AN ONTOLOGY FOR COMPUTER-AIDED MODELING OF CELLULAR PROCESSES

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Emek Demir
October, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Dr. Uğur Doğrusöz(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. İsmail Hakkı Toroslu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Ayşe Elif Erson

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

AN ONTOLOGY FOR COMPUTER-AIDED MODELING OF CELLULAR PROCESSES

Emek Demir
Ph.D. in Computer Engineering
Supervisor: Asst. Prof. Dr. Uğur Doğrusöz
October, 2005

Cellular processes form the *hardware* layer of living organisms. Malfunctions in cellular processes are responsible for most of the currently incurable diseases. Not surprisingly, knowledge about cellular processes are growing at an enormous rate. However, today's molecular biology suffers from lack of a formal representation system for cellular processes. Most of the knowledge is locked in literature, that are not accessible to computational analysis and modeling. Given the complexity of the system we are attacking, the need for a representation system and modeling tools for cellular processes are clear.

In this dissertation, we describe an ontology for modeling processes. Our ontology possesses several unique features, including ability to represent abstractions and multiple levels of detail, cellular compartments and molecular states. Furthermore, it was designed to meet several user and system requirements, including ease of integration, querying, analysis and visualization.

Based on this ontology we also implemented a set of software tools within the Patika project. Primary use cases of Patika are integration, querying and visualization, and we have obtained satisfactory results proving the feasibility of our ontology.

Compared with existing alternative methods of representing and querying information about cellular processes, Patika provides several advantages, including a regular representation system, powerful querying options, an advanced visualization. Moreover Patika models can be analyzed by computational methods such as flux analysis or pathway activity inference. Although it has a more steep learning curve compared to existing *ad hoc* representation systems, we believe that tools like Patika will be essential for molecular biology research in the future.

Keywords: bioinformatics, ontology, pathway.

ÖZET

BİLGİSAYAR DESTEKLİ HÜCRESEL YOLAK MODELLEMESİ İÇİN BİR ONTOLOJİ

Emek Demir
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Asst. Prof. Dr. Uğur Doğrusöz
Ekim, 2005

Hücresel işlemler canlıların en alt seviyedeki donanımlarıdır. Bu düzeydeki bozukluklar halihazırda tedavi edilemeyen pek çok hastalıktan sorumludur. Hücresel işlemler hakkındaki bilgilerimiz hızla artmaktadır. Ancak, günümüz moleküler biyolojisi bu işlemleri kurallı bir şekilde gösterecek yöntemlerden yoksundur. Bilgi dağarcığının büyük bir kısmı bilimsel yazında, bilgisayarlı modelleme ve çözümlemeye uygun olmayan bir biçimde durmaktadır. Çözülmeye çalışılan sistemin karmaşıklığı gözönüne alındığında, uygun bir gösterim sistemi ve araçlarına duyulan ihtiyac açıktır.

Bu çalışmada hücresel sistemleri modellemek icin bir ontoloji öneriyoruz. Bu ontoloji, soyutlamaları, çoklu ayrıntı düzeylerini, hücresel bölmeleri ve molekül hallerini gösterebilmek gibi tekil özelliklere sahiptir. Ayrıca tümleme, sorgulama, çözümleme ve görselleştirme kolaylığı gibi birtakım kullanıcı ve sistem ihtiyaçlarını karşılamak üzere tasarlanmıştır.

Bu ontolojiyi taban alarak bir dizi yazılım aracı geliştirdik. Patika araçlarının temel kullanım hedefleri tümleme, görselleme ve sorgulamadır. Bu hedeflerde elde ettiğimiz tatmin edici sonuçların ontolojinin kullanılabilirliğini doğruladığını düşünüyoruz.

Halihazırdaki yolak gösterme ve sorgulama aracları ile karşılaştırıldığında Patika düzenli bir gösterim sistemi, ileri sorgulama yöntemleri, açık görselleme arabirimi gibi avantajlara sahiptir. Bunun yanısıra Patika'dan elde edilen modeller, akım analizi ya da yolak etkinliği çıkarımı gibi yöntemlerle çözümlenebilir.

Anahtar sözcükler: Biyoenformatik, ontoloji, yolak.

Acknowledgement

I would like to thank my advisor Dr. Uğur Doğrusöz for advising this thesis. I know I will never be able to achieve the standards he set as an advisor.

Prof. Dr. Özgür Ulusoy and Assoc. Prof. Dr. İsmail Hakkı Toroslu receive my gratefulness for reading the manuscript and their helpful comments.

Asst. Prof. Ayşe Elif Erson went in great lengths correcting this manuscript, and her biology expertise was immensely helpful.

I would like to thank Asst. Prof. Uğur Güdükbay for carefully reviewing the manuscript and his invaluable comments.

It is quite difficult to thank properly the PATIKA group. Instead I opt to give a fictional memory of one of our research meetings. I am standing before the whiteboard, drawing a figure to answer the question of Dr. Doğrusöz. He just asked one of those questions, that made all the pieces fall into their places. Özgün Babur is nodding, somehow absently, but I know he is thinking. Our ideas are resonating, then we are probably on the right track. If Dr. Doğrusöz is the judge, he is the jury. Aslı Ayaz has that frown on her face again. She is trying to find a gap, an ambiguity. If there is one, I am sure that she is going to root it out. Several minutes ago Erhan Giral has just listed the plethora of features he fixed/implemented last week. Zeynep Erson is looking distracted, but I know she has a very detailed documentation with her. She is wrestling with one of the most difficult components, yet somehow keeps her sanity. Ahmet Çetintaş will talk about query documentation and the formalisms he came up. I rely on them to do things that I can not do, to succeed where I fail. I know my skills and abilities are highly valued here. It feels like family, a very good one.

During the course of six years I had a chance to work with many excellent undergraduate students. I would like to thank them for the effort they put in PATIKA.

I would like to thank BioPAX group for their suggestions, insights and comments on the BioPAX list.

My parents not only infected me with curiosity, but also thought me how to enjoy it. Every part of this work is a result of their love and care.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Living is not a simple task, even for a cell. A cell struggles to survive, compete and transmit its genetic information to the next generation. This is not an easy task and requires constant scanning of the environment and decision making mechanisms to respond to changes accordingly. The underlying network of interacting genes, proteins, RNAs and other molecules is a massively parallel, inherently complex system. Cellular processes typically span several magnitudes of spatial and temporal parameters.

Reductionist tradition in molecular biology can be traced back to Mendel who, being an atomist, sought genetic atoms that define an organism. Mendel's views were resurrected during the start of this century with identification of chromosomes. What we have witnessed for the last century was essentially a race to identify and catalogue those elements, and associate them with end-effects or phenotypes. In line with the same tradition, the mechanism between the element and the phenotype was often elucidated as an isolated path of interactions. System models exist only in very small scales and simple organisms [31]. Although reductionist approach was very successful in identifying unit components of the cell, it fails when trying to elucidate mechanisms of so called "multi-faceted diseases" such as diabetes and cancer.

One reason is the robustness of the cell. It is possible to think cell's environment as a landscape with many basins, each basin denoting a phenotype, and points themselves being genotypes. Small perturbations in the system is often counter balanced by homeostatic forces or alternative pathways, and are not reflected to the phenotype. However, if somehow a large perturbation occurs, or small perturbations accumulate as in cancer, cell suddenly changes behavior, as it now switches to a different basin. Such behavior is often called *robust but fragile*. There can be combinatorially many paths to achieve this phenotype. In fact attempts to pin down different cancer stages to individual *oncogenes* almost always fails, with the exception of very specific cancer types such as retinoblastoma. Instead what we observe is different genes mutated in different frequencies, supporting our proposition that although mutations in some genes are more critical for inducing cancer, there are multiple (and possibly combinatorially many) paths.

Yeast gene deletion experiments also tell a similar tale [38]. The concept of essential genes are getting less and less important. Instead research is currently focusing on combinations of deletions that has the most effect on the survivability of the system [44].

Finally there are phenomena that can only be detected and analyzed at systems level, such as conserved subgraphs, modules, and emerging patterns, due to the evolution mechanism of the network and the fitness landscape it evolved to such as the topology of the graph, its structure and properties [43]. It is evident that one needs to consider cellular pathways as an interconnected network rather than separate linear signal routes. Perceiving cellular pathways as subgraphs of a single global pathway can provide more meaningful models.

There is a wide array of biological questions that require such a cell scale model. Reasoning about complex biological problems such as mechanisms of multi-faceted diseases using only biological literature is analogous to servicing a Boeing 777 using a textual catalog of its 3.000.000 parts. To make things worse, that catalog is often fragmented, incomplete and contains conflicting information. An integrated model of the cellular processes would help us to fix those missing

and conflicting parts, employ computational methods of analysis and preserve our sanity.

First effort in this direction was creation of models of metabolic networks as early as 1950s. Later, this data was extended and captured by several databases [46, 40]. More recently several advances in experimental and computational methods enabled us to produce cell-scale high-throughput data [34]. Each of these systems, however, capture a certain aspect of the system, and have their own representation system. Finally several efforts were launched to reconstruct signaling networks through human curation. As a result, we are witnessing an array of pathway databases and resources with strikingly diverse representation schema or ontologies, ranging from none to detailed quantitative models, to multi-level qualitative models. Terms such as state, pathway and modules become increasingly popular in the systems biology literature, but one can find different even conflicting definitions for those. Clearly, systems biology is seeking a paradigm, a common way of thinking and communicating. This is not an easy task though, as such a paradigm have to be able to deal with complex, stochastic and combinatorial phenomena that are abound in living organisms [10].

Here we propose an ontology for reconstructing cellular processes. Our ontology is specifically developed for representing metabolic and signaling pathways, and attempts to stay as loyal as possible to current existing notions and concepts used in molecular biology literature. For example entity-state relationships such as *different phosphorylated forms of protein X* can be represented within our ontology, but was a missing concept in existing ontologies when Patika project started. Similarly compartments, molecular complexes and entity level interactions are also covered. This ontology was used as a basis for tools developed within the Patika project in our group. Several unique features of the Patika ontology which allows handling incomplete information, complexity management and integration, arose as a result of requirements analysis during software development. We believe that requirements of the software is closely coupled with the ontology.

The rest of the thesis is organized as follows: Next two chapters provides

background information about cellular processes and describe previous research on modeling them. Chapter 4 attempts to analyze requirements for this ontology. In Chapter 5 we give a textual and formal definition of the ontology follows and discuss open issues that are still to be addressed. Chapter 6 details the implementation of the ontology within the PATIKA system. Chapter 7 gives an overview of PATIKA project and tools, with particular emphasis on how concepts in the ontology were put to use. Finally we discuss the place of PATIKA ontology in the current systems biology landscape and consider future directions.

Chapter 2

Background on Cellular Processes

This chapter briefly gives an overview of cellular processes with emphasis on phenomena that was important on our design choices. It obviously does not attempt to provide a comprehensive overview, but rather focuses on observations that was critical for the design of PATIKA ontology.

2.1 Main Actors

70% of a cell's mass is water. Proteins take the second spot, ranging from 15% to 20%. DNA and RNA form another 2% to 7%. Small molecules make up approximately 4% of a cell's mass, and the remaining 4% to 7% are membranes and lipids forming them.

Proteins are responsible for most of the functional and structural features of cells. Although diverse, all proteins are essentially polymers of 20 types of amino acids. Sizes of proteins are typically hundreds to thousands of amino acids, making a huge set of proteins possible. They act as a skeleton dictating cell's shape and sometimes mobility. They catalyze reactions that are needed

for maintenance and replication. They detect and report environmental changes outside the cell. And more importantly they act as switches, essentially forming one of the basic elements of decision making mechanism we were searching for.

Proteins provide the function and structure, whereas nucleic acids, DNA and RNA, provide memory and inheritance. Similar to proteins, nucleic acids are polymers of 4 different nucleotides. Genetic material in cells reside in several large DNA molecules. Through templating they can self replicate, and through transcription and translation, they act as templates for RNA and indirectly, protein synthesis. Substrings of chromosomes that act as such templates are called genes and the process is called gene expression.

Small molecules such as various ions, saccharides, lipids alcohols and other organic compounds act as structural units, co-factors, energy storage and messengers. Some small molecules are ubiquitously present such as water and ATP, in the sense that they are assumed to be always present, and their consumption is considered insignificant as it is drawn from a very large pool.

2.2 Control Mechanisms

A cell uses a diverse array of mechanisms for controlling and directing the flow of information. An attempt to build an ontology first requires an in-depth analysis of those mechanisms. This analysis will be helpful later while justifying our design choices and assessing our coverage.

When discussing control mechanisms in the cell, it is useful to slightly extend the chemical paradigm, such that some very similar molecules are grouped under the term *biological entity*. For example, different phosphorylated forms of p53 are indeed different molecules, but they are grouped under the term *p53 protein*. Then one can conceptualize the network of reactions as signals carried through changes in the state of entities, rather than individual reactions.

In this modified paradigm, a biological entity starts its life by either being

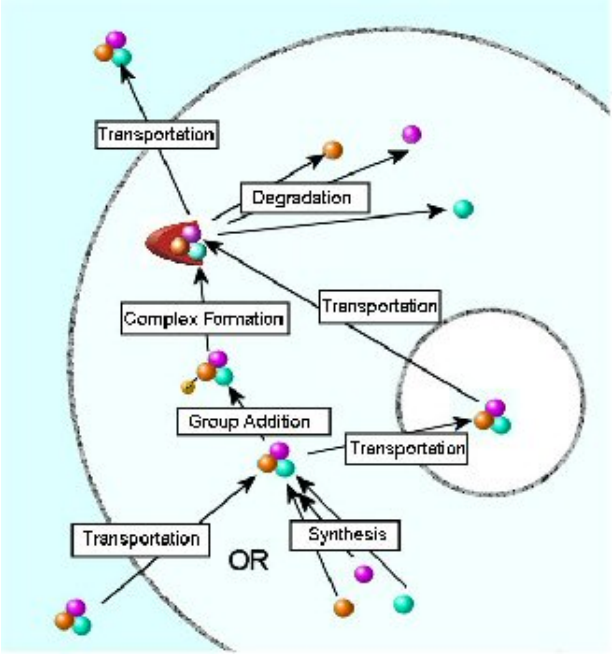

Figure 2.1: Life cycle of an entity in the modified paradigm. From bottom to up, an entity's life starts by being transported into the cell or synthesized, then it goes through an optional series of modifications/transitions. Finally it is degraded or transported out of the cell.

synthesized from its precursors, or transported into the cell, then it goes through a series of transitions, such as receiving and losing chemical groups, forming complexes and different isomers or changing cellular location. An entity's life ends by either being degraded, or transported out of the cell (Figure 2.1).

Each such transition changes the information context of the molecule. The set of transitions an entity goes through is context dependent, i.e. an entity can follow very different paths depending on the environmental, spatial and temporal variables, which in turn can trigger different cellular responses. Throughout evolution, several types of mechanisms were reused to control this flow of information at different levels and different time scales. Rest of this section discusses those mechanisms. However, one should also bear in mind that although these examples cover a majority of cases they are not comprehensive. Every now and then, scientists come up with a new mechanism or a variation of an existing mechanism, to prove that our understanding of even the most elementary mechanisms are far from complete [54].

2.2.1 Transcription Factors

In a cell not all genes are expressed uniformly. Some genes, often called housekeeping genes because they are involved with *everyday tasks* such as metabolism, are expressed with a relatively constant rate. Others such as those involved in controlling cell cycle can vary drastically in their expression rates and times. The most classical example of such regulation is transcription factors, where a protein specific to the sequence in the *vicinity* of the target gene binds to that region and increases or decreases the binding rate of the RNA polymerase to the promoter [37]. Several transcription factors, different RNA polymerases and local changes in DNA structure can combine to provide several different mechanisms [13, 14]. Roeder provides an excellent review of those processes [71]. An alternative method is blocking the gene with small interfering RNA molecules [24, 82, 57].

2.2.2 Chromatin Structure

Alternatively the expression rate can be regulated by changing the chromatin structure. DNA is typically stored in a highly condensed fashion, folded around proteins called histones. In order to be transcribed, some portions of the DNA molecule needs to be unfolded and exposed, a process regulated by modified histones. Histones are subject to an enormous number of post-translational modifications, including acetylation and methylation of lysines, and arginines, phosphorylation of serines and threonines, ubiquitylation and sumoylation of lysines, as well as ribosylation [67] (See Figure 2.2). Each combination might lead to distinct chromatin structures, effectively. Modification of a histone subunit, called H3, is increasingly considered as a generic transcriptional regulation mechanism. Finally gene expression is also regulated by adding methyl groups to the DNA.

2.2.3 Post Transcriptional Control

RNA molecules that act as templates for protein synthesis are called messenger RNA (mRNA). Once an mRNA is synthesized, it goes through a complicated task

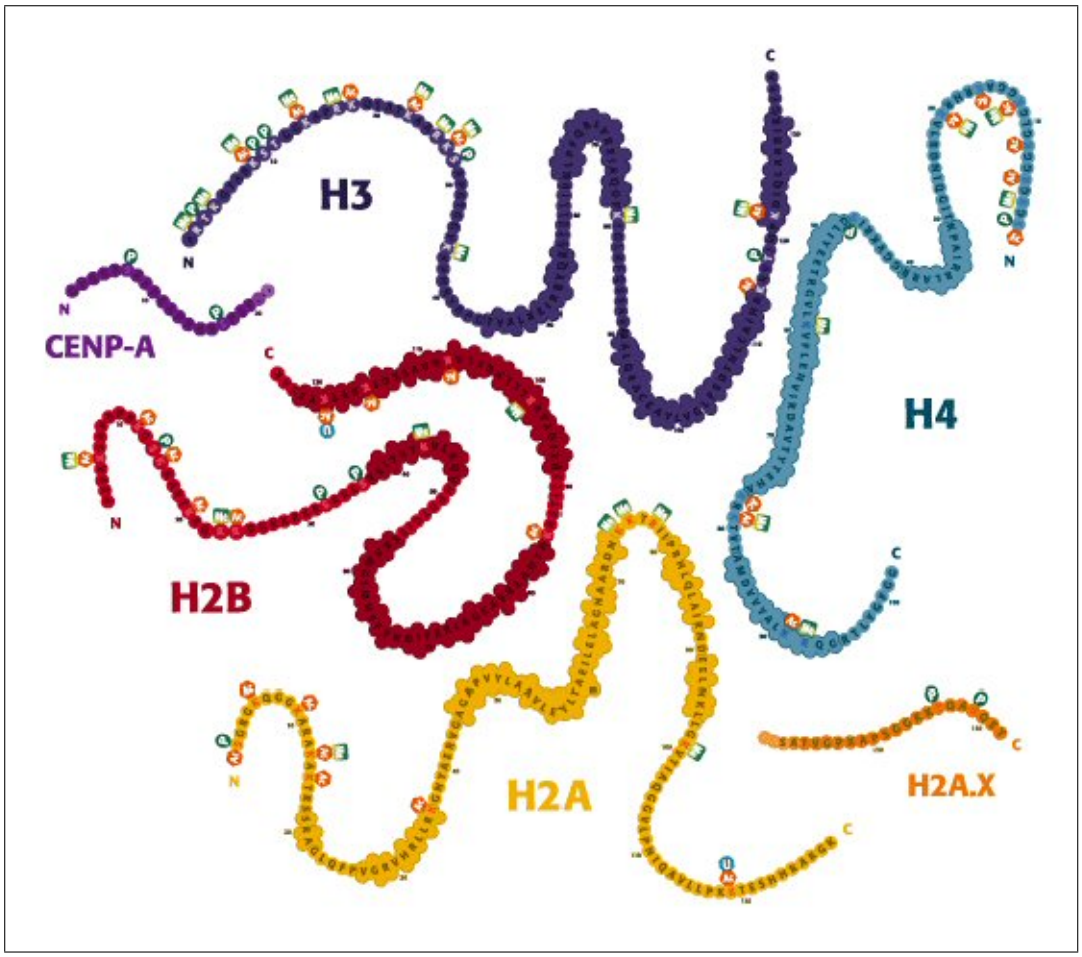

Figure 2.2: A map of histone modifications. Histone subunits come together to form a large protein complex which acts as a spindle for DNA. Note that all subunits have several modification sites and these modifications can be combinatorial in nature. (Courtesy of Peterson *et al* [67]

.

of RNA processing [54]. Several substrings of the RNA, are removed, and specific sequences are added to both ends of the RNA. The process controls the longevity and function of the mRNA. Finally, RNAs produced in the nucleus have to be exported, either to fulfill their function in protein synthesis or to mature into functional particles. All of these steps are controlled by a complex mechanism of proteins and act as another layer of control mechanism [70].

2.2.4 Alternative Splicing

RNA splicing is a post-transcriptional process that occurs prior to mRNA translation. A gene is first transcribed into a pre-messenger RNA (pre-mRNA), which is a copy of the genomic DNA containing intronic regions destined to be removed during pre-mRNA processing (RNA splicing), as well as exonic sequences that are retained within the mature mRNA. During RNA splicing, exons can either be retained in the mature message or targeted for removal in different combinations to create a diverse array of mRNAs from a single pre-mRNA, a process referred to as alternative RNA splicing. Alternative splice events that affect the protein coding region of the mRNA will give rise to proteins which differ in their sequence and possibly, in their activities. Alternative splicing within the non-coding regions of the RNA can result in changes in regulatory elements such as translation enhancers or RNA stability domains, which may have a dramatic effect on the level of protein expression [80]. More than half of human RNAs are estimated to be subject to alternative splicing [61, 60]. A bias for alternatively spliced genes in signaling pathways [59] indicate that in fact this is a very common decision making mechanism for cells. High-throughput experimental methods for detecting alternative splice forms are being developed [77].

2.2.5 Naturally Arising Anti Sense RNA

Some RNA sequences are complementary to other endogenous RNAs, and are often called Natural Antisense RNA (NARs). Their *modus operandi* can be both

cis, where NAR is transcribed from the opposing strand, or *trans*, from a completely separate loci. Although much less is known about *trans* NARs, it is known to induce gene silencing in *Drosophila* [5] and probably humans. Antisense regulation, both at transcription and post-transcription appears to be co-evolved and has a lot of common patterns [62].

2.2.6 Regulons

Experiments reported over the past several years, including genome-wide microarray approaches, have demonstrated that many eukaryotic RNA-binding proteins (RBPs) associate with multiple messenger RNAs (mRNAs) both *in vitro* and *in vivo*, regulating the translation of the bound RNA molecule, often called regulons. Although still a novelty, regulons have been shown to be critical in protein targeting [45].

One should keep in mind that, although these mechanisms are listed separately in fact they are tightly coupled. Figure 2.3 [54], shows the known coupling between these processes.

2.2.7 Post Translational Control

Perhaps the richest layer of control, in terms of diversity of mechanisms, occur after a protein is translated. Also the lifespan of their effects cover a broad spectrum ranging from nanoseconds to days, making them typically very hard to detect using high-throughput methods.

Group Additions

A small molecule added to a specific residue of protein can lead to a change in its function. These modifications occur at highly specific sites which are conserved across species and different proteins in sequence and structure.

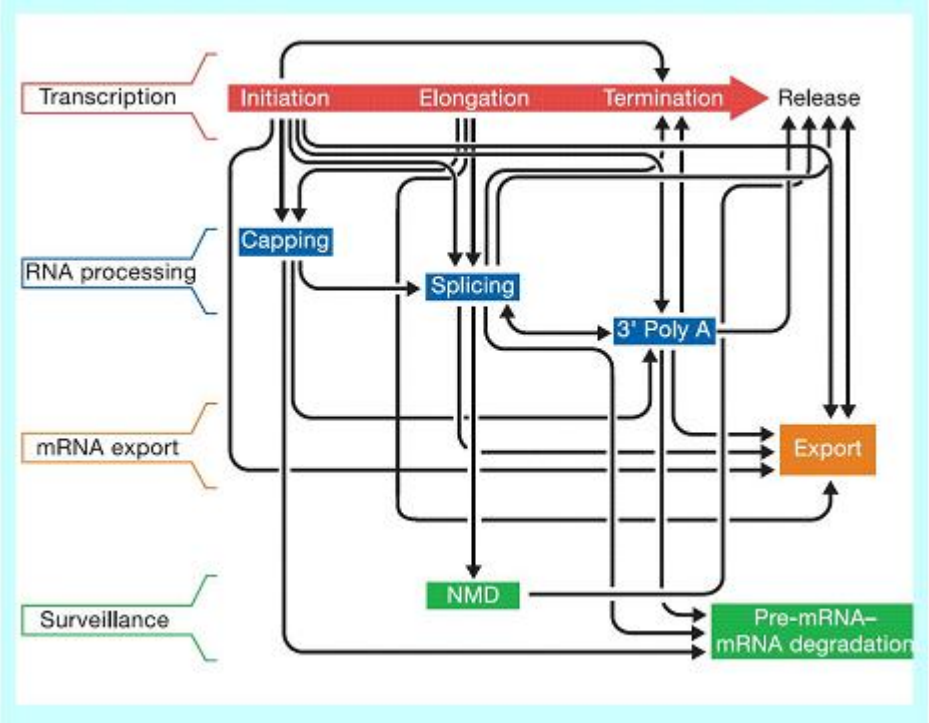

Figure 2.3: Transcriptional and post transcriptional control is highly coupled. In the figure, red arrow indicate the stages of transcription. Steps of RNA processing and export is listed below, in a chronological order, with respect to transcription. Black arrows indicate physical/functional coupling between two steps. Courtesy of Maniatis and Reed [54]

Swiss-prot[1] is a database of curated database sequences. As a part of their curation effort they provide a controlled vocabulary for post translational modifications, which lists 281 different groups that are known to be attached to proteins[2]. Some of these groups allow proteins attach and penetrate membranes, whereas others acts as cofactors for specific reactions. Yet another group induces structural changes in the protein, often leading to activation of a catalytic activity.

Phosphate belong to this latter category and are by far the most common modification. There are 1027 kinases identifed in the human genome, proteins, whose sole purpose is to add phosphate groups to other proteins. Similar to gene networks, kinases can activate other kinases by phosphorylation, leading to a phenomena called signaling cascades. Signaling cascades allow multiplication of the signal, and fine grained control for signal propagation [64].

Another important observation is that a protein might potentially receive multiple group additions, leading to combinatorially many different molecules [2, 88]. In several cases enumerating each combination might not be feasible.

Cleavage

Sometimes the peptide sequence of a protein can change through cleavage of the peptide. Although this is typical of secreted proteins, it is also used as a mechanism of protein activation control to induce major cellular mechanisms such as apoptosis, induced cell suicide [73, 20].

2.2.8 Complex formation

Relatively long lasting specific non-covalent interactions between molecules are very common in a cell, and often called molecular complexes. Purpose of some complexes are purely catalytic, or structural. However there are other complexes

[1] http://ca.expasy.org/sprot/
[2] http://ca.expasy.org/sprot/userman.html#PTM_vocabularies

which serve as an *AND* operator, in the sense that presence of all members of the complex have to be satisfied to perform required function. Alternatively, it can be used for decoupling different functions, and reusing same molecules. For example different transcription factors use the same recruitment mechanism to control expression of different genes.

Recently several high-throughput essays for detecting complex forming interactions between proteins, DNA and RNA have been developed, which in turn resulted in several *interaction databases* that capture these data.

2.2.9 Spatial Aspects

So far, we have considered only static part of signaling. But spatial aspects also plays important roles in signal transduction and cell behavior such as cell cycle.

The most obvious mechanism is sub-cellular targeting or compartmentalization. A cell is far from being a homogeneous environment. It is divided by membranes, and often has special *points* which is specialized in providing a certain service or behavior, such as axon hillock. Concentrations of different molecules in different regions and compartments can be different, forming diverse contexts. For example transportation to nucleus is a critical control point for many transcription factors.

Gradients, on the other hand, occur in a free but *not well stirred* medium. Often formed by two molecules with different diffusion constants and opposite activities, gradients may form two sorts of patterns. If the inhibitor (or substrate) diffuses much more rapidly than the activator, the activator piles up in local regions of space, forming steady-state (time-independent) patterns as in chromosome separation. On the other hand, when the diffusion constant of the inhibitor (or substrate) is about the same as (or less than) the diffusion constant of the activator, traveling waves of activation propagate through the medium. Traveling waves of cyclic AMP, a small molecule often act as a signal messenger, in fields of Dictyostelium amoebae govern the processes of aggregation [52, 11].

2.2.10 Temporal Aspects

Cells are by no means static machines. Oscillation loops and thresholds play an important role on the regulation of cellular processes [12]. An important temporal aspect is cycles, a series of reactions that affect each other in a cyclic manner, either through substrate/product relations as in the Krebs cycle or effector relations as in the cell cycle, Circadian clock or synaptic signaling pathways [9]. Depending on the relations a cycle is either a positive cycle, i.e. it is self enforcing, or negative cycle, it is self controlling. Temporal aspects more then often require quantitative analysis of the signaling network, thus is best captured by simulation and flux analysis studies.

Chapter 3

Related Work

A recent collection of pathway resources available on the net[1] reveals more than 181 *pathway resources*. Most of these resources are pathway databases themselves. What is more striking than the number of resources is the diversity of network paradigms they use.

The Pali Buddhist Udana, tells the story of 6 blind men, who attempt to obtain a picture of an elephant by feeling it. Each one of them touches a different part, tusk, body, ear, trunk, leg and tail, and make different claims about what animal looks like. Similarly, different types of networks of cellular processes capture different aspects of the system, and can present strikingly different paradigms. Still, one should keep in mind that these paradigms arise more from experimental systems and common abstractions rather than physical or chemical features of the cell.

3.1 Gene Networks

Changes in the expression rate of a gene can in turn regulate other genes, an observation which led to one of the first network models in biology, *gene networks*

[1] http://cbio.mskcc.org/prl/index.php

[76]. A gene network, is a directed graph where nodes represent genes and edges represent a *regulation path* from source to target. Lytic/lysogenic switch of the lambda phage gene network was one of the earliest examples of stochastic behavior modeled in living organisms [1]. Models of gene networks were later extended to cover combinatorial effects of the genes. Segal et al. provides an interesting approach where a hierarchy of genes were built which in turn control a module, or a grouping of target genes [78]. Common data sources for these models are microarrays [47, 25] and protein DNA interactions of transcription factors [56, 55]. **Ch**romatin **i**mmuno**p**recipitation chip followed by cDNA microarray analysis (ChIP^2) is also becoming a major high-throughput assay for proteinDNA binding data [49, 15].

The advantage with gene networks is that it is relatively easy to obtain system wide data using microarrays and ChIP^2. The downside is they can not capture mechanisms that does not involve transcriptional regulation. Moreover, the *activation path* from one gene to the other may be subject to control by other genes, through possibly combinatorial mechanisms, which again can not be captured by gene networks.

3.2 Interaction Networks

Interaction networks were a result of several experimental systems that can detect complex forming interactions. Yeast two hybrid assay and protein chips allowed proteome wide analysis of protein-protein interactions. System scale, pair-wise interactions maps, often called *interactomes*, were constructed for several organisms including *S.cerevisiae*, *C.elegans* and *D.melanogaster*. Using sequence homologies it was possible to predict a substantial amount of human interactome as well [17]. Gavin *et al* and Ho *et al* also demonstrated a mechanism for detecting multi protein complexes by first using hundreds of tagged proteins as baits, precipitating complexes including these proteins and finally using mass spectroscopy for detecting complex contents [28, 32]. Additionally, structures of complexes that were detected by X-Ray crystallography also provides a substantial amount of

data [51, 3].

Although interaction networks provide almost system scale data, the number of false positives are still high. Moreover, since all interactions are obtained *in vitro*, chances are some detected interactions never occurs *in vivo* due to temporal and spatial constraints. Nevertheless, interaction information is still a valuable facet of the *elephant* and must be captured by a pathway ontology.

BIND is perhaps the most extensive interaction database [3]. Description of an interaction encompasses cellular location, experimental conditions used to observe the interaction, conserved sequence, molecular location of interaction, chemical action, kinetics, thermodynamics, and chemical state. Molecular complexes are defined as collections of more than two interactions that form a complex, with extra descriptive information such as complex topology. Pathways are defined as collections of two or more interactions that form a pathway, with extra descriptive information such as cell cycle stage. Currently BIND contains 32716 entities and 79820 interactions.

Another important interaction database is Database of Interacting Proteins (DIP) [87]. DIP focuses only on protein-protein interactions and uses a hybrid curation effort, where the core portion is curated by researchers and the rest is by computational methods. It currently contains 44349 interactions among 17048 proteins. It is possible to query these interactions and visualize it using an applet (JDip). DIP is tightly linked to PIR and SwissProt and it accepts submissions from users.

HPRD is a database of human proteins, but also contains a significant amount of protein-protein interactions [66]. Information about the domain and region of interaction, if available, is present as well as the type of experiment done to detect the interaction. Expression, domain architecture and post-translational modifications are also curated for each protein. A number of curated pathways created from the interaction data are available as images.

3.3 Metabolic Networks

Metabolic networks were determined *in vitro* by classical enzyme assays as early as 1950s. The core paradigm of the metabolic network is chemical paradigm with two major differences, first any reactions containing the same substrates and products are considered identical. Second, the molecules catalyzing the reactions (enzymes), and the substrates/products form two distinct sets. An established classification of enzymes, based on the reactions they catalyze are also an important part of this ontology. The Enzyme Commission (EC) system (http://www.chem.qmul.ac.uk/iubmb/enzyme/) [19] is perhaps the earliest, and one of the most widely used, examples of a hierarchical controlled vocabulary in biology. Rather then linking enzymes directly to the reactions, each reaction is instead assigned a set of EC numbers, meaning that any enzyme falling into this category can actually catalyze this reaction. Typically these reactions are not assigned to a cellular compartment. Each reaction is actually a cross-organism, cross-compartment abstraction of actual instances of reactions. This generalization, however, has one very useful feature, it is possible to semi-automatically obtain the metabolic map specific to an organism, once its genome is sequenced [72].

Metabolic network ontology can cover only a certain subset of existing chemical network, because it lacks structures for representing aforementioned control mechanisms. This is mostly due to the fact that enzymes are never substrates and products of reactions. Thanks to recent advancements in metabolic profiling, which allow non intrusive *in vivo* measurements [68, 4, 81], our knowledge about metabolic networks are almost complete, including kinetic constants. This led to successful efforts for simulating *minimal cell*, more correctly *minimal metabolism*. Extending this network to more comprehensive models are currently an active field of study.

Metabolic pathways are more manageable compared to signaling pathways in terms of complexity. Therefore, efforts for drawing every interaction in those pathways as a still image have proved to be successful. These databases have a rigid definition of a pathway and they never create a pathway on the fly. Unfortunately, these features are essential for regulatory pathways.

One of the well-known metabolic databases is Kyoto Encyclopedia for Genes and Genomes (KEGG) [40] . KEGG is composed of a set of still images defining metabolic pathways, a set of tables defining relationships and orthologous entries, and hierarchal texts defining these entries. These components are backed up with a querying system that allows users to extract pathways. Although KEGG started as a metabolic pathways database, it recently started an initiative for modeling cellular signaling processes as well. However, signaling part lacks the ontology of the metabolic part and is not a truly pathways database.

EcoCyc [46] is one of the most serious attempts toward building an ontology for metabolic pathways. EcoCyc features the entire small molecule metabolism in *E.Coli* and provides support for querying and computation. EcoCyc is also the first true attempt to an integrated environment since it also provides visual tools for analyzing and displaying cellular environments [42]. They define different types of molecules, each with its own class, and consider different states of a molecule as different actors. In addition, reactions are defined to be independent entities, and molecules are linked to the reactions by distinct relations, which they call slots. Each molecule may optionally be tagged with a cellular compartment. Their ontology also makes use of the *pathway* concept to define summary abstractions, which may be used for defining data at varying levels of detail.

3.4 Signaling Networks

Signaling Network ontologies can model metabolic networks, and more complex signaling networks. They typically allow any role for any molecule. They also provide methods for representing complexes, spatial constraints, and abstract groupings. Despite some efforts, currently there is no *standard* ontology for modeling signaling networks. Signaling network ontologies provide the most detailed ontology, but the detail of the ontology also dictates manual curation, a very scarce resource. Currently most of the data on signaling networks reside in the literature in free text form [29].

Cell Signaling Networks Data base (CSNDB) is a data- and knowledge- base for signaling pathways of human cells. It compiles the information on biological molecules, sequences, structures, functions, and biological reactions that transfer the cellular signals. Signaling pathways are compiled as binary relationships of biomolecules and represented by graphs drawn automatically. CSNDB's pathfinder querying mechanism is probably one of the pioneering works in the field. Unfortunately, CSNDB suffers from a naive data model in which you may get multiple instances of the same molecule or their orthologous and generic variants in the same graph.

TRANSPATH [48][2] employs a powerful hybrid ontology of both mechanistic (actor-event based) and semantic for describing cellular events. It has a well-defined structure and an extensive content. It focuses on pathways involved in the regulation of transcription factors. All data is extracted by experts from the scientific literature. TRANSPATH features a basic querying system that allows searching for molecules. TRANSPATH currently does not support computations but has a very suitable structure as long as all data entries are made in mechanistic model.

AfCS[3] is a collaboration between 17 universities in USA and Nature Publishing Group, attempting to provide curated pathway models. [63]. AfCS is an interesting project, since it takes collaborative reconstruction as its primary use case. AfCS relies on a relatively loose ontology and linking models using URLs to provide a distributed, collaborative environment. AfCS is focused on signal transduction and as such can model concepts such as complexes and cellular location.

The Reactome project[4] is a collaboration among Cold Spring Harbor Laboratory, The European Bioinformatics Institute, and The Gene Ontology Consortium to develop a curated resource of core pathways and reactions in human biology [39]. The information is authored by biological researchers with expertise in their field, maintained by the Reactome editorial staff, and cross-referenced

[2]http://biobase.de/transpath
[3]http://www.afcs.org
[4]http://www.reactome.org

with with PubMed, GO, and the sequence databases at NCBI, Ensembl and UniProt. Reactome's ontology, which was developed independently, is very similar to PATIKA's mechanistic level , and in fact it is possible to convert Reactome data into PATIKA's. Reactome's manually curated repository, provides the most extensive high quality signaling pathway data for humans. Reactome allows a very loose concept of *generic* which can be used to model generic states, e.g. damaged DNA. However, Reactome does not differentiate between different generic concepts and does not handle ambiguities in the model that might arise due to their semantics.

The aMaze project aims to provide a workbench for modeling which can deal with a large variety of cellular processes including metabolic pathways, protein-protein interactions, gene regulation, sub-cellular localization, transport, and signal transduction [50]. aMaze's data model is again very similar to PATIKA [84, 83] although there are several differences that makes transformation from one to another very lossy. aMaze ontology was one of the first to introduce the concept of state.

Inoh project is another pathway database, that provide several new concepts. Of particular interest is the usage of the compound graphs. If we do not consider KEGG's and EcoCyc's pathways, INOH receives credit for publishing the first concept of abstractions. INOH's abstractions are focused on homologies, it allows a form of *homology templates* [26, 27].

In general, signaling pathway databases focus on the direction of signal flow, showing activation and inhibition relations among signaling molecules. In these systems one can follow the transduction of a signal. However, the mechanisms of regulation is often omitted in favor of simplicity, leading to ambiguities in the model, and hindering any possible functional computations. Considering a molecule to be only in active and inactive states is clearly an oversimplification since a molecule often times has more than one active state, each performing a different activity.

Because of the aforementioned reasons, efforts for developing common, standard ontologies are gaining increasing support in the scientific community. There

are efforts in multiple levels [35, 6, 30, 18, 72]. We believe that coercion between these different levels are important for the integration of biological data at different levels. For example, sequence, yeast two hybrid, microarray and metabolic simulation data have different perspective and level of detail, although they describe the same system. An ontology which could integrate and store data from such different sources and present them seamlessly in different perspectives, isolating a user from such heterogeneities, is critical to modeling of such a complex system.

Chapter 4

Requirements Analysis

In the future, we expect that a biologist who wants to adopt a system-level approach for modeling a disease or a biological phenomena starts by constructing a large network of knowledge, spanning multiple databases and information sources. They do this by specifying queries from a single common interface. They then add their knowledge and data into it, and visualize and analyze the resulting model. They will like to share and integrate their model with their colleagues, especially if they are in a large distributed research community (e.g. European NoEs or AFCS of United States). They often will couple this model with high-throughput data, trying to figure out how the changes in the genotype led to the phenotype they are observing. They will need to change their view port to perceive the model at varying levels of detail and perspective, ranging from medical imaging data to individual reactions to protein structure. They specify complex queries to test their hypothesis or come up with new targets for drugs. They can annotate and couple their models with logical inference methods, so that a medical doctor can use it for diagnostic purposes. Clearly present-day biologist is unprepared and unequipped for this challenge. There are currently numerous tools and databases, providing some of this information. However, none of them provides the tight integration needed by the biologist. This chapter analyzes the requirements for an ontology that can at least partially address above use case history.

4.1 Use-Case overview

Why do we want to define a formal specification, after all? Following are the use-cases we envision, where an ontology is mandatory or helpful.

1. *Rapid knowledge acquisition*: A common representation system would enable users to query, retrieve and visualize pathway data using a common interface, allow a faster and easier way to obtain information on cellular pathways.

2. *Collaborative model building*: A common format is the first natural step for a collaborative environment, allowing integration of different models from different sources.

3. *High-throughput data analysis/integration*: A common ontology would allow building system-scale models, with much more explicit semantics. This would be a major breakthrough for analyzing system-wide, high throughput data.

4. *Scenario/target/hypothesis testing*: Again a system-scale model would allow testing plausibility of ideas, or investigating possible outcomes and side effects of a change.

5. *Simulation*: There are several levels and models for simulation of cellular systems. Although our current ontology does not meet the requirements of most simulation systems, nor does the current biological data. However, an ontology would serve as a blueprint, for building system-scale models, with increasing levels of detail.

6. *Pathway-inference*: There is already a substantial amount of effort to infer "pathways" from high throughput data or literature. However, without a common ontology, results of these efforts remain isolated pieces of knowledge, which cannot be integrated with or compared to each other. Moreover, most of these methods use ad-hoc, loose definitions of a pathway, which result in data that has little biological value. A common ontological framework is also essential for these efforts to flourish.

7. *Customized drug combination design*: A use-case that combines several use-cases mentioned above and of particular importance is the ability to foresee how a cell would respond to a certain combination of drugs. This is especially important for a cure for cancer, as one can couple such a model with high-throughput techniques and computational methods to come up with best drug combination that can most effectively select and kill cancerous tissue, with minimal side effects.

4.2 Complexity of Cellular Processes in Humans

An estimation of the complexity of the problem we are attacking is essential to set our requirements. In this section we try to estimate some statistics related to the cellular processes in humans.

Although there is a constant debate on the issue, the best estimates for the number of genes in the Human Genome is approximately 25000 [65]. Differential expression of these genes leads to approximately 250 different cell types [33], each having different chemical, spatial and temporal contexts. Different cells express different combinations of approximately 1500 different receptors [85], which *listen* different environmental changes, and responsible for most of the difference in cell's reaction. An array of 518 known protein kinases and approximately 150 phosphatases took part in signaling pathways, along with components for other mechanisms, which transfer these signals to the various response mechanisms [63].

Considering all the control mechanism we have revised, a rough estimation indicates 10-100 states per gene on the average(See table 4.1 for more details) . This indicates that the networks human cellular processes contain 10^5-10^6 genetic components only. Considering small molecules, combinatorial and genetic phenomena, our estimation is a network with a magnitude in the order of 10^6 .

Network Component	Number
Cells	10^{14}
Cell Types	200
Genes	25000
Splice Variants per Gene	2.5
PTM[1]s per Proteins	2.5
Protein States per Gene	20

Table 4.1: A rough estimation of numbers of various cellular components, based on currently known numbers in the literature. (PTM stands for post translational modification)

4.3 Clarity, Content and Coverage

While discussing design choices and trade-offs we made in our ontology, it is useful to have an evaluation space. Here, we define three criteria we found most relevant.

1. *Coverage* refers to the amount of data an ontology is able to model, compared to the entire biological knowledge corpus. Increasing coverage has the obvious benefits of being able to model more biological phenomena, thus able to solve more. However, most of the time in order to be able to cover new ground, one needs to introduce new classes/relations/rules into the ontology, or relax the semantics of existing ones to accommodate the new phenomena.

2. *Content* describes an unambiguous and regular structure in the information to be modeled. A higher content is key for better and more powerful analysis methods. For example still image databases has very high coverage but no content at all, as one can not even query the system against the names of proteins. Increasing content often means introducing new rules to handle exceptions, or leaving out these exceptional cases.

3. *Clarity* refers to the intuitiveness and comprehensibility of the model itself. The more classes/relations/rules there are, less the clarity is. The advantage of clarity is obvious, a less steep learning curve. However more then often, it comes with a cost in content.

These principles often conflict with each other, and a compromise must be made, considering the nature of the data at hand.

4.4 Requirements

Below is a brief overview of major requirements for PATIKA project and ontology. One should note that they are not isolated items rather they are coupled with each other, ontology and software components.

4.5 Integration

The primary use case for PATIKA ontology is collaborative reconstruction of human cellular processes. Modeling the human as a system is a task that scales higher in several orders of magnitude in complexity to any engineering model our civilization has so far managed to build. Modeling such a system is clearly beyond the capabilities of a single lab or group. We need a large scale collaborative effort.

A molecular biologist has a very good grasp of a particular subgraph of this complex network. There are reviews which would put several such subgraphs together to form a larger map of a certain pathway. Following the same path, why not build an integration system, that would scale up to the complete graph of the processes in a cell, where scientists could put together their knowledge, in a similar fashion to put pieces of a puzzle together. However there are several obstacles we need to resolve first before we can hope to realize an integrated pathway database. First, unlike Genbank and Protein Data Bank, where user submissions are typically isolated records, a pathway submission needs to be merged with the already existing model in the database. This raises several problems related to identity, concurrency and conflict resolution, which needs to be addressed at the ontology level. It is reasonable to assume that a user will view only a limited portion of the complex network of available cellular pathways

at a time. Hence a modification to the existing data in this small window may affect the integrity of this entire network. In order to deal with this, an ontology should also state the integrity rules of the pathway data, enabling us to construct a robust model. Only with the help of such rules, automated integration of data into the existing knowledge base is possible.Second, models created by different users might be at different levels of detail and precision. Finally these models, similar to journal articles, contain heavily interpreted information and a revision mechanism is often required. Tackling these problems require improvements in existing ontologies and development of new protocols and software tools. This goal creates an array of sub-requirements, some of which needs to be addressed at the ontology level.

As our goal is to annotate and model *all* processes within a cell, we should try to maximize our coverage during design, with making as little as possible sacrifices from clarity and content.

4.6 Incomplete Information

There are fields, as in metabolic pathways, where our understanding is much more complete, with a nearly complete map of reactions, their reaction constants and even typical concentrations. On the other hand, data on most signaling pathways are still vague at best, with indirect relations, ambiguous mechanisms and unknown reaction constants. A more strict model would dismiss a lot of signaling data leading to low coverage, where a lax model would poorly model metabolic pathways.

To make things worse, new high-throughput techniques such as Y2H(yeast two hybrid) system, or (ChIP^2) produces data that are inherently partial. For example, in the case of Y2H, we know that two proteins interact (and that is if it is not a false positive), but we do not know its cellular location, or other participants. We need ontological facilities to separate known from unknown clearly. By adding new classes and rules we decrease the clarity of the model,

balanced by an increase in clarity and content.

4.7 Multiple Levels of Detail

Ability to represent multiple levels of detail is a very important requirement that arises due to the heterogeneous nature of biological knowledge. In collaborative construction, as desired modeling detail level of one user can be drastically different from another, a user may not be able to integrate their knowledge if the existing level of detail in the database does not match theirs. We attempt to address this problem by allowing multiple levels of detail, using different abstractions. A user can represent a metabolic pathway in a very detailed form, and can include a very abstract level signaling pathway regulation in the same graph.

4.8 Complexity Management

A more vigorous model, for most of the time, means a more complex representation, which in turn leads to models cluttered with states and interactions that are possibly of no interest to certain users. It is therefore desirable to manage complexity, such that the part of the model that a user currently focuses on is represented in full detail, where other portions are hidden or represented at a more abstract level. Similar analysis and query facilities must also be provided. The ontology should provide facilities to reduce complexity through capturing groupings and similarities, abstracting them and hiding their details when desired.

4.9 Analysis

Analysis options one can provide to the user has a lot to do with the content of the model. For example ordinary differential equations simulation of a model

requires each transition to be associated with a differential equation, and each state with an initial condition. One often needs to trade off a lot of coverage for such a content, as such data is not often available, however stoichiometry is often known so it is a realistic trade off to leave out data with unknown stoichiometry, so that one can perform flux analysis on the model.

As a minimal requirement, we typically would be able to query and retrieve a subgraph of it. Apart from SQL like queries, we would like to be able to run graph theoretic queries to identify shortest paths, positive feedback loops, common regulators etc. It is important to map such graph theoretic problems to biological ones, identify and verify their relevance to the biological problems and come up with ontology modifications to improve them.

4.10 Visualization

Even though the ultimate goal in analysis of pathway data is support for functional computations and simulations on the model created, a simpler yet very effective form of analysis is possible through visualization. First of all, an effective visualization is only possible through an ontology that permits drawings of pathways with intuitive images (i.e., graphical user interfaces). Another necessary tool for effective visualization is automated layout, with which aesthetically pleasing, comprehensible drawings of pathways can be produced. It is also crucial to have proper complexity management tools for analysis of complex pathways. Such techniques are necessary at both the visualization level and at the level of knowledge base, which is free of geometrical information for pathways. Thus the ontology should suggest various ways to reduce the complexity of the information that the user deals with at one time. Another way of dealing with complexity is by supplying powerful querying mechanisms. Such mechanisms enable researchers to find their ways around in the *jungle* of paths, again requiring a rigid ontology. Visualizing external multi-dimensional data on pathway graphs is also very important for making the model useful. Although visualization seems like a software aspect rather than an ontological one, there are still ontological choices, which can

lead to models that can be more effectively visualized, such as designing objects so that node degrees and depth of compound graphs are reduced or limited.

Chapter 5

Ontology

In this chapter, we describe PATIKA ontology, to model networks of cellular processes through integration of information on individual pathways. Our ontology is suitable for modeling incomplete information and abstractions of varying levels for complexity management. Furthermore, it facilitates concurrent modifications and extensions to existing data while maintaining its validity and consistency.

We first define the fundamentals of our ontology for modeling cellular processes, then we give a formal owl definition.

5.1 PATIKA Objects

Every first class object in the PATIKA ontology is a PATIKA *Object*, which describe the common functionality and information. A PATIKA Object has a *unique id*, a *version*, an *author* (for the purposes of provenance), and a *data source*, which describes how this phenomenon was observed and points to the literature references. A data source is further classified into four classes as follows:

1. *Experimental*: Existence of this object can be tracked to some experimental observables. In this case data source typically points to a journal article.

2. *Inferred*: This object was inferred from other experimental observables, such as complex prediction from 3D structure, or a reaction that was inferred by homology. In this case the data source typically points to the article in which the method was described.

3. *Imported*: This object was automatically imported from another similar database, such as Reactome. In this case data source typically points to that database entry.

4. *Other*: Used when the data source does not fit aforementioned cases, such as psychic revelation and divine intervention.

Although there are more detailed data source ontologies [41], we found this classification quite sufficient for the time being.

Every PATIKA Object also has a name and description, which should comply with external naming conventions and vocabularies (such as HUGO [86], or GO [30]) whenever possible. This however was not enforced in the core ontology in any way. Finally every PATIKA Object is optionally associated with a set of GO terms.

Content of the PATIKA Objects might be extended in order to comply with standards that are defined by various initiatives such as Psi-MI, BioPAX [18] or SBML [35], or for adding in user data such as the results of a microarray experiment.

5.2 Bioentities

More than often actors, especially macromolecules have a common path of synthesis and/or are chemically very similar. For example, a p53 protein may be in native, phosphorylated, and MDM2-bound forms. Another example is cytoplasmic and extracellular calcium. These molecules have different information contexts, and changes in their concentrations leads to clearly different outcomes.

It is possible to model these molecules as separate entities, ignoring the mentioned grouping. However, these entity groupings are very common and deeply embedded in the current biological paradigm. In fact there is a wealth of information that is only available at the entity level. Therefore it is more agreeable to maintain such biological or chemical groupings as *bioentities* while representing these 'minor' changes in their information context with *states*.

As state information is often not available, most genomic and proteomic databases such as Entrez Gene [53], UniProt [7] or GO [30] are entity level databases. In our ontology a bioentity stores a set of external references mapping to these databases, and acts as a gateway to the external resources. Bioentities are further classified into 5 classes:

1. *DNA*: A DNA molecule or certain *region* on a DNA molecule. Typically DNA entity is used to model genes on a chromosome, although it can also be used for other DNA molecules such as viral DNA, or damaged DNA fragments. A DNA entity also has a set of products, used to identify transcription/translation relations between entities.

2. *RNA*: Similar to DNA, this entity is used for RNA molecules. It typically represents gene transcripts. It has an optional source for identifying its source gene, and a set of products for identifying its protein products. Different splice forms of an RNA are considered as different states of the same entity, although their protein products are considered as different entities.

3. *Protein*: Again, this entity is primarily used for protein products of genes, however it can also be used to represent short peptide sequences. A protein entity has an optional source entity, a DNA or RNA.

4. *Small Molecule*: Despite its name this entity is used for all other molecules, lipids, ions, carbohydrates etc. Unlike genetic (DNA, RNA and Protein) entities, entity-state concept for small molecules are somewhat weaker. It is hard to come up with rules defining what sets of molecules should be grouped under a single entity. Clearly cytoplasmic and extracellular Ca++ belong to the same entity, but is 6P-Glucose a state of glucose? Or are

ATP and ADP different states or different entities? We opt to model compartment changes, complex formations and non-covalent isomerizations and homomerizations as state changes. Covalent chemical modifications are considered as a transformation to a different entity.

5. *Physical Factor*: This rare class is reserved for entity-like environmental factors, such as radiation, pressure or heat, since they also act as an input to the signaling pathways.

5.3 Bioentity Interactions

For most high-throughput techniques such as microarray and Y2H, bioentities form the unit entries, and the results of the experiment defines a graph over them. In fact aforementioned gene and interaction networks, which are the major outputs of these methods are defined at this level. Bioentity interactions are used to capture such relations. In a sense they represent incomplete information, as a bioentity interaction always maps to one or more mechanistic level interactions, although latter one often is not identified/elucidated yet. There are four types of bioentity interactions:

1. *Protein protein interaction*: PPI is an undirected relation, indicating that two proteins are observed to interact with each other in a Y2H or co-precipitation system, i.e. there is at least one state of entity A that somehow interacts with B. One or more mechanistic level relations might be associated with this entity level relation. For example a state 1 of protein A might be bound by protein B, where state 2 of protein A might be bound and cleaved by B. Even the nature of the chemical reaction does not necessarily be same / similar. Compartment information and n-ary relations can not be captured by PPI. Some sample databases that contain PPI data include DIP, BIND and IntAct.

2. *Transcriptional Regulation*: TR is a directed relation, indicating that at least one state of source node activates/inhibits expression of at least one

DNA state of the target. Although there is combinatorial information on TR, we are yet to incorporate this to our ontology. Although current ontology can not capture the mechanism of the regulation, incorporation of operons in the future can improve this. Moreover one can always define complete mechanism at the entity level, if the information is available.

3. *Derived*: These edges represent that there is a transition in the mechanistic graph that is adjacent to at least one state of each bioentity. Depending on the exact semantics this edge might have sub-types. For example a control edge might indicate that source bioentity has a state that is an effector of a transition, from which a state of the target is produced.

4. *Generic*: This is an undirected or directed relation which does not fit into one of the previous three. Co-occurence graphs, where an edge between two bioentities indicate that they are referred significantly together in the literature is an example.

5.4 States

Typically in a cellular network, flow and control of information happens through modifications of molecules. Most of the time, it is easy to envision these changes as a state-transition, where an entity modulates another entity by switching it to a different state. The term is very generic and encapsulates macromolecules (e.g., DNAs, RNAs, and proteins), small molecules e.g., ions, ATP, and lipids), or even physical actors (e.g., heat, radiation, and mechanical stress). States also represent molecular complexes, or conceptual abstractions that behave like state. Depending on their nature, states are classified as either compound or simple.

5.4.1 Simple States

Simple states represent tangible and unit phenomena. Each state belong to a bioentity, and represents a change in the information context of the bioentity.

Those changes are represented with the following bioentity variables:

1. *Cellular Localization*: Each state has a compartment in the cell. Changes in compartment means a change in the molecules information context, since the set of molecules it can interact with changes. This property is single and mandatory for all states. Compartments are described later in this document.

2. *Complex Binding*: This property describes a functional change due to long-lived non-covalent bonds between molecules, e.g. p53 bound state of MDM2. This property is multiple and optional.

3. *Homomerization*, i.e. non covalent bonding of two or more of the same state is not considered as a complex formation for the sake of simplicity. Instead we use a separate property for modeling such a state. This property is multiple and optional

4. *Isomerization*: Conformational non-covalent changes within the molecule. This property is multiple and optional.

5. *Chemical Modification*: Cleavages, group additions/removals and other covalent changes are classified in this group. This property is multiple and optional.

Each bioentity variable tuple is formed of a class, a value and a description. Class is one of the variable classes above. Value is chosen from a limited vocabulary, possibly taken from other specific ontologies. Description is free text, however depending on other ontologies, we might also wish to constrain it. Following are some examples:

Defining bioentity variables more formally increases the content of the models, and helps a great deal to the equivalence by context problem. There is still further room for improvement by formalizing the information that was delegated to the free form description slot. This, however, again requires formalization of sequence and sequence position and have subtle complexities such as disulfide bonds which requires two position information to be formally defined.

Class	Value	Description
Cellular Loc.	Cytoplasm	–
Attachment	Cytoplasmic Membrane	Farnesyl attached
Chem. Mod.	Phosphorylation	At 155-Arg
Chem, Mod.	Point Mutation	2444 A-T
Isomerization	Allosteric	Active Enzyme
Homomerization	Trimer	-

Table 5.1: Examples of bioentity variable triples.

Any combination of bioentity variables forms a unique state of this bioentity. It should be noted that only a very small portion of the state space actually occurs in biological systems.

Important points when defining states are context and desired level of detail. One should keep in mind that states map to a class of molecules/entities rather than a single molecule. It might be that this group is not totally homogeneous. For example, it is not desirable for most of the cases to model the rotamers of a protein as different states, as there are combinatorially many of them, they are very short lived (in the range of nanoseconds) and switching from one of them to another is almost instantaneous. However, PATIKA ontology does not define hard lines for the level of abstraction, as it readily provides a framework for modeling and representing multiple levels of detail. So we can say that state variables can be incomplete and overlapping. An example of incomplete state variable is "phosphorylated p53". However, this representation poses a subtle problem. Since we do not know at which site the p53 is phosphorylated, relationship between this state and phosphorylated p53 at 153Arg is not clear. It might be that two authors actually talk about different states, or the latter is a non-proper subset of the first. A sensible approach is to delegate this issue to the submitter, and to the expert, as it is really hard to come up with a context free resolution rule. If the first is the case, than the submitter must modify the phosphorylated p53 entry to bring it into the correct level of detail.On the other hand, if it is the second case they must switch the p53 phosphorylated into an incomplete state to indicate different levels of detail. However, this does not solve all cases, submitter (rightfully) might not know whether it is the first or the second case. Or

he might know that it is the first case, however he might not have enough information to annotate p53 phosphorylated. This is a very difficult issue to handle, and currently unresolved. A formal hierarchy might seem more desirable to solve these problems but then again there might be different, conflicting hierarchies for states of an entity.

It is hard to come up with rules defining what sets of molecules should be grouped under a single entity, especially for small molecules. Clearly cytoplasmic and extracellular Ca++ belong to the same entity, but is 6P-Glucose a type of glucose? Or are ATP and ADP different states or different entities? A crude rule is to allow only compartment and attachment changes as states variables for small molecules. However, this point still needs to be elucidated.

An important side note is in pathway drawings, it is common represent different states as a single biological entity, even when the mechanistic detail is known. This is an oversimplification as different states can have very different and sometimes conflicting effects. Mapping such information to PATIKA graphs might not be trivial, as in most cases the mechanistic detail is unknown. PATIKA allows defining relations at both bioentity and state level to address these levels of detail and abstraction.

In some cases, a bioentity's states are also labeled with various semantic tags, such as active form of an enzyme or open/closed state of a channel protein. Another logical tag is aberrations. Two types of aberrations exist: sequence and structure aberrations (i.e. due to misfolding). Sequence aberrations can be chromosomal or point mutations as well as polymorphisms. We can define a more detailed sequence state variable annotation scheme using GO. Sequence aberrations can be traced both on DNA, RNA and protein, although it must originate on DNA. Structure aberrations can occur in RNA and Protein states and model changes due to aberrant folding. Most of those aberrations are non-specific, i.e. they are groupings of combinatorially many different molecules under a name. Logical variables are manifestations of physical variables in a certain context, and there are times where we do not have the exact physical variable, but can assess the logical state. Leaving out logicals would decrease our coverage in that sense.

However, we decided that modeling contexts would be very difficult at this level, and such data was often not admitted by the databases. Although these tags are rather unambiguous, there are also no standard control vocabularies. Currently it is best to capture such tags with the states name and description.

Another point is states can be ubiquitous, i.e., they participate in a significantly high number of reactions. Typical examples are small molecules such as ATP (Adenosine triphosphate), or water, which have generic and structural roles. Such states can be problematic from two aspects, first for querying purposes, one often wants to ignore these. For example, when querying for shortest path between two protein states that both get phosphorylated, there is a path of size 3 that passes over ATP. This path, however, is rather insignificant, as both events does not really change the ATP concentration in the cell, and there is no real information flow over this path. Second, molecules such as water can have very high degrees even for very small subgraphs, which makes them very difficult to visualize. So for both aspects, we would like to handle ubique states specially. Patika ontology uses ubique state class to describe simple states that exhibit such a behavior. Although we have identified compound states that also show ubique-like behavior (e.g. ribosome) for the current level we opt to ignore such cases. Another problem is in some contexts ubique state might have taken part in decision making mechanisms. For example for oxidative phosphorylation, ATP concentration might act as a regulator. However, it is very difficult to specify such contexts, and currently is not covered by the Patika ontology.

5.4.2 Compound States

A compound state is a grouping of other Patika objects, which exhibits a state-like behavior, and needs to be addressed at this level. There are two types of compound states, complex and abstraction.

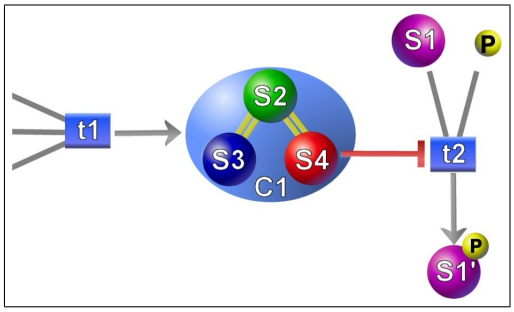

Figure 5.1: Representation of complex in PATIKA . Here C1 is a complex formed by states S2, S3 and S4. Binding relations are also represented. Transition T1, which represents the complex formation event adresses the complex, where the inhibition of t2 by S4 is an example of addressing complex members.

5.4.2.1 Complexes

In biological systems molecules often form clusters for performing proper tasks, behaving like a single state. We consider each member of a molecular complex as a new state of its biological entity. The function of a molecular complex is affected by the specific binding relations within itself. Therefore these binding relations must be represented in the model as well. Moreover, members of a molecular complex may independently participate in different transitions; thus one should be able to address each member individually. In addition, a molecular complex may contain members from multiple neighboring compartments. In that case, always one of those compartments is a member type compartment. It is actually possible to model complexes in a similar fashion to membrane spanning proteins.

Complex states have a set of simple state members which should be complex members.

Complex states do not have a bioentity, as they are not simple. However their members have their own bioentities. This information may be used for complexes as well, e.g. for querying. In a similar manner, each complex member has its own set of bioentity variables, including compartments. So it is possible to specify multi compartment spanning complexes with PATIKA ontology. Since each complex member is specified separately it is possible to address members specifically as activators or inhibitors of other reactions(Figure 5.1).

An important question is "Do we model short lived binding relations as complexes or activation relationships?". There might not be a concrete answer for that, however a best practice is *never use a compound graph unless you need to*, and this also applies to complexes. If an activation relation would be able to represent the current knowledge, it is best to use it. If that level of detail is not sufficient for another user, they can re-edit it to add a complex at that point.

Another important distinction is between a chemical modification and a complex. As a rule of thumb, long lived non-covalent relations are considered as complexes, whereas covalent bonds are considered as chemical modifications.

5.4.2.2 Abstraction States

Abstraction states are groupings of pathway elements that behave like a state. An example is Wnt protein, which in fact represents a set of homologous proteins with similar functions. Abstractions are not limited to states but to transitions and in fact is a cross cutting aspect in our ontology. Therefore we are describing the details of both abstractions later in this chapter.

5.5 Transitions

A cell is not a static entity, neither are its actors. Molecules in a cell are synthesized, modified, transported and degraded constantly to respond to the changes the environment, or to accomplish a task. One can model such changes as quantitative chemical reactions. However this would reduce the coverage of the model, as currently both molecular concentrations and rate constants for most of these reactions are unknown. It is often preferred to represent these changes qualitatively since this better suits existing knowledge in molecular biology. A transition has a set of states as its substrates (inputs) and products (outputs). A transition occurs only when all of its substrates are present and activation conditions are satisfied; a function of the certain other states. These states are called the effectors of a transition. Two types of effectors relations are defined, activator

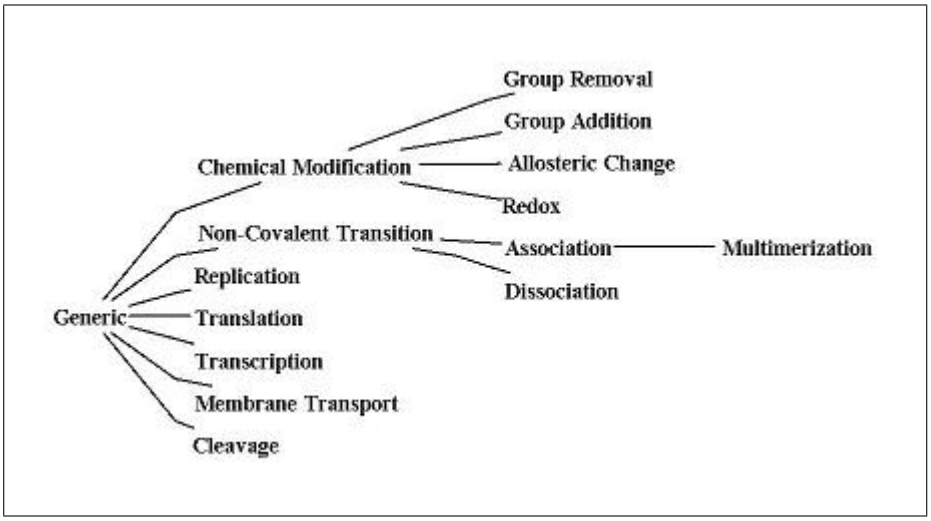

Figure 5.2: PATIKA transition tree decomposes transitions to several classes.

and inhibitor, for positive and negative regulation respectively. When a transition occurs, all of its products are generated. PATIKA uses a pragmatic approach for formally defining transitions: any event that changes one or more states to another set of states is a transition. This definition delegates the exact definition of transition to the exact definition of state, and as mentioned above, level of modeling detail for PATIKA states are very flexible. It follows that PATIKA ontology can model transitions at multiple levels, allowing high coverage, without losing from its content. Two transitions are equal if they have the same set of substrates and products. This reveals two invariants for transitions:

- A transition has at least one substrate and one product
- There cannot be two transitions with same set of substrates and products.

Although transitions can have a very large spectrum, we expect that most of them will fall to the certain classes. Those classes are captured by PATIKA transition tree (Figure 5.2).

5.6 Mechanistic Interactions

Mechanistic interactions define relations between states and transitions at the chemical level of detail. There are five types of them:

1. A substrate relation is a directed relation with a state as its source and a transition as its target, which indicates that the state is consumed by the transition. A substrate relation has a stoichiometry attribute which describe the number of its source states that are consumed per target transition. stoichiometry defaults to one.

2. A product relation is a directed relation with a transition as its source and a state as its target. It indicates that the state is produced by the transition. It also has a stoichiometry attribute to describe states produced per transition.

3. An activator relation is a directed relation with a state as its source and a transition as its target. It describes the enabling or facilitating of the transition via the source state. An activator relation may optionally be one of enzymatic, allosteric or abstract.

4. An inhibitor relation is a directed relation with a state as its source and a transition as its target. It describes the disabling or impeding of the transition via the source state. Irreversible inhibitions should be modeled as a separate state of the modified enzyme. An inhibition relation may optionally be one of competitive, non-competitive, allosteric, enzymatic or abstract.

5. A bind relation is an undirected relation with two complex members as their source and target. It describes a non-covalent bonding between these two states. If all binding relations were known for a complex, than the graph defined by binding relations and members would be connected.

There are cases where the stoichiometry might be unknown. Ron Caspi, a researcher at MetaCyc project[1] kindly provided the following use case at the BioPAX list;

> A different scenario is when the identity of all of the reactants is NOT known. There are cases when some intermediates in a pathway are identified, and a general pathway is suggested, but the exact nature of the reactions has not been figured out. In this case, you have reactions that do not balance, and would not balance because there may be additional reactants involved. In such cases the stoichiometry of the known reactants may be unknown. An example for this can be found in the MetaCyc pathway *indole-3-acetate degradation to anthranilate*.

Currently PATIKA ontology can not cover such incomplete information.

5.7 Abstractions

Network of molecular interactions derived from current biological data is incomplete and complicated. Different types of abstractions are necessary to make effective analysis of cellular processes and dealing with complexity better.

Currently abstractions are only defined in the mechanistic graph level. However, there are other groupings, mostly created by clustering and inference algorithms that produce groupings of bioentities. In order to capture such groupings we might wish to introduce bioentity level groupings.

5.7.1 Regular Abstractions

Pathway is a very common and ambiguous term, and can have several even conflicting meanings. In its broadest meaning it is a subgraph of the cellular process

[1] http://www.metacyc.org

network. Typically, content of a pathway might be effected by the researcher's point of view and target, experimental system and even historical reasons. We still make use of *abstraction* classes for capturing such groupings, because they can immensely help with complexity management. (It is much easier to query *regular abstraction named lysine catabolism* compared to figuring out *1 neighborhood of up-to-3 downstream of cytoplasmic lysine*). We model such groupings using regular abstractions. Regular abstractions can be arbitrarily nested and can intersect. However they can not be addressed directly, i.e. they have no incident edges.

5.7.2 Incomplete Abstractions

Since the data on cellular processes is not complete, different levels of information may be available for certain events. In cases where it is not identified which state among a set of states constitutes the substrate, product or effector of a transition, or where target transition of an effector is obscure, we may need to abstract these states (transitions) as a single state (transition) to represent the available information despite its incomplete nature. An edge defined on an incomplete state means that it is actually defined on at least one state inside but the exact state is not known. A similar semantic applies to incomplete transitions.

5.7.3 Homology Abstractions

In biological systems, a gene is often duplicated throughout its evolution serving a similar but different function. A special case occurs when this differentiation serves as a specialization of a generic mechanism. For example the term WNT gene, actually represents nineteen various similar genes in human [58]. These genes are all activated by different stimulus at different tissues and can lead to different responses even though the signal processing mechanism is similar. Bhalla also describes common process motifs in signaling pathways, which are even more elementary operations that are reused through the entire network [8]. Our ontology supports representation of such homologies using abstractions. Homology

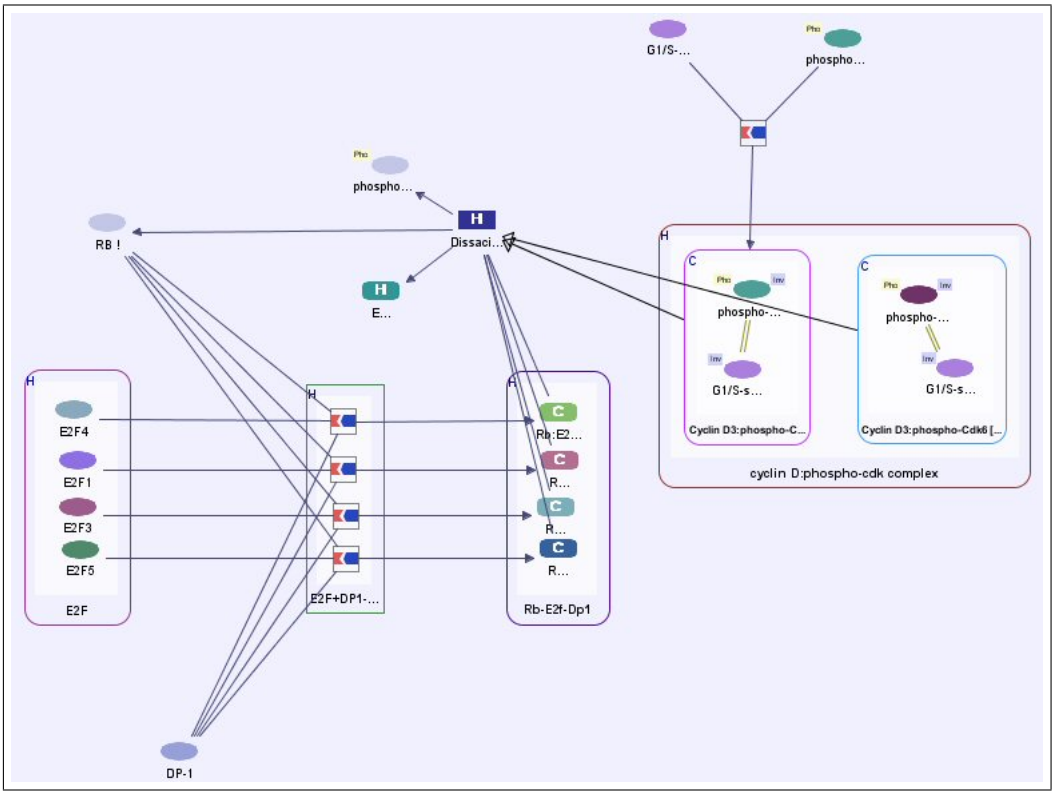

Figure 5.3: An example portion of cell cycle pathways containing homologies

abstractions are also divided into homology states and homology transitions.

Exact semantics of homology abstractions can be slightly ambiguous, below is a more formal definition to clarify this issue: We will define a Homology State simply as a set of member states. For our purpose, it is sufficient to define a transition as T(S,P,A,I) where S is a set of states acting as substrates, P for products etc. A homology transition is defined as $HT(S_H, S_S, P_H, P_H, A_H, A_H, I_H, I_S, M)$. S_S, P_S are sets of simple states where S_H, and P_H 's members are homology states. M is a set of transitions, that belong to this homology. If $m(S^m, P^m, A^m, I^m) \in M$, then $S^m = S_S \cup N_H$ where N_H is a set that satisfies the following:

For every valid N_H there is a set of ordered pairs $\sigma = \{(h,n)|h \in S_H, n \in N_H, n \in h$ and $\forall h \in S_H$ there is exactly one $(h,n) \in \sigma\}$.

P^m is also defined similarly. For A^m and I^m we modify the definition of N_H to include at least one (instead of exactly one) pair (h,n).

Above definition means that a homology transition is like a transition template. To create your own instance of this template, you have to insert your states of choice, to the given slots. Some slots are invariable, and defined by simple states, other slots :

- If a simple state is a S/P/A/I to a homology transition, it is a S/P/A/I to all of its members.
- If a homology state is a S/P to a homology transition, then for every member of the homology transition, there is exactly one member of the homology state, acting as a S/P.
- If a homology state is a A/I to a homology transition, then for every member of the homology transition, there is at least one member of the homology state, acting as an A/I.

This is a hard invariant to check at runtime dynamically, instead we might want to modify the ontology in the future, such that the user actually specifies a mapping.

An important invariant not included in the definition above is that HSs can only interact with HTs. This is due to the fact that HS is an abstraction, and a reaction that has an abstract input/output is also abstract. Reverse is not true, an HT can perfectly interact with a simple state.

Another related invariant is, an HT must have at least one interacting HS. Since biologically there is no concept of reaction homology (events does not evolve, entities does), it actually is a group of reactions with homologous S/P/A/I.

A subtle concern is if an HT has two different homology S/P, from the same homology state (e.g. dimerization of different members within a protein family), then our exactly one constraint becomes exactly two. A similar scenario might occur if two HSs are overlapping, and are S/P to the same homology transition.

Note that the above definition does not remove HS-HT edges. It may be argued that HS-HT edges can always be inferred by finding HSs that satisfy

the mapping. However, the concern above makes inferring these a rather hard problem, so using them explicitly and checking invariants makes a lot more sense.

Another issue that was not covered is homology complexes. Several ontologies, including Reactome allow generic states as complex members. When more then one such generic exist within a complex, this implies that every possible complex combination in fact exists. This is rather a dangerous implication, as users will often use a generic - higher granularity, introducing fictional complexes absent in-vivo. PATIKA does not allow this, but also has no way to nicely represent combinatorial complexes

5.8 Cell Model

While modeling the events in the cell we can not ignore the cellular structure that these events take place. In fact cellular locations have major effects on many cellular events. As the compartments and their adjacencies are cell type dependent, compartmental structure should be modeled as part of the ontology.

We consider a cell as a closed sack. If we think of a sack as a membrane structure each sack encapsulates a space and each space contains other sacks (membrane bound organelles), where specific cellular events can take place. For example cytoplasmic membrane is a sack and encapsulates cytoplasmic space. Cytoplasmic space in turn contains other membrane bound organelles such as mitochondria, nucleus etc. These inclusion relations can be a few levels of depth. Neighborhood relations can be determined by these inclusion relations in our model. There are some exceptions in this model where the model does not fit to the real cell. For example ER (Endoplasmic reticulum) does not have a closed sack structure instead it is like tube extending from nuclear membrane to the extracellular matrix. We model extra neighborhood relations as in ER example (ER-nuclear membrane ER-extracellular matrix neighborhoods) independent from the inclusion model.

For different cell types different cell models can be prepared using ontology

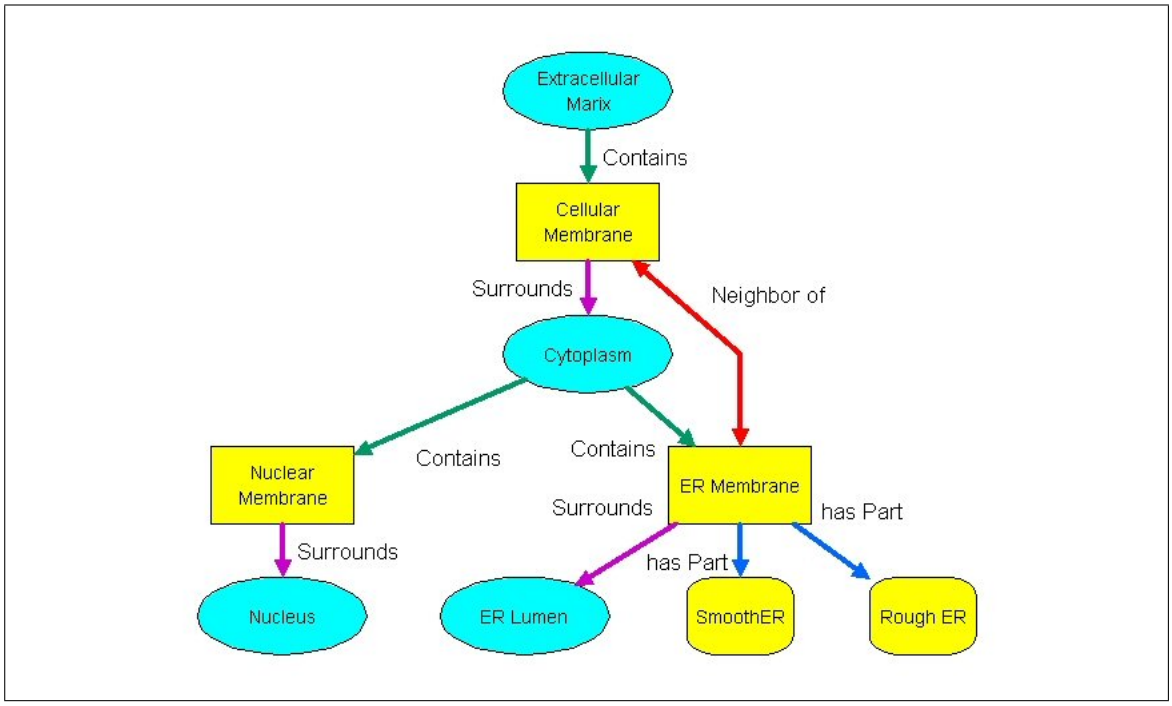

Figure 5.4: An example of cell model relations. Circles are spaces, squares are membranes and rounded rectangles are subregions. Different inter region relations are also shown.

explained above. This cell model can be used as an underlying cell model of PATIKA*pro*. Cell models can be prepared not only for different cell types, but also to examine compartments at different levels of detail. An example of relations are given in Figure 5.4.

Membranes pose an additional problem since not only a molecule may be located completely inside the membrane but also it may span one or both of its neighboring compartments. For membranes there are four types of sub-locations, inner and outer side of the membrane, inside membrane and spanning membrane.

5.9 Formal Definition

PATIKA ontology was partially defined in [21] using a graph notation. However, it is quite cumbersome to define and extend constraints and other relations using

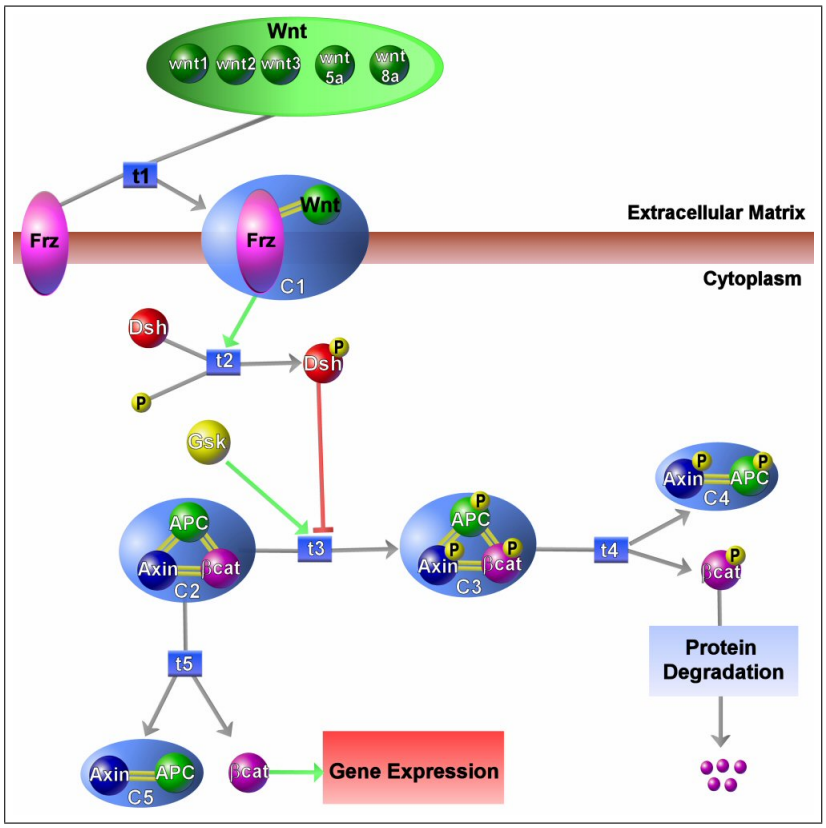

Figure 5.5: A representation of a portion of a Wnt pathway with the PATIKA ontology. Three regions are shown, Extracellular Matrix, cytoplasm and cytoplasmic membrane. Wnt is a homology abstraction containing different Wnts, which are simple states themselves. Frz is also homology state and represents a family of receptors that are important in differentiation during development. C1 is a complex of Wnt and Frz proteins. Note that members can have different compartments. C2-C5 represents different complexes formed by APC, Axin and beta-Catenin, proteins that are also involved in development. Two downstream pathways of protein degradation and gene expression were shown with regular abstractions.

this notation. So far most of these extra constraints were defined informally as internal documentation or software invariants. This makes it difficult to track and communicate such constraints. In order to improve this, we have described our ontology in Web Ontology Language (OWL). OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language[2].OWL is gaining support in many communities, including biology and chemistry as the *de facto* ontology language. Current OWL ontology can be obtained from PATIKA web site[3]. There are numerous tools for visualizing and analyzing OWL. Among them Protègè provides a very friendly and visual environment, and can be obtained from Protègè home page[4]. There are differences between the way PATIKA is implemented compared to its ontology, mostly due to programming constraints, however, these are minor, and does not compromise with the overall semantics of the ontology.

The short version of the ontology is given in Appendix A. By short, we mean that the controlled vocabularies, such as post translational modification types, and actual compartment structure is not included. In PATIKA, these are externalized into separate XML files.

5.10 Open Issues

Following properties are missing from the current ontology, but were considered at some point and left out for future versions.

[2]http://www.w3.org/2001/sw/WebOnt/
[3]http://www.patika.org/ontology/patikapro.owl
[4]http://protege.stanford.edu

5.10.1 Generics

Frequently multiple molecules are grouped and referred as a single actor in literature. For example there are groupings of states that are formed through polymerization such as glycogen. Although the structure of the *complete* glycogen is single, virtually infinite sub-structures occur during its metabolism. It is an example of generic states by polymerization, also applicable to other polysaccharides, cytoskeleton proteins (i.e. tubulin) and fatty acids. Their instances can not be (feasibly) enumerated. It is worth to note that sometimes entities of the states are also generic. For example in the example of removing a glucose from a glycogen, we can not represent all poosible glycogen molecules with different number of glucoses feasibly, yet representing both left and right side glycogen with the same entity is also wrong and breaks quite a number of invariants, including the statement that a molecule can not be a both substrate and product of a reaction. Although semantically we can perceive that, this is nowhere in the ontology, so we can not claim that we are modeling glycogen metabolism unambiguously.

Another example is damaged DNA with 3' incision. This is a classic example used in DNA repair pathways. It is worthy to note that one really can not hope to model DNA repair pathways, without somehow modeling damaged generic DNA.

Another classical example is combinatorial recombination for diverse immune response. This is a very special example however one also needs to consider the extensions of this generic behavior, like major histocompatibility complex, epitopes etc.

Yet again there are proteins which can get semi-quantitatively phosphorylated. The tail of the neurofilament heavy subunit contains an amino acid motif repeated 44-45 times [2]. Each repeat can get phosphorylated, which then regulates axonal caliber, with interfilament spacing determined by phosphorylation of the motifs. This results in potentially 2^{45} different proteins.

Generic states can form complexes in such a manner that they create combinatorially many species, even though species of states themselves can feasibly

be enumerated. Initial events in EGFR signaling, accounting for Sos and PLCg activation can generate as much as 5000 states [16].

How to model generics is still an open and hotly debated issue in the current pathway modeling community.

5.10.2 Modulation

PATIKA ontology does not cover some aspects of control effector molecules can exert on a transition. Specification of inhibitors and activators of a transition does not necessarily establish an exact activation condition. Currently we assume that any combination of effectors can regulate a transition. This might not be the case, for example two inhibitors may never be present together in the cell, or when two inhibitors are present they cancel out each other . An approach is to use extra objects for modeling such modulation logic. If it turns out that actually only a small number of all combinations of effectors are significant, another possible approach is to use the already existing compound graph notion to include children nodes into the transition for all significant combination sets, in order to be able to address them separately. The term modulation coins a rather wide spectrum. In modeling modulation we can use boolean predicates, linear equations, stochastic models, pi-calculus etc. [36, 79, 74, 69]. PATIKA ontology assumes that the representation and stoichiometry of the transition is independent of the transition logic. This is a choice made to increase coverage since vigorously modeling activation condition and substrate concentrations require a linear (and possibly stochastic) set of equations, which are unknown for most signaling pathways. Our primary aim is to build a framework, albeit not precise, with the available biological data. However, it would still be possible to add simulation support, at the software level by using a pluggable interface to a simulation engine. Our ontology would then serve to intuitively represent and investigate a model, where the simulation engine would be used for functional computations.

5.10.3 Exhaustive relations

Another related aspect is under certain circumstances, multiple transitions having the same state as a substrate may affect each other through depleting this common substrate. This happens when the equilibrium constant of a transition is relatively much higher than the others. There are a bunch of control mechanisms, often involving activation of an enzyme such that a certain constant is lowered, that uses this depletion mechanism. If such a difference occurs among the equilibrium constants of transitions, we call the transition with the higher equilibrium constant exhaustive over other transitions for the common substrate. Transitions having the same order of equilibrium constant, on the other hand, are said to be cooperative.These relations currently can not be represented properly with PATIKA ontology, as PATIKA ontology totally forgoes quantitative aspects in favor of coverage. A simple *exhaustive over* edge is not simple sufficient, as there can be as high as n^2 such relations where n is the degree of product edges incident on this state, and other factors such as enzymes can change those constants(again here the assumption is system is *not* in equilibrium.

5.10.4 Reversible Transitions

In theory, all chemical reactions within a well-stirred medium is reversible,i.e. depending on the chemical equilibrium they can produce their substrates from their products or vice versa. However a biological system is hardly a balanced, well stirred medium (in fact more it gets close to a balanced, well stirred medium, less alive it is). In PATIKA all transitions are models as one-way reactions. If a reaction is reversible, then it is modeled as two transitions, where one's product set is other's substrate set and vice versa. These two transitions are called inverse of each other. However this representation has one obvious problem, one can annotate one part of the reversible reaction and forget to annotate other, leading to inconsistencies in the data. It is much more preferable to have both transitions as a single entity.

5.10.5 Context

Contents of a complete network of pathways may be classified according to varying fields of studies such as apoptosis, lipid metabolism, cell cycle, etc. Similar classification may be performed based on tissue or phase specific processes. Looking at such an entire, complex network from the point of specific interest fields, tissues, or phases of cellular processes would simplify the understanding of the network by filtering out the undesired parts. Fields, tissues and phases can also be modeled as bioentity/state level abstractions, but they can be very large and impractical to use. States and transitions can occur in many different cell types and contexts. Therefore these groupings are not isolated.

Phases, which describe a partial ordering and co-occurrence are even more problematic, as they can overlap and include each other, creating combinatorially many intervals, and each of these phases might be specific to each tissue. More importantly cells change their type through differentiation so a phase would also ideally contain such changes. Clearly, a full coverage of this issue in the near future is very unlikely.

5.10.6 Chromosome Structure

Genes are not separate molecules, they are a subsequence of a chromosome. How genes are ordered and located is important as it affects several important phenomena, including their activation and inheritance. Also on a chromosome there are other entities like operons or activator sequences, which behave very like stateful bioentities. They do not simply belong to the bioentity of the gene they regulate, since they can took part in regulation of multiple genes. Other chromosomal features include repeating subsequences, or regions where the frequency of Adenine and Thymine is high. To add further complexity the chromosome is physically non-uniform, some portions are densely folded, effectively turning off genes in that regions, whereas some of them are exposed to increase their expression rate. These features and relations currently can not be expressed with Patika ontology.

Chapter 6

Ontology Implementation

This chapter describes the design and implementation of PATIKA ontology in PATIKA*pro* and PATIKA*web*.

6.1 Model Layer

Model layer defines first class objects as interfaces, allowing a greater flexibility for its implementors. We assume that the reader already has an acquaintance with the ontology so we do not further explain its concepts, unless an implementation specific explanation is required. The implementation was done in Java 1.4 and most of the time clearly parallel to owl description, which is also not detailed here.

An overview graph of first class objects are given in Figure 6.1. Since abstractions are cross-cutting concerns they were implemented with multiple inheritance.

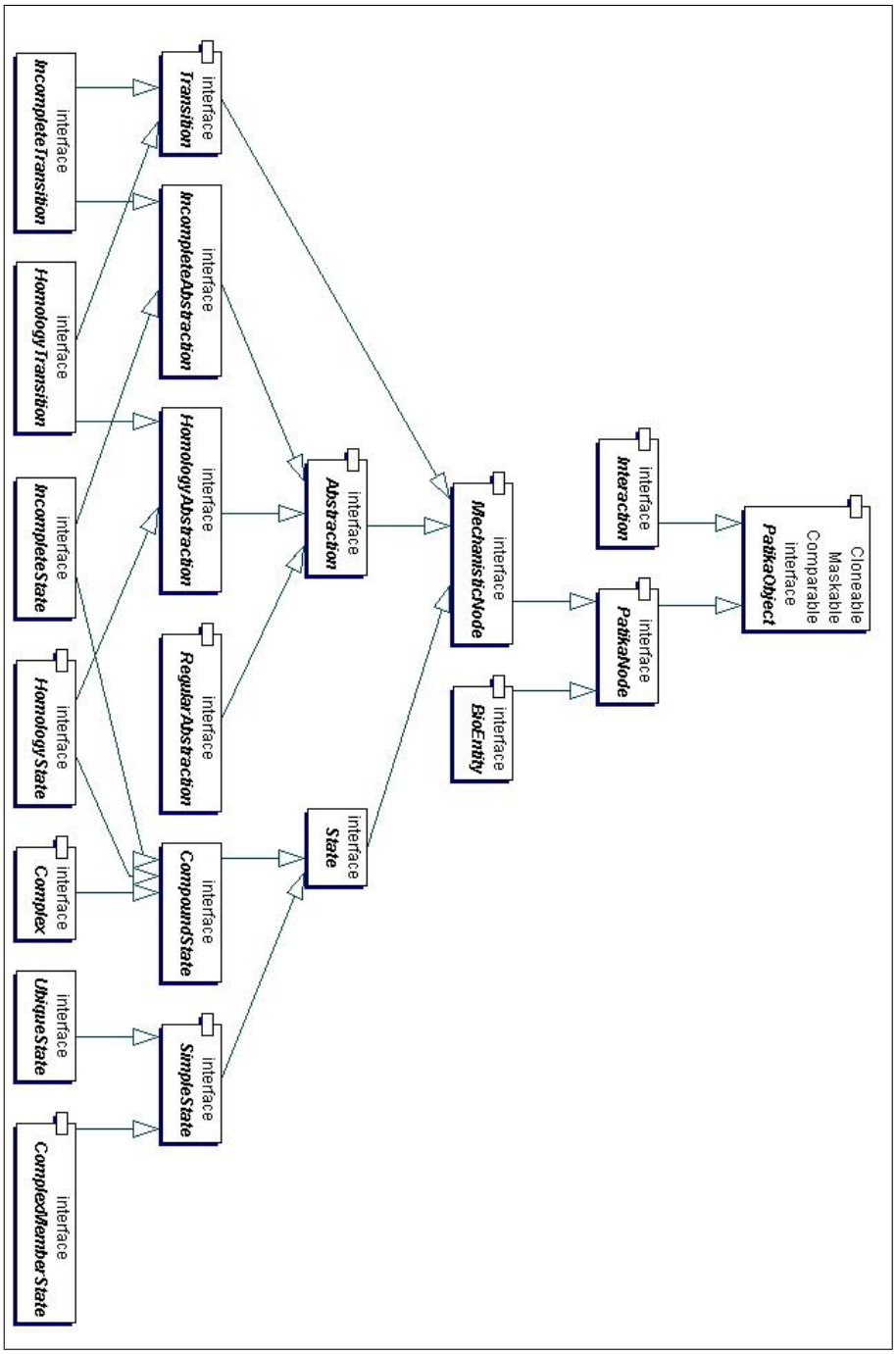

Figure 6.1: Class hierarchy of the primary PATIKA objects.

6.2 Concrete Implementations

There are three concrete model implementations, DB (Database) level, S (Subject) level and V (View) level.

6.2.1 DB Level

The server side employs an MVC framework and DB level acts as the model layer, providing the Patika Model interface which is used for manipulating data. DB level is also a DAO layer hiding persistence related details from the user, and provides the same consistent Patika Model interface. The DB level relies on an in house graph implementation and provides persistence and querying logic.

6.2.2 S Level

The S level relies on Tom Sawyer Software's[1] graph libraries for defining an abstract Patika graph. S level is a model layer and contains only topology of the graph and the related data. In a sense, S level acts as a *cached* subgraph of the database and a temporary storage for user created objects and user modifications.

6.2.3 V Level

V level defines a compound graph which again relies on Tom Sawyer Software's graph libraries. V level is a view layer and contains all the drawing information. However each manipulation that is made to the model is delegated to the S layer, which in turn updates views accordingly.

V level provides extra facilities for managing the visualization such as collapsing compound nodes, fetching and merging more objects from the subject graph

[1] http://www.tomsawyer.com

and laying out external data on them such as expression levels from a microarray experiment.

6.3 Common Properties and Patterns

6.3.1 Info objects

Defining the model at the interface level is preferable, as it allows greater flexibility, but has a drawback. Logic and data common to all PATIKA objects need to be duplicated at each level, since Java does not allow multiple inheritance. To work around this problem we chose to use so called *info objects*.

Each PATIKA interface has an associated info class, which is responsible for containing data and logic that is independent of the implementation. Thus implementors of the interface use the same info class and delegate the calls to the info. For example bioentity of a simple state is kept at the SimpleStateInfo and instance of both DBSimpleState and SSimpleState delegate calls to their getBioEntity and setBioEntity methods to the corresponding methods of their associated SimpleStateInfo instances. This way only the delegation code is duplicated. A hierarchy of info objects are depicted in Figure 6.2.

6.3.2 PATIKA Factory

More often we want to abstract implementation of the model objects from their creation logic. For example XML convertor should be able to convert XML to any model implementation. In order to achieve this goal, all creation logic of the PATIKA objects are abstracted in the PatikaFactory interface. PatikaFactory contains methods in the form createXXX (creates an object with a given id and persistence flags, should be used when the object was already created previously e.g. while reading from XML) and createNewXXX (creates an object with a new id and modified=true, removed=false, orphan=false values, should be used

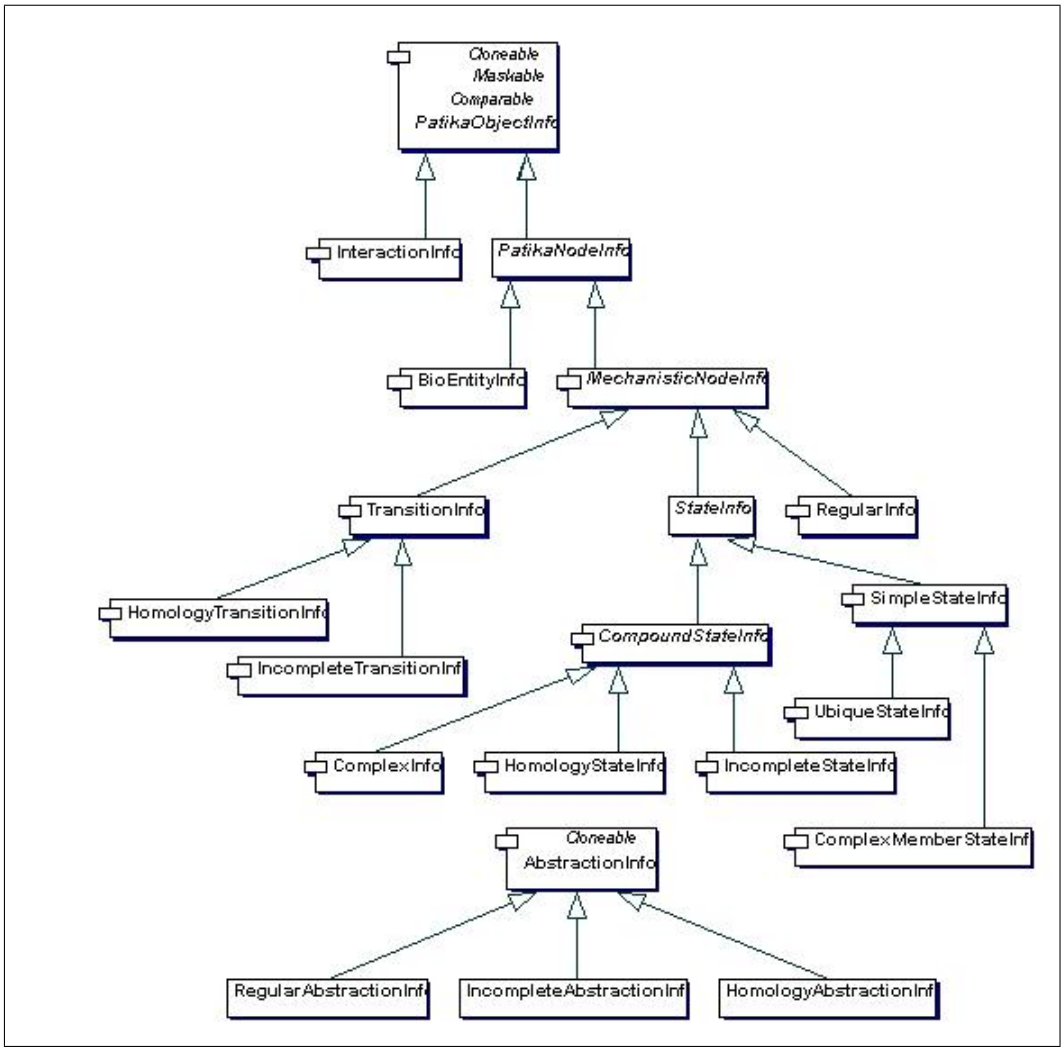

Figure 6.2: Class hierarchy of info objects

when an object is newly created, e.g. by wizards), and its implementors create corresponding objects. For example createSimpleState methods in DBPatikaFactory and SPatikaFactory creates a DBSimpleState and SSimpleState respectively. Returning back to our initial use case, setting a correct factory to the XML convertor delegates all creation calls to the factory, thus creates the correct level of objects. All model implementations should contain a respective factory.

6.3.3 Abstraction Info

Abstraction hierarchy tree overlaps with state/transition hierarchy tree as a homology abstraction can be a state or a transition. Same goes for the incomplete abstraction. Multiple inheritance does not pose an additional problem at the interface level but concrete implementors should implement state/transition tree as the main inheritance tree and implement abstraction specific calls at the leafs.

Abstraction implementations should use AbstractionInfo directly or by extending it to delegate common logic to all abstractions in order to avoid code duplication across abstractions. Although AbstractionInfo uses the same pattern with PatikaObjectInfo they are separate classes. Thus a HomologyStateInfo has two info objects both a StateInfo and a HomologyInfo.

6.4 Services

So far we have covered how the data was modeled within the system and the general layer design. However there are other services which are needed to satisfy the requirements that were previously listed. Other than the data specified in OWL and its accessor methods, model implementations provide several extra services.

6.4.1 Validation

Although OWL also provides its own validation facilities, some PATIKA constraints can not be specified by OWL, and also it is quite impractical to use OWL at performance critical points such as the server side. Therefore all model implementations provide a *check* method to check model invariants and model validity specified by the ontology. Model invariants are constraints that needs to be satisfied between any two atomic edit operations, e.g. every edge must have a source and a target node. Validity constraints are those that needs to be satisfied before a user's changes are submitted to the database. Check methods return an error log when they find a constraint violation, which can be used for debugging, user-feedback and logging.

6.4.2 Graph Traversal

PATIKA nodes support an interface for some graph traversal operations, like neighborhood fetching. Definition of adjacency can depend on the user preferences. For example, a user may or may not want to traverse over the members of a reached homology state for a certain shortest path query. PATIKA model implements a visitor pattern for abstracting such traversal options from the actual query algorithms such as shortest path. Each PATIKA node provide methods for specifying traversal policy and for providing lists of adjacent nodes, owner abstractions, equivalent states etc. upon being visited by an algorithm.

6.4.3 Field Querying

Patika Graph interface provides two methods for querying its contents. *findByPid* method returns the PatikaObject with the given PID. *findByField* method provides a facility for retrieving the contents of the graph via an SQL like query. At different levels different query implementations exist. DB level converts the query into Hibernate queries where S level sports a simple querying system of its own.

Once a query result is obtained as a set of Patika Objects, one often wants to find the minimal valid subgraph that contains all the query results. This operation is called *excising* and is also provided by model implementations. PATIKA nodes support an interface for some graph traversal operations, like neighborhood fetching. Algorithm classes are highly encouraged to use this interface when they need those operations.

6.4.4 Graph traversal

How to traverse a PATIKA graph depends on the user preferences. For example when you reach a homology state you may want the query to assume that it also reached all member states of this homology state. Or you may not want this behavior. One can specify these traversal options, with different flags:

- linkComplexToMember
- linkMemberToComplex
- linkMembersOfComplex
- linkHomologyStateToMember
- linkMemberToHomologyState
- linkMembersOfHomologyState
- linkHomologyTransitionToMember
- linkMemberToHomologyTransition
- linkMembersOfHomologyTransition
- linkIncompleteStateToMember
- linkMemberToIncompleteState
- linkMembersOfIncompleteState

- linkIncompleteTransitionToMember
- linkMemberToIncompleteTransition
- linkMembersOfIncompleteTransition
- traverseUbique

Each link may be considered as edges for traversal whose distance is 0. For example if linkMembersOfIncompleteState is true, then the traversal behaves if there is an edge between members of an incomplete state, whose distance is 0. So, reaching any of them means reaching all of them. The last option, traverseUbique, determines if the ubique states will be traversed or not. If not, then there would be no neighbor of a ubique to traverse.

At the general node level, just a simple neighborhood operation is supported, that do not consider any traversal option. The method *getNeighbors* is supported by any PATIKA Node, with a direction parameter which can be upstream, donwstream or both. Bond edges are not considered and product edges are always traversed if encountered. Other types of edges are subject to edge type parameter. This may be a specific type (e.g. substrate) or general type (e.g. mechanistic interaction). The neighbors are collected into the parameter collection, result. If null is sent, then a new collection is created by the method and returned.

At the mechanistic node level, one may want to find all owner abstractions that explicitly or implicitly contain this node, i.e. owner abstractions and their owner abstractions and so on. Method *getParentAbstractionsRecursive* supports this. Only abstractions that match the abstraction type parameter are considered and further navigated, similar to edge type parameter.

Using the traversal options, one can ask the equivalent states of a state or a transition, i.e. states/transitions that have 0 distance edges from this state/transition. This is provided by the *getEquivalentStates* and *getEquivalentTransitions* respectively.

6.4.5 Integration Support

All PATIKA Objects provide several flags and services that are needed by the submission subsystem.

- *Modified:* This flag is used to track whether the object was modified by the user. Depending on the factory method that was used to create the object (false for createXXX methods, true for createNewXXX methods) the flag is set to the proper value. At the client side this flag is set to true, whenever the user performed an edit on the object. Furthermore to avoid accidental edits, the user is warned and asked to confirm their edit operation. This flag is set to false whenever the object was updated from the database (clean copy). On the server side this object is set to true only as a result of submission, and set to false whenever the submission is successful and the object is updated in the database and its version increased. This way the submission manager can cleverly minimize the work done for submission checks, as there is no need to check unmodified objects or commit them to the database.
- *Removed:* Whenever a user wants to remove an object from the database, it schedules it for removal. On the software side this is tracked by the removed flag. Upon a successful submission these objects are removed from the database.

6.4.6 Excision support

Not all subgraphs of a PATIKA graph is valid. Some PATIKA objects depend on other objects for being valid, the latter being called a prerequisite of the former. Some example dependency relations are:

  - All interactions must have their sources and targets in the view, and if both its source and target is in the view, so must the interaction.
  - Each transition must have all of its substrates and products in the view. Although effectors are optional. A transition with missing substrates

and products is wrong, in the sense that it clearly violated chemical paradigm. On the other hand leaving out effectors makes it simply *partial*.

- All states must have their bioentity in the graph.
- All complexes must have their complexes member states in the graph and *vice versa*.
- All abstractions must have their members in the graph. The reverse does not hold.

All Patika objects know and can provide a list of their prerequisites.

Chapter 7

System Implementation

This chapter gives an overview of the PATIKA tools. It then proceeds to explain design and implementation of several components, with the focus being on components that were based on or complimenting the ontology. Finally we discuss how these components in turn affected the ontology itself.

7.1 System Overview

PATIKA software contains a server side component which provides web services for persistence, querying and integration and two clients for serving different use cases. All clients talk with the server using the same XML based interface over HTPP. PATIKA*pro* is the heavy client, which is a Java application aimed at users whose primary use case is to edit or extensively analyze the data. On the other hand, PATIKA*web* is targeted for users who are more interested in read-only access to the database for rapid knowledge-acquisition.

7.1.1 PATIKA Server

A component diagram for PATIKA Server was given in Figure 7.1.

Underlying container is Tomcat[1], although server can also be configured to run on a J2EE server and a JTA datasource, if a clustered environment is needed.

Hibernate[2] is the current ORM tool, but it is pretty much isolated using a DAO pattern7.2, so it should be possible to migrate to any other ORM easily, including JDO and iBatis, or simply JDBC. Hibernate is a powerful, high performance object/relational persistence and query service for Java. Hibernate allows developing persistent objects using plain old Java classes and relations - including association, inheritance, polymorphism, composition and the Java collections framework.

Spring[3] is a layered Java/J2EE framework, providing several commonly occuring structures in J2EE servers. Spring framework is used for three things: Implementing the IoC pattern for a modular server design, a flexible MVC and managing and isolating Hibernate. Spring layers and how they interact with other server components is given in Figure 7.3.

Of particular importance is how the transaction demarcation and session structure is handled with this architecture. Most of the low-level short-term concurrency is handled by the underlying database. It is database's responsibility to ensure that two concurrent transactions are serializable; i.e. their outcomes are same as if they were not concurrent. This transaction level logic is pretty much abstract and well thought. However RDMS does not give us any guarantee on the order of the serial schedule. Thus it is our responsibility to ensure that work between transactions does not depend on each other. As for transaction handling, each method that needs to be in a transaction context is defined declaratively in a Spring context XML definition. A typical declaration looks like

```
<prop key="get*">PROPAGATION_MANDATORY, readOnly</prop>
```

which defines that all methods of this class starting with get must always be called from a transactional context, and can not modify the DB. Any violation of

[1]http://www.jakarta.org/tomcat
[2]http://www.hibernate.org
[3]http://www.springframework.org

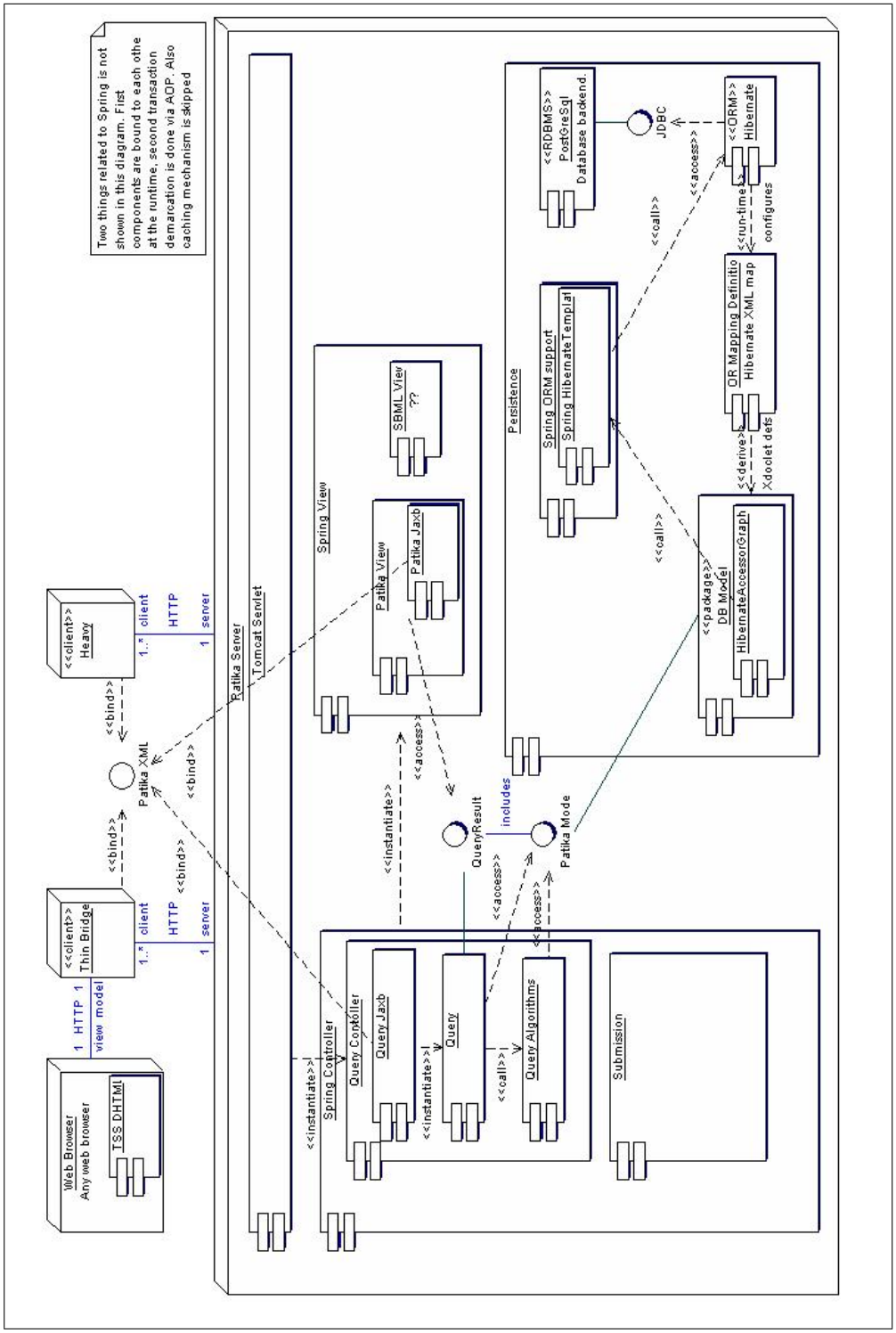

Figure 7.1: Major server side components and their deployment

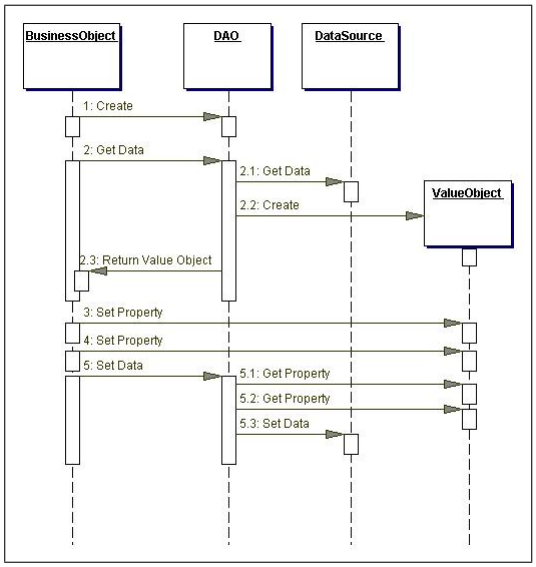

Figure 7.2: DAO pattern allows decoupling business logic from the persistence aspects

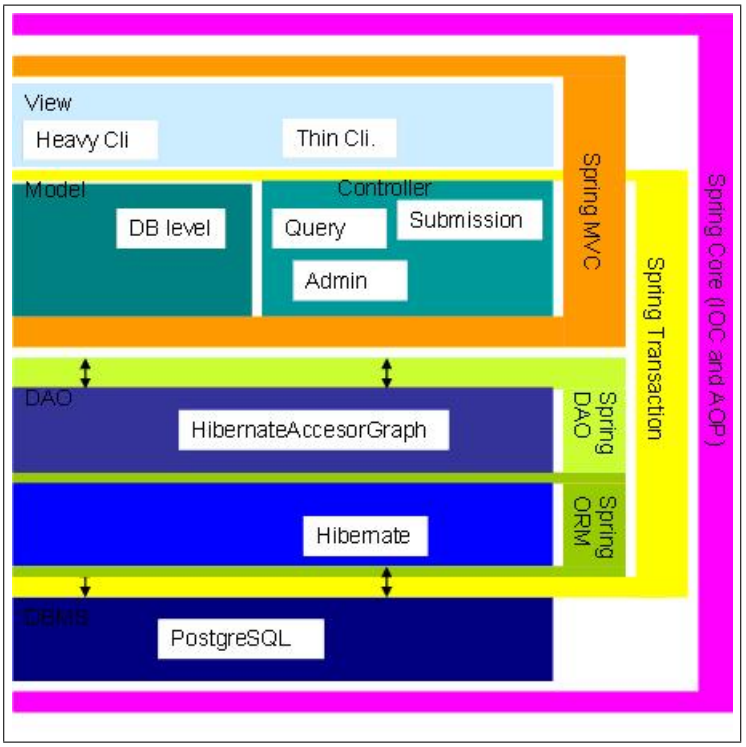

Figure 7.3: Server components within spring framework. Cross cutting concerns, such as transaction damarcation is done via AOP.

these constraints results in a run-time exception. These are implemented using run time class enhancement based on AOP Alliance interface. More interesting things occur when we use a declaration like

```
<prop key="query*">PROPAGATION_REQUIRED,readOnly</prop>
```

which means that if this method is called from a non-transactional context, Spring first opens a new transaction, otherwise it uses the existing transaction. Transactions are closed when the opening method is closed and comitted. You can throw certain exceptions to rollback and cancel the transaction.

Accessor graph methods are propagation mandatory, which means that they do not open and close the transactions, but require to be called from within a transaction context. On the other hand QueryController's query method is propagation required. Each call to query method is an abstraction of Hibernate session, which in turn is an abstraction of a JDBC transaction. Accessor graph serves as the viewpoint of the underlying graph. A user can perform a large number of operations on the accessor graph. Finally the user returns the query method which closes and commits the transaction. This whole process is a single transaction from database point of view and therefore atomic.

Handling of session cache, versioning and similar aspects are handled using this session structure and similar declarative AOP based strategies. This way persistence related code is minimized in the implementation. For example a graph traversal algorithm is unaware of the fact that it is actually opening a transaction when first accessing the database and closing it after the algorithm has returned. Similarly accessing the neighbors of a node is done transparently, where Hibernate tracks database status and runs SQL queries for bringing up objects that are not currently in the cache from the database. All mapping definitions with the only exception of implemented bean interface is specified in separate xml files, decoupling the business logic from persistence.

Current Patika database contains 62568 records, which was obtained by integrating several data sources including Entrez-Gene [53], UniProt [7], HPRD [66]

and Reactome [39].

7.1.2 Clients

PATIKA has two different clients, PATIKA*web* and PATIKA*pro*. Overall architecture of these clients are fairly similar, the only difference in their *modus operandi* is the PATIKA*web* has a three-tier architecture, where the most editor operations are done on the so-called *bridge*, whereas PATIKA*pro* sports a plain two-tier architecture, where all editor operations are done a heavy Java application. Mostly due to keep the client thin and high performant, PATIKA*web* provides only read-access to the database and does not allow users to modify queried models.

Clients provide full graph editor functionality such as do/undo, zoom, move etc. On top of it boasts a multiple view framework where different subgraphs of underlying subject graph is visualized in different windows, allowing user to effectively manage and compare their view ports. An automated layout algorithm [23] which can handle both compartment constraints and compound graphs nicely is provided to effectively visualize these views. Finally user can associate the given graph with external data such as a microarray experiment. Example screen shots from both PATIKA*pro* and PATIKA*web*is provided in Figure 7.4 and 7.5.

The architecture of the clients are quite complicated and are beyond the scope of this thesis.

7.2 Query subsystem

Querying component of PATIKA*pro* aims to provide users with a strong yet easy to use tool for retrieving and analyzing pathway graphs. Queries in PATIKA can be very complicated and have their little subsystem worth mentioning,

In the query scenario server receives an http request to the query URL. Servlet

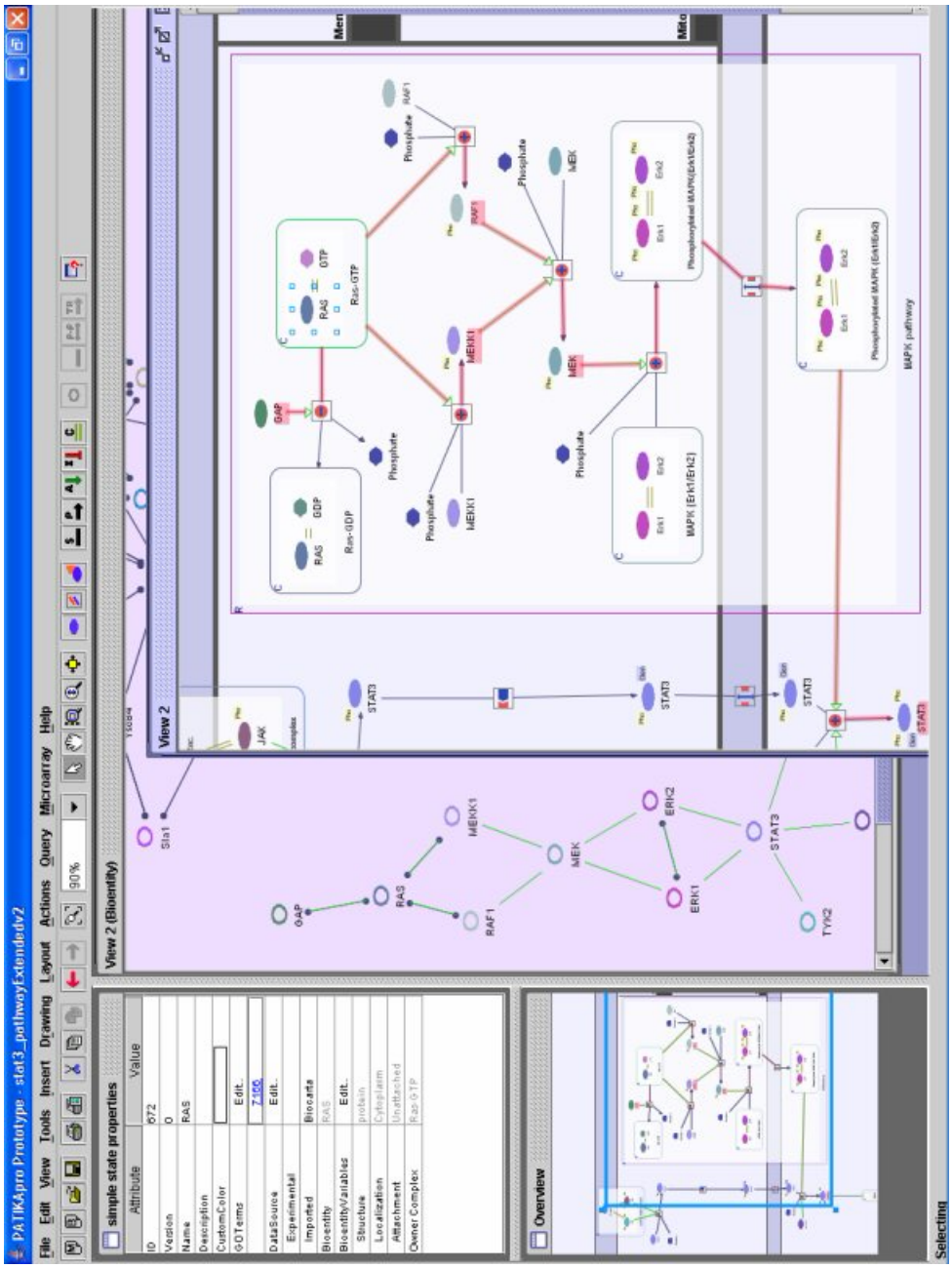

Figure 7.4: A screenshot of PATIKA*pro*.

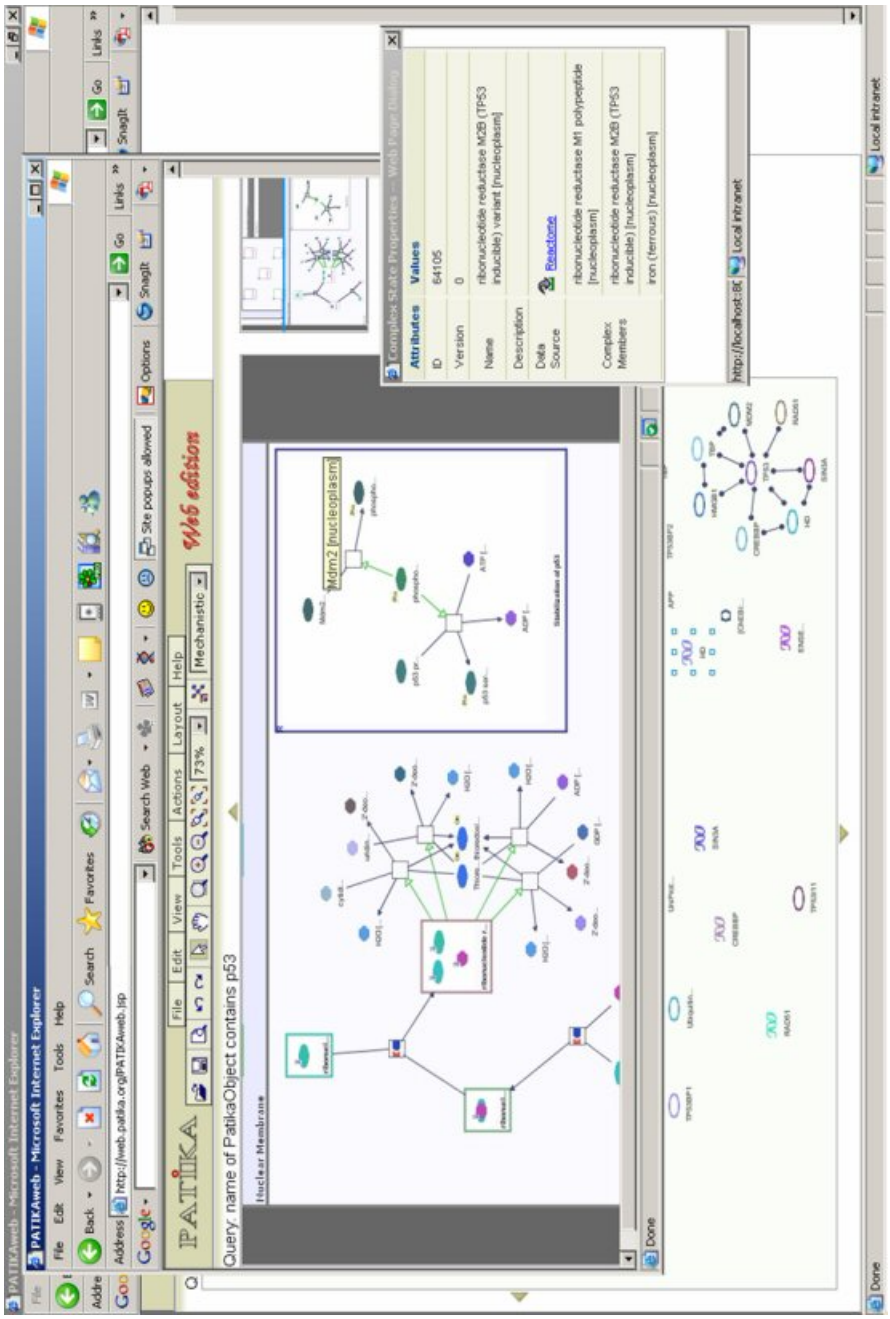

Figure 7.5: A screenshot of PATIKA*web*.

reads the query in the XML format, dispatches correct controller which in turn unmarshals Query type objects from the XML and runs them against the database. All queries return a set of PATIKA objects and some query specific result data e.g. in the case of k-shortest paths a set of lists describing paths found. However a set of PATIKA objects does not necessarily reflect a consistent view. The minimal consistent subgraph of the database containing the set is determined and copied into a DBPatikaGraph, forming a model. This model is marshaled into XML and finally the XML is sent back to the client. On the client side the graph is unmarshaled from XML. Already existing objects are detected, their version is checked and if outdated they are updated, new objects are merged to the model. The resulting view is incrementally laid out.

PATIKA query component has several layers as follows.

7.2.1 Query Interface

As the queries in PATIKA*pro* will be constructed according to user needs, a user interface layer works in client side to get the inputs of queries from user. This layer allows the user to select the type, inputs and other query-specific attributes from a dialog. When the query result is found (client side query) or taken from Query Proxy (server side query), this layer is responsible of retrieval of the query result to the user (see the analysis report).

7.2.2 Query Proxy

As some queries run in server side, the constructed queries with the help of Query Dialog should be sent to the main server to be evaluated. At those times, the queries are converted to XML format (i.e. marshaled) and sent to the main server from client computers. Query Proxy component is responsible from this operation. When the query is evaluated in server side and the result of the query is received from the server, Query Proxy makes the reverse operation and converts the XML data to its original form. When the operation is completed, it sends

the query result to Query Interface.

7.2.3 Query Controller

Query Controller works in server side and is responsible for evaluating the queries that come from client side. To achieve this, it first converts the XML formatted query to its original form (i.e. unmarshalls it) and then creates a Query object using the parameters. When the query result is found, it converts the result to XML format and sends to Query Proxy.

7.2.4 Query

Query is the object constructed by client or server side that contains all the information (i.e. type, running environment, predicates, etc.) about the query created by the user.

7.2.5 Query Algorithms

Query Algorithms are needed for evaluation of queries. As the query types, input parameters and evaluation criteria change from query to query, query algorithm design and implementations differ between them. So, there exist different algorithms for query evaluation. One exception is that Field Queries do not need any Query Algorithm objects; they are straightforward and evaluated directly using the graph model in client side or server side.

Every query can also receive result of another query as input. In this context, it is possible to create nested queries, which provides querying also according to non primitive fields. For example, by using nested field queries, users can inquire ComplexMemberStates whose BioEntities have name something. In that example, the inner primitive field query, returns bioentities according to their names. An overview of class relations are given in Figure 7.6.

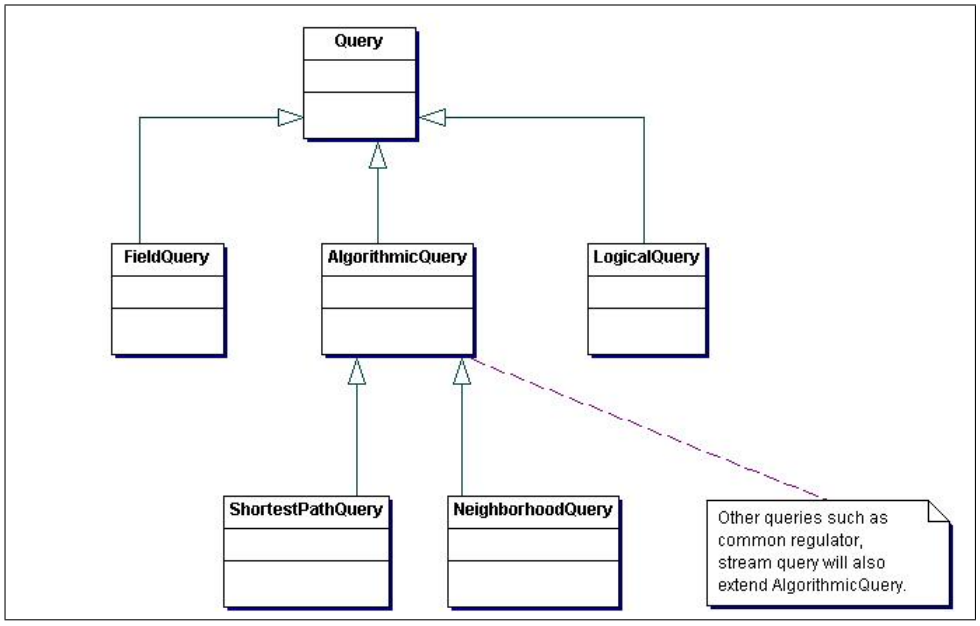

Figure 7.6: An overview of query class relations. Not all algorithmic queries are shown for brevity.

7.2.6 PATIKA Graph Model

The queries of PATIKA*pro* run on the graph model in both client and server sides. If the query will run on view or subject graph, graph model on client side is used as the environment to run the query. However, if the database will be used as the environment; the server side graph model is used.

Queries can be grouped into three class, field queries, algorithmic queries and logical queries. PATIKA system allows composing and combining them, allowing a very powerful querying system.

7.2.7 Query by fields of the objects

: These queries are the simplest queries that ask only the object with given field information. Field queries are composed of clauses and conditions. Clauses are the structures in which conditions and clauses are conjunct with ORs and ANDs, using a composite pattern. There are several kinds of conditions.

- *String condition* in which it is checked whether a field is equal to the specified string
- *Integer condition* in which it is checked whether a field is equal to the specified integer
- *Object condition* in which it is checked whether a field is equal to the specified object. These conditions are not directly created directly by the user, but it is required to check an object is equal to the result of another query, like joins in database queries.

  ```
  from ComplexMemberState where BioEntity =
    { from BioEntity where Name = ? }
  ```

  The above query written in PatikaFieldQuery language is an example to an object condition in which a string condition is used as an object. This query should get ComplexMemberStates of which have BioEntity's naming smt.

- *List condition* in which it is checked whether a field which is a list of something (integer, string or object) has any specified query. List conditions are as object conditions have at least one condition inside.

  ```
  from BioEntity where Names has any ?
  ```

  The above query is an example to a list condition which consists a string condition inside. Bioentities has a an array of Names which are simple strings, a bioentity is chosen if it has any value equal to the specified string in its Names string.

  ```
  from Complex where MemberStates has any
  { from ComplexMemberState where BioEntity =
  { from BioEntity where Names has any ? } }
  ```

  The above query is a list condition query and if one of its MemberStates is equal to the inner condition (an object condition consist of a list condition of a string condition), it is chosen. Figure 7.2.7 summarizes the aforementioned classes.

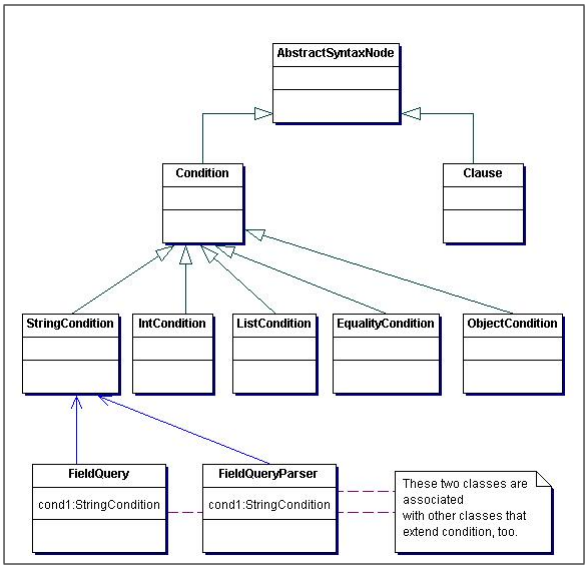

Figure 7.7: The class diagram of field query nodes. A composite pattern was used for arbitrary nesting of query objects.

These strings are then parsed and transformed into several field query objects. A FieldQueryParser object takes a string and then parses it producing an abstractSyntaxNode instance (or rather an instance of one of its subclass) which further may include more clauses and conditions by composition. State diagrams in Figures 7.8 and 7.9 depicts the details of field query parsing.

Field queries are interpreted differently at server and client side. This is achieved by making polymorphing calls to PatikaGraph interface. At the client side iteratively all of the PATIKA objects should be sent to the evaluate method of this abstractSyntaxNode one by one, and the list of the ones which return true, should be returned as the result of the query. A visitor pattern was implemented to achieve polymorphism between different S-level objects. Client side does not do anything for the performance, all queries have O(n) time complexity and there is no query optimizer. On the server side interpretation is even more simple. Since our PATIKA Field Query Language is similar but not equal to the Hibernate Query Language (HQL), conversion from our language to HQL can be done with little effort. An AbstractSyntaxNode object has an synthesizeHibernateQuery method, which can do this conversion, then only remains running of this query via hibernate query, with its full performance benefits.

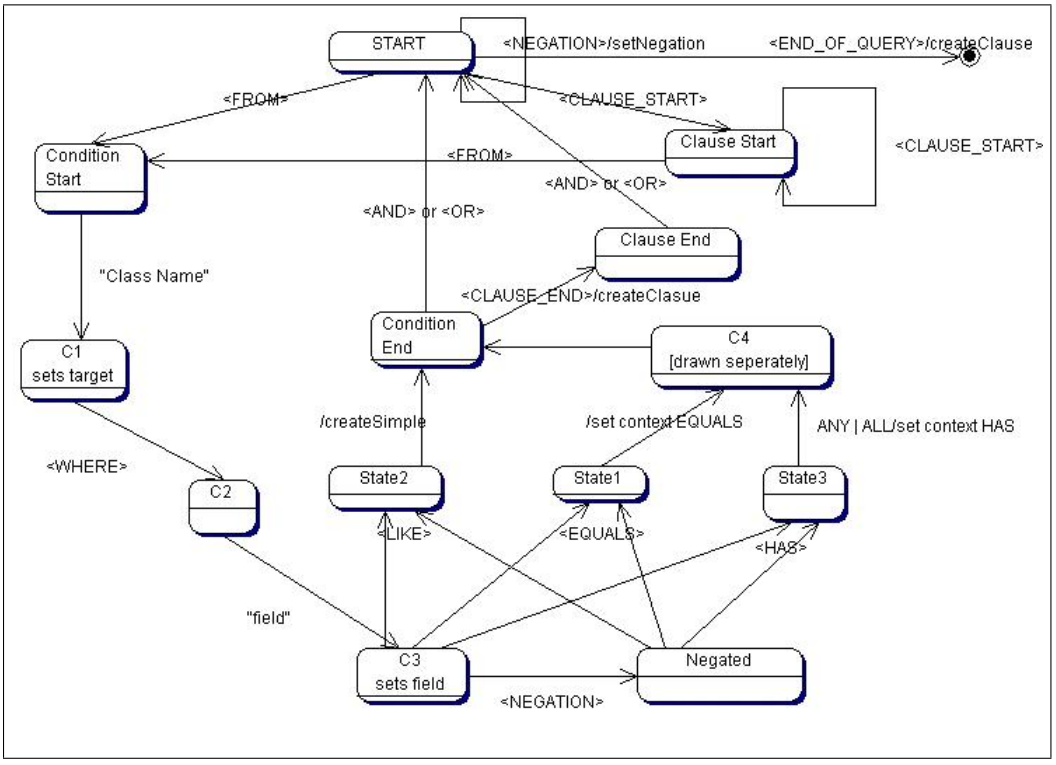

Figure 7.8: General state diagram of fieldQueryParser, for parsing the Patika query languages field queries.

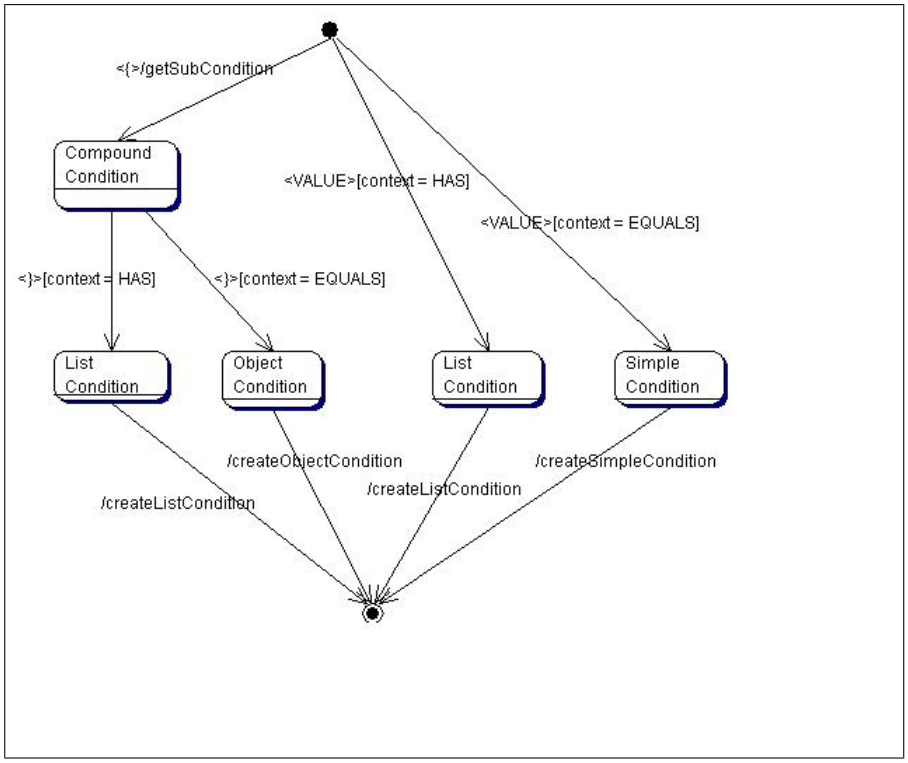

Figure 7.9: State diagram of the FieldQueryParser, for deciding on which condition to create. Through composite conditions it is possible to specify arbitrarily nested object relations.

7.2.8 Algorithmic (Pathway) queries

These types of queries include mostly the graph theoretic queries and the queries that ask about the pathway information. Examples include shortest path, k-neighborhood, or common regulator. PATIKA query system defines different graph theoretic queries for different biological problems.

First of all, let's define the following frequently used terms.

- *Path*: A path is a non-empty graph P = (V, E) where;

  $$V = n_0, n_1, , n_k E = n_0n_1, n_1n_2, , n_{k-1}n_k,$$

  where n_i are all distinct. n_0 and n_k are called the end points of path P. Therefore, we can write P as an ordered set of nodes, as follows

  $$P = n_0n_1n_2n_k$$

  Also we will represent paths as sets. In this manner, a path P = V (P, E) can be represented as $V(P) \cup E(P)$

- *Incoming and Outgoing Paths*: A directed path P is called an incoming path of node n if P ends at node n. Similarly, a directed path P is called an outgoing path of node n if P starts with node n. The sets of incoming and outgoing paths are denoted with $I_n \, and \, O_n$ respectively.

- *A-Path*: A-path is a path where one of the end nodes is from set A and no other nodes and interactions are from set A. Hence P is an A-path iff

  $V(P) \cap A = n_0 \, or \, V(P) \cap A = n_k$

  We denote the set of all A paths as Π_A.

- *A-B Path*: Let A and B be set of nodes and interactions, then an A-B path is a path with one end node is from set A and the other from set B, and no other node of path is from either set A or B. Hence P is an A-B path iff

  $$V(P) \cap A = n_0 \, and \, V(P) \cap B = n_k$$

or

$$V(P) \cap A = n_k \; and \; V(P) \cap B = n_0.$$

Similarly we denote the set of all A paths as Π_{A-B}.

- *Cycle*: If P = $n_0 n_1 n_2 n_k$ is a path, then C = P + $n_k n_0$ is a cycle. We denote cycle C as follows

$$C = n_0 n_1 n_2 n_k n_0$$

We denote the set of all cycles on n_0 as Γ_{n_0}

- *Positive and Negative Paths*: A directed path is called positive iff it contains an even number of inhibitors, and represented with a $\oplus$. Similarly, a directed path is called negative iff it contains an odd number of inhibitors and represented with $\ominus$.

- *Positive and Negative Feedback*: A directed cycle is called positive feedback iff it contains even number of inhibitors. Similarly, a directed cycle is called negative feedback iff it contains odd number of inhibitors.

- *Metabolic Path*: A directed path is called to be metabolic iff it does not contain any activators and inhibitors. A metabolic path is represented with $\odot$.

Following are the list of defined pathway/algorithmic queries in the PATIKA system. For each query type, we give a formal definition and their biological significance.

Shortest Path between two Sets

This query returns paths between two sets, or paths starting from source set and ending at target set. Hence, this query results in a set of paths with the following property:

$$SP(A, B) = \left\{ p \mid p \in \Pi_{(A-B)}, \mid p \mid \leq \mid q \mid \forall q \in \Pi_{A-B} \right\}$$

Note that a user can also specify the sign of the path. For the positive case resulting in such a condition:

$$SP^{\oplus}(A,B) = \left\{ p \mid p \in \Pi^{\oplus}_{(A-B)}, \mid p \mid \leq \mid q \mid \forall q \in \Pi^{\oplus}_{A-B} \right\}$$

and vice versa for the negative case.

Also there is an upper limit on the number of the paths that is to be returned. This acts as a safe guard as occasionally there might be combinatorially many paths.

Biological Significance: It is commonly accepted that graph theoretic distance of two nodes is correlated with their functional distance. This argument is a long one and is beyond the scope of this document. But to put it simply it has three basis:

- In a small-world graph evolved with node duplication events (most biological networks, including reaction networks fall into this category), graph theoretic distance correlates with evolutionary distance.
- Shorter the graph theoretic distance between two nodes, more likely that they are co-regulated, because there are less (control) reactions between them.
- Evolutionarily a very long path with many redundant intermediates should be suboptimal. Intermediates that do not perform control and amplification of the signal, are simply unnecessary vulnerable spots reducing the robustness of the system.

Assuming the above statement is true, and then this query answers the following questions:

- What are the possible route(s) that this protein governs this process?
- How pathway A and pathway B are linked?
- What is the most possible route for this signal to be transmitted to the nucleus?

Positive / Negative Feedback of a Node

This query results in a list of positive ($\Gamma_a^{\oplus}$) or negative cycles ($\Gamma_a^{\ominus}$) that contain a specified node.

Biological Significance: Feedback loops are an important apparatus used by cellular networks. They can have signal amplifying or stabilizing roles. This query answers questions like:

- How is the concentration of this molecule stabilized?
- How this signal gets amplified?

k-Neighborhood of a Node Set

This query returns all the edges and nodes which are in the specified distance, k, to the specified objects. This query results in a set N, where

$$N_k(A) = A \cup \{x \mid \exists p \in \Pi_A, x \in p\}$$

Biological Significance: This query again relies on the graph theoretic argument above, but takes a different point of view. It finds out objects that are closest to the given target(s), thus returns a functional neighborhood. This query answers questions like:

- In which pathways does my protein take part in?
- With which states does this molecule interact with?
- What are the other actors taking part in this process?
- Which proteins catalyze this reaction?

Positive/Negative Upstream and Downstream

A positive 1-upstream of a node is the first node on the incoming path which also activates the preceding transition of that node. Similarly, a k^{th} positive upstream is the k^{th} node that behaves in the same manner. Negative upstream of a node is composed of the nodes on the incoming path that inhibits the preceding transition of that node. In downstream case, the nodes following the proceeding transition of the node is concerned. To give a chemical intuition, positive stream of a node is the set of nodes that keep the node not to be vanished by reactions. Formally stating a node s is in the positive upstream of a node t iff:

$$\exists p : p \in O_s^{\oplus}, p \in I_t^{\oplus}, \mid p \mid \leq k$$

The negative and/or downstream queries are defined similarly.

Biological Significance: Analyzing upstream and downstream nodes of a molecule is important to be able to retrieve cause/effect relationships, which are critical in diagnosis or drug design. This query answers questions like:

- What activated this protein?
- Which processes are affected if this gene is knocked down?
- What are the downstream effects of this drug?

Unambiguous Upstream and Downstream

By unambiguous positive upstream of a node n we mean the set of nodes that have only positive outgoing paths ending at n. And similarly we define the negative and positive upstream and downstream of a node. Formally, a node s is in unambiguous upstream of node t iff:

$$\exists p : p \in O_s^{\oplus}, p \in I_t^{\oplus}, \mid p \mid \leq k$$

and

$$\neq \exists q : q \in O_s^{\ominus}, q \in I_t^{\ominus}, \mid q \mid \leq k$$

Biological Significance: The result of this query is a subset of the previous one. However in this query we require that there are no routes with conflicting effects between the source and target, i.e. all paths are of the same sign. This way we can say unambiguously that to our best knowledge source affects the target in this manner.

Common Target and Common Regulator

If node A is starting node of a directed path that end up with node B, then node B is said to be a target of node A and node A is said to be a regulator of node B. In this context, common target of a node set S is the set of nodes that are targets of all nodes in S; similarly the common regulator of a node set S is the set of nodes that are regulators of all nodes in S. A

Node t is common positive target for set of nodes S iff

$$\forall s \in S \; \exists p : p \in O_s^{\oplus}, p \in I_t^{\oplus}, \mid p \mid \leq k$$

Biological Importance: This query becomes important when analyzing correlated events, like microarray expression levels. It finds common regulators/targets, that can possibly explain observed correlation. This query answers questions like:

- Why the expression levels of these two genes are correlated?
- Why are the final phenotypes of these two different signals the same?

Graph of Interest

This query is actually like a special version of query that will be mentioned next. If the same set is given for both input sets, then the result will be the same as

of this query. This query returns all paths of length at most k between any two nodes of a specified node set. As in the name of the query, we aim to find a graph of interested nodes. And we give a parameter k, to limit the size of the resulting graph.

$$GoI(A,k) = \bigcup \left\{ p \mid p, p \in O_s^{\oplus}, s \in A, p \in I_t^{\oplus}, t \in A, \mid p \mid \leq k, \right\}$$

Biological Importance: Although this query does not attempt to answer a specific question, it allows a quick and easy way for the user the fetch subgraph, that is potentially most interesting for them based on a set of initial nodes. Compared to a neighborhood query GOI has the specific advantage of filtering out dangling subgraphs that is connected to only one "interest node". GOI is also useful in analyzing microarray data as when given a set of correlated genes, it brings in paths between those genes.

7.2.9 Logical queries

These queries allow performing negation, union and intersection operations on other query results.

Some algorithms provide the user with extra information, for example a cycle query might return a set of cycles in the form of lists. Query Result class allows attachment of such information to the result set. This information is then visualized at the client side. A bridge pattern is used to handle different query results.

7.2.10 Server Side Query Sequence

Using the parameters taken from the user, through a GUI,the query is constructed, converted to a custom XML format and sent to server side by Query Proxy. In the server side, Query Manager unmarshals the XML and creates a Query object. Once the query object is created, it is run on database either after the creation of an appropriate algorithm or directly (if it is a Field Query). The

result of the query needs to be sent back to client side and same steps are followed in the reverse order. Query Manager marshals the result of the query into XML format and sends to Query Proxy. Query Proxy unmarshalls the query result and passes the query result to the Query Interface.

7.3 Model Integration and Concurrency

Although not directly a part of ontology, model integration is nevertheless an ontology issue. PATIKA assumes a distributed collaborative environment where scientists put together a model similar to *putting together pieces of a puzzle*.

In the submission scenario, a user makes several modifications to the objects queried from server and/or creates new objects, and decides to submit these changes to the database, to be integrated with the rest of the model. Server receives an http request to submission url. Servlet reads the submission in the XML format, dispatches correct controller which in turn unmarshals a Submission object. Users id and access rights is checked and if passed submission is processed. A submission manager is called which performs several checks. In *validity* check common ontology invariants are checked to verify that the submitted model is in fact valid. Although this check is also done by the PATIKA client before submission, it is only safe to have it double checked. In *version* check submitted objects are checked for being up-to-date to avoid concurrency conflicts. In *orphan* checks it is ensured that objects scheduled for deletion does not invalidate ontological invariants. Finally duplicate checks determine if there is a *suspiciously* similar object in the database and warns user to avoid redundant duplications. If these checks are successfully passed the submission is accepted for validation. Initial XML is persisted as a file, and the submission is sent to an expert. The expert then examines and validates the model, acting as a scientific arbiter. Finally the submission is merged to the database. In trusted environments, such as a local server at a research lab, it is possible to skip this validation step.

PATIKA system tracks the version of each object. Additionally a modified flag

is set to true whenever a user edits an object. Finally a removed flag is set if the object is scheduled for removal.

7.3.1 Identity and Versioning

Every Patika object has a unique identification object called PID (Patika identification), which keeps track of the ID and the version of the object. A PID comes in two types, local and global. When an object is first created, editor assigns it an ID using an incrementing scheme. The version is set to 0. When this object is first committed to the database, a new ID is issued, and the version is set to 1. After this point we call this ID global. When an object with a global ID is submitted to the database, its version is incremented by 1. In a Patika graph, all local ID s are unique among other local IDs, and all global IDs are unique among other global IDs.

One should notice that assigning an ID and version alone does not solve all identity and redundancy problems. For most bioentities, responsibilities for object identity and non-redundancy is delegated to the source database. For example we assume that two genes with different Entrez-Gene IDs are actually different entities, and we also assume that the granularity of the Entrez-Gene is actually satisfying for all users, i.e. no user will need to divide DNA bioentities imported from Entrez-Gene. They are atomic for the current paradigm. This is obviously a strong statement, however it is the only viable and feasible option if we are hoping to integrate different models, and should be acceptable for most use-cases, if not all.

As for the other Patika objects, unfortunately there is nothing we can delegate to. Patika employs several mechanisms to minimize duplicate objects. For example when a user wants to create a state of a protein, a query based on the owner entity is sent to the server, it is first presented the existing states of this protein in the database. Upon reviewing these if they figure that the state they have in mind is already defined in the database, they can simply retrieve this from the database instead of creating a duplicate. As a second measure if a state's all

entity variables are same with an existing state, duplicate check at the server side fails and user is asked to first fix this (either by replacing the duplicate entry with the existing one, or providing the variable that makes this state a new one), Similarly transitions with exact state/product set is not allowed. However these checks would miss semantic mistakes. In that case we rely on expert validation and curation to fix those problems. Although this sounds like rather weak, in fact creating a duplicate entry is quite difficult in the PATIKA system and with sensible curators, it should be extremely rare.

7.3.2 Concurrency

A problem of collaborative systems is to merge user edits. PATIKA employs a CVS like logic for handling concurrencies. As for the concurrency scenarios, there are five events, a user might get an up-to-date copy (checkout), edit its copy (edit), submit its changes to the database (commit) and update its edited copy with the latest database version (merge), schedule it for removal from the database (remove). When multiple users actually perform these actions concurrently, there can be some conflicts.

When a PATIKA object is first checked out it is in up-to-date state, if this object is edited it will pass to the modified state. If a user commits this modified object, its version in the database is incremented, and all other copies of the objects in other clients become out-dated. At any time a user might invoke a checkout to fetch the up-to-date version of the object. If however, the outdated object is also modified, then we say that it is in conflict state. Now instead of checkout, which will simply override user changes and bring it to up-to-date state, user can also perform a merge, which will semi-automatically merge changes between two copies and bring it to modified state. PATIKA provides facilities for querying the state of the current objects and performing various operations. Figure 7.10 is a screenshot from PATIKA editor where the concurrency status of the current objects are highlighted.

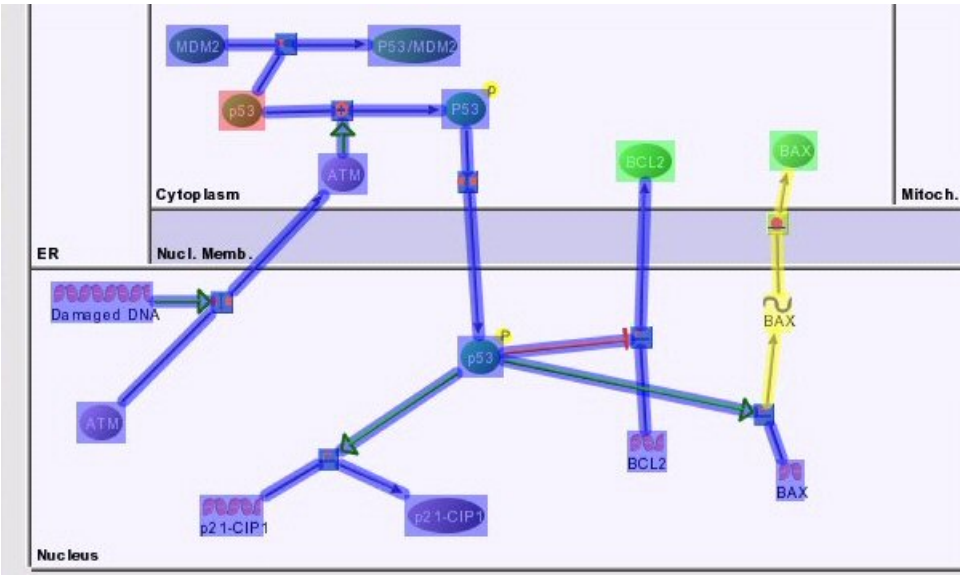

Figure 7.10: A screenshot of PATIKA editor where the concurrency status of the current objects are highlighted, by the *show status* facility. Blue means the object is up-to-date, yellow modified, green local and red conflicting

.

An interesting scenario occurs when two users query an object and concurrently modify it, and then one of them submits the modified version to the database. Now the other user has an edit conflict. If that object is queried again PATIKA will automatically detect this conflict by checking version numbers during merge, and then provide an *update wizard* to allow user to merge its local changes to the new version in the database.

7.3.3 Orphaning

A problem with the current PATIKA integration framework rises due to the fact that each user works on a certain subgraph of the database. This requires that certain elements are partially represented. For example not all reactions that a state participates in is brought to the client. Changes, particularly delete operations, on such elements might lead to consistencies upon merge, even though the submitted graph is valid by itself. For example removing a Bioentity requires all of its states to be removed as well. Such inconsistencies are called orphans,

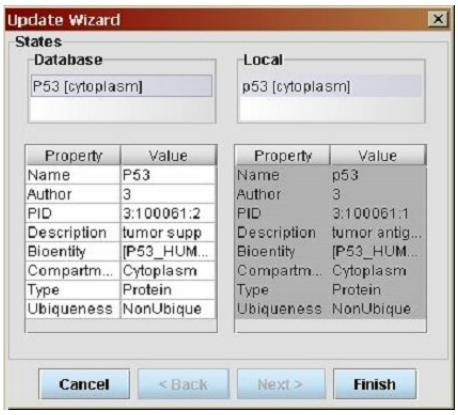

Figure 7.11: An update wizard allows comparing and merging changes

and are checked and detected by the submission system when a submission is received. User is asked to retrieve orphaned objects, edit the graph to make it consistent and resubmit. In the previous example the user can also schedule states of the bioentity for removal as well to restore validity.

7.3.4 Multiple Levels of Detail

A problem with integration efforts is different parts of the knowledge is known at different levels of detail. A scenario is given in figure 7.12. Incomplete abstractions can be used for similar cases for handling multiple levels of detail.

7.4 View Management

A user of the Patika database is interested in a small sub-graph of the actual network in the database. There are two level of views in the Patika system. Each Patika client represents a view, fetched by incremental Patika queries. In the query section, we have already described the scenarios and facilities for retrieving this subgraph, called the subject graph. However we also need to manage the views obtained by the queries. Then at the client, there are multiple views of the subject graph, again subject to same validity conditions.

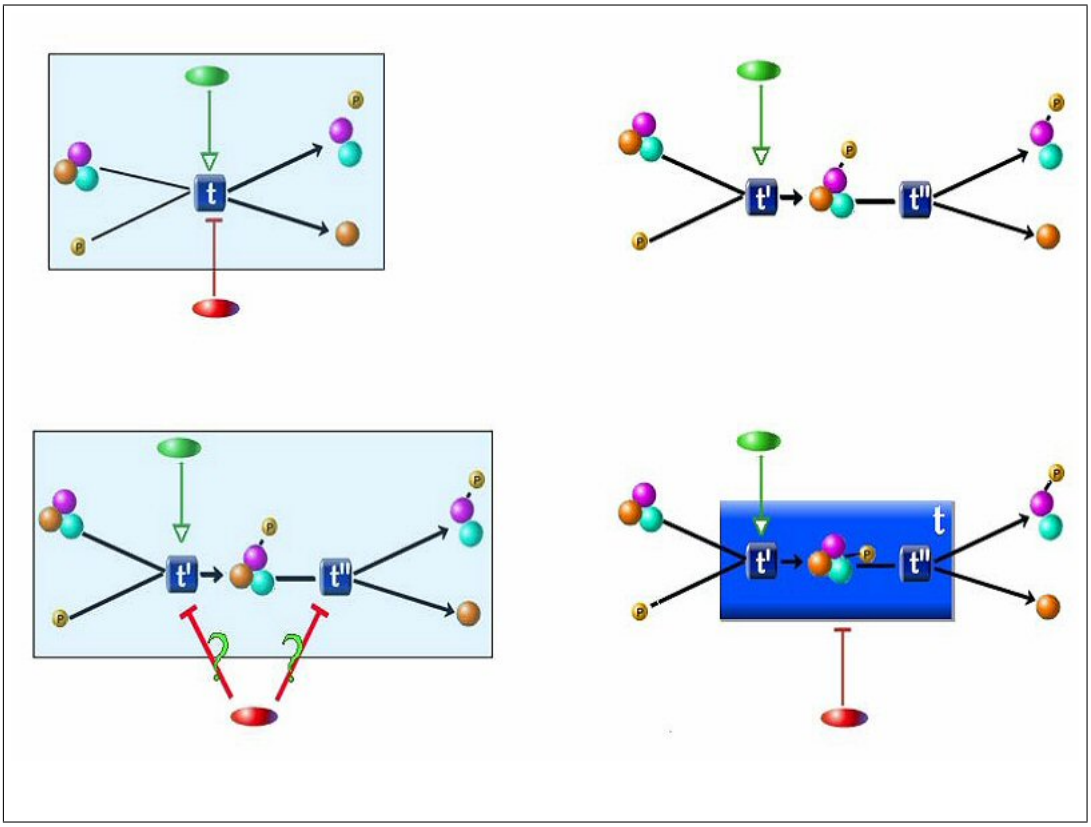

Figure 7.12: A simple reaction in the pathway (upper left) is queried (shaded box) and replaced by the user to include intermediary steps (upper right). However user might not know whether the inhibitor at the bottom inhibits first or second step (lower left). A solution to this problem is to allow user to define an incomplete transition abstraction, and define the inhibition on the abstraction, allowing multiple levels of detail.

On the server side these constraints are satisfied with the *excision* system where the valid minimal graph is obtained from a given set of nodes by recursively calculating the prerequisites of the objects, by calling the *getPrerequisites()* method. Then the expanded object set is retrieved from the database, cloned, inserted into a new graph, and the corresponding relations restored, which is again done polymorphically by *bindToPrerequisites()* method. The created graph is then passed to the client, and either replaces subject graph or merged into it. In the case of a merge, there is still one risk of inconsistency, an interaction between a node in the existing graph and the merged graph does not necessarily excised by the server. Although this problem can easily solved by running an O(n) check on the server, it can sinificantly increase the server traffic as it is a very frequent operation. Thus for the time being, we are allowing it to happen. User however can always. update their view, upon which they will receive the missing interaction.

On the server side things are bit more complicated as view(s) of a subject graph can be visualized/edited at the same time on different graph windows and synchronization of the graph objects is an important issue, since the modification of a view affects the subject, thus affects all other views in the consideration of the user in the client. The mechanism can be simplified as composed of single subject and observers depending on this subject. The design for this mechanism is based on the Observer Pattern, since each view is basically the observer of the subject graph. To complicate issues more, if two abstractions are overlapping in the view only one of them can be visualized fully as due to the geometrical constraints. As an amendment we introduce an alternative method of visualizing abstractions by drawing a holo around its members. From the user's point of view an abstraction can be in four visual states: Hidden, Holo, Collapsed and Expanded. PATIKA users can change visualization states of abstraction by a given interface. However programmatically visual state transitions for an abstraction depends on the other abstractions in the view (Figure 7.13).

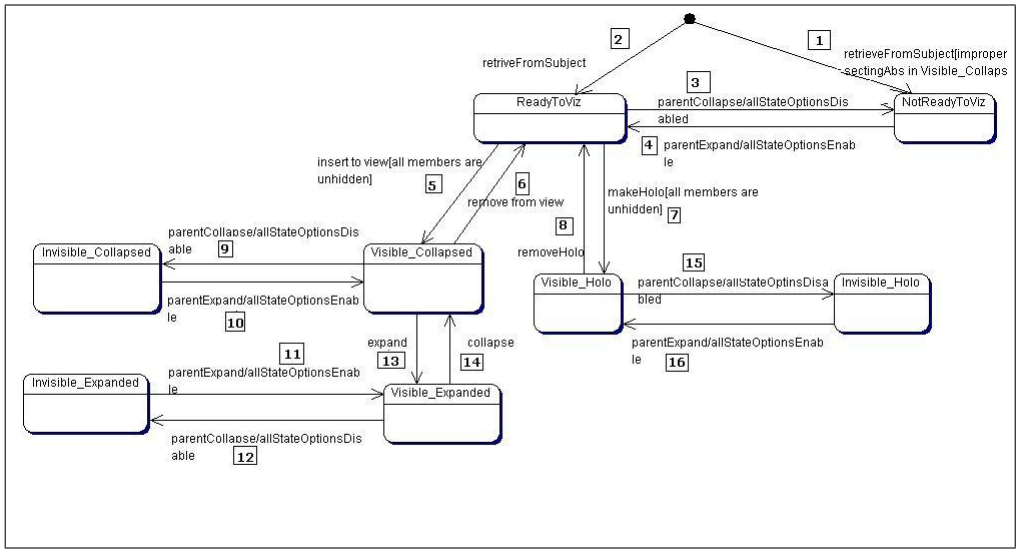

Figure 7.13: A state diagram showing how various PATIKA operations change the visualization state of an abstraction. For example if one of the members of an abstraction is deleted from the view, then it should be also removed from the view (3), or it can not be visualized other than as a holo, if it has an overlapping abstraction that is in expanded state (8).

7.5 System and Ontology

So far we argued a certain flow of direction for the design process. From biological domain to the ontology, from the ontology to the software system. Although this sounds completely rational, there were many cases where the reverse happened actually; the requirements of the system dictated the ontology. Below is a list of several such cases:

1. *Incomplete Abstractions*

   Incomplete abstractions (called blackbox pathways) rarely exist in the biological literature, as representing multiple levels of detail and incompleteness is not a major requirement for them. Initial PATIKA ontology only had one type of abstraction, called a summary back then, but with the discussion of integration scenarios, it became clear that the semantics of these summary nodes were way too ambiguous for our needs. We defined incomplete abstractions to be able to deal with several integration scenarios,

isolating incompleteness and ambiguity from the rest of the system.

2. *Ubique States*

   The introduction of the ubique states into the ontology was only for addressing problems that were created by high degree nodes, in analysis and visualization. Although we have foreseen this problem much earlier, their proper implementation was postponed until we started migrating actual data from Reactome and hit the problems in laying them out and querying them.

   After their introduction, however ubiques needed another refinement in the ontology based on ubique membership in abstractions. This was because they broke one of the invariances of the abstraction visualization system. A deeper analysis revealed that in fact the semantics of having a disconnected ubique in an abstraction was not clear, and should be best not allowed.

3. *Homology Abstraction Semantics*

   Again problem with homology abstraction semantics arise as a result of abstraction visualization efforts. Similar to the ubique case, we realized that we needed a more strong mapping between the owner and owned abstractions and thus have added the additional restrictions detailed in the ontology chapter.

4. *View Validity*

   First concerns related to the view validity arise when we were designing the query subsystem of the first version of the Patika editor, quickly we realized the need for an excision system and added the view validity constraints to the ontology.

5. *State Variables*

   Previously state variables were described using a less formal system. The need for detecting the duplicate states was the first motivation for formalizing state variables with further subclasses and control vocabularies.

As the above examples clearly indicates there is in fact a strong coupling between the software development process and the ontology. The software system is not merely a testing ground for the ontology, it has its own set of problems, which more than often better delegated to the ontology. It also clarifies the minimal information that are needed to perform various goals such as integration or visualization.

Chapter 8

Discussion

8.1 Why a new ontology?

By the time PATIKA project started, there were no real ontology for modeling cellular processes at the level of mechanism, i.e. a network of biochemical reactions, that can possibly cover major signal transduction pathways. One might question the need for developing a new ontology. After all there were very good ontologies for chemical reactions, interaction data and metabolic pathways. In which ways signal transduction pathways are different?

Mechanistic representation requires additional detail that can not be covered by existing ontologies. Chemical reactions have no concept of entity and compartments, Interaction ontologies are defined only at the entity level and metabolic pathways separates substrate/product set clearly from the enzyme set, a feature that is most needed for signal transduction pathways.

Mechanistic level representation often requires an interpretation of the experimental data and can not be produced in a high-throughput manner. Still we strongly believe that the advantage is obvious; well defined, high detail ontology allows integration of different experimental findings and reasoning over them. However, in reality a quick overview of human curated metabolic and signaling

pathway databases employing this representation system reveals a different picture. Although all of them are based on chemical paradigm, there are different extensions in their ontologies, and sometimes these can be quite ambiguous in their semantics. This is especially the case for signaling pathway databases.We believe that the main reason behind this proliferation and divergence is the failure of the chemical paradigm to model biological systems.

Most of the deficiency in covering mechanism of signal transduction was due to lack of state-bioentity duality. In a very short time, several different research groups, including ours [83, 75, 22] published ontologies containing very similar state concepts. Several other concepts including abstractions emerged at the same time. Our ontology was first to bring issues such as formal state definitions via state variables, incomplete and homology abstractions, transition function classifications, cell models etc. and show that it can actually work by developing a platform based on it.

We strongly believe that PATIKA ontology have influenced several other projects as well, including Reactome [39] and BioPAX [18].

8.2 Simulation vs. Pathway Reconstruction

There are two major use cases for a pathway modeling system, simulation and pathway reconstruction.

A simulation use case has two major ultimate goals, ability to represent an observed temporal aspect, and ability to predict. Although not an absolute necessity, most simulation systems are based on quantitative models. Parameters of this reaction needs to be determined. If this data is experimentally available (whether the experimentally available, reduced *in vitro* data is acceptable is another issue) then one can talk about a system that can predict the outcome of a system given an initial condition. That data is most often missing. What happens is that scientists employ different steady state assumptions to come up with constants such that the system successfully displays the same phenotype with

the subject at hand, effectively covering the temporal information. However, one should be careful that these *predicted* constants are strictly context dependent, and different quantitative models are hard to integrate. Even with a predictive system, one need to come up with virtual constants that specify the system boundaries. For example many metabolic systems assume a constant concentration of ATP or glucose. These artificial constraints exist, since we are modeling a subsystem and we need to artificially isolate it from the rest of the system. It is ok though, for the requirements of this system, since the simulation is consistent within itself.

These constants however has one major drawback. It is virtually impossible to integrate two such models, since scaling of the constants between models is arbitrary and there is no naive policy to implement to ensure correct scaling of the constants during model creation time. Pathway reconstruction systems such as Patika let go of the simulation ability to provide increased coverage and integrability. Such systems are always qualitative. They can not represent temporal aspects, and has very limited predictive power. However we expect that once the underlying graph topology is reconstructed, one can use it to create and integrate quantitative models.

8.3 Public Standard Development Efforts and Patika Ontology

A survey of several ontologies that can cover signaling pathways easily reveals that, due to aforementioned divergence in semantics and ontology terms, standardization is of critical importance. Table 8.1 gives an overview of naming clashes between ontologies. Semantic clashes, which are much more serious, are as frequent as naming clashes.

There are several efforts for developing a standard representation system for pathways. Based on two major use cases, simulation and pathway reconstruction, two major efforts are prominent on the scene, SBML and BioPAX respectively.

BioPAX	PATIKA	aMaze	INOH	Reactome	SBML 2
Physical Entity	Bio Entity	Physical Entity	Reference Entity	Reference Entity	–
State	State	Bio Entity	State	Physical Entity	Species
Physical Entity Participant	Interaction	Input/ Output	Input/ Output/ Controller	Input/ Output	Species Reference
Interaction	Transition	Bioevent	Process/ Event	Event	Reaction
State Variable	Bioentity Variable	State	Biological Attribute	-	-

Table 8.1: A comparison of naming of different ontologies. Note that several terms clash with each other.

The Systems Biology Markup Language (SBML) is a machine-readable format for describing qualitative and quantitative models of biochemical networks. It can also be used to express the interactions of biochemical networks with other phenomena. The emphasis in SBML is on supporting quantitative models. Major differences between SBML and PATIKA ontology are:

- the lack of entities and entity level representation. Each state in SBML, called a specie is a separate molecule. Therefore there are no semantic relation between cytoplasmic Ca and Extracellular Ca in an SBML model. Similarly one can not represent entity level interaction information in SBML.
- the ability to represent quantitative models, SBML's focus is not on model integration.
- unstructured cell model. Each compartment in SBML is a bag, where PATIKA compartments are aware of their neighbors.
- no abstraction facilities. Abstractions, by definition are qualitative structures, and can not be simulated, so SBML does not contain them.

The goal of the BioPAX group is to develop a common exchange format for biological pathways data. Since BioPAX's concentration is much more on

pathway reconstruction, it is much more similar to PATIKA ontology compared to SBML. Major differences between BioPAX and PATIKA ontology are:

- the lack of states. Current BioPAX definition considers each participant of a reaction as a separate state of a molecule, but there is no way to represent if two participants are actually the same state.
- the make use of modulation class. Instead of directly defining activators and inhibitors of a transition, BioPAX make use of a separate class called modulation, which then modulates the reaction. This addresses some of the open issues we mentioned at Ontology chapter related to defining cumulative or multiplicative effects of effectors and transition rules.
- abstraction facilities for only representing pathways, analogous to PATIKAś regular abstractions. No ability to represent incomplete information, multiple levels of detail and homology.

We are actively involved in the BioPAX, and have contributed significantly to the state subgroup and cell models. Starting from Level 3 (current release is level 2) BioPAX will cover most of PATIKA structures including states and cell models, allowing import and export of BioPAX models to PATIKA format with minimal information loss.

Currently we have exporters for converting PATIKA models to BioPAX and SBML. However the inverse is not true, since BioPAX and SBML models are relatively more *loose* and may not contain information that are mandatory for PATIKA models, such as a cell model or entity-state relationships.

8.4 Future Directions

We have listed a bunch of open issues that are still waiting to be addressed. Of all, generic entities pose the most challenging obstacle, and also have the utmost priority due to their omnipresence in cellular pathways. They are particularly

important for PATIKA since we are importing large amounts of data from Reactome and losing quite an important bit of it because we can not cover generic entities in Reactome. To a lesser extent, modeling chromosome structure can also provide a significant amount of coverage, allowing importing and integrating data from transcriptional regulation databases.

PATIKA is ongoing project with several subprojects. The requirements of these subprojects affect and are affected by the ontology. They will definitely provide new use cases and requirements for the ontology, which will lead to new additions and improvements to the ontology.

Some very recent ontology changes were resulted from the data integration efforts from external databases. We have identified points where we are incompatible and improved our ontology accordingly. As we will add more and more data sources ontology will be also refined to fit and cover them. In a similar manner new high-throughput techniques will create new data sources and types that needs to be fitted into the system.

8.5 Conclusion

We have described an ontology for modeling cellular processes. The ontology provides novel structures and facilities for describing phenomena and aspects such as entity-state relations, homologies, compartments and incomplete information. Based on this ontology we have built an integrated system for modeling processes, and developed methods for visualizing, analyzing and integrating them. The system itself also acts as a proof of concept for the ontology, as it shows that the PATIKA ontology can be used feasibly to model quite a large amount of cellular processes in biological literature today with minimal information loss.

Today we are witnessing an accelerating progress toward a standardized ontology for cellular processes, and concept that was first proposed by PATIKA ontology, are being converged and adopted by such efforts. We are hoping to contribute to the field in the future as well.

Bibliography

[1] G. Ackers, A. Johnson, and M. Shea. Quantitative model for gene regulation by lambda phage repressor. *Proc Natl Acad Sci U S A*, 79(4):1129–33, Feb 1982.

[2] A. Al-Chalabi, P. M. Andersen, P. Nilsson, B. Chioza, J. L. Andersson, C. Russ, C. E. Shaw, J. F. Powell, and P. N. Leigh. Deletions of the heavy neurofilament subunit tail in amyotrophic lateral sclerosis. *Hum Mol Genet*, 8(2):157–164, Feb 1999.

[3] C. Alfarano, C. Andrade, K. Anthony, N. Bahroos, M. Bajec, K. Bantoft, D. Betel, B. Bobechko, K. Boutilier, E. Burgess, K. Buzadzija, R. Cavero, C. D'Abreo, I. Donaldson, D. Dorairajoo, M. Dumontier, M. Dumontier, V. Earles, R. Farrall, H. Feldman, E. Garderman, Y. Gong, R. Gonzaga, V. Grytsan, E. Gryz, V. Gu, E. Haldorsen, A. Halupa, R. Haw, A. Hrvojic, L. Hurrell, R. Isserlin, F. Jack, F. Juma, A. Khan, T. Kon, S. Konopinsky, V. Le, E. Lee, S. Ling, M. Magidin, J. Moniakis, J. Montojo, S. Moore, B. Muskat, I. Ng, J. Paraiso, B. Parker, G. Pintilie, R. Pirone, J. Salama, S. Sgro, T. Shan, Y. Shu, J. Siew, D. Skinner, K. Snyder, R. Stasiuk, D. Strumpf, B. Tuekam, S. Tao, Z. Wang, M. White, R. Willis, C. Wolting, S. Wong, A. Wrong, C. Xin, R. Yao, B. Yates, S. Zhang, K. Zheng, T. Pawson, B. F. F. Ouellette, and C. W. V. Hogue. The Biomolecular Interaction Network Database and related tools 2005 update. *Nucleic Acids Res*, 33(Database issue):D418–24, Jan 2005.

[4] J. Allen, H. M. Davey, D. Broadhurst, J. K. Heald, J. J. Rowland, S. G. Oliver, and D. B. Kell. High-throughput classification of yeast mutants for

functional genomics using metabolic footprinting. *Nat Biotechnol*, 21(6):692–6, Jun 2003.

[5] A. A. Aravin, M. S. Klenov, V. V. Vagin, F. Bantignies, G. Cavalli, and V. A. Gvozdev. Dissection of a natural RNA silencing process in the Drosophila melanogaster germ line. *Mol Cell Biol*, 24(15):6742–6750, Aug 2004.

[6] M. Ashburner, C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, M. Harris, D. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. Matese, J. Richardson, M. Ringwald, G. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–9, May 2000.

[7] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh. The Universal Protein Resource (UniProt). *Nucleic Acids Res*, 33(Database issue):D154–9, Jan 2005.

[8] U. S. Bhalla. The chemical organization of signaling interactions. *Bioinformatics*, 18(6):855–863, Jun 2002.

[9] U. S. Bhalla. Temporal computation by synaptic signaling pathways. *J Chem Neuroanat*, 26(2):81–86, Oct 2003.

[10] U. S. Bhalla. Models of cell signaling pathways. *Curr Opin Genet Dev*, 14(4):375–381, Aug 2004.

[11] U. S. Bhalla. Signaling in small subcellular volumes. I. Stochastic and diffusion effects on individual pathways. *Biophys J*, 87(2):733–744, Aug 2004.

[12] U. S. Bhalla and R. Iyengar. Robustness of the bistable behavior of a biological signaling feedback loop. *Chaos*, 11(1):221–226, Mar 2001.

[13] L. Bintu, N. E. Buchler, H. G. Garcia, U. Gerland, T. Hwa, J. Kondev, T. Kuhlman, and R. Phillips. Transcriptional regulation by the numbers: applications. *Curr Opin Genet Dev*, 15(2):125–35, Apr 2005.

[14] L. Bintu, N. E. Buchler, H. G. Garcia, U. Gerland, T. Hwa, J. Kondev, and R. Phillips. Transcriptional regulation by the numbers: models. *Curr Opin Genet Dev*, 15(2):116–124, Apr 2005.

[15] A. Blais and B. D. Dynlacht. Devising transcriptional regulatory networks operating during the cell cycle and differentiation using ChIP-on-chip. *Chromosome Res*, 13(3):275–288, 2005.

[16] M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, Nov 2004.

[17] K. R. Brown and I. Jurisica. Online predicted human interaction database. *Bioinformatics*, 21(9):2076–82, May 2005.

[18] M. P. Cary, G. D. Bader, and C. Sander. Pathway information for systems biology. *FEBS Lett*, 579(8):1815–20, Mar 2005.

[19] N. committee of the international union of biochemistry and molecular biology (NC-IUBMB). Enzyme Supplement 5 (1999). *Eur J Biochem*, 264(2):610–50, Sep 1999.

[20] N. N. Danial and S. J. Korsmeyer. Cell death: critical control points. *Cell*, 116(2):205–219, Jan 2004.

[21] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, A. Ayaz, G. Gulesir, G. Nisanci, and R. Cetin-Atalay. An ontology for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 20(3):349–56, Feb 2004.

[22] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, Jul 2002.

[23] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A layout algorithm for undirected compound graphs. *Submitted for Publication*, 2005.

[24] A. Fire, S. Xu, M. Montgomery, S. Kostas, S. Driver, and C. Mello. Potent and specific genetic interference by double-stranded RNA in Caenorhabditis elegans. *Nature*, 391(6669):806–11, Feb 1998.

[25] N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303(5659):799–805, Feb 2004.

[26] K. Fukuda and T. Takagi. Knowledge representation of signal transduction pathways. *Bioinformatics*, 17(9):829–37, Sep 2001.

[27] K. I. Fukuda, Y. Yamagata, and T. Takagi. FREX: a query interface for biological processes with hierarchical and recursive structures. *In Silico Biol*, 4(1):63–79, 2004.

[28] A.-C. Gavin, M. B?sche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J. M. Rick, A.-M. Michon, C.-M. Cruciat, M. Remor, C. H?fert, M. Schelder, M. Brajenovic, H. Ruffner, A. Merino, K. Klein, M. Hudak, D. Dickson, T. Rudi, V. Gnau, A. Bauch, S. Bastuck, B. Huhse, C. Leutwein, M.-A. Heurtier, R. R. Copley, A. Edelmann, E. Querfurth, V. Rybin, G. Drewes, M. Raida, T. Bouwmeester, P. Bork, B. Seraphin, B. Kuster, G. Neubauer, and G. Superti-Furga. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–7, Jan 2002.

[29] D. Hanahan and R. Weinberg. The hallmarks of cancer. *Cell*, 100(1):57–70, Jan 2000.

[30] M. Harris, J. Clark, A. Ireland, J. Lomax, M. Ashburner, R. Foulger, K. Eilbeck, S. Lewis, B. Marshall, C. Mungall, J. Richter, G. Rubin, J. Blake, C. Bult, M. Dolan, H. Drabkin, J. Eppig, D. Hill, L. Ni, M. Ringwald, R. Balakrishnan, J. Cherry, K. Christie, M. Costanzo, S. Dwight, S. Engel, D. Fisk, J. Hirschman, E. Hong, R. Nash, A. Sethuraman, C. Theesfeld, D. Botstein, K. Dolinski, B. Feierbach, T. Berardini, S. Mundodi, S. Rhee, R. Apweiler, D. Barrell, E. Camon, E. Dimmer, V. Lee, R. Chisholm, P. Gaudet, W. Kibbe, R. Kishore, E. Schwarz, P. Sternberg, M. Gwinn, L. Hannick, J. Wortman, M. Berriman, V. Wood, N. de la Cruz, P. Tonellato,

P. Jaiswal, T. Seigfried, R. White, and G. O. Consortium. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res*, 32(Database issue):D258–61, Jan 2004.

[31] H. L. Heine, H. S. Leong, F. M. V. Rossi, B. M. McManus, and T. J. Podor. Strategies of conditional gene expression in myocardium: an overview. *Methods Mol Med*, 112:109–154, 2005.

[32] Y. Ho, A. Gruhler, A. Heilbut, G. D. Bader, L. Moore, S.-L. Adams, A. Millar, P. Taylor, K. Bennett, K. Boutilier, L. Yang, C. Wolting, I. Donaldson, S. Schandorff, J. Shewnarane, M. Vo, J. Taggart, M. Goudreault, B. Muskat, C. Alfarano, D. Dewar, Z. Lin, K. Michalickova, A. R. Willems, H. Sassi, P. A. Nielsen, K. J. Rasmussen, J. R. Andersen, L. E. Johansen, L. H. Hansen, H. Jespersen, A. Podtelejnikov, E. Nielsen, J. Crawford, V. Poulsen, B. D. S?rensen, J. Matthiesen, R. C. Hendrickson, F. Gleeson, T. Pawson, M. F. Moran, D. Durocher, M. Mann, C. W. V. Hogue, D. Figeys, and M. Tyers. Systematic identification of protein complexes in Saccharomyces cerevisiae by mass spectrometry. *Nature*, 415(6868):180–3, Jan 2002.

[33] L. Hood and D. Galas. The digital code of DNA. *Nature*, 421(6921):444–448, Jan 2003.

[34] L. Hood, J. R. Heath, M. E. Phelps, and B. Lin. Systems biology and new technologies enable predictive and preventative medicine. *Science*, 306(5696):640–643, Oct 2004.

[35] M. Hucka, A. Finney, H. Sauro, H. Bolouri, J. Doyle, H. Kitano, A. Arkin, B. Bornstein, D. Bray, A. Cornish-Bowden, A. Cuellar, S. Dronov, E. Gilles, M. Ginkel, V. Gor, I. Goryanin, W. Hedley, T. Hodgman, J.-H. Hofmeyr, P. Hunter, N. Juty, J. Kasberger, A. Kremling, U. Kummer, N. L. Novre, L. Loew, D. Lucio, P. Mendes, E. Minch, E. Mjolsness, Y. Nakayama, M. Nelson, P. Nielsen, T. Sakurada, J. Schaff, B. Shapiro, T. Shimizu, H. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, and S. Forum. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–31, Mar 2003.

[36] N. Ishii, M. Robert, Y. Nakayama, A. Kanai, and M. Tomita. Toward large-scale modeling of the microbial cell for computer simulation. *J Biotechnol*, 113(1-3):281–294, Sep 2004.

[37] F. JACOB and J. MONOD. Genetic regulatory mechanisms in the synthesis of proteins. *J Mol Biol*, 3:318–56, Jun 1961.

[38] H. Jeong, S. Mason, A. Barab?si, and Z. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–2, May 2001.

[39] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. Gopinath, G. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome: a knowledgebase of biological pathways. *Nucleic Acids Res*, 33(Database issue):D428–32, Jan 2005.

[40] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori. The KEGG resource for deciphering the genome. *Nucleic Acids Res*, 32(Database issue):D277–80, Jan 2004.

[41] P. D. Karp, S. Paley, C. J. Krieger, and P. Zhang. An evidence ontology for use in pathway/genome databases. *Pac Symp Biocomput*, pages 190–201, 2004.

[42] P. D. Karp, S. Paley, and P. Romero. The Pathway Tools software. *Bioinformatics*, 18 Suppl 1:S225–S232, 2002.

[43] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford Universty Press, 1993.

[44] A. Kaufman. Obtaining a functional pathway description from multi-knockout data. In *7th BioPathways Meeting at ISMB 2005*, 2005.

[45] J. D. Keene and P. J. Lager. Post-transcriptional operons and regulons co-ordinating gene expression. *Chromosome Res*, 13(3):327–337, 2005.

[46] I. M. Keseler, J. Collado-Vides, S. Gama-Castro, J. Ingraham, S. Paley, I. T. Paulsen, M. Peralta-Gil, and P. D. Karp. EcoCyc: a comprehensive database resource for Escherichia coli. *Nucleic Acids Res*, 33(Database issue):D334–D337, Jan 2005.

[47] S. Y. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic Bayesian networks. *Brief Bioinform*, 4(3):228–35, Sep 2003.

[48] M. Krull, N. Voss, C. Choi, S. Pistor, A. Potapov, and E. Wingender. TRANSPATH: an integrated database on signal transduction and a tool for array analysis. *Nucleic Acids Res*, 31(1):97–100, Jan 2003.

[49] T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, Z. Bar-Joseph, G. K. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, J. Zeitlinger, E. G. Jennings, H. L. Murray, D. B. Gordon, B. Ren, J. J. Wyrick, J.-B. Tagne, T. L. Volkert, E. Fraenkel, D. K. Gifford, and R. A. Young. Transcriptional regulatory networks in Saccharomyces cerevisiae. *Science*, 298(5594):799–804, Oct 2002.

[50] C. Lemer, E. Antezana, F. Couche, F. Fays, X. Santolaria, R. Janky, Y. Deville, J. Richelle, and S. J. Wodak. The aMAZE LightBench: a web interface to a relational database of cellular processes. *Nucleic Acids Res*, 32(Database issue):D443–D448, Jan 2004.

[51] H. Li, J. Li, S. Tan, and S. Ng. Discovery of binding motif pairs from protein complex structural data and protein interaction sequence data. *Pac Symp Biocomput*, pages 312–23, 2004.

[52] D. F. Lusche, K. Bezares-Roder, K. Happle, and C. Schlatterer. cAMP controls cytosolic Ca2+ levels in Dictyostelium discoideum. *BMC Cell Biol*, 6(1):12, Mar 2005.

[53] D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res*, 33(Database issue):D54–8, Jan 2005.

[54] T. Maniatis and R. Reed. An extensive network of coupling among gene expression machines. *Nature*, 416(6880):499–506, Apr 2002.

[55] V. D. Marinescu, I. S. Kohane, and A. Riva. The MAPPER database: a multi-genome catalog of putative transcription factor binding sites. *Nucleic Acids Res*, 33(Database issue):D91–7, Jan 2005.

[56] V. Matys, E. Fricke, R. Geffers, E. Gssling, M. Haubrock, R. Hehl, K. Hornischer, D. Karas, A. Kel, O. Kel-Margoulis, D.-U. Kloos, S. Land, B. Lewicki-Potapov, H. Michael, R. Mnch, I. Reuter, S. Rotert, H. Saxel, M. Scheer, S. Thiele, and E. Wingender. TRANSFAC: transcriptional regulation, from patterns to profiles. *Nucleic Acids Res*, 31(1):374–8, Jan 2003.

[57] M. A. Matzke and J. A. Birchler. RNAi-mediated pathways in the nucleus. *Nat Rev Genet*, 6(1):24–35, Jan 2005.

[58] J. R. Miller. The Wnts. *Genome Biol*, 3(1):REVIEWS3001, 2002.

[59] B. Modrek and C. Lee. A genomic view of alternative splicing. *Nat Genet*, 30(1):13–19, Jan 2002.

[60] B. Modrek and C. J. Lee. Alternative splicing in the human, mouse and rat genomes is associated with an increased frequency of exon creation and/or loss. *Nat Genet*, 34(2):177–180, Jun 2003.

[61] B. Modrek, A. Resch, C. Grasso, and C. Lee. Genome-wide detection of alternative splicing in expressed sequences of human genes. *Nucleic Acids Res*, 29(13):2850–2859, Jul 2001.

[62] S. H. Munroe. Diversity of antisense regulation in eukaryotes: multiple mechanisms, emerging patterns. *J Cell Biochem*, 93(4):664–671, Nov 2004.

[63] J. A. Papin, T. Hunter, B. O. Palsson, and S. Subramaniam. Reconstruction of cellular signalling networks and analysis of their properties. *Nat Rev Mol Cell Biol*, 6(2):99–111, Feb 2005.

[64] T. Pawson and J. D. Scott. Protein phosphorylation in signaling–50 years and counting. *Trends Biochem Sci*, 30(6):286–290, Jun 2005.

[65] E. Pennisi. Why do humans have so few genes? *Science*, 309(5731):80, Jul 2005.

[66] S. Peri, J. D. Navarro, T. Z. Kristiansen, R. Amanchy, V. Surendranath, B. Muthusamy, T. K. B. Gandhi, K. Chandrika, N. Deshpande, S. Suresh,

B. Rashmi, K. Shanker, N. Padma, V. Niranjan, H. Harsha, N. Talreja, B. Vrushabendra, M. Ramya, A. Yatish, M. Joy, H. Shivashankar, M. Kavitha, M. Menezes, D. R. Choudhury, N. Ghosh, R. Saravana, S. Chandran, S. Mohan, C. K. Jonnalagadda, C. Prasad, C. Kumar-Sinha, K. S. Deshpande, and A. Pandey. Human protein reference database as a discovery resource for proteomics. *Nucleic Acids Res*, 32(Database issue):D497–501, Jan 2004.

[67] C. L. Peterson and M.-A. Laniel. Histones and histone modifications. *Curr Biol*, 14(14):R546–R551, Jul 2004.

[68] L. Raamsdonk, B. Teusink, D. Broadhurst, N. Zhang, A. Hayes, M. Walsh, J. Berden, K. Brindle, D. Kell, J. Rowland, H. Westerhoff, K. van Dam, and S. Oliver. A functional genomics strategy that uses metabolome data to reveal the phenotype of silent mutations. *Nat Biotechnol*, 19(1):45–50, Jan 2001.

[69] A. Regev and E. Shapiro. Cells as computation. *Nature*, 419(6905):343, Sep 2002.

[70] M. S. Rodriguez, C. Dargemont, and F. Stutz. Nuclear export of RNA. *Biol Cell*, 96(8):639–655, Oct 2004.

[71] R. G. Roeder. Transcriptional regulation and the role of diverse coactivators in animal cells. *FEBS Lett*, 579(4):909–15, Feb 2005.

[72] P. Romero, J. Wagg, M. L. Green, D. Kaiser, M. Krummenacker, and P. D. Karp. Computational prediction of human metabolic pathways from the complete human genome. *Genome Biol*, 6(1):R2, 2005.

[73] G. S. Salvesen and J. M. Abrams. Caspase activation - stepping on the gas or releasing the brakes? Lessons from humans and flies. *Oncogene*, 23(16):2774–2784, Apr 2004.

[74] M. Samoilov, S. Plyasunov, and A. P. Arkin. Stochastic amplification and signaling in enzymatic futile cycles through noise-induced bistability with oscillations. *Proc Natl Acad Sci U S A*, 102(7):2310–2315, Feb 2005.

[75] F. Schacherer, C. Choi, U. Gtze, M. Krull, S. Pistor, and E. Wingender. The TRANSPATH signal transduction database: a knowledge base on signal transduction networks. *Bioinformatics*, 17(11):1053–7, Nov 2001.

[76] T. Schlitt and A. Brazma. Modelling gene networks at different organisational levels. *FEBS Lett*, 579(8):1859–66, Mar 2005.

[77] F. Schweighoffer, A. Ait-Ikhlef, A. L. Resink, B. Brinkman, D. Melle-Milovanovic, P. Laurent-Puig, J. Kearsey, and L. Bracco. Qualitative gene profiling: a novel tool in genomics and in pharmacogenomics that deciphers messenger RNA isoforms diversity. *Pharmacogenomics*, 1(2):187–197, May 2000.

[78] E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19 Suppl 1:i264–71, 2003.

[79] B. M. Slepchenko, J. C. Schaff, I. Macara, and L. M. Loew. Quantitative cell biology with the Virtual Cell. *Trends Cell Biol*, 13(11):570–576, Nov 2003.

[80] S. Stamm, S. Ben-Ari, I. Rafalska, Y. Tang, Z. Zhang, D. Toiber, T. A. Thanaraj, and H. Soreq. Function of alternative splicing. *Gene*, 344:1–20, Jan 2005.

[81] L. W. Sumner, P. Mendes, and R. A. Dixon. Plant metabolomics: large-scale phytochemistry in the functional genomics era. *Phytochemistry*, 62(6):817–36, Mar 2003.

[82] Y. Tomari and P. D. Zamore. Perspective: machines for RNAi. *Genes Dev*, 19(5):517–29, Mar 2005.

[83] J. van Helden, A. Naim, C. Lemer, R. Mancuso, M. Eldridge, and S. J. Wodak. From molecular activities and processes to biological function. *Brief Bioinform*, 2(1):81–93, Mar 2001.

[84] J. van Helden, A. Naim, R. Mancuso, M. Eldridge, L. Wernisch, D. Gilbert, and S. J. Wodak. Representing and analysing molecular and cellular function using the computer. *Biol Chem*, 381(9-10):921–935, 2000.

[85] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, J. D. Gocayne, P. Amanatides, R. M. Ballew, D. H. Huson, J. R. Wortman, Q. Zhang, C. D. Kodira, X. H. Zheng, L. Chen, M. Skupski, G. Subramanian, P. D. Thomas, J. Zhang, G. L. G. Miklos, C. Nelson, S. Broder, A. G. Clark, J. Nadeau, V. A. McKusick, N. Zinder, A. J. Levine, R. J. Roberts, M. Simon, C. Slayman, M. Hunkapiller, R. Bolanos, A. Delcher, I. Dew, D. Fasulo, M. Flanigan, L. Florea, A. Halpern, S. Hannenhalli, S. Kravitz, S. Levy, C. Mobarry, K. Reinert, K. Remington, J. Abu-Threideh, E. Beasley, K. Biddick, V. Bonazzi, R. Brandon, M. Cargill, I. Chandramouliswaran, R. Charlab, K. Chaturvedi, Z. Deng, V. D. Francesco, P. Dunn, K. Eilbeck, C. Evangelista, A. E. Gabrielian, W. Gan, W. Ge, F. Gong, Z. Gu, P. Guan, T. J. Heiman, M. E. Higgins, R. R. Ji, Z. Ke, K. A. Ketchum, Z. Lai, Y. Lei, Z. Li, J. Li, Y. Liang, X. Lin, F. Lu, G. V. Merkulov, N. Milshina, H. M. Moore, A. K. Naik, V. A. Narayan, B. Neelam, D. Nusskern, D. B. Rusch, S. Salzberg, W. Shao, B. Shue, J. Sun, Z. Wang, A. Wang, X. Wang, J. Wang, M. Wei, R. Wides, C. Xiao, C. Yan, A. Yao, J. Ye, M. Zhan, W. Zhang, H. Zhang, Q. Zhao, L. Zheng, F. Zhong, W. Zhong, S. Zhu, S. Zhao, D. Gilbert, S. Baumhueter, G. Spier, C. Carter, A. Cravchik, T. Woodage, F. Ali, H. An, A. Awe, D. Baldwin, H. Baden, M. Barnstead, I. Barrow, K. Beeson, D. Busam, A. Carver, A. Center, M. L. Cheng, L. Curry, S. Danaher, L. Davenport, R. Desilets, S. Dietz, K. Dodson, L. Doup, S. Ferriera, N. Garg, A. Gluecksmann, B. Hart, J. Haynes, C. Haynes, C. Heiner, S. Hladun, D. Hostin, J. Houck, T. Howland, C. Ibegwam, J. Johnson, F. Kalush, L. Kline, S. Koduru, A. Love, F. Mann, D. May, S. McCawley, T. McIntosh, I. McMullen, M. Moy, L. Moy, B. Murphy, K. Nelson, C. Pfannkoch, E. Pratts, V. Puri, H. Qureshi, M. Reardon, R. Rodriguez, Y. H. Rogers, D. Romblad, B. Ruhfel, R. Scott, C. Sitter, M. Smallwood, E. Stewart, R. Strong, E. Suh, R. Thomas, N. N. Tint, S. Tse, C. Vech, G. Wang, J. Wetter, S. Williams, M. Williams, S. Windsor, E. Winn-Deen, K. Wolfe, J. Zaveri, K. Zaveri, J. F. Abril, R. Guig, M. J. Campbell, K. V. Sjolander, B. Karlak, A. Kejariwal, H. Mi, B. Lazareva, T. Hatton,

A. Narechania, K. Diemer, A. Muruganujan, N. Guo, S. Sato, V. Bafna, S. Istrail, R. Lippert, R. Schwartz, B. Walenz, S. Yooseph, D. Allen, A. Basu, J. Baxendale, L. Blick, M. Caminha, J. Carnes-Stine, P. Caulk, Y. H. Chiang, M. Coyne, C. Dahlke, A. Mays, M. Dombroski, M. Donnelly, D. Ely, S. Esparham, C. Fosler, H. Gire, S. Glanowski, K. Glasser, A. Glodek, M. Gorokhov, K. Graham, B. Gropman, M. Harris, J. Heil, S. Henderson, J. Hoover, D. Jennings, C. Jordan, J. Jordan, J. Kasha, L. Kagan, C. Kraft, A. Levitsky, M. Lewis, X. Liu, J. Lopez, D. Ma, W. Majoros, J. McDaniel, S. Murphy, M. Newman, T. Nguyen, N. Nguyen, M. Nodell, S. Pan, J. Peck, M. Peterson, W. Rowe, R. Sanders, J. Scott, M. Simpson, T. Smith, A. Sprague, T. Stockwell, R. Turner, E. Venter, M. Wang, M. Wen, D. Wu, M. Wu, A. Xia, A. Zandieh, and X. Zhu. The sequence of the human genome. *Science*, 291(5507):1304–1351, Feb 2001.

[86] H. M. Wain, M. J. Lush, F. Ducluzeau, V. K. Khodiyar, and S. Povey. Genew: the Human Gene Nomenclature Database, 2004 updates. *Nucleic Acids Res*, 32(Database issue):D255–7, Jan 2004. Hugo.

[87] I. Xenarios, L. Salwnski, X. J. Duan, P. Higney, S.-M. Kim, and D. Eisenberg. DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions. *Nucleic Acids Res*, 30(1):303–305, Jan 2002.

[88] X.-J. Yang. Multisite protein modification and intramolecular signaling. *Oncogene*, 24(10):1653–1662, Mar 2005.

Appendix A

Owl Definition

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
     <!ENTITY pat
      "http://cs.bilkent.edu.tr/~emek/patika-pro-ontology.owl#" >
   ]>

<rdf:RDF
xmlns:pat="&pat;"
xmlns:rdf="&rdf;"
xmlns:rdfs="&rdfs;"
xmlns:owl="&owl;"
xmlns:xsd="&xsd;"
xml:base="&pat;"
>
<owl:Ontology rdf:about="">
<rdfs:comment>
This is the ontology for cellular processes,
as used in PATIKA project in OWL format. If
you would like to use any part of this ontology,
please cite Demir, Babur et al 2004.
Visit http://www.patika.org for more information.

PATIKA project is at Bilkent University, Turkey.
All rights reserved, 2003 PATIKA project
</rdfs:comment>
<rdfs:label>Patika Pro Ontology</rdfs:label>
```

```
</owl:Ontology>

<!--*****************************************-->
<!--Section: PID           -->
<!--*****************************************-->

<owl:Class rdf:ID="PID">
<rdfs:comment>
This class represents the primary key for PATIKA objects.
</rdfs:comment>
</owl:Class>

<owl:DatatypeProperty rdf:ID="ID">
<rdfs:domain rdf:resource="#PID"/>
<rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="version">
<rdfs:domain rdf:resource="#PID"/>
<rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>


<!--*****************************************-->
<!--Section: PatikaObject   -->
<!--*****************************************-->

<owl:Class rdf:ID="PatikaObject">
<rdfs:comment>This is the root class for the principle
elements of the Patika ontology. </rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="uniquePID">
<rdfs:comment>Unique PID of this PATIKA object</rdfs:comment>
<rdfs:domain rdf:resource="#PatikaObject"/>
<rdfs:range rdf:resource="#PID"/>
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="name">
<rdfs:domain rdf:resource="#PatikaObject"/>
<rdfs:range rdf:resource="&xsd;string"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="description">
<rdfs:domain rdf:resource="#PatikaObject"/>
<rdfs:range rdf:resource="&xsd;string"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
```

```
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="goTerm">
<rdfs:comment>A GO term associated with this object.</rdfs:comment>
<rdfs:domain rdf:resource="#PatikaObject"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="author">
<rdfs:comment> ID of the author who created this object</rdfs:comment>
<rdfs:domain rdf:resource="#PatikaObject"/>
<rdfs:range rdf:resource="&xsd;int"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>

<!--*****************************************-->
<!--Section 2: Interactions   -->
<!--*****************************************-->

<owl:Class rdf:ID="Interaction">
<rdfs:comment> Interaction</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#PatikaObject"/>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="targetNode">
<rdfs:comment>Target Node of this interaction</rdfs:comment>
<rdfs:domain rdf:resource="#Interaction"/>
<rdfs:range rdf:resource="#PatikaNode"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<owl:inverseOf rdf:resource="#inEdges"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="sourceNode">
<rdfs:comment>Source Node of this interaction</rdfs:comment>
<rdfs:domain rdf:resource="#Interaction"/>
<rdfs:range rdf:resource="#PatikaNode"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<owl:inverseOf rdf:resource="#outEdges"/>
</owl:ObjectProperty>

<!--*****************************************-->
<!--Section 2.1: Mechanistic Interactions   -->
<!--*****************************************-->

<owl:Class rdf:ID="MechanisticInteraction">
<rdfs:comment>
Mechanistic interactions define relations between mechanistic nodes
at the chemical level of detail.
```

```
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#Interaction"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#sourceNode"/>
<owl:allValuesFrom rdf:resource="#MechanisticNode"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#targetNode"/>
<owl:allValuesFrom rdf:resource="#MechanisticNode"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ReactionInteraction">
<rdfs:comment>
Abstract interaction class to define relations between states and
transitions
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#MechanisticInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PreTInteraction">
<rdfs:comment>
Abstract interaction class for defining interactions that has a state
as their source and transition as their target.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#ReactionInteraction"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#sourceNode"/>
<owl:allValuesFrom rdf:resource="#State"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#targetNode"/>
<owl:allValuesFrom rdf:resource="#Transition"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Substrate">
<rdfs:comment>
A substrate relation is a pre transition interaction, which indicates
that the state is consumed by the transition.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#PreTInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="exhaustive">
<rdfs:comment>Indicates whether activation of the target transition
exhausts the substrate</rdfs:comment>
<rdfs:domain rdf:resource="#Substrate"/>
<rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="Activator">
<rdfs:comment>
An activator relation is a pre transition interaction. It describes
the enabling or facilitating of the transition via the source state.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#PreTInteraction"/>
</rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:ID="Inhibitor">
<rdfs:comment>
An inhibitor relation is a pre transition interaction. It describes
the disabling or impeding of the transition via the source state.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#PreTInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Product">
<rdfs:comment>
A product relation is a directed relation with a transition as its
source and a state as its target. It indicates that the state is
produced by the transition.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#ReactionInteraction"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#sourceNode"/>
```

```
<owl:allValuesFrom rdf:resource="#Transition"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#targetNode"/>
<owl:allValuesFrom rdf:resource="#State"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:ID="Bind">
<rdfs:comment>
A bind relation is an undirected relation with two
complex members as their source and target. It
describes a non-covalent bonding between these two
states. If all binding relations were known for a
complex, then the graph defined by binding relations
and members would be connected.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#MechanisticInteraction"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#sourceNode"/>
<owl:allValuesFrom rdf:resource="#ComplexMemberState"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#targetNode"/>
<owl:allValuesFrom rdf:resource="#ComplexMemberState"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="Stochiometry">
<rdfs:comment>Stochiometry indicates the relative number of
chemicals that are produced/consumed per reaction</rdfs:comment>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Substrate"/>
<owl:Class rdf:about="#Product"/>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
```

```
<rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<!--*****************************************-->
<!--Section 2.1: Bioentity Interactions    -->
<!--*****************************************-->

<owl:Class rdf:ID="BioentityInteraction">
<rdfs:comment>
Bioentity interactions describe relations between bioentities
but not states. They represent incomplete information, and
always map to one or more mechanistic level interaction,
although latter one might not be identified yet.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#Interaction"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#sourceNode"/>
<owl:allValuesFrom rdf:resource="#BioEntity"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#targetNode"/>
<owl:allValuesFrom rdf:resource="#BioEntity"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PPI">
<rdfs:comment>
PPI is an undirected relation, indicating that two
proteins are observed to interact with each other in
a Y2H or co-precipitation system, i.e. there is at
least one state of entity A that somehow interacts
with B. One or more mechanistic level relations might
be associated with this entity level relation.

For example a state 1 of protein A might be bound by
protein B, where state 2 of protein A might be bound
and cleaved by B. Even the nature of the chemical
reaction does not necessarily be same/similar.
Compartment information and n-ary relations can not
be captured by PPI. Some sample databases that
contain PPI data include incyte, DIP, BIND, IntAct,
PIM, ProNet and Mint.
</rdfs:comment>
<rdfs:subClassOf>
```

```
<owl:Class rdf:about="#BioentityInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Other">
<rdfs:comment>
Other is a reserved for cases that does not fit the other
interaction types, such as literature cooccurence, or
microarray clusters.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#BioentityInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="GeneticRegulation">
<rdfs:comment>
GR is a directed relation, indicating that at least
one state of source node activates/inhibits expression
of at least one DNA state of the target. Although there
is combinatorial information on GR, we are yet to
incorporate this to our ontology. Some sample databases
that contain GR data include Transfac, RegulonDB and  ooTFD.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#BioentityInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Derived">
<rdfs:comment>
Derived is an undirected relation derived from mechanistic graph.
</rdfs:comment>
<rdfs:subClassOf>
<owl:Class rdf:about="#BioentityInteraction"/>
</rdfs:subClassOf>
</owl:Class>

<!--****************************************-->
<!--Section: Patika Node    -->
<!--****************************************-->

<owl:Class rdf:ID="PatikaNode">
<rdfs:subClassOf>
<owl:Class rdf:about="#PatikaObject"/>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="inEdges">
<rdfs:comment>Interactions coming to this node</rdfs:comment>
```

```
<rdfs:range rdf:resource="#Interaction"/>
<rdfs:domain rdf:resource="#PatikaNode"/>
<owl:inverseOf rdf:resource="#targetNode"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="outEdges">
<rdfs:comment>Interactions coming to this node</rdfs:comment>
<rdfs:range rdf:resource="#Interaction"/>
<rdfs:domain rdf:resource="#PatikaNode"/>
<owl:inverseOf rdf:resource="#sourceNode"/>
</owl:ObjectProperty>


<!--******************************************-->
<!--Section: Bioentity    -->
<!--******************************************-->

<owl:Class rdf:ID="BioEntity">
<rdfs:subClassOf rdf:resource="#PatikaNode"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#inEdges"/>
<owl:allValuesFrom rdf:resource="#BioentityInteraction"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#outEdges"/>
<owl:allValuesFrom rdf:resource="#BioentityInteraction"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="states">
<rdfs:comment>States of this bioentity</rdfs:comment>
<rdfs:range rdf:resource="#SimpleState"/>
<rdfs:domain rdf:resource="#BioeEntity"/>
<owl:inverseOf rdf:resource="#ownerBioEntity"/>
</owl:ObjectProperty>


<!--******************************************-->
<!--Section: Mechanistic Node    -->
<!--******************************************-->
<owl:Class rdf:ID="MechanisticNode">
<rdfs:subClassOf rdf:resource="#PatikaNode"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#inEdges"/>
<owl:allValuesFrom rdf:resource="#MechanisticInteraction"/>
```

```
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#outEdges"/>
<owl:allValuesFrom rdf:resource="#MechanisticInteraction"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="ownerAbstractions">
<rdfs:comment>Abstractions that own this node</rdfs:comment>
<owl:inverseOf rdf:resource="#abstractionMembers"/>
<rdfs:range rdf:resource="#Abstraction"/>
<rdfs:domain rdf:resource="#MechanisticNode"/>
<rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>

<!--*****************************************-->
<!--Section: Abstraction    -->
<!--*****************************************-->
<owl:Class rdf:ID="Abstraction">
<rdfs:subClassOf rdf:resource="#MechanisticNode"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="abstractionMembers">
<rdfs:domain rdf:resource="#Abstraction"/>
<rdfs:range rdf:resource="#MechanisticNode"/>
<rdf:type rdf:resource="&owl;TransitiveProperty"/>
<owl:inverseOf rdf:resource="#ownerAbstractions"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="RegularAbstraction">
<rdfs:subClassOf rdf:resource="#Abstraction"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#inEdges"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
0
</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#outEdges"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
0
</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
```

```
</owl:Class>

<owl:Class rdf:ID="IncompleteAbstraction">
<rdfs:subClassOf rdf:resource="#Abstraction"/>
</owl:Class>

<owl:Class rdf:ID="HomologyAbstraction">
<rdfs:subClassOf rdf:resource="#Abstraction"/>
</owl:Class>


<!--******************************************-->
<!--Section: State    -->
<!--******************************************-->

<owl:Class rdf:ID="State">
<rdfs:subClassOf rdf:resource="#MechanisticNode"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#outEdges"/>
<owl:allValuesFrom>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#PreTInteraction"/>
<owl:Class rdf:about="#Bind"/>
</owl:unionOf>
</owl:Class>
</owl:allValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#inEdges"/>
<owl:allValuesFrom>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Product"/>
<owl:Class rdf:about="#Bind"/>
</owl:unionOf>
</owl:Class>
</owl:allValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="CompoundState">
<rdfs:subClassOf rdf:resource="#State"/>
</owl:Class>

<owl:Class rdf:ID="HomologyState">
```

```
<rdfs:subClassOf rdf:resource="#HomologyAbstraction"/>
<rdfs:subClassOf rdf:resource="#CompoundState"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#abstractionMembers"/>
<owl:allValuesFrom rdf:resource="#SimpleState"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="IncompleteState">
<rdfs:subClassOf rdf:resource="#IncompleteAbstraction"/>
<rdfs:subClassOf rdf:resource="#CompoundState"/>
</owl:Class>

<owl:Class rdf:ID="Complex">
<rdfs:subClassOf rdf:resource="#CompoundState"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="complexMembers">
<rdfs:domain rdf:resource="#Complex"/>
<rdfs:range rdf:resource="#ComplexMemberState"/>
<owl:inverseOf rdf:resource="#ownerComplex"/>
</owl:ObjectProperty>

<!--******************************************-->
<!--Section: Simple State    -->
<!--******************************************-->

<owl:Class rdf:ID="SimpleState">
<rdfs:subClassOf rdf:resource="#State"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="ownerBioEntity">
<rdfs:comment>Owner bioentity of this state</rdfs:comment>
<rdfs:range rdf:resource="#BioEntity"/>
<rdfs:domain rdf:resource="#SimpleState"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<owl:inverseOf rdf:resource="#states"/>
</owl:ObjectProperty>
<!--******************************************-->
<!--Section: State Variables    -->
<!--******************************************-->
<owl:DatatypeProperty rdf:ID="chemicalType">
<rdfs:comment>Chemical type of this state</rdfs:comment>
<rdfs:domain rdf:resource="#SimpleState"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<rdfs:range>
<owl:DataRange>
<owl:oneOf>
```

```
<rdf:List>
<rdf:first rdf:datatype="&xsd;string">DNA</rdf:first>
<rdf:rest>
<rdf:List>
<rdf:first rdf:datatype="&xsd;string">RNA</rdf:first>
<rdf:rest>
<rdf:List>
<rdf:first rdf:datatype="&xsd;string">Protein</rdf:first>
<rdf:rest>
<rdf:List>
<rdf:first rdf:datatype="&xsd;string">
Physical Factor
 </rdf:first>
<rdf:rest>
<rdf:List>
<rdf:first rdf:datatype="&xsd;string">
Small Molecule
</rdf:first>
<rdf:rest rdf:resource="&rdf;nil"/>
</rdf:List>
</rdf:rest>
</rdf:List>
</rdf:rest>
</rdf:List>
</rdf:rest>
</rdf:List>
</rdf:rest>
</rdf:List>
</owl:oneOf>
</owl:DataRange>
</rdfs:range>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="insideRegion">
<rdfs:comment>The region which this state resides in</rdfs:comment>
<rdfs:range rdf:resource="#Region"/>
<rdfs:domain rdf:resource="#SimpleState"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="dockedMembrane">
<rdfs:comment>The membrane which this state is docked</rdfs:comment>
<rdfs:range>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Membrane"/>
<owl:Class rdf:about="#MembraneSubregion"/>
</owl:unionOf>
</owl:Class>
</rdfs:range>
```

```
<rdfs:domain rdf:resource="#SimpleState"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>


<owl:Class rdf:ID="ComplexMemberState">
<rdfs:subClassOf rdf:resource="#SimpleState"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="ownerComplex">
<rdfs:domain rdf:resource="#ComplexMemberState"/>
<rdfs:range rdf:resource="#Complex"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<owl:inverseOf rdf:resource="#complexMembers"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="UbiqueState">
<rdfs:subClassOf rdf:resource="#SimpleState"/>
</owl:Class>

<!--*****************************************-->
<!--Section: Transition     -->
<!--*****************************************-->

<owl:Class rdf:ID="Transition">
<rdfs:subClassOf rdf:resource="#MechanisticNode"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#inEdges"/>
<owl:allValuesFrom rdf:resource="#PreTInteraction"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#outEdges"/>
<owl:allValuesFrom rdf:resource="#Product"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="IncompleteTransition">
<rdfs:subClassOf rdf:resource="#IncompleteAbstraction"/>
<rdfs:subClassOf rdf:resource="#Transition"/>
</owl:Class>

<owl:Class rdf:ID="HomologyTransition">
<rdfs:subClassOf rdf:resource="#HomologyAbstraction"/>
<rdfs:subClassOf rdf:resource="#Transition"/>
<rdfs:subClassOf>
<owl:Restriction>
```

```
<owl:onProperty rdf:resource="#abstractionMembers"/>
<owl:allValuesFrom rdf:resource="#Transition"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!--*****************************************-->
<!--Section: Compartments     -->
<!--*****************************************-->
<owl:Class rdf:ID="Region">
<rdfs:comment>
A region is a cellular location in its broadest sense
</rdfs:comment>
</owl:Class>

<owl:DatatypeProperty rdf:ID="RegionName">
<rdfs:comment>Name of this region</rdfs:comment>
<rdfs:domain rdf:resource="#Region"/>
<rdfs:range rdf:resource="&xsd;string"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="RegionID">
<rdfs:comment>Unique ID of this region</rdfs:comment>
<rdfs:domain rdf:resource="#Region"/>
<rdfs:range rdf:resource="&xsd;string"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="Compartment">
<rdfs:comment>
A compartment is a region with a well defined boundary.
Membranes and membrane enclosed spaces are compartments.
Compartments are neighbor to each other.</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Region"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasSubregion">
<rdfs:domain rdf:resource="#Compartment"/>
<rdfs:range rdf:resource="#Subregion"/>
<owl:inverseOf rdf:resource="#subregionOf"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Membrane">
<rdfs:comment>
A membrane is a compartment, that forms a closed lipid sack.
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Compartment"/>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="encapsulates">
<rdfs:comment>
Unique space encapsulated by this membrane.
Encapsulates relation implies neighborhood
</rdfs:comment>
<rdfs:domain rdf:resource="#Membrane"/>
<rdfs:range rdf:resource="#Space"/>
<owl:inverseOf rdf:resource="#encapsulatedBy"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="insideOf">
<rdfs:comment>
Unique space this membrane is insideof.
Insideof relation implies neighborhood
</rdfs:comment>
<rdfs:domain rdf:resource="#Membrane"/>
<rdfs:range rdf:resource="#Space"/>
<owl:inverseOf rdf:resource="#contains"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Space">
<rdfs:comment>
A space is a compartment, encapsulated by a membrane.
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Compartment"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="encapsulatedBy">
<rdfs:comment>
Unique compartment encapsulating this membrane.
EncapsulatedBy relation implies neighborhood
</rdfs:comment>
<rdfs:domain rdf:resource="#Space"/>
<rdfs:range rdf:resource="#Membrane"/>
<owl:inverseOf rdf:resource="#encapsulates"/>
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="contains">
<rdfs:comment>
Membranes inside this space. Contains relation implies neighborhood
</rdfs:comment>
<rdfs:domain rdf:resource="#Space"/>
<rdfs:range rdf:resource="#Membrane"/>
<owl:inverseOf rdf:resource="#contains"/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="Subregion">
<rdfs:comment>
A loosely defined region, must be a subregion of a compartment.
A subregion does not have neighborhood relations but inherits
all neighborhood relations of its compartment.
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Region"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="subregionOf">
<rdfs:domain rdf:resource="#Subregion"/>
<rdfs:range rdf:resource="#Compartment"/>
<owl:inverseOf rdf:resource="#hasSubregion"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="MembraneSubregion">
<rdfs:comment>A subregion of a membrane</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Subregion"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#subregionOf"/>
<owl:allValuesFrom rdf:resource="#Membrane"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="SpaceSubregion">
<rdfs:comment>A subregion of a space</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Subregion"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#subregionOf"/>
<owl:allValuesFrom rdf:resource="#Space"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="neighborOf">
<rdfs:comment>
Extra property for defining neighborhoods that does not fit to
inclusion paradigm. e.g. ER
</rdfs:comment>
<rdfs:domain rdf:resource="#Compartment"/>
<rdfs:range rdf:resource="#Compartment"/>
<rdf:type rdf:resource="&owl;SymmetricProperty"/>
</owl:ObjectProperty>


<!--******************************************-->
```

```
<!--Section: Compartments     -->
<!--*****************************************-->

</rdf:RDF>
```