Visualization for Biological Models, Simulation, and Ontologies

Gary Yngve

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2007

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Gary Yngve

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Linda Shapiro

Reading Committee:

_____

Jim Brinkley

_____

Dan Cook

_____

Linda Shapiro

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Visualization for Biological Models, Simulation, and Ontologies

Gary Yngve

Chair of the Supervisory Committee:
Professor Linda Shapiro
Computer Science and Engineering

In this dissertation, I present three browsers that I have developed for the purpose of exploring, understanding, and analyzing models, simulations, and ontologies in biology and medicine. The first browser visualizes multidimensional simulation data as an animation. The second browser visualizes the equations of a complex model as a network and puts structure and organization on top of equations and variables. The third browser is an ontology viewer and editor, directly intended for the Foundational Model of Anatomy (FMA), but applicable to other ontologies as well. This browser has two contributions. First, it is a lightweight deliverable that lets someone easily dabble with the FMA. Second, it lets the user edit an ontology to create a view of it. For the ontology browser, I also conduct user studies to refine and evaluate the software.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

The modern computer age is changing the way doctors, engineers, scientists, and even economists and social scientists, do research. Previously, they would formulate hypotheses on theoretical models and test them with *in vivo* or *in vitro* experiments. Their models and data would be represented using their own notations, making collaboration and sharing difficult. Now computers are at the center of the confluence of models, experiment data, and knowledge representation. Computer experiments are run on models via simulation, and every piece of the model or the experiment data can be represented by a controlled vocabulary. Further rules and semantics grant this vocabulary considerable reasoning power.

Simulation opens many possibilities for experimentation *in silico*, especially when otherwise researchers would face problems with time-scale, excessive cost or causing harm to the test subjects, and having limited control or observations. A computer simulation consists of a theoretical model that is realized in code and a set of inputs to define system parameters and initial conditions. The output of the simulation is for every point in time, the entire state of the system. Conducting a simulation creates two challenges for researchers, whose backgrounds may not be in computer science or applied math: they need to be able to transfer their knowledge of the theoretical models and experiments they would like to perform to something computers can read, and they need to be able to interpret the floods of data coming out of the simulators. This work addresses these two problems in the context of biological simulation.

Additionally, data acquisition and the abilities to store and process such data

are progressing so fast in biology and medicine that scientists, doctors, and health-care workers are struggling to keep pace, both in terms of being overwhelmed by the mountains of data and in being unaware of how truly powerful the new technology can be. Having a controlled vocabulary means that data so annotated can be universally shared. The challenges are in developing the controlled vocabulary and getting it adopted by users. The latter challenge is two-fold: people want the gains of a controlled vocabulary without the expense of converting their legacy data to it, and controlled vocabularies are often too cumbersome and not customized enough for the average user.

The computational aspects of biological research today intertwine models, data, and knowledge. An example is the work by Kalet et al.[25] on modeling the spread of cancer. Using knowledge of the connectivity of the lymphatic system and a model of how the cancer would spread, they can produce data that predicts the spread of the cancer. Furthermore, their results could be validated against clinical data. Going one step further, they could use their knowledge of anatomy, their predictions of tumor spread, and a model of the effects of radiation treatment to plan how to target the radiation. The goal would be to maximize damage to cancerous cells while minimizing damage to critical areas. Figure 1.1 illustrates the interconnectedness of data, models, and knowledge for a biologist. As biologists are not expected to be experts in the allied fields of math and computer science, there is a cognitive gap between their understanding of data, models, and knowledge and their computational representations. Reducing this gap would enable biologists to be more productive.

1.1 Motivation

Biological simulation is a fast-growing field today with a wide range of applications. Companies such as AneSoft, METI, Immersion, and Simuvision build simulation environments to teach and evaluate both knowledge and motor skills for surgery and anesthesiology. Simulations let students practice infrequent scenarios and learn in en-

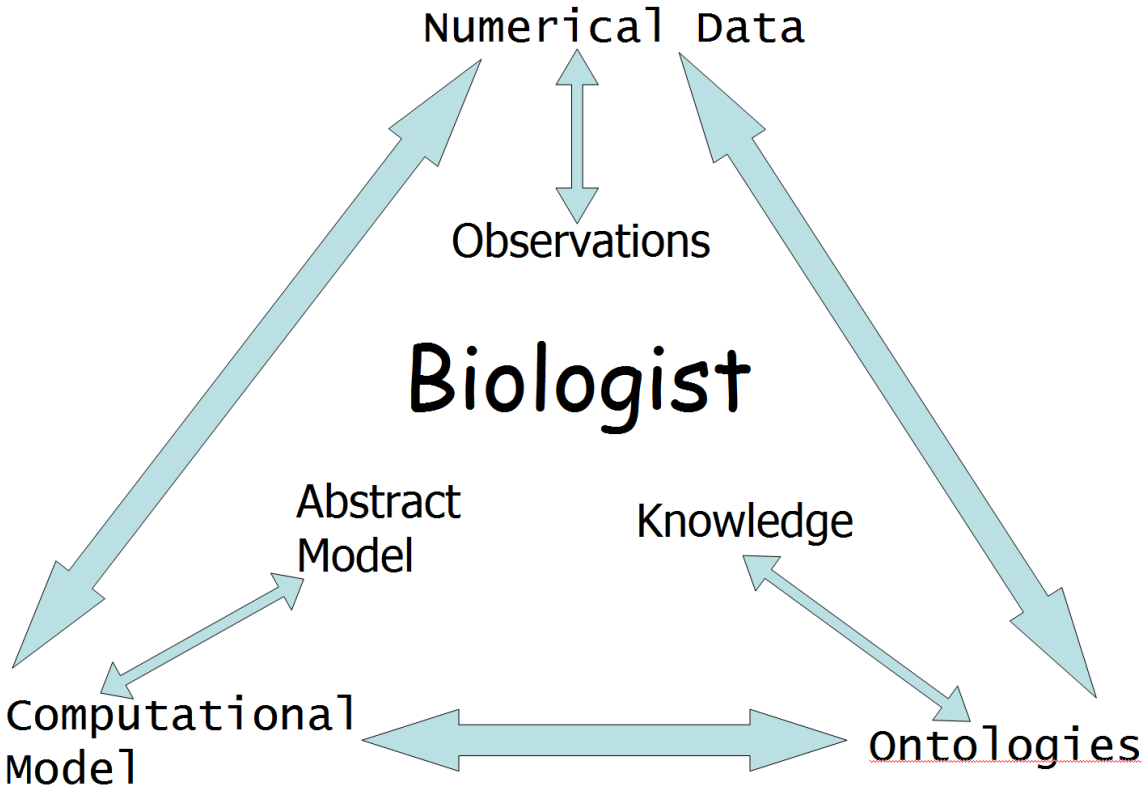

Figure 1.1: Biological research involves synergy of data, models, and knowledge. Each of these areas has a more abstract representation, which a person thinks and reasons about, and a more computational representation, which is what the computer uses. A challenge for the biologist is to bridge the cognitive gap between the abstract representation and the computational representation.

vironments in which failure has no consequences. Researchers are using simulations to study proteins and develop new drugs. They want to use simulation to predict outcomes on patients and design custom optimal treatments. Simulation would allow for virtual clinical trials. Government funding reports have identified these key problems and have declared simulation and visualization as crucial research fields[24, 36]. Simulation has other strong applications beyond biology from engineering to economics. Likewise, clinical devices such as ICU monitors that are cluttered with patient data can benefit from visualization[1].

The virtual human is the holy grail of biological simulation. The ideal model would give the illusion of simulating every molecule in every cell of every tissue of every organ while cleverly simplifying and approximating to make the calculations run faster than real time. Researchers have a long way to go, and the mass of data pumped out of current simulators requires new informatics research to help manage it and new visualization research to help understand it. Researchers across the world, including many who are part of the Physiome Project[1], are building independent small models that others may later want to connect. These models may not have much documentation or a good design for interfacing with other models; they may also be written in a variety of languages such as CellML[2] and SBML[3]. Visualization is a necessary piece of the puzzle for building toward the digital human.

An ontology consists of a controlled vocabulary and a set formal relationships and rules between its terms. They are becoming increasingly popular in the corporate world, with companies such as Ontolica, Ontopia, and SchemaLogic marketing knowledge-base solutions. Ontologies for biology let researchers communicate their models and data with each other, no matter their language or discipline. Furthermore, computers can understand the formalities of an ontology and make deductions.

---

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

[2]http://www.cellml.org

[3]http://www.sbml.org

With respect to simulation of the virtual human discussed previously, variables in the model, clinical data, and experimental results can all be catalogued using an ontology. A computer could use the knowledge to generate a model or code. Outside of simulation, being able to catalog an immense amount of uniformly annotated data from distributed sources is immensely valuable. Annotated radiological images could be used for educating young radiologists. Doctors could query a huge database, asking for patient histories for say, patients with colon cancer with a certain cell type and stage, who are also in this gender and age group and have the following chronic conditions. Without being able to assimilate data from so many sources, people would not be able to amass enough to have a chance of extracting the similar cases they need. Researchers using data-mining and machine-learning techniques would appreciate such a pool of data, as more data means better results. Finally, an individual patient's entire medical records, including imagery, could be easily shared and distributed between people who need it.

1.2 Problem Statement

Although there are strong connections between ontologies and biosimulation, these two will be treated separately when defining what problems this disseration addresses.

1.2.1 Biological Models

For biological models, the focus of this work is on visualizing lumped-parameter simulations of systemic biology within a moderate time scale. Lumped-parameter models network a set of homogenized components, which are either single units treated atomically, such as the left ventricle of the heart, or aggregates treated atomically, such as all the alveoli in the lungs. Lumps may be of vastly different scales, from molecular to organ. The networks represent behavior from fluid dynamics to chemical reactions. Whereas much recent research has focused on visualization for the specific areas of genomics and proteomics, this research addresses the full spatial spectrum

by abstracting to more of an information-visualization approach. It does not focus on multiscale time (protein-folding simulations have timesteps on the order of 10^{-15} seconds, whereas human lifespan is on the order of 10^9 seconds), because multiscale time is very much a challenge for modeling and simulating. Visualization is challenging enough without this extra requirement.

A researcher thinks about a model abstractly as approximations and assumptions of knowledge about anatomy and physiology such that certain behavior can be feasibly simulated. The researcher usually makes an intermediary model such as a lumped network to aid in the task of turning the model into code. Automation via the Foundational Model of Anatomy (FMA)[40] and other ontologies will someday handle the task of building the intermediary model. There exist icon-based model-construction tools today that turn an intermediary model into code; however, they are not generalizable or flexible enough for complex models. Most likely, the researcher writes code for the computational representation of the model. Ideally the code is high-level, and a compiler converts the high-level code into optimized low-level code. For example, JSim[4] has a compiler that translates the Mathematical Modeling Language into Java bytecode.

In addition to going forward from abstract to computational, a researcher would like to go backward. For example, if perusing the code from someone else's model, a researcher might want to find out quickly what assumptions were made about physiology. With the model browser presented in this disseration, the researcher will be able to find the answer. During the process of debugging a model, the researcher may want to mentally invert the code into the intermediary model or isolate smaller parts of the model. Software, such as the model browser presented in this dissertation, can assist one's mind in these tasks.

---

[4]http://www.physiome.org/jsim

1.2.2 Data

Researchers obtain numerical data from a variety of sources, including telemetry, imagery, and simulation. In a strictly computational sense, multidimensional time-varying data is a matrix of numbers. The cognitive gap that needs to be bridged is for the numbers to be interpreted as observations. Simply visualizing all the time-varying data as curves, either adjacent or superimposed, quickly runs out of room or becomes too cluttered. Data needs to be organized, and perhaps even pre-analyzed by the computer. Additionally, the user may discover and understand relationships better through a higher dimensional visualization or through a visualization that incorporates the context of the model, such as by using the knowledge of anatomy. Animation with the animated data browser presented in this disseration adds a spatial dimension that was previously reserved for time and can bring a simulation to life.

1.2.3 Ontologies

An ontology is a controlled vocabulary of terms with formal relationships between the terms. Each term can be composed of other terms, and the rules governing the relationships can be defined in the ontology as well. Often the terms will have a taxonomy that allows traits to be inherited. At a minimum, an ontology can be represented crudely in the resource description framework (RDF) as subject-predicate-object triples, or with full rules and inferences in the web ontology language (OWL). An ontology enables the data it describes by supporting reasoning and data-mining, as well as standardizing nomenclatures across the world. A fundamental challenge is to combine domain-based ontologies into an overarching semantic web. An alternate, and likely easier, approach is to build foundational ontologies from which people can extract application ontologies for their specific domains.

An application ontology is derived from one or more foundational ontologies, with the ontologies predominantly orthogonal to each other. The application ontology

would only contain a small subset of relevant information, both for efficiency and so as not to overwhelm the user in the subdomain. Furthermore, the application ontology may contain subdomain-specific references not present in the foundational ontologies, or it may contain an extra framework to tie the foundational ontologies together. For this thesis, application ontologies deriving from only a single foundational ontology were considered, to avoid name-collision problems. The first research problem is designing the infrastructure to support queries and edits of the application ontologies. The infrastructure should represent the application as a view layer; that is, the new ontology is not necessarily materialized. Rather queries to the application ontology undergo a transformation, and data from all of its sources are assimilated together to produce the results, as if the application ontology were materialized. This construction makes it easy to chain layers of application ontology views on top of each other. The second research problem is to develop an effective visualization for creating such an application ontology view. The visualization should give interactive feedback, despite the challenge of operating on ontologies too large to handle all at once. The visualization should be accompanied with powerful search and query features and possess features expected of any commercial application, such as robust saving, loading, undoing, and help. The ontology browser effectively visualizes ontologies and enables users to construct ontology views.

Another roadblock is that individual labs or departments likely have their own legacy formats and notations, and for reasons of convenience, they do not want to spend the effort to standardize. To spur their efforts, ontology researchers should both 1) try to reach out to them and explain how powerful and useful the ontologies are, and 2) try to make the adoption of an ontology as painless as possible. To communicate the power of ontologies to a biologist or doctor, one must know how the biologist or doctor could use it. Fundamentally, this dilemma is a chicken-and-egg problem. Until they possess the technology and find uses for it, informaticists do not know how it is going to be used; likewise they are not going to waste the time to

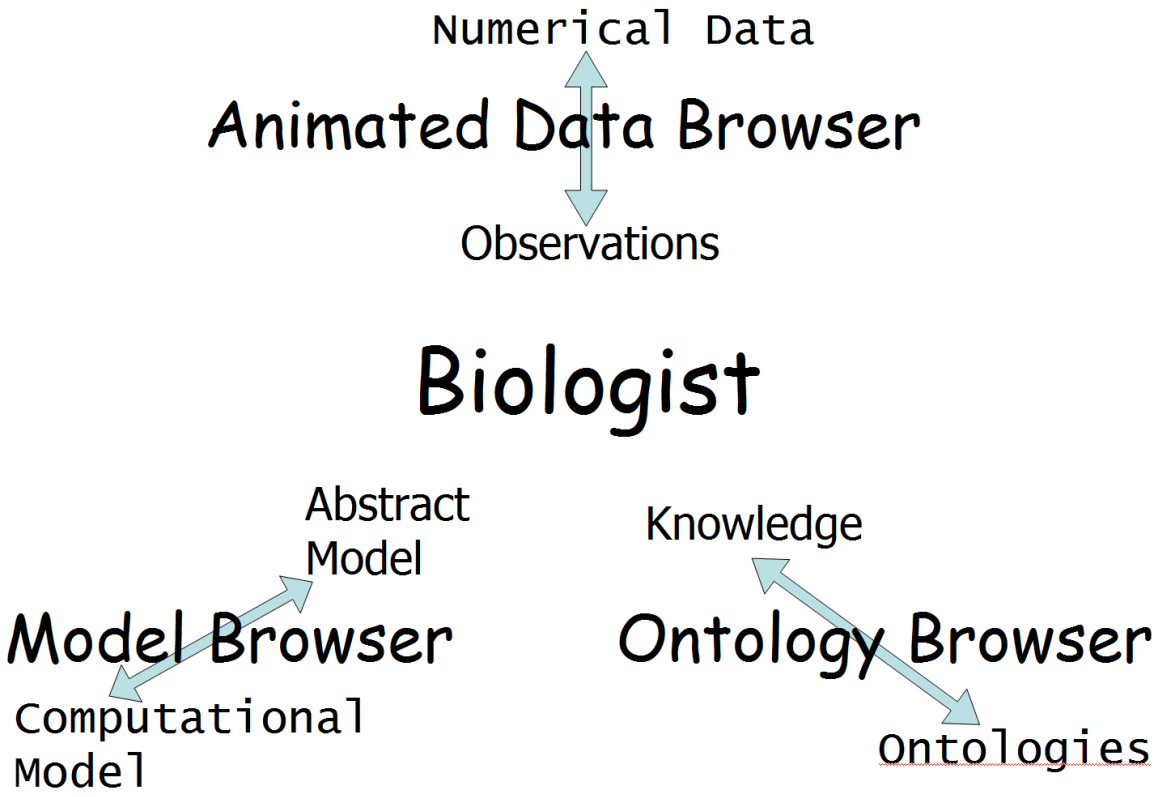

Figure 1.2: The animated data browser, model browser, and ontology browser presented in this dissertation reduce the impedance between thinking about the abstract and its computational equivalent.

adopt new technology if there is not a use for it. The ontology browser presented in this dissertation specifically eliminates the inconveniences associated with using earlier deliveries of the Foundational Model of Anatomy.

1.3 Contributions

This dissertation presents three new browsers designed to bridge the cognitive gaps between human-understandable representations of models, data, and knowledge, and their computational equivalents. The thesis is that these browsers make it easier for biologists to understand and analyze the computational representations. Figure 1.2

illustrates the roles of the three browsers. The animated data browser and model browser are justified by presenting questions about numerical data or computational models and showing how they could be answered using the browsers. The ontology browser has additionally undergone several iterations of feedback and improvements, including a qualitative user study.

The model browser imparts structure and meaning onto what is often unstructured code by transforming variables, metadata, and equations into an interactive visualization. The browser is a novel application to biomedical informatics and offers many possible directions for future research. The browser also supports the hierarchical mapping of variables to a taxonomy.

The ontology browser stands apart from related work by scaling well with respect to the size of the ontology. In addition to being a browser, it also is an editor for creating views of an ontology. The browser manages what has been loaded from the back end, what the user has looked at, and what the user should see given what has been loaded and looked at. The result for the user is a seamless experience that preserves context and has maximum relevance. The browser also includes powerful search and query tools, as well as interactive tutorials.

In addition to the user interface, the ontology browser has a compact representation for ontologies that can offer in excess of a 90% reduction in size for the underlying data. This compression allows ontologies to be delivered as a small payload without additional support software such as databases. The effect on the user is that the barrier of entry to exploring an ontology is greatly reduced. The belief is that the easier access will encourage the biomedical community to incorporate ontologies into their work, which will later reap great benefits.

A substantial contribution beyond the research and creation of the browsers is the engineering necessary to elevate them to the level of commercial software, both in feature sets and robustness. The model browser is in the process of being integrated

into the JSim[5] simulation package, which is part of the Physiome project. The ontology browser is planned to be adopted as a means for lightweight delivery for other ontologies served from the National Center for Biomedical Ontology[6].

1.4 Synopsis

In the rest of this thesis, the relevant related work in the areas of biological modeling and simulation, bioinformatics, ontologies, visualization, and usability is first presented. Next the animated data browser, and then the model browser are introduced. Before the ontology browser is described, the theory and architecture beneath it, which are essential to understanding the design choices in the interface and how it works, are explained. Finally conclusions and future work, much of which is presented specifically in the individual chapters, are discussed. The following subsections summarize the browsers in more detail.

1.4.1 Animated Data Browser

The animated data browser presents multidimensional simulation data from multiple parts as an animation. Coping with data at different scales, both spatially and temporally, is a challenge that this disseration addresses. The browser also produces clear detailed views of multidimensional data, including up to three simultaneous dimensions.

1.4.2 Model Browser

The model browser uses the compilation information from a mathematical model to produce a graph of the dependencies. Variables are enhanced with metadata, which can be used to color-code, filter, or cluster. Sets of variables can be collapsed into a

[5]http://www.physiome.org/jsim

[6]http://www.bioontology.org

single merged variable, using the metadata. Furthermore, if a full hierarchy is available, the layout can take advantage of special hierarchical techniques. The browser grants quick access to the equations associated with variables or sets of variables. The user can browse pathways of variables to specifically study the connectivity of parts of the model.

1.4.3 Ontology Browser

The ontology browser consists of an underlying architecture to support the notion of chained compact view layers and a visualization on top. The visualization supports quick navigation through the ontology, as well as defining of the views by editing the ontology. It is supplemented with powerful search and query features. The browser has been tested on a use case and is currently undergoing usability evaluations. In addition, a compressed representation of the Foundational Model of Anatomy has been developed; it enables the ontology plus the software to be delivered in total as a small payload. The software runs under Java 1.5, making it both lightweight and platform-independent.

Chapter 2

RELATED WORK

In the following sections the related work on bioinformatics, biosimulation, ontologies, and visualization is reviewed.

2.1 Models, simulation, and visualization

Computer simulation of a model has revolutionized science by bridging the gap between theory and experimentation. Dowling[18] describes simulation as a "theory," as the model is a mathematical representation, not reality . She also describes simulation as an "experiment," as the scientist can fiddle with parameters and observe results. She writes, "A sense of direct manipulation encourages simulators to develop a 'feel' for their mathematical models with their hands and their eyes, by *tinkering* with them, *noticing* how they behave, and developing a practical *intuition* for how they work." It is the job of visualization to aid in exploration and intuition.

Tufte's seminal volumes gets to the heart of how to make effective visualizations. Though many of his examples predate computers, concepts such as looking at the ink-to-information ratio still apply. He gives countless examples where tick marks or grid lines are either superfluous or overbearing and offers suggestions from using lighter grays to eliminating lines all together. Tufte notes that interesting data is inherently multivariate and devotes a chapter to escaping the flatland of paper and video display[47]. He discusses the concept of "small multiples," where the same design structure is repeated for different categories or points in time. The viewer can focus on the details of just one object, yet still have the context of adjacent elements. He shows several ways of encoding time series of multivariate data, including by hav-

ing each data point be a number or more complex glyph (a symbol showing multiple features). In a chapter on graphical excellence, he gives a compelling example of decomposing a time series into its different frequencies to emphasize seasonal trends, yearly trends, etc[48]. Tufte also discusses how graphical elements can be multifunctional. For example, instead of regular tick marks on an axis, there could be ticks and numbers for the minimum, maximum, and mean.

As opposed to a static image, a visualization can take advantage of interactivity to explore a multidimensional space, discover patterns, and navigate a dataset too large to see clearly at once. Card et al.[12] discussed three modes of interaction. Data transformations include the dynamic query (an interactive visual alternative to SQL) and details-on-demand. Visual mappings include how data is mapped to color, shape, and other visual attributes. Visual transformations include panning and zooming, having alternate views, non-Euclidean projections, lenses, and highlighting. Ahlberg and Schneiderman[2] applied dynamic query filters to support rapid browsing of starfield displays. Their work was the precursor of the commercial product Spotfire, currently used for analyzing microarray data, among other tasks.

2.2 Ontologies

The Foundational Model of Anatomy[40] set the standard as a bioinformatics ontology of canonical anatomy that allows symbolic representation and reasoning over a wide range of relationships. The principal component is the anatomical taxonomy, which attempts to use standard nomenclature while adhering to rigorous definitions and relationships. The authors took much care in defining such nonleaf terms as organs and tissues but noted that inconsistencies are nearly unavoidable when assigning definitions and logic, for example with embryonic anatomy. Another main component is the structural abstraction, used for describing spatial and part-of relationships. A less realized component is the transformational abstraction. The FMA includes nonmaterial geometric abstractions and supports the creation of anatomical

sets, including sets defined by functionality. In this work the FMA will be called a *foundational ontology*, and the ontologies based on the FMA for subdomains will be called application ontologies, though in more philosophical literature, the FMA is considered an application ontology.

The potential value of the FMA in medicine is endless, though many research and engineering challenges exist to get the FMA and allied technologies in widespread use. One such hurdle is that many labs are handcuffed to their own knowledge bases, and the alignment of their knowledge bases with the FMA can be a difficult or time-consuming task. The alignment software PROMPT[35], which runs inside Protégé, [1] assists the user in comparing two ontologies and resolving conflicts. Perrin[38] created a visualization plugin for PROMPT using tree maps and conducted a user study where a small population unanimously agreed that Prompt-Vis led to more enjoyable and effective experience. Another hurdle is to create an application ontology for a subdomain, which might not only be a subset of the foundational ontology, but could also incorporate additional data or modifications.

Lambrix et al.[31] conducted a study of ontology development tools using a subset of the Gene Ontology (GO)[2]. GO is very different from the FMA in several respects; it is continuously updated as new research happens, rather than being curated with occasional releases, and it has very few relations—just subclass and part—which are nowhere near as topologically complex as in the FMA. They noted that almost all the systems they studied had shortcomings with scalability. In general, they found all the tools they studied to perform well, though they all had learnability issues. Protégé had a clear advantage for being so extendible with both plugins and format import/exports.

The CLOVE framework[50] is a start toward creating an application view from a foundational ontology. The authors defined a language that specifies inclusions or

[1] http://www.protege.org, a free open-source ontology editor and knowledge-base framework

[2] http://www.geneontology.org

exclusions based on simple constraints. There is still much work to do with creating a richer language for views, not even including the possibilities of adding or modifying content or drawing knowledge from multiple ontologies. This research area is likely to be very active in the next few years.

Bernstein et al.[10] developed a natural language system for editing ontologies. Their system prompts the user with possible grammatically correct completions, which addresses both the habitability problem (users expect a limited set of features that the capabilities of the system far exceeds) and the ambiguity problem (by restricting the language to a controlled grammar). They conducted studies showing their system is effective, even in the hands of novices, though the jargon associated with ontologies was a difficulty. As their system supports a relatively simple set of edits, they note that they cannot compare their system fairly to Protégé—rather they would need a simplified version of Protégé to see if the natural language truly has an advantage.

2.3 Bioinformatics and Biosimulation

The fields of bioinformatics and biosimulation are becoming increasingly intertwined. The Physiome Project, focusing on modeling and simulating all aspects of physiology, recognized that ontologies make the modeling environment richer and unambiguous[16]. In the next sections, the relevant work from bioinformatics and biosimulation, as well as visual environments that incorporate both, is summarized.

2.3.1 Bioinformatics

The creators of CellML[32] recognized the need for a standard medium that was both human-readable and computer-readable and could aid in exchanging cellular models, a task that had previously been costly and fraught with error. They discussed three forms of model representation: high-level conceptualization, mathematical model, and computer code. The conceptualization reads as prose but is insufficient to rigorously

define the model. The computer code lacks the semantic information of the model. A cell modeling language should bridge between computational and published models. The Systems Biology Markup Language is such a language but is lacking in hierarchical structure and temporal and spatial scaling. CellML was designed to address these needs. Some key features include variables and connections, units, metadata, and groups. These features allow more powerful analysis, such as model validation.

Several researchers have developed software to generate simulation code from an abstract model, with the implementations ranging from prototype to commercial. Cook et al.[14] proposed using ontologies to generate code for models through an icon-based visualization. They noted that much physiology obeys canonical equations, such as resistance or capacitance laws. Rubin et al.[41] built upon the proposal, using both canonical equations and custom equations to recreate a cardiovascular model. They discovered several errors and inconsistencies in the original model as a result. Cook[15] later developed Chalkboard, a graphical modeling tool for pathways that also lets the user analyze feedback qualitatively. Additionally, it generates quantitative code that can be run by a simulator.

2.3.2 Biosimulation

Lumped parameter models have widespread use in systemic biology, as well as in other unrelated fields. Some common themes are emerging, and visualization can play a key role in progressing the field of biosimulation in these directions. Researchers want to tune models to patient data. In nearly all cases, the patient data that clinicians are capable of noninvasively gathering underdetermines the system, and empirical data or best guesses must be used for the remaining constraints. Researchers need to validate their models against known data. They want to propose hypotheses about physiology and test them against their models. They want to extend or link together models developed by themselves or others.

Olufsen et al.[37] conducted experiments on subjects to measure cardiovascular

response to cerebral hypotension from changing posture while attempting to keep other variables constant. They recorded heart rate, blood pressure, and cerebral blood flow velocity and fit the data to a simple model. They noticed several difficulties in inferring resistances from their data due to the complexities of human physiology (assumed MCA is constant volume, i.e. flow proportional to flow velocity, and didn't account for baroreceptor's effect on measured finger blood pressure). Nevertheless, they were able to model a biphasic change in cerebral resistance and postulate an explanation.

Lakin et al.[30] noted that many models of intracranial pressure operate on a classical assumption that because of the rigid skull, blood flow is volume-preserving within the skull. They cited several reasons why this assumption can be a poor approximation and built a full-body lumped network around a cerebral model that included sources, sinks, and conduits for CSF and ISF, as well as regulators for pressure. They also focused on hypotensive scenarios and were able to reproduce the characteristic physiological behavior. They could not duplicate exact numerical values due to problems arising from tuning parameters—they mostly used empirical data constrained to the flow equations.

As part of the Virtual Soldier Project, Neal et al.[33] used subject-specific models to predict survival in pigs wounded in the left ventricle. Prior to injury, they used data from each pig to tune specific models. Post-injury, they used limited noninvasive data (heart rate and arterial blood pressure) to drive the tuned model to infer total blood volume and cardiac output, both strong indicators of survival. Of note is that their model integrates several smaller models written by others, such as the heart or the baroreceptors.

Neal et al. discussed in detail their parameter optimization, which took several days to complete. Clearly any research that can help speed up this process would be quite useful. They were able to gather a mere fraction of hemodynamic variables from measurements on the pigs, whereas the rest had to be "textbook data." Some of

the parameters, such as compliance, could be calculated from others by laws. Other parameters needed some scaling when applied to a specific patient. They performed three rounds of numerical optimization with some manual intervention. The earlier rounds isolated modules so that they could be tuned without interference. The final round allowed a few parameters to vary to fit the blood pressure curves without changing anything else.

Kerckhoffs et al.[29] coupled a finite-element model of the heart to a lumped network modeling systemic and pulmonary circulation. To initialize the FEM, they used a lumped model of the heart that included interaction between the two ventricles. Their work is only the tip of the iceberg in multiscale modeling. (See the recent IEEE special issue for more on the state of the art of physiological modeling[13].) Cardiac output eventually becomes cardiac input, and this feedback cannot be ignored. However, the circulation can be modeled on a much simpler scale than that of the ventricles. Researchers have built detailed models that can take days to simulate a few heartbeats, such as the Smith's heart model that accounts for electrodynamics and deformations[44]. Though impressive, the model illustrates the need for simplifying whenever possible to make debugging, tuning, and simulating tractable. Visualization tools are needed to navigate these multiscale models and the deluge of data they produce and to assist with tasks such as calibrating coarser with finer models.

2.3.3 Integrated problem-solving environments

Johnson[23] posed the integrated problem-solving environment as an important problem, noting that visualizations are often considered afterthoughts to the model and simulation. The Physiome Project [22] has spawned several visualization packages, e.g. CMGUI and SCIRUN, mostly aimed at scientific visualization. Chalkboard[15] coupled with JSim[3] would be a powerful environment. Pathway Analytics, commer-

[3]http://www.physiome.org/jsim/

cial software produced by TeraNode[46], offers a full suite for the domain of biological pathways using biological databases, authoring models, collaborating, simulating and analyzing, incorporating lab data, and visualizing.

Antoniotti et al.[6] developed a natural language interface for querying simulated biological systems with propositional temporal logic. Their queries use time-dependent words such as sometimes, eventually, and always. For example, asking if a system has a steady state is equivalent to asking if eventually there will always be zero time derivatives. They can also generate sentences of "biologically interesting factoids." Though their system does not involve visualization, it is worth mentioning due to its potential for making complex mathematical systems accessible to biologists with little mathematical training.

2.4 Related visualization research

Much visualization research has relevance to the biological domain. Classic visualization problems such as visualizing stock market data or browsing photograph archives involve multidimensional time series infused with metadata. Simulation or lab data present the same challenges. Sometimes the data has an underlying topology best portrayed by a graph. Clinicians may want to monitor this data in real time to assess a patient, or researchers may spend months experimenting and tinkering with data from repeated experiments. Any visualization tool, no matter how good in theory, must be usable in practice.

2.4.1 Biological/medical data

Baker et al.[8] visualized the dynamics of genetic regulatory networks. Their tool shows both the movements of regulatory proteins and their respective concentrations. It animates the diagram into 3D to show the topology of the network structure without losing the mental map. Bajaj et al.[7] visualized multi-component macromolecules, up to a million atoms. They considered four levels of detail: the backbone

chain, secondary structures, e.g. helices and sheets, residues, e.g. amino acids and nucleotides, and atoms. Their system can visualize structure, functions on a surface, and volumetric data. Akers et al.[3] created a system for researchers to explore the brain's white matter pathways in 3D with dynamic queries. Users can filter on ranges of pathway length, curvature, and fractional anisotropy and can define volumes of interest to see pathways contained in their unions or intersections.

2.4.2 Multidimensional data, time series, and graphs

Wattenberg[52] displayed large graphs containing multidimensional discrete data by rolling the graph onto two categorical dimensions and aggregating nodes and edges. Ham et al.[51] visualized large state transition systems. They built a backbone tree by ranking nodes, clustered based on local structure, and preserved symmetries. Tzeng et al. peered inside neural nets[49], displaying input and output weights for both single data points and sets of data. Their visualizations allowed them to better understand the hidden states of an automatically computed net and reduce the number of nodes in the net without sacrificing accuracy. Saraiya et al. [42] studied alternate ways of visualizing graphs associated with time series. They found that representing a single attribute at a node was more accurate for single time-point analysis and comparing between two time points. Representing multiple points of time on a node was faster for determining when in time a certain behavior occurs or for identifying outliers. They found that a single view was faster for investigating a single node or point in time, whereas multiple views were faster for investigating groups of nodes or intervals of time.

Battista et al.[9] gave an overview of graph-drawing literature, mostly focused on smaller graphs with goals such as minimizing edge crossings, using only right angles, etc. Huang et al.[21] used several techniques to display larger graphs. They filtered away nodes that were below an eigenvector importance threshold and did not disconnect the graph. They also clustered highly connected cliques into subgraphs.

They used a force-based algorithm to reduce overlap between nodes and labels.

Robertson et al.[39] investigated visualizing a polyarchy, that is, showing two orthogonal hierarchies as well as the cross-relations between them. They developed a means of pivoting from one hierarchy to the other while maintaining context, through sliding and rotating. Several user studies guided them to refine their visualization as well as demonstrate its effectiveness. They also examined how the polyarchy visualization would be incorporated into an infrastructure with a web-service back end and a mid-tier cache.

Holten's well-received work this past year on hierarchical edge bundling [20] reduces clutter and enhances clarity in graphs consisting of both hierarchical and adjacency relationships. His examples on call graphs are especially relevant to the model browser. His algorithm renders edges as alpha-blended piecewise cubic B-splines and indicates direction by a color gradient, which has less clutter than arrows. Shorter edges are rendered after longer edges, and shorter edges are more opaque than the more transparent longer edges. A bundling strength determines how edges are grouped; the user can continuously vary it to obtain low-level or high-level connectivity information. The user can also draw a line through bundles of edges to select them and explore them with the other edges absent. His future work involves allowing local bundling strengths, such as by a lens widget, so that the user can investigate individual curves in a bundle without changing the global bundling strength.

2.4.3 Ontology visualization

Interest in ontologies and the semantic web has grown recently, and with that, the need for visualization. Many graph-based visualization techniques have appeared, using tree-maps, force-directed layouts, 3-D navigation, etc. An outstanding survey paper on these techniques[26] attempted to classify them on their functionality and usability for various task domains. They note that large ontologies (on the order of 100000 classes/instances) are especially challenging to visualize for several reasons.

The size of the ontologies requires specialized data structures and graphics. For most visualizations attempting to display that many items on a screen simultaneously, the labels are relatively unimportant, However, for ontologies, the labels are very much important, and it is hard, if at all possible, to have labels be nonoverlapping and legible when the view is cluttered.

A collaboration by many of the same authors had previously conducted a usability study on four techniques[27], including Jambalaya[45] and TGVizTab[4], both of which are available as plugins to Protégé. They defined several general tasks for users to do with the tools to assess the performance of the tools. The ontology they used was small (a few hundred classes, instances, and slots), which is of note, because in my attempts to use the visualization tools on the FMA (hundreds of thousands of classes and instances), they were either too slow (by a factor of 1000) or crashed because they required all the data to be in memory. Another point of discussion is that a particular domain-specific task or a particular knowedge base may be more amenable to a particular tool. This dilemma is a frustrating chicken-and-egg problem where the computer scientist does not know how to best design the tool without knowing how biologists will use it, and the biologists will not use a tool unless it is usable. A further frustration is users' habituation to existing interfaces, exhibited by the fact that the textual class browser, default in Protégé, performed the best in their experiments, which they theorized is due to users' prior familiarity with filesystem navigation.

Jambalaya[45] is a well featured plugin for Protégé that visualizes ontologies. It has a zoomable node-link interface with a variety of layouts and supports drag-and-drop from the default class browser. One of its features is that it can nest nodes inside of nodes according to the subclass hierarchy. The relations displayed as colored curved edges can be filtered to what the user wants to see. As the user navigates, transitions are smooth, including a full zoom into a node producing a class browser view.

DIaMOND[17] is a degree-of-importance model for Jambalaya. It is described as attention-reactive, because the visibility of the nodes is dependent on what the

user focuses his or her attention. Items can be labeled as landmark, interesting, or noninteresting. Landmark items are those selected as truly important and never to be hidden. Noninteresting items are either items specifically designated as noninteresting or items whose importance threshold has decayed below a threshold. Interesting items are items that have been accessed or otherwise indicated as interesting, and have not had their importances decay yet below a threshold.

TGVizTab[4] uses a mass-spring system to solve for the layout using forces. It has many features in common with Jambalaya, but of importance is that children of a hierarchy may not all appear at the same level of depth. In the study of Katifori et al. [27] many users found the layout to be choppy and chaotic, though despite the frustrations, the users performed very well with the tool.

2.5 Usability

North's thesis is that the purpose of visualization is insight, and hence an evaluation of a visualization should determine how well it generates insight [34]. He lists some important characteristics of insight, namely that it is complex, deep, qualitative, and relevant. He argues that the standard experiments based on measuring speed, accuracy, or efficiency of task performance do poor jobs at evaluating insight. Instead, benchmarks need to involve more complex tasks, or ideally, instead of benchmarks, users would participate in an open-ended exploration. During the explorative process, users would think aloud, as is done in formative usability studies. Insights found by the subjects would be documented and given precise numerical ratings by experts ranking how complex, deep, relevant, and correct the insight. Though this vision is grand, implementing such an evaluation requires more time to develop, expert judges, motivated users for extended experiments, and a larger sample population.

Saraiya et al.[43] developed a methodology for evaluating bioinformatic visualizations based on the principles just described. They studied several microarray data visualization tools, both free and commercial, with their methodology. Their sub-

jects found many general insights, some deep insights, but few insights that led to new hypotheses. They theorized that the lack of new hypotheses was due in part to the subjects having limited time and familiarity with the software, but more so because the software did not connect the data with the relevant domain of biology. Surprisingly, domain novices and domain experts performed equally in their studies. This fact underscores the importance of embedding bioinformatics in visualizations, so that users can make higher- level biological inferences.

Albert et al.[5] investigated how to map numerical changes in data to visual changes so that clinically important changes in a patient's condition would be visually apparent. Their target was a cardiovascular information display that could be used for anesthesia.[1] They conducted experiments to determine the just noticeable differences (JND) for changing size and interviewed anesthesiologists to determine what changes in numerical patient data were clinically significant. They coupled these two studies to produce mappings from data to display that were sigmoid in nature. The slope of the obtained line in the middle is 2.5 times the JND. In the extrema, the slopes flatten, as accuracy isn't as important as the notion that the data is under/ oversaturated.

In addition, aesthetics and ease of use are important considerations for visualizations. Saraiya et al.[43] noted that visualizations that offered multiple ways of viewing the same data instilled more confidence in the users and that awkward interfaces detracted from insight. Tufte[47] discusses an example of a topographic map that shows both land and ocean bottom. The blue and brown-toned coloring invokes an obvious analogy to water and land. A downside is that subtle color shades may be hard to discern absolutely, especially in different local contexts. On the other hand, the standard rainbow scientific color spectrum allows for a finer discretization of a numerical range. The rainbow suffers from the order of the spectrum not being intuitive, and the colors being garish.

2.6 Relevance to the Browsers

The related research presented thus far covers a broad array of work ranging from problem domains in bioengineering to tools in bioinformatics to specific techniques in visualization and usability. Each of the browsers presented in this dissertation more directly depends on specific related work.

The original work by Agutter[1].was an inspiration for the animated data browser. They recognized that telemetry needs to be processed by a person into more abstract observations, so one could quickly realize if someone was very sick. Saraiya[42] implemented a time-series graph visualization similar to the animated data browser as part of a user study of several visualization alternatives. The distinction is they only had one time-series variable per node, whereas the animated data browser supports two or three per node.

The model browser is unique in the context of bioinformatics. Certainly visualization of call graphs in the context of software engineering has been around for a while, and Holten[20] had call graphs in mind as a use case for hierarchical edge bundling. Though there are other visualization techniques suitable for networks, the model browser also uses supplemental metadata per node (variable). Cook's Chalkboard[15] bridges the cognitive gap for models in the direction of automatically generating a computational model from an abstract model. The model browser tackles the inverse problem; that is, it helps one reason about more abstract concepts given only an annotated computational model.

Many other researchers have built ontology visualization tools; most are summarized in Katifori's survey[26]. Of these tools, Jambalaya[45] and TGVizTab[4], both plugins to Protégé, performed the most successfully[27]. Protégé, an impressive and successful authoring tool, can also function as a browser, though as browsing is not its primary function, it is not optimized for browsing. The ontology browser's choice of relevant nodes to display is a simplification of the degree-of-importance present

in DIaMOND[17]; the browser could be easily extended to support their full degree-of-importance model. The ontology browser uses an extension of Yee et al.'s radial layout algorithm[53] designed to handle cycles.

Chapter 3

ANIMATED DATA BROWSER

Two browsers for biosimulation have been developed, one that focuses on the model, and the other that focuses on the data produced by a simulation. The research thus far has targeted a lumped-parameter model of the cardiovascular system developed for the Virtual Soldier Project. Because of the model's complexity, scope of physical properties, and range in size of anatomical components, users can find the model and simulations overwhelming. These browsers make the model much more approachable.

Both of these browsers, as well as the ontology browser, use the Java-based Prefuse visualization toolkit [19]. This toolkit is ideal due to its integration of visualization into databases and graphs and its support for panning, zooming, and animated transitions. Currently the browsers use precomputed simulation data for convenience in developing and testing, but all the machinery is in place to obtain data as it is computed (which may be slower than real-time) from JSim.

The goal of the animated data browser is to bridge the cognitive gap from numerical data to observations. One way to visualize multidimensional time- series data is with curves over time. An example of such is figure 3.1. Note that even with just three curves superimposed, the view is starting to get cluttered. If those curves were to be replicated for over a dozen components of the cardiovascular system, much screen real-estate would be needed. The interface for the curves would need to require the ability to zoom and pan in time, an ability that is inherently present with animation. The animated data browser has several advantages over this approach. First, by representing time as a temporal variable instead of a spatial variable, more data

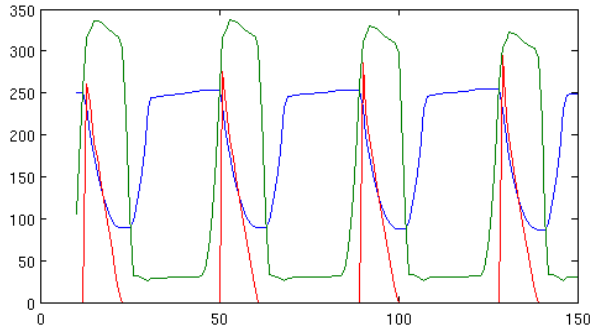

Figure 3.1: Visualization of time-series for pressure, volume, and flow using commerical mathematical software.

can fit onto the screen. Second, different visual cues can be used to represent different dimensions, as opposed to have all dimensions be curves. For example, volume maps intuitively to size of a shape, and pressure, an intrinsic property, maps well to intensity. Additionally, when there is a strong notion of what the variables mean, for example, flow, pressure, and volume in a blood vessel, a carefully crafted animated glyph (complex shape with different features mapped to different dimensions) could be more intuitive than simply three curves; for example it could more approximate a doctor's observation in a clinical setting. Note that the animated data browser could be used in parallel with graphs of time-series, enabling the user to choose the best option for the task.

Using the animated data browser on the cardiovascular system, several questions can be easily answered without having to read any graphs:

- Track a pulse through the circulation.
- What's the approximate period of a pulse?
- What parts have the greatest pressure differentials?
- What parts have the most blood volume?

- Find a vein with near constant pressure. Find one that has a considerably variable pressure.

All these questions can be answered from a single overview.

3.1 Animation

The data browser animates multidimensional data resulting from a simulation. The user can view many nodes at once and filter out irrelevant modules. The tool supports the standard animation controls for playback, including pausing, adjusting playback speed, and seeking. Instead of nodes corresponding to variables, as in the model browser, nodes correspond to several variables from merged anatomy. Each node has the same visualization scheme, which the user can specify. The user can map each physical property (generic unit type, such as volume, pressure, or flow) to a display parameter, such as the radius or color of the node. In addition, if he or she wants time as an axis or wants a more custom view of a specific node, he or she can ask for details. Figure 3.2 shows four screenshots from the animated data browser. The next sections address the scaling that happens between the data and the display and the types of detailed visualizations that have been implemented.

3.1.1 Scaling

The visualization receives numeric data from the simulator and needs to transform it to be displayed, for example to a 0–255 intensity or a range of circle diameters. Given the units for the data, some scaling information is available, for example whether the data must be nonnegative. Further scaling rules could be defined from a configuration file, for example, that all pressures should be scaled by a certain formula. The choice of how to scale can negatively affect the interpretation of the data, perhaps exaggerating the irrelevant or deemphasizing an important trend. For example, fitting to the minimum and maximum would result in flat venous pressure turning into irregular

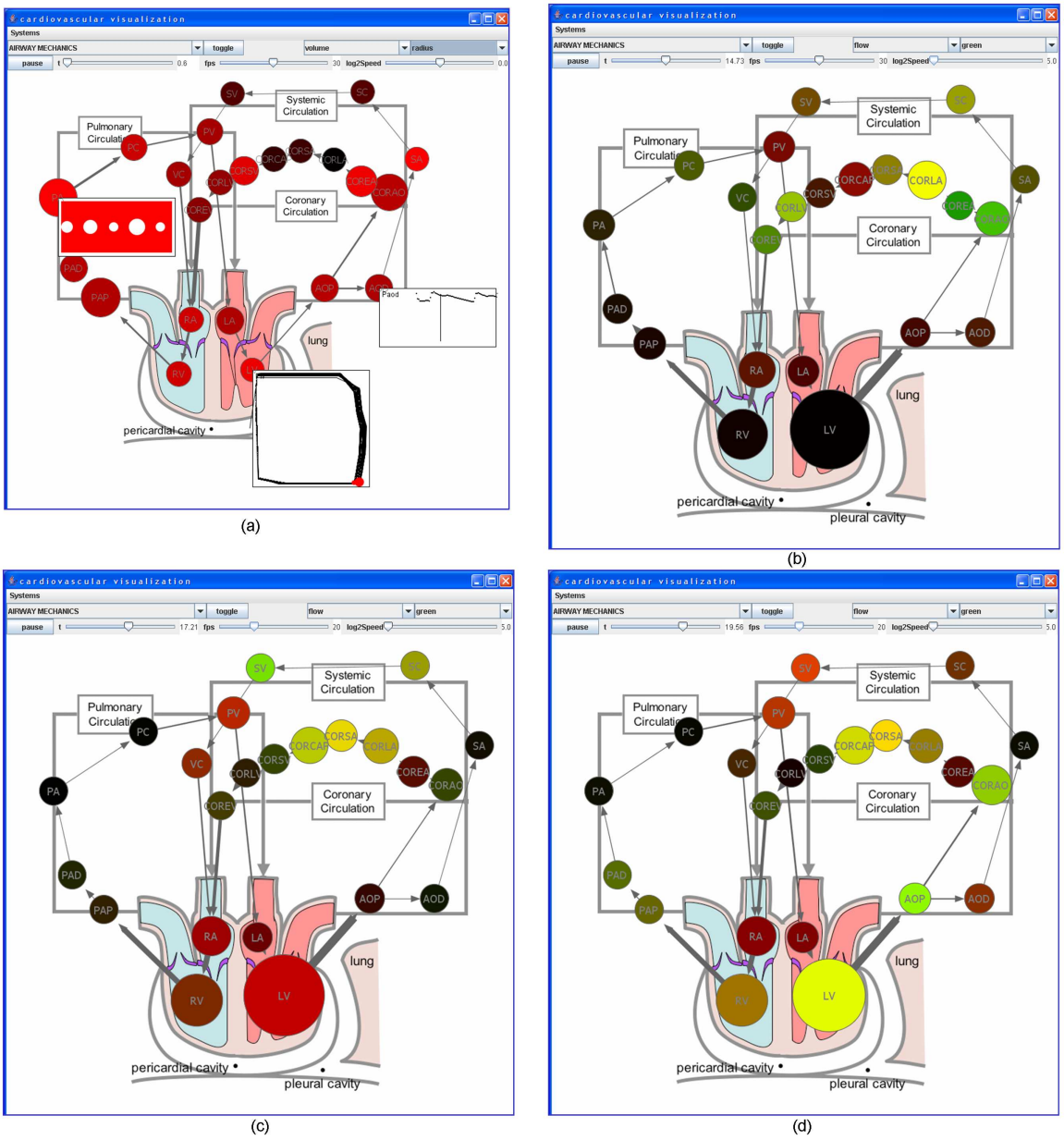

Figure 3.2: (a) The animated data browser with three different styles of details shown. (bcd) Three frames animating left ventricular contraction. The LV fills with blood (inc. volume—size), the muscle contracts (inc. pressure—red), and blood flows into the aorta (inc. flow—green).

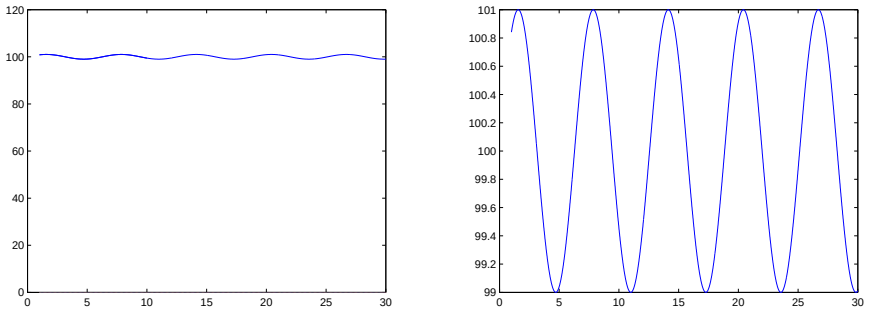

Figure 3.3: These two graphs of the same curve can cause markedly different interpretations.

amplified noise. A better alternative would be fitting to 0 and the maximum. Though this mapping does not amplify noise, it could flatten data. Whereas a businessperson would choose the scaling that is the best sell, scaling should not impinge on scientific discovery.

We give the user the choice of whether to scale data absolutely (by the same value for all nodes) or relatively (different values per node). The former has the advantage of comparing node to node. The latter has the advantage of finding the ideal scale for each node to reveal how that node changes over time. For example, only a small fraction of the blood that is pumped through the heart travels through the coronary circulation system. The former would convey this ratio, but the latter would scale the volumes of blood in both so that the fluctuations are clearly shown in both. Both choices are fully justified, so the user should be able to select which one to use. For the absolute scaling, the user has sliders to control how to scale the values onto the display. Values that are too far outside the display range are hidden. Figure 3.4 shows how different anatomical volumes can be visualized at different scales.

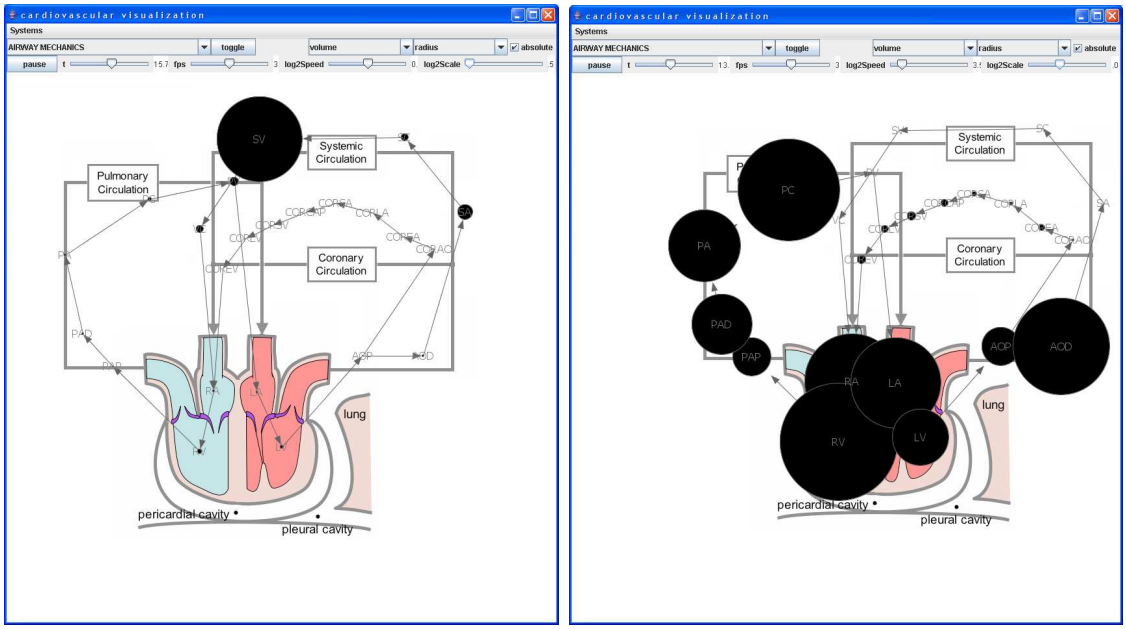

Figure 3.4: (left) Volumes scaled to the size of the systemic aggregates. Pulmonary and heart volumes are tiny, and coronary volumes are pinpoints. (right) Volumes scaled to the size of the heart chamber volumes. Coronary volumes are tiny but visible. Larger volumes are hidden, as they would otherwise occupy the entire screen.

3.1.2 Details on demand

In addition to looking at animations of many nodes simultaneously, the user can look at animations one node at a time. Each node shares the same schemes for visualizing the details so that the user can compare nodes easily. The schemes implemented so far include time plots, phase plots, and vessel diagrams. Though time plots and phase plots can be static images, animation can be helpful. The animations of a time plot let users pan through windows of long time-series. For example, users can see sliding windows of five seconds for five minutes of data. The animated phase plots establish a correspondence of t to (p, v) that would not be present on a static picture.

The vessel visualization displays volume, pressure, and flow simultaneously through a cross-section of an idealized cylindrical vessel. Volume is proportional to the square of the vessel's radius. Color encodes pressure, with the assumption that pressure is spatially uniform in the vessel. A pattern of blobs flows through the vessel. Because the blobs are really representing distance as traveled by a single cell, whereas flow is measured in volume over time, the distance d is calculated from flow f and volume v by

$$d(t) = \int_0^t \frac{f(t)}{v(t)} dt.$$

This representation has an analogy to the flow-velocity measurements that clinicians take with Doppler ultrasound.

As part of a larger simulation package, selecting one or more nodes could display their curves in a linked full pane. Another alternative is to embed the details inside the nodes as more complex glyphs or to have multiple parallel graphs, each focusing on one dimension.

3.2 Conclusions and Future Work

The animated data browser can answer the following questions in an overview animation.

- Track a pulse through the circulatory system.
- What is the approximate period of a pulse?
- What parts have the greatest pressure differentials?
- What parts have the most blood volume?
- Find a vein with near constant pressure. Find one that has a considerably variable pressure.

The animation itself brings to life the periods and phases of the predominant signals. The pressure impulse from the left ventricle can be seen propagating from large arteries to small arteries to capillaries. Though the static frames in figure 3.2 do not do justice to an animation, the change in pressure is quite apparent for the ventricles of the heart. Using an absolute scaling instead of a relative scaling makes it easy to see that most of the blood is in the systemic veins, and a small percentage of blood flows through the coronary vasculature (figure 3.4). Because the authored layout has a correspondence between parts and variables and may have a geographical arrangement of parts, it can be easy to locate where a certain part should be. In this case, the systemic veins in the systemic circulation chain have nearly constant pressure, whereas the pulmonary veins in the pulmonary circulation chain have a variable pressure.

The animated data browser requires some authoring by the user to build a layout that is understandable. People who are not medically trained find the animated data browser approachable thanks to the diagrams in the background that explain the anatomy. A whole separate problem is to design a good authoring tool. However, the authoring only needs to be done once per model, and with ontologies, could be somewhat automated. Similarly the detailed visualizations only need to be written once and can then be applied to any data of corresponding physiology.

Another question is how many dimensions can be represented at once before the user is overwhelmed. Size and one color channel are reasonable, but what about when two color channels are blended together? Other possibilities include changing the shape of a node and the texture inside it. Sound can offer temporal clues that might be hard to visualize otherwise, such as the progression of a pulse through a pipeline or the superposition of two frequencies. An interesting next step would be to have a detailed visualization that loads medical imagery that can animated by the simulation data.

I want to develop a visualization that lets a user compare simulation results, both in overview and detail. The user will need to tell the system what he or she is trying to investigate, and then the system will tune its visualizations to address qualitative questions around the user's needs. The user should retain full exploratory power with assistance from the system, rather than the system forcing the user down a certain path.

A researcher may be interested in one of many patterns in a single curve. Any automated process should suggest to the researcher which patterns he or she may want to investigate and should not make assumptions without asking. For example, blood pressure's main variance is due to the cardiac cycle, with a period a little short of a second. The pressures in the lungs influence the pressures around the heart, so there is an additional component at a period of about 3-5 seconds. There exist more subtle effects at longer periods due to feedback with the baroreceptors. Algorithms need to know the desired period when performing such tasks as tracking the minimum and maximum values per period over time or measuring how a certain frequency changes rate over time.

Once the researcher has indicated the desired frequencies to investigate, the visualization will help answer questions about trends and comparisons. For blood pressure, some common questions include:

- Is the frequency (heart rate) increasing?
- Is the mean pressure decreasing?
- Is the pressure narrowing? (decreasing variance)
- Is the pressure stabilizing?
- What is the shape of the curve?
- How does the pressure here compare to elsewhere in the body?

To address these questions, algorithms need to be able to analyze functions both spatially and temporally.

I also want to develop some effective means for visualizing the comparison between two multidimensional time series from different simulations. Some ideas include rendering them side by side or on top of each other using transparency. I would like to experiment with having maximum or minimum values fade away slowly so that if the data from two simulations are not aligned in time, they can still be compared. I also want to experiment with using dimensionality reduction and other mathematical tools to show epitomes of changes. A user could then cluster variables by "increases," "stays the same," or "decreases." The user should likewise be able to make dynamic queries based on qualitative analysis of the numerical data.

Chapter 4
MODEL BROWSER

Biological simulation is a fast-growing field today with a wide range of applications. Researchers are generating and sharing bigger and more complex models. Many challenges are arising, including debugging, tuning, and validating a model, as well as sharing, publishing, and merging models. There need to be better ways to reason about models beyond perusing source code, which is often cryptic or poorly documented. In this chapter, the model browser, which is a visualization application that assists the researcher in reasoning about a computational model through interaction with a graph of the model, is described.

By interacting with our visualization, one can answer questions about a model such as,

- "Show me the variables for just the vascular flow portions of the model." (figure 4.4)
- "What role does temperature play in the model?" (figure 4.6)
- "Does the model have baroreceptors in the aorta or the carotids or both?" (figure 4.10 and figure 4.12)
- "Do the lungs exert pressure on the heart?" (figure 4.13)
- "What parameters modify the P-V curves?" (figure 4.3)
- "In the model, what are the paths of blood through the cardiovascular system?" (figure 4.11)

These questions would be difficult to answer just by staring at unstructured equations. Over the course of this chapter, figures will illustrate the answers to these questions.

The test example for which the previous questions apply,is a cardiovascular model with baro- and chemoreceptor feedback; it represents anatomy as a lumped network. The lumps (nodes in the network) represent homogenized individual anatomical parts or aggregates at various scales, and the connectivity (edges) represent physiology such as flow or control. Variables and equations range from tangible to abstract. The lumped network represents state and behavior from fluid dynamics to chemical reactions, drawing analogies from L-R-C electrical circuits. A lump encodes several variables of different physical properties, such as pressure, volume, and flow. Other variables exist too, including temporary variables that hold common subexpressions or observational variables such as vital signs.

Realized into code, the model has around 300 parameters, 300 time-dependent variables, and 300 equations, of which 60 are differential. The variables use 25 canonical physical units (liters and cubic centimeters both belong to the same canonical unit of volume). The developer divided the code into eleven sections. The hierarchy that was created for this model contains 75 terms and has a depth of six, most nodes in the hierarchy have six to twelve children. Most of the hierarchical relationships are part-of, but some are is-a or address functionality. The model already had description tags; we worked with the developer to add module and anatomy tags. Figure 4.1 is a screenshot of our system showing the whole model. The different sections are pie wedges. The circles are parameters (outer) and variables (inner), and the arrows represent dependencies. An excerpt of code from the model is in Figure 4.2.

4.1 Fundamentals

The browser consists of an index of displayed variables on the left, a legend of colors on the right, and in the center the graphical representation of the model. Nodes represent variables from the code, and edges represent dependencies from the equations.

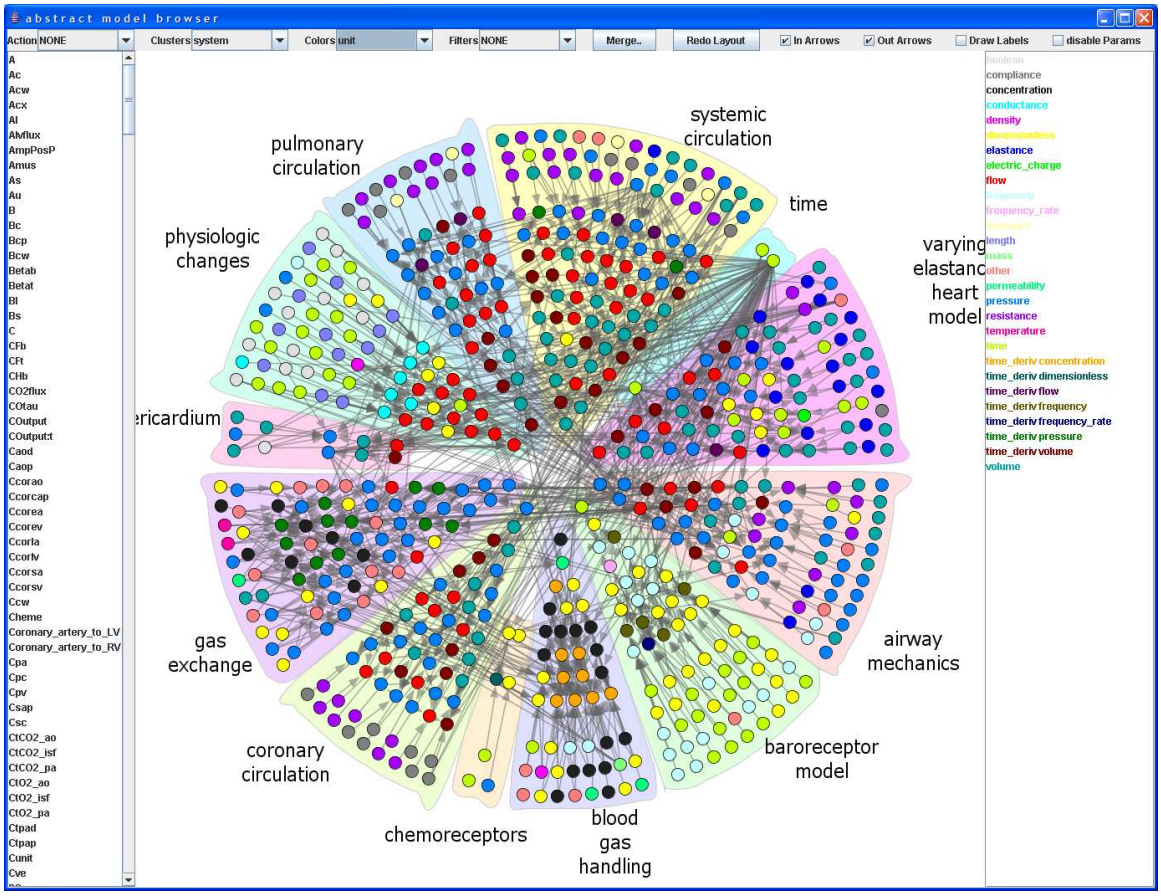

Figure 4.1: Overview of a large model using a pie layout. Parameters and variables are color-coded by scientific unit and clustered by module. Arrows show dependencies.

```
real Ro = 0.025      mmHg*sec/ml;
Ro.system  = "systemic circulation";
Ro.anatomy = "vena cava";
Ro.desc    = "Vena cava resistance offset parameter";

real Rvc(t)   mmHg*sec/ml;
Rvc.module  = "systemic circulation";
Rvc.anatomy = "vena cava";
Rvc.desc    = "Resistance of Vena cava";

Rvc = (KR*(Vmax_vc/Vvc)^2) + Ro;
// Vena cava: Lu et al. Eq.(4)
```

Figure 4.2: Excerpt of code from the test model showing declarations of a parameter and a variable, each with metadata, and an equation.

The interface provides easy navigation through a complex graph, such as traversing neighbors or pruning irrelevant parts.

The browser can run as a stand-alone program or as a plugin to JSim[1], a Java-based simulation system for building quantitative numerical models and analyzing them with respect to experimental reference data. JSim can constrain units in equations to be balanced, and it supports the embedding of metadata in the model file, as part of JSim's mathematical modeling language. JSim's capabilities are focused toward biological simulation, and it can import SBML and CellML models. The model browser uses Prefuse[19], a Java-based toolkit for building interactive information visualization applications.

4.1.1 Metadata

The model browser can filter, color, cluster, or merge variables based on metadata associated with the model. The model browser reaches its full potential when the

[1]http://www.physiome.org/jsim/

variables of the model possess a variety of metadata tags that can carry a variety of extra information. Some of these tags are part of JSim's math modeling language, such as units, datatypes, and comments, but others need to be defined explicitly, such as anatomical, physiological, or organizational tags. Tagging the variables does put extra work onto the developer on par with what is required to document code with a tool such as JavaDoc, but the benefits are huge and would likely save time in the long run. The tags give the user more fields to search and sort so that filtering and coloring can reveal interesting patterns. The test model has metadata for unit, datatype, module, comment, and anatomy, of which the latter three are tagged explicitly (figure 4.2). The module metadata corresponded to the eleven sections of the model's code.

4.1.2 Hierarchies

Hierarchies add significant meaning and capability to code. Though a computer scientist might be capable of designing a carefully architected object-oriented system, the same cannot be expected of a biologist writing a model. Furthermore, a biological model may have more interconnections than the average software. Thus a modular or hierarchical structure can enrich a flat list of equations and variables by using metadata to assign variables to entries in a hierarchy, perhaps derived from an ontology. An extra XML file holds the application hierarchy for the model. The ontology viewer (described in a later chapter) could be extended to develop hierarchies building from existing ontologies such as the FMA.[40]

4.1.3 Dependencies

Each equation in the model consists of a lefthand variable and a righthand expression, which in turn consists of several terms, either implicitly or explicitly. The JSim compiler parses the simulation code and exposes these dependencies, needed for compilation and execution, to the plugin developer. The variables are represented visually

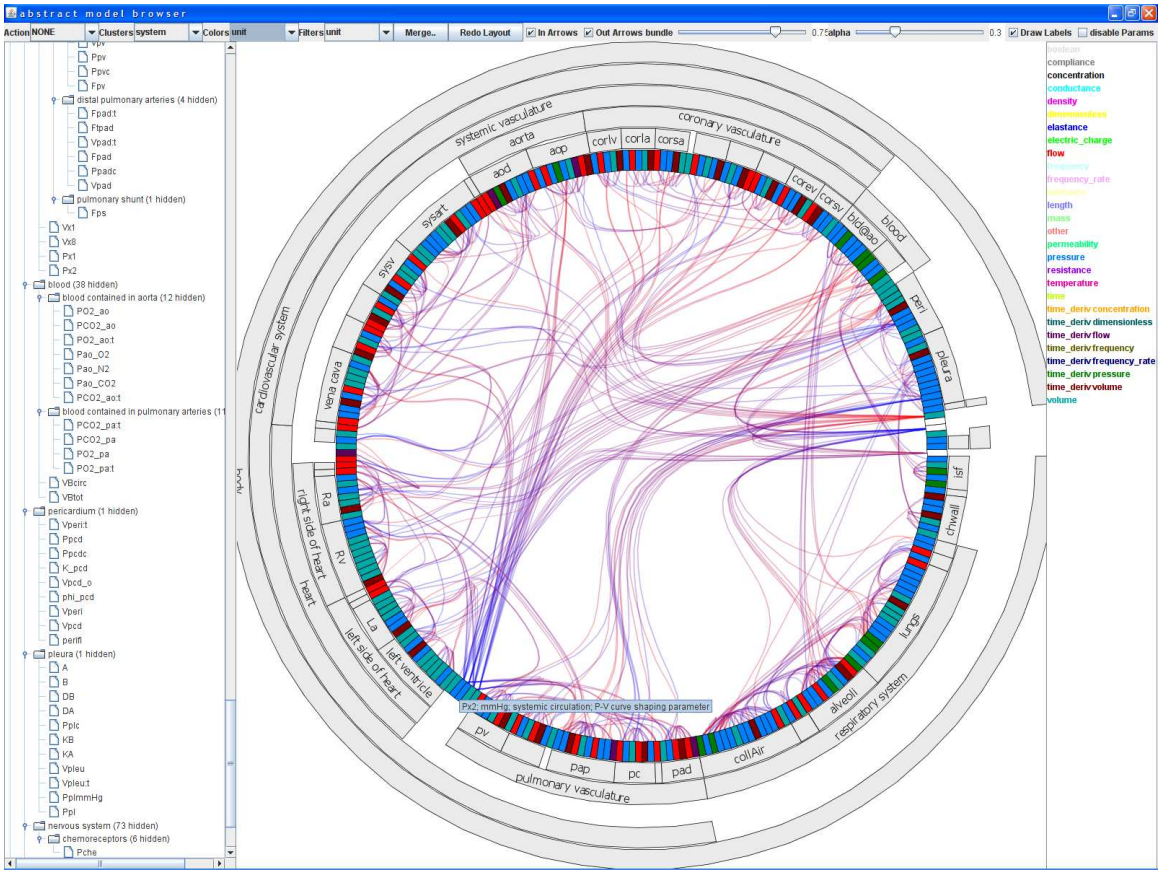

Figure 4.3: View of flow variables. Note the clump of P-V curve parameters at 7-o'clock.

as nodes and the dependencies as edges. For example, the equation $F = m \cdot a$ would have three nodes and two edges, with arrows from m and a to F.

4.2 Interaction

The model browser has three types of interaction, as categorized by Card, et al.[12]. Its use is illustrated with several screenshots.

4.2.1 Data Transformations

Data transformations operate on the data itself, but rather than doing an SQL query, the user can point and click through friendlier interfaces. For each field of discrete metadata, the user can select checkboxes that determine which values are shown and which are hidden. The system evaluates the conjunction over all the fields. If a filtered node becomes invisible, any edges going to or from it likewise become invisible. Figure 4.4 shows a view that has been filtered to show the variables corresponding to the vascular flow portions of the model.

The user can select a set of children in a hierarchy to hide or unhide. To remind the user that nodes are hidden in the hierarchy, the number of hidden nodes is displayed in the tree index, as is done in Jambalaya[45]. Figure 4.3 shows the variables filtered to just pressure, flow, volume, and their time derivatives, using hierarchical edge bundles.

The system can display details on nodes by selection either in the index or on the graph. When focusing on a node, the user can load equations, graphs, and other details for the node.

4.2.2 Visual Mappings

The system supports coloring of nodes based on discrete values in the variables' metadata. Each value is mapped to a color, explained by a legend. Around thirty

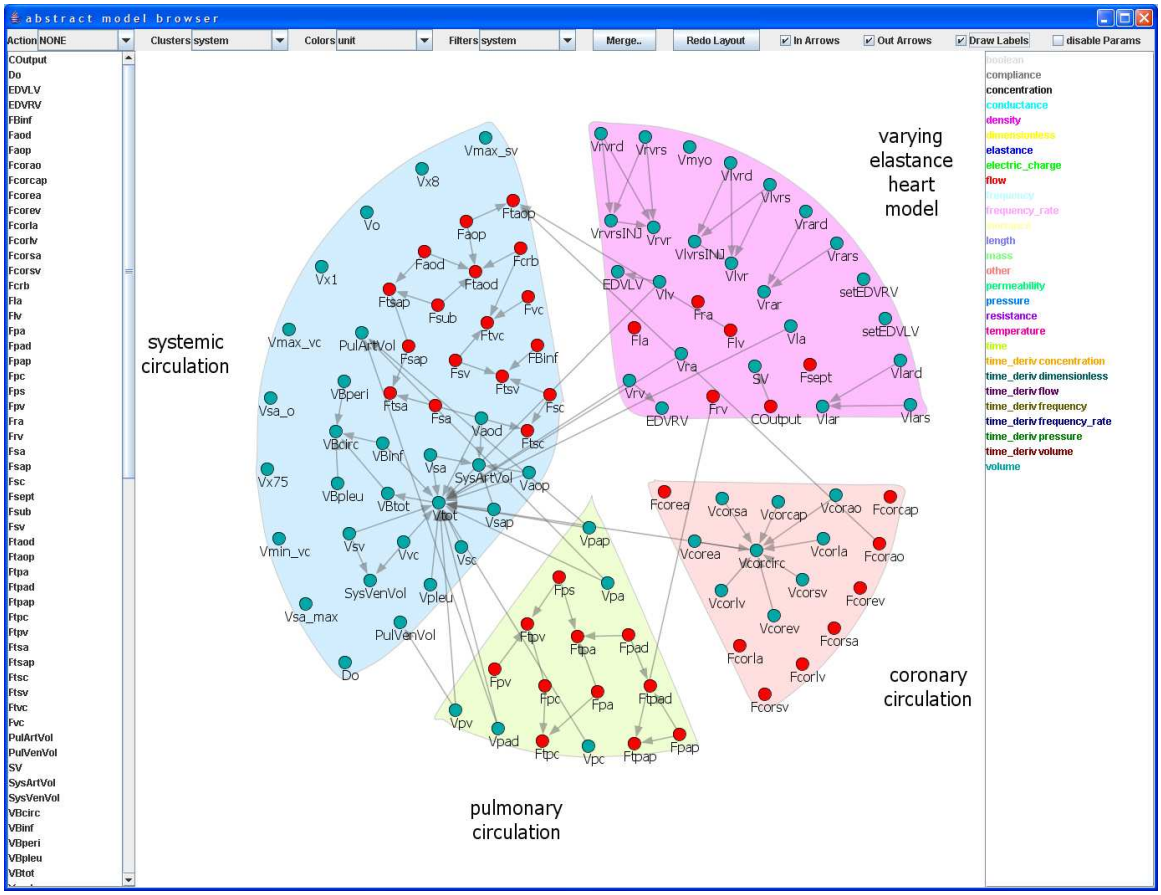

Figure 4.4: The user filtered away modules that do not correspond to vascular components and only selected units corresponding to volume and flow. The resulting set of variables correspond to the vascular flow portions of the model.

different colors can be displayed before shades become hard to discern. Units of measurement are colored by their canonical form, ignoring dimensionless constants (e.g. cubic meters and liters are both volume and are colored the same). The system also colors nodes based on connectivity, using different color channels to represent whether a node has zero, one, or more dependencies or is a dependency of zero, one, or more nodes. The system also allows filtering on the in- and out-degrees of nodes, as if they were metadata.

4.2.3 Visual Transformations

Several visual transformations map the graph, with the hierarchy expanded to some degree, to an interactive diagram.

Expansion / Contraction

The hierarchy can be expanded or contracted by navigating through the tree on the index or by clicking on a node and telling it to expand or contract if legal. When a set of nodes are contracted, any dependencies inside the set disappear. Dependencies to or from an outside node are redirected to the contracted node. Figure 4.5 illustrates the graph transformation from a merge of the yellow nodes. Expansion is the inverse operation of contraction.

Figure 4.7 is fully expanded, and Figure 4.10 is mostly contracted. When no hierarchy is present, contraction can be performed on metadata to merge nodes sharing common attributes, such as the same module or the same unit of measurement. For example, figure 4.6 shows an example where the layout is clustered by canonical physical unit, and the nodes have measurements in temperature are merged together.

With a few collapses and filters, despite no hierarchy present, the user can whittle the model to a small set of nodes of interest. Likewise, focusing on nodes and looking at connectivity and feedback restricts the view down to a few nodes. A compelling example of the information gained from a merge is when all clusters are merged so that

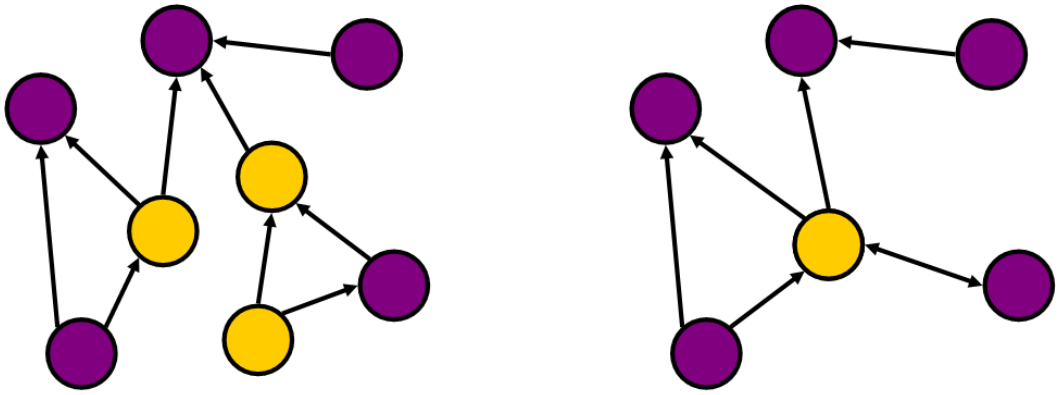

Figure 4.5: The left diagram is the original graph. When the yellow nodes are merged, the graph's connectivity is arranged as shown in the right diagram.

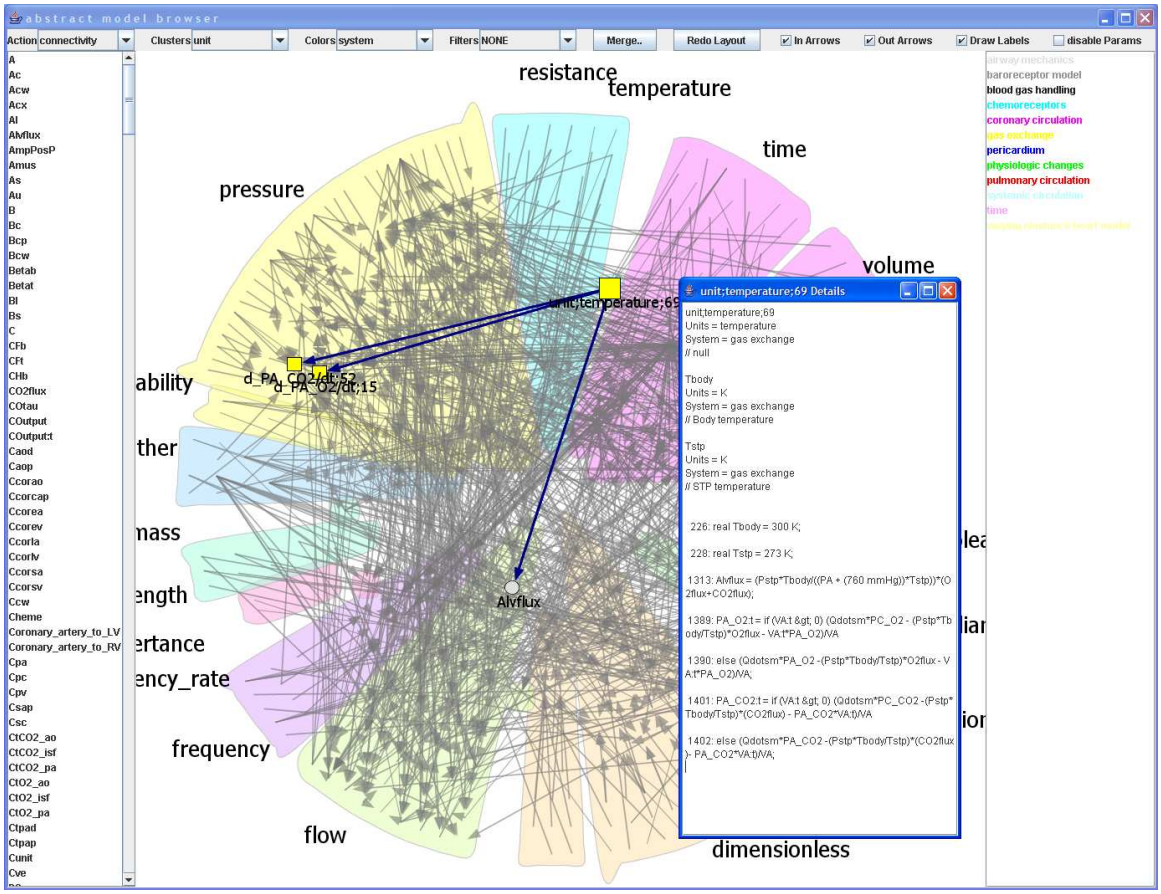

Figure 4.6: The layout is clustered by canonical physical units, and the all the nodes having temperature are merged. The user can quickly see what is influenced by temperature (e.g. no hypothalamus regulation) and can easily access all equations involving temperature.

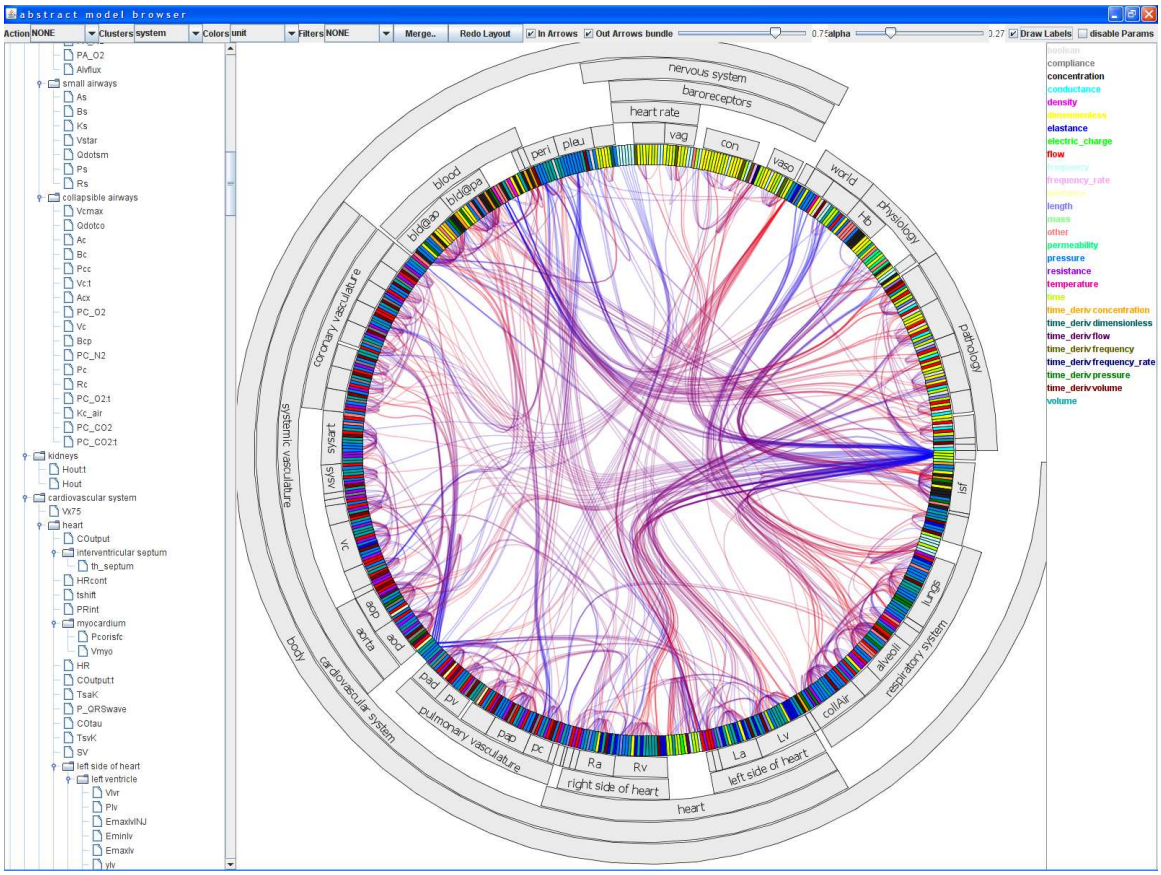

Figure 4.7: Full model, hierarchical edge bundling.

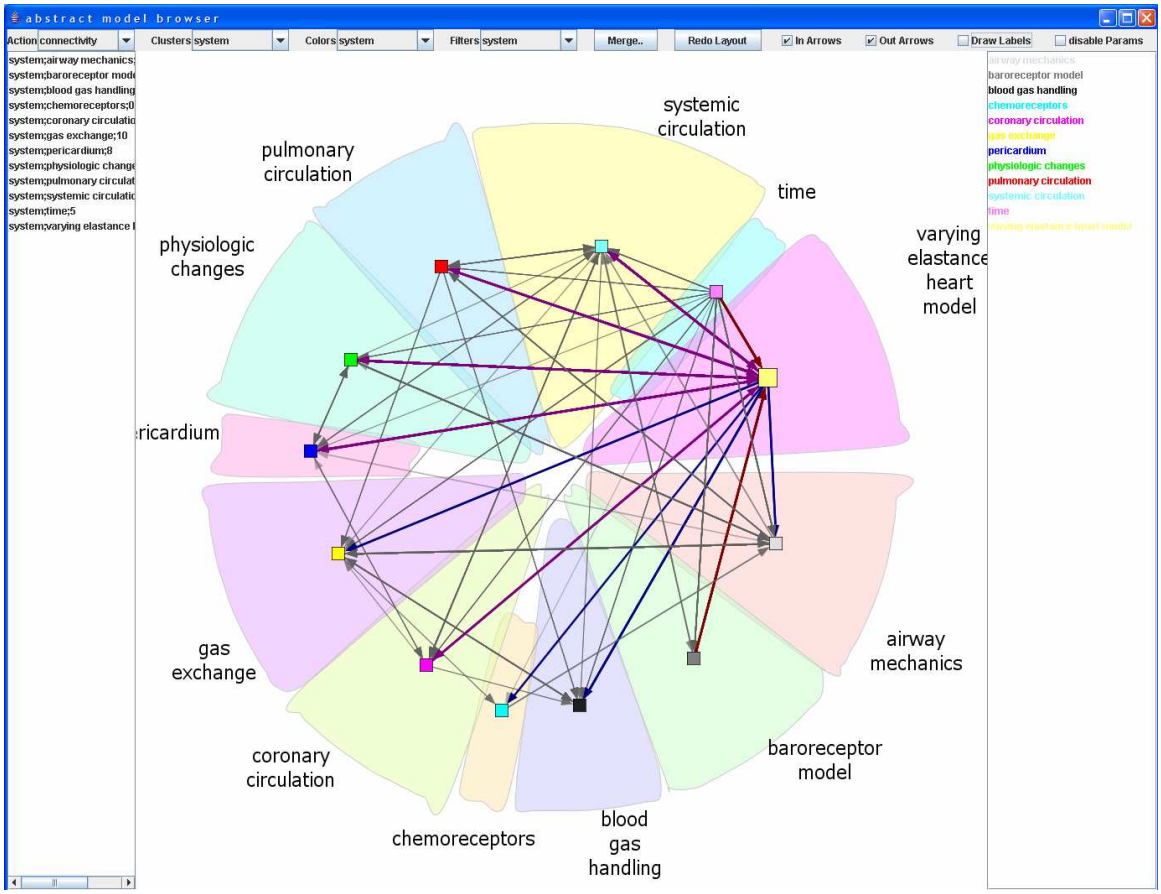

Figure 4.8: All the module clusters are merged into single nodes so that cross- module interaction becomes apparent.

cross-module interaction becomes apparent, as seen in figure 4.8. The user can see how the heart is truly the center of the system, whereas the chemoreceptors monitor the blood gases and drive the air mechanics.

Layout

Two different types of layouts for nodes and edges have been implemented. The icicle layout emphasizes the dependencies across the hierarchies, whereas the pie layout optimizes the arrangement of nodes for dependencies within a group. The pie layout is space-filling and is best for giving an overview of the variables.

The icicle layout represents arbitrary hierarchies by concentric rings growing inward. For this layout, hierarchical edge bundles[20] are used to render edges, as discussed in related work. Directed edges go from blue to red. Figure 4.7 shows a visualization of the full model, with bundling defaulted to 0.75. No node rearrangement is done here, because the layout is intended for edges across hierarchies. Because of the shallower hierarchy and fewer edges, the user chose to reduce bundling to 0.3 and increase the opacity of the edges. Figure 4.10 shows how the baroreceptors sense the aorta and affect the heart and systemic arteries, and the chemoreceptors sense the blood in the aorta and affect the respiratory system.

The pie layout fills the variables into areas that are wedges of a pie, so that pie consists of a uniform density of nodes. It supports a partitioning just one level deep. A discrete optimization permutes nodes within a group to minimize the sum of squared edge lengths, which creates a much cleaner layout. Edges are rendered as straight lines with arrows, so that color can be used for other purposes. Because the browser already has an alphabetical index and colors the nodes according to values of a field, the nodes are arranged to minimize edge distances. This strategy is common in graph layout schemes, though usually the solution involves a continuous optimization by simulating a spring-mass system. In the present system, the positions of the nodes are not moved, as the regular layout is needed given their density. In addition, the topology of the cardiovascular graph is complex and irregular enough that a spring-mass system would likely have difficulty remaining stable or converging. Instead a discrete optimization of repeatedly picking two random nodes and swapping their positions if the swap would result in a reduction of edge distances was chosen. The optimization runs in a fraction of a second, and upon completion (when it reaches a local minimum) yields a layout with an objective consistently twelve to eighteen percent of the original layout's objective. The system can put parameters further from the center than variables or intermix them. Figure 4.11 shows the advantage of a pie layout with edge optimization. The coronary circulation forms a nice chain of

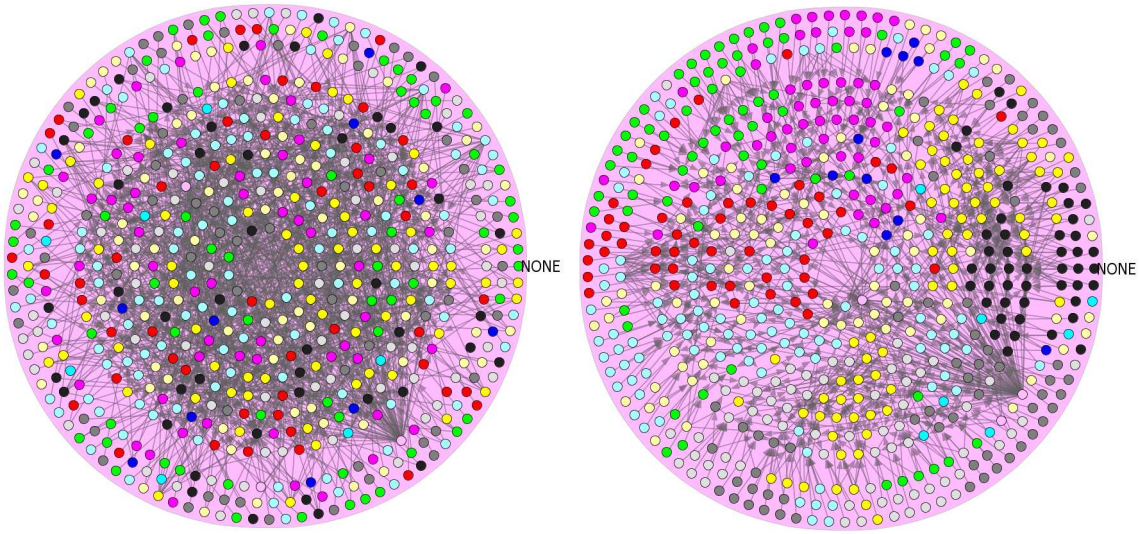

Figure 4.9: The cardiovascular model displayed with no clustering and color encoding the module. On the left, no edge optimization is done, and the graph is cluttered with edges. On the right, edge optimization produces a much cleaner image. Also note the difference in distribution of colors between the two.

flow propagation.

Figure 4.9 shows the cardiovascular model without any clustering and colored by module. Both diagrams show the equal-density circular layout. The left diagram does not have any edge-distance optimization, and the edges are distracting and nearly meaningless. The right diagram has the edge-distance optimization, and the difference is striking. Of further note is that the topology of the equations caused the optimization to cluster modules together, not surprising, since a well-written module should have many intra-module interactions and few cross-module interactions. As a corollary, edge-distance optimization can reveal modularity that is not explicitly defined. Certainly an idea for future work is to optimize for cross-module edges too or to optimize on a module hierarchy.

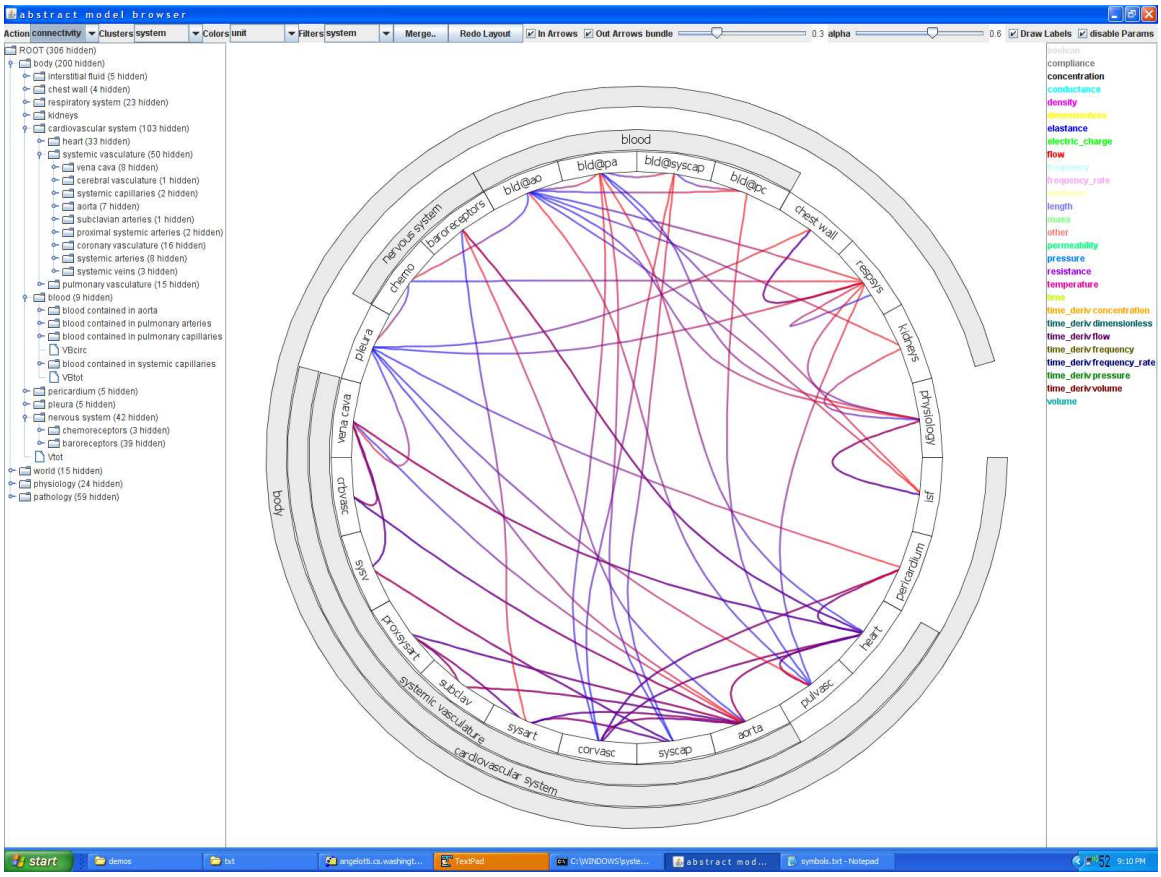

Figure 4.10: Summary view of model with hierarchy. Baro- and chemoreceptor ins/outs are clear.

Focus and Other Interaction

The user can also explore the in- and out-neighbors for a node recursively or browse feedback loops for a node. When focusing on a node and its neighbors, edges are highlighted and colored based on direction, and labels appear on the nodes, temporarily hiding unrelated nodes. Hovering pops up a tooltip that displays information on the variable. Figure 4.12 shows focusing on the set of merged baroreceptor variables and looking at second neighbors. In the screenshot, the user also loaded details of the equations associated with baroreceptors. Figure 4.13 shows that the lungs apply pressure onto the heart by way of the pleural chamber pressures and pericardial chamber pressures.

The whole system supports smooth panning and zooming on the graphs, and transitions are animated when possible to establish temporal coherency. For the hierarchical edge bundling, sliders exist to adjust bundling and opacity.

4.3 Conclusions and Future Work

The model browser was built to aid the researcher in reasoning qualitatively about a model specified in code. By interactively navigating graphs, filtering variables, and hierarchically browsing, a user can answer many questions that would be hard otherwise with just the source code. Revisiting the questions presented at the beginning of the chapter,

- "Show me the variables for just the vascular flow portions of the model." (figure 4.4)

- "What role does temperature play in the model?" (figure 4.6)

- "Does the model have baroreceptors in the aorta or the carotids or both?" (figure 4.10 and figure 4.12)

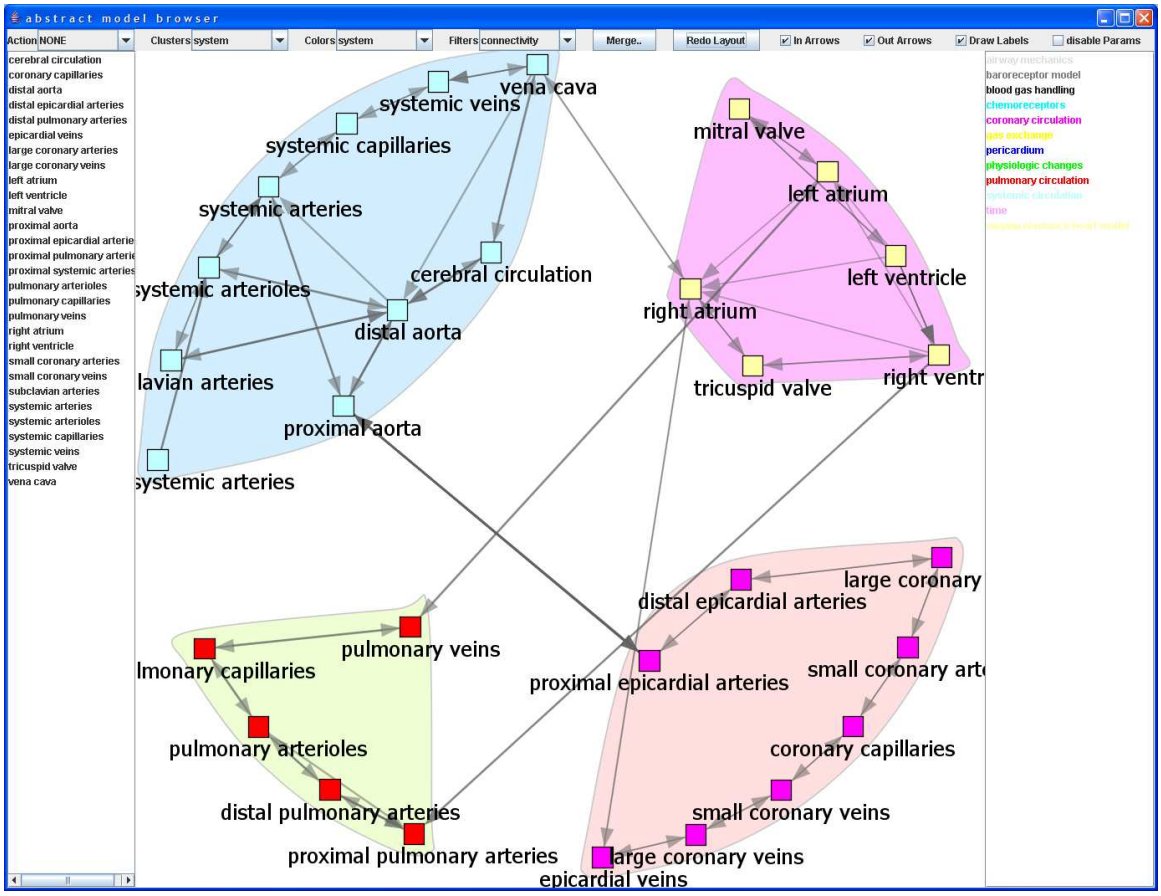

Figure 4.11: Interactions between anatomical parts, mostly fluid flow. Layout optimizes paths.

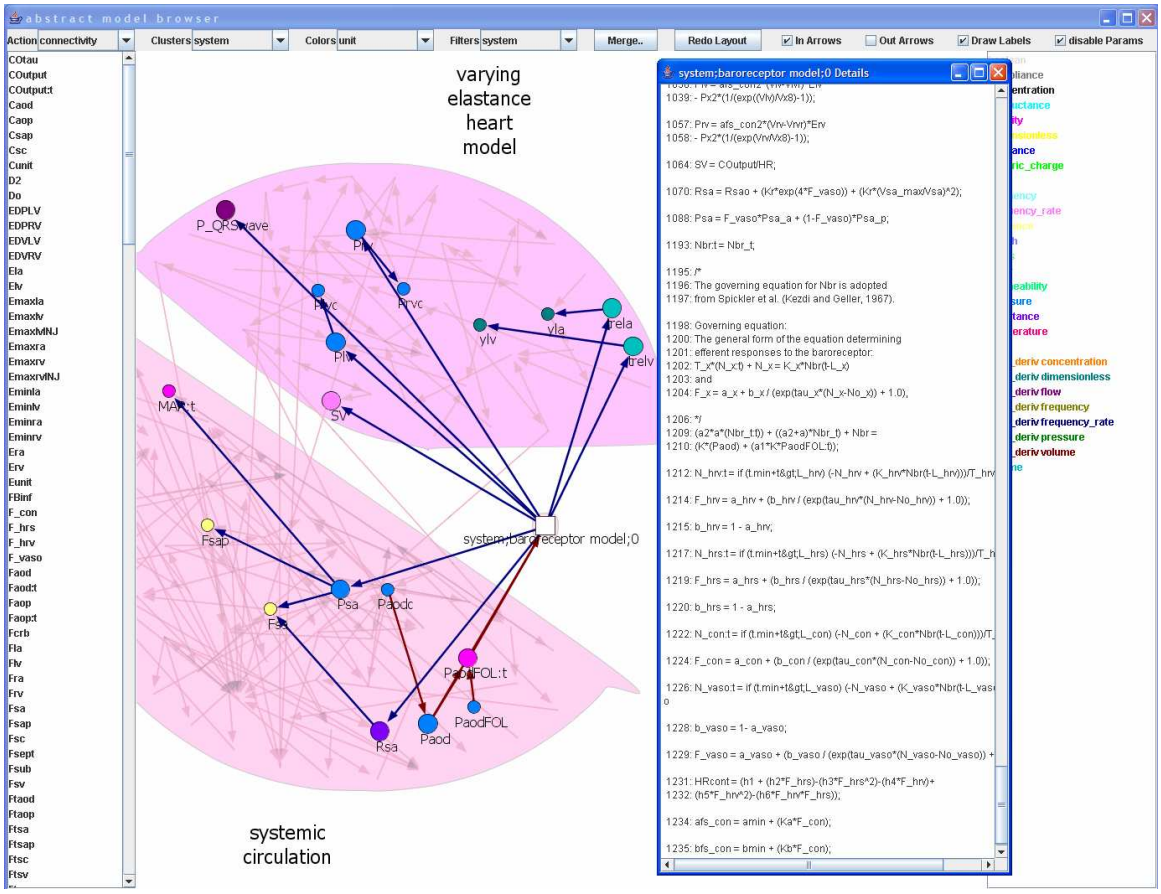

Figure 4.12: Exploring connectivity of baroreceptor module. All baroreceptor equations are shown.

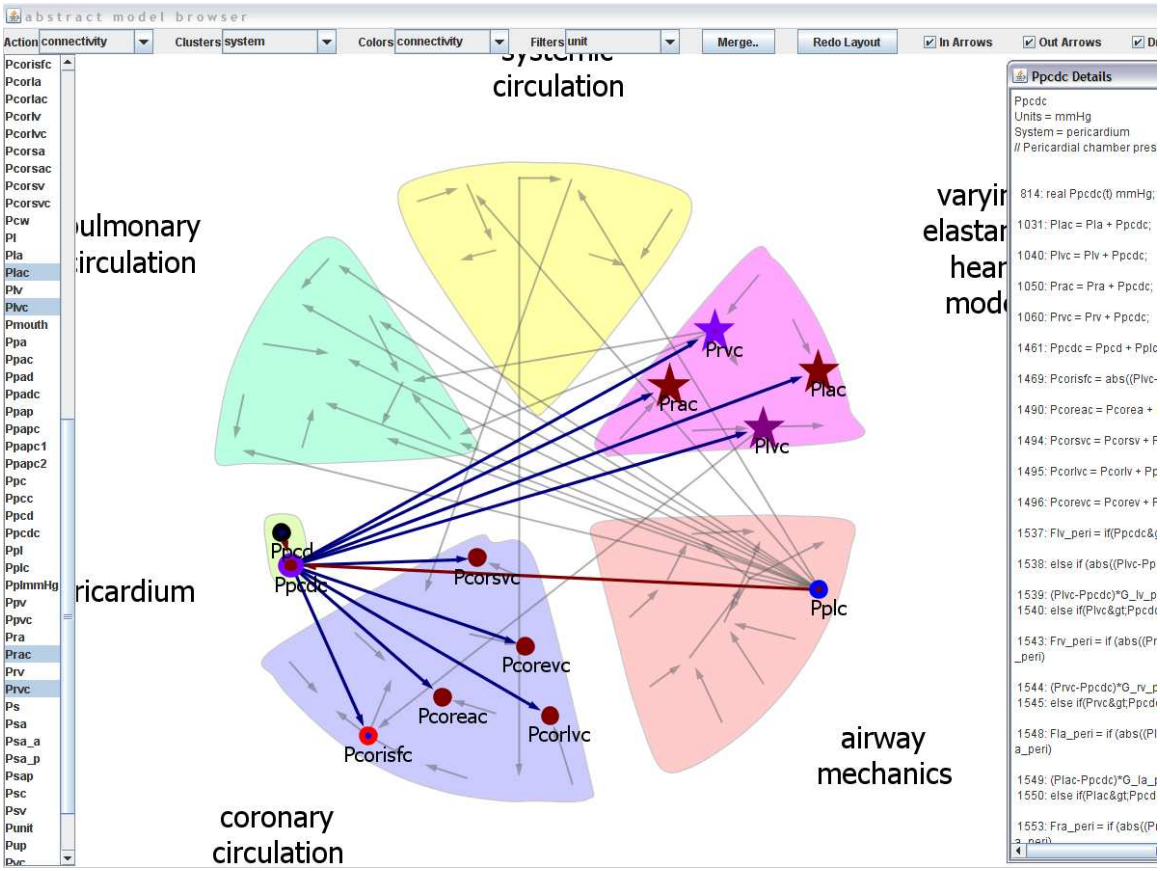

Figure 4.13: In this diagram, it can easily be seen that the pleural chamber pressure influences the pericardial chamber pressure, which respectively influences the chamber pressures in the atria and ventricles of the heart.

- "Do the lungs exert pressure on the heart?" (figure 4.13)
- "What parameters modify the P-V curves?" (figure 4.3)
- "In the model, what are the paths of blood through the cardiovascular system?" (figure 4.11)

one can see that the system can assist a user in bridging the cognitive gap between the computational model and more abstract ideas. Filtering and clustering play key roles in pruning the space of variables and equations to just the relevant. Merging, whether an explicit hierarchy is present or not, adds structure to unstructured equations. Finally, the ability to focus on neighborhoods of a node enables the user to see related equations that may be spatially distant in the model's source code.

Several bioengineers have examined the tool and have provided positive feedback and suggestions for improvement; a user study is next, once the software has reached a more robust and complete state, on as would be expected of a commercial application. Users will be timed while performing specified tasks, and a questionnaire will be used to evaluate the usability of the system. I also would like to try other models, especially ones involving metabolic pathways or ion channels. Another good test of the system would be to obfuscate a model by introducing temporary variables and superfluous equations, e.g. instead of $x : t = y$, have $x : t = z; z = y$. A powerful model browser should be able to cut through the chaff and reveal the underlying structure to the user.

Currently the layout, both inside a cluster and outside, is circular, without any focus on any element. Other strategies may be more elucidating, such as a more linear layout from a focused node. Additionally, the user may want to rearrange the nodes to a personal layout that can be saved and reused.

The browser's operations are mainly on nodes, but operations on edges would be useful too. A few examples include whether to draw cross-cluster edges, coloring

edges based on their physical properties (flow, pressure from adjacent tissue, chemical reaction, etc.) and exploring edges until a terminating condition is reached, upon which the path is compressed into a single edge. This path compression could be useful when connecting differential equations to each other without the intermediate variables and may yield a result close to the modeler's sketch of a circuit diagram.

Many models contain canonical equations that are instantiated many times, e.g. conservation of flow. We would like to have a novice interface for searching for such equations (in essence a query of neighborhoods), or be able to detect repeated patterns of equations. Advanced queries could aid in bug hunting or explaining auto-generated code.

The model browser could be extended to reason about a model quantitatively, by enabling dynamic queries on simulation data, visualizing sensitivity analysis, and clustering variables with similar behavior. Preliminary tests on calculating the principal components of the simulation data suggests that the vast majority of the information is captured in just a handful of bases. In addition to being a generic model browser, the tool could be customized for specific tasks such as debugging or parameter tuning.

4.3.1 Debugging

An extended model browser could help the developer catch many types of errors that he or she may encounter; several are listed below, along with discussions of how the browser could be extended to detect them. A full set of interviews and user studies with bioengineers would establish the frequency of these errors and the potential effectiveness of these tools.

Consider the following two lines of code excerpted from the cardiovascular model.

```
Ftaod = Faop - Faod - Fcrb - Fsub; Ftsap = Faod - Fsap + Fsub;
```

Flow in an elastic vessel consists of two components. The first component is the rate that a unit volume of blood travels through the vessel. This number is calculated by a variant of Ohm's Law, that is, the flow (current) is the pressure

over the resistance. The second component is radial flow, which accounts for the vessel changing its volume (capacity) due to an increase or decrease in flow before or after—an inelastic vessel would have zero radial flow. In the first line of code, the change in volume of the distal aorta is equal to the output of the proximal aorta that is not going to the cerebral or subclavian arteries minus the output of the distal aorta. In the second line, the change in volume of the aggregate proximal arteries is the inputs (distal aorta and subclavian artery) minus the outputs (aggregate of proximal arteries). The subclavian term might seem strange in both equations, but it has a sound explanation. The subclavian artery belongs to the aggregate of proximal arteries, but it branches from the top of the aortic arch as opposed to the descending aorta.

Let us look at three possible mistakes for the second line. In all cases, an astute programmer running a test for conservation of total blood volume will notice a problem but will be unable to localize it to one of a hundred lines of flow equations.

```
Ftsap = Faop - Fsap + Fsub;
```

In this example, as written, blood flows from the proximal aorta, not the distal aorta, a subtle typo that could have resulted from fatigue or sloppy cut-and-pasting. When the visualized with anatomy merged, an arrow would be present from the proximal aorta to the aggregate of proximal arteries, which would be immediately noticeable as errant.

```
Ftsap = Faod - Fsap - Fsub;
```

Here the problem is an incorrect sign on the subclavian term, fixed by switching the sign or enclosing the last two terms in parentheses. Though the topology of the network is correct, the performance is incorrect. Here, if the user could invoke a tool that uses sensitivity analysis to visualize "an increased flow here causes an increase or decrease there" then an errant pattern would appear in the visualization that the user would quickly isolate. The diagram in figure 4.14 illustrates how a visualization could quickly reveal an error that would be difficult to find by just browsing code.

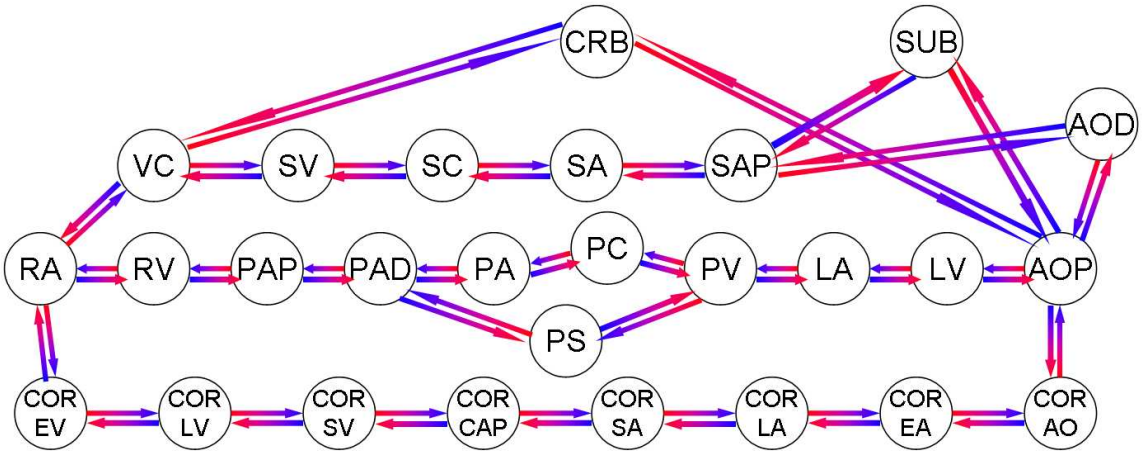

Figure 4.14: Rendition of a possible visualization that can reveal errors through sensitivity analysis. The error is where two adjacent arrows have colors going in opposite directions.

```
Ftsap = Ftaod - Fsap + Fsub;
```

This error is the wrong distal aorta flow variable. Though it may be possible to introduce a sentinel dimensional unit to differentiate flow from radial flow (and correspondingly populate the code with needed conversion factors) so that the bug could be caught by unit-checking, the extra labor on the programmer and the extra clutter in the code is probably counterproductive. This example is the most challenging to catch, as the units and anatomy are correct. However I believe that with appropriate visualization and layout, a user could detect something anomalous in the network topology and isolate the bug.

In addition, the developer should be able to tag variables and equations as "verified." That way, for a difficult bug, the developer could prune the model down to the suspect parts and eliminate them one by one. Similarly, sandboxing by treating merged nodes as black boxes could help isolate and test suspect code.

Chapter 5

ONTOLOGY BROWSER FOUNDATIONS

Ontologies are large collections of terms, relationships between them, and rules for reasoning about the contained knowledge. The ontology browser relies on a substantial backend to manage the data loaded from the ontology and any modifications to the ontology. Furthermore, the constraints on how an ontology can be changed dictate the interactions granted to the user.

5.1 Ontology Views

The ontology browser visualizes a view of an ontology and can be used to produce another view of an ontology. The view can be the identity (the ontology itself) or it can be a derived view produced by the ontology browser or some other application/service. A view acts as a virtual ontology, without actually storing or materializing the ontology. As a result, a view has a compact representation and can be changed with minimal overhead. Views are useful for presenting abstractions to the user, for example to hide irrelevant information from a user with specific needs. One can also add or change content in a view. Theoretically a view could also incorporate multiple ontologies, although if the ontologies are not orthogonal, there could be naming conflicts that need to be resolved. In database theory, a view is specifically defined as a virtual table constructed from the result set of a query. For the ontology view, the recording of the modifications to the ontology is not necessarily a query, but foreseeably, should be translatable into one.

Figure 5.1 illustrates the basic architecture of the foundational ontology, intermediate views, and the eventual visualization or other application. A sequence of

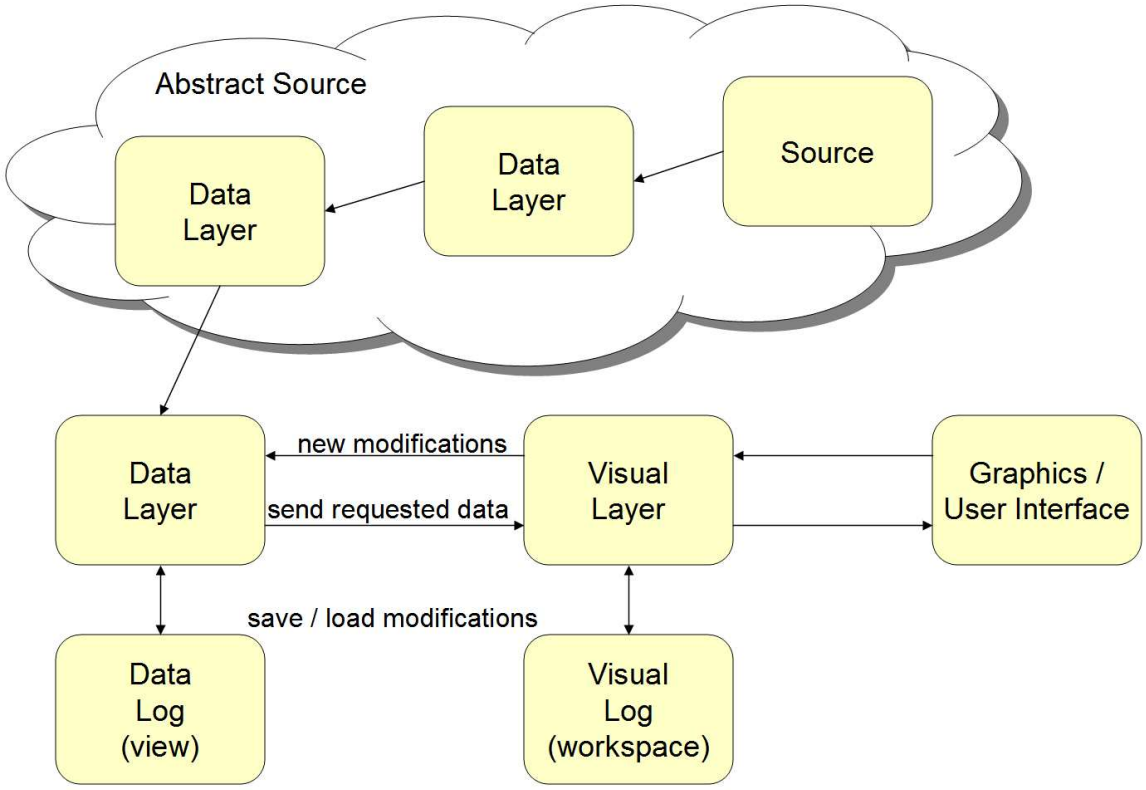

Figure 5.1: The DataLayer produces a view of an ontology. DataLayers can be chained together, and the view can be materialized to a new ontology.

chained views is abstractly the source ontology and indistinguishable from a materialized ontology. Each of these views is called a DataLayer. In the next chapter, the VisualLayer, which adds visual components to the view for the sake of the visualization, will be introduced. The ontology browser specifically allows the user to issue modifications to the DataLayer via the VisualLayer. The modified DataLayer can be saved as a new ontology view and distributed to others compactly or materialized.

5.2 Ontology Theory

In this section the formal definition of an ontology and its semantics will be given. Using this specification, the transformations onto the ontology that guarantee that

the semantics still hold will be defined. These transformations form a view of the ontology. The ontology definition used here does not include constraints on domains and ranges, cardinalities, etc. – for several reasons. First, the original ontology may not be compliant even if it claims to be so, and it is pointless to try to uphold semantics that are not true in the first place. Second, too many constraints may limit users' flexibility, or at the best, create a challenging user-interface problem for the developer. Finally, as a prototype, it is not necessary to solve the problem fully – rather to provide a template that can be refined.

5.2.1 Formalizing an ontology

An ontology O is a triple

$$O = \{\mathcal{E}, \mathcal{A}, \mathcal{R}\},$$

where $\mathcal{E}$ is the set of all defined entities, $\mathcal{A}$ is the set of all defined attributes, and $\mathcal{R}$ is the set of all defined relations. I will also refer to these sets as alphabets. An entity

$$e \in \mathcal{E},\ e = \{\mathcal{A}_e, \mathcal{R}_e\}$$

consists of the set of attributes $\mathcal{A}_e$ allowed for e and the set of relations allowed for e. An attribute $a(e) \mapsto T,\ a \in \mathcal{A}$ maps an entity to a list of values of specified type T (e.g. integer, float, string, boolean). The values may be defined or unspecified. A relation

$$r(e) \mapsto \mathcal{E}_{r(e)},\ r \in \mathcal{R},\ \ \mathcal{E}_{r(e)} \subseteq \mathcal{E}$$

maps an entity to a set of entities.

Elsewhere in the literature, one may see attributes and relations all grouped together as slots, with constraints called facets. Further definitions may follow for distinguishing instances from classes or allowing multiple inheritance, such as for the sake of allowing an entity to be both a class and an instance. The view taken in this

work is that for the user, attributes and relations have significantly different connotations, which warrants reasoning about them separately. A second assumption is that the user does not care about the subtlety of a top-level class inheriting both from root and a template. Reified relations, which are really instances that contain properties associated with the specified relations, are specifically ignored, but adding them to the framework would not be difficult.

Every entity e has a special relation p, the superclass (parent) relation. The relation $p(e) = P$ is further constrained so that the set P is a singleton for all e except for the parent-less root, and the parent graph is acyclic. In other words, every entity has a unique finite ancestry to the root. Inheritance forces further semantics on the sets of allowed relations and attributes for entities. Let $e_p = \{\mathcal{A}_{e_p}, \mathcal{R}_{e_p}\}$ be the parent of e, that is $\{e_p\} = p(e)$. Then

$$\mathcal{A}_{e_p} \subseteq \mathcal{A}_e \qquad \text{and} \qquad \mathcal{R}_{e_p} \subseteq \mathcal{R}_e.$$

We can now define inverse relations rigorously. Every relation r has a unique inverse r^{-1}. Some relations are self-inverses (e.g. continuous with), and some are not (the inverse of superclass is subclass). The following property holds with relations and their inverses (akin to saying that every outgoing edge has a corresponding incoming edge):

$$e' \subseteq r(e) \iff e \subseteq r^{-1}(e')$$

Let the relation r_{sub} be a subclass of a relation r. The following properties hold:

$$r_{sub} \subseteq \mathcal{R}_e \rightarrow r \subseteq \mathcal{R}_e, \qquad r_{sub}(e) \subseteq r(e).$$

The inverse relations must be allowed for the entities in $r(e)$, that is

$$r^{-1} \in \mathcal{R}_{e'},\ e' \in r(e).$$

The superclass and subclass relations are allowed for the root of the ontology (and hence all other entities), and likewise they are constrained to have values that are roots (i.e. any of the entities). For accessibility, unique identifiers such as universal resource identifiers (URIs) refer to each entity, attribute, relation, and attributed relation.

5.2.2 Legal modifications to an ontology

Now that we have rigorously specified the semantics for an ontology, we can define transformations on the ontology, as well as what needs to be done to guarantee that all the semantics still hold. We will consider the transformations of adding, deleting, or changing entities, attributes, relations, attributed relations, and constraints.

Additions

If we introduce a new entity, it must be assigned a parent. The new entity will inherit the parent's set of allowed attributes and relations. Adding a new attribute to the ontology requires no further operations. Adding an attribute to an entity means that all descendents of that entity now inherit the attribute. A relation must be added to the ontology with its inverse. As with adding an attribute to an entity, adding a relation to an entity means that all descendents of the entity now inherit the attribute. A subrelation can be added to an entity only if the relation is already present.

Deletions

Deleting an attribute from the alphabet of attributes for the ontology results in all instances of the attribute in entities being deleted. Deleting an attribute from the allowed list of attributes for an entity is only legal if the attribute is not present in the parent (or the entity is the root). Deleting is straightforward, though one must decide whether to add the attribute to any of the children or to delete the attributes

from the children as well. Deleting a relation from the alphabet of relations for the ontology results in all instances of the relation and its inverse in entities being deleted. Deleting a relation from the list of allowed relations for an entity is only legal if the relation is not allowed for the parent. Again one may decide to add back the relation for a child. Deleting the relation removes its value set from the entity, which in turn means that the value sets for the inverse relation need to be modified (if $e' \in r(e)$ and we delete e, then e must be removed from the set $r^{-1}(e')$). Deleting a relation deletes its subrelations.

Deleting an entity means that either its descendents are deleted too, or their parents need to be reassigned to the parent of the deleted entity. Again the two choices lead to two editing operations for the user. All relations and attributed relations associated with the entity are deleted, and the entity is removed from the value sets from the inverse relations. That is, if $e' \in r(e)$ and we delete e, then e must be removed from the set $r^{-1}(e')$. The logic can be applied to other user operations, for example, undeleting an entity that had previously been deleted. The parents of the undeleted entity would likewise need to be undeleted, and it would be a design choice whether or not to undelete the children.

Changes

The value of an attribute can be changed from undefined to defined, or it can be changed from one value to another. The type of an attribute can only be changed if for all instances of it, the values are undefined. A simple extension is to specify whether or not an attribute can be overriden once defined. Changing the value set of a relation on an entity (akin to adding or deleting an edge) simply requires the corresponding inverse edge be changed.

A special operation is to change the parent of an entity. To make the entity a child of its grandparent or further ancestor, only the parent edge and its inverse need to be reassigned. It will keep all the same sets of allowed attributes, relations, etc. For

implementation purposes, the entity may have some of those attributes, relations, etc. explicitly allowed for it rather than inherited from the parent. To make the entity a child of one of its children or further descendent, that descendent becomes the child of the entity's parent (as just described), and the entity becomes its child. However in this case, the entity may need to inherit some from its new parent. To make the entity a child of an unrelated entity, there may be both attributes and relations that are no longer required and that it needs to inherit.

5.3 DataLayer: Implementation of an Ontology View

Now that we have defined an ontology and legal transformations on the ontology, we must think how to implement the recording of these changes and the necessary queries involved to obtain the needed information. To summarize, the changes include adding or removing from the alphabets of entities, attributes, and relations, adding or removing from the allowed attributes and relations for an entity, modifying the values of attributes for an entity, and modifying the values of relations for an entity. Semantics regarding subclasses, subslots, and inverse relations need to be enforced. Challenges arise when ensuring that the implementation scales well for large ontologies.

5.3.1 Ontology Assumptions

To guide our design choices, we will make the following assumptions about ontologies:

underlying semantics of the original ontology are sound: Though a user may not need a rich set of semantics on a derived view, a minimal set of semantics is required for the efficient logic used in the DataLayer. Relations must have inverses, and each specific edge must have a corresponding inverse (possibly itself). This rule allows for an inexpensive reverse lookup. Additionally simple semantics regarding inheritance must hold (e.g. a slot defined in the parent must be defined in the child), and single inheritance must be present to allow

the delete and undelete subtrees. Note that if a user wants to effectively ignore all inheritance rules in a view, an option is to flatten the taxonomy completely.

ontologies are large: We will assume that it is not feasible to store the whole ontology or even all the transitive descendents of an entity. As a result, any queries or edits will be propagated toward the root. If a child needs to know about a change above, it will ask its parent (recursively), rather than the parent broadcasting to the children.

not too many attributes, relations, or attributed relations: We will assume it is feasible to store the set of defined attributes, relations, and attributed relations in memory. Implementation-wise, whether an entity is likely to have more relations, or a relation have more entities in its domain, may lead to design decisions.

entity-relation graph is sparse: The entity-relation graph is sparse, so it will be best to store edge-relation graphs as adjacency lists rather than as adjacency matrices.

subclass hierarchy is not too deep: It will not be too costly to store, query, or traverse all the ancestors for an entity. In addition, much data, for example allowed attributes, can be stored as differences from parent to child, and the full attribute set integrated computationally by traversing to the root.

small set of changes to create a view: One does not need to be too concerned about making the set of changes have minimal computation or storage, and likewise, the view can be stored concisely by representing the changes (as a log, as query transformations, etc.) rather than materializing the modified ontology.

The current release of the Foundational Model of Anatomy[40] has roughly 78,000 entities, 200 types of attributes and relations, in excess of a hundred thousand in-

stances (many just synonyms but some reified relations), and a million entity-relation edges. Its depth is 19 (Dorsal digital vein of left big toe, along with 17 other veins of the feet, are of that depth). Only the part relation has subrelations. The value sets for most relations on entities are small, often one or two. Notable exceptions are that hundreds of types of tendons have Tendon as a parent, and Bona-fide anatomical space has thousands of direct instances.

5.3.2 A compressed representation for the FMA

The alphabet of attributes, relations, and attributed relations for the FMA can be assigned to different bits, and then each all possible combinations of which are allowed for an entity can be represented by a 32-byte bitfield. In contrast, if pointers were stored per attribute or relation, each would take a full 32 bits. There is much redundancy in this format, and no doubt, cleverly storing differences could save more space.

Each entity is stored in a map linking to all its relations' value sets. The format is the id of the first non-empty relation, the cardinality of the value set, references for each of the values, the next non-empty relation, etc. Though this format is compact, one may have to search through all relations to find the one desired. In practice, this computation is not an issue. Attributes that are strings are stored via Lucene (http://lucene.apache.org). The storage is compact, and the search capabilities are quite impressive.

Though a "proprietary" format goes against the principles of the web, as long as converters exist, there are many advantages to having a compact representation. The Protégé/MySQL incarnation of the FMA takes in excess of 500MB of disk space and takes several hours to download and construct the database. Furthermore, requiring the additional applications MySQL and Protégé discourage the average user from dabbling. An additional problem with MySQL is that it may require administrator privileges to install. Instead of having to communicate over the network because the

FMA is too large to store locally, or spend half a day installing it locally, the whole FMA can be conveniently downloaded in under a minute as a single payload and be ready to go. Of important concern in the medical community is that of patient privacy. Note that the compact proprietary format could be encrypted, should there be sensitive data present. Only authenticated software could then properly read the data, and no security would be lost by storing the data locally.

Another assumption that is applicable is that foundational ontologies such as the FMA do not change often. In other words, the user would not have to re-download often, nor would the expensive conversion of the FMA to a condensed format have to be performed often by the curators. Furthermore, if a curator wanted to transmit minor changes about the FMA, the changes could be encoded in a DataLayer—in essence, a patch.

5.3.3 Implementation

The DataLayer is an abstract interface that loads information from a previous DataLayer, sends information to the next DataLayer, and perform modifications. At the bottom are implementations of the DataLayer that operate directly on specific databases or web services. However, the most important DataLayer is the DataLayer that operates on a previous DataLayer. The ability to chain DataLayers means that any sequence of DataLayers can be perceived as an abstract source, perhaps even materialized as an ontology. In the future, DataLayers might even load information from multiple ontologies, not just a single source. Figure 5.2 shows the insides of a DataLayer. Each of the other DataLayers in the figure self-similarly has its own other insides respectively. The asking arrow from a DataLayer to a previous DataLayer represents the load interface, a set of methods loading information on entities, relations, attributes, etc., and the sending arrow from a DataLayer to the next DataLayer represents the send interface, a set of corresponding methods to send the data. Figure 5.3 shows part of the DataLayer interface, along with different implementations for one

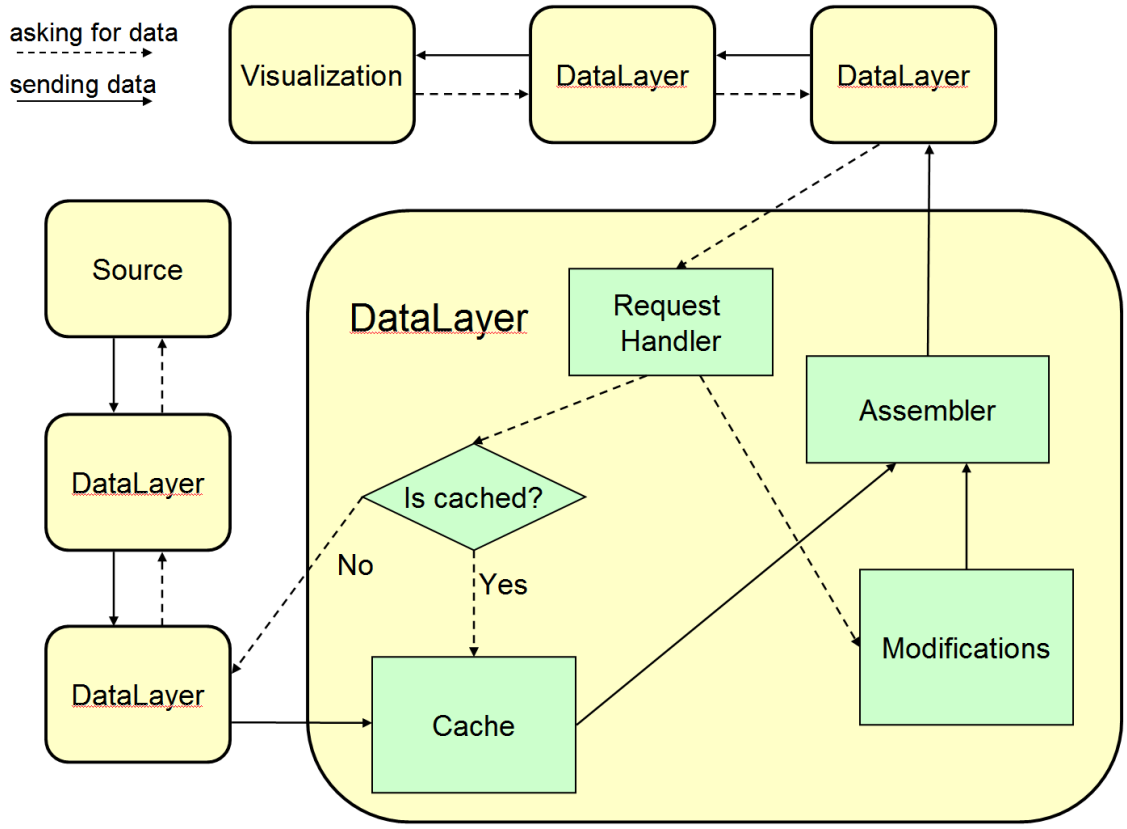

Figure 5.2: The Inner workings of a DataLayer. The cached data from the previous DataLayer's output plus the modifications are assembled to form this DataLayer's output.

of the loading methods. One implementation is loading from another DataLayer, and the other is loading from a Protégé database. The sending of information to the next DataLayer, as well as queries on and modifications to the current DataLayer are implemented in the AbstractDataLayer, so that only the loading operations need to be implemented for derivations that operate on different sources.

```
public class AbstractDataLayer {
   ...
    public abstract BaseEntity[] loadRelateds(BaseRelation r, BaseEntity e);

    public final String[] sendRelateds(String relName, String entityName) {
        BaseRelation relation = getRelation(relName);
        BaseEntity entity = getBaseEntity(entityName);
        ArrayList<String> al = new ArrayList();
        Iterator<BaseEntity> it = relation.modified().relatedDirect(entity);
        while (it.hasNext()) al.add(it.next().getName());
        return al.toArray(STRINGARRAY);
    }
}

public class DataLayerDataLayer extends AbstractDataLayer {
  ...
    public BaseEntity[] loadRelateds(BaseRelation r, BaseEntity e) {
        if (e.isNewEntity()) return EMPTYARRAY;
        String[] names = parent.sendRelateds(r.getName(), e.getName());
        BaseEntity[] ret=new BaseEntity[names.length];
        for(int i=0; i<names.length; i++) ret[i] = getBaseEntity(names[i]);
        return ret;
    }


public class ProtegeDataLayer extends AbstractDataLayer {
  ...
    public BaseEntity[] loadRelateds(BaseRelation r, BaseEntity e) {
        Frame f = protegeKnowledgeBase.getFrame(e.getName());
        Collection<Frame> c = f.getOwnSlotValues(
                                protegeKnowledgeBase.getSlot(r.getName()));
        BaseEntity[] ret = EMPTYARRAY;
        if (!c.isEmpty()) {
            ret = new BaseEntity[c.size()];
            int j = 0;
            for(Frame neighbor:c)
                ret[j++] = getBaseEntity(i.next().getName());
        }
        return ret;
    }
}
```

Figure 5.3: Code for the DataLayer

Deletions and Exceptions

In the spirit of the ontology assumptions, the DataLayer represents the deletion of a subtree of entities by storing a deletion mark plus a timestamp in a hash table indexed by the entity. When an entity is queried, the DataLayer checks to see if any of its ancestors are marked deleted. Given that the number of total deletions is likely small and the number of ancestors for an entity is small, this implementation is inexpensive with respect to both space and time.

Representing only deleted subtrees is not expressive enough to be useful. The DataLayer augments deletions with exceptions to deletions. When an entity is an exception to a deletion, it, all its children, and all its ancestors are no longer deleted. The hashtable storing deletions also stores these exceptions. An exception is marked by denoting the source of the exception with a special mark and then marking the source's ancestors with another mark. When the traversal of ancestors happens, the order of the timestamps resolves which were deleted before exceptions or after. To summarize, an entity is considered deleted in one of two situations:

- the most recent timestamp for a mark of itself or its ancestors is a delete
- the most recent timestamp for a mark of itself or its ancestors is an except, and neither the entity itself is most recently marked excepted nor do the entity's ancestors include the source of the exception

A final sticky point is the interaction with the change-parent operation and subtree deletions and exceptions. The exception chain described previously may be broken by the move, and the new ancestors of the entity may have their own deletions/exceptions. There is no correct answer to what the solution is, but one important consideration is to minimize the confusion of the user. The confusion arises in particular because in the visualization, there is no representation of what deletions or exceptions were performed or when, so the user could be surprised if a subtree were

moved and suddenly all its children changed whether or not they appeared deleted. Another consideration is that any further changes be easily undoable. The rule is to preserve the outward appearance as much as possible upon the move. The additional operations are as follows:

- If the entity and the entity's to-be-reassigned parent both appear deleted, the entity is marked deleted and the parent is reassigned.

- If the entity appears deleted but the entity's to-be-reassigned parent does not appear deleted, then the entity is marked deleted and the parent reassigned.

- If both the entity and the entity's to-be-reassigned parent do not appear deleted, then the appearance of the subtree needs to be preserved. The most recent of the deletes and excepts performed on the entity's ancestors are performed on the entity itself, but its timestep is made current. This mark also has an extra annotation that any timestamps in the descendents trump any timestamps of value less than current residing in the new ancestors. The parent is reassigned.

- If the entity does not appear deleted but the entity's to-be reassigned parent appears deleted, then further specification is needed whether to delete the entity or to except the to-be reassigned parent. Once either choice is performed, the state resolves to one of the previous cases.

ModifiedIterator

At the heart of the DataLayer is the assembler in figure 5.2. Given the data from the previous DataLayer (cached) and the modifications, the assembler needs to produce the result dynamically. Manipulating lists has several disadvantages, among them that huge data structures may need to be allocated, and multiple list copies may occur. Iterators have the advantage of being able to pass on one item at a time,

making it easy to skip an item or insert an item. Iterators can be easily nested. The ModifiedIterator is an iterator that takes in an iterator on base items, a hashset of deleted items, and an iterator on new items. It returns the items that form the union of the base items and the new items, minus the deleted items. With parametric types, it can be used for handling entities, attributes, relations, etc. Figure 5.4 shows the ModifiedIterator code, and figure 5.5 shows the ModifiedIterator being used (see figure 5.3 again to see how they all fit together).

```
public class ModifiedIterator<T> implements Iterator<T> {
    private static final Set EMPTYSET = new HashSet();
    private Set delSet;
    private Iterator<T> iter, newIter;
    boolean ok = false;
    T ret;

    ModifiedIterator(Iterator<T> iter, Iterator<T> newIter, Set delSet) {
        this.iter = iter;
        this.newIter = newIter;
        if (delSet == null) this.delSet = EMPTYSET;
        else this.delSet = delSet;
    }

    public boolean hasNext() {
        while (it.hasNext() && !ok) {
            T t = it.next();
            if (delSet.contains(t)) continue;
            ok = true;
            ret = t;
        }
        if (newIter != null) {
            while (newIter.hasNext() && !ok) {
                T t = newIter.next();
                ok = true;
                ret = t;
            }
        }
        return ok;
    }

    public T next() {
        if (!ok) hasNext();
        if (!ok) throw new NoSuchElementException();
        ok = false;
        return ret;
    }
}
```

Figure 5.4: Code for the ModifiedIterator.

```
public class ModifiedRelation extends AbstractRelation {
  ...
    public Iterator<BaseEntity> relatedDirect(BaseEntity e) {
    return new ModifiedIterator(new SkipDelIterator(base.relatedDirect(e)),
        addEdges.get(e) == null ? null : addEdges.get(e).iterator(),
        delEdges.get(e));
    }
}

public class BaseRelation extends AbstractRelation {
  ...
    protected BaseEntity[] loadRelateds(BaseEntity e) {
        BaseEntity[] outs = edges.get(e);
        if (outs == null) {
            outs = dl.loadRelatedsAsArray(this, e);
            edges.put(e, outs);
        }
        return outs;
    }

    public Iterator<BaseEntity> relatedDirect(BaseEntity e) {
        if (e.isNewEntity()) return EmptyIterator.EMPTY_ITERATOR;
        return new ArrayIterator(loadRelateds(e));
    }
}
```

Figure 5.5: Excerpts of code for original and modified relations.

5.4 Conclusions and Future Work

The architecture for constructing views is powerful and flexible and should be easily extensible for incorporating queries. The vision is that in the future, the architecture would be like figure 5.6, with the database views chained together transforming queries, but still fit smoothly into the existing visualization framework. Straightforward extensions of the DataLayer would include incorporating further constraints (facets), and better support for instances. Additionally, it may be desirable to materialize the logs (or query transformations) plus the source ontology into a new ontology.

An extension of the DataLayer allows it to masquerade as a triple store and serve as the back end for Jena[1], a middleware for Semantic Web applications. A SPARQL[2] query through a webservice is decomposed by Jena into individual requests for sets of triples. One challenge is that a request may ask for many triples (e.g. triples with any subject and any object but a specific predicate), and Jena would internally filter the triples. This inconvenience stems from the fact that SPARQL's expressiveness is richer than that of SQL, so a more specific query may not be capable of being sent to the back end. SPARQL does allow the user to request a query to be passed on by Jena, but this feature removes the notion of an abstract source and requires the user to possess expert-level knowledge on how the view was constructed. The best solution may be for the chaining of views to be performed entirely in SQL, so that queries may be better transformed and passed on, rather than costly extra data be transferred per each link of a potentially complex chain of views.

An excellent direction for future work would be to try to consolidate the modifications by removing redundant operations, refactoring operations, or deducing equivalent queries (presuming the source view does not change)—in essence, taking advantage of the Kolmogrov complexity of a set of modifications. A simple transformation

---

[1] http://jena.sourceforge.net

[2] http://www.w3.org/TR/rdf-sparql-query/

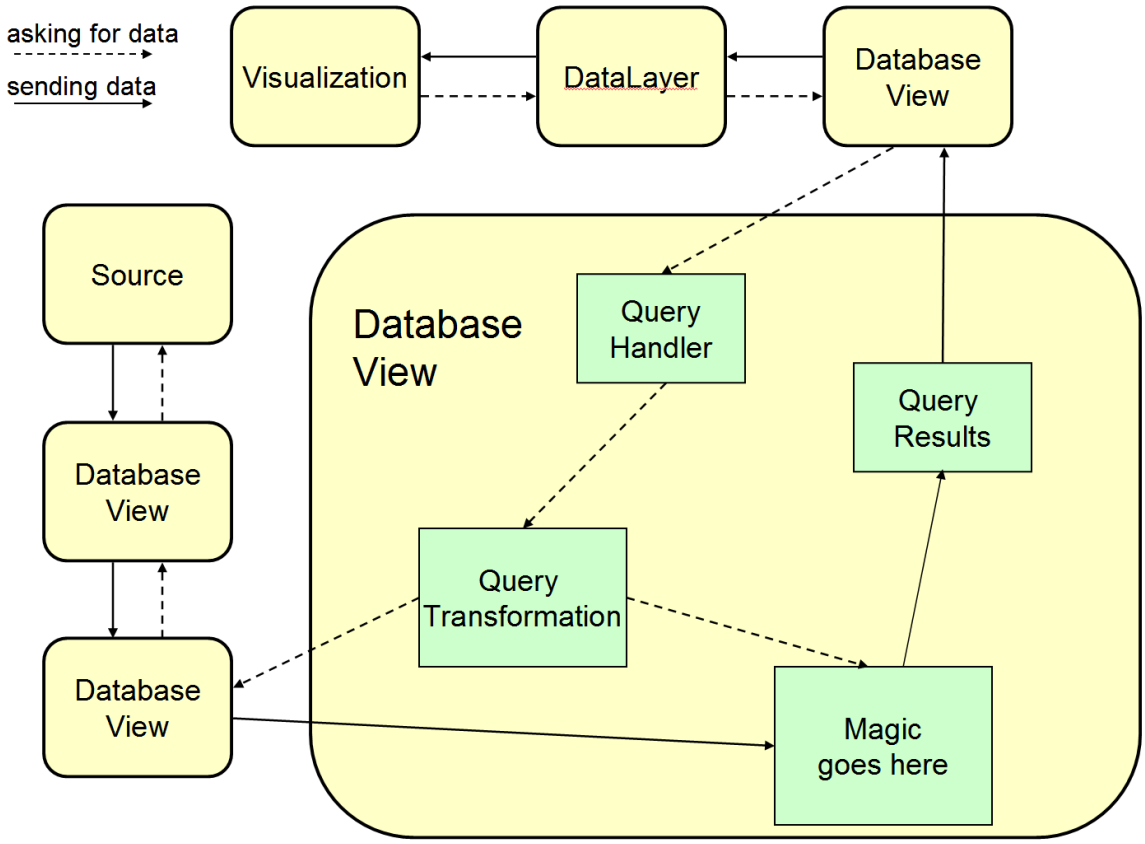

Figure 5.6: Database views chain together via query transformations. On the frontend, a DataLayer queries the database view and provides information for the visualization.

involving deletions and exceptions could involve recognizing when all the children of an entity are deleted (or respectively excepted) and instead mark equivalently that the entity itself is deleted or excepted. Other transformations could be operating on the result of a query instead of operating on an explicit set. Such an optimization is really only appropriate if the queries are on the previous view or the current view is frozen (saved), as any changes to the view could affect the inferred queries. Very likely, any work on optimization would be necessary as a part of translating the modifications into queries.

Originally the design was expected to include more semantic inference.. The inference was much more intricate than expected, and the original ontology was not consistent over the additional logic. The only inferences made were with relations having inverses and inheritance with subclasses. Much work could be done with defining inferences (and their rules with respect to editing) and experimenting with their usefulness. One area that was specifically ignored is multiple inheritance. What happens to an entity when one parent is deleted but not the other? Perhaps one way to solve the problem is by using Java's approach: a class can have just one parent but can implement many interfaces. In this respect, an entity has a single superclass but perhaps multiple templates, which can be deleted. Removing a template removes its respective slots. Additional rules would need to be defined to avoid naming conflicts between slots in multiple templates.

Chapter 6

ONTOLOGY BROWSER INTERFACE

The browser visualizes an ontology as a graph, with the ontology's entities as nodes and relations as edges. The goal is for the visualization to be used for both exploration of the ontology and constructing new views. As opposed to other ontology visualizations that try to visualize all entities or all relations (if not filtered), the approach taken here is a compromise—some entities and one or two relations. In some cases, these other applications may have specific reasons for their choices, such as wanting to reveal to the viewer the overarching topology of a complex network. There are two reasons for the compromise, and both address the needs for the user. The first is that the user wants to see a network with a potential depth of three or more (showing just one or two deep often is not interesting and can be accomplished equally well as a hyperlinked list), and it is important for the user to see the names of all the nodes at once. The second reason is that the user cannot comprehend more than a few relations at once, not even counting the extreme clutter that all those nodes and edges would cause. The clutter of nodes is exacerbated by the fact that for an ontology, unlike a general network, the labels have great significance and should be easily readable, which means that the labels occupy much space.

The interaction was designed with several principles to improve the usability and avoid any confusion.

smooth transitions: Smooth transitions help maintain the cognitive connection between the old view and the new view.

zoomable: A zoomable interface offers flexibility for having a big layout with small

labels that get larger with zooming, or to having a node produce details with zooming.

speed: The software needs to be fast to be interactive. Queries should be kept to a minimum, and information should be cached.

lazy evaluation: The whole reference ontology cannot necessarily exist in memory at once, and there should not be a need to read all of it. Visible nodes and edges should be cached and swapped as needed.

visibility vs existence: It should be clear when nodes and edges are invisible on-screen versus if they exist in the ontology.

reversibility: Any change to the state of the visualization or modification to the ontology should be undoable in succession.

changed vs original: The visualization should have the ability to display what nodes and edges are part of the original reference ontology and what are modified/new/missing.

memory of state: When the user collapses a subtree and re-expands it, or switches from one primary relation to another and back, the older layout should be replicated. Furthermore, the user should be able to save the state of the system (as a workspace), close the program, and start again the following day as if the closing and restarting had not happened.

6.1 VisualLayer

The VisualLayer stores visual data about the entities and relations and acts as the go-between for the DataLayer and any interface. Many of the design principles for the DataLayer apply, but they may be even stronger for the VisualLayer. Notably, more

```
public int getInt(NodeItem ni, Relation r) {
    HashMap<NodeItem,Integer> h=masks.get(r);
    if (h==null) return 0;
    Integer i=h.get(ni);
    if (i==null) return 0;
    return i;
}

public void setInt(NodeItem ni, Relation r, int val) {
    HashMap<NodeItem,Integer> h=masks.get(r);
    if (h==null) masks.put(r,h=new HashMap());
    h.put(ni,val);
}
```

Figure 6.1: Code that lazily builds the visual data as needed.

entities and relations than are stored in the DataLayer's cache are needed for the visualization, and the visualization has even tighter time and space constraints. For each entity, extra visual information includes whether the node is visible, has been touched by the mouse, and is tagged a certain color, as well as temporary variables needed for the layout. Even more space-consuming is for each combination of entity and relation, whether all of its edges have been loaded, if it is currently expanded or collapsed, if there are deleted edges being hidden, what subrelations are visible, etc. All of this information is stored in bitfields for compactness, but still, ten thousand entities times a hundred relations is a bit much. The solution is that the data structure is only created as needed, and the information stored is cleverly chosen such that 0 is the default response. Figure 6.1 shows an excerpt of code that lazily creates the data structure upon a setInt and returns 0 for a getInt if that part of the data structure does not yet exist.

Another function of the VisualLayer is to decide what edges to load. Certainly if the user asks to expand all the outgoing edges for a relation on an entity, all those edges will be expanded. But suppose the user switches to a different relation.

What edges should be loaded? Logically, the edges connecting nodes that have been previously touched should be expanded. Because there are so many different relations and a user may end up looking at only a handful of them, loading these edges happens on an as-needed basis. The answer is that no new nodes are loaded, but any existing edges that have not been loaded into the VisualLayer are loaded. To perform this process efficiently, nodes are tagged as clean or dirty, per relationship. By default, nodes are dirty in all relationships. A node becomes clean for a relationship when all of its incoming and outgoing edges have been explored[1], or the method to search for unloaded edges has processed it. The method searches the dirty and skips the clean ones.

6.2 Navigation

The ontology browser allows a user to interact quickly and easily with a large ontology. In addition to being a browser, the program also supports the construction of a view of an ontology – adding a layer on top of the ontology that gives the perception that the ontology has been modified. Modifications include adding and deleting terms (classes) and redefining the relationships between terms (by adding and deleting individual links). All operations are available by right-clicking on a node and choosing an option from a popup menu. Operations that modify are all contained under a submenu `modify`. Common navigation operations are also available via single-clicks or double-clicks with the left mouse button. Each entity in the ontology is a node with its name inside, and edges represent relations. Additionally, the number of visible edges around the node along with the total edges around the node can be displayed as part of the name, and the name can be abbreviated to conserve space on the display. The user can hover over a node or an edge to get its full name or meaning. Normally only

[1]For relations that are constrained to for trees, it suffices to check consistently either incoming or outgoing edges, and for a relation such as subclass, checking just the incoming edges is far more efficient.

one type of relation can be seen at once. The current relation is printed in the top left. An arrow in the middle of the edge indicates the edge's direction. A bidirectional edge (a symmetric relation) has two arrows. The layout is a radial tree layout based on Yee et al.[53], with extra code to handle cross edges and back edges that do not arise in a pure tree.

Initially just a single node is visible. To see more nodes, data must be loaded from the backend, which the user does by expanding the node. This data is not loaded automatically, as it is in other visualizations, for two reasons. The first is that it may be slow to load the data, and second, the user may not want to see all the other data, as a relevant subset may already be loaded. Figure 6.2 shows a sequence of operations while navigating the ontology. The first three operations are expanding the related entities for the selected relation, in this case, part. The nodes appear in a tree rooted at the original node. Because it is important to distinguish whether the visible neighbors of a node are actually all the neighbors (for the primary relation) or not, a solid border denotes that all neighbors of the node for the current relation have been loaded, whereas a dashed border denotes that there are still more neighbors. For example, in the figure, the stomatognathic system has no neighbors, because there are no visible neighbors and the border is solid. In addition, the edge counts (visible/total) inform the user if all the edges are displayed.

Shown at the bottom of the figure, the fourth operation changes the root of the tree. Whereas other visualizations may show the tree of just parts descending from the root, the ontology viewer also shows the inverse (part of) branching from the root as another tree. These two trees are disjoint aside from the root, that is, there cannot be any edges between the regular tree and the inverse tree. Choosing a new root for the tree can be used both for seeing the relationship and inverse relationship of the new root and for hiding cousin nodes that are irrelevant. If the user does not need the context of the inverse tree, it can be disabled via a checkbox in the menu. For convenience, setting the root can also be performed via a left double-click.

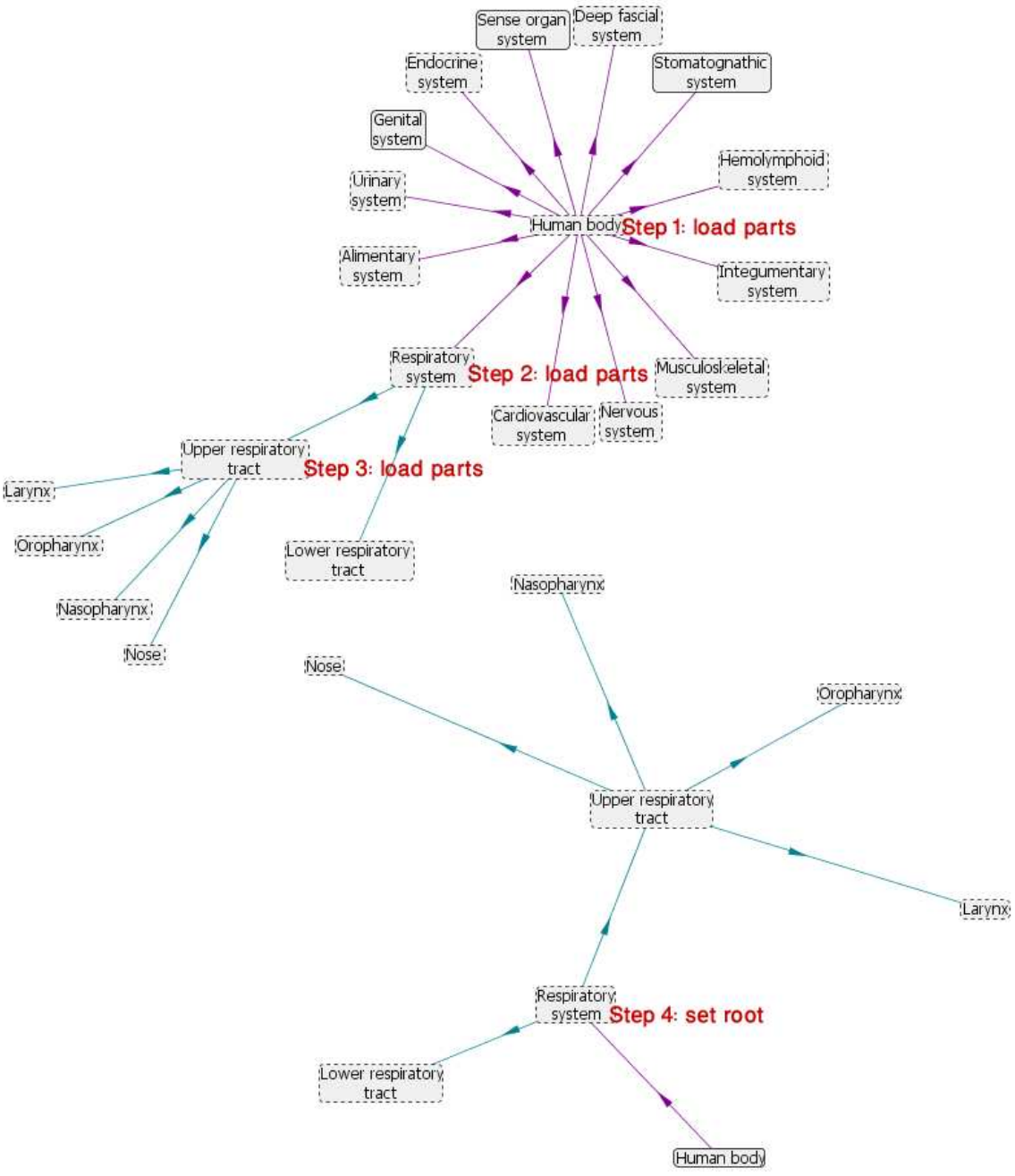

Figure 6.2: The top diagram shows a part hierarchy revealed from three consecutive expansions by the user. Then the user changed the root of the layout, creating the view in the bottom.

Once the user has expanded several nodes and decides there are too many visible on the screen, the user can collapse a subtree of nodes. The operation is available on the right-click menu; additionally a single left click performs an expand or collapse that does not load new data from the back end. Figure 6.3 shows a collapse operation. The top figure shows the view before the collapse, and in the bottom figure, the larynx is collapsed. The blue outline around the larynx denotes that it has been collapsed, rather than that it has no neighbors. Both changing the root and collapsing nodes work well for hiding irrelevant nodes. Figure 6.4 shows a cluttered view of bone subclasses, whereas figure 6.5 has long bones set as root, and figure 6.6 has long bones and flat bones collapsed.

Another navigational tool is to switch the relation being shown, making the clicked node the new root. As mentioned for the VisualLayer, relevant edges are loaded from the back end, but no new nodes are loaded. This context is extremely useful and can be considered an approximation of degree-of-interest. For example, suppose the user loads some nodes in the part relationship as in figure 6.7 but then wants to change the viewed relationship to subclass. The browser produces the subclass view seen in figure 6.8. If all siblings of the visible nodes were displayed, the resulting view would be largely irrelevant; in figure 6.9, only the yellow nodes are the relevant ones (the nodes visible in figure 6.8). When a user changes back to a previously visited relation, the state (e.g. which nodes were expanded/collapsed) is remembered.

6.2.1 Secondary relationships

Sometimes it can be useful to see more than one relationship at a time to understand a more complex situation. Other visualizations display more than one relation at a time by default and rely on the user to filter the relations. The approach taken here is different, in that by default, just one relation (and perhaps its inverse) is shown, and the user can opt to show one or two secondary relationships. The secondary relationships are only shown one level deep beyond the primary hierarchy, so they

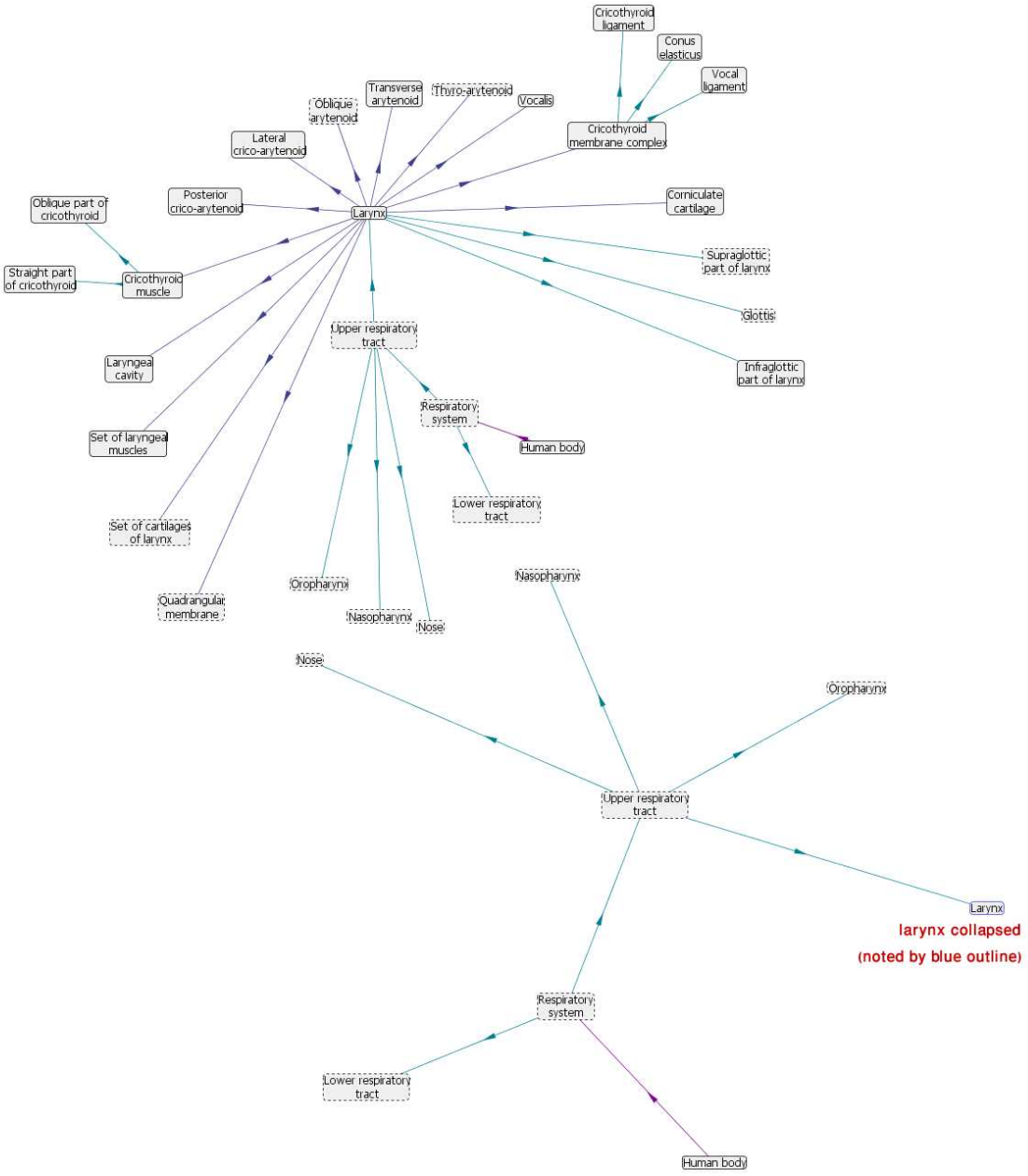

Figure 6.3: The top figure shows the layout before the collapse operation, and the bottom figure shows the layout afterward.

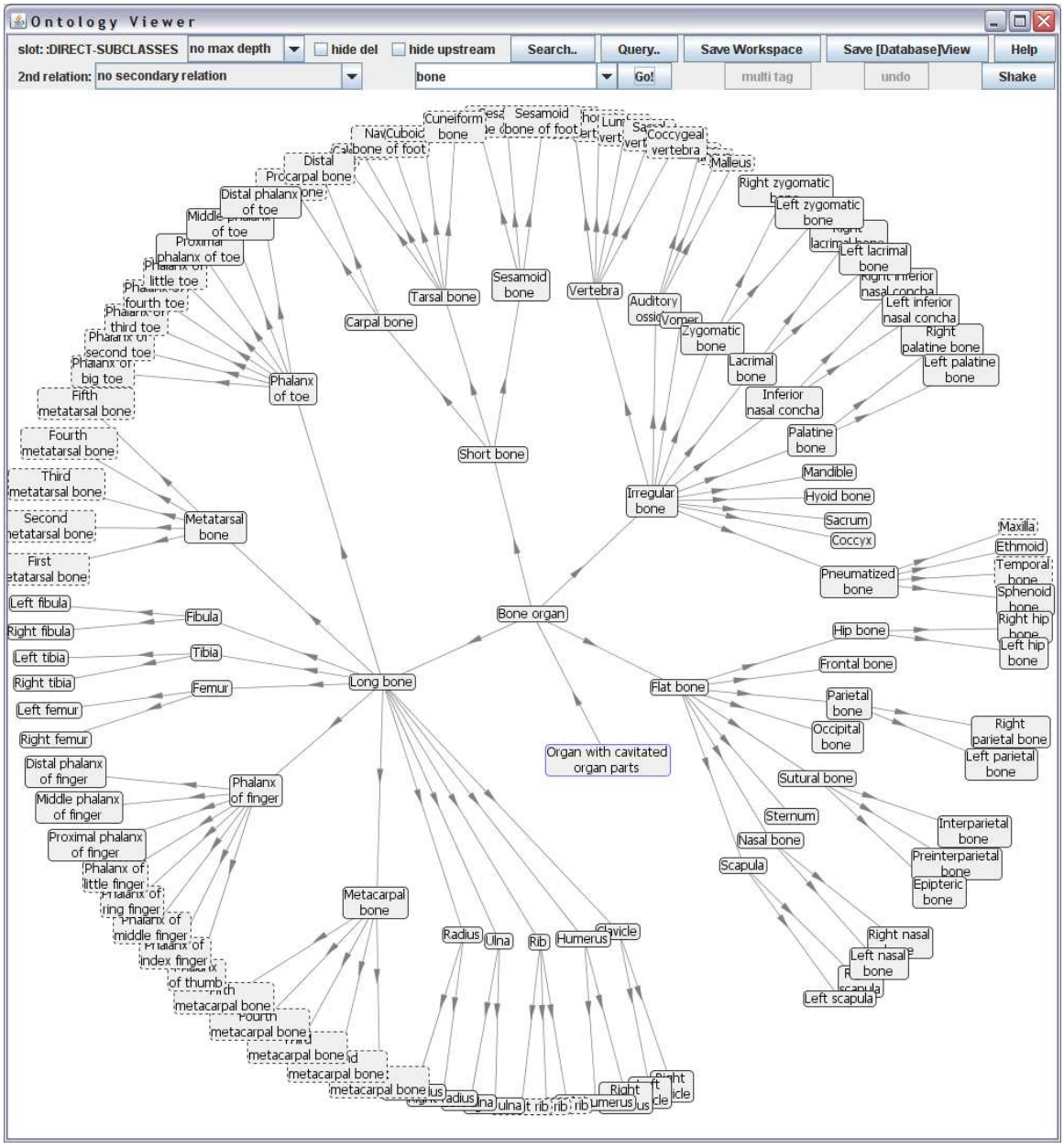

Figure 6.4: A view of the taxonomy of human bones. The view is cluttered, and labels are overlapping.

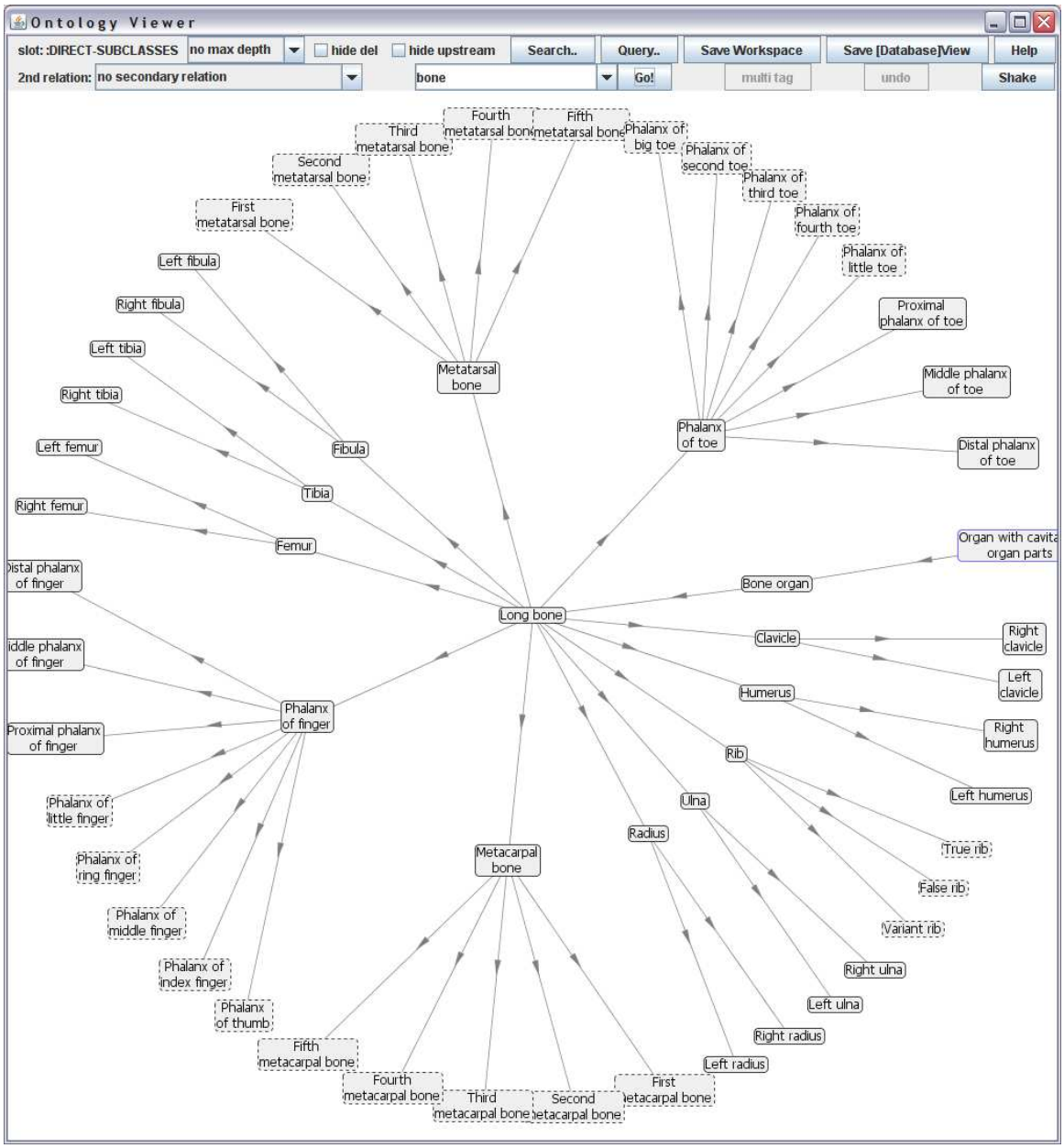

Figure 6.5: The user can set the root to just "Long bone" to study that information with less clutter. The root can be set back to "Bone" later.

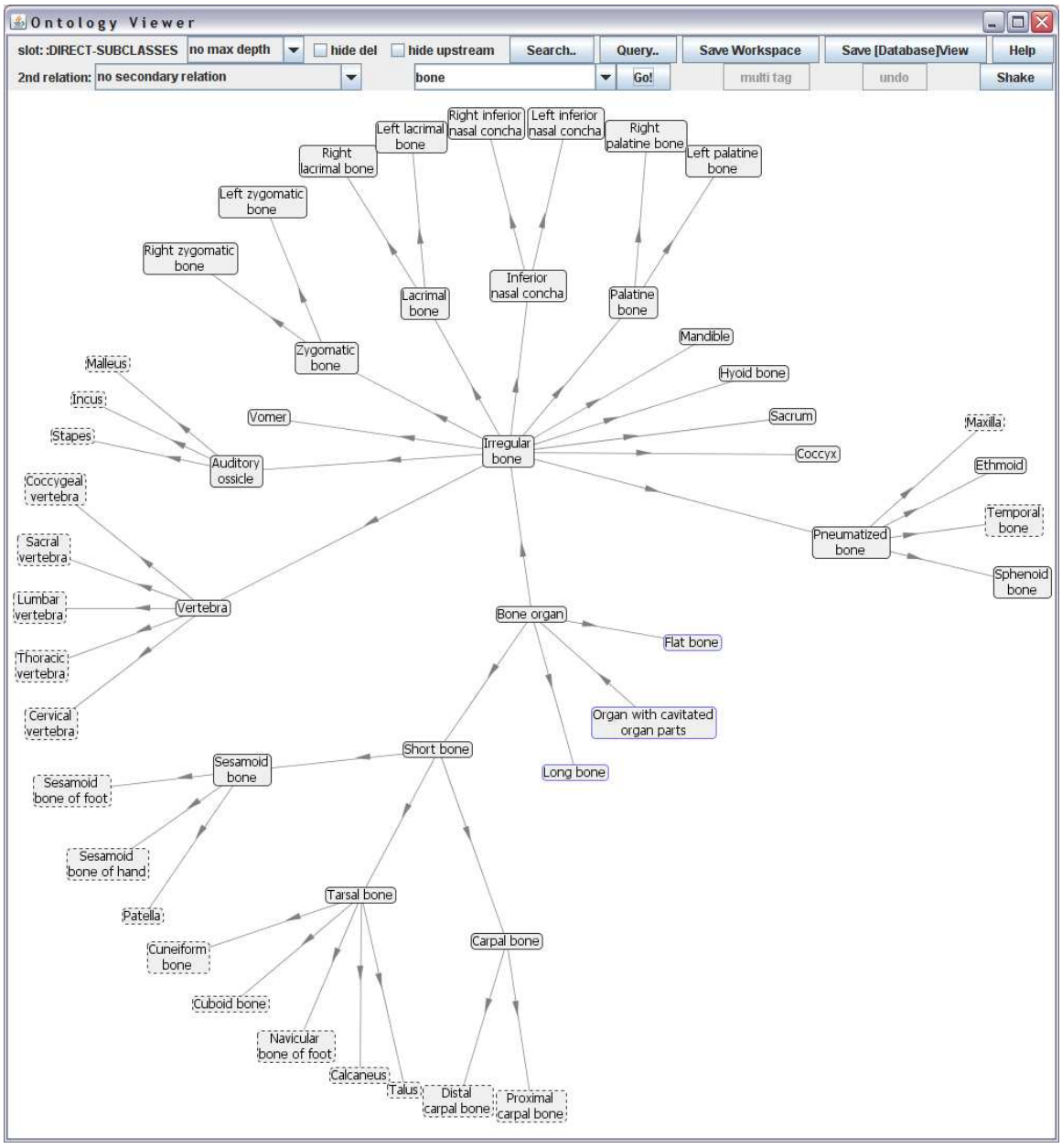

Figure 6.6: Another way to reduce clutter is to collapse other subtrees. Here the short and irregular bones can be easily seen because the long bones and flat bones have been collapsed.

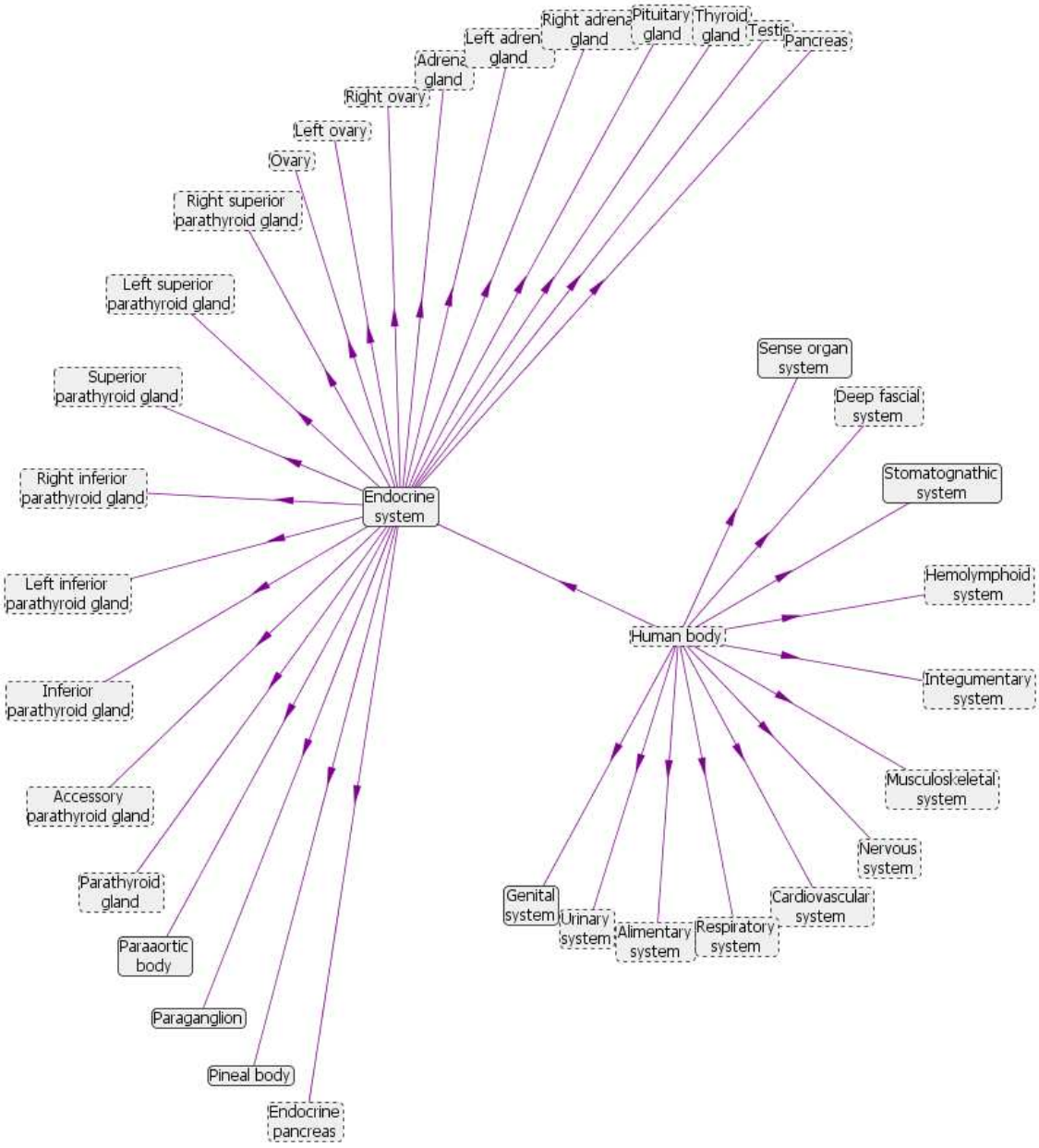

Figure 6.7: View with several parts expanded.

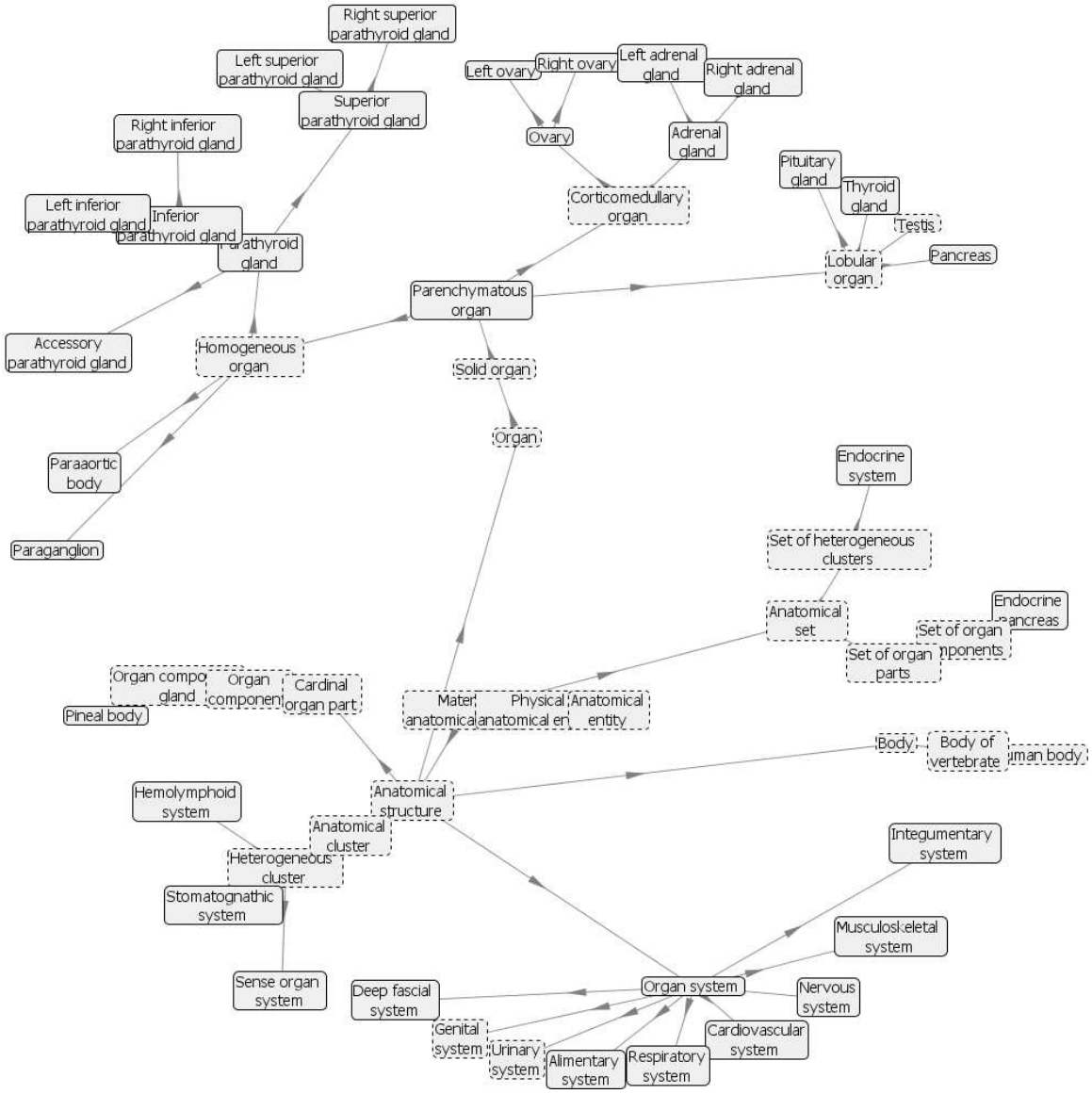

Figure 6.8: View switched from part to subclass. The nodes shown are from the context of the previous views.

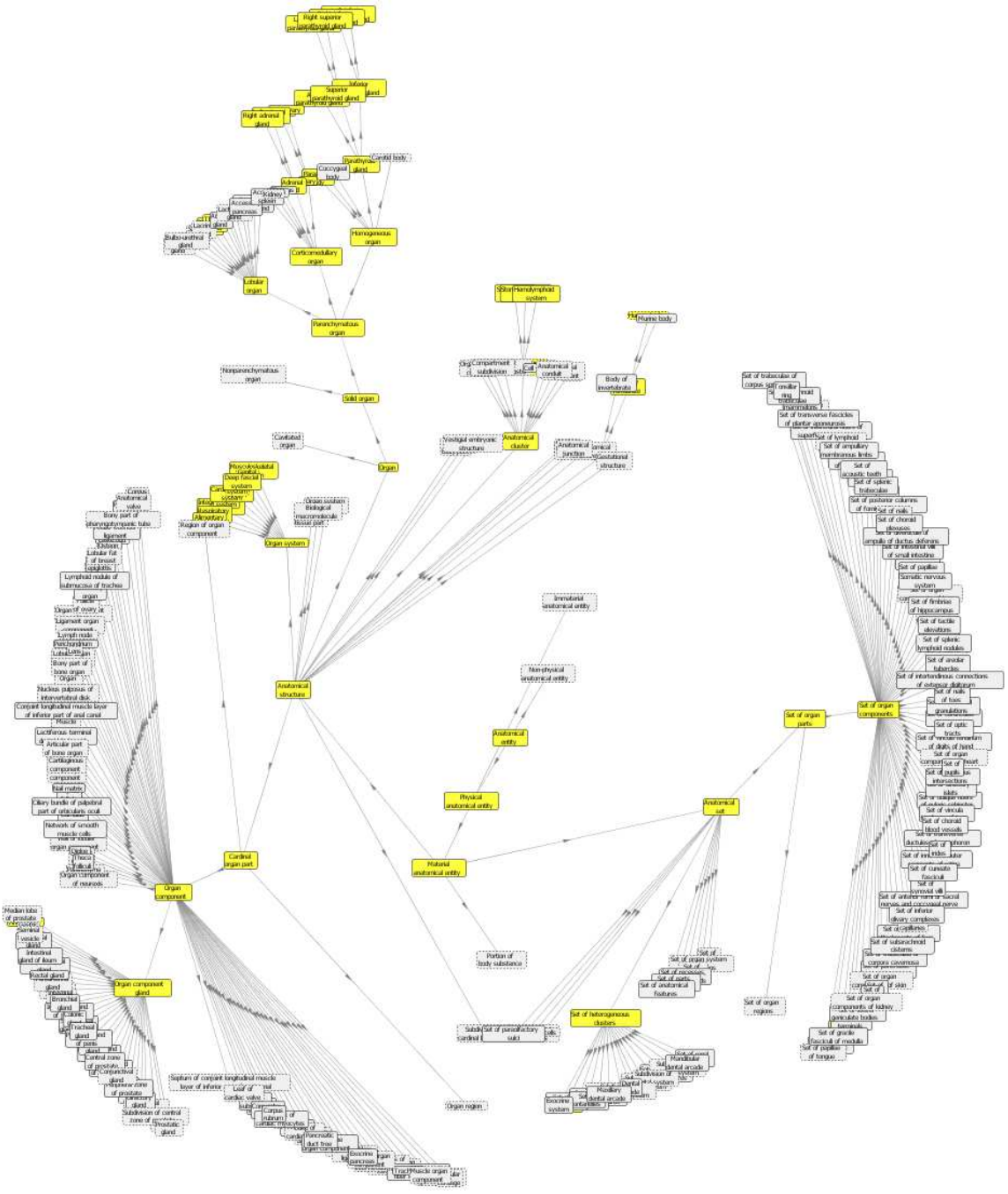

Figure 6.9: If the subclass hierarchy were expanded such that all the nodes in the original part view were visible, it would look like this. The nodes displayed in the relevant view (previous figure) are highlighted in yellow.

deliver context without clutter. Figure 6.10 and figure 6.11 show two examples of secondary relationships. The first primarily shows the branching of the coronary arteries and secondarily shows the regions they supply. The second primarily shows the partitions of the heart and secondarily shows the arteries that supply them. Having more than one secondary relationship is useful for lymphatic chains, where one might be concerned about lymphatic drainage, afference, and tributaries.

6.2.2 Other non-modifying interaction

Several other features are available that have not yet been discussed. If there are too many nodes on the screen, the user can artificially restrict the maximum depth to a small number.

The browser supports differentiating between subrelations for a given relation, e.g. regional part and constitutional part, which are types of part. Subtypes of edges are color-coded differently, and in addition, the user can elect to load just one subtype or restrict the expansion of neighbors to just one subtype. Because the primary relation is the superrelation, a node with all regional parts shown may still have a dashed border, because other types of parts are not being shown,

The user can view the details of a node, which shows a text document of the entity's attributes and relations, all hyperlinked. A work in progress is to toggle between the document showing the original ontology and the current transformations, as well as allowing editing through the document. Having several parallel ways of accomplishing the same task is a blessing to the users, so they can prefer what they are more familiar/comfortable with, or they can use the appropriate tool for their own use, of which I am not yet aware. A potential future item is the slick zoomable interface in Jambalaya, where instead of the document appearing in a new window, it appears by zooming into the node.

Finally nodes can be tagged or untagged one of ten different colors, bright colors roughly spanning the hues of a rainbow. The color choice was designed so that no

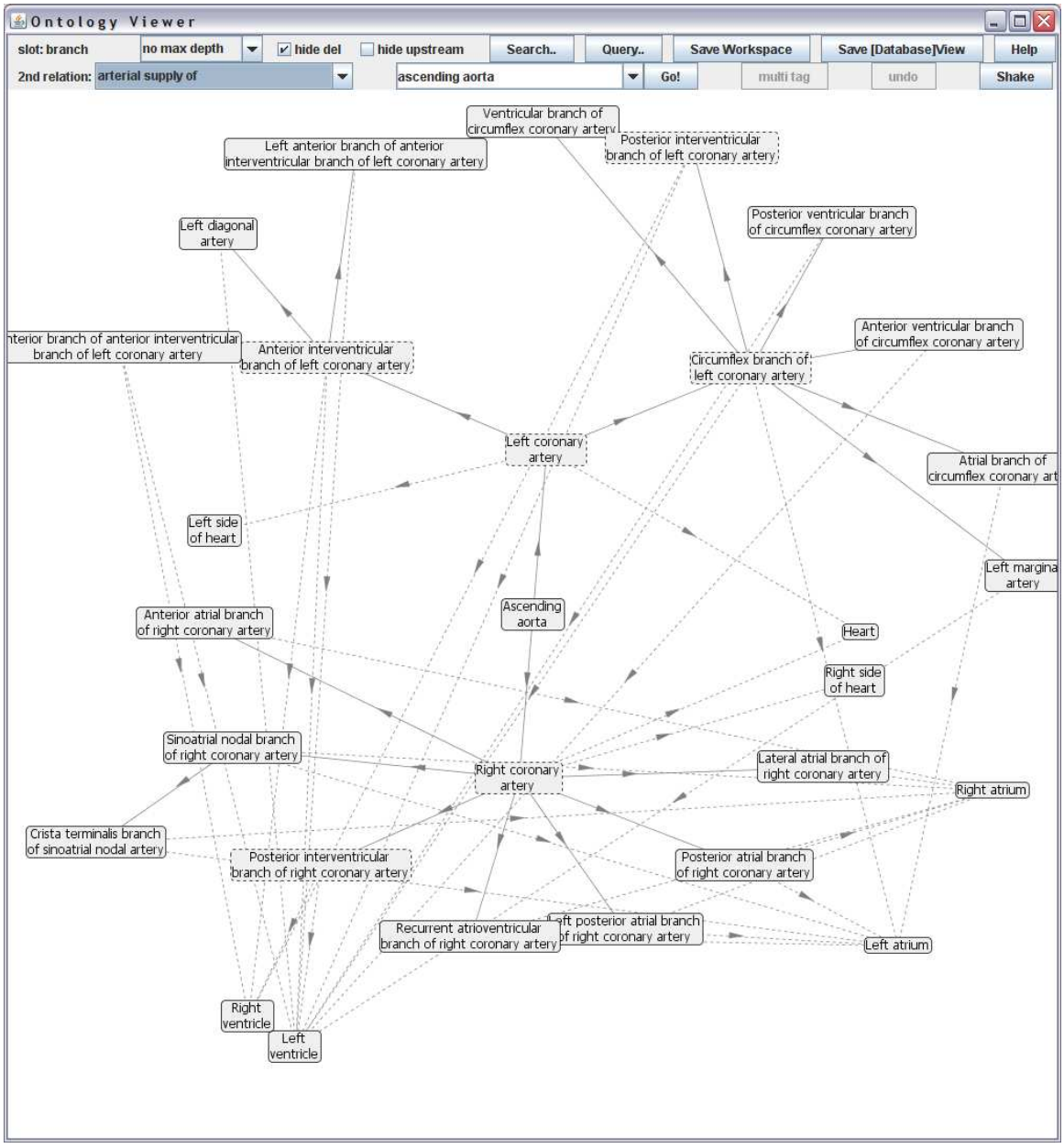

Figure 6.10: Branches of coronary arteries as the primary relationship and arterial supply of as the secondary relationship.

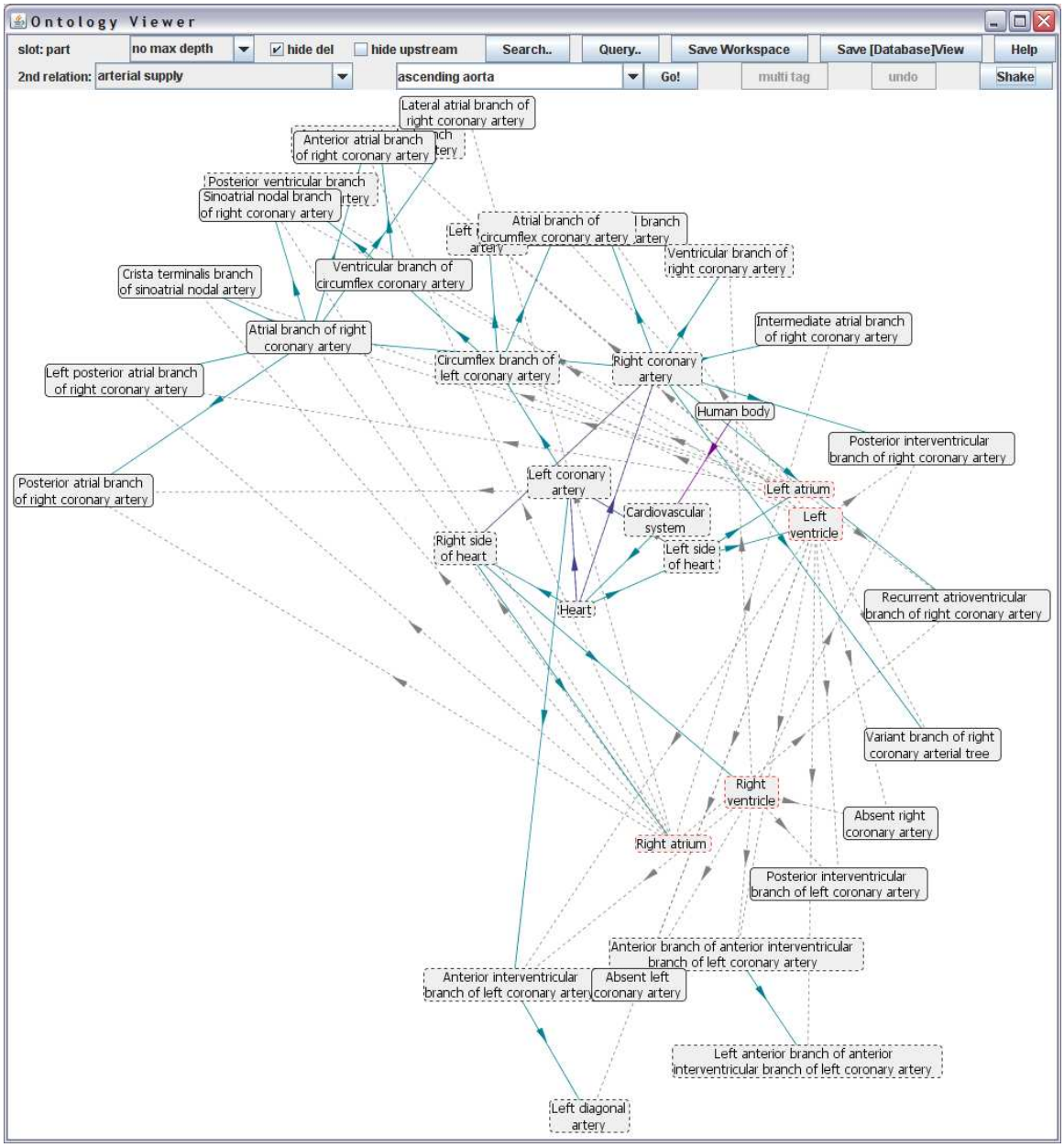

Figure 6.11: Parts of the heart as the primary relationship and arterial supply as the secondary relationship.

two colors are likely to be confused with each other or with the otherwise used colors (pale green and pale red). The purpose of tagging is to provide a visual reminder for the user or to assign some additional meaning to a set of nodes, with the cue being the color. Tagging is additionally supported in the search and query interfaces, to be described in later sections. A later section will also discuss advanced applications of tagging for a specific use case. A work in progress is to provide a means for the user to annotate what the tags mean, for their own reference and for others who may use the workspace later.

A fundamental design decision was that nodes can be tagged multiple colors, which provides much more power to the user, as the user can then operate on intersections, unions, or subtractions on sets of tag colors. The difficulty though is how to display the multiple colors assigned to a given node. Three ideas include having multi-tagged slowly animate their colors in order, having multi-tagged nodes display colored stripes showing all their different tags, and having a button that cycles through the tagged colors (in essence, animation on demand). The first and last, which were the easiest to implement, were tried, and the last was chosen because the first was too distracting. One problem however is how to notify the user if there are indeed multi-tagged nodes on the screen. The first attempt was to have a button that is by default disabled and is only enabled if there are such nodes present. However it was too subtle. The strategy now is to notify the user via a dialog if the current view has multiple tags and the previous doesn't, and then the user could press the same button to cycle between tags. This tactic should provide the needed reminder without being excessively annoying.

6.3 Modification

The user can perform a suite of modifications accessed through the "modify" submenu of the right-click popup menu. To differentiate modifications from the original ontology, new nodes (entities) are shaded pale green, deleted nodes are shaded pale red, new edges are colored green, and deleted edges are colored red. A user may

not want to see deleted nodes (or nodes connected by deleted edges), and if there are many of them, they can clutter the view. A checkbox toggles the deletions to be hidden, which cleans the view. However it is desirable to know if an item is not visible, because an edge to it does not exist or because an edge to it was deleted. A red border (color of the border is orthogonal to its line style) indicates that it has neighbors or incident edges that have been deleted and hidden. Figure 6.12 illustrates all of these cases, except the case of a new node.

A user adds or deletes an edge by first initiating the add or delete by selecting it from the modify submenu from right-clicking on the edge's source node. As the user drags the mouse to the target node, a red or green arrow is drawn from the source to the mouse cursor. Clicking on the target node confirms the operation. The original version had no feedback other than the new edge appearing, but after watching users add edges the wrong way, The current version has a dialog that frames the meaning of the new edge as a sentence, e.g. "heart has part left atrium" and asks the user for confirmation. A difficulty is that the slots in an ontology have names such as "part" and "part of" and to a novice user, "A part B" does not have a clear meaning. The heuristic used is that a slot name that ends in a preposition (e.g. is, of, with, by) or appears to be a verb should be prefixed with "is", and the other slot names should be prefixed with "has."

A practical use of deleting edges is to clean up an unnecessarily dense partition that can be easily resolved by transitivity. Figure 6.13 shows the clutter resulting from the superfluous edges. This redundancy would be difficult to discover in a non-graphical view. The user can select the redundant edges to delete (figure 6.14) and then when the "hide deletions" checkbox is enabled, the graph looks much cleaner (figure 6.15). Again note the red border, which means that there are deletions that are not shown. When the workspace is saved into a view, the view will not contain any sign of those edges.

Deleting and excepting deletions on entities (and their subtrees) follows the rules

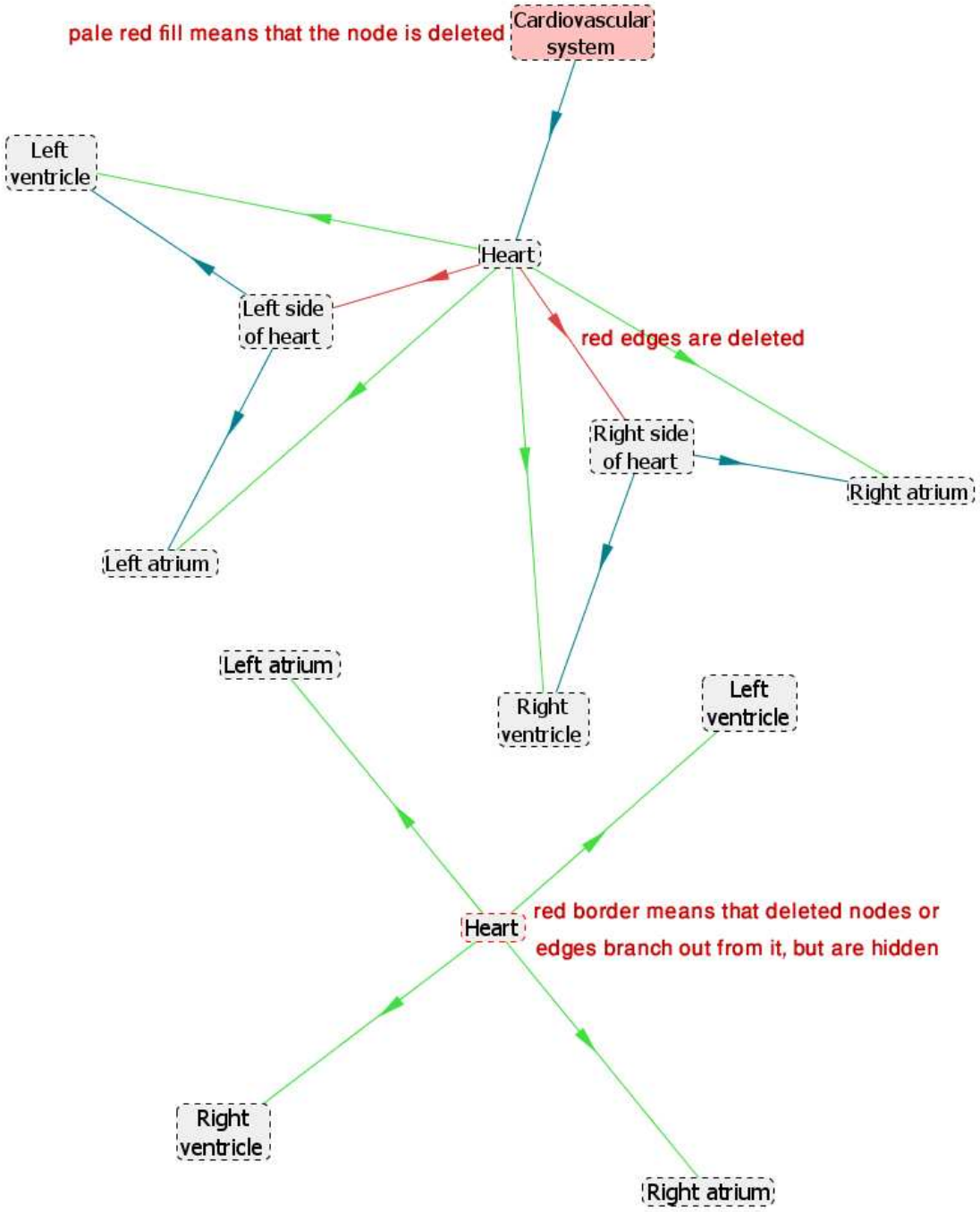

Figure 6.12: This view of parts of the heart shows deleted nodes, deleted edges, and new edges, as well as what the view looks like when deletions are hidden.

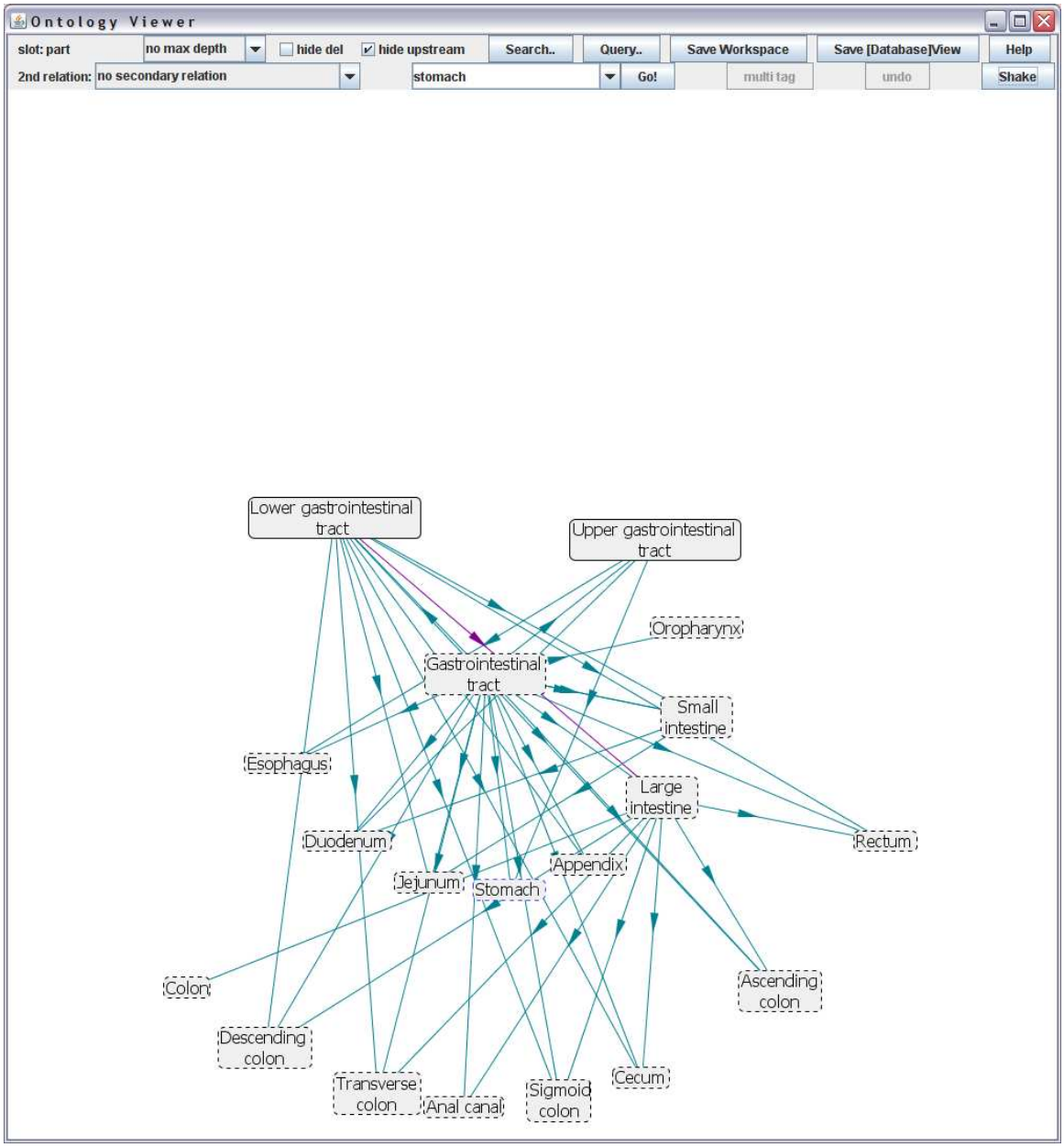

Figure 6.13: The ontology contains a redundant partonomy that could be inferred via transitivity. A user might want to cull the redundant edges.

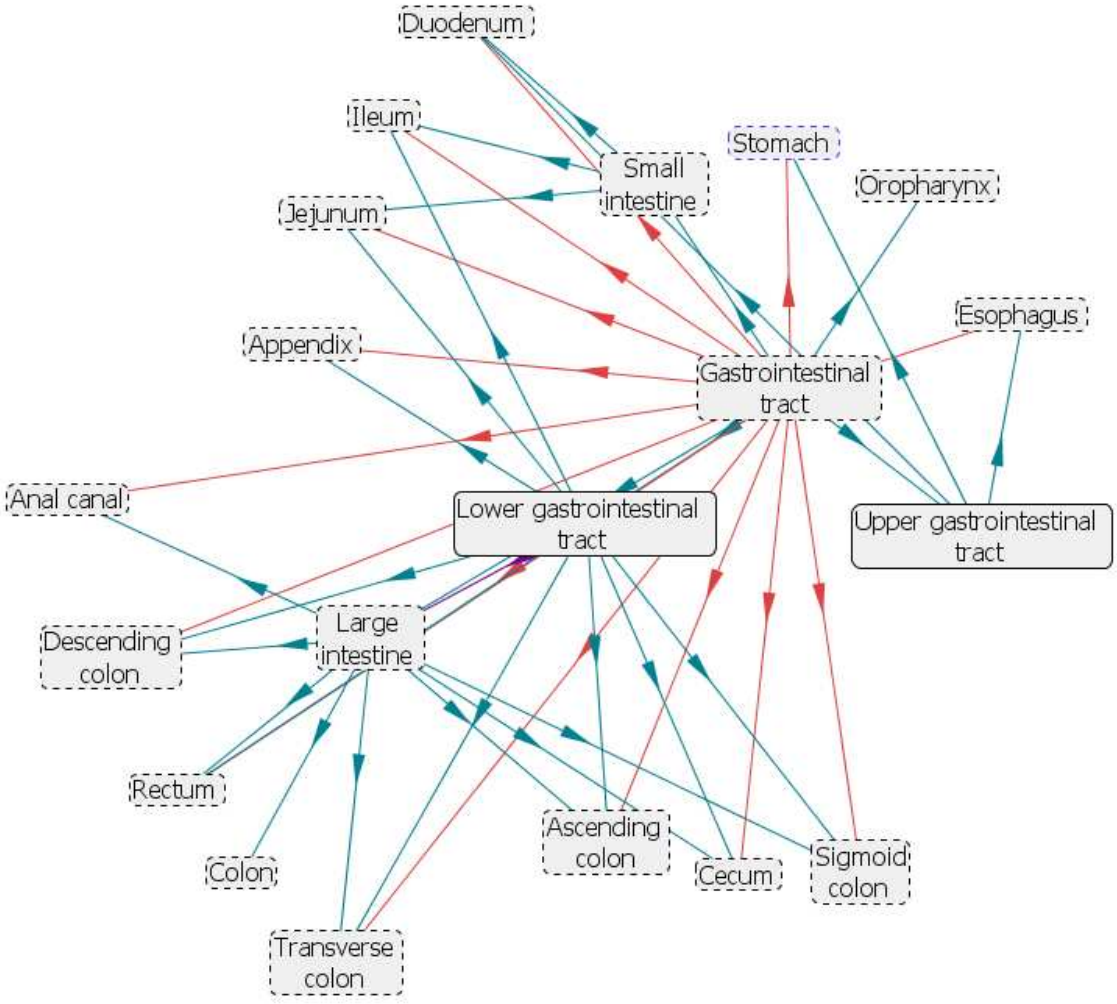

Figure 6.14: One by one, the user deleted the redundant edges, which appear red.

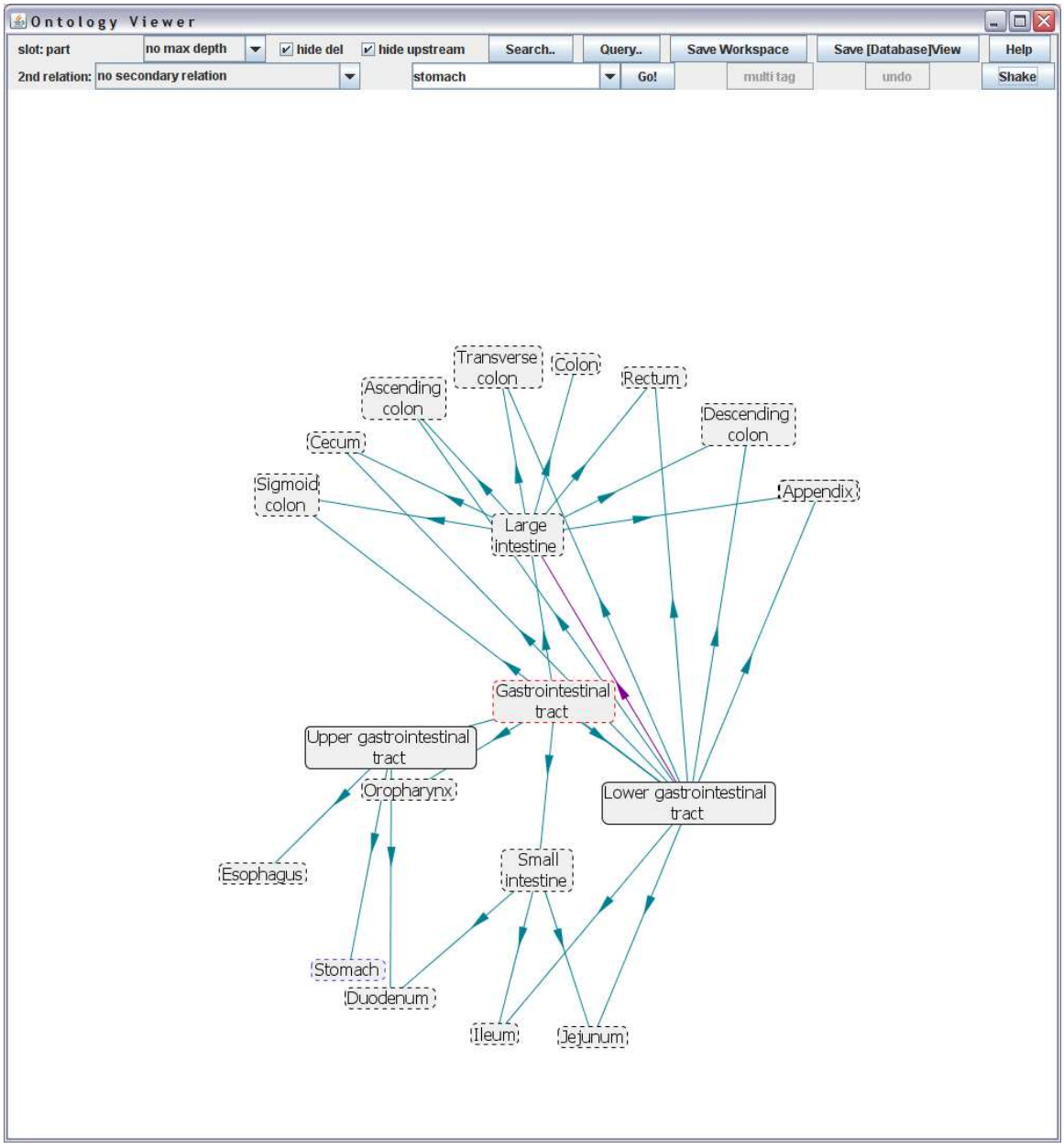

Figure 6.15: Finally, the redundant edges can be hidden from view, with a red border around a node reminding the user that it has outoing edges that were deleted.

described in the previous chapter. So that changing a parent is an atomic operation, rather than an edge addition and an edge deletion, the standard edge additions and deletions are not available in the sub/superclass views, and the change-parents operation is only available in those views.

When adding a new entity, the user needs to specify the parent of the new node. The user does this by initiating the new-entity operation from the to-be-parent node in the sub/superclass views, or from a to-be-sibling node in any other view. In one of these other views, the new node will not be visible because it has no other relationships, so to establish the cognitive connection with the new node, the view is changed so that the primary relation is subclass. The node inherits all the slots of the parent, but not necessarily its values. It only inherits the values of slots from the parent if the grandparent shares those same values. This heuristic infers which properties are inherited (e.g. if the entity has a mass) or are overridden (e.g. the entity's name). When the user wants to add relations to the new node, the node will not be visible in views of relations other than sub/superclass, and hence, the user cannot click on both nodes to add an edge between them. Instead a feature called "connect to unseen" exists that lets the user connect the node to one of a set of recently touched nodes.

The top part of figure 6.16 shows several new nodes, as well as deleted subtrees. When the transformations are saved as a view, the appearance of the view is as if it were an untouched ontology of its own, as is seen in the bottom of the figure. With just a few modifications, the view of an ontology can be changed dramatically.

6.4 Query

The query interface provides a means of making powerful queries, though it is not intended for the novice. Computations can be with unary or binary operators (one or two arguments); the results are returned to an output list. The output list can be named and stored or piped back to input. The various inputs to the computations

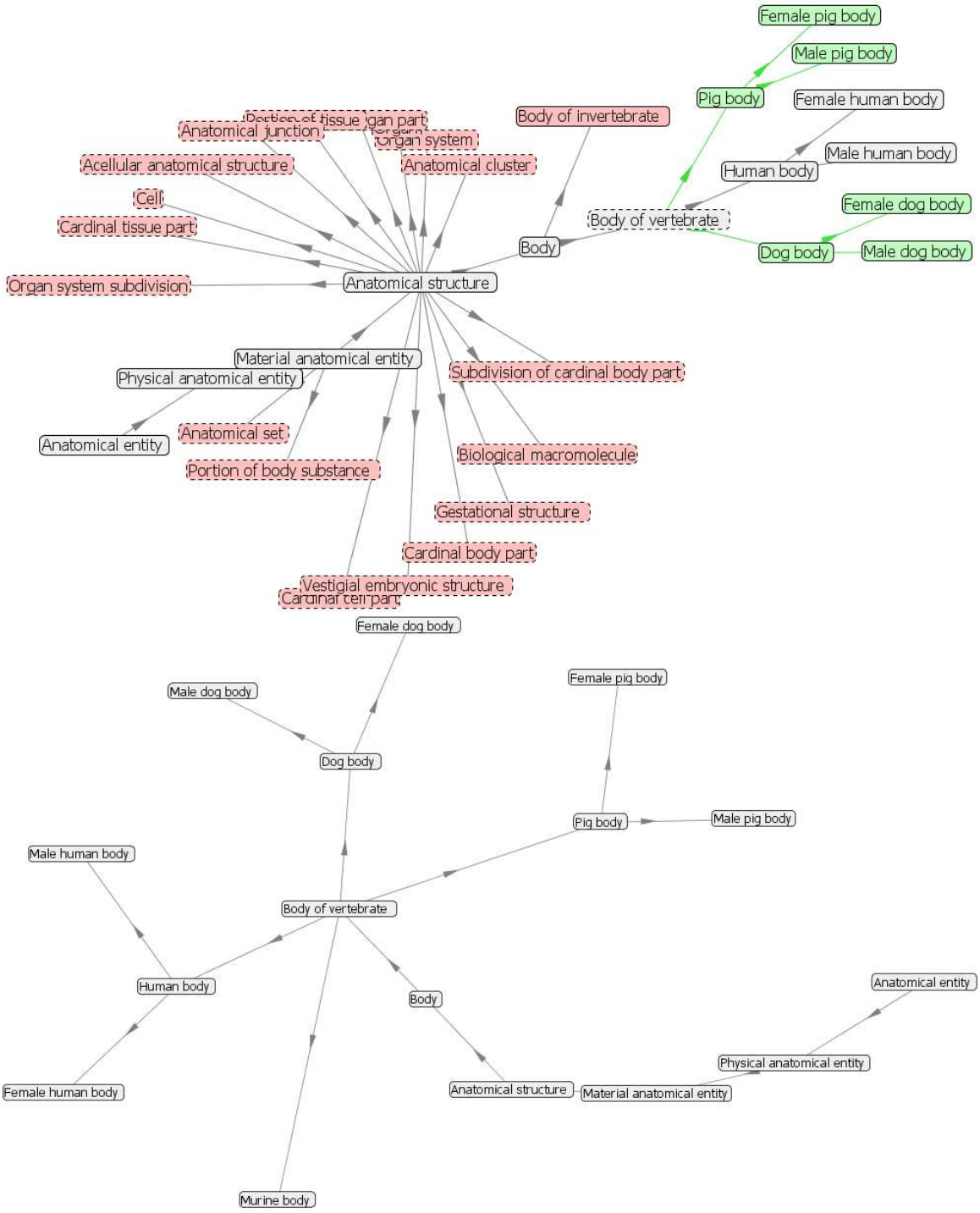

Figure 6.16: This view was created by deleting the root, undeleting "Body of vertebrate," and adding some new nodes (entities).

include all the loaded nodes, nodes that are visible, nodes that have been touched, nodes that have been recently touched, the loaded nodes that are deleted, nodes that are tagged a certain color, the output list, and stored lists. In addition, the results list as a batch can modify the visualization. The interface can be used for several general tasks, such as performing operations that would be too mundane or time-consuming to do manually, such as tagging all visible nodes, or for building complex queries, such as finding entities that have no subclasses and are not part of anything. Figure 6.17 details the computations supported, and figure 6.18 details the actions that can be performed. Figure 6.19 displays two screenshots of the query interface. The first two columns show the inputs to the operations; only one is used for an unary operation. Some of the populations or actions require an additional argument, for example what color to tag the selected nodes, or the name of a saved list.

Inputs can be chosen from the sources mentioned previously. Though an input list can be partially selected, the user will usually want everything selected. For convenience, if nothing is selected, the user will be prompted yes/no upon computation to determine if he or she would like everything to be selected. The source for an input is specified by one or two combo boxes, and the input list is filled using the current data by pressing "Populate." Note that the list will not change dynamically (e.g. a list of visible nodes)—the user needs to press "Populate" again to repopulate the list with current state. One reason for this design choice, as opposed to repopulating on selecting a new choice, is that a JComboBox does not trigger an ActionEvent when selecting the same item—the user must select something else and then select back.

Ideally a powerful query interface could be based on a visual flowchart or natural language, such as those investigated by Bernstein's lab.[28] However, some of the use cases require powerful queries, and this implementation serves as a placeholder for both the users and the researchers studying the users or the queries generated. Also, there exists a tradeoff with what set of computations to reveal to the user. Certainly, if the programmer had unlimited time, any possible operation could be

Computations

Identity: This operator simply sends the input to the output so that actions can be performed on it.

Relations: For each selected item in the input, the outputs are the directly related terms or the transitively related terms, via the specified relation. A checkbox allows the user to say whether to use the original ontology or to use the modifications performed thus far.

Predicates: A variety of predicates exist to evaluate states of the view layer modifying the ontology or the visualization. Some of these predicates check if terms are new or deleted, if nodes are visible or tagged, if a node has been touched by the mouse, etc. A checkbox allows the user to negate the predicate. The computation returns the subset of inputs that satisfies the predicate.

Relation Predicates: Some predicates take a relation as an additional parameter, such as if a node has any outNeighbors (any values for the slot) of the relation. Appropriately, there are checkboxes for whether to use the original ontology or modifications, and whether to negate the predicate. For example, if one wants to filter a list to include just those nodes that have no subclasses (are leaves), one would invoke the relation predicate "has relateds" with relation ":DIRECT-SUBCLASSES" and negate the predicate.

Set Arithmetic: Binary set operators: given two input lists, one can compute their union (in one or the other), their intersection (in both), and their difference (in the first but not the second).

Figure 6.17: Computations supported by the Query interface.

Actions

None: Does nothing. Useful to get the count of the number of items in the result list.

Clear: Removes the selected set from the result list.

Send/Store: *Send:* Copies the selected items of the result list to Source 1 or Source 2. *Store:* Stores the selected items of the result list under a user-specified name. Can be accessed later as a source.

Set root: Sets the root of the visualization to the first selected item.

Details: Displays details for the first selected item.

Load: Loads the selected items into the visualization.

Delete: Modifies the ontology so the selected items are deleted (loading them if necessary).

Except/Undelete: Modifies the ontology so the selected items are undeleted (loading them if necessary).

Tag: Tags the selected items by the specified color (loading them if necessary).

Untag: Untags the selected items of the specified color (loading them if necessary).

Touch: Touch the selected items (loading them if necessary).

Figure 6.18: Actions supported by the Query interface.

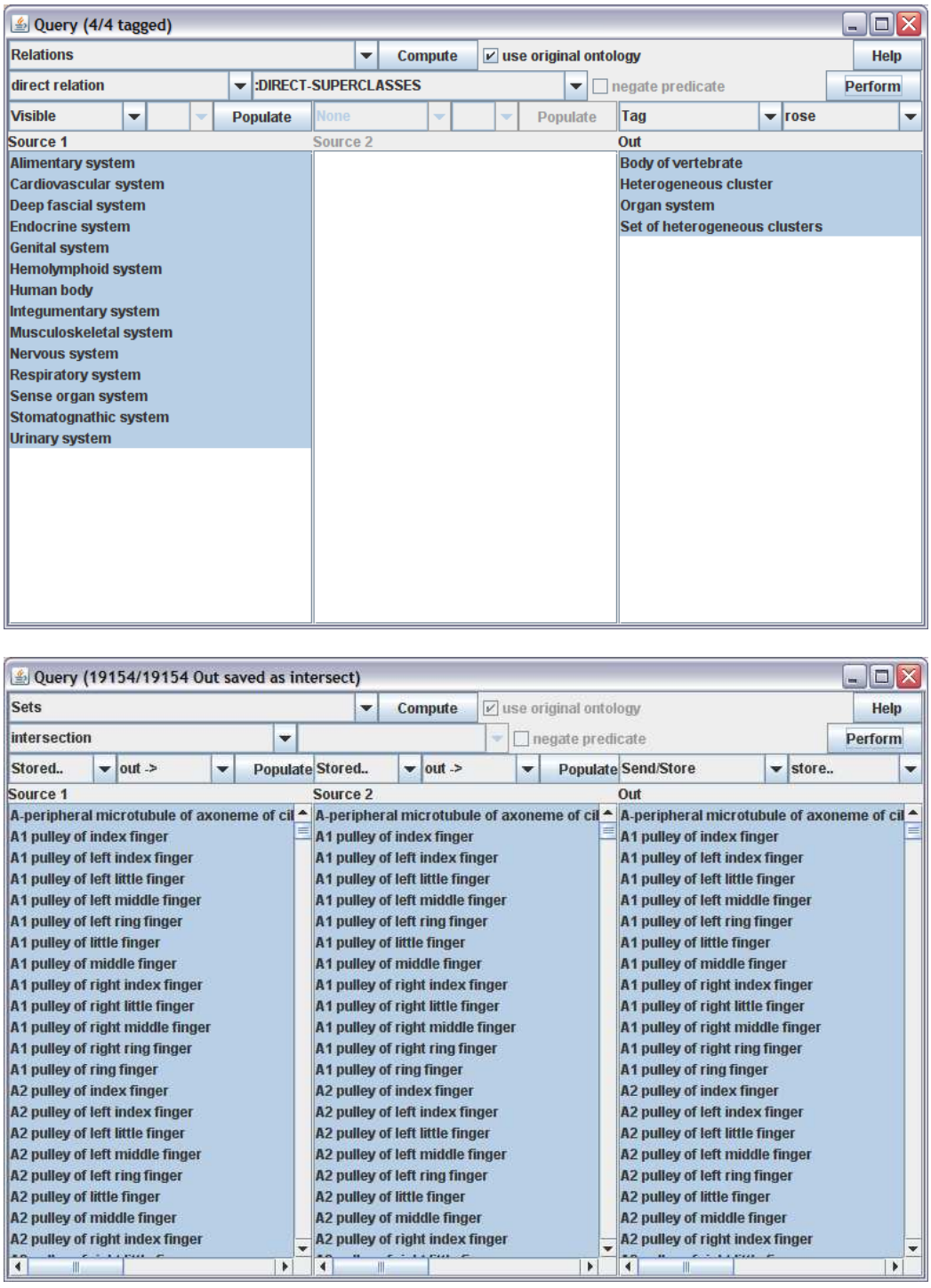

Figure 6.19: (top) a query that computes that superclasses of a set of entities; (bottom) a query that computes the intersection of two sets of entities

precoded, but then the challenge would be how to present it to the user without the user becoming overwhelmed with too many tools. At the same time, a small set of tools may be sufficient to perform any operation through enough compositions, but the logic required to figure out how to chain the computations together is likely too much for the average user. An example is finding nodes that are leaves (have no subclasses). One way to do it is to take the set of all nodes, compute the set of superclasses for this set, and subtract from the original set. However, this logic is counterintuitive and is completely unnecessary with a "has >1 relateds" predicate. A corollary is that there might be multiple ways to express the same computation but one such way is dramatically more efficient to compute. This issue should be the job of a compiler or optimizer that happens behind the scenes, and users should be given the freedom to express the queries in whatever ways they like.

Here are some examples of more complex queries and how they would be performed using the query interface.

- *Load everything that is transitively part of the "Alimentary system."* Search for "Alimentary system" and set as root (or load parts of "Human body"), and populate Source 1 with visible nodes. Click on "Alimentary system", select "Relations" then "Transitive closure" with relation "part", and compute.

- *Find the nodes that are tagged "rose" and are visible, and tag them "yellow."* Populate Source 1 with nodes tagged "rose." Choose the predicate "is visible" and compute. Perform "tag yellow" on the result.

- *Find terms that are material, have no subclasses, and are not part of anything (implying that they are possibly incomplete/dangling).* Populate Source 1 with the loaded nodes and select "Material Anatomical Entity." Compute the transitive closure over ":DIRECT-SUBCLASSES." Send the result back to Source 1. Choose "Relation predicate" then "has relateds" with ":DIRECT-

SUBCLASSES" and select "negate predicate." Compute and send the result back to Source 1. Change the relation to "part of" and compute again.

- *Find terms that are transitively part of both the male and female bodies but not part of the human body (perhaps these terms should be part of the human body).* Search the male and female human bodies or load subclasses of human bodies. Get "Male human body" to appear in Source 1 and select it. Choose "Relations" then "Transitive closure" with "part" and compute. Save the result as "maleOnly." Do the same for the "Female human body" and save as "femaleOnly." Change the operation to "Sets" then "Intersection," set Source 1 to "Stored../femaleOnly" and Source 2 to "Stored../maleOnly," and compute. Store the result as "both." Now select "Human body," compute the transitive closure of part, and send the result to Source 2 (or store and populate later into Source 2). Populate Source 1 with "both." Change the operation to "Sets" then "Difference" and compute.

6.5 Search

Often a user wants to explore a specific entity. Rather than browse the subclass hierarchy, it is much more convenient for the user to type in the name of the entity directly and go to it. Furthermore, the user may not know the exact name of the entity, or despite the impressive completeness of the ontology, the name may not be present. The system supports three modes of search of increasing complexity, intended for different tasks. All three modes use Apache Lucene[2] as a backend. The only modification is that hits with shorter names are given priority, because of the prevalence of compound names in anatomy. For example, a query on "lung" would additionally return, among many others, "left lung" and "lobe of lung."

The first search mode is a search textfield on the main browser window. It has

[2]http://lucene.apache.org

no fancy features and is designed for quick access. The user is not given a choice of hits, which is fine because nothing bad can happen should the hit not be what the user intended. The textfield is augmented with a pulldown menu that has a list of recently visited nodes that the user can select for quick access.

The second search mode is a full-featured search, where multiple hits can be investigated before committing. In an attempt to grant some of the powers of a Lucene query without forcing users to learn the language, the interface gives the options of searching for an exact phrase, searching for words that start with a prefix, and searching for words that sound like the entered word. The hits are culled to show only the most likely. In most cases, this culling keeps the signal-to-noise ratio high, but in a few instances, the desired entity could not be found. A checkbox turns off this culling.

The third search mode is intended for the power-user wanting to do searches on a list of items. Items are categorized either as "found" or "not found." A batch search attempts to match as many as possible. Each matched item is annotated with how it was matched (perfect match, matched a synonym, matched this phrase, etc.) so that the user can evaluate each match. Incorrect matches can be moved from "found" back to "not found." Items that the batch search could not find can be matched manually by the full-featured search previously described. Selections of found entities can be loaded, deleted, excepted, tagged, etc., as can be done through the query actions. The entities also can be annotated so the user can note which had been processed or add any other commentary. The "found" list can be sorted in several ways: the original order of the input, alphabetical by the input terms, alphabetical by the matched entities, how the matches were made (e.g. exact, alternate name, manual, etc.), or alphabetical by the comments. A batch search can be saved and loaded as well.

Figure 6.20 shows a screenshot of the Batch Search in use on the RadLex use case, described in the next section. The left column shows terms that have not been matched to FMA entities, and the right column shows terms that are matched.

The terms in the left column highlighted yellow are ones where the search found a match that the user deemed incorrect, and the user moved the terms from the right column back to the left column. The majority of the terms in the right column were exact matches. The terms highlighted green are examples of terms that did not have an exact match; both the search term and matched term are shown. The terms highlighted red are examples where there was a match with a synonym or other alternate name. The terms highlighted blue are examples of terms that the user manually searched for. The manual query is also stored in the comment. Finally, the user added comments to the nodes denoting that they had been processed.

6.6 RadLex Use Case

The *Rad*iological *Lex*icon came to being while the FMA was still being created. As a result, RadLex is not compatible with the FMA, though it would be desirable to be so. During the testing of the ontology browser, a set of steps was developed to align the lexicon with the FMA. Potentially, the user interface could be tailored to specifically address this use case, but a more generic interface has a wider use and may allow people to discover new ways of manipulating data that developers of an interface could never have envisioned.

One useful, but nonintuitive, technique is to start by deleting the whole FMA and then excepting only the nodes needed (e.g. for RadLex). Then the user can start trying to match terms, as was shown in figure 6.20. The user does not need to match all possible terms; he or she can go back later and match more, using the comments to remember what has already been done. A tactic that works well is to sort the found list by "how found," undelete, tag, and mark the obvious ones, move the obviously wrong ones back to not found, and defer the rest of the terms (less than a quarter) until later.

Then the user can view the tagged nodes to see how well connected their partonomies are. If there is a bridging node that needs to be undeleted but is not part of the lexi-

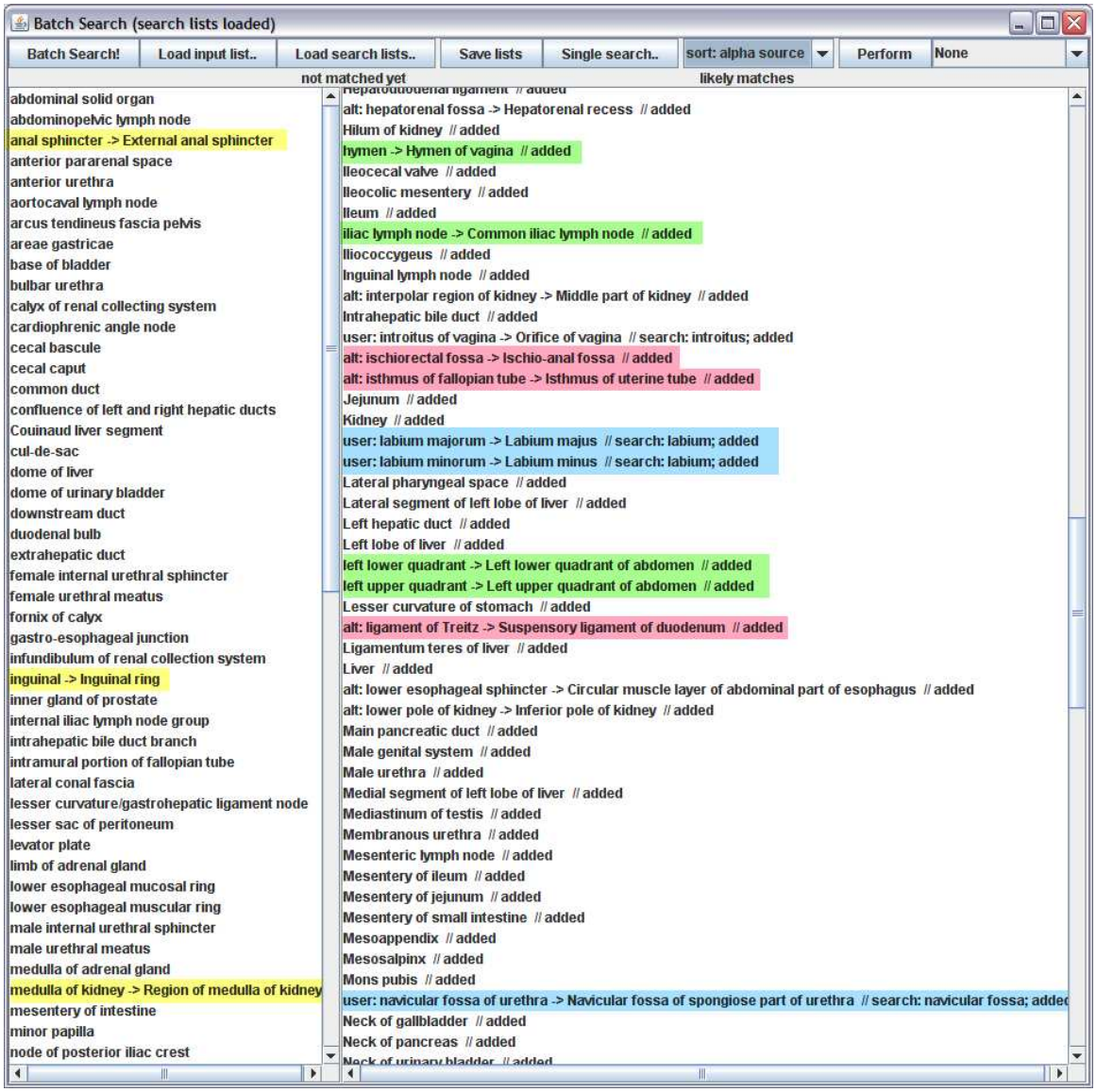

Figure 6.20: The batch search in use. The items highlighted yellow were moved from "found" to "not found" by the user. The other highlighted items illustrate different types of matches.

con, that can be tagged another color. If the user is happy with the connectivity of a set of nodes, they all can be tagged a third color (quickly in the query interface). Once all the terms have been matched, and all the matched terms have been tagged this third color, the alignment can be considered complete, unless the user has additional requirements.

Figure 6.21 shows a set of matched terms that are well connected in the part hierarchy. In this case, orange is used to denote that the user is happy with them and does not need to deal with them further. Figure 6.22 is a view where the user still needs to resolve the part relationships for the lime terms. Once the user fixes these issues and is satisfies, he or she will tag them orange. Finally, looking at the subclass view in figure 6.23 gives an overview of what has been done and what unresolved problems still remain.

6.7 Tutorial

The ontology browser supports an interactive step-by-step tutorial to help users learn how to use the system. As the programmer and most experienced user of my own application, I often forget that it has some learning curve. I designed the controls to be simple, to the point where it feels almost as easy to me as playing a videogame, but it takes users time to learn an application or even a videogame. The program has a rough help document, but even with revisions or perhaps even a document that painstakingly details a walkthrough, users might still get frustrated. One tactic employed in many games is to start off with a live walkthrough/tutorial that focuses on only limited features and expands the capabilities once the user has mastered the basics. The educational software Alice[3], which has a visual programming environment for creating stories/animations, has some excellent interactive tutorials. As an experienced user (the tutorial author) builds a workspace, he or she can also insert comments. The

[3]http://www.alice.org

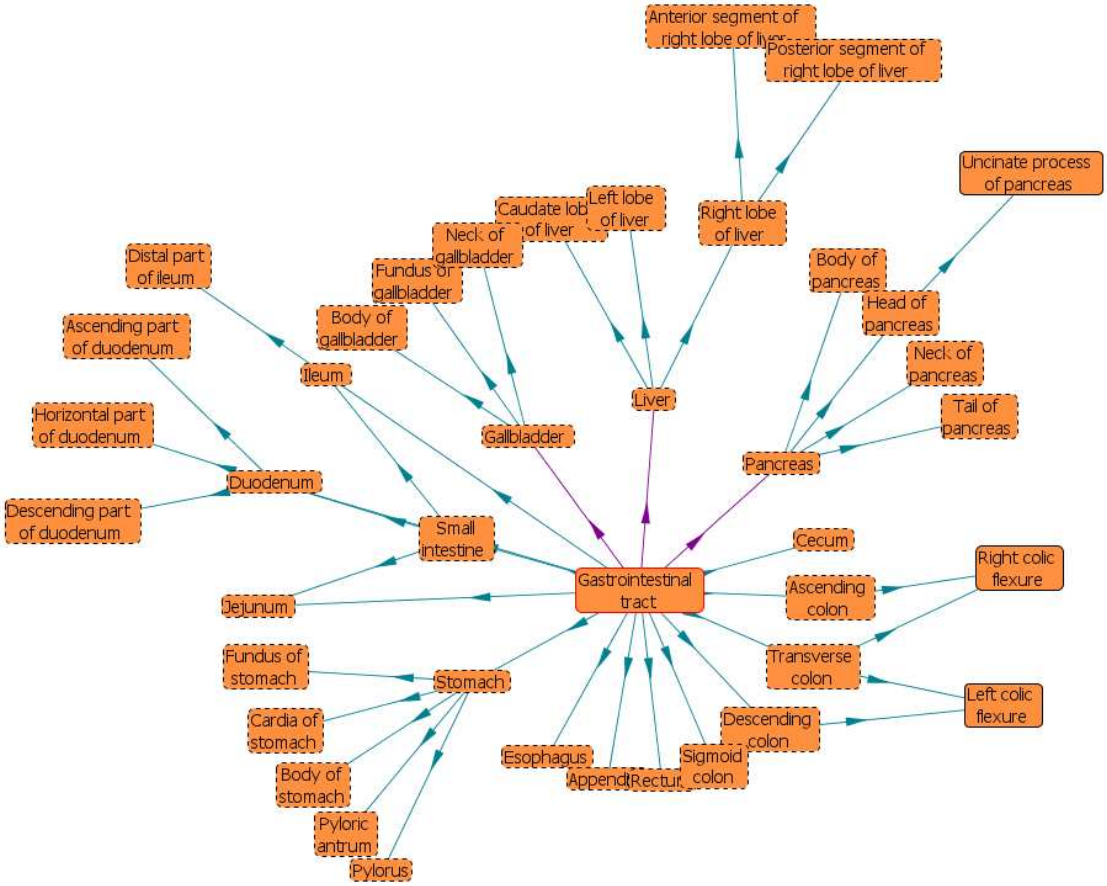

Figure 6.21: A large set of RadLex terms are connected via part relationships. The user highlighted these orange as a reminder that they are content with them.

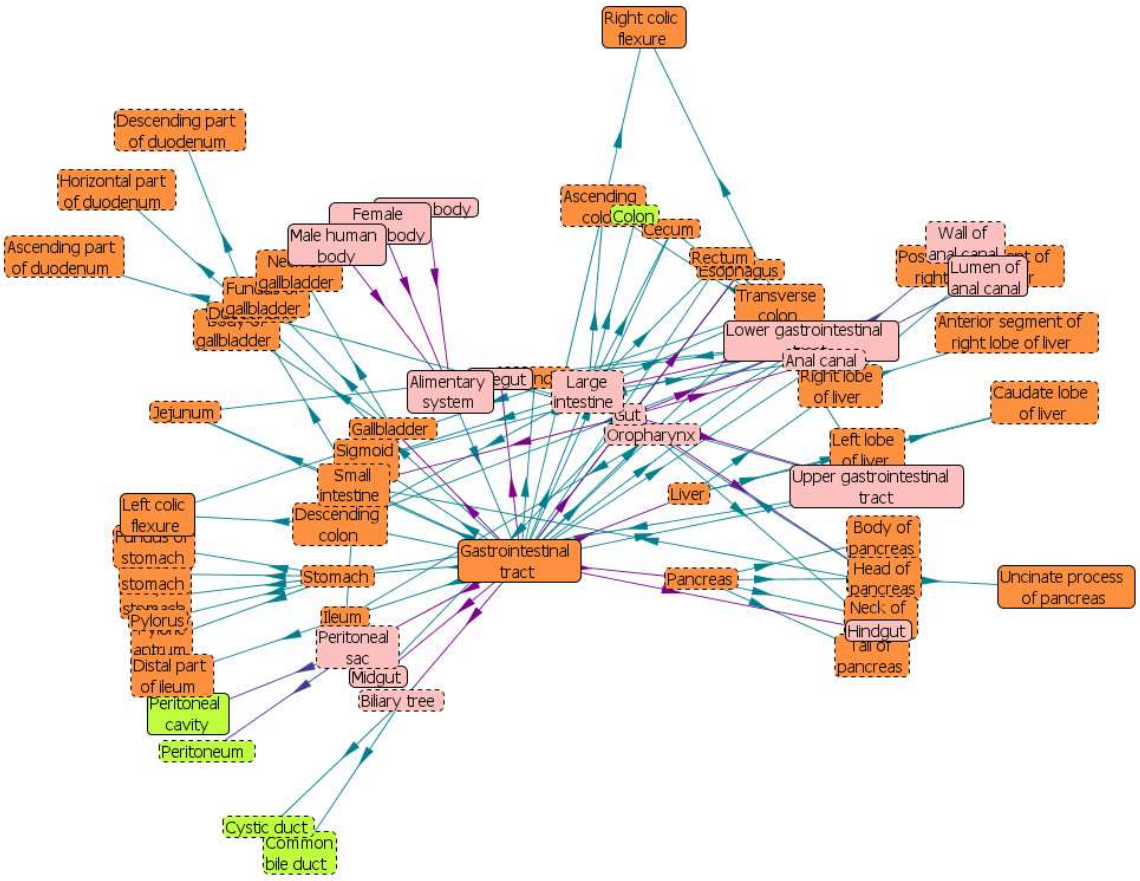

Figure 6.22: Here some terms are connected, but some are not. The lime terms need to be connected by entities that are not part of RadLex.

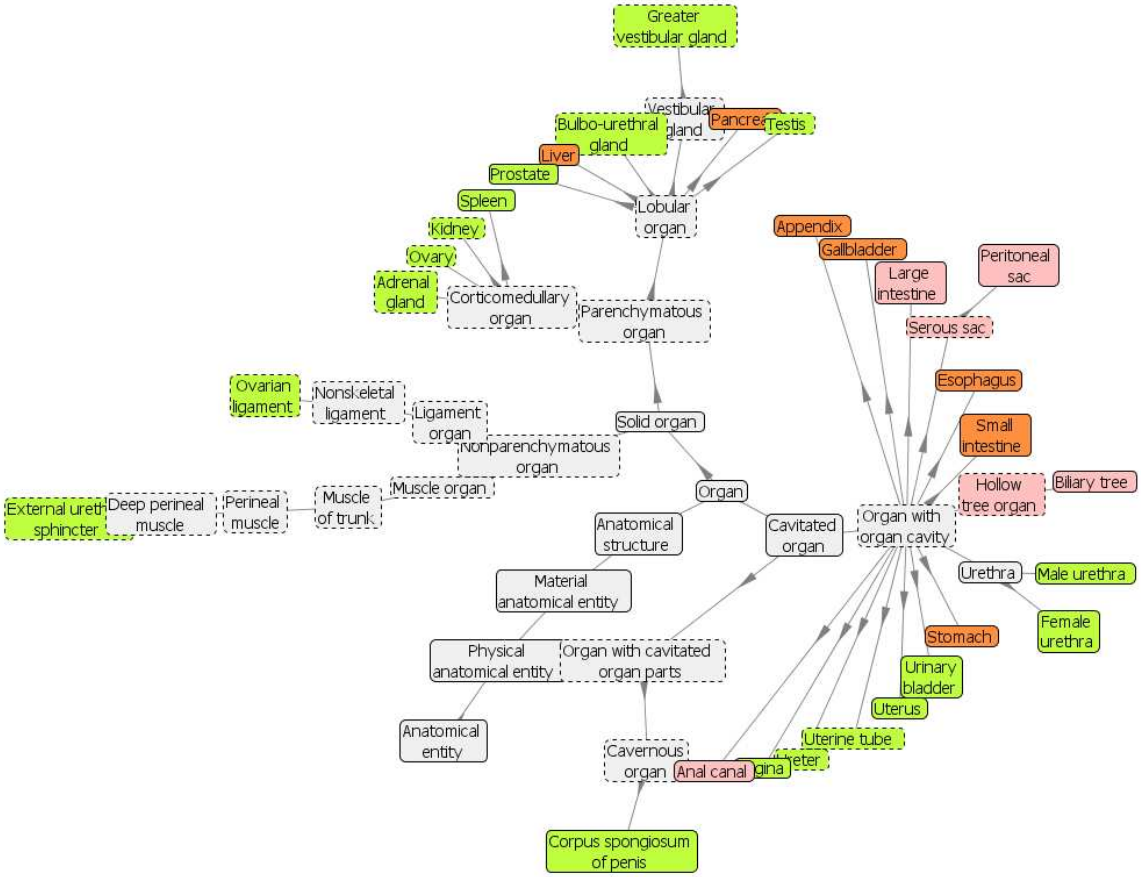

Figure 6.23: This view gives an overview of how many RadLex terms that are organs still need to be arranged in part relationships. The user can then click on one of the lime terms, change the view to part, and work on it, tagging it orange when happy.

workspace can then later be played back in a special mode (the tutorial mode) where the comments appear in a tutorial window one at a time and only advance if the user is able to duplicate the next logged action. When a user performs an incorrect action, the system can check if if was the correct type of action or if the arguments for the action (e.g. clicked on the right node) were correct, and give the user feedback. Additional context-dependent help could be built into each type of logged action. There are currently three tutorials for the system. The first focuses on navigation, the second on adding and deleting nodes, and the third on adding and deleting edges.

6.8 Evaluation

The ontology browser is undergoing iterations of user evaluations and refinement. Many ideas, such as the need for a tutorial, having a single "expand" operator that abstracts away loading from a back end, and displaying the counts of branches, originated from user feedback. Early feedback consisted of much frustration of "not knowing what to do" and being overwhelmed with all the features and the subtle visual cues; each of these has been addressed, and subsequent evaluations have shown marked improvement The evaluations have been productive in identifying annoyances, points of confusion, and other usability problems. Feedback so far is positive regarding use as a browser for the FMA. Users felt they learned about the FMA with the tool, and they would use the tool again. They spoke highly of the search and history capabilities.

Fourteen people who tried the browser unsupervised filled out an online survey. The full survey is reproduced in Appendix A. The first part of the survey assessed their familiarity with ontologies, anatomy, computer applications, database queries, and the FMA. The vast majority rated themselves as having above-average familiarity with computer applications and queries and average to below-average familiarity with anatomy. The one user who was an expert in anatomy had only a layperson's knowledge of anatomy. Another user was very computer savvy, yet had almost no

survey question (1: strongly disagree – 5: strongly agree)	mean	stddev
From using this software, my understanding of the FMA has improved.	3.86	0.363
I would use this software in the future to explore the FMA.	4.00	0.784
Using this software, I could effectively search for terms in the FMA.	4.21	0.802
Using this software, I could do simple queries on the FMA.	3.79	0.802
Using this software, I could do complex queries on the FMA.	3.00	0.877
The help documents were useful.	4.5	0.519
I did not get confused much.	3.29	0.994
I feel like I learned how to use this software.	3.86	0.535

Table 6.1: Users' evaluations of the ontology browser

anatomical knowledge. Another user was savvy in both anatomy and computers. Given that most of the audience were people associated with bioinformatics, most were at least acquainted with ontologies and the FMA. Given the size of the sample and the skew of the users, no reliable correlations can be drawn between their prior knowledge and their evaluation of the system. A future study should target either people who are more allied with biology or people who are downloading ontologies from the National Center for Biomedical Ontology.

A series of questions assessed the users' effectiveness and satisfaction with the tool on a discrete scale of 1 to 5 from strongly disagree to strongly agree. Table 6.1 shows the results, which were generally positive aside from performing complex queries. Half the users spent less than half an hour with the system; one user logged more than two hours. There do not appear to be any significant correlations between the time spent using the system and the subsequent evaluation ratings.

One question asked the user to state the most interesting thing learned about the

FMA from the software. This question was designed to assess insight and discovery from the undirected exploratory process of browsing. Unfortunately users did not reply with exact examples, either because they did not read the survey until afterward and did not remember the exact examples, or the question was too vague. Nevertheless, many users noted that the browser enabled them to fathom the complexity of the FMA especially when looking several levels deep. This complexity cannot be seen in an indented list or a Protégé frame. One user discovered that subclass/superclass and instance/type seemed to be mostly duplicates and was confused. This occurrence further motivates the need for a view. Non-expert users are probably not aware of the difference that Protégé makes between a class and instance; a "cleaned" view of the FMA could have instance and type relations removed.

A series of questions inquired on the likes and dislikes for the primary visualization, the search features, and the query features. Overall, users liked the accessibility of commands in the right-click menu, the navigation, and the layout. Some users had concerns that they would have liked to have a certain feature, but that feature was in fact present and additionally documented in the tutorial. Examples included the ability to restrict the depth of the layout and having a frame-like list of all the relationships for an entity. It is not known if the users in fact actively and whole-heartedly participated in those tutorials. Other concerns included the desires for features such as being able to move nodes manually, better handling of overlaps of nodes, less finicky performance of tool tips from hovering over edges, and disambiguating the subrelationships with a legend and a more specific mouse-over. For the search, a few complained that one had to hit "go" to search, rather than just press enter in the search box — a habituation issue. Others complained that the quick search did not give them multiple search results or did not correct misspellings. The search window gives multiple search results and optionally can search for misspellings. Perhaps the tutorials need to discuss the search window, or the quick search should incorporate features of the search window. No one experimented much with the query window;

clearly the user interface needs to be made more novice-friendly, which would be a significant undertaking. As some users suggested, a tutorial for the query window would be a good start.

Finally, users listed other suggestions or desired features for the ontology browser. They wanted to be able to select several nodes at once, such as with a lasso or by control-clicking, and perform an operation on the selected set, for example deleting. In certain cases, some actions may not produce easily noticeable visual changes; a cue such as a beep may help. One user suggested that if one were to search for a node that is already visible in the layout, it would be highlighted rather than become the new root. Users suggested a legend to explain the colors and styles of the borders and edges, and perhaps the borders and edges being thicker and their colors more distinct.

The evaluations just described are more qualitative in nature than quantitative. I have plans to conduct a comparison of my system with other ontology visualization tools, such as Jambalaya[45] and TGVizTab [4]. The comparison would consist of obtaining timing results on tasks and measuring effectiveness, efficiency, satisfaction via a survey, e.g. the System Usability Scale (SUS). Given that the previous studies have been somewhat biased toward people with a stronger computer background than a biology background, this study should focus on a population with more of a biology background. Unfortunately, I have not been able to find another graphical tool, including the two tools just mentioned, that is capable of practically visualizing the FMA, because it is so large. Instead I plan on using a small meaningful subset of the FMA, namely a knowledge base of the RadLex abdominal terms, for running the user experiments.

6.9 Conclusions and Future Work

The ontology browser enables a user to navigate and edit (create a view) of an ontology quickly and easily. Furthermore, the compact representation of the FMA that can

be used by the browser allows both the browser and the ontology to be delivered as a small payload, letting users experiment with minimal startup expense. To make the system ready to distribute to anyone, much more work still needs to be done for usability, especially with constructing tutorials and walkthroughs to get a user up to pace. Additionally, the system could prompt the user with hints as the user is exploring. Also, the editing features could be stripped down so that it is simpler to use (as a pure browser). On a different note, the browser could be embedded into Protégé, though for best results, it should have the option of spawning a new window, rather than remain embedded in a pane or tab (the creators of JSim added this feature to support the Model Browser).

As more feedback is obtained from users, more advanced edits will be made available that are compositions of more fundamental edits. For example, "bypass me" for subclasses could consist of redirecting all the node's children to have their parents be the node's parents. The challenge would be to work these feature into the user interface without making the menus too complex or the operations (click, drag, etc.) too overloaded. Right now dragging is reserved for panning and zooming. Other software takes advantage of a third mouse button (not present on every computer, especially laptops) or use of the control keys. Having to remember when to press shift, alt, or control is hard for the user to learn, so gestural mouse strokes may be better. Another option is to have different modes of interaction that the user toggles among.

Many ontologies have a variety of constraints, including on the domain and range for relations, on the values of attributes, and on the cardinality of the values. It would be a challenge to have a simple editing interface that does justice to these constraints. Some changes can only be done by temporarily violating the constraint so that it can be satisfied upon the next step. An example of such a change is changing a parent of a subclass, where the system forces the change to be performed atomically. As more constraints get enforced (e.g. cardinality constraints, of which single-inheritance is an example of), defining the set of operations that need to atomically compose more

primitive operations would get harder and harder. When users attempts to perform a change, they should be warned that an action may violate the constraints, and if so, why. Ideally the interface would give cues to what operations would be legal so that it is not trial-and-error. For that matter, some users may specifically want to ignore the constraints. The danger is that once constraints are ignored, it would be difficult to restore the system to a constrained state again if so desired, without undoing the intervening changes. A system such as Prompt could help resolve the conflicts.

The ontology browser uses a radial tree layout, which tends to perform well for ontology connectivity, even if there is the occasional cycle. There are certainly other layouts, such as force-directed layouts or treemaps. A fundamental challenge for any layout is label placement, since labels are so crucial for the understanding of an ontology. Distortion, such as a fish-eye lens or hyperbolic geometry, may help in dealing with many nodes on the screen. Similarly, a large ontology displayed on a virtual 10Kx10K screen would benefit from a picture-in-picture showing an overview with the current view framed.

All types of layout will suffer from the situation when a node has thousands of children, as is the case with "tendon" or "ligament." For this situation, it is best not to render each of the children as separate nodes, but rather as an "imposter" representing all of them (or at least all of them that have no other context) as a sector of a ring. The imposter could have a special interface to scroll through the entities, perhaps with an embedded fisheye.

The FMA has several types of instances. Some are trivial, such as synonyms that are really just the name plus an author and date. However, the reified relations (e.g. attributed part, attributed continuous with) map to attributes that have specific information qualifying the relationships. These instances may be best viewed as text, though it may be possible to encode some of the attributes of the instances as colors or some other schema. Additionally, some of the instances have directions/orientations associated with them, and these could be used geographically for the layout.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This dissertation presented three browsers bridging the cognitive gaps from computational to abstract, for models, data, and ontologies. As these three areas are intertwined in biological research, an interesting direction for future work would be to build browsers to link these areas. Certainly data from simulation could be incorporated into the model browser; however, animating several hundred nodes at a time would not be effective. The model browser supports a hierarchy upon the variables; this hierarchy can be based on the FMA, or potentially an application ontology derived from anatomy and physiology foundational ontologies. In line with the ontology browser, variables from a model or simulation could be related to anatomical entities by a "has variable" relation, and the variables could be linked to each other by a "has dependency" relation. The challenge would be to find a suitable layout. Recall that the model browser does not display labels unless either selected by the user and there was enough room, or the user hovered over a variable. Its layout is suitable to accommodating on the order of a thousand nodes at a time. The ontology browser always displays labels, and its radial layout supports less nodes visible simultaneously than a space-filling layout. Additionally, straight edges do no perform well for items that are dense and collinear; such edges would appear for equations representing local anatomy. The hierarchical edge bundling layout accommodates the collinearity by rendering the edges as curves. Finally, specific software for building application models from ontologies could incorporate specific aspects of the visualizations to assist the user with the specific task. The visualization and interaction would be dependent on the exact needs of the user of such a tool.

Over ten years ago, Fred Brooks[11] wrote a piece on the computer scientist as a toolsmith. The toolsmith makes tools that are *intelligence amplifying*, that is mind and machine working together, potentially superior to artificial intelligence (cutely posed as IA>AI). He continues that the great tool users are the scientists—physicists, chemists, biologists—and the toolsmiths should feel rewarded by the tools' application. As a such a toolsmith myself, I am very much delighted by the chance to collaborate and help someone with my tools. Brooks notes that solving a multidisciplinary problem may lead to better computer science research, due to the fact that the problem is real-world, not a toy problem. Throughout working on this thesis, I have appreciated the challenge of the real world, especially the scalability problems arising with large ontologies such as the Foundational Model of Anatomy.

The three browsers described in this dissertation amplify the intelligence of those using them. The animated data browser lets one reason about multiple dimensions simultaneously, seeing viscerally how the whole system is behaving. The model browser gives a user a much richer navigation through the set of model equations than is afforded by the code itself. The hierarchical and clustering features of the browser can put additional structure onto the equations. The ontology browser lets a user explore a large complex ontology through local contexts. The ontology format can be delivered with minimal effort needed by the user. The ontology browser also supports editing of an ontology for the purpose of producing views of it. These views allow people to customize knowledge bases for their particular domains. Though these browsers were designed to be used as research tools, they could perhaps be adapted for educational purposes. Likely some amount of customization and authoring would be necessary. At the very least, I hope the tools inspire interest and awareness of computer science, biology, and the interdisciplinary bridge between.

BIBLIOGRAPHY

[1] James Agutter, Noah Syroid, Frank Drews, Dwayne Westenskow, Julio Bermudez, and David Strayer. Graphic data display for cardiovascular system: Case study. In *IEEE Symposium on Information Visualization*, 2001.

[2] C. Ahlberg and B. Schneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of CHI*, 1994.

[3] David Akers, Anthony Sherbondy, Rachel Mackenzie, Robert Dougherty, and Brian Wandell. Exploration of the brain's white matter pathways with dynamic queries. In *IEEE Visualization*, 2004.

[4] Harith Alani. TGVizTab: An ontology visualisation extension for Protégé. In *Proceedings of Knowledge Capture, Workshop on Visualization Information in Knowledge Engineering*, 2003.

[5] Robert Albert, Noah Syroid, Yinqi Zhang, Jim Agutter, Frank Drews, Dave Strayer, George Hutchinson, and Dwayne Westenskow. Psychophysical scaling of a cardiovascular information display. In *IEEE Visualization*, 2003.

[6] M. Antoniotti, I. T. Lau, and B. Mishra. Naturally speaking: A system biology tool with natural language based interfaces. In *Biological Language Conference*, 2004.

[7] Chandrajit Bajaj, Peter Djeu, Vinay Siddavanahalli, and Anthony Thane. TexMol: Interactive visual exploration of large flexible multi-component molecular complexes. In *IEEE Visualization*, 2004.

[8] C. A. H. Baker, M. S. T. Carpendale, P. Prusinkiewicz, and M. G. Surette. GeneVis: Visualization tools for genetic regulatory network dynamics. In *IEEE Visualization*, 2002.

[9] Giusseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis Tollis. Annotated bibliography on graph drawing algorithms. *Computer Geometry: Theory and Applications*, 4:235–282, 1994.

[10] Abraham Bernstein and Esther Kaufmann. GINO - a guided input natural language ontology editor. In *5th International Semantic Web Conference (ISWC 2006)*, pages 144–157. Springer, November 2006.

[11] Fred Brooks. The computer scientist as a toolsmith II. *Communications of the ACM*, 39(3):63–70, 1996.

[12] Stuart K. Card, Jock D. Mackinlay, and Ben Schneiderman. *Readings in Information Visualization: Using Vision To Think*. Morgan Kaufmann Publishers, Inc., 1999.

[13] Jean-Louis Coatrieux and James Bassingthwaighte. Scanning the issue: Special issue on the physiome and beyond. *Proceedings of the IEEE*, 94(4), 2006.

[14] Daniel L. Cook, Jose L. V. Mejino, and Cornelius Rosse. Evolution of a foundational model of physiology: Symbolic representation for functional bioinformatics. In *MEDINFO*, 2004.

[15] Daniel L. Cook, Jesse C. Wiley, and John H. Gennari. Chalkboard: Ontology-based pathway modeling and qualitative inference. *Preprint*, 2007.

[16] Edmund J. Crampin, Matthew Halstead, Peter Hunter, Poul Nielsen, Denis Noble, Nicolas Smith, and Merryn Tawhai. Computational physiology and the physiome project. *Experimental Physiology*, 89(1):1–26, 2004.

[17] Tricia d'Entremont and Margaret-Anne Storey. Using a degree-of-interest model for adaptive visualizations in Protégé. In *Proceedings of the 9th International Protégé Conference*, 2006.

[18] Deborah Dowling. Experimenting on theories. *Science in Context*, 12(2):261–274, 1999.

[19] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *ACM CHI*, 2005.

[20] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[21] Xiaodi Huang, Peter Eades, and Wei Lai. A framework of filtering, clustering and dynamic layout graphs for visualization. *Conferences in Research and Practice in Information Technology*, 38, 2005.

[22] Peter J. Hunter. Modeling human physiology: The IUPS/EMBS physiome project. *Proceedings of the IEEE*, 94(4), 2006.

[23] Chris Johnson. Top scientific visualization research problems. *IEEE Computer Graphics and Applications*, 24(4), 2004.

[24] Chris Johnson, Robert Moorhead, Tamara Munzner, Hanspeter Pfister, Penny Rheingans, and Terry S. Yoo. *NIH/NSF Visualization Research Challenges Report*. IEEE Press, 2006.

[25] Ira Kalet, Mark Whipple, Silvia Pessah, Jerry Barker, Mary Austin-Seymour, and Linda Shapiro. A rule-based model for local and regional tumor spread. In *Proceedings of AMIA*, 2002.

[26] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods — a survey. *ACM Computing Surveys*, 2007 (to appear).

[27] Akrivi Katifori, Elena Torou, Constantin Halatsis, Georgios Lepouras, and Costas Vassilakis. A comparative study of four ontology visualization techniques in Protégé: Experiment setup and preliminary results. In *Proceedings of Information Visualization*, 2006.

[28] Esther Kaufmann and Abraham Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In *6th International Semantic Web Conference (ISWC 2007)*, pages 281–294, 2007.

[29] Roy C.P. Kerckhoffs, Maxwell L. Neal, Quan Gu, James B. Bassingthwaighte, Jeff H. Omens, and Andrew D. McCulloch. Coupling of a 3d finite element model of cardiac ventricular mechanics to lumped systems models of the systemic and pulmonic circulation. *Annals of Biomedical Engineering*, 2007.

[30] William D. Lakin, Scott A. Stevens, Bruce I. Tranmer, and Paul L. Penar. A whole-body mathematical model for intracranial pressure dynamics. *Journal of Mathemetical Biology*, 46:347–383, 2003.

[31] Patrick Lambrix, Manal Habbouche, and Marta Perez. Evaluation of ontology development tools for bioinformatics. *Bioinformatics*, 19(12):1564–1571, 2003.

[32] Catherine M. Lloyd, Matt D.B. Halstead, and Poul F. Nielsen. CellML: its futre, present and past. *Progress in Biophysics and Molecular Biology*, 85:433–450, 2004.

[33] Max Lewis Neal and James B. Bassingthwaighte. Subject-specific models for the estimation of cardiac output and blood volume during hemorrhage. *submitted to Critical Care Medicine*, 2007.

[34] Chris North. Toward measuring visualization insight. *IEEE Computer Graphics and Applications*, 26(3), 2006.

[35] Natasha F. Noy and Mark A. Musen. The PROMPT suite: Interactive tools for ontology merging and mapping. Technical report, Stanford Medical Informatics, 2003.

[36] J. Tinsley Oden, Ted Belytschko, Jacob Fish, Thomas J.R. Hughes, Chris Johnson, David Keyes, Alan Laub, Linda Petzold, David Srolovitz, and Sidney Yep. Revolutionizing engineering science through simulation. *NSF Blue Ribbon Panel on Simulation-based Engineering Science*, 2006.

[37] Mette S. Olufsen, Ali Nadim, and Lewis A. Lipsitz. Dynamics of cerebral blood flow regulation explained using a lumped parameter model. *Am J Physiol Regulatory Integrative Comp Physiol*, 282:R611–R622, 2002.

[38] David Stephen John Perrin. PROMPT-Viz: Ontology version comparison visualizations with treemaps. Master's thesis, University of Victoria, 2001.

[39] George Robertson, Kim Cameron, Mary Czerwinski, and Daniel Robbins. Polyarchy visualization: Visualizing multiple intersecting hierarchies. In *CHI*, 2002.

[40] Cornelius Rosse and Jose L. V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36:478–500, 2003.

[41] Daniel L. Rubin, David Grossman, Maxwell Neal, Daniel L. Cook, James B. Bassingthwaighte, and Mark A. Musen. Ontology-based representation of simulation models of physiology. In *AMIA Annual Symposium Proceedings*, 2006.

[42] Purvi Saraiya, Peter Lee, and Chris North. Visualization of graphs with associated timeseries data. In *IEEE Symposium on Information Visualization*, 2005.

[43] Purvi Saraiya, Chris North, and Karen Duca. An insight-based methodology for evaluating bioinformatics visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 11(4), 2005.

[44] N. P. Smith, D. P. Nickerson, E. J. Crampin, and P. J. Hunter. Multiscale computational modelling of the heart. *Acta Numerica*, pages 371–431, 2004.

[45] M. A. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy. Jambalaya: an interactive environment for exploring ontologies. In *Intl Conference on Intelligent User Interfaces*, 2002.

[46] Teranode. Leveraging pathway analytics for life sciences research and development, 2005.

[47] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1999.

[48] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1999.

[49] Fan-Yin Tzeng and Kwan-Liu Ma. Opening the black box – data driven visualization of neural networks. In *IEEE Visualization*, 2005.

[50] Rosario Uceda-Sosa, Cindy X. Chen, and Kajal T. Claypool. CLOVE: A framework to design ontology views. In *ER*, pages 844–849, 2004.

[51] Frank van Ham, Huub van de Wetering, and Jarke J. van Wijk. Interactive visualization of state transition systems. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):319–329, 2002.

[52] Martin Wattenberg. Visual exploration of multivariate graphs. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2006.

[53] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. Animated exploration of dynamic graphs with radial layout. In *IEEE Symposium on Information Visualization*, 2001.

Appendix A

USABILITY SURVEY

The following pages show the web survey that was given to the users of the ontology browser.

OntologyView User Survey

The OntologyView is a browsing software designed for exploration of ontologies such as the FMA. Though it may be used for specific tasks such as query, search, and alignment, those directed tasks are not the primary focus of the software. As there really isn't any suitable software available to do a comparison (e.g. Protege is not designed for exploration), this survey will not be used to show quantitatively how effective it is, nor will it ask the user to do specific tasks for speed or accuracy. Rather, it will qualitatively evaluate how much better the user understands the FMA, assess the usability of the components, and ask for feedback. Thank you for participating.

Question 1.

Required.

	don't know much at all	layperson's knowledge	aquainted	saavy	expert understanding
Familiarity with ontologies	○	○	○	○	○
Familiarity specifically with FMA	○	○	○	○	○
Familiarity with anatomy/medicine	○	○	○	○	○
Familiarity with computer applications	○	○	○	○	○
Familiarity with database queries	○	○	○	○	○

Question 2.

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
From using this software, my understanding of the FMA has improved.	○	○	○	○	○
I would use this software in the future to explore the FMA.	○	○	○	○	○
Using this software, I could effectively search for terms in the FMA.	○	○	○	○	○
Using this software, I could do simple queries on the FMA.	○	○	○	○	○
Using this software, I could do complex queries on the FMA.	○	○	○	○	○
The help documents were useful.	○	○	○	○	○
I did not get confused much.	○	○	○	○	○
I feel like I learned how to use this software.	○	○	○	○	○

Question 3.

What's the coolest / most interesting thing you learned about the FMA from using this software?

Question 4.

Time spent using the software:

- ○ less than 15 minutes
- ○ 15-30 minutes
- ○ 30-60 minutes
- ○ 60-120 minutes
- ○ more than 120 minutes

Next Cancel

OntologyView User Survey

Question 5.

For the main visualization, what worked well?

Question 6.

For the main visualization, what was irritating/confusing?

Question 7.

For the search components, what worked well?

Question 8.

For the search components, what was irritating/confusing?

Question 9.

For the query window, what worked well?

Question 10.

For the query window, what was irritating/confusing?

Question 11.

Were the tutorials useful?

What would you like to see in another tutorial?

Question 12.

Other suggestions or feature requests?

Question 13.

Any other comments (please email bug reports directly to me (gyngve@cs.washington.edu) so that I can obtain further information from you about how the bugs manifested themselves)?

Previous Submit responses Cancel

VITA

Gary Yngve earned a Bachelor of Science in Computer Science at Georgia Institute of Technology in 2000. In 2007, he earned a Doctor of Philosophy in Computer Science at the University of Washington.