DIOGO FERNANDO VEIGA
PEDRO DE STEGE CECCONELLO

Metabolic IsaViz: Representando Vias Metabólicas em Grafos RDF Customizados

FLORIANÓPOLIS
2004

DIOGO FERNANDO VEIGA
PEDRO DE STEGE CECCONELLO

Metabolic IsaViz: Representando Vias Metabólicas em Grafos RDF Customizados

Relatório final do trabalho de conclusão do curso de Bacharelado em Ciências da Computação da Universidade Federal de Santa Catarina.

Orientador: Prof. Dr. José Eduardo De Lucca, INE/CTC/UFSC
Coorientador: Prof. Dr. Luismar Marques Porto, EQA/CTC/UFSC

Membros da Banca:
Prof. Dr. Leandro José Komosinski, INE/CTC/UFSC
Prof. Dr. João Bosco Mangueira Sobral, INE/CTC/UFSC

FLORIANÓPOLIS
2004

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

As vias metabólicas são conjuntos de reações nos organismos que sintetizam ou degradam moléculas, como proteínas e carboidratos. Criar modelos computacionais que possam ser analisados é fundamental para aplicações em biotecnologia, como a obtenção da máxima quantidade de um composto químico de interesse. O propósito deste trabalho foi combinar uma tecnologia de representação de dados, chamada RDF (Resource Description Framework) com a representação de vias metabólicas, ambas baseadas em grafos direcionados. Para isto, implementou-se as modificações lógicas/gráficas em um ambiente de desenvolvimento RDF já existente, conhecido como IsaViz. As modificações necessárias para a adaptação do software ao domínio das vias bioquímicas juntamente com uma biblioteca de classes e propriedades RDF – a Biblioteca Genômica, que foi acoplada ao sistema – resultou no Metabolic IsaViz, uma extensão do IsaViz destinado à criação de modelos de vias metabólicas e regulatórias. O Metabolic IsaViz, no âmbito do projeto de pesquisa do Grupo de Engenharia Genômica da UFSC, será a base de um sistema de informação genômica, chamado GenIS (Genome Information System). Esta plataforma computacional será utilizada para a simulação de vias bioquímicas e pretende reduzir as atividades experimentais, obtendo, por meio das simulações in silico, indícios para conduzir os trabalhos em laboratório. Os modelos de vias serão criados com o Metabolic IsaViz informando parâmetros necessários às equações de simulação, como as constantes cinéticas enzima-substrato, constante de inibição e outras e a partir daí o código RDF/XML será interpretado por um simulador de vias, baseados em RPHs (Redes de Petri Híbridas).

Palavras-chave: vias metabólicas, RDF, IsaViz.

1 – Introdução e Justificativa

A determinação das vias metabólicas e de controle regulatório de organismos, a partir da informação genômica seqüenciada, é uma importante etapa para estudos de análise pós-seqüenciamento. Esta caracterização é possível porque a seqüência de DNA contém os genes que expressam as enzimas que, por sua vez, atuam nas vias bioquímicas como catalisadores biológicos, tornando possível a conversão entre compostos intermediários de uma certa via. O conhecimento das vias bioquímicas permite que sejam aplicadas técnicas que induzam à acumulação de um certo composto de interesse, por exemplo, para ser utilizado na fabricação de medicamentos, alimentos, etc.

Desta forma, projetar um ambiente gráfico para criação de modelos de vias metabólicas é uma iniciativa em relação à necessidade de análise e compreensão dos dados genômicos que, pelo desenvolvimento de novas técnicas de seqüenciamento, estão sendo gerados em grande volume.

O propósito deste trabalho foi combinar uma tecnologia de representação de dados, chamada RDF (KLYNE e CARROL, 2003; MANOLA e MILLER, 2003) com a representação de vias metabólicas (Vide seção 2.2 para conceituação básica), ambas baseadas em grafos direcionados. Para isto, implementamos as modificações lógicas/gráficas em um ambiente de desenvolvimento RDF já existente, conhecido como IsaViz (PIETRIGA, 2003). As modificações necessárias para a adaptação do software ao domínio das vias bioquímicas juntamente com uma biblioteca de classes e propriedades RDF – a Biblioteca Genômica, que foi acoplada ao sistema – resultou no Metabolic IsaViz, uma extensão do IsaViz destinado à

criação de modelos de vias metabólicas e regulatórias. O Metabolic IsaViz, no âmbito do projeto de pesquisa do Grupo de Engenharia Genômica da UFSC, será a base de um sistema de informação genômica, chamado GenIS (Genome Information System). Esta plataforma computacional será utilizada para a simulação de vias bioquímicas e pretende reduzir as atividades experimentais, obtendo, por meio das simulações *in silico*, indícios para conduzir os trabalhos em laboratório. Os modelos de vias serão criados com o Metabolic IsaViz informando parâmetros necessários às equações de simulação, como as constantes cinéticas enzima-substrato, constante de inibição e outras e a partir daí o código RDF/XML será interpretado por um simulador de vias, baseados em RPHs (Redes de Petri Híbridas). Este simulador ainda se encontra em processo de estudo.

1.1 – O RDF e a Web Semântica

No contexto da Web Semântica, o RDF desempenha o papel de representar os recursos e seus relacionamentos, ao lado da XML (Ver Tabela 1). A Web Semântica é considerada a próxima geração da Web, na qual o conteúdo será estruturado com aspectos sintáticos e, sobretudo, semânticos, agregando-se significado tanto para as máquinas quanto para as pessoas. Na Web atual, através dos documentos HTML, a informação é projetada para visualização dos usuários que a acessam. Adicionando páginas Web com dados destinados aos computadores, a Web se tornará a Web Semântica (BERNERS-LEE et al., 2000).

Tabela 1 - Tecnologias utilizadas na Arquitetura 1 da Web Semântica.

Tecnologia	Comentário
RDF (Resource Description Framework) / XML	Camada de representação de recursos, para descrição dos recursos e de seus relacionamentos.
URI (Uniform Resource Identifier)	Para identificação de recursos.
RDF Schema	Para definição de vocabulários.

A Web Semântica afirma que os conceitos e relacionamentos no domínio da aplicação devem ser descritos em uma linguagem comum, que defina todas as propriedades dos elementos do domínio. Pode-se estabelecer uma relação direta desta linguagem com as Tabelas de um banco de dados relacional. A outra suposição da Web Semântica é que sejam construídas ontologias para representar o domínio da aplicação. Os sistemas baseados em conhecimento (Knowledge-Base Systems) – área de desenvolvimento da Inteligência Artificial – as utilizam para extração do conhecimento por meio de regras de inferência.

A representação da informação nestes moldes permitirá alcançar níveis elevados de interoperabilidade entre os computadores que estão conectados à Internet, de forma que possam realizar tarefas em colaboração com outros recursos, de maneira automática, em um ambiente de informação distribuída.

Outra conseqüência será o acesso à informação com mais qualidade, o que torna a Web um grande banco de dados passível à descoberta de conhecimento e à dedução de novos fatos.

Diante deste potencial que emerge com a Web Semântica, uma das necessidades que diz respeito à sua difusão é a construção de aplicativos que utilizem as idéias e tecnologias da área para resolução de problemas cujo conhecimento está distribuído e necessita integração. Este projeto acredita que a representação RDF, inserida na Web Semântica, pode ser uma nova fronteira para inferência de conhecimento a partir da grande quantidade de dados relacionados ao

genoma.

1.2 – Adaptação Gráfica e Lógica do IsaViz

Considere a forma pela qual o aMAZE (van HELDEN et al., 2000), modelo para vias metabólicas adotado, denota um passo da via desde a expressão do gene (Figura 1). Esta forma de representação gráfica é adequada para conceber modelos biológicos de vias.

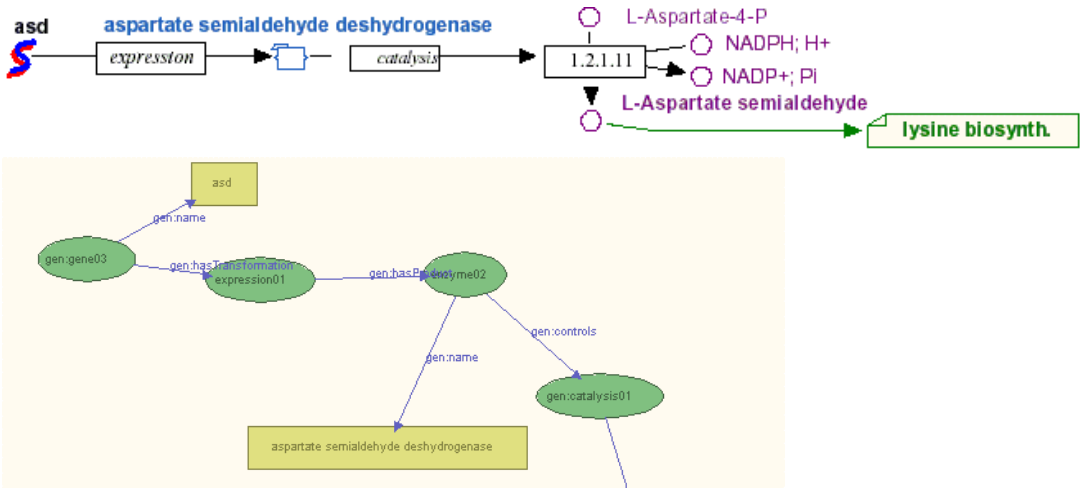

Figura 1 - Representação gráfica sugerida pelo modelo de vias bioquímicas aMAZE (acima). Esta representação é intuitiva para os profissionais da área. Abaixo está a representação gerada pelo IsaViz.

Com a ferramenta IsaViz (vide seção 2.3), para autoria de grafos RDF, a representação visual teria o aspecto da Figura 1 (figura de baixo). Ela não distingue tipos de nodos, o que torna não intuitiva a visualização do fenômeno biológico. Isto mostra a necessidade da adaptação da ferramenta para trabalhos específicos bem como para atuar na anotação das vias metabólicas. A manipulação gráfica das vias

é mais adequada para profissionais usuários destes softwares de análise, como biólogos, bioquímicos, profissionais da computação, etc.

Em resumo, podemos destacar aqui os fatores importantes nas escolhas das tecnologias propostas no trabalho:

1) A representação gráfica de vias metabólicas é muito semelhante a grafos direcionados. Isto tornou adequado utilizar RDF para descrição das vias já que seu modelo de dados é um multigrafo direcionado;

2) Representar em RDF traz vantagens:

- Utilizar tecnologia da Web Semântica (BERNERS-LEE et al., 2001), permite interoperabilidade entre softwares, através de agentes. A camada de representação da Web Semântica é em RDF ou XML Schema.
- RDF traz a idéia de trabalho cooperativo (MANOLA e MILLER, 2002), o que significa dizer que dados sobre um recurso RDF podem ser distribuídos entre vários *namespaces*. Relacionado às vias metabólicas, poder-se-ia dizer que enquanto um grupo de pesquisa faz a anotação das enzimas outro pode estar interessado em fazer simulações; fornece, portanto, ao recurso (neste caso a via em questão) dados sobre a cinética das reações. Um agente pode integrar estas informações.

3) IsaViz é a ferramenta recomendada pela W3C para a autoria de modelos RDF. Ela está em conformidade com as especificações RDF e oferece uma grande quantidade de recursos, como mecanismo de zoom nos grafos, aplicação de folhas de estilos GSS (Graph Stylesheets) aos modelos, bem como exporta RDF para muitos formatos, como SVG (Scalable Vector Graphics), PNG (Portable Network

Graphics) e propriamente para RDF/XML.

2 – Estudos Realizados

2.1 – Vias Metabólicas

As vias metabólicas são seqüências de reações que ocorrem nos organismos com a função de produzir energia através da degradação de macromoléculas (vias do metabolismo) ou de consumir energia (vias de anabolismo), buscando produzir moléculas complexas, tais como proteínas, com o objetivo de duplicação ou muitas outras funções celulares.

Considere o exemplo da via de biossíntese de carotenóides (ARACYC, 2003) encontrada na *Arabidopsis thaliana* (uma espécie de planta). Os carotenóides são uma classe de compostos produzidos pelo metabolismo secundário de plantas, que conferem pigmentação laranja e vermelha, sendo encontrados em laranjas, tomates, cenouras e flores amarelas. São usados como corantes em alimentos industrializados, bebidas e rações animais. Os carotenóides também são encontrados em bactérias, fungos e algas. São essenciais à fotossíntese, são antioxidantes, além da função de atrair animais. Para o homem, é uma importante fonte de vitamina A e combate vários tipos de câncer e doenças degenerativas da visão, atuando como reguladores do sistema imunológico.

A via de produção pode ser vista na Figura 2. O gráfico menor, ao lado da via, denota como as reações foram identificadas no processo de anotação da *A. thaliana*. As reações em verde significam que a enzima está presente no organismo

e a reação é única para esta via. Já para a reação denotada por uma linha preta a

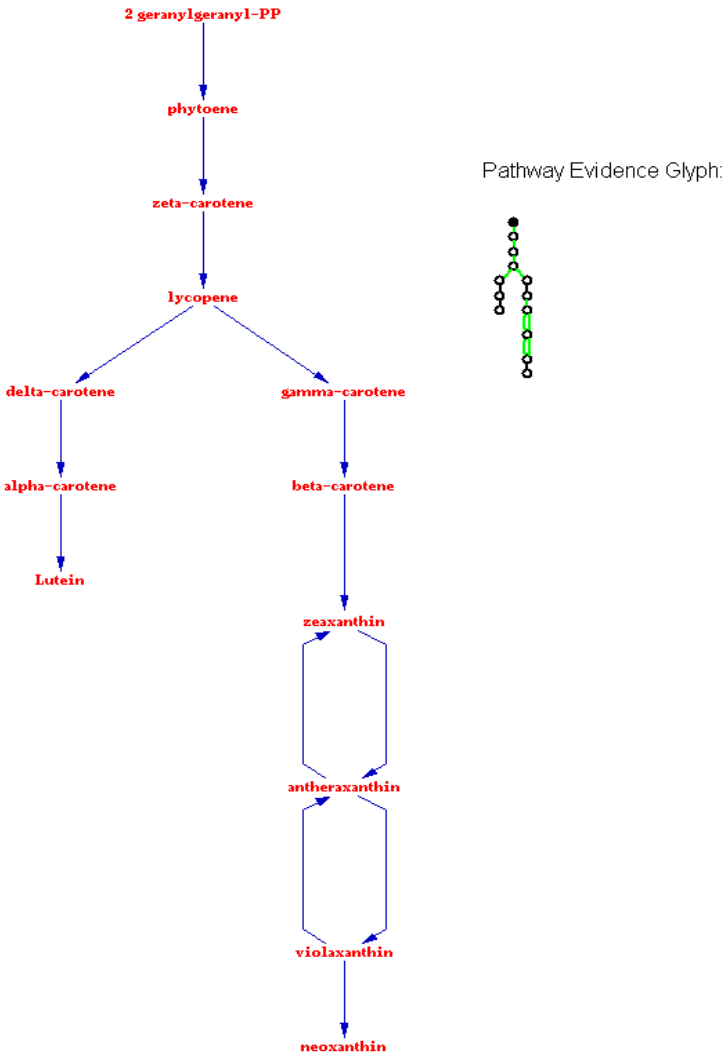

Figura 2 - Exemplo de via metabólica.

enzima não foi identificada e esta reação é única para esta via. Isto significa dizer que nem todos os passos da via acontecem para qualquer organismo e isto vai depender do conjunto de enzimas codificadas pelo genoma da espécie em estudo.

2.2 – Resource Description Framework

O RDF (*Resource Description Framework*) surgiu com o intuito de se tornar a linguagem de representação dos dados disponíveis na Web, constituindo-se uma camada essencial da Web Semântica. No contexto desta próxima Web, o RDF desempenha o papel de representar os recursos e seus relacionamentos, ao lado do XML. Um recurso web é considerado algo que possa ser acessível por um navegador, como um documento ou uma foto, quanto algo não acessível, porém também identificado por uma URI (*Uniform Resource Identifier*), tais como funcionários de uma empresa, carros em estoque, livros em uma biblioteca.

2.2.1 – Modelo de Dados e suas Diversas Sintaxes

Para cumprir seu objetivo de representação, a tecnologia RDF é composta de um modelo de dados, que é um multigrafo direcionado, e algumas linguagens de descrição para este modelo, tais como RDF/XML, N-triples e Notation-3.

O modelo de dados define a forma pela qual a informação de um certo domínio é representada. Com um grafo direcionado, podemos expressar facilmente os dados sobre um certo negócio, objeto, pessoa, etc. Em RDF, a descrição dos recursos ocorre por meio de *Statements* ou asserções, que graficamente denotam a ligação entre dois nodos de um grafo. Na terminologia RDF, o nodo de onde parte a aresta é chamado de *Subject*, enquanto o nodo onde chega a aresta é conhecido como *Object*. A própria aresta é denominada *Predicate*. Todos os elementos são rotulados por suas URIs.

A Figura 3 mostra um grafo RDF com uma única asserção, que descreve o modelo do carro `http://www.montadoraXX.com.br/estoque#carro0578` da montadoraXX. Neste exemplo, podemos identificar as partes de um *statement*, utilizando a terminologia comentada:

1) *Subject*: se refere ao recurso que está sendo descrito. Neste caso, o recurso é `http://www.montadoraXX.com.br/estoque#carro0578`.
2) *Predicate*: indica a propriedade ou característica do recurso. No exemplo, a propriedade é `http://www.montadoraXX.com.br/estoque#modelo`. Outras propriedades que podem identificar o carro são: número de portas, cilindradas, preço etc.
3) *Object*: parte do *statement* que indica o valor da propriedade. Em RDF o *object* pode ser um tipo simples, como uma string, uma data ou inteiro, bem como pode ser um outro recurso. O *object* da *statement* da Figura 3 é o recurso `http://www.montadoraXX.com.br/modelos#Gurgel`.

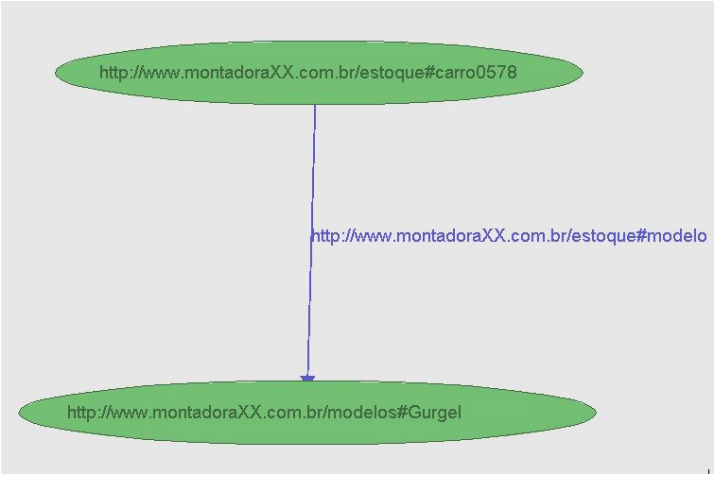

Figura 3 - Modelo RDF com uma única asserção.

Os URIs, como `http://www.montadoraXX.com.br/modelos#Gurgel`, têm

como objetivo identificar univocamente um recurso no espaço da Web e, como abordado anteriormente, não pretende, necessariamente, fornecer uma página que pode ser acessada por um navegador. URIs podem ser criadas para referenciar qualquer recurso, incluindo:

a) objetos acessíveis pela rede, como documentos eletrônicos, uma imagem, um serviço web;
b) coisas que não são acessíveis através de browsers, como funcionários de uma empresa, clientes, carros no estoque, etc;
c) conceitos abstratos que não existem fisicamente, como o conceito "*creator*" do vocabulário Dublin Core (DUBLIN CORE, 2003).

Outro exemplo, exposto na Figura 4 e referente ao mesmo domínio do exemplo anterior, torna claro que um modelo de dados pode ser visto como uma coleção de *statements*.

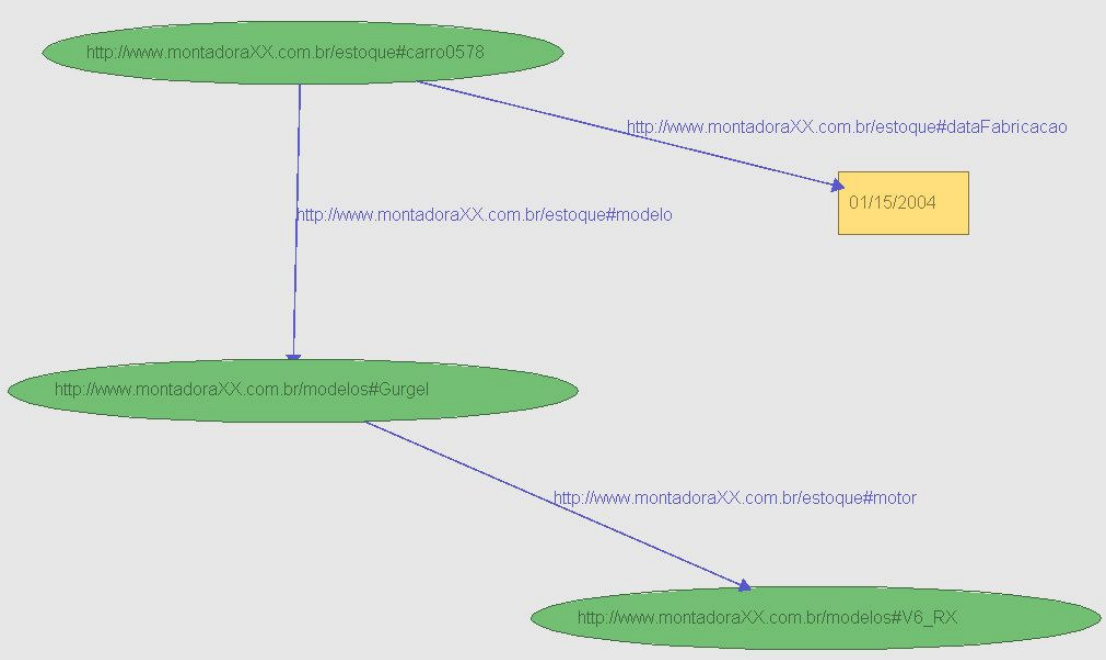

Figura 4 - Grafo RDF envolvendo várias statements, combinando recursos e literais.

Existem várias sintaxes para representação de grafos RDF. Podemos descrever o grafo acima em triplas ou em uma linguagem proveniente de XML. As sintaxes N-triples e RDF/XML para o modelo da Figura 4 são apresentadas nas Tabelas 2, 3 e 4.

Tabela 2 – Notação N-triple. O prefixo ou namespace car corresponde a http://www.montadoraXX.com.br/estoque# enquanto o prefixo carMd corresponde a http://www.montadoraXX.com.br/modelos#.

Notação N-triple
car:carro0578 car:dataFabricacao "01/15/2004".
car:carro0578 car:modelo carMd:Gurgel.
carMd:Gurgel car:motor carMd:V6_RX.

Tabela 3 – Notação RDF/XML. O prefixo carMd, existente na Tabela 2, é extinto.

Notação RDF/XML

```
1. <rdf:RDF   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
              xmlns:car='http://www.montadoraXX.com.br/estoque#'>
2. <rdf:Description
rdf:about='http://www.montadoraXX.com.br/estoque#carro0578'>
3.    <car:modelo
rdf:resource='http://www.montadoraXX.com.br/modelos#Gurgel'/>
4.    <car:dataFabricacao xml:lang='date'>01/15/2004</car:dataFabricacao>
5.  </rdf:Description>
6.  <rdf:Description
rdf:about='http://www.montadoraXX.com.br/modelos#Gurgel'>
7.    <car:motor rdf:resource='http://www.montadoraXX.com.br/modelos#V6
RX'/>
   8.  </rdf:Description>
   9. </rdf:RDF>
```

O XML/RDF pode ser escrito como na Tabela 4, com abreviações, porém o grafo resultante é o mesmo se escrevermos cada elemento rdf:description separadamente, como no exemplo da Tabela 3.

Tabela 4 – Notação RDF/XML abreviada. O prefixo carMd, existente na Tabela 2, também é extinto.

Notação RDF/XML Abreviada

```
1. <?xml version='1.0'?>
2. <rdf:RDF
   xmlns:car='http://www.montadoraXX.com.br/estoque#'
   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
3.  <rdf:Description
rdf:about='http://www.montadoraXX.com.br/estoque#carro0578'>
4.    <car:modelo>
5.      <rdf:Description
rdf:about='http://www.montadoraXX.com.br/modelos#Gurgel'>
6.        <car:motor
rdf:resource='http://www.montadoraXX.com.br/modelos#V6_RX'/>
7.      </rdf:Description>
8.    </car:modelo>
9.    <car:dataFabricacao xml:lang='date'>01/15/2004</car:dataFabricacao>
10. </rdf:Description>

11.</rdf:RDF>
```

Um recurso localizado em outro local pode ser referenciado pela sua URI completa. Considere o exemplo no qual uma barraca, que está em um certo catálogo, está sendo avaliada por um *site* de esporte. Este próximo exemplo mostra idéias-chave da Web Semântica: podemos afirmar qualquer coisa sobre recursos existentes e a informação sobre um determinado recurso não precisa estar em um só local, mas sim distribuída na Web (Ver Tabela 5).

Tabela 5 - Descrição de um recurso distribuído.

O recurso destacado em negrito, que contém sua descrição em outro local da web, está aqui sendo avaliado por um *site* de esportes.

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.           xmlns:sportex="http://www.exampleRatings.com/terms/">
4.    <rdf:Description
rdf:about="http://www.example.com/2002/04/products#10245">
5.         <sportex:ratingBy>Richard Roe</sportex:ratingBy>
6.         <sportex:numberStars>5</sportex:numberStars>
7.    </rdf:Description>
8.     </rdf:RDF>
```

XML/RDF prevê o uso de classes (tipos), construídos em RDFS (RDF Schema) (BRICKLEY & GUHA, 2003) ou através de linguagens específicas para ontologias como DAIML+OIL (CONNOLY, 2003) e OWL (SMITH, 2003). O exemplo abaixo (Tabela 6) mostra a notação utilizada, com o recurso sendo declarado com o

elemento rdf:type, proveniente de um esquema RDF.

Tabela 6 - Utilização de classes em RDF/XML.

Neste exemplo, o recurso cujo ID é 10245 é do tipo http://www.example.com/terms/Tent.

```
1.   <?xml version="1.0"?>
2.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.               xmlns:ex="http://www.example.com/terms/"
4.               xml:base="http://www.example.com/2002/04/products">
5.     <rdf:Description rdf:ID="10245">
6.          <rdf:type rdf:resource="http://www.example.com/terms/Tent" />
deve haver a definição da classe!!
7.          <ex:model>Overnighter</ex:model>
8.          <ex:sleeps>2</ex:sleeps>
9.          <ex:weight>2.4</ex:weight>
10.         <ex:packedSize>14x56</ex:packedSize>
11.       </rdf:Description>
12.  </rdf:RDF>
```

2.2.2 – Padronização de Vocabulários: a RDF Schema

Um vocabulário RDF é composto de classes e propriedades. A linguagem com a qual definimos estas classes e propriedades, ou seja, o vocabulário, é por si própria um vocabulário, chamado RDFS, que significa RDF Schema. O RDFS possui o *namespace* `http://www.w3.org/2000/01/rdf-schema#`, com o prefixo rdfs.

Uma classe, como um carro, imóvel ou pessoa, possui propriedades. Uma característica interessante em RDF é que as propriedades são definidas de forma independente, promovendo a sua reutilização.

Outro aspecto sobre RDFS é que não há necessariamente uma restrição às informações atribuídas a um recurso caso ele seja de uma certa classe, diferente do conceito de tipos do XML Schema, cujo objetivo é a validação dos dados através de uma sintaxe precisa.

A seguir mostra-se como são definidos os elementos dos esquemas RDF.

a) Definição de classes:

São utilizados os seguintes elementos RDF (Tabela 7):

Tabela 7 – Elementos RDF para definição de classes.

Recursos	Propriedades
rdfs:Class (criação da classe) rdfs:Resource	rdf:type rdfs:subClassOf

A atribuição de uma classe a um recurso se faz com a propriedade rdf:type. Um recurso pode ser instância de uma ou mais classes. Todas as classes são subclasses da classe rdfs:Resource (Ver Figura 5 e Tabela 8).

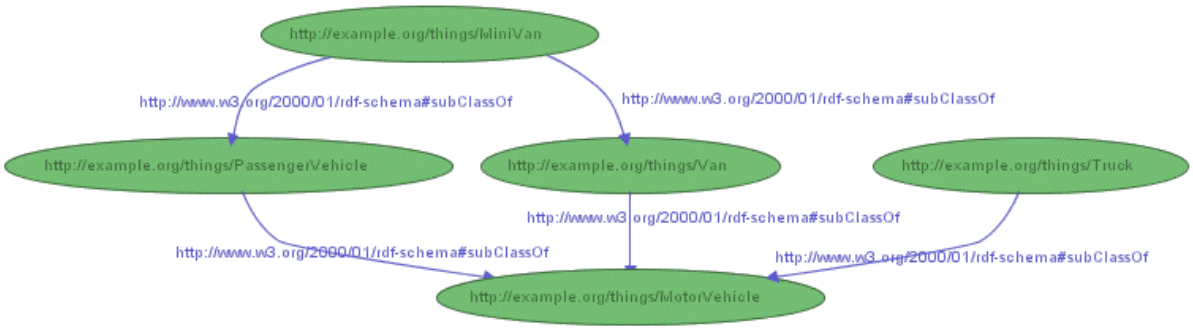

Figura 5 - Hierarquia de classes RDF descrevendo a relação entre veículo automotores e utilizando os elementos da Tabela 7.

Tabela 8 – Código RDF/XML correspondente à hierarquia de classes da Figura 5.

As classes estão delimitadas por elementos `rdf:Description`. O `rdf:ID` serve somente para referenciar o recurso dentro do mesmo esquema. Caso contrário, deve ser utilizada a URI completa como referência.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:ID="MotorVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/> /* subclasse de recurso!!!
</rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>


<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```

b) Definição de Propriedades:

A Tabela 9 mostra os elementos RDF utilizados na definição de propriedades, que são utilizados no exemplo da Tabela 10.

Tabela 9 – Elementos RDF para definição de propriedades.

Recursos	Propriedades
rdf:Property (classe RDF disponível)	rdfs:domain – indica a qual classe a propriedade deve ser aplicada rdfs:range –classe ou tipo de dado (inteiro, data) que a propriedade pode assumir rdfs:subPropertyOf

Tabela 10 – Cria propriedades para a estrutura de classes do exemplo anterior (Figura 5).

A propriedade `registeredTo` , de acordo com a definição abaixo, só pode ser aplicada a classe `MotorVehicle` e só assume valores da classe `Person`.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:ID="registeredTo">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="http://www.example.org/classes#Person"/>
</rdf:Description>

<rdf:Description rdf:ID="rearSeatLegRoom">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</rdf:Description> </rdf:RDF>
```

Propriedades que são subpropriedades de outras herdam características de *range* e *domain*. Agora, pode-se utilizar o vocabulário para definir o conteúdo RDF.

c) Interpretação de Esquemas RDF:

As descrições definidas por um esquema RDF são aplicadas de acordo com a aplicação, ou seja, nenhuma definição se constitui em restrições aos dados RDF, a menos que a aplicação que interpreta os dados assim o deseje.

Por exemplo, podemos ter instâncias de classe sem propriedades declaradas pelo esquema, ou mesmo com propriedades a mais, ou ainda com classes diferentes do que é determinado no rdf:range.

2.3 – IsaViz: Um Ambiente para Autoria em RDF

O IsaViz (PIETRIGA, 2003) é um ambiente visual para edição e navegação de modelos RDF, representados por grafos direcionados. O software é construído com

a linguagem Java e de código-livre. Recursos e literais são os nodos do grafo (elipses e retângulos respectivamente), com propriedades representando os vértices que ligam estes nodos. Desde a versão 2.0, o IsaViz suporta GSS (*Graph Stylesheets*), uma linguagem *stylesheet* derivada de CSS e SVG, para atribuir estilos aos elementos dos grafos. A interface gráfica do software pode ser dividida em quatro janelas principais, que estão descritas pelas Figuras 7 a 10.

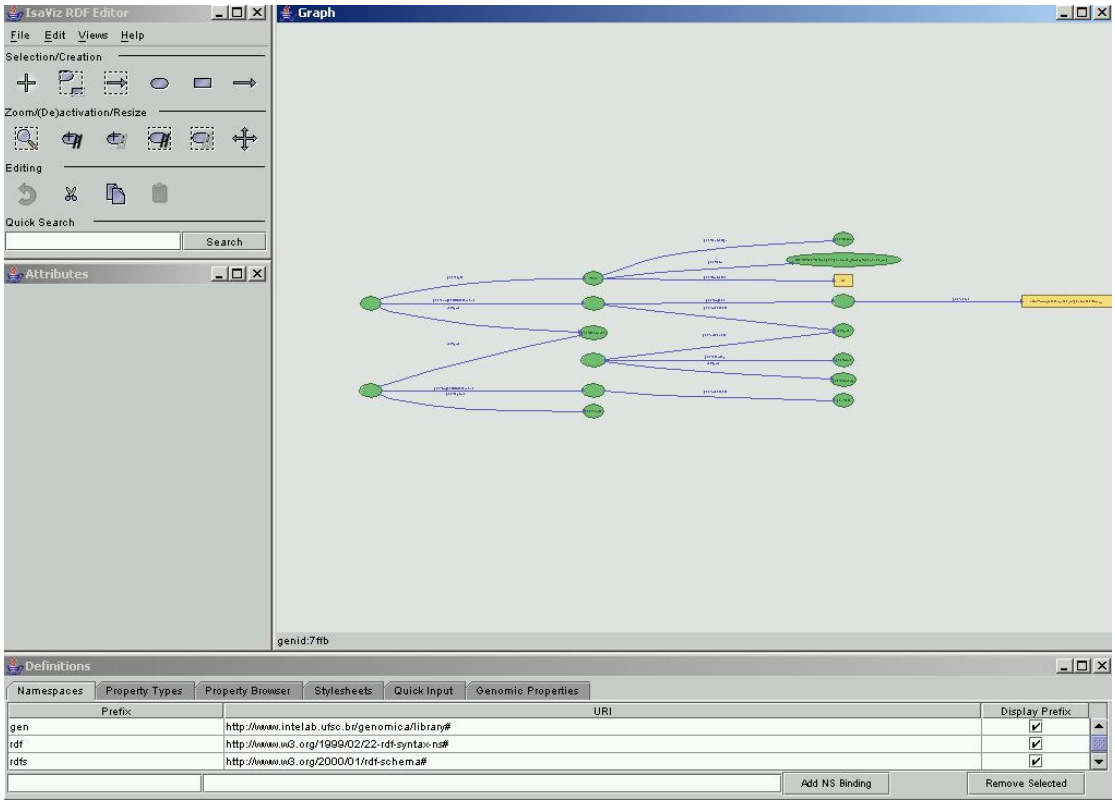

Figura 6 – Interface de desenvolvimento do software IsaViz. A janela principal, à direita, permite a construção dos modelos RDF. A versão original do IsaViz representa graficamente os recursos como elipses verdes, os literais como caixas amarelas e as propriedades através de arestas azuis.

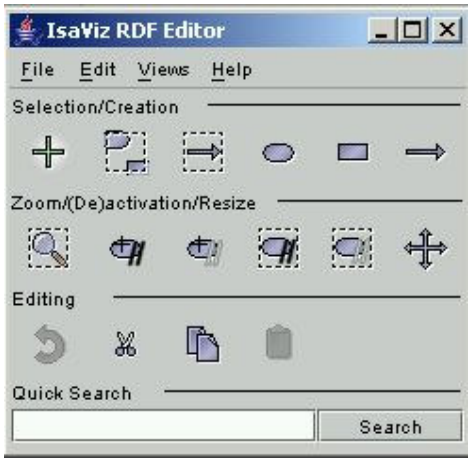

Figura 7 – Janela IsaViz RDF Editor – provê acesso aos menus principais e a uma paleta de ferramentas, que altera entre os modos de edição do sistema: inserção de recursos, inserção de propriedade, seleção de nodo, seleção múltipla e outros.

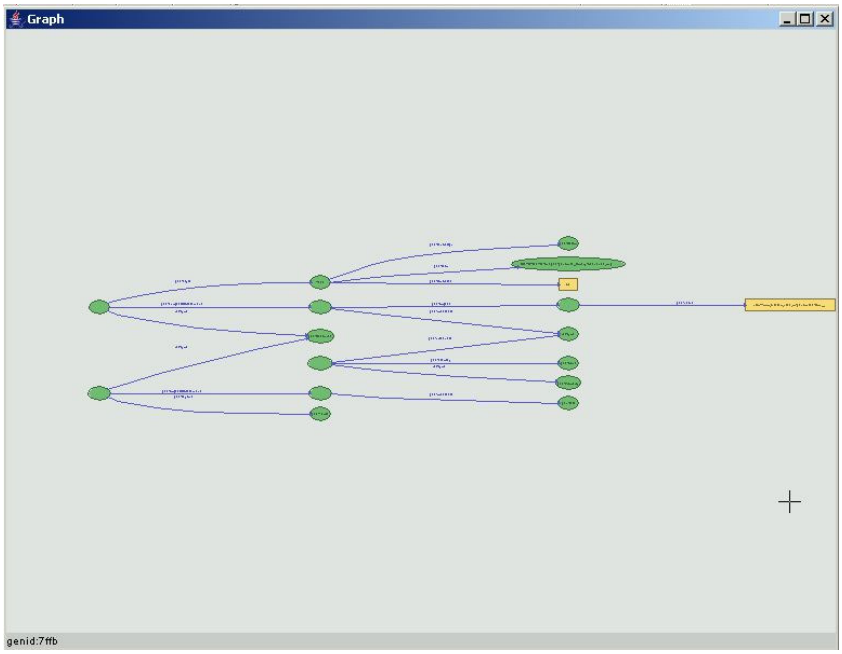

Figura 8 – A Janela *Graph* exibe o grafo RDF em 2D. Esta janela, construída sobre a biblioteca ZVTM, possibilita uma navegabilidade otimizada, oferecendo recursos de zoom sobre regiões específicas do grafo.

Figura 9 – Em *Attributes*, são mostradas informações sobre o item selecionado no grafo. No caso de recursos, é exibido a URI ou ID do nodo (quando não for *blank node*), além das suas propriedades e de uma opção para excluí-lo.

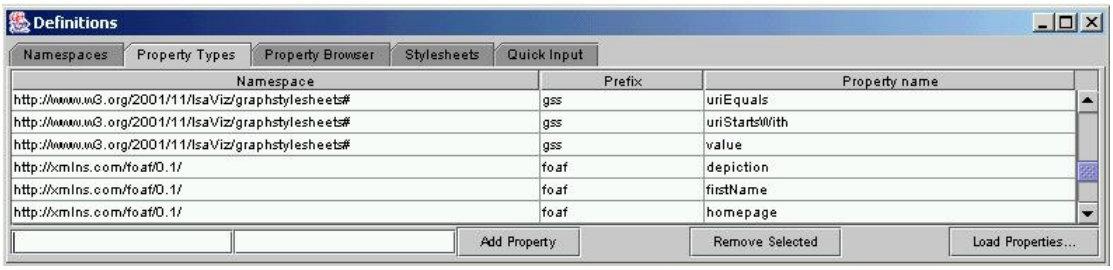

Figura 10 – A janela *Definitions* é composto por cinco abas, que desempenham uma variedade de funções.

A Figura 10, representada por suas abas, tem suas funções descritas abaixo:

- Aba *Namespaces*: contém os prefixos e URIs dos *namespaces* utilizados.
- Aba *Property Types*: contém as propriedades para cada um dos *namespaces* disponíveis para modelagem.
- Aba *Property Browser*: apresenta as propriedades do recurso selecionado no momento, com possibilidade de navegação entre elas.

- Aba *Stylesheets* – gerenciamento do(s) *stylesheet*(*s*) disponíveis para ser(em) aplicado(s) no grafo através dos comandos *Load*, *Apply*, *Remove*, *Edit*.
- Aba *Quick Input* – área de texto para inserção de *statements* em RDF/XML, N-Triples ou Notation 3.

2.3.1 – Arquivos de Projeto

O IsaViz possui um formato de arquivo chamado ".isv", em XML, que guarda informações não somente do modelo de dados, mas também da representação gráfica, como localização espacial dos elementos, regiões desativadas ou mesmo de elementos não visíveis no grafo. Este formato será abordado com mais detalhes à frente (vide seção 3.8).

Os arquivos ISV podem ser recuperados através da opção de menu "*Open Project*...". A partir da versão 2.0, informações de estilo de nodos e arestas também são armazenadas.

2.3.2 – Navegação no Grafo

A área de trabalho do IsaViz é infinita, isto é, o conceito de limitação por *scrollbars* não é aplicável. Sendo assim, a navegação no modelo é feita através de câmeras que podem ser movidas em todos os sentidos, afastando e aproximando do grafo.

Teclas de atalho facilitam a mobilidade pelo grafo. São elas:

- Setas: translação (cima, baixo, direta, esquerda);
- *Home*: visão global;
- *Page Up*/*Page Down*: *zoom out*/*zoom in*.

Outro recurso de visualização é o *Radar View* (Figura 11), que é acessado pelo menu *View* ou pressionando Ctrl+R, e fornece uma visão global do grafo. A janela de Radar contém um retângulo adicional que representa a região atualmente visualizada na janela principal. Este retângulo pode ser movido com o mouse ou com as setas do teclado.

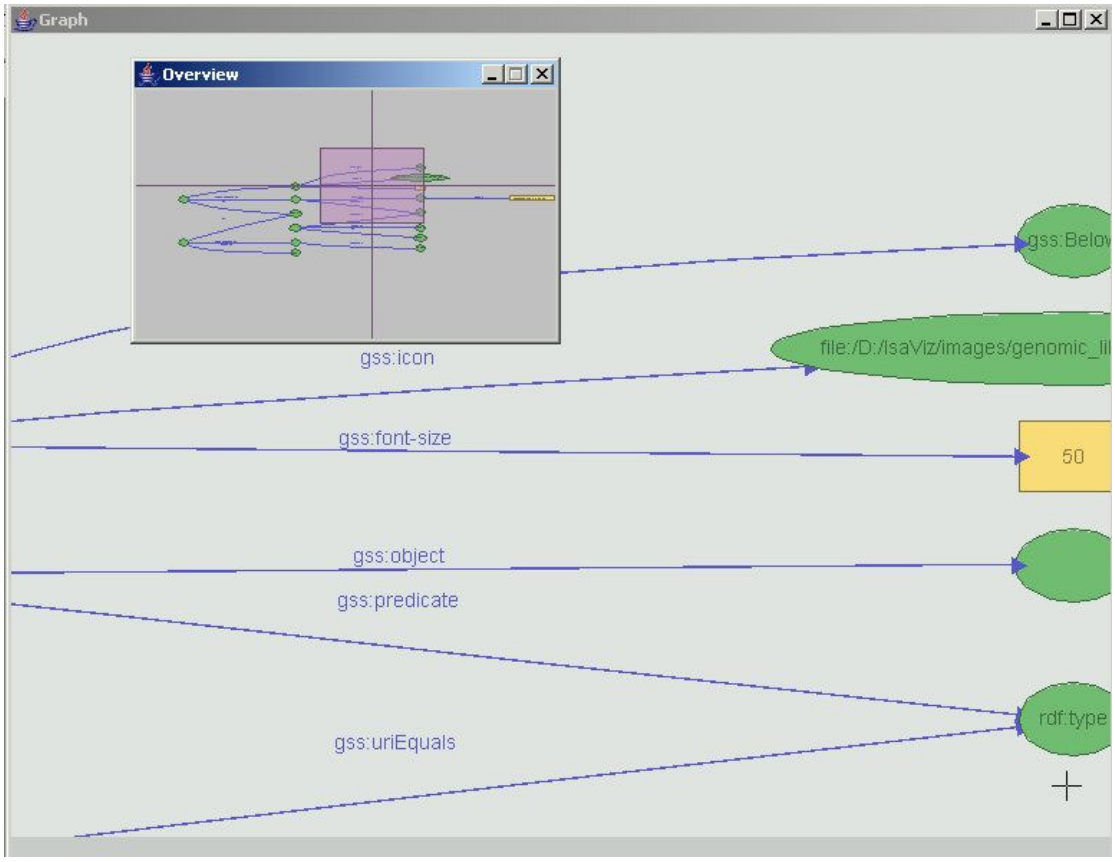

Figura 11 - O *Radar View* é um recurso de navegação bastante útil para modelos com uma grande quantidade de objetos, já que permite a localização rápida de regiões específicas do grafo RDF.

2.3.3 – Esquema de Funcionamento

Nesta seção vamos abordar o funcionamento interno do IsaViz, analisando suas principais classes e estruturas de dados. As classes que serão comentadas e que dizem respeito ao projeto lógico do IsaViz podem ser vistas no diagrama Figura 12.

A classe abstrata *INode* representa qualquer item do grafo e seus principais atributos denotam se o recurso está selecionado e/ou comentado e qual é o alinhamento do *label* para o item (atributo align).

As classes *IResource*, *IProperty* e *ILiteral* têm atributos e métodos específicos.

A classe *IResource* representa um recurso RDF, que pode ter o papel de *subject* ou *object* de uma asserção. O atributo `incomingPredicates` é um vetor de *IProperties* que armazena as propriedades que "chegam" a este nodo e o atributo `outgoingPredicates`, as propriedades que partem deste recurso. A `URI` é o nome atribuído ao recurso na sua criação e deve identificá-lo univocamente. Não existem dois recursos com a mesma URI em um mesmo grafo RDF. A *string* `classUri` identifica a que classe pertence o recurso, caso ele seja pertencente ao *namespace* gen.

O atributo `g11`, do tipo *Glyph*, representa a forma gráfica (elipse, triângulo, círculo, imagem, etc) que o nodo assume no espaço de visualização. Já o `g12`, da classe *VText*, é o *label* do recurso.

Um método importante desta classe é o `toISV()`. Sua função é escrever no arquivo .isv um elemento XML com as informações do nodo, tais como coordenadas

x e y, sua `URI`, `shape` (forma do Glyph) e outros atributos. É utilizada a API DOM para escrita e leitura destes arquivos XML.

```
<isv:resources>
   <isv:iresource
            display="true" fill="0" h="19" id="0" shape="icon"
            stroke="1" w="30" x="76"
            xlink:href="pathway_sample02_files/Chemicals.png" y="112">
            <isv:URIorID x="76" y="85">
           <isv:uri>http://www.intelab.ufsc.br/genomica/library#Anthranilate</isv:uri>
            </isv:URIorID>
   </isv:iresource>
...

<isv:resources>
```

A classe *IProperty* possui dois atributos essenciais, que são o `subject`, um *IResource* e `object`, da classe *INode*. O `object` é desta classe pois uma *statement* RDF admite como *object* tanto um recurso quanto um literal. Os atributos `namespace` e `localname` identificam a propriedade. Por exemplo, a URI `http://www.intelab.ufsc.br/genomica/library#Metabolic_Pathway` é composta pelo *namespace* `http://www.intelab.ufsc.br/genomica/library#` e pelo localname `Metabolic_Pathway`.

A classe *VPath* é do ZVTM e informa a posição gráfica da propriedade.

Já os objetos da classe *ILiteral* armazenam o valor, um `datatype`, uma referência à propriedade que o aponta e outros atributos semelhantes ao IResource. Por exemplo, um literal pode ter um *datatype* `http://www.w3.org/2001/XMLSchema#float` e ter o valor 20.56. Os *datatypes* são tipos da linguagem XML Schema.

A classe *ISVManager* tem as rotinas de salvamento e recuperação para os arquivos de projeto. Os métodos utilizam a API DOM para criação dos arquivos XML.

A classe Editor é a inicial do sistema e armazena referências para as janelas gráficas: `cmp`, `tblp`, `propsp`, `navp`. Os *Hashtables* `resourcesByUri` e

`propertiesByUri`, além do vetor dinâmico de literais são as estruturas de dados do grafo. No caso do *hash* `propertiesByUri`, dada uma certa URI, retorna-se um vetor, já que podemos ter a mesma propriedade várias vezes no modelo. No `resourcesByUri` são armazenados recursos, únicos para cada chave ou URI. O objeto `genLibrary`, da classe *GenomicLibrary*, é o responsável por efetuar o *parse* da Biblioteca Genômica expressa na linguagem RDFS e deixar em memória as classes, as propriedades e um *hash* com as propriedades de cada classe, dados estes que serão utilizados na inserção de recursos e de propriedades.

Estão também nestas classes os métodos de `export()` e `load()` para os vários formatos aceitos pelo IsaViz. Os métodos `createNewResource()` e `createNewProperty()` criam JDialogs para obtenção das informações pelo usuário.

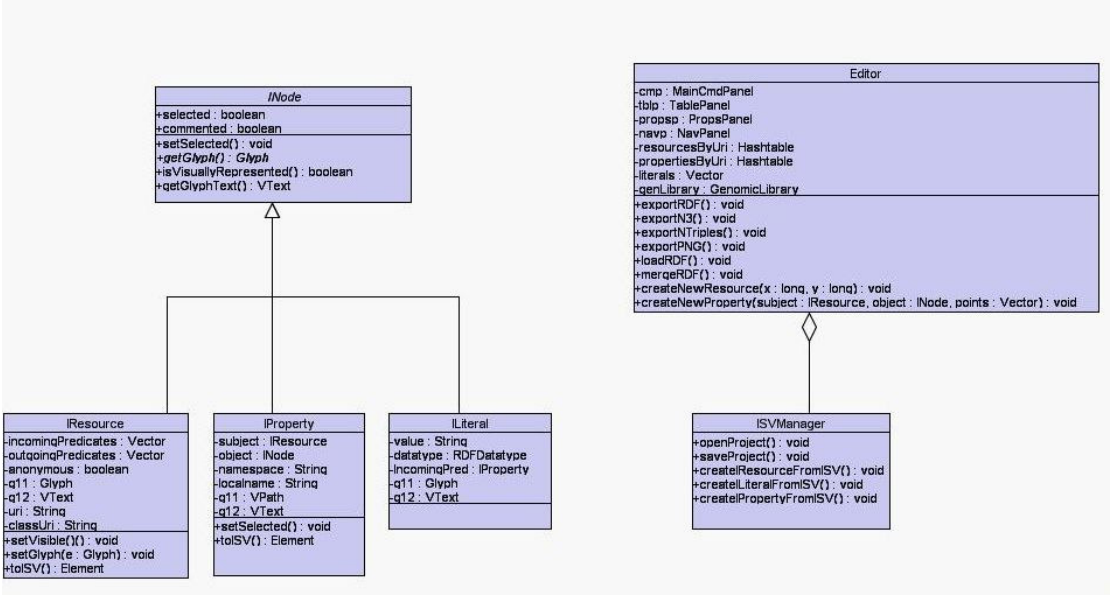

Figura 12 - Diagrama UML de classes do IsaViz que fazem parte do núcleo do sistema. Os comentários de métodos e atributos que aparecem nas classes são feitos no texto.

3 – Metodologia

Com o objetivo de possibilitar a modelagem de vias metabólicas e regulatórias, resultando no Metabolic IsaViz, foram feitas extensões na parte lógica e gráfica do sistema original. Nesta seção descreve-se como as tecnologias analisadas foram empregadas nas modificações propostas no sistema original.

3.1 – Biblioteca Genômica em RDF Schema

Para definição das classes e suas propriedades, utiliza-se um vocabulário RDF com termos do domínio genômico. A biblioteca estava originalmente disponibilizada em XML Schema e foi modelada com bases em ontologias e registros de bases de dados genômicas primárias, como o NCBI e o EMBL (VEIGA & PORTO, 2003). Procedeu-se a conversão para a linguagem RDF Schema.

As classes que estão na biblioteca referem-se à terminologia de seqüenciamento, como partes de uma seqüência (intron, exon, UTR), bem como à terminologia envolvida na modelagem de vias metabólicas (*products*, *chemicals*, *activators* entre outros). Uma tabela com os tipos da biblioteca pode ser observada na Figura 13. As classes e propriedades da biblioteca são utilizadas no Metabolic IsaViz no momento da inserção de recursos e de propriedades no modelo RDF. Na criação de um recurso (Vide Seção 3.3), é informada a que classe ele pertence; por exemplo, Glycolysis pertence à gen:Metabolic_Pathway. Já na criação de

propriedades (Vide Seção 3.5), a biblioteca auxilia a modelagem mostrando quais as propriedades que podem ser inseridas para um determinado recurso. Por exemplo, para um recurso da classe gen:Activator (um ativador de uma reação enzimática), a biblioteca sugere a inserção de propriedades como: mecanismo de ativação, descrição, fórmula estrutural e outras.

Conforme visto na Seção 2.2.2 – Padronização de vocabulários –, um esquema RDF é composto por classes e propriedades. Pode-se analisar, utilizando como exemplo o código da Tabela 11, como foram definidas as propriedades e classes da Biblioteca Genômica.

O primeiro elemento definido no código é uma propriedade, chamada gen:reaction_direction, que especifica o sentido da reação: reversível, irreversível, sem direção conhecida. Esta propriedade é aplicada à classe gen:Enzymatic_Data (rdfs:domain) e seu valor é uma *string* (rdfs:range). O elemento a seguir também é uma propriedade, a gen:Description. Verifique que ela é reutilizada em várias classes tais como Gene, Protein, Chemicals, Metabolic_Pathway. O último elemento é uma classe, a gen:ChromosomeFrag, que é subclasse de gen:Sequence.

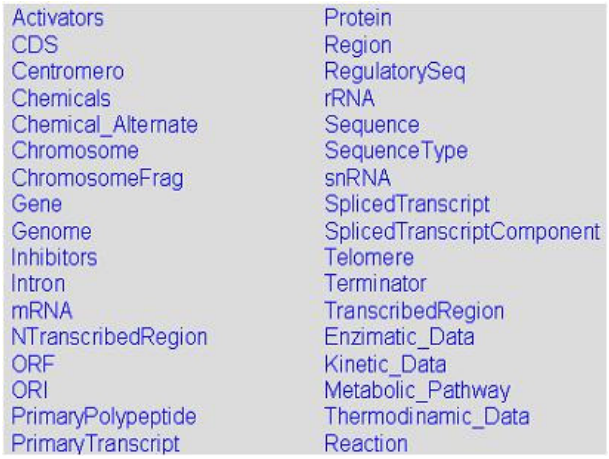

Figura 13 - Classes da Biblioteca Genômica, base para a criação dos modelos biológicos com o Metabolic IsaViz.

Tabela 11- Fragmento que compõe a Biblioteca Genômica.

De acordo com a sintaxe de RDF/XML, cada elemento rdf:description (destacado em negrito) define um elemento, cujo tipo vai ser determinado pela propriedade rdf:type. Neste trecho de código, temos a definição de duas propriedades e de uma classe, a *ChromosomeFrag*.

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
          …

<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#reaction_direction">
   <rdfs:comment>Property Mechanism - This property must be restricted to the follow values: reversible,
   irreversible, physiologically-unidirectional.</rdfs:comment>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Enzymatic_Data"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Description>

<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Description">
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Gene"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#PrimaryPolypeptide"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Protein"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Genome"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Metabolic_Pathway"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Description>

<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#ChromosomeFrag">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://www.intelab.ufsc.br/genomica/library#Sequence"/>
 </rdf:Description>
                        ...
</rdf:RDF>
```

3.2 – A Classe GenomicLibrary

A classe *GenomicLibrary* é responsável por efetuar o *parsing* da Biblioteca Genômica e criar as estruturas de dados específicas que serão disponibilizadas ao Metabolic IsaViz. Esta foi a única classe adicionada ao IsaViz original, já que as outras alterações foram em classes já existentes.

Os atributos e métodos da *GenomicLibrary* aparecem na Figura 14. Os atributos da classe têm o seguinte conteúdo:

- *HashMap* `class_prop`: guarda as propriedades de cada classe, a chave é a URI da classe;
- *HashMap* `class_subclass`: relaciona classe e subclasse, a chave é a URI da classe;
- *HashMap* `prop_value`: armazena o valor que cada propriedade pode assumir (rdf:range), a chave é a URI da propriedade;
- *ArrayList* `all_properties`: lista de todas as propriedades da Biblioteca Genômica.

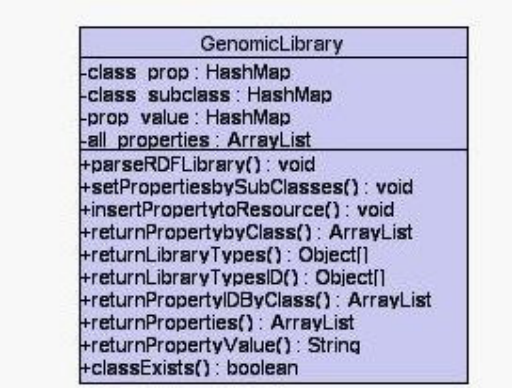

Figura 14 - Diagrama UML da classe GenomicLibrary, responsável pela interpretação do código RDF/XML da Biblioteca Genômica.

O *parsing* é realizado utilizando métodos oferecidos pela Jena API, uma biblioteca de manipulação de arquivos RDF/XML. O método principal da *GenomicLibrary* é o `parseRDFLibrary()`, que monta as estruturas de dados, de acordo com o que foi comentado acima. O trecho de código da Tabela 12 mostra uma parte do método `parseRDFLibrary()`. Para identificar as classes, criou-se um iterador sobre recursos, com a restrição de que estes recursos tenham a propriedade rdf:type, cujo valor seja rdfs:class. Isto pode ser observado na linha 10.

A partir daí, é verificado se o recurso é subclasse de algum outro (linha 18) para preencher corretamente o *Hashmap* `class_subclass` (linha 20).

Tabela 12 - Trecho de código do método parseRDFLibrary() da classe GenomicLibrary.

O método `parseRDFLibrary()` da classe *GenomicLibrary* monta as estruturas de dados disponíveis para o Metabolic IsaViz.

```
1. public void parseRDFLibrary () {
2.     try {
3.          // create an empty model
4.          Model model = ModelFactory.createDefaultModel();
5.
6.         FileReader freader=new FileReader(inputFileName);
7.         RDFReader parser = model.getReader("RDF/XML");
8.         parser.read(model,freader,"");
9.
10.          ResIterator iter = model.listSubjectsWithProperty(RDF.type,RDFS.Class);
11.          Resource aux;
12.          Statement subClass_stmt;
13.
14.          if (iter.hasNext()) {
15.             while (iter.hasNext()) {
16.                aux = iter.nextResource();
17.
18.                if (aux.hasProperty(RDFS.subClassOf)){
19.                   subClass_stmt = aux.getProperty(RDFS.subClassOf);
20.                   class_subclass.put(aux.toString(),subClass_stmt.getObject().toString());.
21.                   class_prop.put(aux.toString(),new ArrayList());
22.                }else {
23.                   class_subclass.put(aux.toString(),"");
24.                   class_prop.put(aux.toString(),new ArrayList());
25.                   //System.out.println(aux);
26.                }
27.             }
28.          } else {
29.             System.out.println("No classes were found in the database");
30.          }
…
```

Os outros métodos da classe, como `returnPropertyByClass()` e `returnLibraryTypes()`, foram criados para facilitar o acesso aos dados pelas várias janelas do sistema.

3.3 – Inclusão de Recurso

Para os recursos que são inseridos sob o *namespace* gen, ou seja, que referenciam uma classe da biblioteca, o procedimento de inclusão no grafo foi alterado. A Figura 15 mostra a janela de inclusão do IsaViz original, enquanto a Figura 16 mostra como ficou o *dialog* de inserção de recursos no Metabolic IsaViz. A diferença entre as duas é que, no sistema original, informa-se somente a URI ou ID para o novo recurso, enquanto no Metabolic IsaViz informa-se, além da URI, a classe do recurso. Este procedimento também insere automaticamente a propriedade rdf:type no recurso criado, evitando que o usuário insira esta propriedade toda vez que criar um recurso da biblioteca (Veja Figuras 17 e 18) .

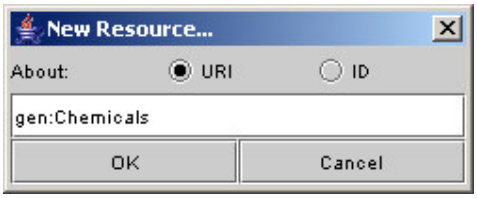

Figura 15 - Criação de recurso no IsaViz original.

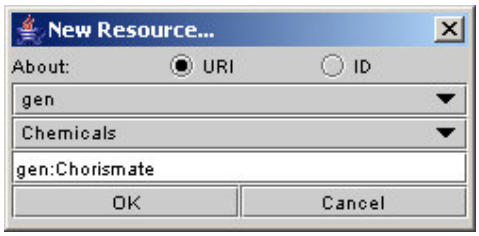

Figura 16 - Criação de recurso no Metabolic IsaViz.

Ainda na Figura 16, o primeiro ComboBox contém a lista de todos os *namespaces* disponíveis (neste caso, está selecionado o *namespace* "gen", prefixo de `http://www.intelab.ufsc.br/genomica/library#`). Quando o prefixo da biblioteca for selecionado, o segundo ComboBox é habilitado, contendo todas suas classes. No exemplo desta Figura, o recurso com URI gen:Chorismate pertence ao *namespace* "gen" e à classe *Chemicals*. O grafo resultante, com apenas 1 (um) elemento, pode ser visto na Figura 18.

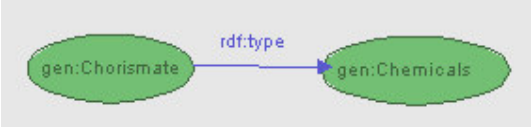

Figura 17 - Grafo RDF no IsaViz original. A propriedade rdf:type indica a classe do recurso gen:Chorismate.

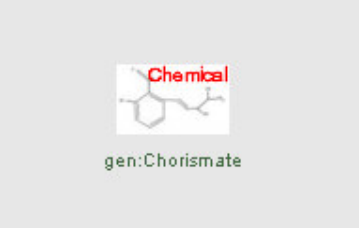

Figura 18 - Grafo RDF no Metabolic IsaViz. Para melhorar a visualização, as propriedades rdf:type são ocultadas e o ícone representa a classe.

As Figuras 17 e 18 representam o mesmo RDF. Com o Metabolic IsaViz, os três passos que o usuário tinha que executar para criar o grafo da Figura 17 – criar o recurso gen:Chorismate, o recurso gen:Chemicals e a propriedade rdf:type – foram diminuídos para apenas um. A propriedade rdf:type e o recurso que indica a classe continuam existindo, mas passam a não ser visíveis e o ícone indica a classe do recurso.

3.4 – Visualização das Propriedades de Classe

Após se inserir um recurso é necessário conhecer quais propriedades ele possui. Para solucionar este aspecto, na janela *Definitions* foi adicionada a aba *Genomic Properties*. Quando o recurso selecionado no grafo pertencer à Biblioteca Genômica, esta aba irá exibir todas as propriedades da classe à qual o recurso pertence, listando o *namespace*, classe, descrição da propriedade e o valor que ela pode assumir.

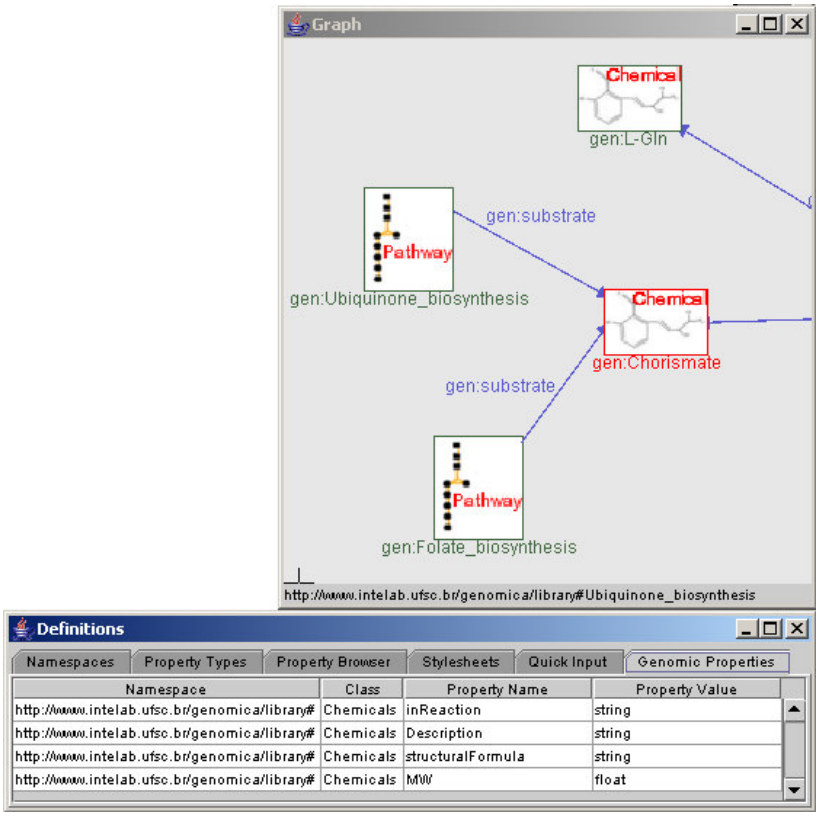

Figura 19 - Efeito da seleção de um recurso da biblioteca na aba *Genomic Properties*.

Na Figura 19, o recurso gen:Chorismate aparece selecionado (em destaque); por conseguinte, a aba *Genomic Properties* mostra o *namespace* (das propriedades), o nome da classe, *Chemicals*, suas propriedades, *inReaction*, *Description*, *structuralFormula*, *MW* e o valor de cada uma delas, *string* e *float*.

3.5 – Inclusão de Propriedade

Na inserção de propriedades, uma importante modificação foi implementada para facilitar a criação dos modelos. As propriedades que podem ser inseridas para o recurso selecionado, considerando a sua classe, são mostradas na janela *New*

Statement. Pela Figura 20, o recurso gen:Aspartate_biosynthesis, que é uma via metabólica, possui as seguintes propriedades aplicáveis: product, reaction, Description, global_reaction, synonymous. Para formar uma *statement* com o recurso gen:L-aspartate, a escolha seria product, resultando no grafo da Figura 21.

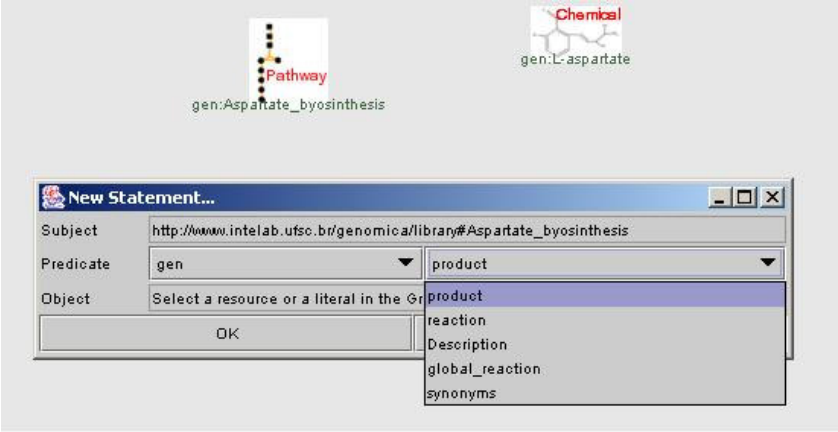

Figura 20 - Inserção de propriedades no Metabolic IsaViz. As propriedades de cada recurso aparecem ao usuário, de acordo com o que foi definido na Biblioteca Genômica.

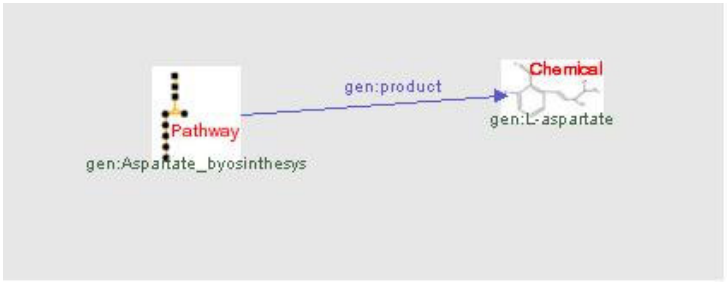

Figura 21 - Resultado da inserção da propriedade gen:product ligando os recursos gen:Asparte_biosynthesis e gen:L-aspartate.

3.6 – Lista das Propriedades

Na janela *Definitions*, aba *Property Types*, que contém a lista de propriedades disponíveis, todas as propriedades presentes na biblioteca foram inseridas.

3.7 – Inclusão do *Namespace*

O namespace `http://www.intelab.ufsc.br/genomica/library#`, com prefixo gen, que representa a Biblioteca Genômica, foi incluído na janela *Definitions*, aba *Namespaces.* A partir de então, todos os recursos e propriedades que tiverem o prefixo “gen:” pertencerão à biblioteca.

3.8 – Armazenamento do Modelo

Quando os recursos vão ser armazenados, os atributos da classe `IResource` são analisados e um pequeno resumo dessas informações é montado para que, no futuro, o recurso seja “recriado”. Para facilitar o processo de recuperação do grafo RDF, o atributo classUri (que indica a classe do recurso) foi adicionado a este resumo.

Outra alteração está associada ao armazenamento, junto ao modelo, de todas as imagens utilizadas no grafo. No IsaViz original, as imagens eram salvas em arquivos com uma numeração seqüencial e sem a preocupação de repetição, ou seja, imagens iguais, se utilizadas mais de uma vez, eram salvas, repetidamente, o número de vezes que aparecessem no grafo. No Metabolic IsaViz, a imagem é salva apenas uma vez e todos os recursos que utilizaram a mesma imagem têm uma referência a tal arquivo. Desta maneira, o espaço utilizado para o armazenamento de um modelo diminui consideravelmente, facilitando, por exemplo, o transporte destes arquivos.

```
92  <isv:iresource
93     classUri="http://www.intelab.ufsc.br/genomica/library#Chemicals"
94     display="true" fill="0" h="19" id="0" shape="icon"
95     stroke="1" w="30" x="76"
96     xlink:href="pathway_sample02_files/Chemicals.png" y="112">
97     <isv:URIorID x="76" y="85">
98        <isv:uri>http://www.intelab.ufsc.br/genomica/library#Anthranilate</isv:uri>
99     </isv:URIorID>
100 </isv:iresource>
```

Figura 22- Trecho do arquivo de armazenamento de um modelo RDF.

A Figura 22 mostra o trecho do arquivo de armazenamento de um modelo RDF (.isv) que representa um recurso existente no grafo. Na linha 93, aparece a primeira alteração, a adição do atributo *classUri* indicando que o recurso pertence à classe *Chemicals*. Na linha 96, o atributo do elemento *xlink:href* fornece a localização do arquivo que contém a imagem associada a este recurso, neste caso `pathway_sample02_files/Chemicals.png`.

A partir das modificações discutidas acima, o software se mostrou pronto para iniciar a modelagem e criação dos grafos para representação das vias, incluindo seus aspectos de regulação e de cinética enzimática, como será visto a seguir.

4 – Resultados

4.1 – Mecanismo de Criação de Modelos

A modelagem das vias metabólicas com o Metabolic IsaViz se baseia na criação dos recursos (nodos do grafo), informando a que classe pertencem, e atribuindo-lhes propriedades, que graficamente aparecem como as arestas do grafo. Quando é informada a classe do recurso, o sistema inclui no espaço de visualização um ícone que indica a classe escolhida.

O processo de modelagem supõe o conhecimento da Biblioteca Genômica e do domínio das vias bioquímicas. Pelo que foi constatado, uma dificuldade é saber como combinar os recursos da biblioteca, como um composto químico (Chemicals) e uma via (Metabolic_Pathway). Para transpor isto, o usuário pode utilizar a Aba *Genomic Properties* da janela *Definitions* (Vide Seção 3.4) e visualizar quais as propriedades que são aplicáveis à classe.

Além disto, junto com o software estão sendo distribuídos modelos-base, que mostram como representar uma reação, uma seqüência de reações (via metabólica) ou mesmo uma reação com parâmetros cinéticos e enzimáticos.

4.2 – Modelos RDF Produzidos

Com o objetivo de testar a eficácia na modelagem de vias metabólicas e regulatórias com o auxílio da Biblioteca Genômica e com as adaptações que geraram o Metabolic IsaViz, foram criados modelos RDF que serão detalhados a seguir. A análise destes modelos permitirá elucidar o mecanismo de modelagem comentado acima.

4.2.1 – Representação de uma Reação

A Figura 23 mostra o modelo RDF gerado com o Metabolic IsaViz para representar um passo de uma via metabólica, ou seja, uma reação enzimática. O recurso central é o gen:2.5.1.54, que é o objeto que representa a reação. Deste recurso saem duas propriedades gen:left, que representam os substratos: os compostos gen:D-Erythrose-4-phosphate (E4P) e o gen:Phosphoenol-pyruvate (PEP). O produto da reação é o recurso identificado por gen: 7P-2-Dehydro-3-deoxy-D-arabino-heptonate, e é indicado com a propriedade gen:right. Estas são as informações mínimas para se construir uma reação.

Este modelo também denota como representar intersecções entre vias. O recurso E4P é o produto final da via glicolítica, e isto é indicado pela propriedade gen:product que parte de gen:glycolysis.

4.2.2 – Uma Reação com Dados sobre a Enzima

Neste próximo modelo mostra-se como uma reação pode conter informações sobre a enzima catalisadora, através do recurso Enzymatic_Data. A reação apresentada na Figura 24 é a gen:4.1.3.27, e transforma o Corismato + L-Gln em Piruvato + L-glutamato + Antranilato. Isto pode ser observado pelas propriedades gen:left e gen:right do objeto da classe Reaction.

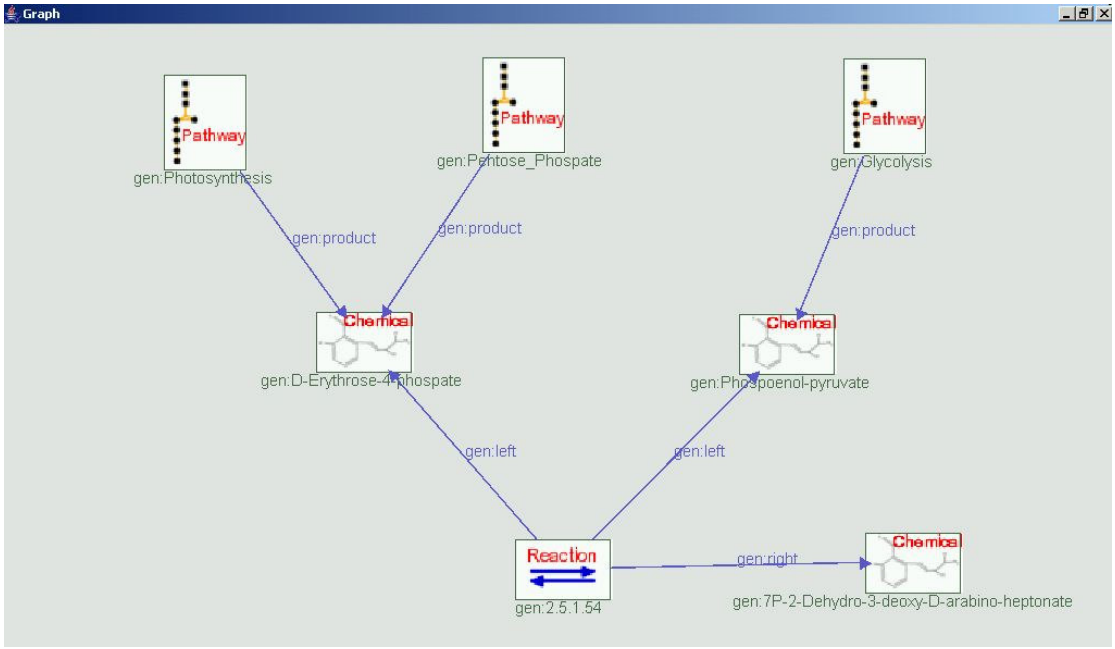

Figura 23 - Representação de uma reação enzimática no Metabolic IsaViz.

As informações sobre a enzima são adicionadas com a propriedade gen:catalyst_parameters, cujo valor é um recurso Enzymatic_Data. Esta classe, de acordo com o nome, tem informações sobre a atividade enzimática tais como a enzima, cofatores, ativadores, inibidores, grupos prostéticos e sentido da reação. Na Figura 24, incluiu-se a propriedade gen:enzyme que leva para a enzima antranilato sintase, representada pelo ícone da classe Protein (exibe uma proteína em sua

conformação tridimensional). A propriedade gen:inhibitor_elem, que é o composto L-Triptofano, também é utilizada. Por fim é adicionada a seqüência de aminoácidos da enzima por meio da propriedade gen:sequence da classe Protein.

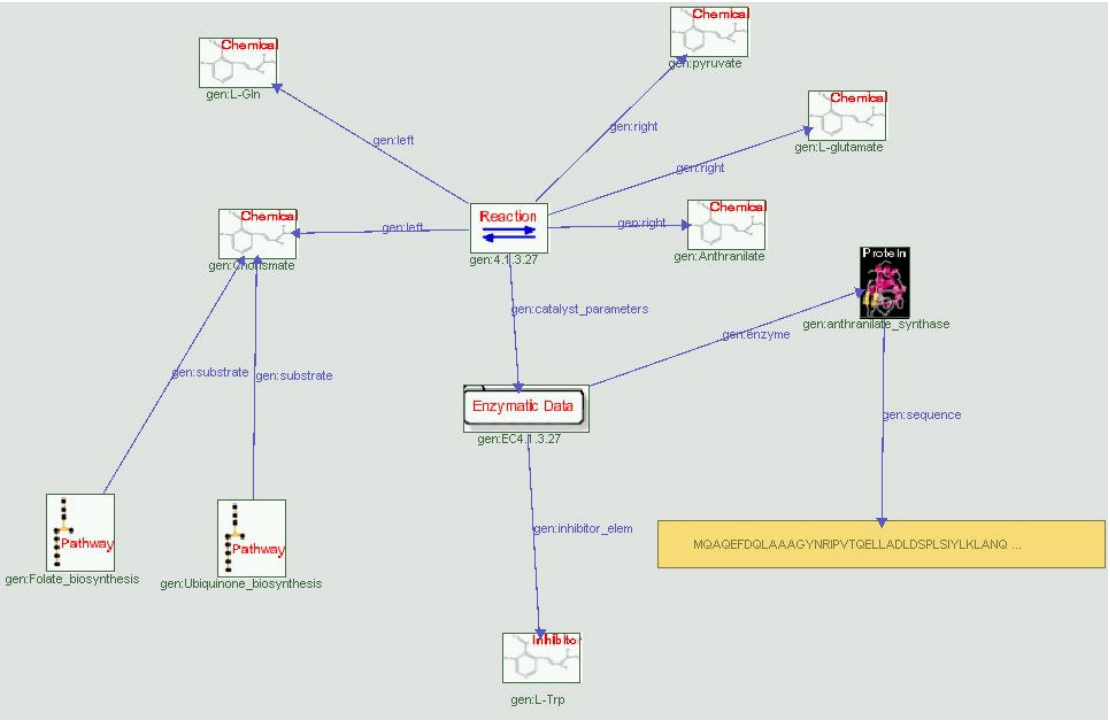

Figura 24 - Esquema de representação de uma reação com dados sobre a enzima catalisadora.

4.2.3 – Uma Reação Englobando Parâmetros Enzimáticos e Cinéticos

Nas Figuras 25 e 26, apresenta-se um modelo de reação que engloba os dados funcionais ou cinéticos e os dados sobre a enzima e seus ligantes, respectivamente.

Na primeira Figura, para adicionar parâmetros de cinética enzimática utiliza-se um recurso da classe Kinetic_Data. O gen:functional_param possui as seguintes propriedades: gen:pH_optimum, gen:Ki, gen:Turnover_number, gen:Specific_activity e gen:Km. Estes parâmetros podem ser utilizados na definição das equações cinéticas, como de Michaelis-Menten e são adicionados ao modelo quando o propósito for de simulação.

A Figura 26 mostra como representar dados sobre os ligantes que participam da reação química. São utilizadas as propriedades gen:inhibitor_elem para denotar os inibidores da reação além das propriedades gen:activators_elem, gen:prosthetic_group e gen:reaction_direction, que fornecem mais detalhes do conhecimento biológico desta reação.

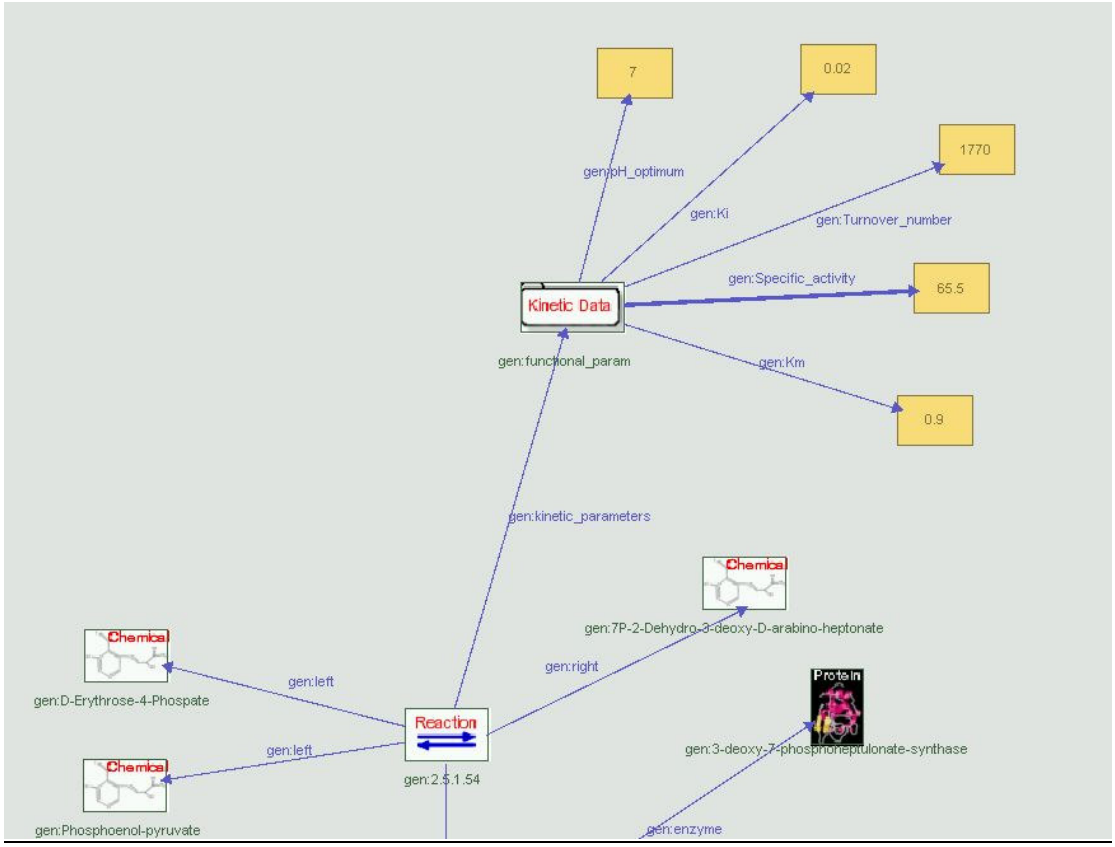

Figura 25 - Dados cinéticos da reação podem ser modelados com um recurso da classe Kinetic_Data. Neste exemplo, parâmetros de pH ótimo, constante de velocidade (Km), de inibição (Ki) entre outros são informados (veja as propriedades que partem do recurso gen:functional_param, classe Kinetic_Data.

O RDF/XML gerado pelo modelo discutido acima encontra-se na Figura 27. Da linha 37 a 47 ocorre a descrição do recurso gen:edata, da classe Enzymatic_Data. Como pode-se ver, cada propriedade é identificada por elemento sob o elemento rdf:description: gen:inhibitor_elem, gen:enzyme, gen:prosthetic_Group, gen:activator e gen:reaction_direction. Com o mesmo mecanismo é descrito o recurso gen:functional_param, entre as linhas 51 e 58.

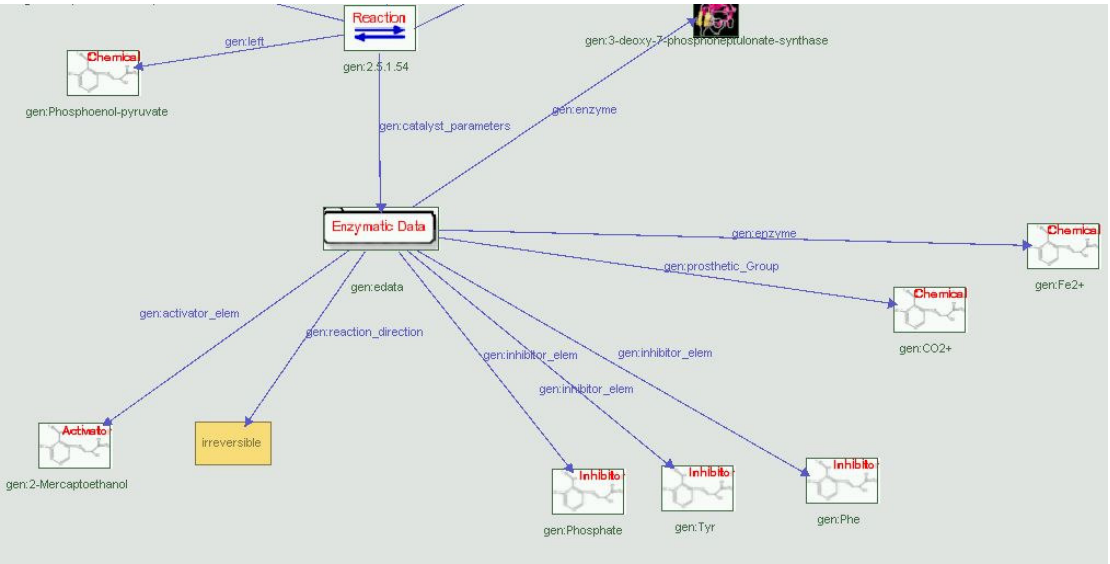

Figura 26 - Representação de ativadores, inibidores e grupos prostéticos que fazem parte da reação 2.5.1.54, e são incluídos por meio do recurso gen:edata, da classe Enzymatic_Data.

```
30    </rdf:Description>
31    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#C02+">
32      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
33    </rdf:Description>
34    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Fe2+">
35      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
36    </rdf:Description>
37    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#edata">
38      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Phosphate"/>
39      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Tyr"/>
40      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Phe"/>
41      <gen:enzyme rdf:resource="http://www.intelab.ufsc.br/genomica/library#Fe2+"/>
42      <gen:enzyme rdf:resource="http://www.intelab.ufsc.br/genomica/library#3-deoxy-7-phosphoheptulonate-synthase
        "/>
43      <gen:prosthetic_Group rdf:resource="http://www.intelab.ufsc.br/genomica/library#C02+"/>
44      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Enzymatic_Data"/>
45      <gen:activator_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#2-Mercaptoethanol"/>
46      <gen:reaction_direction
        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">irreversible</gen:reaction_direction>
47    </rdf:Description>
48    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Tyr">
49      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Inhibitors"/>
50    </rdf:Description>
51    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#functional_param">
52      <gen:Ki rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.02</gen:Ki>
53      <gen:Turnover_number rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1770</gen:Turnover_number>
54      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Kinetic_Data"/>
55      <gen:pH_optimum rdf:datatype="http://www.w3.org/2001/XMLSchema#float">7 </gen:pH_optimum>
56      <gen:Specific_activity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">65.5</gen:Specific_activity>
57      <gen:Km rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.9</gen:Km>
58    </rdf:Description>
59    <rdf:Description
      rdf:about="http://www.intelab.ufsc.br/genomica/library#7P-2-Dehydro-3-deoxy-D-arabino-heptonate">
60      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
61    </rdf:Description>
62  </rdf:RDF>
```

Figura 27 - Trecho do código RDF/XML do modelo de uma reação com informações sobre seus parâmetros cinéticos e enzimáticos. As propriedades ocorrem como elementos aninhados sob o rdf:description.

4.2.4 – Representação de Regulação Transcricional e por Produto

A Biblioteca Genômica disponibiliza ainda classes para representação de alguns níveis de regulação. A Figura 28 mostra o modelo da reação gen:2.3.1.46 envolvendo estes aspectos. A regulação por produto (ou *feedback regulation*), que ocorre entre produtos intermediários de uma mesma via ou de vias diferentes, é representada no grafo pelo recurso gen:reg. A propriedade gen:inhibited_By, que parte deste recurso, especifica qual é o composto que regula a produção de gen:HSCoA e gen:alpha-succinyl-L-homoserine. A classe Regulation também possui a propriedade gen:activated_By.

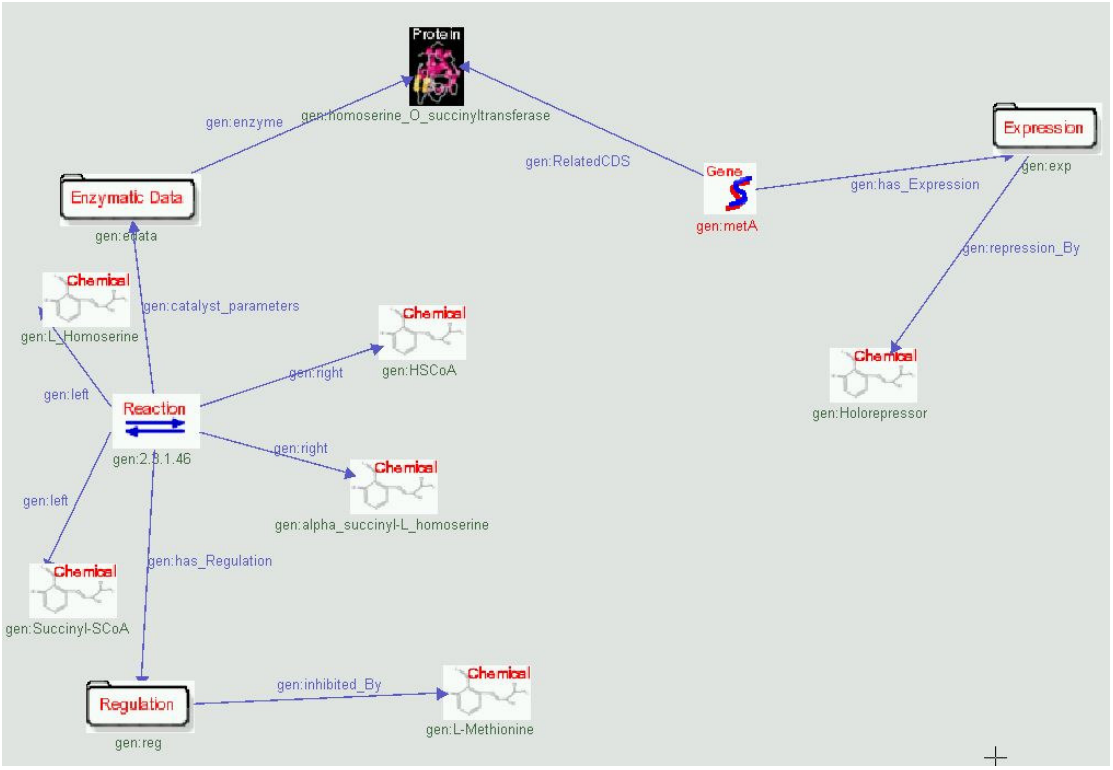

Figura 28 - Representação da regulação a nível de transcrição e entre metabólitos intermediários.

Outro tipo de regulação representado no modelo é a regulação a nível de transcrição. Isto ocorre com a repressão da transcrição do gene metA, realizada por um composto conhecido como Holorepressor. Para esta forma de regulação utiliza-se um recurso da classe Expression combinado com a propriedade gen:repression_By. Esta classe também possui as propriedades gen:activated_By e gen:inhibited_By.

5 – Conclusões e Trabalhos Futuros

A extensão do IsaViz implementada neste trabalho pode ser caracterizada por dois itens: a alteração na representação gráfica dos grafos, através de ícones que lembram o tipo do recurso, e a integração com um vocabulário RDF, denominado de *Genomic Library*, com base em uma ontologia no domínio da informação genômica. Notou-se que restringindo um ambiente de modelagem RDF genérico, como o IsaViz, o processo de criação dos modelos foi otimizado. Desta forma, um IsaViz orientado a bibliotecas, como o Metabolic IsaViz, que indica as classes e as propriedades que os recursos podem assumir, torna mais fácil a representação de dados usando o RDF. Além disso, quando houver quaisquer modificações na biblioteca, o software automaticamente estará atualizado, dinamizando e simplificando o trabalho.

Como abordado, a idéia que fundamentou este projeto foi a de utilizar o modelo de dados RDF, um multigrafo direcionado, para a representação do conhecimento acerca das vias metabólicas e seus diferentes aspectos. Este objetivo foi plenamente alcançado, uma vez que os modelos gerados pelo Metabolic IsaViz conseguem expressar dados sobre a cinética enzimática, mecanismos de regulação e interações entre enzima e seus ligantes. Pretende-se que este software seja utilizado como uma ferramenta de análise de vias de interesse, combinado com programas de simulação baseados em equações diferenciais, técnicas de análise em Engenharia Metabólica ou mesmo com simuladores baseados em RPH (Redes de Petri Híbridas). Logo, o Metabolic IsaViz desempenhará um papel de *front-end*, já

que através da criação do grafo e de sua representação em XML irá coletar os dados necessários para os ensaios de simulação.

De forma geral, adquiriu-se um conhecimento profundo sobre o RDF, suas ferramentas e APIs de manipulação, além da criação de ontologias em RDF. Também é preciso destacar o trabalho de engenharia reversa do IsaViz, pois a única documentação do software era o seu próprio código. Desta forma, desenvolveu-se um software de grande aplicabilidade prática, baseado em tecnologias extremamente recentes e espera-se estar contribuindo para o desenvolvimento da Web Semântica e das pesquisas na área pós-genômica. Atualmente, um processo de conversação com a W3C está em andamento para disponibilizar o software como um produto de código-livre pelos servidores desta instituição. Uma página web está disponível (versão português e inglês) com as informações deste projeto e software para download: **`http://www.intelab.ufsc.br/isaviz.`**

Como trabalhos futuros, sugere-se que o Metabolic IsaViz exporte seus modelos na linguagem de representação padrão para vias metabólicas, a SBML (*Systems Biology Markup Language*) (SBML, 2003), o que tornaria de fato o software deste projeto possível de ser utilizado em escalas maiores. Outro benefício que advém desta característica seria que os modelos de vias gerados com o Metabolic IsaViz poderiam ser acoplados diretamente com softwares de simulação.

Além disso, novas ontologias podem ser integradas, como é o caso da *Gene Ontology* (Gene Ontology, 2003), cuja finalidade é recomendar os nomes de compostos químicos e suas funções. Por fim, outra melhoria importante seria implementar um gerenciador de biblioteca internamente ao software, evitando a edição manual da Biblioteca Genômica.

6 – Referências Bibliográficas

ARACYC – Bases de dados para a espécie. Disponível em: <www.arabidopsis.org>. Acesso em: 28 jun. 2003.

SBML – Systems Biology Markup Language. Disponível em: <www.sbml.org>. Acesso em: 28 jan. 2004.

BERNERS-LEE, Tim, HENDLER, James, LASSILA, Ora. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. **Scientific American**, [S.l.], mai. 2001. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2>. Acesso em: 14 fev. 2003.

BRICKLEY, Dan, GUHA, R. V. **RDF Vocabulary Description Language 1.0: RDF Schema**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em: 14 fev. 2003.

CONNOLY, Dan et. al. DAML+OIL (March 2001) Reference Description. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/daml+oil-reference />. Acesso em: 26 jan. 2004.

DUBLIN CORE Metadata Initiative. Disponível em: <http://dublincore.org>. Acesso em: 12 dez. 2003.

Gene Ontology. Disponível em: <www.geneontology.org>. Acesso em: 28 jan. 2004.

KEGG - **Kyoto Encyclopaedia of Genes and Genomes** - Metabolic pathways Disponível em: <http://www.genome.ad.jp/kegg/>. Acesso em: 28 jun. 2003.

KLYNE, Graham, CARROLL, Jeremy J. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em: 14 fev. 2003.

MANOLA, Frank, MILLER, Eric. **RDF Primer**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-primer/>. Acesso em: 14 fev. 2003.

PIETRIGA, Emmanuel. Software. **IsaViz: A Visual Authoring Tool for RDF**. IsaViz 2.0 alpha - preview release (2003-05-12). Disponível em <http://www.w3c.org>. Acesso em: 28 jun. 2003.

PIETRIGA, Emmanuel. **IsaViz User Manual**. IsaViz 2.0 released (2003-08-08). Disponível em <http://www.w3.org/2001/11/IsaViz/usermanual.html>. Acesso em: 16 jan. 2004.

PIETRIGA, Emmanuel. **Graph Stylesheets (GSS) User Manual**. 2003. Disponível em <http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html>. Acesso em: 16 jan. 2004.

SMITH, Michael K. et al. **OWL Web Ontology Language Guide**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em: 26 jan. 2004.

van HELDEN, Jacques et. al. **Representing and analysing molecular and cellular function in the computer**. Biol Chem 381(9-10), 921-35, 2000.

VEIGA, Diogo F., PORTO, Luismar M. **XML Schema Representation as a Way to Interchange and Customize Genomic and Metabolic Models.** International Conference on Bioinformatics and Computational Biology, Ribeirão Preto, SP, Brasil, mai. 2003.

ANEXO 1 – Parte do Código-Fonte do Metabolic IsaViz (classes fundamentais do sistema)

Classe Inode

```
/*   FILE: INode.java
 *   DATE OF CREATION:   10/18/2001
 *   AUTHOR :            Emmanuel Pietriga (emmanuel@w3.org)
 *   MODIF:              Wed Jul 09 11:06:26 2003 by Emmanuel Pietriga
(emmanuel@w3.org, emmanuel@claribole.net)
 *   MODIF:              Tue Dec 9 2003 by Diogo Veiga/Pedro Cecconello
 */

/*
 *
 *  (c) COPYRIGHT World Wide Web Consortium, 1994-2001.
 *  Please first read the full copyright statement in file copyright.html
 *
 */
package org.w3c.IsaViz;

import com.xerox.VTM.glyphs.Glyph;
import com.xerox.VTM.glyphs.VText;

/*Parent of IResource, IProperty, ILiteral*/

abstract class INode {

    /*is the entity selected in the GUI*/
    boolean selected=false;
    /*is the node/property deactivated*/
    boolean commented=false;

    /*is the node in a table form (for resources, this is from the point of
view of resource as objects)*/
    boolean table=false;

    private int align=Style.TA_CENTER.intValue();

    /*color*/
    int strokeIndex;
    int fillIndex;

    public void setSelected(boolean b){selected=b;}

    public boolean isSelected(){return selected;}

    public abstract Glyph getGlyph();

    public abstract VText getGlyphText();

    public abstract String getText(); //a meaningful string depending on
the node's type (uri, etc)
```

```
    public abstract void comment(boolean b,Editor e);

    public abstract void displayOnTop();

    public boolean isCommented(){return commented;}

    public boolean isVisuallyRepresented(){//note IProperty defines its own
method
      //the entity might not be present in the graph (visual) if it has a
visibility attribute set to display=none or visibility=hidden
      //the test for isVisible() is necessary because INodes for which
visibility=hidden do have glyphs associated with them, they are just not
visible
      if (this.getGlyph()!=null && this.getGlyph().isVisible()){return
true;}
      else return false;
    }

    public abstract void setVisible(boolean b);

    public void setTableFormLayout(boolean b){
      table=b;
    }

    public boolean isLaidOutInTableForm(){
      return table;
    }

    public void setTextAlign(int al){
      align=al;
    }

    public int getTextAlign(){
      return align;
    }


}
```

Classe IProperty

```
/*   FILE: IProperty.java
 *   DATE OF CREATION:   10/18/2001
 *   AUTHOR :            Emmanuel Pietriga (emmanuel@w3.org)
 *   MODIF:              Fri Aug 08 17:37:14 2003 by Emmanuel Pietriga
(emmanuel@w3.org, emmanuel@claribole.net)
 *   MODIF:              Tue Dec 9 2003 by Diogo Veiga/Pedro Cecconello
*/

/*
 *
 *  (c) COPYRIGHT World Wide Web Consortium, 1994-2001.
 *  Please first read the full copyright statement in file copyright.html
 *
 */

package org.w3c.IsaViz;

import java.util.Vector;
import java.awt.BasicStroke;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.xerox.VTM.glyphs.VPath;
import com.xerox.VTM.glyphs.VTriangleOr;
import com.xerox.VTM.glyphs.VText;
import com.xerox.VTM.glyphs.VRectangle;
import com.xerox.VTM.glyphs.Glyph;
import com.xerox.VTM.engine.VirtualSpace;

import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFException;

/*Our internal model class for RDF Properties. Instances of this class are
not property types, but predicates (instances of a property type). So there
can be many IProperty with the same URI.*/

class IProperty extends INode {

    int textIndex;

    IResource subject;
    INode object;

    private String namespace="";          //namespace+localname = URI
    private String localname="";

    //spline (points to the edge common to all properties if laid out in
table form)
    VPath gl1;
    //arrow head (null if edge points to a table form)
    VTriangleOr gl2;
    //property label
    VText gl3;    //if no text has been entered yet, this glyph is null
(use this test to find out if there is text)
    //cell for table form layout (null if laid out as edge/node)
    VRectangle gl4;

    String mapID;

    /**
     *@param rs Jena property representing this edge
     */
    public IProperty(Property p){
      strokeIndex=ConfigManager.defaultPBIndex;
      textIndex=ConfigManager.defaultPTIndex;
      namespace=p.getNameSpace();
      localname=p.getLocalName();
    }

    /**Create a new IProperty from scratch (information will be added
later)*/
    public IProperty(){
      strokeIndex=ConfigManager.defaultPBIndex;
      textIndex=ConfigManager.defaultPTIndex;
    }

    void setNamespace(String n){namespace=n;}

    void setLocalname(String l){localname=l;}
```

```
    public String getIdent(){
      try {
          String res=namespace+localname;
          return (res.equals("nullnull")) ? null : res ;
      }
      catch (NullPointerException ex){return null;}
    }

    public String getNamespace(){
      return namespace;
    }

    public String getLocalname(){
      return localname;
    }

    public void setMapID(String s){mapID=s;}

    public String getMapID(){return mapID;}

    public void setSubject(IResource r){
      subject=r;
    }

    public IResource getSubject(){
      return subject;
    }

    public void setObject(IResource r){
      object=r;
    }

    public void setObject(ILiteral l){
      object=l;
    }

    /**can be an IResource or an ILiteral*/
    public INode getObject(){
      return object;
    }

    /**selects this node (and assigns colors to glyph and text)*/
    public void setSelected(boolean b,boolean selectTableCell){
      super.setSelected(b);
      if (this.isVisuallyRepresented()){
          if (selected){

      gl1.setHSVColor(ConfigManager.selTh,ConfigManager.selTs,ConfigManager
.selTv);
            if
(gl2!=null){gl2.setHSVColor(ConfigManager.selTh,ConfigManager.selTs,ConfigM
anager.selTv);}
            if
(gl3!=null){gl3.setHSVColor(ConfigManager.selTh,ConfigManager.selTs,ConfigM
anager.selTv);}
            if (selectTableCell && gl4!=null){

gl4.setHSVColor(ConfigManager.selFh,ConfigManager.selFs,ConfigManager.selFv
);
```

```
gl4.setHSVbColor(ConfigManager.selTh,ConfigManager.selTs,ConfigManager.selT
v);
            }
            //the stroke gets thicker (+2 w.r.t its original width)
            if (!table){//don't do it for table layout as the thickness
gets increased as many times as the number of rows in the table
                ConfigManager.makeGlyphStrokeThicker(gl1,2.0f);
            }
            displayOnTop();
          }
          else {
            if (commented){

gl1.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManager.comTv
);
                if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
                if
(gl3!=null){gl3.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
                if (gl4!=null){

      gl4.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager
.comFv);

      gl4.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManage
r.comTv);
                }
            }
            else {
                gl1.setColor(ConfigManager.colors[strokeIndex]);
                if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
                if
(gl3!=null){gl3.setColor(ConfigManager.colors[textIndex]);}
                if (gl4!=null){
                  gl4.setColor(ConfigManager.colors[fillIndex]);
                  gl4.setBorderColor(ConfigManager.colors[textIndex]);
                }
            }
            //the stroke gets back to its original thickness (-2 w.r.t
selected width)
            if (!table){//don't do it for table layout as the thickness
gets decreased as many times as the number of rows in the table
                ConfigManager.makeGlyphStrokeThicker(gl1,-2.0f);
            }
          }
      }
    }

    public void comment(boolean b,Editor e){
      if (b){//comment
          commented=b;
          if (this.isVisuallyRepresented()){

      gl1.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManager
.comTv);
            if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
```

```
            if
(gl3!=null){gl3.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
            if (gl4!=null){

gl4.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager.comFv
);

gl4.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManager.comT
v);
            }
          }
      }
      else {//uncomment
          if (subject!=null){//do not uncomment predicate if either subject
or object is still null
            if (object!=null){
                if ((!subject.isCommented()) && (!object.isCommented())){
                  commented=b;
                  if (this.isVisuallyRepresented()){
                      gl1.setColor(ConfigManager.colors[strokeIndex]);
                      if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
                      if
(gl3!=null){gl3.setColor(ConfigManager.colors[textIndex]);}
                      if (gl4!=null){
                        gl4.setColor(ConfigManager.colors[fillIndex]);

      gl4.setBorderColor(ConfigManager.colors[textIndex]);
                      }
                  }
                }
            }
            else {
                if (!subject.isCommented()){
                  commented=b;
                  if (this.isVisuallyRepresented()){
                      gl1.setColor(ConfigManager.colors[strokeIndex]);
                      if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
                      if
(gl3!=null){gl3.setColor(ConfigManager.colors[textIndex]);}
                      if (gl4!=null){
                        gl4.setColor(ConfigManager.colors[fillIndex]);

      gl4.setBorderColor(ConfigManager.colors[textIndex]);
                      }
                  }
                }
            }
          }
          else {
            if (object!=null){
                if (!object.isCommented()){
                  commented=b;
                  if (this.isVisuallyRepresented()){
                      gl1.setColor(ConfigManager.colors[strokeIndex]);
                      if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
                      if
(gl3!=null){gl3.setColor(ConfigManager.colors[textIndex]);}
```

```
                    if (gl4!=null){
                      gl4.setColor(ConfigManager.colors[fillIndex]);

      gl4.setBorderColor(ConfigManager.colors[textIndex]);
                    }
                  }
                }
              }//else should never happen (a predicate alone cannot exist)
            }
      }
    }

    public void setVisible(boolean b){
      if (gl1!=null){gl1.setVisible(b);gl1.setSensitivity(b);}
      if (gl2!=null){gl2.setVisible(b);gl2.setSensitivity(b);}
      if (gl3!=null){gl3.setVisible(b);gl3.setSensitivity(b);}
      if (gl4!=null){gl4.setVisible(b);gl4.setSensitivity(b);}
    }

    public void setGlyph(VPath p,VTriangleOr t){
      gl1=p;
      gl1.setType(Editor.propPathType);   //means predicate glyph (glyph
type is a quick way to retrieve glyphs from VTM)
      gl1.setOwner(this);
      gl2=t;
      if (gl2!=null){
          gl2.setType(Editor.propHeadType);   //means predicate head (glyph
type is a quick way to retrieve glyphs from VTM)
          gl2.setOwner(this);
          gl2.setPaintBorder(false);
      }
    }

    //called when initializing/destroying a resizer for this property's
path - accepts null as input
    protected void setGlyphHead(VTriangleOr t){
      gl2=t;
      if (gl2!=null){
          gl2.setPaintBorder(false);
          gl2.setType(Editor.propHeadType);   //means predicate head (glyph
type is a quick way to retrieve glyphs from VTM)
          gl2.setOwner(this);
      }
    }

    public void setGlyphText(VText t){
      gl3=t;
      gl3.setType(Editor.propTextType);   //means predicate glyph (glyph
type is a quick way to retrieve glyphs from VTM)
      gl3.setOwner(this);
    }

    public void setTableCellGlyph(VRectangle r){
      gl4=r;
      gl4.setType(Editor.propCellType);
      gl4.setOwner(this);
    }

    public Glyph getGlyph(){
      return gl1;
    }
```

```
    /*can be null*/
    public VTriangleOr getGlyphHead(){
      return gl2;
    }

    public VText getGlyphText(){
      return gl3;
    }

    /*can return null if not laid out in a table*/
    public VRectangle getTableCellGlyph(){
      return gl4;
    }

    public boolean isVisuallyRepresented(){
      //the entity might not be present in the graph (visual) if it has a
visibility attribute set to GraphStylesheet._gssHide
      //- added gl4 for robustness, but the gl1 test should be sufficient
      //the test for isVisible() is necessary because INodes for which
visibility=hidden do have glyphs associated with them, they are just not
visible
      if ((this.gl1!=null && this.gl1.isVisible()) || (this.gl4!=null &&
this.gl4.isVisible())){return true;}
      else return false;
    }

   public Element toISV(Document d,ISVManager e,Vector fonts){
      Element res=d.createElementNS(Editor.isavizURI,"isv:iproperty");
      Element uri=d.createElementNS(Editor.isavizURI,"isv:uri");
      Element
namespaceEL=d.createElementNS(Editor.isavizURI,"isv:namespace");
      namespaceEL.appendChild(d.createTextNode(namespace));
      Element
localnameEL=d.createElementNS(Editor.isavizURI,"isv:localname");
      localnameEL.appendChild(d.createTextNode(localname));
      uri.appendChild(namespaceEL);uri.appendChild(localnameEL);
      res.appendChild(uri);
            if (this.isVisuallyRepresented()){//it might actually be worth
to save the geom info when visibility=hidden (since it exists)
          //for now, we do not save anything geom info, no matter whether
display=none or visibility=hidden
              res.setAttribute("display","true");
            }else {
              res.setAttribute("display","false");
            }

            Element path=d.createElementNS(Editor.isavizURI,"isv:path");
          if (gl3!=null){
            uri.setAttribute("x",String.valueOf(gl3.vx));
            uri.setAttribute("y",String.valueOf(gl3.vy));
            //save font
            int index=fonts.indexOf(gl3.getFont());
            if (index==-1){
              fonts.add(gl3.getFont());
              index=fonts.size()-1;
            }
            //do not save font info if font is default zvtm/graph font
            if (index!=0){res.setAttribute("font",String.valueOf(index));}
          }
          StringBuffer coords=new StringBuffer();
```

```
          java.awt.geom.PathIterator pi=gl1.getJava2DPathIterator();
          float[] seg=new float[6];
          int type;
          char lastOp='Z';  //anything but M, L, Q, C since we want the
first command to explicitely appear in any case
          while (!pi.isDone()){//save the path as a sequence of
instructions following the SVG model for "d" attributes
            //we save it in SVG coordinates (not VTM) because we already
have an SVG path interpreter built in VTM's SVGCreator
            type=pi.currentSegment(seg);
            switch (type){
            case java.awt.geom.PathIterator.SEG_MOVETO:{
                if (lastOp!='M'){coords.append('M');} else {coords.append('
');}
                lastOp='M';
                coords.append(Utils.abl2c(seg[0])+" "+Utils.abl2c(seg[1]));
                break;
            }
            case java.awt.geom.PathIterator.SEG_LINETO:{
                if (lastOp!='L'){coords.append('L');} else {coords.append('
');}
                lastOp='L';
                coords.append(Utils.abl2c(seg[0])+" "+Utils.abl2c(seg[1]));
                break;
            }
            case java.awt.geom.PathIterator.SEG_QUADTO:{
                if (lastOp!='Q'){coords.append('Q');} else {coords.append('
');}
                lastOp='Q';
                coords.append(Utils.abl2c(seg[0])+" "+Utils.abl2c(seg[1])+"
"+Utils.abl2c(seg[2])+" "+Utils.abl2c(seg[3]));
                break;
            }
            case java.awt.geom.PathIterator.SEG_CUBICTO:{
                if (lastOp!='C'){coords.append('C');} else {coords.append('
');}
                lastOp='C';
                coords.append(Utils.abl2c(seg[0])+" "+Utils.abl2c(seg[1])+"
"+Utils.abl2c(seg[2])+" "+Utils.abl2c(seg[3])+" "+Utils.abl2c(seg[4])+"
"+Utils.abl2c(seg[5]));
                break;
            }
            }
            pi.next();
          }
          path.setAttribute("d",coords.toString());
          res.appendChild(path);
          if (table){
            res.setAttribute("table","true");//omitted if node-edge (but
parser will understand table=false)
            res.setAttribute("fill",String.valueOf(fillIndex));
            res.setAttribute("tstroke",String.valueOf(textIndex));
            if (gl4!=null){
                Element
cell=d.createElementNS(Editor.isavizURI,"isv:cell");
                cell.setAttribute("x",String.valueOf(gl4.vx));
                cell.setAttribute("y",String.valueOf(gl4.vy));
                cell.setAttribute("w",String.valueOf(gl4.getWidth()));
                cell.setAttribute("h",String.valueOf(gl4.getHeight()));
                res.appendChild(cell);
            }
```

```
          }
          else {//arrow head exists only for node-edge props, not for table
layout
            if (gl2!=null){
                Element
head=d.createElementNS(Editor.isavizURI,"isv:head");
                head.setAttribute("x",String.valueOf(gl2.vx));
                head.setAttribute("y",String.valueOf(gl2.vy));

head.setAttribute("w",Utils.abl2c(String.valueOf(gl2.getSize())));  //only
this one will be saved/read (w=h)
                head.setAttribute("or",String.valueOf(gl2.getOrient()));
                res.appendChild(head);
            }
          }
          res.setAttribute("stroke",String.valueOf(strokeIndex));
          if (gl1.getStroke()!=null){
            if
(gl1.getStroke().getLineWidth()!=Glyph.DEFAULT_STROKE_WIDTH){
                res.setAttribute("stroke-
width",String.valueOf(gl1.getStroke().getLineWidth()));
            }
            if (gl1.getStroke().getDashArray()!=null){
                res.setAttribute("stroke-
dasharray",Utils.arrayOffloatAsCSStrings(gl1.getStroke().getDashArray()));
            }
          }
          if (gl1.getStrokeWidth()!=1.0f){

          }
          if (this.getTextAlign()!=Style.TA_CENTER.intValue()){
            res.setAttribute("text-
align",String.valueOf(this.getTextAlign()));
          }

      if (subject!=null){
        res.setAttribute("sb",e.getPrjId(subject));
      }
      if (object!=null){

            if (object instanceof IResource)
            {
                  res.setAttribute("ob",e.getPrjId(object));
            }else{
                  res.setAttribute("ob",e.getPrjId(object));
            }
      }
      if (commented){res.setAttribute("commented","true");}  //do not put
this attr if not commented (although commented="false" is supported by the
ISV loader)
      return res;
    }

    public String toString(){return super.toString()+" "+getIdent();}

    //a meaningful string representation of this IProperty
    public String getText(){return getIdent()!=null ? getIdent() : "";}

    public void displayOnTop(){
      if
(gl4!=null){Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl4);
```

```
}
      Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl1);
      if
(gl2!=null){Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl2);
}
      if
(gl3!=null){Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl3);
}
    }

    public void setTextColor(int i){//index of color in
ConfigManager.colors
      textIndex=i;
      if (gl3!=null){
          gl3.setColor(ConfigManager.colors[textIndex]);
      }
      if (gl4!=null){
          gl4.setBorderColor(ConfigManager.colors[textIndex]);
      }
    }

    public int getTextIndex(){return textIndex;}

    public void setStrokeColor(int i){//index of color in
ConfigManager.colors
      strokeIndex=i;
      gl1.setColor(ConfigManager.colors[strokeIndex]);
      if (gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
    }

    public int getStrokeIndex(){return strokeIndex;}

    public void setCellFillColor(int i){//index of color in
ConfigManager.colors
      fillIndex=i;
      if (gl4!=null){gl4.setColor(ConfigManager.colors[fillIndex]);}
    }

    public int getCellFillIndex(){return fillIndex;}

    /*returns true if the VPath representing the edge for IProperty p is
shared with at least one other IProperty (in table layout)*/
    public static boolean sharedPropertyArc(IProperty p){
      boolean res=false;
      if (p.getSubject()!=null){
          Vector v=p.getSubject().getOutgoingPredicates();
          if (v!=null){
            IProperty tmpP;
            for (int i=0;i<v.size();i++){
                tmpP=(IProperty)v.elementAt(i);
                if (tmpP.getGlyph()==p.getGlyph()){return true;}
            }
          }
      }
      return res;
    }

    /*returns a vector containing all properties in the same table as p
(but not p)*/
    public static Vector getAllPropertiesInSameTableAs(IProperty p){
      Vector res=new Vector();
```

```
      if (p.isLaidOutInTableForm() && p.getSubject()!=null){
          Vector v=p.getSubject().getOutgoingPredicates();
          if (v!=null){
            IProperty tmpP;
            for (int i=0;i<v.size();i++){
                tmpP=(IProperty)v.elementAt(i);
                if (tmpP.getGlyph()==p.getGlyph() &&
tmpP!=p){res.add(tmpP);}
            }
          }
      }
      return res;
    }

    public static Vector getTableIncomingEdge(IProperty p){
      Vector res=null;
      if (p.getSubject()!=null){
          Vector props=p.getSubject().getOutgoingPredicates();
          if (props!=null){
            IProperty tmpP;
            for (int i=0;i<props.size();i++){
                tmpP=(IProperty)props.elementAt(i);
                if (tmpP.getGlyph()!=null &&
p.getGlyph()==tmpP.getGlyph()){//this test works, even for p being in the
undo delete stack
                  res=new Vector();//as deleted properties keep a ref to
their VPath edge ; it is a little weird, and I am
                  res.add(tmpP.getGlyph());//not sure this is really
robust, but for now it works, and I don't know of any other
                  res.add(tmpP.getGlyphHead());//way to do that efficiently
(and I like efficiency :-)
                  return res;
                }
            }
          }
      }
      return res;
    }

}
```

Classe IResource

```
/*   FILE: IResource.java
 *   DATE OF CREATION:   10/18/2001
 *   AUTHOR :            Emmanuel Pietriga (emmanuel@w3.org)
 *   MODIF:              Fri Aug 08 17:37:31 2003 by Emmanuel Pietriga
(emmanuel@w3.org, emmanuel@claribole.net)
 *   MODIF:              Tue Dec 9 2003 by Diogo Veiga/Pedro Cecconello
 */

/*
 *
 *  (c) COPYRIGHT World Wide Web Consortium, 1994-2001.
 *  Please first read the full copyright statement in file copyright.html
 *
 */



package org.w3c.IsaViz;
```

```
import java.util.Vector;
import java.util.Hashtable;
import java.io.File;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.xerox.VTM.glyphs.*;
import com.xerox.VTM.engine.VirtualSpaceManager;
import com.xerox.VTM.engine.VirtualSpace;

import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.AnonId;
import com.hp.hpl.jena.rdf.model.RDFException;

/*Our internal model class for RDF Resources*/

class IResource extends INode {

    /**returns the substring of a Jena AnonID that is unique for each
anonymous resource (i.e. what is after the first stroke)
     * e.g. e184cb:ea21ce7dcf:-7fda  will return 7fda because everything
that else is common to all AnonIDs fro a given implementation/execution of
the application
     */
    public static String getJenaAnonId(AnonId id){
      StringBuffer sb=new StringBuffer(id.toString());
      int i=0;
      while (i<sb.length()){
          if (sb.charAt(i)=='-'){sb.delete(0,i+1);break;}
          i++;
      }
      return sb.toString();
    }

    Vector incomingPredicates;   //list of IProperty - null if empty
    Vector outgoingPredicates;   //list of IProperty - null if empty

    private boolean anonymous=false;   //anonymous resource or not
    private String anonymousID;        //anonymous id (null if not
anonymous)
//     private String namespace;          //namespace+localname = URI
//     private String localname;
    private String uri, classUri;
    private boolean fragmentID=false;

    private String label;  //set only if there is an rdfs:label property
for this resource

    String mapID;

    Glyph gl1;
    VText gl2;    //if not text has been entered yet, this glyph is null
(use this test to find out if there is text)

    /**
     *@param r Jena resource representing this node
     */
    public IResource(Resource r){
      fillIndex=ConfigManager.defaultRFIndex;
```

```
      strokeIndex=ConfigManager.defaultRTBIndex;
      try {
          if (r.isAnon()){
            anonymous=true;

      anonymousID=Editor.ANON_NODE+IResource.getJenaAnonId(r.getId());
            fragmentID=false;
          }
          else {
            anonymous=false;
            uri=r.getURI();
            if
(uri.startsWith(Editor.BASE_URI)){this.setURIFragment(true);}
          }
      }
      catch (RDFException ex){System.err.println("Error: IResouce(Resource
- Jena): "+ex);}
    }

    /**Create a new IResource from scratch (information will be added
later)*/
    public IResource(){
      fillIndex=ConfigManager.defaultRFIndex;
      strokeIndex=ConfigManager.defaultRTBIndex;
    }

    //the split between namespace and localname is made automatically by
IsaViz/Jena (it is just a guess)
    void setURI(String u){
      this.uri=u;
    }

    void setURIClass(String u) {
        this.classUri=u;
    }

    void setURIFragment(boolean b){
      fragmentID=b;
    }

    boolean isURIFragment(){
      return fragmentID;
    }

    //id MUST contain the anon prefix
    void setAnonymousID(String id){anonymousID=id;}

      boolean isAnon(){return anonymous;}

    void setAnon(boolean b){anonymous=b;}

    public String getIdentity(){
      if (anonymous){return anonymousID;}
      else {return uri;}
    }

    public String getName() {
            if (anonymous){return anonymousID;}
      else {
          return uri.substring(uri.indexOf("#")+1);
        }
```

```
    }

    public String getClassName() {
            if (classUri == null){return "";}
      else {
          return classUri.substring(classUri.indexOf("#")+1);
        }
    }

    public String getClassUri() {
            if (classUri == null){return "";}
      else {
          return classUri;
        }
    }

    public String getGraphLabel(){
      if (anonymous){return anonymousID;}
      else {
          try {
            String res=uri;
            if (fragmentID &&
res.startsWith(Editor.BASE_URI)){res=res.substring(Editor.BASE_URI.length()
);}
            return res;
          }
          catch (NullPointerException ex){return "";}
      }
    }

    //rdfs:label property (set to null if empty)
    public void setLabel(String s){label=(s.length()==0) ? null : s;}

    //rdfs:label property (can be null)
    public String getLabel(){return label;}

    public void setMapID(String s){mapID=s;}

    public String getMapID(){return mapID;}

    public void addIncomingPredicate(IProperty p){
      if (incomingPredicates==null){
          incomingPredicates=new Vector();
          incomingPredicates.add(p);
      }
      else {
          if (!incomingPredicates.contains(p)){incomingPredicates.add(p);}
      }
    }

    public void removeIncomingPredicate(IProperty p){
      if (incomingPredicates!=null && incomingPredicates.contains(p)){
          incomingPredicates.remove(p);
          if (incomingPredicates.isEmpty()){incomingPredicates=null;}
      }
    }

    /**returns null if none*/
    public Vector getIncomingPredicates(){
      return incomingPredicates;
    }
```

```
    public void addOutgoingPredicate(IProperty p){
      if (outgoingPredicates==null){
          outgoingPredicates=new Vector();
          outgoingPredicates.add(p);
//        System.err.println(getIdent()+" -a->
"+Utils.vectorOfObjectsAsCSString(outgoingPredicates));
      }
      else {
          if (!outgoingPredicates.contains(p)){outgoingPredicates.add(p);}
//System.err.println(getIdent()+" -b->
"+Utils.vectorOfObjectsAsCSString(outgoingPredicates));}
          //else {System.err.println(getIdent()+" -c->
"+Utils.vectorOfObjectsAsCSString(outgoingPredicates));}
      }
    }

    public void removeOutgoingPredicate(IProperty p){
      if (outgoingPredicates!=null && outgoingPredicates.contains(p)){
          outgoingPredicates.remove(p);
          if (outgoingPredicates.isEmpty()){outgoingPredicates=null;}
      }
    }

    /**returns null if none*/
    public Vector getOutgoingPredicates(){
      return outgoingPredicates;
    }

    /**selects this node (and assigns colors to glyph and text)*/
    public void setSelected(boolean b){
      super.setSelected(b);
      if (this.isVisuallyRepresented()){
          if (selected){

      gl1.setHSVColor(ConfigManager.selFh,ConfigManager.selFs,ConfigManager
.selFv);

      gl1.setHSVbColor(ConfigManager.selTh,ConfigManager.selTs,ConfigManage
r.selTv);
            if
(gl2!=null){gl2.setHSVColor(ConfigManager.selTh,ConfigManager.selTs,ConfigM
anager.selTv);}
            VirtualSpace
vs=Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace);
            vs.onTop(gl1);vs.onTop(gl2);
          }
          else {
            if (commented){

gl1.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager.comFv
);

gl1.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManager.comT
v);
                if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
            }
            else {
                gl1.setColor(ConfigManager.colors[fillIndex]);
```

```
                gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
                if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
            }
          }
        }
    }

    public void comment(boolean b,Editor e){
      commented=b;
      if (commented){//comment
          if (this.isVisuallyRepresented()){

      gl1.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager
.comFv);

      gl1.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManage
r.comTv);
            if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
          }
          if (incomingPredicates!=null){
            for (int i=0;i<incomingPredicates.size();i++){

e.commentPredicate((IProperty)incomingPredicates.elementAt(i),true);
            }
          }
          if (outgoingPredicates!=null){
            for (int i=0;i<outgoingPredicates.size();i++){

e.commentPredicate((IProperty)outgoingPredicates.elementAt(i),true);
            }
          }
      }
      else {//uncomment
          if (this.isVisuallyRepresented()){
            gl1.setColor(ConfigManager.colors[fillIndex]);
            gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
            if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
          }
          if (incomingPredicates!=null){
            for (int i=0;i<incomingPredicates.size();i++){

e.commentPredicate((IProperty)incomingPredicates.elementAt(i),false);
            }
          }
          if (outgoingPredicates!=null){
            for (int i=0;i<outgoingPredicates.size();i++){

e.commentPredicate((IProperty)outgoingPredicates.elementAt(i),false);
            }
          }
      }
    }

    public void setVisible(boolean b){
      if (gl1!=null){gl1.setVisible(b);gl1.setSensitivity(b);}
      if (gl2!=null){gl2.setVisible(b);gl2.setSensitivity(b);}
    }
```

```
    public void setGlyph(Glyph e){
      gl1=e;
      gl1.setType(Editor.resShapeType);   //means resource glyph (glyph
type is a quick way to retrieve glyphs from VTM)
      gl1.setOwner(this);
    }

    public void setGlyphText(VText t){
      if (t!=null){
          gl2=t;
          gl2.setType(Editor.resTextType);  //means resource text (glyph
type is a quick way to retrieve glyphs from VTM)
          gl2.setOwner(this);
      }
      else {gl2=null;}
    }

    public Glyph getGlyph(){
      return gl1;
    }

    public VText getGlyphText(){
      return gl2;
    }

    public Element toISV(Document d,ISVManager e,Hashtable
bitmapImages,File prjFile,Vector fonts){
      Element res=d.createElementNS(Editor.isavizURI,"isv:iresource");
      Element identif=d.createElementNS(Editor.isavizURI,"isv:URIorID");
      if (!anonymous){
            res.setAttribute("classUri",classUri);
          Element uriEL=d.createElementNS(Editor.isavizURI,"isv:uri");
          uriEL.appendChild(d.createTextNode(uri));
          if (fragmentID){uriEL.setAttribute("fID","true");}
          identif.appendChild(uriEL);
      }
      else {
          if (anonymousID!=null){
            Element
anonIDEL=d.createElementNS(Editor.isavizURI,"isv:anonID");
            //do not save prefix since it is a user preference (read from
the config file)

      anonIDEL.appendChild(d.createTextNode(Utils.erasePrefix(anonymousID))
);
            identif.appendChild(anonIDEL);
          }
      }
      res.appendChild(identif);
            if (this.isVisuallyRepresented()){
          //it might actually be worth to save the geom info when
visibility=hidden (since it exists)
          //for now, we do not save anything geom info, no matter whether
display=none or visibility=hidden
                res.setAttribute("display","true");
            }else {
                res.setAttribute("display","false");
            }
          if (table){res.setAttribute("table","true");}//omitted if node-
edge (but parser will understand table=false)
```

```
          if (gl2!=null){
            identif.setAttribute("x",String.valueOf(gl2.vx));
            identif.setAttribute("y",String.valueOf(gl2.vy));
          }
          res.setAttribute("x",String.valueOf(gl1.vx));
          res.setAttribute("y",String.valueOf(gl1.vy));
          res.setAttribute("fill",String.valueOf(fillIndex));
          res.setAttribute("stroke",String.valueOf(strokeIndex));
          if (gl1.getStroke()!=null){
            if
(gl1.getStroke().getLineWidth()!=Glyph.DEFAULT_STROKE_WIDTH){
                res.setAttribute("stroke-
width",String.valueOf(gl1.getStroke().getLineWidth()));
            }
            if (gl1.getStroke().getDashArray()!=null){
                res.setAttribute("stroke-
dasharray",Utils.arrayOffloatAsCSStrings(gl1.getStroke().getDashArray()));
            }
          }
//        if (this.getTextAlign()!=Style.TA_CENTER.intValue()){
            res.setAttribute("text-
align",String.valueOf(this.getTextAlign()));
//        }
          if (gl1 instanceof VEllipse){
            res.setAttribute("shape",Style.ELLIPSE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VRectangle){
            res.setAttribute("shape",Style.RECTANGLE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VRoundRect){
            res.setAttribute("shape",Style.ROUND_RECTANGLE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VImage){
            //here we must save the bitmap icon using a mechanism close to
what we have for SVG export in the ZVTM
            File
bitmapFile=Utils.exportBitmap((VImage)gl1,prjFile,bitmapImages,this.getClas
sName());
            /*relative URI as the png files are supposed
              to be in img_subdir w.r.t the SVG file*/
            if (bitmapFile!=null){
                res.setAttribute("shape","icon");
```

```
res.setAttributeNS(com.xerox.VTM.svg.SVGWriter.xlinkURI,"xlink:href",ISVMan
ager.img_subdir.getName()+"/"+bitmapFile.getName());
            }
            else {//if the bitmap export process fails in any way, replace
it by a standard ellipse
                res.setAttribute("shape",Style.ELLIPSE.toString());
            }

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VPolygon){

      res.setAttribute("shape","{"+((VPolygon)gl1).getVerticesAsText()+"}")
;
          }
          else if (gl1 instanceof VShape){

      res.setAttribute("shape","["+((VShape)gl1).getVerticesAsText()+"]");
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
            res.setAttribute("or",String.valueOf(gl1.getOrient()));
          }
          else if (gl1 instanceof VCircle){
            res.setAttribute("shape",Style.CIRCLE.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VDiamond){
            res.setAttribute("shape",Style.DIAMOND.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VOctagon){
            res.setAttribute("shape",Style.OCTAGON.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VTriangle){
            if (gl1.getOrient()==(float)Math.PI){
                res.setAttribute("shape",Style.TRIANGLES.toString());
            }
            else if (gl1.getOrient()==(float)-Math.PI/2.0f){
                res.setAttribute("shape",Style.TRIANGLEE.toString());
            }
            else if (gl1.getOrient()==(float)Math.PI/2.0f){
                res.setAttribute("shape",Style.TRIANGLEW.toString());
            }
            else {
                res.setAttribute("shape",Style.TRIANGLEN.toString());
            }
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else {//for robustness
            res.setAttribute("shape",Style.ELLIPSE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
```

```
          //save font
          if (gl2!=null){
            int index=fonts.indexOf(gl2.getFont());
            if (index==-1){
                fonts.add(gl2.getFont());
                index=fonts.size()-1;
            }
            //do not save font info if font is default zvtm/graph font
            if (index!=0){res.setAttribute("font",String.valueOf(index));}
          }

        if (anonymous){res.setAttribute("isAnon","true");}  //do not put
this attr if not anon (although isAnon="false" is supported by the ISV
loader)
      if (commented){res.setAttribute("commented","true");}  //do not put
this attr if not commented (although commented="false" is supported by the
ISV loader)

      res.setAttribute("id",e.getPrjId(this));

      return res;
    }

    public String toString(){return super.toString()+" "+getIdentity();}

    //a meaningful string representation of this IResource
    public String getText(){return (getIdentity()==null) ? "" :
getIdentity();}

    public void displayOnTop(){
      Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl1);
      Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl2);
    }

    public void setFillColor(int i){//index of color in
ConfigManager.colors
      fillIndex=i;
      gl1.setColor(ConfigManager.colors[fillIndex]);
    }

    public int getFillIndex(){return fillIndex;}

    public void setStrokeColor(int i){//index of color in
ConfigManager.colors
      strokeIndex=i;
      gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
      if (gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
    }

    public int getStrokeIndex(){return strokeIndex;}

    /*returns true if this resource has a property rdf:type with value
type*/
    public boolean hasRDFType(String type){
      if (outgoingPredicates!=null){
          IProperty p;
          for (int i=0;i<outgoingPredicates.size();i++){
            p=(IProperty)outgoingPredicates.elementAt(i);
            if (p.getIdent().equals(GraphStylesheet._rdfType) &&
(p.getObject() instanceof IResource) &&
((IResource)p.getObject()).getIdentity().equals(type)){return true;}
```

```
            }
        }
        return false;
    }

}
```

Classe Iliteral

```
/*   FILE: ILiteral.java
 *   DATE OF CREATION:   10/18/2001
 *   AUTHOR :            Emmanuel Pietriga (emmanuel@w3.org)
 *   MODIF:              Fri Aug 08 17:37:07 2003 by Emmanuel Pietriga
(emmanuel@w3.org, emmanuel@claribole.net)
 *   MODIF:              Tue Dec 9 2003 by Diogo Veiga/Pedro Cecconello
 */

/*
 *
 *  (c) COPYRIGHT World Wide Web Consortium, 1994-2001.
 *  Please first read the full copyright statement in file copyright.html
 *
 */



package org.w3c.IsaViz;

import java.util.Hashtable;
import java.util.Vector;
import java.io.File;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.xerox.VTM.glyphs.*;
import com.xerox.VTM.engine.VirtualSpaceManager;
import com.xerox.VTM.engine.VirtualSpace;

import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.RDFException;
import com.hp.hpl.jena.datatypes.RDFDatatype;

/*Our internal model class for RDF Literals*/

class ILiteral extends INode {

    private boolean escapeXML=true;
    private String language;
    private String value;
    private RDFDatatype datatype; //null if plain literal

    IProperty incomingPred;

    Glyph gl1;
    VText gl2;   //if not text has been entered yet, this glyph is null
(use this test to find out if there is text)

    String mapID;

    /**
```

```
     *@param lt Jena literal representing this node
     */
    public ILiteral(Literal l){
      fillIndex=ConfigManager.defaultLFIndex;
      strokeIndex=ConfigManager.defaultLTBIndex;
      try {
          escapeXML=l.getWellFormed();  //right now, Jena always say false
- do not know why Bug? - anyway does not seem to have an impact on
serialization, even if it is indeed well-formed
          if (l.getLanguage().length()>0){language=l.getLanguage();}
          datatype=l.getDatatype();
          value=l.getLexicalForm();
          //value=l.getString();
      }
      catch (RDFException ex){System.err.println("Error: ILiteral(Literal -
Jena): "+ex);}
    }

    /**Create a new ILiteral from scratch (information will be added
later)*/
    public ILiteral(){
      fillIndex=ConfigManager.defaultLFIndex;
      strokeIndex=ConfigManager.defaultLTBIndex;
    }

    void setLanguage(String l){language=l;}

    public String getLang(){return language;}

    void setEscapeXMLChars(boolean b){escapeXML=b;}

    public boolean escapesXMLChars(){return escapeXML;}

    void setValue(String v){value=v;}


//     /**returns the Jena literal*/
//     public Literal getJenaLiteral(){
//    return l;
//     }

//     public void setJenaLiteral(Literal lit){
//    l=lit;
//     }

    public String getValue(){
      return value;
    }

    //null for plain literal
    public void setDatatype(String uri){
      if (uri==null){datatype=null;}
      else {
          if (uri.length()!=0 &&
!Utils.isWhiteSpaceCharsOnly(uri)){datatype=com.hp.hpl.jena.datatypes.TypeM
apper.getInstance().getSafeTypeByName(uri);}
          else {datatype=null;}
      }
    }

    //null for plain literal
```

```
    public void setDatatype(RDFDatatype dt){
      datatype=dt;
    }

    //null if plain literal
    public RDFDatatype getDatatype(){
      return datatype;
    }

    public void setMapID(String s){mapID=s;}

    public String getMapID(){return mapID;}

    public void setIncomingPredicate(IProperty p){
      incomingPred=p;
    }

    public IProperty getIncomingPredicate(){
      return incomingPred;
    }

    /**selects this node (and assigns colors to glyph and text)*/
    public void setSelected(boolean b){
      super.setSelected(b);
      if (this.isVisuallyRepresented()){
          if (selected){

      gl1.setHSVColor(ConfigManager.selFh,ConfigManager.selFs,ConfigManager
.selFv);

      gl1.setHSVbColor(ConfigManager.selTh,ConfigManager.selTs,ConfigManage
r.selTv);
            if
(gl2!=null){gl2.setHSVColor(ConfigManager.selTh,ConfigManager.selTs,ConfigM
anager.selTv);}
            VirtualSpace
vs=Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace);
            vs.onTop(gl1);vs.onTop(gl2);
          }
          else {
            if (commented){

gl1.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager.comFv
);

gl1.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManager.comT
v);
                if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
            }
            else {
                gl1.setColor(ConfigManager.colors[fillIndex]);
                gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
                if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
            }
          }
      }
    }
```

```
    public void comment(boolean b,Editor e){
      commented=b;
      if (commented){//comment
          if (this.isVisuallyRepresented()){

      gl1.setHSVColor(ConfigManager.comFh,ConfigManager.comFs,ConfigManager
.comFv);

      gl1.setHSVbColor(ConfigManager.comTh,ConfigManager.comTs,ConfigManage
r.comTv);
           if
(gl2!=null){gl2.setHSVColor(ConfigManager.comTh,ConfigManager.comTs,ConfigM
anager.comTv);}
          }
          if (incomingPred!=null){
            e.commentPredicate(incomingPred,true);
          }
      }
      else {//uncomment
          if (this.isVisuallyRepresented()){
            gl1.setColor(ConfigManager.colors[fillIndex]);
            gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
            if
(gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
          }
          if (incomingPred!=null){
            e.commentPredicate(incomingPred,false);
          }
      }
    }

    public void setVisible(boolean b){
      if (gl1!=null){gl1.setVisible(b);gl1.setSensitivity(b);}
      if (gl2!=null){gl2.setVisible(b);gl2.setSensitivity(b);}
    }

    public void setGlyph(Glyph r){
      gl1=r;
      gl1.setType(Editor.litShapeType);  //means literal glyph (glyph type
is a quick way to retrieve glyphs from VTM)
      gl1.setOwner(this);
    }

    public void setGlyphText(VText t){
      gl2=t;
      if (gl2!=null){
          gl2.setType(Editor.litTextType);  //means literal text (glyph
type is a quick way to retrieve glyphs from VTM)
          gl2.setOwner(this);
      }
    }

    public Glyph getGlyph(){
      return gl1;
    }

    public VText getGlyphText(){
      return gl2;
    }

    public Element toISV(Document d,ISVManager e,Hashtable
```

```
bitmapImages,File prjFile,Vector fonts){
      Element res=d.createElementNS(Editor.isavizURI,"isv:iliteral");
      if (gl2!=null && getValue()!=null && getValue().length()>0){
          Element val=d.createElementNS(Editor.isavizURI,"isv:value");
          val.appendChild(d.createTextNode(getValue()));
          val.setAttribute("x",String.valueOf(gl2.vx));
          val.setAttribute("y",String.valueOf(gl2.vy));
          res.appendChild(val);
      }
      if (!escapeXML){res.setAttribute("escapeXML","false");} //if value is
not well-formed XML, signal it
      if (language!=null){
          res.setAttribute("xml:lang",language);
      }
      if (datatype!=null){
          res.setAttribute("dtURI",datatype.getURI());
      }
      if (this.isVisuallyRepresented()){
          //it might actually be worth to save the geom info when
visibility=hidden (since it exists)
          //for now, we do not save anything geom info, no matter whether
display=none or visibility=hidden
          res.setAttribute("display","true");
      }
      else {
          res.setAttribute("display","false");
      }

        if (table){res.setAttribute("table","true");}//omitted if node-edge
(but parser will understand table=false)
          res.setAttribute("x",String.valueOf(gl1.vx));
          res.setAttribute("y",String.valueOf(gl1.vy));
          res.setAttribute("fill",String.valueOf(fillIndex));
          res.setAttribute("stroke",String.valueOf(strokeIndex));
          if (gl1.getStroke()!=null){
            if
(gl1.getStroke().getLineWidth()!=Glyph.DEFAULT_STROKE_WIDTH){
                res.setAttribute("stroke-
width",String.valueOf(gl1.getStroke().getLineWidth()));
            }
            if (gl1.getStroke().getDashArray()!=null){
                res.setAttribute("stroke-
dasharray",Utils.arrayOffloatAsCSStrings(gl1.getStroke().getDashArray()));
            }
          }
          if (this.getTextAlign()!=Style.TA_CENTER.intValue()){
            res.setAttribute("text-
align",String.valueOf(this.getTextAlign()));
          }
          if (gl1 instanceof VEllipse){
            res.setAttribute("shape",Style.ELLIPSE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
()));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VRoundRect){
            res.setAttribute("shape",Style.ROUND_RECTANGLE.toString());
```

```
      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VImage){
            res.setAttribute("shape","icon");
            //here we must save the bitmap icon using a mechanism close to
what we have for SVG export in the ZVTM
            File
bitmapFile=Utils.exportBitmap((VImage)gl1,prjFile,bitmapImages,value);
            /*relative URI as the png files are supposed
              to be in img_subdir w.r.t the SVG file*/
            if (bitmapFile!=null){
                res.setAttribute("shape","icon");

res.setAttributeNS(com.xerox.VTM.svg.SVGWriter.xlinkURI,"xlink:href",ISVMan
ager.img_subdir.getName()+"/"+bitmapFile.getName());
            }
            else {//if the bitmap export process fails in any way, replace
it by a standard ellipse
                res.setAttribute("shape",Style.RECTANGLE.toString());
            }

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
)));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else if (gl1 instanceof VPolygon){

      res.setAttribute("shape","["+((VPolygon)gl1).getVerticesAsText()+"]")
;
          }
          else if (gl1 instanceof VShape){

      res.setAttribute("shape","{"+((VShape)gl1).getVerticesAsText()+"}");
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
            res.setAttribute("or",String.valueOf(gl1.getOrient()));
          }
          else if (gl1 instanceof VCircle){
            res.setAttribute("shape",Style.CIRCLE.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VDiamond){
            res.setAttribute("shape",Style.DIAMOND.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VOctagon){
            res.setAttribute("shape",Style.OCTAGON.toString());
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VTriangle){
            if (gl1.getOrient()==(float)Math.PI){
                res.setAttribute("shape",Style.TRIANGLES.toString());
            }
            else if (gl1.getOrient()==(float)-Math.PI/2.0f){
                res.setAttribute("shape",Style.TRIANGLEE.toString());
            }
```

```
            else if (gl1.getOrient()==(float)Math.PI/2.0f){
                res.setAttribute("shape",Style.TRIANGLEW.toString());
            }
            else {
                res.setAttribute("shape",Style.TRIANGLEN.toString());
            }
            res.setAttribute("sz",String.valueOf(gl1.getSize()));
          }
          else if (gl1 instanceof VRectangle){
            res.setAttribute("shape",Style.RECTANGLE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          else {//for robustness
            res.setAttribute("shape",Style.ELLIPSE.toString());

      res.setAttribute("w",String.valueOf(((RectangularShape)gl1).getWidth(
));

      res.setAttribute("h",String.valueOf(((RectangularShape)gl1).getHeight
()));
          }
          //save font
          if (gl2!=null){
            int index=fonts.indexOf(gl2.getFont());
            if (index==-1){
                fonts.add(gl2.getFont());
                index=fonts.size()-1;
            }
            //do not save font info if font is default zvtm/graph font
            if (index!=0){res.setAttribute("font",String.valueOf(index));}
          }

      if (commented){res.setAttribute("commented","true");}  //do not put
this attr if not commented (although commented="false" is supported by the
ISV loader)
      res.setAttribute("id",e.getPrjId(this));
      return res;
    }

    public String toString(){
      String res=super.toString();
      if (getValue()!=null){res+=" "+getValue();}
      if (getDatatype()!=null){res+=" ["+getDatatype().getURI()+"]";}
      return res;
    }

    //a possibly truncated version of this ILiteral's value
    public String getText(){
      if (value!=null){return (value.length()>=Editor.MAX_LIT_CHAR_COUNT) ?
value.substring(0,Editor.MAX_LIT_CHAR_COUNT) : value;}
      else return "";
    }

    public void displayOnTop(){
      Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl1);
      if
```

```
(gl2!=null){Editor.vsm.getVirtualSpace(Editor.mainVirtualSpace).onTop(gl2);
}
    }

    public void setFillColor(int i){//index of color in
ConfigManager.colors
      fillIndex=i;
      gl1.setColor(ConfigManager.colors[fillIndex]);
    }

    public int getFillIndex(){return fillIndex;}

    public void setStrokeColor(int i){//index of color in
ConfigManager.colors
      strokeIndex=i;
      gl1.setBorderColor(ConfigManager.colors[strokeIndex]);
      if (gl2!=null){gl2.setColor(ConfigManager.colors[strokeIndex]);}
    }

    public int getStrokeIndex(){return strokeIndex;}

}
```

Classe GenomicLibrary

```
package org.w3c.IsaViz;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;

import java.io.*;
import java.util.Map;
import java.util.ArrayList;
import java.util.Set;
import java.util.TreeSet;
import java.util.HashMap;

public class GenomicLibrary extends Object {

    public Map class_prop,class_subclass, prop_value;
    public String inputFileName;
      public ArrayList all_properties;

      public GenomicLibrary(){
            class_prop = new HashMap();
        class_subclass = new HashMap();
            prop_value = new HashMap();
        inputFileName = "genomicRDFlibrary2.rdf";
            all_properties = new ArrayList();
        parseRDFLibrary();
      }

    public void parseRDFLibrary () {
       try {
            // create an empty model
            Model model = ModelFactory.createDefaultModel();

           FileReader freader=new FileReader(inputFileName);
           RDFReader parser = model.getReader("RDF/XML");
           parser.read(model,freader,"");
```

```
            // select all the resources with a VCARD.FN property

            ResIterator iter =
model.listSubjectsWithProperty(RDF.type,RDFS.Class);
            Resource aux;
            Statement subClass_stmt;

            if (iter.hasNext()) {
                while (iter.hasNext()) {
                    aux = iter.nextResource();

                    if (aux.hasProperty(RDFS.subClassOf)){
                        subClass_stmt = aux.getProperty(RDFS.subClassOf);

class_subclass.put(aux.toString(),subClass_stmt.getObject().toString());
                        class_prop.put(aux.toString(),new ArrayList());
                    }else {
                        class_subclass.put(aux.toString(),"");
                        class_prop.put(aux.toString(),new ArrayList());
                        //System.out.println(aux);
                    }
                }
            } else {
                System.out.println("No classes were found in the
database");
            }

           iter = model.listSubjectsWithProperty(RDFS.domain);
           StmtIterator domIter;
           Statement stmtAux, stmtRange;
           String resUri;

           if (iter.hasNext()) {
                while (iter.hasNext()) {
                    aux = iter.nextResource();
                              all_properties.add(aux.toString());
                    domIter = aux.listProperties(RDFS.domain);   //get all
domain statements

                    if (domIter.hasNext()){

                        while (domIter.hasNext()){
                            stmtAux = domIter.nextStatement();
                            resUri = stmtAux.getObject().toString();

//System.out.println(stmtAux.getObject().toString()+"
"+stmtAux.getSubject().toString());

insertPropertytoResource(resUri,stmtAux.getSubject().toString());
                        }

                    }else{
                        System.out.println("No domain properties were found
in this resource");
                    }

                        //get the range statement of each properties
                        stmtRange = aux.getProperty(RDFS.range);

      prop_value.put(stmtRange.getSubject().toString(),stmtRange.getObject(
).toString());
```

```
              }
            } else {
                System.out.println("No properties were found in the
database");
            }

        setPropertiesbySubClasses();

        } catch (Exception e) {
            System.out.println("Failed: " + e);
        }
    }

    public void setPropertiesbySubClasses(){
     Set typesG = class_subclass.keySet();
     Object[] types = typesG.toArray();
     ArrayList propertiesType;
     String classURI;
     boolean changed = true;
     String subclassOf;

    while (changed){
        changed = false;
        for (int i=0;i< types.length ;i++ )
        {
            classURI = (String) types[i];
            subclassOf = (String)class_subclass.get(classURI);
            propertiesType = (ArrayList) class_prop.get(subclassOf);

            if (!subclassOf.equals("")){
                for (int j=0; j < propertiesType.size(); j++){

                   if ( insertPropertytoResource(classURI, ((String)
propertiesType.get(j)) ) )
                         changed = true;
                }
               }
        }

    }

 }

    public boolean insertPropertytoResource(String resourceUri, String
propertyValue){
        ArrayList properties = (ArrayList) class_prop.get(resourceUri);

        if (properties.indexOf(propertyValue) == -1){
            properties.add(propertyValue);
            return true;
        }else
           return false;
    }

      public Object[] returnLibraryTypes(){

        Set typesG = class_prop.keySet();
      Object[] types = typesG.toArray();
        String uriType;
        TreeSet orderedSet = new TreeSet();
```

```
            for (int i=0;i< types.length ;i++ )
            {
                  uriType = (String) types[i];
                  orderedSet.add(uriType);
            }

       return (orderedSet.toArray());

      }

    public Object[] returnLibraryTypesID(){
        Set typesG = class_prop.keySet();
        Object[] types = typesG.toArray();
      String uriType, id;
        TreeSet orderedSet = new TreeSet();

        for (int i=0;i< types.length ;i++ ) {
            uriType = (String) types[i];
            id = uriType.substring(uriType.indexOf("#")+1);
            orderedSet.add(id);
      }
        return (orderedSet.toArray());
    }

    public ArrayList returnPropertyByClass(String c) {
        ArrayList properties = (ArrayList) class_prop.get(c);
        return properties;
    }

    public ArrayList returnPropertyIDByClass(String c) {
        ArrayList properties = (ArrayList) class_prop.get(c);
        ArrayList propertiesID = new ArrayList();
        String uri, id;
        for (int i = 0; i < properties.size(); i++) {
            uri = (String) properties.get(i);
            id = uri.substring(uri.indexOf("#")+1);
            propertiesID.add(id);
        }
        return propertiesID;
    }

    public ArrayList returnProperties() {
        return all_properties;
    }

      public String returnPropertyValue(String propShortName) {
       return ((String)
prop_value.get("http://www.intelab.ufsc.br/genomica/library#"+propShortName
));
    }

    public boolean classExists(String c) {
       return (class_prop.get(c)!=null);
    }

}
```

Anexo 2 – Artigo

Metabolic IsaViz: Representando Vias Metabólicas em Grafos RDF Customizados

Diogo Fernando Veiga[1] e Pedro de Stege Cecconello[2]

Universidade Federal de Santa Catarina – Depto. de Informática e Estatística
Caixa Postal 476 – 88040-900 Florianópolis – SC – E-mail: ine-ctc@inf.ufsc.br
Telefone: +55 048 331 9739 Fax: +55 048 331 9770
[1]veiga@inf.ufsc.br, [2]stege@inf.ufsc.br

RESUMO

As vias metabólicas são conjuntos de reações nos organismos que sintetizam ou degradam moléculas, como proteínas e carboidratos. Criar modelos computacionais que possam ser analisados é fundamental para aplicações em biotecnologia, como a obtenção da máxima quantidade de um composto químico de interesse. O propósito deste trabalho foi combinar uma tecnologia de representação de dados, chamada RDF (Resource Description Framework) com a representação de vias metabólicas, ambas baseadas em grafos direcionados. Para isto, implementamos as modificações lógicas/gráficas em um ambiente de desenvolvimento RDF já existente, conhecido como IsaViz. As modificações necessárias para a adaptação do software ao domínio das vias bioquímicas juntamente com uma biblioteca de classes e propriedades RDF – a Biblioteca Genômica, que foi acoplada ao sistema – resultou no Metabolic IsaViz, uma extensão do IsaViz destinado à criação de modelos de vias metabólicas e regulatórias. O Metabolic IsaViz, no âmbito do projeto de pesquisa do Grupo de Engenharia Genômica da UFSC, será a base de um sistema de informação genômica, chamado GenIS (Genome Information System). Esta plataforma computacional será utilizada para a simulação de vias bioquímicas e pretende reduzir as atividades experimentais, obtendo, por meio das simulações in silico, indícios para conduzir os trabalhos em laboratório. Os modelos de vias serão criados com o Metabolic IsaViz informando parâmetros necessários às equações de simulação, como as constantes cinéticas enzima-substrato, constante de inibição e outras e a partir daí o código RDF/XML será interpretado por um simulador de vias, baseados em RPHs (Redes de Petri Híbridas).

Palavras-chave: vias metabólicas, RDF, IsaViz.

1. INTRODUÇÃO

A determinação das vias metabólicas e de controle regulatório de organismos, a partir da informação genômica seqüenciada, é uma importante etapa para estudos de análise pós-seqüenciamento. Esta caracterização é possível porque a seqüência de DNA contém os genes que expressam as enzimas que, por sua vez, atuam nas vias bioquímicas como catalisadores biológicos, tornando possível a conversão entre compostos intermediários de uma certa via. O conhecimento das vias bioquímicas permite que sejam aplicadas técnicas que induzam à

acumulação de um certo composto de interesse, por exemplo, para ser utilizado na fabricação de medicamentos, alimentos, etc.

Desta forma, projetar um ambiente gráfico para criação de modelos de vias metabólicas é uma iniciativa em relação à necessidade de análise e compreensão dos dados genômicos que, pelo desenvolvimento de novas técnicas de seqüenciamento, estão sendo gerados em grande volume.

O propósito deste trabalho foi combinar uma tecnologia de representação de dados, chamada RDF (KLYNE e CARROL, 2003; MANOLA e MILLER, 2003) com a representação de vias metabólicas (Vide seção 2.2 para conceituação básica), ambas baseadas em grafos direcionados. Para isto, implementamos as modificações lógicas/gráficas em um ambiente de desenvolvimento RDF já existente, conhecido como IsaViz (PIETRIGA, 2003). As modificações necessárias para a adaptação do software ao domínio das vias bioquímicas juntamente com uma biblioteca de classes e propriedades RDF – a Biblioteca Genômica, que foi acoplada ao sistema – resultou no Metabolic IsaViz, uma extensão do IsaViz destinado à criação de modelos de vias metabólicas e regulatórias. O Metabolic IsaViz, no âmbito do projeto de pesquisa do Grupo de Engenharia Genômica da UFSC, será a base de um sistema de informação genômica, chamado GenIS (Genome Information System). Esta plataforma computacional será utilizada para a simulação de vias bioquímicas e pretende reduzir as atividades experimentais, obtendo, por meio das simulações *in silico*, indícios para conduzir os trabalhos em laboratório. Os modelos de vias serão criados com o Metabolic IsaViz informando parâmetros necessários às equações de simulação, como as constantes cinéticas enzima-substrato, constante de inibição e outras e a partir daí o código RDF/XML será interpretado por um simulador de vias, baseados em RPHs (Redes de Petri Híbridas). Este simulador ainda se encontra em processo de estudo.

No contexto da Web Semântica, o RDF desempenha o papel de representar os recursos e seus relacionamentos, ao lado da XML. A Web Semântica é considerada a próxima geração da Web, na qual o conteúdo será estruturado com aspectos sintáticos e, sobretudo, semânticos, agregando-se significado tanto para as máquinas quanto para as pessoas. Na Web atual, através dos documentos HTML, a informação é projetada para visualização dos usuários que a acessam. Adicionando páginas Web com dados destinados aos computadores, a Web se tornará a Web Semântica (BERNERS-LEE et al., 2000).

Em resumo, podemos destacar aqui os fatores importantes nas escolhas das tecnologias propostas no trabalho:

1) A representação gráfica de vias metabólicas é muito semelhante a grafos direcionados. Isto tornou adequado utilizar RDF para descrição das vias já que seu modelo de dados é um multigrafo direcionado;

2) Representar em RDF traz vantagens:

- Utilizar tecnologia da Web Semântica (BERNERS-LEE et al., 2001), permite interoperabilidade entre softwares, através de agentes. A camada de representação da Web Semântica é em RDF ou XML Schema.
- RDF traz a idéia de trabalho cooperativo (MANOLA e MILLER, 2002), o que significa dizer que dados sobre um recurso RDF podem ser distribuídos entre vários *namespaces*. Relacionado às vias metabólicas, poder-se-ia dizer que enquanto um grupo de pesquisa faz a anotação das enzimas outro pode estar interessado em fazer simulações; fornece, portanto, ao recurso (neste caso a via em questão) dados sobre a cinética das reações. Um agente pode integrar estas informações.

3) IsaViz é a ferramenta recomendada pela W3C para a autoria de modelos RDF. Ela está em conformidade com as especificações RDF e oferece uma grande quantidade de recursos, como mecanismo de zoom nos grafos, aplicação de folhas de estilos GSS (Graph

Stylesheets) aos modelos, bem como exporta RDF para muitos formatos, como SVG (Scalable Vector Graphics), PNG (Portable Network Graphics) e propriamente para RDF/XML.

2. ESTUDOS REALIZADOS

2.1. Vias Metabólicas

As vias metabólicas são seqüências de reações que ocorrem nos organismos com a função de produzir energia através da degradação de macromoléculas (vias do metabolismo) ou de consumir energia (vias de anabolismo), buscando produzir moléculas complexas, tais como proteínas, com o objetivo de duplicação ou muitas outras funções celulares.

Considere o exemplo da via de biossíntese de carotenóides (ARACYC, 2003) encontrada na Arabidopsis thaliana (uma espécie de planta). Os carotenóides são uma classe de compostos produzidos pelo metabolismo secundário de plantas, que conferem pigmentação laranja e vermelha, sendo encontrados em laranjas, tomates, cenouras e flores amarelas. São usados como corantes em alimentos industrializados, bebidas e rações animais. Os carotenóides também são encontrados em bactérias, fungos e algas. São essenciais à fotossíntese, são antioxidantes, além da função de atrair animais. Para o homem, é uma importante fonte de vitamina A e combate vários tipos de câncer e doenças degenerativas da visão, atuando como reguladores do sistema imunológico.

A figura 1 denota como as reações foram identificadas no processo de anotação da A. thaliana. As reações em verde significam que a enzima está presente no organismo e a reação é única para esta via. Já para a reação denotada por uma linha preta a enzima não foi identificada e esta reação é única para esta via. Isto significa dizer que nem todos os passos da via acontecem para qualquer organismo e isto vai depender do conjunto de enzimas codificadas pelo genoma da espécie em estudo.

Figura 29 – Exemplo de uma via metabólica da *Arabidopsis thaliana*

2.2. Resource Description Framework

O RDF (*Resource Description Framework*) surgiu com o intuito de se tornar a linguagem de representação dos dados disponíveis na Web, constituindo-se uma camada essencial da Web Semântica. No contexto desta próxima Web, o RDF desempenha o papel de representar os recursos e seus relacionamentos, ao lado do XML. Um recurso web é considerado algo que possa ser acessível por um navegador, como um documento ou uma foto, quanto algo não acessível, porém também identificado por uma URI (*Uniform Resource*

Identifier), tais como funcionários de uma empresa, carros em estoque, livros em uma biblioteca.

Para cumprir seu objetivo de representação, a tecnologia RDF é composta de um modelo de dados, que é um multigrafo direcionado, e algumas linguagens de descrição para este modelo, tais como RDF/XML, N-triples e Notation-3. Em RDF, a descrição dos recursos ocorre por meio de *statements* ou asserções, que graficamente denotam a ligação entre dois nodos de um grafo. Na terminologia RDF, o nodo de onde parte a aresta é chamado de *subject*, enquanto o nodo onde chega a aresta é conhecido como *object*. A própria aresta é denominada *predicate*. Todos os elementos são rotulados por suas URIs, que têm como objetivo identificar univocamente um recurso no espaço da Web e não pretende, necessariamente, fornecer uma página que pode ser acessada por um navegador. URIs podem ser criadas para referenciar qualquer recurso.

A Figura 2 mostra um grafo RDF com uma única asserção, que descreve o modelo do carro `http://www.montadoraXX.com.br/estoque#carro0578` da montadoraXX. Neste exemplo, podemos identificar as partes de um *statement*, utilizando a terminologia comentada:

1) *Subject*: se refere ao recurso que está sendo descrito. Neste caso, o recurso é `http://www.montadoraXX.com.br/estoque#carro0578`.
2) *Predicate*: indica a propriedade ou característica do recurso. No exemplo, a propriedade é `http://www.montadoraXX.com.br/estoque#modelo`.
3) *Object*: parte do *statement* que indica o valor da propriedade. O *object* pode ser um tipo simples, como uma string, uma data ou inteiro, bem como pode ser um outro recurso. Na Figura 2, este elemento é o recurso `http://www.montadoraXX.com.br/modelos#Gurgel`.

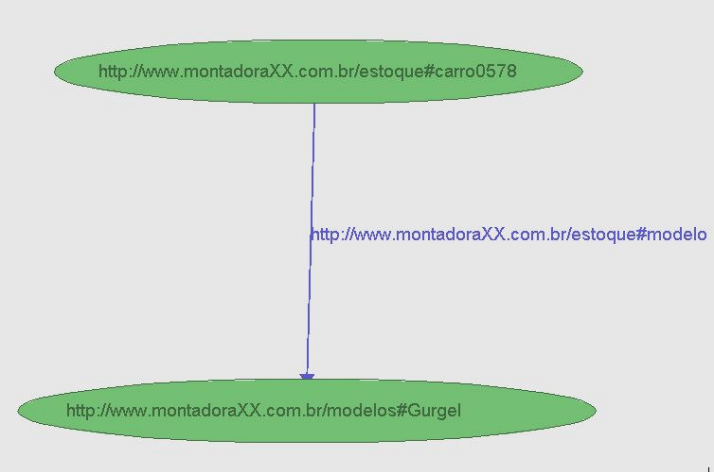

Figura 30 – Modelo RDF com uma única asserção

2.2.1. Padronização de Vocabulário: RDF Schema

Um vocabulário RDF é composto de classes e propriedades. A linguagem com a qual definimos estas classes e propriedades, ou seja, o vocabulário, é por si própria um vocabulário, chamado RDFS, que significa RDF Schema. O RDFS possui o *namespace* `http://www.w3.org/2000/01/rdf-schema#`, com o prefixo rdfs.

Uma classe, como um carro, imóvel ou pessoa, possui propriedades. Uma característica interessante em RDF é que as propriedades são definidas de forma independente, promovendo a sua reutilização.

Outro aspecto sobre RDFS é que não há necessariamente uma restrição às informações atribuídas a um recurso caso ele seja de uma certa classe, diferente do conceito de tipos do XML Schema, cujo objetivo é a validação dos dados através de uma sintaxe precisa.

A seguir mostra-se como são definidos os elementos dos esquemas RDF.

a) Definição de Classe: a atribuição de uma classe a um recurso se faz com a propriedade rdf:type. Um recurso pode ser instância de uma ou mais classes. Todas as classes são subclasses da classe rdfs:Resource. Veja o código da definição de uma classe chamada *MotorVehicle*:

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">


       <rdf:Description rdf:ID="MotorVehicle">
          <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
   schema#Class"/>
          <rdfs:subClassOf
   rdf:resource="http://www.w3.org/2000/01/rdf- schema#Resource"/>
       </rdf:Description>
       ...
     </rdf:RDF>
```

b) Definição de Propriedade: esta definição também é feita com uma propriedade rdf:type com valor rdf:Property. Além disto, está disponível a propriedade rdf:domain, indica a qual classe tal propriedade deve ser aplicada, rdf:range, classe ou tipo de dado (inteiro, string etc) que a propriedade pode assumir e rdf:subPropertyOf para o caso desta propriedade herdar características de outra.

```
<rdf:Description rdf:ID="registeredTo">
      <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#Property"/>
      <rdfs:domain rdf:resource="#MotorVehicle"/>
      <rdfs:range
rdf:resource="http://www.example.org/classes#Person"/>
</rdf:Description>
```

No caso do trecho de código acima, a propriedade *registeredTo* só pode ser aplicada a classe *MotorVehicle* e só assume valores da classe *Person*.

2.3. IsaViz: Um Ambiente para Autoria em RDF

O IsaViz (PIETRIGA, 2003) é um ambiente visual para edição e navegação de modelos RDF, representados por grafos direcionados. O software é construído com a linguagem Java e de código-livre. Recursos e literais são os nodos do grafo (elipses e retângulos respectivamente), com propriedades representando os vértices que ligam estes nodos. Desde a versão 2.0, o IsaViz suporta GSS (Graph Stylesheets), uma linguagem stylesheet derivada de CSS e SVG, para atribuir estilos aos elementos dos grafos. A interface gráfica do software (Figura 3) pode ser dividida em quatro janelas principais, que estão descritas abaixo:

1. IsaViz RDF Editor (canto superior esquerdo) – provê acesso aos menus principais e a uma paleta de ferramentas, que altera entre os modos de edição do sistema:

inserção de recursos, inserção de propriedade, seleção de nodo, seleção múltipla e outros.

2. *Graph* (lado direito) – exibe o grafo RDF em 2D. Esta janela, construída sobre a biblioteca ZVTM, possibilita uma navegabilidade otimizada, oferecendo recursos de zoom sobre regiões específicas do grafo.
3. Em *Attributes* (lado esquerdo) – são mostradas informações sobre o item selecionado no grafo. No caso de recursos, é exibido a URI ou ID do nodo (quando não for *blank node*), além das suas propriedades e de uma opção para excluí-lo.
4. *Definitions* (abaixo) – é composta por cinco abas, que desempenham uma variedade de funções:
   - Aba *Namespaces* – contém os prefixos e URIs dos *namespaces* utilizados.
   - Aba *Property Types*: contém as propriedades para cada um dos *namespaces* disponíveis para modelagem.
   - Aba *Property Browser*: apresenta as propriedades do recurso selecionado no momento, com possibilidade de navegação entre elas.
   - Aba *Stylesheets* – gerenciamento do(s) *stylesheet*(*s*) disponíveis para ser(em) aplicado(s) no grafo através dos comandos *Load*, *Apply*, *Remove*, *Edit*.
   - Aba *Quick Input* – área de texto para inserção de *statements* em RDF/XML, N-Triples ou Notation 3.

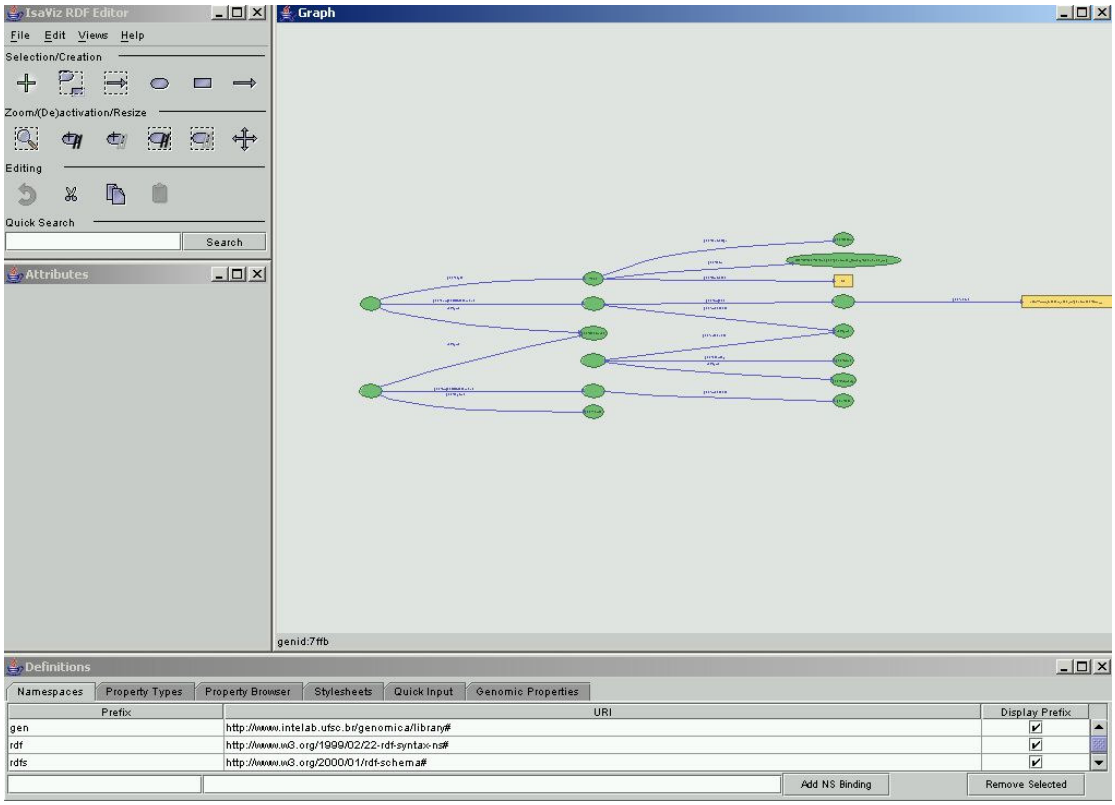

Figura 31 – Interface de desenvolvimento do software IsaViz

2.3.1. Esquema de Funcionamento

Nesta seção vamos abordar o funcionamento interno do IsaViz, analisando suas principais classes e estruturas de dados. As classes que serão comentadas e que dizem respeito ao projeto lógico do IsaViz podem ser vistas no diagrama Figura 4.

A classe abstrata *INode* representa qualquer item do grafo e seus principais atributos denotam se o recurso está selecionado e/ou comentado e qual é o alinhamento do `label` para o item (atributo `align`).

As classes *IResource*, *IProperty* e *ILiteral* têm atributos e métodos específicos.

A classe *IResource* representa um recurso RDF, que pode ter o papel de *subject* ou *object* de uma asserção. O atributo `incomingPredicates` é um vetor de *IProperties* que armazena as propriedades que "chegam" a este nodo e o atributo `outgoingPredicates`, as propriedades que partem deste recurso. A URI é o nome atribuído ao recurso na sua criação e deve identificá-lo univocamente. Não existem dois recursos com a mesma URI em um mesmo grafo RDF. A string `classUri` identifica a que classe pertence o recurso, caso ele seja pertencente ao *namespace* gen.

Existem um atributo, do tipo Glyph, que representa a forma gráfica (elipse, triângulo, círculo, imagem, etc) que o nodo assume no espaço de visualização. Outro atributo, da classe *VText*, representa o *label* do recurso.

Um método importante desta classe é o `toISV()`. Sua função é escrever no arquivo .isv (arquivo que armazena o modelo RDF quando este é salvo) um elemento XML com as informações do nodo, tais como coordenadas x e y, sua URI, *shape* (forma do Glyph) e outros atributos. É utilizada a API DOM para escrita e leitura destes arquivos XML.

```
<isv:resources>
   <isv:iresource
      display="true" fill="0" h="19" id="0" shape="icon"
      stroke="1" w="30" x="76"
      xlink:href="pathway_sample02_files/Chemicals.png" y="112">
      <isv:URIorID x="76" y="85">
<isv:uri>http://www.intelab.ufsc.br/genomica/library#Anthranilate</i
sv:uri>
       </isv:URIorID>
   </isv:iresource>
   ...
<isv:resources>
```

A classe *IProperty* possui dois atributos essenciais, que são o `subject`, um IResource e `object`, da classe *INode*. Os atributos `namespace` e `localname` identificam a propriedade, os dois juntos forma a URI.

Já os objetos da classe *ILiteral* armazenam o valor, um `datatype`, uma referência à propriedade que o aponta e outros atributos semelhantes ao *IResource*.

A classe *ISVManager* tem as rotinas de salvamento e recuperação para os arquivos de projeto. Os métodos utilizam a API DOM para criação dos arquivos XML.

A classe *Editor* é a inicial do sistema e armazena referências para as janelas gráficas: `cmp`, `tblp`, `propsp`, `navp`. Os *Hashtables* `resourcesByUri` e `propertiesByUri`, além do vetor dinâmico de literais são as estruturas de dados do grafo. No caso do hash `propertiesByUri`, dada uma certa URI, retorna-se um vetor, já que podemos ter a mesma propriedade várias vezes no modelo. No `resourcesByUri` são armazenados recursos, únicos para cada chave ou URI. O objeto `genLibrary`, da classe *GenomicLibrary*, é o responsável por efetuar o *parse* da Biblioteca Genômica expressa na linguagem RDFS e

deixar em memória as classes, as propriedades e um hash com as propriedades de cada classe, dados estes que serão utilizados na inserção de recursos e de propriedades.

Estão também nestas classes os métodos de `export()` e `load()` para os vários formatos aceitos pelo IsaViz. Os métodos `createNewResource()` e `createNewProperty()` criam *JDialogs* para obtenção das informações pelo usuário.

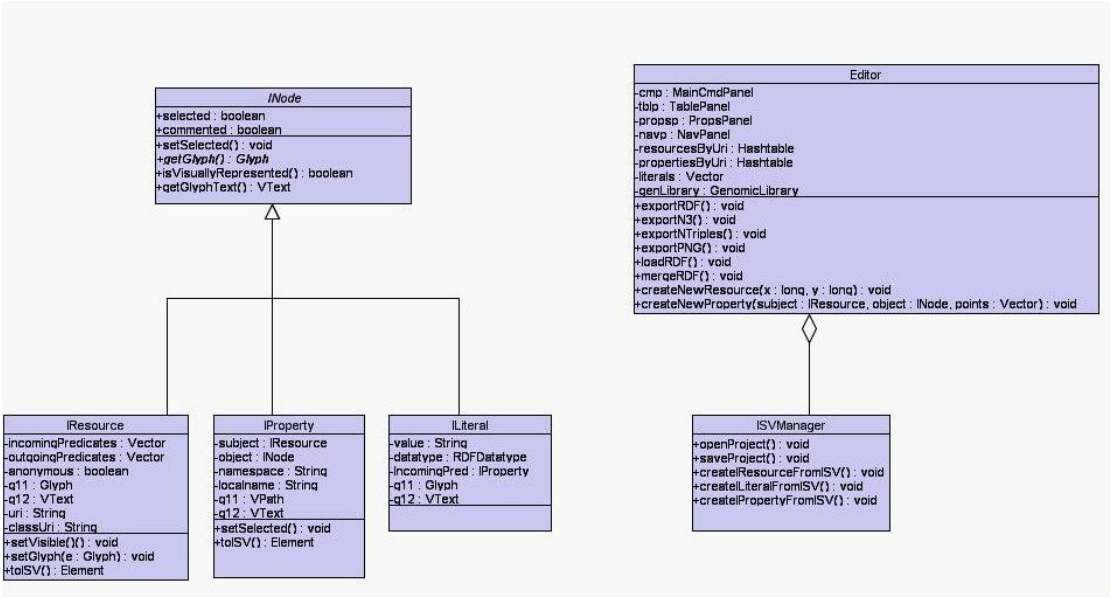

Figura 32 – Diagrama UML de classes do IsaViz que fazem parte do núcleo do sistema. Os comentários de métodos e atributos que aparecem nas classes são feitos no texto.

3. METODOLOGIA

Com o objetivo de possibilitar a modelagem de vias metabólicas e regulatórias, resultando no Metabolic IsaViz, foram feitas extensões na parte lógica e gráfica do sistema original. Nesta seção descreve-se estas extensões de forma detalhada.

3.1. Biblioteca Genômica em RDF Schema

Para definição das classes e suas propriedades, utiliza-se um vocabulário RDF com termos do domínio genômico. A biblioteca foi modelada com bases em ontologias e registros de bases de dados genômicas primárias, como o NCBI e o EMBL (VEIGA & PORTO, 2003).

As classes que estão na biblioteca referem-se à terminologia de seqüenciamento, como partes de uma seqüência (*intron*, *exon*, UTR), bem como à terminologia envolvida na modelagem de vias metabólicas (*products*, *chemicals*, *activators* entre outros). Uma tabela com os tipos da biblioteca pode ser observada na Figura 5. As classes e propriedades da biblioteca são utilizadas no Metabolic IsaViz no momento da inserção de recursos e de propriedades no modelo RDF. Na criação de um recurso (Vide Seção 3.3), é informada a que classe ele pertence; por exemplo, Glycolysis pertence à gen:Metabolic_Pathway. Já na criação de propriedades (Vide Seção 3.5), a biblioteca auxilia a modelagem mostrando quais as

propriedades que podem ser inseridas para um determinado recurso. Por exemplo, para um recurso da classe gen:Activator (um ativador de uma reação enzimática), a biblioteca sugere a inserção de propriedades como: mecanismo de ativação, descrição, fórmula estrutural e outras.

Conforme visto na Seção 2.2.1 – Padronização de Vocabulários –, um esquema RDF é composto por classes e propriedades. Pode-se analisar, utilizando como exemplo o código da Tabela 1, como foram definidas as propriedades e classes da Biblioteca Genômica.
O primeiro elemento definido no código é uma propriedade, chamada gen:reaction_direction, que especifica o sentido da reação: reversível, irreversível, sem direção conhecida. Esta propriedade é aplicada à classe gen:Enzymatic_Data (rdfs:domain) e seu valor é uma string (rdfs:range). O elemento a seguir também é uma propriedade, a gen:Description. Verifique que ela é reutilizada em várias classes tais como Gene, Protein, Chemicals, Metabolic_Pathway. O último elemento é uma classe, a gen:ChromosomeFrag, que é subclasse de gen:Sequence.

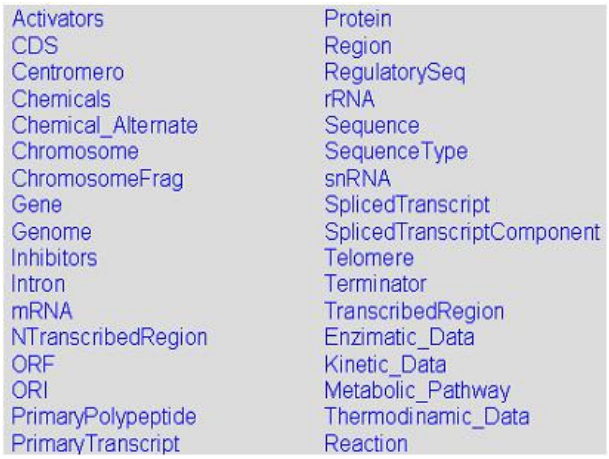

Figura 33 – Classes da Biblioteca Genômica

Tabela 13 – Fragmento que compõe a Biblioteca Genômica

De acordo com a sintaxe de RDF/XML, cada elemento rdf:description (destacado em negrito) define um elemento, cujo tipo vai ser determinado pela propriedade rdf:type. Neste trecho de código, temos a definição de duas propriedades e de uma classe, a *ChromosomeFrag*.

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
      …
<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#reaction_direction">
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Enzymatic_Data"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Description>
<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Description">
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Gene"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#PrimaryPolypeptide"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Protein"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Genome"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
   <rdfs:domain rdf:resource="http://www.intelab.ufsc.br/genomica/library#Metabolic_Pathway"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Description>
<rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#ChromosomeFrag">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://www.intelab.ufsc.br/genomica/library#Sequence"/>
 </rdf:Description>
  ...
</rdf:RDF>
```

3.2. A Classe GenomicLibrary

A classe *GenomicLibrary* é responsável por efetuar o *parsing* da Biblioteca Genômica e criar as estruturas de dados específicas que serão disponibilizadas ao Metabolic IsaViz. Esta foi a única classe adicionada ao IsaViz original, já que as outras alterações foram em classes já existentes.

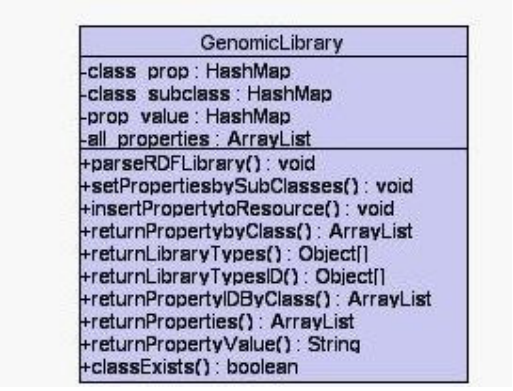

Figura 34 – Diagrama UML da classe GenomicLibrary

O *parsing* é realizado utilizando métodos oferecidos pela Jena API, uma biblioteca de manipulação de arquivos RDF/XML. O método principal da *GenomicLibrary* é o `parseRDFLibrary()`, que monta as estruturas de dados que aparecem na Figura 6. Os outros métodos da classe, como `returnPropertyByClass()` e `returnLibraryTypes()`, foram criados para facilitar o acesso aos dados pelas várias janelas do sistema.

3.3. Inclusão de Recurso

Para os recursos que são inseridos sob o *namespace* gen, ou seja, que referenciam uma classe da biblioteca, o procedimento de inclusão no grafo foi alterado. A Figura 7 mostra a janela de inclusão do IsaViz original, enquanto a Figura 8 mostra como ficou o *dialog* de inserção de recursos no Metabolic IsaViz. A diferença entre as duas é que, no sistema original, informa-se somente a URI ou ID para o novo recurso, enquanto no Metabolic IsaViz informa-se, além da URI, a classe do recurso. Este procedimento também insere automaticamente a propriedade rdf:type no recurso criado, evitando que o usuário insira esta propriedade toda vez que criar um recurso da biblioteca (Veja Figuras 9 e 10) .

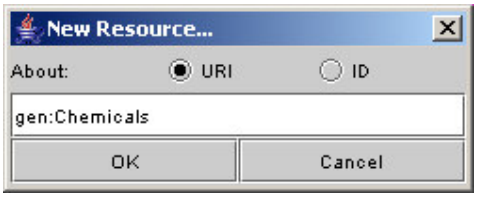

Figura 35 – Criação de recurso no IsaViz original

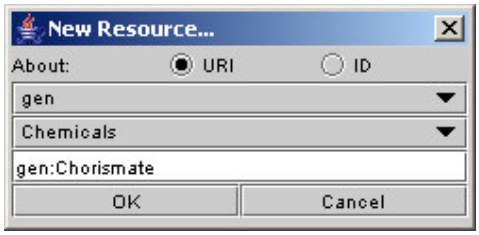

Figura 36 – Criação de recurso no Metabolic IsaViz

Ainda na Figura 8, o primeiro *ComboBox* contém a lista de todos os *namespaces* disponíveis (neste caso, está selecionado o *namespace* "gen", prefixo de `http://www.intelab.ufsc.br/genomica/library#`). Quando o prefixo da biblioteca for selecionado, o segundo *ComboBox* é habilitado, contendo todas suas classes. No exemplo desta Figura, o recurso com URI gen:Chorismate pertence ao *namespace* "gen" e à classe Chemicals. O grafo resultante, com apenas 1 (um) elemento, pode ser visto na Figura 10.

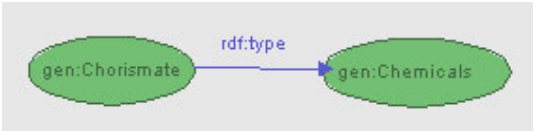

Figura 37 – Grafo RDF no IsaViz original

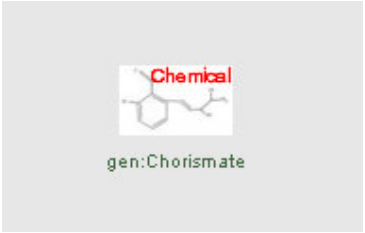

Figura 38 – Grafo RDF no Metabolic IsaViz

As Figuras 9 e 10 representam o mesmo RDF. Com o Metabolic IsaViz, os três passos que o usuário tinha que executar para criar o grafo da Figura 9 – criar o recurso gen:Chorismate, o recurso gen:Chemicals e a propriedade rdf:type – foram diminuídos para apenas um. A propriedade rdf:type e o recurso que indica a classe continuam existindo, mas passam a não ser visíveis e o ícone indica a classe do recurso.

3.4. Visualização das Propriedades de Classe

Na janela *Definitions* foi adicionada a aba *Genomic Properties*. Quando o recurso selecionado no grafo pertencer à Biblioteca Genômica, esta aba irá exibir todas as propriedades da classe à qual o recurso pertence, listando o *namespace*, classe, descrição da propriedade e o valor que ela pode assumir.

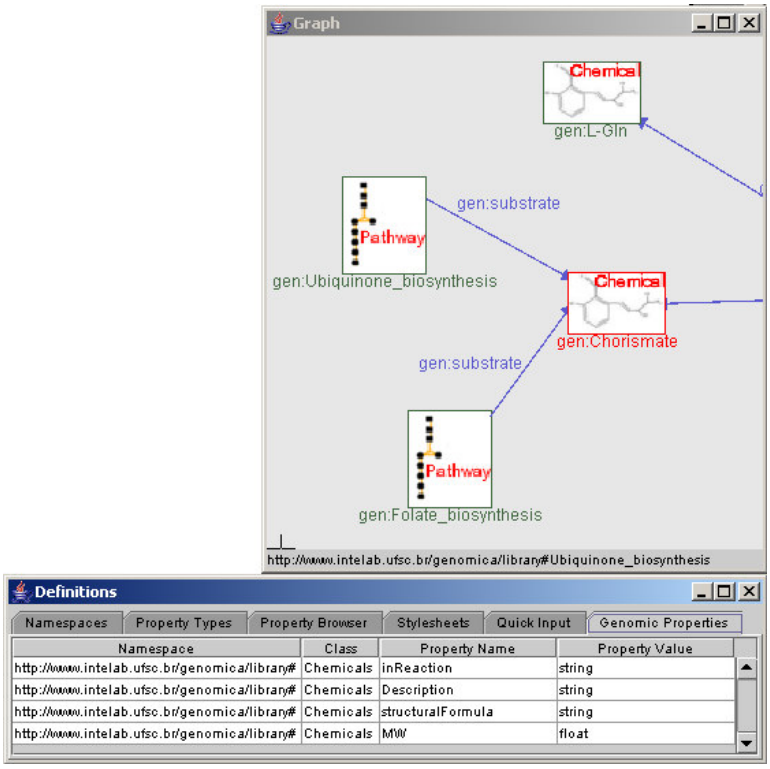

Figura 39 - Efeito da seleção de um recurso da biblioteca na aba *Genomic Properties*

Na Figura 11, o recurso gen:Chorismate aparece selecionado (em destaque); por conseguinte, a aba *Genomic Properties* mostra o *namespace* (das propriedades), o nome da classe, *Chemicals*, suas propriedades, *inReaction*, *Description*, *structuralFormula*, *MW* e o valor de cada uma delas, *string* e *float*.

3.5. Inclusão de Propriedade

Na inserção de propriedades, uma importante modificação foi implementada para facilitar a criação dos modelos. As propriedades (determinadas pela biblioteca) que podem ser inseridas para o recurso selecionado, considerando a sua classe, são mostradas na janela *New Statement*. Pela Figura 12, o recurso gen:Aspartate_biosynthesis, que é uma via metabólica, possui as seguintes propriedades aplicáveis: *product*, *reaction*, *Description*, *global_reaction*, *synonymous*.

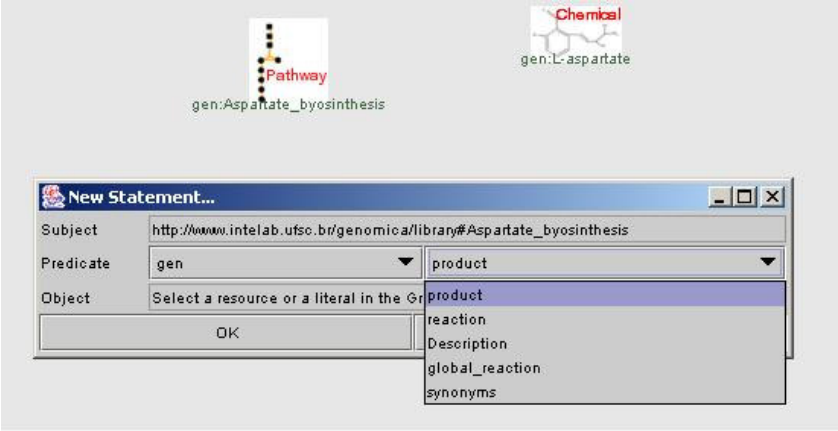

Figura 40 – Inserção de propriedades no Metabolic IsaViz

3.6. Lista de Propriedades

Na janela *Definitions*, aba *Property Types*, que contém a lista de propriedades disponíveis, todas as propriedades presentes na biblioteca foram inseridas.

3.7. Inclusão de Namespace

O *namespace* `http://www.intelab.ufsc.br/genomica/library#`, com prefixo gen, que representa a Biblioteca Genômica, foi incluído na janela *Definitions*, aba *Namespaces*. A partir de então, todos os recursos e propriedades que tiverem o prefixo "gen:" pertencerão à biblioteca.

3.8. Armazenamento do Modelo

Quando os recursos vão ser armazenados, os atributos da classe *IResource* são analisados e um pequeno resumo dessas informações é montado para que, no futuro, o recurso seja "recriado". Para facilitar o processo de recuperação do grafo RDF, o atributo `classUri` (que indica a classe do recurso) foi adicionado a este resumo.

Outra alteração está associada ao armazenamento, junto ao modelo, de todas as imagens utilizadas no grafo. No IsaViz original, as imagens eram salvas em arquivos com uma numeração seqüencial e sem a preocupação de repetição, ou seja, imagens iguais, se utilizadas mais de uma vez, eram salvas, repetidamente, o número de vezes que aparecessem no grafo. No Metabolic IsaViz, a imagem é salva apenas uma vez e todos os recursos que utilizaram a mesma imagem têm uma referência a tal arquivo. Desta maneira, o espaço utilizado para o armazenamento de um modelo diminui consideravelmente, facilitando, por exemplo, o transporte destes arquivos.

4. RESULTADOS

A partir das modificações discutidas acima, o software se mostrou pronto para iniciar a modelagem e criação dos grafos para representação das vias, incluindo seus aspectos de regulação e de cinética enzimática, como será visto a seguir.

4.1. Mecanismo de Criação de Recursos

A modelagem das vias metabólicas com o Metabolic IsaViz se baseia na criação dos recursos (nodos do grafo), informando a que classe pertencem, e atribuindo-lhes propriedades, que graficamente aparecem como as arestas do grafo. Quando é informada a classe do recurso, o sistema inclui no espaço de visualização um ícone que indica a classe escolhida.

O processo de modelagem supõe o conhecimento da Biblioteca Genômica e do domínio das vias bioquímicas. Pelo que foi constatado, uma dificuldade é saber como combinar os recursos da biblioteca, como um composto químico (Chemicals) e uma via (Metabolic_Pathway). Para transpor isto, o usuário pode utilizar a Aba *Genomic Properties* da janela *Definitions* (Vide Seção 3.4) e visualizar quais as propriedades que são aplicáveis à classe.

Além disto, junto com o software estão sendo distribuídos modelos-base, que mostram como representar uma reação, uma seqüência de reações (via metabólica) ou mesmo uma reação com parâmetros cinéticos e enzimáticos.

4.2. Modelos RDF Produzidos

4.2.1. Representação de uma Reação

A Figura 13 mostra o modelo RDF gerado com o Metabolic IsaViz para representar um passo de uma via metabólica, ou seja, uma reação enzimática. O recurso central é o gen:2.5.1.54, que é o objeto que representa a reação. Deste recurso saem duas propriedades gen:left, que representam os substratos: os compostos gen:D-Erythrose-4-phosphate (E4P) e o gen:Phosphoenol-pyruvate (PEP). O produto da reação é o recurso identificado por gen: 7P-2-Dehydro-3-deoxy-D-arabino-heptonate, e é indicado com a propriedade gen:right. Estas são as informações mínimas para se construir uma reação.

Este modelo também denota como representar intersecções entre vias. O recurso E4P é o produto final da via glicolítica, e isto é indicado pela propriedade gen:product que parte de gen:glycolysis.

4.2.2. Reação com Dados sobre Enzima

Neste próximo modelo mostra-se como uma reação pode conter informações sobre a enzima catalisadora, através do recurso Enzymatic_Data. A reação apresentada na Figura 14 é a gen:4.1.3.27, e transforma o Corismato + L-Gln em Piruvato + L-glutamato + Antranilato. Isto pode ser observado pelas propriedades gen:left e gen:right do objeto da classe Reaction.

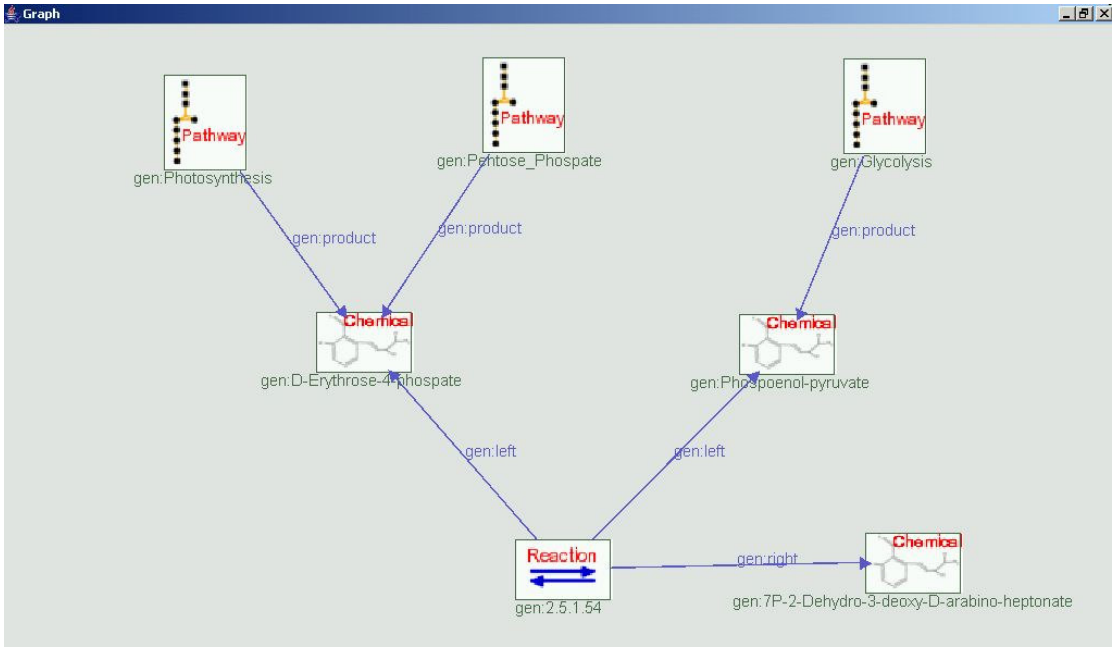

Figura 41 – Representação de uma reação enzimática no Metabolic IsaViz

As informações sobre a enzima são adicionadas com a propriedade gen:catalyst_parameters, cujo valor é um recurso Enzymatic_Data. Esta classe, de acordo com o nome, tem informações sobre a atividade enzimática tais como a enzima, *cofatores*, *ativadores*, *inibidores*, grupos prostéticos e sentido da reação. Na Figura 14, incluiu-se a propriedade gen:enzyme que leva para a enzima antranilato sintase, representada pelo ícone da classe Protein (exibe uma proteína em sua conformação tridimensional).

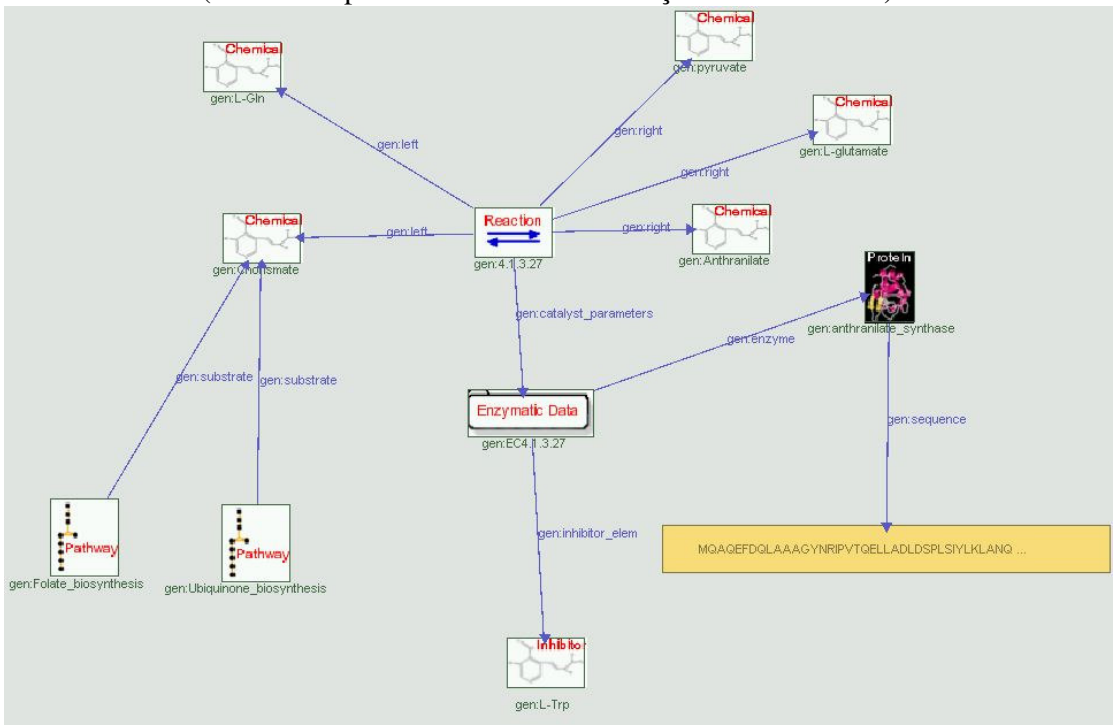

Figura 42 - Representação de uma reação com dados sobre a enzima catalisadora

A propriedade gen:inhibitor_elem, que é o composto L-Triptofano, também é utilizada. Por fim é adicionada a seqüência de aminoácidos da enzima por meio da propriedade gen:sequence da classe Protein.

4.2.3. Uma Reação Englobando Parâmetros Enzimáticos e Cinéticos

Nas Figuras 15 e 16, apresenta-se um modelo de reação que engloba os dados funcionais ou cinéticos e os dados sobre a enzima e seus ligantes, respectivamente.

Na primeira Figura, para adicionar parâmetros de cinética enzimática utiliza-se um recurso da classe Kinetic_Data. O gen:functional_param possui as seguintes propriedades: gen:pH_optimum, gen:Ki, gen:Turnover_number, gen:Specific_activity e gen:Km. Estes parâmetros podem ser utilizados na definição das equações cinéticas, como de Michaelis-Menten e são adicionados ao modelo quando o propósito for de simulação.

A Figura 16 mostra como representar dados sobre os ligantes que participam da reação química. São utilizadas as propriedades gen:inhibitor_elem para denotar os inibidores da reação além das propriedades gen:activators_elem, gen:prosthetic_group e gen:reaction_direction, que fornecem mais detalhes do conhecimento biológico desta reação.

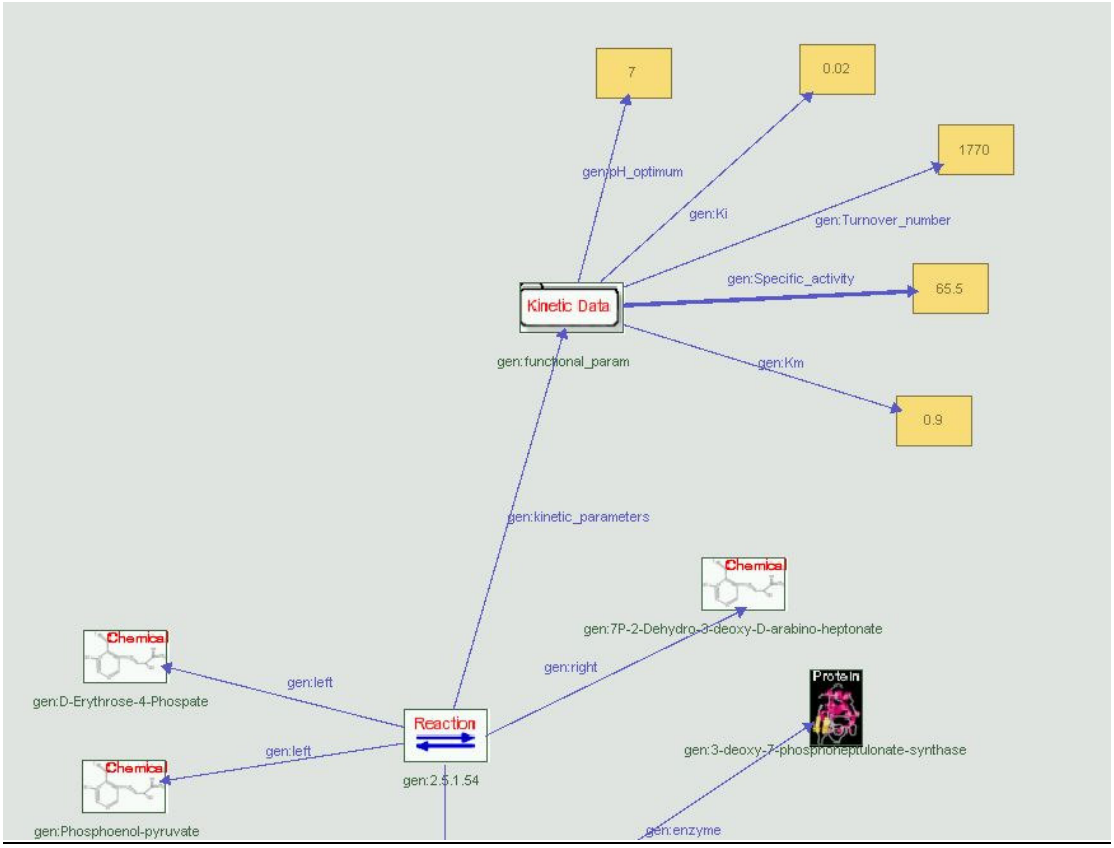

Figura 43 – Dados cinéticos da reação podem ser modelados com um recurso da classe Kinetic_Data

O RDF/XML gerado pelo modelo discutido acima encontra-se na Figura 17. Da linha 37 a 47 ocorre a descrição do recurso gen:edata, da classe Enzymatic_Data. Como pode-se ver, cada propriedade é identificada por elemento sob o elemento rdf:description: gen:inhibitor_elem, gen:enzyme, gen:prosthetic_Group, gen:activator e gen:reaction_direction. Com o mesmo mecanismo é descrito o recurso gen:functional_param, entre as linhas 51 e 58.

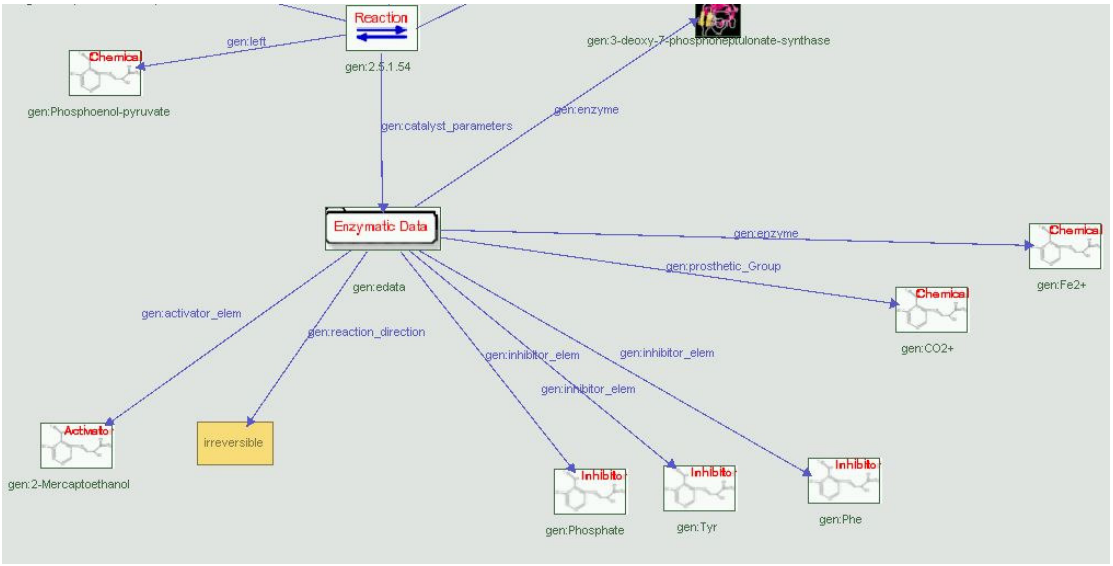

Figura 44 – Representação de ativadores, inibidores e grupos prostéticos

```
30    </rdf:Description>
31    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#CO2+">
32      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
33    </rdf:Description>
34    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Fe2+">
35      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
36    </rdf:Description>
37    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#edata">
38      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Phosphate"/>
39      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Tyr"/>
40      <gen:inhibitor_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#Phe"/>
41      <gen:enzyme rdf:resource="http://www.intelab.ufsc.br/genomica/library#Fe2+"/>
42      <gen:enzyme rdf:resource="http://www.intelab.ufsc.br/genomica/library#3-deoxy-7-phosphoheptulonate-synthase
        "/>
43      <gen:prosthetic_Group rdf:resource="http://www.intelab.ufsc.br/genomica/library#CO2+"/>
44      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Enzymatic_Data"/>
45      <gen:activator_elem rdf:resource="http://www.intelab.ufsc.br/genomica/library#2-Mercaptoethanol"/>
46      <gen:reaction_direction
        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">irreversible</gen:reaction_direction>
47    </rdf:Description>
48    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#Tyr">
49      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Inhibitors"/>
50    </rdf:Description>
51    <rdf:Description rdf:about="http://www.intelab.ufsc.br/genomica/library#functional_param">
52      <gen:Ki rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.02</gen:Ki>
53      <gen:Turnover_number rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1770</gen:Turnover_number>
54      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Kinetic_Data"/>
55      <gen:pH_optimum rdf:datatype="http://www.w3.org/2001/XMLSchema#float">7 </gen:pH_optimum>
56      <gen:Specific_activity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">65.5</gen:Specific_activity>
57      <gen:Km rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.9</gen:Km>
58    </rdf:Description>
59    <rdf:Description
      rdf:about="http://www.intelab.ufsc.br/genomica/library#7P-2-Dehydro-3-deoxy-D-arabino-heptonate">
60      <rdf:type rdf:resource="http://www.intelab.ufsc.br/genomica/library#Chemicals"/>
61    </rdf:Description>
62  </rdf:RDF>
```

Figura 45 – Trecho do código RDF/XML do modelo de uma reação com informações sobre seus parâmetros cinéticos e enzimáticos

4.2.4. Representação de Regulação Transcricional e por Produto

A Biblioteca Genômica disponibiliza ainda classes para representação de alguns níveis de regulação. A Figura 18 mostra o modelo da reação gen:2.3.1.46 envolvendo estes aspectos. A regulação por produto (ou *feedback regulation*), que ocorre entre produtos intermediários de uma mesma via ou de vias diferentes, é representada no grafo pelo recurso gen:reg. A propriedade gen:inhibited_By, que parte deste recurso, especifica qual é o composto que

regula a produção de gen:HSCoA e gen:alpha-succinyl-L-homoserine. A classe Regulation também possui a propriedade gen:activated_By.

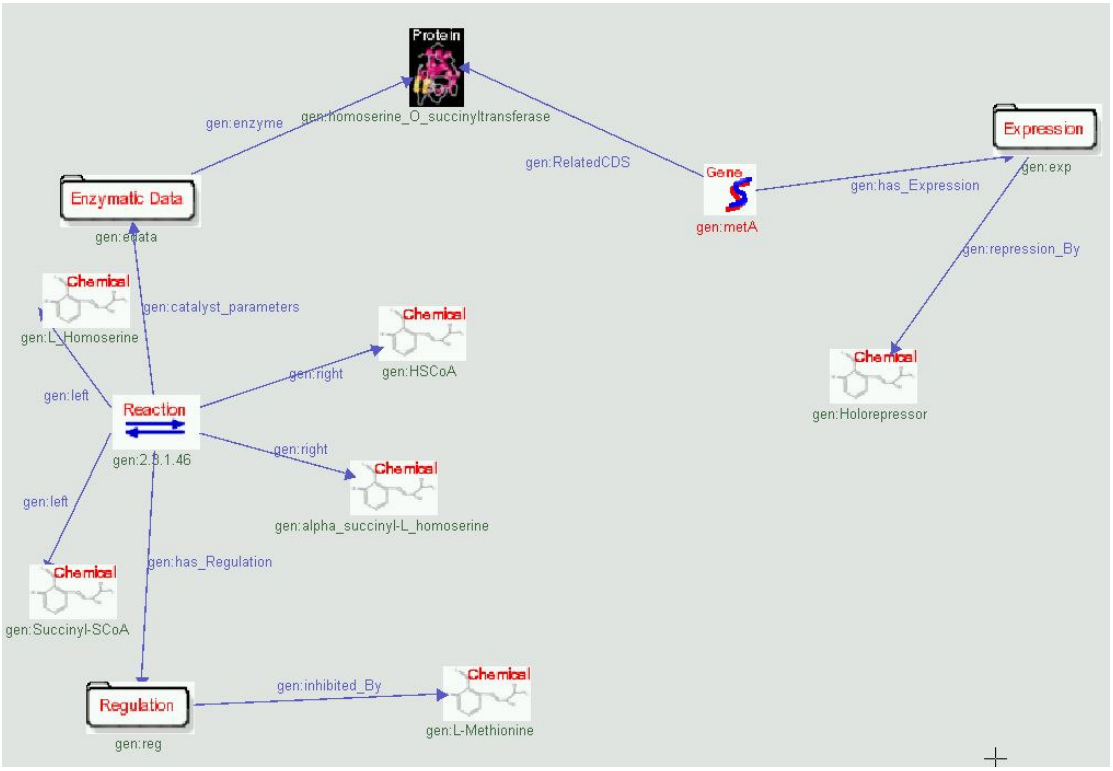

Figura 46 – Representação da regulação a nível de transcrição e entre metabólitos intermediários

Outro tipo de regulação representado no modelo é a regulação a nível de transcrição. Isto ocorre com a repressão da transcrição do gene metA, realizada por um composto conhecido como Holorepressor. Para esta forma de regulação utiliza-se um recurso da classe Expression combinado com a propriedade gen:repression_By. Esta classe também possui as propriedades gen:activated_By e gen:inhibited_By.

5. CONCLUSÕES E TRABALHOS FUTUROS

A extensão do IsaViz implementada neste trabalho pode ser caracterizada por dois itens: a alteração na representação gráfica dos grafos, através de ícones que lembram o tipo do recurso, e a integração com um vocabulário RDF, denominado de *Genomic Library*, com base em uma ontologia no domínio da informação genômica. Notou-se que restringindo um ambiente de modelagem RDF genérico, como o IsaViz, o processo de criação dos modelos foi otimizado. Desta forma, um IsaViz orientado a bibliotecas, como o Metabolic IsaViz, que indica as classes e as propriedades que os recursos podem assumir, torna mais fácil a representação de dados usando o RDF. Além disso, quando houver quaisquer modificações na biblioteca, o software automaticamente estará atualizado, dinamizando e simplificando o trabalho.

Como abordado, a idéia que fundamentou este projeto foi a de utilizar o modelo de dados RDF, um multigrafo direcionado, para a representação do conhecimento acerca das vias metabólicas e seus diferentes aspectos. Este objetivo foi plenamente alcançado, uma vez que os modelos gerados pelo Metabolic IsaViz expressam dados sobre a cinética enzimática,

mecanismos de regulação e interações entre enzima e seus ligantes. Pretende-se que este software seja utilizado como uma ferramenta de análise de vias de interesse, combinado com programas de simulação baseados em equações diferenciais, técnicas de análise em Engenharia Metabólica ou mesmo com simuladores baseados em RPH (Redes de Petri Híbridas). Logo, o Metabolic IsaViz desempenhará um papel de *front-end*, já que através da criação do grafo e de sua representação em XML irá coletar os dados necessários para os ensaios de simulação.

Como trabalhos futuros, sugere-se que o Metabolic IsaViz exporte seus modelos na linguagem de representação padrão para vias metabólicas, a SBML (Systems Biology Markup Language) (SBML, 2003), o que tornaria de fato o software deste projeto possível de ser utilizado em escalas maiores. Outro benefício que advém desta característica seria que os modelos de vias gerados com o Metabolic IsaViz poderiam ser acoplados diretamente com softwares de simulação.

Além disso, novas ontologias podem ser integradas, como é o caso da *Gene Ontology* (Gene Ontology, 2003), cuja finalidade é recomendar os nomes de compostos químicos e suas funções. Por fim, outra melhoria importante seria implementar um gerenciador de biblioteca internamente ao software, evitando a edição manual da Biblioteca Genômica.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ARACYC – Bases de dados para a espécie. Disponível em: <www.arabidopsis.org>. Acesso em: 28 jun. 2003.

SBML – Systems Biology Markup Language. Disponível em: <www.sbml.org>. Acesso em: 28 jan. 2004.

BERNERS-LEE, Tim, HENDLER, James, LASSILA, Ora. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. **Scientific American**, [S.l.], mai. 2001. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2>. Acesso em: 14 fev. 2003.

BRICKLEY, Dan, GUHA, R. V. **RDF Vocabulary Description Language 1.0: RDF Schema**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em: 14 fev. 2003.

CONNOLY, Dan et. al. DAML+OIL (March 2001) Reference Description. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/daml+oil-reference />. Acesso em: 26 jan. 2004.

DUBLIN CORE Metadata Initiative. Disponível em: <http://dublincore.org>. Acesso em: 12 dez. 2003.

Gene Ontology. Disponível em: <www.geneontology.org>. Acesso em: 28 jan. 2004.

KEGG - **Kyoto Encyclopaedia of Genes and Genomes** - Metabolic pathways Disponível em: <http://www.genome.ad.jp/kegg/>. Acesso em: 28 jun. 2003.

KLYNE, Graham, CARROLL, Jeremy J. **Resource Description Framework (RDF):**

Concepts and Abstract Syntax. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em: 14 fev. 2003.

MANOLA, Frank, MILLER, Eric. **RDF Primer**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-primer/>. Acesso em: 14 fev. 2003.

PIETRIGA, Emmanuel. Software. **IsaViz: A Visual Authoring Tool for RDF**. IsaViz 2.0 alpha - preview release (2003-05-12). Disponível em <http://www.w3c.org>. Acesso em: 28 jun. 2003.

PIETRIGA, Emmanuel. **IsaViz User Manual**. IsaViz 2.0 released (2003-08-08). Disponível em <http://www.w3.org/2001/11/IsaViz/usermanual.html>. Acesso em: 16 jan. 2004.

PIETRIGA, Emmanuel. **Graph Stylesheets (GSS) User Manual**. 2003. Disponível em <http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html>. Acesso em: 16 jan. 2004.

SMITH, Michael K. et al. **OWL Web Ontology Language Guide**. [S.l.]. World Wide Web Consortium, 2003. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em: 26 jan. 2004.

van HELDEN, Jacques et. al. **Representing and analysing molecular and cellular function in the computer**. Biol Chem 381(9-10), 921-35, 2000.

VEIGA, Diogo F., PORTO, Luismar M. **XML Schema Representation as a Way to Interchange and Customize Genomic and Metabolic Models.** International Conference on Bioinformatics and Computational Biology, Ribeirão Preto, SP, Brasil, mai. 2003.