CelOWS: UM FRAMEWORK BASEADO EM ONTOLOGIAS COM SERVIÇOS WEB PARA MODELAGEM CONCEITUAL EM BIOLOGIA SISTÊMICA

Ely Edison da Silva Matos

Universidade Federal de Juiz de Fora
Mestrado em Modelagem Computacional

Juiz de Fora, MG - Brasil
Abril de 2008

Matos, Ely Edison da Silva

CelOWS : um framework baseado em ontologias com serviços web para modelagem conceitual em Biologia Sistêmica / Ely Edison da Silva Matos ; orientadora: Profª Drª Fernanda Claudia Alves Campos. – 2008.

134 f. il.

Dissertação (Mestrado em Modelagem computacional)–Universidade Federal de Juiz de Fora, Juiz de Fora, 2008.

1. Ciência da computação – Modelos. 2. Ontologia. 3. Engenharia de software. 4. Biologia. I. Campos, Fernanda Claudia Alves, orientadora. II. Título.

CDU 681.3

CelOWS: UM FRAMEWORK BASEADO EM ONTOLOGIAS COM SERVIÇOS WEB PARA MODELAGEM CONCEITUAL EM BIOLOGIA SISTÊMICA

Ely Edison da Silva Matos

DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE DOS REQUISITOS NECESSÁRIOS À OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.SC.) EM MODELAGEM COMPUTACIONAL.

Aprovada por:

_____

Profª Fernanda Cláudia Alves Campos, D.Sc.

(Orientadora)

_____

Profª Rodrigo Weber dos Santos, D.Sc.

_____

Profª Regina Maria Maciel Braga , D.Sc.

_____

Profª Neide Santos, D.Sc.

JUIZ DE FORA, MG - BRASIL

ABRIL DE 2008

Argue for your limitations
and sure enough, they're yours.
(Richard Bach, Illusions)

DEDICATÓRIA

Dedico este trabalho aos meus pais,
que me deram a oportunidade desta vida
e me ensinaram o que fazer com ela.

AGRADECIMENTOS

Agradeço a Deus, causa primária, inteligência suprema.

A meus pais, Rosemary e Edison, por me oferecerem tudo que o Espírito precisa quando está encarnado e muito mais.

A Ligia, pois tão importante quanto amar é saber que se é amado.

Aos meus meninos, Felipe, Paulo e Daniel, por terem entendido os momentos de ausência e mau-humor neste período, mesmo sem saberem “para que tanto papel e tanto livro”.

A Fernanda, pela paciência, incentivo e perseverança durante a tarefa de orientação.

A Regina, por ter despertado meu interesse pelo assunto em estudo.

Aos membros da Banca Examinadora, pelo trabalho de avaliação.

Ao Carlos Alberto, pelo apoio constante desde que nos conhecemos, reforçado nesta fase em que tive de conciliar estudo e trabalho.

A coordenação do Programa em Modelagem Computacional, pela iniciativa de criação do curso e pelos esforços em fazê-lo crescer.

Aos amigos do CGCO, da FEAK, do CVDEE, pelas oportunidades de trabalho e aprendizado em comum.

Às centenas de pesquisadores espalhados pelo mundo, cujas idéias, artigos e livros publicados permitiram a existência do presente trabalho.

MATOS, Ely Edison da Silva. CelOWS: um framework baseado em ontologias com serviços web para modelagem conceitual em Biologia Sistêmica. Orientadora: Fernanda Cláudia Alves Campos. Juiz de Fora: UFJF/ICE/Faculdade de Engenharia, 2008. Dissertação (Mestrado em Modelagem Computacional).

O advento das tecnologias de alta-performance ocasionou um crescimento exponencial do volume de dados gerado pelas pesquisas em Ciências Biológicas. O desenvolvimento contínuo de ferramentas, métodos e técnicas têm ampliado o entendimento das várias funções, estruturas e processos relacionados à biofísica e fisiologia. O aumento do poder computacional e a utilização de métodos numéricos direcionam o desenvolvimento e uso de modelos mais complexos. A necessidade de entendimento em nível sistêmico dos sistemas biológicos levou à recente definição da chamada Biologia Sistêmica, que adota uma abordagem orientada a sistemas para descrever os processos dinâmicos dentro e entre as células biológicas.

Três níveis de representação de modelos em Biologia Sistêmica podem ser considerados: biológico, matemático e computacional. O desafio apresentado é como representar os detalhes desses níveis de uma forma que possa ser usada para explorar o significado de idéias e observações nos vários níveis. A descrição do modelo em um nível de maior abstração é fundamental neste contexto. A modelagem conceitual permite uma representação abstrata dos dados, a qual se assemelha com a forma que os usuários percebem realmente o mundo real, reduzindo a distância semântica entre o domínio e sua representação.

O presente trabalho propõe a descrição conceitual de modelos biológicos, com o uso de ontologia, lógica descritiva e regras semânticas. É apresentado um framework orientado a serviços, chamado CelOWS, para uso em um ambiente de Modelagem e Simulação, cuja arquitetura é apoiada na ontologia CelO (*Cell Component Ontology*). Os modelos, descritos em uma linguagem de ontologia, são representados computacionalmente como serviços web, podendo ser utilizados em ferramentas de workflow científico para composição de modelos mais complexos.

MATOS, Ely Edison da Silva. CelOWS: an ontology-driven framework using web services for conceptual modeling in Systems Biology. Orientadora: Fernanda Cláudia Alves Campos. Juiz de Fora: UFJF/ICE/Faculdade de Engenharia, 2008. Dissertação (Mestrado em Modelagem Computacional).

The intensive research at biological science has generated large amounts of data. Experimental tools, methods and technologies made possible a better understanding of various functions, structures and processes related to biophysics and physiology. The increase in the computational power and the application of numerical methods lead to development and use of more complex physiological models. The need of understanding of biological systems, at systemic level, lead to recent definition of Systems Biology, that adopts a systems-driven approach to describe dynamic process inside the cells.

One can consider three levels of models in Systems Biology: biological, mathematical and computational. The challenge is how to represent the details of these levels in a way that can be used to explore, at several levels, the ideas and observations. The description of a model with a higher level of abstraction is fundamental. The conceptual modeling allows an abstract data representation, closer to user world, reducing the semantic gap between the domain and its representation.

This work proposes a conceptual description of biological models, using ontologies, description logics and semantic rules. A service-oriented framework named CelOWS is introduced. Its architecture is based on an ontology named CelO (Cell Component Ontology). The models, described using an ontology language, are computationally represented as web services and can be used with scientific-workflows tools to compose more complex models.

LISTA DE SIGLAS

AMS	Ambiente de Modelagem e Simulação
API	*Application Program Interface*
CCO	*Cell Cycle Ontology*
CellML	*Cell Markup Language*
DL	*Description Logics*
DTD	*Document Type Definition*
GO	*Gene Ontology*
KB	*Knowledge Base*
LP	*Logic Programming*
MathML	*Mathematical Markup Language*
OBO	*Open Biomedical Ontologies*
ODE	*Ordinary Differencial Equation*
OWL	*Web Ontology Language*
OWL-S	*Web Ontology Language for Services*
RDF	*Resource Description Framework*
SBML	*Systems Biology Markup Language*
SEI	*Software Engineering Institute*
SI	Sistema Internacional de Unidades
SOA	*Service Oriented Architeture*
SPARQL	*SPARQL Protocol and RDF Query Language*
SWRL	*Semantic Web Rule Language*
UML	*Unified Modeling Language*
UoD	*Universe of Discourse*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. Introdução

1.1. Motivação

As ciências relacionadas com o estudo da vida passaram por uma revolução com a transição dos métodos tradicionais de descoberta in-vivo para os métodos de descoberta in-silico. Esses métodos permitiram reduzir o tempo e o custo associados com a descoberta do conhecimento biológico. Conseqüentemente, uma nova disciplina - chamada Bioinformática - foi criada, unindo em uma única disciplina a Biologia e a Ciência da Computação. A Bioinformática objetiva o gerenciamento e análise dos dados biológicos usando técnicas avançadas de computação (MACEDO, 2005).

O advento das tecnologias de alto desempenho ocasionou um crescimento exponencial do volume de dados gerado pelas pesquisas em Ciências Biológicas. O desenvolvimento contínuo de ferramentas, métodos e técnicas têm ampliado o entendimento das várias funções, estruturas e processos relacionados à biofísica e fisiologia. O aumento do poder computacional e a utilização de métodos numéricos direcionam o desenvolvimento e uso de modelos mais complexos, por exemplo, na área de fisiologia (GARNY et al., 2007). Estes modelos permitem que informações diversas, capturadas de diferentes experimentos físicos e em diferentes escalas, possam ser combinadas, complementando-se mutuamente e provendo uma visão mais acurada das estruturas e processos envolvidos nos fenômenos biológicos.

Desta forma, a Bioinformática é somente um dos passos iniciais na remodelagem das ciências da vida. Para progressos futuros, será necessário o estudo dos sistemas biológicos como um todo, a compreensão do funcionamento dos órgãos (e.g. coração) e dos sistemas associados (e.g. sistema cardiovascular). Este estudo é parte de uma disciplina emergente chamada de Biologia Sistêmica (tradução adotada para Systems Biology) (MACEDO, 2005, KITANO, 2002). A Biologia Sistêmica objetiva desenvolver o entendimento em nível sistêmico dos sistemas biológicos, adotando uma abordagem orientada a sistemas para descrever os processos dinâmicos dentro e entre as células biológicas. Um exemplo dos esforços desenvolvidos nesta direção é o projeto Physiome (HUNTER, 2004).

Embora avanços metodológicos na análise de dados sejam necessários para transformar técnicas experimentais em informação e conhecimento, os problemas na era pós-genômica não serão apenas experimentais ou técnicos, mas também conceituais (WOLKENHAUER, 2003). Um esquema típico do ciclo de pesquisa da biologia computacional é proposto por MACEDO (2005) e apresentado na Figura 1. Primeiramente, (1) modelos qualitativos e modelos quantitativos baseados em dados in vivo e in vitro são propostos e hipóteses são apresentadas. Em seguida (2), simulações são executadas usando propriedades numéricas e discretas do modelo quantitativo, gerando predições sobre o comportamento do sistema. Os resultados das análises (3) sugerem novas hipóteses (análise e interpretação), os quais subsequentemente são testados por experimentos em bancada, iniciando um novo ciclo.

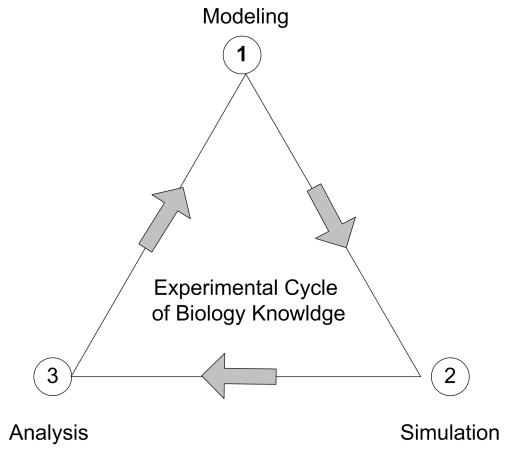

Figura 1. Ciclo experimental do conhecimento da Biologia (MACEDO, 2005).

Para que um modelo possa ser simulado computacionalmente, duas questões importantes devem ser tratadas. A primeira se refere à própria representação do modelo. Embora diagramas, descrição textual e equações possam ser usadas na publicação dos modelos, elas não só estão sujeitas aos erros tipográficos como também à falta de definição das condições iniciais ou de contorno[1] necessárias à simulação. A segunda questão está relacionada à implementação. A necessidade de aplicar métodos numéricos avançados atua como fator limitante para o efetivo uso e estudo do modelo.

Estas questões incentivaram o desenvolvimento da linguagem CellML (*Cell Markup Language*) (CELLML, 2008), uma linguagem de marcação criada

1 Limitações na resolução do modelo; condições de fronteira; restrições.

especificamente para representação de modelos biológicos. Baseada na linguagem XML (*eXtensible Markup Language*) (XML, 2006), CellML especifica elementos que podem ser usados para representar um modelo de maneira formal, sem ambigüidades, legível por humanos e processável por máquina. As equações matemáticas são representadas em MathML (*Mathematical Markup Language*) (MATHML, 2001), o que as torna independentes de uma implementação específica. CellML pode ser usada para representar, armazenar e compartilhar modelos, ampliando sua disponibilidade e facilitando o uso e validação dos mesmos pelos usuários.

CellML, no entanto, não é totalmente adequada para o processo de criação de novos modelos, uma vez que não provê mecanismos de anotação que facilitem o reuso e modificação de novos componentes. Outra questão importante está associada à validação do modelo descrito em CellML. Por ser baseada em XML, a validação do modelo é eminentemente sintática. Através do uso de DTD[2] ou de Schemas (SPERBERG-MCQUEEN, THOMPSON, 2005) o modelo pode ser validado quanto a erros de sintaxe e em relação à aderência à especificação CellML. Porém, questões semânticas não podem ser efetivamente tratadas ou são deixadas para a fase de implementação.

Por outro lado, como observado no Projeto Physiome (HUNTER, 2004), três níveis de representação de modelos em Biologia Sistêmica podem ser considerados: (a) Nível Biológico (representação do problema usando termos do domínio biológico); (b) Matemático (formulação matemática do problema biológico) e (c) Computacional (representação do modelo em uma linguagem formal processável por máquina). Estes três níveis de representação estão intimamente ligados. Conceitos da Biologia estão relacionados a conceitos matemáticos e arquiteturas de simulação, que estão ligados à representação em linguagens computacionais. Em cada nível há uma descrição do modelo, e cada nível possui informações que envolvem os demais níveis.

O desafio que se apresenta, então, é como representar os detalhes desses níveis. A descrição do modelo em um nível de maior abstração é fundamental neste contexto. A modelagem conceitual permite uma representação abstrata dos dados, a qual se assemelha com a forma que os usuários percebem realmente o mundo real, reduzindo

---

2 DTDs (Document Type Definition) estabelecem a gramática válida em um documento XML

(com respeito aos modelos tradicionais) a distância semântica entre o domínio e sua representação.

Assim, o uso de uma linguagem de modelagem conceitual que permita descrever de forma declarativa e reutilizável o domínio de aplicação, usando um vocabulário compartilhado, e o desenvolvimento de ferramentas para que os pesquisadores possam trabalhar com os modelos, são requisitos importantes para o projeto de um framework para modelagem em multiníveis.

1.2. Objetivos

Essa dissertação aborda, em geral, a descrição conceitual de modelos biológicos, com o uso de ontologia, lógica descritiva e regras semânticas e, em particular, os processos de criação, validação, armazenamento, compartilhamento, composição e execução destes modelos em um ambiente de computação orientada a serviço. O trabalho inclui duas das etapas do ciclo de pesquisas biológicas apresentado anteriormente, a modelagem e a simulação. O acrônimo AMS (Ambiente de Modelagem e Simulação) será usado para referenciar o conjunto de ferramentas, modelos e procedimentos adotados pelo pesquisador em sua atividade.

O trabalho propõe um framework para um AMS, organizado em módulos que implementam a execução de cenários de uso em tais ambientes. Esse framework, denominado CelOWS, disponibiliza três serviços: o armazenamento/validação/busca de modelos, a composição de modelos e a execução (simulação) de modelos. Implementado como um serviço web, o framework integra diversas ferramentas necessárias ao processo de modelagem e simulação, de maneira independente da plataforma computacional usada pelo pesquisador. Como os modelos (descritos em uma linguagem de ontologia) são representados computacionalmente também como serviços web, eles podem ser utilizados, posteriormente, em ferramentas de workflow científico para geração de modelos mais complexos.

A construção de uma ontologia implica em adquirir o conhecimento do domínio sendo tratado e coletar as informações apropriadas que definam, com consistência, os termos usados formalmente para descrever tal domínio. Embora a ontologia proposta neste trabalho possa ser aplicada ao domínio mais amplo da fisiologia celular, para fins de pesquisa e avaliação de idéias, o escopo foi restrito ao sub-domínio da

eletrofisiologia da célula cardíaca. Este sub-domínio apresenta um desenvolvimento crescente nos últimos anos e possui diversos modelos já propostos com diferentes níveis de complexidade, o que permite uma avaliação gradativa da ontologia, tanto em termos de completude quanto em termos de funcionalidade.

Assim, a dissertação apresenta uma ontologia particular para descrição conceitual dos modelos, denominada CelO (*Cell Component Ontology*), derivada de um estudo cuidadoso de modelos biológicos existentes, descritos em CellML ou na literatura, associados ao contexto da eletrofisiologia. Além disso, considerando a aplicabilidade de CellML para a descrição e simulação de modelos, além da disponibilidade de centenas de modelos existentes, é possível gerar uma descrição conceitual inicial do modelo CelO a partir de um modelo CellML, bem como é mantida uma referência para o modelo CellML a ser usado no processo de simulação. A implementação, em termos de protótipo, dos principais componentes do framework CelOWS é apresentada, bem como algumas limitações e pontos para expansão futura são discutidos.

A principal contribuição do trabalho consiste na definição de uma arquitetura inicial que permite a integração dos níveis de representação descritos anteriormente, apresentando dois aspectos importantes para a Biologia Sistêmica: o uso de semântica para descrever modelos e uma visão voltada para computação orientada a serviços. A descrição semântica de um modelo facilita o entendimento compartilhado de seus aspectos biológicos, uma vez que componentes, parâmetros, interações e processos são descritos utilizando um vocabulário também compartilhado. A orientação a serviços facilita o uso universal do modelo, independente de plataforma, bem como a sua utilização em ambientes de workflow científicos e grids computacionais.

Algumas questões importantes nortearam o desenvolvimento da ontologia e a implementação do framework CelOWS, e o presente trabalho se propõe a respondê-las:

1. O conhecimento implícito, extraído de um modelo CellML, é realmente significativo?
2. Já existem várias ontologias na área biológica, como OBO (*Open Biomedical Ontologies*) (OBO, 2007), GO (*Gene Ontology*)(GO, 2000), CCO (*Cell Cycle Ontology*) (ANTEZANA, 2006) e CCO (*Cell Component Ontology*) (ZHANG et al., 2005). É realmente necessário desenvolver uma nova ontologia na área?

3. A simples descrição do modelo em uma linguagem de ontologia não é suficiente para integrá-lo em um ambiente de workflow, uma vez que o modelo não é um componente executável. De que forma o uso de ontologias pode efetivamente ser útil neste cenário?
4. A introdução de mais um nível de abstração (no caso deste trabalho, o nível conceitual ou semântico) não torna mais complexo o trabalho de modelagem para o pesquisador?

1.3. Estrutura do trabalho

Esta dissertação está organizada em seis capítulos. O Capítulo 2 apresenta o contexto computacional, fazendo uma revisão dos conceitos associados à modelagem conceitual e ao uso de ontologias. O Capítulo 3 fornece uma visão geral do contexto biológico estudado, relacionado à eletrofisiologia celular, e apresenta a linguagem CellML.

O Capítulo 4 apresenta a ontologia CelO (*Cell Component Ontology*), usada para descrição conceitual dos modelos. A ontologia é parte fundamental do framework proposto, servindo como linguagem comum entre o pesquisador e os serviços oferecidos, permitindo que ele possa definir, redefinir, validar, compor e simular os modelos de forma adequada. A estrutura da ontologia, bem como as regras semânticas utilizadas nos processos de inferência, é descrita. Também são discutidos os métodos utilizados na construção de modelos a partir da extração do conhecimento implícito em modelos CellML.

O Capítulo 5 descreve a arquitetura do framework CelOWS, seus módulos, seus pontos de flexibilização e suas responsabilidades. Quatro cenários de uso comuns em um AMS são apresentados, juntamente com a descrição da implementação computacional do framework e das ferramentas utilizadas. São descritos três procedimentos típicos dos cenários de uso (definição inicial do modelo CelO, composição de modelos e pesquisa de modelos) e os principais trabalhos relacionados.

Finalmente, o Capítulo 6 apresenta as conclusões e sugere trabalhos futuros.

2. Contexto Computacional

2.1. Modelagem Conceitual

Segundo MARJOMAA (2002) a modelagem conceitual tem sido caracterizada de várias maneiras. Entretanto, uma definição importante para essa dissertação é oferecida por KANGASSALO (1992), que apresenta a modelagem conceitual como um processo de formular e coletar conhecimento conceitual sobre um Universo de Discurso, em inglês UoD (*Universe of Discourse*), e documentar os resultados na forma de um esquema conceitual. O termo Universo de Discurso se refere, simplificadamente, ao conjunto de todas as entidades de interesse na realidade que está sendo modelada e seu uso é familiar na Lógica e na Matemática. Na área de Sistemas de Informação, o termo "domínio da informação", ou simplesmente "domínio" é usado com finalidade semelhante.

Este processo de "conceitualização" promovido pela modelagem visa descrever um domínio (ou universo de discurso) através das entidades que o compõe e dos relacionamentos entre essas entidades. Em (SINGH, HUHNS, 2005) o processo de modelagem é apresentado com detalhes, inclusive com as questões relativas à representação da semântica dos modelos.

Nesta seção, são apresentados os conceitos básicos relativos à modelagem, discutidos os diversos níveis em que a semântica pode ser expressa e, finalmente, é feita uma descrição resumida de OWL (*Web Ontology Language*) BECHHOFER et al., 2004) e de SWRL (*Semantic Web Rule Language*) (HORROCKS et al., 2004), as linguagens usadas para desenvolver o modelo semântico proposto nessa disertação.

2.1.1. Modelo, informação e conhecimento

Modelagem é o processo de produzir um modelo, que é uma representação de algum sistema de interesse (MARIA, 1997). Um modelo é similar ao sistema que ele representa, mas muito mais simplificado. Seu propósito básico é permitir ao analista entender o sistema, através da reprodução de seu funcionamento, e predizer os efeitos de mudanças no sistema.

Um modelo é uma abstração de algo, que omite detalhes que não são essenciais e permite lidar com situações e objetos complexos (RUMBAUGH et al., 1994). Através

da abstração, alguns aspectos do mundo real são removidos ou simplificados. A ênfase é colocada nas características essenciais, criando uma “visão”, naturalmente incompleta ou parcial, do ambiente ou contexto modelado. Via de regra a modelagem é feita através de um processo de análise, em que o todo é quebrado em partes componentes que podem ser abordadas separadamente, de forma mais simples. Como são muitos os tipos e quantidade de abstrações que podem ser realizadas, elas são organizadas em níveis. A modelagem conceitual é o processo em que o conhecimento de um domínio é organizado em níveis de abstração a fim de se obter uma compreensão melhor do domínio, encapsulando detalhes. Assim, a descrição de um objeto X em um nível de abstração N1 contém mais detalhes do que a descrição do mesmo objeto X em um nível de abstração N2, se N2 estiver em um nível superior a N1.

Os modelos, sendo abstratos, necessitam de alguma representação feita através de sinais – como ícones, imagens, objetos, tokens e símbolos. No contexto de um dado modelo, estes sinais possuem uma forma e um significado associado. A semântica estuda os aspectos relativos ao significado, enquanto a sintaxe está relacionada à forma. A representação de um modelo, portanto, é feita através de uma linguagem de modelagem que, por sua vez, possui uma sintaxe e uma semântica associadas. Um meta-modelo é um modelo que estabelece a gramática da linguagem utilizada para construir outros modelos.

A partir da construção do modelo, podemos obter informações que sejam de interesse. Informação é uma abstração informal (SETZER, 2006) que representa algo significativo em um certo contexto. A informação pode ser representada através de dados. Em um nível superior de abstração está o conhecimento. Conhecimento pode ser caracterizado como informação combinada com experiência, contexto, interpretação e reflexão (DAVENPORT, 1998). Segundo NONAKA (1997) existem basicamente dois tipos de conhecimento: tácito e explícito. O conhecimento tácito é aquele disponível com as pessoas e que não se encontra formalizado em meios concretos. Já o conhecimento explícito é aquele que pode ser armazenado, por exemplo, em documentos, manuais, bancos de dados ou em outras mídias.

2.1.2. Sintaxe e Semântica

A semiótica é o estudo dos sinais e de como o significado dos sinais é construído, transmitido e compreendido (CARDOSO, 2006). Sinais ou símbolos são geralmente definidos como algo que representa uma outra coisa. No caso específico dos modelos, uma linguagem é usada para representação. Essa linguagem é formada por sinais ou símbolos, usados para construir e transmitir um conhecimento. Modelos formais são construídos com linguagens formais, ou seja, linguagens que possuem regras precisas para sua construção e interpretação. As regras referentes à construção definem a sintaxe da linguagem, enquanto que as regras referentes à interpretação definem sua semântica.

A sintaxe lida com as relações formais ou estruturais entre os sinais ou tokens, bem como com a produção de novos sinais ou tokens. A sintaxe de uma linguagem envolve a definição do conjunto de palavras reservadas, seus parâmetros e a ordem correta em que as palavras serão usadas em uma expressão. Um arquivo XML, por exemplo, usado para interoperabilidade e integração entre sistemas, deve ter uma sintaxe precisa. Se as regras sintáticas não forem observadas, o arquivo não poderá ser processado.

A semântica estuda as relações entre o sistema de sinais (como palavras ou tokens) e seu significado. O objetivo da semântica é totalmente diferente da sintaxe. Esta lida com a estrutura formal em que algo pode ser expresso, enquanto a semântica preocupa-se com o que algo significa. Um aspecto do estudo da semântica, em termos computacionais, é a definição de uma linguagem de representação formal para capturar a semântica de forma processável por máquinas, obtendo uma interpretação consistente. Outro aspecto importante é relacionado à expressividade de tal linguagem, uma vez que a computabilidade é inversamente proporcional à riqueza de conhecimento capturado do mundo real.

Considerando estas questões, três formas de semântica foram definidas em (CARDOSO, 2006):

- Semântica implícita: referente ao significado que está implícito nos dados e que não é representado explicitamente em nenhuma sintaxe processável por máquina;
- Semântica formal: representada em alguma forma sintaticamente bem definida, ou seja, governada por regras sintáticas;

- Semântica *powerful*: habilidade para representar e utilizar conhecimento que é impreciso, incerto, parcialmente verdadeiro ou aproximado (já que a semântica formal envolve limitar a expressividade para permitir características computacionais aceitáveis).

Claramente, um modelo que possua alguma forma de semântica formal é mais expressivo do que um que apresente apenas semântica implícita. Uma forma de agregar alguma semântica aos modelos é através de metadados.

2.1.3. Metadados

Metadados podem ser definidos como “dados sobre os dados” (CARDOSO, 2006). O objetivo de agregar metadados a modelos ou fontes de dados é permitir ao usuário encontrar itens relevantes de acordo com o contexto. O uso de metadados é influenciado pela estrutura dos dados. Os dados podem ser não-estruturados, semi-estruturados ou estruturados. Dados não-estruturados podem ser de qualquer tipo e não seguem necessariamente nenhum formato, regra ou seqüência. Dados semi-estruturados possuem alguma estrutura, mas esta não é rígida. Dados estruturados possuem uma estrutura rígida, descrevendo os objetos através de atributos fortemente tipados.

Metadados podem existir em diferentes níveis. Esses níveis não são mutuamente exclusivos e incluem informação sobre o conteúdo, a estrutura ou a semântica dos dados. Metadados sintáticos descrevem informações não contextuais sobre o conteúdo, geralmente provendo informações de caráter geral (e.g. tamanho do documento, data de criação, etc.). Metadados estruturais, por outro lado, provêm informações sobre a estrutura dos dados, independentes do conteúdo. Eles descrevem como os itens estão organizados no documento e regras para esta organização. Um DTD, por exemplo, apresenta os metadados estruturais de um documento XML. Metadados semânticos acrescentam regras, relacionamentos e restrições aos metadados sintáticos e estruturais.

Metadados semânticos descrevem informações sobre os dados, que são importantes em dado contexto ou domínio, permitindo uma certa interpretação. Dados semânticos provêm um meio para pesquisas de alta precisão e possibilitam a interoperabilidade entre sistemas ou fonte de dados heterogêneos. Estes dados são usados para fornecer significado aos elementos descritos pelos metadados sintáticos ou estruturais. Um aspecto importante, na criação de um meta-modelo semântico para os

dados, é a possibilidade de usar a capacidade de inferência para obter conclusões lógicas baseadas no meta-modelo, de acordo com o nível de semântica adotado (CARDOSO, 2007).

2.1.4. Níveis de Semântica

Segundo (CARDOSO, 2007), dependendo da abordagem, modelos e métodos usados para adicionar metadados semânticos, quatro representações podem ser usadas para organizar os conceitos que descrevem semanticamente os termos: vocabulários controlados, taxonomias, thesaurus e ontologias.

Um vocabulário controlado é uma lista de termos que foram enumerados explicitamente. Todos os termos têm uma definição não redundante e sem ambigüidade. Este é o método mais simples de metadados e tem sido extensivamente usado para classificação. Por exemplo, em um catálogo de produtos é apresentada ao usuário uma lista de opções, limitadas ao conjunto de termos pré-definidos. Seu principal objetivo é evitar que os usuários definam seus próprios termos, que poderiam ser ambíguos, pouco significativos ou sintaticamente errados.

Uma taxonomia é um vocabulário controlado que classifica os termos com base no assunto e os organiza em uma estrutura hierárquica. Um termo é descrito através de um relacionamento explícito com outro termo. Este relacionamento é do tipo “pai-filho”, tal como “x é subclasse de y” ou “y é superclasse de x”. Os usuários da taxonomia podem compreender o significado de um termo analisando os relacionamentos entre o termo e os termos em torno na hierarquia.

Um thesaurus é um vocabulário controlado com relacionamentos conceituais entre os termos. Um thesaurus estende uma taxonomia permitindo outros relacionamentos entre os termos, além do hierárquico. De acordo com (NISO, 2005), são usados quatro tipos diferentes de relacionamentos em um thesaurus: equivalência (mesmo significado), homografia (mesma grafia com diferente significado), hierarquia (subclassificação) e associação (significados próximos, sem relacionamento hierárquico).

2.2. Ontologias

Segundo ALMEIDA (2003), historicamente o termo ontologia tem origem no grego *ontos*, ser e *logos*, palavra. É um termo introduzido na filosofia com o objetivo de

distinguir o estudo do ser humano como tal, do estudo de outros seres das ciências naturais. A origem é a palavra aristotélica “categoria”, que pode ser usada para classificar e caracterizar alguma coisa.

Na área da Ciência da Computação, uma ontologia define uma especificação formal e explícita dos termos de um domínio e das relações entre eles, de uma conceitualização compartilhada (GRUBER, 1992; GRUBER, 1994; BORST, 1997). ALMEIDA (2003) explica esta definição, dizendo que “formal” significa legível para computadores; “especificação explícita” diz respeito a conceitos, propriedades, relações, funções, restrições, axiomas que são explicitamente definidos; “compartilhado” quer dizer conhecimento consensual; e, “conceitualização” diz respeito a um modelo abstrato de algum fenômeno do mundo real.

Uma ontologia provê um mecanismo para capturar a compreensão comum sobre objetos e seus relacionamentos, em um certo domínio de interesse, e prover um modelo formal e manipulável desse domínio. A especificação formal do significado dos termos utilizados possibilita a criação de novos termos através da combinação dos já existentes, bem como permite a integração com outras ontologias.

O estudo e o uso de ontologias na área de software foram popularizados com a idéia da **web semântica** introduzida em (BERNERS-LEE, HENDLER, LASSILA, 2001). Nesse artigo os autores apresentam a visão da **web semântica** como uma extensão da web atual, na qual a informação recebe um significado bem definido, permitindo o trabalho cooperativo de computadores e pessoas. Neste contexto, o tipo mais comum de ontologia consiste de uma taxonomia e um conjunto de regras de inferência, que permitem capturar o conhecimento que não está explícito na taxonomia.

As ontologias podem estender os relacionamentos hierárquicos das taxonomias, permitindo relacionamentos horizontais entre os termos, em uma estrutura tipo grafo. Desta forma, as ontologias facilitam a modelagem de requisitos de informação típicos do mundo real. Sua expressividade é ampliada com o uso de máquinas de inferência. Segundo CORCHO (2007) diferentes formalismos para representação do conhecimento existem para implementação de ontologias. Embora os componentes de cada um sejam diferentes, um conjunto mínimo é comum a todos:

- Classes: representam conceitos do domínio; são definidas por termos geralmente organizados em uma taxonomia;

- Relações: representam um tipo de associação entre as classes. Geralmente as associações são binárias, com o primeiro argumento sendo chamado **domínio** (*domain*) da relação e o segundo argumento sendo o **limite** (*range*). As relações são instanciadas segundo o conhecimento disponível sobre o domínio. As relações binárias são também usadas para expressar atributos ou propriedades das classes. Atributos são diferentes de relações, porque estão limitadas a um tipo de dados (e não a uma classe).
- Instâncias: representam elementos ou indivíduos em uma ontologia.

Uma importante propriedade das ontologias é que sua representação permite o processamento computacional da mesma, sendo baseada em linguagens lógicas, o que permite a definição formal da semântica dos conceitos.

2.2.1 Linguagens Lógicas para Representação do Conhecimento

O uso de linguagens lógicas para representação do conhecimento permite escrever declarações sem ambigüidade e derivar novos conhecimentos que estão implícitos nas descrições existentes. Três linguagens lógicas, comumente associadas a ontologias, são a Lógica de Primeira Ordem, a Lógica Descritiva e a Programação Lógica (BRUIJIN, 2007).

Lógica de Primeira Ordem

A Lógica de Primeira Ordem é um sistema de raciocínio no qual cada sentença, ou declaração, é dividida em sujeito e predicado. O predicado modifica ou define as propriedades do sujeito. Em Lógica de Primeira Ordem, um predicado pode se referir a um único sujeito. Uma sentença em lógica de primeira ordem é escrita na forma Px ou P(x), onde P é o predicado e x é o sujeito, representado como uma variável. Sentenças completas são combinadas logicamente e manipuladas de acordo com as mesmas regras usadas na álgebra booleana. Uma descrição mais completa é feita em (BRUIJIN, 2007).

A Lógica de Primeira Ordem é de interesse nas pesquisas em Inteligência Artificial pela capacidade de expressar raciocínios mais comuns. No entanto, é também uma linguagem muito expressiva, o que torna difícil seu uso em termos computacionais, uma vez que o processo de inferência é muito difícil e os problemas mais interessantes são geralmente não-decidíveis (RUSSELL, NORVIG, 1995). Por isso, subconjuntos

desta lógica foram amplamente estudados e formam a base das linguagens usadas comumente com ontologias na **web semântica**.

Lógica Descritiva

As Lógicas Descritivas, em inglês DL (*Description Logics*), constituem uma família de formalismos baseados em lógica de primeira ordem para representação do conhecimento (NARDI, 2003). Segundo LUTZ (2006), elas são consideradas o formalismo mais importante para representação de conhecimento, unificando e fornecendo uma base lógica para os sistemas tradicionais nesta área (*frames*, redes semânticas, representações orientadas a objeto, modelos semânticos de dados e sistemas de tipos). As Lógicas Descritivas são utilizadas no projeto de sistemas que possuem uma linguagem para definir uma base de conhecimento, em inglês KB (*Knowledge Base*), e ferramentas para realizar inferências sobre a base definida.

Com Lógicas Descritivas a representação do conhecimento é feita através de uma abordagem funcional, ou seja, são fornecidas especificações precisas das funcionalidades a serem providas pela base de conhecimento e das inferências a serem realizadas. Segundo NARDI (2003), na prática, a descrição funcional do sistema é feita através de uma interface chamada “Tell&Ask”, que especifica operações que possibilitem a construção da base de conhecimento (operações Tell) e operações que permitam obter informações a partir da base (operações Ask).

Dentro da base de conhecimento pode ser feita uma clara distinção entre “conhecimento intensional” (o conhecimento geral sobre o domínio do problema ou universo do discurso) e “conhecimento extensional” (conhecimento específico em um problema em particular). De forma análoga, a arquitetura de um sistema DL vai possuir dois componentes – um “TBox” e um “ABox”. O TBox contém o conhecimento intensional, na forma de uma terminologia ou taxonomia, sendo construído através de declarações que descrevem os conceitos gerais e suas propriedades. O ABox contém o conhecimento extensional, também chamado de conhecimento “assercional”, que é específico para os indivíduos do domínio. A Figura 2 (HORROCKS, 2002) mostra a arquitetura típica de um sistema baseado em Lógicas Descritivas.

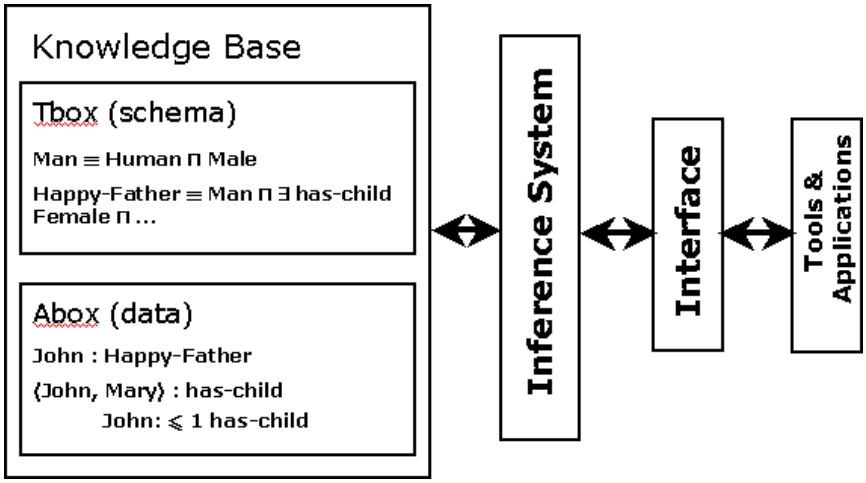

Figura 2. Arquitetura de um sistema DL (HORROCKS, 2002).

As Lógicas Descritivas descrevem o domínio em termos de conceitos (classes), papéis (propriedades, relacionamentos) e indivíduos. Em termos de lógica, os nomes de conceitos são equivalentes a predicados unários – em geral, fórmulas com uma variável livre. Os nomes de papéis são equivalentes a predicados binários – em geral, fórmulas com duas variáveis livres. Os nomes de indivíduos são equivalentes a constantes. O conjunto de operadores é restrito para que a linguagem seja decidível e com baixa complexidade.

Um aspecto importante no uso de Lógicas Descritivas são os **mecanismos de inferência,** em inglês *DL Reasoners*. Através destes mecanismos, não apenas a base de conhecimento pode ser validada em relação à sua correção, mas também o conhecimento implícito pode ser explicitado a partir do conhecimento expresso na base. Estes dois aspectos podem ser reduzidos às seguintes tarefas de inferência (RUSSELL, NORVIG, 1995):

- satisfatibilidade: um conceito C é satisfatível com respeito a um TBox T se existe pelo menos um modelo de T onde a interpretação de C, C', é não-vazia. Esta tarefa é também denominada classificação, ou seja, a verificação se um objeto pertence a uma classe;
- subsunção: um conceito C é subordinado a um conceito D, com respeito a T, se e somente se C' $\subseteq$ D' para cada modelo de T. Ou seja, a verificação se uma classe é subconjunto de outra pela comparação de suas definições.

A inferência na maioria das Lógicas Descritivas é decidível e existem algoritmos otimizados para algumas lógicas. Várias implementações destes algoritmos têm sido usadas com as linguagens de ontologia, tais como FACT++ (FACT, 2008), Racer (RACER, 2008) e Pellet (SIRIN et al., 2007).

Programação Lógica

A Programação Lógica, em inglês LP (*Logic Programming*), é baseada na Lógica Horn (HORN LOGIC, 2006), que também é um subconjunto da Lógica de Primeira Ordem. A semântica da Programação Lógica, porém, é levemente diferente da Lógica de Primeira Ordem. Um programa lógico consiste de regras na forma "se A então B". Se A é verdadeiro, então B deve ser verdadeiro. A Programação Lógica é importante no contexto da **web semântica** por ser usada na inferência com as linguagens para ontologia e também por representar o conhecimento na forma de **regras** (BRUIJIN, 2007).

As regras podem ser vistas como um paradigma de representação do conhecimento complementar à Lógica Descritiva. A Lógica Descritiva é considerada conveniente para definir classes, hierarquias, propriedades e relacionamentos entre eles (BRUIJIN, 2007). Através dela é possível expressar quantificação existencial, disjunção e negação clássica. Programas Lógicos, por outro lado, podem expressar predicados com aridade arbitrária, encadeamento de variáveis sobre predicados e a negação não-monotônica (negação como falha).

O exemplo clássico que mostra o poder das regras sobre as ontologias expressas em Lógicas Descritivas é (MATHEUS et al., 2005): "se x é irmão de um dos pais de y, então x é um tio de y":

```
irmaoDe(X,Z) and filhoDe(Y,Z) → tioDe(X,Y)
```

O relacionamento tioDe(X,Y) requer a possibilidade de restringir o valor de uma propriedade (irmaoDe) de um termo (X) para que ele seja o valor de uma propriedade (filhoDe) de um outro termo (Y). Esse exemplo não pode ser expresso através de Lógica Descritiva, pois as variáveis X e Y são usadas em ambos os lados da implicação, o que não é possível em lógica descritiva.

2.2.2. OWL – Web Ontology Language

Segundo BERNERS-LEE, HENDLER e LASSILA (2001), uma linguagem para representar ontologias deve possuir uma sintaxe e uma semântica bem definidas, ter suporte a inferência, ser eficiente e expressiva. Em 10 de fevereiro de 2004 o W3C (*World Wide Web Consortium*) apresentou como recomendação formal a OWL (*Web Ontology Language*), uma linguagem de ontologia para Web, baseada em lógica descritiva.

A OWL foi projetada para atender às necessidades de uma linguagem de ontologia para a Web, visto que as linguagens que a precederam possuem algumas limitações. XML fornece uma sintaxe para documentos semi-estruturados, mas não associa semântica aos marcadores. Assim, os marcadores XML não possuem nenhum significado no processamento do documento por máquinas. XML *Schema* apenas estende a XML com um conjunto de tipos de dados e fornece um esquema para documentos XML. RDF (*Resource Description Framework*) (RDF, 2002) padroniza a definição e uso de metadados, mas apresenta um modelo de dados muito simples, baseado em triplas (sujeito, predicado, objeto), para representar o relacionamento entre recursos[3]. A RDF *Schema* (RDF, 2004) provê um sistema de tipos para RDF, que permite aos usuários definir recursos com classes, propriedades e valores. No entanto, algumas características tais como restrição de cardinalidade, disjunção de classes e escopo local de propriedades, não podem ser expressas em RDF *Schema*.

OWL é o padrão recomendando pelo W3C para ontologias Web. Ela é construída sobre RDF e RDF *Schema*, usando uma sintaxe RDF baseada em XML. As instâncias são definidas através de descrições RDF e a maioria das primitivas de modelagem RDF são usadas.

Dialetos OWL

OWL é formada por um conjunto de três sub-linguagens, ou dialetos, com diferentes níveis de expressividade:

- OWL *Lite*: suporta as necessidades de classificação hierárquica e restrições mais simples;

---

3 Recursos (resource) são objetos identificado univocamente através de uma URI (*Uniform Resource Identifier*)

- OWL DL: é mais expressiva que OWL Lite, mas, ainda se mantém completa e decidível, ou seja, todos os cálculos de inferência têm computação e término garantidos. A sigla DL se refere a *Description Logic* (Lógica Descritiva), usada como base da sub-linguagem;
- OWL *Full*: possui máxima expressividade, mas não há garantia de decidibilidade. Usa a sintaxe OWL, sendo possível a combinação arbitrária das primitivas OWL com RDF e RDF *Schema*.

Conforme é apresentado em (OWL, 2004), a escolha de qual sub-linguagem OWL os desenvolvedores de ontologias devem usar depende das necessidades da ontologia. A escolha entre OWL *Lite* e OWL DL dependerá da necessidade das propriedades computacionais de OWL *Lite* ou das construções mais expressivas providas pela OWL DL. A escolha entre OWL DL e OWL *Full* dependerá da necessidade de expressividade, decidibilidade e completude computacional da OWL DL ou da expressividade e das facilidades do meta-modelo RDF *Schema* sem a previsibilidade computacional de OWL *Full*.

Estrutura das ontologias OWL

O objetivo da OWL é prover uma linguagem de ontologia que possa ser usada para descrever, de um modo natural, classes e relacionamentos entre classes em documentos e aplicações Web. Os termos usados em uma ontologia devem ser escritos de forma que eles possam ser usados por diferentes softwares.

OWL distingue entre construtores e axiomas (SINGH, HUHNS, 2005). Construtores OWL são as primitivas usadas para especificar novas classes e os axiomas são as primitivas para fazer asserções adicionais sobre as classes e propriedades. Os dialetos OWL provêem construtores de classes baseados em lógica descritiva. Estes construtores usam os tipos de dados definidos no XML Schema.

Inferência em OWL

Como visto anteriormente, os mecanismos de inferência permitem explicitar conhecimentos que estão implícitos em uma base de conhecimento. Como é baseada em lógica descritiva, OWL possibilita o uso destes mecanismos. Como conseqüência, um documento OWL não deve ser considerado apenas sob o ponto de vista de sua sintaxe, mas também de sua semântica. Isso significa que dois documentos superficialmente

diferentes em termos sintáticos podem expressar o mesmo conhecimento, se eles legitimam as mesmas inferências.

Segundo SINGH e HUHNS (2005) a vantagem de se pensar em termos de inferências é que conhecimentos de diferentes fontes podem ser colocados juntos. Modelos conceituais diferentes podem ser combinados e mapeados um em relação ao outro. Os mecanismos de inferência podem detectar potenciais inconsistências, ajudando a resolver erros rapidamente.

2.2.3. SWRL – Semantic Web Rule Language

Muitos processos de negócio, tais como gerenciamento de workflows, treinamento auxiliado por computador, diagnósticos e controle de processos são freqüentemente melhor modelados através de uma abordagem declarativa (O'CONNOR et al., 2005), através de sistemas baseados em regras. Considerando a quantidade e variedade destes sistemas, a SWRL (*Semantic Web Rule Language*) é uma proposta para padronização de uma linguagem de regras, visando interoperabilidade entre estes vários sistemas.

A SWRL é baseada em uma combinação de duas sub-linguagens de OWL (OWL *Lite* e OWL DL) com uma sub-linguagem de RuleML (*Rule Markup Language*) (RULEML, 2008). A SWRL permite aos usuários escrever regras em lógica Horn em termos de conceitos OWL, para realizar inferência sobre indivíduos. As regras podem ser usadas para inferir novos conhecimentos a partir de uma base de conhecimento expressa em OWL.

Como em outras linguagens de regras, as regras SWRL são escritas em pares antecedente-consequente. Nessa terminologia, o antecedente é o corpo (*body*) da regra, enquanto o conseqüente é a cabeça (*head*) da regra. O corpo e a cabeça consistem de uma conjunção de um ou mais átomos (*atoms*). Na proposta submetida ao W3C (SWRL, 2004), SWRL não suporta combinações lógicas de átomos mais complexas.

As regras SWRL realizam inferência sobre indivíduos OWL, em termos de classes e propriedades OWL. Por exemplo, utilizando o exemplo apresentado na seção 2.2.1, uma regra expressando que o irmão de um dos pais de uma pessoa é tio desta pessoa requer a captura de conceitos de ‘pessoa’, ’pais’, ’irmão’ e ‘tio’ em OWL. Esta regra poderia ser escrita como:

```
Person(?x1) ^ hasParent(?x1,?x2) ^ hasBrother(?x2,?x3) → hasUncle(?x1,?x3)
```

onde *Person* é uma classe e *hasParent*, *hasBrother* e *hasUncle* são propriedades OWL. A execução da regra tem o efeito de definir a propriedade *hasUncle* para x3 em todos os indivíduos x1 que satisfazem a regra.

As regras SWRL também podem se referir explicitamente a indivíduos OWL. Por exemplo, a regra seguinte é uma variação da regra anterior, inferindo que um indivíduo específico, Fred, tem um tio:

```
Person(Fred) ^ hasParent(Fred,?x2) ^ hasBrother(?x2,?x3) → hasUncle(Fred,?x3)
```

SWRL também suporta os conceitos de igual-a (*same-as*) e diferente-de (*different-from*). Por exemplo, o átomo *sameAs* pode determinar se dois indivíduos Fred e Fredrick são o mesmo indivíduo:

```
sameAs(Fred,Frederick)
```

SWRL também suporta o uso de vários predicados pré-construídos (*built-in*), o que expande seu poder de expressão. Estes predicados podem aceitar vários argumentos e são descritos em detalhes na especificação de SWRL. Os *built-ins* mais simples são operações de comparação. Por exemplo, o *built-in greaterThan* pode ser usado para determinar se um indivíduo tem um irmão mais velho:

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^ hasAge(?x2,?age2) ^
swrlb:greaterThan(:age2,?age1) → hasOlderBrother(?x1,?x2)
```

2.3. Comentários Finais

Este capítulo apresentou o contexto computacional em que o trabalho está inserido, fazendo um revisão de conceitos associados a modelagem conceitual e ontologias. Modelos conceituais descrevem um domínio de forma simplificada, através da representação das entidades que compõe esse domínio e dos relacionamentos entre essas entidades. Ontologias podem ser usadas para representar o conhecimento expresso no modelo de maneira formal, através do uso de linguagens lógicas.

O capítulo também introduziu as linguagens OWL (Web Ontology Language) e SWRL (Semantic Web Rule Language), utilizadas no trabalho para representação de modelos biológicos, que são discutidos no Capítulo 3, a seguir.

3. Contexto Biológico

As células são as unidades básicas, tanto em termos funcionais quanto estruturais, do corpo humano. A fisiologia celular estuda as funções da célula e compreende essas como interações de vários elementos funcionais. A eletrofisiologia celular estuda as propriedades elétricas das células, envolvendo as medidas de potencial e corrente elétrica. Modelos matemáticos que representam o comportamento dinâmico de cada elemento funcional são continuamente propostos por pesquisadores. No campo da fisiologia e da eletrofisiologia, estes modelos são geralmente formulados por sistemas de equações diferenciais ordinárias (SHIMAYOSHI et al., 2006).

Ainda segundo SHIMAYOSHI et al. (2006), modelos dos elementos funcionais podem ser compreendidos como componentes usados na construção de modelos da célula inteira. Um sistema de equações diferenciais é obtido através da combinação das equações dos modelos componentes. As variáveis desse sistema representam quantidades referentes a objetos ou eventos que ocorrem na atividade celular, tais como concentração de íons, potencial da membrana ou condutância dos canais iônicos.

Esta seção tem por objetivo apresentar os conceitos básicos do domínio da eletrofisiologia da célula cardíaca, bem como a linguagem CellML (CELLML, 2008), usada para representar computacionalmente modelos deste domínio.

3.1. Eletrofisiologia da célula cardíaca

Segundo NICKERSON (2006), a modelagem matemática da eletrofisiologia da célula cardíaca recebeu um significativo desenvolvimento nas últimas décadas devido, principalmente, à evolução da tecnologia, que gerou melhores técnicas experimentais e compiladores de alto desempenho. Os modelos em eletrofisiologia atualmente englobam desde modelos simples da ativação elétrica baseados em polinômios até modelos estocásticos complexos tridimensionais.

A célula tem seu interior delimitado por uma membrana que controla o fluxo das substâncias que entram e saem da célula. Esta membrana é constituída de uma bi-camada fosfolipídica que impede o transporte de moléculas polares e íons. Como existe uma diferença na composição iônica dos meios intra- e extra-celular, com predominância de potássio no interior e sódio, cálcio e cloro no exterior, um gradiente

eletroquímico é gerado entre os dois meios. O chamado **potencial de repouso** (*resting potential*) da membrana é a diferença de potencial entre o interior e o exterior da célula (cerca de –90 mV nas células musculares e fibras de Purkinje e cerca de –55 mV no nó atrioventricular). A atividade elétrica do coração é conseqüência dos fenômenos elétricos gerados por esta diferença de potencial (MALMIVUO, PLONSEY, 1995).

A membrana possui várias características elétricas: resistência (dificuldade para livre fluxo dos íons), condutância (permissividade à condução de íons) e capacitância (capacidade de receber e liberar cargas elétricas). A constituição da membrana a torna semi-permeável, existindo arranjos especiais formados por proteínas que permitem a passagem de íons para dentro ou para fora da célula. Estes arranjos são denominados canais iônicos (*ion channels*). Estes canais são pólos macromoleculares através dos quais íons de sódio (Na), potássio (K) e cloro (Cl) fluem através da membrana. Os canais são especializados e somente uma substância ou um pequeno grupo de moléculas pode passar através de um canal em particular. O fluxo (ou corrente) de íons é determinado pelo gradiente eletroquímico existente entre o interior e o exterior da célula. Além dos canais iônicos, o transporte de íons também é realizado por bombas de íons (*ion pumps*) e permutadores Na+/Ca+ (*ion exchangers*). As bombas de íons são responsáveis por realizar o transporte ativo, ou seja, com gasto de energia (ATP) pela célula, uma vez que o transporte é feito contra o gradiente eletroquímico.

Existem centenas de canais iônicos diferentes e um dos mecanismos de classificação dos canais está baseado no mecanismo denominado *gating*. Esse mecanismo se refere ao fato do fluxo de íons através dos canais iônicos ser regulado por *gates* (portas), que podem estar abertos ou fechados. O estado do *gate* no canal iônico é regulado por diversos fatores, tais como sinais elétricos ou químicos, temperatura ou forças mecânicas. Por exemplo, o estado do *gate* pode estar associado à diferença de potencial elétrico através da membrana (*voltage-gated íon channel*) ou a ligação com moléculas sinalizadoras (*ligand-gated íon channel*) (MALMIVUO, PLONSEY, 1995).

A diferença de potencial na membrana (voltagem da membrana ou voltagem transmembrânica) sofre alterações na recepção de um estímulo. Fenômenos eletroquímicos com liberação de correntes iônicas ocorrem com a recepção do estímulo, reduzindo o potencial elétrico até atingir um certo limiar. A partir daí é desencadeado um **potencial de ação** (*action potential*)**,** um impulso elétrico transiente que percorre a

membrana (Figura 3). Existem dois tipos de respostas ao potencial de ação (rápida ou lenta) dependendo do tipo de célula. Durante a resposta ao potencial de ação ocorrem os fenômenos de fluxo de íons entre o interior e o exterior da célula, bem como os fenômenos de despolarização e repolarização devido à mudança na concentração de íons. A despolarização é caracterizada por uma mudança de potencial na direção positiva. A hiperpolarização (ou repolarização) acontece no sentido contrário. O entendimento destes fenômenos é crucial no estudo do funcionamento da célula.

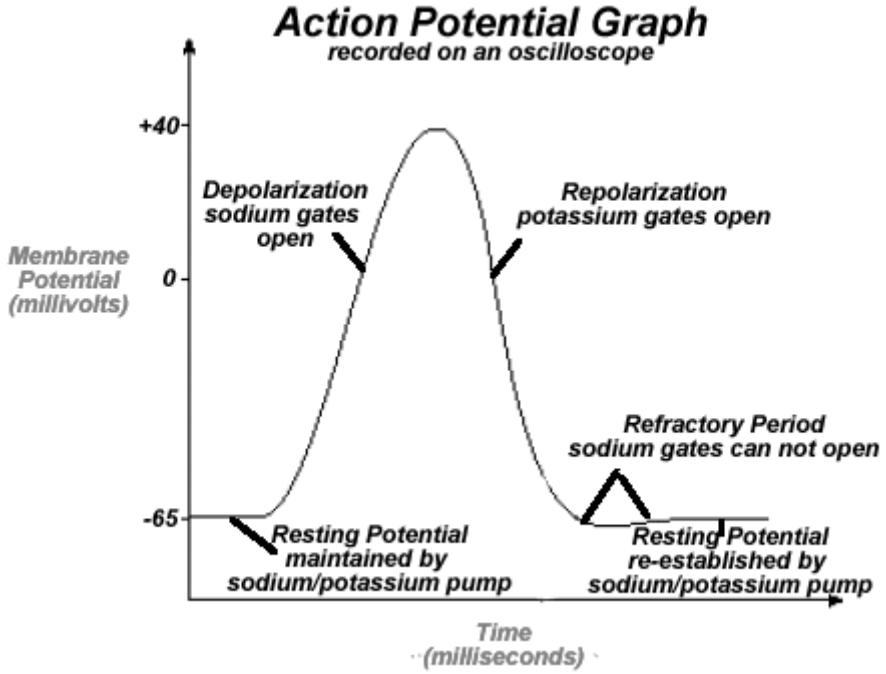

Figura 3. Variação do potencial da membrana e do estado dos *gates* associados ao potencial de ação – (COOLSCHOOL, 2008)

Diversos modelos foram (e ainda são) criados para representar e compreender os fenômenos associados à atividade elétrica na célula. A seção seguinte apresenta brevemente a estrutura desses modelos.

3.2. Modelos em eletrofisiologia celular

Os diversos modelos existentes do comportamento elétrico das células cardíacas podem ser vistos como modificações e refinamentos do modelo desenvolvido por Hodgkin e Huxley (NELSON, 2005) através de uma série de experimentos eletrofisiológicos com

o axônio gigante de lula. Através desses experimentos foi demonstrado que as correntes iônicas macroscópicas poderiam ser compreendidas em termos de mudanças na condutância dos canais de Na+ e K+ na membrana. A condutância mede a facilidade para passagem de corrente elétrica em um material. Através de um conjunto de equações diferenciais, eles modelaram a base iônica do potencial de ação.

No entanto, o modelo Hodgkin-Huxley, desenvolvido a partir de células nervosas, não é um modelo satisfatório para células cardíacas. Outros experimentos similares realizados com as células cardíacas deram origem a modelos mais específicos, tais como o modelo de Beeler-Reuter (BEELER, REUTER, 1977) e os modelos de Luo-Rudy (LUO, RUDY, 1991; LUO, RUDY, 1994).

Os modelos da célula podem ser representados através de diagramas em esquema. A Figura 4, extraída do artigo original, mostra os canais iônicos, bombas e permutadores de íons do modelo Luo-Rudy II, um modelo do potencial de ação do ventrículo cardíaco (LUO, RUDY, 1994). Os diagramas possibilitam visualizar, não apenas a arquitetura dos componentes do modelo, mas também as principais variáveis consideradas. No entanto, para que o modelo possa ser simulado computacionalmente, duas questões importantes devem ser tratadas.

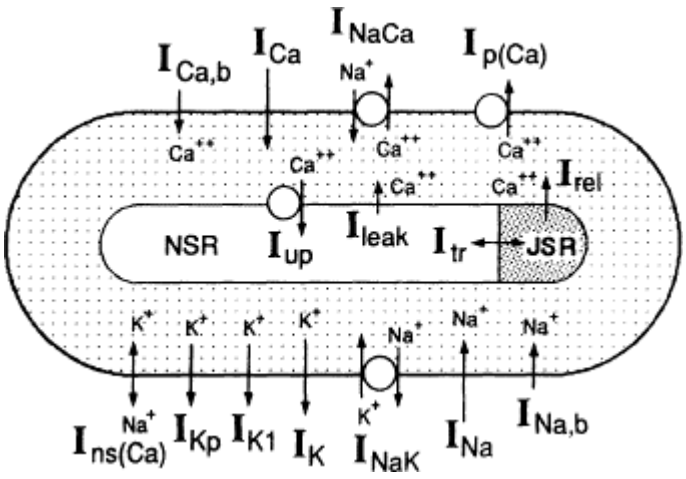

Figura 4. Diagrama em esquema do modelo LR-II, mostrando as correntes iônicas, as bombas e permutadores (LUO, RUDY, 1994).

A primeira questão se refere à própria representação do modelo. NICKERSON e HUNTER (2006) alegam que uma das principais dificuldades para um uso amplo de qualquer modelo de célula é a falta de habilidade dos cientistas em compartilhar facilmente seus modelos com colaboradores e outros pesquisadores no mundo. Segundo

esses autores, a principal fonte de modelos atualmente é a publicação (revisada por pares) do modelo e dos resultados que ele produz. A partir da publicação do conjunto de equações, outros cientistas tentam incorporar o modelo em um software de simulação. No entanto, essa não é uma tarefa fácil. Embora diagramas, descrição textual e equações possam sejam usadas na publicação dos modelos, elas não só estão sujeitas a erros tipográficos como também à falta de definição das condições iniciais ou de contorno necessárias à simulação. Como ilustração, a Figura 5 mostra uma parte das equações do modelo Luo-Rudy II, extraída do artigo original.

b. Currents through the L-type Ca^{2+} channel

$$I_{Ca,t}=I_{Ca}+I_{Ca,K}+I_{Ca,Na};$$

$$I_{Ca}=d\cdot f\cdot f_{Ca}\cdot \bar{I}_{Ca};\quad I_{Ca,K}=d\cdot f\cdot f_{Ca}\cdot \bar{I}_{Ca,K};\quad \text{and } I_{Ca,Na}=d\cdot f\cdot f_{Ca}\cdot \bar{I}_{Ca,Na}.$$

For ion S, including Ca^{2+}, Na^{+}, and K^{+},

$$\bar{I}_S=P_S\cdot z_s^2\cdot\frac{VF^2}{RT}\cdot\frac{\gamma_{si}\cdot[S]_i\cdot\exp(z_sVF/RT)-\gamma_{so}\cdot[S]_o}{\exp(z_sVF/RT)-1};$$

$$P_{Ca}=5.4\times10^{-4}\text{ cm/s};\quad \gamma_{Cai}=1;\quad \gamma_{Cao}=0.341;$$

$$P_{Na}=6.75\times10^{-7}\text{ cm/s};\quad \gamma_{Nai}=\gamma_{Nao}=0.75;$$

$$P_K=1.93\times10^{-7}\text{ cm/s};\quad \gamma_{Ki}=\gamma_{Ko}=0.75;$$

$$f_{Ca}=1/[1+([Ca^{2+}]_i/K_{m,Ca})^2];\quad K_{m,Ca}=0.6\ \mu\text{mol/L};$$

$$d_\infty=1/\{1+\exp[-(V+10)/6.24]\};$$

$$\tau_d=d_\infty\cdot\{1-\exp[-(V+10)/6.24]\}/[0.035\cdot(V+10)];$$

$$f_\infty=1/\{1+\exp[(V+35.06)/8.6]\}+0.6/\{1+\exp[(50-V)/20]\};$$

$$\tau_f=1/\left(0.0197\cdot\exp\{-[0.0337\cdot(V+10)]^2\}+0.02\right);$$

$$\alpha_d=d_\infty/\tau_d;\quad \beta_d=(1-d_\infty)/\tau_d;\quad \alpha_f=f_\infty/\tau_f;\quad \text{and } \beta_f=(1-f_\infty)/\tau_f.$$

Figura 5. Definição parcial das equações do modelo Luo-Rudy II, extraída do artigo original (LUO, RUDY, 1994).

A segunda questão está relacionada à implementação do modelo. A necessidade de implementação de métodos numéricos avançados atua como fator limitante para o efetivo uso e estudo do modelo. Ainda segundo NICKERSON e HUNTER (2006), mesmo quando o código-fonte de uma aplicação de simulação está disponível, ainda existe possibilidade de erros serem introduzidos quando o modelo é traduzido para uma nova linguagem de programação, uma outra técnica de solução numérica é utilizada ou o código fonte é incorporado em um pacote mais genérico de simulação.

O tratamento destas questões incentivou o surgimento de um novo paradigma para a especificação dos modelos, de uma forma independente do método de solução numérica ou de implementação, através do desenvolvimento da linguagem CellML (CELLML, 2008).

3.3. CellML

CellML é uma linguagem de marcação criada especificamente para representação de modelos biológicos. Baseada na linguagem XML, CellML especifica elementos que podem ser usados para representar um modelo de maneira formal, sem ambigüidades, legível por humanos e processável por máquinas. Metadados do modelo podem ser armazenados juntamente com o modelo. As equações matemáticas são representadas de maneira independente de uma implementação específica. CellML pode ser usada para representar, armazenar e compartilhar modelos, ampliando sua disponibilidade e facilitando o uso e validação dos mesmos pelos usuários. A descrição completa da linguagem CellML pode ser encontrada na especificação CellML 1.1 (CUELLAR et al., 2006).

3.3.1. Elementos da linguagem CellML

Os elementos da linguagem CellML são usados para descrever modelos, componentes, variáveis e unidades. Componentes podem ser agrupados para representar contenção ou encapsulamento. A partir da versão 1.1 unidades e componentes podem ser compartilhados usando um mecanismo de importação. A Figura 6 mostra um fragmento do modelo Hodgkin-Huxley descrito em CellML (CELLML REPOSITORY, 2008), com alguns dos principais elementos da linguagem (dois componentes, suas variáveis, a expressão matemática associada ao componente e a conexão entre os dois componentes).

```
<component name="membrane">
  <variable units="millivolt" public_interface="out" name="V" initial_value="-
87.0"/>
  <variable units="microF_per_cm2" name="Cm" initial_value="12.0"/>
  <variable units="millisecond" public_interface="in" name="time"/>
  <variable units="microA_per_cm2" public_interface="in" name="i_Na"/>
  <variable units="microA_per_cm2" public_interface="in" name="i_K"/>
  <variable units="microA_per_cm2" public_interface="in" name="i_Leak"/>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
   <apply><eq/><apply><diff/><bvar><ci>time</ci>
   </bvar><ci>V</ci></apply><apply><divide/><apply>
   <minus/><apply><plus/><ci>i_Na</ci><ci>i_K</ci>
   <ci>i_Leak</ci></apply></apply><ci>Cm</ci>
   </apply></apply>
  </math>
</component>
<component name="sodium_channel">
  <variable units="microA_per_cm2" public_interface="out" name="i_Na"/>
  <variable units="milliS_per_cm2" name="g_Na_max" initial_value="400.0"/>
  <variable units="milliS_per_cm2" name="g_Na"/>
  <variable units="millivolt" name="E_Na" initial_value="40.0"/>
  <variable units="millisecond" public_interface="in" private_interface="out"
name="time"/>
  <variable units="millivolt" public_interface="in" private_interface="out"
name="V"/>
  <variable units="dimensionless" private_interface="in" name="m"/>
  <variable units="dimensionless" private_interface="in" name="h"/>
  <math xmlns="http://www.w3.org/…">...</math>
 </component>
...
 <connection>
  <map_components component_2="sodium_channel" component_1="membrane"/>
  <map_variables variable_2="V" variable_1="V"/>
  <map_variables variable_2="i_Na" variable_1="i_Na"/>
 </connection>
```

Figura 6. Um fragmento do modelo Hodgkin-Huxley descrito em CellML (CELLML REPOSITORY, 2008).

A Figura 7 apresenta um esquema dos elementos que compõem um modelo CellML (SCHILSTRA,2005). A seguir são descritos, de forma resumida, os principais elementos da linguagem, com base na especificação disponível em (CUELLAR et al., 2006).

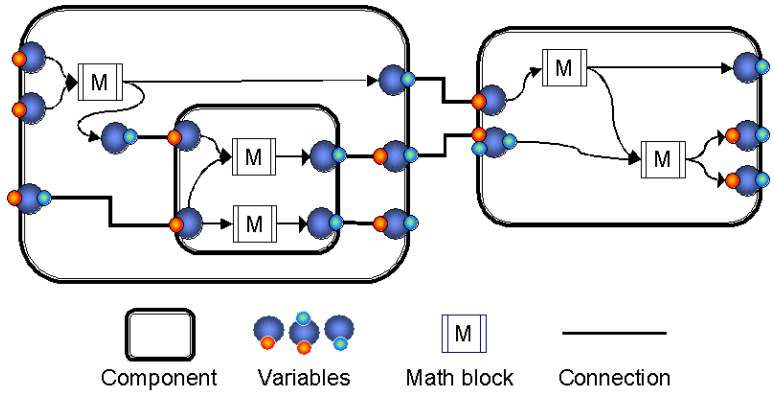

Figura 7. Representação em esquema dos elementos de um modelo CellML (SCHILSTRA,2005).

Modelos e Componentes

Modelos CellML são representados como uma rede de componentes interconectados. Um modelo recebe um nome e um identificador único (cmeta:id), usado para referenciar o modelo através de uma URI.

Um componente é a menor unidade funcional de um modelo. Cada componente deve conter pelo menos uma variável, e pode conter definição de unidades e equações matemáticas que descrevem como o componente se comporta dentro do modelo.

Variáveis

Uma variável CellML é uma entidade nomeada que pertence a um único componente. O objetivo principal de uma variável é representar quantidades usadas nas equações matemáticas. Uma variável pode ter um valor inicial e pode ter atributos (como unidade, interface pública e interface privada).

Matemática

As equações matemáticas são expressas em Content MathML 2.0 (MATHML, 2001), uma linguagem de marcação baseada em XML. Em um modelo CellML são definidas dentro dos componentes, sob o espaço de nomes MathML. MathML provê um extenso conjunto de operadores e containers, possibilitando a combinação destes em expressões matemáticas. Apesar de um documento CellML poder conter qualquer elemento da linguagem MathML, os autores da linguagem definiram um subconjunto de elementos para seu uso em um modelo CellML. Este subconjunto visa facilitar a

interoperabilidade entre diferentes aplicações de software e é considerada suficiente para descrever a matemática da maioria dos modelos biológicos.

Conexões

É possível realizar o mapeamento de uma variável definida em um componente com o valor de outra variável, definida em outro componente. Entretanto, o valor da variável não pode ser matematicamente alterado dentro do componente, se este valor foi importado de uma variável que pertence a outro componente.

Conexões permitem a troca de informações entre componentes. Uma conexão consiste de um conjunto de mapeamentos entre as variáveis de dois componentes. Somente uma conexão pode ser definida entre dois componentes quaisquer em um modelo.

Unidades

A especificação de CellML exige que todas as variáveis e números usados no modelo sejam declarados com um conjunto de unidades. Esta característica assegura o reuso dos componentes e modelos CellML, uma vez que componentes e modelos contendo variáveis com unidades diferentes mas com as mesmas dimensões podem ser conectadas. CellML provê um dicionário de unidades padrão, baseado no Sistema Internacional de Unidades (SI) (SI, 2006), com algumas unidades derivadas do SI e outras unidades não definidas no SI que são usadas com freqüência em modelos biológicos. É possível criar unidades definidas pelo usuário a partir das unidades já presentes no dicionário. Isto permite que os autores de modelos possam trabalhar com as unidades de sua escolha, enquanto assegura que os modelos poderão ser integrados com modelos que usam outras unidades.

Grupos

Grupos são usados para fornecer uma estrutura ao modelo. Eles definem relacionamentos nomeados entre os componentes. Dois tipos de agrupamento são definidos na especificação de CellML: encapsulamento e contenção.

Encapsulamento (*encapsulation*) é um tipo lógico de agrupamento, usado como uma conveniência de modelagem. O encapsulamento permite ao autor do modelo

ocultar um grupo de componentes do resto do modelo, usando um único componente como uma interface através da qual as variáveis podem ser permutadas entre o conjunto encapsulado e os demais componentes do modelo.

Contenção (*containment*) é usada para descrever a organização física ou geométrica do modelo, como a estrutura biológica. Este tipo de agrupamento especifica que os componentes estão fisicamente contidos dentro do componente pai. Por exemplo, um componente representando um canal na membrana celular pode estar fisicamente contido no componente representando a membrana, mas não necessariamente encapsulado neste componente, principalmente se houver variáveis exportadas do componente canal que são usadas em outros componentes do modelo.

Importação

A partir da versão 1.1, a linguagem CellML permite a importação de componentes, conexões e unidades de outros modelos, a fim de serem reusados no modelo corrente. Esta característica possibilita que modelos anteriores possam ser usados como um framework no qual novos componentes são adicionados para a construção de novos modelos.

Metadados

Metadados são embutidos em modelos CellML usando a linguagem RDF. Metadados contém informações que provêem um contexto para o modelo, descrevendo a literatura de referência na qual o modelo está baseado, identificando o criador do modelo e suprindo informações biológicas sobre o modelo, tais como o tipo específico de célula para o qual o modelo foi desenvolvido.

3.3.2. Considerações sobre a linguagem CellML

A linguagem CellML pode ser usada para representar, armazenar e compartilhar modelos, ampliando sua disponibilidade e facilitando o uso e validação dos mesmos pelos usuários. No entanto, CellML não é totalmente adequada para o processo de criação de novos modelos, uma vez que não provê mecanismos de anotação que facilitem o reuso e modificação de novos componentes.

Geralmente o desenvolvimento de um novo modelo em CellML usa modelos já existentes como ponto de partida. Assim, primeiramente um novo componente é inserido em um modelo tomado como base. O modelo completo é ajustado através do estabelecimento de conexões com o novo componente e da definição de novas variáveis. O modelo é simulado várias vezes e, dependendo dos resultados obtidos, o novo componente é alterado ou substituído. Este processo pode se repetir várias vezes. Uma vez que em CellML as variáveis são locais aos componentes, a cada troca ou definição de componentes as variáveis devem ser redefinidas e as conexões refeitas. Isto torna o processo de modelagem mais trabalhoso e sujeito a erros.

Outra questão importante está associada à validação do modelo descrito em CellML. Por ser baseada em XML, a validação do modelo é eminentemente sintática. Através do uso de DTD ou de Schemas o modelo pode ser validado quanto a erros de sintaxe e em relação à aderência à especificação CellML. Questões semânticas ou não podem ser efetivamente tratadas (e.g. evitar que um componente “canal iônico” contenha um componente “membrana”) ou são deixadas para a fase de implementação (e.g. a conformidade no uso de diferentes unidades de medida para as variáveis, como “segundos” e “milisegundos”).

Para tratar estas questões esta dissertação propõe o uso de um modelo baseado em ontologia para criação de modelos em eletrofisiologia.

3.4. Comentários Finais

Este capítulo mostrou uma discussão preliminar sucinta dos contexto biológico abordado no trabalho. Conceitos básicos sobre a eletrofisiologia da célula cardíaca, modelos biológicos e a linguagem CellML foram apresentados. De acordo com o que foi argumentado, mostrou-se a necessidade da modelagem conceitual aplicada a modelos biológicos, a fim de superar algumas dificuldades atualmente existentes neste campo.

O Capítulo 5 apresentará então uma proposta de framework para um AMS (Ambiente de Modelagem e Simulação) que busca atender às necessidades levantadas anteriormente. O framework será acessível via web e está baseado em uma ontologia de modelos biológicos, descrita no Capítulo 4.

4. CelO – Ontologia para Modelagem em Eletrofisiologia

Os esforços para padronizar a representação de modelos em sistemas biológicos geraram linguagens como CellML (*Cell Markup Language*) e SBML (*Systems Biology Markup Language*) (SBML, 2008). Ambas provêem uma representação formal dos principais componentes em um sistema, incluindo entidades biológicas, definição de parâmetros e as equações representativas dos processos biológicos. Ontologias na área biomédica, por outro lado, têm buscado atender a necessidade de padronização de conceitos e termos, tais como a OBO (*Open Biomedical Ontologies*) (OBO, 2007), GO (*Gene Ontology*)(GO, 2000), CCO (*Cell Cycle Ontology*) (ANTEZANA, 2006) e CCO (*Cell Component Ontology*) (ZHANG et al., 2005). Essas ontologias, no entanto, não objetivam especificamente a tarefa de modelagem.

Este capítulo propõe o uso de uma ontologia, chamada CelO (*Cell Component Ontology*), aplicada à representação de modelos biológicos, particularmente modelos em eletrofisiologia celular. O uso de uma linguagem lógica expressiva para representar modelos permite enriquecê-los através da aplicação de mecanismos de inferência e da definição de regras semânticas. Objetivos, características e limitações da ontologia são apresentados, bem como uma visão geral e uma descrição detalhada dos seus elementos.

4.1. Modelagem Conceitual com OWL

A linguagem escolhida para representação dos modelos foi OWL-DL. Como apresentado na seção 2.2.2, esta sub-linguagem de OWL está baseado em Lógica Descritiva, provendo expressividade suficiente para representar os componentes e propriedades dos modelos, ao mesmo tempo permitindo o uso de mecanismos de inferência e a definição de regras semânticas.

O uso de um formato padronizado e recomendado pelo W3C possibilita que os modelos biológicos criados sejam portáteis e integráveis, independentes de uma aplicação, formato ou linguagem de programação específica. O número crescente de ferramentas para trabalhar com OWL-DL, como Protégé-OWL (PROTEGE, 2008), Jena (JENA, 2008), Swoop (KALYANPUR, PARSIA, HENDLER, 2005), a existência de APIs (*Application Program Interface*) como Protege-OWL API (PROTEGE, 2008) e OWLAPI (OWLAPI, 2008) e a implementação de mecanismos de inferência como

FACT++ (FACT, 2008), Racer (RACER, 2008) e Pellet (SIRIN et al., 2007) facilitam seu uso e permitem a criação de aplicações customizadas.

BORGIDA e BRACHMAN (2003) discutem alguns aspectos da modelagem conceitual usando Lógica Descritiva, úteis para a modelagem com OWL-DL. Segundo esses autores, uma ontologia de modelos conceituais inclui objetos individuais, que estão associados através de relacionamentos (geralmente binários) e agrupados em classes. Descrições complexas podem ser construídas a partir de conceitos (*concepts*) e papéis (*roles*) atômicos. As classes modelam os conceitos, enquanto os relacionamentos modelam os papéis.

Entre as questões discutidas por BORGIDA e BRACHMAN (2003) relevantes para este trabalho, pode-se ressaltar:

a. Certas noções podem ser modeladas como indivíduos ou como conceitos. Uma heurística geral é que se é esperado que a noção possa ser contável, ela deve ser modelada como indivíduo. Se a noção não possui um momento de criação ela é usualmente modelada como conceito.
b. É importante distinguir entre objetos e valores associados aos indivíduos. Os objetos possuem uma identidade intrínseca e imutável e precisam ser criados na base de conhecimento. Os valores (inteiros, strings, etc.) são abstrações matemáticas cuja identidade é determinada por algum procedimento envolvendo a estrutura do indivíduo.
c. Indivíduos são diferentes das referências feitas a eles. Por exemplo, o indivíduo “José” é diferente da referência “a pessoa com número de cartão 12345”.
d. Uma das propriedades fundamentais da Lógica Descritiva é a distinção entre conceitos **atômicos** (ou primitivos), para os quais as instâncias somente podem ser declaradas explicitamente, e os conceitos **definidos**, que oferecem condições necessárias e suficientes para a inclusão das instâncias. A classificação de indivíduos sob conceitos definidos é geralmente feita através dos mecanismos de inferência (seção 2.4.1).
e. Em Lógica Descritiva não existe um conceito semelhante aos atributos estáticos da Orientação a Objetos. Caso uma informação deva ser associada

ao conceito inteiro (e não a cada indivíduo), deve ser criado um "meta-indivíduo" relacionado ao conceito através de alguma convenção de nome.

f. Relacionamentos binários são modelos através de papéis e atributos. Esses relacionamentos podem ter restrições de cardinalidade e de domínio. Relacionamentos inversos entre os papéis também devem ser modelados.
g. Pode ser necessário modelar "propriedades de propriedades". No Modelo de Entidades e Relacionamentos (MER) (HEUSER, 1998) isso pode ser feito através da criação de uma entidade associativa. Em Lógica Descritiva pode-se considerar uma "concretização" (*reification*) do relacionamento, através da criação de uma nova classe que o represente.
h. Sistemas de Lógicas Descritivas, diferente de bancos de dados, não assumem um mundo-fechado (*closed-world assumption*) mas sim um mundo-aberto (*open-world assumption*). Assim, em contraste com bancos de dados, se algum relacionamento não é encontrado, não é assumido que ele seja falso. Como conseqüência, qualquer questão sobre se um indivíduo está associado a um conceito, ou se está relacionado a outro indivíduo, possui três possíveis respostas: definitivamente sim, definitivamente não ou desconhecido. O aspecto positivo é que a modelagem pode ser feita a partir de informações parciais. No entanto, existirão situações em que um indivíduo não será reconhecido como satisfazendo as definições como seria esperado.

Outros aspectos relevantes da modelagem com OWL-DL que foram considerados neste trabalho são: a representação de classes como valores de propriedades (NOY, 2005), a modelagem de relacionamentos do tipo todo-parte (RECTOR, WELTY, 2005), a definição de relacionamentos n-ários (NOY, RECTOR, 2006) e a representação de valores específicos em OWL, através de partição de valores ou conjunto de valores (RECTOR, 2005).

4.2. Objetivos

Os objetivos específicos da construção e aplicações da ontologia CelO são:

a. Representar um modelo biológico com uma linguagem lógica, com capacidade de inferência e utilização de regras.

b. Representar os modelos existentes descritos em CellML com anotações semânticas, que permitam superar as limitações descritas na seção 2.3.3. Isto implica em representar explicitamente o conhecimento que está implícito nos modelos CellML, associando semântica aos componentes e variáveis. Este conhecimento permite a criação de mecanismos de validação mais rígidos e a automação parcial do processo de inserção e remoção de componentes no modelo.
c. Permitir a criação de modelos em que as variáveis sejam definidas a partir de seu significado no modelo e não apenas sintaticamente.
d. Permitir a criação de um repositório de modelos que possa ser pesquisado semanticamente, seja diretamente através da web, seja através de um banco de dados com capacidade para inferência. Segundo LLOYD, HALSTEAD e NIELSEN (2004) um dos objetivos principais do desenvolvimento de bancos de dados de ontologias é prover interfaces entre o humano, que compreende os conceitos, e a máquina, que provê representações acuradas, consistentes e sem ambigüidades dos modelos biológicos.
e. Promover o reuso de componentes existentes, através da possibilidade de composição automática ou semi-automática de modelos complexos a partir de modelos existentes mais simples. Modelos simples podem representar uma entidade biológica unitária (como um componente representando um único canal iônico), auto-contida, que pode ser reutilizada (e customizada) várias vezes em um modelo de célula inteira.

4.3. Características da Ontologia

4.3.1. Integração com CellML

Considerando a aplicabilidade de CellML para a descrição e simulação computacional de modelos, a existência e disponibilidade de centenas de modelos e várias ferramentas que trabalham com a linguagem, o projeto da ontologia CelO buscou refletir parcialmente a estrutura de CellML para facilitar a utilização de modelos existentes.

A integração com CellML pode ser vista sob a perspectiva de uma arquitetura em três níveis de abstração, conforme a Figura 8.

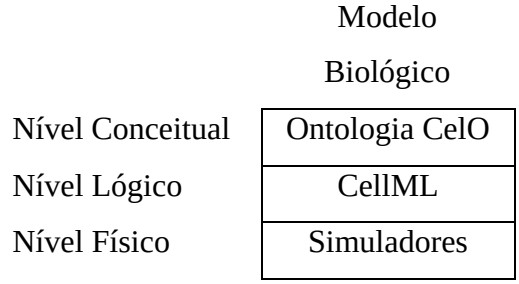

Figura 8. Níveis de abstração no processo de modelagem.

No caso dos modelos em eletrofisiologia, a ontologia CelO é utilizada para descrever os componentes em um nível mais alto de abstração, considerando sua semântica. Uma variável de um componente pode ser descrita como sendo um "canal iônico de sódio", com a respectiva parametrização. Este modelo estará associado a um modelo CellML onde a variável será representada apenas sintaticamente, como ocorre com as linguagens de programação tradicionais. Um simulador que trabalhe com modelos CellML, como por exemplo as aplicações AGOS (BARBOSA, SANTOS, AMORIM, 2006), COR (GARNY, KOHL, NOBLE, 2003) ou PCEnv (PCENV, 2008), vai traduzir o modelo para um implementação específica, a fim de que ele possa ser executado.

A integração da ontologia CelO com CellML permite, portanto, que sejam construídas ferramentas que integrem simuladores já existentes, de forma que o modelador trabalhe com um nível alto de abstração e possa executar o modelo de forma transparente.

4.3.2. Extensibilidade

O processo de desenvolver e evoluir uma ontologia na área da fisiologia é um ciclo iterativo, que necessariamente agrupa especialistas de diferentes campos da Biologia, Matemática e Ciência da Computação (LLOYD, HALSTEAD, NIELSEN, 2004) em um trabalho cooperativo. Assim, se o autor de um modelo deseja representar algo que não está presente na ontologia atual, ou testar novos componentes, ele pode simplesmente estende-la, através da criação de novas classes, indivíduos, propriedades ou regras. A adição de novos elementos cria, de certa forma, uma nova ontologia, sem alterar a estrutura da ontologia original.

Naturalmente esta possibilidade deve ser utilizada com cuidado, a fim de que não ocorra uma proliferação de ontologias que não possam ser integradas umas com as

outras. Além disso, considerando a integração com CellML para execução das simulações, tais extensões devem estar restritas ao nível conceitual, de maneira a não perder a consistência com o modelo CellML representado.

4.3. Visão geral da Ontologia

A estrutura proposta para a ontologia CelO é motivada pela necessidade de prover três tipos essenciais de conhecimento sobre um modelo, cada um caracterizado pelas seguintes questões:

- Quais grandezas são medidas e quais unidades são utilizadas pelas variáveis do modelo?
- Quais conceitos do domínio da Biologia estão associados ao modelo?
- Quais são os componentes do modelo e como eles podem ser acessados (quais são suas interfaces)?

Estas questões levaram à definição de três classes gerais no nível mais alto da ontologia, apresentadas na Figura 9. Uma referência detalhada da ontologia é apresentada no apêndice A1.

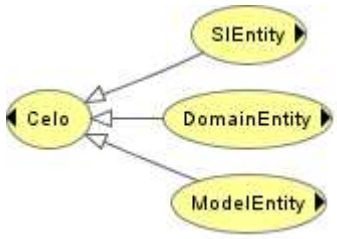

Figura 9. Nível topo da ontologia CelO.

4.3.1. Classe SIEntity

Esta classe e suas sub-classes, que têm uma aplicação genérica, definem um dicionário de unidades e grandezas que são utilizadas em conjunto com os conceitos do domínio e as variáveis dos modelos. Este dicionário está baseado no Sistema Internacional de Unidades (SI). Além disso, novas unidades podem ser definidas pelos usuários, para permitir a integração com CellML As grandezas (quantidades) medidas também estão baseadas no SI. A Figura 10 apresenta a estrutura da classe **SIEntity**.

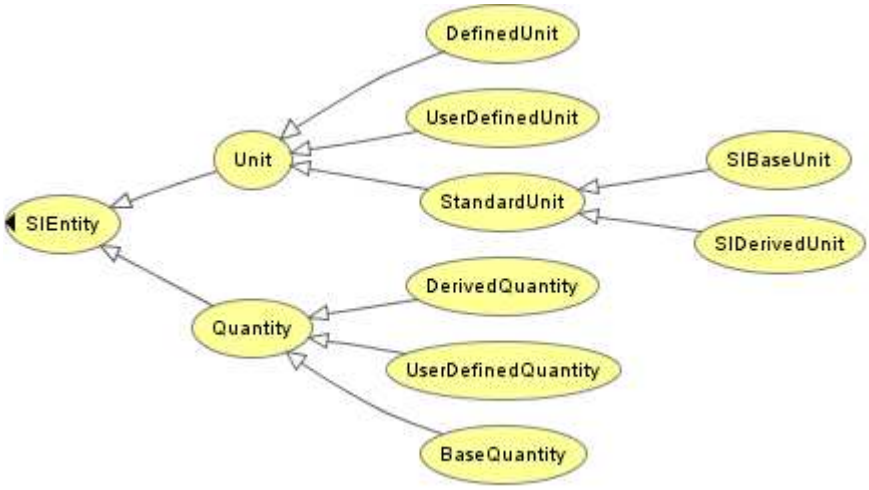

Figura 10. Classe SIEntity da ontologia CelO.

4.3.2. Classe DomainEntity

Esta classe e suas sub-classes definem os termos que serão usados como um vocabulário compartilhado pelos pesquisadores na criação de modelos. Esses termos possuem uma semântica associada, permitindo a descrição do modelo em nível mais alto. A Figura 11 apresenta a estrutura da classe **DomainEntity**.

BiologicalEntity se refere aos conceitos da área da Biologia. Como nessa implementação inicial o escopo está restrito a eletrofisiologia celular, apenas conceitos relativos à célula foram anotadas. **CellElement** se refere à estrutura física da célula (**CellStructure**) e aos processos associados à eletrofisiologia (**CellProcess**). Os elementos de interesse no modelo da célula são registrados em **CellPart** e a localização destes elementos, em termos espaciais, em **CellSpace**. **GatingProcess** está associado aos diversos tipos de *gating*; **PolarizationProcess**, aos fenômenos de despolarização, polarização e repolarização; **MembraneTransport** se refere aos fenômenos de transporte através da membrana. **CellType** registra os diversos tipos de células.

ChemicalEntity se refere aos conceitos da área da Química. **ChemicalElement** se refere aos elementos químicos (tais como sódio, potássio, etc). **ChemicalCompound** se refere aos compostos químicos (**Nucleotide** é apresentado como um exemplo). **ChemicalObject** é um conceito genérico para definição de átomos, íons e moléculas.

BioChemicalEntity se refere a conceitos da área da Bioquímica, que estuda a química dos processos biológicos que ocorrem nos seres vivos. **Protein** é colocado como um exemplo de conceito no nível bioquímico.

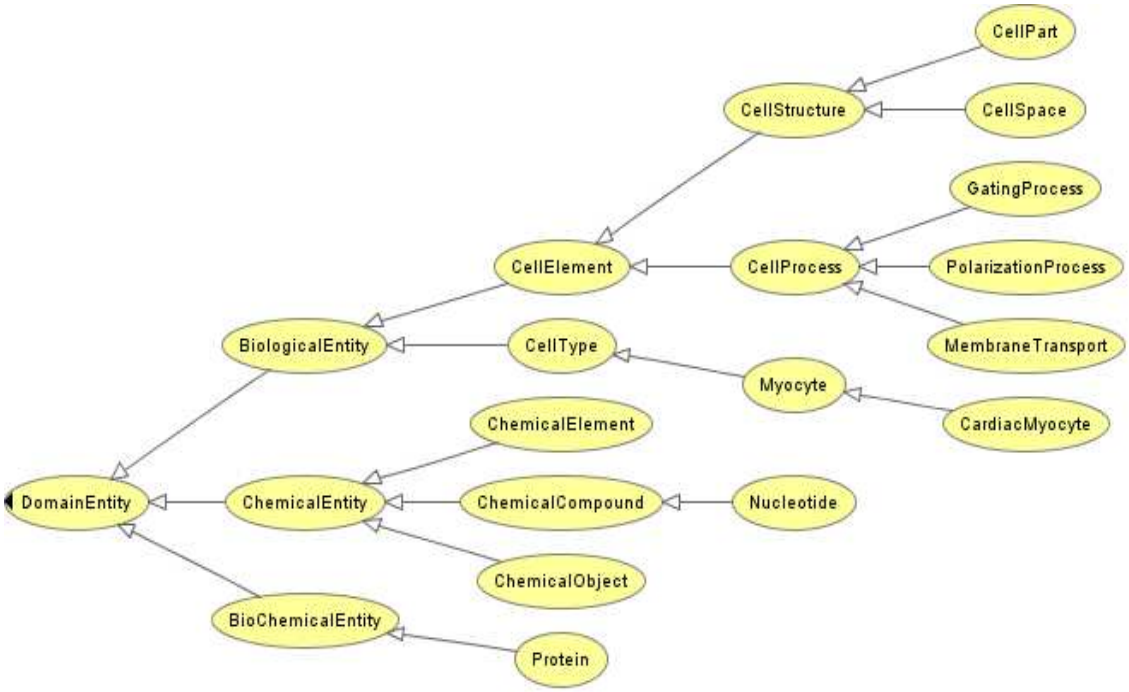

Figura 11. Classe DomainEntity da ontologia CelO.

4.3.3. Classe ModelEntity

Esta classe e suas sub-classes definem os conceitos que serão usados no modelo biológico representado. O objetivo é ter uma descrição de alto nível, fazendo referência ao modelo CellML para as questões de simulação. A Figura 12 apresenta a estrutura da classe **ModelEntity** (as classes em destaque laranja são classes ditas "definidas", pois sua definição está baseada em restrições feitas em outras classes, chamadas "primitivas").

Como a premissa na arquitetura do framework CelOWS, descrito no capítulo 4, é poder tratar um modelo como um serviço web, que possui interfaces e pode ser "executado" (através da simulação), a classe **ModelService** provê uma forma de organizar a visão de um modelo como um serviço. Sua estrutura é inspirada na ontologia OWL-S (MARTIN et al., 2004), usada para descrição semântica de serviços web. Uma instância de **ModelService** existe para cada modelo. Cada instância vai estar associada com uma instância de **ModelProfile**, **ModelGrounding** e **ModelProcess**.

ModelProfile tem o objetivo de informar sobre o que o modelo trata: quais são os componentes do modelo, se está associado a algum compartimento específico da

célula e quais são as entidades biológicas (instância de **BiologicalEntity**) associadas ao modelo.

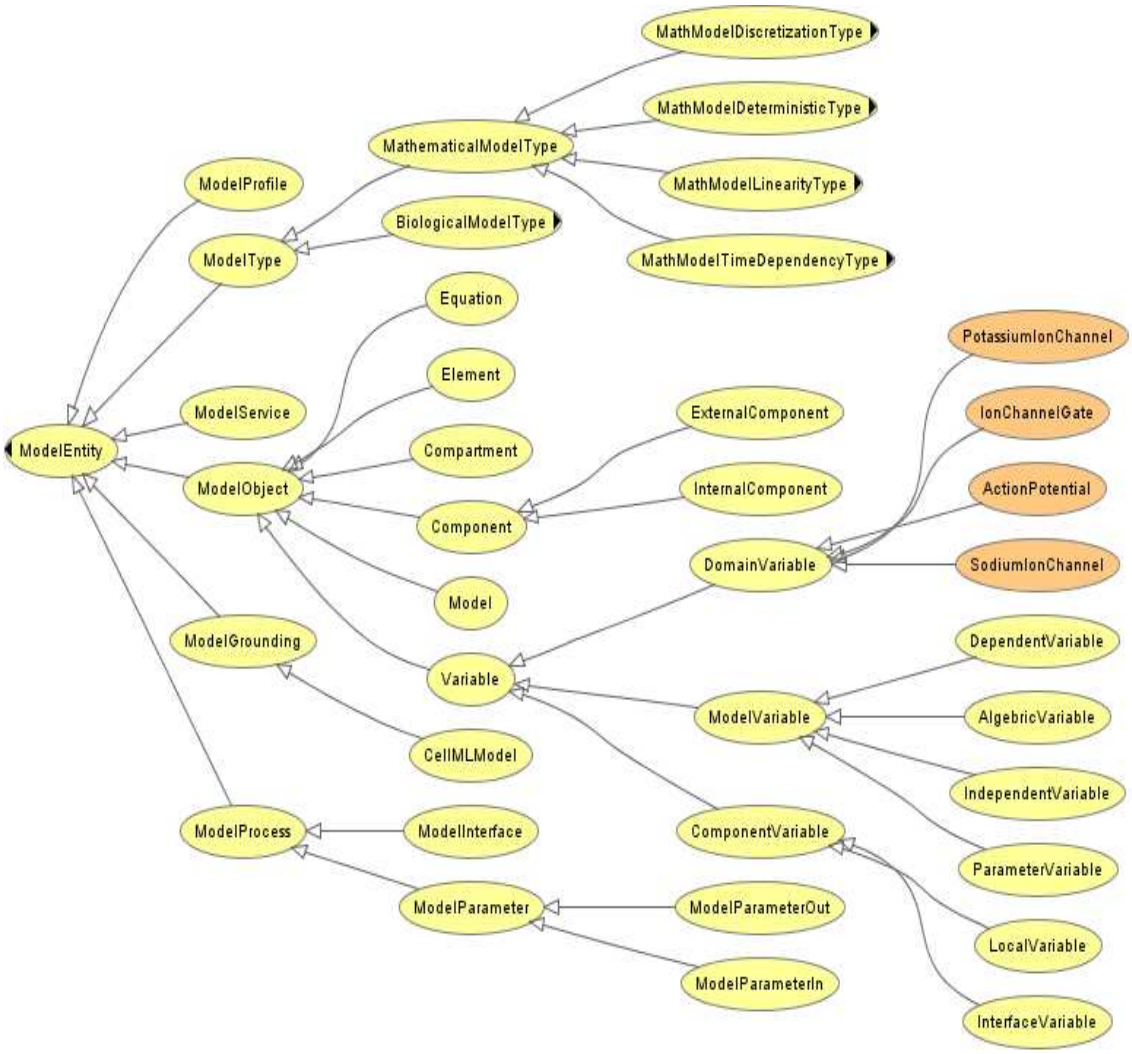

Figura 12. Classe ModelEntity da ontologia CelO

ModelGrounding especifica qual o modelo "lógico" associado, no caso de uma simulação. Como exemplo, a classe **CellMLModel** é usada para armazenar a URI do modelo CellML associado.

ModelProcess basicamente indica como o modelo pode ser usado, ou seja, quais são os parâmetros de entrada (**ModelParameterIn**) e de saída (**ModelParameterOut**) que estão associados à interface (**ModelInterface**) do modelo. Esses parâmetros estão diretamente associados às variáveis do modelo e podem ser usados no processo de simulação do modelo ou na sua composição com outros modelos.

ModelType é uma classe genérica, usada apenas para caracterizar o problema biológico tratado pelo modelo (um modelo de Eletrofisiologia, por exemplo, **Electrophysiology**) e o tipo de modelagem matemática utilizada.

ModelObject agrupa os objetos que compõe um modelo, estando diretamente relacionado com o tipo de modelo subjacente. **Equation** relaciona as equações que implementam matematicamente o modelo, como strings expressas em MathML. **Variable** representa as variáveis do modelo. **ModelVariable** descreve como as variáveis participam nas equações do modelo e **ComponentVariable** relaciona as variáveis de cada componente que compõe o modelo e que podem ser variáveis usadas na interface do componente (**InterfaceVariable**) ou variáveis locais ao componente (**LocalComponent**). **DomainVariable** associa as variáveis do modelo com conceitos expressos pelas classes de **DomainEntity**.

Model representa o modelo propriamente dito, que pode ser atômico (contendo apenas um único componente) ou composto (contendo dois ou mais componentes). A composição de modelos permite a definição de modelos simples que podem ser usados na construção de modelos mais complexos, promovendo o reuso. **Component** representa os componentes do modelo. Esses componentes podem estar descritos no próprio modelo (**InternalComponent**) ou podem armazenar referências URI a modelos externos (**ExternalComponent**). Um modelo pode ser composto por modelos internos e externos simultaneamente.

As **regras** descritas em SWRL são usadas para a definição de relacionamentos no processo de integração com os modelos descritos em CellML.

4.3.4. Regras

As regras semânticas são usadas na ontologia CelO com o propósitos básico de inferir conhecimento que está implícito em modelos CellML já existentes, convertidos para a ontologia; isto é feito através do preenchimento das propriedades associadas às variáveis dos componentes, com indivíduos definidos na ontologia, a partir dos detalhes obtidos no modelo existente (por exemplo, a partir do nome das variáveis), considerando também o conhecimento já definido (por exemplo, variáveis sem dimensão não podem estar associadas a elementos químicos).

Eventualmente as regras poderiam ser usadas também para validar semanticamente o modelo em relação à sua consistência e completude; um exemplo de validação é a verificação se a composição dos componentes não fere a hierarquia anatômica.

As regras são expressas em SWRL e a Figura 13 apresenta algumas das regras definidas para a ontologia CelO, aplicadas a um modelo.

Define a grandeza medida por uma variável:

```
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:DefinedUnit(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^
celo:measures(?x3, ?x5) → celo:hasMeasure(?x1, ?x5)

celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:UserDefinedUnit(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^
celo:measures(?x3, ?x5) → celo:hasMeasure(?x1, ?x5)

celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:StandardUnit(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^
celo:measures(?x3, ?x5) → celo:hasMeasure(?x1, ?x5)
```

Associa uma variável a um elemento químico (exceto as variáveis sem dimensão):

```
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:ChemicalEntity(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^
celo:hasVariableUnit(?x1, ?x5) ^ celo:hasName(?x5, ?x6) ^
swrlb:notEqual(?x6, "dimensionless") → celo:hasDomainEntity(?x1, ?x3)
```

Associa uma variável a uma instância da classe BioChemicalEntity:

```
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:BioChemicalEntity(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) →
celo:hasDomainEntity(?x1, ?x3)
```

Associa uma variável a uma instância da classe BiologicalEntity:

```
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:BiologicalEntity(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) →
celo:hasDomainEntity(?x1, ?x3)
```

Associa um componente a uma instância da classe DomainEntity:

```
celo:Component(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:DomainEntity(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) →
celo:hasDomainEntity(?x1, ?x3)
```

Associa uma variável com uma unidade de medida:

```
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:Unit(?x3) ^
celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) →
celo:hasVariableUnit(?x1, ?x3)
```

Figura 13. Exemplos de regras SWRL aplicadas a um modelo.

4.4. Limitações

A versão inicial da ontologia CelO proposta neste trabalho apresenta as seguintes limitações:

a. Não foi considerado nenhum tipo de integração ou alinhamento com outras ontologias de domínio. Este tema foi discutido por LLOYD, HALSTEAD E NIELSEN (2004) e também no Workshop CellML (2007). Outras ontologias são usadas basicamente para anotação de metadados. No entanto, a existência de elementos semanticamente mais ricos na ontologia CelO indica a possibilidade de integração com ontologias do domínio, tais como *Open Biomedical Ontologies* (OBO) , *Gene Ontology* (GO) e *Cell Component Ontology* (CCO) para descrição dos fenômenos biológicos ou da estrutura da célula de forma padronizada.
b. A ontologia foi testada apenas com modelos em eletrofisiologia cardíaca, tendo sido criados elementos na ontologia exclusivamente referentes a esta área. No entanto, pelas características de extensibilidade (seção 3.2.4), ela pode ser estendida para tipos de modelos diferentes, da mesma forma que CellML pode ser usada para vários tipos de modelos (CELLML, 2008).
c. Como ocorre com qualquer ontologia expressa em OWL, não é praticável a edição da ontologia diretamente no arquivo texto. OWL possui diversas representações diferentes e os editores OWL estão preparados para trabalhar com essas diversas representações. A complexidade da sintaxe torna inviável lidar diretamente com os marcadores OWL. Uma vez que a ontologia CelO é expressa em OWL (bem como os modelos gerados a partir dela), pressupõe-se o uso de ferramentas específicas para se trabalhar com os modelos. O uso de editores OWL, tais como o Protege-OWL (KNUBLAUCH, FERGERSON, NOY, 2004) e SWOOP (KALYANPUR, PARSIA, HENDLER, 2005), para acessar os modelos é possível. No entanto, esses editores possibilitam o acesso a diversas classes, indivíduos e propriedades que são elementos na ontologia e que não deveriam ser visíveis ao usuário que está construindo o modelo.
d. Da mesma forma que CellML possui uma especificação própria para o uso de metadados (CUELLAR, NELSON, HEDLEY, 2006), pressupõe-se que ontologias apropriadas serão usadas para registrar os metadados nos modelos gerados a partir da ontologia CelO. Ainda que OWL permita o uso de

anotações (anotations), a fim de manter a integração com CellML (seção 4.3.1) recomenda-se o uso das mesmas ontologias de metadados.

4.5. Comentários Finais

Este capítulo apresentou CelO (*Cell Component Ontology*), uma ontologia usada para modelagem conceitual em Biologia Sistêmica. A ontologia possui descrições das classes relevantes ao domínio da Biologia, classes referentes ao Sistema Internacional de Unidades e classes para representação de modelos.

Como será mostrado no próximo capítulo, a ontologia é a base para a arquitetura do framework CelOWS, permitindo que o sistema possa auxiliar os pesquisadores a definir, redefinir, validar, compor e executar modelos de forma automatizada e semi-automatizada.

5. CelOWS: um framework para definição, pesquisa e composição de modelos biológicos

A ontologia CelO (*Cell Component Ontology*) faz parte de um contexto mais amplo que é a especificação e desenvolvimento de uma infraestrutura para aplicações em *e-science*, cujo objetivo é o suporte computacional a projetos de pesquisa e a pesquisadores que desejam compartilhar experiências e resultados. Esta infraestrutura visa apoiar a busca por artefatos de pesquisa relacionados a um dado domínio de aplicação. A ontologia CelO é usada, então, como base para pesquisa e composição de modelos biológicos no framework denominado CelOWS. O framework proposto nessa dissertação usa os conceitos de repositório de ontologias e serviços web semânticos, aproveitando a experiência adquirida com o broker MathWS (MATOS et al, 2007), um trabalho desenvolvido no contexto de busca e composição de serviços web matemáticos.

Este capítulo descreve os objetivos do framework CelOWS e a arquitetura proposta para sua efetiva implementação. Quatro cenários de uso, comuns em um MAS (Ambiente de Modelagem e Simulação), são apresentados, juntamente com a descrição da implementação computacional do framework e das ferramentas utilizadas. São descritos três procedimentos típicos dos cenários de uso, além dos principais trabalhos relatados na literatura, relacionados ao tema.

5.1. Objetivos

O objetivo do framework CelOWS é prover uma infraestrutura para registro, pesquisa, recuperação, composição e execução (simulação) de componentes usados em modelos biológicos utilizando ontologias. Especificamente o framework deve permitir:

a. Registro e armazenamento em bancos de dados (repositórios) distribuídos de modelos semânticos que representem os componentes de modelos biológicos.
b. Pesquisas no banco de dados com base na semântica expressa nos modelos, com recuperação dos modelos como ontologias.
c. Composição de modelos, gerando modelos mais complexos que podem ser armazenados no banco de dados.

d. Execução de modelos, ou seja, a submissão dos mesmos a uma ferramenta que permita a simulação do modelo e o retorno dos resultados gerados.

A descrição semântica de componentes, em um formato que permita sua composição com outros componentes, ao mesmo tempo se referindo a modelos que possam ser simulados em ferramentas já estabelecidas (ou criando tais modelos, se necessário), traz uma grande flexibilidade para os processos de modelagem, comuns no ambiente de *e-science*.

O framework proposto é implementado como um serviço web, em inglês *WS* (*Web Service*). Isso permite a distribuição dos repositórios de modelos e facilita sua implementação em ambientes de workflow científico e grids computacionais. Cada componente é também encapsulado em um serviço web, podendo então ser executado remotamente (de forma independente ou composto com outros componentes) ou simplesmente fornecendo sua localização, através da qual o código pode ser obtido para execução local.

5.2. Arquitetura

Uma visão geral da arquitetura do framework CelOWS é apresentada na Figura 14. Na figura *URI CelO* representa a URI da ontologia do modelo CelO. O framework implementa conceitos de Arquitetura Orientada a Serviços, em inglês SOA (*Service Oriented Architeture*), e considera três camadas (*tiers*) distintas. A representação da arquitetura busca seguir as orientações do SEI (*Software Engineering Institute*), utilizando a notação UML 2.0, conforme apresentada por MERSON (2005):

- CelOWS: representa o próprio framework. Implementado como um serviço web, o framework pode ser instalado em sites distintos, permitindo a distribuição dos repositórios, além de ser independente da interface que vai acessar os seus serviços, facilitando a integração com ferramentas já existentes.
- Backend: corresponde a uma camada de serviços usados pelo framework, como o acesso a banco de dados (CelOWS-DB) e a execução de uma ferramenta utilizada para simulação do modelo CellML.

- Client: implementa uma interface para interação com o usuário do framework, podendo ser desenvolvida em qualquer linguagem com acesso a serviços web.

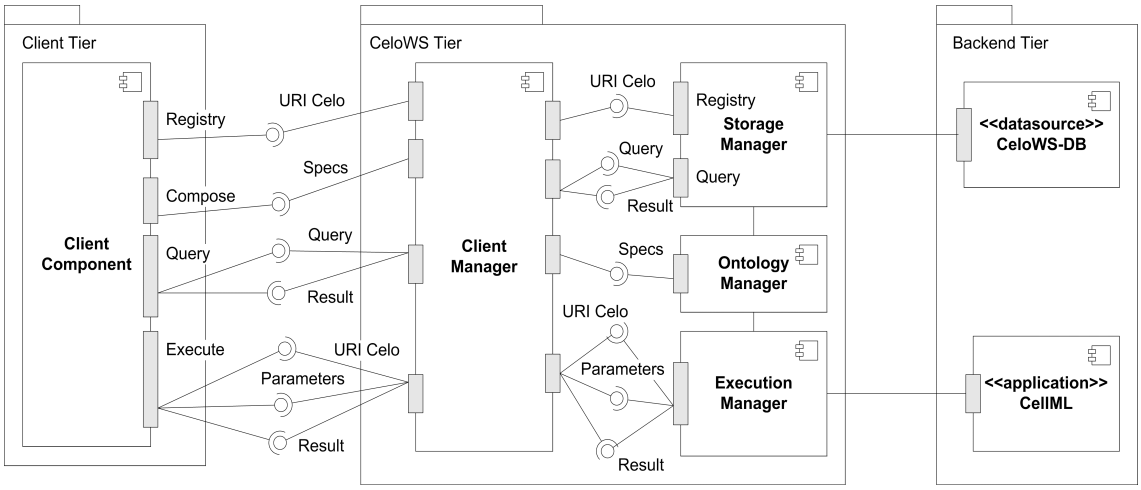

Figura 14. Visão geral da arquitetura do framework CelOWS.

O framework oferece quatro serviços para seus usuários:

- **Registry**: Registro de um modelo no sistema. A partir da URI fornecida pelo usuário, o modelo é recuperado, um mecanismo de inferência é aplicado e o modelo é armazenado em um banco de dados.
- **Compose**: Composição de modelos. O usuário fornece um arquivo XML (Specs) com a especificação de quais modelos devem ser compostos e a arquitetura dessa composição (como os componentes dos modelos serão conectados entre si). Um novo modelo é gerado e armazenado no banco de dados.
- **Query**: Pesquisa de modelos. O usuário fornece uma consulta codificada em SPARQL (SPARQL, 2008) e recebe como resultado as URIs dos modelos, componentes ou variáveis, que atendem à consulta.
- **Execute**: Execução de modelos. O usuário fornece a URI do modelo a ser executado e os parâmetros a serem utilizados (os parâmetros necessários para a execução podem ser obtidos por uma consulta SPARQL prévia). A partir do modelo é recuperada a URI do modelo CellML correspondente e

invocada uma aplicação que faça a simulação do modelo, retornando os resultados obtidos.

Os serviços da camada CelOWS estão distribuídos por quatro módulos gerenciais:

- **Client Manager**: responsável por toda interação com os usuários do framework, implementado com o padrão de projeto Facade (GAMMA, 1995). Sua finalidade é impedir que os clientes tenham acesso à estrutura interna do framework, além de fornecer um ponto único de entrada/saída no sistema.
- **Storage Manager**: responsável pelos processos de armazenamento/recuperação de ontologias na base de dados, além do processamento das consultas (queries) realizadas pelo usuário. Encapsula o acesso ao banco de dados, permitindo que o framework seja independente do Sistema Gerenciador de Banco de Dados.
- **Ontology Manager**: responsável pelos processos de inferência sobre os modelos, bem como por propiciar uma interface programável, em inglês API (*Application Program Interface*), para acesso à ontologia CelO.
- **Execution Manager**: responsável pela execução do modelo CellML associado ao modelo CelO. Deve encapsular o acesso às aplicações de simulação, permitindo que o framework seja independente de uma aplicação específica.

5.3. Cenários de uso

Os seguintes cenários apresentam, de forma resumida, as possibilidades de utilização do framework CelOWS pelos pesquisadores:

- **C1**: A partir de um conceito ou fenômeno biológico de interesse (por exemplo, “canal iônico de sódio” ou “potencial da membrana”), o pesquisador busca na ontologia CelO a sua representação semântica. De posse desta representação, ele pesquisa em um repositório quais modelos estão de alguma forma associados com o conceito e com o fenômeno. Obtendo uma lista de modelos candidatos, ele escolhe um que descreva o

sistema desejado e pode executar simulações ou obter o código CellML para análise.

- **C2**: O pesquisador pode obter um dado modelo CellML, executar simulações e, com base em sua experiência ou análise, acrescentar anotações semânticas ao modelo conceitual associado ao modelo CellML.
- **C3**: O pesquisador pode localizar um modelo existente, possivelmente composto por componentes atômicos, e fazer simulações substituindo estes componentes por outros semelhantes. Os componentes semelhantes, tanto em termos de descrição de um sistema, quanto em relação aos parâmetros de entrada e saída, podem ser localizados através de uma busca semântica.
- **C4**: O pesquisador pode construir um novo modelo, a partir de um modelo vazio ou de um modelo existente, através da composição de componentes atômicos já existentes e descritos semanticamente. Pode ser gerado um modelo CellML para este novo modelo, permitindo sua simulação. O novo modelo também pode ser usado na composição de modelos mais complexos.

5.4. Implementação Computacional

Para implementação e testes do protótipo foram utilizadas as seguintes ferramentas:

- Linguagem PHP versão 5 (PHP, 2008): linguagem de programação de propósito geral, freqüentemente utilizada como linguagem de script para aplicações via web. No caso do protótipo, PHP é usada na criação das interfaces com o usuário.
- Linguagem Java versão 5 (JAVA, 2008): linguagem de programação de propósito geral. No caso do protótipo, Java foi usada para implementação da aplicação no lado servidor, para acessar as diversas APIs relacionadas à ontologias e para as diversas manipulações dos arquivos XML, através de XSLT (*Extensible Stylesheet Language Transformations*) (XSLT, 1999).
- Eclipse versão 3.2 (ECLIPSE, 2008): Ambiente integrado de desenvolvimento, em inglês IDE (*Integrated Development Enviroment*), usado para criação do protótipo. É amplamente extensível, com o uso de plugins, permitindo a edição de diversos tipos de arquivos.

- Protege-OWL Editor versão 3.4 (PROTEGE, 2008): Editor de ontologias OWL, extensível através de plugins.
- Protege-OWL API versão 3.4 (PROTEGE, 2008): API Java para manipulação de ontologias OWL e de regras SWRL, utilizando como base o framework Jena (JENA, 2008).
- Pellet Reasoner versão 1.5.1 (PELLET, 2008): API Java que implementa os mecanismos de inferência. Utilizado juntamente com a API do Protege.
- Jess Rule Engine versão 7 (JESS, 2008): Mecanismo para execução da regras SWRL. O plugin existente no Protege permite apenas a edição das regras. Sua execução é feita através da implementação de uma *bridge* com o framework Jess.
- XML2OWL (AMIN, MORBACH, 2007): aplicação em linguagem C++ que permite a geração da base de um arquivo OWL com indivíduos de uma ontologia base a partir de um arquivo XML, através de regras definidas em um arquivo de configuração. A aplicação original foi adaptada na implementação do protótipo, para permitir maior flexibilidade. Uma nova regra foi incluída para permitir a criação de indivíduos "auxiliares" (que não são incluídos na ontologia) e as regras de criação de propriedades de objeto foram estendidas.
- AGOS (BARBOSA, SANTOS, AMORIM, 2006): AGOS (*API Generator for ODE Solution*) é uma ferramenta que constrói automaticamente uma biblioteca de classe em linguagem C++, que permite ao usuário manipular e resolver numericamente Problemas de Valor Inicial (PVI) baseados em sistemas de Equações Diferenciais Ordinárias, em inglês ODE (*Ordinary Differencial Equation*). Na implementação do protótipo AGOS foi usado para extrair informações sobre as variáveis usadas nas expressões matemáticas dos componentes descritos em CellML. A ferramenta foi adaptada para gerar informações sobre as variáveis de equações algébricas.
- PCEnv versão 0.3 (PCENV, 2008): PCEnv (*Physiome CellML Enviroment*) é uma ferramenta *open-source* para trabalhar com modelos CellML, incluindo a edição de novos modelos, edição de modelos existentes e execução de simulações.

- SOR (*Scalable Ontology Repository*) (LU et al., 2007): uma API especificamente criada para armazenamento e recuperação de ontologias OWL, com suporte ao processo de inferência.

5.5. Integração entre CellML e CelO

A disponibilidade de centenas de modelos descritos de CellML é a principal motivação para a criação de um mecanismo de integração destes modelos com modelos CelO. O modelo CelO representa um modelo CellML enriquecido semanticamente e, quando disponibilizado convenientemente em um repositório, pode ser usado na composição de novos modelos.

5.5.1. Descrição de Modelos CelO

O mapeamento entre modelos CellML/CelO foi feito segundo o seguinte processo (com referência aos elementos CellML descritos na Seção 2.3.3 e às classes da ontologia CelO descritas na Seção 3.3):

- Cada **model** em CellML é um **Model** em CelO;
- Cada **component** em CellML é um **InternalComponent** em CelO;
- Cada **variable** em CellML é uma **InterfaceVariable** ou **LocalVariable** em CelO; de acordo com o atributo **public_interface** é considerada um **ModelParameterIn** ou **ModelParameterOut,** associado a um **ModelInterface;**
- Cada **math** em CellML (expressões matemáticas descritas em MathML) é associada a uma **Equation,** que está associada a um **Component** CelO;

Buscou-se manter alguma compatibilidade estrutural dos modelos CelO com os modelos CellML, visando facilitar a integração dos modelos. O acréscimo, em termos semânticos, aos modelos CellML é feito através das diversas propriedades dos indivíduos. Por exemplo, para cada variável existem propriedades que definem a qual elemento químico ela está associada, qual grandeza ela mede e a qual elemento da célula ela se refere. Da mesma forma, para cada componente é possível obter quais são as variáveis independentes, dependentes, algébricas ou parâmetros.

O workflow para obter um modelo CelO inicial a partir de um modelo CellML é ilustrado na Figura 15, através de um diagrama de atividades UML (*Unified Modeling Language*). A premissa para uso deste workflow é que o modelo CellML seja

“atômico”, ou seja, descreva apenas um componente que possa ser reusado. Os modelos “compostos” (contendo vários componentes) devem ser particionados em modelos atômicos. Estes modelos atômicos podem ser mais facilmente reutilizados, através do processo de composição, descrito na seção seguinte.

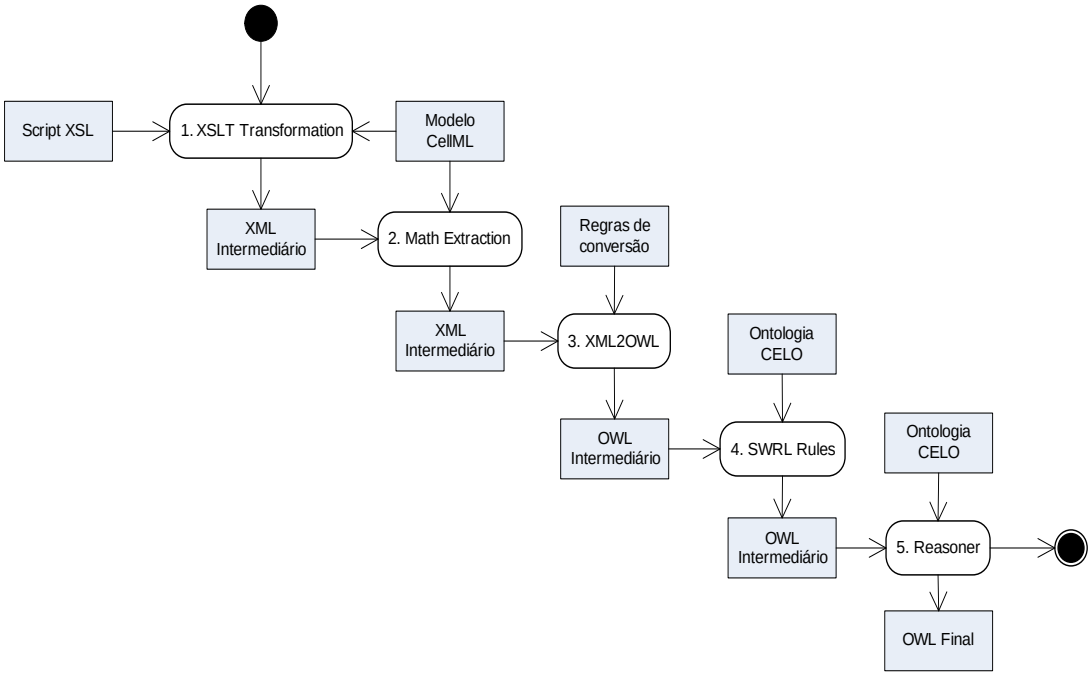

Figura 15. Workflow para obtenção de um modelo CelO inicial a partir de um modelo CellML.

As atividades ilustradas na Figura 15 são:

1. XSLT Transformation: Inicialmente é realizada uma transformação no arquivo XML que representa o modelo CellML, gerando um arquivo XML intermediário, usando um script XSLT. Esta transformação tem dois objetivos: adaptar a estrutura XML tornando-a mais conveniente para a geração do arquivo OWL (Atividade 3) e extrair informação semântica a partir dos identificadores e nomes dos elementos XML. Apesar da falta de padronização de nomes para a identificação dos elementos em CellML, especialmente em variáveis e em componentes, deve-se notar que os nomes carregam alguma semântica. Por exemplo, um componente relacionado a uma concentração do sódio em um canal iônico é etiquetado significativamente e freqüentemente "sodium_channel". Assim, é possível extrair algum conhecimento sobre os componentes a partir de seu nome. A transformação XSLT considera esta

informação e cria no XML intermediário um elemento nomeado "detail" que é usado pelas regras SWRL (Atividade 4) para preencher as propriedades de algumas classes.

2. Math Extraction: Nesta etapa o modelo CellML é analisado e para cada componente (marcador **component**) encontrado é executado o programa AGOS para extrair as variáveis e parâmetros das equações matemáticas. Estas variáveis e parâmetros são formatados como elementos XML e inseridos no arquivo XML intermediário da Atividade 1.
3. XML2OWL: O programa XML2OWL é executado, tendo como entrada o arquivo XML gerado na Atividade 2. Com base nas regras definidas em um arquivo de configuração (Apêndice), é gerado um arquivo OWL intermediário, onde os elementos XML são transformados em indivíduos da ontologia CelO.
4. SWRL Rules: As regras SWRL definidas como parte da ontologia CelO são executadas sobre a ontologia resultante da Atividade 3. Nesta etapa é usado o *framework* Jess Rule Engine através da API Protege. Estas regras preenchem as propriedades das variáveis com as informações obtidas (e registradas) na Atividade 1. Em especial o marcador *detail* é comparado com o nome de vários elementos da ontologia (através da propriedade *hasName*).
5. Reasoner: Após o preenchimento das propriedades, o mecanismo de inferência (*reasoner*) é executado, a fim de verificar a consistência do modelo e classificar as variáveis sob os elementos existentes na ontologia.

Exemplo

Para teste e validação do protótipo foram usados 8 (oito) modelos CellML. Como exemplo, o modelo “A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials” (CELLML, 2008b) é representado na Figura 16 através de um Diagrama de Componentes UML. O diagrama apresenta os componentes e suas conexões (realizadas através das conexões entre os parâmetros das interfaces).

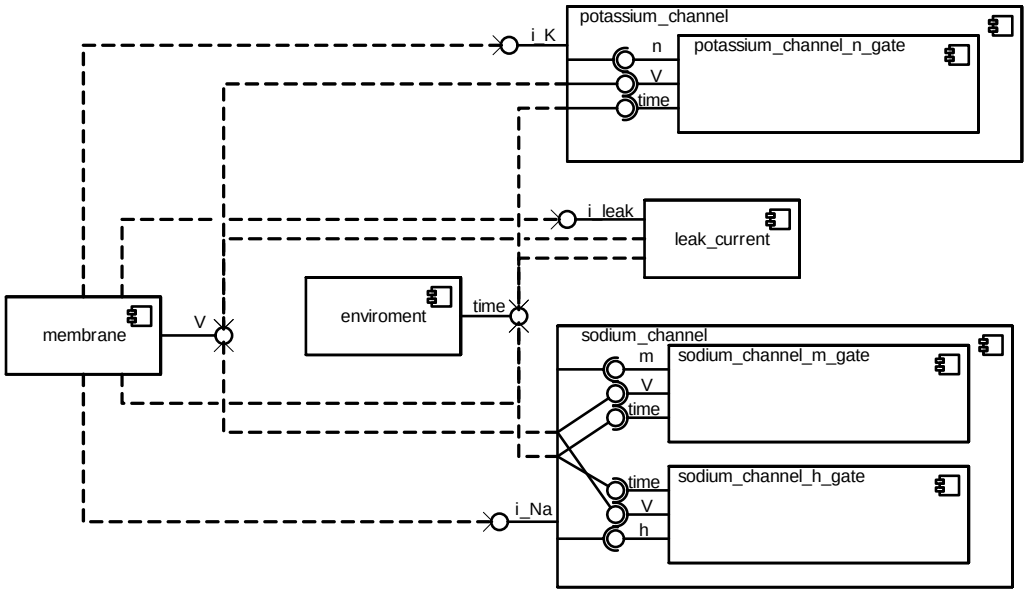

Figura 16. Diagrama de componentes do modelo (CELLML, 2008b).

Como pode ser visto no diagrama, o modelo é composto por oito componentes: *enviroment, membrane, leak_current, potassium_channel, potassium_channel_n_gate, sodium_channel, sodium_channel_m_gate, sodium_channel_h_gate*. Nos testes, o modelo foi particionado, criando-se oito arquivos CellML (modelos atômicos). Em cada arquivo foi aplicado o workflow descrito anteriormente, dando origem a oito modelos CelO. Os arquivos referentes ao modelo original, o modelo atômico da membrana e o modelo CelO da membrana são apresentados no Apêndice 2. Os demais arquivos estão disponíveis no site do projeto (http://celo.mmc.ufjf.br).

5.5.2. Composição de Modelos CelO

A composição de modelos atômicos é um dos objetivos específicos da ontologia CelO. A intenção é promover a reutilização de componentes existentes, através da composição de modelos mais simples, produzindo modelos mais complexos. A composição produz um novo modelo CellML, através da importação de componentes previamente definidos e da conexão automática desses componentes. A composição de modelos é realizada em apenas uma etapa, conforme a Figura 17.

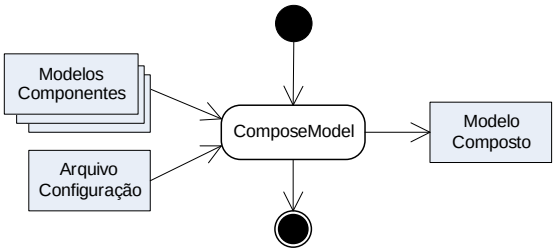

Figura 17. Atividade de composição de modelos CelO.

O processo de composição usa um arquivo de configuração XML, para indicar quais modelos serão compostos e qual a estrutura do novo modelo. A conexão dos componentes é feita através da combinação semântica dos parâmetros do primeiro componente com os parâmetros do segundo componente. É verificado se ambos os parâmetros medem as mesmas grandezas, se ambos estão associados ao mesmo elemento químico e, finalmente, se ambos têm o mesmo nome. Uma consulta SPARQL (Figura 18) é usada para encontrar os parâmetros de saída de um componente que sejam compatíveis com os parâmetros de entrada de outro componente.

```
select ?x0 ?y0 ?x1 ?y1 ?x2 ?y2 ?x3 ?y3 ?x4 ?y4 ?x5 ?y5 ?x6 ?y6
WHERE {<ns1:_ModelInterface> celo:hasInterface ?x1.
?x1 a celo:ModelParameterIn.
OPTIONAL{    ?x1 celo:hasDomainEntity ?x2.
?x2 a celo:CellPart. ?x2 celo:contains ?x5. ?x2 celo:isContainedIn ?x6}
OPTIONAL{    ?x1 celo:hasMeasure ?x3}
OPTIONAL{    ?x1 celo:hasDomainEntity ?x4.
?x4 a celo:ChemicalElement}
?x1 celo:hasName ?x0.
<ns2:_ModelInterface> celo:hasInterface ?y1.
?y1 a celo:ModelParameterOut.
OPTIONAL{    ?y1 celo:hasDomainEntity ?y2.
?y2 a celo:CellPart. ?y2 celo:contains ?y5. ?y2 celo:isContainedIn ?y6}
OPTIONAL{    ?y1 celo:hasMeasure ?y3}
OPTIONAL{    ?y1 celo:hasDomainEntity ?y4.
?y4 a celo:ChemicalElement}
?y1 celo:hasName ?y0
}
```

Figura 18. Consulta SPARQL para busca de parâmetros compatíveis.

A árvore de decisão aplicada aos resultados da consulta para verificar se os parâmetros de dois componentes são compatíveis é apresentada na Figura 19 (as

variáveis X? e Y? são obtidas a partir da consulta SPARQL e correspondem aos valores correspondentes a cada componente).

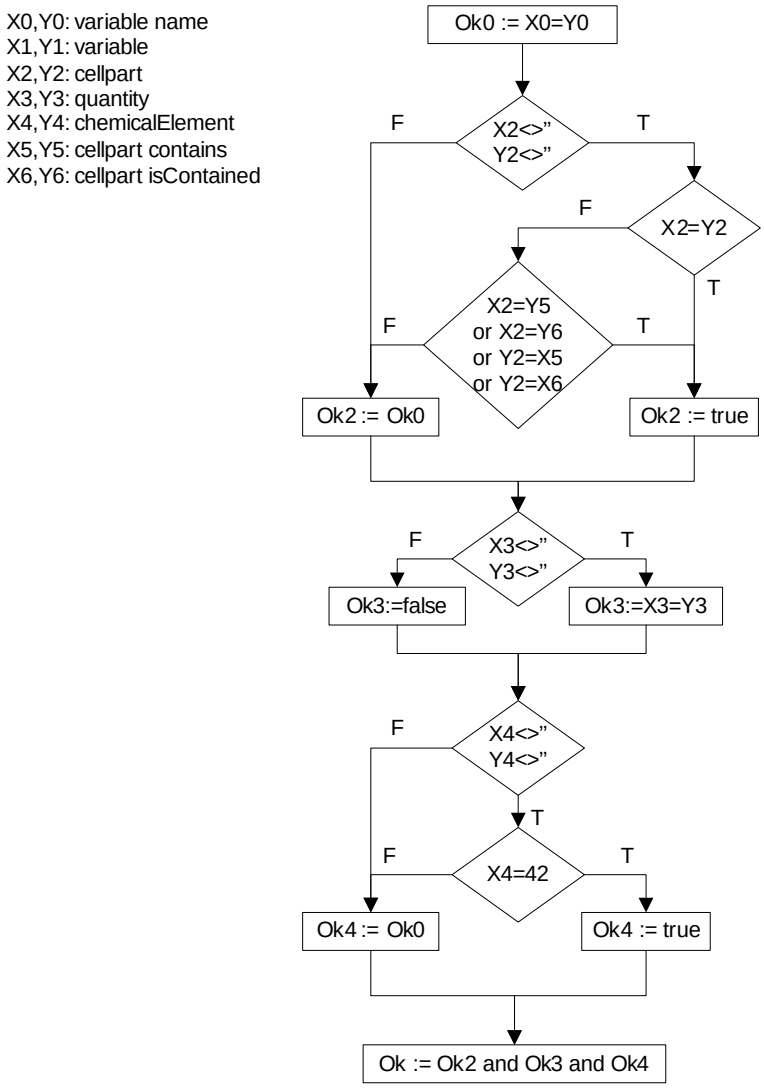

Figura 19. Verificação de compatibilidade entre parâmetros.

Exemplo

Utilizando os oito modelos atômicos descritos no exemplo da seção anterior, foi criado um arquivo de configuração XML (Figura 20) que fizesse a composição desses modelos, dando origem a um modelo CellML compatível com o modelo original.

```
<?xml version="1.0" encoding="UTF-8"?>
<composition>
 <global uri="http://celo.mmc.ufjf.br/owl/noble_1962/environment.owl"/>
 <model uri="http://celo.mmc.ufjf.br/owl/noble_1962/membrane.owl"
uses="global">
  <model uri="http://celo.mmc.ufjf.br/owl/noble_1962/leakage_current.owl"
uses="global"/>
  <model uri="http://celo.mmc.ufjf.br/owl/noble_1962/sodium_channel.owl"
uses="global">
   <model
uri="http://celo.mmc.ufjf.br/owl/noble_1962/sodium_channel_h_gate.owl"/>
   <model
uri="http://celo.mmc.ufjf.br/owl/noble_1962/sodium_channel_m_gate.owl"/>
  </model>
  <model uri="http://celo.mmc.ufjf.br/owl/noble_1962/potassium_channel.owl"
uses="global">
   <model
uri="http://celo.mmc.ufjf.br/owl/noble_1962/potassium_channel_n_gate.owl"/>
       </model>
 </model>
</composition>
```

Figura 20. Arquivo de configuração para composição do modelo.

A Figura 21 mostra o formulário da aplicação cliente protótipo usado para realizar a composição de modelos, com base nos modelos atômicos armazenados no repositório de ontologias. O uso do formulário consiste em 5 etapas:

1. Todos os modelos armazenados no repositório são listados (em uma aplicação real esta etapa implementaria uma consulta que retornasse apenas os modelos de interesse).
2. Os modelos a serem compostos são selecionados.
3. A estrutura do novo modelo é definida, indicando-se quais são os modelos globais, quais modelos usam os modelos globais e quais modelos são internos a outros modelos.
4. O código XML com a configuração é exibido, podendo ser submetido pelo usuário para a criação efetiva do modelo.
5. A localização do arquivo XML contendo o novo modelo CellML gerado é exibida.

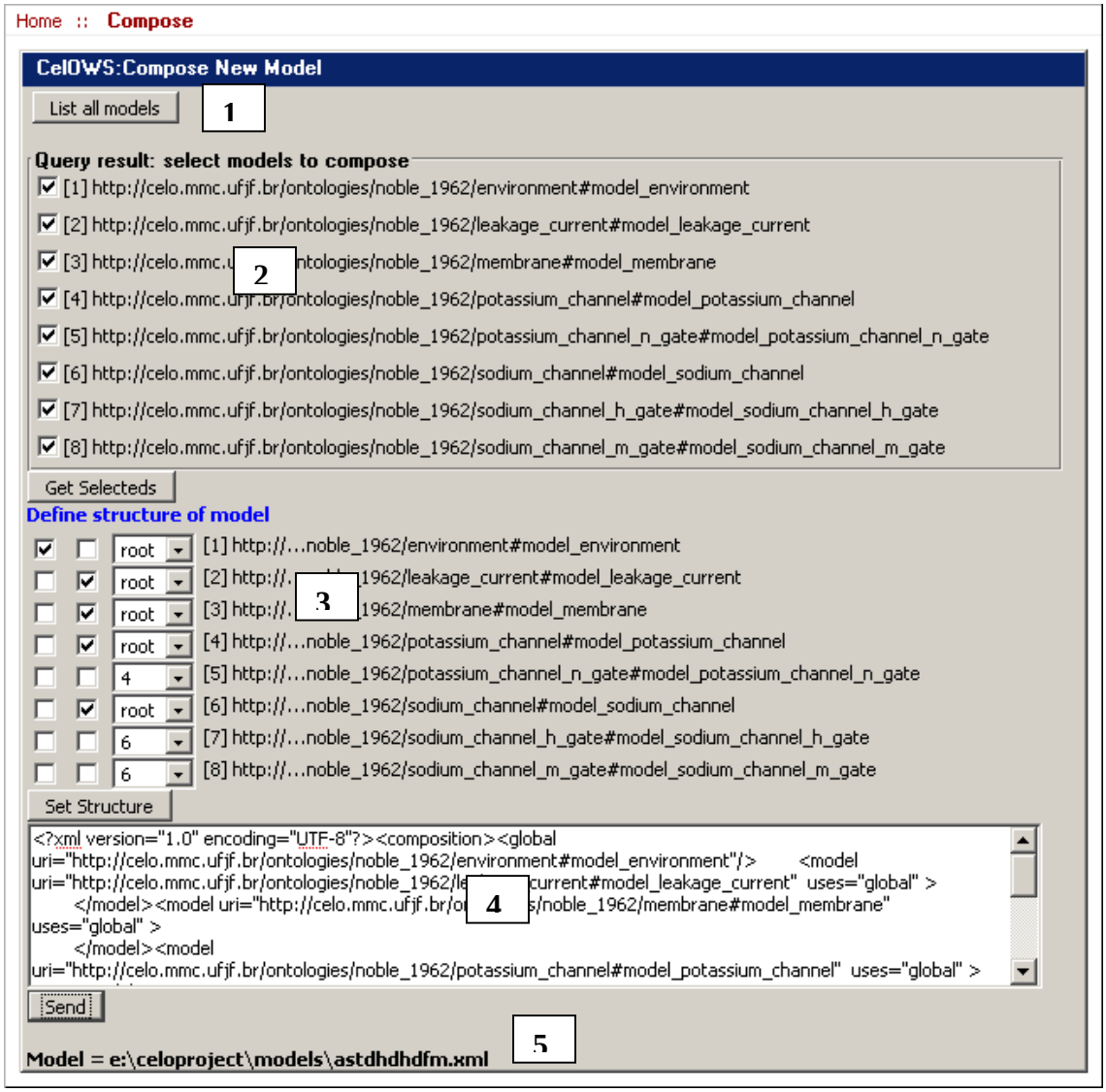

Figura 21. Formulário da aplicação cliente para composição de modelos.

A fim de validar o modelo CellML gerado a partir do modelo CELO, foi utilizado o programa PCEnv versão 0.3, para simulação de modelos. O gráfico *V x Time*, representando o potencial de ação, foi gerado com o arquivo CellML original do modelo "A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials" (Figura 22) e com o modelo gerado através da composição e conexão de componentes (Figura 23). Esses gráficos (como outros gerados em caráter de teste) são idênticos, comprovando a validade do processo de composição.

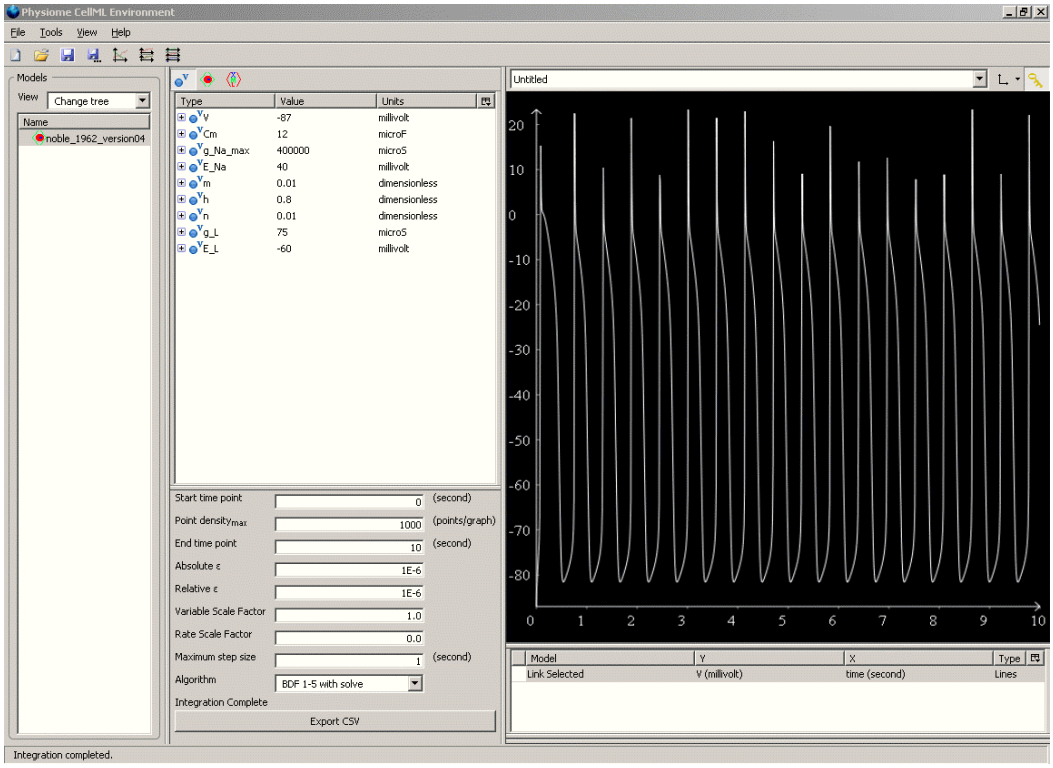

Figura 22. Gráfico *V x time* gerado usando o modelo CellML original.

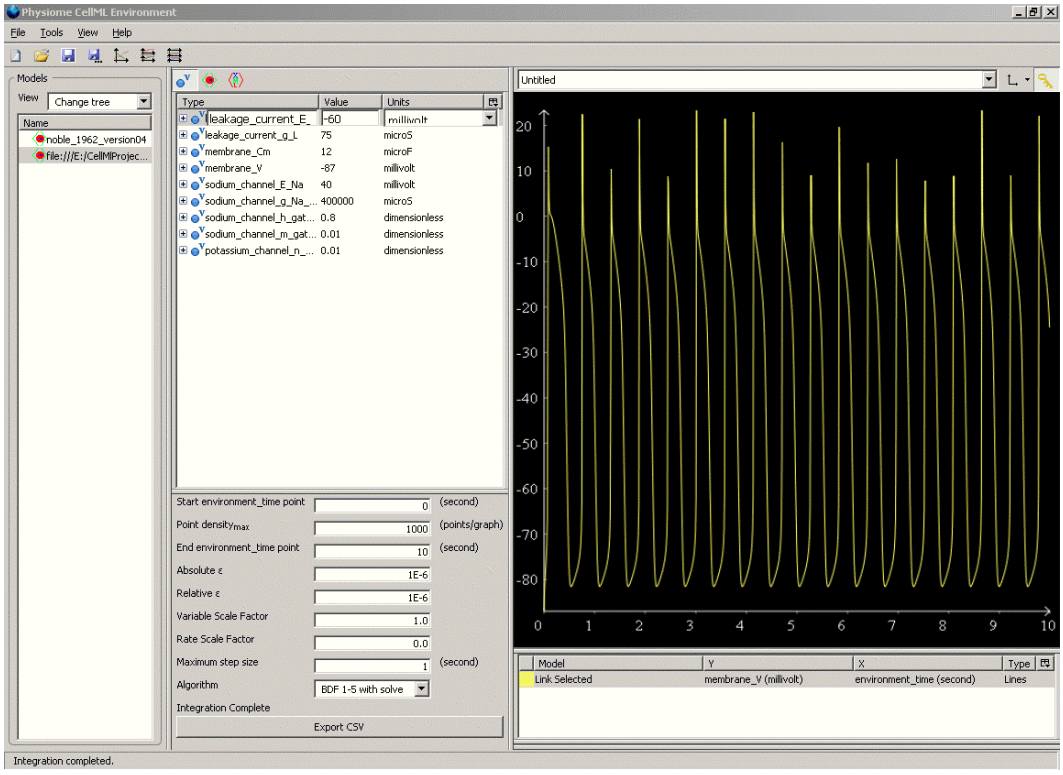

Figura 23. Gráfico *V x time* gerado usando o modelo CellML gerado a partir do modelo CelO.

5.5.3. Pesquisas em Modelos CelO

A fim de possibilitar a busca semântica em modelos CelO foi utilizado na implementação computacional o repositório de ontologias SOR (*Scalable Ontology Repository*). A biblioteca SOR possibilita o armazenamento de ontologias OWL-DL (tanto o componente TBox quanto o componente ABox) em bancos de dados relacionais. SOR suporta o uso de mecanismos de inferência, armazenando não apenas o conhecimento explícito na ontologia, mas também o conhecimento inferido. A ontologia pode ser consultada através do uso de SPARQL.

Segundo LU et al. (2007), da perspectiva do gerenciamento de dados, ontologias poderiam ser consideradas como um modelo de dados (semelhante a um esquema relacional ou esquema XML) e os dados podem ser vistos como instâncias da ontologia (correspondendo ao ABox de uma base de conhecimento). A Figura 23 apresenta os principais componentes da arquitetura SOR.

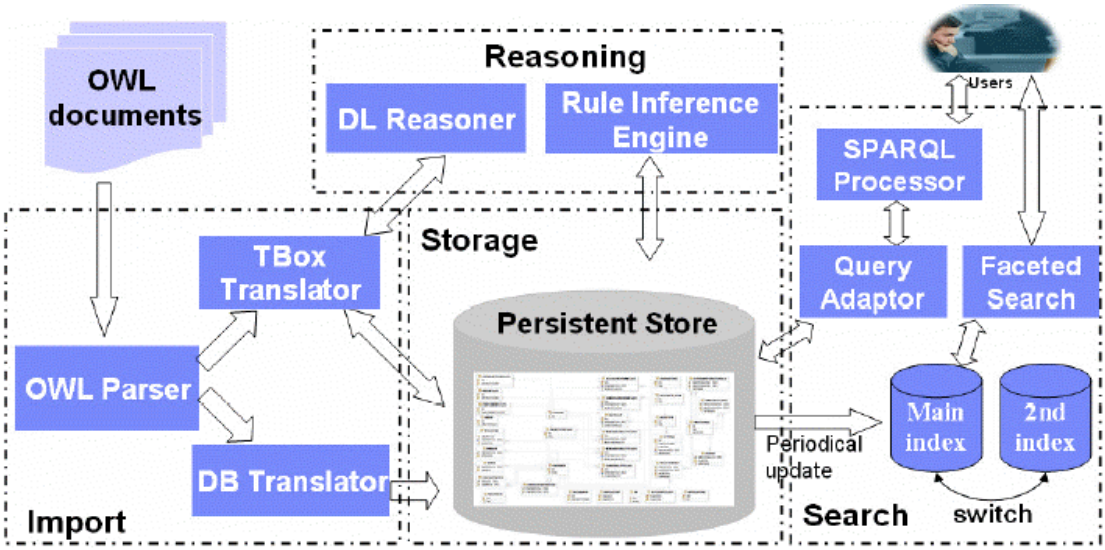

Figura 24. Componentes do SOR (LU et al., 2007).

Exemplo

A Figura 24 apresenta algumas consultas SPARQL usadas para localizar variáveis ou componentes de interesse, que podem ser usados na composição de um novo modelo.

```
Listar todos os modelos
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
      ?x a celo:Model }
```

```
Listar todos os componentes
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
       ?x a celo:Component }

Listar as variáveis associadas com Potencial de Ação
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
       ?x a celo:ActionPotential }

Listar os componetes cujas variáveis estejam associadas com Potencial de Ação
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
       ?x a celo:Component .
       ?y celo:isInterfaceVariableOf ?x
       ;a celo:ModelParameter
       ;a celo:ActionPotential }

Exibir os componentes com parâmetros associados a medição de corrente elétrica
e elemento químico sódio
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:Component .
?y celo:isInterfaceVariableOf ?x
       ;a celo:ModelParameter
       ;celo:hasDomainEntity celo:Sodium
       ;celo:hasMeasure celo:EletricCurrent
}
```

Figura 25. Consultas SPARQL para buscas no repositório de ontologias.

A Figura 26 mostra o formulário da aplicação cliente usado para realizar consultas no repositório de ontologias.

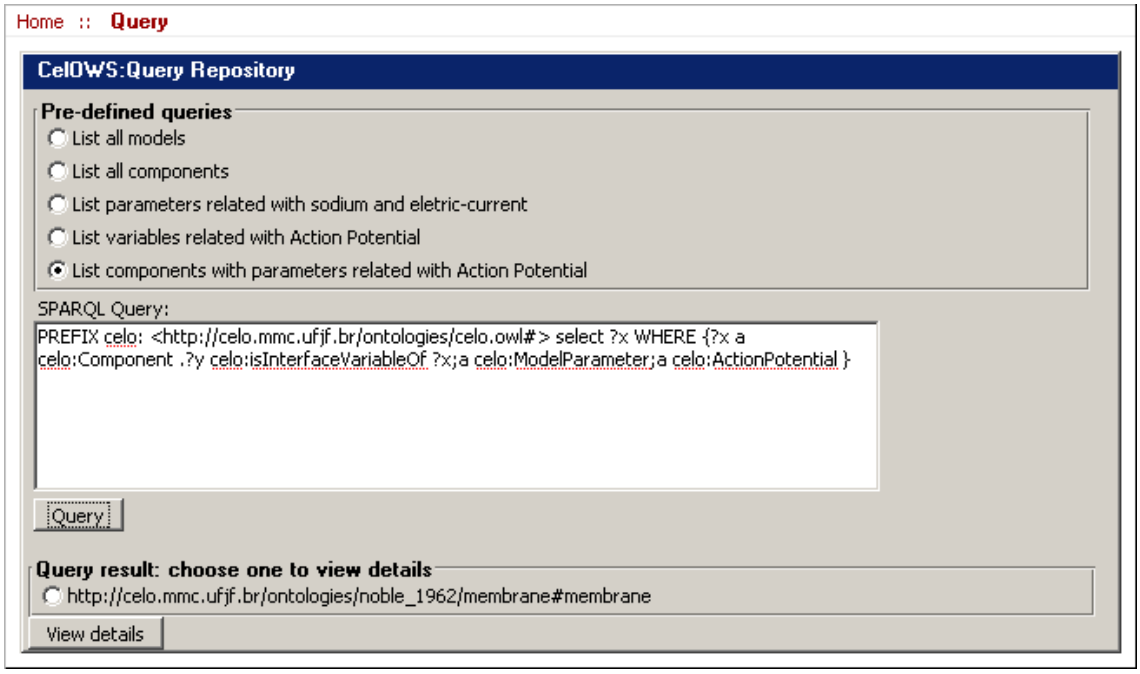

Figura 26. Formulário da aplicação cliente para consultar modelos.

5.6. Trabalhos relacionados

A discussão sobre o uso de ontologias em CellML vem ocorrendo entre os autores de CellML há alguns anos (NIELSEN et al, 2003). Durante o CellML Workshop 2007 o tema foi novamente discutido (HALSTEAD, 2007). O debate levantou algumas questões sobre ontologias em CellML. A ontologia CelO, proposta nesse trabalho, é uma contribuição para esta discussão por representar os modelos de célula usando linguagens ontológicas e estar integrado com CellML.

Um trabalho com ontologias aplicadas aos modelos de célula é descrito em (SHIMAYOSHI et al., 2006). Os autores apresentam métodos para suporte ao desenvolvimento de estruturas de modelos complexos das células, usando a linguagem de marcação PMSML (*Physiological Model Structure Markup Language*) e uma ontologia chamada *Cell Model Ontology*. A ontologia CelO tem um objetivo muito similar, mas usa linguagens mais padronizadas, como OWL, SWRL e XSLT. Isto permite o uso de uma grande variedade de ferramentas, além da possibilidade de um trabalho colaborativo no desenvolvimento da ontologia.

Uma abordagem usando a associação de ontologias e serviços web para apoiar a modelagem em sistemas biológicos é descrita em (SUN, FINKELSTEIN, ASHMORE,

2007). É apresentada uma ontologia para representar o meta-modelo em OWL. OWL-S é usado para especificar a parametrização e composição semi-automática de serviços web voltados para a execução do modelo. No entanto, o foco está no controle das transformações em bancos de dados biológicos e transformação dos métodos de análise em serviços web. A proposta do framework CelOWS, embora também trabalhando com modelos, é propiciar facilidades ao pesquisador nas etapas de modelagem e simulação, e não na etapa de análise.

Em (MARGONINSKI et al., 2006) é apresentada uma linguagem para especificação de modelos e um framework para execução de modelos compostos em Biologia Sistêmica. A linguagem proposta é chamada CDML (*Composite Model Description Language*), e embora seja mais expressiva que CellML, é também baseada em XML, apresentando as dificuldades já discutidas anteriormente.

5.7. Comentários Finais

Este capítulo apresentou o framework CelOWS, uma aplicação de infraestrutura para gerência de modelos biológicos descritos semanticamente, apoiado na ontologia CelO. Foram apresentados os módulos do framework e seus serviços disponibilizados para os pesquisadores. Implementado como um serviço web, ele pode ser utilizado e integrado em aplicações existentes. A implementação computacional do framework foi descrita, juntamente com as ferramentas utilizadas. Exemplos de utilização do framework ilustraram sua viabilidade.

6. Considerações Finais

O crescente volume e a distribuição de dados e processos em Bioinformática tornam cada vez mais fácil a descoberta de novas informações biológicas. Entretanto, como são inúmeras as análises que podem ser feitas, os pesquisadores precisam da ajuda de sistemas que os auxiliem em seus trabalhos. Quando esses dados e processos precisam ser combinados e gerenciados de forma automatizada e escalável, o uso de workflows científicos torna-se imprescindível (GANNON et al., 2007).

Por outro lado, os modelos desenvolvidos na área da Biologia possuem diversas representações possíveis. Essas representações podem ser classificadas em níveis de abstração. No projeto Physiome (HUNTER, 2004), as representações conceitual, matemática e computacional são destacadas, apresentando-se a necessidade de integração entre esses diversos níveis.

Embora seja desejável a associação destas duas áreas, workflows e modelagem, a literatura pesquisada para esta dissertação não apresenta propostas específicas nessa direção. Acreditamos que isto acontece porque geralmente os modelos são considerados apenas no seu aspecto de “representações”, enquanto workflows lidam com componentes de software que podem ser “executados”.

Os estudos em Computação Orientada a Serviços (SINGH, HUHNS, 2007) tendem à modificação desse quadro. A necessidade de representações semanticamente mais ricas já foi detectada tanto na área de workflows científicos (GIL, 2007), quanto na área de modelagem em Biologia (LLOYD, HALSTEAD, NIELSEN, 2004). Representações semânticas de workflows podem ajudar na composição assistida de modelos, enquanto expressar os modelos biológicos em um nível mais alto facilita a pesquisa e o reuso dos mesmos.

Neste contexto, este trabalho propõe mecanismos inovadores, embora baseados em padrões já bem estabelecidos, que propiciam a aproximação das duas áreas. Associamos a representação semântica com o tratamento de modelos como serviços web, permitindo que as tarefas de modelagem possam ser coordenadas em uma Arquitetura Orientada a Serviços, em inglês, SOA (*Service-Oriented Architeture*).

Esta dissertação justificou a importância do tema e da abordagem adotada e apresentou a proposta do framework CelOWS baseado em ontologia para um AMS (Ambiente de Modelagem e Simulação) em Biologia Sistêmica. Um protótipo foi

construído instanciando-se o framework com o uso de diversas ferramentas usadas em aplicações para **web semântica**. Foram realizados testes e experimentos com oito modelos em eletrofisiologia celular. Estes modelos, expressos em CellML, foram utilizados nos processos de integração, composição e pesquisa descritos no capítulo 4.

Finalmente, as questões propostas na Introdução (seção 1.2), podem ser respondidas, levando-se em consideração o número de modelos testados, a área da biologia sob análise e o protótipo desenvolvido:

1. O conhecimento implícito, extraído de um modelo CellML, é significativo somente se o nome dos elementos (componentes e variáveis) também for significativo. Por "significativo" entenda-se o fato dos nomes utilizados expressarem, *per si*, alguma característica semântica. Uma variável associada à medição da corrente elétrica provocada pelo fluxo de íons de sódio, pode ser chamada "I_Na". Este nome é considerado significativo, neste contexto, pois "I" é um símbolo para representar "corrente elétrica" e "Na" é o símbolo do elemento químico sódio. A abordagem usada no trabalho busca explorar estes significados. No entanto, como CellML (assim como outras linguagens propostas para modelagem em biologia) é uma linguagem baseada em XML, nada impede que a mesma variável seja chamada de "x", perdendo-se toda a semântica associada. O uso de um modelo conceitual, por exemplo expresso com a ontologia CelO, registra a semântica através das propriedades associadas à variável, e não ao seu nome.
2. Embora a ontologia CelO (Cell Component Ontology), proposta no trabalho, apresente uma série de classes para mapemento dos conceitos do domínio da Biologia, Química e BioQuímica (definidas sob a classe DomainEntity), a integração destas classes com ontologias consagradas em Biologia é fundamental em uma implementação efetiva. Esta integração permitirá uma compreensão mais clara do modelo em estudo pelo pesquisador, pelo uso de um vocabulário compartilhado já conhecido. Mesmo em um domínio restrito, como o adotado, o conhecimento implícito em um modelo é muito amplo e a ontologia proposta é incapaz de cobrir este conhecimento. Por outro lado, as classes referentes à modelagem (ModelEntity) e ao Sistema Internacional de Unidades

(SIEntity) são efetivas contribuições dentro da proposta apresentada pelo trabalho.

O framework CeloWS é implementado como um serviço web, sendo uma infraestrutura que pode ser integrada em diferentes tipos de ambientes de modelagem. Os serviços oferecidos auxiliam os pesquisadores na definição, validação, composição e execução de modelos, podendo ser usados tanto em um ambiente de workflow científico quanto em outros ambientes em que o processamento seja orientado a serviços.

3. A abordagem proposta tratou a questão da integração entre modelos e workflows considerando que modelos biológicos podem ser, e efetivamente são, representados também de forma computacional para que possam ser simulados. Desta forma, os modelos podem ser considerados "serviços". Mais especificamente podem ser encapsulados em serviços web. O uso de ontologias neste cenário permite que estes modelos sejam tratados como serviços web semânticos, tornando possível que ferramentas de workflow, já maduras, possam ser usadas nos processos de composição e análise de modelos, embora este uso não tenha sido implementado, sendo proposto como um trabalho futuro.
4. A introdução de mais um nível de abstração facilita o trabalho de modelagem pelo pesquisador somente se ele dispuser de ferramentas adequadas. O protótipo não foi desenvolvido com a intenção de ser usado diretamente pelo usuário final (o pesquisador), mas sim para testar a validade da abordagem proposta. Um nível mais alto de abstração, com o uso de ontologias, possibilita o uso de um vocabulário consagrado na área, a validação conceitual do modelo, a associação direta com metadados existentes, a aplicação de regras semânticas e uma representação formal compartilhável. No entanto, ferramentas gráficas, visuais, interativas e de fácil utilização são fundamentais para que o pesquisador possa acessar estes benefícios.

6.1. Trabalhos Futuros

Como a abordagem proposta na dissertação possui um caráter inovador, embora todos os elementos que a compõe (modelagem conceitual, ontologias, serviços web) sejam

bastante conhecidos e maduros, é possível destacar vários trabalhos futuros a partir da base proposta.

O primeiro trabalho, naturalmente, é a ampliação da ontologia CelO em relação aos conceitos associados ao domínio, para abranger outros tipos de modelos biológicos e para referenciar as ontologias de domínio que já estão bem estabelecidas. A ontologia deve ser ampliada, também, em relação aos conceitos associados ao domínio da matemática, para que o conhecimento intrínseco nas equações possa ser explorado. Essas ampliações permitirão melhorar a classificação semântica dos parâmetros e componentes dos modelos, provendo maiores recursos para a pesquisa e a composição dos modelos.

As classes relativas à modelagem (**ModelEntity**) devem ser avaliadas com um número maior e mais diversificado de modelos biológicos, com atenção especial relativa à representação de modelos compostos. Estas informações poderão melhorar a definição e a execução otimizada dos workflows científicos.

Outro trabalho derivado é relativo ao uso da ontologia CelO com outras linguagem de modelagem baseadas em XML, como SBML (*Systems Biology Markup Language*) e CMDL (*Composite Model Description Language*). O trabalho inicial, ainda que buscando uma visão generalista, foi fortemente influenciado pela especificação de CellML.

A integração do framework CelOWS com frameworks para workflow científico também pode ser explorada a fim de que a tarefa de modelagem, principalmente a composição de modelos existentes, possa ser facilitada.

Por fim, a criação de editores gráficos e interativos é fundamental. O uso de elementos visuais que possam ser facilmente reconhecidos e compostos, tendo uma semântica bem definida, com alto grau de interação, é um meio muito natural para os pesquisadores construírem e interpretarem modelos. O uso de ontologias, definindo de forma inequívoca os conceitos do domínio, é um requisito já atendido pelo framework CelOWS.

Referências Bibliográficas

ALMEIDA, M. B., BAX, M. P. “Taxonomia para projetos de integração de fontes de dados baseados em ontologias”. *V Encontro Nacional de Pesquisa em Ciência da Informação*. Belo Horizonte, 2003.

AMIN, M. A., MORBACH, J. “XML to OWL Converter”. Disponível em: http://www.lpt.rwth-aachen.de/Publication/abstractND.php?Nummer=LPT-2007-02. Acesso em 30 mar. 2008.

ANTEZANA, E., TSIPORKOVA, E., MIRONOV, V. Et al. “A cell-cycle knowledge integration framework”. *DILS 2006*, LNBI 4075, pp. 19-34, 2006.

BARBOSA, C., SANTOS, R., AMORIM, R. et al. “A Transformation Tool for ODE based models”. *Lecture Notes in Computer Science 3991*, pp. 69–75. 2006.

BECHHOFER, S. et al. “OWL Web Ontology Language 1.0 Reference”. Disponível em: http://www.w3.org/TR/owl-ref/. Acesso em 30 mar. 2008.

BEELER, G.W.,REUTER, H. “Reconstruction of the action potential of ventricular myocardial fibres”. *Journal of Physiology* , 268, 177-210. 1977.

BERNERS-LEE, T. HENDLER, J., LASSILA, O. “The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”. *Scientific American*, 01/05/2001. Disponível em: http://www. scientificamerican.com/ 2001/0501issue/0501bernerslee. Acesso em 30 mar. 2008.

BORGIDA, A., BRACHMAN, R. J. “Conceptual Modeling with Description Logics”. In: BAADER, F. et al. *Description Logic Handbook: theory, Implementation, and Applications*, Eds. Cambridge University Press, New York, NY, 2003. p. 359-381.

BORST, W.N., 1997. *Construction of Engineering Ontologies*. Phd Thesis.Disponível em: http://www.ub.utwente.nl/webdocs/inf/1/t0000004.pdf. Acesso em 30 mar. 2008.

BRUIJIN, J. de. “Logics for Semantic Web”. In: CARDOSO, J. *Semantic Web Services: Theory, Tools and Applications*. New York: Information Science Reference. 2007. Cap. 3, p. 24-43.

CARDOSO, J. “The Syntactic and the Semantic Web”. In: CARDOSO, J. *Semantic Web Services: Theory, Tools and Applications*. New York: Information Science Reference. 2007. Cap. 1, p. 1-23.

CARDOSO, J., SHETH, A. “The Semantic Web and its applications”. In: CARDOSO, J. *Semantic Web Services, Process and Applications*. Springer Science+Business Media, LLC. 2006. Cap. 1, p. 3-33.

CellML. 2008. “CellML – Cell Markup Language”. Disponível em http://www.cellml.org. Acesso em 30 mar. 2008.

_____. 2008b. “A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials”. Disponível em: http://www.cellml.org/models/noble_1962_version05. Acesso em 30 mar. 2008.

CELLML REPOSITORY. 2008. “The Modification of the Hodgkin-Huxley Equations Applicable you it Purkinje Fibre Action and Pace-Maker Potentials” Disponível em: http://www.cellml.org/models/ noble_1962_version05. Acesso em 30 mar. 2008.

COOLSCHOOL. 2008. “Transmission of a Nerve Impulse”. Disponível em: http://www.coolschool.ca/lor/BI12/unit12/U12L02.htm. Acesso em 30 mar. 2008.

CORCHO, O., FERNANDEZ-LOPEZ, M., GOMEZ-PEREZ, A. “Ontological Engineering: What are ontologies and how can we build them?” . In: CARDOSO, J. *Semantic Web Services: Theory, Tools and Applications*. New York: Information Science Reference. 2007. Cap. 3, p. 44-70.

CUELLAR, A.A. et al.. “CellML Specification 1.1”. 2006. Disponível em: http://www.cellml.org/specifications/cellml_1.1. Acesso em 30 mar. 2008.

CUELLAR, A. A., NELSON, M., HEDLEY, W. “CellML Metadata 1.0 Specification”. 2006. Disponível em: http://www.cellml.org/specifications/metadata/cellml_metadata_ 1.0. Acesso em 30 mar. 2008.

DAVENPORT, T. H., PRUSAK, L., *Conhecimento empresarial: como as organizações gerenciam o seu capital intelectual*. 3º ed. Rio de Janeiro: Campus, 1998.

ECLIPSE. 2008. “Eclipse - an open development platform”. Disponível em: http:// www.eclipse.org. Acesso em 30 mar. 2008.

FACT. 2008. “FACT++”. Disponível em: http://owl.man.ac.uk/factplusplus/. Acesso em 30 mar. 2008.

FITZHUGH, R.A. “Impulses and physiological states in theoretical models of nerve membrane”. 1961, Biophys. J., 1, 445-466.

FREITAS, F. “Ontologias e a Web Semântica”. In: *Anais do XXIII Congresso da Sociedade Brasileira de Computação*. Volume 8: Jornada de Mini-Cursos em Inteligência Artificial ed.Campinas : Sociedade Brasileira de Computação (SBC), v.8, p. 1-52. 2003.

GAMMA et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.

GANNON, D. et al. “Introduction”. In: TAYLOR, I. J et al. *Workflows for e-Science – Scientif Workflows for Grids*. Springer Science+Business Media, LLC. 2007. Cap. 1, p. 1-8.

GARNY A. et al. “CellML and Associated Tools and Techniques”. *Phil Trans Roy Soc A*, 2007.

GARNY, A., KOHL, P., NOBLE, D. “Cellular Open Resource (COR): a public CellML based environment for modelling biological function”. *International Journal of Bifurcation and Chaos* 13:3579-3590. 2003.

GIL, Y. “Workflow Composition: Semantic representations for flexible Automation”. In: TAYLOR, I. J et al. *Workflows for e-Science – Scientif Workflows for Grids*. Springer Science+Business Media, LLC. 2007. Cap. 16, p. 244-257.

GO. “Gene ontology: tool for the unification of biology”. *Nature Genetics 25*, pp. 25-29. 2000.

GÓMEZ-PÉREZ, A., 1999. “Tutorial on Ontological Engineering”. *Internacional Joint Conference on Artificial Intelligence – IJCAI’1999*. Estcolmo, Suécia. Disponível em: http://www.ontology.org/main/papers/madrid-tutorials.html. Acesso em 30 mar. 2008.

GRUBER, T. R., 1994. “Towards principles for the design of ontologies used for knowledge sharing”. In N. Guarino and R. Poli (Eds.), *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer.

GRUBER, T. “A Translation Approach to Portable Ontology Specification”, *Proceedings of Japanese Knowledge Acquisition Workshop (JKAW92)*, 1992.

HALSTEAD, M. “Ontologies and Repositories”. *CellML Workshop*. 2007. Disponível em: http://www.cellml.org/workshop2007. Acesso em 30 mar. 2008.

HEUSER, C. A. *Projeto de Banco de Dados*. Editora Saggra-Luzzato. Porto Alegre, 1998.

HORN LOGIC. 2006. “What is Horn Logic”. Disponível em: http://www.w3.org/2005/rules/wg/ wiki/Horn_Logic. Acesso em 30 mar. 2008.

HORROCKS, I. et al. 2004. “SWRL: A Semantic Web Rule Language Combining OWL and RuleML.W3C Member Submission 21 May 2004”. Disponível em: http://www.w3.org/Submission/SWRL. Acesso em 30 mar. 2008.

HORROCKS, I.. “Reasoning with expressive description logics: Theory and practice.” In Andrei Voronkov, editor, *Proc. of the 19th Int. Conf. on Automated Deduction (CADE 2002)*, number 2392 in Lecture Notes in Artificial Intelligence, pages 1-15. Springer, 2002.

HUNTER, P. J., “The IUPS Physiome Project: a framework for computational physiology”. *Progress in Biophysics & Molecular Biology 85*, pp. 551–569. 2004.

JAVA. 2008. “Java Platform, Enterprise Edition (Java EE)”. Disponível em: http://java.sun.com/javaee/. Acesso em 30 mar. 2008.

JENA. 2008. “Jena – A Semantic Web Framework for Java”. Disponível em: http://jena.sourceforge.net/index.html. Acesso em 30 mar. 2008.

JESS. 2008. “JESS, the Rule Engine for the Java Platform”. Disponível em: http://herzberg.ca.sandia.gov. Acesso em 30 mar. 2008.

KALYANPUR, A., PARSIA, B., HENDLER, J. “A Tool for Working with Web Ontologies”, In: *Proceedings of the International Journal on Semantic Web and Information Systems*, Vol.1, No.1, Jan-Mar 2005

KANGASSALO, H. “On the concept of concept for conceptual modeling and concept detection”. In: OHSUGA, S. et al., *Information Modelling and Knowledge Bases III*. IOS Press, 1992.

KITANO, H. “Systems Biology: A Brief Overview”. *Science* 1 March 2002: Vol. 295. no. 5560, pp. 1662 – 1664. DOI: 10.1126/science.1069492

KNUBLAUCH, H., FERGERSON, R. W., NOY, N. F. et al. “The Protégé owl plugin: An open development environment for semantic web applications”. In: 3rd International Semantic Web Conference. Hiroshima, Japan. 2004.

LLOYD, C. M., HALSTEAD, M. D. B., NIELSEN, P. “CellML: its future, present and past”. *Progress in Biophysics & Molecular Biology 85*. pp 433–450. 2004.

LU, J. et al. “SOR: a practical system for ontology storage, reasoning and search”. In *Proceedings of the 33rd international Conference on Very Large Data Bases* (Vienna, Austria, September 23 - 27, 2007). Very Large Data Bases. VLDB Endowment, 1402-1405. 2007.

LUO, C, RUDY, Y. “A Dynamic Model of the Cardiac Ventricular Action Potential - Simulations of Ionic Currents and Concentration Changes”, Circulation Research , 74, 1071-1097. 1994.

LUO, C, RUDY, Y. “A Model of the Ventricular Cardiac Action Potential - Depolarisation, Repolarisation and Their Interaction”, *Circulation Research* , 68, 1501-1526. 1991.

LUTZ, C., 2006. “Description Logics Home Page”. Disponível em: http://dl.kr.org. Acesso em 30 mar. 2008.

MACEDO, J.A. F.; HAEUSLER, E. H., 2005, *Um modelo conceitual para biologia molecular*. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, RJ, Brasil.

MALMIVUO, J.; PLONSEY, R. *Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields*. Oxford University Press, New York, 1995

MARGONINSKI, O. et al. “A Specification Language and a Framework for the Execution of Composite Models in Systems Biology”. *Lecture Notes in Computer Science Transactions on Computational Systems Biology, VII*, LNBI 4230. 2006

MARIA, A. “Introduction to Modeling and Simulation”. Proceedings of the *Winter Simulation Conference*. 1997.

MARJOMAA, E. “Necessary Conditions for High Quality Conceptual Schemata: Two Wicked Problems”. *Journal of Conceptual Modeling* .December 2002, issue: 27. Disponível em http://www.inconcept.com/JCM/Dec%202002/Marjomaa.html. Acesso em 30 mar. 2008.

MARTIN, D. et al. 2004. “OWL-S: Semantic Markup for Web Services; W3C Member Submission 22 November 2004”. Disponível em: http://www.w3.org/Submission/ OWL-S/. Acesso em 30 mar. 2008.

MATHEUS, C. J. et al. “Using SWRL and OWL to Capture Domain Knowledge for a Situation Awareness Application Applied to a Supply Logistics Scenario”. In: Asaf Adi, Suzette Stoutenburg, Said Tabet (Eds.): *Rules and Rule Markup Languages for the Semantic Web*, First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005.

MATHML. 2001. “Mathematical Markup Language (MathML)”. Disponível em http://www.w3.org/Math/. Acesso em 30 mar. 2008.

MATOS, E. et al. “MathWS: Broker de Serviços Web para e-Science”. *1st Brazilian e-Science WorkShop. in conjunction with 22nd Brazilian Symposium on Data Base*, João Pessoa, Brazil, 2007.

MERSON, P. “Minicurso: Como documentar arquitetura de software”. *XIX Simpósio Brasileiro de Engenharia de Software*, 2005. Disponível em: http://www.sbbd-sbes2005.ufu.br/arquivos/Merson05_minicurso_SBES1.pdf. Acesso em 30 mar. 2008.

NARDI, D., BRACHMAN, R. J., “An introduction to description logics”. In: F. BAADER et al, *Description Logic Handbook: theory, Implementation, and Applications*, Eds. Cambridge University Press, New York, NY, 2003. p. 1-40.

NELSON, M. E. “Electrophysiological Models” In: *Databasing the Brain: From Data to Knowledge*. S. Koslow and S. Subramaniam (eds.) pp. 285-301, Wiley, New York. 2005.

NICKERSON, D., HUNTER, P. “The Noble cardiac ventricular electrophysiology models in CellML”. *Prog Biophys Mol Biol*, Jan-Apr;90(1-3):346-59, 2006.

NIELSEN P. et al. “Ontologies in CellML: A Versatile Method to Describe Cellular Models”. *IIth International conference on Intelligent Systems for Molecular Biology (ISMB)*. 2003.

NISO. 2005. “Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies”. *National Information Standards Organization.* Disponível em http://www.niso.org/standards/standard_gather.cfm? pdflink= http:// www.niso.org/standards/resources/Z39-19-2005.pdf&std_id=814. Acesso em 30 mar. 2008.

NONAKA, I.; TAKEUCHI, H., *Criação do conhecimento na empresa*. Rio de Janeiro: Campus, 1997.

NOY, N. 2005. “Representing Classes As Property Values on the Semantic Web. W3C Working Group Note 5 April 2005”. Disponível em: http://www.w3.org/TR/swbp-classes-as-values. Acesso em 30 mar. 2008.

NOY, N., RECTOR, A. 2006. “Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006”. Disponível em: http://www.w3.org/TR/swbp-n-aryRelations. Acesso em 30 mar. 2008.

OBO. “The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration”. *Nature Biotechnology 25*, 1251 – 1255. 2007.

O'CONNOR, M. et al. “Supporting Rule System Interoperability on the Semantic Web with SWRL”. *Fourth International Semantic Web Conference (ISWC2005)*, Galway, Ireland, 974-986. 2005.

OWLAPI. 2008. “The OWL API”. Disponível em: http://owlapi.sourceforge.net. Acesso em 30 mar. 2008.

PCENV. 2008. “Physiome CellML Environment” http://www.cellml.org/downloads/ pcenv. Acesso em 30 mar. 2008.

PELLET. 2008. “Pellet: The Open Source OWL DL Reasoner”. Disponível em: http://pellet.owldl.com. Acesso em 30 mar. 2008.

PHP. 2008. “PHP Home Page”. Disponível em: http://www.php.net/. Acesso em 30 mar. 2008.

PROTEGE. 2008. “Protege OWL – Ontology Editor for Semantic Web”. http://protege.stanford.edu/plugins/owl/. Acesso em 30 mar. 2008.

RACER. 2008. “RACER Manager”. Disponível em: http://www.racer-systems.com /index.phtml. Acesso em 30 mar. 2008.

RDF. 2002. “Resource description framework (RDF)”. Disponível em http://www.w3.org/RDF/. Acesso em 30 mar. 2008.

RDFS. 2004. “RDF vocabulary description language 1.0: RDF schema”. Disponível em http://www.w3.org/TR/rdf-schema/ Acesso em 30 mar. 2008.

RECTOR, A. 2005. “Representing Specified Values in OWL: "value partitions" and "value sets". W3C Working Group Note 17 May 2005”. Disponível em: http://www.w3.org/TR/swbp-specified-values. Acesso em 30 mar. 2008.

RECTOR, A; WELTY, C. 2005. “Simple part-whole relations in OWL Ontologies. W3C Editor's Draft 11 Aug 2005”. Disponível em: http://www.w3.org/2001/sw/ BestPractices/OEP/SimplePartWhole/. Acesso em 30 mar. 2008.

RULEML. 2008. “The Rule Markup Initiative”. Disponível em: http://www.ruleml.org/. Acesso em 30 mar. 2008.

RUMBAUGH, J. et al., *Modelagem e Projetos Baseados em Objetos*, Rio de Janeiro: Campus, 1994.

RUSSELL, S. J.; NORVIG, P., *Artificial intelligence : a modern approach*. New Jersey: Prentice-Hall, Inc. 1995.

SBML. 2008. "Systems Biology Markup Language". Disponível em: http://sbml.org/Main_Page. Acesso em 30 mar. 2008.

SCHILSTRA, M. J. "SBML and CellML: XML-based markup languages for models of biochemical systems". Presentation at *3th International Symposium on Networks in Bioinformatics*, University of Amsterdam, Amsterdam, The Netherlands, 2005.

SETZER, W. "Data, information, knowledge and competence". *3rd CONTECSI (International Conference on Information Systems)*, June 2006. Disponível em: http://www.ime.usp.br/~vwsetzer/data-info.html. Acesso em 30 mar. 2008.

SHIMAYOSHI, T. et al. "A Method to Support Cell Physiological Modelling Using Description Language and Ontology". *IPSJ Digital Courier*, 2:726-735, 2006.

SI. 2006. "The International System of Units (SI)", 8th edition. Disponível em: http://www.bipm.org/en/si/. Acesso em 30 mar. 2008.

SINGH, M. P., HUHNS, M. N., *Service-Oriented Computing – Semantics, Process, Agents*. John Willey & Sons Ltd. 2005.

SIRIN, E. et al. "Pellet: A practical OWL-DL reasoner**"**, *Journal of Web Semantics*, 5(2), 2007.

SPARQL. 2008. "SPARQL Query Language for RDF. W3C Recommendation 15 January 2008". Disponível em: http://www.w3.org/TR/rdf-sparql-query. Acesso em 30 mar. 2008.

SPERBERG-MCQUEEN, C.M.; THOMPSON, H. 2005. "XML Schema". Disponível em http://www.w3.org/XML/Schema. Acesso em 30 mar. 2008.

SUN, Z., FINKELSTEIN, A., ASHMORE, J. "Using Ontology with Semantic Web Services to Support Modeling in Systems Biology". *International Workshop on Approaches and Architectures for Web Data Integration and Mining in Life Sciences (WebDIM4LS)*, Nancy, France, 2007.

WOLKENHAUER, O., KITANO, H., CHO, K.-H. "Systems biology"**.** *Control Systems Magazine*, IEEE Volume 23, Issue 4, Aug. 2003 Page(s):38 – 48. Digital Object Identifier 10.1109/MCS.2003.1213602.

XML. 2006. "Extensible markup language (XML) 1.0 (Fourth edition). W3C recommendation. 2006". Disponível em http://www.w3.org/TR/REC-xml/. Acesso em 30 mar. 2008.

XSLT. 1999. “XSL Transformations (XSLT). Version 1.0. W3C Recommendation 16 November 1999”. Disponível em: http://www.w3.org/TR/xslt. Acesso em 30 mar. 2008.

ZHANG et al. “MetaCyc and AraCyc. Metabolic Pathway Databases for Plant Research”. *Plant Physiology*, 138: 27-37, 2005.

Apêndice 1. Ontologia CelO

Este apêndice apresenta, para referência, uma descrição sumária das classes da ontologia CelO e o código completo da ontologia OWL, usando a notação Turtle.

A1.1. Descrição das classes

A1.1.1. SIEntity

```
SIEntity: Classes associadas ao Sistema Internacional de Unidades (SI).
   Quantity: Grandeza medida por uma unidade do SI.
      BaseQuantity: Grandeza básica definida pelo SI.
      DerivedQuantity: Grandezas derivadas das unidades básicas do SI.
      UserDefinedQuantity: Grandeza derivada definida pelo usuário.
   Unit: Unidades usadas para medida das grandezas.
      StandardUnit: Unidades padrão definidas no SI.
         SIBaseUnit: Unidades básicas definidas no SI.
         SIDerivedUnit: Unidades derivadas das unidades básicas.
      DefinedUnit: Unidades pré-definidas, não constantes do SI.
      UserDefinedUnit: Unidades definidas pelos usuários usadas não modelos biológicos.
```

A1.1.2. DomainEntity

```
DomainEntity: Classes associadas ao domínio da Biologia.
   BiologicalEntity: Entidades do domínio da Biologia.
      CellElement: Elementos da célula.
         CellStructure: Estrutura da célula.
            CellPart: Partes constituintes da célula.
            CellSpace: Localização espacial dos elementos da célula.
            CellProcess: Processos celulares de interesse em modelos.
            GatingProcess: processo de gating em canais iônicos.
            MembraneTransport: processos de transporte através da membrana.
            PolarizationProcess: processo de polarização/depolariza-ção da membrana.
      CellType: Tipos de células.
         Myocyte: Células dos músculos.
            CardicMyocyte: Células do músculo cardíaco.
   ChemicalEntity: Entidades do domínio da Química.
      ChemicalElement: Elementos químicos.
      ChemicalCompound: Compostos químicos.
      ChemicalObject: Objetos genéricos (íons, átomos, moléculas).
   BioChemicalEntity: Entidades do domínio da Bioquímica.
```

A1.1.3. ModelEntity

```
ModelEntity: Classes usadas no processo de modelagem.
   ModelService: Classe usada para caracterizar um modelo como um serviço web.
      ModelProfile: Informações que caracterizam o modelo.
      ModelProcess: Informações necessárias para execução do modelo.
         ModelInterface: Interface (lista de parâmetros do modelo).
         ModelParameter: Parâmetros do modelo.
            ModelParameterIn: Parâmetros de entrada (associados às variáveis do modelo).
            ModelParameterOut: Parâmetros de saída (associados às variáveis do modelo).
      ModelGrounding: Informações sobre o modelo computacional (executável).
         CellMLModel: Informações sobre o modelo CellML.
   ModelType: Classes usadas para caracterização do modelo
      MathematicalModelType: Tipo de modelo matemático.
         MathModelDiscretizationType
            ContinuousType: modelos contínuos.
            DiscreteType: modelos discretos.
         MathModelLinearityType:
            LinearModel: modelos baseados em equações lineares.
            NonLinearModel: modelos baseados em equações não-lineares.
         MathModelDeterministicType:
            DeterministicModel: modelos determinísticos.
```

```
         StochasticModel: modelos estocásticos.
      MathModelTimeDependencyType:
         StationaryMode: modelos estacionários (independents do tempo).
         TimeDependentModel: modelos dependentes do tempo.
   BiologicalModelType: Tipo de modelo biológico.
      PhysiologicalModel: Modelos em Fisiologia.
         Electrophysiological: Modelos em Eletrofisiologia.
ModelObject: Objetos genéricos dos modelos.
   Model: Identificação do modelo.
   Equation: Lista de equações do modelo.
   Variable: Variáveis do modelo.
      ModelVariable: Variáveis do modelo.
         AlgebricVariable: variáveis algébricas.
         ParameterVariable: parâmetros das equações.
         DependentVariable: variáveis dependentes.
         IndependentVariable: variáveis independentes.
      ComponentVariable: Variáveis de cada componente.
         InterfaceVariable: variáveis de interface (parâmetros) do componente.
         LocalVariable: variáveis usadas internamente no componente.
      DomainVariable: classificação das variáveis pelo mecanismo de inferência.
         ActionPotential: potencial de ação.
         SodiumIonChannel: canal iônico de sódio.
         Component: Componentes do modelo.
         InternalComponent: Componentes definidos internamente ao modelo.
      ExternalComponent: Componentes definidos externamente.
```

A1.2. Código da ontologia

As classes, propriedades e indivíduos da ontologia CelO são descritas nesta seção. Duas propriedades importantes, comuns a várias classes e que não são descritas separadamente são:

- *hasName* String: cada indivíduo em um modelo descrito com a ontologia possui um nome, registrado nesta propriedade. O principal objetivo é permitir que elementos diferentes, em especial variáveis, possam ter o mesmo nome, usado no processo de geração de um modelo CellML. Em OWL, cada elemento deve possuir um nome único. No caso de variáveis isto é conseguido através da concatenação do nome da variável com o nome do componente. A variável *time*, por exemplo, pode ser usada tanto em um componente *membrane* quanto em um componente *sodium_channel*. Na ontologia as variáveis serão chamadas respectivamente de *membrane_time* e *sodium_channel_time*, ambas com a propriedade *hasNam*e igual a *time*.
- *hasDetail* String: esta propriedade não tem valor semântico. Seu objetivo é armazenar as informações obtidas durante o processo de conversão de um modelo CellML a fim de, com a aplicação das regras, preencher as propriedades das variáveis e componentes com indivíduos da ontologia.

A1.2.1. Classes

```
celo:Nucleotide
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ChemicalCompound .

celo:VentricularCell
      rdf:type celo:CardiacMyocyte .

celo:BiologicalEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainEntity .

celo:ModelType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:LinearModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelLinearityType ;
      owl:disjointWith celo:NonLinearModel .

celo:MembraneTransport
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellProcess .

celo:CellSpace
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellStructure .

celo:ChemicalCompound
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ChemicalEntity .

celo:DependentVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelVariable .

celo:PolarizationProcess
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellProcess .

celo:InterfaceVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ComponentVariable .

celo:MarkovModel
      rdf:type celo:ElectrophysiologicalModel .

celo:ExternalComponent
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Component .

celo:InternalComponent
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Component .

celo:Component
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject .

celo:IonChannelGate
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainVariable ;
      owl:equivalentClass
              [ rdf:type owl:Restriction ;
                owl:hasValue celo:Gate ;
                owl:onProperty celo:hasDomainEntity
              ] .

celo:ModelParameterOut
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelParameter .
```

```
celo:ModelObject
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:Protein
      rdf:type owl:Class ;
      rdfs:subClassOf celo:BioChemicalEntity .

celo:ChemicalEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainEntity .


celo:ModelVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Variable .

celo:MathematicalModelType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelType .

celo:ParameterVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelVariable .

celo:SIEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Celo .

celo:Unit
      rdf:type owl:Class ;
      rdfs:subClassOf celo:SIEntity .

celo:Compartment
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject ;
      rdfs:subClassOf
              [ rdf:type owl:Restriction ;
                owl:allValuesFrom celo:Compartment ;
                owl:onProperty celo:modelCompartment
              ] .

celo:Model
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject .

celo:PhysiologicalModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:BiologicalModelType .

celo:ModelService
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:ModelProfile
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:DefinedUnit
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Unit ;
      owl:disjointWith celo:SIDerivedUnit , celo:SIBaseUnit .

celo:PotassiumIonChannel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainVariable ;
      owl:equivalentClass
              [ rdf:type owl:Class ;
                owl:intersectionOf ([ rdf:type owl:Restriction ;
                            owl:hasValue celo:Potassium ;
                            owl:onProperty celo:hasDomainEntity
                          ] [ rdf:type owl:Restriction ;
                            owl:hasValue celo:Channel ;
```

```
                          owl:onProperty celo:hasDomainEntity
                        ] [ rdf:type owl:Restriction ;
                          owl:hasValue celo:EletricCurrent ;
                          owl:onProperty celo:hasMeasure
                        ])
              ] .

celo:ContinuousModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelDiscretizationType ;
      owl:disjointWith celo:DiscreteModel .

celo:LocalVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ComponentVariable .

celo:SIBaseUnit
      rdf:type owl:Class ;
      rdfs:subClassOf celo:StandardUnit ;
      owl:disjointWith celo:DefinedUnit , celo:SIDerivedUnit .

celo:Quantity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:SIEntity .

celo:CellElement
      rdf:type owl:Class ;
      rdfs:subClassOf celo:BiologicalEntity ;
      rdfs:subClassOf
              [ rdf:type owl:Restriction ;
                owl:allValuesFrom celo:CellElement ;
                owl:onProperty celo:contains
              ] ;
      rdfs:subClassOf
              [ rdf:type owl:Restriction ;
                owl:allValuesFrom celo:CellElement ;
                owl:onProperty celo:isContainedIn
              ] .

celo:BiologicalModelType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelType .

celo:SIDerivedUnit
      rdf:type owl:Class ;
      rdfs:subClassOf celo:StandardUnit ;
      owl:disjointWith celo:DefinedUnit , celo:SIBaseUnit .

celo:UserDefinedUnit
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Unit ;
      owl:disjointWith celo:StandardUnit .

celo:ModelInterface
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelProcess .

celo:ModelParameter
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelProcess .

celo:MathModelDiscretizationType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathematicalModelType .

celo:GatingProcess
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellProcess .

celo:TimeDependentModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelTimeDependencyType ;
      owl:disjointWith celo:StationaryModel .
```

```
celo:CellType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:BiologicalEntity .

celo:ChemicalObject
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ChemicalEntity .

celo:Element
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject .

celo:Celo
      rdf:type owl:Class .


celo:CellProcess
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellElement .

celo:StochasticModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelDeterministicType ;
      owl:disjointWith celo:DeterministicModel .

celo:MathModelTimeDependencyType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathematicalModelType .

celo:SodiumIonChannel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainVariable ;
      owl:equivalentClass
              [ rdf:type owl:Class ;
                owl:intersectionOf ([ rdf:type owl:Restriction ;
                            owl:hasValue celo:Sodium ;
                            owl:onProperty celo:hasDomainEntity
                          ] [ rdf:type owl:Restriction ;
                            owl:hasValue celo:Channel ;
                            owl:onProperty celo:hasDomainEntity
                          ] [ rdf:type owl:Restriction ;
                            owl:hasValue celo:EletricCurrent ;
                            owl:onProperty celo:hasMeasure
                          ])
              ] .

celo:Myocyte
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellType .

celo:CardiacMyocyte
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Myocyte .

celo:ModelEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Celo .

celo:DomainEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Celo .

celo:MathModelDeterministicType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathematicalModelType .

celo:ModelGrounding
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:DeterministicModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelDeterministicType ;
      owl:disjointWith celo:StochasticModel .
```

```
celo:IndependentVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelVariable .

celo:CellPart
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellStructure .

celo:ChemicalElement
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ChemicalEntity .

celo:DiscreteModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelDiscretizationType ;
      owl:disjointWith celo:ContinuousModel .

celo:AlgebricVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelVariable .

celo:CellMLModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelGrounding .

celo:ElectrophysiologicalModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:PhysiologicalModel .

celo:ComponentVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Variable .

celo:Equation
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject .


celo:BioChemicalEntity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainEntity .

celo:ModelParameterIn
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelParameter .

celo:Variable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelObject ;
      rdfs:subClassOf
              [ rdf:type owl:Restriction ;
                owl:allValuesFrom celo:Unit ;
                owl:onProperty celo:hasVariableUnit
              ] .

celo:DomainVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Variable .

celo:ModelProcess
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelEntity .

celo:CellStructure
      rdf:type owl:Class ;
      rdfs:subClassOf celo:CellElement .

celo:StationaryModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelTimeDependencyType ;
      owl:disjointWith celo:TimeDependentModel .

celo:StandardUnit
```

```
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Unit ;
      owl:disjointWith celo:UserDefinedUnit .

celo:UserDefinedQuantity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Quantity ;
      owl:disjointWith celo:DerivedQuantity , celo:BaseQuantity .

celo:BaseQuantity
      rdf:type owl:Class ;
      rdfs:subClassOf celo:Quantity ;
      owl:disjointWith celo:DerivedQuantity , celo:UserDefinedQuantity .

celo:MathModelLinearityType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathematicalModelType .

celo:ActionPotential
      rdf:type owl:Class ;
      rdfs:subClassOf celo:DomainVariable ;
      owl:equivalentClass
              [ rdf:type owl:Class ;
                owl:intersectionOf ([ rdf:type owl:Restriction ;
                            owl:hasValue celo:Membrane ;
                            owl:onProperty celo:hasDomainEntity
                          ] [ rdf:type owl:Restriction ;
                            owl:hasValue celo:Voltage ;
                            owl:onProperty celo:hasMeasure
                          ])
              ] .

celo:NonLinearModel
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathModelLinearityType ;
      owl:disjointWith celo:LinearModel .
```

A1.2.2. Propriedades

```
celo:isMeasuredBy
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:Quantity ;
      rdfs:range celo:Unit ;
      owl:inverseOf celo:measures .

celo:hasChemicalSymbol
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:ChemicalElement ;
      rdfs:range xsd:string .

celo:hasInitialValue
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Variable ;
      rdfs:range xsd:string .

celo:composedOf
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelProfile ;
      rdfs:range celo:Component .

celo:hasPrivateInterface
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:InterfaceVariable ;
      rdfs:range xsd:string .

celo:hasMeasure
      rdf:type owl:ObjectProperty ;
      rdfs:domain
              [ rdf:type owl:Class ;
                owl:unionOf (celo:Variable celo:CellProcess)
              ] ;
      rdfs:range celo:Quantity .
```

```
celo:hasMath
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Equation ;
      rdfs:range xsd:string .

celo:connectsTo
      rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
      rdfs:domain celo:Component ;
      rdfs:range celo:Component ;
      owl:inverseOf celo:connectsTo .

celo:refersTo
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelService ;
      rdfs:range celo:ModelGrounding .

celo:describedBy
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelService ;
      rdfs:range celo:ModelInterface .

celo:hasComponentURI
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:ExternalComponent ;
      rdfs:range xsd:string .

celo:hasURI
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:CellMLModel ;
      rdfs:range xsd:string .

celo:hasBiologicalEntity
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelProfile ;
      rdfs:range celo:BiologicalEntity .

celo:hasName
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Celo ;
      rdfs:range xsd:string .

celo:hasInterface
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelInterface ;
      rdfs:range celo:ModelParameter .

celo:hasPublicInterface
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:InterfaceVariable ;
      rdfs:range xsd:string .

celo:presents
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelService ;
      rdfs:range celo:ModelProfile .

celo:hasSymbol
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Unit ;
      rdfs:range xsd:string .

celo:hasVariableUnit
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:Variable ;
      rdfs:range celo:Unit .

celo:contains
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:CellStructure ;
      rdfs:range celo:CellStructure ;
      owl:inverseOf celo:isContainedIn .

celo:hasDescription
```

```
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Celo ;
      rdfs:range xsd:string .

celo:hasDomainEntity
      rdf:type owl:ObjectProperty ;
      rdfs:domain
              [ rdf:type owl:Class ;
                owl:unionOf (celo:Variable celo:CellProcess celo:Component)
              ] ;
      rdfs:range celo:DomainEntity .

celo:isInterfaceVariableOf
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:InterfaceVariable ;
      rdfs:range celo:InternalComponent ;
      owl:inverseOf celo:hasInterfaceVariable .

celo:hasDetail
      rdf:type owl:DatatypeProperty ;
      rdfs:domain
              [ rdf:type owl:Class ;
                owl:unionOf (celo:InternalComponent celo:Variable)
              ] ;
      rdfs:range xsd:string .

celo:hasLocalVariable
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:InternalComponent ;
      rdfs:range celo:LocalVariable ;
      rdfs:subPropertyOf celo:hasVariable .

celo:measures
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:Unit ;
      rdfs:range celo:Quantity ;
      owl:inverseOf celo:isMeasuredBy .

celo:hasCellElement
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:Component ;
      rdfs:range celo:CellElement .

celo:hasInterfaceVariable
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:InternalComponent ;
      rdfs:range celo:InterfaceVariable ;
      rdfs:subPropertyOf celo:hasVariable ;
      owl:inverseOf celo:isInterfaceVariableOf .

celo:hasVariable
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:InternalComponent ;
      rdfs:range celo:Variable .

celo:isContainedIn
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:CellStructure ;
      rdfs:range celo:CellStructure ;
      owl:inverseOf celo:contains .

celo:encapsules
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:Component ;
      rdfs:range celo:Component .
```

A1.2.3. Indivíduos

```
celo:VentricularCell
      rdf:type celo:CardiacMyocyte .

celo:microA_per_cm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "microA_per_cm2"^^xsd:string ;
      celo:measures celo:EletricCurrentDensity .

celo:microF
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "microF"^^xsd:string ;
      celo:measures celo:Capacitance .

celo:Voltage
      rdf:type celo:DerivedQuantity ;
      celo:hasName "potencial"^^xsd:string , "voltage"^^xsd:string , "E"^^xsd:string ;
      celo:isMeasuredBy
              celo:mV , celo:volt .

celo:ElectricConductancePerCapacitance
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:nanoS_per_picoF .

celo:meter
      rdf:type celo:SIBaseUnit ;
      celo:hasName "meter"^^xsd:string ;
      celo:hasSymbol "m"^^xsd:string ;
      celo:measures celo:Length .

celo:Time
      rdf:type celo:BaseQuantity ;
      celo:hasName "time"^^xsd:string ;
      celo:isMeasuredBy
              celo:second , celo:millisecond , celo:ms .

celo:hertz
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "hertz"^^xsd:string ;
      celo:hasSymbol "Hz"^^xsd:string ;
      celo:measures celo:Frequency .

celo:mm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "mm2"^^xsd:string ;
      celo:measures celo:Area .

celo:volt
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "volt"^^xsd:string ;
      celo:hasSymbol "V"^^xsd:string ;
      celo:measures celo:Voltage .

celo:His-PurkinjeFibre
      rdf:type celo:CardiacMyocyte .

celo:squaremeter
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "squaremeter"^^xsd:string ;
      celo:hasSymbol "m2"^^xsd:string ;
      celo:measures celo:Area .

celo:weber
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "weber"^^xsd:string ;
      celo:hasSymbol "Wb"^^xsd:string ;
      celo:measures celo:MagneticFlux .

celo:SarcoplasmicReticulum
      rdf:type celo:CellPart ;
      celo:hasName "sarcoplasmicreticulum"^^xsd:string , "SR"^^xsd:string ;
```

```
      celo:isContainedIn
              celo:Myoplasm .

celo:per_mm
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_mm"^^xsd:string ;
      celo:measures celo:PerLength .

celo:MarkovModel
      rdf:type celo:ElectrophysiologicalModel .

celo:Sodium
      rdf:type celo:ChemicalElement ;
      celo:hasChemicalSymbol
              "Na"^^xsd:string ;
      celo:hasName "sodium"^^xsd:string .

celo:microF_per_cm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "microF_per_cm2"^^xsd:string ;
      celo:measures celo:CapacitancePerArea .

celo:LuminousIntensity
      rdf:type celo:BaseQuantity ;
      celo:hasName "luminous"^^xsd:string ;
      celo:isMeasuredBy
              celo:candela .

celo:EletricCurrentDensity
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:microA_per_cm2 , celo:uA_per_mm2 .

celo:farad
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "farad"^^xsd:string ;
      celo:hasSymbol "F"^^xsd:string ;
      celo:measures celo:Capacitance .

celo:Substance
      rdf:type celo:BaseQuantity ;
      celo:hasName "substance"^^xsd:string ;
      celo:isMeasuredBy
              celo:mole .

celo:Potassium
      rdf:type celo:ChemicalElement ;
      celo:hasChemicalSymbol
              "K"^^xsd:string ;
      celo:hasName "potassium"^^xsd:string .

celo:millimolar
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "millimolar"^^xsd:string ;
      celo:measures celo:Concentration .

celo:Volume
      rdf:type celo:DerivedQuantity ;
      celo:hasName "volume"^^xsd:string ;
      celo:isMeasuredBy
              celo:litre , celo:micrometre3 , celo:cubicmeter .

celo:ElectricCurrentPerConcentration
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:nanoA_per_millimolar .

celo:cm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "cm2"^^xsd:string ;
      celo:measures celo:Area .

celo:Pump
      rdf:type celo:CellPart ;
```

```
      celo:hasName "pump"^^xsd:string ;
      celo:isContainedIn
              celo:Membrane .

celo:ADimensional
      rdf:type celo:UserDefinedQuantity ;
      celo:hasName ""^^xsd:string ;
      celo:isMeasuredBy
              celo:dimensionless .

celo:ElectricConductance
      rdf:type celo:DerivedQuantity .

celo:NetworkModel
      rdf:type celo:ElectrophysiologicalModel .

celo:per_concentration_units
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_concentration_units"^^xsd:string ;
      celo:measures celo:PerConcentration .

celo:joule_per_mole_kelvin
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "joule_per_mole_kelvin"^^xsd:string ;
      celo:measures celo:MolarEntropy .

celo:joule_per_kilomole_kelvin
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "joule_per_kilomole_kelvin"^^xsd:string ;
      celo:measures celo:MolarEntropy .

celo:per_mV
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_mV"^^xsd:string ;
      celo:measures celo:PerVoltage .

celo:Condutivity
      rdf:type celo:UserDefinedQuantity .

celo:ExtraCelllularSpace
      rdf:type celo:CellSpace ;
      celo:contains celo:Membrane .

celo:EletricCurrent
      rdf:type celo:BaseQuantity ;
      celo:hasName "current"^^xsd:string , "i"^^xsd:string ,
"eletriccurrent"^^xsd:string ;
      celo:isMeasuredBy
              celo:picoA , celo:ampere , celo:nanoA .

celo:Calcium
      rdf:type celo:ChemicalElement ;
      celo:hasChemicalSymbol
              "Ca"^^xsd:string ;
      celo:hasName "calcium"^^xsd:string .

celo:ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "ms"^^xsd:string ;
      celo:hasSymbol "ms"^^xsd:string ;
      celo:measures celo:Time .

celo:AtrioVentricularNode
      rdf:type celo:CardiacMyocyte .

celo:PerVoltage
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:per_millivolt , celo:per_mV .

celo:Myoplasm
      rdf:type celo:CellPart ;
      celo:contains celo:Buffer , celo:SarcoplasmicReticulum ;
      celo:hasName "myoplasm"^^xsd:string ;
```

```
      celo:isContainedIn
              celo:IntraCellularSpace .

celo:millisecond
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "millisecond"^^xsd:string ;
      celo:measures celo:Time .

celo:Mitochondria
      rdf:type celo:CellPart ;
      celo:hasName "mitochodria"^^xsd:string ;
      celo:isContainedIn
              celo:IntraCellularSpace .

celo:nanoS_per_picoF
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "nanoS_per_picoF"^^xsd:string ;
      celo:measures celo:ElectricConductancePerCapacitance .

celo:AtrialCell
      rdf:type celo:CardiacMyocyte .

celo:mS_per_mm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "ms_per_mm2"^^xsd:string ;
      celo:measures celo:ElectricConductancePerArea .

celo:PerLength
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:per_mm .

celo:per_mV_ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_mV_ms"^^xsd:string ;
      celo:measures celo:PerVoltageTime .

celo:Gate
      rdf:type celo:CellPart ;
      celo:hasName "gate"^^xsd:string ;
      celo:isContainedIn
              celo:Channel .

celo:nanoA
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "nanoA"^^xsd:string ;
      celo:measures celo:EletricCurrent .

celo:Velocity
      rdf:type celo:DerivedQuantity ;
      celo:hasName "velocity"^^xsd:string ;
      celo:isMeasuredBy
              celo:mm_per_ms .

celo:ElectricCurrentPerCapacitance
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:picoA_per_picoF .

celo:EletricResistence
      rdf:type celo:DerivedQuantity ;
      celo:hasName "eletricresistence"^^xsd:string .

celo:kelvin
      rdf:type celo:SIBaseUnit ;
      celo:hasName "kelvin"^^xsd:string ;
      celo:hasSymbol "K"^^xsd:string ;
      celo:measures celo:Temperature .

celo:Mass
      rdf:type celo:BaseQuantity ;
      celo:hasName "mass"^^xsd:string ;
      celo:isMeasuredBy
              celo:kilogram .
```

```
celo:mV
      rdf:type celo:UserDefinedUnit ;
      celo:measures celo:Voltage .

celo:hasName
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:Celo ;
      rdfs:range xsd:string .

celo:ElectricConductancePerArea
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:mS_per_mm2 , celo:milliS_per_cm2 .

celo:mV_per_ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "mV_per_ms"^^xsd:string ;
      celo:measures celo:VoltagePerTime .

celo:Area
      rdf:type celo:DerivedQuantity ;
      celo:hasName "area"^^xsd:string ;
      celo:isMeasuredBy
              celo:cm2 , celo:mm2 , celo:squaremeter .

celo:ConcentrationPerTime
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:millimolar_per_millisecond , celo:mM_per_ms .

celo:Nucleus
      rdf:type celo:CellPart ;
      celo:hasName "nucleus"^^xsd:string ;
      celo:isContainedIn
              celo:IntraCellularSpace .

celo:faradays_constant_units
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "faradays_constant_units"^^xsd:string ;
      celo:measures celo:ElectricChargePerAmount .

celo:CapacitancePerArea
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:uF_per_mm2 , celo:microF_per_cm2 .

celo:ElectricChargePerAmount
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:coulomb_per_mole , celo:coulomb_per_millimole ,
celo:faradays_constant_units .

celo:picoA
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "picoA"^^xsd:string ;
      celo:measures celo:EletricCurrent .

celo:uF_per_mm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "uF_per_mm2"^^xsd:string ;
      celo:measures celo:CapacitancePerArea .

celo:per_millivolt
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_millivolt"^^xsd:string ;
      celo:measures celo:PerVoltage .

celo:second
      rdf:type celo:SIBaseUnit ;
      celo:hasName "second"^^xsd:string ;
      celo:hasSymbol "s"^^xsd:string ;
      celo:measures celo:Time .
```

```
celo:milliS_per_cm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "milliS_per_cm2"^^xsd:string ;
      celo:measures celo:ElectricConductancePerArea .

celo:Chlorid
      rdf:type celo:ChemicalElement ;
      celo:hasChemicalSymbol
              "Cl"^^xsd:string ;
      celo:hasName "chlorid"^^xsd:string .

celo:hasPublicInterface
      rdf:type owl:DatatypeProperty ;
      rdfs:domain celo:InterfaceVariable ;
      rdfs:range xsd:string .

celo:mm_per_ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "mm_per_ms"^^xsd:string ;
      celo:measures celo:Velocity .

celo:ElectricChargePerConcentration
      rdf:type celo:UserDefinedQuantity .

celo:Acceleration
      rdf:type celo:DerivedQuantity ;
      celo:hasName "acceleration"^^xsd:string .

celo:DiadicSpace
      rdf:type celo:CellSpace ;
      celo:isContainedIn
              celo:IntraCellularSpace .

celo:Atom
      rdf:type celo:ChemicalObject .

celo:Eletriclnductance
      rdf:type celo:DerivedQuantity ;
      celo:hasName "eletricelonductance"^^xsd:string ;
      celo:isMeasuredBy
              celo:siemens .

celo:Capacitance
      rdf:type celo:DerivedQuantity ;
      celo:hasName "capacitance"^^xsd:string ;
      celo:isMeasuredBy
              celo:farad , celo:microF .

celo:PerTime
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:per_millisecond .

celo:HodgkinHuxleyModel
      rdf:type celo:ElectrophysiologicalModel .

celo:cubicmeter
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "cubicmeter"^^xsd:string ;
      celo:hasSymbol "m3"^^xsd:string ;
      celo:measures celo:Volume .

celo:uA_per_mm2
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "uA_per_mm2"^^xsd:string ;
      celo:measures celo:EletricCurrentDensity .

celo:gas_constant_units
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "gas_constant_units"^^xsd:string ;
      celo:measures celo:MolarEntropy .

celo:metre
      rdf:type celo:SIBaseUnit ;
```

```
      celo:hasName "metre"^^xsd:string ;
      celo:hasSymbol "m"^^xsd:string ;
      celo:measures celo:Length .

celo:Channel
      rdf:type celo:CellPart ;
      celo:contains celo:Gate ;
      celo:hasName "ionchannel"^^xsd:string , "channel"^^xsd:string ;
      celo:isContainedIn
              celo:Membrane .

celo:Buffer
      rdf:type celo:CellPart ;
      celo:hasName "buffer"^^xsd:string ;
      celo:isContainedIn
              celo:Myoplasm .

celo:MathModelDeterministicType
      rdf:type owl:Class ;
      rdfs:subClassOf celo:MathematicalModelType .

celo:Frequency
      rdf:type celo:DerivedQuantity ;
      celo:hasName "frequency"^^xsd:string ;
      celo:isMeasuredBy
              celo:hertz .

celo:picoA_per_picoF
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "picoA_per_picoF"^^xsd:string ;
      celo:measures celo:ElectricCurrentPerCapacitance .

celo:per_ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_ms"^^xsd:string ;
      celo:hasSymbol "/ms"^^xsd:string ;
      celo:measures celo:PerTime .

celo:mM_per_ms
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "mM_per_ms"^^xsd:string ;
      celo:measures celo:ConcentrationPerTime .

celo:VoltagePerTime
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:mV_per_ms .

celo:millivolt
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "millivolt"^^xsd:string ;
      celo:hasSymbol "mV"^^xsd:string ;
      celo:measures celo:Voltage .

celo:PerConcentration
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:per_concentration_units .

celo:PerVoltageTime
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:per_mV_ms , celo:per_millivolt_millisecond .

celo:coulomb_per_mole
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "coulomb_per_mole"^^xsd:string ;
      celo:measures celo:ElectricChargePerAmount .

celo:Ion
      rdf:type celo:ChemicalObject ;
      celo:hasName "ion"^^xsd:string .

celo:candela
```

```
      rdf:type celo:SIBaseUnit ;
      celo:hasName "candela"^^xsd:string ;
      celo:hasSymbol "cd"^^xsd:string ;
      celo:measures celo:LuminousIntensity .

celo:concentration_units
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "concentration_units"^^xsd:string ;
      celo:measures celo:Concentration .

celo:SinoAtrialNode
      rdf:type celo:CardiacMyocyte .

celo:millimolar_per_millisecond
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "millimolar_per_millisecond"^^xsd:string ;
      celo:measures celo:ConcentrationPerTime .

celo:Molecule
      rdf:type celo:ChemicalObject ;
      celo:hasName "molecule"^^xsd:string .

celo:AlgebricVariable
      rdf:type owl:Class ;
      rdfs:subClassOf celo:ModelVariable .

celo:Exchanger
      rdf:type celo:CellPart ;
      celo:hasName "exchanger"^^xsd:string ;
      celo:isContainedIn
              celo:Membrane .

celo:Density
      rdf:type celo:DerivedQuantity ;
      celo:hasName "density"^^xsd:string .

celo:Membrane
      rdf:type celo:CellPart ;
      celo:contains celo:Pump , celo:Exchanger , celo:Channel ;
      celo:hasName "membrane"^^xsd:string ;
      celo:isContainedIn
              celo:ExtraCelllularSpace .

celo:ampere
      rdf:type celo:SIBaseUnit ;
      celo:hasName "ampere"^^xsd:string ;
      celo:hasSymbol "A"^^xsd:string ;
      celo:measures celo:EletricCurrent .

celo:kilogram
      rdf:type celo:SIBaseUnit ;
      celo:hasName "kilogram"^^xsd:string ;
      celo:hasSymbol "kg"^^xsd:string ;
      celo:measures celo:Mass .

celo:per_millivolt_millisecond
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_millivolt_millisecond"^^xsd:string ;
      celo:measures celo:PerVoltageTime .

celo:SynapticModel
      rdf:type celo:ElectrophysiologicalModel .

celo:Concentration
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:mM , celo:concentration_units , celo:millimolar .

celo:micrometre3
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "micrometre3"^^xsd:string ;
      celo:measures celo:Volume .

celo:per_millisecond
```

```
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_millisecond"^^xsd:string ;
      celo:measures celo:PerTime .

celo:dimensionless
      rdf:type celo:DefinedUnit ;
      celo:hasName "dimensionless"^^xsd:string ;
      celo:measures celo:ADimensional .

celo:mole
      rdf:type celo:SIBaseUnit ;
      celo:hasName "mole"^^xsd:string ;
      celo:hasSymbol "mol"^^xsd:string ;
      celo:measures celo:Substance .

celo:ATP
      rdf:type celo:Nucleotide .

celo:mM
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "mM"^^xsd:string ;
      celo:measures celo:Concentration .

celo:micrometre
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "micrometre"^^xsd:string ;
      celo:measures celo:Length .

celo:Inductance
      rdf:type celo:DerivedQuantity ;
      celo:hasName "inductance"^^xsd:string .

celo:Length
      rdf:type celo:BaseQuantity ;
      celo:hasName "length"^^xsd:string , "L"^^xsd:string ;
      celo:isMeasuredBy
              celo:meter , celo:metre , celo:micrometre .

celo:litre
      rdf:type celo:SIDerivedUnit ;
      celo:hasName ""^^xsd:string ;
      celo:hasSymbol ""^^xsd:string ;
      celo:measures celo:Volume .

celo:MagneticFlux
      rdf:type celo:DerivedQuantity ;
      celo:hasName "magneticflux"^^xsd:string ;
      celo:isMeasuredBy
              celo:weber .

[]    rdf:type owl:AllDifferent .

celo:MolarEntropy
      rdf:type celo:UserDefinedQuantity ;
      celo:isMeasuredBy
              celo:joule_per_mole_kelvin , celo:joule_per_kilomole_kelvin ,
celo:gas_constant_units .

celo:IntraCellularSpace
      rdf:type celo:CellSpace ;
      celo:contains celo:Nucleus , celo:Myoplasm , celo:Mitochondria , celo:DiadicSpace
.

celo:modelCompartment
      rdf:type owl:ObjectProperty ;
      rdfs:domain celo:ModelProfile ;
      rdfs:range celo:Compartment .

celo:Temperature
      rdf:type celo:BaseQuantity ;
      celo:hasName "temperature"^^xsd:string ;
      celo:isMeasuredBy
              celo:kelvin .
```

```
celo:siemens
      rdf:type celo:SIDerivedUnit ;
      celo:hasName "siemens"^^xsd:string ;
      celo:hasSymbol "S"^^xsd:string ;
      celo:measures celo:Eletriclnductance .

celo:nanoA_per_millimolar
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "nanoA_per_millimolar"^^xsd:string ;
      celo:measures celo:ElectricCurrentPerConcentration .

celo:MulticompartmentModel
      rdf:type celo:ElectrophysiologicalModel .

celo:coulomb_per_millimole
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "coulomb_per_millimole"^^xsd:string ;
      celo:measures celo:ElectricChargePerAmount .
```

Apêndice 2. Arquivos utilizados no estudo de caso

Este apêndice contém os principais arquivos que foram utilizados para testes da ontologia CelO e do framework CelOWS.

A2.1. Modelo CellML

A seguir é apresentado o código original do model CellML “A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials” (CELLML, 2008b).

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This CellML file was generated on 21/08/2007 at 17:38:21 using:

COR (0.9.31.751)
Copyright 2002-2007 Dr Alan Garny
http://COR.physiol.ox.ac.uk/ - COR@physiol.ox.ac.uk

CellML 1.0 was used to generate this cellular model
http://www.CellML.org/
-->
<model xmlns="http://www.cellml.org/cellml/1.0#"
xmlns:cmeta="http://www.cellml.org/metadata/1.0#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:bqs="http://www.cellml.org/bqs/1.0#"
xmlns:cellml="http://www.cellml.org/cellml/1.0#"
xmlns:dcterms="http://purl.org/dc/terms/" xmlns:vCard="http://www.w3.org/2001/vcard-
rdf/3.0#" cmeta:id="noble_1962_version04" name="noble_1962_version04">

   <units name="per_second">
      <unit units="second" exponent="-1"/>
   </units>
   <units name="millivolt">
      <unit units="volt" prefix="milli"/>
   </units>
   <units name="per_millivolt">
      <unit units="volt" prefix="milli" exponent="-1"/>
   </units>
   <units name="per_millivolt_second">
      <unit units="millivolt" exponent="-1"/>
      <unit units="second" exponent="-1"/>
   </units>
   <units name="microS">
      <unit units="siemens" prefix="micro"/>
   </units>
   <units name="microF">
      <unit units="farad" prefix="micro"/>
   </units>
   <units name="nanoA">
      <unit units="ampere" prefix="nano"/>
   </units>
   <component name="environment">
      <variable units="second" public_interface="out" name="time"/>
   </component>
   <component name="membrane">
      <variable units="millivolt" public_interface="out" name="V" initial_value="-87"/>
      <variable units="microF" name="Cm" initial_value="12"/>
      <variable units="second" public_interface="in" name="time"/>
      <variable units="nanoA" public_interface="in" name="i_Na"/>
      <variable units="nanoA" public_interface="in" name="i_K"/>
      <variable units="nanoA" public_interface="in" name="i_Leak"/>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
         <apply>
            <eq/>
            <apply>
```

```
               <diff/>
               <bvar>
                  <ci>time</ci>
               </bvar>
               <ci>V</ci>
            </apply>
            <apply>
               <divide/>
               <apply>
                  <minus/>
                  <apply>
                     <plus/>
                     <ci>i_Na</ci>
                     <ci>i_K</ci>
                     <ci>i_Leak</ci>
                  </apply>
               </apply>
               <ci>Cm</ci>
            </apply>
         </apply>
      </math>
   </component>
   <component name="sodium_channel">
      <variable units="nanoA" public_interface="out" name="i_Na"/>
      <variable units="microS" name="g_Na_max" initial_value="400000"/>
      <variable units="microS" name="g_Na"/>
      <variable units="millivolt" name="E_Na" initial_value="40"/>
      <variable units="second" public_interface="in" private_interface="out"
name="time"/>
      <variable units="millivolt" public_interface="in" private_interface="out"
name="V"/>
      <variable units="dimensionless" private_interface="in" name="m"/>
      <variable units="dimensionless" private_interface="in" name="h"/>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
         <apply>
            <eq/>
            <ci>g_Na</ci>
            <apply>
               <times/>
               <apply>
                  <power/>
                  <ci>m</ci>
                  <cn cellml:units="dimensionless">3</cn>
               </apply>
               <ci>h</ci>
               <ci>g_Na_max</ci>
            </apply>
         </apply>
         <apply>
            <eq/>
            <ci>i_Na</ci>
            <apply>
               <times/>
               <apply>
                  <plus/>
                  <ci>g_Na</ci>
                  <cn cellml:units="microS">140</cn>
               </apply>
               <apply>
                  <minus/>
                  <ci>V</ci>
                  <ci>E_Na</ci>
               </apply>
            </apply>
         </apply>
      </math>
   </component>
   <component name="sodium_channel_m_gate">
      <variable units="dimensionless" public_interface="out" name="m"
initial_value="0.01"/>
      <variable units="per_second" name="alpha_m"/>
      <variable units="per_second" name="beta_m"/>
      <variable units="millivolt" public_interface="in" name="V"/>
      <variable units="second" public_interface="in" name="time"/>
```

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <apply>
      <eq/>
      <ci>alpha_m</ci>
      <apply>
         <divide/>
         <apply>
            <times/>
            <cn cellml:units="per_millivolt_second">100</cn>
            <apply>
               <minus/>
               <apply>
                  <minus/>
                  <ci>V</ci>
               </apply>
               <cn cellml:units="millivolt">48</cn>
            </apply>
         </apply>
         <apply>
            <minus/>
            <apply>
               <exp/>
               <apply>
                  <divide/>
                  <apply>
                     <minus/>
                     <apply>
                        <minus/>
                        <ci>V</ci>
                     </apply>
                     <cn cellml:units="millivolt">48</cn>
                  </apply>
                  <cn cellml:units="millivolt">15</cn>
               </apply>
            </apply>
            <cn cellml:units="dimensionless">1</cn>
         </apply>
      </apply>
   </apply>
   <apply>
      <eq/>
      <ci>beta_m</ci>
      <apply>
         <divide/>
         <apply>
            <times/>
            <cn cellml:units="per_millivolt_second">120</cn>
            <apply>
               <plus/>
               <ci>V</ci>
               <cn cellml:units="millivolt">8</cn>
            </apply>
         </apply>
         <apply>
            <minus/>
            <apply>
               <exp/>
               <apply>
                  <divide/>
                  <apply>
                     <plus/>
                     <ci>V</ci>
                     <cn cellml:units="millivolt">8</cn>
                  </apply>
                  <cn cellml:units="millivolt">5</cn>
               </apply>
            </apply>
            <cn cellml:units="dimensionless">1</cn>
         </apply>
      </apply>
   </apply>
   <apply>
      <eq/>
      <apply>
```

```
               <diff/>
               <bvar>
                  <ci>time</ci>
               </bvar>
               <ci>m</ci>
            </apply>
            <apply>
               <minus/>
               <apply>
                  <times/>
                  <ci>alpha_m</ci>
                  <apply>
                     <minus/>
                     <cn cellml:units="dimensionless">1</cn>
                     <ci>m</ci>
                  </apply>
               </apply>
               <apply>
                  <times/>
                  <ci>beta_m</ci>
                  <ci>m</ci>
               </apply>
            </apply>
         </apply>
      </math>
   </component>
   <component name="sodium_channel_h_gate">
      <variable units="dimensionless" public_interface="out" name="h"
initial_value="0.8"/>
      <variable units="per_second" name="alpha_h"/>
      <variable units="per_second" name="beta_h"/>
      <variable units="millivolt" public_interface="in" name="V"/>
      <variable units="second" public_interface="in" name="time"/>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
         <apply>
            <eq/>
            <ci>alpha_h</ci>
            <apply>
               <times/>
               <cn cellml:units="per_second">170</cn>
               <apply>
                  <exp/>
                  <apply>
                     <divide/>
                     <apply>
                        <minus/>
                        <apply>
                           <minus/>
                           <ci>V</ci>
                        </apply>
                        <cn cellml:units="millivolt">90</cn>
                     </apply>
                     <cn cellml:units="millivolt">20</cn>
                  </apply>
               </apply>
            </apply>
         </apply>
         <apply>
            <eq/>
            <ci>beta_h</ci>
            <apply>
               <divide/>
               <cn cellml:units="per_second">1000</cn>
               <apply>
                  <plus/>
                  <cn cellml:units="dimensionless">1</cn>
                  <apply>
                     <exp/>
                     <apply>
                        <divide/>
                        <apply>
                           <minus/>
                           <apply>
                              <minus/>
```

```
                           <ci>V</ci>
                        </apply>
                        <cn cellml:units="millivolt">42</cn>
                     </apply>
                     <cn cellml:units="millivolt">10</cn>
                  </apply>
               </apply>
            </apply>
         </apply>
         <apply>
            <eq/>
            <apply>
               <diff/>
               <bvar>
                  <ci>time</ci>
               </bvar>
               <ci>h</ci>
            </apply>
            <apply>
               <minus/>
               <apply>
                  <times/>
                  <ci>alpha_h</ci>
                  <apply>
                     <minus/>
                     <cn cellml:units="dimensionless">1</cn>
                     <ci>h</ci>
                  </apply>
               </apply>
               <apply>
                  <times/>
                  <ci>beta_h</ci>
                  <ci>h</ci>
               </apply>
            </apply>
         </apply>
      </math>
   </component>
   <component name="potassium_channel">
      <variable units="nanoA" public_interface="out" name="i_K"/>
      <variable units="microS" name="g_K1"/>
      <variable units="microS" name="g_K2"/>
      <variable units="second" public_interface="in" private_interface="out"
name="time"/>
      <variable units="millivolt" public_interface="in" private_interface="out"
name="V"/>
      <variable units="dimensionless" private_interface="in" name="n"/>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
         <apply>
            <eq/>
            <ci>i_K</ci>
            <apply>
               <times/>
               <apply>
                  <plus/>
                  <ci>g_K1</ci>
                  <ci>g_K2</ci>
               </apply>
               <apply>
                  <plus/>
                  <ci>V</ci>
                  <cn cellml:units="millivolt">100</cn>
               </apply>
            </apply>
         </apply>
         <apply>
            <eq/>
            <ci>g_K1</ci>
            <apply>
               <plus/>
               <apply>
                  <times/>
                  <cn cellml:units="microS">1200</cn>
```

```
                  <apply>
                     <exp/>
                     <apply>
                        <divide/>
                        <apply>
                           <minus/>
                           <apply>
                              <minus/>
                              <ci>V</ci>
                           </apply>
                           <cn cellml:units="millivolt">90</cn>
                        </apply>
                        <cn cellml:units="millivolt">50</cn>
                     </apply>
                  </apply>
               </apply>
               <apply>
                  <times/>
                  <cn cellml:units="microS">15</cn>
                  <apply>
                     <exp/>
                     <apply>
                        <divide/>
                        <apply>
                           <plus/>
                           <ci>V</ci>
                           <cn cellml:units="millivolt">90</cn>
                        </apply>
                        <cn cellml:units="millivolt">60</cn>
                     </apply>
                  </apply>
               </apply>
            </apply>
         </apply>
         <apply>
            <eq/>
            <ci>g_K2</ci>
            <apply>
               <times/>
               <cn cellml:units="microS">1200</cn>
               <apply>
                  <power/>
                  <ci>n</ci>
                  <cn cellml:units="dimensionless">4</cn>
               </apply>
            </apply>
         </apply>
      </math>
   </component>
   <component name="potassium_channel_n_gate">
      <variable units="dimensionless" public_interface="out" name="n"
initial_value="0.01"/>
      <variable units="per_second" name="alpha_n"/>
      <variable units="per_second" name="beta_n"/>
      <variable units="millivolt" public_interface="in" name="V"/>
      <variable units="second" public_interface="in" name="time"/>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
         <apply>
            <eq/>
            <ci>alpha_n</ci>
            <apply>
               <divide/>
               <apply>
                  <times/>
                  <cn cellml:units="per_millivolt_second">0.1</cn>
                  <apply>
                     <minus/>
                     <apply>
                        <minus/>
                        <ci>V</ci>
                     </apply>
                     <cn cellml:units="millivolt">50</cn>
                  </apply>
               </apply>
```

```
            <apply>
               <minus/>
               <apply>
                  <exp/>
                  <apply>
                     <divide/>
                     <apply>
                        <minus/>
                        <apply>
                           <minus/>
                           <ci>V</ci>
                        </apply>
                        <cn cellml:units="millivolt">50</cn>
                     </apply>
                     <cn cellml:units="millivolt">10</cn>
                  </apply>
               </apply>
               <cn cellml:units="dimensionless">1</cn>
            </apply>
         </apply>
      </apply>
      <apply>
         <eq/>
         <ci>beta_n</ci>
         <apply>
            <times/>
            <cn cellml:units="per_second">2</cn>
            <apply>
               <exp/>
               <apply>
                  <divide/>
                  <apply>
                     <minus/>
                     <apply>
                        <minus/>
                        <ci>V</ci>
                     </apply>
                     <cn cellml:units="millivolt">90</cn>
                  </apply>
                  <cn cellml:units="millivolt">80</cn>
               </apply>
            </apply>
         </apply>
      </apply>
      <apply>
         <eq/>
         <apply>
            <diff/>
            <bvar>
               <ci>time</ci>
            </bvar>
            <ci>n</ci>
         </apply>
         <apply>
            <minus/>
            <apply>
               <times/>
               <ci>alpha_n</ci>
               <apply>
                  <minus/>
                  <cn cellml:units="dimensionless">1</cn>
                  <ci>n</ci>
               </apply>
            </apply>
            <apply>
               <times/>
               <ci>beta_n</ci>
               <ci>n</ci>
            </apply>
         </apply>
      </apply>
   </math>
</component>
<component name="leakage_current">
```

```
   <variable units="nanoA" public_interface="out" name="i_Leak"/>
   <variable units="microS" name="g_L" initial_value="75"/>
   <variable units="millivolt" name="E_L" initial_value="-60"/>
   <variable units="second" public_interface="in" name="time"/>
   <variable units="millivolt" public_interface="in" name="V"/>
   <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
         <eq/>
         <ci>i_Leak</ci>
         <apply>
            <times/>
            <ci>g_L</ci>
            <apply>
               <minus/>
               <ci>V</ci>
               <ci>E_L</ci>
            </apply>
         </apply>
      </apply>
   </math>
</component>
<group>
   <relationship_ref relationship="containment"/>
   <component_ref component="membrane">
      <component_ref component="sodium_channel">
         <component_ref component="sodium_channel_m_gate"/>
         <component_ref component="sodium_channel_h_gate"/>
      </component_ref>
      <component_ref component="potassium_channel">
         <component_ref component="potassium_channel_n_gate"/>
      </component_ref>
      <component_ref component="leakage_current"/>
   </component_ref>
</group>
<group>
   <relationship_ref relationship="encapsulation"/>
   <component_ref component="sodium_channel">
      <component_ref component="sodium_channel_m_gate"/>
      <component_ref component="sodium_channel_h_gate"/>
   </component_ref>
   <component_ref component="potassium_channel">
      <component_ref component="potassium_channel_n_gate"/>
   </component_ref>
</group>
<connection>
   <map_components component_2="environment" component_1="membrane"/>
   <map_variables variable_2="time" variable_1="time"/>
</connection>
<connection>
   <map_components component_2="environment" component_1="sodium_channel"/>
   <map_variables variable_2="time" variable_1="time"/>
</connection>
<connection>
   <map_components component_2="environment" component_1="potassium_channel"/>
   <map_variables variable_2="time" variable_1="time"/>
</connection>
<connection>
   <map_components component_2="environment" component_1="leakage_current"/>
   <map_variables variable_2="time" variable_1="time"/>
</connection>
<connection>
   <map_components component_2="sodium_channel" component_1="membrane"/>
   <map_variables variable_2="V" variable_1="V"/>
   <map_variables variable_2="i_Na" variable_1="i_Na"/>
</connection>
<connection>
   <map_components component_2="potassium_channel" component_1="membrane"/>
   <map_variables variable_2="V" variable_1="V"/>
   <map_variables variable_2="i_K" variable_1="i_K"/>
</connection>
<connection>
   <map_components component_2="leakage_current" component_1="membrane"/>
   <map_variables variable_2="V" variable_1="V"/>
   <map_variables variable_2="i_Leak" variable_1="i_Leak"/>
```

```
   </connection>
   <connection>
      <map_components component_2="sodium_channel_m_gate" component_1="sodium_channel"/>
      <map_variables variable_2="m" variable_1="m"/>
      <map_variables variable_2="time" variable_1="time"/>
      <map_variables variable_2="V" variable_1="V"/>
   </connection>
   <connection>
      <map_components component_2="sodium_channel_h_gate" component_1="sodium_channel"/>
      <map_variables variable_2="h" variable_1="h"/>
      <map_variables variable_2="time" variable_1="time"/>
      <map_variables variable_2="V" variable_1="V"/>
   </connection>
   <connection>
      <map_components component_2="potassium_channel_n_gate"
component_1="potassium_channel"/>
      <map_variables variable_2="n" variable_1="n"/>
      <map_variables variable_2="time" variable_1="time"/>
      <map_variables variable_2="V" variable_1="V"/>
   </connection>
</model>
```

A2.2. Stylesheet XSLT

A seguir é apresentada a folha de estilo usada para a transformação inicial do modelo CellML (seção 4.5.1).

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:mathml="http://www.w3.org/1998/Math/MathML"
xmlns:cmeta="http://www.cellml.org/metadata/1.0#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:bqs="http://www.cellml.org/bqs/1.0#"
xmlns:cellml="http://www.cellml.org/cellml/1.0#"
xmlns:dcterms="http://purl.org/dc/terms/" xmlns:vCard="http://www.w3.org/2001/vcard-
rdf/3.0#">
   <xsl:output method="xml"/>
   <xsl:template match="/">
      <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="mathml:math"/>
   <xsl:template match="*|@*">
      <xsl:copy>
         <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
   </xsl:template>
   <xsl:template match="cellml:model">
      <xsl:element name="model">
         <xsl:copy-of select="@*"/>
         <xsl:variable name="mid">model_<xsl:value-of select="@name"/></xsl:variable>
         <xsl:attribute name="id">
            <xsl:value-of select="$mid"/>
         </xsl:attribute>
         <xsl:apply-templates/>
      </xsl:element>
   </xsl:template>
   <xsl:template match="cellml:units">
      <xsl:element name="units">
         <xsl:attribute name="name">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
         <xsl:attribute name="model">
            <xsl:value-of select="../../cellml:model/@name"/>
         </xsl:attribute>
      </xsl:element>
   </xsl:template>
   <xsl:template match="cellml:component">
      <xsl:element name="component">
         <xsl:attribute name="name">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
```

```
         <xsl:attribute name="model">
            <xsl:value-of select="../../cellml:model/@name"/>
         </xsl:attribute>
         <xsl:apply-templates select="@*|node()"/>
         <xsl:variable name="name">
            <xsl:value-of select="@name"/>
         </xsl:variable>
         <xsl:for-each select="cellml:variable[@public_interface=&apos;in&apos; or
@public_interface=&apos;out&apos; or @private_interface=&apos;in&apos; or
@private_interface=&apos;out&apos;]">
            <xsl:call-template name="component_interface">
               <xsl:with-param name="variable" select="."/>
               <xsl:with-param name="public" select="@public_interface"/>
               <xsl:with-param name="private" select="@private_interface"/>
            </xsl:call-template>
         </xsl:for-each>
         <xsl:for-each select="cellml:variable[(not(@private_interface) and
not(@public_interface)) or (@private_interface=&apos;none&apos;) or
(@public_interface=&apos;none&apos;)]">
            <xsl:call-template name="component_localvar">
               <xsl:with-param name="variable" select="."/>
            </xsl:call-template>
         </xsl:for-each>
            <xsl:call-template name="component_math">
               <xsl:with-param name="component" select="mathml:math"/>
               <xsl:with-param name="name" select="$name"/>
               <xsl:with-param name="component_name" select="@name"/>
            </xsl:call-template>
         <xsl:call-template name="extract_sub">
            <xsl:with-param name="n" select="./@name"/>
            <xsl:with-param name="d" select="0"/>
         </xsl:call-template>
      </xsl:element>
   </xsl:template>
   <xsl:template name="component_interface">
      <xsl:param name="variable"/>
      <xsl:param name="public"/>
      <xsl:param name="private"/>
      <xsl:element name="parameter">
         <xsl:variable name="pid">
            <xsl:value-of select="$variable/../@name"/>_<xsl:value-of
select="$variable/@name"/></xsl:variable>
         <xsl:variable name="vid">
            <xsl:value-of select="$variable/../@name"/>_<xsl:value-of
select="$variable/@name"/></xsl:variable>
         <xsl:variable name="mid">model_<xsl:value-of
select="$variable/../../@name"/></xsl:variable>
         <xsl:variable name="cid"><xsl:value-of
select="$variable/../@name"/></xsl:variable>
         <xsl:variable name="pub">public_<xsl:value-of select="$public"/></xsl:variable>
         <xsl:variable name="pri">private_<xsl:value-of
select="$private"/></xsl:variable>
         <xsl:attribute name="id">
            <xsl:value-of select="$pid"/>
         </xsl:attribute>
         <xsl:attribute name="vid">
            <xsl:value-of select="$vid"/>
         </xsl:attribute>
         <xsl:attribute name="mid">
            <xsl:value-of select="$mid"/>
         </xsl:attribute>
         <xsl:attribute name="cid">
            <xsl:value-of select="$cid"/>
         </xsl:attribute>
         <xsl:attribute name="public">
            <xsl:value-of select="$pub"/>
         </xsl:attribute>
         <xsl:attribute name="private">
            <xsl:value-of select="$pri"/>
         </xsl:attribute>
         <xsl:copy-of select="$variable/@*"/>
         <xsl:call-template name="extract_sub">
            <xsl:with-param name="n" select="$vid"/>
            <xsl:with-param name="d" select="0"/>
```

```
         </xsl:call-template>
      </xsl:element>
   </xsl:template>
   <xsl:template name="component_localvar">
      <xsl:param name="variable"/>
      <xsl:element name="localvar">
         <xsl:variable name="vid">
            <xsl:value-of select="$variable/../@name"/>_<xsl:value-of
select="$variable/@name"/></xsl:variable>
         <xsl:attribute name="id">
            <xsl:value-of select="$vid"/>
         </xsl:attribute>
         <xsl:copy-of select="$variable/@*"/>
         <xsl:call-template name="extract_sub">
            <xsl:with-param name="n" select="$vid"/>
            <xsl:with-param name="d" select="0"/>
         </xsl:call-template>
      </xsl:element>
   </xsl:template>

   <xsl:template match="cellml:variable"/>
   <xsl:template match="cellml:connection"/>
   <xsl:template match="cellml:group"/>

   <xsl:template name="component_math">
      <xsl:param name="component"/>
      <xsl:param name="name"/>
      <xsl:param name="component_name"/>
      <xsl:choose>
         <xsl:when test="count($component/*) &gt; 0">
            <xsl:for-each select="$component/*">
                <xsl:element name="equation">
            <xsl:attribute name="id">
               <xsl:value-of select="$component_name"/>_eq_<xsl:value-of
select="position()"/>
             </xsl:attribute>

                <xsl:element name="math">
            &lt;apply&gt;
            <xsl:call-template name="component_math_apply">
               <xsl:with-param name="component" select="."/>
               <xsl:with-param name="name" select="$name"/>
            </xsl:call-template>
            &lt;/apply&gt;
              </xsl:element>
              </xsl:element>
            </xsl:for-each>
         </xsl:when>
      </xsl:choose>
   </xsl:template>

   <xsl:template name="component_math_apply">
      <xsl:param name="component"/>
      <xsl:param name="name"/>
      <xsl:choose>
         <xsl:when test="count(./*) &gt; 0">
            <xsl:for-each select="$component/*"> &lt;
               <xsl:value-of select="name()"/>
               <xsl:for-each select="@*">
                  <xsl:value-of select="&apos; &apos;"/>
                  <xsl:value-of select="name()"/>="
                  <xsl:value-of select="."/>"
               </xsl:for-each> >
               <xsl:choose>
                  <xsl:when test="name() = &apos;ci&apos;">
                     <xsl:value-of select="$name"/>_
                     <xsl:value-of select="./text()"/></xsl:when>
                  <xsl:otherwise>
                     <xsl:value-of select="./text()"/>
                  </xsl:otherwise>
               </xsl:choose>
               <xsl:call-template name="component_math_apply">
                  <xsl:with-param name="component" select="."/>
                  <xsl:with-param name="name" select="$name"/>
```

```
               </xsl:call-template>&lt;/
               <xsl:value-of select="name()"/> >
            </xsl:for-each>
         </xsl:when>
      </xsl:choose>
   </xsl:template>


   <xsl:template name="extract_sub">
      <xsl:param name="n"/>
      <xsl:param name="d"/>
      <xsl:variable name="nid">detail_<xsl:value-of select="$d"/></xsl:variable>
      <xsl:variable name="dd">
         <xsl:value-of select="$d"/>
      </xsl:variable>
      <xsl:choose>
         <xsl:when test="contains($n,&apos;_&apos;)">
            <xsl:element name="{$nid}">
               <xsl:value-of select="substring-before($n,&apos;_&apos;)"/>
            </xsl:element>
            <xsl:call-template name="extract_sub">
               <xsl:with-param name="n" select="substring-after($n,&apos;_&apos;)"/>
               <xsl:with-param name="d" select="$d + 1"/>
            </xsl:call-template>
         </xsl:when>
         <xsl:otherwise>
            <xsl:element name="{$nid}">
               <xsl:value-of select="$n"/>
            </xsl:element>
            <xsl:if test="(number($dd) &lt; 5)">
               <xsl:call-template name="extract_sub">
                  <xsl:with-param name="n" select="&apos;-&apos;"/>
                  <xsl:with-param name="d" select="$d + 1"/>
               </xsl:call-template>
            </xsl:if>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>
</xsl:stylesheet>
```

A2.3. Regras de Conversão

A seguir é apresentado o arquivo de configuração com as regras de transformação de XML para OWL, usado com o software XML2OWL (seção 4.5.1).

```
; Define namespaces
; Syntax: NAMESPACE(namespace_prefix,namespace_uri)
NAMESPACE("celo","http://celo.mmc.ufjf.br/ontologies/celo.owl")
NAMESPACE("sqwrl","http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl")
NAMESPACE("swrlxml","http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlxml.owl")
NAMESPACE("temporal","http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl")
NAMESPACE("abox","http://swrl.stanford.edu/ontologies/built-ins/3.3/abox.owl")
NAMESPACE("tbox","http://swrl.stanford.edu/ontologies/built-ins/3.3/tbox.owl")
NAMESPACE("swrlm","http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlm.owl")
NAMESPACE("swrlx","http://swrl.stanford.edu/ontologies/built-ins/3.3/swrlx.owl")
NAMESPACE("swrla","http://swrl.stanford.edu/ontologies/3.3/swrla.owl")

; Base namespace
BNAMESPACE("model","--Model_URI--")

; Default namespace
DNAMESPACE("model","--Model_URI--")

; Default namespace prefix
NSPREFIX("model","--Model_URI--")

; Define a list of namespace prefixes for the imported ontologies
; This rule is mandatory; should be inluded even if the ontology being written
; does not import any other ontology.
; Syntax: IMPORT("ns_prefix1","ns_prefix2", ...)
IMPORT("celo","sqwrl","swrlxml","temporal","abox","tbox","swrlm","swrlx","swrla")
```

```
;-------------------------------------------------------------------------
; Create Model

IND(celo:Model,model,model@id)
IND(celo:ModelService,model,model@id)
DTP(celo:hasName,ModelService,model@name, string)

IND(celo:CellMLModel,model,"_CellMLModel")
DTP(celo:hasURI,CellMLModel,model@reference, string)
OTP(celo:refersTo,ModelService,CellMLModel)

IND(celo:ModelProfile,model,"_ModelProfile")
OTP(celo:presents,ModelService,ModelProfile)

IND(celo:ModelInterface,model,"_ModelInterface")
OTP(celo:describedBy,ModelService,ModelInterface)

;-------------------------------------------------------------------------
; Create Units

IND(celo:UserDefinedUnit,units,units@name)
DTP(celo:hasName,UserDefinedUnit,units@name, string)

;-------------------------------------------------------------------------
; Create InternalComponent

IND(celo:InternalComponent,component,component@name)
;OTP(celo:hasMainComponent,ModelProfile,InternalComponent)
DTP(celo:hasName,InternalComponent,component@name, string)

DTP(celo:hasDetail,InternalComponent,component/detail_0, string)
DTP(celo:hasDetail,InternalComponent,component/detail_1, string)
DTP(celo:hasDetail,InternalComponent,component/detail_2, string)
DTP(celo:hasDetail,InternalComponent,component/detail_3, string)
DTP(celo:hasDetail,InternalComponent,component/detail_4, string)
DTP(celo:hasDetail,InternalComponent,component/detail_5, string)

;-------------------------------------------------------------------------
; Create Equation

IND(celo:Equation,equation,equation@id)
DTP(celo:hasMath,Equation,equation/math, string)

OTP(celo:hasEquation,InternalComponent,Equation)

;-------------------------------------------------------------------------
; Create LocalVariable

IND(celo:LocalVariable,localvar,localvar@id)
DTP(celo:hasName,LocalVariable,localvar@name, string)
DTP(celo:hasInitialValue,LocalVariable,localvar@initial_value, string)
DTP(celo:hasDetail,LocalVariable,localvar@units, string)

OTP(celo:hasLocalVariable,InternalComponent,LocalVariable)
DTP(celo:hasDetail,LocalVariable,localvar/detail_0, string)
DTP(celo:hasDetail,LocalVariable,localvar/detail_1, string)
DTP(celo:hasDetail,LocalVariable,localvar/detail_2, string)
DTP(celo:hasDetail,LocalVariable,localvar/detail_3, string)
DTP(celo:hasDetail,LocalVariable,localvar/detail_4, string)
DTP(celo:hasDetail,LocalVariable,localvar/detail_5, string)

OTP(celo:hasLocalVariable,InternalComponent,LocalVariable)

;-------------------------------------------------------------------------
; Create InterfaceVariable

IND(celo:InterfaceVariable,parameter,parameter@id)
DTP(celo:hasName,InterfaceVariable,parameter@name, string)
DTP(celo:hasInitialValue,InterfaceVariable,parameter@initial_value, string)
DTP(celo:hasDetail,InterfaceVariable,parameter@public, string)
DTP(celo:hasDetail,InterfaceVariable,parameter@private, string)
DTP(celo:hasDetail,InterfaceVariable,parameter@units, string)
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_0, string)
```

```
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_1, string)
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_2, string)
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_3, string)
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_4, string)
DTP(celo:hasDetail,InterfaceVariable,parameter/detail_5, string)

OTP(celo:hasInterfaceVariable,InternalComponent,InterfaceVariable)
OTP(celo:hasInterface,ModelInterface,InterfaceVariable,model@id,parameter@mid)
OTP(celo:isInterfaceVariableOf,InterfaceVariable,InternalComponent,parameter@cid,compone
nt@name)

IND(celo:ParameterVariable,modelvariabletype,modelvariabletype@varid,modelvariabletype@t
ype=":parametervariable")
IND(celo:AlgebricVariable,modelvariabletype,modelvariabletype@varid,modelvariabletype@ty
pe=":algebricvariable")
IND(celo:IndependentVariable,modelvariabletype,modelvariabletype@varid,modelvariabletype
@type=":independentvariable")
IND(celo:DependentVariable,modelvariabletype,modelvariabletype@varid,modelvariabletype@t
ype=":dependentvariable")

IND(celo:ModelParameterIn,modelvariabletype,modelvariabletype@varid,modelvariabletype@ty
pe=":parametervariable")
IND(celo:ModelParameterOut,modelvariabletype,modelvariabletype@varid,modelvariabletype@t
ype=":dependentvariable")
OTP(celo:hasInterface,ModelInterface,ModelParameterIn)
OTP(celo:hasInterface,ModelInterface,ModelParameterOut)
```

A2.4. Modelo CelO

A seguir é apresentada a representação em OWL do modelo “A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials” (CELLML, 2008b).

```
@prefix celo:    <http://celo.mmc.ufjf.br/ontologies/celo.owl#> .
@prefix dc:      <http://purl.org/dc/elements/1.1/> .
@prefix protege:  <http://protege.stanford.edu/plugins/owl/protege#> .
@prefix swrlx:   <http://swrl.stanford.edu/ontologies/built-ins/3.3/swrlx.owl#> .
@prefix protegedc:  <http://protege.stanford.edu/plugins/owl/dc/protege-dc.owl#> .
@prefix abox:    <http://swrl.stanford.edu/ontologies/built-ins/3.3/abox.owl#> .
@prefix dcterms:  <http://purl.org/dc/terms/> .
@prefix sqwrl:   <http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix swrl:    <http://www.w3.org/2003/11/swrl#> .
@prefix swrlm:   <http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlm.owl#> .
@prefix celo:  <http://celo.mmc.ufjf.br/ontologies/noble_1962#> .
@prefix swrlxml:  <http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlxml.owl#> .
@prefix temporal:  <http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix swrlb:   <http://www.w3.org/2003/11/swrlb#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix tbox:    <http://swrl.stanford.edu/ontologies/built-ins/3.3/tbox.owl#> .
@prefix swrla:   <http://swrl.stanford.edu/ontologies/3.3/swrla.owl#> .

celo:leakage_current_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nleakage_current_\ni_Leak</\nci > \n <\napply > \n\n                <\ntimes >
\n</\ntimes > \n <\nci > \nleakage_current_\ng_L</\nci > \n <\napply > \n\n<\nminus >
\n</\nminus > \n <\nci > \nleakage_current_\nV</\nci > \n <\nci >
\nleakage_current_\nE_L</\nci > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:leakage_current
      rdf:type celo:InternalComponent ;
      celo:hasDetail "current"^^xsd:string , "leakage"^^xsd:string , "-"^^xsd:string ;
      celo:hasEquation celo:leakage_current_eq_1 ;
      celo:hasInterfaceVariable
```

```
              celo:leakage_current_V , celo:leakage_current_i_Leak ,
celo:leakage_current_time ;
      celo:hasLocalVariable
              celo:leakage_current_g_L , celo:leakage_current_E_L ;
      celo:hasName "leakage_current"^^xsd:string .

<http://celo.mmc.ufjf.br/ontologies/noble_1962>
      rdf:type owl:Ontology ;
      owl:imports <http://celo.mmc.ufjf.br/ontologies/celo.owl> .

celo:sodium_channel_h
      rdf:type celo:ModelParameterOut , celo:DependentVariable , celo:InterfaceVariable
;
      celo:hasDetail "dimensionless"^^xsd:string , "h"^^xsd:string ,
"private_in"^^xsd:string , "sodium"^^xsd:string , "channel"^^xsd:string , "-
"^^xsd:string , "public_"^^xsd:string ;
      celo:hasDomainEntity
              celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "h"^^xsd:string ;
      celo:hasPrivateInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:sodium_channel .

celo:membrane_i_Na
      rdf:type celo:AlgebricVariable , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "i"^^xsd:string , "Na"^^xsd:string ,
"public_in"^^xsd:string , "membrane"^^xsd:string , "nanoA"^^xsd:string , "-"^^xsd:string
;
      celo:hasDomainEntity
              celo:Sodium , celo:Membrane ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_Na"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:membrane .

celo:sodium_channel_m_gate_eq_2
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_m_gate_\nbeta_m</\nci > \n <\napply > \n\n                <\ndivide >
\n</\ndivide > \n <\napply > \n\n<\ntimes > \n</\ntimes > \n <\ncn
cellml:units=\"\nper_millivolt_second\" \n > \n120</\ncn > \n <\napply > \n\n<\nplus >
\n</\nplus > \n <\nci > \nsodium_channel_m_gate_\nV</\nci > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n8</\ncn > \n</\napply > \n</\napply > \n <\napply >
\n\n<\nminus > \n</\nminus > \n <\napply > \n\n<\nexp > \n</\nexp > \n <\napply > \n\n
<\ndivide > \n</\ndivide > \n <\napply > \n\n<\nplus > \n</\nplus > \n <\nci >
\nsodium_channel_m_gate_\nV</\nci > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n8</\ncn
> \n</\napply > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n5</\ncn > \n</\napply >
\n</\napply > \n <\ncn cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n</\napply >
\n</\napply > \n\n</apply>\n        "^^xsd:string .

celo:potassium_channel_n_gate_V
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "V"^^xsd:string , "potassium"^^xsd:string , "gate"^^xsd:string
, "channel"^^xsd:string , "n"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPublicInterface
```

```
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:potassium_channel_n_gate .

celo:per_second
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_second"^^xsd:string .

celo:sodium_channel_m
      rdf:type celo:ModelParameterOut , celo:DependentVariable , celo:InterfaceVariable
;
      celo:hasDetail "dimensionless"^^xsd:string , "m"^^xsd:string ,
"channel"^^xsd:string , "sodium"^^xsd:string , "private_in"^^xsd:string , "-
"^^xsd:string , "public_"^^xsd:string ;
      celo:hasDomainEntity
              celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "m"^^xsd:string ;
      celo:hasPrivateInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:sodium_channel .

celo:per_millivolt
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_millivolt"^^xsd:string .

celo:potassium_channel_n_gate_n
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "dimensionless"^^xsd:string ,
"potassium"^^xsd:string , "gate"^^xsd:string , "n"^^xsd:string , "channel"^^xsd:string ,
"-"^^xsd:string , "public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "0.01"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "n"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:potassium_channel_n_gate .

celo:sodium_channel_m_gate
      rdf:type celo:InternalComponent , celo:IonChannelGate ;
      celo:hasDetail "m"^^xsd:string , "gate"^^xsd:string , "channel"^^xsd:string ,
"sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasEquation celo:sodium_channel_m_gate_eq_2 , celo:sodium_channel_m_gate_eq_1
, celo:sodium_channel_m_gate_eq_3 ;
      celo:hasInterfaceVariable
              celo:sodium_channel_m_gate_m , celo:sodium_channel_m_gate_V ,
celo:sodium_channel_m_gate_time ;
      celo:hasLocalVariable
              celo:sodium_channel_m_gate_beta_m , celo:sodium_channel_m_gate_alpha_m ;
      celo:hasName "sodium_channel_m_gate"^^xsd:string .

celo:potassium_channel_i_K
      rdf:type celo:PotassiumIonChannel , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "i"^^xsd:string , "K"^^xsd:string ,
"nanoA"^^xsd:string , "potassium"^^xsd:string , "channel"^^xsd:string , "-"^^xsd:string
, "public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasInitialValue
```

```
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_K"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:potassium_channel .

celo:sodium_channel_m_gate_alpha_m
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "per_second"^^xsd:string , "alpha"^^xsd:string , "m"^^xsd:string ,
"gate"^^xsd:string , "sodium"^^xsd:string , "channel"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "alpha_m"^^xsd:string ;
      celo:hasVariableUnit
              celo:per_second .

celo:sodium_channel_h_gate_eq_2
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_h_gate_\nbeta_h</\nci > \n <\napply > \n\n                <\ndivide >
\n</\ndivide > \n <\ncn cellml:units=\"\nper_second\" \n > \n1000</\ncn > \n <\napply >
\n\n<\nplus > \n</\nplus > \n <\ncn cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n
<\napply > \n\n<\nexp > \n</\nexp > \n <\napply > \n\n   <\ndivide > \n</\ndivide > \n
<\napply > \n\n<\nminus > \n</\nminus > \n <\napply > \n\n   <\nminus > \n</\nminus > \n
<\nci > \nsodium_channel_h_gate_\nV</\nci > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n42</\ncn > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n10</\ncn > \n</\napply > \n</\napply > \n</\napply >
\n</\napply > \n\n</apply>\n        "^^xsd:string .

celo:potassium_channel_n_gate_alpha_n
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "per_second"^^xsd:string , "potassium"^^xsd:string ,
"alpha"^^xsd:string , "gate"^^xsd:string , "channel"^^xsd:string , "n"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "alpha_n"^^xsd:string ;
      celo:hasVariableUnit
              celo:per_second .

celo:sodium_channel_m_gate_m
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "dimensionless"^^xsd:string ,
"m"^^xsd:string , "gate"^^xsd:string , "sodium"^^xsd:string , "channel"^^xsd:string , "-
"^^xsd:string , "public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "0.01"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "m"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_m_gate .

celo:potassium_channel_n_gate_beta_n
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "beta"^^xsd:string , "per_second"^^xsd:string ,
"potassium"^^xsd:string , "gate"^^xsd:string , "channel"^^xsd:string , "n"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "beta_n"^^xsd:string ;
```

```
      celo:hasVariableUnit
              celo:per_second .

celo:sodium_channel_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_\ng_Na</\nci > \n <\napply > \n\n                <\ntimes > \n</\ntimes
> \n <\napply > \n\n<\npower > \n</\npower > \n <\nci > \nsodium_channel_\nm</\nci > \n
<\ncn cellml:units=\"\ndimensionless\" \n > \n3</\ncn > \n</\napply > \n <\nci >
\nsodium_channel_\nh</\nci > \n <\nci > \nsodium_channel_\ng_Na_max</\nci > \n</\napply
> \n\n</apply>\n        "^^xsd:string .

celo:membrane
      rdf:type celo:InternalComponent ;
      celo:hasDetail "membrane"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Membrane ;
      celo:hasEquation celo:membrane_eq_1 ;
      celo:hasInterfaceVariable
              celo:membrane_i_Na , celo:membrane_time , celo:membrane_V ,
celo:membrane_i_Leak , celo:membrane_i_K ;
      celo:hasLocalVariable
              celo:membrane_Cm ;
      celo:hasName "membrane"^^xsd:string .

celo:potassium_channel_time
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "second"^^xsd:string , "private_out"^^xsd:string ,
"public_in"^^xsd:string , "time"^^xsd:string , "potassium"^^xsd:string ,
"channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPrivateInterface
              "out"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:potassium_channel .

celo:potassium_channel_eq_3
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\npotassium_channel_\ng_K2</\nci > \n <\napply > \n\n                <\ntimes >
\n</\ntimes > \n <\ncn cellml:units=\"\nmicroS\" \n > \n1200</\ncn > \n <\napply >
\n\n<\npower > \n</\npower > \n <\nci > \npotassium_channel_\nn</\nci > \n <\ncn
cellml:units=\"\ndimensionless\" \n > \n4</\ncn > \n</\napply > \n</\napply >
\n\n</apply>\n        "^^xsd:string .

celo:sodium_channel_h_gate_time
      rdf:type celo:InterfaceVariable , celo:IonChannelGate ;
      celo:hasDetail "second"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "h"^^xsd:string , "time"^^xsd:string , "gate"^^xsd:string ,
"channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_h_gate .

celo:_CellMLModel
      rdf:type celo:CellMLModel ;
```

```
      celo:hasURI "http://celo.mmc.ufjf.br/models/noble_1962.cellml"^^xsd:string .

celo:leakage_current_time
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "current"^^xsd:string , "second"^^xsd:string ,
"private_"^^xsd:string , "public_in"^^xsd:string , "leakage"^^xsd:string ,
"time"^^xsd:string , "-"^^xsd:string ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:leakage_current .

celo:potassium_channel_n_gate_time
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "second"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "time"^^xsd:string , "potassium"^^xsd:string ,
"gate"^^xsd:string , "channel"^^xsd:string , "n"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:potassium_channel_n_gate .

celo:membrane_time
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "second"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "membrane"^^xsd:string , "time"^^xsd:string , "-"^^xsd:string
;
      celo:hasDomainEntity
              celo:Membrane ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:membrane .

celo:sodium_channel_m_gate_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_m_gate_\nalpha_m</\nci > \n <\napply > \n\n                <\ndivide >
\n</\ndivide > \n <\napply > \n\n<\ntimes > \n</\ntimes > \n <\ncn
cellml:units=\"\nper_millivolt_second\" \n > \n100</\ncn > \n <\napply > \n\n<\nminus >
\n</\nminus > \n <\napply > \n\n   <\nminus > \n</\nminus > \n <\nci >
\nsodium_channel_m_gate_\nV</\nci > \n</\napply > \n <\ncn cellml:units=\"\nmillivolt\"
\n > \n48</\ncn > \n</\napply > \n</\napply > \n <\napply > \n\n<\nminus > \n</\nminus >
\n <\napply > \n\n<\nexp > \n</\nexp > \n <\napply > \n\n   <\ndivide > \n</\ndivide >
\n <\napply > \n\n<\nminus > \n</\nminus > \n <\napply > \n\n   <\nminus > \n</\nminus >
\n <\nci > \nsodium_channel_m_gate_\nV</\nci > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n48</\ncn > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n15</\ncn > \n</\napply > \n</\napply > \n <\ncn
cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n</\napply > \n</\napply >
\n\n</apply>\n        "^^xsd:string .

celo:membrane_Cm
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:LocalVariable ;
```

```
      celo:hasDetail "microF"^^xsd:string , "membrane"^^xsd:string , "Cm"^^xsd:string ,
"-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Membrane ;
      celo:hasInitialValue
              "12"^^xsd:string ;
      celo:hasMeasure celo:Capacitance ;
      celo:hasName "Cm"^^xsd:string ;
      celo:hasVariableUnit
              celo:microF , celo:microF .

celo:nanoA
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "nanoA"^^xsd:string .

celo:sodium_channel_h_gate_V
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "V"^^xsd:string , "h"^^xsd:string , "gate"^^xsd:string ,
"channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_h_gate .

celo:sodium_channel_h_gate_h
      rdf:type celo:InterfaceVariable , celo:IonChannelGate ;
      celo:hasDetail "private_"^^xsd:string , "dimensionless"^^xsd:string ,
"h"^^xsd:string , "gate"^^xsd:string , "sodium"^^xsd:string , "channel"^^xsd:string , "-
"^^xsd:string , "public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "0.8"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "h"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_h_gate .

celo:sodium_channel_E_Na
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:LocalVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "Na"^^xsd:string , "sodium"^^xsd:string ,
"channel"^^xsd:string , "E"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "40"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "E_Na"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt .

celo:sodium_channel_h_gate_eq_3
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\napply > \n\n
<\ndiff > \n</\ndiff > \n <\nbvar > \n\n<\nci > \nsodium_channel_h_gate_\ntime</\nci >
\n</\nbvar > \n <\nci > \nsodium_channel_h_gate_\nh</\nci > \n</\napply > \n <\napply >
\n\n               <\nminus > \n</\nminus > \n <\napply > \n\n<\ntimes > \n</\ntimes >
\n <\nci > \nsodium_channel_h_gate_\nalpha_h</\nci > \n <\napply > \n\n<\nminus >
\n</\nminus > \n <\ncn cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n <\nci >
\nsodium_channel_h_gate_\nh</\nci > \n</\napply > \n</\napply > \n <\napply >
\n\n<\ntimes > \n</\ntimes > \n <\nci > \nsodium_channel_h_gate_\nbeta_h</\nci > \n
```

```
<\nci > \nsodium_channel_h_gate_\nh</\nci > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:potassium_channel_n
      rdf:type celo:ModelParameterOut , celo:DependentVariable , celo:InterfaceVariable
;
      celo:hasDetail "dimensionless"^^xsd:string , "potassium"^^xsd:string ,
"private_in"^^xsd:string , "channel"^^xsd:string , "n"^^xsd:string , "-"^^xsd:string ,
"public_"^^xsd:string ;
      celo:hasDomainEntity
              celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:ADimensional ;
      celo:hasName "n"^^xsd:string ;
      celo:hasPrivateInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:dimensionless ;
      celo:isInterfaceVariableOf
              celo:potassium_channel .

celo:potassium_channel_g_K2
      rdf:type celo:LocalVariable , celo:AlgebricVariable ;
      celo:hasDetail "g"^^xsd:string , "microS"^^xsd:string , "potassium"^^xsd:string ,
"channel"^^xsd:string , "-"^^xsd:string , "K2"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "g_K2"^^xsd:string ;
      celo:hasVariableUnit
              celo:microS .

celo:millivolt
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "millivolt"^^xsd:string .

celo:leakage_current_i_Leak
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "current"^^xsd:string , "private_"^^xsd:string , "i"^^xsd:string ,
"Leak"^^xsd:string , "nanoA"^^xsd:string , "leakage"^^xsd:string , "-"^^xsd:string ,
"public_out"^^xsd:string ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_Leak"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:leakage_current .

celo:sodium_channel_h_gate_beta_h
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "beta"^^xsd:string , "per_second"^^xsd:string , "h"^^xsd:string ,
"gate"^^xsd:string , "channel"^^xsd:string , "sodium"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "beta_h"^^xsd:string ;
      celo:hasVariableUnit
              celo:per_second .

celo:potassium_channel_n_gate_eq_3
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\napply > \n\n
<\ndiff > \n</\ndiff > \n <\nbvar > \n\n<\nci > \npotassium_channel_n_gate_\ntime</\nci
> \n</\nbvar > \n <\nci > \npotassium_channel_n_gate_\nn</\nci > \n</\napply > \n
<\napply > \n\n               <\nminus > \n</\nminus > \n <\napply > \n\n<\ntimes >
\n</\ntimes > \n <\nci > \npotassium_channel_n_gate_\nalpha_n</\nci > \n <\napply >
\n\n<\nminus > \n</\nminus > \n <\ncn cellml:units=\"\ndimensionless\" \n > \n1</\ncn >
```

```
\n <\nci > \npotassium_channel_n_gate_\nn</\nci > \n</\napply > \n</\napply > \n
<\napply > \n\n<\ntimes > \n</\ntimes > \n <\nci >
\npotassium_channel_n_gate_\nbeta_n</\nci > \n <\nci >
\npotassium_channel_n_gate_\nn</\nci > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:sodium_channel_eq_2
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_\ni_Na</\nci > \n <\napply > \n\n                <\ntimes > \n</\ntimes
> \n <\napply > \n\n<\nplus > \n</\nplus > \n <\nci > \nsodium_channel_\ng_Na</\nci > \n
<\ncn cellml:units=\"\nmicroS\" \n > \n140</\ncn > \n</\napply > \n <\napply >
\n\n<\nminus > \n</\nminus > \n <\nci > \nsodium_channel_\nV</\nci > \n <\nci >
\nsodium_channel_\nE_Na</\nci > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:membrane_i_Leak
      rdf:type celo:AlgebricVariable , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "i"^^xsd:string , "membrane"^^xsd:string ,
"Leak"^^xsd:string , "public_in"^^xsd:string , "nanoA"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Membrane ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_Leak"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:membrane .

celo:sodium_channel_V
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "private_out"^^xsd:string ,
"V"^^xsd:string , "public_in"^^xsd:string , "channel"^^xsd:string , "sodium"^^xsd:string
, "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPrivateInterface
              "out"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:sodium_channel .

celo:sodium_channel_h_gate
      rdf:type celo:InternalComponent , celo:IonChannelGate ;
      celo:hasDetail "h"^^xsd:string , "gate"^^xsd:string , "sodium"^^xsd:string ,
"channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasEquation celo:sodium_channel_h_gate_eq_2 , celo:sodium_channel_h_gate_eq_3
, celo:sodium_channel_h_gate_eq_1 ;
      celo:hasInterfaceVariable
              celo:sodium_channel_h_gate_h , celo:sodium_channel_h_gate_V ,
celo:sodium_channel_h_gate_time ;
      celo:hasLocalVariable
              celo:sodium_channel_h_gate_alpha_h , celo:sodium_channel_h_gate_beta_h ;
      celo:hasName "sodium_channel_h_gate"^^xsd:string .

celo:potassium_channel_eq_2
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\npotassium_channel_\ng_K1</\nci > \n <\napply > \n\n                <\nplus >
\n</\nplus > \n <\napply > \n\n<\ntimes > \n</\ntimes > \n <\ncn
cellml:units=\"\nmicroS\" \n > \n1200</\ncn > \n <\napply > \n\n<\nexp > \n</\nexp > \n
```

```
<\napply > \n\n   <\ndivide > \n</\ndivide > \n <\napply > \n\n<\nminus > \n</\nminus >
\n <\napply > \n\n   <\nminus > \n</\nminus > \n <\nci > \npotassium_channel_\nV</\nci >
\n</\napply > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n90</\ncn > \n</\napply > \n
<\ncn cellml:units=\"\nmillivolt\" \n > \n50</\ncn > \n</\napply > \n</\napply >
\n</\napply > \n <\napply > \n\n<\ntimes > \n</\ntimes > \n <\ncn
cellml:units=\"\nmicroS\" \n > \n15</\ncn > \n <\napply > \n\n<\nexp > \n</\nexp > \n
<\napply > \n\n   <\ndivide > \n</\ndivide > \n <\napply > \n\n<\nplus > \n</\nplus > \n
<\nci > \npotassium_channel_\nV</\nci > \n <\ncn cellml:units=\"\nmillivolt\" \n >
\n90</\ncn > \n</\napply > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n60</\ncn >
\n</\napply > \n</\napply > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:sodium_channel_g_Na_max
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:LocalVariable ;
      celo:hasDetail "max"^^xsd:string , "g"^^xsd:string , "Na"^^xsd:string ,
"microS"^^xsd:string , "channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "400000"^^xsd:string ;
      celo:hasName "g_Na_max"^^xsd:string ;
      celo:hasVariableUnit
              celo:microS .

celo:per_millivolt_second
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "per_millivolt_second"^^xsd:string .

celo:leakage_current_g_L
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:LocalVariable ;
      celo:hasDetail "current"^^xsd:string , "L"^^xsd:string , "g"^^xsd:string ,
"leakage"^^xsd:string , "microS"^^xsd:string , "-"^^xsd:string ;
      celo:hasInitialValue
              "75"^^xsd:string ;
      celo:hasName "g_L"^^xsd:string ;
      celo:hasVariableUnit
              celo:microS .

celo:sodium_channel_h_gate_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\nsodium_channel_h_gate_\nalpha_h</\nci > \n <\napply > \n\n                <\ntimes >
\n</\ntimes > \n <\ncn cellml:units=\"\nper_second\" \n > \n170</\ncn > \n <\napply >
\n\n<\nexp > \n</\nexp > \n <\napply > \n\n<\ndivide > \n</\ndivide > \n <\napply > \n\n
<\nminus > \n</\nminus > \n <\napply > \n\n<\nminus > \n</\nminus > \n <\nci >
\nsodium_channel_h_gate_\nV</\nci > \n</\napply > \n <\ncn cellml:units=\"\nmillivolt\"
\n > \n90</\ncn > \n</\napply > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n20</\ncn >
\n</\napply > \n</\napply > \n</\napply > \n\n</apply>\n        "^^xsd:string .

celo:_ModelInterface
      rdf:type celo:ModelInterface ;
      celo:hasInterface celo:potassium_channel_n , celo:sodium_channel_g_Na_max ,
celo:sodium_channel_h , celo:leakage_current_g_L , celo:environment_time ,
celo:leakage_current_E_L , celo:membrane_Cm , celo:membrane_V , celo:sodium_channel_m ,
celo:sodium_channel_E_Na .

celo:potassium_channel_V
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "private_out"^^xsd:string ,
"public_in"^^xsd:string , "V"^^xsd:string , "potassium"^^xsd:string ,
"channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPrivateInterface
              "out"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
```

```
              celo:potassium_channel .

celo:sodium_channel_g_Na
      rdf:type celo:LocalVariable , celo:AlgebricVariable ;
      celo:hasDetail "Na"^^xsd:string , "g"^^xsd:string , "microS"^^xsd:string ,
"channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "g_Na"^^xsd:string ;
      celo:hasVariableUnit
              celo:microS .

celo:environment
      rdf:type celo:InternalComponent ;
      celo:hasDetail "environment"^^xsd:string , "-"^^xsd:string ;
      celo:hasInterfaceVariable
              celo:environment_time ;
      celo:hasName "environment"^^xsd:string .

celo:noble_1962_version04
      rdf:type celo:Model , celo:ModelService ;
      celo:describedBy celo:_ModelInterface ;
      celo:hasName "noble_1962_version04"^^xsd:string ;
      celo:presents celo:_ModelProfile ;
      celo:refersTo celo:_CellMLModel .

celo:potassium_channel_g_K1
      rdf:type celo:LocalVariable , celo:AlgebricVariable ;
      celo:hasDetail "K1"^^xsd:string , "g"^^xsd:string , "microS"^^xsd:string ,
"potassium"^^xsd:string , "channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "g_K1"^^xsd:string ;
      celo:hasVariableUnit
              celo:microS .

celo:potassium_channel_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\npotassium_channel_\ni_K</\nci > \n <\napply > \n\n               <\ntimes >
\n</\ntimes > \n <\napply > \n\n<\nplus > \n</\nplus > \n <\nci >
\npotassium_channel_\ng_K1</\nci > \n <\nci > \npotassium_channel_\ng_K2</\nci >
\n</\napply > \n <\napply > \n\n<\nplus > \n</\nplus > \n <\nci >
\npotassium_channel_\nV</\nci > \n <\ncn cellml:units=\"\nmillivolt\" \n > \n100</\ncn >
\n</\napply > \n</\napply > \n\n</apply>\n        "^^xsd:string .

celo:sodium_channel_m_gate_time
      rdf:type celo:IonChannelGate , celo:InterfaceVariable ;
      celo:hasDetail "second"^^xsd:string , "private_"^^xsd:string ,
"public_in"^^xsd:string , "m"^^xsd:string , "time"^^xsd:string , "gate"^^xsd:string ,
"channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_m_gate .

celo:potassium_channel_n_gate_eq_2
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\npotassium_channel_n_gate_\nbeta_n</\nci > \n <\napply > \n\n                <\ntimes >
\n</\ntimes > \n <\ncn cellml:units=\"\nper_second\" \n > \n2</\ncn > \n <\napply >
\n\n<\nexp > \n</\nexp > \n <\napply > \n\n<\ndivide > \n</\ndivide > \n <\napply > \n\n
```

```
<\nminus > \n</\nminus > \n <\napply > \n\n<\nminus > \n</\nminus > \n <\nci >
\npotassium_channel_n_gate_\nV</\nci > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n90</\ncn > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n80</\ncn > \n</\napply > \n</\napply > \n</\napply >
\n\n</apply>\n        "^^xsd:string .

celo:sodium_channel
      rdf:type celo:InternalComponent ;
      celo:hasDetail "channel"^^xsd:string , "sodium"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasEquation celo:sodium_channel_eq_1 , celo:sodium_channel_eq_2 ;
      celo:hasInterfaceVariable
              celo:sodium_channel_h , celo:sodium_channel_m , celo:sodium_channel_V ,
celo:sodium_channel_i_Na , celo:sodium_channel_time ;
      celo:hasLocalVariable
              celo:sodium_channel_g_Na , celo:sodium_channel_g_Na_max ,
celo:sodium_channel_E_Na ;
      celo:hasName "sodium_channel"^^xsd:string .

celo:membrane_i_K
      rdf:type celo:AlgebricVariable , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "i"^^xsd:string , "K"^^xsd:string ,
"public_in"^^xsd:string , "membrane"^^xsd:string , "nanoA"^^xsd:string , "-"^^xsd:string
;
      celo:hasDomainEntity
              celo:Membrane , celo:Potassium ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_K"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:membrane .

celo:leakage_current_V
      rdf:type celo:InterfaceVariable ;
      celo:hasDetail "current"^^xsd:string , "millivolt"^^xsd:string ,
"private_"^^xsd:string , "public_in"^^xsd:string , "V"^^xsd:string ,
"leakage"^^xsd:string , "-"^^xsd:string ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:leakage_current .

celo:potassium_channel_n_gate
      rdf:type celo:InternalComponent , celo:IonChannelGate ;
      celo:hasDetail "potassium"^^xsd:string , "gate"^^xsd:string ,
"channel"^^xsd:string , "n"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Gate , celo:Potassium , celo:Channel ;
      celo:hasEquation celo:potassium_channel_n_gate_eq_3 ,
celo:potassium_channel_n_gate_eq_2 , celo:potassium_channel_n_gate_eq_1 ;
      celo:hasInterfaceVariable
              celo:potassium_channel_n_gate_time , celo:potassium_channel_n_gate_V ,
celo:potassium_channel_n_gate_n ;
      celo:hasLocalVariable
              celo:potassium_channel_n_gate_alpha_n ,
celo:potassium_channel_n_gate_beta_n ;
      celo:hasName "potassium_channel_n_gate"^^xsd:string .

celo:sodium_channel_time
      rdf:type celo:InterfaceVariable ;
```

```
      celo:hasDetail "second"^^xsd:string , "private_out"^^xsd:string ,
"public_in"^^xsd:string , "time"^^xsd:string , "sodium"^^xsd:string ,
"channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPrivateInterface
              "out"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:sodium_channel .

celo:sodium_channel_h_gate_alpha_h
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "per_second"^^xsd:string , "h"^^xsd:string , "alpha"^^xsd:string ,
"gate"^^xsd:string , "channel"^^xsd:string , "sodium"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "alpha_h"^^xsd:string ;
      celo:hasVariableUnit
              celo:per_second .

celo:sodium_channel_m_gate_V
      rdf:type celo:InterfaceVariable , celo:IonChannelGate ;
      celo:hasDetail "millivolt"^^xsd:string , "private_"^^xsd:string , "V"^^xsd:string
, "public_in"^^xsd:string , "m"^^xsd:string , "gate"^^xsd:string , "sodium"^^xsd:string
, "channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPublicInterface
              "in"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:sodium_channel_m_gate .

celo:potassium_channel
      rdf:type celo:InternalComponent ;
      celo:hasDetail "potassium"^^xsd:string , "channel"^^xsd:string , "-"^^xsd:string ;
      celo:hasDomainEntity
              celo:Potassium , celo:Channel ;
      celo:hasEquation celo:potassium_channel_eq_1 , celo:potassium_channel_eq_3 ,
celo:potassium_channel_eq_2 ;
      celo:hasInterfaceVariable
              celo:potassium_channel_V , celo:potassium_channel_n ,
celo:potassium_channel_time , celo:potassium_channel_i_K ;
      celo:hasLocalVariable
              celo:potassium_channel_g_K1 , celo:potassium_channel_g_K2 ;
      celo:hasName "potassium_channel"^^xsd:string .

celo:leakage_current_E_L
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:LocalVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "current"^^xsd:string , "L"^^xsd:string ,
"leakage"^^xsd:string , "E"^^xsd:string , "-"^^xsd:string ;
      celo:hasInitialValue
              "-60"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "E_L"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt .

celo:sodium_channel_m_gate_beta_m
```

```
      rdf:type celo:LocalVariable , celo:AlgebricVariable , celo:IonChannelGate ;
      celo:hasDetail "beta"^^xsd:string , "per_second"^^xsd:string , "m"^^xsd:string ,
"gate"^^xsd:string , "channel"^^xsd:string , "sodium"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Gate , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasName "beta_m"^^xsd:string ;
      celo:hasVariableUnit
              celo:per_second .

celo:environment_time
      rdf:type celo:ModelParameterIn , celo:ParameterVariable , celo:IndependentVariable
, celo:InterfaceVariable ;
      celo:hasDetail "second"^^xsd:string , "private_"^^xsd:string , "time"^^xsd:string
, "environment"^^xsd:string , "-"^^xsd:string , "public_out"^^xsd:string ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:Time ;
      celo:hasName "time"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:second ;
      celo:isInterfaceVariableOf
              celo:environment .

celo:membrane_V
      rdf:type celo:ModelParameterOut , celo:ActionPotential , celo:DependentVariable ,
celo:InterfaceVariable ;
      celo:hasDetail "millivolt"^^xsd:string , "private_"^^xsd:string , "V"^^xsd:string
, "membrane"^^xsd:string , "-"^^xsd:string , "public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Membrane ;
      celo:hasInitialValue
              "-87"^^xsd:string ;
      celo:hasMeasure celo:Voltage ;
      celo:hasName "V"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:millivolt , celo:millivolt ;
      celo:isInterfaceVariableOf
              celo:membrane .

celo:sodium_channel_m_gate_eq_3
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\napply > \n\n
<\ndiff > \n</\ndiff > \n <\nbvar > \n\n<\nci > \nsodium_channel_m_gate_\ntime</\nci >
\n</\nbvar > \n <\nci > \nsodium_channel_m_gate_\nm</\nci > \n</\napply > \n <\napply >
\n\n               <\nminus > \n</\nminus > \n <\napply > \n\n<\ntimes > \n</\ntimes >
\n <\nci > \nsodium_channel_m_gate_\nalpha_m</\nci > \n <\napply > \n\n<\nminus >
\n</\nminus > \n <\ncn cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n <\nci >
\nsodium_channel_m_gate_\nm</\nci > \n</\napply > \n</\napply > \n <\napply >
\n\n<\ntimes > \n</\ntimes > \n <\nci > \nsodium_channel_m_gate_\nbeta_m</\nci > \n
<\nci > \nsodium_channel_m_gate_\nm</\nci > \n</\napply > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:_ModelProfile
      rdf:type celo:ModelProfile .

celo:membrane_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\napply > \n\n
<\ndiff > \n</\ndiff > \n <\nbvar > \n\n<\nci > \nmembrane_\ntime</\nci > \n</\nbvar >
\n <\nci > \nmembrane_\nV</\nci > \n</\napply > \n <\napply > \n\n
<\ndivide > \n</\ndivide > \n <\napply > \n\n<\nminus > \n</\nminus > \n <\napply >
\n\n<\nplus > \n</\nplus > \n <\nci > \nmembrane_\ni_Na</\nci > \n <\nci >
\nmembrane_\ni_K</\nci > \n <\nci > \nmembrane_\ni_Leak</\nci > \n</\napply >
\n</\napply > \n <\nci > \nmembrane_\nCm</\nci > \n</\napply > \n\n</apply>\n
"^^xsd:string .

celo:microS
      rdf:type celo:UserDefinedUnit ;
```

```
      celo:hasName "microS"^^xsd:string .

celo:potassium_channel_n_gate_eq_1
      rdf:type celo:Equation ;
      celo:hasMath "\n<apply>\n <\neq > \n</\neq > \n <\nci >
\npotassium_channel_n_gate_\nalpha_n</\nci > \n <\napply > \n\n                <\ndivide
> \n</\ndivide > \n <\napply > \n\n<\ntimes > \n</\ntimes > \n <\ncn
cellml:units=\"\nper_millivolt_second\" \n > \n0.1</\ncn > \n <\napply > \n\n<\nminus >
\n</\nminus > \n <\napply > \n\n   <\nminus > \n</\nminus > \n <\nci >
\npotassium_channel_n_gate_\nV</\nci > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n50</\ncn > \n</\napply > \n</\napply > \n <\napply >
\n\n<\nminus > \n</\nminus > \n <\napply > \n\n<\nexp > \n</\nexp > \n <\napply > \n\n
<\ndivide > \n</\ndivide > \n <\napply > \n\n<\nminus > \n</\nminus > \n <\napply > \n\n
<\nminus > \n</\nminus > \n <\nci > \npotassium_channel_n_gate_\nV</\nci > \n</\napply >
\n <\ncn cellml:units=\"\nmillivolt\" \n > \n50</\ncn > \n</\napply > \n <\ncn
cellml:units=\"\nmillivolt\" \n > \n10</\ncn > \n</\napply > \n</\napply > \n <\ncn
cellml:units=\"\ndimensionless\" \n > \n1</\ncn > \n</\napply > \n</\napply >
\n\n</apply>\n        "^^xsd:string .

celo:sodium_channel_i_Na
      rdf:type celo:SodiumIonChannel , celo:InterfaceVariable ;
      celo:hasDetail "private_"^^xsd:string , "i"^^xsd:string , "Na"^^xsd:string ,
"nanoA"^^xsd:string , "sodium"^^xsd:string , "channel"^^xsd:string , "-"^^xsd:string ,
"public_out"^^xsd:string ;
      celo:hasDomainEntity
              celo:Sodium , celo:Channel ;
      celo:hasInitialValue
              "-"^^xsd:string ;
      celo:hasMeasure celo:EletricCurrent ;
      celo:hasName "i_Na"^^xsd:string ;
      celo:hasPublicInterface
              "out"^^xsd:string ;
      celo:hasVariableUnit
              celo:nanoA , celo:nanoA ;
      celo:isInterfaceVariableOf
              celo:sodium_channel .

celo:microF
      rdf:type celo:UserDefinedUnit ;
      celo:hasName "microF"^^xsd:string .
```