

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

Carla Adriana Barvinski Zanchett

**METAMODELAGEM MOF E SUA
APLICAÇÃO PARA MODELAGEM DE
SISTEMAS IMUNOLÓGICOS ARTIFICIAIS**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. João Bosco Mangueira Sobral

Florianópolis, Maio de 2005

METAMODELAGEM MOF E SUA APLICAÇÃO PARA MODELAGEM DE SISTEMAS IMUNOLÓGICOS ARTIFICIAIS

Carla Adriana Barvinski Zanchett

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação

Prof. Dr. Raul Sidnei Wazlawick

**Coordenador do Curso de Pós-Graduação
em Ciências da Computação**

Prof. Dr. João Bosco Manguiera Sobral
(Orientador)

Banca Examinadora:

Prof. Dra. Nina Krahe Edelweiss
Instituto de Informática - UFRGS

Prof. Dr. Ricardo Pereira e Silva
Departamento de informática e Estatística - UFSC

Prof. Dr. Luiz Carlos Zancanella
Departamento de Informática e Estatística - UFSC

“...E à ciência temperança, e à temperança paciência e à paciência piedade.”

II Pedro, 1:6

Dedico esta dissertação a todos aqueles que de forma voluntária ou involuntária me incentivaram a prosseguir e a superar os obstáculos externos e as minhas próprias limitações.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. João Bosco Manguiera Sobral, pelo apoio, dedicação e confiança, fundamentais para a elaboração deste trabalho.

Ao Prof. Dr. Ricardo Raul Sidnei Wazlawick, Coordenador do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina, extensivo a todo o seu Corpo Docente.

Aos professores membros da banca examinadora: Prof. Dr. Ricardo Pereira e Silva, Prof. Dr. Luiz Carlos Zancanella, Profa. Dra.

Aos meus colegas Profs. Renato Bobsin Machado e Hélio Lopes dos Santos, pelo incentivo e colaboração para o desenvolvimento deste estudo.

À minha mãe Profa. Stella Maria Moreira Barvinski, pelas inúmeras correções do trabalho, apesar das limitações impostas pela saúde debilitada.

Aos professores Manoel dos Passos e Claídes Rejane Schneider pela revisão final e;

à minha família: marido e filhas, inclusive Daiane, que tanto me auxiliaram na superação deste desafio.

Sumário

| | |
|---|----------|
| Lista de Figuras | x |
| Lista de Abreviaturas | xi |
| Resumo | xii |
| Abstract | xiii |
| 1 INTRODUÇÃO | 1 |
| 2 O PARADIGMA DE METAMODELAGEM E O MOF | 4 |
| 2.1 Metamodelagem | 4 |
| 2.1.1 Arquiteturas de Metamodelagem | 6 |
| 2.1.2 Benefícios da Metamodelagem | 7 |
| 2.1.3 A estrutura de Metalinguagens | 9 |
| 2.1.3.1 Sintaxe Abstrata | 9 |
| 2.1.3.2 Sintaxe Concreta | 10 |
| 2.1.3.3 Semântica | 10 |
| 2.1.4 A construção de metamodelos | 11 |
| 2.1.5 Metalinguagens Orientadas a Objeto | 11 |
| 2.2 O <i>Meta-Object Facility</i> | 13 |
| 2.2.1 O MOF Aplicado ao Desenvolvimento de Software | 14 |
| 2.2.2 O MOF Aplicado ao Gerenciamento da Informação | 16 |

| | | |
|---------|--|----|
| 2.2.3 | A Arquitetura do MOF | 16 |
| 2.2.4 | Os Construtores do MOF | 18 |
| 2.2.4.1 | Classes | 18 |
| 2.2.4.2 | Associações | 19 |
| 2.2.4.3 | Tipo de Dados - <i>Data Types</i> | 20 |
| 2.2.4.4 | Pacotes | 20 |
| 2.2.5 | A Estrutura do Modelo MOF | 22 |
| 2.2.5.1 | Principais Classes Abstratas do Modelo MOF | 23 |
| 2.2.5.2 | Os Relacionamentos do Modelo MOF | 26 |
| 2.2.5.3 | Os Tipos de Dados do Modelo MOF | 27 |
| 2.2.5.4 | As Exceções do Modelo MOF | 28 |
| 2.2.5.5 | Constantes do Modelo MOF | 28 |
| 2.2.5.6 | Restrições do Modelo MOF | 29 |
| 2.2.5.7 | O Pacote de Tipos Primitivos | 30 |
| 2.2.6 | O Mapeamento, As Interfaces e Metaobjetos MOF | 30 |
| 2.2.6.1 | Mapeamento entre níveis do MOF | 30 |
| 2.2.6.2 | Composições de Pacotes e o Mapeamento no Modelo MOF | 33 |
| 2.2.6.3 | Domínio de instâncias M1 - Extensões | 34 |
| 2.2.6.4 | Interfaces MOF | 35 |
| 2.2.6.5 | Interfaces <i>Reflective</i> de MOF | 36 |
| 2.2.6.6 | O Mapeamento MOF-CORBA | 38 |
| 2.2.6.7 | O Mapeamento MOF para Java | 39 |
| 2.2.7 | MOF e a UML | 39 |
| 2.2.8 | Ferramentas MOF | 41 |

| | | |
|----------|--|-----------|
| 3 | SISTEMAS IMUNOLÓGICOS | 42 |
| 3.1 | Sistema Imunológico Humano | 42 |
| 3.1.1 | Imunologia | 42 |
| 3.1.2 | <i>Self</i> e <i>NonSelf</i> | 44 |
| 3.1.2.1 | Genes e Características <i>Self</i> | 45 |
| 3.1.3 | Anatomia do Sistema Imunológico Humano | 45 |
| 3.1.3.1 | Linfócitos | 46 |
| 3.1.3.2 | Células-B e Anticorpos | 47 |
| 3.1.3.3 | Células-T e linfocinas | 48 |
| 3.1.3.4 | Fagócitos | 48 |
| 3.1.3.5 | Reconhecimento de Antígenos | 49 |
| 3.1.3.6 | Imunidade Inata e Adquirida | 49 |
| 3.1.4 | Propriedades do Sistema Imunológico | 50 |
| 3.1.4.1 | Detecção | 50 |
| 3.1.4.2 | Diversidade | 50 |
| 3.1.4.3 | Aprendizado | 51 |
| 3.1.4.4 | Tolerância | 52 |
| 3.2 | O Sistema Imunológico Artificial | 52 |
| 3.2.1 | Arquitetura Para um Sistema Imunológico Artificial | 54 |
| 4 | METAMODELO MOF PARA UM MODELO DE SISTEMA IMUNOLÓGICO ARTIFICIAL | 57 |
| 4.1 | Convenção Utilizada Na Representação Do Metamodelo | 57 |
| 4.2 | As Classes do Metamodelo SIA | 59 |
| 4.2.1 | SIAElemento | 59 |
| 4.2.2 | Computador | 59 |

| | | |
|--|---|-----------|
| 4.2.3 | Serviço | 59 |
| 4.2.4 | Elemento | 60 |
| 4.2.5 | RegistroDeLogs | 60 |
| 4.2.6 | ItemDeLog | 61 |
| 4.2.7 | AnalizadorDeEventos | 61 |
| 4.2.8 | Evento | 61 |
| 4.2.9 | Agente | 61 |
| 4.2.10 | Memória | 62 |
| 4.3 | Em Direção à Validação do Metamodelo | 62 |
| 4.4 | Considerações sobre a Metamodelagem MOF | 63 |
| CONCLUSÕES E PERSPECTIVAS | | 65 |
| Anexo A – Classes Do Modelo MOF | | 67 |
| Anexo B – Relacionamentos Do Modelo MOF | | 72 |
| Apêndice A – Classes do Metamodelo SIA | | 75 |
| A.1 | SIAElemento | 75 |
| A.2 | Computador | 76 |
| A.3 | Serviço | 76 |
| A.4 | Elemento | 77 |
| A.5 | RegistroDeLogs | 78 |
| A.6 | ItemDeLog | 79 |
| A.7 | AnalizadorDeEventos | 80 |
| A.8 | Evento | 81 |
| A.9 | Agente | 82 |
| A.10 | Memória | 83 |

| | |
|----------------------------------|-----------|
| A.11 Ataque | 84 |
| A.12 Violação | 85 |
| A.13 Ignorada | 85 |
| A.14 ViolaçãoIgnorada | 85 |
| A.15 EventoDeViolação | 86 |
| A.16 EventoDeAtaque | 86 |
| A.17 EventoDeSegurança | 86 |
| A.18 AgenteEstático | 86 |
| A.19 AgenteMóvel | 87 |
| A.20 AgenteReativo | 88 |
| A.21 AgentePassivo | 89 |
| | |
| Apêndice B – Glossário | 91 |
| | |
| Referências | 94 |
| | |
| Índice Remissivo | 99 |

Lista de Figuras

| | | |
|---|--|----|
| 1 | Arquiteturas de metamodelagem | 6 |
| 2 | O Modelo MOF. Fonte: (OMG, 2002a) | 22 |
| 3 | O <i>NameSpace</i> e suas subclasses. Fonte: (OMG, 2002a) | 24 |
| 4 | O <i>TypedElement</i> e elementos tipados. Fonte: (OMG, 2002a) | 26 |
| 5 | O Pacote <i>Reflective</i> . Fonte: (OMG, 2002a) | 37 |
| 6 | Exemplo de metamodelo MOF. Adaptado de OMG (2002a) | 38 |
| 7 | Metamodelo SIA. | 60 |
| 8 | O relacionamento <i>RefersTo</i> | 73 |

Lista de Abreviaturas

| | | |
|-------|-------|--|
| API | | <i>Application Programming Interface</i> |
| CASE | | <i>Computer Aided Software Engineering</i> |
| CORBA | | <i>Common Object Request Broker Architecture</i> |
| CWM | | <i>Common Warehouse Metamodel</i> |
| EMF | | <i>Eclipse Modelling Framework</i> |
| IDE | | <i>Integrated Development Environment</i> |
| IDL | | <i>Interface Definition Language</i> |
| JMI | | <i>Java Metadata Interface</i> |
| MDA | | <i>Model Driven Architecture</i> |
| MHC | | <i>Major Histocompatibility Complex</i> |
| MIC | | <i>Model Integrated Computing</i> |
| MOF | | <i>Meta-Object Facility</i> |
| OCL | | <i>Object Constraint Language</i> |
| OMG | | <i>Object Management Group</i> |
| OO | | <i>Orientado a Objetos</i> |
| SIA | | <i>Sistema Inmunológico Artificial</i> |
| UML | | <i>Unified Modeling Language</i> |
| XMI | | <i>XML Metadata Interchange</i> |
| XML | | <i>eXtensible Markup Language</i> |

Resumo

Este trabalho versa sobre metamodelagem com o *Meta-Object Facility* (MOF), linguagem padrão para metamodelagem proposta pelo *Object Management Group* (OMG), e sua aplicação na concepção de um metamodelo para um Sistema Imunológico Artificial (SIA). O MOF é uma linguagem abstrata, tecnologicamente neutra, utilizada para a criação de metamodelos expansíveis, portáteis, interoperáveis e independentes de plataforma. Seus metamodelos viabilizam à transformação automática de modelos para diferentes plataformas de desenvolvimento, o que acelera o processo de desenvolvimento de softwares ao mesmo tempo em que reduz custos. Seus conceitos foram aplicados na criação de um metamodelo para o Sistema Imunológico Artificial, abordagem freqüentemente utilizada em pesquisas em segurança computacional em redes de computadores. O metamodelo gerado é genérico, o que torna-o aplicável não somente na concepção de sistemas de detecção de intrusão em redes de computadores mas, também, em outros nos quais os princípios definidos para um sistema imunológico artificial se faça adequada.

Palavras-chave: metamodelagem, metamodelo, MOF, SIA, MDA.

Abstract

This work is about meta-modeling with the Meta-Object Facility (MOF), the standard language for meta-modeling, proposed by the Object Management Group (OMG) and its application on the conception of a meta-model to an Artificial Immune System (AIS). MOF is an abstract language, which is neutral; it is used for the creation of expanded meta-models, portable, interactive and independent in the platform. Their meta-models make it easier the automatic transformation of models into different developed platforms, which accelerates the process of software development, at the same time that reduces the costs. Their concepts were applied in the creation of a meta-model for an Artificial Immune System (AIS), an approach often used in researches of computing security in computer networks. The created meta-model is generic, what makes it applied not only in the conception of invaded detection systems in computer networks, but also in other ones, in which the defined principles for an Artificial Immune Systems are fitted.

Key-words: meta-modelling, meta-model, MOF, AIS, MDA.

1 INTRODUÇÃO

Impulsionado por um mercado que busca na Tecnologia da Informação (TI) recursos que proporcionem diferenciais de competitividade aguçada, a área de desenvolvimento de software tem-se deparado com uma complexidade crescente, cujas atuais tecnologias e conseqüentes ferramentas são, algumas vezes, inadequadas ou insuficientes. As recentes tendências (OMG, 2002b), (DAVIS, 2002), (FLATER, 2002), (FRANKEL, 2003) na área de desenvolvimento de software apontam para o crescimento cada vez maior da construção de tecnologias de software baseadas em metamodelos, ou seja, modelos que descrevem outros modelos. Em virtude da abstração que lhe é inerente, vislumbra-se que eles possam contribuir significativamente para a equação das dificuldades resultantes do atual cenário de desenvolvimento de software.

Dão ênfase a esta tendência as abordagens *Model Integrated Computing* (MIC) (DAVIS, 2002), (LEDECZI, 1999) e *Model Driven Architecture* (MDA) , (OMG, 2002b) (FRANKEL, 2003), as quais têm no uso de metamodelos, o seu princípio. As duas sugerem que o processo de desenvolvimento de software se focalize na elaboração de modelos independentes de plataforma. Posteriormente, estes modelos seriam transformados através de ferramentas CASE (*Computer Aided Software Engineer*) para plataformas específicas.

Segundo FRANKEL (2003) e ATKINSON (2002), a metamodelagem propicia uma elevação na produtividade do desenvolvimento de software; favorece a integração de aplicações; soluciona os problemas de intercâmbio e o reuso de modelos entre ferramentas; contribui para a interoperabilidade entre aplicações. Além disso, auxilia no gerenciamento e no uso de metadados (OMG, 2002a) (SANTOS, 2003).

No contexto da Orientação a Objetos (OO), há várias linguagens para metamodelagem (ULRICH, 2002), (LEDECZI, 1999), (GERBER; RAYMOND, 2003), (OMG,

2002a), em sua maioria proprietárias. Contudo, o uso de uma metalinguagem padrão tanto conduziria às vantagens acima descritas, quanto evitaria a repetição dos fatos vivenciados nas décadas de 80 e 90, quando eclodiram inúmeros métodos de modelagem OO cujas divergências somente foram superadas pela unificação de conceitos em uma linguagem de modelagem unificada nomeada *Unified Modeling Language* (UML) (FURLAN, 1998).

Por essa razão (BEZIVIN, 2001), a *Object Management Group* (OMG) propôs o *Meta-Object Facility* (MOF) (OMG, 2002a) como metalinguagem padrão para a geração de metamodelos. O MOF disponibiliza uma linguagem abstrata, em tecnologia neutra, para criação e gerenciamento de metamodelos e propõe *frameworks* para especificação, construção e implementação de repositórios de metadados (OMG, 2002a).

O MOF, no contexto desse trabalho, será utilizado para definir um metamodelo para o sistema imunológico artificial.

Nas últimas décadas, analogias do sistema imunológico humano tem servido de inspiração para a proposição de soluções computacionais variadas que vão desde a robótica até a problemas relacionados com a segurança de redes de computadores.

Baseados nos conceitos da imunologia foram propostos diversos modelos arquiteturais (SOMAYAJI; HOFMEYR; FORREST, 1997) definindo estruturas e ações ideais para um sistema imunológico artificial. Dentre várias aplicações utilizando esses princípios estão os trabalhos desenvolvidos por pesquisadores da Universidade Federal de Santa Catarina (UFSC) (JUCA, 2001), (JUCA; BOUKERCHE; SOBRAL, 2002), (JUCA et al., 2003), (MACHADO, 2005) (MACHADO et al., 2005) que utilizaram tais conceitos na concepção de um sistema de detecção de intrusão em uma rede de computadores. Diante da constante recorrência ao tema propomos a elaboração de um metamodelo para um sistema imunológico artificial, atuando como inteligência computacional usada no desenvolvimento de sistemas de segurança de detecção de intrusão em redes de computadores.

Este trabalho teve como objetivo construir um metamodelo, baseado em MOF, para um Sistema Imunológico Artificial aplicável no contexto de redes de computadores.

Os objetivos específicos deste trabalho foram os seguintes:

- Realizar um estudo sobre metamodelagem adquirindo conhecimentos sobre o tema;
- Elencar as principais linguagens de metamodelagem orientadas a objetos;
- Efetuar um estudo da linguagem de metamodelagem MOF;
- Realizar revisão bibliográfica sobre o Sistema Imunológico Humano e o Sistema Imunológico Artificial estabelecendo a base para a construção de um metamodelo;
- Projetar um metamodelo para um Sistema Imunológico Artificial utilizando o MOF;
- Avaliar o metamodelo proposto.

Este trabalho está estruturado em cinco capítulos distribuídos da forma a seguir:

O capítulo 2 discorre sobre metamodelagem, atividades e recursos afins, propiciando o embasamento para o desenvolvimento de metamodelos; fornece também, uma visão geral do MOF, sua arquitetura, áreas de aplicação, sua fundamentação teórica, juntamente com a descrição dos principais construtores da metalinguagem, a estrutura do Modelo MOF com suas classes, relacionamentos e tipos de dados; além de abordar as formas de mapeamentos do MOF, suas interfaces e metaobjetos, mostrando como ele é mapeado para outras tecnologias.

A descrição dos Sistemas Imunológicos Humano e Artificial, que embasa a construção de um metamodelo MOF inspirado no sistema imunológico caracteriza o capítulo 3.

No capítulo 4 aplicamos os conceitos de metamodelagem MOF na criação de um metamodelo para o Sistema Imunológico Artificial e validamos o metamodelo.

E, por fim, são apresentados as conclusões deste trabalho e perspectivas de trabalhos futuros.

2 O PARADIGMA DE METAMODELAGEM E O MOF

O uso de metamodelos é a base dos paradigmas de desenvolvimento orientado a modelos. Com este intuito, provemos neste capítulo conceituação acerca da metamodelagem de forma a embasar o desenvolvimento do trabalho, enquanto realizamos uma definição do MOF, sua arquitetura, áreas de aplicações e sua fundamentação teórica, juntamente com a descrição de seus principais construtores.

Também, para entender o Modelo MOF e sua semântica, abordamos as convenções utilizadas na definição dos elementos do modelo juntamente com suas características e exceções, bem como, fornecemos a descrição do mapeamento dos metamodelos MOF para tecnologias de implementação juntamente com as interfaces reflexivas do MOF.

Aliado a isso, objetivando o melhor entendimento do trabalho, utilizamos diagramas UML para ilustrar e exemplificar alguns dos conteúdos.

2.1 Metamodelagem

Na computação o significado do termo "META" varia de acordo com a forma com que é empregado. Em banco de dados o termo metadados significa "dados sobre dados" (OMG, 2002a) e pode se referir a dicionários de dados, a descritores de tabelas ou repositório de dados. E, na programação, o termo metalinguagem se refere a uma linguagem que descreve a sintaxe de outra linguagem (SEBESTA, 2003).

Neste trabalho, o prefixo "META" sempre expressa uma descrição de algo, ou seja, metamodelagem é o ato de elaborar um metamodelo. Um metamodelo é um

modelo que descreve outro modelo e uma metalinguagem é uma linguagem utilizada para criar uma linguagem de modelagem que, segundo BOOCH et al.(2000), é uma das técnicas científicas mais difundidas para descrição de sistemas, em seu comportamento e estrutura.

A finalidade da modelagem é a representação abstrata de conceitos pertinentes a um determinado domínio. As linguagens de modelagem associam descrições textuais com uma sintaxe gráfica, como é o caso da *Unified Modeling Language* (UML)(OMG, 2003a), por exemplo.

A modelagem é estreitamente relacionada com metamodelagem, pois esta última é quem define os conceitos que farão parte da linguagem de modelagem. Em outras palavras, a metamodelagem está a um nível acima da modelagem e seu propósito é definir uma linguagem de modelagem. Os artefatos da atividade de modelagem e metamodelagem são respectivamente modelos e metamodelos. Enquanto um modelo descreve a semântica, aspectos físicos e visuais da análise, projeto ou implementação de um sistema, através da representação de conceitos do mundo real (BOOCH; JACOBSON; RUMBAUGH, 2000), os metamodelos declaram os elementos de uma linguagem de modelagem (FRANKEL, 2003) (KLEPPE; WARMER; BAST, 2003).

Segundo MELLOR et al.(2004, p.44), "um metamodelo pode ser considerado como um modelo cujas instâncias são tipos em outros modelos". Assim, verifica-se que todo tipo utilizado na representação dos conceitos de um domínio, em um modelo deve estar definido no metamodelo da linguagem utilizada. Essa relação influencia na modelagem pois o metamodelo determina as características da linguagem de modelagem. Dessa forma, se desejamos modelar objetos, o metamodelo da linguagem de modelagem selecionada para a tarefa deve conter definições para os conceitos de classe, atributo, operação, relacionamentos, os quais são inerentes ao paradigma de orientação a objetos. Conseqüentemente, o vocabulário e regras representados em um modelo são em essência, o instanciamento dos tipos pertencentes ao metamodelo da linguagem de modelagem utilizada. Por isso, se houver a necessidade de representar no modelo um novo tipo de objeto, este deve ser incorporado ao metamodelo para que possa ser utilizado (ODELL, 1998), (KLEPPE; WARMER; BAST, 2003). A definição contida em um metamodelo abrange completamente a linguagem, o que inclui a sua sintaxe e semântica.

2.1.1 Arquiteturas de Metamodelagem

De acordo com SEBESTA (2003), denomina-se metalinguagem a linguagem utilizada para a descrição de outras linguagens. Uma metalinguagem é descrita por um metamodelo. Um metamodelo é autodescritivo, ou seja, ele representa a si mesmo. Este aspecto lhe concede um alto poder de abstração e provê capacidade de descrever quaisquer tipos de metalinguagens. O relacionamento modelo, metamodelo, metamodelo compõem uma arquitetura de metamodelagem.

Há duas abordagens para a composição de arquiteturas para metamodelagem. Uma que utiliza três camadas e a outra que distribui objetos, modelos, metamodelos e metamodelos em quatro camadas. São elas as arquiteturas de metamodelagem de três camadas e quatro camadas. Estas arquiteturas são ilustradas pela Figura 1.

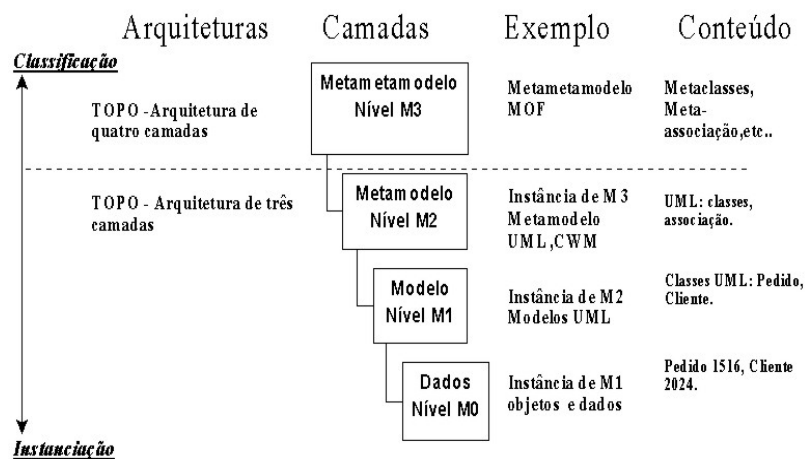


Figura 1: Arquiteturas de metamodelagem

Observa-se que cada uma das camadas superiores contém informações referentes ao seu nível inferior imediato conforme descrito a seguir:

1. A camada mais inferior, denominada M0, de ambas as arquiteturas contém ou dados dos objetos em tempo de execução ou elementos originais do sistema representado;

2. Na camada acima, chamada M1, está o modelo que define os objetos instanciados por M0;
3. O nível posterior, nomeado M2, contém o metamodelo que descreve a linguagem de modelagem utilizada para elaborar o modelo. Este é o nível topo para a arquitetura de três camadas;
4. O quarto nível, denominado M3, existente somente na arquitetura de quatro camadas, contém o metametamodelo, que descreve a metalinguagem utilizada para definir o metamodelo (nível M2).

Ressalta-se que, a definição da quantidade de camadas utilizado para compor uma arquitetura de metamodelagem depende do nível de abstração fornecido pelo topo da hierarquia (KARAGIANNIS; KUHN, 2002) (FRANKEL, 2003).

O nível de abstração requerido pelos atuais sistemas, é suprido pelo modelo de quatro camadas (OMG, 2002a) (ULRICH, 2002) (SPRINKLE et al., 2001) que disponibiliza uma infra-estrutura abstrata altamente flexível que propicia um ambiente de modelagem configurável a uma ampla variedade de domínios. Outra vantagem dessa arquitetura são seus *frameworks* que disponibilizam recursos avançados para a interoperabilidade entre ferramentas CASE (*Computer Aided Software Engineering*) (NORDSTROM, 1999).

Além disso, outro fator que favorece a arquitetura de quatro camadas é a composição do núcleo de um metametamodelo, que normalmente contém uma quantidade reduzida de conceitos que, por serem poucos, tendem a permanecer estáveis facilitando o intercâmbio de modelos e a integração de modelos.

2.1.2 Benefícios da Metamodelagem

O uso de metamodelos no desenvolvimento de software produz benefícios nesta área que inclui melhoramentos na produtividade, facilidades para a portabilidade de modelos entre ferramentas e para a interoperabilidade e integração de aplicações, além de aumentar o ciclo de vida de softwares, pois favorece sua manutenção e documentação.

Estudos realizados pelo desenvolvedor MetaCASE (METACASE, 2004), baseado em experiências vivenciadas pelas empresas Nokia, Lucent e US Air Force na uti-

lização de métodos para domínios específicos, outra denominação dada a abordagem de desenvolvimento orientado a modelos, concluiu que adotar a abordagem de desenvolver modelos para domínios específicos, principalmente aqueles que tratam de soluções para famílias de produtos, tais como, aplicações para celulares e softwares embarcados, eleva de modo expressivo a produtividade do desenvolvimento de software, pois que a solução é desenvolvida uma única vez. Isto se explica porque um modelo concebido independente de plataforma deve representar apenas os conceitos do domínio modelado, abstendo-se de conter conceitos que o tornem dependente de quaisquer tecnologias (Java, XMI, .Net, etc...), o que torna a atividade mais rápida (KLEPPE; WARMER; BAST, 2003).

Esta isenção permite aos desenvolvedores se concentrar na representação dos aspectos relevantes dos objetos que compõem o problema, sem preocupações na implementação da solução. O resultado é uma solução única especificada em uma terminologia familiar a todos os *stakeholders*, que poderá ser implementada em inúmeras aplicações, em tecnologias variadas, de acordo com as necessidades dos usuários. Todavia, há que se ressaltar que os ganhos em produtividade só serão possíveis mediante o uso de ferramentas que agreguem a um modelo independente de plataforma as informações necessárias para que este funcione em uma plataforma específica. Sem a automatização das transformações a metamodelagem em muito pouco contribui para a produtividade no desenvolvimento de software.

A manutenção e documentação de sistemas também são algumas das atividades que podem ser favorecidas em um ambiente de metamodelagem (ODELL, 1998), (KLEPPE; WARMER; BAST, 2003). Nos métodos que incorporam esta prática, as alterações no modelo ocorre de uma forma coerente e integrada, uma vez que os modelos não estão fragmentados em níveis (ODELL, 1998). Isso permite manter a documentação do sistema atualizada, bem como agiliza a liberação de novas versões de software.

Para CLARK et al.(2004), afirmação corroborada por FRANKEL (2003), a presença de metamodelos no âmbito do desenvolvimento de ferramentas é particularmente importante. Para eles, o acesso aos metadados, contidos no metamodelo, fornece às ferramentas informações relevantes sobre forma de apresentação e o armazenamento dos modelos ou programas nesta linguagem. Eles também facilitam a interação entre os usuários e ferramentas enquanto subsidiam a realização

de atividades como a análise, testes, entre outras. O fato de inúmeras linguagens compartilharem uma mesma base conceitual, habilitaria a construção de ambientes de desenvolvimento semanticamente expressivos. Este mesmo fator fundamentaria a interoperabilidade entre aplicações, em que modelos para plataformas específicas teriam no modelo independente de plataforma os elementos necessários para transformar os conceitos oriundos de uma plataforma em conceitos para uma outra plataforma. Um exemplo de aplicabilidade de metamodelos na interoperabilidade entre aplicações na área de telefonia pode ser obtida através da referência (FARSHCHIAN; JAKOBSSON; BERG, 2002).

2.1.3 A estrutura de Metalinguagens

Todas as linguagens independente de paradigma, aplicação ou escopo, compartilham alguns aspectos comuns que são uma sintaxe abstrata (essencial), uma sintaxe concreta e uma semântica (FRANKEL, 2003) (SEBESTA, 2003) (CLARK et al., 2004). A sintaxe de uma metalinguagem tem por objetivo definir os conceitos, ou tipos, utilizados para construir os modelos, os relacionamentos que poderão existir entre os conceitos, juntamente com as regras concernentes à construção de modelos.

Em particular, a sintaxe abstrata de um metamodelo define que elementos pertencerão à linguagem enquanto que a sintaxe concreta estabelece uma notação para esses tipos. Já a semântica especifica condições invariantes que asseguram a boa formação dos construtores da linguagem (NORDSTROM, 1999).

Estes conceitos são abordados nas tópicos seguintes.

2.1.3.1 Sintaxe Abstrata

A especificação desta sintaxe é fundamental para a construção de linguagens de modelagem, pois que ela descreve os tipos, os relacionamentos, as regras de validação dos modelos derivados de uma determinada linguagem de modelagem. A partir de uma sintaxe abstrata podem-se derivar inúmeras sintaxes concretas (FRANKEL, 2003), todas interoperáveis e compatíveis entre si.

A sintaxe abstrata, para o nível M2, consiste na especificação do próprio meta-modelo (CLARK et al., 2004) . A ênfase dada na sua elaboração está na definição dos relacionamentos estruturais entre os tipos que a formarão, não havendo preo-

cupações quanto à parte semântica.

Pode-se dizer então, que o foco está em estabelecer os tipos, seus relacionamentos, bem como, regras aplicáveis a cada um.

A especificação da sintaxe abstrata dos metamodelos varia em decorrência de paradigmas e objetivos que se pretende disponibilizar à metalinguagem que ele dará origem. Para tanto, sua definição deve focar na análise do paradigma em que esta se baseará, nos tipos, relacionamentos e regras que ela deve apresentar, verificando se estes permitirão a representação dos conceitos dentro do padrão de desenvolvimento escolhido (CLARK et al., 2004). Por exemplo, para representar conceitos utilizando orientação a objetos a metalinguagem deve possuir os elementos essenciais para a representação de classes, pacotes, relacionamentos, atributos, operações, entre outros.

2.1.3.2 Sintaxe Concreta

A sintaxe concreta, utilizando uma notação, expressa os conceitos da sintaxe abstrata (SEBESTA, 2003) (FRANKEL, 2003). A notação utilizada pode estar na forma textual ou gráfica. O exemplo padrão de sintaxe gráfica orientada a objetos são os diagramas da linguagem UML como, por exemplo, os diagramas de classes, de objetos, de interação, entre outros. Esta modalidade de sintaxe concreta tem como benefício a capacidade de expressar uma grande quantidade de detalhes de uma forma compreensível e intuitiva.

2.1.3.3 Semântica

A semântica tem como finalidade determinar o significado de cada elemento dentro de uma linguagem, estabelecendo como esta será executada e o que acontecerá mediante a execução.

A sua definição demanda descrever a finalidade de cada tipo, relacionamento, restrições, estados e ações pertencentes a sintaxe abstrata da linguagem. Esta atividade requer clareza e precisão de forma a evitar ambigüidades, não dando margem a interpretações errôneas quanto ao uso da linguagem.

A semântica de um metamodelo é descrita através de textos informais e restrições formais.

2.1.4 A construção de metamodelos

O processo de metamodelagem segue os mesmos passos da modelagem tradicional (CLARK et al., 2004), ou seja, envolve a identificação e modelagem de conceitos, a definição da arquitetura, validação e teste do modelo. Segundo Clark et. al (2004), na fase de identificação são detectados os conceitos e regras do domínio, dos quais resultam o vocabulário e parâmetros para a validação do modelo. O vocabulário provê a base para definição da sintaxe abstrata do metamodelo. A descrição da finalidade de cada item que compõem esta sintaxe bem como das regras a ele aplicáveis provê a semântica do metamodelo. Na seqüência, a aplicação de uma notação que expresse os elementos da sintaxe abstrata estabelece uma sintaxe concreta para o metamodelo.

Ao final, a correção do metamodelo deve ser testada através da construção de instâncias da sintaxe abstrata deste.

2.1.5 Metalinguagens Orientadas a Objeto

Expomos a seguir algumas metalinguagens disponíveis na literatura e adequadas a construção de metamodelos orientados a objetos. Dentre as mais atuais estão: o MEMO *Meta-metamodel* (ULRICH, 2002), a MetaGME (LEDECZI, 1999), o *model Ecore* (GERBER; RAYMOND, 2003) , e o *Meta Object Facility* (MOF) (OMG, 2002a), tema central deste trabalho.

O MEMO ("*multi perspective enterprise modelling*") *meta-metamodel* surgiu na Universidade de *Koblenz-Landau* na Alemanha, proposto por Frank (ULRICH, 2002). O MEMO *Meta-metamodel* disponibiliza uma estrutura voltada à criação de metamodelos específicos para atividades empresariais tais como planejamento, desenvolvimento e sistemas de informação corporativos. Dele derivam um conjunto de metamodelos que definem um método para o desenvolvimento de modelos empresariais, também denominado MEMO, cujas abstrações proporcionam análise e projeto de aspectos importantes para o processo de desenvolvimento, tais como, processos de negócio, estrutura organizacional, modelos de informação, entre outros.

O MetaGME (LEDECZI, 1999) (SPRINKLE et al., 2001), especifica o metamodelo do *Generic Modeling Environment* (GME) constituindo-se num conjunto de ferramentas para criação de modelagem para domínios específicos (*Domain-Specific*

Modeling - DSM), criado pelo *Institute for Software Integrated Systems* da Universidade de *Vanderbilt*¹ (E.U.A). Metamodelos para domínios específicos visam atender às necessidades do paradigma *Model Integrated Computing* (MIC)(DAVIS, 2002) (LEDECZI, 1999) cuja finalidade é prover metamodelos para área de *Large Computer Based Systems* (CBS's). Estes são desenvolvidos através da ferramenta GME 2000, versão mais recente do GME.

O GME 2000 utiliza a notação do diagrama de classes *Unified Modeling Language* (UML) (OMG, 2003a), para metamodelagem e a *Object Constraints Language* (OCL) para definir restrições para os metamodelos. Outros conceitos tradicionais da orientação a objeto também são utilizados no MetaGME: os relacionamentos de composição, agregação e hierarquia, associação entre classes atributos literais e numéricos.

O ambiente do GME 2000 pode ser configurado para diferentes paradigmas de modelagem, pois permite ao usuário selecionar o paradigma de modelagem mais adequado às suas necessidades e, a partir disso, a ferramenta disponibiliza os recursos necessários para a criação dos modelos desejados dentro da abordagem selecionada.

O metamodelo *model Ecore* é usado para definir o *Eclipse Modelling Framework* (EMF) (GERBER; RAYMOND, 2003). O EMF é um projeto *open-source* parte integrante do projeto Eclipse, e capacita este último a aplicar o conceito de desenvolvimento orientado a modelos da MDA.

O *model Ecore* foi influenciado pelo MOF e compartilha com este alguns conceitos porém, com um objetivo diferente (GERBER; RAYMOND, 2003), pois o EMF é um *framework* para construir ferramentas e aplicações Java. Ele disponibiliza aos desenvolvedores dessa linguagem os benefícios da modelagem formal (GERBER; RAYMOND, 2003) (POWELL, 2004). Estes modelos podem ser intercambiados entre diferentes ferramentas e aplicações, através da conversão dos objetos para o padrão *XML MetaData Interchange* (XMI)(OMG, 2003b).

Os principais construtores do *model Ecore* são as classes (com seus atributos tipados e operações com parâmetros e exceções), o relacionamento de múltipla herança e pacotes (BUDINSKY et al., 2003) (GERBER; RAYMOND, 2003). O enfoque dado à descrição e construção de modelos para o ambiente Java, tornam o *model*

¹www.isis.vanderbilt.edu/Projects/gme

Ecore uma metalinguagem para uma plataforma específica (GERBER; RAYMOND, 2003), distinguindo-o do MOF cujo propósito é o desenvolvimento de metamodelos tecnologicamente neutros, isto é, independentes de quaisquer plataformas.

2.2 O *Meta-Object Facility*

O MOF é uma tecnologia neutra que define uma linguagem abstrata para metamodelagem e um *framework* para especificação, construção e gerenciamento de metamodelos Orientado a Objetos (OO)(OMG, 2002a).

A especificação MOF inclui os seguintes componentes:

- Uma definição formal de seu metametamodelo, que é a linguagem abstrata usada para especificar metamodelos MOF e a si próprio;
- Um mapeamento dos metamodelos MOF para IDL do *Common Object Request Broker Architecture* (CORBA), resultando em interfaces IDL (*Interface Definition Language*) para o gerenciamento de qualquer tipo de metadado;
- Um conjunto de interfaces reflexivas IDL CORBA para gerenciar metadados independente de metamodelo;
- Um conjunto de interfaces IDL CORBA usado para representar e gerenciar MOF metamodelos;
- Um formato XMI para troca de metamodelos MOF.

Os conjuntos destes elementos dão suporte à criação de padrões e aplicações variados podendo ser utilizado tanto para a criação de metamodelos quanto para o gerenciamento de metadados. Na área da metamodelagem, ele pode ser usado para definir quaisquer tipos de metamodelos para domínios de interesse específicos. Os metamodelos da UML (OMG, 2003a), CWM (*Common Warehouse Metamodel*) (OMG, 2001), XMI (OMG, 2003b), e mesmo o próprio MOF (OMG, 2002a) constituem alguns exemplos dessa aplicabilidade. Suas propriedades de OO permitem a extensão e modificação de metamodelos através da herança e composição, de modo que é possível constituir modelos de informação amplos e diferentes filosoficamente do MOF, ou seja, podem-se criar metamodelos não orientados a objetos a partir do MOF.

Os metadados descritos pelos metamodelos podem ser manipulados através de regras de mapeamento que fazem parte da especificação e que dão condições para a implementação de repositórios MOF. Através do mapeamento, metamodelos MOF de nível M2 podem se relacionar com outros metamodelos, no mesmo nível, ou no nível inferior. A especificação do MOF disponibiliza três formas de mapeamentos (OMG, 2002a):

- Um mapeamento abstrato que traduz um metamodelo MOF em um modelo de informação abstrato;
- Um mapeamento IDL que fornece a IDL padrão da OMG junto com semânticas de comportamento para os metaobjetos do metamodelo. Metaobjeto é uma abreviação de objeto de metadado. Esses metaobjetos representam metadados que, através de APIs, (*Application Programming Interfaces*) resultantes do mapeamento, podem ser gerenciados. As APIs tornam seguros a criação, o acesso e a alteração das informações contidas em um metamodelo, pois garantem a consistência lógica e estrutural, e as restrições do mesmo. O mapeamento do MOF é exclusivo para a API do CORBA (*Common Object Request Broker Architecture*). Esta API capacita um cliente CORBA a obter as descrições das informações contidas no metamodelo e implementar suporte à reflexão. Outros mapeamentos que não se refiram a tecnologias da própria OMG são providenciadas por desenvolvedores das mesmas. Como exemplo, cita-se o mapeamento para Java, provido através do padrão Java *Metadata Interface* (JMI) (JCP, 2002) que define as regras de mapeamento para a API do Java;
- O terceiro mapeamento se refere à produção de um *Document Type Definition* (DTD)/XML (*eXtensible Markup Language*) para o intercâmbio de metadados. Este mapeamento denomina-se Mapeamento XML.

As características do MOF fazem-no aplicável tanto ao desenvolvimento de software, quanto ao gerenciamento de informações e metadados, os quais são abordados na seqüência.

2.2.1 O MOF Aplicado ao Desenvolvimento de Software

Dentro da área de desenvolvimento, o MOF pode ser usado no suporte ao ciclo de vida do desenvolvimento de software em áreas que requeiram o uso de metadados.

Por exemplo, no campo do desenvolvimento de aplicações distribuídas orientadas a objetos, a implementação das interfaces MOF na criação de repositórios de serviços, favorece a construção de ambientes destinados ao armazenamento e gerenciamento de modelos de software (SANTOS, 2003), tais como modelos UML, sem que haja dependência de plataforma. Ressalta-se que as ferramentas utilizam a notação padrão XMI para permitir o intercâmbio de modelos entre si e viabilizar transformações nos modelos. No ambiente de repositório, os modelos criados pelos projetistas em ferramentas, são consultados pelos programadores e implementados a partir do repositório, cuja função é mediar o acesso dos projetistas a ferramentas, editores gráficos e outros recursos necessários à criação e carregamento dos modelos. Caso o repositório disponibilize um gerador de interfaces, poder-se-á automatizar os mapeamentos dos metamodelos para as plataformas CORBA e Java e outras tecnologias que possuam mapeamentos para o MOF.

No segmento de modelagem de sistemas, o MOF contribui com sua flexibilidade, que na visão de HOLZ (2003), permite definir quaisquer tipos de metamodelos independentes do domínio de aplicação, fornecendo um conjunto conciso e único de conceitos para a atividade.

A metamodelagem é importante para o desenvolvimento de aplicativos, pois, cada metamodelo define por si próprio uma linguagem capaz de descrever um domínio de interesse específico, como por exemplo, sistemas legados, *data warehouses*, processos de software, testes ou telecomunicações. Alguns desses domínios não conseguem ser adequadamente representados por linguagens de modelagem de uso geral como a UML, em virtude da insuficiência das mesmas na representação plena de especificidades. Neste contexto, os metamodelos se mostram mais convenientes, pois podem representar quaisquer tipos de conceitos de um dado domínio.

Ainda na modelagem de sistemas, a implementação das interfaces reflexivas do MOF possibilita as ferramentas realizarem operações sobre os objetos do modelo, sem prévio conhecimento deste, bem como, consultas ao repositório para verificação de modelos ali armazenados, e exame aos objetos instanciados.

No projeto *Odyssey* (MURTA, 2004), a ferramenta *Odyssey-VCS* (OLIVEIRA; MURTA; WERNER, 2004) provê versionamento de modelos utilizando-se o MOF e metamodelos derivados (UML e XMI).

2.2.2 O MOF Aplicado ao Gerenciamento da Informação

No âmbito de gerenciamento da informação, o MOF é aplicável tanto no projeto, quanto na implementação e gerenciamento de informações estruturadas, pois pode prover um repositório para os modelos de informação de um dado domínio. O repositório fornece meios para a criação e gerenciamento dos objetos instanciados em tempo de execução. Em sistemas de informação de larga escala, o uso de metadados pode descrever páginas Web armazenadas em banco de dados.

Por exemplo, a técnica de gerenciamento de dados empregada em armazém de dados (*Data Warehouse*), preconiza a extração da informação de bancos de dados lógicos e fisicamente diferentes, integrando e armazenando a informação em um depósito de banco de dados para somente leitura e sem acesso concorrente. Nele, o processo de integração e extração das informações é realizado através do mapeamento de esquemas dos bancos de dados individuais para o esquema do armazém de dados. O papel de metadados neste ambiente é fundamental e o uso de metamodelos como o CMW (OMG, 2001), especificação baseada em MOF para *Data Warehousing*, possibilita a criação de ferramentas de apoio sofisticadas para a atividade cuja capacidade automatizaria a tarefa de gerenciamento de metadados.

Dessa forma, o MOF, neste cenário, proporciona a integração de modelos, em tempo de execução, do armazém de dados com o ambiente de desenvolvimento (SANTOS, 2003).

Abordamos nos itens seguintes os conceitos básicos e as definições relevantes para o entendimento do MOF, juntamente com os aspectos de sua arquitetura conceitual utilizada para descrever metadados, compor metamodelos e realizar mapeamentos.

2.2.3 A Arquitetura do MOF

O modelo arquitetural adotado por MOF (OMG, 2002a) (KLEPPE; WARMER; BAST, 2003), disponibiliza a estrutura para a descrição de metadados, metamodelos e mapeamentos. Na descrição desta arquitetura são utilizados alguns termos cujo entendimento adequado de seu significado é imprescindível para este trabalho, pelo que tecemos a seguir algumas observações referentes à terminologia utilizada.

1. São consideradas camadas metaníveis aquelas que se encontram acima da ca-

mada de informação (nível M0 do modelo arquitetural descrito na seção 2.1.1, página 6). Contudo, não há um número pré-determinado de metaníveis que devam constar em determinado modelo.

2. Os conceitos de modelagem orientado a objetos poderão estar presentes nos diversos níveis de um *framework* metadados MOF, ou seja, desde o nível M1 até M3.
3. O prefixo "meta" possui dentro do contexto de MOF os seguintes significados:
 - Metadados - É o termo usado para referenciar dados cujo propósito é descrever outros dados;
 - Metamodelo - É o termo usado para referenciar um modelo de algum tipo de metadados;
 - Metaobjeto - É o termo usado para referenciar um objeto abstrato ou tecnologia específica representando metadados.
4. Doravante o núcleo de MOF, isto é, o metametamodelo de MOF, será referenciado como Modelo MOF. Igualmente, ao invés de utilizar termos como classe, metaclasses ou ainda metametaclasses, utilizar-se-ão os termos instância de classe do nível M1 ou instância de classe do nível M2.

O MOF é um *framework* extensível planejado para suportar uma ampla variedade de padrões de uso e de aplicações, tanto na área de metamodelagem quanto na de metadados (OMG, 2002a). Sua arquitetura de metamodelagem segue o modelo tradicional de quatro camadas. De acordo com esta abordagem, modelos da camada inferior podem ser vistos como uma instância de algum modelo da camada superior. A estrutura clássica de quatro camadas aplicada à arquitetura de MOF é ilustrada pela Figura 1, página 6.

No topo da arquitetura, nível M3, encontra-se a infra-estrutura para a criação de linguagens de modelagem: o Modelo MOF. Na camada subsequente (nível M2), encontram-se os metamodelos criados a partir do Modelo MOF, dos quais são exemplos os metamodelos da UML, do CWM, e do XMI. No nível M1 estão os modelos instanciados a partir dos metamodelos MOF da camada M2. Os diagramas UML, definições de coluna e tabelas de bancos de dados e interfaces IDL, são alguns dos

exemplos dos elementos dessa camada. A camada mais inferior da arquitetura, nomeada nível M0, é povoada por elementos que são instância dos modelos da camada M1, ou seja, dado um diagrama de classes (M1), encontraremos em M0 os objetos que são o resultado de instanciamento das classes.

Não existem limites para a quantidade de metaníveis dentro da arquitetura de metadados de MOF, podendo existir mais ou menos de quatro metaníveis. O que define o número de metaníveis são as necessidades verificadas pelos desenvolvedores. Por exemplo, se o nível de abstração necessário for obtido utilizando apenas três níveis, esse será o limite de camadas. A quantidade de metaníveis citados na especificação de MOF fazem parte de uma convenção e não determinam uma quantidade mínima ou máxima de metaníveis que podem ser utilizados.

2.2.4 Os Construtores do MOF

O MOF possui uma estrutura de modelagem orientada a objetos formada por quatro elementos principais, a saber:

1. Classes - Modelo de metaobjetos;
2. Associações - modelador de relacionamentos binários entre metaobjetos;
3. Tipos de dados - modelador de dados;
4. Pacotes - o organizador lógico de modelos.

Esses elementos são utilizados na construção de metamodelos, juntamente com regras de utilização. As subseções seguintes descrevem cada um deles.

2.2.4.1 Classes

As classes são o elemento central do MOF. Elas descrevem metaobjetos MOF. Um metaobjeto é uma instância de uma classe definida em um nível superior ao dele. Por exemplo, as classes definidas no nível M2 possuem instâncias, metaobjetos, no nível M1. Um metaobjeto possui uma identidade, estado e comportamento; estes dois últimos, definidos pela metaclassa instanciada.

Uma classe possui atributos, operações e referências, podendo também conter exceções, constantes, tipos de dados, restrições ou outros elementos. Essas características podem ser herdadas de outras classes, inclusive por herança múltipla. O relacionamento entre classes é feito através de associações.

Uma classe MOF é uma especificação abstrata que inclui seus estados, interfaces e comportamento informal, de forma suficiente para propiciar a concepção de interfaces concretas, com semântica bem definida portanto, adequadas ao gerenciamento dos estados dos metaobjetos.

2.2.4.2 Associações

As associações expressam os relacionamentos entre instâncias de classes dos diferentes níveis no MOF. As associações relacionam os modelos com seus respectivos metamodelos. Isto significa que para cada classe nível M1, há uma associação MOF nível M2 definindo relacionamentos (*links*) entre os pares de instâncias das classes.

Conceitualmente, *links* não possuem identidade e, portanto, não contêm atributos ou operações. Um *links* contém duas propriedades denominadas extremidades de associação (*associationEnd*). Uma extremidade de associação é o ponto final de uma associação. Cada extremidade de associação está relacionada a uma classe e suas funções são:

- Definir quais são as classes que participam da associação;
- Definir o papel da associação;
- Estabelecer o número mínimo e máximo de instâncias que cada uma das classes pode ter no relacionamento.

Toda extremidade de associação referencia um dos objetos envolvidos no relacionamento. Além disso, cada uma delas contém informações sobre o nome e tipo da extremidade, sua multiplicidade, características de agregação, configuração de navegabilidade e mutabilidade (OMG, 2002a).

2.2.4.3 Tipo de Dados - *Data Types*

O tipo de dados MOF provê definições para tipos de atributos e valores de parâmetros e operações que são elementos que não possuem identidade de objeto.

Os tipos de dados são classificados em duas categorias:

1. Tipos primitivos - são os tipos elementares como booleano, *string* e inteiro.
2. Tipos construtores (não primitivos) - são definidos pelo metamodelador e permitem a definição de tipos de dados mais complexos. Os tipos dessa categoria classificam-se em: enumeração, estruturado, coleção ou alias.

Os tipos primitivos de MOF objetivam expressar o estado de um objeto e proporcionar recursos para uma tecnologia de metamodelagem neutra.

2.2.4.4 Pacotes

Os pacotes servem para agrupar logicamente elementos dentro do metamodelo. Seus objetivos variam de acordo com o nível em que estão sendo utilizados:

1. No nível M2, os pacotes proporcionam o particionamento de metamodelos através do agrupamento de elementos tais como outros pacotes, classificadores, associações, tipos de dados, exceções, constantes, entre outros;
2. No nível M1, eles atuam como contêineres para metadados provendo o acesso aos objetos, instância do metamodelo.

Há quatro mecanismos que podem ser usados para a composição e reuso de pacotes; são eles (OMG, 2002a):

- Generalização - A herança permite que os pacotes herdem conteúdos de outros pacotes, segundo algumas regras clássicas. Estas regras estabelecem que um pacote (subpacote) herda todos os elementos pertencentes ao pacote-pai (superpacote), observando restrições que previnem a colisão dos nomes herdados com aqueles existentes no subpacote;

- Pacotes Aninhados (*Nesting*) - O aninhamento de pacotes estabelece um mecanismo permitindo que um pacote contenha outro pacote que, por sua vez, contém outro pacote e assim por diante. Como a sua finalidade é prover um recurso de modularização e não de reuso, eles não podem ser diretamente instanciados ou removidos. O instanciamento ou remoção é feito pelo pacote mais externo. Assim sendo, ao criar uma instância do pacote mais externo, é disparado o instanciamento dos pacotes internos e, ao excluir o pacote externo, são eliminados os pacotes internos. Pacotes aninhados não podem ser utilizados pelos mecanismos de generalização, importação ou agrupamento;
- Importação (*Importing*) - A importação fornece uma forma diferenciada para o reuso ou composição de pacotes permitindo a seleção de elementos de um metamodelo. Utilizando-a pode-se compor um pacote importando elementos de vários ao mesmo tempo. Pode-se importar classes, inclusive superclasses, tipos de dados, exceções e associações cujas extremidades de associações pertençam a classes do pacote importado;
- Agrupamento (*Clustering*) - O agrupamento relaciona o pacote importador e o pacote importado englobando-os dentro de pacote maior denominado *Cluster* ou agrupador. Na importação, tanto um pacote pode agrupar vários outros pacotes quanto pode pertencer a vários agrupamentos. As instâncias de um agrupamento se comportam igual aos pacotes aninhados, o que torna o ciclo de vida dos pacotes no agrupamento dependente do pacote agrupador: ao criar uma instância de um agrupador são geradas, automaticamente, instâncias para todos os pacotes que nele estão contidos. Todas as instâncias pertencem ao agrupador, então, a exclusão do mesmo implicará na remoção de todos os pacotes nele agrupados. E é esta a única forma de se remover um agrupamento de pacotes. Todavia, ao contrário do aninhamento de pacotes, pode-se criar uma instância de um pacote sem que seja necessário instanciar o agrupamento.

Além dos construtores acima citados, há outros construtores adicionais como as constantes, as exceções, as restrições e os identificadores (*tags*), os quais fazem parte da estrutura do MOF.

2.2.5 A Estrutura do Modelo MOF

O Modelo MOF, em sua versão 1.4, consiste de um único pacote não aninhado denominado *Model* (OMG, 2002a). Este pacote importa do pacote *PrimitiveType* os tipos de dados primitivos utilizados pelo Modelo MOF. No pacote estão inclusos um conjunto de classes e relacionamentos que conferem ao metamodelo o potencial para metamodelagem. Modelo MOF completo é ilustrado pela Figura 2. O papel das principais classes, relacionamentos, pacotes, tipos de dados, exceções, constantes e restrições no Modelo MOF são abordados na seqüência.

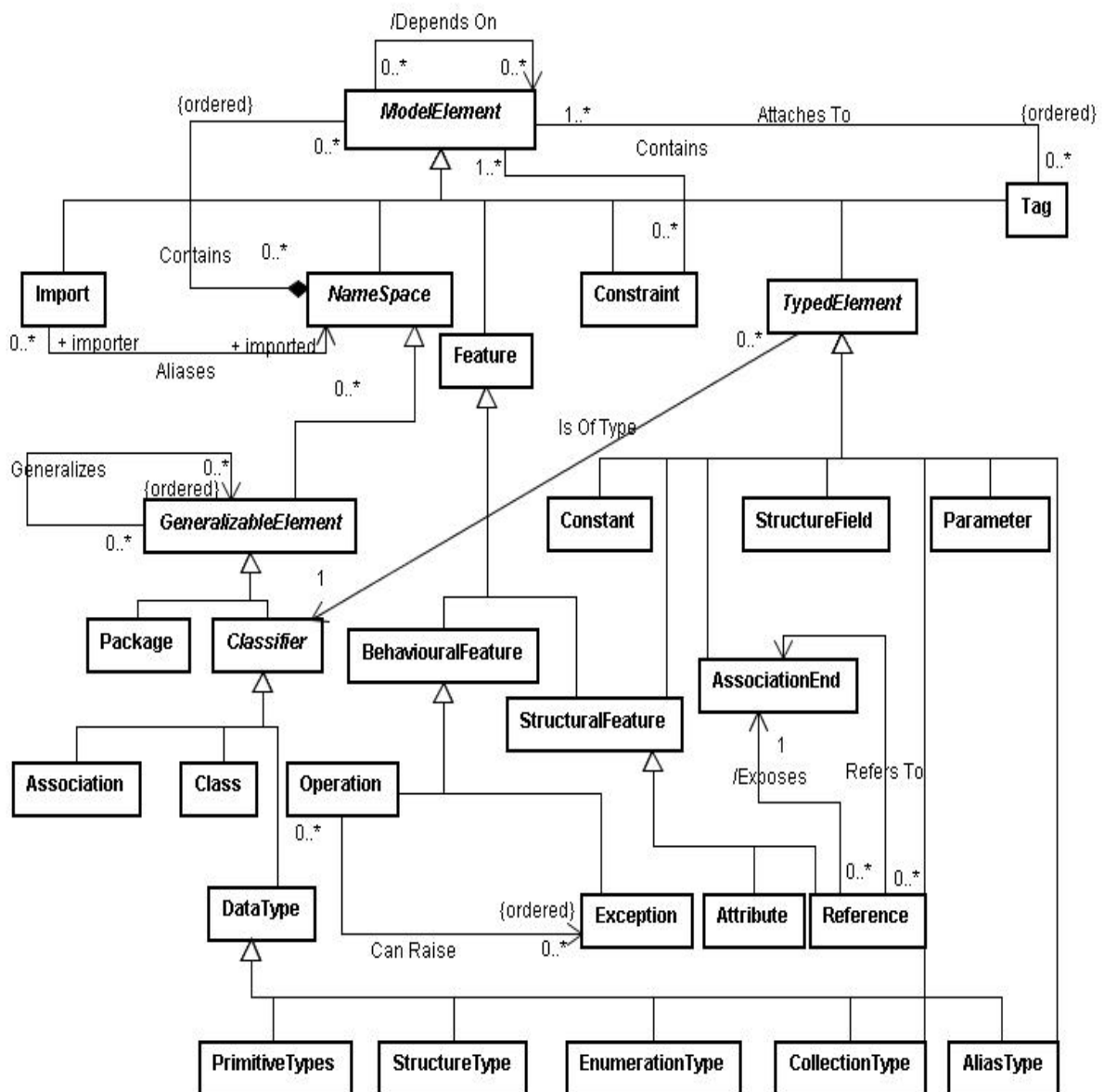


Figura 2: O Modelo MOF. Fonte: (OMG, 2002a)

2.2.5.1 Principais Classes Abstratas do Modelo MOF

Dentre as inúmeras classes que compõem o Modelo MOF destacam-se as classes abstratas *ModelElement*, *NameSpace*, *GeneralizableElement*, *Classifier* e *TypedElement*. Estas classes organizam o metamodelo e generalizam propriedades e operações especializadas pelas demais classes do MOF. A descrição das principais classes abstratas do MOF são apresentadas nas subseções seguintes, enquanto que as demais classes concretas estão descritas no ANEXO A.

***ModelElement*:** A *ModelElement* é a superclasse de todo o Modelo MOF. Ela declara atributos e operações fundamentais para o metamodelo. Os atributos são *name*, *qualifiedName* e *annotation* e suas finalidades são:

- O atributo *name* fornece um nome e identifica um *ModelElement* de uma forma única dentro do espaço de nome de seu contêiner.
- O atributo *qualifiedName* garante que o nome definido para o *ModelElement* é único no contexto de seu pacote mais externo.
- O atributo *annotation* contém informações relacionadas com o *ModelElement*.

As operações desta classe estruturam o relacionamento dela com as demais classes do metamodelo. São elas:

- *findRequiredElements* - esta operação permite localizar um subconjunto de *ModelElements* do qual um *ModelElement* é dependente. A natureza da dependência é descrito pela classe *Constant* e seus significados são obtidos através do relacionamento *DependsOn*.
- *isRequiredBecause* - Através desta operação pode-se verificar o grau de dependência (direta ou indireta) entre *ModelElements*.
- *isFrozen* - a finalidade desta operação é indicar se o estado do elemento de foi alterado ou não (*frozen*).

Ela também possui as referências *container*, *requiredElements* e *constraints*, que lhe permitem acessar as classes *NameSpace*, *ModelElement* e *Constraint*, e obter informações sobre o espaço de nome de um determinado *ModelElemente*

os elementos dos quais ele depende, bem como, as restrições as quais ele se subordina.

NameSpace : Esta classe tem a função de classificar e caracterizar *ModelElements* que podem conter outros *ModelElements* através da definição de um conjunto unívoco de nomes para aqueles *ModelElements* que fazem parte de um determinado espaço de nomes. Para tanto, estabelece restrições à nomeação dos mesmos. Algumas de suas subclasses, representadas na Figura 3, podem atuar contêineres desses elementos.

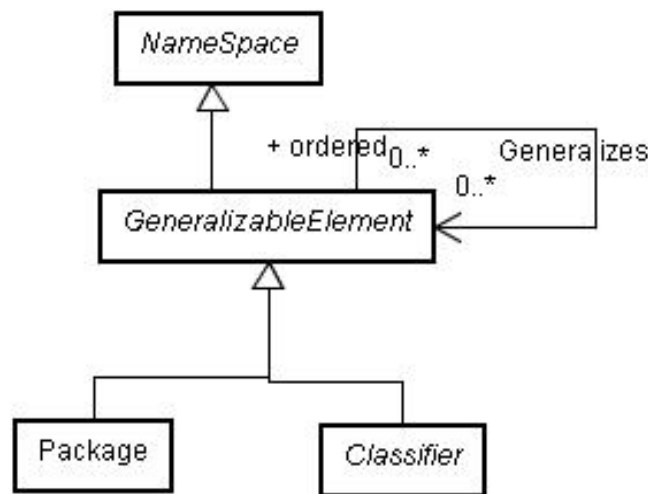


Figura 3: O *NameSpace* e suas subclasses. Fonte: (OMG, 2002a)

Com relação a associações, a *NameSpace* possui a referência *contents* que dá acesso à classe *ModelElement* e identifica as coleções de elementos que estão contidos em determinado espaço de nome.

As operações da classe *lookupElement*, *resolveQualifiedName*, *findElementsByType*, *nameIsValid* executam as atividades seguintes:

- Procurar por um elemento dentro do espaço de nome cujo nome corresponda exatamente ao nome fornecido (operação *lookupElement*).
- Obter um elemento específico contido pelo espaço de nomes identificado através do atributo *qualifiedName* (operação *resolveQualifiedName*).
- Obter uma lista de todos os elementos de um determinado tipo contido no espaço de nomes (operação *findElementsByType*).

GeneralizableElement: A classe *GeneralizableElement* é a superclasse de todas as classes que suportam herança no MOF. Na herança, todos os objetos descendentes herdam as características (atributos, referências, operações) definidas para cada um de seus ascendentes, sejam eles imediatos ou não. No MOF, ao herdar características de suas classes-mãe, as subclasses passam a fazer parte do espaço de nomes do *GeneralizableElement*. Estes nomes devem ser exclusivos, isto é, não podem colidir com nomes já existentes dentro do espaço de nomes do *GeneralizableElement*. O MOF define um conjunto de restrições que tratam deste aspecto e outros relacionados ao uso de herança.

A classe apresenta os atributos:

- *isRoot*- determina se a super classe poderá ou não ter supertipos.
- *isLeaf*- define se a classe pode ou não atuar como superclasse.
- *isAbstract* - indica se a classe terá ou não instâncias; uma classe abstrata não pode gerar objetos.

A classe possui as operações *allSuperTypes*, que retornam uma lista de as todas superclasses relacionadas direta ou indiretamente com a superclasse em foco; *lookupElementExtended*, que obtém um elemento (classe ou pacote) relacionado com o nome fornecido, direta ou indiretamente; *findElementsByTypeExtended*, que realiza buscas dentro do contexto da superclasse.

A *GeneralizableElement* tem apenas uma referência denominada *supertypes*, que relaciona sub-classes com suas superclasses.

Classifier: Esta classe classifica instâncias de acordo com um conjunto de características específicas. É a superclasse de *Association*, *Class* e *DataType*.

TypedElement: A *TypedElement* serve de base para todas as classes nível M3 que precisem especificar um tipo como é o caso de atributos, operações, parâmetros. A definição de tipos é feita através do relacionamento *IsOfType* que liga a classe *TypedElement* com a classe *Classifier*, que efetivamente determina o tipo através de suas especializações.

As classes *TypedElement*, *DataType* e seus relacionamentos de tipagem estão ilustrados na Figura 4.

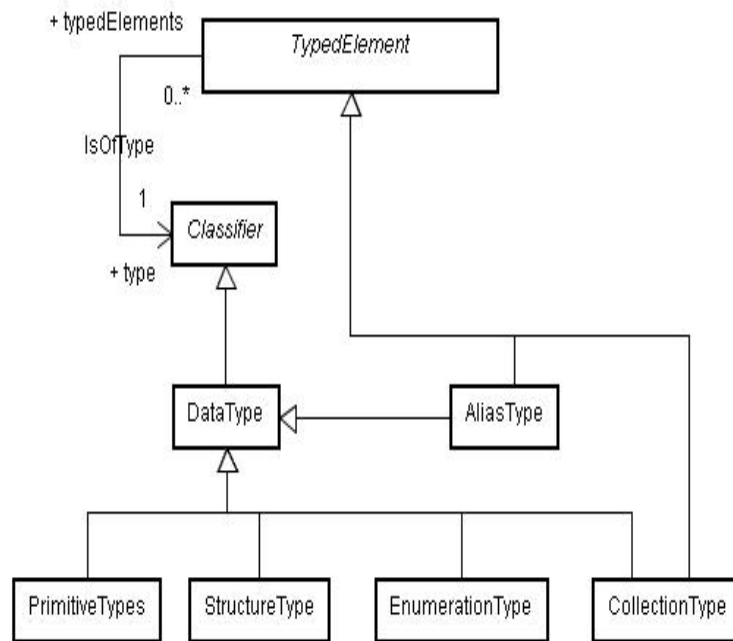


Figura 4: O *TypedElement* e elementos tipados. Fonte: (OMG, 2002a)

2.2.5.2 Os Relacionamentos do Modelo MOF

Além do conjunto de classes o Modelo MOF apresenta algumas associações, consideradas especiais, porque agregam características diferenciadas aos relacionamentos. Estas associações são: *Contains*, *Generalizes*, *IsOfType* e *DependsOn*. O papel destas associações são abordados nesta seção. Todavia, afora essas, há também as associações *RefersTo*, *CanRaise*, *Constrains*, *AttachesTo*, as quais encontram-se descritas no ANEXO B.

***Contains*:** A associação *Contains* representa um relacionamento de composição que interliga as classes *NameSpace* e *ModelElement*. Como um metamodelo MOF é uma composição de objetos do tipo *ModelElement*; também por que estas nem sempre são oriundas do mesmo pacote, há a necessidade de consistir o espaço de nome. Neste caso, a associação *Contains* relacionando as classes *NameSpace* e *ModelElement*, garante a unicidade do nome da classe e permite a importação e reuso de pacotes. Por essa razão, ele é considerado o relacionamento mais importante do Modelo MOF. Suas extremidades de associação são *container* e *containedElement*. O Modelo MOF estabelece regras através da OCL que especificam quais subclasses podem estar contidas ou conter outras

subclasses.

Generalizes: Esta associação tem em suas extremidades a classe *GeneralizableElement*; seu objetivo é tornar possível a generalização (herança), ou seja, a *Generalizes* representa o relacionamento entre os objetos nível M2 de classes-mãe e classes-filha. Suas duas extremidades indicam quais elementos foram definidos como supertipo(extremidade *supertype*) ou subtipos (extremidade *subtype*).

IsOfType: Este relacionamento liga as classes *TypedElement* e *Classifier*, estabelecendo um vínculo entre um elemento tipado e o classificador que possui a definição para este elemento. As suas extremidades são *type* e *typedElement*. A extremidade *type* estabelece o classificador enquanto que *typedElement* define o elemento tipado.

DependsOn: Este relacionamento estabelece uma auto-associação sobre os objetos da classe *ModelElement*. Ela permite identificar dependências estruturais que um determinado *ModelElement* tenha de outros *ModelElements*. Essas dependências podem se referir a vários aspectos e envolver contêineres, restrições, especialização, importação, entre outros. As extremidades desta associação são *dependent* e *provider*. *DependsOn* é uma associação derivada das associações *Contains*, *Generalizes*, *IsOfType* entre outras.

2.2.5.3 Os Tipos de Dados do Modelo MOF

Os tipos de dados do Modelo MOF são representados como instâncias de subclasses. Se enquadram nessa categoria os objetos das classes:

PrimitiveTypes: As instâncias de *PrimitiveTypes* são a única representação de tipos primitivos de dados usados no Modelo MOF.

MultiplicityType: O *MultiplicityType* é um registro que especifica a multiplicidade de atributos, parâmetros, referências ou extremidades de associação. Este registro contém os campos *lower*, *upper*, *isOrdered* e *isUnique*, que definem para os limites inferiores e superiores do registro se os objetos são ou não ordenados.

VisibilityKind: A enumeração dos três tipos de visibilidade possíveis para um elemento de modelo fora de seu contêiner é a função deste tipo de dado. As visibilidades podem ser:

- *public vis:* torna o elemento de modelo visível para todos os elementos de modelo no mesmo contêiner;
- *protected vis:* restringe o uso do elemento de modelo a apenas as subclasses;
- *private vis:* nega o acesso a qualquer outro elemento de modelo.

DirectionKind: Este tipo de dado define as direções possíveis para transferência de informação entre parâmetro de operações e exceções. As direções podem ser: *in dir* (entrada), *outdir* (saída), *inoutdir* (entrada e saída) e *returndir* (retorno).

ScopeKind: Os possíveis escopos para atributos e operações são elencar por *ScopeKind*. Estes são nível instância (*instancelevel*) e nível classificador (*classifierlevel*).

AggregationKind: Este tipo de dado contém as semânticas de agregação possíveis para uma associação. São elas: *none*, *shared* e composição *composite*.

EvaluationKind: Os possíveis tipos de dados para avaliação de restrições estão definidos em *EvaluationKind*. São eles o tipo *immediate* e *deferred*.

2.2.5.4 As Exceções do Modelo MOF

A exceção *NameNotFound* é uma das exceções definidas para o Modelo MOF e *NameNotResolved* a outra. A primeira é executada toda vez que uma operação de busca de nome falha e a segunda ocorre quando a resolução de um nome qualificado falha.

2.2.5.5 Constantes do Modelo MOF

As constantes do Modelo MOF são *Unbounded* e *DependenceKinds*. A constante inteira *Unbounded* é utilizada para expressar limites superiores indefinidos em multiplicidade. Já as constantes do tipo *DependenceKinds* são várias e denotam categorias de dependência que podem existir entre *ModelElements* de um metamodelo. São

elas: *ContainerDep*, *ContentsDep*, *SignatureDep*, *ConstraintDep*, *ConstrainedElementsDep*, *SpecializationDep*, *ImportDep*, *TypeDefinitionDep*, *ReferenceEndsDep*, *TaggedElementsDep*, *IndirectDep* e *AllDep*.

2.2.5.6 Restrições do Modelo MOF

As restrições estabelecidas para o MOF e seus metamodelos derivados (CWM, XMI, UML) foram definidas em OCL (*Object Constraint Language*), uma linguagem que fornece um conjunto de expressões apropriado para definir restrições a modelos.

WARMER(2003) define a OCL como uma linguagem de restrição com a qual pode-se construir modelos de software. As expressões escritas nessa linguagem são importantes para os modelos orientados a objeto e para seus artefatos, pois agregam informações relevantes para a implementação do sistema. No MOF, as restrições OCL apenas expressam limitações ao modelo, não modificando o estado dos objetos que restringem.

Contudo, o MOF não está atrelado a uma linguagem exclusiva para a especificação de restrições, nem tampouco define mecanismos particulares de avaliação (OMG, 2002a). Assim, semântica das restrições aplicáveis às operações, atributos e associações derivadas podem tanto ser expressas via OCL ou através de qualquer outra linguagem adequada a definição de restrição.

Ressalta-se porém, que as restrições são necessárias para que se possa representar metaobjetos ou ainda representar o Modelo MOF no formato XMI, pois expressam regras de consistências aplicáveis aos componentes de modelos tais como classes e pacotes. Toda restrição é composta pelos seguintes elementos:

- Um nome;
- Um conjunto de elementos restritos ao qual elas são aplicadas;
- Uma linguagem adequada para especificar as regras de consistência (restrição);
- Os elementos do metamodelo aos quais a restrição é aplicada;
- A definição da política de avaliação (*evaluation policy*) que determina se uma regra de consistência será aplicada de imediato ou futuramente.

Em uma expressão, escrita em OCL ou em qualquer outra linguagem de restrição, os elementos restringidos podem ser avaliados no momento em que houver mudança nos elementos ou posteriormente, de acordo com as definições da política de avaliação estabelecidas através dos atributos da classe *Constraint*.

2.2.5.7 O Pacote de Tipos Primitivos

Os tipos primitivos de dados do MOF, que compreendem os tipos *Boolean*, *Integer*, *Long*, *Float*, *Double* e *String*, estão contidos em um pacote denominado *PrimitivesTypes*. Neste pacote, estão as instâncias desses tipos, cuja utilização pode ser feita por qualquer metamodelo MOF. Os tipos *String*, *Boolean* e *Integer* são utilizados na especificação do Modelo MOF como instâncias de si mesmos.

2.2.6 O Mapeamento, As Interfaces e Metaobjetos MOF

Um dos aspectos mais relevantes na metamodelagem é a capacidade de se realizar o mapeamento de um metamodelo para outras tecnologias ou outros níveis de abstração. A exportação de metamodelos entre diferentes ferramentas, o reuso e extensibilidade dos modelos são algumas das possibilidades resultantes do mapeamento. Neste contexto, o pacote *Model* pode ser mapeado para as tecnologias CORBA ou Java através da geração de interfaces CORBA e JMI respectivamente. Então, estas interfaces são utilizadas para a criação de metamodelos, instâncias do Modelo MOF para aplicações CORBA e JAVA. Estes tipos de mapeamentos juntamente com o mapeamento do MOF de um nível de abstração para outro, por exemplo, nível M2 para nível M1, são tratados nesta seção.

2.2.6.1 Mapeamento entre níveis do MOF

Ao se criar um metamodelo MOF está se criando uma linguagem abstrata adequada à definição de tipos de metadados (OMG, 2002a). Estes são criados através do mapeamento dos construtores de nível M2 que estão no metamodelo em representações de nível M1 denominadas valores MOF. Cada um dos construtores MOF são mapeados de uma forma específica, conforme será exposto nos itens seguintes.

Classes: Os objetos (nível M1) são instâncias de classes M2 e uma vez mapeados apresentam as seguintes propriedades (OMG, 2002a):

- Todas as instâncias possuem identificador de objeto intrínseco, permanente e independente de qualquer propriedade externa, que permite distinguir de forma unívoca um objeto do outro. Ele também serve para determinar se instâncias de determinada classe são iguais.
- Instâncias de objetos podem se conectar umas com as outras através de associações. Instâncias do tipo *Null* devem ser evitadas pois causam problemas nos mapeamentos.

A igualdade entre instâncias não é determinada apenas pela identificação de objetos, à atributos ou vínculos existentes entre as instâncias e sim pela tríade desses valores. As instâncias de nível M1 seguem regras que definem o seguinte:

- O tempo de vida de instâncias nível M1 (criação e destruição de instâncias) é determinado por eventos definidos no modelo computacional.
- Somente classes concretas nível M2 podem gerar instâncias nível M1, ou seja, classes abstratas não geram instâncias.

Tipos de Dados: As instâncias M1 de tipos de dados M2 do MOF não possuem identificação, não podem estabelecer vínculos via associação e não geram instâncias *null* (OMG, 2002a). Além disso, os tipos de dados MOF se enquadram ou no grupo tipo de dados neutros composto pelos tipos de dados primitivos ou nativos, formados por outros dados em tecnologia não neutra.

Atributos: Os atributos são um dos mecanismos usados para definir relacionamento entre valores no nível M1 (OMG, 2002a) e relacionar as classes M2 com suas instâncias no nível M1. A semântica computacional de atributos no nível M1 pode variar conforme o tipo de mapeamento utilizado. Porém, algumas independem do mapeamento, são elas (OMG, 2002a):

- Todo atributo tem uma assinatura composta pelo seu nome e seu tipo, definindo um relacionamento binário entre uma instância de classe e um valor(es) de atributo(s). Uma vez que, no nível M2, o atributo pode ser do tipo Classe ou um tipo de dados, o mapeamento em M1 relaciona, na primeira situação, duas classes (uma representando a classe do atributo e outra seu tipo de dados) e no outro caso, haverá uma instância representando o atributo e outra correspondendo ao seu tipo de dado.

- A multiplicidade M2 de um determinado atributo influencia em seu valor no nível M1, determinando se ele é ou não uma coleção. Se houver limites inferiores e superiores, será necessário avaliar em tempo de execução se os limites estão sendo respeitados, se não há membros duplicados, e disponibilizar mecanismos que permitam especificar a ordem dos elementos de atributos multivalorados.
- Um atributo cujo escopo seja classificador terá uma única instância do tipo de atributo ou coleção deste, vinculado as instâncias da classe que contém o atributo. Se o escopo for instância, haverá relacionamentos independentes entre instâncias da classe a que pertence o atributo e instâncias de tipo de atributo.
- A ativação do sinalizador "*is derived*" indica que o relacionamento entre a instância de uma classe qualquer e a instância do atributo é computada.

Associação: A associação é o outro mecanismo que relaciona valores MOF no nível M1. Uma associação é uma relação binária entre pares de instâncias no nível M1 e reflete uma associação nível M2. Os relacionamentos em uma associação são denominados vínculos. Estes vínculos somente são navegáveis se o valor do sinalizador "*isNavigable*" for *TRUE* e se a classe oposta possuir uma instância de *Reference*. Isto significa que a existência de uma declaração de referência em uma classe habilita a navegação à extremidade de associação da classe com a qual esta se relaciona (FRANKEL, 2003) (OMG, 2002a). Portanto, a extremidade de associação será visível e navegável a todas as classes que possuam uma referência para a extremidade. O valor padrão para a visibilidade de extremidades de associação no MOF é público, a menos que outro valor seja especificado. O mapeamento de MOF (OMG, 2002a) estipula várias possibilidades no mapeamento de associações de classe nível M2 para vínculos ou conjuntos de vínculos no nível M1 envolvendo multiplicidade, ordenação (*ordered*) e estado das extremidades de uma associação. Estas definições determinam algumas características nas instâncias de nível M1, como por exemplo:

- Somente existem vínculos se houver instâncias cujo estado seja bem formado;
- Vínculos são direcionados, o que torna um vínculo[x,y] diferente de [y,x];

- Não há identificador para vínculos, eles são identificados pelas instâncias em suas extremidades;
- Não há conexões entre instâncias de classes nulas com outras classes ou consigo mesma;
- Não existem duplicatas de vínculos em associações deste nível;
- Vínculos ordenados (*ordered*) podem ser representados por uma seqüência linear.
- Conjuntos de vínculos podem apresentar múltiplos estados para uma mesma associação nível M2.

A ativação de atributos de sinalização (*flags*) no nível M2 também influencia na semântica de instâncias M1 de associações possibilitando, ou não, modificações, navegação, derivação nas extremidades das associações. Destes sinalizadores, destaca-se o que se refere à agregação.

A agregação estabelece um forte relacionamento entre duas classes, em que a exclusão da classe agregadora elimina as classes agregadas.

O MOF define três semânticas para a agregação a saber, *none*, *composite* e *shared*. Todas elas correspondem semanticamente aos seus similares na UML. Todavia, com relação a agregação *shared*, *aggregate* na UML, o seu uso não é estimulado no MOF.

2.2.6.2 Composições de Pacotes e o Mapeamento no Modelo MOF

Existem quatro mecanismos de composição de metamodelos no Modelo MOF que são o aninhamento, a generalização, a importação e o agrupamento de pacotes.

Destes, o aninhamento é o mais simples. Seu objetivo é ocultar a informação, pois uma classe pertencente a um pacote mais interno torna-se invisível a outros metamodelos. No nível M1, o comportamento da instância de classificador depende de seus pacotes mais externos e internos.

Nos pacotes generalizados, no nível M2, há o reuso de definições de pacotes; neste gênero de composição, o comportamento das instâncias de M1 de um subpacote depende do comportamento dos elementos do superpacote nível M2.

Na composição por importação também ocorre o reuso, contudo, isto acontece de uma forma seletiva. Uma importação pode ser considerada uma agregação ou composição, motivo pelo qual a semântica para nível M1 não tem maiores impactos, não havendo estados de relacionamentos entre instâncias de pacotes importados ou importando (OMG, 2002a).

No agrupamento de pacotes igualmente há o reuso seletivo, com a diferença de que nele a infraestrutura de implementação nível M1 é compartilhada. Sua semântica é similar a dos pacotes aninhados.

2.2.6.3 Domínio de instâncias M1 - Extensões

No MOF, as informações relativas ao contexto de classes, associações e pacotes são fornecidas pelas extensões, também chamadas domínio de instâncias de M1.

As extensões informam conteúdos que um objeto compartilha com os demais, o que é importante para a implementação de servidores, por exemplo. O conceito de contexto para MOF se refere aos aspectos de uma classe cujo escopo tenha sido definido como classificador e que são, portanto, comuns a todas as instâncias da classe em um mesmo domínio.

As extensões seguem rigorosamente a hierarquia, pacote, classe, associação. Assim, cada vínculo ou instância de classe, pertence obrigatoriamente a uma associação e a uma classe e, estas duas, a um pacote. As extensões funcionam da seguinte forma: quando se define o escopo de uma classe para o nível *classifier*, automaticamente a extensão da classe disponibiliza todas as informações sobre as instâncias no nível M1. Essas instâncias são criadas dentro do domínio da classe e ali permanecem até a sua exclusão.

Pelo mesmo princípio, as extensões de associações contêm todos os vínculos relativos ao nível M1 de uma associação. Vínculos são criados no domínio de uma associação e ali permanecem durante toda sua existência.

As extensões de pacote são criadas de acordo com regras específicas cujos detalhes podem ser obtidos consultando a especificação (OMG, 2002a). Uma extensão de pacote é um conjunto de extensões de classes, associações e outros pacotes.

2.2.6.4 Interfaces MOF

No MOF, as interfaces resultantes do processo de mapeamento para qualquer plataforma utilizam obrigatoriamente quatro tipos diferentes de metaobjetos (OMG, 2002a). São eles: *association*, *instance*, *class proxy*, e *package*.

O relacionamento entre estes tipos e seu papel dentro do mapeamento MOF são abordados na seqüência.

Package: Um metaobjeto *Package* é uma instância da classe nível M3 MOF *Package*. Ele representa um contêiner de metaobjetos. O pacote mais externo é definido como o metaobjeto raiz dos demais.

Class Proxy: Para toda classe do nível M2 existe uma classe especial, denominada *Class proxy*, e um objeto *proxy* correspondente. A função destes metaobjetos pode ser:

1. Instanciar metaobjetos *instance*. O metaobjeto *proxy* atua como uma fábrica de metaobjetos *instance*;
2. Atuar como contêiner para os metaobjetos do tipo *instance*;
3. Fornecer métodos para acesso e modificação do estado dos atributos com escopo classificador.

Instance: Os metaobjetos *instance* representam instâncias de classes do nível M2. Eles permitem a modificação nos estados dos atributos. Os metaobjetos *instance* e *Class proxy* relacionam-se da seguinte forma:

1. Todo metaobjeto possui desde o momento de sua criação um metaobjeto contêiner que é do tipo *class proxy*. Uma *class proxy* fornece os métodos que criam os metaobjetos *instance*. Os parâmetros desses métodos são os valores dos atributos cujo escopo foi definido como instância.
2. Durante toda a sua vida um metaobjeto *instance* estará associado ao seu metaobjeto contêiner. Ao ser destruído ele é removido desse contêiner.

As interfaces dos metaobjetos *instance* fornecem operações importantes para o seu contexto, pois propiciam o acesso, modificação do estado de atributos, invocação de operações de escopo instância, o suporte a identificação de metaobjetos e a exclusão de metaobjetos *instance*, entre outros itens.

Association: Os metaobjetos *Association* manipulam coleções de vínculos relacionada às associações do nível M2. Seus contêineres são os metaobjetos do tipo *package*. As operações de suas interfaces permitem consultar vínculos específicos em um conjunto de vínculos, adicionar, modificar e remover vínculos a partir do conjunto ou ainda obter todo o conjunto de vínculos.

2.2.6.5 Interfaces *Reflective* de MOF

O MOF possui um pacote adicional denominado pacote *Reflective*. Este pacote contém um conjunto de interfaces que disponibiliza mecanismos para a criação, consulta, alteração de metaobjetos sem que haja a necessidade do conhecimento prévio de suas interfaces.

Estas interfaces são descritas brevemente nesta seção. Uma exposição mais abrangente a respeito de suas operações pode ser encontrada na especificação MOF (OMG, 2002a). As interfaces são:

- *RefBaseObject* - que é a interface raiz do pacote *Reflective*. Ela fornece operações comuns a quase todos os metaobjetos MOF, menos para as exceções, tipos enumerados e tipos estruturados. As operações permitem testar a identidade dos metaobjetos, retornar um metaobjeto de um objeto, entre outras.
- A interface *RefPackage* fornece acesso aos metaobjetos de um determinado pacote. Os métodos *refClass* e *refAssociation*, por exemplo, retornam informações sobre uma classe ou associação específica.
- A interface *RefAssociation* permite a manipulação de metaobjetos tipo *Association* através de operações genéricas para consultar e alterar vínculos.
- A interface *RefObject* disponibiliza aos metaobjetos do tipo *instance*, operações de acesso e alteração dos estados dos atributos e permite a invocação de operações com escopos de instância.

Essas interfaces, ilustradas na Figura 5, fornecem mecanismos para a realização diferenciada de operações via mapeamento de interfaces. Este aspecto é importante para aplicações que manipulam diversos metamodelos sem conhecê-los, como por exemplo, um *browser* de metaobjetos.

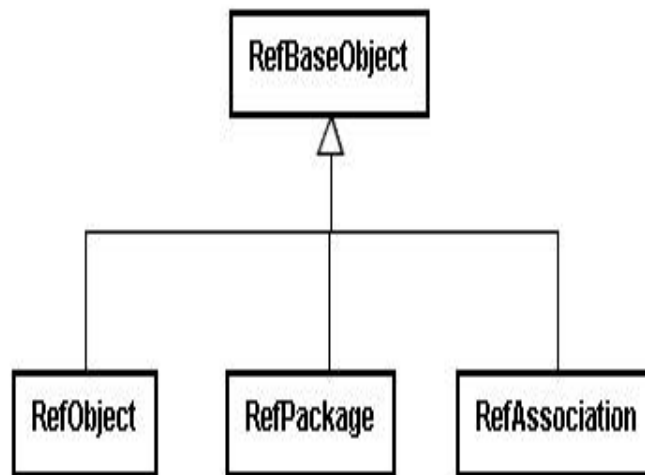


Figura 5: O Pacote *Reflective*. Fonte: (OMG, 2002a)

As regras utilizadas para o mapeamento de interfaces MOF para outras plataformas utilizando-se do pacote *Reflective* estabelecem que:

- Para cada pacote e cada associação do metamodelo devem ser criadas uma interface *package* e uma *association*, respectivamente;
- Para cada classe do metamodelo são criadas uma interface *proxy* e uma interface *instance*.

E, no que se refere à herança entre metaobjetos, os procedimentos de mapeamento são os seguintes:

- Um metaobjeto *instance*, sem supertipo, herda de *RefObject*. Todos os demais estendem seus supertipos;
- Um metaobjeto *package*, sem supertipo, herda de *RefPackage*. Todos os demais estendem seus supertipos;
- Todos os metaobjetos *association* herdam de *RefAssociation*.

Por exemplo, dado um metamodelo, ilustrado pela Figura 6, contendo dois pacotes denominados Pack1 e Pack2. Destes, Pack1 é o superpacote de Pack2. O Pack1 contém duas classes. Uma delas (Classe1), atua como superclasse da outra,

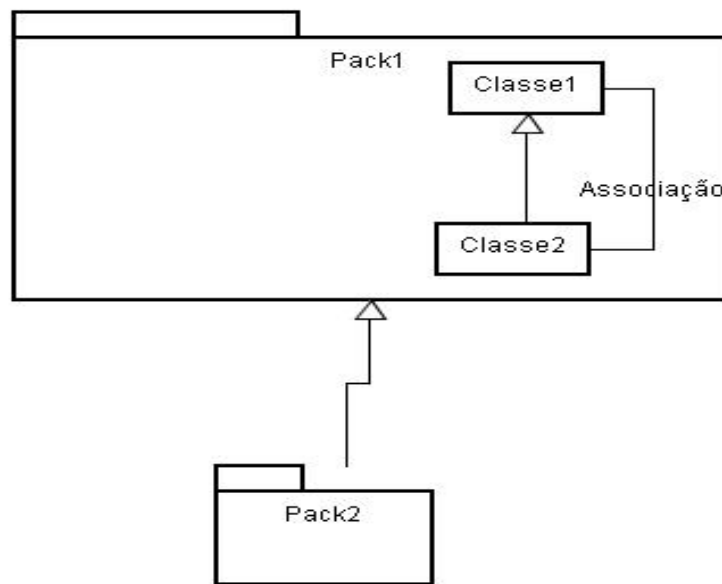


Figura 6: Exemplo de metamodelo MOF. Adaptado de OMG (2002a)

chamada Classe2, ou seja, Classe2 é subclasse de Classe1. Estas classes se conectam através de uma associação definida como "Associação".

Ao aplicar as regras de mapeamento acima descritas, obtém-se como resultado duas interfaces *package* referente a Pack1 e Pack2, respectivamente. Entretanto, como o pacote *Pack1* não apresenta supertipos, ele herda diretamente de *RefPackage*, enquanto que Pack2, por ser um subtipo de Pack1, herda as propriedades de seu antecedente.

Para cada classe do metamodelo são geradas duas classes: uma referente aos metaobjetos *instances* e outra relativa aos metaobjetos *class proxy*.

2.2.6.6 O Mapeamento MOF-CORBA

O primeiro mapeamento do MOF para uma plataforma específica foi para o CORBA. Este mapeamento possibilita a representação de modelos UML como metaobjetos CORBA utilizando-se de um conjunto de interfaces CORBA IDL.

Estas interfaces oferecem suporte às operações de criação, alteração, exclusão e recuperação de informações sobre os modelos UML. O objetivo é permitir a clientes de um repositório de metadados baseados em CORBA usar interfaces para realizar operações (FRANKEL, 2003).

Na atualidade, este mapeamento foi estendido aos metamodelos CWM e XMI, também oriundos de MOF, os quais, mapeados, têm interfaces CORBA baseadas em IDL ou JMI. Dessa forma, modelos CWM podem ser representados como meta-objetos CORBA ou Java.

2.2.6.7 O Mapeamento MOF para Java

Faz parte da especificação do MOF o seu mapeamento para IDL CORBA através de regras gerais independente de plataforma. Baseado nisso, a comunidade Java definiu o mapeamento do MOF para Java através do *Java Metadata Interface* (JMI) (JCP, 2002).

A especificação JMI define o mapeamento entre metamodelos MOF e a linguagem Java. Este mapeamento leva a interfaces para objetos Java que representam elementos da camada M2 e permitem a realização de operações sobre os metamodelos. Dentre as operações estão a criação, alteração, exclusão e recuperação de informações.

As interfaces JMI estão organizadas em dois pacotes (JCP, 2002): *javax.jmi.model* e *javax.jmi.reflect*. O primeiro representa o Modelo MOF e o segundo as interfaces reflexivas. O JMI torna possível o acesso às interfaces do MOF através de clientes Java.

2.2.7 MOF e a UML

A especificação do MOF define uma linguagem abstrata utilizada para definir linguagens de modelagem. Por ser abstrata, a sua descrição é inteiramente textual, não há representações gráficas para o MOF. Contudo, como seus construtores de metamodelagem são, em sua maioria, compatíveis com os construtores da UML, afinal o metamodelo desta foi concebido através do MOF, pode-se utilizá-la para representá-lo graficamente. Ressalta-se, porém, que semanticamente o Modelo MOF é totalmente independente de quaisquer modelos, inclusive da UML. Esta seção apresenta a relação entre o MOF e o padrão de modelagem UML.

A UML é uma linguagem de modelagem universal adotada pela OMG como, o padrão para modelagem de sistemas discretos (OMG, 2001). Totalmente orientado a objetos, ela fornece uma linguagem de modelagem independente de plataforma.

Dentre seus principais construtores de modelagem estão as classes, os relacionamentos e tipos de dados, os quais também são utilizados pelo MOF.

A capacidade da UML em representar metamodelos MOF decorre do fato de que ambos compartilham uma terminologia comum. Apesar disso cada um deles possui uma aplicabilidade diferente. Enquanto a UML é utilizada como linguagem de modelagem orientado a objetos de propósito geral, o MOF é utilizado para definir linguagens de modelagem.

As diferenças estruturais entre os construtores de UML e MOF são as seguintes:

1. O MOF suporta apenas associações binárias, já que, em metamodelagem, raramente se define uma associação com mais de duas classes envolvidas, enquanto que UML suporta n-árias. Todavia, futuramente o MOF deve suportar associações n-árias.
2. Em MOF não existem classes associativas, ou seja, as associações MOF não contêm características tais como atributos e métodos, aspecto existente na UML.
3. O MOF suporta o conceito de referência que estabelece o seguinte: se há uma associação entre as classes A e B, então há também uma referência em A de B e de B em A. Através dessas referências o cliente navega de uma classe para outra sem precisar utilizar a associação. Este mesmo conceito existe na UML na forma do atributo *associationEnd* ou extremidade de associação, contudo, a navegação é realizada pela associação.

Então, o que há de diferente entre as duas abordagens é o propósito de uso e em alguns itens a forma da implementação já que ambos podem ser representados na mesma notação, pois, além dos pontos já citados, outros conceitos como, por exemplo, generalização, dependência e refinamento são implementados como classes em UML e como associações em MOF.

Apesar dessas divergências, na atualidade, a UML é a linguagem de modelagem adotada para representar graficamente metamodelos MOF.

2.2.8 Ferramentas MOF

Na atualidade, os principais desenvolvedores da área de modelagem, de alguma forma, adaptaram suas ferramentas para criar metamodelos compatíveis com MOF, tais como, o *Rational Rose* da IBM ², o *Objecteering*³, ou ainda, o Poseidon ⁴, os quais proporcionam a criação de metamodelos compatíveis com o MOF utilizando-se da UML.

Uma vez elaborado o metamodelo, este pode ser importado por um ambiente integrado de desenvolvimento (*Integrated Development Environment -IDE*) como, por exemplo, o *NetBeans* ⁵ ou *Arcstyler* ⁶ ou, UML ainda, para repositórios de modelos como dMOF ⁷, nos quais pode ser executado e gerar os *frameworks* desejados. Estas ferramentas são apenas algumas das disponíveis no mercado implementando diferentes aspectos das atividades relacionadas com o desenvolvimento orientado a modelos.

²www-306.ibm.com/software/rational

³www.objecteering.com

⁴www.gentleware.com

⁵www.netbeans.org

⁶www.arcstyler.com

⁷www.dstc.edu.au

3 SISTEMAS IMUNOLÓGICOS

Uma vez que este trabalho se propõe a metamodelar o Sistema Imunológico Artificial, efetua-se neste capítulo a revisão de parte do conhecimento necessário para o entendimento dos Sistemas Imunológicos Humanos e Artificial. Nele, abordamos as características funcionais e estruturais do Sistema Imunológico Humano de forma condensada, contudo, suficiente para o escopo do trabalho e os princípios básicos do Sistema Imunológico Artificial.

3.1 Sistema Imunológico Humano

3.1.1 Imunologia

Para KEPHART (1994), o Sistema Imunológico Humano é constituído de uma rede complexa formada por órgãos e células envolvidos na defesa do corpo contra ataques de invasores externos, conclusão esta corroborada por SOMAYAJI (1997), HOFMEYR (1999) e FORREST (2000). Em seu funcionamento normal, o sistema imunológico combate as infecções ocasionadas por bactérias, fungos, vírus ou parasitas, do contrário, o corpo fica sujeito a uma grande quantidade de doenças.

O nosso meio ambiente está repleto destes elementos potenciais causadores de enfermidades que podem se multiplicar descontroladamente e levar o hospedeiro ao óbito. Assim como o homem, estes organismos obedecem ao princípio natural de preservação das espécies.

O corpo humano é o habitat ideal para muito deles, motivo pelo qual tentam invadi-lo, cabendo ao sistema imunológico impedir tal intento. Caso venha a falhar acionará outros mecanismos para caçar e destruir o invasor. Dentre as características

principais do sistema imunológico está a capacidade de distinguir entre o *self* e o *nonself* (HELMAN; DHAESELEER; FORREST, 1996) (DHAESELEER, 1996). Todas as células do corpo possuem células que são caracterizadas como *self*. As células dos organismos estrangeiros possuem outras moléculas denominadas *nonself*.

O sistema imunológico apresenta duas características opostas: a especificidade e a diversidade (HELMAN; DHAESELEER; FORREST, 1996). Em decorrência disso, ele é capaz de reconhecer milhões de células *nonself* distintas e de elaborar combinações de moléculas e células para combatê-las de modo eficiente.

O êxito do sistema imunológico na defesa do corpo é dependente de uma rede de comunicação dinâmica e elaborada, formado por células que, organizadas em conjuntos e subconjuntos, trocam informações entre si. O resultado é um sistema de verificação sensível que produz uma resposta imunológica conveniente e efetiva.

Sem a capacidade do sistema imunológico de detectar e extinguir a maioria das enfermidades, a humanidade estaria em risco constante de morte.

O sistema imunológico possui dois níveis de defesa contra agentes invasores externos: a imunidade inata e a imunidade adaptativa ou adquirida (STITES; TERR, 1992). Destes, a imunidade inata é o nível mais elementar. Ela tem, na pele, sua primeira linha de defesa que, apesar de constituir uma proteção externa eficiente para a maioria das invasões, algumas vezes é suplantada por organismos agressores específicos. Quando isso ocorre, outros elementos do sistema imunológico inato tentam deter o invasor. Contudo, se todas as barreiras fornecidas pela imunidade inata fracassam, desenvolve-se a resposta imune adaptativa. Nela, a fisiologia do organismo atua como defesa utilizando as condições de PH e temperatura para tornar o ambiente inadequado aos invasores. O sistema imunológico adaptativo distingue-se do sistema imunológico inato por apresentar especificidade e memória, características inexistentes neste último.

O sistema imunológico adaptativo inicia sua aprendizagem a partir do primeiro contato com algum organismo desconhecido que consegue suplantar o sistema inato, pois, para eliminar uma infecção, ele deverá reconhecer e destruir as células infectadas. A cada novo contato com um mesmo patógeno (elemento causador de doenças), a resposta do sistema imunológico adaptativo torna-se mais eficiente, pois ele memoriza o agente infeccioso capacitando-se para combatê-lo futuramente. Assim, caso o corpo venha a sofrer o mesmo processo infeccioso, o sistema imunológico adaptativo

reconhece a patogenia e elabora uma resposta de melhor qualidade e maior intensidade a partir desse segundo contato. É dessa forma que a memória imunológica protege o organismo durante toda a sua existência.

O sistema imunológico adaptativo é formado de células sanguíneas denominadas linfócitos que existem aos milhões dentro do organismo humano. Eles cooperam na detecção e eliminação de doenças, atuando como detectores móveis e independentes que circulam pelo corpo através do sistema sanguíneo e linfático.

3.1.2 *Self e NonSelf*

A essência do sistema imunológico está na sua capacidade de discernir entre o *self* e o *nonself* (KEPHART, 1994) (SOMAYAJI; HOFMEYR; FORREST, 1997)(HOFMEYR, 1999) (FORREST; HOFMEYR, 2000), e a partir daí, eliminar este último.

Todas as células do corpo portam moléculas que as identificam como *self*. Células com essa identificação não são atacadas pelas defesas imunológicas, porém, ao detectarem as classificadas como *nonself*, mecanismos de defesas são rapidamente acionados de modo a combater e eliminar os organismos intrusos.

Uma molécula capaz de induzir uma resposta imunológica em um determinado hospedeiro denomina-se antígeno, que pode ser um fungo, uma bactéria, um parasita, um vírus, ou mesmo células ou tecidos de outras pessoas, introduzidos por transfusão de sangue ou transplante. Uma única molécula é um único antígeno, contudo, células podem conter vários antígenos (STITES; TERR, 1992).

O local de resposta imune, ou seja, a ação de defesa do sistema imunológico, depende parcialmente da via de acesso do antígeno no organismo. Assim, se ele permanece na pele ocorre uma resposta inflamatória local. Se a invasão ocorre por via sanguínea, ele é levado para o baço, que se torna o principal local da resposta imune (STITES; TERR, 1992). Se a área de entrada são as vias respiratórias e gastrointestinais então, estes mesmos órgãos acionam suas defesas em resposta à invasão.

Um antígeno é detectado pelo sistema imunológico em decorrência de protuberâncias existentes em sua superfície, chamadas de **epítomos**. Antígenos carregam diferentes tipos de **epítomos** e estes, por sua vez, apresentam diferentes níveis de resposta imunológica.

3.1.2.1 Genes e Características *Self*

As moléculas que marcam uma célula como *self* são codificadas por um grupo de genes que estão contidas em seções de um cromossomo específico conhecido como *Major Histocompatibility Complex* (MHC).

Um antígeno seja *self* ou *nonself* somente é reconhecido por células T, linfócitos derivados do timo (STITES; TERR, 1992). Se estas estiverem em conjunto com moléculas MHC. Os detalhes da estrutura destes genes e as moléculas por eles codificadas variam de um indivíduo para outro. Esta diversidade é conhecida como polimorfismo.

O MHC é essencial para as defesas imunológicas, pois suas características determinam a quais antígenos uma resposta imune seria benéfica e a quais esta seria nociva, definindo também, a intensidade da resposta. As moléculas que apresentam os antígenos para as células B, linfócitos derivados da medula óssea (STITES; TERR, 1992), células T e macrófagos são codificadas principalmente no MHC.

As primeiras apresentam propriedades que habilitam as últimas a se reconhecerem mutuamente e se comunicarem. Essas moléculas formam um grupo de glicoproteínas, que codificadas pelos genes MHC fornecem as características *self* que aparecem em quase todas as células sanguíneas. Elas são conhecidas como:

- Antígenos MHC de classe I - estas moléculas alertam células-T *killer* sobre a presença no organismo de células infectadas com um vírus intracelular ou transformadas em câncer e que precisam ser eliminadas.
- Antígenos MHC de classe II - estas moléculas estão presentes nas células B, nos macrófagos e em outras células responsáveis por presentirem a presença de antígenos estrangeiros. Os produtos dos antígenos de classe II combinam com a partícula do antígeno estrangeiro de modo a expô-lo e chamar a atenção das células-T *helper*.

3.1.3 Anatomia do Sistema Imunológico Humano

As células que formam o sistema imunológico estão disseminadas por todo o organismo. Todavia, elas se concentram mais em órgãos linforreticulares e em tecidos linfóides relacionados com as vias respiratórias e gastrointestinais (STITES; TERR,

1992). A denominação de cada uma está relacionada com suas funções, que são o crescimento, o desenvolvimento e a utilização de linfócitos.

Os órgãos linfóides incluem a medula óssea, o baço, o timo, assim como os linfonodos, fígado, adenóide e o apêndice. O sangue e os vasos linfáticos, que fazem o transporte dos linfócitos de uma estrutura para outra, também são considerados como componentes dos órgãos linfóides. Tanto as células que estão destinadas a tornarem-se células imunológicas, como as demais células do sangue são produzidas na medula óssea. A medula óssea é rica em células embrionárias das quais originam os linfócitos e os fagócitos responsáveis pela absorção de partículas extracelulares e das células brancas.

Alguns elementos são essenciais na estrutura do sistema imunológico. Nas subseções seguintes abordamos cada um deles.

3.1.3.1 Linfócitos

Os linfócitos são pequenas células de glóbulos brancos produzidas na medula óssea e timo. Algumas delas migram, através da circulação sanguínea, para outros locais como baço, linfonodos e tecidos linfóides associados às mucosas. As duas maiores classes de linfócitos são:

- Células-B ou linfócitos B. As células-B crescem e amadurecem fora do timo. Elas trabalham principalmente segregando substâncias solúveis chamadas de imunoglobulinas dentro dos fluidos do corpo. Os anticorpos são incapazes de penetrar em células vivas. A maioria das células B possui antígenos de classe II do MHC, que são importantes para ações cooperativas com as células T.
- Células-T. As células T se desenvolvem e amadurecem em sua maioria no timo. Elas interagem diretamente com seus alvos, atacando a célula do corpo que tenha sido contaminada por alguma virose.

As distinções entre as células-B e as células-T não se baseiam somente nas moléculas presentes em suas superfícies, mas também no modo de atuação, que mesmo entre células de um mesmo tipo podem sofrer variações.

3.1.3.2 Células-B e Anticorpos

Cada célula-B do sistema imunológico está programada para gerar anticorpos específicos também conhecidas como imunoglobulinas. Imunoglobulinas são proteínas, formadas por cadeias polipeptídicas, cujas unidades básicas são conhecidas como aminoácidos (STITES; TERR, 1992). Cada anticorpo tem duas cadeias polipeptídicas fortes idênticas e duas cadeias leves tomando a forma de um Y, que atuam no reconhecimento do antígeno estrangeiro e na subsequente resposta do Sistema imunológico adaptativo.

Elas são criadas a partir do primeiro contato de uma célula-B virgem com um antígeno. A partir do que a célula B torna-se ativa. Em um momento seguinte, com a colaboração da célula-T, ela amadurece em plasmócitos, que por sua vez, passam a gerar grandes quantidades de anticorpos. Os plasmócitos são produzidos em um processo de clonagem em que um plasmócito é criado a partir de uma célula-B qualquer, replicado, ele origina alguns milhares de cópias que são colocados no fluxo sangüíneo.

Uma determinada imunoglobulina, combina com um determinado antígeno da mesma maneira que uma chave combina com uma fechadura. Na maioria dos mamíferos foram identificadas cinco classes principais de imunoglobulinas designadas IgG, IgA, IgM, IgE e IgD (STITES; TERR, 1992). Alguma delas, contudo, apresentam modificações que lhes permitem compor outras subclasses. Cada uma das classes desempenha uma diferente tarefa na estratégia da defesa imunológica cujas atuações estão resumidas a seguir:

- IgG - é a mais numerosa imunoglobulina no fluxo sangüíneo. É capaz de entrar nos tecidos, como por exemplo a placenta, e trabalha eficientemente para cobrir microorganismos. Ela é a responsável pela proteção dos recém-nascidos durante o primeiro mês de vida.
- IgM - esta imunoglobulina mantém-se no fluxo sangüíneo e atua matando bactérias.
- IgA - a IgA concentra-se nos fluidos do corpo, como saliva, secreções gastrointestinais e respiratórias, vigiando as entradas do organismo.
- IgE - esta imunoglobulina ocorre somente em pequenas porções. Está relaci-

onada na defesa contra infecções parasitárias e reações alérgicas, situação em que ocasiona o surgimento de edemas e rubor na pele.

- IgD - esta imunoglobulina está presente dentro das membranas das células-B agindo na ativação celular.

3.1.3.3 Células-T e linfocinas

As células T contribuem para com o sistema imunológico de duas formas. Inicialmente, atuando na coordenação do sistema, uma vez que as células-B não podem fazer anticorpos contra algumas substâncias sem a ajuda das células-T.

A segunda é através da célula-T citotóxica, que ataca diretamente células infectadas do corpo. A atuação nas atividades de controle é, dentre as duas atividades anteriores, a propriedade mais relevante das células-T, porque nesse processo são elas que induzem ou ajudam outras células (JUCA, 2001). As células-T atuam pela secreção de substâncias conhecidas como linfocinas. Estas podem incrementar a resposta imune por outros linfócitos B e T, pois atuam como reguladores das respostas imunológicas, inflamatórias e da recuperação de lesões. Dentre as linfocinas estão os linfócitos letais que são grandes células que aniquilam outras células, inclusive tumorais, denominados de *Natural Killer* (NK). Estes ao reconhecer uma célula tumoral a destroem imediatamente, dispensando o reconhecimento dos antígenos do MHC.

3.1.3.4 Fagócitos

Os fagócitos são grandes células brancas que envolvem e digerem microorganismos invasores como bactérias ou parasitas. Alguns fagócitos também possuem a capacidade de mostrar o antígeno para o linfócito.

Os fagócitos mais importantes são os monócitos e os macrófagos (JUCA, 2001). Os monócitos circulam no sangue, mas quando migram para dentro de tecidos tornam-se macrófagos, células versáteis que desempenham muitas atividades, tais como, a eliminação de bactérias e outros patógenos através da inflamação.

3.1.3.5 Reconhecimento de Antígenos

Para ordenar o reconhecimento e a resposta para um antígeno que é seu alvo, as células-T e as células-B carregam em suas superfícies receptores moleculares específicos.

Em células-B, o receptor é um protótipo de um anticorpo agarrado em sua superfície.

Em Células-T o receptor é mais complexo, estruturalmente. Ele possui a mesma estrutura de um anticorpo, apresentando um par de cadeias ligadas, que para funcionar necessita da ajuda das células T *helper*.

Entretanto, ambas as células não reconhecem o antígeno em seu estado natural; o antígeno inicialmente precisa ser fragmentado nas moléculas MHC. Após este processo, as células-T *helper* unem-se nas moléculas MHC classe II. Esta combinação expõe os antígenos para os macrófagos e para as células-B enquanto que algumas células-T citotoxinas reagem a outros antígenos ligando-se nas moléculas MHC classe I. O resultado é uma abordagem que proporcionam as células-T atuação somente em alvos específicos, evitando o contágio de outras células por contato direto ou pelo surgimento de secreções.

3.1.3.6 Imunidade Inata e Adquirida

Quando um organismo é exposto a uma moléstia, tanto as células-B como as células-T são ativadas e algumas retêm uma memória da moléstia. Ao reencontrar o mesmo antígeno o sistema imunológico (STITES; TERR, 1992) irá reconhecê-lo e destruí-lo rapidamente. Contudo, o grau e a duração da imunidade depende do tipo do antígeno, de sua quantidade e da forma que este entrou no corpo. A intensidade da resposta imunológica também depende da hereditariedade; em alguns indivíduos a resposta é mais fraca, já em alguns não existe nenhuma resposta.

As crianças recém-nascidas possuem respostas imunológicas mais tênues. Elas apresentam uma imunidade natural passiva decorrente do recebimento dos anticorpos IgG e IgA de suas mães. Esta mesma imunidade pode ser obtida através de vacinas que imunizam o indivíduo para um agente infeccioso específico.

Uma resposta imunológica pode ser ativada por uma infecção ou por vacinação.

A vacinação envolve a imunidade adaptativa. Seu princípio básico consiste na produção de vacinas a partir de patógenos que de uma forma inteiramente segura produzem uma resposta imunológica sem, contudo, ocasionar a enfermidade (JUCA, 2001).

3.1.4 Propriedades do Sistema Imunológico

O sistema imunológico apresenta quatro propriedades importantes ((HOFMEYR, 1999)): detecção, diversidade, aprendizado e tolerância, descritas a seguir.

3.1.4.1 Detecção

A detecção ou reconhecimento ocorre quando reações químicas são estabelecidas entre os receptores localizados na superfície de uma célula imunológica e os epítomos (STITES; TERR, 1992) que estão localizados na superfície de uma patogénia: a estrutura dos epítomos e dos receptores é tridimensional. A força entre um receptor e um epítomo é chamada de afinidade. Conseqüentemente, um receptor é específico pois que ele se liga somente a poucos epítomos.

As patogénias possuem diferentes e múltiplos epítomos refletindo sua estrutura molecular: então, diferentes linfócitos podem ser específicos para um simples tipo de patogénia. O modo de agir de um linfócito é regulado pela sua afinidade. Quando o número de receptores conectados excede um montante, o linfócito é ativado.

Um linfócito somente será ativado pelas patogénias caso seus receptores tenham afinidades suficientemente altas para uma particular estrutura de uma patogénia e se existir um número suficiente de patógenos no espaço de um linfócito.

3.1.4.2 Diversidade

A detecção no sistema imunológico (SOMAYAJI; HOFMEYR; FORREST, 1997) (HOFMEYR, 1999) está relacionado com os elementos *nonsel* do organismo, então, o sistema imunológico deve ter uma diversidade suficiente de receptores para assegurar que alguns de seus linfócitos possam reagir a determinado elemento patogênico. A geração de um repertório suficientemente diverso é um problema, porque o corpo humano não produz tantas proteínas quanto os possíveis epítomos patogênicos.

A solução adotada pelo sistema imunológico é renovar continuamente os linfócitos, gerando-os diariamente, de modo que no prazo de dez dias o conjunto de linfócitos de qualquer ser é totalmente renovado. Esta operação eleva a proteção oferecida pelo sistema imunológico.

3.1.4.3 **Aprendizado**

O sistema imunológico deve ser capaz de detectar e eliminar rapidamente as patogenicias, porque o crescimento destas é exponencial.

O sistema imunológico incorpora princípios que possibilita ao linfócito aprender (SOMAYAJI; HOFMEYR; FORREST, 1997) (HOFMEYR, 1999) ou adaptar-se as estruturas de proteínas estrangeiras específicas e de recordar-se destas estruturas rapidamente. Estes dois princípios são implementados pelas células-B. Quando uma célula-B é ativada (BROSTOFF; M; MALE, 2002), ela produz cópias de si mesma e estes clones são produzidos através de divisão celular.

Este processo de clonagem está sujeito a sofrer um processo de mutação, no qual podem ser geradas células-B que possuem diferentes receptores da célula original e assim diferentes afinidades com as patogenicias. Quanto maior a afinidade da célula-B com a patogenia presente, mais semelhante irá ser seu clone. Quanto maior afinidade, mais eficiente o linfócito será na captura de uma patogenia específica, conseqüentemente mais rapidamente a infecção será eliminada.

Isto é particularmente importante quando o sistema imunológico está combatendo a replicação de uma patogenia: nesta situação, ocorre uma corrida entre a reprodução da patogenia e a reprodução das célula-B. Uma resposta imunológica vencedora é resultado da proliferação das células-B em uma taxa de afinidade superior que as patogenicias estrangeiras que estimularam a resposta imunológica.

A retenção da informação codificada nas células-B constitui a memória do sistema imunológico. Caso a mesma patologia seja encontrada no futuro, a população de células-B pré-adaptadas reagem rapidamente. Uma das conseqüências do aprendizado e da memória é o fato de que duas situações no sistema imunológico podem ser distinguidas: a reação do sistema imunológico quando encontra uma patogenia que já tenha sido encontrada anteriormente. No primeiro contato, durante o qual as células-B irão adaptar-se à patogenia, o processo é relativamente lento.

Entretanto, o segundo é rápido, porque o sistema imunológico retém a memória da patogenia. A reação posterior não é somente desencadeada pelo reencontro da mesma patogenia, mas também pela infecção com uma nova patogenia que seja similar à patogenia conhecida.

3.1.4.4 Tolerância

As moléculas que sinalizam a célula como *self* são codificadas por um grupo de genes que estão contidos em seções de um cromossomo conhecido como MHC. As moléculas do MHC determinam quais antígenos um indivíduo pode reagir e com que força. O MHC permite que as células-B, as células-T e os fagócitos, reconheçam uns aos outros e comuniquem-se entre si.

As proteínas MHC podem ser classificadas em dois grupos (BROSTOFF; M; MALE, 2002): Um grupo de proteínas, os anticorpos MHC de classe I, constituem as características de *self* que estão presentes em todas as células do corpo.

Outro grupo de proteínas, conhecidas como MHC de classe II estão presentes nas células-B, fagócitos e em outras células responsáveis pelos antígenos para ajudar as células-T.

3.2 O Sistema Imunológico Artificial

O Sistema Imunológico Humano, conforme abordado nos tópicos anteriores, subdivide-se em vários níveis de defesa, que constituem barreiras progressivas de proteção diante da invasão do organismo por elementos causadores de doenças.

No nível mais elementar está a pele e as mucosas, que tentam impedir fisicamente a instalação das doenças no corpo. Na seqüência, encontra-se a temperatura e o PH, que à medida em que tornam o ambiente impróprio para a multiplicação do elemento invasor, formam um obstáculo fisiológico. Todavia, uma vez que essas barreiras tenham sido suplantadas, os sistemas imunológicos inato e adquirido são acionados visando deter a investida das patogenias.

O relacionamento entre a segurança computacional e a imunologia foi introduzida em 1987 através da concepção do termo vírus de computador por COHEN (MACHADO, 2005) *apud* (COHEN, 1987). Posteriormente os trabalhos de (FOR-

REST; PERELSON, 1994) e (KEPHART, 1994) estabeleceram uma conexão entre estes assuntos.

A partir disso foram elaborados diferentes planos, aplicáveis no contexto de segurança computacional, combinando elementos do sistema imunológico e arquiteturas de sistemas de computadores. As arquiteturas propostas foram: **Proteção de Dados Estáticos, Proteção de Processos Ativos em um Computador, Protegendo uma Rede Confiável de Computadores, e Protegendo uma Rede de Computadores Mutuamente Disponíveis e Confiáveis**. A descrição destas arquiteturas podem ser obtidas em (SOMAYAJI; HOFMEYR; FORREST, 1997).

O projeto de Sistemas Imunológicos Artificiais utiliza-se desta base conceitual, aliada a reprodução no ambiente artificial de algumas características essenciais do Sistema Imunológico Humano (SOMAYAJI; HOFMEYR; FORREST, 1997), quais sejam:

1. **Proteção Distribuída:** o Sistema Imunológico compreende milhões de componentes distribuídos através do corpo. A proteção do organismo é executada pela interação, em âmbito local, destes componentes, ou seja, não existe um controle centralizado, conseqüentemente não existe um ponto único de falha no sistema.
2. **Diversidade:** há o predomínio da diversidade no sistema imunológico afinal, não existem dois indivíduos com o mesmo sistema imunológico. Esta particularidade reduz os índices de vulnerabilidade da população diante da ocorrência de uma mesma patogenia. A personalização do sistema imunológico e a diferença existente entre seus componentes, podem oferecer diversos modelos de reconhecimento a uma gama de patogenias.
3. **Robustez:** a arquitetura do sistema imunológico prevê que a perda de alguns componentes causará leves desequilíbrios na sua funcionalidade, isto porque os componentes do sistema são numerosos e redundantes. A possibilidade de reposição de componentes, associada com o controle descentralizado, propicia ao sistema imunológico robustez e tolerância à falhas.
4. **Adaptabilidade:** o sistema imunológico é capaz de rapidamente e com precisão, identificar e decodificar patogenias respondendo de uma forma efetiva e veloz.

A resposta efetiva a um patógeno, em um segundo contato, é sempre superior ao da primeira exposição.

5. Memória: a memória imunológica permite que o sistema imunológico responda rapidamente quando o corpo é exposto uma segunda vez a uma mesma patogenicia. Após ativada, a memória mantém-se ativa durante todo o tempo de vida de um organismo. Este princípio evidencia que o sistema imunológico implementa a detecção por abuso, o que proporciona o monitoramento das patogenias a que o organismo é acometido.
6. Especificação: a definição de *self* adotada pelo sistema imunológico é empírica. Os componentes do conjunto *self* são determinados pelo monitoramento das proteínas que estão presentes no corpo, em um dado instante, sem a preocupação com os demais elementos que deveriam estar disponíveis.
7. Flexibilidade: a alocação de recursos para a proteção do corpo no sistema imunológico é flexível.
8. Crescimento: o sistema imunológico também permite a expansão. Para combater uma infecção séria, o sistema imunológico providencia a produção de novos componentes. Todos os componentes se comunicam e interagem localmente. O aumento na quantidade de elementos somente acontece no local em que ocorre a infecção, o que ameniza sobrecargas no sistema decorrente da elevação populacional de componentes.
9. Detecção por Anomalia: utilizando-se de mecanismos variados, o sistema imunológico é capaz de detectar uma diversidade de patogenias. O mecanismo mais eficaz baseia-se na detecção por abuso todavia, é a detecção por anomalia que é ativada no primeiro contato com um patógeno. E por essa razão a detecção por anomalia é essencial para a sobrevivência do organismo, uma vez que o organismo irá encontrar uma grande variedade de patogenias durante o seu tempo de vida.

3.2.1 Arquitetura Para um Sistema Imunológico Artificial

Dentre as propostas de arquiteturas para Sistemas Imunológicos Artificiais citadas na seção 3.2, página 52, selecionamos a arquitetura denominada **Prote-**

gendo uma Rede Confiável de Computadores (SOMAYAJI; HOFMEYR; FORREST, 1997). Esta arquitetura propõe que cada computador, em uma rede de computadores, corresponde a um órgão de um organismo. Neste modelo, cada processo representa uma célula e um indivíduo é considerado uma rede de computadores mutuamente confiáveis. A atitude proposta por esta arquitetura proporciona a eliminação rápida, em toda a rede, das anomalias detectadas inicialmente em um único computador.

Nela, o sistema imunológico inato é composto pelos mecanismos de segurança de um computador, associado a mecanismos de segurança da rede. O sistema imunológico adquirido é implementado por processos linfócitos que migram entre computadores e que podem ser implementados como agentes móveis.

Estes atuam em um processo de detecção por anomalia, o qual determina que se um processo tiver seu funcionamento alterado em decorrência de um ataque ou esteja sob um ataque, o linfócito deve agir suspendendo, encerrando ou reiniciando o processo.

A detecção de anomalias baseia-se na observação de divergências no uso de serviços monitorados. Os componentes *self*, nessa abordagem, podem ser definidos como o conjunto de registros do sistema, extraídos em uma situação normal; enquanto que o *nonself* seria o oposto, isto é, seriam os registros catalogados por ocasião de invasão.

A arquitetura é baseada em *host*, uma vez que cada servidor efetua análise de registros de seus *logs*. Além disso, é distribuída, pois que não há restrições quanto a quantidades de servidores disponíveis na rede, nem quanto aos componentes responsáveis pela detecção, análise, memorização e geração de respostas imunológicas descentralizadas. Estas respostas podem ser propagadas pela rede, o que supõe uma ação cooperativa entre os elementos que a compõem.

As respostas imunológicas são elaboradas segundo duas metodologias (MACHADO, 2005), resultado da classificação do evento como ataque ou tentativa de ataque a um dos serviços monitorados. A primeira define uma resposta passiva, iniciada por um *host* seguro, o qual consiste no envio de um email de alerta aos administradores relatando a ocorrência de eventos intrusivos. A segunda opção, denominada ativa, é implementada por um agente móvel que suspende o serviço sob ataque.

Esta arquitetura, tradicionalmente, faz parte de uma estrutura maior que distribui dispositivos de segurança em vários níveis. No primeiro nível, em analogia à pele e às mucosas (barreira física do sistema imunológico inato), está um *firewall* que controla o acesso à rede de computadores.

No segundo nível, (equivalente às barreiras químicas), estão dispostos mecanismos de controle e monitoração da rede. E, no terceiro nível, em atividades correlatas às existentes no sistema imunológico inato que compreendem a detecção de patógenos no organismo, estão os mecanismos tradicionais de segurança tais como sistemas de arquivos, permissões de arquivos, controle de acessos, políticas de segurança de computadores e serviços e a geração de registros de atividades. Neste nível encontra-se, também, o sistema imunológico adaptativo, responsável pelos processos de reconhecimento e elaboração de respostas imunológicas.

Essas ações são realizadas através das imunizações (*patches* e corretivos) em conjunto com um sistema computacional utilizando a tecnologia de agentes móveis. Cabe aos agentes móveis a identificação de patógenos, a manutenção da memória imunológica e o estabelecimento de respostas imunológicas.

4 METAMODELO MOF PARA UM MODELO DE SISTEMA IMUNOLÓGICO ARTIFICIAL

Este capítulo propõe uma definição de um metamodelo para um Sistema Imunológico Artificial segundo os princípios abordados em capítulo anterior. Inicialmente, apresentamos as convenções e notações utilizadas, seguido da proposta de metamodelo e sua validação. Finalizando, tecemos algumas considerações sobre o metamodelo criado e o processo de metamodelagem MOF.

4.1 Convenção Utilizada Na Representação Do Metamodelo

Constituem a base do metamodelo do SIA, as classes, suas propriedades e as associações. Uma classe, no contexto deste trabalho, é considerada como uma especificação abstrata de metaobjetos incluindo o seu estado, suas interfaces e comportamento mínimo, razão pela qual a sua definição não necessariamente requer a especificação de métodos para implementar o comportamento de metaobjetos.

Os construtores do metamodelo serão descritos em uma notação textual também em UML, pois que até a conclusão deste trabalho não existiam ferramentas específicas para modelagem MOF. Todavia, utilizamo-la, conscientes dos desalinhamentos existentes entre ambas as linguagens, conforme relatado na seção 2.2.7 localizada na página 39.

A notação textual utilizada nomeia as classes do metamodelo, descreve suas associações, seu tipos de dados, suas operações, restrições e atributos de acordo com

a estrutura a seguir especificada:

- Nome da classe: toda classe possui esta seção definindo o seu nome, o qual pode ser seguido pela palavra "abstract" indicando se ela é abstrata ou "substitute name", indicando um nome alternativo para mapeamento.
- Superclasses: este tópico lista as superclasses da classe que está sendo descrita.
- *ContainerElements*: este é um item opcional, que, se presente, indica que instâncias da classe atuam como contêiner para instâncias de outras classes. Isto implica em definir uma agregação por composição entre as classes envolvidas. Poderão ser contidas classes que fazem parte do mesmo pacote ou provenientes de pacotes dependentes.
- *Attributes*: nesta seção são listados os atributos pertencentes à classe, com seu tipo, visibilidade e sinalizadores indicando se o atributo é ou não modificável, se ele é derivado e qual o seu escopo: *instance level* (padrão) ou *class level*. Atributos herdados não são relacionados. No MOF, o valor padrão para visibilidade é público. A ausência da seção significa que a classe não possui atributos.
- *References*: este cabeçalho lista todas as referências disponíveis para a classe e sua ausência simboliza que a classe não permite que se navegue de uma instância para outra. As referências herdadas não são listadas. Uma referência possui um nome e campos para indicar a classe e a multiplicidade da extremidade oposta, o relacionamento e a orientação (sentido) da navegabilidade.
- *Operation*: este cabeçalho relaciona as operações para a classe, com exceção das herdadas. A ausência deste tópico indica que não há operações para a classe. As operações apresentam: um tipo de retorno, especificado em *Return type* com sua multiplicidade; um sinalizador opcional, que ativa *isQuery*; uma lista de parâmetros (para cada parâmetro são definidos o nome, a direção, tipo e multiplicidade) e uma lista de exceções que podem ser ocasionadas pela operação, cuja ausência determina que a operação não tem exceção.
- *Constraints*: este item relaciona as restrições aplicáveis à classe. Sua ausência significa que não há restrições para a classe, porém, se especificadas, as mesmas serão descritas de acordo com o seguinte:

- Descrição textual da restrição;
- XX nome da classe *restrição*;
- *format1*: nome da classe e um formato da restrição;
 - *format2*: idem - formato alternativo;
 - *evaluation policy*: definição da política de avaliação (imediata ou adiada);
 - **context** Contexto de aplicação da restrição em OCL;
 - **inv**: Descrição da restrição em OCL.

4.2 As Classes do Metamodelo SIA

As principais classes que compõem o metamodelo, suas funcionalidades, atributos, operações e relacionamentos estão resumidas nas subseções seguintes, enquanto que a descrição completa juntamente com as demais classes do metamodelo, encontram-se descritas no Apêndice A. O Metamodelo SIA é ilustrado pela Figura 7.

4.2.1 SIAElemento

A classe abstrata **SIAElemento** é a superclasse do metamodelo SIA. Ela generaliza algumas características que são comuns a vários dos elementos do imunológico artificial.

4.2.2 Computador

A classe **Computador**, no metamodelo, é análoga aos órgãos no organismo. Os órgãos (Computadores), contêm aglomerados de células (**Serviços**).

4.2.3 Serviço

A classe **Serviço** representa as células do organismo humano. As células constituem a base de todo sistema imunológico. No SIA, ela contém os serviços monitoráveis.

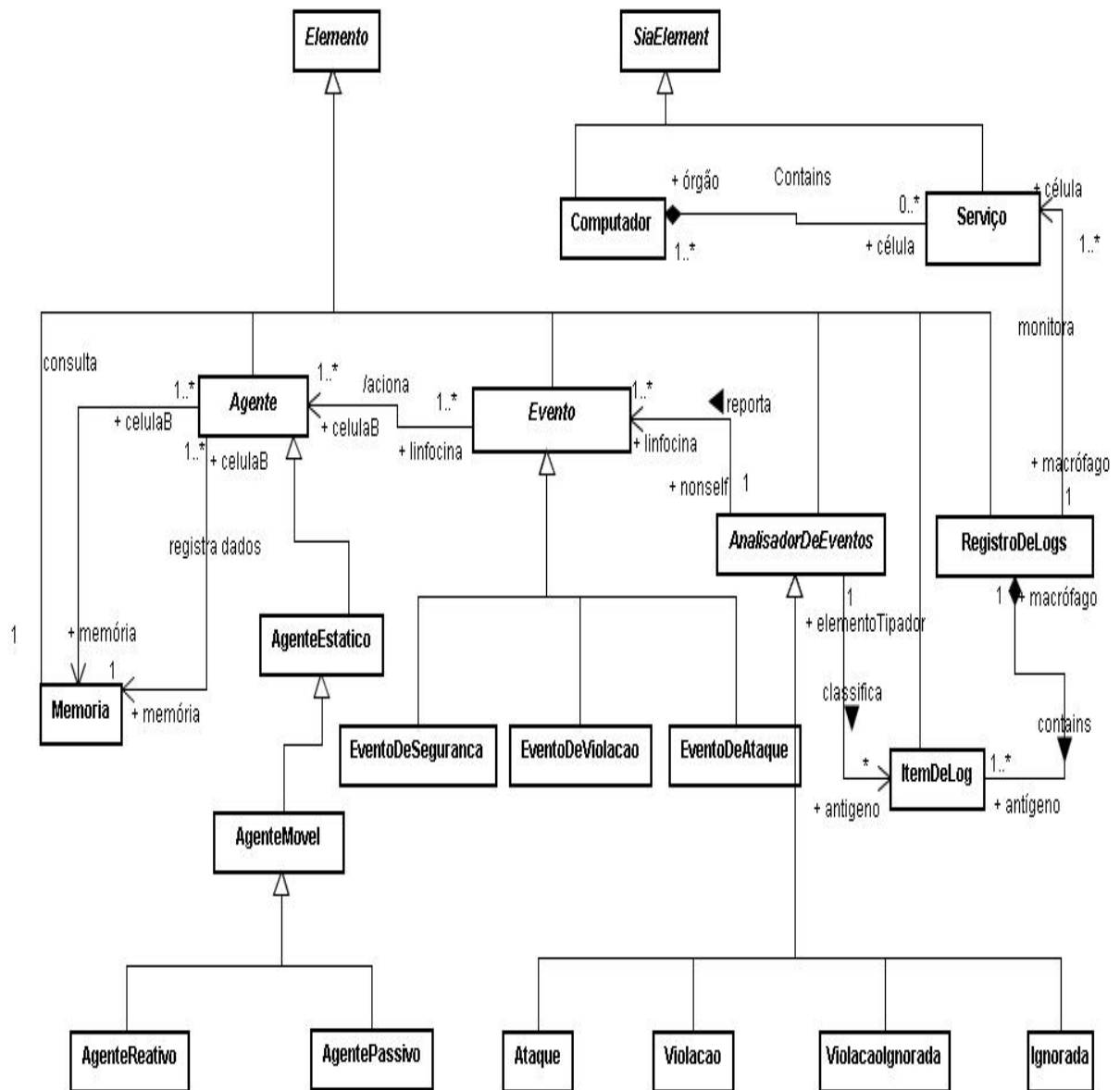


Figura 7: Metamodelo SIA.

4.2.4 Elemento

A classe abstrata **Elemento** define propriedade comum a várias outras classes no metamodelo.

4.2.5 RegistroDeLogs

A classe **RegistroDeLogs** define o registro de atividades das solicitações de **Servico**. Esta classe atua como um macrófago, ou seja, ela envolve os antígenos

(requisições de serviços), fragmenta-os em peptídeos(*strings*) e os expõe para as Células-*T*Helper representada no metamodelo pelo **AnalisadordeEventos**. Ela contém uma classe denominada **ItemDeLog**, que mantém as ocorrências de registros de atividades para um determinado **Serviço**.

4.2.6 ItemDeLog

A classe **ItemDeLog**, no metamodelo, que mantém as ocorrências de registros de atividades para um determinado **Serviço**.

4.2.7 AnalisadorDeEventos

A classe abstrata **AnalisadorDeEventos** atua no sistema como uma Célula *T*Helper. Ela é uma superclasse que possui diversas especializações. Suas especializações digerem os antígenos que foram detectados pelo macrófago (**RegistroDeLogs**) e os expõe a células especializadas do sistema adaptativo.

4.2.8 Evento

Evento é a superclasse das classes que atuam como linfocinas: **EventoDeViolação**, **EventoDeAtaque**, **EventoDeSegurança**. A função de uma linfocina é gerar um sinal, provocando uma reação de demais elementos no sistema imunológico. No ambiente computacional, a atualização em qualquer uma das três classes estimula a ação de células-B.

4.2.9 Agente

Esta classe abstrata descreve as características comuns a células B especialistas (anticorpos), as quais têm a responsabilidade de construir respostas imunológicas adaptativas. Ao ser ativada, são instanciados objetos da classe **AgentesEstáticos** para tratar os antígenos *nonsel*f.

4.2.10 Memória

Esta classe representa as células de memória do sistema imunológico, cuja função é armazenar informações sobre os antígenos, possibilitando uma resposta mais efetiva e rápida em uma exposição futura a uma mesma patogenia. As células de memória são analisadas pelos agentes para tomada de decisão quanto a reação imunológica a ser tomada, enquanto que cabe aos plasmócitos (agentes móveis) difundir essas informações para as demais células de memórias do organismo.

4.3 Em Direção à Validação do Metamodelo

Esta seção pretende dar uma visão de como se pode validar o metamodelo para sistemas imunológicos artificiais.

Modelos informais de sistemas imunológicos artificiais, foram definidos em JUCA(2001) e MACHADO(2005a). O modelo informal em JUCA(2001) diz respeito ao sistema imunológico inato, enquanto em MACHADO(2005a) considerou uma extensão ao trabalhos em JUCA(2001), abordando o sistema imunológico artificial inato e o adquirido.

Este trabalho construiu um metamodelo MOF que pode derivar modelos de sistemas imunológicos artificiais para representar formalmente os modelos informais dos trabalhos anteriores.

No nível de abstração mais baixo, os modelos em JUCA(2001) e MACHADO(2005) foram implementados e testados em ambientes reais de rede. As análises de resultados apresentadas em JUCA(2002) e MACHADO(2005a) mostram níveis aceitáveis, dentro dos quais, os sistemas de detecção de intrusão sendo construídos, podem funcionar. Embora tais implementações não tenham sido completas, mesmo assim, se pode verificar que as propriedades das partes implementadas, refletem a semântica das mesmas partes especificadas nos modelos de sistemas imunológicos artificiais propostos em JUCA(2001) e MACHADO(2005). As semânticas desses modelos refletem a semântica do metamodelo, como se pode aferir através do processo de software proporcionado pelo ambiente de desenvolvimento *NetBeans*¹.

¹www.netbeans.org

4.4 Considerações sobre a Metamodelagem MOF

Para elaboração desse metamodelo, foram observadas as regras de mapeamento que definem como um elemento do nível M2 será mapeado para o nível M1. Para tanto, foram avaliadas as necessidades de navegabilidade bidirecional, pois estas afetam o conteúdo das API geradas.

Também levou-se em conta, a atribuição de multiplicidade multivalorada, tanto de atributos quanto de extremidades de associação, porque, se a multiplicidade for definida como *ordered*, a API gerada utilizará uma interface usando listas, do contrário, utilizará coleções. Isso faz com que o uso adequado de atributos e associações multivalorados propicie resultados mais adequados ao metamodelo.

Outro aspecto relevante na metamodelagem com o MOF é o uso da agregação. O mapeamento de um relacionamento de agregação reflete no comportamento da API gerada, acrescentando-lhe propriedades diferenciadas.

Também a especificação de operações em um metamodelo difere da atividade tradicional de modelagem: como a proposta de um metamodelo é fornecer uma abstração para um domínio específico, porém com um alto grau de independência, operações que possam ser inferidas pelos próprios geradores devem ser evitadas, devendo-se constar apenas aquelas que não possam ser originadas a partir dos atributos e operações MOF declarados.

No que concerne às restrições de pré e pós-condições aplicáveis às operações, a especificação do MOF nada menciona sobre o mapeamento destas.

Observando-se estes aspectos, concebemos o metamodelo, que apresenta as características inerentes ao sistema imunológico artificial percorridas no capítulo anterior:

1. O metamodelo SIA disponibiliza a **proteção distribuída e a flexibilidade** através de inúmeros agentes móveis que percorrem redes de computadores contendo o avanço dos invasores;
2. Além disso, ele é aplicável a qualquer computador em uma rede, independente da diversidade de configurações de segurança existentes, aspecto comum em todo sistema imunológico.

3. A existência de vários agentes confere **robustez** ao metamodelo, permitindo que um sistema que o aplique se mantenha operante ainda que tenha perdido alguns componentes em um ataque.
4. A capacidade dos agentes móveis consorciados aos agentes de memória em reagir a novos e velhos invasores, provendo uma resposta rápida e adequada confere ao metamodelo a **adaptabilidade** e a **memória** disponível no sistema imunológico natural.
5. Enquanto que a atuação dos filtros fornece a **especificação de política implícita e a detecção por anomalias**, que prevê a distinção de antígenos *self* de *nonself*.
6. Finalizando, uma vez que é aplicável a qualquer computador na rede, além de, utilizando-se de uma propriedade do MOF, que permite a adição de extensões, o metamodelo também contempla o aspecto de crescimento, o qual determina que o sistema imunológico seja expansível.

Salientamos que, no metamodelo, utilizamos o MOF para representar conceitos que muitas vezes não são OO, como é o caso das configurações de segurança ou o registro de eventos, que são aplicativos tradicionalmente incorporados aos sistemas operacionais.

CONCLUSÕES E PERSPECTIVAS

Dentro do processo de desenvolvimento de software, o uso de metamodelos insere um nível mais elevado de abstração ampliando a possibilidade de reuso de modelos, especialmente em famílias de aplicações, pois que os metamodelos são criados de forma a serem independentes de plataforma o que os torna reutilizáveis em inúmeras aplicações dentro de um mesmo contexto.

O MOF é a linguagem padrão para metamodelagem, e como tal, seus metamodelos suprem a limitação que as linguagens de modelagem genéricas, tais como a UML, têm para expressar conceitos de domínios específicos. Todavia, para se consolidar como padrão nesta área, o mesmo deve ser disponibilizado aos desenvolvedores através do maior número de ferramentas de desenvolvimento livres possível.

Os metamodelos baseados em MOF são portáteis, em decorrência do uso de XMI (XML Metadata Interchange) e adequados à transformação automática de modelos para diferentes plataformas de desenvolvimento. A proximidade dos conceitos do MOF com os da UML favorece sua divulgação e aceitação, enquanto sua capacidade de extensibilidade prolonga o ciclo de vida das aplicações baseadas em seus metamodelos.

No contexto da engenharia de software, os metamodelos MOF podem ser gerados em ferramentas com suporte a metamodelagem e posteriormente importados para ambientes de desenvolvimento integrados compatíveis que geram, automaticamente, *frameworks* para algum domínio de aplicação, o que diminui o tempo de desenvolvimento e conseqüentemente aumenta a produtividade do desenvolvedor.

A adoção do Sistema Imunológico Artificial como domínio de aplicação dos conceitos de metamodelagem do MOF, oportunizou testar as capacidades deste na representação de conceitos complexos, portadores de características específicas concentradas como a diversidade, a proteção distribuição, a robustez e a flexibilidade,

sem contudo, restringir a possibilidade de futuramente adicionar novos conceitos, resultantes tanto de avanços tecnológicos quanto de descobertas científicas no campo da biologia.

Também, pode-se dizer que o uso do MOF na concepção do Metamodelo para um Sistema Imunológico Artificial propicia a geração de modelos variantes aplicados a sistemas de detecção de intrusão, tanto de redes convencionais quanto na pesquisa sobre redes *ad hoc* móveis.

Além da possibilidade acima, este trabalho oferece outras oportunidades de continuidade como, por exemplo, a criação de uma notação gráfica específica para representar o MOF, tornando-o independente da representação da UML. Tal aplicação poderia ser concebida na forma de *plugin* para alguma IDE *free*, o que facilitaria a sua disseminação, aceitação e uso.

A validação completa e a implementação do metamodelo SIA, utilizando-se dos princípios de transformação de modelos proposto pela MDA constitui, igualmente, uma atividade de prosseguimento desta dissertação.

Consideramos, ainda, que seja oportuno a condução de pesquisas que avaliem os possíveis resultados que possam ser produzidos mediante a aplicação de padrões de projeto na metamodelagem OO.

ANEXO A – Classes Do Modelo MOF

Abaixo estão descritas as demais classes pertencentes ao Modelo MOF.

Class: Uma *Class* representa um objeto classe. Seu único atributo é *isSingleton* que especifica se pode ou não existir mais de uma instância de objetos no nível M1, uma vez que as demais características triviais de uma classe, tais como, atributos, operações, referências, restrições, entre outros, são representados no MOF de forma independente.

Feature: A classe *Feature* estabelece características estruturais e comportamentais dos *ModelElements*. Os aspectos comportamentais são especificados pelas classes *Operation* e *Exception* e as estruturais pelas classes *Attribute* e *Reference*. *Feature* possui os atributos *scope* e *visibility*.

DataType: A *DataType* é uma especialização de *Classifier*, e atua como super-classe de todas as classes que são construtores ou representam tipos de dados MOF.

PrimitiveType: Esta classe representa um tipo de dado atômico, cujas instâncias, (seis ao todo), são usadas para representar tipos de dados em um metamodelo. As instâncias são: *Integer*, *Boolean*, *String*, *Long*, *Double* e *Float*.

CollectionType: A *CollectionType* é o construtor MOF para tipos de coleções. Ela propicia definição de tipos de dado cujos valores são coleções finitas de instâncias de outro tipo, para tanto são atribuídos limites inferiores e superiores para as coleções, e também propriedades de ordenação e de unicidade.

EnumerationType: Este é o construtor MOF para tipos de dados cujos valores enumerados são representados por *Strings*.

AliasType: Esta classe é o construtor de subtipos de quaisquer outras classes MOF ou tipo de dados. Estes subtipos são conhecidos como *Alias*.

Association: A finalidade da *Association* é representar uma associação no nível M2 . Cada instância de uma associação é denominada vínculo ou ligação, indica se há existência de uma conexão entre dois objetos classificadores. O MOF permite somente associações binárias, restringindo o número máximo de objetos participantes em um vínculo a dois. As associações contêm duas extremidades, cada uma delas relacionadas com um dos classificadores. Uma associação contém referência para estas extremidades. Seu único atributo é *isDerived*, o qual define uma associação derivada. Uma associação derivada não possui instâncias, uma vez que seus vínculos são resultantes de outras informações no metamodelo.

StructureField: A classe *StructureField* descreve os campos de uma instância da classe *StructureType*.

StructuralFeature: Esta classe estabelece características estáticas, também denominadas propriedades estruturais, de um *ModelElement*. As propriedades estruturais são definidas pelas suas subclasses *Attribute* e *Reference* .

Attribute: A classe *Attribute* define uma característica estrutural contendo valores. Todos os atributos das classes dos metamodelos MOF são por ela representados.

Package: Um pacote, na modelagem, é um construtor organizacional representando um mecanismo de propósito geral para a organização de elementos em grupos (BOOCH; JACOBSON; RUMBAUGH, 2000). No MOF, um pacote contém elementos de modelos relacionados que formam um modelo lógico. Todos os pacotes de metamodelos MOF são armazenados no repositório como instâncias de *Package*. Os pacotes podem ser utilizados de formas variadas através de:

- Aninhamento - nesta situação um pacote contém um aninhamento de pacotes. Este tipo de uso é obtido através da associação *Contains*.
- Herança - Nela um pacote herda de outro pacote através da associação *Generalizes*.

- Importação ou agrupamento - Neste caso, pacotes de outros espaços de nome são importados através da associação *Alias*.

StructureType: A *StructureType* atua como construtor MOF para tipos de dados estruturados. A sua estrutura é descrita pela classe *StructureFields*.

Reference: *Reference* representa, nos metamodelos MOF, as referências que uma classe pode fazer a outras, proporciona acesso ao conjunto de vínculos (*links*) existentes para uma associação. Seus dois atributos são *referencedEnd* e *exposeEnd*, e permitem respectivamente, especificar a extremidade de associação que referencia o outro classificador e a extremidade da associação referenciada pelo próprio classificador.

BehavioralFeature: Esta classe define os aspectos comportamentais de um *ModelElement*. Essas características são especificadas através de suas subclasses *Operation* e *Exception*.

Operation: Todas as operações das classes dos metamodelos MOF são representadas pela classe *Operation*. Seu único atributo e referência são *isQuery* e *exceptions*. O booleano *isQuery* estabelece se o comportamento da operação poderá ou não alterar o estado do objeto, no escopo da classe ou da instância. Enquanto que *exceptions* viabiliza a execução de exceções que tratam erros ou condições anormais que eventualmente possam ocorrer durante uma operação.

Parameter: Esta classe fornece uma forma de comunicação, através da transmissão de parâmetros, entre a classe *Operation* e outras classes que especificam comportamentos. Seus atributos são *direction* e *multiplicity*. O primeiro determina o propósito do parâmetro indicando se este seria de entrada ou saída ou ainda ambos. Enquanto a *multiplicity* declara a cardinalidade do parâmetro.

Exception: A classe *Exception* define um erro ou qualquer condição anormal que pode ser disparada durante a execução de uma operação. O conceito de exceções MOF é semelhante ao das linguagens de programação orientada a objetos.

AssociationEnd: A finalidade da *AssociationEnd* é representar as duas extremidades existentes em uma associação. Seus atributos mais significativos são:

- *isNavigable* - define se a extremidade da associação suporta ou não vínculos de navegação. A definição da navegação é importante pois, torna o objeto na outra extremidade da associação passível de acesso direto. A acessibilidade das associações somente é possível porque o objeto de origem armazena referências acerca do objeto destino.
- *aggregation* - indica a existência de uma associação com uma semântica especial. Neste caso, a extremidade da associação é definida ou como composição ou como compartilhada.
- *multiplicity* - atribui valores mínimos e máximos para ocorrência de vínculos entre objetos. Sua presença em uma extremidade de associação restringe a quantidade de instâncias nesta extremidade.

Import: Esta classe permite que um pacote faça uso de *ModelElements* definidos em outros pacotes pertencentes a outros espaços de nome. A importação é feita através da associação *Aliases*. Na importação, o pacote que recebe o objeto importado contém também o espaço de nome relativo ao pacote de que é originário tal objeto. Assim, tanto importador quanto importado se tornam visíveis uns aos outros, a menos que haja restrições quanto à visibilidade. Neste intuito, existe a referência *importedNamespace*, que obtém informações sobre o espaço de nome.

Constraint: A classe *Constraint* tem por finalidade especificar restrições ao comportamento ou estado de *ModelElements* de um metamodelo. As regras nela contidas são aplicadas a todas as instâncias do *ModelElement* ao qual esteja anexada. Estas regras são definidas através dos atributos da classe. Elas são representadas e aplicadas com uma expressão (*expression*), codificadas em qualquer forma, denotada em qualquer linguagem e esquema de codificação desde que este seja especificado no atributo *language*. A aplicação da restrição pode ser configurada de imediato (atributo *immediate*) ou adiada (atributo *deferred*). A classe tem a referência *constrainedElements* que proporciona acesso aos *ModelElements* restringidos.

Constant: As constantes têm a função de associar um nome a um valor constante e esta classe fornece um mecanismo para definição de valores constantes para tipos de dados simples. Seus valores estão limitados aos tipos definidos como

instâncias da classe *PrimitiveTypes*. O conceito de constantes MOF é similar ao utilizado pelas linguagens de programação.

Tag: Uma etiqueta (*Tag*) é o mecanismo que torna possível a extensão ou modificação de um metamodelo MOF. Uma etiqueta possui um nome que a identifica dentro do contêiner; um identificador (*Tag Id*) que diferencia o seu tipo (o MOF dispõe de algumas categorias de identificadores); uma coleção de zero ou mais valores a ela associados. Este par nome/valor pode então, ser anexado às instâncias de classes, disponibilizando um mecanismo de extensão que adiciona informação a um metamodelo. Seus atributos são:

- *tagID* - que especifica o significado da *Tag*;
- *values* - que contém *strings* ordenadas de valores para a *Tag*.

A atribuição de nomes para *tagIds*, deve observar os mesmos critérios de nomeação utilizados por Java para atribuição de nomes aos pacotes.

ANEXO B – Relacionamentos Do Modelo MOF

Abaixo estão descritos os demais relacionamentos do Modelo MOF.

CanRaise: Esta associação identifica as exceções que podem ocorrer durante a execução de uma operação. *CanRaise* representa o relacionamento entre as classes *Operation* e *Exception*. As suas extremidades são: *operation* que define o conjunto de operações que podem causar a exceção, e *except* que define o conjunto de exceções que podem ser executados pelas operações. Uma operação pode ocasionar zero ou mais exceções, e uma exceção podem ser disparadas por uma ou mais operações.

Constrains: Este relacionamento identifica as restrições aplicáveis a um *ModelElement*. Para tanto, a associação liga uma classe *ModelElement*, que é o elemento restringido, a uma classe *Constraint*, que detém as restrições. Cada *Constrains* valida um ou mais *ModelElements*. Suas extremidades são *constrains*, que identifica as restrições, e *constrainedElement* que define os elementos a que se aplicam as restrições.

AttachesTo: A finalidade da *AttachesTo* é listar *Tag(s)* associadas a um determinado *ModelElement*. Ela relaciona as classes *Tag* e *ModelElement*. As extremidades desta associação detêm informações sobre os *ModelElements* descritos ou modificados pela *Tag* anexada.

Aliases: Esta associação *Aliases* representa a importação de *ModelElements*. Ela relaciona as classes *Import* e *Namespace*. Suas extremidades são *importer*, que faz a importação do objeto com ou sem alterações no espaço de nome, e *imported* que contém o *Namespace* importado. A extremidade *importer*

pertence à classe *Import* enquanto que extremidade *imported* pertence à classe *Namespace*.

RefersTo: O relacionamento *RefersTo* liga as classes *Reference* e *AssociationEnd*. Ao criar um objeto do tipo *Association*, o MOF cria, também, dois objetos do tipo *AssociationEnds* representando as duas classes envolvidas na associação. Para poder acessar o objeto com o qual se relaciona, a classe necessita ter uma referência deste. Esta referência é fornecida pela associação *RefersTo* que liga as classes nível M3 *AssociationEnd* e *Reference*. Isto significa que, toda vez que no nível M2 uma classe se relaciona com outra classe, é criado um objeto *Reference* para que os objetos de ambas extremidades de associação (*AssociationEnd*), possam se acessar mutuamente, conforme exemplifica a Figura 8.

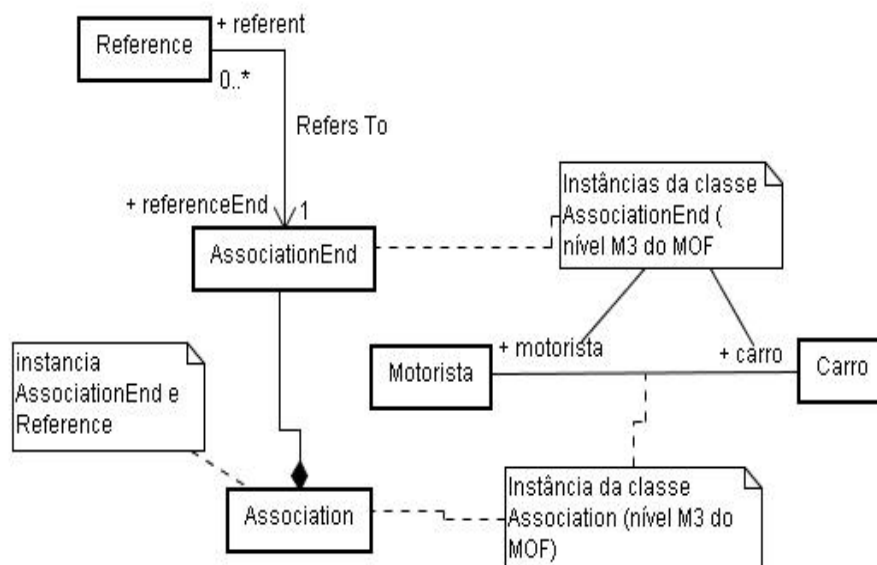


Figura 8: O relacionamento *RefersTo*

Nela, as classes *Motorista* e *Carro* mantêm uma associação com as extremidades de associação de mesmo nome (*motorista* e *carro*). A associação entre as classes é estabelecida através do instanciamento da classe nível M3 *Association*. Uma *Association* contém a classe *AssociationEnd*, e portanto, cria o objeto *AssociationEnd*. Para que as extremidades se tornem acessíveis é criado um objeto *Reference* e o relacionamento *RefersTo* que passa a identificar as extremidades. Assim, a função do relacionamento é indicar ambas as extremidades de associação.

Exposes: O relacionamento *Expose* também liga as classes *Reference* e *AssociationEnd*. Esta é uma associação derivada, ou seja, o estado do objeto *Reference* é dependente da *AssociationEnd* ao qual está relacionado. Suas extremidades são *referencedEnd* e *referent*.

APÊNDICE A – Classes do Metamodelo SIA

Este apêndice contém a descrição completa das classes do metamodelo para o Sistema Imunológico Artificial.

A.1 SIAElemento

A classe abstrata SIAElemento é a superclasse do metamodelo SIA. Ela generaliza algumas características que são comuns a vários dos elementos do imunológico artificial.

Nome da classe/tipo: SIAElemento *Abstract*.

Atributos

Nome/tipo: nome: *String*.

Multiplicidade: 1.

Finalidade: nomeia o elemento do modelo.

Nome/tipo: idRede: *String*.

Multiplicidade: 1.

Finalidade: Mantém informações que identificam da rede.

Nome/tipo: Domínio: *String*.

Multiplicidade: 1.

Finalidade: Define o nome o domínio da rede.

A.2 Computador

A classe Computador, no metamodelo, é análoga aos órgãos no organismo. Os órgãos (Computadores) contêm aglomerados de células (Serviços).

Nome da classe/tipo: Computador.

Superclasse: SIAElemento

Contained Elements: Serviço

References

célula - Referencia um conjunto de Serviços (Célula).

classe: Serviço.

defined by: *Contains::célula.*

multiplicity: 0..*.

inverse: Serviço::Computador.

A.3 Serviço

A classe Serviço representa as células do organismo humano. As células constituem a base de todo sistema imunológico. No SIA, ela contém os serviços monitoráveis. Sua descrição é:

Nome da classe/tipo: Serviço.

Superclasse: SIAElemento

Atributos

Nome/tipo: serviço: *String*.

Multiplicidade: 0..*.

Finalidade: contém o nome do serviço.

References

célula - Referencia um computador .

classe: Computador.

defined by: *Contains::*órgão.

multiplicity: 1..*.

inverse: Computador::Serviço.

macrófago - Referencia o RegistroDeLogs (Macrófago).

classe: RegistroDeLogs.

defined by: *monitora::*macrófago.

multiplicity: 1.

inverse: RegistroDeLogs::Serviço.

A.4 Elemento

A classe abstrata Elemento define propriedade comum a várias outras classes no metamodelo.

Nome da classe/tipo: Elemento *Abstract*.

Atributos

Nome/tipo: *log: String*.

Multiplicidade: 1..*.

Finalidade: Um *log* contém uma cadeia de caracteres (antígeno) extraída da solicitação de serviço para análise do SIA.

A.5 RegistroDeLogs

A classe **RegistroDeLogs** define o registro de atividades das solicitações de Serviço. Esta classe atua como um macrófago, ou seja, ela envolve os antígenos (requisições de serviços), fragmenta-os em peptídeos (*strings*) e os expõe para as Células-THelper, representada no metamodelo pelo **AnalisadordeEventos**. Ela contém uma classe denominada **ItemDeLog**, que mantém as ocorrências de registros de atividades para um determinado **Serviço**.

Nome da classe/tipo: RegistroDeLogs .

Superclasse: Elemento.

Contained Elements: ItemDeLog.

References

célula - fornece acesso aos serviços monitorados.

classe: Serviço (Célula).

defined by: monitora::célula.

multiplicity: 1..*.

inverse: Serviço::RegistroDeLogs

antígeno - Referencia um antígeno.

classe: ItemDeLog.

defined by: Contains::antígeno.

multiplicity: 1.

inverse: ItemDeLog::RegistroDeLog.

Constraints Um RegistroDeLogs não pode conter outro RegistroDeLogs.

C1 *RegistroDeLogssisBeSingleton*

- *format1: REGISTRODELOGSSISBESINGLETON*

- *format2*: *registrodelogssisbesingleton*
- *evaluation policy*: *immediate*
- *context* RegistroDeLogs
- *inv*: *self.isUnique implies self.isSingleton*

A.6 ItemDeLog

A classe **ItemDeLog**, no metamodelo, mantém as ocorrências de registros de atividades para um determinado **Serviço**.

Nome da classe/tipo: **ItemDeLog**.

Superclasse: **Elemento**

References

macrófago - Referencia um macrófago.

classe: **RegistroDeLog**.

defined by: *Contains::macrófago*.

multiplicity: 0..*.

inverse: **RegistroDeLog::ItemDeLog**.

elementoTipador: - permite ao **AnalisadorDeEventos** tipar os *logs* (antígenos) registrados.

classe: **AnalisadorDeEventos**.

defined by: *classifica::antígeno*.

multiplicity: 1.

inverse: **AnalisadorDeEventos::ItemDeLog**.

A.7 AnalisadorDeEventos

A classe abstrata **AnalisadorDeEventos** atua no sistema como uma Célula *THelper*. Ela é uma superclasse que possui diversas especializações. Estas especializações digerem os antígenos que foram detectados pelo macrófago (**RegistroDeLogs**) e os expõe a células especializadas do sistema adaptativo.

Nome da classe/tipo: **AnalisadorDeEventos** (Célula *THelper*).

Superclasse: Elemento.

Atributos

Nome/tipo: nome: palavra *String*.

Multiplicidade: 0..*/ ordered.

Finalidade: constitui uma base ordenada de dados que caracteriza eventos que devem ser registrados.

References

tipo - Referência que fornece as representações de tipos disponíveis para **ItemDeLog**.

classe: **ItemDeLog** (antígeno).

defined by: classifica::antígeno.

multiplicity: *.

inverse: **ItemDeLog::AnalisadorDeEventos**.

linfocina - permite reportar o antígeno *nonsel*f para a classe **Evento** e suas especializações, as quais são responsáveis pela ativação do sistema imunológico adaptativo através da emissão de linfocinas (sinais químicos).

classe: **Evento** (Linfocina).

defined by reporta::linfocina.

multiplicity: 1..*.

inverse: Evento::AnalisadorDeEventos.

Constraints Um elementoTipador fornece visibilidade a um único tipo.

C2 Filtro *TypeMustBeVisible*

- *format1:* *TYPEMUSTBEVISIBLE*
- *format2:* *typemustbevisible*
- *evaluation policy:* *deferred*
- **context** AnalisadorDeEventos
- **inv:** `self.isVisible(self.type)`

A.8 Evento

Evento é a superclasse das classes que atuam como linfocinas: **EventoDeViolação**, **EventoDeAtaque**, **EventoDeSegurança**. A função de uma linfocina é gerar um sinal, provocando uma reação dos demais elementos no sistema imunológico. No ambiente computacional, a atualização em qualquer uma das três classes estimula a ação de células-B.

Nome da classe/tipo: **Evento** (Linfocina) *Abstract*.

Superclasse: **Elemento**.

References

nonsel - Expõe o antígeno *nonsel* para emissão das linfocinas adequadas a ativação do sistema imunológico adquirido.

classe: **AnalisadorDeEventos** (Célula-*T*Helper).

defined by `reporta::nonsel`.

multiplicity: 1.

inverse: AnalisadorDeEventos::Evento.

célulaB - referência que permite ativar o sistema imunológico adaptativo.

classe: Agente (CélulaB).

defined by aciona::célulaB.

multiplicity: 1..*.

inverse: Agente::Evento.

A.9 Agente

Esta classe abstrata descreve as características comuns a células B especialistas (anticorpos), as quais têm a responsabilidade de construir respostas imunológicas adaptativas. Ao ser ativada, são instanciados objetos da classe **AgenteEstáticos** para tratar os antígenos *nonself*.

Nome da classe/tipo: Agente (CélulaB) *Abstract*.

Superclasse: Elemento.

Operações

Nome/tipo: geraAnticorpo.

Finalidade: operação encarregada de gerar anticorpos adequados ao antígeno detectado.

Nome/tipo: avaliaLinfocina.

Finalidade: operação encarregada de avaliar qual foi a linfocina acionada.

References

linfocina - Inspecciona os eventos (linfocinas) gerados, analisando-os e gerando respostas imunológicas.

classe: Evento.

defined by aciona::linfocina.

multiplicity: 1..*.

inverse: Evento::Agente.

memória - Referência que proporciona acesso a base de dados sobre as patogenias cadastradas.

classe: Agente.

defined by consulta::memória.

multiplicity: 1.

inverse: Memória::Agente.

memória - Referência que permite o registro de informação na base de dados de patogenias.

classe: Agente.

defined by registra::memória.

multiplicity: 1.

inverse: Memória::Agente.

A.10 Memória

Esta classe representa as células de memória do sistema imunológico, cuja função é armazenar informações sobre os antígenos, possibilitando uma resposta mais efetiva e rápida em uma exposição futura a uma mesma patogenia. As células de memória são analisadas pelos agentes estáticos para tomada de decisão quanto a reação imunológica a ser tomada, enquanto que cabe aos plasmócitos (agentes móveis) difundir essas informações para as demais células de memórias do organismo.

Nome da classe/tipo: Memória.

Superclasse: Elemento.

References

célulaB - Referência que permite a difusão de informações sobre as patogenias cadastradas.

classe: Memória.

defined by consulta::célulaB.

multiplicity: 1..*.

inverse: Agente::Memória.

célulaB - Referência que permite o registro de informação na base de dados sobre patogenias.

classe: Memória.

defined by registradados::célulaB.

multiplicity: 1..*.

inverse: Agente::Memória.

Constraints Só existe uma única memória no sistema.

C3 Memória *isBeSingleton*

- *format1:* MEMÓRIAISBESINGLETON
- *format2:* memóriaisbesingleton
- *evaluation policy:* immediate
- *context* Memória
- *inv:* self.isUnique implies self.isSingleton

A.11 Ataque

Representa um tipo de filtro que detecta antígenos específicos. A classe Ataque contém um conjunto de palavras chaves que caracterizam os receptores de célula *THelper*, os quais são capazes de identificar vírus ou células cancerígenas, caracterizadas computacionalmente por ataques a servidor ou serviço.

Nome da classe/tipo: Ataque.

Superclasse: AnalisadorDeEventos

A.12 Violação

Representa um tipo de filtro que detecta antígenos de violação, ou seja, a classe contém um conjunto de palavras chaves que caracterizam os receptores de célula *T*Helper capazes de reconhecer ações de violação às políticas de segurança do sistema.

Nome da classe/tipo: Violação.

Superclasse: AnalisadorDeEventos

A.13 Ignorada

Esta classe representa um tipo de filtro que detecta antígenos que não representam perigo ao SIA. Sua base de dados é constituída de palavras chaves que caracterizam os receptores de célula *T*Helper que detectam a presença de microorganismos inofensivos às políticas de segurança do sistema.

Nome da classe/tipo: Ignorada.

Superclasse: AnalisadorDeEventos.

A.14 ViolaçãoIgnorada

Esta classe representa um tipo de filtro que detecta antígenos que não representam perigo ao SIA. Ela contém um conjunto de palavras chaves que caracterizam os receptores de célula *T*Helper que detectam ações inofensivas ao sistema, por serem de natureza *self*.

Nome da classe/tipo: ViolaçãoIgnorada.

Superclasse: AnalisadorDeEventos

A.15 **EventoDeViolação**

Esta classe emite sinais de alerta indicando que houve um evento de Violação de segurança, ativando a produção de anticorpos adequados para este tipo de antígeno.

Nome da classe/tipo: `EventoDeViolação`.

Superclasse: `Evento`.

A.16 **EventoDeAtaque**

A função desta classe é gerar sinais de alerta indicando que houve um evento de ataque, ativando a produção de anticorpos adequados para este tipo de antígeno.

Nome da classe/tipo: `EventoDeAtaque`.

Superclasse: `Evento`

A.17 **EventoDeSegurança**

A classe **EventoDeSegurança** emite sinais de alerta indicando que ocorreu um evento de violação das políticas de segurança do sistema, ativando a produção de anticorpos adequados para este tipo de antígeno.

Nome da classe/tipo: `EventoSegurança`.

Superclasse: `Evento`

A.18 **AgenteEstático**

A classe **AgenteEstático** (Anticorpo) é um monitor de evento. Um monitor de evento interage com objetos `Evento` (`Linfocina`) verificando a ocorrência de novos eventos de ataque ou segurança. Quando ativado (evento detectado), clona-se e passa por um processo de mutação.

Nome da classe/tipo: **AgenteEstático.**

Superclasse: **Agente.**

Operações

Nome/tipo: *isviolation.*

Finalidade: avalia se a linfocina gerada é uma violação. Caso verdadeiro ativa os agentes móveis.

Nome/tipo: *clonarse.*

Finalidade: operação que produz um agente estático com propriedades especiais (agentes móveis).

A.19 AgenteMóvel

Esta classe representa os plasmócitos do sistema imunológico, cuja função é controlar e eliminar o invasor.

No sistema computacional, eles analisam a intensidade da violação e geram resposta computacionais visando eliminar o antígeno. Um agente móvel pode se deslocar na rede para realizar a resposta imunológica adequada à violação recebida.

Nome da classe/tipo: **AgenteMóvel.**

Superclasse: **AgenteEstático**

Atributos

Nome/tipo: grauViolação int.

Multiplicidade: 1.

Finalidade: contém um valor que permite analisar o grau de violação ocorrido visando efetuar a resposta imunológica adequada.

Nome/tipo: origem *String.*

Multiplicidade: 1.

Finalidade: armazena o endereço de origem do agente móvel.

Nome/tipo: destino *String*.

Multiplicidade: 1.

Finalidade: armazena o endereço de destino do agente móvel.

Nome/tipo: imunoG *Boolean*.

Multiplicidade: 1.

Finalidade: mantém o estado do AgentePassivo, se verdadeiro o agente é ativado e inicia resposta imunológica.

Operações

Nome/tipo: analisaAvaliação.

Finalidade: avalia o grau de violação e ativa a resposta imunológica adequada, através de suas especializações.

Nome/tipo: disparaAgente.

Finalidade: envia o agente para um endereço seguro na rede.

Nome/tipo: retornaAgente.

Finalidade: o agente retorna ao seu endereço de origem.

A.20 AgenteReativo

Esta classe representa anticorpos do tipo IgG eficientes no combate a tipos específicos de microorganismos invasores. Computacionalmente ele atua deslocando-se até o servidor que está sendo atacado e encerra o serviço que disponibiliza a conexão.

Nome da classe/tipo: `AgenteReativo`.

Superclasse: `AgenteMóvel`

Atributos

Nome/tipo: serviço `Serviço`.

Multiplicidade: 1.

Finalidade: contém informações sobre o serviço que será finalizado.

Operações

Nome/tipo: `finalizarServiço`.

Finalidade: finaliza o serviço que foi violado.

A.21 `AgentePassivo`

Esta classe representa os anticorpos IgM eficiente no combate a bactérias. Computacionalmente ele envia *emails* administrativos, a partir de uma máquina segura, alertando sobre a violação.

Nome da classe/tipo: `AgentePassivo`.

Superclasse: `AgenteMóvel`

Atributos

Nome/tipo: serviço `Serviço`.

Multiplicidade: 1.

Finalidade: contém informações sobre o serviço que a ser finalizado.

Nome/tipo: emailAdm *String*.

Multiplicidade: 1..*.

Finalidade: contém uma lista de endereços de administradores de redes que devem receber email alertando-os sobre a violação ocorrida.

Operações

Nome/tipo: nome: enviarEmail.

Finalidade: envia email aos administradores da rede alertando sobre o evento.

APÊNDICE B – Glossário

Anticorpos - Substância específica, com os caracteres da gamaglobulina, produzida pelo organismo, para defesa contra substâncias estranhas capazes de produzir moléstias.

Antígeno - Substância orgânica nociva, de natureza protéica, que, inoculada no organismo, é capaz de desenvolver a formação de um antagonista específico de defesa, o anticorpo.

Célula B - São pequenas células de glóbulos brancos produzidas na medula óssea, responsáveis por respostas imunológicas.

Célula T - São pequenas células de glóbulos brancos amadurecidas no Timo, responsáveis por detecção e reconhecimento de antígenos.

Célula THelper - Conjunto de Células T que atuam na distinção de antígenos *self* e *nonself*. São responsáveis pela emissão de linfocinas (sinais químicos) que ativam as Células B.

Epítopo - Protuberâncias existentes na superfície dos antígenos, as quais permitem o seu reconhecimento e o ativamento de uma resposta imunológica.

Fagócitos - São grandes células brancas que envolvem e digerem microorganismos invasores como bactérias ou parasitas.

Imunoglobulina - Anticorpos.

Linfócitos - São pequenas células de glóbulos brancos produzidas na medula óssea e timo.

Macrófagos - São células versáteis que desempenham muitas atividades, tais como, a eliminação de bactérias e outros patógenos através da inflamação.

Major Histocompatibility Complex (MHC)- Grupo de genes que estão contidas em seções de um cromossomo específico. Estão presentes em todas as células e marcando-as como *self*.

Medula Óssea - Substância mole que se encontra no interior dos ossos. Atua como fonte de todas as células sanguíneas.

Metadado - Dados que descrevem outros dados.

Metaobjeto - Referente a uma instância de classe definida em um nível superior ao do objeto.

Metalinguagem - Linguagem que descreve a sintaxe de outra linguagem.

Meta - Termo que indica a descrição de algo.

Metamodelagem - Ato de elaborar um metamodelo.

Micróbio - Ser unicelular, animal ou vegetal, com dimensões microscópicas; inclusos bactéria, fungos, viroses, protozoários.

Monócitos - Categoria de fagócitos que circulam no sangue, ao migrar para dentro de tecidos tornam-se macrófagos.

Monoespecificidade - Propriedade de um linfócito com todos os receptores idênticos, em tornar-se específico para um determinado conjunto de epítopos.

Nonsel - Células ou moléculas de organismos estrangeiros não pertencentes ao hospedeiro. Estas células desencadeiam uma resposta imunológica.

Órgãos Linfóides - Órgãos do sistema imunológico nos quais os linfócitos desenvolvem-se.

Patogenia - Parte da patologia que estuda a origem das doenças.

Patogênico - Que provoca doença.

Patógeno - Elemento causador de doenças.

Self - Células do próprio corpo.

Timo - Órgão linfóide no qual os linfócitos se reproduzem e amadurecem.

Vasos Linfáticos - componentes dos órgãos linfóides que transportam os linfócitos para os órgãos do sistema imunológico e para o fluxo sanguíneo.

Referências

- ATKINSON, C.; KUHNE, T. The role of metamodeling in mda. In: INTERNATIONAL WORKSHOP IN SOFTWARE MODEL ENGINEERING. *Proceedings*. Dresden: WiSME UML 2002, 2002. p. 5.
- BEZIVIN, J. From object composition to model transformation with the mda. In: 39TH INTERNATIONAL CONFERENCE AND EXHIBITION ON TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS: TOOLS 39. *Proceedings*. Santa Barbara: Institute Of Electrical and Eletronics Engineer, 2001. v. 1, p. 350–355.
- BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *UML Guia do Usuário*. Rio de Janeiro: Campus, 2000.
- BROSTOFF, J.; M, I. R.; MALE, D. K. *Imunologia*. 6^a. ed. São Paulo: MANOLE, 2002. 1-115 p.
- BUDINSKY, F. et al. *Eclipse Modeling Framework*. Boston: Addison Wesley, 2003.
- CASTRO, L. N. de. *Engenharia Imunológica: Desenvolvimento e Aplicação de Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais*. Tese (Tese de Doutorado em Engenharia da Computação) — Departamento de Engenharia de Computação e Automação Industrial - Universidade Estadual de Campinas, Campinas, 2001.
- CLARK, T. et al. *Applied Metamodelling. A Foundation for Language Driven Development*. Sheffield: Xactium, 2004.
- DASGUPTA, D. An immune agent architecture for intrusion detection: A general framework. In: WORKSHOP ON ARTIFICIAL IMMUNE SYSTEMS AND THEIR APPLICATIONS. *Proceedings...* Arlington, 1999.
- DASGUPTA, D.; FORREST, S. An anomaly detection algorithm inspired by the immune system. In: _____. *Artificial immune systems and their applications*. Berlin: Springer-Verlag Berlin and Heidelberg, 1999. cap. Part.III, p. 262–275.
- DAVIS, J. R. Model integrated computing: A framework for creating domain specific design environments. In: THE 6TH WORLD MULTICONFERENCE ON SYSTEMS, CYBERNETICS, AND INFORMATICS (SCI). *Proceedings*. Orlando: Multiconference on Systems, Cybernetics, and Informatics, 2002. p. 6.
- DHAESELEER, P. An immunological approach to change detection: Theoretical results. In: 9TH IEEE COMPUTER SECURITY FOUNDATIONS WORKSHOP. *Proceedings*. Dromquinna Manor: IEEE Computer Society Press, 1996. p. 9.

- FARSHCHIAN, B. A.; JAKOBSSON, S.; BERG, E. Coupling mda and parlay to increase reuse in telecommunication application development. In: WORKSHOP IN SOFTWARE MODEL ENGINEERING. *Proceedings*. Dresden: WiSME UML 2002, 2002. p. 5. Disponível em: <<http://www.metamodel.com>>.
- FLATER, D. Impact of model-driven standards. In: 35TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS02). *Proceedings*. Big Island: Institute Of Electrical and Eletronics Engineer, 2002. v. 9, p. 285–295.
- FORREST, S.; HOFMEYR, S. A. Architecture for an artificial immune system. *Evolutionary Computation*, Cambridge, v. 8, n. 4, p. 443–473, 2000. Disponível em: <<http://www.cs.unm.edu>>.
- FORREST, S.; PERELSON, A. S. Self-nonsel self discrimination in a computer. In: IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY. *Proceedings*. Oakland: IEEE Computer Society Press, 1994. p. 202–212.
- FOWLER, M.; SCOTT, K. *UML Essencial*. 2a. ed. Porto Alegre: Bookman, 2003.
- FRANKEL, D. S. *Model Driven Architecture: Applying mda to enterprise computing*. Indianapolis: Wiley Publishing, 2003. 80-187 p.
- FURLAN, J. D. *Modelagem de objetos através da UML - The Unified Modeling Language*. São Paulo: Makron Books, 1998. 23-31 p.
- GERBER, A.; RAYMOND, K. Mof to emf: There and back again. In: 2003 OOPSLA WORKSHOP ON ECLIPSE TECHNOLOGY EXCHANGE. *Proceedings*. Anaheim: Association for Computing Machinery - ACM, 2003. p. 60–64.
- HELMAN, P.; DHAESELEER, P.; FORREST, S. An immunological approach to change detection: Algorithms, analysis, and implications. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY. *Proceedings*. Oakland: IEEE Computer Society Press, 1996. p. 110–119.
- HOFMEYR, A. S.; FORREST, S. Immunity by design: An artificial immune system. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE. *Proceedings*. Orlando: Morgan Kaufmann, 1999. p. 289–296. Disponível em: <citeseer.ist.psu.edu/hofmeyr99immunity.html>.
- HOFMEYR, S. A. *An Immunological Model of Distributed Detection and its Application to Computer Security*. 135 p. Tese (Doctor Of Philosophy in Computer Science) — University of New Mexico, Albuquerque, 1999.
- HOLZ, E. A meta-model based approach for the combination of models in multiple languages. In: IASTED INTERNATIONAL CONFERENCE MODELLING AND SIMULATION MSO 2003. *Proceedings*. Banff: IASTED ACTA Press, 2003. p. 308–313.
- HOUAISS, A.; VILLAR, M. de S.; FRANCO, F. M. de M. *Dicionário Houaiss da Língua Portuguesa*. Rio de Janeiro: Objetiva, 2001. 1906-1907 p.

JCP. *Java Metadata Interface(JMI) Specification, Version 1.0*. [S.l.], Junho 2002. Disponível em: <<http://www.jcp.org/>>.

JUCA, K. R. L. *Um Sistema de Detecção de Intrusão baseado no Sistema Imunológico Humano com Análise dos Registros de Atividades*. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina (UFSC), Florianópolis, 2001.

JUCA, K. R. L.; BOUKERCHE, A.; SOBRAL, J. B. M. Intrusion detection based on the immune human system. In: 16TH INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM. *Proceedings*. Florida, 2002. p. 199. Disponível em: <<http://computer.org>>.

JUCA, K. R. L. et al. Human immune anomaly and misuse based detection for computer system operations part ii. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS03). *Proceedings*. Nice, 2003. p. 146. Disponível em: <<http://csdl.computer.org>>.

KARAGIANNIS, D.; KUHN, H. Metamodelling platform. In: THIRD INTERNATIONAL CONFERENCE EC WEB 2002 DEXA 2002. *Proceedings*. Aix en Provence: Springer Verlag, 2002. p. 182.

KEPHART, J. O. A biologically inspired immune system for computers. In: FOURTH INTERNATIONAL WORKSHOP ON SYNTHESIS AND SIMULATION OF LIVING SYSTEMS. *Proceedings*. Banff: MIT Press, 1994. p. 130–139.

KIM, J. The human immune system and network intrusion detection. In: EUFIT 99. *Proceedings*. London: IEEE Computer Society Press, 1999. Disponível em: <<http://www.cs.ucl.ac.uk>>.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained: The model driven architecture: Practice and promise*. Boston: Pearson Education, 2003. 131-141.

LEDECZI, A. Model construction for model-integrated computing. In: 13TH INTERNATIONAL CONFERENCE ON SYSTEMS ENGINEERING. *Proceedings*. Las Vegas, 1999. p. 103–108.

MACHADO, R. B. *Um Sistema de Detecção de Intrusão de Host baseado em Sistema Imunológico Artificial*. Dissertação (Mestrado em Ciências da Computação) — Universidade Federal de Santa Catarina (UFSC), Florianópolis, 2005.

MACHADO, R. B. et al. A hybrid artificial immune and mobile agent intrusion detection based model for computer network operations. In: 19TH IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS05) WORKSHOP 6. *Proceedings*. Denver, 2005. Disponível em: <<http://doi.ieeecomputersociety.org>>.

MELLOR, S. et al. *MDA Distilled: Principles of Model Driven Architecture*. Massachusetts: Addison Wesley, 2004. 131-141 p.

- METACASE. *Domain-specific modeling: 10 times faster than UML*. Jyvaskyla, 2004. Disponível em: <<http://eai.ittoolbox.com>>. Acesso em: 16 dez. 2004.
- MURTA, L. G. P. *Odyssey-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes*. Dissertação (Exame de Qualificação Mestrado em Engenharia de Sistemas e Computação) — Universidade Federal do Rio de Janeiro - COPPE/UFRJ, Rio de Janeiro, 2004.
- NORDSTROM, G. G. *Metamodeling Rapid Design And Evolution Of Domain Specific Modeling Environments*. 170 p. Tese (Doctor Of Philosophy in Electrical Engineering) — Faculty of the Graduate School of Vanderbilt University, Nashville, Maio 1999.
- ODELL, J. J. *Advanced Object-Oriented Analysis & Design using UML*. New York: SIGS Books & Multimedia, 1998. 15-21 p.
- OLIVEIRA, H.; MURTA, L. G. P.; WERNER, C. M. L. Odyssey-vcs: Um sistema de controle de versões para modelos baseados no mof. In: 18º SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. *Anais eletrônicos...* Brasília: Sociedade Brasileira de Computação - SBC, 2004. p. 6.
- OMG. *Common Warehouse Metamodel Specification, Version 1.0*. Needham, 2001. Disponível em: <<http://www.omg.org>>.
- OMG. *Meta-Object Facility Specification, Version 1.4*. Needham, Abril 2002. Disponível em: <<http://www.omg.org>>.
- OMG. *Model-Driven Architecture*. Needham, Março 2002. Disponível em: <<http://www.omg.org/mda/index.html>>.
- OMG. *UML Unified Modeling Language Specification, Version 1.5*. Needham, march 2003. Disponível em: <<http://www.omg.org>>.
- OMG. *XML Metadata Interchange Specification, Version 2.0*. Needham, may 2003. Disponível em: <<http://www.omg.org>>.
- PFLEEGER, S. L. *Engenharia de Software Teoria e Prática*. 2ª. ed. São Paulo: Prentice Hall, 2004.
- POWELL, A. Model with the eclipse modeling framework, part 1. *developerworks*, Vancouver, p. 12, Abril 2004. Disponível em: <<http://www-128.ibm.com/developerworks/opensource/library>>. Acesso em: 20 nov. 2004.
- SANTOS, H. L. dos. *Uma Solução de Metadados Baseada nos Padrões MOF e XML*. 141 p. Dissertação (Mestrado em Ciências da Computação) — Universidade Federal de Pernambuco (UFPE), Recife, 2003.
- SEBESTA, R. W. *Conceitos de Linguagens de Programação*. 5ª. ed. Porto Alegre: Bookman, 2003. 110-116 p.

SOMAYAJI, A.; HOFMEYR, S.; FORREST, S. Principles of a computer immune system. In: NEW SECURITY PARADIGMS WORKSHOP. *Proceedings*. New Castle Upon Tyne: Association for Computing Machinery - ACM, 1997. p. 75–82.

SOMMERVILLE, I. *Engenharia de Software*. 6ª. ed. São Paulo: Addison Wesley, 2003.

SPRINKLE, J. et al. The new metamodeling generation. In: 8TH ANNUAL IEEE INTERNATIONAL CONFERENCE AND WORKSHOP ON THE ENGINEERING OF COMPUTER-BASED SYSTEMS. *Proceedings*. Washington: IEEE Computer Society Press, 2001. p. 275.

STITES, D. P.; TERR, A. I. *Imunologia Básica*. Rio de Janeiro: Prentice Hall do Brasil, 1992. 7-101 p.

TOLVANEN, J.-P. *Incremental Method Engineering with Modeling Tools Theoretical Principles and Empirical Evidence*. 301 p. Dissertação (Dissertation In Computer Science, Economics And Statistics) — University of Jyväskylä, Jyvaskyla, 1998.

ULRICH, F. Multi-perspective enterprise modeling (memo) - conceptual framework and modeling languages. In: 35TH HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES. *Proceedings...* Big Island, 2002. p. 21.

WARMER, J.; KLEPPE, A. *MDA and the use of OCL*. Boston: Addison Wesley, 2003. 1-77 p.

Índice Remissivo

- EMF, 12
- MDA, 1
- MEMO, 11
- Meta, 4
- MetaGME, 11
- Metalinguagem, 4, 6
- Metalinguagens
 - estrutura, 9
 - semântica, 10
 - sintaxe abstrata, 9
 - sintaxe concreta, 10
- Metametamodelo, 6, 7, 9
- Metamodelagem, 1, 4, 5
 - arquiteturas de , 6
- Metamodelo, 5
 - construção, 11
- Metamodelo SIA
 - classes, 59
- MOF
 - arquitetura, 16
 - associação, 32
 - associações, 18, 19
 - atributos, 31
 - classes, 18, 30
 - definição, 13
 - interfaces, 34, 36
 - mapeamento, 14, 30, 38, 39
 - metadados, 14
 - pacotes, 18, 20, 33
 - tipo de dados, 18, 20
- Sistema imunológico
 - nonsel*f, 43, 44
 - self*, 43–45
 - adaptativo, 43
 - anatomia, 45
 - antígeno, 49
 - anticorpos, 47
 - células-B, 47
 - células-T, 48
 - epítomos, 44
 - imunidade, 43
 - imunidade adquirida, 49
 - imunidade inata, 43, 49
 - imunoglobulina, 47
 - linfócitos, 46
 - linfocinas, 48
 - MHC, 45
 - plasmócito, 47
 - propriedades, 50
 - resposta imune, 44
- Sistema Imunológico Artificial
 - nonsel*f, 55
 - self*, 55
 - características, 53