COPPE/UFRJ

EMPREGO DE PRINCÍPIOS DE SISTEMAS EMERGENTES NA CONSTRUÇÃO DE UMA PLATAFORMA PARA APLICAÇÕES PONTO-A-PONTO

Mutaleci de Góes Miranda

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro

Abril de 2010

EMPREGO DE PRINCÍPIOS DE SISTEMAS EMERGENTES NA CONSTRUÇÃO DE UMA PLATAFORMA PARA APLICAÇÕES PONTO-A-PONTO

Mutaleci de Góes Miranda

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Bonorino Xexéo, D. Sc.

_____

Prof. Marco Antonio Casanova, Ph.D.

_____

Profª. Marta Lima de Queirós Mattoso, D.Sc

_____

Prof. Geraldo Zimbrão da Silva, D.Sc.

_____

Profª Maria Cláudia Reis Cavalcanti, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 2010

Miranda, Mutaleci de Góes

Emprego de Princípios de Sistemas Emergentes na Construção de uma Plataforma para Aplicações Ponto-a-ponto/ Mutaleci de Góes Miranda. – Rio de Janeiro: UFRJ/COPPE, 2010.

VIII, 94 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 90–94.

1. Sistemas Ponto-a-ponto. 2. Sistemas Emergentes. 3. Descoberta de Recursos. 4. Colaboração. I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Ao meu filho, Augusto Samuel. Que este trabalho possa de alguma forma lhe servir de inspiração no futuro.

AGRADECIMENTOS

Este trabalho teve um forte cunho colaborativo. Agradeço a todos os colegas que se viram envolvidos nesta pesquisa e contribuíram para seu aperfeiçoamento. Em particular, agradeço ao Marcelo Mayworm, ao Rafael Leonardo, ao José Rodrigues Neto, à Adriana Vivacqua e ao Rodrigo Padula, com os quais tive oportunidade de trabalhar de forma mais próxima. Agradeço também a todos os outros colegas da COPPE, que proporcionaram um ambiente de trabalho agradável e motivante.

Agradeço aos integrantes do Instituto Militar de Engenharia e de outros Órgãos do Exército Brasileiro, que tornaram esta empreitada possível.

Agradeço aos membros da Banca Examinadora, pelo trabalho empenhado e pelas contribuições oferecidas.

Agradeço aos professores Geraldo Xexéo e Jano Souza, pela paciência, pela confiança e pelo direcionamento seguro.

Agradeço aos meus pais, Waldemar (*In Memoriam*) e Alice, por terem acreditado e investido em mim desde o princípio. Agradeço também a todos os outros amigos e familiares, que me auxiliaram das mais diversas formas.

Agradeço à minha esposa, Sandra, pelo apoio e incentivo incondicionais.

Agradeço a Deus, por nos ter dado a capacidade de estudar Sua Criação e por ter me inspirado e protegido durante mais esta jornada.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

EMPREGO DE PRINCÍPIOS DE SISTEMAS EMERGENTES NA CONSTRUÇÃO DE UMA PLATAFORMA PARA APLICAÇÕES PONTO-A-PONTO

Mutaleci de Góes Miranda

Abril/2010

Orientadores: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

O objetivo principal deste trabalho é investigar a viabilidade da aplicação de princípios observados em sistemas naturais emergentes no desenvolvimento de sistemas de computação ponto-a-ponto e, mais particularmente, na abordagem do problema da descoberta de recursos. Os principais resultados obtidos são:

- a elaboração de um modelo de computação ponto-a-ponto que incorpora um conjunto de princípios de emergência;
- a utilização desse modelo na implementação de uma plataforma para execução de aplicações ponto-a-ponto, baseada em agentes móveis que se comunicam por intermédio de espaços compartilhados;
- o desenvolvimento de um mecanismo de descoberta de recursos em redes ponto-a-ponto, inspirado na forma como alguns organismos utilizam substâncias químicas para se orientar em sistemas naturais; e
- implementações e propostas de aplicações colaborativas e de recuperação de informações para a plataforma desenvolvida.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

EMPLOYMENT OF EMERGENT SYSTEMS PRINCIPLES IN BUILDING A PLATFORM FOR PEER-TO-PEER APPLICATIONS

Mutaleci de Góes Miranda

April/2010

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

The main goal of this work is to investigate the viability of applying principles observed in natural systems that present emergent behavior to building peer-to-peer computing systems and, more particularly, to approaching the resource discovery problem. The main results obtained are:

- the elaboration of a model for peer-to-peer computing which embeds a set of emergency principles;
- the use of that model in implementing a platform for peer-to-peer applications, based on mobile software agents which communicate among themselves through shared spaces;
- the development of a peer-to-peer resource discovery mechanism, inspired by the way some organisms use chemicals to orient themselves in natural systems; and
- implementation and proposals of collaborative and information retrieval applications for the developed platform.

SUMÁRIO

1. INTRODUÇÃO

O advento da tecnologia de comutação de pacotes, já presente na Arpanet - a rede do Departamento de Defesa dos Estados Unidos da América que foi o embrião da Internet, viabilizou o desenvolvimento de sistemas globais de computação distribuída capazes de permanecerem em funcionamento a despeito da ocorrência de falhas em suas subredes componentes (LEINER, 2008). Embora a descentralização, a escalabilidade e a resiliência tenham se tornado características intrínsecas da Internet em função de sua arquitetura e de seus protocolos, grande parte dos sistemas de informações distribuídos é construída atualmente de acordo com o paradigma cliente-servidor, que presume a existência de um computador principal em cada sistema. Esse nó principal centraliza a responsabilidade pelas funções essenciais do sistema, tornando a obtenção da escalabilidade e a resiliência muito dispendiosas.

No início da década corrente, ferramentas de compartilhamento de arquivos como Freenet (CLARKE et al., 2000), Napster (NAPSTER, 2001), e outras reacenderam o interesse geral pelos assim chamados sistemas ponto-a-ponto (*peer-to-peer*), que se baseiam também em requisitos de descentralização, escalabilidade e resiliência. Embora tenham servido como prova de conceito, as ferramentas de compartilhamento de arquivos populares são funcionalmente restritas e implementam apenas parcialmente os requisitos de sistema ponto-a-ponto. Diversos grupos de pesquisa têm procurado ampliar a aplicabilidade dos sistemas ponto-a-ponto gerando contribuições em aspectos tais como interoperabilidade (JXTA,2001), semântica e metadados (NEJDL, 2002), indexação (RATNASAMY et al, 2001,STOICA et al, 2001, ROWSTRON, e DRUSCHEL, 2001), e recuperação de informações(ABERER et al 2004, TANG & DWARKADAS, 2004).

A alternância entre paradigmas de centralização-hierarquia e descentralização-rede pode ser verificada também nos históricos de outros segmentos da tecnologia da informação, como por exemplo na modelagem de bancos de dados. De uma perspectiva mais ampla podemos observar que sistemas naturais e sociais são, em geral, redes descentralizadas em essência, ao passo que modelos tecnológicos freqüentemente se apóiam em solução hierárquicas ou centralizadas por questões de simplicidade, custo e

facilidade de controle. Empresas, por exemplo, embora sejam mais precisamente modeladas por meio da teoria das redes sociais, são normalmente organizadas em torno de uma hierarquia administrativa, com um grande número de níveis em alguns casos.

1.1. Motivação

A adesão dogmática ao paradigma de centralização-hierarquia pode comprometer severamente a capacidade de inovação em tecnologia da informação, particularmente em aplicações onde o foco não é o processamento automático de informações, mas sim a intermediação entre seres humanos. Em contrapartida, a conjunção de conceitos advindos da teoria das redes sociais e da tecnologia de sistemas computacionais ponto-a-ponto pode apoiar o desenvolvimento de uma classe de aplicações colaborativas e de gerência do conhecimento capaz de promover a eficácia operacional e o aproveitamento do capital intelectual de organizações acadêmicas, governamentais e de negócios.

Além dos aspectos supracitados, a tecnologia ponto-a-ponto oferece perspectivas particularmente interessantes para sistemas relacionados à defesa nacional. Uma dessas perspectivas é a possibilidade da concepção de sistemas mais resistentes às condições de combate, como por exemplo, sistemas de comando e controle com postos de comando distribuídos. Um posto de comando é uma instalação na qual um comandante militar e seu estado-maior reúnem-se para realizar, o comando e o controle das operações realizadas por suas tropas subordinadas. Cada nível de comando possui seus próprios postos de comando que ligam-se aos postos de nível superior e subordinado, formando uma hierarquia por onde fluem ordens, no sentido descendente, e informações sobre o estado atual do ambiente, das tropas subordinadas, das operações e dos adversários, no sentido ascendente. Embora os postos de comando sejam estabelecidos tipicamente em posições afastadas das áreas de combate principais, a tecnologia atual de sensoreamento e o emprego de armamento de longo alcance os torna vulneráveis. Operações de decapitação, ou seja, destruição ou incapacitação de postos de comando, podem provocar redução na disponibilidade de informações para o escalão superior e desorganização nos escalões subordinados, reduzindo drasticamente o poder

de combate da tropa atingida. A aplicação da tecnologia de aplicações colaborativas com arquitetura ponto-a-ponto pode permitir que:

- um grupo de pessoas disperso geograficamente realize as atividades de um posto de comando com baixo risco de ser alvo de uma operação de decapitação;
- cada componente do grupo permaneça na área mais propícia para a obtenção de informações ligadas às suas atribuições, e que componentes incapacitados pelos adversários sejam substituídos rapidamente;
- informações sejam coletadas, processadas e distribuídas colaborativamente, entre elementos do mesmo nível hierárquico ou de níveis diferentes, promovendo o aumento da consciência situacional de todos os participantes;

Outro cenário relacionado à defesa nacional no qual a tecnologia ponto-a-ponto parece ser uma abordagem promissora é a de gerência do conhecimento baseada em simulações computacionais. A atividade bélica contemporânea é marcada pela assimetria tática e tecnológica entre inimigos, pela possibilidade de emprego de tropas em regiões remotas e pela rápida evolução dos sistemas de armas, de apoio e de comando e controle. Conseqüentemente, as forças armadas devem ser capazes de gerar e disseminar conhecimento adequado continuamente, ou seja, devem implementar um processo eficiente de gerência de conhecimento para manter sua operacionalidade, como por exemplo, o uso intensivo de simulações de combate. Entretanto, a implementação de simulações de grande escala em arquitetura cliente-servidor gera a necessidade de centros de simulações complexos, dotados de servidores com grande poder de processamento. Em contrapartida, o emprego de uma arquitetura ponto-a-ponto pode permitir que a capacidade de processamento de computadores pessoais disponíveis nas organizações militares para tarefas rotineiras seja combinada para executar simulações de combate em escalas variadas, sem a necessidade de manutenção de um centro de simulações complexo e da realização de grandes deslocamentos de pessoal e equipamentos.

Talvez um passo fundamental para fomentar o aproveitamento do potencial de arquiteturas de distribuição mais gerais seja a implementação de um infraestrutura de *software* baseada em um paradigma que permita que os desenvolvedores escolham livremente o grau de centralização adequado para suas aplicações, sem prejuízo dos

requisitos de escalabilidade e resiliência. Uma alternativa promissora de paradigma é o modelo de sistemas complexos adaptativos, ou sistemas emergentes, inspirado em sistemas naturais tais como colônias de insetos por exemplo. Um sistema complexo adaptativo é composto por um grande número de agentes que interagem entre si e com um ambiente de forma que o sistema possa desempenhar suas funções mesmo na presença de algum grau de falha individual de agentes ou mudanças imprevistas no ambiente. Nesse tipo de sistema a comunicação entre agentes é mediada pelo ambiente e cada agente pode influenciar o comportamento de qualquer outro de acordo com necessidades circunstanciais do sistema. Além disso, o comportamento desses sistemas é emergente, ou seja, o relacionamento causal entre o comportamento individual dos agentes e o comportamento global do sistema não é evidente.

A implementação de uma infraestrutura baseada nesse paradigma envolve uma série de problemas de pesquisa tais como indexação, busca e recuperação de informações distribuída e controle de consistência e disponibilidade dos dados, que precisam ser tratados através de técnicas diferentes das empregadas em sistemas de bancos de dados convencionais.

1.2. Objetivo

O objetivo da presente pesquisa é investigar a viabilidade da aplicação de princípios observados em sistemas naturais emergentes no desenvolvimento de sistemas ponto-a-ponto, com foco principal no problema da descoberta de recursos. As principais etapas realizadas foram a concepção e implementação de uma plataforma para desenvolvimento de aplicações ponto-a-ponto, o desenvolvimento de algoritmos de descoberta de recursos, a realização de simulações para avaliação da escalabilidade e da resiliência desses mecanismos, e o desenvolvimento de aplicações para validação.

O restante desse trabalho está organizado da seguinte forma: no capítulo 2 são apresentados conceitos, técnicas e ferramentas atuais de sistemas ponto-a-ponto; no capítulo 3 são apresentadas uma série de princípios de engenharia encontrados em sistemas naturais com comportamento emergente, e as diretrizes extraídas dos mesmos para a concepção da arquitetura lógica da plataforma; no capítulo 4 são apresentados os

componentes da arquitetura lógica, incluindo o método de descoberta de recursos concebido no contexto da pesquisa; no capítulo 5 a plataforma é caracterizada; no capítulo 6 são relacionadas algumas aplicações implementadas sobre a plataforma e apresentadas outras propostas de implementação; finalmente, no capítulo 7 são apresentadas nossas considerações finais.

2. SISTEMAS PONTO-A-PONTO

Um sistema ponto-a-ponto (*peer-to-peer*) é uma aplicação de rede computacional na qual cada nó é administrado de forma autônoma e possui a capacidade de contribuir com recursos computacionais para a execução distribuída de tarefas solicitadas por outros nós do sistema. As propriedades de um sistema ponto-a-ponto ideal são a descentralização, a resiliência e a escalabilidade.

A descentralização significa a inexistência de nós específicos responsáveis por funções essenciais do sistema. Na prática, entretanto, existem sistemas ponto-a-ponto que admitem algum grau de centralização em prol de melhoria de desempenho.

A resiliência é conseqüência da descentralização, pois uma vez que a responsabilidade por cada função seja distribuída por vários nós, um certo número de falhas individuais não deverá comprometer o sistema como um todo.

Finalmente, a escalabilidade advém do fato que cada nó que entra na rede oferece recursos computacionais adicionais para aumentar a capacidade do sistema.

As três grandes áreas de aplicação para sistemas ponto-a-ponto são o compartilhamento de recursos computacionais, a colaboração e a computação em colméia(*hive computing*).

O compartilhamento de recursos computacionais ocorre quando cada participante do sistema utiliza espaço de armazenamento de dados, dados armazenados, capacidade de processamento ou capacidade de transmissão de dados de outros nós para atingir seus objetivos individuais e ao mesmo tempo oferece seus recursos para o restante da rede.

A colaboração ocorre quando grupos de usuários são formados para a execução de tarefas de interesse coletivo e utilizam a rede para, além de compartilhar recursos utilizados nessa atividade, realizar comunicação e coordenação, possivelmente de forma síncrona.

Na computação em colméia, um único problema computacional complexo, cujo tratamento centralizado é inviável, é particionado e distribuído pelos nós do sistema.

2.1. Grau de heterogeneidade de sistemas ponto-a-ponto

Em relação a seu grau de heterogeneidade, um sistema ponto-a-ponto pode ser puro, híbrido ou baseado em supernós(*super-peers*). Sistemas puros são caracterizados pela equivalência funcional dos nós participantes. Sistemas híbridos, por sua vez, utilizam nós servidores para centralizar a execução de algumas tarefas do sistema. Sistemas baseados em supernós são uma solução intermediária que tem por objetivo superar problemas de desempenho, característicos de sistemas puros, e simultaneamente evitar a criação de gargalos de processamento e pontos críticos de falha que ocorre em sistemas híbridos. Os supernós são nós do sistema que possuem a capacidade de participar de todas tarefas do mesmo, interligados de modo a formar um subsistema ponto-a-ponto puro. Nesse tipo de sistema, cada um dos nós ordinários precisa ser conectado obrigatoriamente a pelo menos um supernó, podendo ser ou não conectado também a outros nós ordinários. A figura 2.1 apresenta de forma esquemática exemplos dos três tipos de sistema descritos.

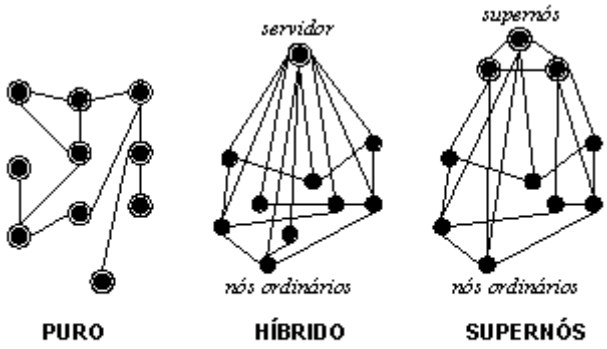

Figura 2.1 – Graus de heterogeneidade de sistemas ponto-a-ponto

2.2. Topologia de sistemas ponto-a-ponto

Em geral, os sistemas ponto-a-ponto são classificados como estruturados ou não-estruturados. Um sistema ponto-a-ponto é dito estruturado quando existe uma correlação entre a alocação dos dados e a topologia da rede, e é dito não-estruturado

quando essa propriedade não ocorre (MILOJICIC et al. , 2002). As propostas de sistemas estruturados encontradas na literatura normalmente estão associadas a algum algoritmo de roteamento específico que explora suas propriedades topológicas para obter garantias de desempenho na descoberta de recursos. Sistemas não-estruturados, em contrapartida, baseiam-se em algoritmos de busca heurísticos tais como inundação (*flooding*) e caminhamento aleatório (*random walking*).

Os defensores da abordagem de não-estruturação argumentam que o ganho de desempenho obtido pelo emprego de uma topologia estruturada não compensa o consumo de recursos computacionais necessários para manter a estrutura da rede, uma vez que a autonomia dos nós pode implicar em participação transiente dos mesmos. Além disso, as propostas básicas de sistemas estruturados não tratam adequadamente as diferenças de desempenho individual dos nós e não oferecem suporte a consultas complexas.

As tentativas de elaboração de sistemas estruturados funcionalmente mais ricos suscitaram o aparecimento de abordagens mais complexas, como a descrita em (CASTRO et al., 2005), que não se enquadram graciosamente no esquema de classificação apresentado acima. A fim de agrupar as diversas possibilidades de forma mais acurada, a topologia dos sistemas e os métodos de descoberta de recursos serão examinados neste trabalho de forma independente.

2.2.1. Redes sobrepostas

Os sistemas ponto-a-ponto podem ser estruturados como redes sobrepostas (*overlay networks*), ou seja, redes lógicas que utilizam os serviços de uma rede subjacente independente para efetuar a comunicação entre seus nós. A existência de uma ligação direta no nível da rede sobreposta entre dois nós quaisquer significa, na verdade, que as mensagens trocadas entre esses nós percorrem caminhos da rede subjacente que não são controlados pela rede sobreposta, podendo inclusive conter nós que não fazem parte da mesma.

A figura 2.2 ilustra uma rede sobreposta à internet, cujos nós são as estações de trabalho numeradas de 1 a 6. As ligações da rede sobreposta existentes entre os nós 2 e

3 e entre os nós 4 e 5 são diretas também na Internet, uma vez que cada par de nós está contido inteiramente em uma única rede local. Todas as demais ligações da rede sobreposta são entre pares que abrangem mais de uma rede local e, portanto, implicam a utilização de gateways e roteadores da Internet para transportarem mensagens. Deve-se notar que uma troca de mensagens entre os nós 6 e 7, que pertencem à mesma rede local, também utilizaria o roteamento da Internet, uma vez que esses nós não estão ligados entre si no nível da rede sobreposta e, conseqüentemente, necessitam da intermediação do nó 5, que está localizado em outra rede local.

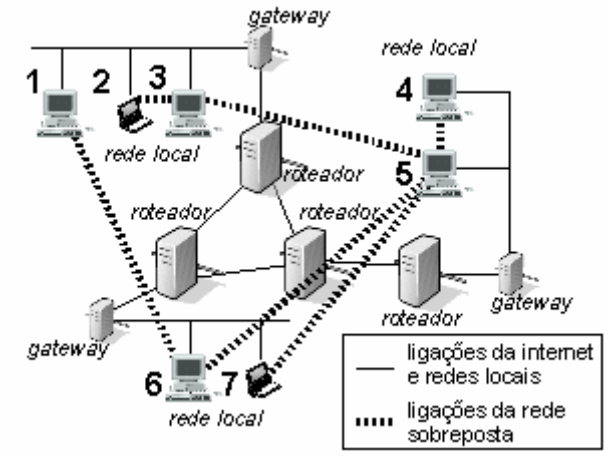

Figura 2.2 – Exemplo de rede sobreposta

Embora as aplicações no nível da rede sobreposta não possam controlar a rede subjacente, a rede sobreposta pode ser construída de forma explorar a localidade da rede, ou seja, privilegiar o estabelecimento de ligações diretas entre nós que estejam conectados por meio de poucos dispositivos na rede subjacente, podendo portanto trocar mensagens de forma eficiente.

Outro aspecto relevante relacionado às redes subjacentes é a necessidade da inclusão nos sistemas ponto-a-ponto de mecanismos de travessia de dispositivos de roteamento ou segurança que impõe restrições ao tráfego de mensagens tais como os NAT's (*Network Address Translators*) e *firewalls* da Internet.

2.2.2. Tipos de topologia

Um sistema ponto-a-ponto será classificado, neste trabalho, como de topologia estruturada quando seu nós e ligações puderem ser mapeados nos elementos de uma estrutura matemática dotada de um procedimento formal de construção, e de topologia não-estruturada em caso contrário.

Uma variedade de abordagens de construção de topologias estruturadas para sistemas ponto-a-ponto utiliza aproximações de grafos regulares como ponto de partida e adiciona conjuntos de ligações apropriados para gerar propriedades desejadas de roteamento e resiliência. Genericamente, tais abordagens associam a cada nó do sistema um identificador formado por seqüências de d dígitos de b bits cada. O número de dígitos corresponde ao número de dimensões ocupadas pela representação espacial do grafo e o número de bits corresponde ao logaritmo na base 2 da quantidade máxima de nós em cada eixo dimensional. Alguns exemplos de grafos regulares, ilustrados na figura 2.3, e suas regras de construção são relacionados a seguir:

- **malha (*mesh*):** seja $d > 1$ o número de dígitos dos identificadores dos nós e $b > 1$ o número de bits de cada dígito. Cada nó A é ligado aos nós cujos identificadores possuem d-1 dígitos coincidentes a dígitos correspondentes no identificador de A e cujo dígito remanescente é igual ao sucessor ou predecessor aritmético do dígito correspondente no identificador de *A;*
- **hipercubo:** a construção é realizada de forma similar à da malha, empregando-se identificadores de nó de apenas um bit por dígito;
- **toróide:** a construção é realizada de forma similar à da malha, empregando-se aritmética módulo 2^b na determinação de sucessores e predecessores;

- **anel:** a construção é realizada de forma similar à do toróide, empregando-se identificadores de nó de apenas um dígito;

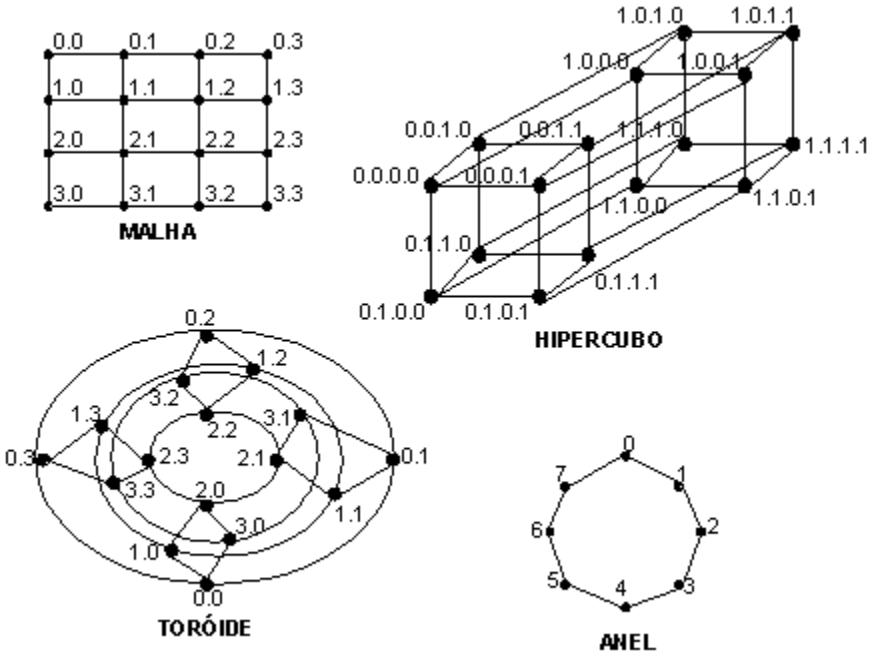

Figura 2.3 – Exemplos de grafos regulares

Uma alternativa à utilização de grafos é a associação de nós da rede a regiões de um espaço cartesiano. Nessa abordagem dois nós da rede serão ligados sempre que estiverem associados a regiões vizinhas no espaço cartesiano.

Outra possível abordagem de estruturação é a construção de topologias semânticas. Nesse caso, os dados de aplicações contidos nos nós são representados por entidades matemáticas e então a rede é organizada com base em medidas de similaridade entre essas entidades.

Uma vez estabelecidas as características estruturais da topologia de um sistema ponto-a-ponto, outro aspecto fundamental a ser observado é a definição de um comportamento dinâmico adequado para a mesma, ou seja, de protocolos de entrada e saída de nós na rede que assegurem a escalabilidade da topologia preservando as suas propriedades desejadas, bem como a continuidade do funcionamento do sistema, seja através da transferência de recursos imediatamente antes da saída programada de um nó, ou através do emprego de replicação, capaz manter a disponibilidade de recursos

mesmo em caso de saídas inesperadas. Tais protocolos são naturalmente dependentes da estrutura escolhida e de sua eficiência depende a viabilidade da utilização da rede em cenários de entradas e saídas freqüentes de nós.

2.2.3 Topologia de mundo pequeno

A despeito do tipo de topologia de um sistema ponto-a-ponto, em geral é desejável que a mesma apresente propriedades que facilitem a disseminação de informações pela rede.

O termo rede de mundo pequeno refere-se a experimentos sociológicos baseados em comunicação por correio tradicional, nos Estados Unidos da América (MILGRAM, 1967), e por correio eletrônico, abrangendo o mundo inteiro (DODDS, 2003), nos quais foi verificado que indivíduos residentes em regiões geograficamente muito afastadas são socialmente conectados por cadeias formadas por em média seis pessoas.

Nesses experimentos, um conjunto de destinatários finais foi estabelecido e participantes residentes em cidades muito afastadas desses destinatários foram convidados a iniciar o roteamento das mensagens. Cada participante, caso não conhecesse o destinatário final de uma mensagem, deveria enviá-la para um conhecido que supusesse socialmente mais próximo do destinatário final. Os resultados do experimento revelaram não só a existência de cadeias curtas ligando indivíduos quaisquer em uma vasta rede social, mas também a habilidade dos indivíduos de rotear mensagens eficientemente por essas cadeias com conhecimento local e limitado da rede.

De forma um pouco mais rigorosa, uma rede é dita de mundo pequeno quando apresenta simultaneamente baixo caminho mínimo médio (crescimento logarítmico sobre o número de nós), e alto coeficiente de agrupamento (próximo a 1).

O caminho mínimo médio de uma rede é a média do número mínimo de ligações que precisam ser percorridas para transportar uma mensagem entre cada par de nós na rede. O coeficiente de ligação de uma rede é a média do coeficiente de agrupamento de todos os seus nós. Por sua vez, o coeficiente de agrupamento de um nó é definido como a razão entre o número de ligações existentes entre seus vizinhos e o

número de ligações que existiriam se a vizinhança fosse completamente conectada.

Uma forma simples de construir uma rede de mundo pequeno é o método de WATTS & STROGATZ (1998), que consiste em organizar a rede como um reticulado regular em forma de anel no qual cada nó é ligado aos outros k nós mais próximos no anel e então substituir uma pequena porção de ligações escolhidas aleatoriamente por ligações entre nós distantes no anel. A regularidade da estrutura básica da rede gera um alto coeficiente de agrupamento, ao passo que as ligações entre nós distantes no anel reduz o caminho mínimo médio.

Embora o trabalho de Watts e Strogatz caracterize matematicamente as redes de mundo pequeno e demonstre a ocorrência das mesmas em uma variedade de sistemas naturais e artificiais, não fornece subsídios para a questão da eficiência de roteamento de mensagens. Uma construção mais adequada para esse fim consiste em organizar a rede como uma malha bidimensional de nxn nós na qual, além das ligações a nós adjacentes, cada nó possui um atalho unidirecional com destino a um nó distante na malha, escolhido segundo alguma distribuição de probabilidades. KLEINBERG (1999) considerou o problema de, a partir de um nó arbitrário, rotear uma mensagem para outro nó dada sua posição na malha, sob a restrição de que cada nó conheça apenas as posições de seus vizinhos adjacentes na malha e do vizinho ligado pelo atalho. Nesse trabalho ele demonstrou que, se a distribuição de probabilidades utilizada na escolha de destinos de atalhos for inversamente proporcional ao quadrado da distância entre os nós sobre ligações na malha básica, é possível rotear mensagens em O ($\log^2 n$) passos apenas encaminhando-as a cada passo para o vizinho mais próximo ao destino final, e que esse algoritmo de roteamento é ótimo entre aqueles que utilizam apenas conhecimento local da rede.

2.3 Descoberta de recursos

Como os sistemas ponto-a-ponto são caracterizados pela utilização de recursos computacionais distribuídos para a execução de suas funções, um aspecto fundamental para os mesmos é o emprego de mecanismos destinados a informar eficientemente aos nós quais são os outros nós da rede que podem contribuir com os recursos necessários

para uma dada tarefa. Esses mecanismos podem pertencer às seguintes categorias:

- **diretório centralizado:** metadados sobre todos os recursos presentes no sistema, incluindo suas localizações, são reunidos em um diretório centralizado. Para determinar quais nós podem oferecer um recurso compatível com uma dada consulta, as aplicações devem realizar uma busca sobre esse diretório. Esse método é empregado tipicamente em sistemas híbridos, nos quais o nó servidor encarrega-se de armazenar o diretório. Dessa forma, só é necessária a comunicação do nó iniciador da consulta primeiramente com o nó servidor e em seguida com o nó provedor do recurso.
- **caminhamento aleatório (*random walking*):** cada consulta é encaminhada para um vizinho do nó iniciador escolhido aleatoriamente. Caso o nó receptor possua um recurso compatível ele envia esse recurso para o nó iniciador. Caso contrário o processo de encaminhamento para um vizinho aleatório é repetido por cada nó receptor até que um recurso compatível seja encontrado ou que um número preestabelecido de nós tenha sido percorrido. Esse método está tipicamente associado a redes não-estruturadas.
- **inundação de consultas (*query flooding*):** cada consulta, é rotulada com um número correspondente à distância máxima entre o nó iniciador e qualquer nó onde os recursos serão pesquisados. Se o valor do rótulo for positivo, cópias da consulta com o valor do rótulo decrementado de uma unidade são encaminhadas para todos os vizinhos do nó iniciador. Os nós receptores que possuam um recurso compatível enviam o recurso para o nó iniciador e os demais nós repetem o processo de encaminhamento seus vizinhos. A fim de reduzir o número de mensagens desnecessárias, os nós descartam consultas recebidas em duplicata e não encaminham consultas para o vizinho por meio do qual elas chegaram. Esse método também está tipicamente associado a redes não-estruturadas.
- **propagação seletiva:** generalização dos métodos de caminhamento aleatório e de inundação. Cópias de cada consulta são encaminhadas para um subconjunto dos vizinhos do nó iniciador. Os nós receptores que possuam um

recurso compatível enviam o recurso para o nó iniciador e os demais nós repetem o processo de encaminhamento. Em cada passo do procedimento, um nó que examina uma consulta realiza a construção do subconjunto de nós receptores, de forma probabilística ou determinística, com base em informações recebidas previamente dos vizinhos ou anexadas à própria consulta. Uma forma eficiente de armazenar e transmitir informações sobre vizinhanças é a utilização de filtros Bloom (BLOOM, 1970) para representar o conteúdo de outros nós. Um filtro bloom é um vetor de bits de tamanho fixo usado para armazenar informações de pertinência a conjuntos de tamanho indeterminado, mediante a aceitação de alguma probabilidade de ocorrência de falsos positivos. O conjunto vazio é representado por um vetor com valor *0* em todos os bits. Para incluir um elemento em um filtro bloom de m bits, $k < m$ funções de espalhamento diferentes, todas mapeando o espaço dos elementos ao espaço de números inteiros correspondentes a posições no vetor, são aplicadas ao elemento a ser inserido. Então, os k bits localizados nas posições correspondentes às saídas das funções de espalhamento recebem o valor *1*. Para fazer um teste de pertinência, as mesmas funções utilizadas para inclusão são aplicadas ao elemento a ser testado, e o elemento é considerado pertencente ao conjunto caso os bits localizados nas k posições obtidas possuam o valor *1*. Esse esquema produz uma probabilidade de falsos positivos p que cresce juntamente com o número n de elementos inseridos segundo a expressão $p = [1- (1 – m^{-1})^{kn}]^{k.}$ Em conseqüência, para valores pré-estabelecidos de m e n, o valor de k que minimiza a probabilidade de falsos positivos é de aproximadamente $0,7m/n$.

- **roteamento de recursos:** todo recurso colocado à disposição do sistema, ou um índice com metadados sobre o recurso, é roteado, em função de suas características, para algum nó do sistema. Consultas são igualmente roteadas pela rede de modo a atingir nós para os quais recursos ou índices compatíveis tenham sido roteados previamente. Esse método é tipicamente empregado em sistemas de topologia estruturada, por meio da técnica conhecida como tabela de espalhamento distribuída (*distributed hash table* – DHT). Uma DHT associa identificadores únicos aos recursos e atribui a cada nó do sistema a

responsabilidade por uma partição do espaço de identificadores. Para realizar consultas em DHT's é necessário conhecer aprioristicamente o identificador do recurso desejado, o que as caracteriza como mecanismos de busca baseados em palavra-chave. Em geral, as redes associadas a DHT's são projetadas para apresentar simultaneamente baixo número de vizinhos por nó, baixo diâmetro, roteamento baseado em decisões locais, balanceamento de carga e alto número de caminhos alternativos entre os nós (HELLERSTEIN, 2003).

2.4. Exemplos

Nas seções 2.4.1 a 2.4.9, serão apresentados alguns exemplos de protocolos e aplicações que ilustram algumas das técnicas supramencionadas.

2.4.1. Pastry (ROWSTRON e DRUSCHEL, 2001)

Pastry é uma infraestrutura de DHT que mantém a rede estruturada em uma topologia baseada em hipercubo. Os identificadores dos nós são escolhidos aleatoriamente dentro de um espaço circular de identificadores de 128 bits formados por 128/*b* dígitos de *b* bits cada.

Cada recurso colocado à disposição do sistema é associado a uma chave de identificação pertencente a esse mesmo espaço, que é utilizada para rotear o recurso para o nó com identificador numericamente mais próximo da chave.

A estrutura básica de uma rede Pastry de até *N* nós é obtida por meio da manutenção, em cada nó, de uma tabela de roteamento composta por $\log_{(2^b)} N$ linhas com $2^b - 1$ entradas cada uma, na qual a *n*-ésima linha contém endereços, na rede subjacente, de nós cujos identificadores possuem os *n* primeiros dígitos com valores coincidentes aos dos dígitos do identificador do nó que hospeda a tabela. O conjunto formado pelos valores dos *(n+1)*-ésimos dígitos dos identificadores dos nós referenciados na linha da tabela deve corresponder ao conjunto dos valores não coincidentes ao *(n+1)*-ésimo dígito do identificador nó que hospeda a tabela. Dessa forma, se durante um roteamento um nó recebe um recurso cujo identificador possui os

k primeiros dígitos correspondentes aos *k* primeiros dígitos de seu próprio identificador, ele será capaz de encaminhar o recurso a um outro nó cujo identificador possui os *k+1* primeiros dígitos correspondente aos *k+1* primeiros dígitos do identificador do recurso. Em conseqüência, os roteamentos podem ser completados em O ($\log_{(2}{}^{b)}N$) passos.

A especificação Pastry explora a localidade da rede estabelecendo a proximidade dos nós na rede subjacente como critério adicional para a composição das tabelas de roteamento. Além da tabela de roteamento propriamente dita, cada nó mantém referências para os *M* nós mais próximos em termos de localidade de rede, a fim de auxiliar o processo de composição da tabela de roteamento, bem como para os *L* nós com identificadores numericamente mais próximos ao seu identificador, utilizados para completar o processo de roteamento quando os encaminhamentos baseados na tabela de roteamento já levaram o recurso a um nó suficientemente próximo do nó alvo e para dar suporte à replicação.

O custo de entrada de cada nó nas redes Pastry é também de O ($\log_{(2}{}^{b)}N$) mensagens.

2.4.2. CAN (RATNASAMY et al., 2001)

CAN (*content addressable network*) é uma infraestrutura de DHT que emprega a abordagem de mapeamento dos nós da rede em regiões de um espaço cartesiano de *d* dimensões. Ao entrar na rede, um nó seleciona aleatoriamente um ponto do espaço cartesiano e contacta o nó que está mapeado à região correspondente. Então o nó contactado divide a região com o nó que está entrando e ambos, bem como os nós vizinhos, têm as ligações ajustadas para refletir o novo mapeamento.

Cada recurso colocado à disposição do sistema é associado a uma chave de identificação à qual é aplicada uma função de espalhamento uniforme que a mapeia também em um ponto do espaço cartesiano. O recurso é então roteado para o nó mapeado à região correspondente.

Em redes CAN, cada nó é ligado a *2d* vizinhos, ou seja, a um predecessor e a um sucessor para cada dimensão, resultando em uma configuração de toróide. Em cada passo de um roteamento, o recurso é encaminhado ao vizinho mapeado à região mais

próxima do ponto correspondente ao recurso. Dessa forma, os roteamentos são realizados em $O(n^{1/d})$ encaminhamentos , onde n é o número de nós na rede.

A especificação CAN define ainda um mecanismo de múltiplas realidades, que consiste em mapear a rede a múltiplos espaços cartesianos e utilizar funções de espalhamento diferentes para rotear recursos em cada um desses espaços. Esse mecanismo dá suporte à replicação, uma vez que permite que cada recurso seja roteado para vários nós diferentes, produzindo impacto positivo tanto sobre a resiliência quanto ao tempo de resposta para as buscas.

O custo de entrada de nós em redes CAN é de O (d) mensagens.

2.4.3. Chord (STOICA et al., 2001)

Chord é um protocolo de DHT que mantém a rede estruturada em uma topologia baseada em anel. Os identificadores dos nós são obtidos por meio da aplicação de uma função de espalhamento aos endereços da rede IP subjacente.

Cada recurso colocado à disposição do sistema é associado a uma chave de identificação à qual é aplicada a mesma função de espalhamento utilizada na obtenção dos identificadores dos nós. O identificador de chave assim obtido é utilizado para rotear o recurso para o nó cujo identificador é o sucessor mais próximo do identificador da chave. A função de espalhamento consistente promove o balanceamento de carga entre os nós da rede. A fim de obter eficiência de roteamento, o protocolo Chord mantém em cada nó uma tabela de roteamento construída de forma que a i-ésima entrada da tabela corresponde ao primeiro nó cujo identificador supera em pelo menos 2^{i-1} unidades o identificador do nó que contém a tabela, onde $1 \leq i \leq b$, b é o número de bits dos identificadores, e a aritmética módulo 2^b é empregada em todos os cálculos. Dessa forma, o anel base da topologia é complementado por até b ligações unidirecionais partindo de cada nó da rede, que permitem que durante um roteamento a distância ao nó alvo seja reduzida à metade em cada encaminhamento. Como conseqüência, roteamentos em redes de N nós podem ser realizados com alta probabilidade em $O(\log N)$ encaminhamentos.

O protocolo Chord emprega a transmissão de $O(\log^2 N)$ mensagens para manter

a topologia correta após cada entrada ou saída de nó na rede. A replicação de recursos não é parte integrante do protocolo sendo deixada a cargo das aplicações.

2.4.4. Symphony (MANKU et al., 2003)

Assim como Chord, Symphony é um protocolo de DHT que mantém a rede estruturada em uma topologia baseada em anel. Os identificadores dos nós são números inteiros escolhidos no intervalo [0,1).

Cada recurso colocado à disposição do sistema é associado a um número nesse intervalo e roteado para o nó cujo identificador é o sucessor mais próximo do número associado. O anel base da topologia é transformado em uma rede de mundo pequeno similar à do modelo de Kleinberg por meio da adição de k ligações unidirecionais partindo de cada nó da rede com destinos a nós responsáveis por identificadores escolhidos de acordo com a função de distribuição de probabilidades $p_n(x) = 1/(x.\ln n)$, onde n é o número de nós estimado na rede. Como conseqüência, os roteamentos podem ser realizados em $O((\log^2 n)/k)$ encaminhamentos.

O protocolo Symphony também emprega a transmissão de $O(\log^2 n)$ mensagens para manter a topologia correta após cada entrada ou saída de nó na rede.

2.4.5 Napster (NAPSTER, 2001)

Napster é um sistema híbrido de compartilhamento de arquivos que reacendeu o interesse geral pela computação ponto-a-ponto, tornando-se uma aplicação popular da Internet.

A topologia do Napster é não-estruturada, uma vez que cada participante do sistema mantém armazenados em sua própria estação de trabalho os arquivos que decide compartilhar e estabelece ligações diretas com as estações de participantes que tenham solicitado cópias desses arquivos. A função do servidor Napster é prover um diretório centralizado no qual são pesquisados os nós aptos a fornecer os arquivos solicitados por cada participante.

2.4.6. SETI@home (BERKELEY, 1999)

SETI@home é um sistema híbrido de computação em colméia destinado à análise de grandes massas de dados obtidas por rádios telescópios para identificação de padrões de sinais que possam indicar a existência de inteligência extraterrestre.

O sistema apresenta uma topologia em estrela, na qual apenas o servidor central liga-se a todos os outros participantes. A função do servidor é particionar os dados que devem ser analisados, enviá-los para os participantes, receber os resultados das análises parciais e realizar a integração esses resultados.

2.4.7. Gnutella (GNUTELLA, 2001)

Gnutella é um protocolo de busca distribuída para redes de topologia não-estruturada, que admite consultas complexas, definidas pelo usuário.

As primeiras versões do Gnutella aderiam à arquitetura ponto-a-ponto pura e eram baseadas em inundação simples de consultas, mas posteriormente o protocolo passou a adotar uma arquitetura de supernós. Esses supernós aumentam a eficiência das buscas centralizando parte do processamento de consultas de seus nós ordinários, mas ainda dependem do mecanismo de inundação para comunicarem-se entre si, limitando assim a escalabilidade do protocolo.

2.4.8. Freenet (CLARKE et al., 2000)

Freenet é um sistema ponto-a-ponto puro de topologia não-estruturada destinado ao compartilhamento de espaço de armazenamento que utiliza um método de propagação seletiva de consultas.

Cada nó do sistema mantém uma tabela de roteamento que associa nós vizinhos a chaves de identificação de arquivos, sendo o conjunto inicial de vizinhos de um nó fornecido por meio de algum mecanismo externo ao sistema.

Quando um nó recebe uma consulta do usuário ou encaminhada por outro nó, ele envia o arquivo solicitado, caso o possua, de volta pelo caminho reverso ao da

consulta. À medida em que um arquivo percorre o caminho reverso em direção ao originador da consulta, os nós visitados o armazenam e atualizam suas tabelas de roteamento, eliminando arquivos anteriores e entradas de tabela pouco utilizadas se for necessário. Caso um nó não possa responder uma consulta, ele a encaminha para o vizinho associado à chave mais similar da chave da consulta. E caso um nó não possa encaminhar uma consulta para o vizinho com a chave mais similar devido a indisponibilidade ou para evitar ciclos, ele a envia para o vizinho associado à segunda chave mais similar e assim por diante. Se nenhum vizinho puder receber a consulta ou se o tamanho de caminho máximo tiver sido atingido, o nó realiza a retropropagação (*backtracking*) de uma mensagem de falha, a fim de que nós anteriores possam também tentar outras opções de encaminhamento. Solicitações de armazenamento recebem tratamento análogo ao das consultas.

2.4.9. Jxta (SUN, 1999)

Jxta (*Juxtapose*) é um conjunto de protocolos para desenvolvimento de aplicações ponto-a-ponto que abrange as funcionalidades de descoberta de nós e serviços oferecidos pelos nós, publicação de serviços disponíveis, troca de dados com outros nós, roteamento de mensagens, consultas sobre o estado dos nós e gerenciamento de grupos de nós.

Como um dos objetivos centrais da plataforma Jxta é a interoperabilidade, ela pode ser implementada em dispositivos computacionais variados e é fortemente baseada em padrões abertos, tais como o XML para formatação de mensagens e informações armazenadas pelos nós. A plataforma Jxta utiliza uma arquitetura de supernós denominados *rendez-vous peers* ou *relay peers*, caso assumam funções relacionadas ao gerenciamento de grupos de nós ou à travessia de NAT's e *firewalls* respectivamente.

2.5 Conclusão

No presente capítulo foi realizada uma exposição das principais características dos sistemas ponto-a-ponto, de técnicas fundamentais relacionadas à arquitetura e de alguns exemplos de protocolos e aplicações que empregam essas técnicas. No próximo capítulo será apresentado o paradigma de sistemas emergentes ou sistemas complexos adaptativos, que busca inspiração na forma como a natureza aborda a questão do relacionamento entre descentralização, resiliência e escalabilidade. Em seguida, será delineada uma proposta de arquitetura de ferramenta para computação ponto-a-ponto baseada nesse paradigma.

3. EMERGÊNCIA

Emergir significa vir à tona, tornar aparente, deixar de estar oculto. O emprego do termo emergência no domínio da engenharia de sistemas decorre da observação dos sistemas segundo uma perspectiva ascendente, ou seja, como um conjunto de componentes facilmente identificáveis que interagem entre si através de interfaces bem definidas. Em muitos casos essa perspectiva é suficiente para inferir-se com facilidade um comportamento sistêmico uma vez compreendidos separadamente os comportamentos de cada um dos componentes do sistema. Entretanto, existe uma categoria de sistemas na qual a compreensão de componentes em separado não revela de forma óbvia o comportamento sistêmico, o qual parece emergir quando o sistema é observado como um todo.

A necessidade de alternância de perspectivas para compreender os componentes de um sistema e o sistema como um todo caracteriza uma espécie de teste de emergência (RONALD et al., 1999). Nesse contexto, emergência pode então ser definida como a ocorrência de diferenças qualitativas entre os comportamentos de um sistema e de seus componentes, decorrentes da existência de relações causais complexas entre os mesmos.

O interesse da engenharia pela emergência decorre de dois fatos: primeiro, um comportamento sistêmico emergente é em geral muito mais complexo que o comportamento dos componentes do sistema em questão, o que implica a possibilidade de obtenção de sistemas complexos por meio da integração de componentes simples e, portanto, mais confiáveis, compreensíveis e baratos; segundo, sistemas com comportamento emergente em alguns casos possuem resiliência e capacidade de auto-organização para adaptação a mudanças ambientais. Entretanto assim como é difícil inferir o comportamento sistêmico emergente com base no conhecimento do comportamento dos componentes, também é difícil determinar quais comportamentos de componentes resultarão em um comportamento sistêmico especificamente desejado.

3.1. Animais coletivos

Uma possível abordagem para possibilitar o aproveitamento do fenômeno do comportamento emergente na engenharia é a análise da emergência em sistemas naturais, visando a determinação de princípios gerais que possam ser aplicados a projetos de sistemas. Alguns exemplos clássicos de modelos computacionais de comportamentos emergentes de sistemas naturais (PARUNAK, 1997), são descritos a seguir:

- **Planejamento de caminhos em colônias de formigas –** uma formiga, que pode ser vista como um componente de uma colônia de formigas, tem as capacidades básicas de caminhar aleatoriamente levando alguma carga ou não, identificar objetos e detectar no ambiente gradientes de substâncias químicas denominadas feromônios, produzidas por outras formigas em atividade. Entretanto, colônias de formigas são capazes de estabelecer caminhos curtos, e não de tamanho aleatório, seguidos pela maioria das formigas para deslocarem-se entre o ninho e as fontes de comida disponíveis. Para encontrar alimento, as formigas caminham aleatoriamente pelo ambiente, privilegiando a direção do gradiente de feromônios, caso esteja presente. Ao encontrarem uma porção de alimento, elas a carregam em direção ao ninho, depositando feromônios em todo o caminho no retorno. Dessa forma, caminhos curtos são inteiramente marcados por feromônios mais rapidamente e seguidos por outras formigas com mais facilidade, estabelecendo um processo de realimentação positiva. Em contrapartida, feromônios em caminhos mais longos ou relativos a fontes de comida já esgotadas são progressivamente evaporados pelo ambiente.
- **Organização interna das colônias de formigas –** uma colônia de formigas possui também a capacidade de formar agrupamentos de objetos tais como ovos, larvas e alimentos, organizados por tipo, dentro do ninho. Cada formiga caminha aleatoriamente pelo ninho e, caso não esteja carregando nada, ao encontrar um objeto decide se carrega o objeto com uma probabilidade que é tanto maior quanto menor tiver sido a proporção de objetos similares vistos no passado

recente. E caso esteja carregando um objeto, a intervalos regulares de tempo decide se deposita o objeto com uma probabilidade que é tanto maior quanto maior tiver sido a proporção de objetos similares vistos no passado recente. Este procedimento é suficiente para promover a organização global do ninho embora cada formiga execute apenas organizações locais a cada tarefa executada.

- **Diferenciação de tarefas em colônias de vespas** – em colônias de vespas, cada vespa toma apenas decisões individuais sobre que tarefa irá assumir, mas a colônia como um todo as divide em grupos de tamanho adequado para as necessidades de cada momento. A tarefa de cada vespa é definida por duas variáveis numéricas denominadas força e limiar. Quando duas vespas se encontram, elas se enfrentam e a vencedora é escolhida aleatoriamente, com maior probabilidade de vitória para a mais forte. Como resultado do enfrentamento, uma parte da força da perdedora é transferida para a vencedora. Cada vespa com força suficiente para se afastar do ninho decide também aleatoriamente se vai procurar comida. A probabilidade de decidir ir é tanto maior quanto maior for a demanda por comida do ninho e quanto menor for o limiar da vespa. Se uma vespa decide ir buscar comida, o seu limiar diminui; e se decide ficar perto do ninho, o mesmo aumenta. Esse comportamento tende a dividir as vespas em três grupos: um com baixa força e baixo limiar, o qual permanece perto do ninho para cuidar das larvas; um segundo grupo com alta força e baixo limiar, que vai buscar comida; e um terceiro grupo, unitário, correspondente a um líder com alta força e alto limiar, cuja função é permanecer perto do ninho, enfrentando as outras vespas para balancear o processo de diferenciação.
- **Movimento de cardumes e revoadas** - cada peixe ou ave movimenta-se tentando manter a mesma velocidade e direção dos vizinhos, uma distância mínima dos mesmos e uma distância pequena do centro do grupo. Conseqüentemente, o grupo inteiro consegue fazer manobras para evitar obstáculos ou predadores sem desfazer a formação. Embora os componentes desses grupos consigam fazer isoladamente manobras semelhantes às dos próprios grupos, o controle do movimento de uma coletividade de elementos

fisicamente desconectados e dotados de iniciativa própria é qualitativamente diferente do controle de um único corpo de indivíduo formado por partes fisicamente unidas e sem vontade independente.

- **Caça cooperativa em alcatéias** – um animal sozinho só pode caçar presas mais lentas que ele mesmo. Entretanto, se a velocidade da presa não for grande demais, um certo número de lobos conseguirá cercá-la ainda que ela seja mais rápida que os indivíduos do grupo. Para isso, cada lobo tenta chegar o mais perto possível da presa e manter-se longe dos outros lobos simultaneamente.

Em todos esses modelos, escolhas adequadas de funções e parâmetros resultaram em comportamentos similares aos dos sistemas naturais modelados, demonstrando experimentalmente como comportamentos individuais simples podem gerar comportamentos coletivos complexos permeados por auto-organização. Os sete princípios de engenharia observados por Parunak nesses modelos são os seguintes:

1) Modelagem orientadas a agentes: cada componente do sistema deve corresponder a um agente ou ao ambiente sobre o qual os agentes atuam. Parunak frisa que os agentes devem corresponder a objetos do domínio do problema e não a funções abstratas do sistema, como ocorre com os componentes no paradigma da decomposição funcional. As funções do sistema devem emergir das atividades dos agentes no ambiente;

2) Simplicidade de agentes: cada agente deve ser simples comparado ao sistema como um todo. No contexto de sistemas de *software*, simplicidade significa pequeno código, pequeno espaço de estados, localidade de interações, tempo curto de instanciação e memória de curto prazo. Se os agentes respeitarem esse princípio, as seguintes propriedades poderão ocorrer:

- a falha individual de um agente não terá impacto significativo sobre o sistema;
- cada agente será fácil de construir e compreender;
- uma grande população de agentes, cada um com um pequeno espaço de estados, permitirá a construção de um sistema com grande espaço de estados;

- a exclusão de informações e agentes obsoletos dará lugar a informações mais atualizadas e agentes mais adaptados ao estado corrente do sistema;
- cada agente terá que processar uma quantidade limitada de informações;
- falhas ficarão circunscritas a uma pequena região do ambiente;
- as interações serão de mais fácil compreensão;

3) Heterarquia: os agentes devem ser estruturalmente semelhantes e cada agente deve possuir a capacidade potencial de influenciar o comportamento de outros agentes de acordo com necessidades circunstanciais do sistema. Heterarquias são estrutura dinâmicas e não possuem pontos de centralização, ao contrário, por exemplo, das hierarquias, nas quais os componentes são organizados rigidamente em níveis e cada componente pode centralizar o controle do comportamento de vários componentes de nível inferior. A ausência de componentes centralizadores nas heterarquias elimina pontos singulares de falha e gargalos de desempenho, facilita a escalabilidade e favorece a simplicidade dos agentes;

4) Diversificação: embora a semelhança estrutural seja desejável, os agentes devem apresentar algum tipo de diversificação, ou seja, diferenças em seus estados individuais que os permitam monitorar aspectos distintos do ambiente. Dessa forma, um ambiente complexo poderá ser monitorado sem que o princípio de simplicidade dos agentes seja violado. A diversificação pode ser obtida através dos mecanismos de aleatoriedade e repulsão. A aleatoriedade consiste em atribuições probabilísticas de valores a variáveis de estado. A repulsão consiste em ajustar esses valores em cada agente para que se afastem daqueles atribuídos a variáveis de outros agentes. Uma exemplo típico de aleatoriedade é o caminhamento das formigas, que produz diversificação espacial. A repulsão por sua vez, pode ser exemplificada pelo espalhamento dos lobos, também destinado à diversificação espacial;

5) Drenagem de entropia: o sistema deve possuir um mecanismo de drenagem de entropia, que envolve a dissipação de alguma espécie de campo no ambiente ou entre agentes. Essa dissipação gera um fluxo que é percebido pelos agentes e utilizado para orientar suas atividades. Em contrapartida, as atividades dos agentes reorganizam o campo, realimentado-o positivamente. Dessa forma, a organização do

sistema em um nível macro, ou seja, a diminuição da entropia resultante da organização do campo, é compensada pelo aumento da entropia em nível micro representada pela dissipação do campo. Como resultado desse ciclo, qualquer estado de equilíbrio é imediatamente desfeito a fim de que o sistema possa atingir sucessivamente outros estados de equilíbrio mais apropriados em função da evolução das condições. Um exemplo desse mecanismo é o ciclo dos feromônios nas colônias de formigas, que são produzidos pelos agentes, utilizados para marcar caminhos e evaporados pelo ambiente. Outro exemplo são os enfrentamentos que promovem dissipação e concentração de força nos ninhos de vespas. Este princípio explicita ainda a necessidade de haver separação nítida entre os estados dos agentes e do ambiente, e do ambiente ter capacidade de propagar e descartar informações sem interferência dos agentes;

6) Mecanismo de aprendizado: o sistema deve prover mecanismos de aprendizado para agentes individualmente, por meio de alterações em seus estados, e para coletividades, por meio de alterações no estado do ambiente. Em ambos os casos, o aprendizado implica uma reorganização na estrutura do sistema (JOHNSON, 2003), em vez do simples armazenamento de um par *(padrão, resposta)* em algum tipo de componente de memória;

7) Planejamento e execução concorrentes: o planejamento e a execução de tarefas devem ser realizados concorrentemente, pois planejamentos antecipados não são eficazes em ambientes sujeitos a mudanças freqüentes. O planejamento concorrente com a execução tende a ser sub-ótimo, porém possibilita a adaptação às mudanças do ambiente, favorecendo a obtenção de soluções melhores que as planejadas antecipadamente com base em um estado do sistema que pode estar obsoleto no momento da execução.

3.3.1. Estigmatismo

Uma outra característica presente em diversos sistemas naturais com comportamento emergente é o estigmatismo (*stigmergy*), ou seja, a realização da comunicação entre agentes por meio de estigmas, ou seja, marcas, deixados no ambiente. O estigmatismo desempenha papéis fundamentais tanto no funcionamento

desses sistemas naturais, como em uma série de algoritmos computacionais inspirados nos mesmos (DORIGO et al., 2000). Um exemplo típico de comunicação estigmática é a geração de trilhas de feromônios nas colônias de formigas.

Embora não seja citada entre os princípios de engenharia de Parunak, a comunicação estigmática interage com esses princípios de diversas maneiras:

- favorece a simplicidade dos agentes, eliminando a necessidade de protocolos e interfaces de comunicação síncronos com outros agentes;
- é compatível com o princípio da organização heterárquica, uma vez que permite comunicação sem estabelecimento de referências explícitas;
- é conveniente para a implementação dos campos de informação dos mecanismos de drenagem de entropia;
- pode ser utilizada como ferramenta de integração entre mecanismos de aprendizado individual e sistêmico.

3.2. Cérebros e super-cérebros

Outro exemplo notável de emergência na natureza é o funcionamento dos sistemas nervosos e, mais particularmente, do cérebro humano. Os componentes básicos do cérebro são os neurônios, células que, embora possuem capacidades muito simples de processamento de informações quando observadas isoladamente, formam em conjunto um sistema de controle para as complexas manifestações cognitivas e comportamentais do ser humano. Os neurônios criam entre si conexões unidirecionais de força variável denominadas sinapses, que permitem que cada neurônio informe seu estado de ativação a outros neurônios por meio de mecanismos eletro-químicos. O estado de ativação de cada neurônio é função dos estados de ativação informados por outros neurônios e da força das sinapses correspondentes. Os processos de aprendizado (e esquecimento) no cérebro correspondem, grosso modo, ao ajuste da estrutura da rede de neurônios em termos de formação, destruição, reforço e enfraquecimento de sinapses. Embora o estágio ainda superficial de conhecimento sobre o funcionamento do cérebro e as limitações tecnológicas atuais coloquem a realização de simulações apuradas fora de alcance, a pesquisa e o desenvolvimento na área de redes neurais

artificiais já demonstraram a efetividade dessa abordagem conexionista para processamento de informações em uma gama variada de aplicações (WIDROW et al., 1994).

Além da abordagem conexionista, outro princípio estrutural que diferencia o cérebro das coletividades de animais apresentadas na seção precedente é a aparente organização recursiva dos neurônios em subredes às quais MINSKY (1988) denominou agências. Segundo esse modelo, cada agência é especializada em alguma função, como em uma sociedade, e qualquer comportamento produzido pelo cérebro, por mais complexo que seja, emerge da colaboração de agências apropriadas quando adequadamente conectadas.

Um dos comportamentos que emergem no cérebro é a sociabilidade, que resulta na criação de agrupamentos humanos organizados, como as cidades, por exemplo. Segundo a perspectiva de JOHNSON (2003), cidades podem ser consideradas super-cérebros, ou seja, sistemas complexos cujos componentes primordiais são os cérebros humanos. Esses sistemas enquadram-se novamente no nível de coletividade de seres autônomos, mas que, nesse caso, são dotados de capacidades de empregar padrões complexos de comunicação síncrona ou assíncrona, de obter e empregar conhecimento global sobre o sistema, de acumular e aprimorar conhecimento obtido por gerações anteriores, e de construir artefatos para a implementação de novas formas de processamento e transmissão de informações, como, por exemplo, as redes de computadores, que introduzem nesse nível de organização mecanismos conexionistas não fornecidos inicialmente pela natureza.

Johnson ressaltou ainda a importância da organização espacial nesse tipo de sistema, ou seja, a formação de agrupamentos de elementos que desempenham funções similares, e do estabelecimento de mecanismos de retroalimentação positiva e negativa, como instrumentos de aprendizado e manutenção da homeostase respectivamente.

3.3. Emergência em sistemas ponto-a-ponto

Nas seções anteriores, relacionamos uma série de princípios de engenharia observados em sistemas naturais de comportamento emergente. Uma vez que tais

sistemas geralmente envolvem grande número de componentes, são auto-organizáveis, são adaptáveis a situações imprevistas e são tolerantes a falhas de componentes, a aplicação de seus princípios na construção de sistemas ponto-a-ponto parece uma abordagem promissora para a obtenção de escalabilidade e resiliência de forma descentralizada em sistemas ponto-a-ponto. Alguns desses princípios podem ser incorporados em uma infraestrutura para desenvolvimento de sistemas ponto-a-ponto, ao passo que os demais podem servir de base para recomendações ou padrões de projeto de aplicações construídas sobre essa infraestrutura. Nossa proposta para a arquitetura dessa infraestrutura baseia-se na combinação dos paradigmas de agentes móveis de *software* e de espaços compartilhados.

Agentes são componentes de *software* que encapsulam encapsulam código, estado e controle de invocação. PARUNAK (1997) relaciona como vantagens da orientação a agentes sobre a orientação a objetos tradicional a redução de custos de integração e manutenção, e o aumento da adaptabilidade e da robustez. Um agente móvel é um agente que possui a capacidade adicional de se transportar entre os nós de uma rede preservando seu estado interno e de realizar processamentos localmente nesses nós.

Espaços compartilhados (EUGSTER et al., 2003) são abstrações de comunicações que permitem trocas de informações entre entidades que não possuem qualquer conhecimento uma da outra, que podem estar hospedadas em nós diferentes do sistema e que podem existir em intervalos de tempo disjuntos. Um espaço compartilhado tem a capacidade de armazenar dados e assinaturas por dados compatíveis com modelos especificados. Quando uma entidade escreve um dado, o espaço compartilhado identifica assinaturas associadas a modelos compatíveis com o dado e notifica as entidades que as depositaram. Entidades podem recuperar dados escritos no sistema apresentando um modelo compatível com os dados desejados, seja em resposta a uma notificação recebida ou por iniciativa própria.

Da observação dos princípios de sistemas naturais estudados foram derivadas as seguintes diretrizes para orientar a concepção da arquitetura lógica proposta neste trabalho:

- um sistema abrange um ou mais ambientes lógicos, cada um deles composto por

um conjunto de espaços compartilhados interconectáveis, denominados células;

- cada nó do sistema hospeda no máximo uma célula de cada ambiente;
- um ambiente é utilizado por agentes de *software* que podem realizar operações sobre a célula hospedada em seus próprios nós ou sobre células conectadas à mesma;
- além das operações tradicionais de espaços compartilhados, agentes podem conectar a uma célula a outra célula do ambiente identificada pelo endereço do nó que a hospeda, ou eliminar conexões existentes;
- agentes de *software* podem mover-se do nó corrente para outro nó de endereço conhecido que contenha uma célula do mesmo ambiente;
- objetos escritos nas células pelos agentes encapsulam comportamentos de controle de propagação para células conectadas.

A utilização de agentes móveis de *software* dá suporte aos princípios da modelagem orientada a agentes e do controle heterárquico. Sua adoção como primitiva do modelo facilita o mapeamento de objetos de domínios de aplicações e elimina a necessidade dos programadores estabelecerem hierarquias artificiais de controle e de gerenciarem explicitamente múltiplas linhas de execução (*threads*). Entretanto, a interface computacional entre os agentes humanos e os espaços compartilhado pode ser implementada adequadamente por meio de objetos de aplicação tradicional. O modelo não estabelece restrições ao grau de complexidade individual dos agentes, por tratar-se de uma questão dependente de domínio de aplicação.

As primitivas de estabelecimento e fechamento de conexões entre espaços compartilhados dão suporte à implementação de sistemas auto-organizáveis em estilo conexionista. Por outro lado, ao representar partições do ambiente como espaços compartilhados que podem ser utilizados por entidades residentes no mesmo nó ou nós vizinho, o modelo incorpora os princípios de comunicação estigmática e de localidade de interações. Entretanto, não há perda de generalidade em relação às possibilidades de implementação de algoritmos distribuídos, uma vez que outros paradigmas de comunicação podem ser simulados à partir das operações primitivas do espaço compartilhado.

A especificação das entradas dos espaços compartilhados como objetos dotados de comportamento de propagação provê ao ambiente a capacidade de propagar e descartar informações sem interferência dos agentes.

O modelo proposto assemelha-se em vários pontos ao da infraestrutura Anthill (BABAOGLU, 2002). Entretanto, Anthill não implementa a propagação de dados pelo ambiente sem interferência de agentes. Outra proposta semelhante é o *middleware* TOTA (MAMEI et al., 2005), que oferece mecanismos de coordenação estigmática entre agentes para aplicações de redes ad hoc.

No capítulo seguinte, será apresentada a arquitetura lógica proposta.

4. ARQUITETURA Coppeer

A plataforma Coppeer é um projeto de pesquisa ativo do Laboratório de Bancos de Dados da COPPE/UFRJ. A primeira versão da plataforma foi concebida como uma infraestrutura ponto-a-ponto para apoiar arquiteturas de computação em grade tais como a *Open Grid Services Architecture* (OGSA – Arquitetura Aberta para Serviços em Grades), por exemplo. Em especial, ela foi a plataforma para o *Collaborative Ontology Editor* (COE – Editor Cooperativo de Ontologias), que apóia descrições semânticas de serviços em grade por meio de ontologias e permite a troca desse tipo de conhecimento entre pessoas em um domínio (BRAGA, 2004).

Atualmente, o principal objetivo da plataforma é a implementação de aplicações colaborativas em ambientes ponto-a-ponto. Na seção 4.1 é descrita uma arquitetura lógica para a plataforma, concebida com base em alguns dos princípios de sistemas emergentes estudados. Na seção 4.2, é apresentado um método de descoberta de recursos compatível com a arquitetura.

4.1. Descrição da arquitetura

A arquitetura lógica da plataforma Coppeer foi concebida tendo em vista a necessidade de conjugar elementos voltados para a implementação de princípios de emergência com facilidades para a integração de componentes externos, em geral construídos segundo paradigmas não-emergentes, necessários para suprir requisitos de aplicações colaborativas. Além disso, foi considerada a conveniência de prever um elemento responsável pela descoberta de recursos na rede ponto-a-ponto.

A arquitetura é baseada em agentes e espaços compartilhados, conforme as diretrizes propostas no capítulo anterior, a fim de contemplar a questão da emergência; a abordagem para conjugar as demais necessidades é atribuição de diferentes papéis aos agentes. Os componentes da arquitetura são os seguintes:

- Agência - é o componente da arquitetura presente em cada nó de uma rede Coppeer, responsável por oferecer interfaces de administração ao usuário e de serviços para clientes externos à plataforma Coppeer, por gerenciar agentes e

células presentes no nó e por intermediar transmissões de entradas entre agentes e células localizadas em nós distintos.

- Célula - uma célula é um espaço compartilhado similar aos da especificação Javaspace (SUN, 1999). Cada célula dentro de uma agência é identificada por um nome de ambiente e oferece para os agentes primitivas para escrita e leitura de entradas, para assinatura por notificações de escrita de entradas contendo dados especificados, e para criação e exclusão de conexões com células do mesmo ambiente localizadas em outras agências.
- Entrada - Uma entrada é um *container* de dados trocados entre agentes capaz de, quando armazenado em uma célula, gerar novas células para serem escritas em células vizinhas e alterar seu estado interno sem interferência de agentes.
- Agente - Um agente é uma entidade dotada de código e estado, associada a um ambiente, e que invoca as primitivas das células e move-se entre agências para executar computações distribuídas. Os agentes podem criar novos agentes, mover-se da agência corrente para uma agência contendo uma célula vizinha do mesmo ambiente e invocar primitivas da célula corrente e das células vizinhas.

Os agentes podem desempenhar os seguintes papéis:

- Micro-agente – corresponde a agentes destinados a realizar computações distribuídas segundo o paradigma da emergência. Agentes que desempenham este papel devem possuir código e espaço de estados simples e comunicar-se exclusivamente por intermédio das células.
- Agente fachada – corresponde a agentes que, além de comunicar-se com outros agentes por intermédio das células, podem ser invocados por sua agência para tratar requisições de serviços feitas por clientes externos à plataforma Coppeer. Dessa forma, esse papel permite que um sistema emergente, composto por diversos agentes, ofereça uma interface única para sistemas externos, mesmo que estes sejam construídos sob outros paradigmas.
- Agente adaptador - corresponde a agentes que, além de comunicar-se com outros agentes por intermédio das células, podem invocar primitivas de componentes externos à plataforma Coppeer. Dessa forma, esse papel permite que um sistema

emergente as interfaces de outros sistemas, mesmo que estes sistemas sejam construídos sob outros paradigmas.

- Agente diretório – corresponde a agentes destinados a organizar informações sobre recursos presentes em outros nós do sistema e disponibilizá-las para os outros agentes.

A arquitetura descrita é ilustrada na figura 4.1.

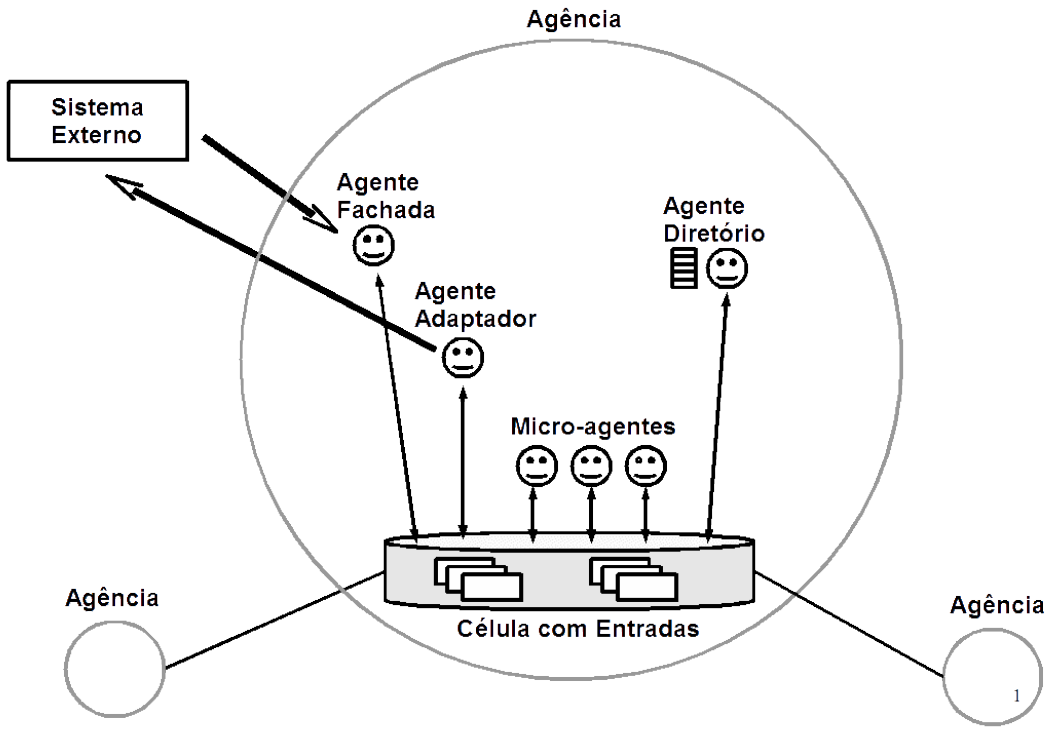

Figura 4.1. Arquitetura Coppeer

4.2 Descoberta de recursos na arquitetura Coppeer

Conforme o exposto no capítulo 2, a descoberta de recursos em um sistema ponto-a-ponto é uma funcionalidades essencial para que o mesmo possa aproveitar as capacidades de seus nós constituintes. A arquitetura da plataforma Coppeer prevê a existência de agentes diretórios nos nós da rede, destinados a organizar informações sobre recursos presentes em outros nós. Nesta seção, será descrita uma abordagem de

descoberta de recursos apropriada para a plataforma, denominada método TIGRAS, acrônimo para *Topology-Independent Gradient Search* – busca de gradiente independente de topologia (MIRANDA et al., 2008). A abordagem conjuga características de descoberta por propagação seletiva e por roteamento de recursos, procurando permitir a realização de buscas eficientes em redes não-estruturadas. O método TIGRAS é dividido em um método básico, baseado nos princípios de emergência previamente estudados, e por um método multi-estágios, no qual um conjunto instâncias do método básico é empregado para obtenção de melhor escalabilidade.

O método básico tem como princípio inspirador principal a capacidade das formigas orientarem-se em um ambiente pelo rastreamento de gradientes de feromônios. No contexto do método básico, o elemento correspondente ao campo de feromônios é um campo de informações indicativas das distâncias estimadas de cada nó a cada recurso presente na rede como um todo. As buscas são realizadas por micro-agentes que, a cada passo, movem-se para o nó vizinho que aparenta estar mais próximo do nó que hospeda o recurso desejado, de acordo com as informações do campo disponíveis localmente.

Os recursos são representados por chaves únicas, de forma que o conteúdo de cada nó pode ser representado por filtros Bloom. Conforme o exposto no capítulo 2, um filtro bloom é um vetor de bits de tamanho fixo capaz de armazenar informações de pertinência a conjuntos de tamanho indeterminado, mediante a aceitação de alguma probabilidade de ocorrência de falsos positivos.

Os filtros Bloom construídos no método TIGRAS são propagados pela rede para permitir a construção do campo de informações nos outros nós. No referente à construção desse campo, o método básico diverge ligeiramente do princípio encontrado nas colônias de formigas. Enquanto as trilhas de feromônios são produzidas por formigas que localizaram um recurso e percorreram o ambiente previamente, o campo de informação no método básico é construído por agentes diretório que permanecem fixos em seus nós, coletando e combinando filtros Bloom que são propagados novamente pela rede. Esta opção se deve ao fato de que, nas colônias de formigas, a formação das trilhas de feromônios é iniciada por indivíduos que encontraram a fonte de

recursos após buscas aleatórias, por vezes implicando a perda de parte da população da colônia, caracterizando um processo que seria pouco eficiente ao ser transposto para um ambiente computacional no qual uma grande variedade de recursos precisa ser potencialmente localizado à partir de qualquer nó do sistema. O método básico procura, então, construir deterministicamente um campo que corresponderia, em termos conceituais, a todas as trilhas possíveis para cada um dos recursos presentes na rede.

O restante dessa seção está organizado como se segue: na subseção 4.2.1, o funcionamento do método básico é descrito em maiores detalhes ; na subseção 4.2.2 o método multi-estágios é caracterizado ; e na seção 4.2.3 alguns resultados da avaliação experimental do método TIGRAS são apresentados.

4.2.1 Método básico

O método básico utiliza filtros Bloom para armazenar em cada nó da rede sumários das localizações das chaves presentes na rede inteira. Estes sumários são construídos de forma que cada nó armazena e encaminha para seus vizinhos uma quantidade de dados suficiente para permitir a execução de buscas eficientes sem, entretanto, ocasionar sobrecarga no sistema.

O agente diretório de cada nó da rede armazena o último sumário que foi construído e enviado por cada um dos agentes diretórios dos nós vizinhos. Cada sumário consiste em uma tabela contendo uma quantidade variável de filtros Bloom que representa o conhecimento que o agente que o construiu tem à respeito do conteúdo da rede como um todo.

Vamos denominar a *i*-ésima entrada de uma tabela como filtro nível-*i*. Cada tabela é construída de forma que o filtro nível-*i* descreve o conteúdo de todos os nós que podem ser alcançados por meio de *i* saltos à partir do nó que a construiu, incluindo nós que podem ser alcançados por meio de um caminho alternativo com menos de *i* saltos. Dessa forma, o filtro nível-0 descreve o conteúdo do próprio nó que construiu a tabela; o filtro nível-1 descreve o conteúdo de todos os vizinhos imediatos; o filtro nível-2 descreve o conteúdo dos vizinhos dos vizinhos e assim por diante. Em consequência, quanto maior é o nível de um filtro, maior é a porção da rede descrita pelo mesmo, e

mais alta é a taxa de falsos positivos para verificação de pertinência de chaves no filtro. Entretanto, se existe uma relação apropriada entre o tamanho do filtro e o número de chaves distintas presentes na rede, buscas que utilizem apropriadamente as informações dos filtros poderão compensar passos errôneos por meio do mecanismo descrito mais adiante.

As informações sobre o conteúdo de um dado nó são propagadas pela rede por meio de um mecanismo de "fofoca" (*gossiping*). Para construir uma tabela de filtros Bloom, o agente diretório do nó usa informações locais e tabelas enviadas anteriormente pelos agentes dos nós vizinhos. A tabela resultante é enviada de volta para todos os vizinhos do nó e o procedimento é repetido iterativamente. As informações locais são utilizadas para construir o filtro nível-0 e , para todo *i* maior que zero, o filtro nível-(*i*+1) recebe o valor da união dos filtros nível-*i* das tabelas recebidas dos vizinhos.

Este procedimento resulta em tabelas corretas, uma vez que nós que podem ser atingidos depois de *i* saltos à partir de um vizinho de um dado nó podem ser atingidos após (*i*+1) saltos à partir do próprio nó. A natureza iterativa do processo mantém as tabelas atualizadas, propagando pela rede alterações nos filtros correspondentes a alterações ocorridas no conteúdo dos nós e na topologia. A figura 4.2 ilustra a construção da tabela de um nó *P* após sua entrada em uma rede.

Redes muito dinâmicas exigem alta taxa de atualização, e portanto alta utilização de banda, para manter as tabelas precisas. Contudo, aplicações que não exigem consistência forte em redes relativamente estáveis demandam baixas taxas de atualização e de utilização de banda. Seja *t* um parâmetro do sistema representando o intervalo entre duas atualizações de tabela nos nós. A estimativa pessimista para o tempo total necessário para publicar uma modificação de conteúdo ou topologia será *t.d*, onde *d* é o diâmetro da rede.

A simplicidade do procedimento descrito implica a introdução de circularidade nas informações representadas pelos filtros. Dessa forma, alguma condição precisa ser estabelecida para limitar o número de níveis na tabelas de filtros. Uma solução possível é estimar o diâmetro *d* da rede e construir tabelas com no máximo *d* níveis. Um método mais preciso pode ser aplicado se o filtro que representa todo o conjunto de chaves da rede for conhecido. Neste caso, pode-se interromper a adição de

novos níveis a uma tabela à partir no momento em que este filtro for encontrado em seu último nível. Por exemplo, se o tamanho dos filtros for menor ou igual ao número de chaves distintas na rede, o conjunto de todas as chaves é representado por um filtro contendo apenas bits com valor 1.

Com o auxílio das tabelas de filtros Bloom construídas conforme o descrito acima, é possível executar buscas em redes de topologia arbitrária com complexidade algorítmica menor que a de métodos de inundação de consultas. No método básico, o nó que gera uma consulta cria um micro-agente de busca que procura mover-se em cada salto para o nó vizinho ainda não visitado que aparenta estar mais próximo do nó que contém a chave desejada. Durante a busca, o micro-agente constrói uma lista ordenada de nós já visitados, na qual cada nó aparece apenas uma vez. A distância de um determinado vizinho ao nó que contém a chave desejada é estimada por meio da verificação do filtro de menor nível na tabela enviada pelo vizinho no qual a chave desejada está presente. Se todas as tabelas na rede estão precisas, o micro-agente segue o caminho mais curto até o nó que contém a chave desejada, conforme o ilustrado na figura 4.3 . Caso contrário, ao detectar que está em um caminho errôneo, o micro-agente retorna para algum nó já visitado e reinicia a busca daquele ponto, mantendo a lista de nós já visitados em sua memória.

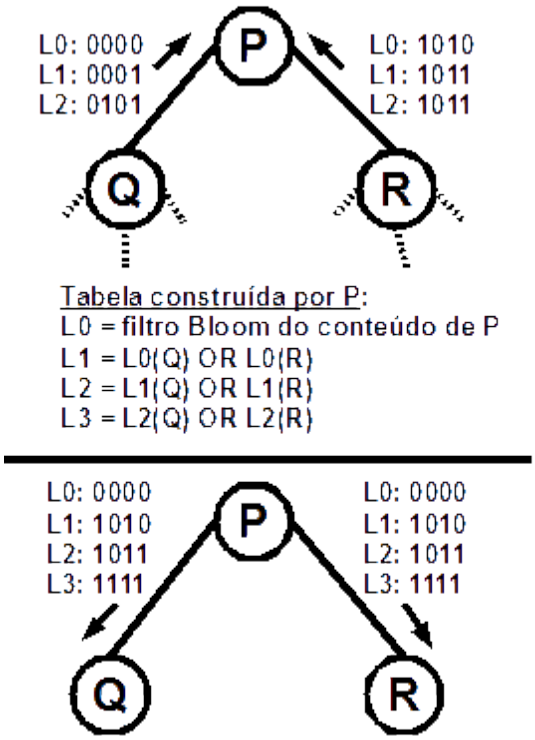

Figura 4.2. Construção de tabela de filtros

Imprecisões nas tabelas podem ser causadas por alterações na topologia da rede ou pela imprecisão intrínseca dos filtros Bloom. Um agente detecta que está em um caminho errôneo e reinicia a busca em um dos três casos:

- se nenhuma tabela de vizinho contém a chave desejada em algum filtro;
- se todos os vizinhos estão presentes na lista de nós já visitados; ou
- se a distância estimada ao nó contendo a chave desejada não decresce após um salto.

Como a lista de nós já visitados é preservada, cada vez que a busca é reiniciada um novo caminho é examinado. Inicialmente, os micro-agentes selecionam o primeiro nó da lista de nós visitados – ou seja, o nó que gerou a consulta – como ponto de reinício. Se em algum momento o agente detecta que está em um caminho errôneo, mas verifica que o ponto de reinício é o próprio nó onde ele está presente, o sucessor do ponto de reinício corrente na lista de nós já visitados é selecionado como novo ponto de reinício para que a busca possa prosseguir; e caso o ponto de reinício corrente seja o

último nó da lista de visitados, a busca é abortada.

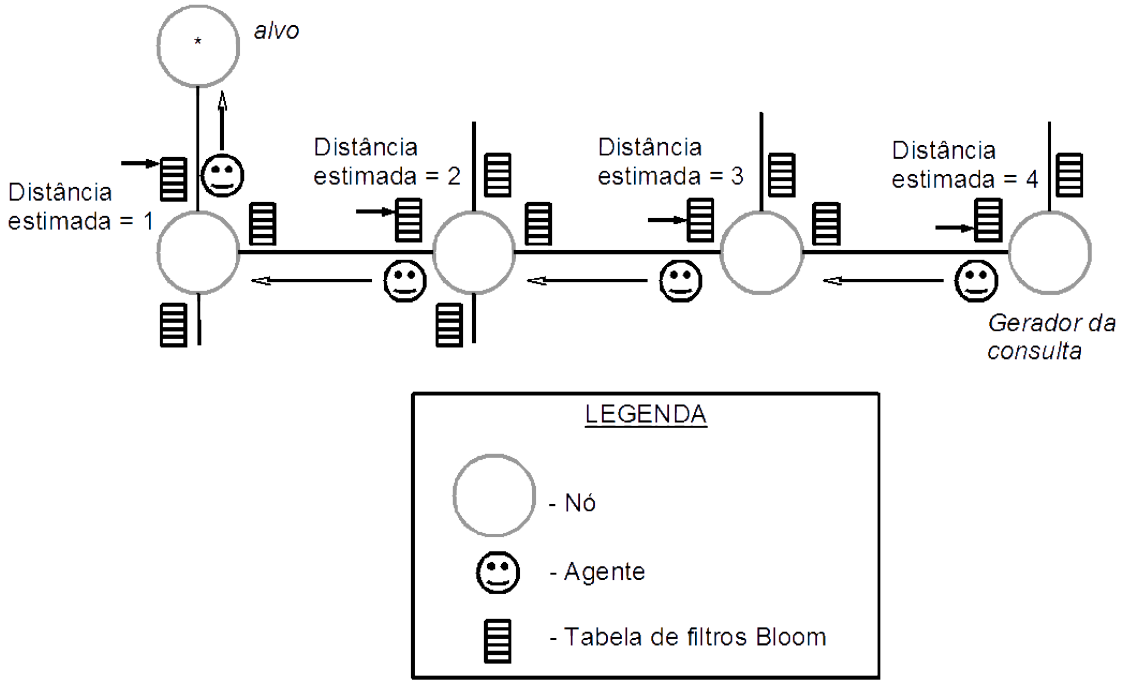

Figura 4.3 – Busca no Método Básico

O comportamento de um micro-agente de busca é resumido em formato de pseudo-código a seguir:

/ <u>Funções auxiliares</u>

Nó_Corrente( ): retorna uma referência para o nó em que o agente está.

Contém(lista_de_nós,nó): verifica se um nó está presente em uma lista.

Insira(nó, lista_de_nós): insere um nó em uma lista.

Armazena(nó,chave): verifica se um nó armazena uma chave.

Vizinhos( ): retorna uma lista de referências para os nós vizinhos do nó corrente.

Primeiro_Nível(nó,chave): verifica nas tabela de filtros Bloom de um nó, a distância estimada de uma chave ao nó, ou seja, o nível do primeiro filtro onde a chave é encontrada.

```
Tamanho(lista_de_nós): retorna o número de elementos de uma lista.
Mova_Para(nó): move o agente para outro nó. */


/* Esta função é invocada quando o agente é criado */
Inicializa(Chave_Alvo){
        /* o estado destas variáveis é preservado quando o agente move-se entre os nós
           da rede */
        Chave := Chave_Alvo;
        Próximo_Nó := NIL;
        Terminou := 'F';
        Distância_Estimada := INFINITO;
        Reinício := 0;
        Visitados := VAZIO;
}

/* Esta função é invocada quando o agente é criado e cada vez que se move para um
novo nó */
Age( ){
        SE (Terminou = 'F') {
                SE (NÃO Contém(Visitados,Nó_Corrente( )))
                        Insira(Nó_Corrente(), Visitados);
                SE ( Armazena(Nó_Corrente( ),Chave) ) Terminou := 'V'; /*ACHOU*/
                SENÃO {
                        Nível := 0;
                        Nível_Mínimo := INFINITO;

                        /* Avalia qual vizinho ainda não visitado tem menor distância
                           estimada ao objetivo*/
                        PARA CADA Vizinho EM Vizinhos( ) FAÇA {
                                Nível := Primeiro_Nível(Vizinho,Chave);
                                SE (Nível >= 0) ENTÃO {
                                        SE (Nível < Nível_Mínimo) ENTÃO {
                                                SE (NÃO Contém(Visitados,Vizinho)) {
                                                        Nível_Mínimo := Nível;
                                                        Próximo_Nó := Vizinho;
                                                }
                                        }
                                }
                        }

                        /* Avalia se haverá progresso ao mover-se para o vizinho
                           selecionado. Caso afirmativo, prossegue. Caso negativo,
                           retorna para um nó já visitado ou aborta a busca */
                        SE ((Nível_Mínimo >= Distância_Estimada) OU
                             (Próximo_Nó = NIL)) {
```

```
                    Próximo_Nó := Visitados[Reinício];
                    SE (Próximo_Nó := Nó_Corrente( )) {
                        Reinício := Reinício + 1;
                        SE (Reinício < Tamanho(Visitados)) {
                            Próximo_Nó := Visitados[Reinício];
                        } SENÃO {
                            Terminou := 'V'; /*ABORTOU*/
                        }
                    }
                    Distância_Estimada := INFINITO;
                } SENÃO Distância_Estimada := Nível_Mínimo;
                Mova_Para(Próximo_Nó);
                Próximo_Nó = NIL;
            }
        }
}
```

4.2.2 Método multi-estágios

Conforme visto na seção anterior, o método básico apóia-se na relação entre o tamanho dos filtros o número de chaves distintas na rede. Mais especificamente, quanto maior for o número de chaves distintas, maior será o tamanho de filtro necessário para permitir buscas eficientes. Filtros grandes consumirão mais recursos computacionais para serem processados, armazenados e transmitidos pela rede. Para obter melhor escalabilidade, foi concebido um método de busca multi-estágios que utiliza um pequeno número de instâncias do método básico como blocos constituintes.

O fundamento do método multi-estágios é a distribuição probabilística das chaves pela rede, de forma que cada estágio da busca ocorra em uma região apropriada da mesma, com base em tabelas de filtros muito menores que as necessárias para executar uma busca equivalente de único estágio, na rede inteira, em tempo semelhante.

A publicação no método multi-estágios consiste na supracitada distribuição probabilística de chaves pela rede. Em um sistema de s estágios, uma chave deve ser vista como a concatenação de s dígitos $D_0D_1...D_{s-1}$, onde cada dígito pode assumir k valores distintos. Dessa forma, tal sistema poderá gerenciar k^s chaves distintas.

O sistema é composto por s instâncias independentes do método básico e a i-ésima instância é responsável por gerenciar subchaves de i dígitos na forma $D_0D_1...D_{i-1}$,

com i variando de 1 a s. Além disso, existe uma seqüência $h_0, h_1,..., h_{s-1}$ de constantes do sistema.

Quando um nó publica uma dada chave $D_0D_1...D_{s-1}$, o sistema distribui a chave e todas suas subchaves – caso as mesmas ainda não estejam presentes no sistema, de acordo com a seguinte regra: a distância do nó que armazena a subchave $D_0D_1...D_{i-1}$ – usando a i-ésima instância do método básico, ao nó que armazena a subchave $D_0D_1...D_{i-1}D_i$ – usando a $(i+1)$-ésima instância, deve ser no máximo h_i.

Para cumprir a regra, o nó que publica a chave inicia uma publicação descobrindo o nó que contém a subchave D_0 ou selecionando, de acordo com o protocolo apresentado no próximo parágrafo, um nó distante no máximo h_0 saltos para armazená-la. Depois disso, o nó descoberto ou selecionado repete o procedimento para a subchave D_0D_1 considerando a distância h_1 e assim por diante. Ao final da publicação, o nó contendo a subchave $D_0D_1...D_{s-2}$ descobre o nó que contém a chave $D_0D_1...D_{s-1}$ ou seleciona um nó distante no máximo h_{s-1} saltos para armazená-la.

Como múltiplos nós podem iniciar a publicação de chaves contendo a mesma subchave simultaneamente, é necessário o emprego de um protocolo para evitar a eventual seleção de mais de um nó intermediário para armazenar essa subchave. O protocolo determina que as subchaves sejam armazenadas com dois estados possíveis: estado *publicando* e estado *final*. Quando um nó P precisa descobrir ou selecionar outro nó para armazenar uma subchave, ele usa a instância apropriada para realizar uma busca do método básico pela subchave, o que resulta em três situações possíveis:

- se a busca é bem sucedida e o estado da subchave localizada é *publicando*, o nó P aguarda um período aleatório de tempo e repete o procedimento;
- se a busca é bem sucedida e o estado da subchave localizada é *final*, o nó que armazena a subchave torna-se responsável por realizar o próximo estágio da publicação, conforme descrito anteriormente;
- se a busca fracassa, o nó P armazena localmente a subchave no estado *publicando*, aguarda um período de tempo aleatório e realiza uma nova busca pela subchave. Se esta nova busca é bem sucedida, o nó P remove a subchave de sua coleção e age como nas situações anteriores. E caso a nova busca fracasse, um agente de caminhamento aleatório é empregado para selecionar um nó Q

localizado a uma distância apropriada de *P*, a subchave é armazenada no estado *final* no nó *Q* e é então removida de *P*.

Os períodos de tempo utilizados no protocolo devem ser maiores que o tempo estimado para a propagação das informações de filtros Bloom de qualquer nó por toda a rede, ou seja, o tempo necessário para finalizar uma publicação no método básico.

A busca multi-estágios pela chave $D_0D_1...D_{s-1}$ consiste em uma seqüência de *s* buscas básicas nas instâncias apropriadas pelas subchaves D_0, D_0D_1, $D_0D_1D_2$, e assim por diante, finalizando com a busca pela chave $D_0D_1...D_{s-1}$ na última instância. A figura 4.4 ilustra um exemplo de uma busca de três estágios. Para localizar a chave *311*, o sistema primeiramente busca a subchave *3* utilizando a primeira instância TIGRAS. Depois disso, o sistema utiliza a segunda instância para buscar a subchave *31* e, finalmente, utiliza a terceira instância para buscar a chave *311*.

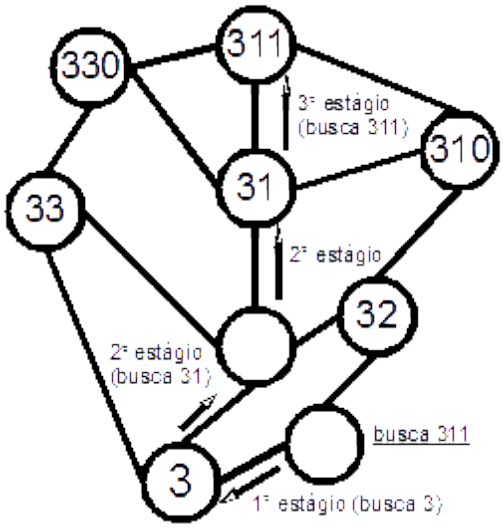

Figura 4.4. Busca multi-estágios

O método de publicação multi-estágios assegura que se um agente de busca atinge o nó contendo a subchave $D_0D_1...D_{i-1}$ e se as tabelas da $(i+1)$-ésima instância de método básico forem acuradas, o agente pode orientar-se por estas tabelas para atingir o nó contendo a subchave $D_0D_1...D_{i-1}D_i$ após no máximo h_i saltos. Imprecisões nas tabelas

são tratadas pelo método básico de busca conforme o descrito na seção anterior.

4.2.3 Avaliação experimental

Alguns experimentos de simulação foram executados a fim de verificar os seguintes pontos:

- a influência do tamanho dos filtros no desempenho do método básico;
- a influência do tamanho da rede no desempenho do método básico;
- o desempenho da busca no método multi-estágios.

Todas as simulações foram executadas em um programa Java desenvolvido especificamente com esse fim. O programa gerou redes aleatórias de forma que cada novo nó adicionado à simulação era conectado a cerca de *log (n)* nós escolhidos entre os que foram previamente adicionados, sendo *n* o número final de nós na rede simulada. Após a geração da rede, o programa distribuía 1000 chaves distintas pela rede, criava a tabelas de filtros e simulava a propagação das informações na rede. Finalmente, o programa buscava todas as chaves à partir de um dado nó da rede, calculando o número médio de saltos necessários para localizar as chaves. Cada resultado mostrado nesta seção é a média de 10 execuções do programa.

No primeiro grupo de simulações, o programa simulou redes contendo 10000 nós, com tamanho de filtro variando de 100 a 2000 bits. Os resultados mostrados na figura 4.5 indicam a estabilização do desempenho do método básico quando o tamanho do filtro aproxima-se do número de chaves distintas. O número de saltos nessa região da figura é de aproximadamente 3 saltos, o que se aproxima do mínimo possível para as redes simuladas nos experimentos.

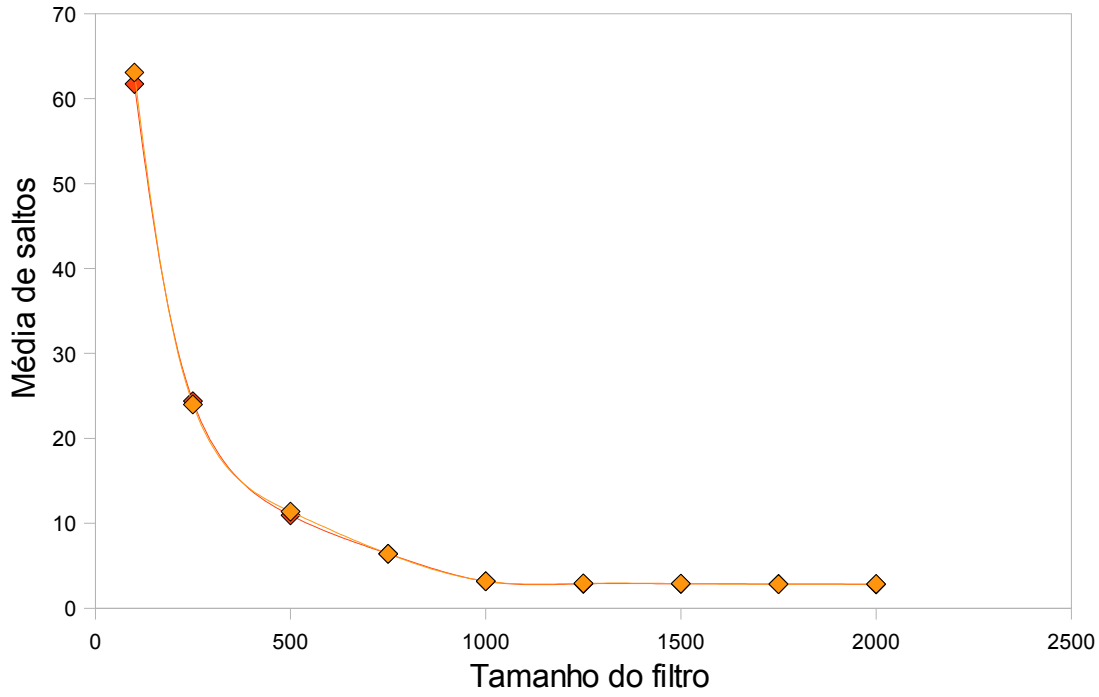

Figura 4.5. Influência do tamanho de filtro no método básico

No segundo grupo de simulações, o tamanho do filtro foi fixado em 1000 bits, com tamanho da rede variando entre 100 e 1000 nós. A aparência logarítmica da curva resultante, mostrada na figura 4.6, é consistente com o resultado anterior, uma vez que o número médio de saltos cresce de forma similar ao diâmetro da rede aleatória.

O último grupo de experimentos simulou uma rede de 3 estágios em um sistema de chaves de dígitos decimais com $h_0 = 3$, $h_1 = 2$ e $h_2 = 1$. O tamanho da rede foi fixado em 10000 nós, com tamanho de filtro variando de 10 – o número de dígitos distintos, a 333 bits. Os resultados são mostrados na figura 4.7. Como cada nó hospeda 3 tabelas de filtros, a figura apresenta o triplo do tamanho de filtro, a fim de permitir comparações razoáveis com os resultados do método básico. Um tamanho de filtro de 10 bits – que significa 30 bits ao considerarmos as 3 tabelas, resulta em 5.6 saltos em média, o que é consistente com as escolhas das constantes h. O método básico necessita de cerca de 750 bits para apresentar resultados similares. Verificamos ainda que é necessário triplicar o tamanho de filtro para obter 10% de ganho e que o desempenho

permanece praticamente estável deste ponto em diante.

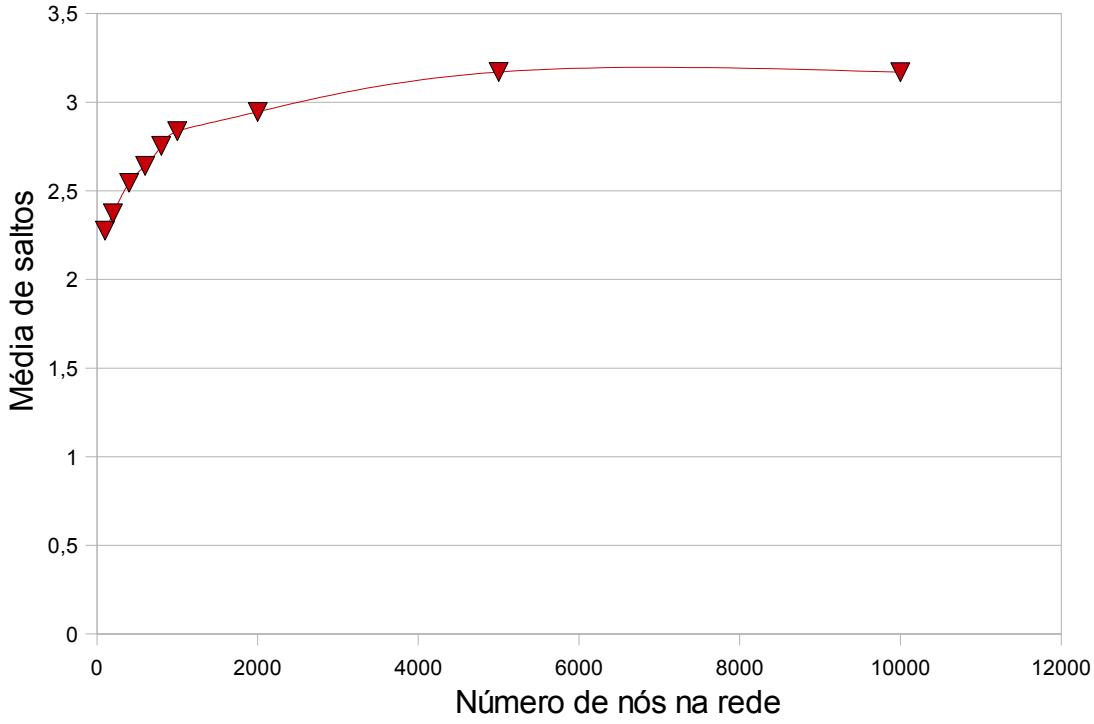

Figura 4.6. Influência do tamanho da rede no método básico

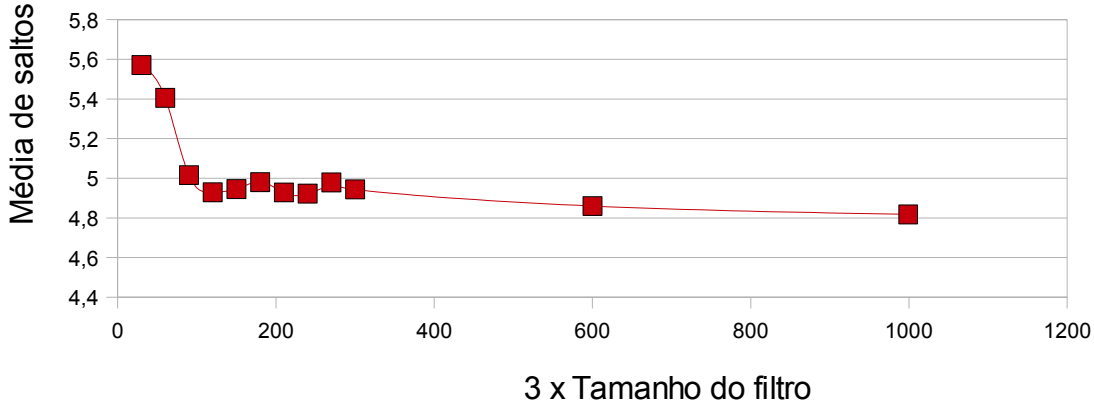

Figura 4.7. Influência do tamanho de filtro no método multi-estágios

5. PLATAFORMA CoppeerCAS

Um dos resultados centrais da presente pesquisa é uma implementação de uma agência da Arquitetura Coppeer na forma de um ambiente de tempo de execução para aplicações ponto-a-ponto denominado CoppeerCAS. O CoppeerCAS oferece suporte para a utilização de princípios de Sistemas Complexos Adaptativos (*Complex Adaptive Systems*) nas aplicações hospedadas. Por ter sido implementado em linguagem de programação Java, o CoppeerCAS pode ser usado em uma variedade de plataformas computacionais.

No CoppeerCAS, conexões entre células são bidirecionais e significam que cada uma das células envolvidas mantém um objeto procurador (proxy), que permite a realização de operações remotas sobre a outra célula. Para receber as requisições dos procuradores, cada célula mantém um objeto adaptador. Entretanto, esta estrutura é transparente para aplicação, que pode realizar operações sobre células remotas utilizando a mesma interface das células locais.

Um cliente de uma célula pode ser um agente local ou remoto de uma aplicação. Um mecanismo de *leasing* é empregado para controlar por quanto tempo uma célula irá atender uma requisição de um cliente. Quando um cliente requisita o armazenamento de uma entrada em uma célula, ele especifica uma duração pretendida para o serviço. Então, a célula alvo utiliza objetos da classe LeaseStrategy específicos da aplicação para determinar a duração efetiva do serviço.

Desenvolvedores de aplicações podem criar entradas com as regras de propagação que desejarem por meio da implementação de um conjunto de métodos do ciclo de propagação.

Para determinar o comportamento dos agentes, os desenvolvedores de aplicações devem criar objetos de comportamento e então passar o objeto para a agência durante a criação do agente. Os objetos de comportamento devem conter métodos que serão invocados pela plataforma para tratar eventos relevantes tais como criação do agente, movimentação ou operações sobre a célula.

Uma agência é composta por um gerente de agentes, uma coleção de células e um gerente de conexões. O gerente de agentes é o componente encarregado da criação,

da execução e da movimentação dos agentes. A coleção de células gerencia a criação das células e o controle do acesso das aplicações às mesmas. O gerente de conexões manipula as ligações entre as células e outros objetos necessários para efetuar trocas de dados entre agências.

O conteúdo das células é armazenado em tabelas de espalhamento (*hash tables)* com vistas à obtenção de bom desempenho. O CoppeerCAS é estruturado segundo uma arquitetura *Microkernel* (SPRING, 2006), que facilita a adição de módulos ao sistema em tempo de desenvolvimento ou execução. A figura 5.1 ilustra a visão geral dos componentes do CoppeerCAS.

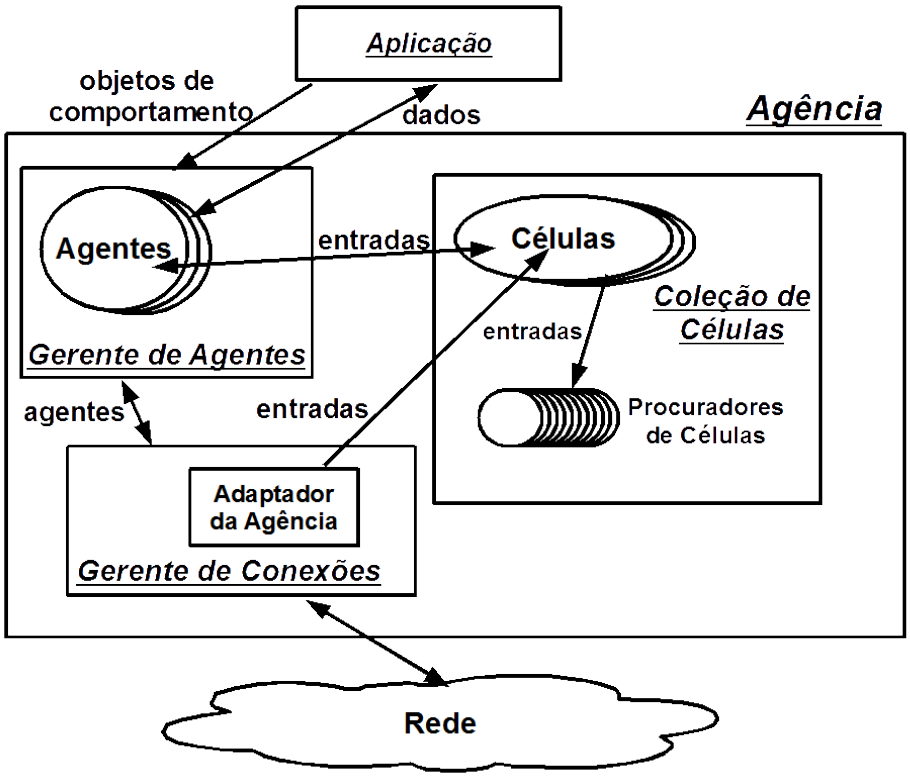

Figura 5.1 – Componentes do CoppeerCAS

5.1 Programação de aplicações no CoppeerCAS

O CoppeerCAS destina-se à execução de aplicações orientadas-a-objeto codificadas em linguagem Java. O objeto responsável pela linha de execução principal

da aplicação deve implementar a interface CoppeerApplication. A plataforma passa para esse objeto uma referência para a interface Agency, cujas funções principais são o fornecimento de referências para células locais e a criação de agentes Coppeer. As aplicações criadas pelos usuários precisam ser registradas em um arquivo de configuração, a fim de que o CoppeerCAS as instancie e crie itens de menu correspondentes em sua janela principal para que os usuários possam invocá-las.

A interface Cell especifica as operações que as células oferecem a seus clientes. Um cliente de uma célula pode ser um objeto de comportamento de agente ou qualquer outro objeto de uma aplicação. A célula é o mecanismo oferecido pela plataforma para comunicação entre clientes de uma mesma aplicação ou de aplicações diferentes. Um cliente pode ler entradas depositadas por qualquer outro cliente na mesma célula ou propagadas de células vizinhas. Para receber notificações de eventos de escrita em uma célula, um cliente deve implementar a interface CellEventHandler, sendo que os agentes Coppeer a implementam por padrão. Cada célula pertence a um ambiente e é hospedada por uma agência. Portanto, uma célula pode ser identificada dentro de uma agência pelo identificador de seu ambiente. Se um cliente e a agência que contém a célula estão no mesmo espaço de endereços, ou seja, são executados na mesma máquina virtual Java, então o cliente realiza operações diretamente na célula real. Caso contrário, o mecanismo transparente de procuração já citado é utilizado. Esse mecanismo de comunicação remota será melhor detalhado na seção 5.2.

As entradas que uma aplicação deposita em uma célula devem implementar a interface Entry. Os métodos especificados nessa interface são invocados pela plataforma no momento em que a entrada é depositada na célula para controlar a propagação da mesma para as células vizinhas ou para compará-la com assinaturas previamente realizadas e modelos especificados em operações de leitura. Portanto, além de especificar campos para conter os dados específicos da aplicação, o desenvolvedor deve implementar os métodos da interface Entry de forma a obter os comportamentos de propagação e notificação e recuperação desejados.

A lógica de uma aplicação pode ser total ou parcialmente embutida em agentes do Coppeer. Para criar um agente, a aplicação deve instanciar um objeto de comportamento pertencente a alguma subclasse da classe abstrata AbstractBehavior, e

passá-lo para o controle da plataforma no momento da criação do agente. O objeto especializado deve implementar adequadamente os métodos abstratos que são invocados pela plataforma como conseqüência de eventos relevantes como a criação do agente, o recebimento de notificações referentes a assinaturas, e a movimentação para outras agências. A classe AbstractBehavior oferece métodos para que o código especializado dos comportamentos tenha acesso à interface da célula que abriga o agente e solicite movimentações.

Os desenvolvedores de aplicação podem utilizar recursos variados da linguagem Java para implementar comportamentos correspondentes a micro-agentes, agentes adaptadores e agentes diretórios. A implementação de um agente fachada, por sua vez, pode ser realizada mediante a utilização de um comportamento que implemente a interface CoppeerWebService. Um comportamento que implemente essa interface pode ser registrado na agência, tornando-se acessível para programas externos via protocolo HTTP.

As seções 5.1.1 a 5.1.7 relacionam os principais métodos das interfaces e classes acima referidas.

5.1.1. Métodos da Interface CoppeerApplication

- **start(Agency *agency*)**: invocado por um item do menu de aplicações da janela principal da plataforma quando o usuário o seleciona.

5.1.2. Métodos da Interface Agency

- **Cell createCell(EnvironmentId *envId*, LeaseStrategy *leaseStrategy*):** cria e retorna uma referência para a célula local do ambiente identificado pelo parâmetro *envId* usando a estratégia de lease definido pelo usuário no parâmetro *leaseStrategy;*
- **Cell getCell(EnvironmentId *envId*):** retorna uma referência para a célula local do ambiente identificado pelo parâmetro *envId*. Caso a célula ainda não exista, cria a mesma utilizando uma estratégia de *leasing* padrão;

- **CellEventListener createCellEventListener(CellEventHandler *handler*):** cria e retorna uma referência para um objeto ouvinte para ser usado na criação de assinaturas em células, especificando que os eventos de interesse serão encaminhados para o objeto especificado no parâmetro *handler.*
- **void createAgent(Behavior *behavior*, EnvironmentId *envId*, Object *init*):** cria um agente Coppeer na célula local do ambiente identificado pelo parâmetro *envId,* com o comportamento especificado pelo parâmetro *behavior*, e com os dados de inicialização contidos no parâmetro *init.*

5.1.3. Métodos da Interface Cell

- **Entry readIfExists(Entry *tmpl*, Transaction *txn*, long *timeout*):** lê de uma célula uma entrada existente que seja compatível com a entrada padrão fornecida no parâmetro *tmpl*;
- **Entry takeIfExists(Entry *tmpl*, Transaction *txn*, long *timeout*):**similar ao método readIfExists(...), mas remove da célula a entrada obtida;
- **EventRegistration notify(Entry *tmpl*, Transaction *txn*,CellEventListener *listener*, long *lease*, Serializable *handback*):** requisita a criação de uma assinatura vigente durante o tempo em milissegundos especificado no parâmetro *lease.* A assinatura provoca o envio de uma notificação ao objeto ouvinte especificado no parâmetro *listener* sempre que uma entrada compatível com a entrada padrão fornecida no parâmetro *tmpl* for escrita na célula. Uma cópia do objeto especificado pelo parâmetro *handback é incluída* no evento de notificação.
- **Cell connect(AgencyId *agencyId*):** cria uma conexão bidirecional entre a célula e uma outra célula do mesmo ambiente hospedada na agência especificada pelo parâmetro *agencyId*;
- **Collection getNeighbors( ):** retorna uma coleção de objetos procuradores para as células vizinhas;
- **Cell getNeighbor(AgencyId *agencyId*):** retorna um procurador para a célula vizinha hospedada na agência especificada pelo parâmetro *agencyId*, caso ela

exista.

5.1.4. Métodos da Interface CellEventHandler

- **handleCellEvent(CellEvent *ev*):** invocado por um ouvinte de célula quando uma entrada compatível com um modelo fornecido em uma assinatura criada pelo método Cell.notify(...) é escrita na célula. O parâmetro *ev* contém dados de controle, tais como um número seqüencial para o evento e o uma cópia do objeto *handback* fornecido durante a criação da assinatura.

5.1.5. Métodos da Interface Entry

- **boolean matches(Entry *tmpl*):** invocado por células após uma escrita ou durante o processamento de uma leitura. É utilizada para determinar se a entrada combina com os modelos contidos nas assinaturas realizadas na célula. Os modelos são passados para o método no parâmetro *tmpl*;
- **boolean beforePropagation( ):** invocado por células após a escrita da entrada. A entrada é entregue ao propagador da célula somente se o valor de retorno for verdadeiro (*true*);
- **Entry generatePropagation(Cell *neighbor*):** invocado por propagadores uma vez para cada vizinho da célula associada. Uma referência para um objeto procurador do vizinho é passada no parâmetro *neighbor* e o valor de retorno é escrito pelo propagador neste mesmo procurador;
- **long getPropagationLease(Cell *neighbor*):** invocado por propagadores uma vez para cada vizinho da célula associada. Uma referência para um procurador do vizinho é passada no parâmetro *neighbor* e o valor de retorno é utilizado como duração de lease solicitada pelo propagador à célula vizinha;
- **boolean afterPropagation( ):** invocada por propagadores para determinar se o ciclo de propagação deve continuar. O ciclo continua somente se o valor retornado for verdadeiro (*true*).

5.1.6. Métodos da Classe AbstractBehavior

- **Cell getCell( )**: retorna uma referência para a célula corrente do agente associado;
- **CellEventListener getListener( ):** retorna uma referência para o objeto ouvinte que deve ser utilizado em invocações do método Cell.notify(...);
- **void move(AgencyId *agencyId*)**: transfere o agente associado para um célula do mesmo ambiente hospedada na agência especificada pelo parâmetro *agencyId*, caso ela exista.
- **void registerAs(String serviceName):** registra o serviço oferecido pelo comportamento, tornando-o acessível para programas externos via protocolo HTPP. Após o registro, o serviço pode ser invocado por clientes HTTP, sendo identificado pela URI (*Uniform Resource Identifier*), da agência concatenada ao nome contido no parâmetro *serviceName*;
- **onAgentCreation(Object *init*):** invocado pelo agente associado logo após sua criação. Os dados de inicialização são passados por meio do parâmetro *init*;
- **onArriving( ):** invocado pelo agente associado na agência de destino, imediatamente após uma transferência;
- **onLeaving( ):**invocado pelo agente associado na agência de origem, antes de uma transferência ser realizada;
- **behave( ):** invocado pelo agente associado após a invocação dos métodos onAgentCreation( ) e onArriving( );
- **handleCellEvent(CellEvent *ev*):** invocado pelo agente associado quando uma entrada compatível com um modelo fornecido em uma assinatura criada pelo método Cell.notify(...) é escrita na célula. O parâmetro *ev* contém dados de controle, tais como um número seqüencial para o evento e o uma cópia do objeto *handback* fornecido durante a criação da assinatura.

5.1.7. Métodos da Interface CoppeerWebService

- **ServiceResponse serve(String uri, String method, Properties header,**

Properties parms): invocado pela agência quando o serviço registrado correspondente ao comportamento que implementa a interface é solicitado por um cliente HTTP. O método deve devolver o resultado adequado em um objeto da classe ServiceResponse.

5.2. Componentes do CoppeerCAS

As seções 5.2.1 a 5.2.6 apresentam as características fundamentais de implementação dos componentes do CoppeerCAS.

5.2.1. Browser

O Browser é o componente do CoppeerCas responsável pela interface com o usuário, que é independente das interfaces das aplicações. Suas classe principais são SwingBrowserImpl, BrowserMainWindow e StartApplicationMenuItem, conforme o mostrado na figura 5.2 . Considerando um padrão de projeto MVC, SwingBrowserImpl é uma classe controladora, empregada para observar eventos de interesse no modelo representado pelo componente Agency, provocar as atualizações correspondentes na visão BrowserMainWindow, receber desta última entradas do usuário e repassar essas entradas ao componente Agency. Na versão implementada, os principais eventos e entradas tratados por SwingBrowserImpl são relacionados às conexões e desconexões com outros nós em cada ambiente.

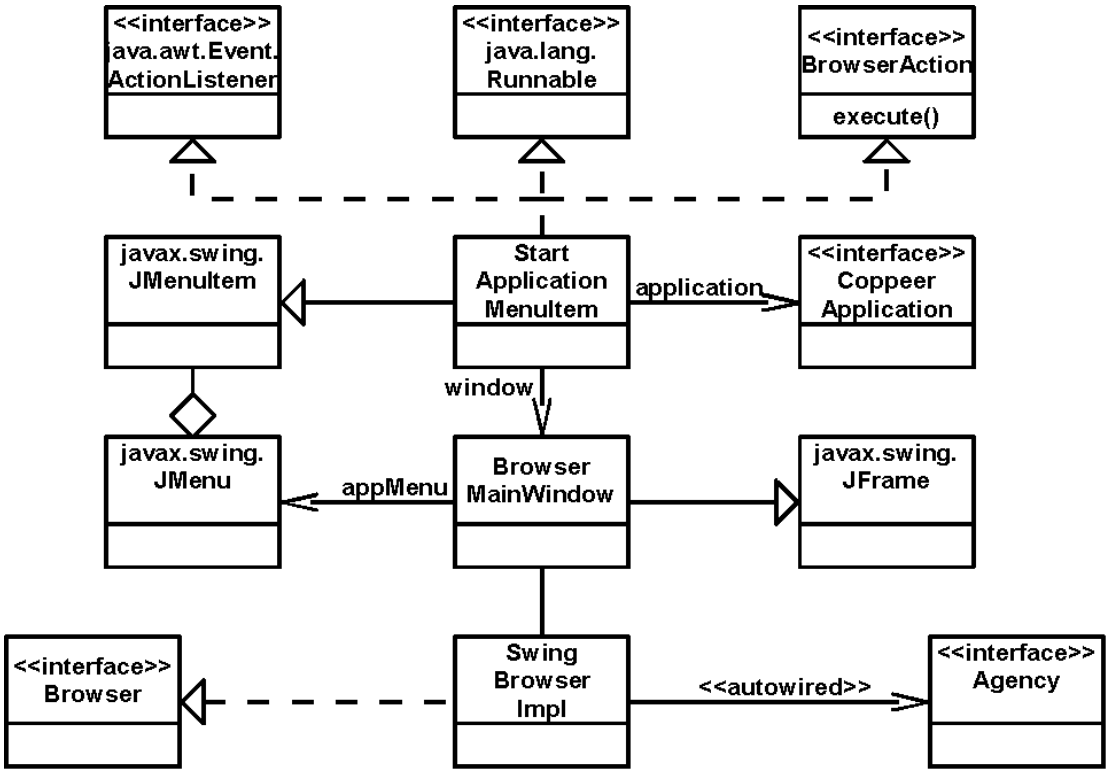

Figura 5.2 – Browser

Outra função básica de SwingBrowserImpl é construir em BrowserMainWindow um menu das aplicações disponíveis para o usuário. Cada item desse menu é controlado por um objeto da classe StartApplicationMenuItem que, em conformidade com o padrão de projeto Command, encapsula uma referência para a aplicação correspondente. Os objetos responsáveis por cada aplicação são instanciados pelo Kernel e entregues ao Browser durante a inicialização do CoppeerCAS. Quando um usuário invoca uma aplicação, o objeto StartApplicationMenuItem apenas invoca o método CoppeerApplication.start(Agency) correspondente em uma *thread* independente do CoppeerCAS, passando como parâmetro uma referência para a agência, de forma que a aplicação possa requisitar serviços da plataforma Coppeer.

5.2.2. Agency

O componente Agency, cuja estrutura é mostrada na figura 5.3, centraliza as funções principais do CoppeerCAS. Suas classes principais são

RMIConnectionManagerImpl, SimpleMultiThreadAgentManagerImpl e InMemoryCellPoolImpl que implementam respectivamente o gerente de conexões, o gerente de agentes e a Coleção de células. O componente possui ainda um objeto da classe java.util.concurrent.ScheduledExecutorService que gerencia a utilização das *threads* necessárias para a execução das aplicações.

O componente Agency é estruturado segundo os padrões de projeto Facade e Mediator, de forma que uma classe denominada AgencyMediator é responsável por implementar a interface com componentes externos e intermediar os serviços que os componentes internos requisitam uns dos outros.

5.2.3. AgentManager

Para atender a uma solicitação de criação de agente, o AgentManager cria um objeto da classe AgentBody, que encapsula o objeto de comportamento fornecido na solicitação. Em seguida, uma *thread* é requisitada ao componente Agency para executar os métodos de inicialização do objeto de comportamento, ou seja, *onAgentCreation(...)* e *behave()*.

De forma similar, quando um agente móvel oriundo de outra agência é recebido, o AgentManager é responsável por criar um novo objeto AgentBody para encapsular o comportamento do agente e requisitar uma *thread* para executar os métodos de *onArriving(...)* e *behave()*.

5.2.4. ConnectionManager

O ConnectionManager é o componente responsável pelo estabelecimento de conexões entre células, pela realização de assinaturas em células remotas, pela transferência de agentes e pela execução do serviço HTTP na agência. Este último permite o acesso de programas externos a agentes Coppeer e a troca de arquivos entre agências.

O componente mantém um procurador RMI (SUN, 2004) para sua própria agência. Quando uma conexão com uma célula de outra agência deve ser realizada, o

ConnectionManager encarrega-se de criar um procurador da célula local utilizando um objeto da classe RMICellProxyImpl, que encapsula o procurador RMI da agência. Em seguida, o ConnectionManager obtém um procurador RMI da agência remota utilizando o serviço de nomeação RMI. Finalmente, o componente invoca o método *connect(...)* do procurador RMI da agência remota, que envia o RMICellProxyImpl criado localmente e obtem o procurador correspondente da célula remota. As estruturas de conexão contém ainda objetos propagadores, da classe RMIPropagatorImpl, que também são criados pelo ConnectionManager. A função desses objetos é descrita na seção 5.2.5 .

A realização de uma assinatura em uma célula remota envolve a criação de um procurador de ouvinte, da classe RMICellEventListenerProxyImpl, que também encapsula o procurador RMI da agência. Mais detalhes sobre procuradores de ouvintes são fornecidos na seção 5.2.6 .

Para realizar uma movimentação de agente, o componente ConnectionManager obtém o procurador RMI da agência de destino utilizando o serviço de nomeação RMI e invoca o método *sendAgent(...)* desse procurador, a fim enviar o objeto de comportamento do agente.

O serviço HTTP é implementado pela classe CoppeerWebServer, que consiste em uma extensão da classe NanoHTTPD, implementada por terceiros. O servidor HTTP primeiramente procura interpretar as URI recebidas como uma solicitação a um serviço registrado por um agente. O registro de um serviço consiste no armazenamento de uma referência para o agente responsável em uma tabela de espalhamento indexada pelo nome do serviço. Para responder a uma solicitação de URI, o servidor HTTP tenta localizar na tabela a referência para o agente correspondente e, se for bem sucedido, invoca o método *serve(...)* do mesmo. O agente constrói a resposta e o servidor HTTP despacha-a para o cliente. Caso nenhum serviço correspondente a uma URI seja localizado na tabela, o servidor HTTP comporta-se de forma típica, tentando localizar o arquivo correspondente na agência e enviá-lo ao cliente. O serviço HTTP oferece ainda aos agentes locais a funcionalidade de criação de referências para arquivos, na forma de objetos da classe FileRef. Um objeto FileRef encapsula uma URI para um dado arquivo da agência em que é criado, e pode ser incluído em entradas para ser escrito em células e trasnmitido entre agências. Um agente em uma agência remota pode posteriormente

invocar o método *saveAs(...)* de um FileRef, provocando uma requisição do arquivo correspondente ao servidor HTTP da agência que hospeda o arquivo e a gravação de uma cópia na agência do requisitante.

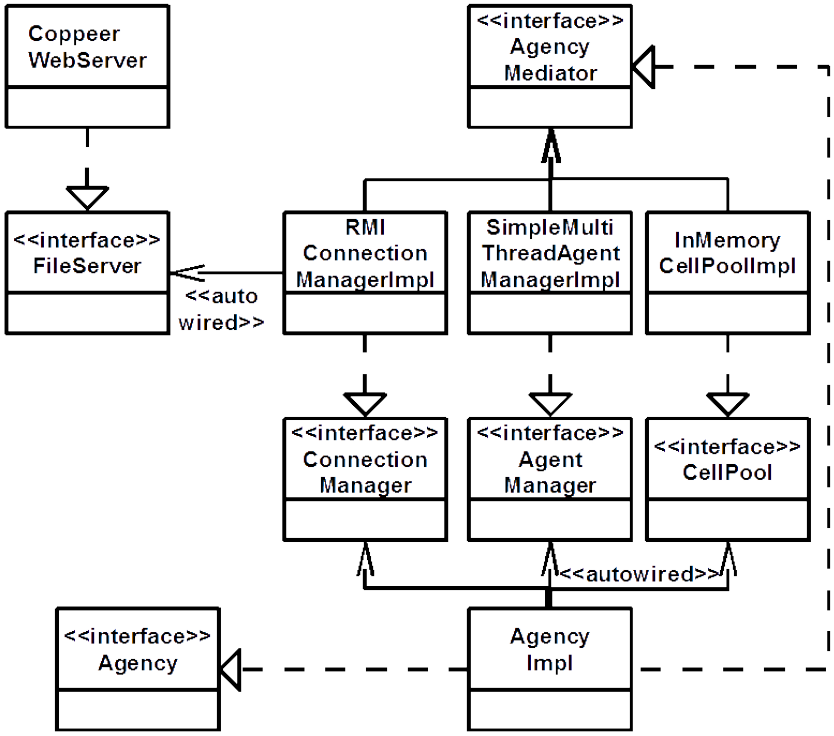

Figura 5.3- Agency

5.2.5 CellPool

O componente CellPool, cuja estrutura é mostrada na figura 5.4, é uma coleção de células implementada basicamente como uma tabela de espalhamento, na qual objetos que implementam as células são inseridos tendo como chave o identificador do ambiente a que pertencem. Cada célula é implementada por um objeto da classe InMemoryCell, que especializa a classe AbstractCell.

AbstractCell encapsula as estruturas de conexão de uma célula com suas vizinhas. Ela é composta por uma tabela de espalhamento, que contém procuradores

para as células vizinhas indexados pelo identificador da agência correspondente, e por um objeto propagador.

Os procuradores implementam a classe Cell, em conformidade ao padrão de projeto Proxy, a fim de poderem ser utilizados transparentemente em operações sobre células remotas. Cada procurador corresponde a um objeto da classe RMICellProxyImpl, que por sua vez encapsula um procurador RMI da agência que contém a célula vizinha. Toda requisição feita a um procurador de célula é convertida em uma invocação remota de um método do objeto RMIAgencyAdapter no nó remoto correspondente, que por sua vez realiza a operação adequada sobre a célula real.

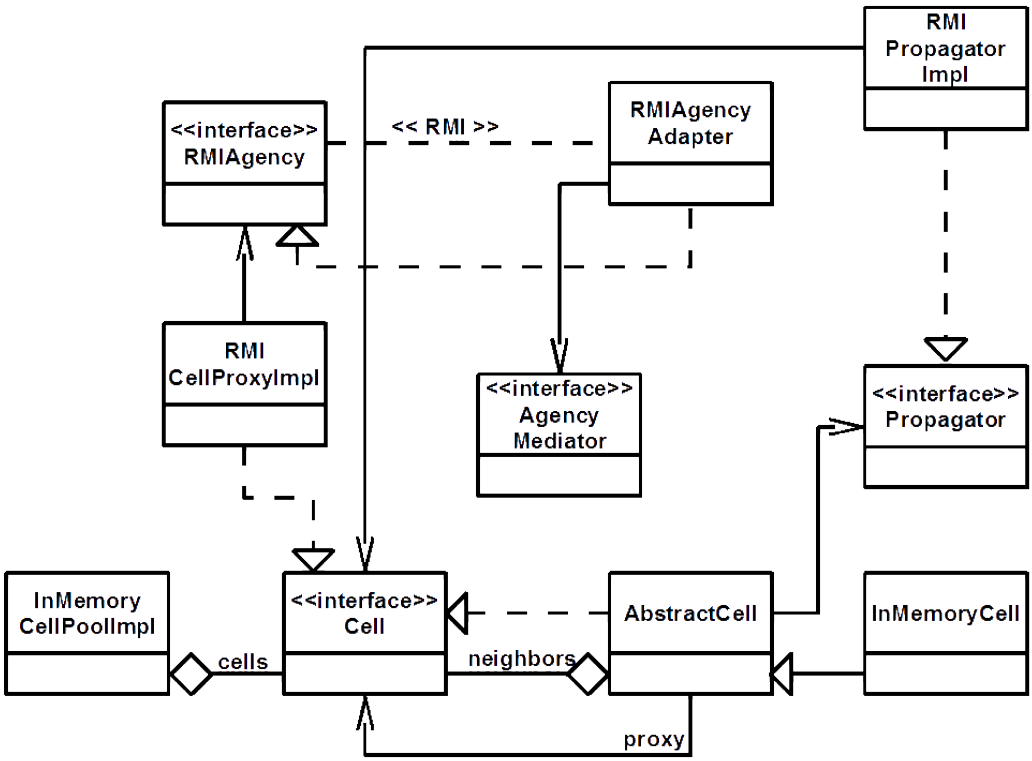

Figura 5.4 – InMemoryCellPoolImpl

Propagadores correspondem a objetos da classe RMIPropagatorImpl e têm a responsabilidade de invocar os métodos de controle de propagação de cada entrada escrita na célula. De acordo com os resultados retornados por esse métodos, o

propagador utiliza os procuradores das células vizinhas para realizar operações de escrita remotas.

A classe InMemoryCell, mostrada na figura 5.5, é uma simplificação da implementação proposta por MIGLIARDI et al. (2000), que insere cada entrada escrita na célula em duas tabelas de espalhamento independentes, denominadas entriesByClass e entriesByField. A tabela entriesByClass utiliza como chaves as classes das entradas já escritas na célula e mantém associada a cada chave a lista de entradas cuja classe corresponde à referida chave. Sua função é permitir que uma solicitação de leitura contendo um modelo de conteúdo nulo busque uma entrada pertencente à classe do modelo fornecido.

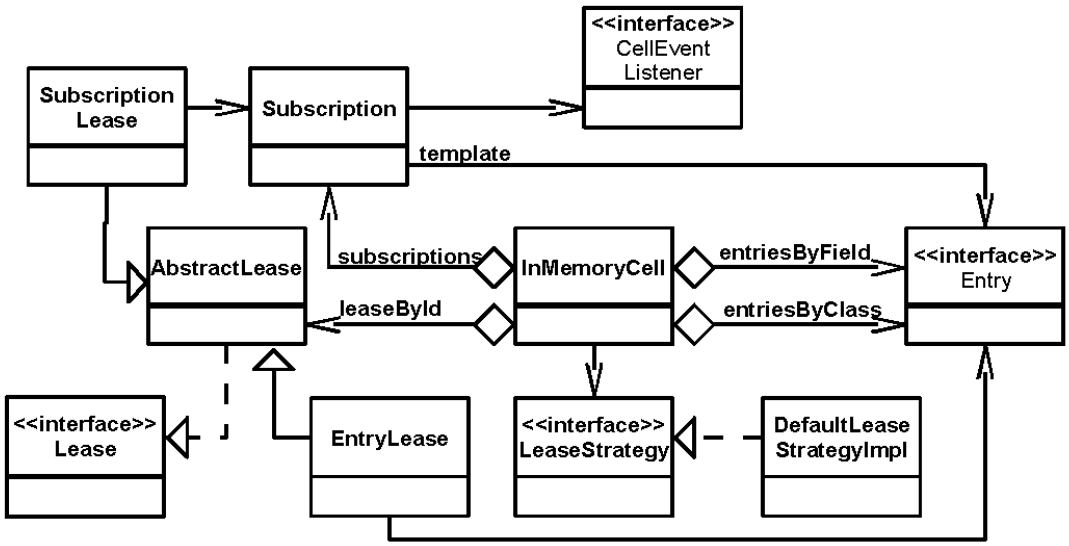

Figura 5.5 – InMemoryCell

A tabela entriesByField, por sua vez, armazena múltiplas vezes cada entrada escrita na célula, utilizando cada um dos campos não nulos da entrada como chave. Conseqüentemente, a cada chave presente na tabela é associada uma lista com todas

entradas que contém algum campo com mesmo nome e valor da chave. Quando uma solicitação de leitura contendo um modelo não nulo é realizada, as listas cujas chaves correspondem a cada um dos campos não nulos do modelo são recuperadas, caso existam. Em seguida, a menor lista é percorrida seqüencialmente e a primeira entrada compatível com o modelo, de acordo com as regras expressas no método *Entry.matches(Entry)*, é retornada como resultado.

Cada célula mantém também uma lista de objetos da classe Subscription, responsáveis pelos dados das assinaturas feitas na célula. Cada objeto Subscription contém o modelo fornecido durante a realização da assinatura e uma referência para o ouvinte interessado. Quando uma entrada é escrita na célula, a lista de assinaturas é percorrida seqüencialmente, e sempre que a entrada é compatível com o modelo, um evento é gerado e enviado para o ouvinte.

Além disso, para cada entrada escrita e para cada assinatura feita, a célula mantém objetos das classes EntryLease e SubscriptionLease, respectivamente. A função desses objetos é armazenar o tempo de expiração da entrada ou da assinatura e removê-las da célula caso esse tempo já tenha sido ultrapassado no momento de uma das verificações que são feitas periodicamente pela plataforma. Os tempos de expiração são calculados em função do tempo especificado pelo cliente que requisitou o serviço correspondente e das regras estabelecidas no objeto da classe LeaseStrategy, fornecido no momento da criação da célula.

5.2.6 CellEventListener

Os ouvintes das assinaturas das células podem ser objetos das classes AgentBody ou RMICellEventListenerProxy, conforme o mostrado na figura 5.6 . Objetos AgentBody correspondem às assinaturas feitas por agentes Coppeer localizados na mesma agência da célula ao passo que objetos RMICellEventListenerProxy correspondem às assinaturas feitas por agentes remotos. Ambos implementam a interface CellEventListener, também em conformidade com o padrão Proxy, a fim de que as assinaturas os referenciem indistintamente. Mas, enquanto AgentBody implementa um ouvinte real, RMICellEventListenerProxyImpl é um procurador de

ouvinte que encapsula o procurador RMI da agência que hospeda o agente remoto. Toda notificação feita a um procurador de ouvinte é convertida em uma invocação remota de do método *notify(...)* do objeto RMIAgencyAdapter no nó remoto, que por sua vez notifica o AgentBody do agente remoto. Um AgentBody, ao receber um evento, o envia para um CellEventHandler, ou seja, o objeto Behavior associado ao agente.

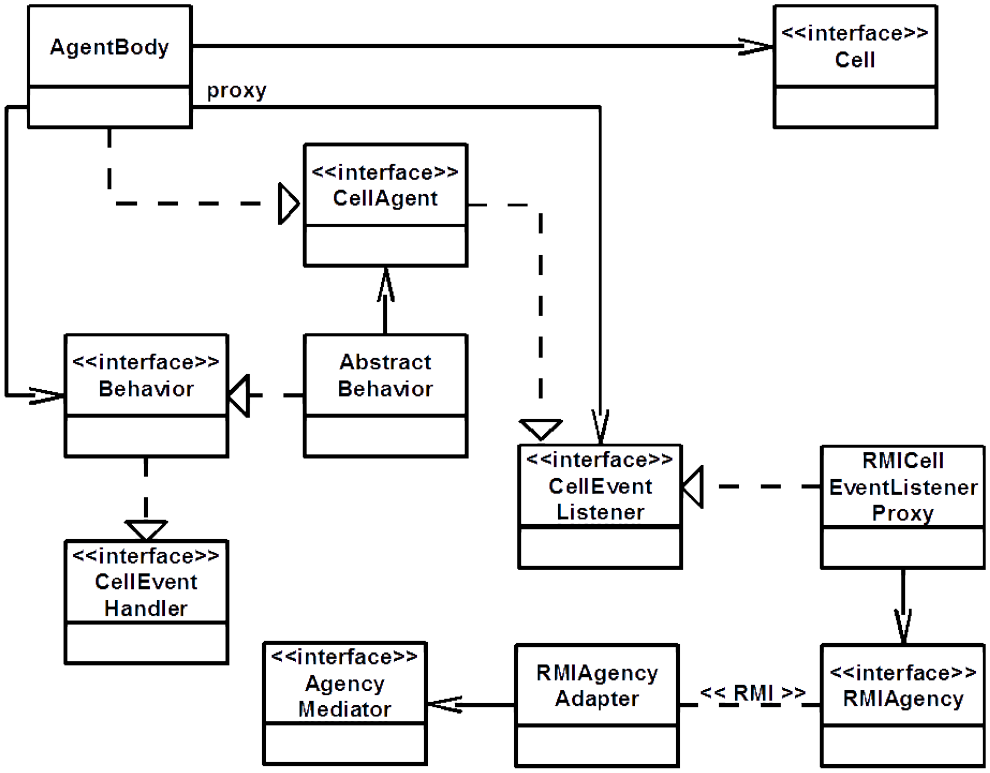

Figura 5.6 – CellEventListener

6. APLICAÇÕES CoppeerCAS

O CoppeerCAS oferece abstrações de alto nível para a implementação de comunicação assíncrona e execução de múltiplas *threads*, na forma dos espaços compartilhados e dos agentes respectivamente. Tais abstrações reduzem o esforço de programação de aplicações distribuídas, o que favoreceu a implementação de uma série de ferramentas, em sua maioria colaborativas, pelo grupo de pesquisas do Laboratório de Bancos de Dados da COPPE/UFRJ e outros pesquisadores externos, tendo como base o CoppeerCAS. Entre essas ferramentas destacam-se:

- ***FoxPeer*** (VIVACQUA et al., 2009) – é uma aplicação de navegação colaborativa da *Web*, composta por um *plug in* do navegador Mozilla Firefox e por um agente fachada do Coppeer. Caso um usuário avalie uma página *Web* carregada pelo navegador, o *plug in* envia o endereço da página e a avaliação do usuário para o agente. Este então carrega a página novamente, constrói um sumário composto pelas palavras mais freqüentes e armazena o sumário e a avalição em uma base local. Quando um usuário indica que deseja encontrar páginas semelhantes à que está carregada no navegador, o *plug in* também envia o endereço para o agente, que neste caso, após carregá-la e construir o sumário, pesquisa sua base local e envia o sumário para que agentes de nós vizinhos também pesquisem suas respectivas bases, formadas por sumário de páginas visitadas por outros usuários da comunidade. Os resultados são coletados pelo agente, e uma página de resposta, contendo endereços e médias das avaliações, é enviada ao navegador para ser exibida ao usuário que iniciou o processo.
- ***Crawler Ponto-a-ponto*** (MAYWORM, 2007) – é um *Web Crawler* multiagentes, no qual as tarefas necessárias para o funcionamento da ferramenta são executadas por diferentes agentes Coppeer. Os agentes são distribuídos por diversas agências, para fim de balanceamento de carga e para aumentar a eficiência da ferramenta, ao atribuir cada página que deve ser examinada à agência que possa obter o menor tempo de resposta para carregá-la.
- ***Business Process Cooperative Editor*** **(BPCE – Editor Cooperativo de Processos de Negócios) –** destinado a apoiar o reuso, a avaliação e a evolução

cooperativa de processos de negócios de uma organização;

- ***Knowledge Chains Editor*** **(KCE – Editor de Cadeias de Conhecimento) –** destinado a facilitar processos de aprendizado mediante o gerenciamento e compartilhamento de estratégia pessoais de aprendizado representadas graficamente como cadeias de conhecimento(SOUZA et al., 2005b);
- ***Project Management Collaborative Editor*** **(PMCE – Editor Colaborativo de Gerenciamento de Projetos) –** destinado a facilitar a disseminação do conhecimento gerado e das experiências adquiridas durante a realização das atividades do projeto por seus respectivos responsáveis;
- **CUMBIA 2.0 –** é a evolução da aplicação para colaboração oportunística CUMBIA (SOUZA et al., 2005a), destinada à descoberta de similaridades entre interesses de usuários do sistema por meio da análise de informações coletadas em suas estações de trabalho;
- ***Collaborative Ontology Editor 2.0 (*****COE - Editor Colaborativo de Ontologias) –** destinado à criação, edição e compartilhamento de ontologias representadas como árvores hiperbólicas;
- **OntoEmerge** – protótipo de ferramenta de integração de ontologias que utiliza o CoppeerCAS para realizar transferências de arquivos e gerenciamento dos nós em uma rede ponto-a-ponto (SOUZA Jr. et al, 2010).

Nas seções 6.1 e 6.2, serão apresentadas duas propostas de aplicações colaborativas complexas, que incorporam princípios de sistemas emergentes em suas próprias concepções, sendo, portanto particularmente interessantes para demonstrar possibilidades do CoppeerCAS.

6.1. Infraestrutura de Apoio ao Projeto Emergente (MIRANDA et al., 2006)

6.1.1 Modelo de Projeto Emergente

Projeto emergente é o processo de tradução de um conjunto de requisitos de sistema interrelacionados para um conjunto de modelos de elementos de sistema

interrelacionados, realizado por um grande número de agentes sem emprego de comunicação direta, coordenação central ou conhecimento global.

Seja R o conjunto de todos os requisitos possíveis expressos em uma dada linguagem de requisitos e $M \subset R$ o conjunto de todos os elementos de modelo expressos em uma dada linguagem de modelagem. No contexto de nossa proposta de modelo de projeto emergente, um projeto é representado pela tupla $D = (S , I)$ onde:

- $S \subset R$ é um conjunto de especificações;
- I é um conjunto de arestas direcionadas, da forma ($x \in S\text{-}M , y \in S$) ou ($x \in S \cap M$, $y \in S \cap M$), representando que a especificação x cita a especificação y.

Da definição de R e M nota-se que a linguagem de modelagem é uma especialização da linguagem de requisitos. Portanto, a primeira de sentença declara que combinações de requisitos e elementos de modelo podem representar um projeto, abrangendo desde a especificação de requisitos inicial até o projeto finalizado, composto somente por elementos de modelo. A segunda sentença, por sua vez, declara que uma especificação de requisitos pode citar especificações de elemento de modelo, mas uma especificação de elemento de modelo só pode citar outro elemento de modelo.

Os agentes envolvidos, que no caso são projetistas humanos e não agentes de *software*, devem executar uma seqüência de operações sobre especificações iniciais de requisitos para gerar projetos finalizados. Essas operações devem gerar apenas projetos sintática e semanticamente corretos. A correção sintática é assegurada se não forem criadas arestas partindo de elementos de modelo em direção a especificações de requisitos. A correção semântica, por sua vez, é garantida pela seguinte regra: se uma aresta partindo de uma especificação x em direção a uma especificação y é substituída por uma aresta partindo de x em direção a uma especificação z, então z deve ser uma *extensão* de y, ou seja, o significado original de x deve ser preservado na operação. Uma definição mais detalhada das operações de projeto permitidas em nossa proposta é apresentada em seguida.

6.1.2. Atividades de Projeto Emergente

Nesta seção serão definidas as atividades de projeto emergente, ou seja, a

operações que os agentes podem realizar sobre um projeto. Seja *A* um conjunto de agentes e $T = A^*$ o conjunto de todos os subconjuntos de *A*. A alocação do projeto é a função $t:S \rightarrow T$ que mapeia cada especificação de um projeto em uma equipe de agentes responsável pela especificação. Se uma equipe é responsável por uma especificação, ela pode realizar operações para particioná-la, mesclá-la, realocá-la e traduzi-la.

A operação de *particionamento* substitui uma especificação *s* por um conjunto *P* de especificações interrelacionadas do mesmo tipo (requisito ou elemento de modelo) de *s, e* todas as arestas envolvendo *s* por arestas envolvendo algum membro de *P* na mesma direção. A equipe torna-se responsável pelos membros de P.

A operação de *mesclagem* substitui um conjunto *P* de especificações interrelacionadas e do mesmo tipo por uma especificação *s* e todas as arestas envolvendo membros de *P* por arestas envolvendo *s* na mesma direção. A equipe torna-se responsável por *s*.

A operação de *realocação* passa a responsabilidade por uma especificação para outra equipe de agentes, modificando a alocação do projeto;

A operação de *tradução* substitui uma especificação de requisitos *r1* por um elemento de modelo *m1*, e todas as arestas envolvendo *r1* por arestas envolvendo *m1*. Além disso, devem ser observadas as duas situações a seguir:

- se uma especificação de requisitos *r2* é citada por *r1* e não estiver sendo traduzida em paralelo, a operação criará um elemento de modelo *i*(*r2*) para ser citado por *m1* no lugar de *r2*, e uma citação de *r2* para *i*(*r2*), determinando que sua tradução tem que ser uma extensão de *i*(*r2*). A equipe torna-se responsável por *m1* e *i*(*r2*). Quando *r2* for traduzida para um elemento de modelo *m2*, a aresta de *m1* para *i*(*r2*) deverá ser substituída por uma aresta de *m1* para *m2*;
- se *r2* é traduzida por outra equipe para um elemento de modelo *m2* paralelamente à criação de *i*(*r2*), então essa equipe - que não teria como conhecer i(r2) e elaborar m2 como sua extensão - deve, após a finalização de *i*(*r2*) e *m2*, criar um elemento de modelo adicional *a*(*m2*). O elemento *a*(*m2*) deve estender *i*(*r2*) e citar *m2*, para poder substituir *i*(*r2*) e desempenhar um papel de intermediador entre *m1* e *m2*.

Como as operações foram definidas em termos de equipes, os agentes precisam se comunicar uns com os outros para selecionar as operações coletivamente e para aplicar as operações selecionadas. Além disso, uma operação sobre uma especificação pode implicar a substituição ou inclusão de citações em outras especificações controladas por outras equipes. Conseqüentemente, a comunicação entre entre agentes de equipes diferentes também é necessária. Como nosso modelo de projeto emergente exclui a comunicação direta entre os agentes por definição, a alternativa restante é a comunicação estigmática: as informações sobre as operações tem que ser colocadas em uma infraestrutura de suporte de forma que possam ser recuperadas por qualquer agente interessado.

6.1.3. *Brainstorming* Emergente

Conforme visto na seção 6.1.2, o modelo de projeto emergente proposto demanda uma infraestrutura para implementação de comunicação indireta, ou seja, assíncrona, entre os agentes que participam do projeto. No caso da comunicação entre os agentes que participam de uma mesma equipe, faz-se também necessário um mecanismo que auxilie a concepção e escolha colaborativa da operação que deve ser aplicada em cada etapa do projeto. Como a proposta de projeto emergente não impõe limitação de tamanho às equipes, o mecanismo deve possibilitar a participação de um número potencialmente grande de agentes no processo de elaboração de cada operação. Para atingir essses objetivos, a proposta de projeto emergente inclui o mecanismo que denominamos *brainstorming* emergente.

O mecanismo de *brainstorming* emergente mantém a estação de trabalho de cada agente conectada a k vizinhos no máximo em uma rede sobreposta. O número k deve ser pequeno, de forma que os agentes recebam uma quantidade limitada de informações, para que seja respeitado o princípio da simplicidade de agentes apresentado no capítulo 3. No início do processo de *brainstorming*, o mecanismo fornece uma especificação do projeto para os agentes e permite que os mesmos registrem uma proposta de operação. Além disso, o mecanismo permite que cada agente leia as propostas de seus vizinhos, analise-as e atualize sua própria proposta com base

no conhecimento adquirido, ou endosse uma das propostas dos vizinhos. Após cada agente atualizar sua proposta diversas vezes, o resultado esperado é a emergência de algumas poucas boas propostas, consolidadas pela ação colaborativa dos integrantes da rede. Cada equipe deve possuir um projetista moderador, cuja estação de trabalho pode funcionar como um supernó para coordenar o processo de votação para seleção da melhor proposta e aplicação da operação correspondente.

6.1.4. Exemplo de Projeto Emergente

Nesta seção vamos apresentar um exemplo simplista, para fins ilustrativos, de um projeto emergente de sistemas de *software*, no qual a linguagem de requisitos é a linguagem natural e a linguagem de modelagem é um equivalente textual do modelo de classes da UML. Nesse caso, a linguagem de modelagem pode claramente ser considerada uma especialização da linguagem de requisitos.

A especificação de requisitos em linguagem natural é mostrada na tabela 6.1 e uma possível transformação em um projeto terminado é ilustrada na figura 6.1 . A especificação corresponde à descrição de um histórico de empréstimo bibliotecário e é inicialmente alocada a uma equipe denominada *equipeA*.

Depois que os agentes de *equipeA* concordam em executar uma partição e uma relocação, o projeto resultante é composto por uma *descrição de histórico,* uma *descrição de pessoa* e uma *descrição de livro*, alocadas respectivamente às equipes *equipeA*, *equipeB* e *equipeC*, conforme o mostrado na tabela 6.2. Nessa tabela e nas tabelas subseqüentes, citações são exibidas sublinhadas.

Tabela 6.1 - Especificação de requisitos inicial

Nome	***Equipe***	***Conteúdo***
Descrição de histórico de empréstimo bibliotecário	*equipeA*	"... o histórico registra todo livro que cada pessoa já retirou ... uma pessoa é descrita por seu nome, endereço,... um livro é descrito por seu título, autor, ..."

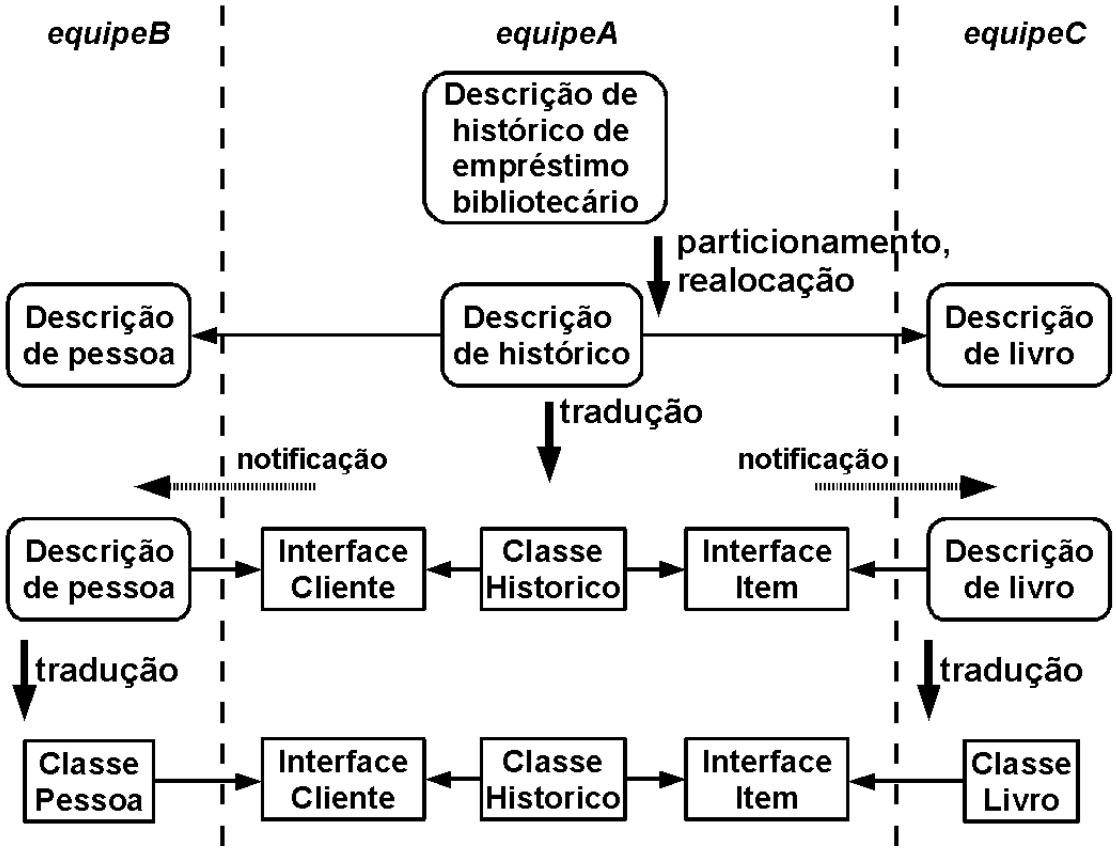

Figura 6.1 - Exemplos de operações

Tabela 6.2 - Resultado do particionamento e da realocação

Nome	***Equipe***	***Conteúdo***
Descrição de histórico	*equipeA*	"... o histórico registra todo livro que cada pessoa já retirou."
Descrição de pessoa	*equipeB*	"... uma pessoa é descrita por seu nome,endereço,..."
Descrição de livro	*equipeC*	"... um livro é descrito por seu título,autor ..."

No passo seguinte, os agentes de *equipeA* concordam em realizar uma tradução, substituindo a *descrição de histórico* por três elementos de modelo: *classe Historico*, *interface Cliente* e *interface Item*. As outras equipes são notificadas para adicionar citações em suas especificações. O resultado é mostrado na tabela 6.3.

Finalmente agentes de *equipeB* e *equipeC* concordam em executar traduções gerando os modelos de elemento *classe Pessoa* e *classe Livro*, respectivamente, mostrados na tabela 6.4.

.

Table 5.3. Resultado da tradução da descrição de histórico

Nome	***Equipe***	***Conteúdo***
Classe histórico	*TeamA*	Class Log{ Item logItem;Client logClient; ... }
Interface cliente	*equipeA*	Interface Cliente{...}
Interface item	*equipeA*	Interface Item {...}
Descrição de pessoa	*equipeB*	"... uma pessoa é descrita por seu nome,endereço ... precisa estender Cliente"
Descrição de livro	*equipeC*	"... um livro é descrito por seu título,autor ... precisa estender Item"

Tabela 6.4. resultado da tradução das descrições de pessoa e livro

Nome	***Equipe***	***Conteúdo***
Classe pessoa	*equipeB*	Class Pessoa implements Cliente{String nome; String endereco ... }
Classe livro	*equipeC*	Class Livro implements Item{String titulo; String autor ... }

6.1.5. Implementação no CoppeerCAS

Na implementação de uma infraestrutura para projeto emergente no CoppeerCAS, cada equipe deve ter seu próprio ambiente Coppeer para manipular as especificações que lhe forem atribuídas. Uma especificação pode ser armazenada como uma entrada em uma célula do ambiente e replicada em outras células por propagação para reduzir tempos de busca e aumentar a disponibilidade do sistema. Para permitir a rápida disseminação de informações dentro de uma equipe, o ambiente correspondente pode ser organizado como uma rede de mundo pequeno.

O mecanismo de *brainstorming* emergente, deve ser composto por agentes Coppeer responsáveis por oferecer para os agentes humanos (projetistas), interfaces de escrita e leitura de propostas de operações. O agente de escrita armazena cada proposta do projetista como uma entrada na célula local. O agente de leitura lê as propostas dos

nós vizinhos e as apresenta para o projetista quando requisitado. Agentes adicionais devem ser utilizados para implementar os processos de votação e aplicação da operação eleita.

Como uma operação pode implicar modificações em especificações localizadas em ambientes diferentes, um ambiente global deve abranger todas as equipes. Sempre que uma operação for executada sobre uma especificação, uma entrada descrevendo essa operação deve ser armazenada em uma célula do ambiente global. Então, o ambiente global deve propagar essa entrada a fim de que todas as equipes que tenham feito assinaturas para eventos na especificação sejam notificadas. A fim de garantir a correção do projeto, as equipes devem fazer assinaturas no ambiente global para eventos em especificações que citem ou sejam citadas pelas especificações por elas controladas.

A plataforma Coppeer oferece como mecanismo básico de comunicação a notificação de eventos de escrita para clientes que realizaram assinaturas na própria célula onde um evento ocorre. Entretanto, a ferramenta de projeto emergente precisa de um mecanismo que permita notificações de eventos em nível de ambiente, ou seja, que em resposta à escrita de uma entrada em uma célula, realize uma busca distribuída na rede a fim de que as assinaturas compatíveis feitas em outras células do ambiente sejam localizadas. Tal mecanismo pode ser implementado em sistemas ponto-a-ponto por meio de técnicas tais como as propostas de EUGSTER & GUERRAOUI (2002), COSTA et al. (2003), TERPSTRA et al. (2003) e GUPTA et al. (2004).

6.2. Infraestrutura para desenvolvimento incremental e colaborativo de Sistemas Distribuídos de Informações (MIRANDA et al., 2007)

O estado atual das tecnologias de rede facilita o desenvolvimento de sistema de computação distribuída para organizações complexas dispersas em grandes áreas geográficas. Entretanto, o projeto deste tipo de sistema pode ser dificultado por problemas tais como a excessiva complexidade de um modelo global para o sistema ou a necessidade de lidar com idiossincrasias das unidades organizacionais. Além disso, o acréscimo de novas características a um sistema em produção pode gerar procedimentos intrincados de reimplantação.

Como um exemplo, podemos citar o Exército Brasileiro, cujas unidades organizacionais são dispersas por todo o território nacional, que é composto por áreas urbanas, florestas, montanhas, regiões desérticas e vários outros ambientes. Conseqüentemente, apesar das estruturas comuns determinadas por regulamento, estas unidades apresentam particularidades diversas em termos de missão, tamanho, equipamentos, orçamento, cultura, pessoal e recursos computacionais.

De uma forma geral, o projeto desse tipo de sistema parece enquadrar-se em um modelo:

- colaborativo – o sistema deve ser distribuído em uma rede ponto-a-ponto de tal forma que equipes autônomas possam construir ou modificar suas partes;
- incremental – o sistema deve ser construído sob a hipótese de que seus requisitos são apenas parcialmente conhecidos e irão mudar com o tempo;
- dinâmico – o sistema deve ser construído de tal forma que a implementação de novos requisitos não exigirão a reimplantação do sistema ou a conversão de dados armazenados para novos modelos.

Nesta seção, é apresentada uma proposta de infraestrutura para projetos com as características acima relacionadas, baseada no uso de documentos semi-estruturados como modelo de dados, no particionamento de código do sistema em diversos agentes fracamente acoplados e no uso de ontologias para reconciliar diferenças existentes entre os vocabulários das unidades organizacionais.

6.2.1. Abordagem básica

O estilo arquitetural da *World Wide Web* (FIELDING, 2000) representa uma abordagem de projeto colaborativo, incremental e dinâmica bem sucedida. Em nossa proposta, aplicamos alguns aspectos desse estilo arquitetural e, partindo de suas características de sistema cliente-servidor hipermídia, acrescentamos novos elementos a fim de derivar uma abordagem apropriada para sistemas ponto-a-ponto de propósito geral. As principais características da abordagem resultante são as seguintes:

- entidades do mundo real são representadas primariamente por documentos independentes de linguagem de programação, indexáveis, categorizáveis,

legíveis para humanos e capazes de referenciar outros documentos;

- documentos possuem uma estrutura, que representa o estado de uma entidade, e por uma interface, que declara as operações que podem ser executadas sobre o documento;
- cada documento é hospedado por um nó autônomo do sistema, que pode armazená-lo persistentemente;
- a execução de operações sobre os documentos é realizada por unidades de código fracamente acopladas, que podem ser substituídas independentemente. Estas unidades podem modificar tanto a estrutura como a interface dos documentos.

6.2.2. Modelo de documentos

Em nossa proposta, entidades do mundo real são mapeadas a documentos textuais contendo um identificador local de documento, um conjunto possivelmente vazio de identicadores de classe de documento, um papel de documento, dados representando o estado da entidade e assinaturas de métodos representando as operações que podem ser executadas sobre o documento. A figura 6.2 mostra uma possível representação XML para um documento correspondente a uma venda.

O identificador local de documento é único no escopo do nó que hospeda o documento. Para fazer referência a um documento, o sistema emprega URI's compostas por um identificador de nó e um identificador local de documento.

Cada identificador de classe de documento refere-se a uma categoria do mundo real específica de aplicação que contém a entidade representada pelo documento. O propósito das classes de documento é organizá-los para apoiar a indexação e a busca dos documentos. Uma classe de documento não impõe uma estrutura ou uma interface aos documentos, uma vez que esses elementos são definidos no corpo de cada documento individual.

Os possíveis papéis de documento são: instância; modelo; espelho; mensagem; e ontologia. Um documento instância representa uma ocorrência individual de um entidade categorizável. Um documento modelo, por sua vez, é a base para a criação de

documentos instância com estruturas, interfaces e classes iniciais similares. Documentos espelho são cópias remotas de um documento que delegam a execução de métodos aos documentos originais e replicam qualquer atualização ocorrida nos mesmos, sendo úteis para apoiar o desenvolvimento de aplicações colaborativas. O papel mensagem designa documentos não-persistentes que são gerados como resultado de operações e podem ser passados entre nós do sistema. Finalmente, documentos do papel ontologia explicitam relações diretas entre classes de documentos e outros metadados do sistema. As informações dos documentos ontologia permitem a expansão de consultas escritas com um dado vocabulário para gerar consultas escritas com vocabulários mais antigos ou vocabulários de outros nós.

Cada assinatura de método em uma interface de documento é composta de um nome de método, uma lista de argumentos e um identificador de versão. Para invocar um método, uma aplicação precisa conhecer apenas uma referência ao documento alvo, um nome de método e uma lista de argumentos. O identificador de versão é utilizado internamente pelo sistema para direcionar a invocação a um agente apropriado. Um mesmo par (nome de método, lista de argumentos) pode ser associado a identificadores de versão diferentes em documentos distintos. Então, duas invocações de método idênticas sob a perspectiva da aplicação invocadora podem ser executadas por agentes diferentes se os alvos forem documentos distintos ou se o identificador de versão do documento for modificado entre as invocações. Desta forma o sistema dá suporte a um mecanismo de polimorfismo aos métodos dos documentos, semelhante ao encontrado nas linguagens de programação orientadas a objeto.

6.2.3. Implementação no CoppeerCAS

Na abordagem proposta, não há separação clara entre tempo de desenvolvimento e tempo de produção. Em vez disso, é prevista a execução simultânea de ações de desenvolvedores, que modificam as características da aplicação, e ações de usuários, que modificam o estado de um modelo sendo executado pela aplicação. Ações de desenvolvedores típicas são a criação e a atualização de documentos modelo e ontologia, ou a implementação de comportamentos de agentes responsáveis pela

execução de métodos presentes em interfaces de documentos. Por sua vez, ações de usuários típicas são a criação de documentos instância à partir de documentos modelo, a criação de documentos espelho à partir de documentos instância, invocações de métodos de documentos e a busca por documentos que satisfaçam algum critério especificado.

A implementação da proposta no CoppeerCAS, que denominamos Coppeer Documents, pode ser realizada na forma de um conjunto padrão de agentes composto por um agente de repositório, um agente de busca, um agente de interface de usuário e um agente de interface de desenvolvedor, conforme esquematizado na figura 6.3 . Desenvolvedores de aplicação devem criar agentes responsáveis por métodos estendendo a classe *AbstractMethodBehavior* para implementar comportamentos específicos de cada aplicação. Além disso, desenvolvedores mais avançados podem criar agentes para substituir os agentes padrão.

```
<doc id="s41154" role_type ="instance">
    <class id="Sale"/>
    <item cod = "bk82948238 qty="3"
        value="98.54"/>
    </item>
    <method name="getValue"
        version_id="getValue01"/>
    <method name="convertToCreditSale"
        version_id="convertToCreditSale01">
        <arg>number_of_instalments</arg>
    </method>
</doc>
```

Figura 6.2 - Exemplo de um documento de venda

Agente de repositório. o agente de repositório é responsável pelo armazenamento de documentos em um repositório persistente e pela recuperação destes documentos em respostas a requisições de outros agentes. Para executar estas funções, o agente pode manipular os tipos de entradas descritos a seguir:

- *DocumentCreationEntry* – transportam todos os dados necessários para a criação de um novo documento. Quando um agente de repositório recebe este tipo de entrada, ele cria o documento correspondente no repositório;

- *MethodInvocationEntry* – é composta por uma referência de documento, um nome de método e uma lista de parâmetros. Quando um agente de repositório recebe este tipo de entrada, ele recupera o documento apropriado do repositório, extrai o identificador de versão correspondente ao método invocado e escreve na célula local uma entrada do tipo *AgentInvocationEntry* contendo o documento, o nome do método, a lista de parâmetros e o identificador de versão;

- *RepositoryQueryEntry* – contem critérios de busca de documentos. Quando um agente de repositório recebe este tipo de entrada ele recupera do repositório referências para todos os documentos que satisfaçam os critérios de busca e escreve na célula local uma entrada do tipo *SearchResultEntry* contendo um documento mensagem composto por essas referências.

- *DocumentRequestEntry* – transporta uma referência para documento. Quando um agente de repositório recebe este tipo de entrada, ele recupera o documento apropriado e escreve na célula local uma entrada do tipo *DocumentEntry* contendo um documento mensagem que será manipulado pelo agente que requisitou o documento.

- *DocumentUpdateEntry* – contém dados necessários para atualizar a estrutura e a interface de um documento pré-existente. Quando um agente de repositório recebe este tipo de entrada ele recupera o documento apropriado, executa as alterações requeridas e armazena a nova versão do documento no repositório.

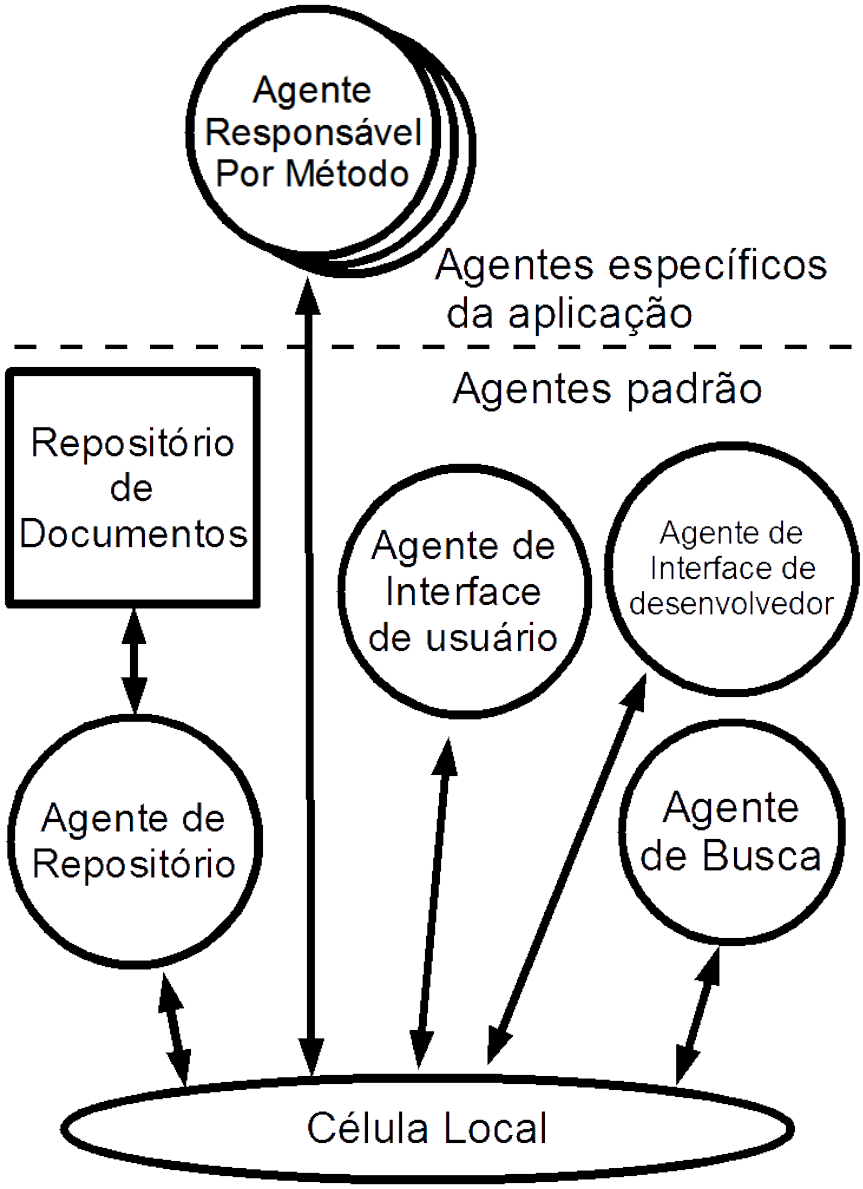

Figura 6.3. Arquitetura da plataforma Coppeer Documents

Agente de busca. O agente de busca é responsável pela busca distribuída no sistema. Ele é capaz de manipular entrada do tipo *NetQueryEntry*, que representam consultas emitidas por agentes locais ou remotos.

Uma entrada do tipo *NetQueryEntry* contem critérios de busca e um identificador de

fonte que identifica a origem da consulta. Quando um agente de busca recebe este tipo de entrada ele primeiramente escreve a entrada nas células vizinhas. Então ele escreve na célula local uma entrada do tipo *RepositoryQueryEntry* para obter do agente de repositório documentos ontologia relacionados a fim de poder expandir o critério de busca. Depois de receber os resultados, o agente de busca utiliza as ontologias para expandir o critério de busca. Novamente, o agente escreve na célula local uma entrada do tipo *RepositoryQueryEntry*, desta vez contendo o critério de busca expandido. Finalmente, o agente combina resultados recebidos do agente de repositório local e de outros nós, e escreve uma entrada do tipo *SearchResultEntry* contendo estes resultados na célula local ou em outro nó, dependendo do especificado no identificador de fonte da entrada original do tipo *NetQueryEntry*.

Agente de interface de usuário. O Agente de interface de usuário provê uma interface universal para usuários de aplicações. Ele pemite a busca de documentos, a invocação de métodos e a criação de novos documentos instância ou espelho.

Quando o agente recebe um critério de busca de um usuário, ele escreve uma entrada correspondente do tipo *NetQueryEntry* na célula local. Depois de receber os resultados, o agente os exibe para o usuário. que pode então requisitar um documento. Nesse caso, o agente escreve uma entrada correspondente do tipo *DocumentRequestQuery* na célula local. Depois de receber o docuento, o agente exibe os dados e a interface do documento para o usuário. Então, o usuário pode invocar um método do documento e fornecer os parâmetros apropriados. Para executar a invocação, o agente escreve na célula local uma entrada do tipo *MethodInvocationEntry*. Depois de receber os resultados da invocação em um entrada do tipo *InvocationResultEntry*, o agente finalmente exibe os resultados ao usuário.

Para criar uma documento instância à partir de um documento modelo ou um documento espelho à partir de um documento instância pré-existente, o agente primeiramente utiliza uma entrada do tipo *DocumentRequestEntry*, conforme descrito acima, para recuperar o documento modelo ou instância, e então utiliza os dados recuperados para criar uma entrada apropriada do tipo *DocumentCreationEntry* que é então escrita na célula local.

Agente de interface de desenvolvedor. O Agente de interface de desenvolvedor provê uma interface para pessoas que atualizam características das aplicações. Este agente pode executar todas as funções do agente de interface de usuário e além disso permite a criação de documentos instância, modelo ou ontologia sem a necessidade de utilização de documentos pré-existentes como base. No caso dessas últimas funções, o agente recebe todas as informações necessárias do desenvolvedor, utiliza-as para construir uma entrada do tipo *DocumentCreationEntry* e escreve a entrada na célula local.

Outra função do Agente de interface de desenvolvedor é a instanciação dos agentes responsáveis pela execução dos métodos presentes nas interfaces de documentos.

Agentes responsáveis por métodos. Agentes responsáveis por métodos são as unidades de código que implementam o comportamento específico de cada aplicação. Depois da criação de uma interface de documento, um desenvolvedor deve criar agentes para tratar a invocação de cada método presente na interface.

Agentes responsáveis por métodos são de certa forma similares às implementações de métodos em linguagens de programação orientadas a objeto (LPOO). Entretanto, enquanto os métodos das LPOO normalmente alteram somente o estado de um objeto, agentes responsáveis por métodos podem atualizar a estrutura, o estado e a interface de um documento.

Para criar um agente responsável por método, um desenvolvedor deve estender a classe *AbstractMethodBehavior* . Esta classe é responsável por fazer uma assinatura na célula local a fim de que o agente seja notificado a respeito da escrita de entradas do tipo *AgentInvocationEntry* dirigidas a ele, e por construir uma representação orientada a objeto do documento alvo contido nas entradas. Essa representação é disponibilizada para o código especializado da classe concreta que implementa o comportamento do agente. Esse código, por sua vez, deve construir e escrever na célula local uma entrada do tipo *DocumentUpdateEntry*, para ser tratada pelo agente de repositório, e outra do tipo *InvocationResultEntry*, para ser tratada pelo agente que invocou o método em questão, ou seja, que iniciou o processo gerando uma entrada do tipo

MethodInvocationEntry.

6.2.4 Exemplo de projeto incremental e colaborativo

Nesta seção, vamos descrever um exemplo simplista que ilustra como a plataforma Coppeer Documents permite o desenvolvimento incremental e distribuído de um sistema de informações. Sejam A e B uma livraria e sua filial localizadas em cidades diferentes. Inicialmente, a companhia realiza apenas vendas a vista e seu sistema de gerenciamento de vendas, implementado sobre a plataforma Coppeer Documents, funciona de forma similar nos dois nós. O sistema gera documentos de venda à partir de um documento modelo denominado *SaleTemplate*. Um documento de venda pertence à classe *Sale* e sua interface contém um método denominado *getValue*. O agente responsável pelo método *getValue* retorna a soma dos valores de todos os itens de venda descrito em um documento de venda.

Se o gerente da filial B decide introduzir vendas a crédito depois que o sistema já está em produção, as seguintes ações devem ser realizadas no nó B para implementar a nova característica:

- criação de um novo modelo *SaleTemplate* contendo um método *convertToCreditSale* em sua interface. Documentos baseados nesse modelo serão similares ao apresentado na figura 6.2;
- criação de um agente responsável pelo método *convertToCreditSale*. Este agente deve substituir o identificador de classe do documento de *Sale* para *CreditSale*, adicionar ao documento os valores das prestações, remover a assinatura do método *convertToCreditSale* e substituir o identificador de versão do método *getValue*. A figura 6.4 apresenta uma possível representação XML do documento da figura 6.2 após uma invocação de *convertToCreditSale*;
- criação de um novo agente responsável pelo método *getValue* nas invocações sobre documentos de venda a crédito. Esse agente deve retornar a soma dos valores de todas as prestações adicionadas ao documento;
- criação de um documento ontologia definindo *CrediSale* como uma subclasse de *Sale*.

Como nenhum código é removido por essas ações, vendas a vista são tratadas como antes. Um invocação do método *getValue* sobre uma venda a crédito, por sua vez, será tratada pelo novo agente responsável pelo método. Embora o nó A não contenha informações sobre a classe *CreditSale*, uma busca global por todos os documentos pertencentes à classe *Sale* incluirão vendas a crédito, uma vez que a consulta gerada pelo nó A é expandida no nó B pela ontologia que define *CreditSale* como uma subclasse de *Sale*.

```
<doc id="s41154" role_type ="instance">
    <class id="CreditSale"/>
    <item cod = "bk82948238 qty="3"
        value="98.54"/>
    </item>
    <instalment value="49.50"
        exp_date="2007-08-10" />
    <instalment value="49.50"
        exp_date="2007-09-10"/>
    <method name="getValue"
        version_id="getValue02"/>
</doc>
```

Figura 6.4 - Examplo de documento de venda a crédito

6.3. Redes de Mundo Pequeno no CoppeerCAS

Nesta seção, apresentaremos uma proposta de mecanismo para obtenção de redes de mundo pequeno para o CoppeerCAS. A geração de tais redes visa aumentar a eficiência da propagação de informação em aplicações CoppeerCAS, como, por exemplo, a proposta na seção 6.1 .

O mecanismo é inspirado no método de Watts e Strogatz para geração de redes de mundo pequeno. Em conseqüência, as células são distribuídas aleatoriamente pelo mecanismo em um anel imaginário e dois tipos de conexões são permitidas: conexões para vizinhos próximos no anel e atalhos para vizinhos distantes. Os parâmetros principais do procedimento são *n*, o número máximo estimado de nós na rede, *d*, a máxima distância dentro da qual um vizinho é considerado próximo, *h*, o número máximo de saltos que um agente pode realizar, e *r*, a razão a ser mantida entre o número

de atalhos e de conexões para vizinhos próximos. Para implementar a distribuição pelo anel imaginário, cada agência que entra em um ambiente gera um número aleatório $l \in [0,1]$, que representa a posição de sua célula no anel. Para entrar no ambiente, uma agência deve conhecer o identificador de pelo menos uma outra agência já presente no ambiente. A célula da agência que está entrando conecta-se à célula conhecida e periodicamente envia agentes que caminham aleatoriamente pela rede para coletar identificadores de outras agências já presentes no ambiente, juntamente com as respectivas posições no anel imaginário. Depois de no máximo h saltos, cada agente retorna para a origem e estabelece conexões com células de algumas das agências percorridas de forma a manter próximo de r o valor da razão entre o número de atalhos e conexões a vizinhos próximos.

Tabela 6.5. Resultados do simulador de topologia de mundo pequeno

n	***d***	**Caminho Minimo Médio**	**Coeficiente de Agrupamento**
100	0.030	2.71	0.42
500	0.013	2.93	0.49
1000	0.007	3.19	0.49
2500	0.004	3.27	0.52

Um simulador *ad hoc*, baseado na API Java de simulação por eventos discretos JIST (BARR, 2005), foi implementado para observarmos o comportamento do mecanismo proposto. Os experimentos realizados, com o valor de n variando de 100 a 2500 nós, sugerem um crescimento logarítmico para o tamanho do caminho mínimo médio em função do número de nós. A tabela 6.5 mostra alguns resultados típicos, obtidos com $d \approx \ln(n)/n$, $r = 0.1$ e $h = 12$. A função utilizada para determinar d tem por objetivo assegurar a escalabilidade da abordagem – considerando que os n nós serão uniformemente distribuídos no anel, cada um irá se conectar cerca de $2nd$ vizinhos próximos em média. O valor h foi ajustado empiricamente e a escolha de r foi baseada nos resultados de Watts e Strogatz.

Em todos os experimentos, o nós entraram na rede a taxas fixas e a simulação

era executada até ocorrer a convergência. Alguns dos experimentos mostraram que o caminho mínimo médio e o coeficiente de agrupamento variam com r conforme o esperado e que a rede era capaz de manter seus parâmetros mesmo depois da saída simultânea de cerca de 15% dos nós.

7. CONSIDERAÇÕES FINAIS

Neste trabalho, procuramos investigar a viabilidade do emprego de princípios de sistemas emergentes na construção de sistemas ponto-a-ponto. Os princípios de estruturação do sistema em agentes, de heterarquia e de estigmatismo foram diretamente incorporados a uma arquitetura lógica de agentes móveis autônomos que se comunicam por intermédio de espaços compartilhados. Esta arquitetura serviu de base, então, para o desenvolvimento de uma plataforma para aplicações ponto-a-ponto.

A funcionalidade de propagação de informações entre espaços compartilhados foi incluída na arquitetura de forma a ampliar as possibilidades da comunicação estigmática previstas no modelo original de espaços compartillhados e a facilitar o emprego do princípio de drenagem de entropia em aplicações.

O problema da descoberta de recursos, que se apresenta como uma questão fundamental em sistemas ponto-a-ponto, foi abordado no escopo da arquitetura lógica, resultando na concepção do método TIGRAS. O método baseia-se no princípio de drenagem de entropia, ou seja, na manutenção de um campo de informações que permeia a rede ponto-a-ponto com a finalidade de orientar agentes móveis que realizam buscas de gradiente para localizar recursos. O campo de informações mantido pelo método TIGRAS é implementado em estruturas baseadas em filtros Bloom, que são propagadas e processadas pelos nós da rede de forma a oferecer estimativas de distâncias de cada nó aos recursos dispersos pelo sistema. O método TIGRAS é completamente distribuído e as propriedades de escalabilidade e de resiliência foram observadas em experimentos de simulação. Os agentes do método, embora apresentem comportamento relativamente simples quando observados isoladamente, colaboram para a resolução de um problema complexo. Dessa forma, os resultados obtidos com o método TIGRAS sugerem a conveniência do emprego de princípios de sistemas emergentes na construção de sistemas ponto-a-ponto.

A fim de ilustrar o emprego dos princípios estudados e da plataforma desenvolvida na área de colaboração, duas infraestruturas foram propostas. A Infraestrutura de Apoio ao Projeto Emergente consiste em um arcabouço no qual usuários vistos como agentes que realizam operações simples, organizando-se de forma

heterárquica e utilizando comunicação estigmática, podem colaborar na execução de projetos complexos. A Infraestrutura para desenvolvimento incremental e colaborativo, por sua vez, prevê a divisão de sistemas de informação em um grande número de agentes que empregam comunicação estigmática. Esta abordagem procura gerar sistemas de baixo acoplamento que podem ser estendidos pela adição de novos agentes sem necessidade de qualquer modificação em componentes pré-existentes.

Além das infraestruturas propostas, uma série de aplicações de colaboração e recuperação de informações foram implementadas por outros pesquisadores. Estas aplicações utilizaram as primitivas da plataforma para implementar estruturação em agentes, comunicação entre os nós da rede e buscas distribuídas.

7.1. Principais contribuições

A arquitetura lógica Coppeer, que incorpora princípios de sistemas emergentes na combinação dos paradigmas de agentes móveis autônomos e espaços compartilhados, foi empregada na implementação da plataforma CoppeerCAS, destinada ao desenvolvimento de aplicações ponto-a-ponto em linguagem Java. Além de favorecer o mapeamento de algoritmos concebidos com base em princípios de sistemas emergentes, a plataforma oferece abstrações de alto nível para a implementação de comunicação assíncrona e execução de múltiplas *threads*, reduzindo assim o esforço de programação de aplicações distribuídas de uma forma geral.

Como conseqüência, a plataforma contribuiu para implementação de diversas ferramentas de pesquisa, nas áreas de trabalho colaborativo, recuperação de informação e integração de informações, e recebeu aperfeiçoamentos motivados pelas demandas dessas ferramentas.

Outra contribuição central deste trabalho foi a concepção do método TIGRAS, que emprega princípios de sistemas emergentes para tratar de forma satisfatória o problema da descoberta de recursos em redes dinâmicas e não-estruturadas.

Finalmente, este trabalho incluiu a elaboração de propostas de infraestruturas que empregam princípios de sistemas emergentes para apoiar, respectivamente, o projeto colaborativo e o desenvolvimento incremental e colaborativo de sistemas de

informações.

7.2. Trabalhos propostos

Tendo em vista os objetivos iniciais de pesquisa e os desdobramentos encontrados no tratamento dos problemas abordados, consideramos que a continuidade natural deste trabalho inclui as seguintes atividades:

- implementação de publicação/subscrição distribuída utilizando princípios de sistemas emergentes;
- implementação de mecanismos de segurança de sistemas emergentes, destinados a permitir que as aplicações atinjam seus objetivos mesmo na presença de nós maliciosos;
- aplicação de princípios emergentes no desenvolvimento de algoritmos distribuídos clássicos tais como, por exemplo, acordo bizantino e eleição de líder;
- avaliação completa do método TIGRAS visando determinar seus limites de desempenho e escalabilidade, bem como seu comportamento em topologias particulares e redes *ad hoc*;
- avaliação de alternativas aos filtros Bloom para construção de sumários para o método TIGRAS, como por exemplo, Wavelets (MALLAT, 1989).
- implementação de um sistema de recuperação de informações sobre o método TIGRAS ou utilizando diretamente princípios de sistemas emergentes;
- desenvolvimento de um ambiente de desenvolvimento integrado e de um simulador genérico para o CoppeerCAS;
- incorporação à plataforma de ferramentas para provisão e consumo de Serviços Web (W3C, 2000, RICHARDSON, 2007);
- emprego da plataforma para a implementação de Simulações Distribuídas e Sistemas de Comando e Controle.

REFERÊNCIAS BIBLIOGRÁFICAS

ABERER, K., KLEMM, F., RAJMAN, M.,WU, J., 2004, "An Architecture for Peer-to-Peer Information Retrieval", *27th Annual International ACM SIGIR Conference (Workshop on Peer-to-Peer Information Retrieval)*.

BABAOGLU, O, MELING, H., MONTRESOR, A.,2002, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems", In: *Proceedings of the 22th International Conference on Distributed Computing Systems*, pp. 15-22, Vienna, July.

BARR, R., HAAS, Z. J., RENESSE, R., 2005, "JiST: An efficient approach to simulation using virtual machines", *Software Practice & Experience*, n.35, v.6, pp. 539-576, May.

BERKELEY, 1999, *SETI@home*, <http://setiathome.berkeley.edu>, Acesso em Setembro/2006.

BLOOM, B., 1970, "Space/time trade-of's in hash coding with allowable errors", *Communications of the ACM*, n.13, v.7, pp. 422-426.

BRAGA, B., 2004, *COPPEER-DB:Uma Plataforma para Gerência de Dados Distribuída em Redes Peer-to-Peer,* Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

CASTRO, M., COSTA, M., AND ROWSTRON, A., 2005, "Debunking some myths about structured and unstructured overlays", In: *Proceedings of the NSDI '05,* Boston, May.

CLARKE, I., SANDBERG, O., WILEY, B., HONG, T., 2000, "Freenet: A distributed anonymous information storage and retrieval system", In: *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, pp. 46-66, July.

COSTA, P., MIGLIAVACCA, M., PICCO, G., CUGOLA, G., 2003, "Introducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms", In: *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03).*

DODDS, P. S., MUHAMAD, R., WATTS, J., 2003, "An Experimental Study of Search in Global Social Networks", *Science,* v. 301, n. 5634, pp. 827 – 829, Aug.

DORIGO, M., BONABEAU, E.,THERAULAZ, G.,2000, "Ant algorithms and

stigmergy", *Future Generation Computer Systems*,v. 16, n. 9, pp. 851-871.

EUGSTER, P.T., GUERRAOUI, R., 2002, "Probabilistic multicast", In: Proceedings of the 3rd IEEE International Conference on Dependable Systems and Networks (DSN 2002), pp 313–322.

EUGSTER, P. T., FELBER, P. A., GUERRAOUI, R., KERMARREC, A. M., 2003, "The many faces of publish/subscribe", *ACM Computing Surveys*, v. 35(2), pp 114-131.

FIELDING, R. T., 2000. *Architectural styles and the design of network-based software architectures,* Doctoral dissertation, University of California, Irvine.

GNUTELLA,2001, *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, <http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf>, Acesso em Setembro/2006.

GUPTA, A., SAHIN, O. D., AGRAWAL, D., ABBADI, A. E., 2004, "Meghdoot: Content-Based Publish/Subscribe over P2P Networks", In: *Proceedings of Middleware*.

HELLERSTEIN, J. M., 2003, "Toward Network Data Independence", *SIGMOD Record*, v. 32, n. 3, pp 34-40, Sep.

JOHNSON, S., 2003, *Emergência, A vida integrada de formigas, cérebros, cidades e software,* Rio de Janeiro, Jorge Zahar.

JXTA, 2001, *JXTA v2.0 Protocols Specification*, <http://spec.jxta.org/nonav/v1.0/ docbook/ JXTAProtocols.pdf.>, Acesso em Setembro/2006.

KLEINBERG, J., 1999, *The small-world phenomenon: An algorithmic perspective*, Cornell Computer Science, TR 99-1776.

LEINER, B.M., CERF, V.G., CLARK, D.D. et al. "A Brief History of the Internet", <http://www.isoc.org/internet/history/brief.shtml>, Acesso em Dezembro/2008

MALLAT, S., 1989, "A theory for multiresolution signal decomposition: The wavelet representation", *IEEE Transactions on Pattern Analysys and Machine Intelligence*, v. 11, n. 7, pp. 674-693, July.

MAMEI, M., ZAMBONELLI, F.,2005, "Programming stigmergic coordination with

the TOTA middleware", In: *Proceedings of the Fourth international Joint Conference on Autonomous Agents and Multiagent Systems*, pp 415-422, Netherlands, July.

MANKU,G. S., BAWA, M., RAGHAVAN, P., 2003, "Symphony: Distributed hashing in a small world", In: *4th USENIX Symposium on Internet Technologies and Systems.*

MAYWORM, M., *Um Crawler Peer-to-peer Baseado em Agentes,* Dissertação de Mestrado, Universidade Federal do Rio de Janeiro.

MIGLIARDI, M., SCHUBIGER, S., SUNDERAM, V., 2000, "A Distributed JavaSpace Implementation for Harness", *Journal of Parallel and Distributed Computing,* v. 60, n.10, pp. 1325-1340.

MILGRAM, S., 1967, "The small world problem", *Psychology Today*, pp. 60-67, May.

MILOJICIC, D., KALOGERAKI, V., LUKOSE, R. et al., 2002, *Peer-to-peer computing*, HP Labs, HPL-2002-57.

MINSKY, M., 1988, *The Society of Mind*, New York, Simon & Schuster.

MIRANDA, M. , XEXEO, G. B. , SOUZA, J. M., 2008, "TIGRAS: A Topology-Independent Gradient Search Approach for Peer-to-Peer Key Look Up", In: *Proceedings of IEEE 11th International Conference on Computational Science and Engineering* , pp. 197-202, São Paulo, July.

MIRANDA, M. , XEXEO, G. B. , SOUZA, J. M., 2007, "A Framework to Collaborative and Incremental Development of Distributed Information Systems", *CSCWD (Selected Papers)*, pp. 273-281.

MIRANDA, M. , XEXEO, G. B. , SOUZA, J. M., 2006, "Building Tools for Emergent Design with COPPEER", In: *Proceedings of 10th International Conference on Computer Supported Cooperative Work in Design*,.v. I., pp. 550-555, Nanjing, May.

NAPSTER, 2001, *Napster protocol specification*, <http://opennap.sourceforge.net/ napster.txt>, Acesso em Setembro/2006.

NEJDL, W., WOLF, B., QU, C. et al., 2002, "Edutella: A p2p networking infrastructure based on RDF", In: *Proceedings of the 11th International World Wide Web Conference*, pp. 604-615, Hawaii, May.

PARUNAK, V., 1997,. "Go to the Ant: Engineering Principles from Natural Agent

Systems", *Annals of Operations Research,*.n. 75 (Special Issue on Artificial Intelligence and Management Science), pp. 69 – 101.

RATNASAMY, S., FRANCIS, P., HANDLEY, S. et al., 2001, "A scalable content-addressable network", In: *Proceedings of the ACM SIGCOMM*, pp. 161-172, San Diego, Aug.

RICHARDSON, L., RUBY, S., 2007, RESTful Web Services. Web services for the real world, O'Reilly, May.

RONALD, E. M., SIPPER, M., CAPCARRERE, M. S., 1999, "Design, observation, surprise! A test of emergence"*, Artificial Life, n. 5(3), pp. 225-239.*

ROWSTRON, A., DRUSCHEL, P., 2001, "Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems", In: *Proceedings of the International Conference on Distributed Systems Platforms (Middleware)*, pp. 329-350, Heideberger, Germany, Nov.

SOUZA, J. M. , VIVACQUA, A. , MORENO, M., 2005, "CUMBIA: An Agent Framework to Detect Opportunities for Collaboration", In: *Proceedings of 9th International Conference on Computer Supported Cooperative Work in Design,* pp 434-445, Coventry, U.K., May.

SOUZA, J. M., REZENDE, J. L., SILVA, R. L. S., 2005, "Building Personal Knowledge through Exchanging Knowledge Chains", In: *Proceedings of IADIS 2005 - Web Based Communities*, Algarve, Feb.

SOUZA Jr., H. C., MOURA, A. M. C., CAVALCANTI, M. C., 2010, "Integrating Ontologies Based on P2P Mappings", *IEEE Transactions on Systems, Man and Cybernetics. Part A, Systems and Humans*, vol. 40: pp 1-12.

SPRING, 2006, *Spring Java/J2EE Application Framework Reference Documentation version 1.2.8*, <http://static.springframework.org/spring/docs/1.2.x/spring-reference.pdf>, Acesso em Setembro/2006.

STOICA, I., MORRIS, R., KARGER, D. et al., 2001, "Chord: A scalable peer-to-peer lookup service for internet applications", In: *Proceedings of the ACM SIGCOMM*, pp. 149-160, San Diego, Aug.

SUN Microsystems, 1999, *JavaSpaces Specification,* Palo Alto, CA, Jan.

SUN Microsystems, 2004, *Java Remote Method Invocation Specification*, Santa Clara, CA.

TANG, C., DWARKADAS, S., 2004, "Hybrid global-local indexing for efficient peer-to-peer information retrieval*",* In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, pp 211- 224, March.

TERPSTRA, W. W., BEHNEL, S, FIEGE, L et al., 2003, "A peer-to-peer approach to content-based publish/subscribe". In: *Proceedings of the 2nd international workshop on Distributed event-based systems*, San Diego, California, June

VIVACQUA, A., RODRIGUES Nt., J.A., MACHADO, M. et al., 2009, "Community-supported collaborative navigation with FoxPeer", *International Journal of Web Based Communities*, vol. 5(1): pp 126-138.

W3C, 2000, *Soap specifications*, <http://www.w3.org/TR/soap>, Acesso em Setembro/2006.

WATTS, D. J., STROGATZ, S. H., 1998, "Collective dynamics of 'small-world' networks", *Nature*,vol. 393, pp. 440-442.

WIDROW, B., RUMELHART,D.E., LEHR, M.A, 1994, "Neural networks: applications in industry, business, and science", *Communications of the ACM*, n. 37, pp 93 - 105