COPPE/UFRJ

UM MODELO DE AUTOGERENCIAMENTO DE SISTEMAS DISTRIBUÍDOS INSPIRADO NO SISTEMA CIRCULATÓRIO

Alberto Arkader Kopiler

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 Ciências em Engenharia de Sistemas e Computação.

Orientadores: Felipe Maia Galvão França
Inês de Castro Dutra

Rio de Janeiro
Março de 2010

UM MODELO DE AUTOGERENCIAMENTO DE SISTEMAS DISTRIBUÍDOS INSPIRADO NO SISTEMA CIRCULATÓRIO

Alberto Arkader Kopiler

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. Felipe Maia Galvão França, Ph.D.

_____
Prof. Adilson Elias Xavier, D.Sc.

_____
Prof. Luis Alfredo Vidal de Carvalho, D.Sc.

_____
Profa. Inês de Castro Dutra, Ph.D.

_____
Profa. Ana Cristina Bicharra Garcia, Ph.D.

_____
Prof. Max Suell Dutra, Dr.-Ing.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2010

Kopiler, Alberto Arkader

Um Modelo de Autogerenciamento de Sistemas Distribuídos Inspirado no Sistema Circulatório/ Alberto Arkader Kopiler. – Rio de Janeiro: UFRJ/COPPE, 2010.

XIV, 193 p.: il.;29,7 cm.

Orientadores: Felipe Maia Galvão França

Inês de Castro Dutra.

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

Referências Bibliográficas: p. 114-125.

1. Autogerenciamento de Sistemas. 2. Inspiração Biológica. 3. Computação Circulatória. 4. Computação Autonômica. 5. Multiagentes Móveis. 6. Inteligência Coletiva. I. França, Felipe Maia Galvão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título

Aos meus pais, minha esposa e meus filhos

Agradeço aos meus orientadores Inês de Castro Dutra e Felipe Maia Galvão França, a compreensão e orientação desta tese.

Agradeço à direção do Departamento de Automação de Sistemas do Centro de Pesquisas de Energia Elétrica – ELETROBRAS/CEPEL, Raul Balbi Sollero, a oportunidade de realização deste trabalho.

Agradeço aos colegas de trabalho do Departamento de Automação de Sistemas, inclusive Alvaro da Silva Ferreira, o apoio e a troca de ideais durante a realização da pesquisa.

Agradeço a toda equipe do OurGrid, em especial a Francisco Brasileiro, Rodrigo Vilar, Abmar Barros e Adabriand Furtado, o suporte para implementação do estudo de caso real e obtenção de resultados.

Agradeço a Marlon Rocha seu apoio nos testes realizados no LabIA do PESC/COPPE.

Agradeço a meu pai (*in memoriam*) seu exemplo de vida, honestidade e perseverança na busca de seus objetivos e superação de obstáculos.

Agradeço a minha mãe todo seu carinho e incentivo aos estudos.

Agradeço a meus filhos Alan, Alexandre e André, a compreensão neste período e o questionamento: "— Pai, quando você vai terminar seu trabalho? — Doutorado serve pra quê?".

Agradeço por último, mas não menos importante, a minha esposa Rosana sua dedicação e amor.

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

UM MODELO DE AUTOGERENCIAMENTO DE SISTEMAS DISTRIBUÍDOS INSPIRADO NO SISTEMA CIRCULATÓRIO

Alberto Arkader Kopiler

Março/2010

Orientadores: Felipe Maia Galvão França
Inês de Castro Dutra

Programa: Engenharia de Sistemas e Computação

O interesse pelas tecnologias de autogerenciamento de sistemas distribuídos, isto é, que possibilitam que um sistema seja controlado por ele mesmo em ambientes descentralizados, vem crescendo na mesma proporção que aumenta a complexidade do próprio sistema. O paradigma do homem como o centro do gerenciamento do sistema não é mais apropriado para os sistemas computacionais distribuídos atuais. É necessário colocá-lo em um papel de supervisão do processo definindo políticas e metas gerais de mais alto nível, delegando ao sistema autogerenciado de mais baixo nível, alcançá-las. Um modelo para autogerenciamento de sistemas distribuídos inspirado no sistema circulatório humano é apresentado como contribuição para atender esta necessidade. Os resultados de simulações e da aplicação do modelo a um ambiente de computação em grade são apresentados, a partir de estudo de casos. O modelo inspirado biologicamente no sistema circulatório se mostra viável para aplicação a problemas reais nos quais se deseja elevar os níveis de autonomia de gerenciamento.

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

A SELF-MANAGEMENT MODEL INSPIRED BY THE CIRCULATORY SYSTEM APLLIED TO DISTRIBUTED SYSTEMS

Alberto Arkader Kopiler

March/2010

Advisors: Felipe Maia Galvão França
Inês de Castro Dutra

Department: Computing and Systems Engineering

The interest in self-management techniques applied to distributed systems, that is, techniques that allow systems to control themselves in decentralized environments, is increasing in the same proportion as the system increases its complexity. The paradigm of a man as the centre of the management is no longer appropriate for the modern distributed computing systems. Man must be reallocated to a supervisory role, define high-level policies and general goals, and delegate to the low-level self-managed system reaching them. A model for distributed systems' self-management inspired in the human circulatory system is then presented as a contribution to attend this demand. Based on case studies, results obtained by simulation and application of the model to a grid computing environment are reported. The biologically inspired model is feasible for real applications in which the increase in autonomy levels is desired.

Índice

Figuras

Tabelas

1 Introdução

As organizações modernas atuais são fortemente dependentes da tecnologia para manter a qualidade de seus produtos e serviços frente à demanda cada vez maior do cliente e a exigência de fazer frente à concorrência. Pode-se tomar como exemplo destas organizações: empresas de cartão de crédito, bancos, empresas do setor de energia elétrica, telecomunicações, rede de supermercados, empresas do setor de petróleo, entre outras. Para atender à demanda estas organizações dispõem de computadores interligados em rede e conectados à Internet ininterruptamente, processando os programas computacionais necessários para atender seus objetivos. Ou seja, os computadores ficam ligados sem interrupção, vinte e quatro horas por dia, sete dias por semana, recebendo, transmitindo e armazenando informação. Pode-se enxergar, então, essa organização como sendo um sistema de informação e não apenas como possuindo um. Esta infraestrutura de hardware e software precisa ser mantida funcionando ininterruptamente, de forma regulada, integrada, otimizada e protegida. Além disso, prevê-se que para cada usuário existirão dezenas ou centenas de “computadores” para serem gerenciados. Neste trabalho é buscada inspiração em sistemas biológicos para modelar estes sistemas computacionais complexos, de forma a enfrentar os novos desafios apresentados. Sistemas biológicos são os que mais se assemelham e possuem as características de sistemas computacionais complexos, como os descritos.

Assim como os organismos vivos precisam ter seus subsistemas regulados internamente, de forma a que seja mantido um equilíbrio entre eles, e externamente, de acordo com o ambiente em que vivem, bem como prover formas de defesa contra ameaças externas (camuflagem, sensores, veneno, velocidade, habilidades entre outras), as organizações precisam manter sua infra-estrutura operando permanentemente de forma regulada, otimizada, protegida, com qualidade e lucratividade para atender a sua atividade fim, provendo formas de defesa contra a concorrência, vandalismo, má utilização, sobrecarga, intempéries, etc.

O interesse pelas tecnologias de autogerenciamento de sistemas distribuídos, isto é, que possibilitam que um sistema seja controlado por ele mesmo em ambientes descentralizados, vem crescendo na mesma proporção que aumenta a complexidade do próprio sistema. O paradigma do homem como o centro do gerenciamento do

sistema (“*man inside the loop*”), controlando os processos manualmente e participando interativamente no processo de decisão, não é mais apropriado para os sistemas computacionais distribuídos atuais (Figura 1.1). Por outro lado, um sistema completamente automático sem necessidade de intervenção humana, utilizando técnicas puras de inteligência artificial é uma meta muito difícil de ser alcançada em curto prazo [90]. Como novo paradigma é necessário reposicionar o operador humano colocando-o acima do processo (“*man above the loop*”) de forma que ele passe a definir políticas e metas gerais de mais alto nível, delegando ao sistema autogerenciado, de mais baixo nível, alcançá-las (Figura 1.2).

Figura 1.1 – *Man inside the loop*

A proposta desta tese é apresentar um modelo de autogerenciamento inspirado biologicamente para prover o gerenciamento autonômo da infraestrutura de sistemas distribuídos baseados em computador. Este modelo, denominado de computação circulatória (CIACOM), se baseia no sistema circulatório humano e está focado na operação supervisionada pelo homem, ou seja, a ênfase é dada a sistemas autogerenciados supervisionados por administradores humanos e não a sistemas completamente automáticos.

Figura 1.2 – *Man above the loop*

1.1 Motivação

Desde os primórdios da teoria da computação são buscadas idéias inspiradas na natureza que muitas vezes se transformam em técnicas computacionais de sucesso. As seguintes técnicas podem ser citadas como exemplo: redes neurais artificiais, algoritmos genéticos, computação evolucionária, inteligência de enxames, sistema imunológico artificial e colônia de formigas [1].

Em um nível mais metafórico, a IBM tomou a iniciativa de desenvolver pesquisa em computação inspirada em sistemas biológicos [2] para o gerenciamento de sistemas computacionais. Mais especificamente, a IBM lançou a idéia de Computação Autonômica em 2001, a partir de um manifesto [3]. Ela realiza abstrações desejáveis em sistemas computacionais complexos possibilitando seu controle, assim como o sistema nervoso autônomo (SNA) libera a parte consciente do cérebro de ter que manejar as funções vitais de mais baixo nível do organismo. O objetivo é obter um sistema que seja capaz de se configurar e reconfigurar em condições imprevisíveis (incertas) e que estejam constantemente variando, para continuamente otimizar sua operação, para se recuperar de eventos rotineiros ou extraordinários que podem fazer com que algumas partes funcionem mal, de forma análoga ao processo de cura em sistemas biológicos, e para se proteger contra ameaças em seu ambiente (externo). A funcionalidade de autogerenciamento é alcançada através de aspectos chave como

autoconfiguração, autorrecuperação, auto-otimização e autoproteção, chamadas de propriedades auto* ou autopropriedades [4].

Sistemas autonômicos podem ser aplicados ao gerenciamento da infraestrutura de sistemas elétricos, sistemas de telecomunicações, sistemas financeiros, sistemas de transporte, sistemas de logística e sistemas espaciais, entre outros.

1.2 Contexto

Este trabalho está inserido no contexto de sistemas computacionais inspirados na natureza, para desenvolver um modelo para o autogerenciamento da infraestrutura de sistemas computacionais. Então, como pode ser visto na Figura 1.3, dentre os diversos sistemas inspirados na natureza, encontram-se aqueles inspirados em sistemas biológicos, e dentre estes estão os sistemas inspirados no sistema nervoso e por sua vez os inspirados no sistema nervoso autônomo. A computação autonômica se encaixa dentro deste último.

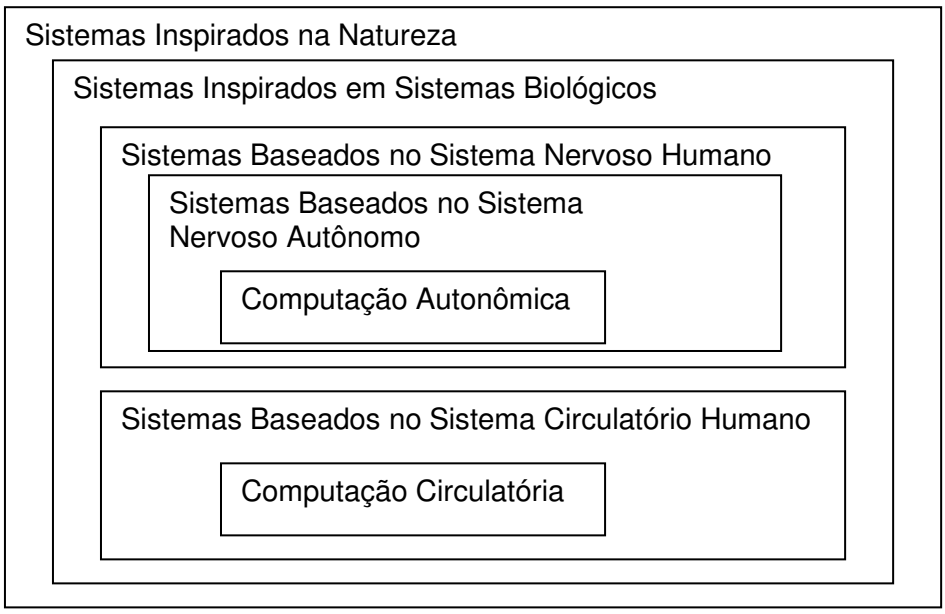

Figura 1.3 – Contextualização do Modelo de Computação Circulatório-Autonômico Proposto.

Para atingir o autogerenciamento, outras inspirações biológicas também são propostas, como por exemplo, o sistema circulatório humano, dando origem à computação circulatória, termo cunhado neste trabalho e que ainda não tinha sido

formalizado anteriormente, apesar de algumas de suas características poderem ser encontradas em [17,54].

1.3 Objetivos

Nosso principal objetivo com este trabalho é o estudo investigativo de um modelo de autogerenciamento com inspiração biológica, baseado no sistema circulatório. Nossos objetivos específicos são:

a) Estudar os trabalhos relacionados e suas limitações, vantagens e desvantangens;
b) Projetar um novo modelo de autogerenciamento baseado no sistema circulatório com características próximas a de sistemas autonômicos, mas com maior autonomia e maior coesão entre as propriedades auto*;
c) Simular e implementar um protótipo;
d) Investigar sua viabilidade aplicando-o a um ambiente distribuído alvo (em nosso caso, ambiente *grid*).

1.4 Contribuições

Dentre as contribuições do trabalho podem-se destacar:

a) Uma proposta inédita de autogerenciamento de sistemas computacionais baseada no sistema circulatório humano;
b) Dotação de forma nativa de mecanismos que suportem o autogerenciamento;
c) Modelo de sistema de autogerenciamento que possui intrinsicamente processos de controle fechados (*loops* de controle) que funcionam de forma integrada, ora cooperativamente, ora competitivamente, mas que fazem emergir o comportamento global desejado;

1.5 Justificativas

Existe uma tendência para que os sistemas atuais cresçam e se tornem mais complexos dificultando o gerenciamento pelo usuário. O desenvolvimento de novos sistemas deve prever características de autogerenciamento de forma a colocar o

usuário em uma posição supervisória [90]. Ou seja, novas aplicações devem ser dotadas, de forma nativa, de mecanismos que suportem seu autogerenciamento. Projetando sistemas que intrinsicamente possuam processos de controle fechados (*loops* de controle) provê uma abstração útil para atingir este objetivo.

Apesar de existirem iniciativas como a Computação Autonômica, o modelo proposto se destaca por ser mais integrado, levando em consideração o interrelacionamento das propriedades autonômicas (autopropriedades) e por buscar mais inspirações biológicas.

1.6 Desafios

Diversos temas de pesquisa em ciência da computação são discutidos pela comunidade científica e organizados e lançados como desafios a serem enfrentados nos próximos anos. Podem ser citados:

a) “Grandes Desafios da Pesquisa em Computação no Brasil – 2006 – 2016” [86], dos quais dos cinco grandes desafios listados pode ser selecionado o desenvolvimento tecnológico de qualidade: sistemas disponíveis, corretos, seguros, escaláveis, persistentes e ubíquos;

b) “Journey of Non-Classical Computation” [87] que na página 23, cita como uma das jornadas desafiantes, o desenvolvimento de um hardware computacional (inspirado biologicamente) capaz de se adaptar, evoluir, se curar, replicar e aprender; além disso, esse mesmo documento apresenta outras jornadas em que sistemas computacionais inspirados em biologia são colocados como possíveis linhas de pesquisa para enfrentar esses desafios (ex: Sistemas Imunológicos Artificiais).

Uma das questões relativas aos “Grandes Desafios” é a seguinte: uma razão para que o problema se torne um desafio crescente é o fato de que os sistemas de software estão se tornando cada vez mais complexos (volume e abrangência da funcionalidade e requisitos de qualidade mais restritivos), precisam estar disponíveis por mais tempo (sistemas 24h / 7 dias por semana) e serem utilizados por pessoas sem garantia de treinamento.

Questões relativas à “Jornada” são: Como podemos fazer “evoluir” sistemas com a complexidade dos sistemas produzidos com a tecnologia de daqui há 10 ou 20 anos? Como podemos construir sistemas que possam aprender e se adaptar a seus ambientes, interno e externo, de maneira a melhorar o seu desempenho, que se torne imune a ataques, internos e externos, e que possam aprender a usar todos os recursos disponíveis para eles?

Outros documentos apresentam uma antevisão do que será a computação no futuro, como por exemplo, “The Computer for the 21st Century” [88]. Em trecho extraído: “Centenas de computadores em uma sala podem parecer intimidantes a princípio. Assim como, centenas de *volts* transitando pelos fios elétricos pareceram assim um dia. Mas assim como os fios elétricos no interior das paredes, estas centenas de computadores ficarão invisíveis para a consciência comum. As pessoas simplesmente usam-nos inconscientemente para cumprir seus afazeres diários”, chama-se a atenção para aspectos relacionados à computação pervasiva, onipresente, fazendo papel de assistente pessoal, mas apresentando autonomia e características descritas na computação autonômica, onde a técnica possibilitará ao homem gerenciar sistemas complexos como um todo pelo foco em objetivos de mais alto nível (consciente), deixando os de baixo nível (inconsciente) para serem gerenciados automaticamente pelos sistemas autonômicos.

Concluindo, buscar modelos que inspirados em sistemas biológicos, auxiliem no gerenciamento de sistemas computacionais, se mostra uma jornada ao mesmo tempo desafiante e promissora.

1.7 Estrutura do Texto

O texto está organizado em seis capítulos, referências bibliográficas e sete anexos. O primeiro capítulo é introdutório contendo os seguintes itens: o tema da tese, a motivação para sua realização, o seu contexto, os objetivos a serem alcançados, as contribuições proporcionadas, os desafios a serem enfrentados e sua justificativa.

No segundo capítulo é apresentada uma revisão bibliográfica dos trabalhos relacionados com o tema da tese. Destaca-se, entre os modelos com inspiração biológica, o modelo computacional inspirado no sistema nervoso autônomo (Computação Autonômica).

No terceiro capítulo é apresentado o novo modelo computacional proposto, inspirado nos sistemas circulatório e nervoso autônomo humano: o modelo de computação circulatório-autonômico. São apresentadas as características deste modelo para complementar a modelagem autonômica já existente.

Na implementação do CIACOM, tratada no quarto capítulo, foi utilizado o paradigma de multiagentes móveis.

No quinto capítulo são apresentados estudos de caso para os quais o modelo proposto foi aplicado: sistemas computacionais distribuídos, ambiente de computação em grade e operação do setor elétrico. Os testes e resultados obtidos por simulação (sistemas computacionais distribuídos) ou implementação (ambiente em grade) são apresentados.

No sexto capítulo as conclusões deste trabalho são apresentadas em face à análise dos resultados obtidos, ressaltando as contribuições esperadas e trabalhos futuros.

Em seguida são apresentadas as referências bibliográficas e sete anexos contendo: (1) abordagem sobre vários aspectos do sistema circulatório humano, (2) considerações adicionais sobre o modelo de computação circulatória, (3) programas e arquivos auxiliares para *jobs* OurGrid, (4) histórico e tendências da aplicação de inspiração biológica à computação, (5) programa fonte de simulação do CIACOM em ambiente NetLogo, (6) programa fonte de implementação do CIACOM em ambiente JADE escritos na linguagem JAVA e (7) um glossário de termos.

2 Trabalhos Relacionados

Apesar deste trabalho se basear principalmente na Computação Autonômica, outros trabalhos relacionados ao autogerenciamento de sistemas são pertinentes, seja porque apresentam atributos de autonomia, como, por exemplo, os sistemas autônomos, estabilidade, como o sistema ultraestável de Ashby, seja porque apresentam atributos de mobilidade e distribuição, como é o caso de agentes móveis, seja ainda porque apresentam atributos de emergência e coletividade, como a inteligência de enxames e algoritmos de formigas. Como todos estes trabalhos têm em comum inspiração biológica, esta parte é apresentada inicialmente seguida de outros trabalhos relacionados.

2.1 Inspiração Biológica

Os sistemas computacionais atuais são muito complexos e frágeis [6]. Muitos dos sistemas atuais, apesar de proverem muitas funcionalidades aos usuários, também estão sujeitos a falhas com efeitos catastróficos, dificuldades de manutenção e repletos de vulnerabilidades a ataques externos [7]. Um objetivo importante da computação é ser capaz de construir sistemas que possam funcionar com altos níveis de autonomia, gerenciar enormes quantidades de dados de forma robusta, configurar a si mesmos automaticamente pelas redes de computadores, reconfigurar a si mesmos quando partes estão danificadas ou destruídas, processar rapidamente grandes quantidades de dados de uma forma maciçamente paralela, aprender a partir de seu ambiente com o mínimo de intervenção humana, e “evoluir” para se adaptar melhor para a tarefa na qual foi projetado [7].

Não há dúvida que tais sistemas computacionais com essas propriedades são muito desejáveis [7]. Apesar do desenvolvimento de tais sistemas ser uma área ativa de pesquisa em ciência da computação, a própria Internet é um exemplo de um sistema que é capaz de operar sem uma autoridade central e se reconfigurar quando partes estão danificadas ou destruídas, os cientistas continuam pesquisando novos sistemas, novos tipos de hardware, software, algoritmos e vêm buscando fontes de inspiração mais amplas dos que as existentes na ciência da computação atual. Dentro deste

enfoque de abertura científica, algo inteiramente diferente e desconhecido pode se tornar atraente.

Uma possível área de pesquisa está focada em um conjunto de técnicas inspiradas nas ciências biológicas, porque organismos biológicos frequentemente exibem propriedades desejáveis em sistemas computacionais. Eles funcionam com grande nível de autonomia. Algumas entidades biológicas como, por exemplo, os neurônios do cérebro, podem se configurar automaticamente em redes (e se reconfigurar em algum nível quando partes estão danificadas ou destruídas). Sistemas sensoriais rapidamente descobrem características relevantes em um amontoado de dados. Muitos animais se adaptam ao ambiente que os cerca para desempenhar melhor seu papel na natureza. Muitos organismos biológicos apresentam mecanismos para autorrecuperação e todos os organismos multicelulares crescem a partir de um estado inicial que é menos complexo que seus estados finais.

Os sistemas biológicos inspiram áreas de pesquisa em sistemas computacionais há muitos anos. Inteligência Artificial [11], Inteligência Artificial Distribuída [91], Redes Neurais Artificiais, Algoritmos Genéticos [13], Algoritmos de Formigas [15] (*Ant Colony Optimization*), Sistema Imunológico Artificial [10], Neurociêcia Computacional [14], Inteligência de Enxames [8] (S*warm Intelligence*), Computação Amorfa [12], Vida Artificial [89] e Inteligência Coletiva são algumas dessas áreas de estudo [17].

2.2 Sistemas Autônomos

Um sistema pode ser considerado autônomo se consegue atingir seus objetivos sem a intervenção humana [18]. Um sistema autônomo é um sistema que pode reagir de forma inteligente e flexível em condições de mudanças operacionais e demandas de processos que estão a sua volta [19].

Pode ser feita uma analogia com os paradigmas de orientação a objetos e sistemas baseados em componentes, em que para uma requisição feita para um objeto ou um componente é esperada uma reação (comportamento) compatível. Em um sistema autônomo esta requisição é enviada para um componente autônomo que age ou responde de forma apropriada seja com uma ação ou uma informação. Autônomo significa que apenas o objetivo é definido, sendo que a solução é obtida como uma autorresposta do sistema.

Para ilustrar o significado de ação autônoma, pode ser citado o exemplo de um robô que está na Lua e é telecontrolado da Terra e que tem como objetivo chutar uma bola. Em um processo convencional o robô usaria seus sensores para localizar a bola, e enviaria as informações para a Terra, esperando uma mensagem de retorno sobre qual ação tomar.

Devido ao atraso na comunicação as condições do ambiente podem ter se alterado e a ação comandada naquele momento pode não ser mais a ideal. A alternativa é de dotar o robô de autonomia para realizar a tarefa especificada em alto nível através de ações concretas de baixo nível, por ele mesmo.

Ou seja, as ações de mais alto nível continuam sendo realizadas pelos operadores, ao passo que as ações simples ou de baixo nível são definidas e tomadas pelo robô em tempo de execução.

Um sistema autônomo pode ser formado por subsistemas também autônomos. Se o objetivo e a ação resultante do sistema autônomo servir para controlar um processo, este sistema é chamado de um sistema de controle autônomo. Se ele servir para o desenvolvimento de informação de alto nível, este sistema é chamado de sistema de informações autônomo. Um exemplo é um componente para supervisionar um processo ou um sistema, gerando informação de diagnóstico como resultado, que é disponibilizada tanto para o operador do processo quanto para outros componentes autônomos. Além da auto-organização permite também a autodiagnose, que possibilita gerar informações sobre a possibilidade de sucesso em se alcançar a meta desejada.

A vantagem desta arquitetura é que ela permite que o sistema opere durante muito tempo sem a necessidade de intervenção humana. Especialmente em situações extraordinárias ou de mudanças rápidas, o sistema age de forma adequada sem requerer intervenção externa.

Sistemas autônomos podem ser utilizados em sistemas complexos que demandam ações flexíveis e de alto nível, que precisam ser tomadas apesar de se lidar muitas vezes com a incerteza e pouco conhecimento prévio.

O conhecimento incerto está fortemente ligado à área de inteligência artificial e inteligência computacional. Habilidades inteligentes, como aprendizagem pela experiência, planejamento de ações ou deteção e identificação de erros, usadas para serem analisadas pelo operador do processo, são cada vez mais integradas em sistemas autônomos. As vantagens são evitar erros de operação, melhorar o tempo de resposta e aliviar o operador.

2.3 Computação Autonômica

O termo Computação Autonômica (*Autonomic Computing*) foi cunhado em 2001 pela IBM para designar sistemas capazes de se manter de forma autônoma, tendo capacidade semelhante a sistemas de agentes inteligentes, onde autonomia e iniciativa própria são características importantes.

A computação autonômica é uma abordagem global e estratégica para projetar sistemas computacionais distribuídos complexos. Global no sentido de que “enxerga” todas as partes que compõem um sistema formando um todo, sendo que as partes devem ser coordenadas estrategicamente de forma a que este todo cumpra seu objetivo. Esta técnica é inspirada no funcionamento do sistema nervoso autônomo humano e tem como meta o projeto e a construção de sistemas que são autogerenciados. De forma mais específica, um sistema autonômico é um ambiente computacional autogerenciado, autônomo e onipresente que “esconde” sua complexidade, provendo o usuário com uma interface que satisfaz perfeitamente suas necessidades. O sistema sempre decidirá por conta própria o que deve ser feito para mantê-lo estável, porém guiado por seres humanos, com políticas em alto nível de abstração. O sistema estará sempre monitorando e otimizando seu próprio estado, e automaticamente se adaptando para mudanças de condições, sejam estas internas, ou externas.

O funcionamento do sistema nervoso autônomo é apresentado a seguir, mais especificamente dos subsistemas simpático e parassimpático que atuam de forma antagônica. Em seguida são apresentadas as características da computação autonômica através de suas propriedades auto* e sua arquitetura.

2.3.1 Sistema Nervoso Autônomo

O sistema nervoso humano está dividido em Sistema Nervoso Periférico (SNP) e Sistema Nervoso Central (SNC). O SNP é composto por neurônios sensoriais (sensores) e motores (atuadores). Os neurônios sensoriais são ativados por receptores de estímulos que fazem chegar esta informação ao SNC. Por sua vez os neurônios motores estão ligados por feixes de nervos aos músculos e glândulas e executam a ação ditada pelo SNC. O SNC está dividido em duas partes: o sistema nervoso somático-sensorial e o sistema nervoso autônomo.

O sistema nervoso somático-sensorial controla os órgãos de forma voluntária, principalmente os músculos. O Sistema Nervoso Autônomo (SNA) controla a função dos órgãos individualmente e a homeostase (equilíbrio), sendo também conhecido como sistema automático [20,21].

Na maior parte das vezes, não percebemos o funcionamento do SNA, pois ele funciona de uma maneira involuntária e reflexiva. Por exemplo, não notamos quando nossos vasos sanguíneos se contraem ou dilatam, ou quando nosso coração bate um pouco mais rápido ou um pouco mais devagar [22].

O SNA está sempre ativo trabalhando em conjunto com o sistema nervoso somático.

Existem duas situações opostas que servem para caracterizar seu funcionamento:

a) Em emergências que causam estresse e necessitam de reações de fuga ou luta, e;
b) Em situações de conforto (não emergenciais) que permitem o descanso e a digestão.

A primeira situação é suportada pela parte do SNA chamada de sistema nervoso simpático, enquanto o sistema nervoso parassimpático é responsável pela segunda. A Figura 2.1 apresenta um esquema com os dois subsistemas que compõem o SNA.

a) Sistema Nervoso Simpático

Imagine o seguinte cenário: o dia está bonito e ensolarado. Você está passeando tranquilamente no parque quando, de repente, aparece um ladrão pronto para lhe assaltar.

Você fica e luta ou se vira e foge? Estas são reações de fuga ou luta. Nestas situações, o sistema nervoso simpático entra em ação, ele usa energia do corpo, sua pressão sanguínea aumenta, seu coração dispara e sua digestão desacelera.

O sistema nervoso simpático é predominantemente responsável pelo controle da frequência e força das contrações cardíacas. Dor, ansiedade, medo e ferimento involuntariamente aumentam a estimulação simpática para a aceleração cardíaca.

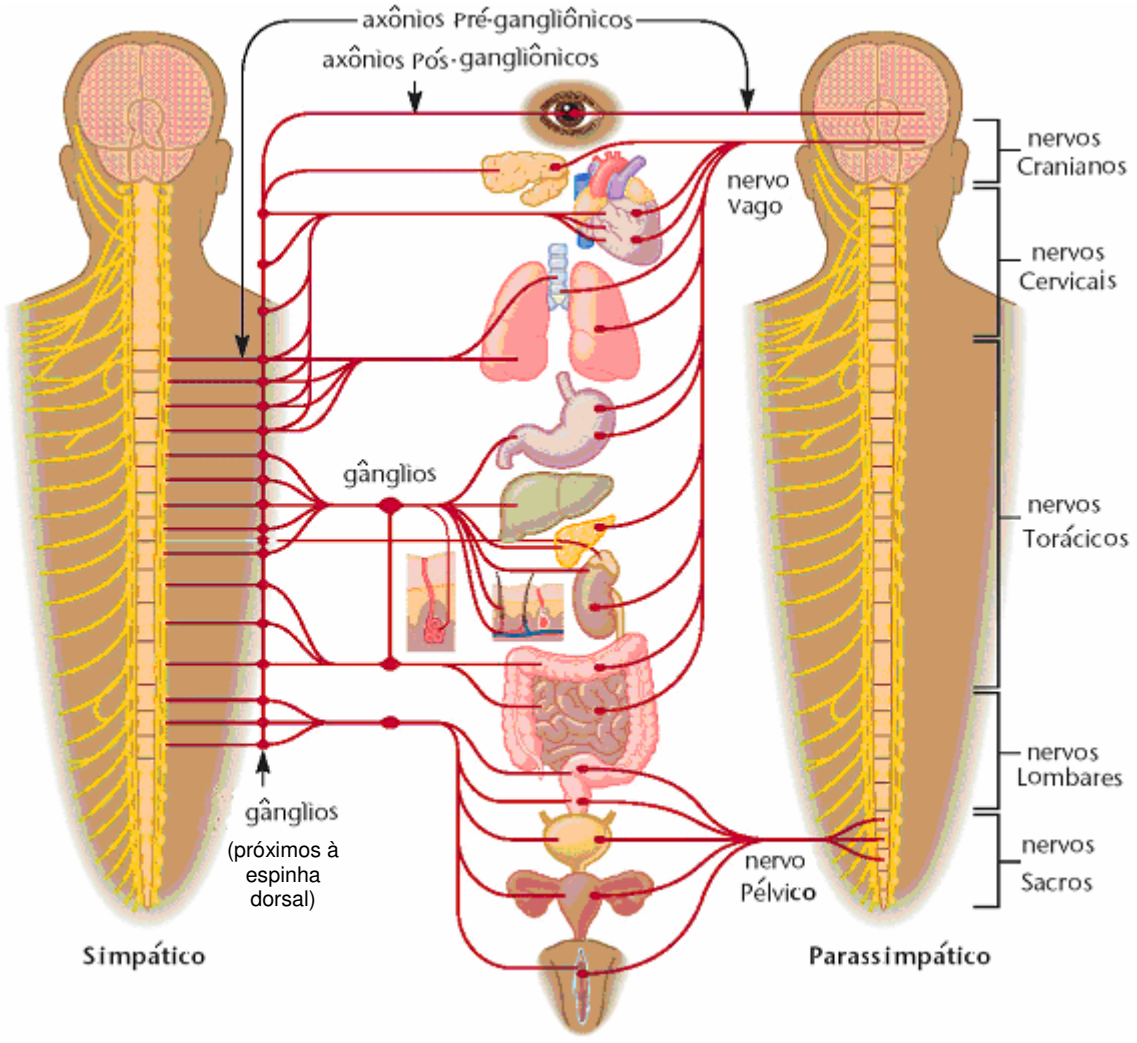

Figura 2.1 – Sistema Nervoso Autônomo. Fonte: [23]

O sistema nervoso simpático também é responsável pelo controle automático da temperatura do corpo. Quando a temperatura do ambiente aumenta em um dia quente de verão, receptores térmicos na pele estimulam os centros de controle simpáticos do cérebro, resultando na dilatação dos vasos sanguíneos cutâneos, aumentando o fluxo de sangue na superfície do corpo de onde o calor é dissipado por irradiação. A dilatação dos vasos sanguíneos, desta forma, faz com que pressão sanguínea diminua. O sistema nervoso simpático também responde ao calor do ambiente estimulando as glândulas sudoríparas, resultando no suor [21].

b) Sistema Nervoso Parassimpático

Imagine agora um segundo cenário: o dia está bonito e ensolarado. Você está passeando tranquilamente no parque. Desta vez, entretanto, você decide descansar confortavelmente em uma cadeira e comer um lanche. Esta situação faz com que a resposta de repouso e digestão seja ativada. Neste momento o sistema nervoso parassimpático entra em ação para poupar energia, a pressão sanguínea do corpo diminui, o coração bate mais devagar e a digestão pode começar.

Quando um estímulo chega a um órgão, como, por exemplo, a luz brilhante alcança nossos olhos, esta mensagem é conduzida através de fibras sensoriais até o cérebro, que por sua vez estimula as fibras parassimpáticas dos nervos das pupilas, resultando na contração automática dos músculos pupilares que diminuem sua abertura, reduzindo a quantidade de luz que chega às células sensitivas na retina dos olhos. Analogamente, os estímulos associados à entrada de comida em nosso estômago são sentidos pelas fibras do nervo vago e são transmitidas para o cérebro que responde, através do mesmo nervo vago de volta para o estômago. Isto estimula a secreção de sucos gástricos e contrações peristálticas do estômago para misturar a comida aos sucos digestivos secretados e gradualmente levar o conteúdo gástrico para os intestinos, onde um processo similar é iniciado através essencialmente do mesmo caminho de nervos parassimpático. Felizmente, o esvaziamento do reto e da bexiga não é inteiramente automático, mas sujeito aos impulsos parassimpáticos que são controlados voluntariamente [21].

2.3.2 Propriedades auto*

Um sistema autonômico alcança o autogerenciamento através de aspectos chave como autoconfiguração, autorrecuperação, auto-otimização e autoproteção [24]. Estes aspectos são chamados de propriedades auto* ou autopropriedades. Estas propriedades têm que estar presentes de forma harmoniosa para permitir o bom funcionamento do sistema como um todo.

A propriedade de autoconfiguração se refere à capacidade do sistema em se adaptar automaticamente às mudanças do ambiente. Os sistemas autoconfiguráveis se adaptam seguindo as políticas ditadas pelos especialistas do sistema. Havendo reconfiguração de parte do sistema, o restante se ajusta automaticamente.

A propriedade de auto-otimização se refere à capacidade dos sistemas de procurarem melhorar continuamente. Recursos podem ser adicionados ou retirados dinamicamente para melhorar o desempenho e a eficiência do sistema.

A propriedade de autoproteção é a capacidade que um sistema tem de antecipar, detectar, identificar e se proteger de ataques mal intencionados externos (fatores extrínsecos). Esta propriedade possibilita, portanto, detectar comportamentos hostis e de tomar as medidas necessárias para conter acessos não autorizados, infecção por vírus, etc., permitindo a melhoria das políticas de segurança.

A propriedade de autorrecuperação é a capacidade que um sistema tem de antecipar, detectar, identificar e se proteger do mau funcionamento do sistema devido a fatores internos (fatores intrínsecos), evitando, por exemplo, efeitos em cascata.

2.3.3 Arquitetura

A arquitetura de computação autonômica da IBM provê um *framework* no qual podem ser construídos sistemas autogerenciáveis [25]. Na Figura 2.2 são mostrados os componentes e as principais interações entre eles para um único gerenciador autonômico e um único recurso gerenciado por ele. O recurso, também chamado de elemento gerenciado, é o que está sendo transformado em um elemento mais

autogerenciável. Este pode ser um único sistema (ou mesmo uma aplicação dentro de um sistema), ou pode ser uma coleção de vários sistemas logicamente relacionados. Os sensores provêm uma maneira de se obter dados de medição do recurso, e os atuadores provêm uma forma de modificar o comportamento do recurso. Os gerenciadores autonômicos leem os dados dos sensores e manipulam os atuadores para fazer com que os recursos se tornem mais autogerenciáveis. O gerenciador autonômico contém componentes para monitorar, analisar, planejar e executar (arquitetura MAPE), e comum a todos eles estão os seguintes itens: o conhecimento do ambiente computacional, os acordos de nível de serviço e outras considerações relacionadas. O conjunto formado pelo elemento gerenciador (gerenciador autonômico) e pelo elemento gerenciado (recurso) é denominado elemento autonômico.

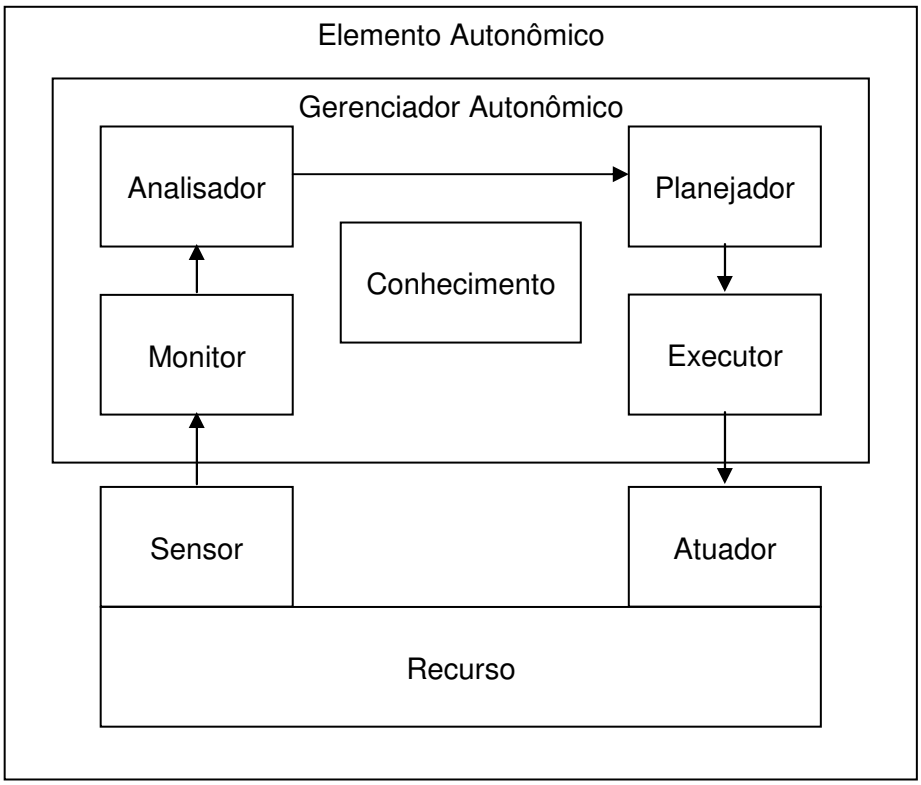

Figura 2.2 – Gerenciador Autonômico e o Recurso Gerenciado formando o Elemento Autonômico. Fonte: [25].

O componente de monitoração (monitor) filtra e faz a correlação com os dados lidos pelo sensor. O componente de análise (analisador) processa estes dados refinados para fazer previsões e determinação de problemas, entre outras atividades. O

planejamento (planejador) constrói um fluxo de informação que especifica uma ordem parcial de ações para se alcançar a meta especificada pelo componente de análise. O componente de execução (executor) controla a execução deste fluxo de informação e coordena caso existam múltiplos fluxos de informação concorrentes. O termo "execução" pode ser ampliado para incluir também as ações manuais. Em essência a arquitetura da computação autonômica descreve os laços de controle de realimentação para sistemas de autogerenciamento.

O modelo genérico de um *loop* de controle com realimentação para computação autonômica [26] está representado na Figura 2.3. Ele é composto por três passos:

a) Monitoramento: medição (através dos sensores S) dos parâmetros do sistema controlado que são relevantes para a função controlada (por exemplo, carga corrente para otimização de desempenho, disponibilidade do servidor para tolerância a falhas, etc.);
b) Decisão (D): decidindo a ação corretiva a ser tomada. Isto por sua vez pode ser feito em dois passos: análise e planejamento;
c) Execução: executando efetivamente (através dos atuadores A) as ações no sistema controlado.

Para atingir estes objetivos, o sistema possui uma base de conhecimento (C), representando a informação do sistema controlado.

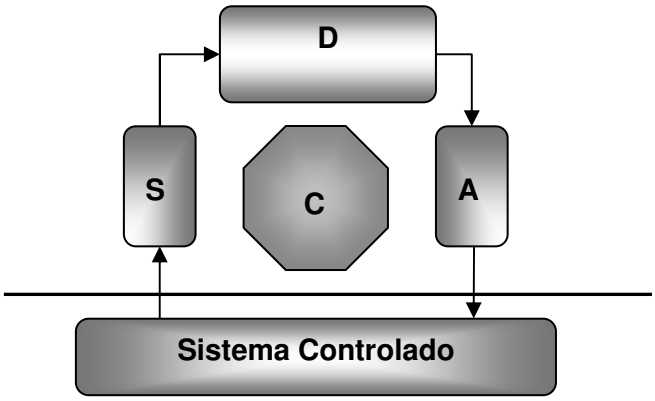

Figura 2.3 – Modelo de Computação Autonômica.

2.4 Sistema Ultraestável de Ashby

Qualquer máquina real incorpora um grande número de variáveis [26]. O sistema (corpo) humano também pode ser representado por um conjunto similar de variáveis, das quais podemos considerar algumas: as essenciais. Para um organismo vivo sobreviver, suas variáveis essenciais precisam ser mantidas entre limites viáveis (Figura 2.4). Caso contrário o organismo tem a possibilidade de desintegração e/ou perda de identidade (isto é, dissolução ou morte).

Os mecanismos internos do corpo trabalham continuamente e em conjunto para manter suas variáveis essenciais dentro de seu limite viável. A definição de Ashby para comportamento adaptativo deriva da observação do corpo humano. Ele afirma que uma forma de comportamento é adaptativa se ela mantém as variáveis essenciais dentro dos limites fisiológicos. Isto define a zona viável.

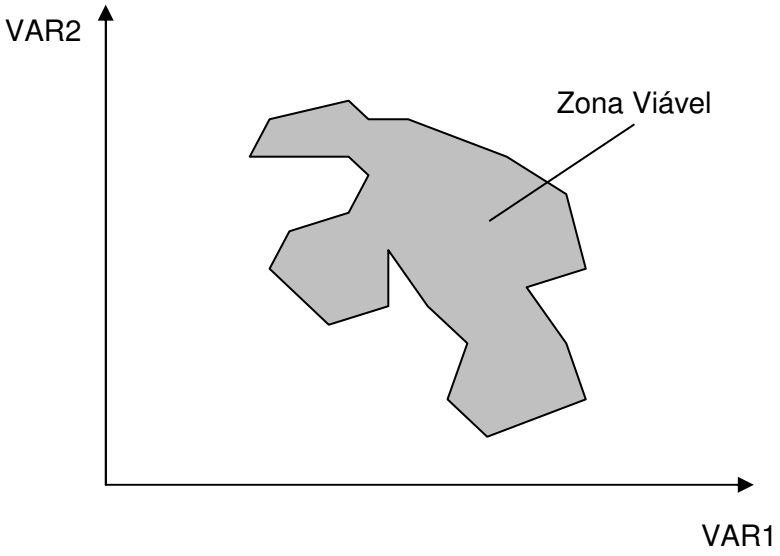

Figura 2.4 – Variáveis Essenciais. Fonte: [26].

Duas importantes observações podem ser feitas:

a) O objetivo do comportamento adaptativo está diretamente ligado com a sobrevivência do sistema;
b) Se os ambientes externo e interno colocam o sistema fora de seu estado de equilíbrio fisiológico, o sistema sempre trabalhará no sentido de retornar ao estado de equilíbrio original.

Ashby observou que muitos organismos sofrem dois tipos de perturbações: (1) pequenos impulsos frequentes nas variáveis principais e (2) grandes mudanças ocasionais em seus parâmetros.

Baseado nesta observação, ele definiu a arquitetura de sistemas ultraestáveis que são constituídos por dois *loops* fechados (Figura 2.5): um *loop* controla as pequenas perturbações e um segundo *loop* é responsável pelas perturbações com maior duração.

Como pode ser observado na Figura 2.5, o sistema ultraestável é composto por três partes: (1) o ambiente, (2) a parte reativa *R* e (3) o mecanismo de passo ou parte *S*. A parte *R* representa o subsistema do organismo que é responsável pelo comportamento ou a percepção. Ele usa os canais de sensores como parte de sua capacidade de percepção e canais motores para responder a mudanças provocadas pelo ambiente.

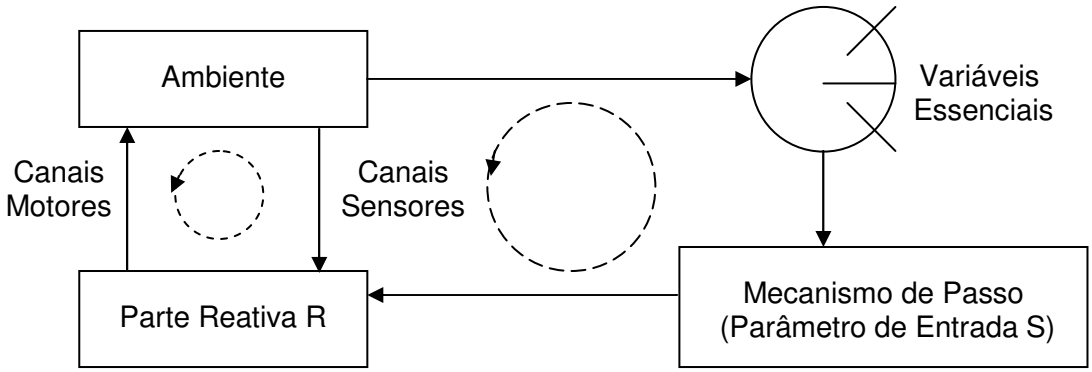

Figura 2.5 – Dois *loops* de controle do Sistema Ultraestável de Ashby. Fonte: [26].

Este conjunto de sensores e canais motores compõe a primeira realimentação entre *R* e o ambiente. A parte *R* pode ser considerada como sendo um conjunto de comportamentos do organismo, que é disparado baseado nas mudanças provocadas pelo ambiente. A parte *S* representa o conjunto de parâmetros que dispara as mudanças em características relevantes do conjunto de comportamentos. Observe que a parte *S* somente dispara mudanças quando o ambiente afeta as variáveis essenciais de uma maneira que faça com que elas saiam dos limites fisiológicos. Como mencionado anteriormente, estas variáveis precisam ser mantidas dentro dos limites fisiológicos para que qualquer organismo ou sistema adaptativo sobreviva. Assim, a segunda realimentação entre o ambiente e a parte *R* é responsável pelo disparo do comportamento adaptativo do organismo. Quando as mudanças

provocadas pelo ambiente no organismo são grandes o suficiente para colocar as variáveis essenciais fora de seus limites fisiológicos, a segunda realimentação fica ativa e modifica o conjunto de comportamentos existentes no sistema para fazê-lo se adaptar a essas novas mudanças.

Note que quaisquer modificações no ambiente tendem a passar um sistema, anteriormente em um estado estável, para um estado instável. O objetivo do sistema como um todo é manter seus subsistemas (o ambiente e a parte *R*) em um estado de equilíbrio estável. A primeira realimentação controla as pequenas modificações no ambiente com o conjunto de comportamentos existentes de forma a trazer o sistema como um todo para um equilíbrio estável. A segunda realimentação controla mudanças maiores e de longo prazo no ambiente através da modificação do conjunto de comportamentos existentes originalmente e eventualmente trazem de volta para o estado de equilíbrio o sistema todo. Desta forma, o ambiente e o organismo sempre estão em um estado de equilíbrio estável e qualquer atividade do organismo dispara um mecanismo para manter este equilíbrio.

2.5 Agentes Móveis

Agentes móveis são agentes de *software* capazes de se autotransmitir – seu código e seu estado – através de uma rede de computadores e recomeçar sua execução em um computador (*site*) remoto [27].

A principal motivação por trás dos agentes móveis é simples: levar o código a ser executado (processamento) para perto dos dados e, com isso, reduzir o tráfego de mensagens através da rede.

As sete principais razões para utilização de agentes móveis, segundo LANGE *et al.* [28], são as seguintes:

1) Redução da carga da rede;
2) Melhoria da latência da rede;
3) Encapsulamento de protocolos;
4) Execução assíncrona e autônoma;
5) Adaptação dinâmica;
6) Natural heterogeneidade;
7) Robustez e tolerância a falhas.

O modelo computacional, baseado em mobilidade, pode ser visto como uma forma de substituição, refinamento ou extensão do tradicional paradigma cliente/servidor. Com a adoção deste modelo é esperado, portanto: a melhoria da latência e banda passante de aplicações cliente-servidor e a diminuição da vulnerabilidade devido à desconexão da rede.

O modelo de agente móvel é um modelo geral que serve para expressar e desenvolver aplicações distribuídas. Seu alto nível de abstração, escondendo detalhes de comunicação do desenvolvedor, faz com que o sistema seja melhor compreendido e usado se comparado a modelos de mais baixo nível, baseados em troca de mensagens, RPC (*remote procedure call*) ou outras combinações de modelos tradicionais. Isto é importante principalmente na construção de aplicações distribuídas compostas por vários componentes interconectados.

A principal motivação para a migração de agentes, que é acessar e processar dados localmente em prol de um recurso ou usuário remoto, levanta, por outro lado, questões de segurança e eficiência.

A segurança é a principal questão levantada para justificar a utilização incipiente de agentes móveis em aplicações práticas. Os problemas de segurança podem ser causados tanto pelo agente móvel que foi recebido por um computador remoto e tem acesso a seus recursos, podendo utilizá-los de forma indevida, quanto por este ambiente remoto tentando violar o agente móvel recém chegado. Por isso, nesta tese, o escopo de utilização das aplicações propostas fica limitado à *intranet*, pois neste ambiente a segurança já é garantida por *firewalls* e antivírus. A simulação de ataques ou mau funcionamento é feito por agentes maliciosos ou alteração de comportamento de agentes que fazem parte do próprio sistema.

A seguir são apresentados aspectos importantes sobre geração de agentes de fruto de pesquisa em publicações que tratam de população de agentes.

2.5.1 Geração de Agentes

SUZUKI, IZUMI, OOSHITA *et al*. [72] apresentam um mecanismo de autoadaptação de uma população de agentes móveis distribuídos em uma rede usando uma

abordagem inspirada biologicamente. A inspiração biológica é a do modelo populacional de uma única espécie que tem sua população controlada pela quantidade de alimento presente no ecossistema. Neste modelo é proposta a resolução do problema do controle populacional de agentes móveis em redes dinâmicas. O objetivo é ajustar a população de agentes para uma fração da população original em um tempo *t*. São apresentadas duas soluções para o problema: a primeira depende do conhecimento *a priori* do número total de nós da rede, enquanto que a segunda independe deste número. Além disso, cuida para evitar a extinção dos agentes pela criação de novos agentes em nós pouco visitados. A movimentação dos agentes pelos nós é aleatória. Se em um nó acabar o alimento necessário para a sobrevivência de um agente, ele morre (se mata), ao passo que se sobrar muito alimento há a duplicação de agentes.

AMIN e MIKLER [73] descrevem um controle adaptativo do número de agentes de uma rede usando feromônios aplicado a roteamento de redes. Eles observaram que a maioria das aplicações baseadas em agentes assume uma população de agentes estática. Alguns sistemas implementados usando colônias de insetos criam e eliminam agentes de forma periódica. Uma grande população de agentes pode acelerar a convergência de um processo, mas ao custo de aumentar o *overhead* de recursos do sistema. Mudar dinamicamente a população de agentes em resposta a seu ambiente não é uma tarefa trivial na ausência de um controlador central. É proposta uma coordenação entre agentes usando feromônios, ou seja, a partir de informação local o agente executa ações (comportamento individual) que fazem emergir um comportamento global desejado do sistema (convergência). Feromônios chamados nodais ajudam agentes no controle populacional. A intensidade dos feromônios nodais é expresso pela equação $e^{-\lambda(\Delta t)}$ onde λ representa o grau de volatilidade do feromônio e Δt é o intervalo de tempo decorrido desde seu depósito. Se um agente visitando um nó percebe que a intensidade do feromônio depositado está acima de um *limite de terminação* (ψ), mas não houve atualização da tabela de roteamento, ele se mata. Esta situação caracteriza uma superpopulação de agentes. Se um agente visitando um nó percebe que a intensidade do feromônio depositado está abaixo de um *limite de clonagem* (Ω), ele se clona. Esta situação caracteriza uma população de agentes insuficiente. Não há dependência do número inicial de agentes.

BAKHOUYA e GABER [74] apresentam um sistema de agentes autônomos adaptativos móveis com controle populacional inspirado no sistema imunológico. Ele é constituído por uma cadeia de estimulação e supressão de antígenos que faz com que

o número total de agentes convirja para um número que independe do número inicial de agentes. Os elos da cadeia são os comportamentos: clonar, mover e morrer. Cada elo está associado a um antígeno. Ou seja, os agentes percebem a concentração de antígenos e dependendo da maior resultante se clonam, se movem ou se matam. Como esta cadeia é fechada, existe uma realimentação que acaba fazendo com que o sistema entre em equilíbrio.

LINDEN [13], em seu livro sobre algoritmos genéticos, apresenta, no capítulo 4.6, módulo de população em que são discutidos aspectos de controle de população, como por exemplo, os modelos de população constante. Um dos mecanismos de controle é para cada indivíduo que nasce um morre. Outro mecanismo é para cada par de pais são gerados dois filhos. Quando o número de filhos gerados se iguala à população, esta nova geração substitui a dos pais. No capítulo 7 do mesmo livro, outros módulos de população são tratados onde a geração e a morte de indivíduos não são simplesmente fixas ou aleatórias, mas dependem de características do indivíduo. No item 7.1 são abordados aspectos como o tamanho variável da população em que a expectativa de vida para cada indivíduo é proporcional à qualidade do indivíduo. No item 7.2 o elitismo é tratado, ou seja, os melhores indivíduos de cada geração não devem "morrer" junto com sua geração. No item 7.3 é abordada a ideia que na maioria dos algoritmos genéticos toda geração nasce ao mesmo tempo e morre de uma só vez. No "mundo real" os indivíduos nascem aos poucos e morrem por velhice, doença, acidentes. É sugerida a estratégia *steady state*, na qual indivíduos são criados aos poucos e os "piores" pais são substituídos por estes novos indivíduos. No item 7.4 é citada a estratégia μ+λ, na qual o indivíduo atual só é escolhido para morrer ou ser substituído quando ele for suficientemente diferente do melhor indivíduo (distância de *hamming*). No item 7.5 são tratadas populações de tamanho variável, sendo citado exemplo de problema em que o tamanho da população varia com o tempo e cada indivíduo só morre quando sua idade for igual ao seu tempo de vida.

É desejável que o algoritmo (mecanismo) de geração de agentes do CIACOM também apresente como característica a independência do número inicial de agentes célula do sangue, ou seja, que o número de agentes iniciais aumente ou diminua, convergindo para se adaptar ao número de células do corpo. Outra característica desejável é que esta população represente um percentual apenas do número de células do corpo, assim como do ajuste dinâmico da população de agentes caso aumente ou diminua o número total de células do corpo. Aplicação de técnicas de algoritmos genéticos é dificultada pela distribuição de agentes e dificuldade de avaliação de indicadores

(função *fitness*), mas não devem ser descartadas. Técnicas de inteligência coletiva devem ser exploradas para fazer emergir um número de agentes adequado ao CIACOM.

2.5.2 Distribuição de Agentes

TAHARA, OHSUGA e HONIDEN [75] apresentam uma gama de padrões para o desenvolvimento de agentes incluindo padrões para agentes móveis (*mobility pattern*) e para sua migração (*itinerary*, *branching* e *star-shaped*). LIMA, MACHADO *et al.* [76] apresentam além desses mais alguns padrões de distribuição para agentes móveis: *master-slave*, *meeting e facilitator*.

BAKHOUYA e GABER [74] preveem a movimentação do agente, para sua melhor distribuição pela rede, com controle inspirado no sistema imunológico, semelhante ao apresentado no subitem anterior de geração.

AMIN e MIKLER [73] apresentam as estratégias de migração *Random Walk* e *Least Recently Used* (LRU). Esta última se caracteriza pelos agentes executarem uma busca em profundidade na rede baseada na informação carregada por eles. Como a partir da troca de informação entre agentes, as informações passam a ser similares, acaba emergindo uma concentração de agentes em áreas da rede. Para solucionar este problema foi proposta estratégia inspirada no depósito de feromônios. Estes depósitos, chamados de *feromônios de ligação,* servem para orientar os agentes para seguir os caminhos menos visitados, evitando a concentração. Ou seja, o agente é repelido pela maior concentração deste feromônio.

2.6 Outros Trabalhos Relacionados

Existem outros projetos relacionados ao gerenciamento de sistemas complexos que merecem ser citados. Como, por exemplo, o projeto da NASA inspirado no genoma humano [30] cujo objetivo é criar uma arquitetura multiagente que proveja um ambiente computacional que se adapte a mudanças inesperadas no hardware, através da reconfiguração do novo hardware, sem perda de funcionalidade, mas com provável perda de desempenho. A comunicação celular a partir de sinais químicos apresentada no projeto da NASA é semelhante à apresentada no modelo proposto no Capítulo 3.

A IBM, além de lançar a computação autonômica, apresentou um projeto de um sistema multiagente para sua implementação. Trata-se da arquitetura descentralizada para computação autonômica *Unity* [31,32] baseada em múltiplos agentes interativos, chamados de elementos autonômicos.

PANIT e LUKE [33] exploraram algoritmos para modelar o mecanismo de comunicação com feromônios usado pela colônia de formigas (formigueiro). Apesar de não modelar infraestruturas, a comunicação indireta apresentada está relacionada à inteligência de enxames.

HELSINGER, KLEINMAN e BRINN [34] abordam um *framework* para controlar a "sobrevivência" de sistemas multiagente. O objetivo é manter o funcionamento de sistemas multiagente distribuídos mesmo na ocorrência de adversidades. A correlação das informações monitoradas para seleção das ações de controle são contribuições importantes deste projeto.

O projeto ABLE [35] (*Agent Building and Learning Environment*), desenvolvido pela IBM, é um ambiente de desenvolvimento para construção de agentes inteligentes autonômicos e sistemas multiagente. Novas funcionalidades que permitem a construção de sistemas multiagentes autonômicos são apresentadas.

O gerenciamento inteligente das infraestruturas do século XXI [37], como é o caso das infraestruturas de transporte, bem como das chamadas infraestruturas inteligentes (telégrafo, sistemas de supervisão e controle, controle de tráfego aéreo e sistemas de sinalização telefônicos), se mostra necessário para que estas infraestruturas alcancem seu potencial máximo. Como exemplo, pode se citada a iniciativa do governo do Reino Unido para controlar estas infraestruturas [41] e da Holanda com uma nova geração de infraestruturas por meio de uma fundação homônima [42,43]. A tecnologia de multiagentes é apontada como uma das ferramentas fundamentais para a modelagem das infraestruturas a serem gerenciadas. Na Figura 2.6 são mostrados os seis princípios da infraestrutura inteligente: escalabilidade, segurança, interoperabilidade, disponibilidade, adaptabilidade e visibilidade.

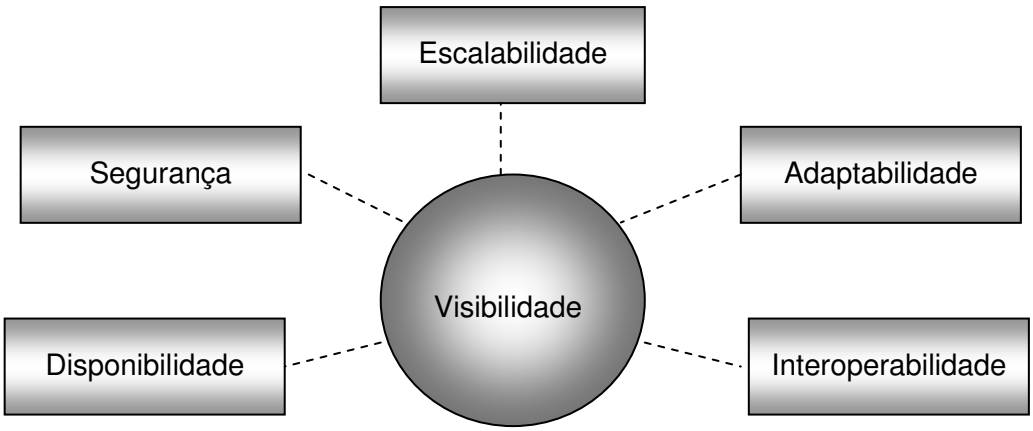

Figura 2.6 – Os seis princípios da Infraestrutura Inteligente. Fonte: [37].

FERGUSON e CHARRINGTON [44] mostram como construir uma estrutura inteligente de Tecnologia da Informação e citam a computação autonômica. Além disso, a computação autonômica é considerada como tendência entre as tecnologias emergentes [40]. Outro trabalho relacionado é a comunicação autonômica [38, 39] guiada pelos mesmos princípios de inspiração biológica da computação autonômica.

O gerenciamento por delegação também é um trabalho relacionado [45,46]. A ideia é utilizar *scripts* que possibilitem criar um sistema onde as tarefas de monitoração e de controle de um gerente de mais alto nível podem ser repassadas para subgerentes, responsáveis por uma área específica determinada pela proximidade com os dispositivos gerenciados.

Uma vez lançada a semente da Computação Autonômica pela IBM, outras empresas de computação [2] seguiram seu caminho como: a HP com sua Infraestrutura Adaptativa (*Adaptive Infrastructure*), a Fujitsu-Siemens com seus Sistemas Orgânicos (*Organic Systems*), a Microsoft com seus Sistemas Dinâmicos (*Dynamic Systems*), a Hitachi com sua Computação Harmoniosa (*Harmonious Computing*) e a INTEL com sua Computação Pró-ativa (*Proactive Computing*).

Em outros setores também existe interesse em desenvolver sistemas baseados no paradigma autonômico, como é o caso do setor elétrico [47] com o EPRI [48], de comunicações [49] e aeroespaciais com a NASA [18].

Outros trabalhos relacionados que tratam de computação autonômica podem ser citados [50,51,52,53].

PARASHAR e HARIRI [26] sugerem a divisão dos sistemas autonômicos em dois grupos: os dos sistemas que possuem propriedades autonômicas e os dos sistemas que permitem o desenvolvimento de aplicações autonômicas. Podem ser classificados no primeiro grupo os projetos: *OceanStore*, *Storage Tank*, *Oceano*, *Smart DB2*, *AutoAdmin*, *Sabio* e *Q-Fabric*. São classificados no segundo grupo os projetos: KX (*Kinesthetics eXtreme*), *Anthill*, *Astrolabe*, *Gryphon*, *Smart Grid*, *Autonomia* e *AutoMate*.

Derivada da computação autonômica, a computação autonômica pessoal (*Personal Autonomic Computing* ou PAC) [5] é a integração da computação pessoal (*Personal Computing*) com a computação autonômica. Outras metáforas biológicas [54,55,56] são a apoptose (certeza da morte), os batimentos cardíacos, a pulsação e a inteligência de enxames, que serviram para a prova de conceito de ferramentas PAC [57,58,59].

Alguns trabalhos são baseados em computação autonômica [60], sendo que em muitos deles há especialização em determinadas características autonômicas [61], enquanto que em outros, todas as características autonômicas estão presentes [62,63].

2.7 Considerações

Já existem sistemas autonômicos? A Internet pode ser considerada, em nível de rede, como um sistema autonômico, uma vez que, por exemplo, uma pessoa usando seu computador endereça uma mensagem a outro usuário, geograficamente distante ou não, tem o êxito deste envio (operação de alto nível) na maior parte das vezes, sem estar consciente da complexidade envolvida no gerenciamento da infraestrutura computacional e de comunicação necessária para o cumprimento desta “simples tarefa”.

O próprio Sistema Elétrico Nacional Interligado brasileiro pode ser considerado um sistema autonômico, uma vez que a energia elétrica que chega a nossas casas pode ser proveniente de uma região remota, sendo que só percebemos a existência do sistema quando falta luz. O mesmo vale para infraestrutura de água, comunicações telefônicas, televisão, etc. Alguns dos sistemas computacionais atuais já incorporam

uma série de características autonômicas. Por exemplo, os sistemas operacionais contêm módulos para fazer atualizações periódicas de softwares, os sistemas editores de texto têm módulos para fazer correção automática de ortografia, estilo etc. O preenchimento automático de campos no Windows Explorer, ou no ambiente de desenvolvimento (de software) integrado Eclipse, a complementação automática de classes JAVA, conforme são digitadas teclas de caracteres em sequência, podem ser considerados também como automatismos que antecipam e agem de forma pró-ativa em benefício do usuário para que este alcance seu objetivo. Mas, estes tipos de sistema estão mais próximos da iniciativa de assistentes pessoais do que de sistemas autonômicos. Por exemplo, enquanto teclamos usando o Microsoft Word, um personagem animado fica monitorando e caso perceba uma sequência de seleção de comandos pode recomendar alguma ação a ser tomada.

Além disso, existem pequenos sistemas, por exemplo, *softbots*, que espiam operações do usuário, tais como navegação *web* entre outras, tentando prever as próximas interações do usuário. Por exemplo, sites de livros tais como Amazon "lembram" de operações passadas para fazer sugestões aos usuários. Estas características estão presentes em muitos sistemas, não somente os computacionais, porém o objetivo é desenvolver um sistema autorregulador que integre estas características para gerenciamento de infraestruturas em geral. Por exemplo, numa empresa, o fluxo de informações e a manutenção da ordem geralmente são realizados de forma hierárquica pelos indivíduos que ocupam os cargos de chefia. Um sistema computacional inspirado em um sistema biológico poderia atuar nesta empresa, de forma que o sistema tomasse ciência de todas as informações trocadas entre as chefias, de forma a antever situações de conflito, estresse, incompetências, etc.

Sistemas autonômicos apresentam caraterísticas de sistemas de agentes inteligentes e multiagentes, acrescidas de características diretamente ligadas a sistemas biológicos, ou seja:

a) Devem ficar "ligados" (ou pelo menos em estado de vigília) 24 horas por dia;
b) Devem ser capazes de se autorrecuperar;
c) Devem ser capazes de chegar a alguma situação estável (equilíbrio);
d) Devem ser capazes de se autoproteger;
e) Devem ser capazes de se auto-otimizar;
f) Devem ser capazes de se autoconfigurar.

2.8 Conclusão

Neste capítulo foram apresentados os trabalhos relacionados a sistemas computacionais inspirados em sistemas biológicos, com ênfase naqueles relativos ao autogerenciamento de sistemas distribuídos.

Nos trabalhos sobre infraestrutura inteligente é mencionada a utilização de multiagentes (móveis) e de agentes inteligentes. A computação autonômica pessoal foi utilizada como alternativa para prova de conceito. Foram citadas também as ferramentas ABLE e JADE para a implementação de sistemas autonômicos.

Outros trabalhos relacionados, como inteligência de enxames, gerenciamento por delegação, colônia de formigas e outros, apresentam subsídios dos quais podem ser extraídos técnicas de inteligência computacional e algoritmos interessantes.

Nosso trabalho é inspirado em sistemas autonômicos, pois estes são dotados das propriedades auto* e já demonstraram sua utilidade em algumas tarefas de autogerenciamento. Porém, apesar de ser inspirada em sistemas biológicos, a computação autonômica está relacionada ao sistema nervoso de uma forma extremamente metafórica. Em nossa opinião, e isto é comprovado pela operacionalidade dos sistemas autonômicos, isto traz duas consequências: (1) os sistemas implementados, por muitas vezes, afastam-se do modelo ideal de origem biológica, o que, por sua vez, contribui para (2) que estes sistemas não abranjam todas as funcionalidades integradas do modelo biológico subjacente. Com isso, outras funcionalidades proporcionadas por outras inspirações biológicas acabam não sendo contempladas.

A computação autonômica pessoal [5], ou seja, a aplicação da computação autonômica ao ambiente de computação pessoal, traz mais metáforas biológicas (apoptose, batimento cardíaco, pulsação cardíaca e inteligência de enxames) além das do sistema nervoso autônomo. Este exemplo já justifica a necessidade de integração e utilização de outras inspirações biológicas. Para preencher esta lacuna é apresentado, no próximo capítulo, o modelo de computação circulatório-autonômico que está mais próximo de um sistema biológico, sendo menos metafórico e incorporando mais inspirações biológicas, além de integrar as propriedades auto*.

3 Um Modelo Computacional Inspirado no Sistema Circulatório Humano

Neste capítulo é apresentado um modelo de autogerenciamento inspirado no sistema circulatório humano. Chamaremos este modelo de "CIACOM", do inglês CIrculatory-Autonomic COmputing Model [64,65,66,67].

A partir da descrição do sistema circulatório humano (Anexo A) foi utilizada a metáfora da circulação sanguínea humana (Computação Circulatória) como extensão à metáfora do sistema nervoso autônomo (Computação Autonômica), para modelagem do autogerenciamento de sistemas computacionais (computação circulatório-autonômica).

Foi feita uma analogia entre elementos biológicos e computacionais e foram descritas as características, as propriedades e a interação dos elementos do modelo.

3.1 Modelo Curculatório-Autonômico

O principal modelo biológico no qual o CIACOM foi inspirado é o sistema circulatório, como ilustrado na Figura 3.1. Neste modelo biológico as células sanguíneas (plaquetas, glóbulos vermelhos e brancos) são geradas de forma centralizada no baço e de forma distribuída na medula óssea. Uma vez na corrente sanguínea, elas são bombeadas pelo coração para todas as partes do corpo (sangue arterial), chegando a quase todas as células (exceto os neurônios). Estas células desempenham papel específico em prol de seu "usuário": as plaquetas recuperam os tecidos, os glóbulos brancos protegem o corpo contra ataques de vírus e os glóbulos vermelhos abastecem as células de recursos (oxigênio e nutrientes). As células sanguíneas, portanto, são móveis, atendem às necessidades das células do corpo e mantêm uma relação de muitos para um (n:1), ou seja, muitas células do corpo podem ser atendidas por uma única célula sanguínea. Quando retornam ao coração (sangue venoso) são bombeadas para o pulmão, onde os glóbulos vermelhos são reabastecidos de oxigênio e nutrientes.

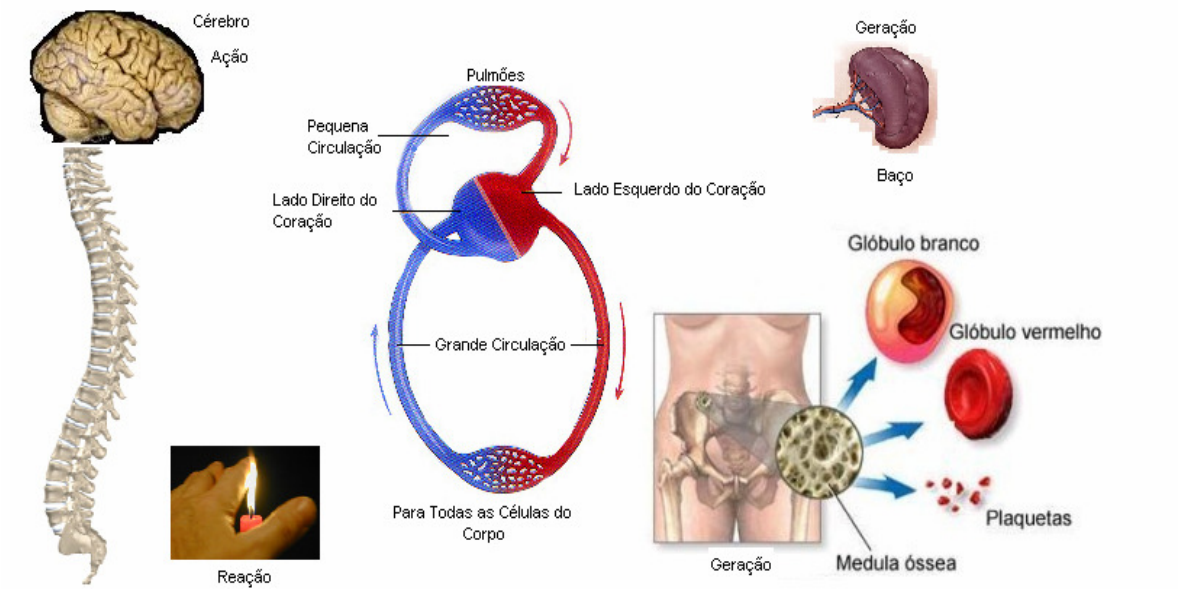

Figura 3.1 - Modelo Biológico.

Outra inspiração biológica característica do modelo é a apoptose: o glóbulo vermelho tem um tempo de vida pré-estabelecido ao final do qual ele se destrói. A chamada apoptose serve para simplificar o gerenciamento dos agentes móveis que são usados na implementação das células sanguíneas.

Além disso, o cérebro pode, de forma hierárquica e voluntária, direcionar a política de ação do corpo, influenciando diretamente o funcionamento do sistema circulatório. A espinha dorsal representa a reação automática correspondente a reflexos, como por exemplo, quando afastamos a mão quando nos queimamos, representando a forma involuntária de reação influenciando o sistema circulatório.

Para que o modelo possibilite que o usuário exerça o papel de supervisão, foram previstos mecanismos para que o sistema se mantenha funcionando de forma permanente e autônoma, similar à sobrevivência dos organismos vivos.

O modelo utiliza também a forma indireta de comunicação entre agentes, inspirado no depósito químico efetuado pelas plaquetas para atrair outras plaquetas e glóbulos vermelhos no processo de tampão plaquetário, cujo objetivo é cessar o sangramento.

3.1.1 Analogia entre Elementos Biológicos e Computacionais

Os elementos principais do modelo biológico do sistema circulatório são: coração, células sanguíneas (glóbulos brancos, glóbulos vermelhos e plaquetas) e as vias do sangue (artérias/arteríolas/capilares e veias/vênulas). Apesar de não fazerem parte do sistema circulatório, outros elementos do corpo humano também foram considerados na analogia: cérebro, órgãos, baço, medula óssea, células do corpo e as células-tronco. Isto foi necessário para completar o modelo, pois a medula óssea e o baço fazem o papel de geradores de células sanguíneas a partir de células-tronco; os órgãos e as células do corpo por serem abastecidos pelo sangue, fazem papel de usuários do serviço; o cérebro faz o papel voluntário de operador do sistema e o coração e as células sanguíneas, o papel involuntário de controle autônomo, distribuído e ininterrupto do sistema.

Nas Tabelas 3.1 e 3.2 são mostrados cada um dos elementos do sistema proposto, a função biológica em que houve a inspiração, uma analogia com o sistema computacional e a característica da computação circulatório-autonômica representada por ele.

3.1.2 Características do Modelo

A seguir são listadas as principais características presentes no modelo de computação circulatória, contendo inspiração biológica aplicada a sistemas computacionais, a saber:

a) O glóbulo vermelho artificial tem o papel de fornecedor de recursos para o sistema, ou seja, exercez a função de otimização dos sistemas computacionais, administrando os recursos necessários para o funcionamento otimizado do sistema (ex.: memória, processamento, etc.);
b) O glóbulo branco artificial tem o papel de defensor do sistema, ou seja, faz a função de proteção dos sistemas computacionais (de forma preventiva);
c) A plaqueta artificial tem o papel de mantenedora do sistema, ou seja, faz a função de recuperação dos sistemas computacionais (de forma corretiva);
d) Está disponível mecanismo que permite que estes elementos artificiais se comuniquem (similar a mensagens químicas – estigmergia e feromônios das formigas) [17], funcionando de forma integrada, coordenada, correlacionada e distribuída;

Tabela 3.1 – Quadro Comparativo dos Elementos do Sistema Circulatório-Autonômico Proposto

Elemento Biológico	Função Biológica	Analogia Computacional	Computação Circulatório-Autonômica
Coração	Bomba que leva o sangue para todas as partes do corpo (batimentos cardíacos)	*Clock* do sistema	Funcionamento Ininterrupto
Glóbulo Branco	Defesa do organismo contra ataques	Antivírus, *firewall*, vacinas	Proteção, configuração e autoajuste
Glóbulo Vermelho	Suprimento das células com oxigênio e Nutrientes	Gerenciadores de rede, balanceamento de carga.	Otimização, configuração e autoajuste
Plaqueta	Manutenção dos vasos Sanguíneos	*Reset*, *shutdown*, salvar contexto do S.O.	Recuperação, configuração e autoajuste
Artérias, arteríolas, capilares	Comunicação com todas as partes do corpo	Redes de Comunicação, Redes de Computadores	Canal de comunicação autonômico
Veias, vênulas	Comunicação com todas as partes do corpo	Redes de Comunicação, Redes de Computadores	Canal de comunicação autonômico

Tabela 3.2 – Quadro Comparativo dos Elementos do Sistema Circulatório-Autonômico Proposto (continuação)

Elemento Biológico	Função Biológica	Analogia Computacional	Computação Circulatório-Autonômica
Órgãos (Fígado, Rim, Pâncreas, etc.)	Diversas: filtro, regulador, renovador, etc.	Componente de software	Elemento Gerenciado
Cérebro	Raciocínio, controle centralizado	Processamento, análise, tomada de decisão	Estratégias (política)
Células	Em conjunto compõem os órgãos	Parte de um Componente de software	Elemento Gerenciado
Células-tronco	Geram as outras células	Nova versão de parte de um Componente de software	Adaptação
Medula-óssea	Geram as células-tronco de forma distribuída	Sistema Operacional	Geração/Regeneração
Baço	Geram as células-tronco de forma centralizada	Sistema Operacional	Geração/Regeneração

e) As células sanguíneas artificiais são criadas com tempo de vida predeterminado ao final do qual se destroem (apoptose) [17];
f) Está disponível mecanismo que avisa que a célula sanguínea está “viva” (similar ao batimento cardíaco) [17];
g) Está disponível mecanismo que informa a “pulsação” da célula sanguínea que ainda está “viva” (similar à medição de frequência cardíaca) [17], para fins de avaliação de desempenho do elemento;
h) Cada célula sanguínea é um gerenciador autonômico;
i) Está disponível mecanismo de *loop* global que permite a adaptação e a criação de novas células sanguíneas e sua distribuição pelo sistema;
j) Está disponível internamente em cada célula sanguínea mecanismo de *loop* local que permite a monitoração, análise, planejamento e execução de ações;
k) O funcionamento do sistema é ininterrupto;
l) São fornecidas estratégias globais a serem seguidas, na forma de regras e parâmetros;
m) São fornecidas estratégias locais, na forma de regras e parâmetros, a serem seguidas dependentes da função de especialização (proteção, recuperação e otimização);
n) São fornecidas estratégias locais, na forma de regras e parâmetros, a serem seguidas independentes da função de especialização (autoajuste e configuração).

3.1.3 Propriedades

A partir das características apresentadas podem-se associar a computação circulatória às seguintes propriedades positivas:

a) Abrangência: caracterizada pela chegada dos vasos sanguíneos a (quase) todas células do corpo (exceto os neurônios);
b) Adaptação: de acordo com mudanças no ambiente, futuras gerações de células sanguíneas podem ser criadas e modificadas se tornando mais adaptadas a ele;
c) Autonomia: caracterizada por ações tomadas pela própria iniciativa do sistema como, por exemplo, as tarefas executadas pelas células sanguíneas, a apoptose e o batimento cardíaco involuntário;

d) Comunicação: caracterizada pelos sinais químicos liberados pelas plaquetas para atrair os glóbulos vermelhos no processo de cicatrização, chamados de comunicação indireta, e estendida para as demais células sanguíneas;
e) Continuidade: a operação do sistema é contínua;
f) Cooperação: verificada pela cooperação entre células sanguíneas, a exemplo do trabalho em equipe entre as plaquetas e os glóbulos vermelhos na formação do tampão plaquetário;
g) Competição: verificada pela grande quantidade de células de mesmo tipo que concorrem para execução da mesma tarefa;
h) Distribuição: caracterizada pela geração distribuída de células sanguíneas;
i) Escala: verificada pela relação de células sanguíneas ser pelo menos uma ordem de grandeza menor que as células do corpo;
j) Escalabilidade: há maior ou menor produção de células sanguíneas de acordo com as exigências do sistema (infecção por vírus ou aumento da atividade);
k) Estabilidade: verificada pela tendência do sistema voltar a um ponto de equilíbrio, após alguma mudança no ambiente;
l) Mobilidade: caracterizada pela movimentação das células sanguíneas segundo o fluxo do sangue;
m) Monitoração: caracterizada pela medição frequência cardíaca, da pressão sanguinea[1] e a análise sanguínea[2] que refletem o funcionamento do sistema possibilitando seu controle;
n) Proteção: verificada pela ação dos glóbulos brancos na defesa do corpo;
o) Reação: inerente aos glóbulos brancos frente a uma infecção, as plaquetas frente a um sangramento e aos glóbulos vermelhos quando fornecem recursos às células;
p) Recuperação: verificada pela ação das plaquetas no processo de cicatrização;
q) Redundância: grande quantidade de células sanguíneas do mesmo tipo;
r) Renovação: células sanguíneas “morrem” e novas células são criadas;
s) Sincronismo: o coração é uma bomba com uma frequência de batimento que comanda a velocidade do fluxo sanguíneo;
t) Sobrevivência: quando há muita perda de sangue, o organismo é colocado em estado de choque, para que as funções essenciais continuem ativas.

O sistema circulatório também apresenta características associadas a propriedades negativas (*drawbacks*):

[1] Fora do escopo deste trabalho.
[2] Fora do escopo deste trabalho.

a) Fragilidade: falta de segurança e estabilidade afetada por doenças vírus ou ataques externos;
b) Pontos de falha únicos: não existe redundância do coração;
c) Dependência: o sistema circulatório depende de outros sistemas;
d) Perda de controle: geração de células sanguíneas podem não acompanhar necessidades sistêmicas, assim como, aumentar mais do que a demanda. O sistema como um todo pode entrar em colapso quando muito exigido.

3.1.4 Interação entre os Elementos

Na Figura 3.2 é mostrado como o modelo circulatório-autonômico proposto integra as várias funções biológico-computacionais. Por questão de simplificação foram omitidas as células-tronco, por se tratarem de novas instâncias de células. Pela mesma razão, os glóbulos brancos, vermelhos e plaquetas foram agrupados como células sanguíneas. Pode-se observar que cada uma delas possui um símbolo representando um coração. No nosso modelo não existe um único coração, cada célula sanguínea possui o seu.

A seguir é descrita a comunicação entre os elementos, numerada de 1 a 7:

1) Comunicação medula óssea – células sanguíneas: como a proposta do modelo é o autogerenciamento do sistema, a Medula Óssea deve minimizar o envio de mensagens diretamente para os órgãos e as células. Ela cria as células sanguíneas, as embute com regras e configurações pré-definidas, inclusive com a informação de quando elas devem se autodestruir e as coloca em execução. A medula óssea está, portanto, delegando a estas células seu gerenciamento.

2) Envio de mensagens pelas células sanguíneas: as células sanguíneas podem enviar mensagens para as células e para os órgãos oferecendo acoplamento.

3) Recepção e Envio de mensagens pelos órgãos: Recebe oferta de acoplamento. Caso aceite a oferta responde ao emissor. Quando precisa informar à medula óssea de algo relevante pode enviar mensagem diretamente para ela.

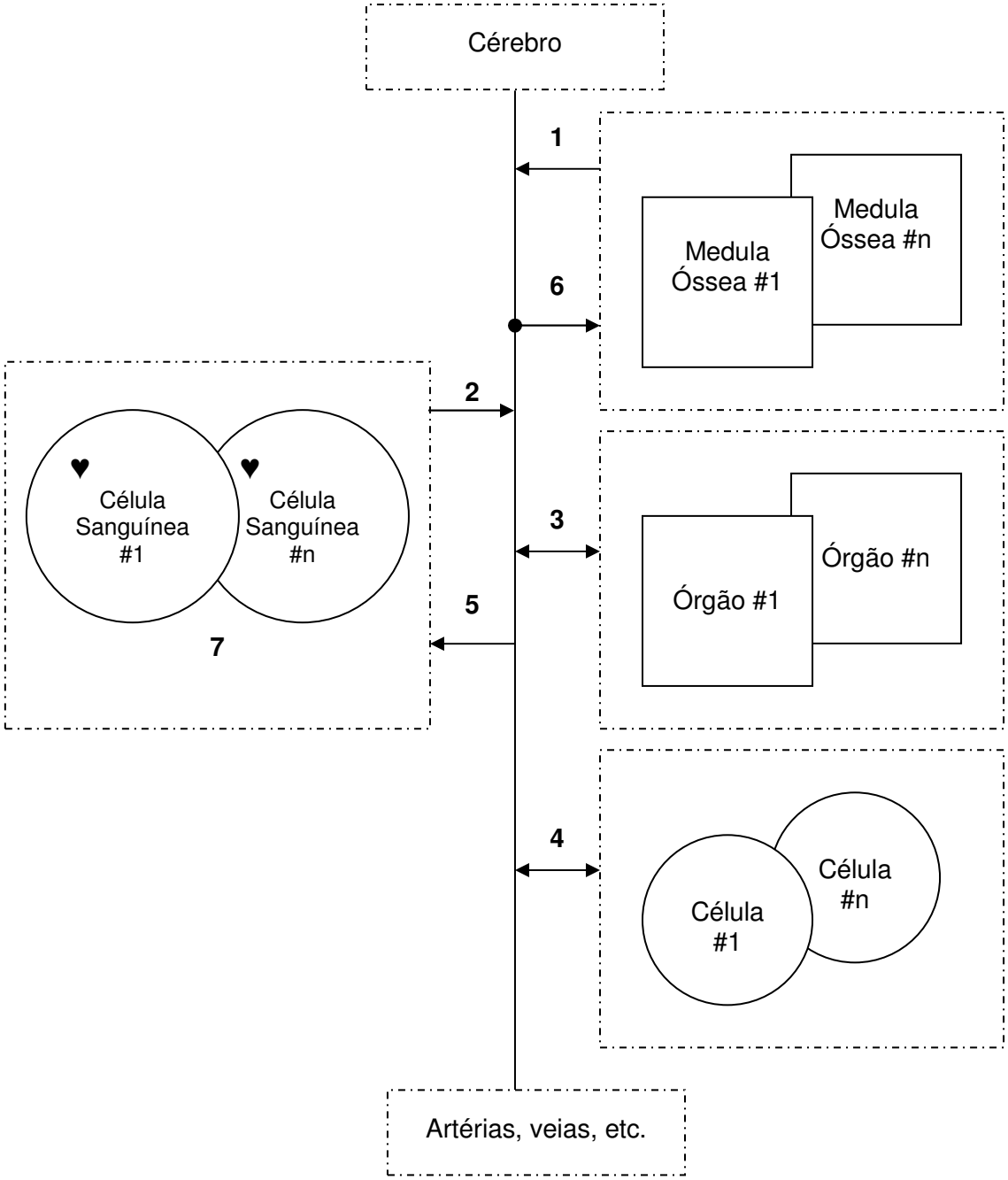

Figura 3.2 – Interação entre os Elementos do Modelo Proposto.

4) Recepção e Envio de mensagens pelas células: idêntica à opção 3.

5) Recepção de mensagens pelas células sanguíneas: são as mensagens provenientes de órgãos ou células reconhecendo o acoplamento oferecido por ela. Dependendo da mensagem e do emissor, podem aceitar o acoplamento ou descartar a mensagem.

6) Recepção de mensagens pela medula óssea: a medula óssea pode receber mensagens das células sanguíneas, das células e dos órgãos. Além disso, ela pode contabilizar a quantidade de células sanguíneas, de forma a criar novas em um processo de replicação. Neste processo ela deve levar em conta o histórico de mensagens, a correlação entre elas, a política e estratégias correntes, de modo que, dependendo destas informações novas regras e configurações possam ser embutidas.

7) Comunicação intercélulas sanguíneas: enquanto as células sanguíneas estiverem "vivas" emitem mensagens indicativas, com um valor de "pulso" indicando o quão saudável elas estão. Além disso, deixam mensagens químicas ("rastros") para que células sanguíneas do mesmo tipo ou de tipos diferentes, localizadas nas proximidades ou que se movimentaram para este local, ao perceberem, analisem a necessidade de tomar alguma atitude (ação).

Apesar ser possível a comunicação interórgãos e intercélulas no sistema biológico, a princípio esta comunicação não é levada em consideração em nosso modelo.

Apesar de nem todas as mensagens serem endereçadas diretamente para a medula óssea, esta tem acesso a todas elas, podendo levá-las em consideração em um processo de inferência e decisão sobre ações a tomar. Ou seja, busca-se a autonomia através da criação de elementos autonômicos na forma de células sanguíneas, para os quais são delegadas funções de gerenciamento. Mas isto não descarta a possibilidade de atuação direta de um órgão central em uma situação extrema (exceção) ou mesmo habitual. Por exemplo, a necessidade de manutenção de um hardware, ou troca de versão de um componente de software, pode solicitar uma ação manual do operador, caso estes mecanismos não estejam totalmente automatizados. Ou seja, caso o sistema não seja totalmente autonômico, a previsão de ações manuais se faz necessária.

O cérebro representa a atuação de um operador humano para modificação de políticas que estão ativas naquele momento. Não está explícito na Figura 3.2, mas o cérebro pode enviar mensagens às medulas ósseas distribuídas informando sobre novas políticas de atuação, o que será refletido em futuras gerações de células sanguíneas.

3.1.5 Pequena Circulação e Grande Circulação

Assim como o sistema circulatório humano possui dois ciclos, a pequena circulação e a grande circulação, nosso modelo é composto por dois ciclos equivalentes: a pequena circulação artificial e a grande circulação artificial.

A pequena circulação artificial municia os glóbulos vermelhos artificiais com recursos computacionais, sendo que esta estratégia é comandada pela regras gerais (estratégia) estabelecidas pelo cérebro. Na pequena circulação é que se dá a replicação das células sanguíneas artificiais. Este processo deve ser protegido, pois é crítico, para evitar ataques e mau funcionamento.

A medula óssea no modelo circulatório-autonômico tem um papel decisório descentralizado, ele aponta as metas a serem alcançadas e as estratégias globais (definidas pelo cérebro) e ao criar as células sanguíneas, embute nelas regras e configurações locais e globais. Por exemplo, os glóbulos brancos e vermelhos e as plaquetas levam sempre consigo regras (similar ao DNA) para depositar mensagens químicas, caso detectem problema local ou para informar simplesmente que passaram por ali. Este depósito pode ser percebido por outros glóbulos próximos, ou que se deslocaram para lá, que executam ações pertinentes de acordo com suas regras e parâmetros. Com isso, a medula óssea (e o cérebro indiretamente) está delegando sua função de gerenciamento para as células sanguíneas artificiais.

A grande circulação artificial é responsável pela distribuição das células sanguíneas artificiais pelo “corpo” (sistema) de forma a alcançar todas as células e, consequentemente, os órgãos.

Em resumo, na pequena circulação as células sanguíneas são criadas (pela medula óssea) e distribuídas pela grande circulação para poderem exercer suas funções. O modelo computacional proposto, analogamente ao sistema circulatório biológico é considerado “fechado”, pois nele são formados dois ciclos: a pequena e a grande circulação.

Por ter como seus elementos básicos os glóbulos e plaquetas, que estão presentes em todas as partes do "corpo" (leia-se componentes do sistema computacional ou sistema), o modelo proposto é, por sua natureza, distribuído, ubíquo[3] e pervasivo[4] .

Como pode ser visto na Figura 3.3, além dos dois ciclos ou *loops* descritos, cada célula sanguínea artificial possui um *loop* interno que lhe possibilita exercer suas funções.

As localizações possíveis de um elemento gerenciador (glóbulos vermelhos, brancos e plaquetas) são:

a) Pequena Circulação Arterial (**PCA**);
b) Grande Circulação Arterial (**GCA**);
c) Grande Circulação Venosa (**GCV**);
d) Pequena Circulação Venosa (**PCV**).

Cada um dos elementos gerenciadores possui um *loop* local interno para realizar, sem interrupção, as ações adequadas que são dependentes da localização em que estiver.

Por exemplo, assumindo que os elementos gerenciadores estão inicialmente localizados em PCA, estes são recém-criados, ou regenerados para permanecerem em funcionamento, ou chegaram ao final de seu ciclo de vida.

A partir daí, estes elementos são distribuídos para a grande circulação, passando para a localização GCA. Então, eles procuram se associar a um elemento gerenciado (acoplamento[5]) formando um elemento autonômico. Caso se acople passa para a localização GCV, sendo que seu *loop* interno, que está sempre em funcionamento, monitora as informações vitais do sistema gerenciado, passando a controlá-lo. Caso ele não se acople, decorrido um período de tempo pré-estabelecido, passa para a localização PCV. Desta localização, decorrido um período de tempo pré-estabelecido, ele flui para a localização PCA e o ciclo recomeça. Ele também pode chegar à localização PCV, a partir do GCV, com a ocorrência do evento de desacoplamento ou com o final de sua vida útil.

[3] que está ao mesmo tempo em toda a parte, onipresente.
[4] que se infiltra, que penetra; espalhado, difuso; penetrante.
[5] O acoplamento ocorre quando um elemento gerenciador se une ao elemento gerenciado formando um elemento circulatório-autonômico.

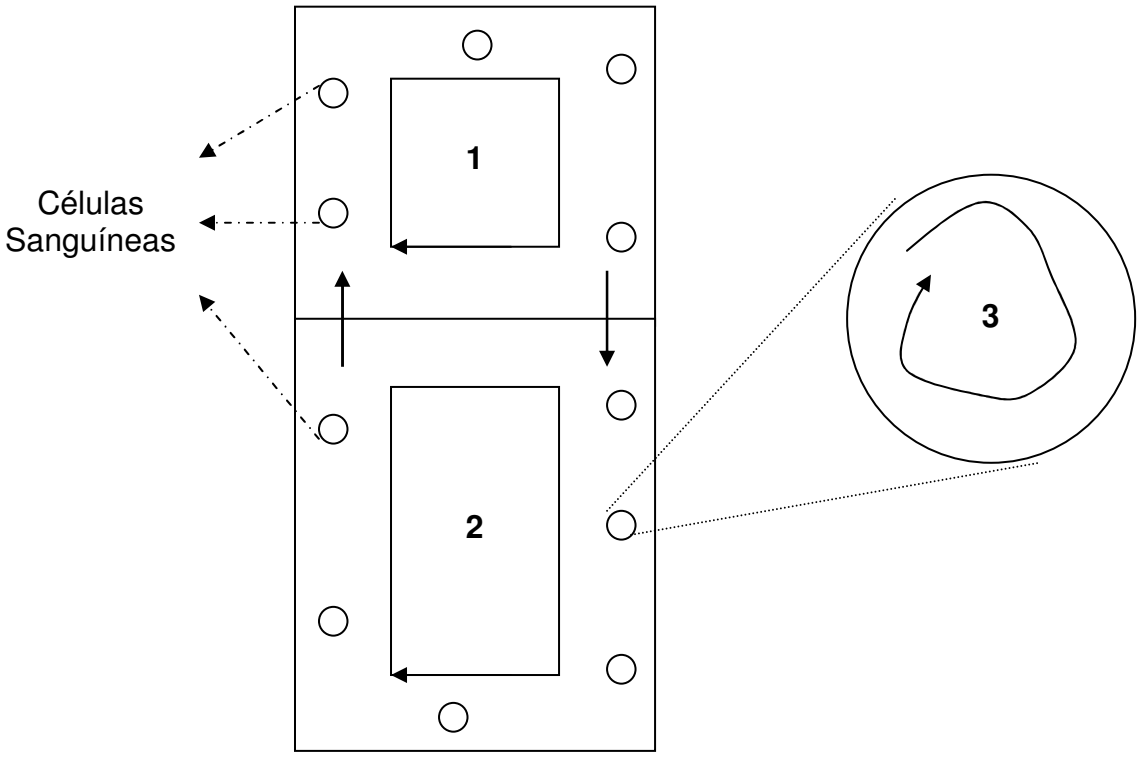

Figura 3.3 – À esquerda, *loops* correspondentes à Pequena (1) e à Grande Circulação (2) e, à direita, em detalhe ampliado, *loop* local interno (3).

Na Figura 3.4 é proposta máquina de estados para representar o ciclo de vida do elemento gerenciador (células sanguíneas). Os círculos representam os estados (possíveis localizações). A mudança de estado é representada por setas que se originam em um estado e apontam para o estado destino. Cada seta possui uma ou mais condições booleanas para controle da mudança de estado, que quando satisfeita(s) promove(m) a mudança de estado do elemento. Exceto no caso em que as setas se originam e se destinam ao mesmo estado, representando condição ou condições para permanência no mesmo estado.

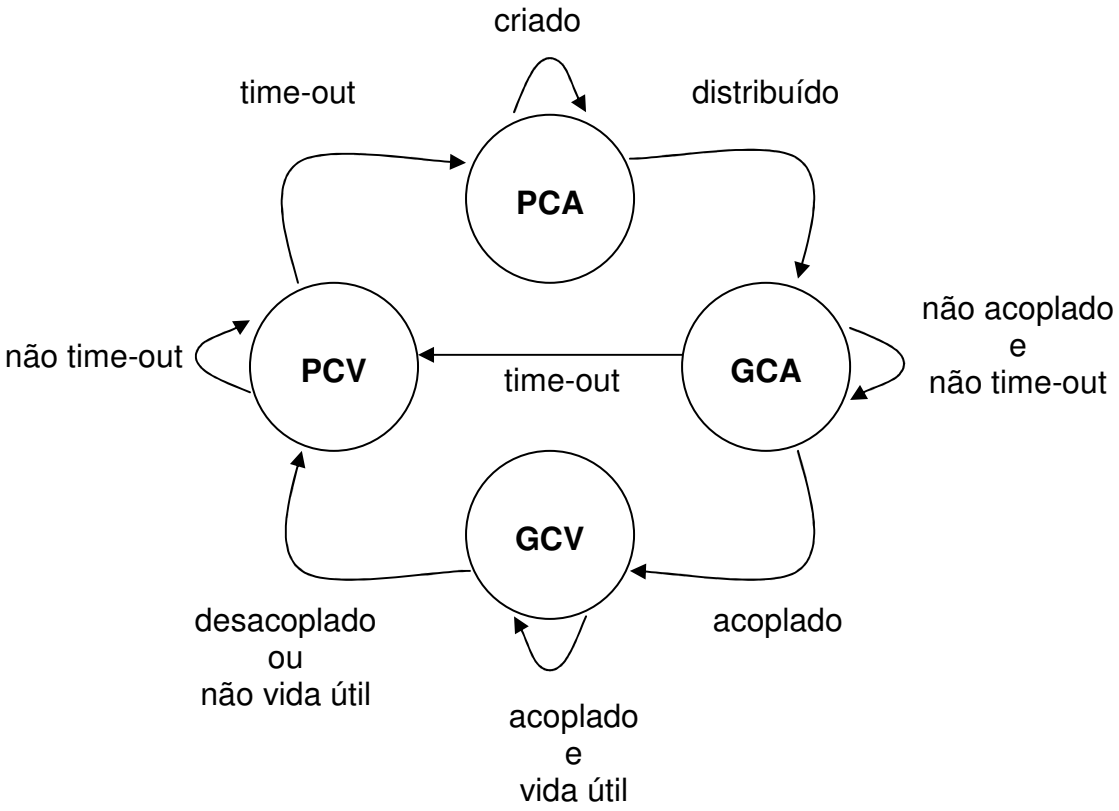

Figura 3.4 – Máquina de Estados Representando os Canais Arterial e Venoso correspondentes à Pequena e à Grande Circulação para os elementos gerenciadores.

A Figura 3.5 contém a máquina de estados proposta para o elemento medula óssea restrita à pequena circulação.

O elemento medula óssea possui apenas os estados PCV e PCA. Quando está no estado PCA, caso seja necessário, cria novos elementos gerenciadores com regras e parâmetros atualizados, levando em consideração, por exemplo, o *log* de mensagens criado pelos elementos gerenciadores já existentes e pelas políticas de alto nível estabelecidas. Já no estado PCV analisa o *log* de mensagens criado pelos elementos gerenciadores já existentes para decidir, por exemplo, se nenhum, um ou mais parâmetros devem ser modificados para novos gerenciadores a serem criados.

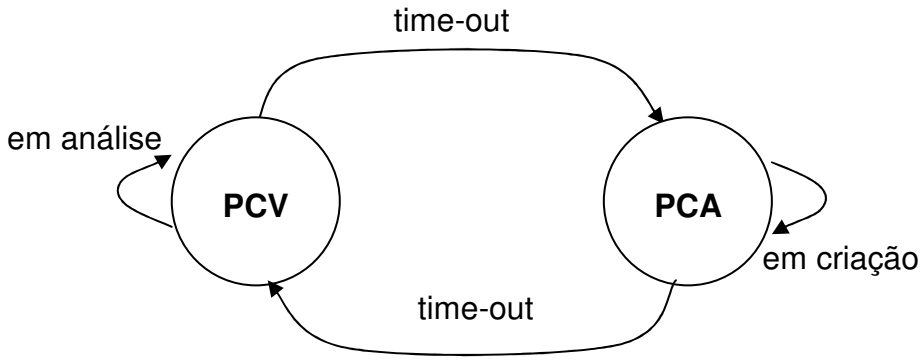

Figura 3.5 – Máquina de Estados Representando os Canais Arterial e Venoso correspondentes à Pequena Circulação para o elemento medula óssea.

3.1.6 Elemento Circulatório-Autonômico

Na Figura 3.6 está representado um elemento autonômico constituído pelo gerenciador autonômico e pelo recurso gerenciado. Estão representados os processos MAPE de Monitorar, Analisar, Planejar e Executar; e a base de conhecimento, do lado do gerenciador e o sensor e atuador do lado do recurso.

Na Figura 3.7 está representado um elemento circulatório-autonômico. Nesta figura a base de conhecimento está dividida em parte global e local, para chamar a atenção que existem parâmetros e regras em níveis global e local. Note também, que foi acrescido o processo de Reflexo (reação) para caracterizar ações rápidas (reações) muitas vezes necessárias para preservação da integridade do elemento gerenciado ou do próprio gerenciador. Ou seja, o MAPE pode se transformar apenas em MRE.

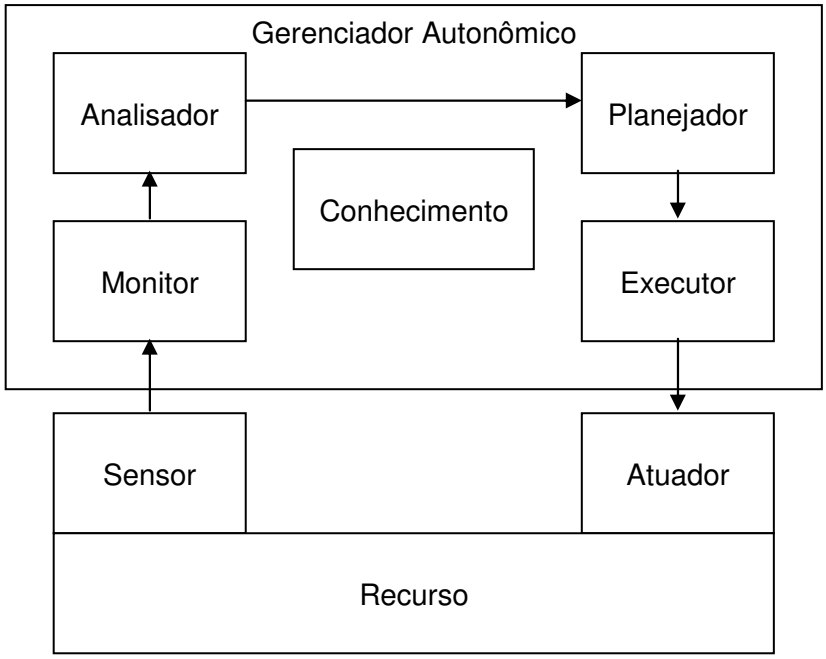

Figura 3.6 – Esquema de Elemento Autonômico: Gerenciador Autonômico e Recurso.

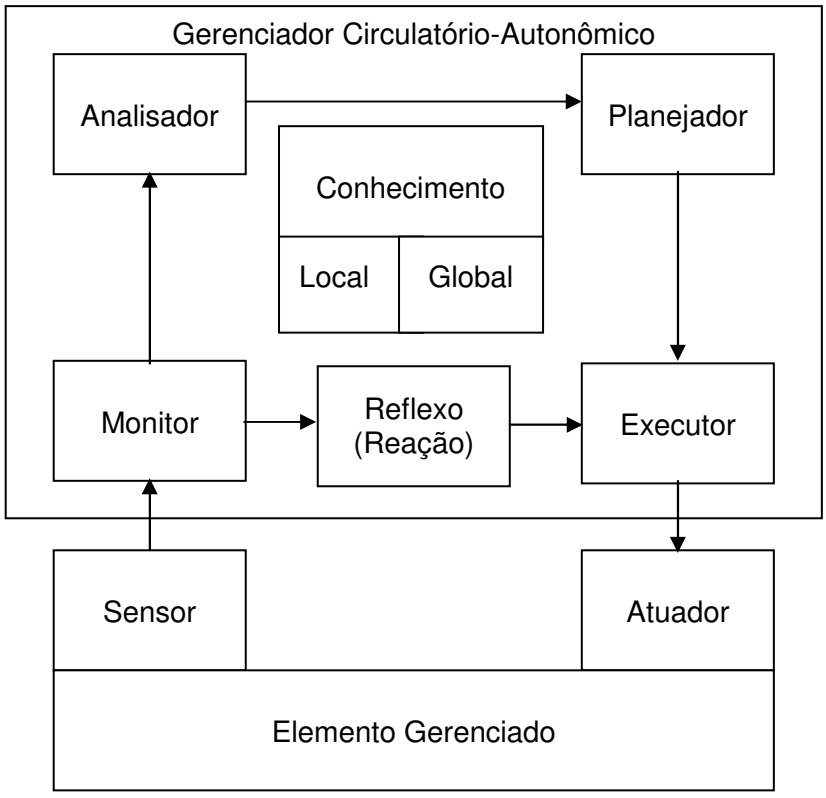

Figura 3.7 – Esquema de Elemento Circulatório-Autonômico.

Outras observações são que este gerenciador circulatório-autonômico pode ser especializado para se transformar em uma “plaqueta”, “leucócito” ou “hemácia” e não fica acoplado permanentemente ao componente de software (“órgão” ou “célula”). Ele só se acopla a um componente quando, em tempo de execução, passar em uma localização “próxima” a ele e o componente responder afirmativamente à oferta dos serviços do gerenciador. A recusa de acoplamento pode ocorrer caso o componente já tenha sido atendido por um gerenciador neste ciclo, ou não necessite de assistência no momento.

3.2 Modularização

A partir das peças de origem biológica ilustradas na Figura 3.1 foi feito um diagrama em blocos para modularizar o modelo de autogerenciamento do CIACOM ilustrado na Figura 3.8.

Na figura são apresentados os seguintes blocos: Ação/Reação, Geração, Distribuição, Acoplamento, Renovação, Análise Ativa e Análise Passiva. A cor vermelha (cinza mais claro) corresponde à circulação arterial e a cor azul (cinza mais escuro) corresponde à circulação venosa. Note que o bloco de Geração está repetido para representar a geração centralizada (no baço) e distribuída (medula óssea). As setas indicam a direção do fluxo de células característica do dinamismo do modelo baseado em agentes móveis. O bloco de renovação representa a geração de novas células sanguíneas em substituição àquelas que cometeram apoptose. A análise é dividida em duas partes: ativa e passiva. A Análise Ativa permite que a célula sanguínea execute ações como resposta rápida a estímulos externos (do ambiente ou de outras células); enquanto que a Análise Passiva permite uma análise mais profunda, geralmente mais lenta, influenciando novas gerações de células. O bloco Ação/Reação está representado na cor preta correspondendo ao papel de supervisão do homem acima do *loop* de controle.

O bloco de geração é responsável pela criação das células sanguíneas artificiais. Para tanto deve ser dotado de algoritmo que, para a geração de novas células, leve em consideração o número de células do corpo, o número de células sanguíneas já criadas por tipo, sua taxa de natalidade e sua taxa de mortalidade. Estas células podem ser implementadas como agentes móveis cujos comportamentos dependem de seu tipo, ou seja, de sua finalidade: proteção, recuperação ou otimização. Estes

agentes móveis se responsabilizariam pela gerência dos recursos distribuídos implementados como agentes estáticos representando as células do corpo.

Figura 3.8 – Diagrama em blocos ilustrando o modelo CIACOM de autogerenciamento.

Uma vez criadas as células sanguíneas são distribuídas de acordo com um algoritmo de distribuição. O bloco de distribuição é dotado de um algoritmo que cuida de enviar o agente móvel a partir da máquina que o criou para outra máquina mais adequada como destino.

Ao chegar ao destino, a célula tenta se acoplar às células do corpo localizadas nesta máquina através de uma estratégia de acoplamento. Tanto a distribuição quanto o acoplamento levam em consideração rastros químicos locais deixados por outras células do sistema, como comunicação indireta, de forma a auxiliar na sua autoconfiguração e/ou decisão de próxima ação a ser tomada. Trata-se de inspiração biológica baseada na forma de comunicação das formigas (estigmergia), mas também similar às mensagens químicas (feromônios) utilizadas pelas plaquetas para

comunicar às outras plaquetas e aos glóbulos vermelhos que detectou um local que necessita de reparo e que precisa de sua ajuda no processo de coagulação.

Células sanguíneas que não conseguiram se acoplar são redistribuídas para outras máquinas até que se esgote o número máximo de migrações permitido ou se esgote seu tempo de vida (apoptose). Durante seu ciclo de vida, estejam acopladas ou não, vão deixando rastros para outras células se guiarem configurando uma análise ativa com cooperação. Podem ler informações e mensagens (sensores) e atuar sobre outras células (ambiente) ou enviar mensagens (atuadores).

Células sanguíneas que esgotaram seu tempo de vida ou o número máximo de migrações retornam à máquina que as criou. É feita, então, uma análise passiva das informações colhidas pelas células sanguíneas de forma a subsidiar na renovação de células adaptadas às novas condições do sistema. Esta característica pode ser considerada como de otimização ou competição, uma vez que as células sanguíneas com melhor característica são reproduzidas em detrimento de outras "menos capazes". Os módulos de população (renovação e geração) devem garantir a produção de novas células de forma a manter um número de células adequado ao bom funcionamento do sistema. No entanto, uma nova geração de células só deve ser criada, modificando gerações anteriores, quando critérios estabelecidos forem alterados além ou aquém de um determinado limiar a ser definido e dependente da aplicação. Além disso, devem ser levados em consideração o dinamismo e a necessidade de adaptação da população de agentes uma vez que máquinas podem ser desligadas "matando" células, assim como, novas máquinas podem ser inseridas no sistema aumentando a demanda por mais agentes.

O módulo de ação e reação permite que o operador humano modifique configurações de parâmetros e políticas iniciais ou atuais, caracterizando o posicionamento do homem em um nível superior de supervisão.

Para permitir a atuação, comunicação, movimentação, análise ativa, reação, depósito de rastros, recebimento de mensagens, percepção e acoplamento, as células são dotadas de *loop* interno conforme ilustrado na Figura 3.9. Na implementação utilizando sistemas de multiagentes, as células gerenciadoras são agentes móveis e suas ações são comportamentos executados dentro de seu *loop* interno. Já as células gerenciadas podem ser representadas por agentes fixos.

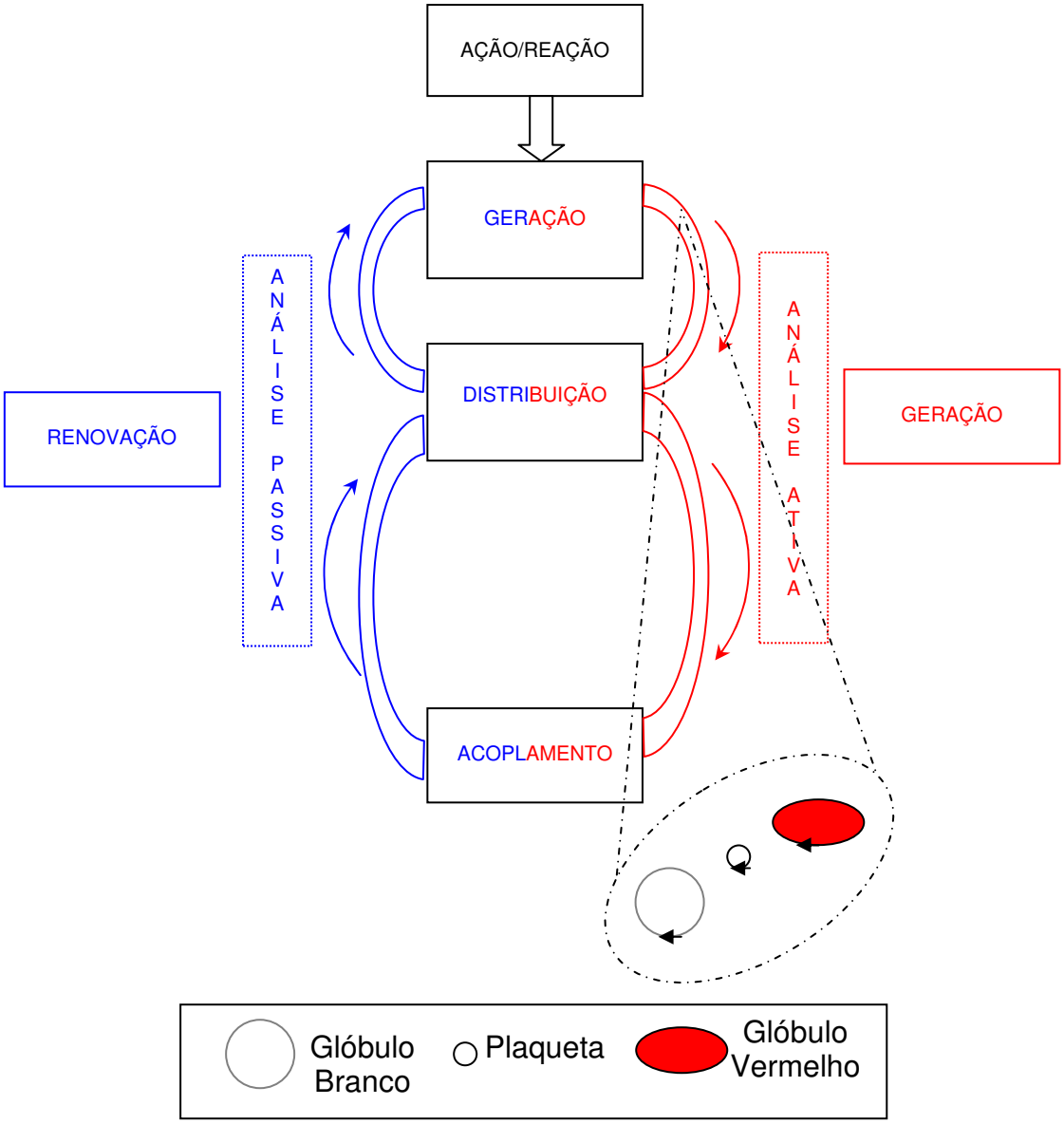

Figura 3.9 – Detalhe do modelo: *loop* interno das células.

Na Figura 3.10 estão representados os conjuntos (“enxames”) de células e suas funcionalidades: recuperação (plaquetas), otimização (glóbulos vermelhos) e proteção (glóbulos brancos). Elas são dotadas de mecanismo que permite o depósito de “rastros químicos” de forma a orientar não somente seus pares, mas também outros tipos de células. Estes rastros são voláteis, isto é, vão reduzindo sua intensidade com o passar do tempo. Esta comunicação indireta possibilita a correlação entre as diversas funcionalidades, possibilitando, assim como a cooperação dos glóbulos vermelhos com as plaquetas no processo de coagulação do sangue (Figura 3.11), seu gerenciamento de forma integrada.

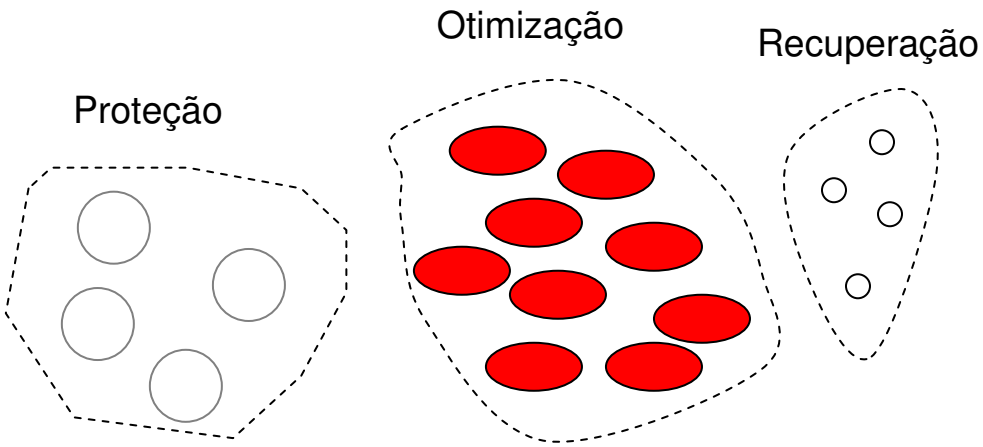

Figura 3.10 – Conjuntos (“enxames”) de células.

Cocluindo, trata-se de um modelo distribuído de autogerenciamento com supervisão em que os índices, parâmetros e outras variáveis do sistema vão depender do problema no qual ele for aplicado.

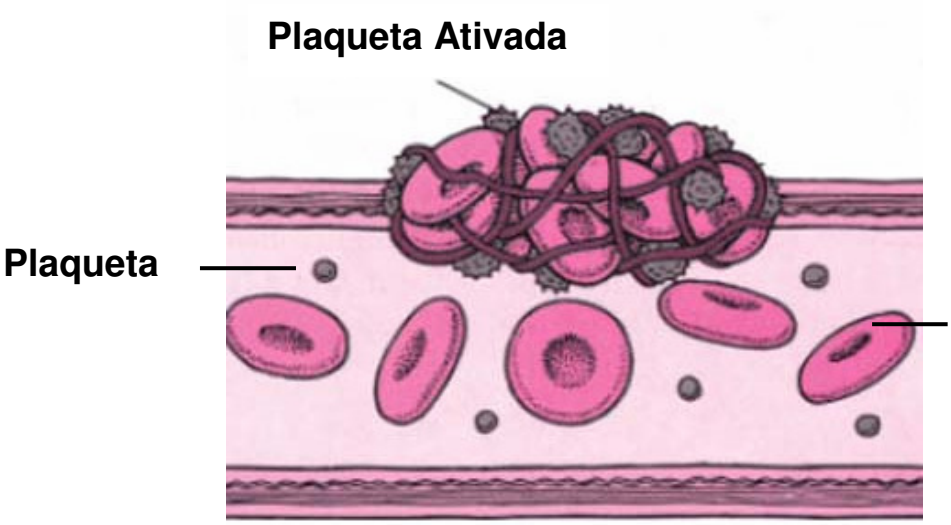

Figura 3.11 – Formação do Coágulo com Plaquetas e Glóbulos Vermelhos.

3.3 Detalhamento

A especificação do modelo é baseada em multiagentes móveis e estáticos cujos comportamentos executados são controlados por dois *loops*: interno (local) e externo (global). Os comportamentos servem para executar funções intrínsecas ao modelo, assim como especificidades inerentes à aplicação específica. A seguir são descritos os tipos de agentes utilizados na construção do modelo, sua dinâmica de criação e seus comportamentos principais. Em seguida são descritos os módulos componentes

para construção do modelo e a estratégia de execução de seus comportamentos controlados pelos dois *loops*.

3.3.1 Tipos de Agentes

A construção do modelo é baseada em multiagentes móveis e estáticos que podem ser dos seguintes tipos: agente de ação ou de interface, agente criador ou de geração, agente célula sanguínea ou gerenciador e agente célula do corpo ou gerenciado.

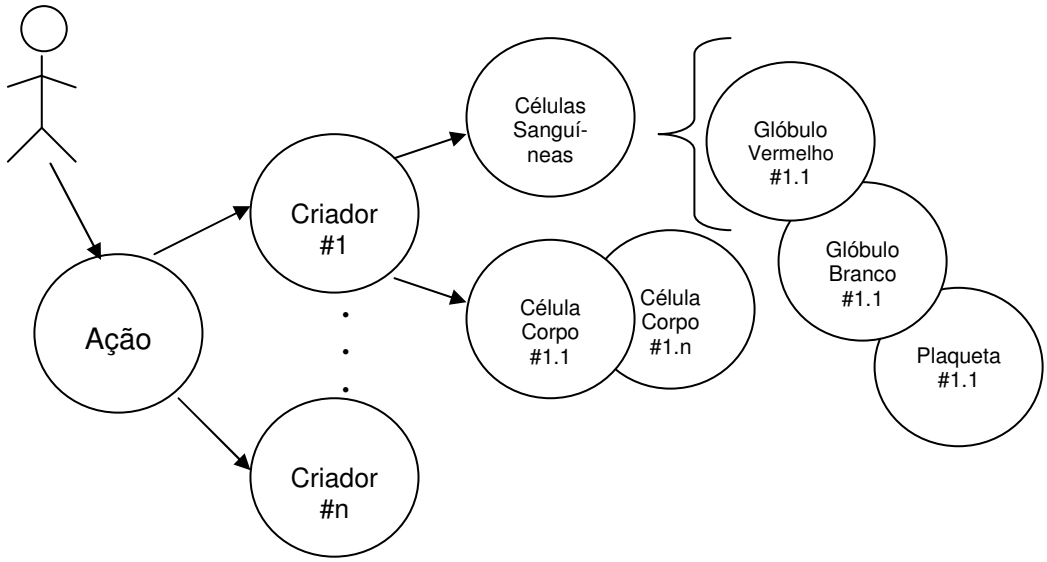

Figura 3.12 – Fluxo de criação de agentes.

É proposto um fluxo de criação de agentes a partir de um único agente ação, conforme apresentado na Figura 3.12. Neste fluxo, o operador humano cria um agente ação contendo políticas e estratégias de operação. Este agente ação, por sua vez, é responsável pela geração de outros agentes chamados de criadores, além de promover a interface de nível supervisório entre o operador humano (usuário) e o sistema de gerenciamento. O agente ação gera um número inicial de **n** agentes criadores e os distribui pela rede. Estes agentes criadores se movem pela rede e se

instalam nos nós identificados como com mais conectividade ("small world"[6] e "scale-free"[7] [71]) conforme Figura 3.13.

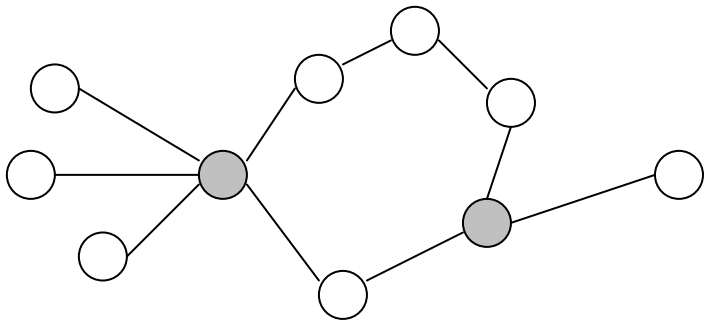

Figura 3.13 – Nós da rede. Os nós com mais conectividade estão destacados.

Estes agentes criadores instalados nestes nós se clonam de forma a "povoar" toda a rede, isto é, se instalar em todos os nós identificados como nós com mais conectividade. Esta clonagem (Figura 3.14) deve ser feita com uma determinada frequência de modo a garantir a cobertura da rede e compensar agentes criadores "mortos". Agentes criadores são classificados como híbridos, por serem móveis e estáticos durante seu ciclo de vida.

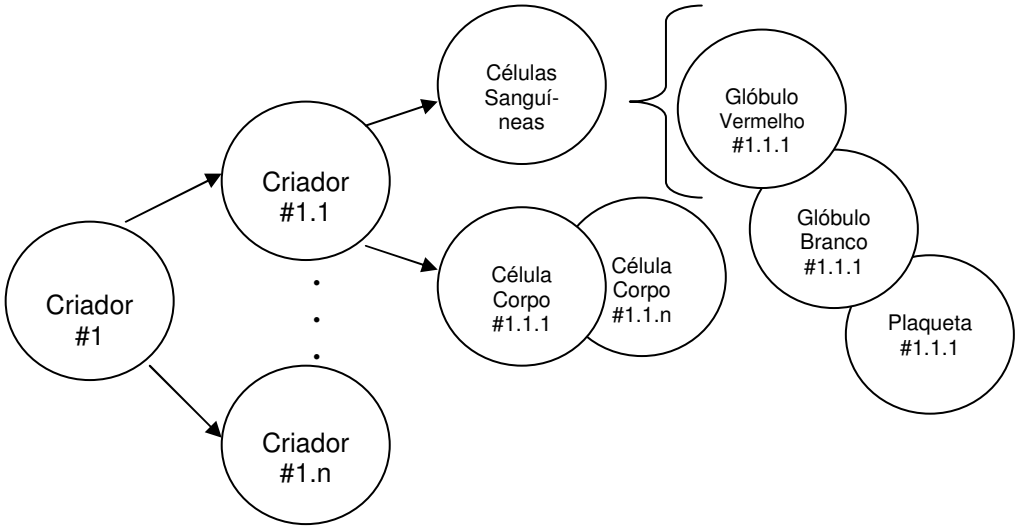

Figura 3.14 – Clonagem de agentes Criadores.

[6] Conclusão do experimento de Stanley Milgram realizado em 1967 para identificação de redes sociais em que foram verificados "seis níveis de separação" de relacionamento entre as pessoas.

[7] A noção de redes sem escala ("scale-free networks") que caracteriza uma variedade de sistemas complexos nos quais alguns nós apresentam um enorme número de conexões enquanto que outros apresentam poucas conexões.

Uma vez instalados os agentes criadores passam a criar outros agentes (células sanguíneas). Estes agentes (células sanguíneas) apresentam como tipos: os glóbulos brancos, os glóbulos vermelhos e as plaquetas conforme Figura 3.15. Para cada tipo de célula sanguínea é gerada uma população responsável pela recuperação (plaquetas), proteção (glóbulos brancos) e otimização (glóbulos vermelhos).

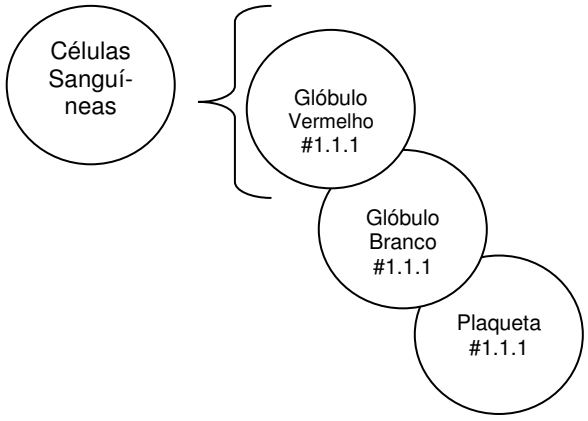

Figura 3.15 – Detalhe dos tipos de agentes células sanguíneas.

Para fins de teste são criados tantos agentes células do corpo, quantos nós do sistema distribuído a ser gerenciado conforme Figura 3.16. Estes agentes migram para todas as máquinas (exceto um agente célula do corpo local para simular aplicativo na própria máquina) e nelas se instalam, se transformando em agentes estáticos.

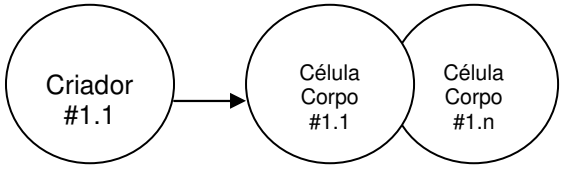

Figura 3.16 – Detalhe dos agentes células corpo.

Tanto os agentes célula sanguínea quanto os células do corpo são criados pelos agentes criadores e distribuídos com estratégias de migração (distribuição) diferentes.

A quantidade de células sanguíneas criadas vai depender da quantidade de células do corpo, pois a metáfora do sistema sanguíneo sugere uma relação de 1:10 (um para dez), ou seja, um glóbulo vermelho sendo capaz de atender até 10 células do corpo. Esta relação cai para 1:100 se relacionamos os glóbulos brancos às células do corpo e 1:1000 se relacionamos as plaquetas às células do corpo.

Quando solicitada pelo operador humano, o agente ação se comunica com os agentes criadores para informá-los sobre mudanças nas políticas de operação (Figura 3.17).

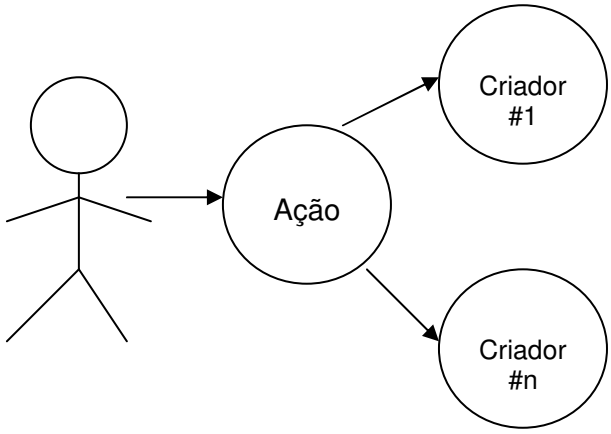

Figura 3.17 – Detalhe da comunicação entre os agentes ação e criadores.

Os agentes célula se transformam em Gerenciadores Circulatórios se e somente se acoplarem a um agente corpo.

Os agentes são implementados como multiagentes móveis e estáticos. Na modelagem proposta as ações que um agente pode realizar são traduzidas na forma de comportamentos.

A seguir são reapresentados os tipos de agentes e seus respectivos comportamentos.

O agente ação apresenta os comportamentos: comunicação (com agentes criadores), interface (operador humano) e criação (gera agentes criadores e os distribui pela rede);

Agentes criadores apresentam os comportamentos: clonagem, criação, comunicação, distribuição, análise, migração, identificação e instalação;

Agentes corpo apresentam os seguintes comportamentos: acoplamento, desacoplamento e comunicação;

Agentes células sanguíneas apresentam os seguintes comportamentos: acoplamento, comunicação, migração, rastreamento, configuração, ação, monitoração, análise e apoptose.

3.3.2 Dinâmica de Funcionamento

Na Figura 3.8 foram apresentados os módulos componentes para construção do modelo proposto. Para alinhavar estes módulos são apresentados na Figura 3.18 os *loops* global e local. Em seguida, cada módulo é explorado individualmente e indicado(s) o(s) algoritmo(s) necessários para sua consecução.

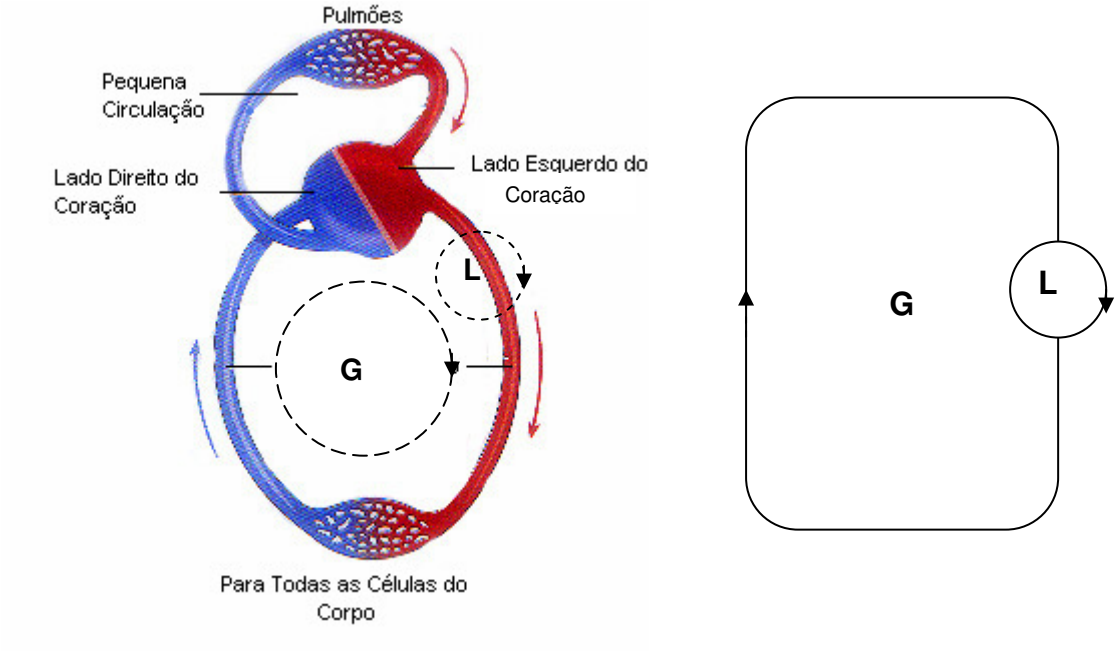

Figura 3.18 – *Loop* Global (G) e Local (L) executado pelo agente célula sanguínea e analogia com o Sistema Circulatório.

Na Figura 3.18 é feita uma analogia entre a grande circulação do sistema circulatório (à esquerda) e o *loop* global (G) de controle da Computação Circulatória (à direita), onde na primeira as células sanguíneas são transportadas pelo corpo enquanto na segunda os agentes trafegam pela rede. Já o *loop* local (L) corresponde à troca efetuada pelas células com seu ambiente enquanto é feito o transporte global no caso biológico, enquanto na Computação Circulatória corresponde aos comportamentos executados pelos agentes (seus atuadores) de acordo com os estímulos recebidos ou percebidos por seus sensores.

Na Figura 3.19 é apresentado o modelo Computação Circulatória como um sistema de controle ultraestável de Ashby. Note que os *loops* global (G) e local (L) foram montados sobre a Figura 2.5, assim como os agentes célula para análise ativa (mais rápida) e controle sobre o agente corpo (ambiente) e agente criador para análise passiva (mais lenta) com possibilidade de ajuste das variáveis essenciais.

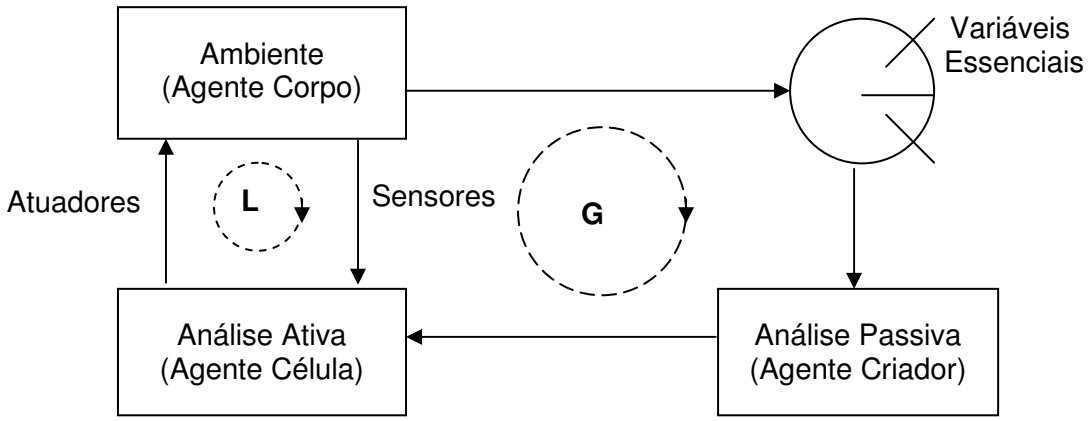

Figura 3.19 – Computação Circulatória como um sistema de controle ultraestável de Ashby.

Na Figura 3.20 é feito um resumo onde é apresentada a relação entre os agentes que compõem a Computação Circulatória e os módulos por eles executados.

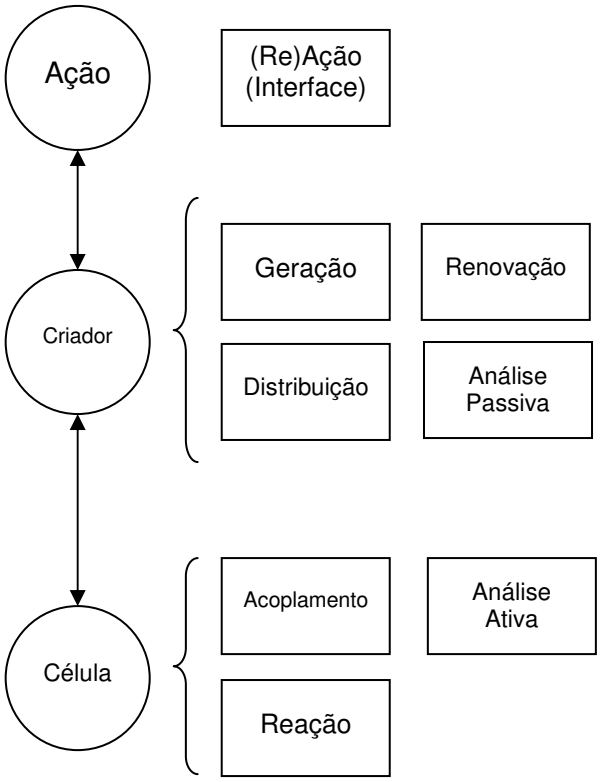

Figura 3.20 – Relação entre Agentes e módulos.

O agente ação é responsável pelo módulo ação que cria os agentes criadores e realiza a interface com o sistema através de interface gráfica. As setas interligando os agentes representam a relação de criação e de comunicação entre os agentes. Sendo

que os agentes célula podem se comunicar entre si de forma direta (mensagens) ou indireta (rastros).

Na Figura 3.21 é feita uma revisão da máquina de estados proposta (subseção 3.1.5, Figura 3.4) para controlar os agentes célula durante seu ciclo de vida que vai da geração, passando pela distribuição, acoplamento, e culminando com a análise onde é decidida sua renovação ou apoptose (morte). Note novamente montagem com os *loops* global (G) e local (L). Esta revisão do nome dos estados teve como objetivo melhorar a compreensão, uma vez que os nomes anteriores, na forma de siglas GCV, PCV, PCA e GCA, são parecidos causando confusão.

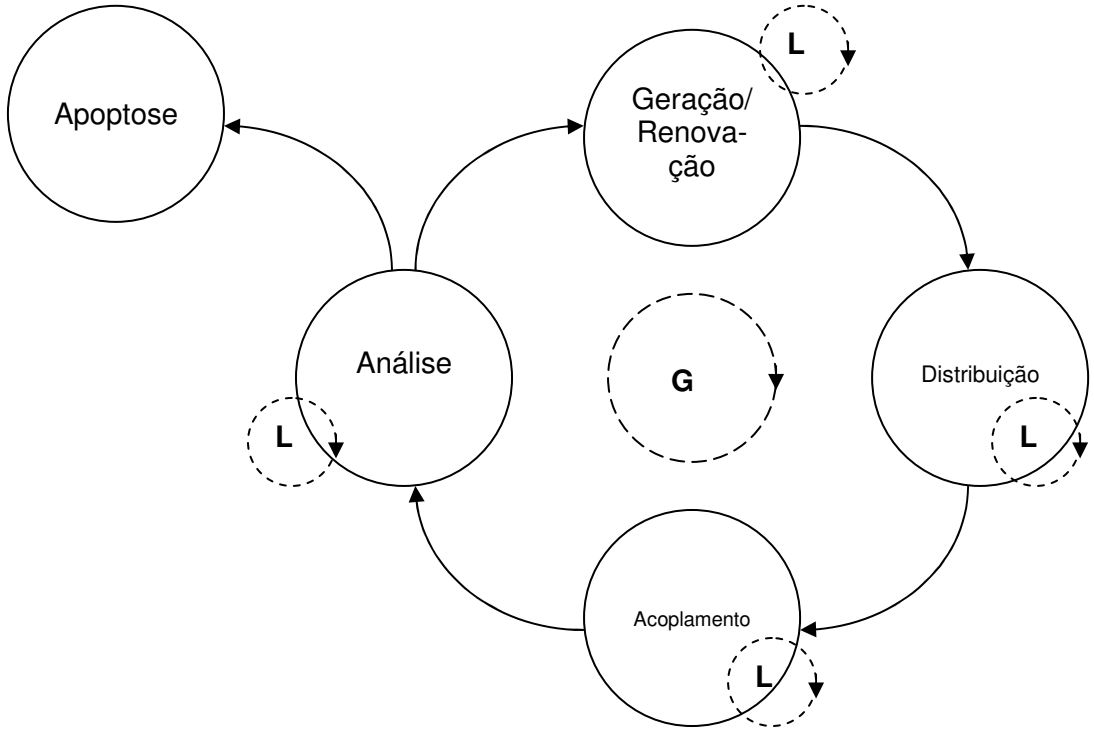

Figura 3.21 – Máquina de estados para controle do Agente Célula no *loop* Global com representação de *loop* Local para cada estado.

Na Figura 3.22 é mostrada a hierarquia de comando entre agentes: Ação → Criador → Célula. É dada ênfase na atuação do Agente Criador nas fases de Geração e Análise do ciclo de vida do Agente Célula.

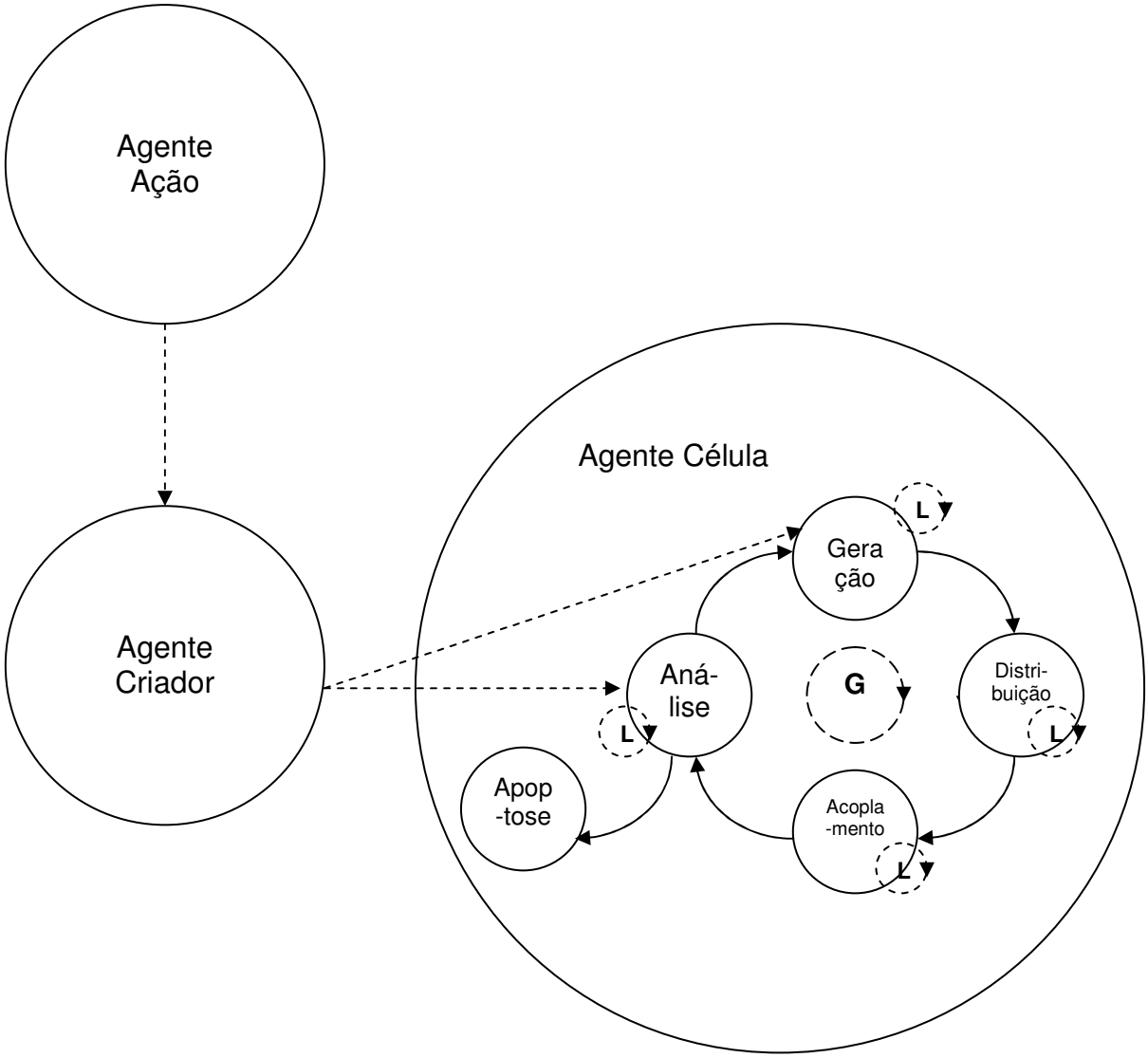

Figura 3.22 – Hierarquia de comando: Agente Ação atuando sobre Agente Criador que por sua vez atua nas fases de Geração e Análise do ciclo de vida do Agente Célula.

3.3.3 Módulo de Geração

O agente criador cria novos agentes célula por tipo: glóbulo vermelho, glóbulo branco ou plaqueta. A quantidade a ser criada varia de acordo com a quantidade de agentes já existentes e a projeção de quantos agentes por tipo são necessários para atender a demanda atual.

Para o funcionamento ininterrupto do CIACOM é fundamental que haja uma geração contínua de agentes, mas é fundamental que haja controle para evitar tanto o seu crescimento exponencial quanto sua extinção. Trata-se, portanto, do problema de

controle populacional de agentes móveis em redes dinâmicas. A questão da movimentação dos agentes, para promover uma distribuição que atenda à demanda (homogênea ou heterogênea), é tratada no módulo de distribuição, no próximo item.

No capítulo 5 é apresentado estudo de caso que utiliza o mecanismo de controle populacional do CIACOM com independência do número inicial de agentes célula do sangue, do número total de agentes corpo e que, com parâmetros estáticos ajustados inicialmente, converge para uma população que corresponde a um percentual da população de agentes célula do corpo.

3.3.4 Módulo de Distribuição

Um agente célula depois de gerado é distribuído pela rede para o próximo nó pertencente ao conjunto de nós gerenciados pelo *agente criador*. O tipo de padrão de migração pode ser itinerário (*itinerary pattern*) ou alternativamente, outro padrão possível é o *random walk,* onde a migração se dá por sorteio aleatório e não segue um itinerário pré-estabelecido.

Por questões de simplificação, o módulo de distribuição do CIACOM é baseado no *random walk*. Em alguns exemplos é explorado o depósito de feromônios, no sentido de otimizar a migração de agentes, ou seja, o feromônio está “repelindo” agentes para promover uma distribuição mais homogênea.

3.3.5 Módulo de Acoplamento

A estratégia de acoplamento do agente gerenciador (células sanguíneas) ao agente gerenciado (células do corpo) é fracamente acoplada. Ou seja, o gerenciador não fica permanentemente ligado ao elemento gerenciado. A política de acoplamento (conectividade) é baseada em nível de rastro (estigmergia). Um agente gerenciador se instala em um nó e aguarda por um tempo o acoplamento com o elemento gerenciado.

Caso expire o tempo e não haja acoplamento ele migra para o próximo nó do itinerário (*itinerary pattern*), ou seja, usando a mesma estratégia do algoritmo de distribuição. Neste período ele adquire informações que servem para o cálculo da quantidade de feromônios a serem depositados. Estes níveis são utilizados pelos próximos

gerenciadores para decidirem se se instalam neste nó ou se migram direto para o próximo. Estes feromônios são voláteis, isto é se evaporam com o tempo. Estes níveis dependendo do tipo de agente podem fornecer informações para correlação. Como o desligamento de um nó pode acarretar a perda destas informações, pois o agente pode ser desligado junto, deve haver um número mínimo de agentes para garantir redundância de cobertura. Um novo nó pode ser incorporado também a qualquer momento.

3.3.6 Módulo de Análise

São dois os módulos de análise: a análise ativa e a passiva.

Na análise ativa, os agentes célula executam algoritmo (comportamento) de depósito de rastro e de análise de rastro para auxiliar na tomada de decisão (reação) quanto ao acoplamento, distribuição ou ação específica da aplicação. Além disso, parâmetros (variáveis essenciais) são colhidas. Quando no estado acoplamento o agente realiza efetivamente o que demanda a aplicação, ou seja, sua atividade fim.

Na análise passiva, os agentes criadores analisam indicadores colhidos pelos agentes célula e seus parâmetros para verificar se estão acima ou abaixo de um determinado limite. Caso os limites estabelecidos sejam ultrapassados podem caracterizar a necessidade de novas gerações de agentes célula adaptados com novos parâmetros que melhor atendam ao sistema. Outra ação possível é a emissão de aviso ao operador sobre a situação atual do sistema, para que este em uma posição de supervisão tome as devidas providências.

3.3.7 Módulo de Ação

Este módulo deve conter o algoritmo de geração de *agentes criadores*. O próprio *agente ação* pode distribuir os *agentes criadores* ou delegar ao *agente criador* se autodistribuir. O algoritmo de distribuição deve levar em consideração nós com mais conectividade para estabelecimento dos *agentes criadores*. Além disso, contém a interface homem-máquina para permitir que o operador (“agente humano”) envie comandos (de alto nível) com novas políticas de atuação do sistema. Estas são

recebidas pelos *agentes criadores* que tomam as ações apropriadas: geração de mais agentes de um determinado tipo ou redução, alteração de parâmetros, entre outras.

3.3.8 Módulo de Reação

Está sendo contemplada a reação do agente célula em relação a seu ambiente, as mensagens diretas e indiretas de outros agentes. Ou seja, ação de atuadores do agente a partir da percepção de seus sensores. A reação do conjunto (espinha dorsal) de forma involuntária está fora do escopo desta tese, sendo deixada para trabalhos futuros.

3.3.9 Módulo de Renovação

O *agente criador* que criou determinado agente célula determina, no módulo de análise ativa, se este é renovado ou não. Aquele que cometeu apoptose é eliminado. A princípio a renovação se resume a simplesmente redistribuir o agente célula, mas com possibilidade de "recarregar" seu tempo de vida.

Este módulo tem forte ligação com o módulo de geração e a renovação pode estar embutida neste módulo. Entretanto, uma das principais virtudes do modelo CIACOM é deixar explícitos seus módulos de forma a facilitar e nortear sua aplicação a problemas reais. Serve também como um *checklist* de forma que o usuário do modelo não esqueça de implementar nenhuma característica de sua aplicação.

3.4 Conclusão

Foi apresentada uma proposta baseada na circulação sanguínea para modelagem de autogerenciamento de sistemas complexos. Como o modelo proposto também surgiu da influência da computação autonômica e de sua metáfora com o sistema nervoso autônomo, foi denominado Modelo Computacional Circulatório-Autonômico, em inglês, CIrculatory-Autonomic COmputing Model, ou "CIACOM". Mais considerações sobre o modelo podem ser encontradas no Anexo B.

As contribuições deste modelo para o modelo autonômico já existente são: o mapeamento das células sanguíneas nas propriedades *auto** e o acréscimo de mais inspirações biológicas, como, por exemplo, os batimentos e pulsações do coração, a apoptose e a liberação de substâncias químicas pelas células sanguíneas, que dão ênfase à forma integrada como as propriedades *auto** podem ser analisadas para obter-se o autogerenciamento. Ou seja, o novo modelo dá ênfase à necessidade de análise integrada das informações.

Para facilitar o entendimento para posterior implementação do modelo CIACOM, procurou-se dividi-lo em módulos. Neste capítulo, cada módulo foi explorado.

4 Implementação

4.1 Introdução

O CIACOM tem como objetivo prover autogerenciamento em sistemas computacionais distribuídos, interligados de pequeno (telefone celular ou PDA) ou grande porte (servidores ou computadores pessoais). O problema de autogerenciamento em foco alavanca soluções que envolvam redução na troca de mensagens entre agentes com maior ênfase em sua mobilidade. A tecnologia de multiagentes foi utilizada por ser apropriada para implementar autonomia e mobilidade em sistemas computacionais distribuídos. Um protótipo foi desenvolvido para prova de conceito. Foi utilizado em sua implementação o *framework* JADE [29] (Java Agent Development Environment) para sistema multiagentes, a linguagem de programação JAVA e o ambiente de desenvolvimento integrado (IDE – Integrated Development Environment) Eclipse. Para simulação dos algoritmos propostos, com um grande número de agentes, utilizamos o ambiente de programação NetLogo [77].

4.2 Implementação

No modelo proposto cada elemento é um agente em que suas ações são implementadas por comportamentos (*behaviours*). O Anexo F contém o código correspondente a esta implementação.

Ação: Responsável pela interface com o operador humano, geração e distribuição dos agentes Criadores. Possui os comportamentos Interface Behaviour, Creation Behaviour e Distribution Behaviour (Figura 4.1).

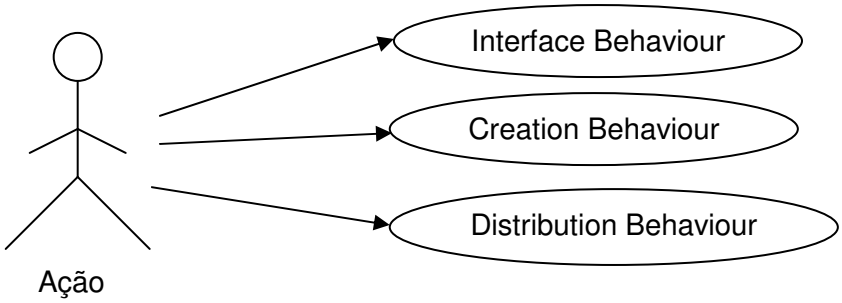

Figura 4.1 – Agente Ação do CIACOM e seus respectivos comportamentos.

Criador: Responsável pela criação das células. Possui os comportamentos Creation Behaviour e Analysis Behaviour. Inspirado na medula óssea que produz as células sanguíneas de forma distribuída. No protótipo, para fins de simulação, também cria as células do corpo (Figura 4.2).

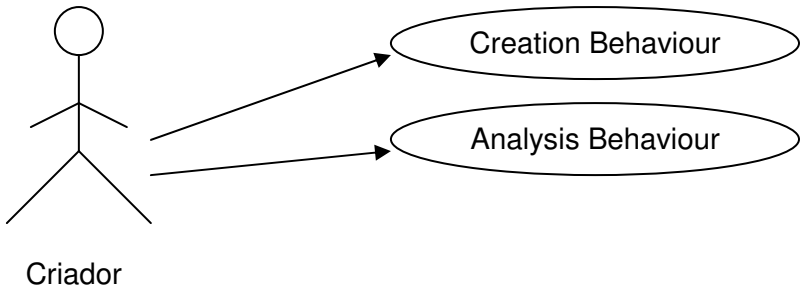

Figura 4.2 – Agente Criador do CIACOM e seus respectivos comportamentos.

Célula: Responsável pelo gerenciamento distribuído. Inspirados nas células sanguíneas que têm a função de proteção (glóbulos brancos), recuperação (plaquetas) e manutenção (glóbulos vermelhos) das células do corpo. Possui os comportamentos Reflection Behaviour, Execution Behaviour, Publication Behaviour, Monitoring Behaviour, Apoptosis Behaviour, Configuration Behaviour, IamAlive Behaviour, Pulsing Behaviour, State Behaviour, Mobile Behaviour e Communication Behaviour (Figura 4.3). Nesta versão do protótipo não houve diferenciação entre as células sanguíneas (glóbulos brancos, glóbulos vermelhos e plaquetas).

Corpo: Responsável pelo aplicativo propriamente dito. Possui os comportamentos Application Behaviour, Communication Behaviour e Subscription Behaviour (Figura 4.4). As células do corpo, chamadas apenas de corpo, por simplificação e para evitar confusão com as células sanguíneas, foram criadas para fins de simulação do gerenciamento de aplicativos.

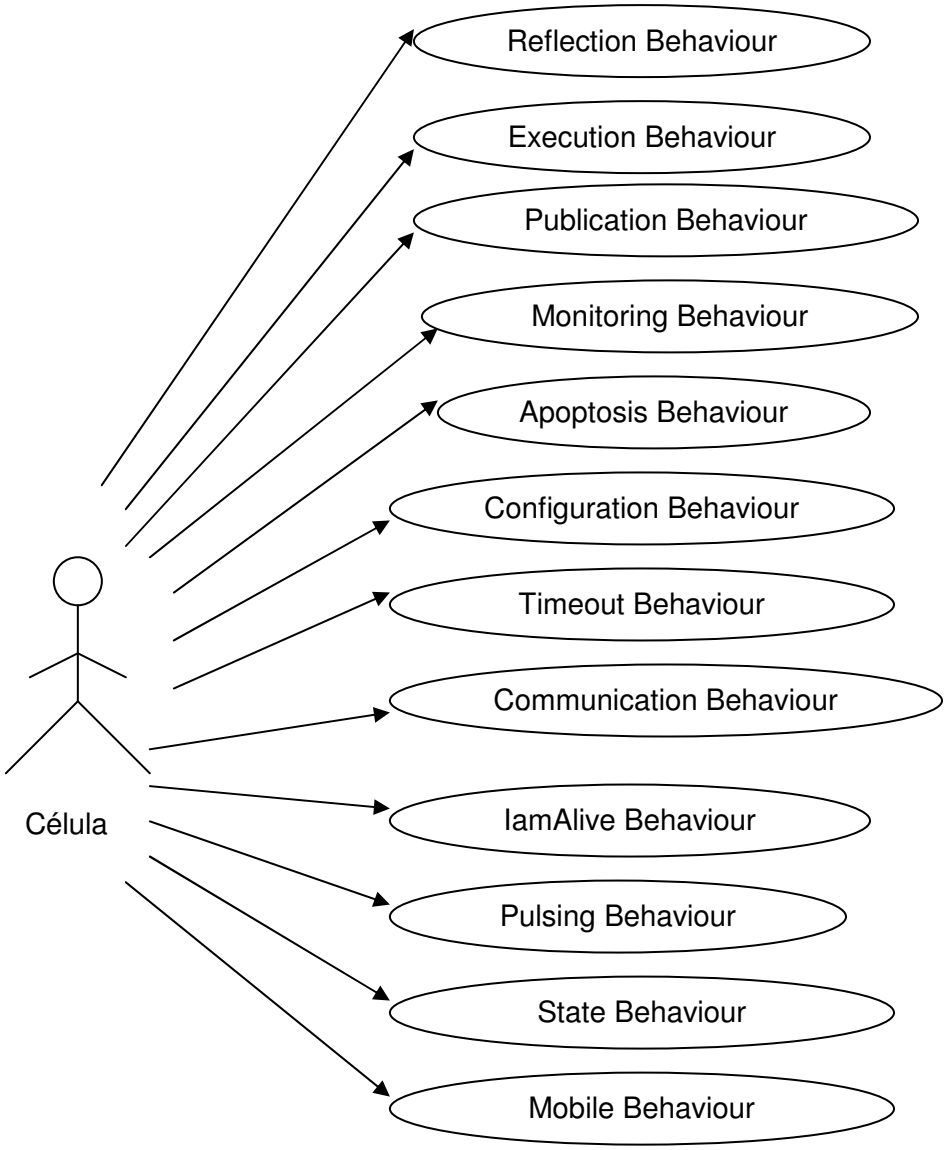

Figura 4.3 – Agente Célula do CIACOM e seus respectivos comportamentos.

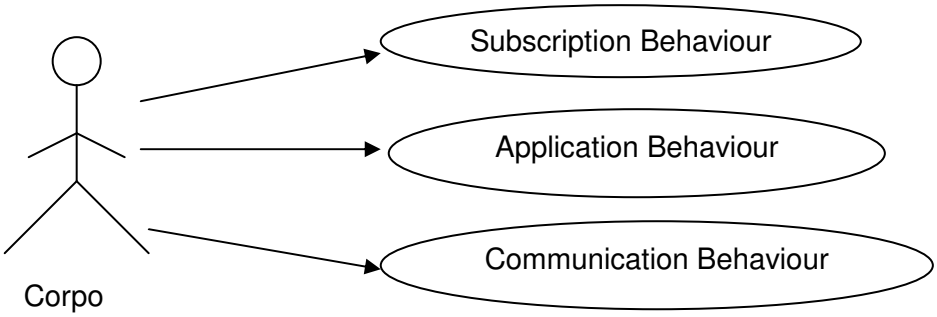

Figura 4.4 – Agente Corpo do CIACOM e seus respectivos comportamentos.

Na Figura 4.5 está ilustrada a máquina de estados que simula a pequena e a grande circulação sanguínea. Cada célula possui sua própria máquina de estados que a controla durante seu ciclo de vida.

O ciclo de vida de uma célula começa no estado CRIAÇÃO. O agente Criador é o responsável pela criação de novas células e sua distribuição pelo sistema (nós da rede ou *containers* JADE). Para tanto o criador possui o comportamento Creation. Células antigas que estão neste estado e ainda não expiraram seu tempo de vida devem ser redistribuídas. Portanto, as células novas são distribuídas e as antigas redistribuídas passando para o estado DISTRIBUIÇÃO. Esta passagem se dá pela condição booleana CRIADO que passou de FALSE para TRUE. O comportamento State cuida de verificar as condições e mudar de estado quando necessário caracterizando sua autonomia depois da criação.

A quantidade de novas células (população) criadas depende do número de células do corpo a serem gerenciadas e células sobreviventes. Para elaborar este cálculo foi utilizada heurística conforme descrito no item 2.5.1 O importante é guardar uma proporção entre células sanguíneas e células do corpo, de forma que as últimas sejam mais numerosas (por exemplo, relação 1:10). Note que o estado e as condições são internos ao agente célula e estão encapsulados nele.

Quando um agente célula é criado, é estipulado um tempo de vida aleatório (entre um patamar mínimo e um patamar máximo), um tempo de espera máximo (timeout) e uma quantidade máxima de nós (ou *containers*) a serem visitados pelo agente célula.

Para controlar o tempo de vida de uma célula são utilizados o comportamento Apoptosis e a condição booleana Fim_de_Vida. Para controlar o tempo de espera máximo e a quantidade máxima de nós são utilizados o comportamento Timeout e o contador N_MIGRAÇÕES.

Para implementar a característica de mobilidade da célula foi elaborado o comportamento Mobile. Ele cuida de mover a célula para o destino adequado de acordo com as condições e o estado da célula.

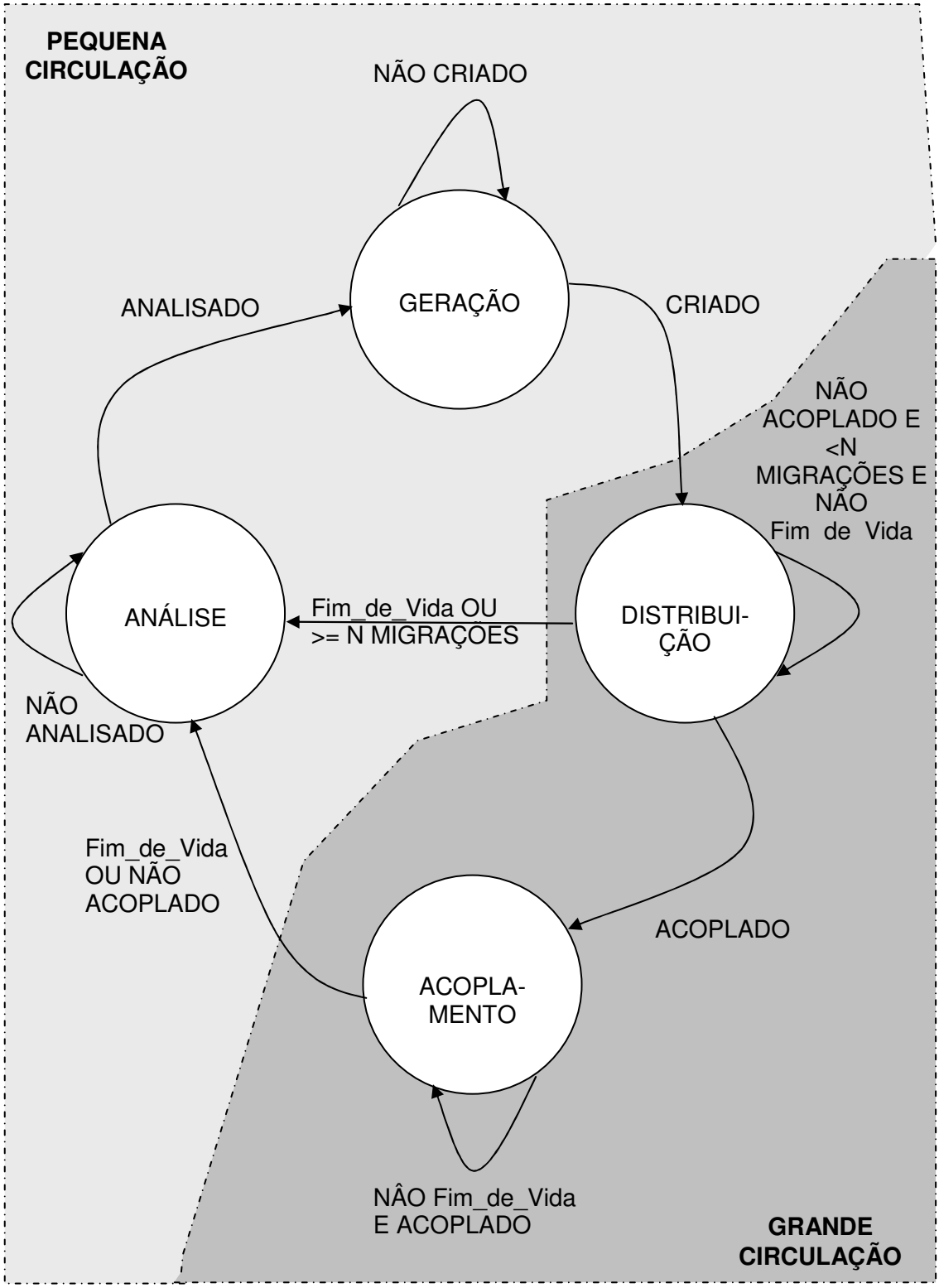

Figura 4.5 – Máquina de estados simulando a pequena e a grande circulação.

Portanto, quando uma célula passa para o estado DISTRIBUIÇÃO, ela migra para um *container* remoto usando o comportamento Mobile. Lá chegando, ela está pronta para servir e publica sua disponibilidade em um agente “página amarelas”, local ao *container*, usando o comportamento Publication. O agente “páginas amarelas” (DF –

directory facilitator – agente *default* do JADE), utilizado nesta versão é único e local ao "main container" (também *default* do JADE). O JADE permite a criação de mais de um DF (redundância) aumentando a disponibilidade e a confiabilidade do sistema.

Quando uma célula do corpo necessita dos serviços de uma célula ela utiliza seu comportamento Subscription para solicitar (estabelecer) uma relação de gerenciamento, ou seja, receber mensagens com informações pertinentes, ser monitorada através do comportamento Monitoring e executar controles através do comportamento Execution. Mais de uma célula do corpo pode subscrever uma mesma célula para satisfazer a relação *n* para *m*, onde *n* é o número de células, *m* é o número de células do corpo e $n < m$.

O comportamento Communication serve para estabelecer a comunicação entre célula e corpo. A célula passa para o estado ACOPLAMENTO quando a condição ACOPLADO passar para TRUE. Isto ocorre após troca de mensagens entre célula e corpo, de acordo com protocolo de comunicação. No protótipo foi utilizado um protocolo simples (THANKS) no qual o agente célula "cumprimenta" o agente corpo com uma mensagem específica, e este por sua vez responde com o mesmo cumprimento.

Caso a célula ainda esteja no estado DISTRIBUIÇÃO e estoure o tempo máximo (timeout) de espera pelo acoplamento, e seu tempo de vida não tenha expirado e o número máximo de migrações não tenha sido alcançado (N_MIGRAÇÕES), ela migra para o próximo Container da lista e incrementa seu contador de migrações, mas permanece no estado DISTRIBUIÇÃO. Caso seu tempo de vida tenha expirado ou o número máximo de migrações tenha sido alcançado (N_MIGRAÇÕES) o estado passa a ser ANÁLISE.

Quando no estado ACOPLAMENTO há uma interação entre célula e corpo, de forma a que a célula gerencie o corpo. Pode haver mais de um corpo sendo gerenciado pela mesma célula, desde que todos estejam contidos no mesmo *container*.

O comportamento Configuration é um metacomportamento e serve para adicionar todos os outros comportamentos da célula.

O comportamento IamAlive serve para a célula informar que está viva. Esta informação é emitida na forma de mensagem, periodicamente ou sob demanda

contendo seu destinatário (outras células e/ou agente corpo). O comportamento Pulsing é semelhante ao IamAlive, mas informa além de que ela está viva, sua saúde (por exemplo, 100% corresponderia a saúde ótima e 50% a saúde precária).

O comportamento Reflection serve para monitorar o ambiente externo à célula e, caso seja necessário, executar ações que preservem a própria célula.

As células do corpo, além dos comportamentos Subscription e Communication, possui o comportamento Application apresentando funções específicas para sua aplicação. Ou seja, está executando um programa para uma finalidade específica.

Uma célula, quando passa para o estado ANÁLISE, pode ter sido proveniente do estado DISTRIBUIÇÃO devido ao Fim_de_Vida ou estouro do número máximo de migrações; ou do estado ACOPLAMENTO devido ao desacoplamento do corpo e também por Fim_de_Vida.

Quando uma célula está no estado ANÁLISE a condição booleana ANALISADO é que controla sua passagem para o estado CRIAÇÃO. Neste estado o criador cuida de analisar informações provenientes das células que migraram para o *container* onde se originaram usando o comportamento Analysis. Esta análise permite ao criador quando da criação de novas células aperfeiçoá-las, ou seja, adaptá-las de forma a criar nova geração de células mais aptas ao autogerenciamento. Uma vez completada a análise a condição ANALISADO passa para TRUE e o estado passa a ser CRIADO (desde que FIM_de_Vida seja FALSE) e o ciclo de vida da célula reinicia. Caso seja a condição FIM_de_Vida seja TRUE, a apoptose é efetivada com a autodestruição da célula, encerrando definitivamente seu ciclo de vida.

4.3 Simulação

Foi utilizado o software NetLogo [77] para simular o CIACOM em uma grande quantidade de agentes. Foi utilizado um *loop* de controle genérico [78] semelhante ao exibido na Tabela 4.1.

Tabela 4.1 – O *loop* de controle.

```
1: inicializa o mundo
2: inicializa os indivíduos
3: enquanto verdadeiro
4:        para cada indivíduo
5:                observe o mundo
6:                execute uma ação
7:        fim para
8:        mundo: determina novas observações
9:        mundo: processa custos e benefícios para todos os indivíduos
10: fim enquanto
```

Este *loop de* controle inclui a inicialização (passos 1 e 2), simula o *loop* global (passos 3 a 10) e o *loop* local (passos 4 a 7).

4.4 Conclusão

Também foram implementados alguns comportamentos do sistema multiagente CIACOM utilizando o ambiente de desenvolvimento JADE. Verificou-se a viabilidade de utilização deste ambiente para o desenvolvimento de um protótipo do CIACOM no qual o comportamento e a máquina de estado dos agentes criador, célula e corpo foram implementados. A implementação se restringiu à criação de uma única plataforma e de dois *containers* em um único computador. Ao iniciarmos o sistema um único agente criador cuida de criar todos os outros agentes, ou seja, células e corpo. A natural extensão deste protótipo para uma rede de computadores e implementação em outros dispositivos, como por exemplo, telefones celulares e PDAs é deixada para futuros trabalhos.

Depois da implementação da primeira versão do protótipo do CIACOM, usando o ambiente de desenvolvimento JADE, foi observada a necessidade de teste de algoritmos de geração e distribuição para uma grande quantidade de agentes. Ou seja, apesar do protótipo mostrar ser factível a implementação do modelo proposto a partir da criação de agentes e seus comportamentos, a implementação e testes de algoritmos mais elaborados não foi adequada para uma grande quantidade de

agentes. Optou-se, portanto, pela utilização do ambiente de simulação NetLogo para validar o modelo frente a uma grande quantidade de agentes.

No próximo capítulo são apresentados estudos de caso, testes e resultados utilizando simulações no NetLogo, uma aplicação real em um *grid* computacional e um estudo puramente teórico de uma aplicação no setor elétrico.

Mais considerações sobre a implementação e a simulação podem ser encontradas nos Anexos E e F.

5 Estudos de Caso, Testes e Resultados

Foram elaborados três estudos de caso para testar a viabilidade de se construir aplicativos segundo o modelo CIACOM. O primeiro estudo de caso é o balanceamento de carga e a desinfecção utilizando enxames de agentes simulados no NetLogo. O segundo estudo de caso é a aplicação do modelo ao escalonamento de tarefas no ambiente de *grid* oportunista OurGrid. O terceiro estudo de caso visa aplicar o modelo à operação do setor elétrico, porém sem simulação, ficando restrito à teoria. Os resultados dos dois primeiros estudos de caso são apresentados.

5.1 Estudo de Caso: Balanceamento de Carga e Desinfecção

O balanceamento de carga é proposto como um problema a ser resolvido para prova de conceito do modelo CIACOM. Observe que apesar da motivação deste estudo de caso ser balanceamento de carga, este problema foi tratado de forma simplificada de modo a facilitar a simulação, acompanhamento e análise crítica dos resultados.

O problema a ser tratado pode ser definido da seguinte forma: os computadores de uma empresa estão interligados com uma determinada topologia, formando uma rede de computadores, têm acesso à Internet e estão sujeitos a falhas e ataques externos. Em cada computador pode ser executado no máximo *n* processos. Um operador responsável pelo gerenciamento dos computadores dita a política de distribuição dos processos pelos computadores: alta, média ou baixa. A política de alta permite o desbalanceamento na distribuição dos processos, ou seja, que alguns computadores executem muitos processos enquanto que outros fiquem com poucos ou nenhum processo. A política de média possibilita um equilíbrio maior nesta distribuição sem, no entanto, obter o balanceamento ótimo. Quando o operador estabelece como política a baixa deseja que o balanceamento seja otimizado, ou seja, a distribuição dos processos pelos computadores seja o melhor possível. Por questão de simplificação consideramos que a capacidade de processamento e memória dos computadores são as mesmas, assim como, que os processos ocupam o mesmo tamanho de memória e têm o mesmo tempo de execução. O operador distribui os processos de forma manual sem automação.

O problema foi definido desta forma para realçar a impossibilidade de gerenciamento manual de uma grande quantidade de computadores e processos, sem que se procure como alternativa o aumento do contingente de operadores, a automatização de processos ou ambos. Mas o aumento da complexidade dos próprios sistemas dificulta a sua automação. Na maioria dos casos é necessária a automação de diversos processos, cada uma destas automações (gerenciamentos) executando concorrentemente, de forma assíncrona, competindo por recursos computacionais e muitas vezes provocando oscilações indesejadas que comprometem o bom funcionamento do sistema como um todo. Ou seja, a otimização de um determinado gerenciamento pode prejudicar outro podendo causar o colapso geral do sistema, ou simplesmente não possibilitar que suas metas de disponibilidade (sobrevivência/estabilidade/confiabilidade) sejam atendidas. Deve ser provido mecanismo que possibilite a interação entre os diversos processos de gerenciamento de forma a evitar a emersão de comportamento de oscilação indesejado no sistema.

A solução de autogerenciamento proposta com utilização do modelo CIACOM é mostrada na Figura 5.1.

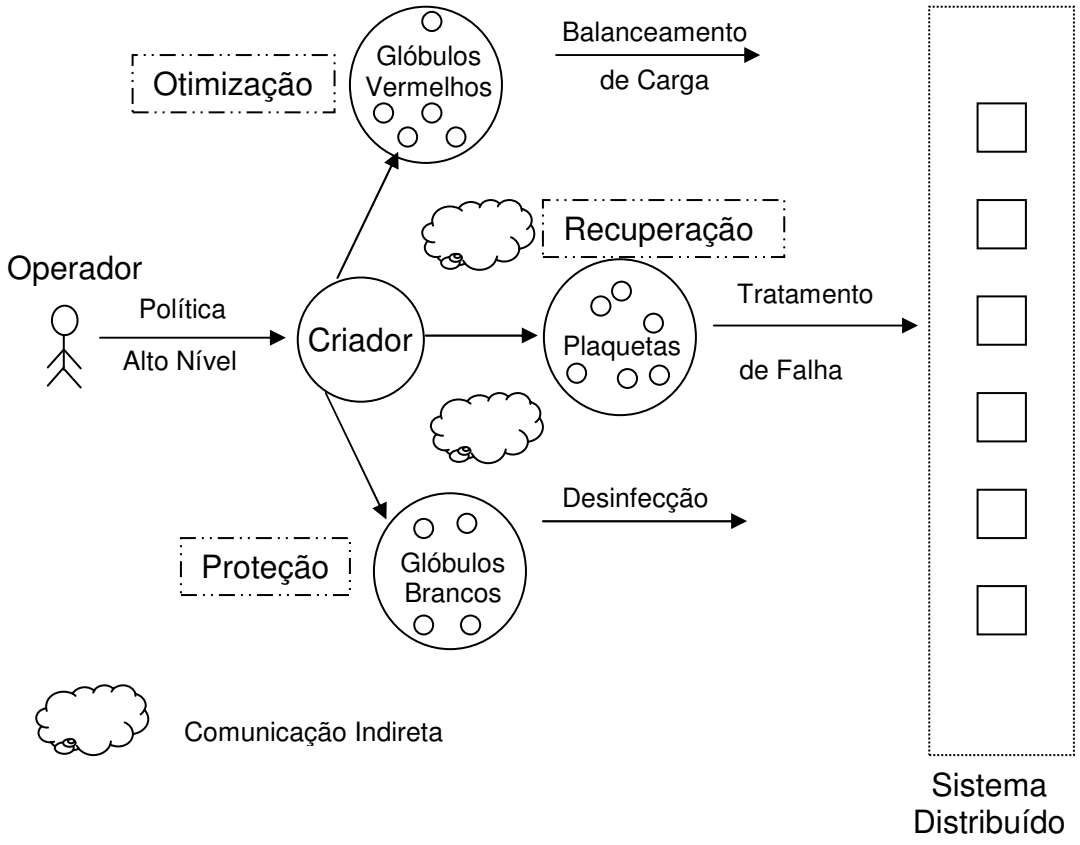

Figura 5.1 – Solução de autogerenciamento com o CIACOM.

O operador dita a política de alto nível (parâmetros) e aciona o programa Criador que cuida de criar três conjuntos de agentes: otimização (glóbulos vermelhos), recuperação (plaquetas) e proteção (glóbulos brancos). Os agentes de otimização são responsáveis pelo balanceamento de carga, os agentes de proteção são responsáveis pela identificação de nós (computadores) infectados e sua desinfecção e os agentes de recuperação são responsáveis pela identificação e tratamento das falhas dos nós.

O comportamento do glóbulo vermelho, de balanceamento de carga, tem como inspiração a busca de comida pelas formigas e seu posterior depósito em uma pilha. Os glóbulos vermelhos, uma vez criados com uma política de distribuição de processos (alta, média ou baixa), passam a executar o comportamento de busca por nós carregados que consiste em migrar pelo espaço de busca (nós ou computadores da rede) e através de sensores identificar os nós carregados. Caso identifique um nó carregado executa uma ação que consiste na captura de um processo deste nó que passa a ficar menos carregado. O estado local deste glóbulo vermelho passa de ”livre” a “com processo” e passa a executar outra ação de busca, mas desta feita para encontrar um nó descarregado e depositar lá o processo transferido. A atuação de um enxame de glóbulos vermelhos faz emergir o balanceamento de carga.

Um nó é considerado carregado, descarregado ou normal se o número de processos no nó divididos pelo número máximo de processos for maior ou igual a um limiar máximo, for menor ou igual a um limiar mínimo ou estiver entre estes dois limiares, respectivamente.

Além do comportamento de busca, o glóbulo vermelho pode depositar feromônios (comunicação indireta) que se evaporam com o decorrer do tempo, de forma a sinalizar que outros agentes passaram por este nó e permitir que outro glóbulo vermelho, que também passe por aí, decida se deve também executar uma ação de busca ou simplesmente migrar direto para outro nó, ampliando a cobertura do conjunto de glóbulos vermelhos e otimizando o processo. Pode perceber também os feromônios depositados por outros tipos de agentes e tomar decisões, como por exemplo, evitar (repulsão) nós que foram marcados como infectados.

O comportamento do glóbulo branco, de proteção contra vírus, se caracteriza pela identificação de nós infectados e por sua posterior desinfecção. Tem como inspiração a busca de comida pelas formigas (caça ao vírus) e no retorno com a comida para o ninho com o depósito de feromônios marcando o caminho. Ou seja, o enxame de glóbulos brancos é distribuído pelo espaço de busca (rede de computadores)

monitorando através de sensores para identificação de nós infectados. Caso os glóbulos brancos identifiquem nós infectados, depositam feromônios para marcar esta condição. Com isso, outros glóbulos brancos que migravam de forma aleatória, são atraídos (atração) para contribuir com a identificação de outros vírus, reforçar a condição de nó infectado e iniciar o processo de desinfecção. O glóbulo branco que identificou um nó infectado deposita o feromônio e continua vagando pelo espaço de busca para identificar outros nós infectados.

O comportamento do glóbulo branco, de desinfecção do nó, se caracteriza pela atração exercida pela quantidade de feromônios depositados por outros glóbulos brancos. Com isso o glóbulo branco migra para o nó infectado ou sua vizinhança e executa a desinfecção do nó. Esta ação inclusive pode determinar a retirada temporária deste nó da rede.

Observe que para a inspiração biológica inicial no sistema circulatório acrescentou-se, na questão específica de depósitos químicos, inspiração no comportamento das formigas por esse ser de fácil compreensão e implementação, além de apresentar vasta literatura sobre o tema. Ou seja, o entendimento e aprofundamento de como realmente é que agem as células sanguíneas foi relegado a um segundo plano, em prol da simplificação, sem, no entanto, perder o mérito do modelo se basear no sistema circulatório. No caso específico do comportamento real das plaquetas que emitem sinais químicos atraindo outras plaquetas e posteriormente glóbulos vermelhos, por estes apresentarem tamanho muito superior aos das plaquetas, formando o tampão plaquetário, em nosso estudo de caso optou-se de mostrar este comportamento de atração de tipos diferentes de células sendo exercido pelo glóbulo branco. Nada impede que se opte na modelagem da propriedade de autorrecuperação por um binômio identificação/recuperação de forma que as plaquetas identifiquem falhas e solicitem auxílio de outras plaquetas para recuperar o sistema destas falhas, assim como sinalizar para os glóbulos brancos que esta falha pode ser porta de entrada de vírus, bem como esta informação servir para os glóbulos vermelhos ficarem atentos sobre os problemas encontrados.

A ênfase do modelo proposto é a sobrevivência do sistema, ou seja, seu funcionamento estável aumentando sua disponibilidade, e não seu funcionamento ótimo.

Na Figura 5.2 são apresentadas ideias que se opõem uma a outra. Inibição em oposição a estímulo, castigo em oposição à recompensa, repulsão em oposição à

atração, simpático em oposição ao parassimpático, competição em oposição à cooperação. Estas ideias servem de base para diversos algoritmos na área de inteligência artificial incluindo algoritmos de busca e aprendizagem. Percebe-se pela descrição deste estudo de caso que a oposição atração/repulsão é amplamente utilizada no modelo. Podemos considerar o depósito de feromônios pelas formigas como um estímulo que atrai outras formigas, caracterizando-se como um mecanismo de cooperação. Adicionamos então a idéia oposta de inibição para repelir agentes, de modo ora a acelerar (catalisar) o processo, ora para controlar o sistema. Desta forma busca-se que aja interação no sistema, com forças que se opõem e que fazem emergir o autogerenciamento (regulação) do sistema.

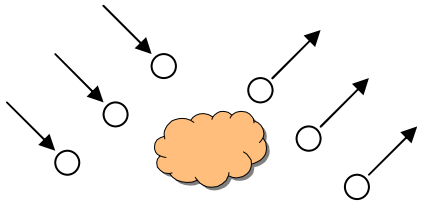

Figura 5.2 – Atração (estímulo) e Repulsão (inibição).

5.1.1 Testes

Utilizando o ambiente NetLogo foram desenvolvidos aplicativos para simulação do CIACOM com foco inicial na propriedade de auto-otimização para balanceamento de carga.

Estes programas pretendem ilustrar os resultados do modelo CIACOM através da simulação da evolução dos agentes glóbulos vermelhos como se fosse uma população isolada num certo ambiente, executando a tarefa de balanceamento de carga segundo uma política global ditada por um operador.

Um dos fatores que influenciam a variação do número de indivíduos é o balanço entre natalidade e mortalidade, que supostamente é positivo (caso contrário a espécie extingue-se). Desta forma é que a apoptose está inserida no modelo, pois um indivíduo, durante seu ciclo de vida, nasce, se reproduz (ou não) e morre.

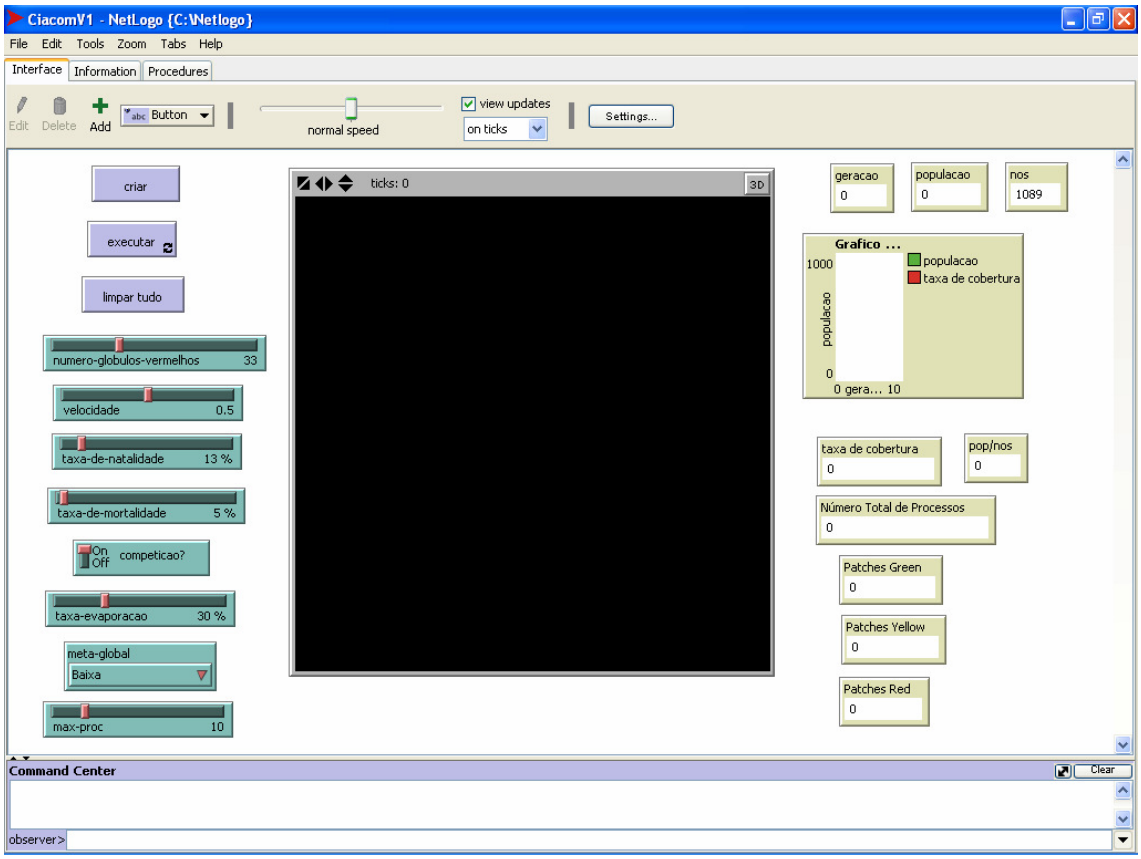

Figura 5.3 – Tela do aplicativo CIACOM_V1 no NetLogo.

Outro fator que se faz necessário para evitar o descontrole do crescimento populacional e que deve ser incorporado ao modelo é o efeito de competição pelos mesmos recursos, que limita o crescimento de qualquer população.

Neste modelo supõe-se que cada território (nó da rede) só pode ser ocupado por um indivíduo e que os indivíduos podem mover-se entre os vários territórios durante o seu período de vida. Caso dois indivíduos se encontrem no mesmo território, um deles morre.

O NetLogo fornece uma maneira simples de modelar os indivíduos de uma espécie (chamados *turtles*) e os territórios (chamados *patches*), pois vem munido de um mundo retangular dividido em quadrados (*patches*) de coordenadas inteiras.

Como pode ser visto na Figura 5.3 foi definida área quadrada contendo 33 quadrados na vertical e na horizontal, perfazendo um total de 1089 quadrados (*patches*). Cada um dos quadrados representa um nó da rede de simulação, sendo considerados vizinhos os oito quadrados com os quais tem fronteira (Figura 5.4). Ou seja, cada computador da rede é representado por um quadrado que está conectado a seus vizinhos simulando uma rede de computadores. No NetLogo as coordenadas (x,y)

deste quadrado são mapeadas como um toro (*doughnut, bagle* ou rosquinha), de modo que sempre existirão oito vizinhos mesmo em quadrados localizados na moldura da área quadrada. Além disso, o movimento dos indivíduos quando localizados em algum extremo é circular, ou seja, um indivíduo que está na extrema direita do quadrado e se move um pouco mais à direita aparece no extremo esquerdo do quadrado.

V1	V2	V3
V8	Q	V4
V7	V6	V5

Figura 5.4 – Vizinhança de um quadrado (*patch*) Q.

Uma vez apresentados o elenco e o cenário, é apresentado o enredo:

1. Coloca-se um número inicial de glóbulos vermelhos (*turtles*) ao acaso no mundo. Ao contrário dos *patches* as coordenadas de cada indivíduo não têm de ser inteiras;
2. Cada indivíduo tem uma certa probabilidade de se reproduzir (taxa de reprodução). Em cada geração, cada indivíduo pode dar origem no máximo a um novo descendente, o qual é colocado num ponto ao acaso no interior da circunferência de raio 2 centrada no progenitor;
3. Em cada geração os glóbulo vermelhos se deslocam (migram) no mundo com uma certa velocidade *v*, ou seja, se movem no seu ambiente. Neste caso a posição final de cada glóbulo vermelho é um ponto escolhido ao acaso no interior da circunferência de raio *v* centrada na posição inicial de cada indivíduo. Desta forma, está sendo simulado o movimento dos glóbulos vermelhos como agentes móveis dentro de seu ambiente (sangue e vasos sanguíneos);
4. Opcionalmente pode-se introduzir um fator de competição. Se dois glóbulos vermelhos ocupam o mesmo território (*patch*), um deles, escolhido aleatoriamente, morre.

5.1.2 Interação com o Modelo

A interação com os modelos em NetLogo é feita através de uma interface gráfica que se encontra dividida em três partes (controle, configuração e apresentação) segundo um código de cores:

- Controle: Botões que preparam, começam, param ou colocam o modelo nas condições iniciais (cor azul).
- Configuração: Cursores e interruptores que alteram os parâmetros do modelo (cor verde).
- Apresentação: Monitores e gráficos que apresentam os resultados de uma forma compreensível para o homem (cor bege).

Para iniciar este modelo em NetLogo é necessário clicar primeiro em criar.

5.1.3 Controle do Modelo

Quando um botão é pressionado, o modelo responde com uma ação, de acordo com o tipo de botão.

Botões do tipo *uma vez* executam uma ação (pedaço de código) uma só vez. Quando a ação termina, o botão volta ao normal.

Botões do tipo *para sempre* executam repetidamente uma ação (porção de código). Quando se quer terminar a ação, deve-se pressioná-lo novamente. Este retorna ao estado inicial depois de terminada a ação que está sendo executada naquele momento.

Nosso modelo, assim como a maior parte dos modelos, tem um botão do tipo *uma vez*, chamado criar, e um botão do tipo *para sempre* chamado executar. Nosso modelo apresenta também um botão do tipo *uma vez* chamado limpar tudo.

5.1.4 Configuração do Modelo

Existem essencialmente duas formas de variar os parâmetros de um modelo em NetLogo, através de cursores ou de interruptores. O seu propósito é testar diferentes cenários ou hipóteses. A alteração dos parâmetros e a posterior execução do modelo permitem ver como este reage a essas alterações, e desta forma obter uma compreensão mais profunda do fenômeno que está sendo modelado.

Cada cursor tem um intervalo de valores que pode ser ajustado. À medida que se move o marcador do valor mínimo ao valor máximo, o parâmetro toma o valor do lado direito do cursor. Por exemplo, o cursor numero-globulos-vermelhos tem o valor

mínimo de 0 e o valor máximo de 100. O modelo pode ser executado com qualquer número inicial de indivíduos entre 0 e 100.

Os interruptores são usados para explorar universos alternativos. Por exemplo, o interruptor competicao? permite ver como a existência ou ausência de competição afeta a população de glóbulos vermelhos.

5.1.5 Apresentação dos Resultados

Um dos propósitos da criação de modelos é a integração de dados sobre um assunto ou tópico que seriam difíceis de obter experimentalmente, quer seja por questões de recursos, éticas, dificuldade no controle de algumas variáveis ou outras. O NetLogo tem duas formas de apresentar os dados ao usuário: gráficos e monitores.

Os gráficos permitem visualizar de que forma as variáveis em estudo estão relacionadas, por exemplo, como é que a população varia com o tempo (geração). Por outro lado os monitores permitem visualizar os valores das variáveis em cada instante.

5.1.6 Execução e Aperfeiçoamento do Modelo

É feito um sorteio aleatório para preencher cada nó (*patch*) com o número mínimo de zero e máximo de dez processos (máximo configurável). Para guardar esta informação foi criada uma variável local para cada *patch* denominada de n_proc.

Na inicialização do aplicativo NetLogo CIACOM_V1 (tela na Figura 5.5 e código no Anexo E) é executada uma regra baseada no número de processos. Caso este número seja superior a oito, o *patch* é considerado carregado e pintado na cor vermelha. Caso o número de processos esteja entre seis e oito, o *patch* é considerado mediamente carregado e é pintado de amarelo. Finalmente, se o número de processos for inferior a seis, o *patch* é considerado pouco carregado e é pintado de verde. O objetivo é balancear carga, ou seja, migrar processos de *patches* mais carregados e transferi-los para *patches* com mais disponibilidade, de forma a promover uma distribuição de carga mais homogênea.

Para resolver este problema que classificamos como de otimização, criamos alguns glóbulos vermelhos representados no NetLogo como *turtles*, e os distribuímos aleatoriamente pelos *patches.* Eles são dotados de sensores para perceber se um *patch* está carregado (cor vermelha), mediamente carregado (cor amarela) ou com pouca carga (cor verde). Além disso, possuem variável local para armazenar a política ditada pelo operador: alta para permitir carregamento qualquer, média para provocar a migração de processos de nós carregados para nós mediamente carregados e baixa para provocar a migração de processos de nós carregados ou mediamente carregados para nós com pouca carga.

Quando são criados, os glóbulos vermelhos nascem com uma cópia desta política (variável global) para sua variável local. Como cometem apoptose após o fim de seu ciclo de vida (tempo de vida aleatório), novas gerações de agentes são criadas e caso a política tenha mudado, esta nova geração se atualiza. Como existe uma coexistência de uma geração antiga com uma nova, o comportamento global do sistema muda mais suavemente.

São dois os comportamentos do glóbulo vermelho. No início, quando são criados, os glóbulos estão "vazios", ou seja, não carregam nenhum processo. O primeiro comportamento equivale a uma formiga em busca de alimento. Os glóbulos se movimentam seguindo uma estratégia de migração (aleatória) pelos *patches* em busca de nós muito carregados ou muito e mediamente carregados, caso a política seja de média ou baixa, respectivamente. Caso encontrem "capturam" um processo deste nó e o transportam para outro nó que esteja menos carregado e o deposita lá. O segundo comportamento é similar a uma formiga que carrega o alimento de volta para seu ninho.

Estas regras simples executadas por um conjunto de glóbulos vermelhos fazem emergir o resultado (comportamento) global esperado de balanceamento de carga.

Além disso, para garantir um contingente mínimo de glóbulos vermelhos, estes são periodicamente reproduzidos (clonados), tomando-se o cuidado de configurar os parâmetros de taxa de crescimento populacional, taxa de natalidade e taxa de mortalidade, de forma a resultarem em um valor positivo.

A partir da verificação experimental que a estratégia de reprodução resultava em um crescimento exponencial da população, foi necessário acrescentar o controle biológico

de competição. Este controle consiste na eliminação de um dos glóbulos vermelhos quando dois deles se encontram em um mesmo nó.

Conforme pode ser visto na Figura 5.3, a tela está dividida em quatro partes: comandos, configurações, monitoração e visualização. A área de comando apresenta os comandos Criar, Executar e Limpar tudo. O comando Criar inicializa as variáveis e cria os agentes e os *patches*. O comando de execução dispara o *loop* eterno que simula a execução em paralelo dos agentes e de seu ambiente (*patches)*. Para interromper a simulação basta pressionar novamente o botão. O comando Limpar tudo é autoexplicativo.

A área de configurações estabelece valores iniciais para as variáveis de controle que podem ser alteradas em tempo de execução.

A área de visualização permite que se acompanhe a evolução dos agentes e sua movimentação pelos nós, simulando uma rede de computadores com agentes móveis.

A área de monitoração possibilita o acompanhamento de grandezas como a população total de agentes, total de nós da rede, quantidade de gerações daquela população, gráfico de evolução de população no tempo (gerações), taxa de cobertura, relação de população por total de nós, número total de processos, número de *patches* carregados (cor vermelho), mediamente carregados (cor amarela) e pouco carregados (cor verde).

As variáveis foram configuradas inicialmente para os seguintes valores: número inicial de glóbulos vermelhos (33), velocidade de migração ou movimentação (0.5), taxa de natalidade (13%), taxa de mortalidade (5%), competição (ON), taxa de evaporação (30%), meta global (baixa) e número máximo de processos (10).

Na Figura 5.5 é apresentada a tela após ser pressionado o botão Criar. Os quadrados coloridos correspondem aos *patches* e os círculos azuis aos glóbulos vermelhos "vazios" distribuídos aleatoriamente. Na área de monitoração podem ser observados o número total de processos (4897), o número de nós verdes pouco carregados (655), o número de nós amarelos mediamente carregados (327) e o número de nós vermelhos muito carregados (107), relação população/ nó (3,03 %) e número total de nós (1089).

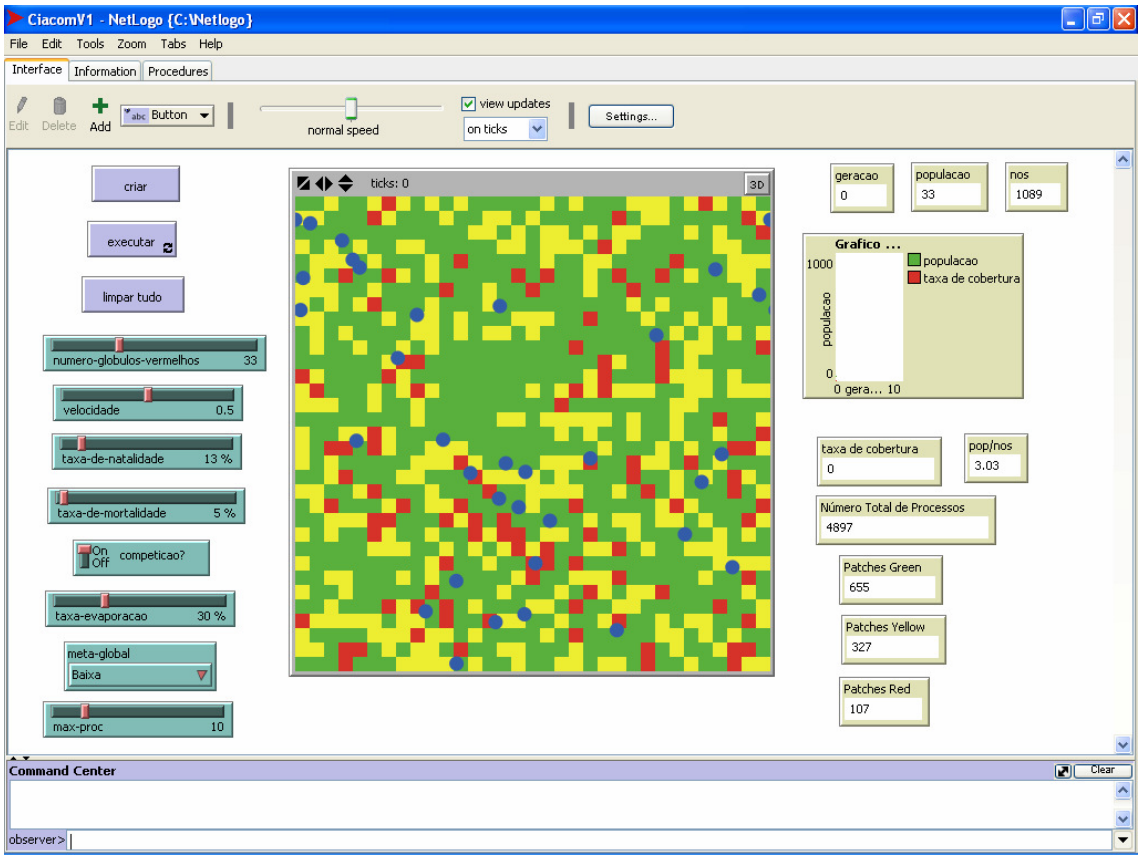

Figura 5.5 – Tela do aplicativo CIACOM_V1 após pressionar o botão Criar.

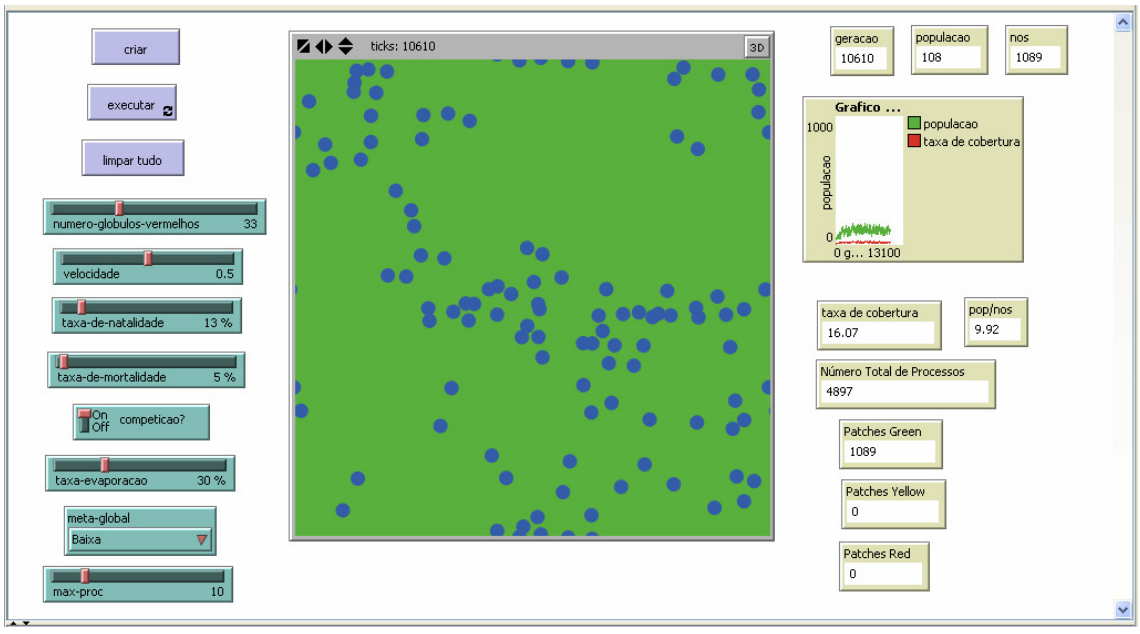

Figura 5.6 – Convergência do balanceamento de carga no aplicativo CIACOM_V1.

Pressionando o botão executar, após cerca de 10000 gerações, emerge toda verde a área de visualização da tela, com a relação pop/nós com valor 9,92% e uma população de glóbulos vermelhos de 108, conforme Figura 5.6. Apesar de não estar

representado nesta figura, um glóbulo vermelho quando captura um processo e deixa de ficar “vazio”, muda da cor azul para magenta, indicando esta condição.

Se passarmos o controle de competição para OFF ocorre o fenômeno de descontrole populacional, conforme Figura 5.7.

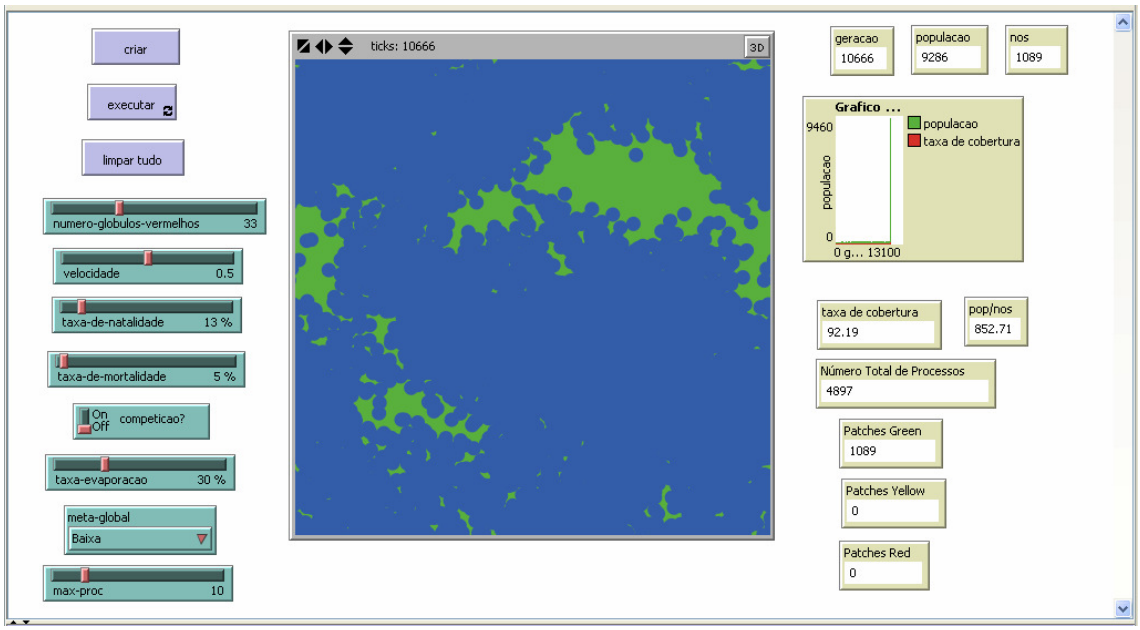

Figura 5.7 – Descontrole populacional (opção competicao? OFF).

O NetLogo possui três abas: *Interface*, *Information* e *Procedures*. A aba *Interface* é a que acompanhamos até agora que serve tanto para montar os componentes da tela como para executar os aplicativos para realizar a simulação propriamente dita. A aba *Information* serve para explicarmos o funcionamento do aplicativo CIACOM_V1 e de seus monitores (saídas) e parâmetros de configuração (entradas). A aba *Procedures* serve para a construção do programa propriamente dito, definição de variáveis globais, locais, métodos e código associado aos botões da interface, leitura de configurações iniciais, atualização de monitores e gráficos.

Após ser pressionado o botão Executar é iniciado o *loop* de cada *turtle,* correspondendo aos glóbulos vermelhos, que executa os módulos de análise, depósito, movimento, reprodução, morte e choque por competição. Ou seja, é feito o sensoriamento do carregamento de um nó pelo agente que decide se deve retirar um processo de um nó carregado, se ele mesmo estiver “vazio” ou se deve colocar um processo em um nó, caso possua um. Em seguida o agente migra para outro nó de forma aleatória, mas restrita à vizinhança do nó, e repete a operação de análise. Neste

intervalo ele deposita feromônios (antes de migrar), se reproduz de acordo com a taxa de natalidade, comete apoptose de acordo com a taxa de mortalidade e em caso de “choque” (competição ativada) com outro agente em um mesmo nó, é feito um sorteio, e um dos dois é eliminado (desde que não contenha um processo).

Nesta versão do programa já foi previsto e implementado o depósito químico pelo agente e a correspondente evaporação, mas ainda não está sendo usado efetivamente.

Neste ponto, podem ser feitas observações importantes. O aplicativo é sensível à diferença entre as taxa de natalidade e mortalidade, que deve ser positiva. Diferenças pequenas provocam a indesejável extinção dos glóbulos vermelhos e o fracasso do modelo. Por exemplo, a redução da taxa de natalidade de 13% para 10% já provoca este efeito, mantida a taxa de mortalidade em 5% (Figura 5.8).

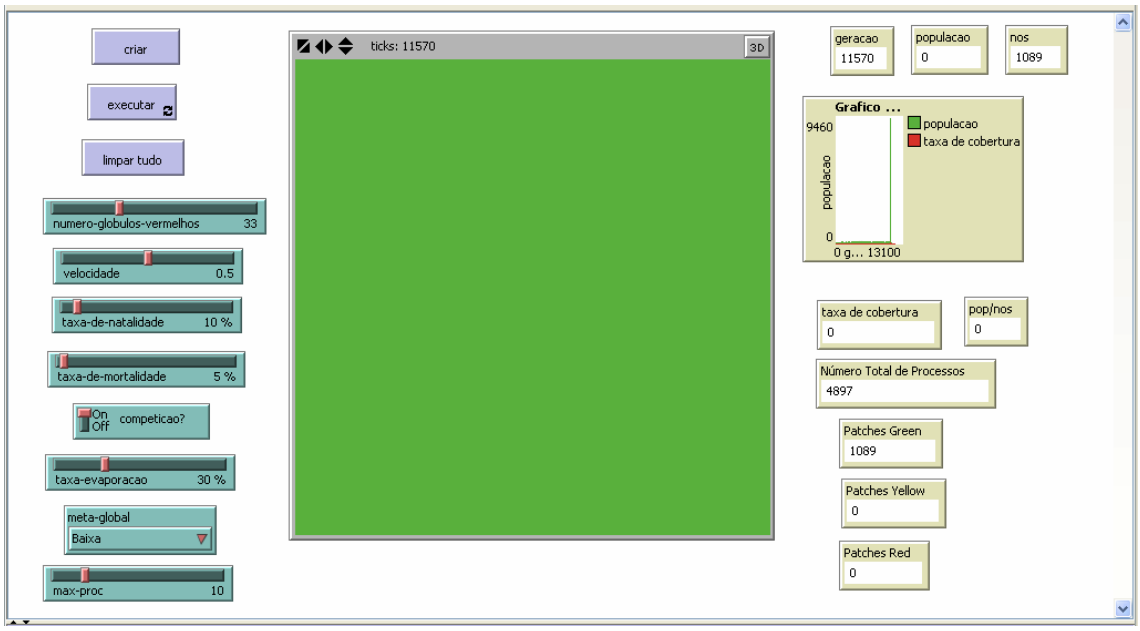

Figura 5.8 – Extinção dos Glóbulos Vermelhos por pequena diferença positiva de crescimento populacional (5%).

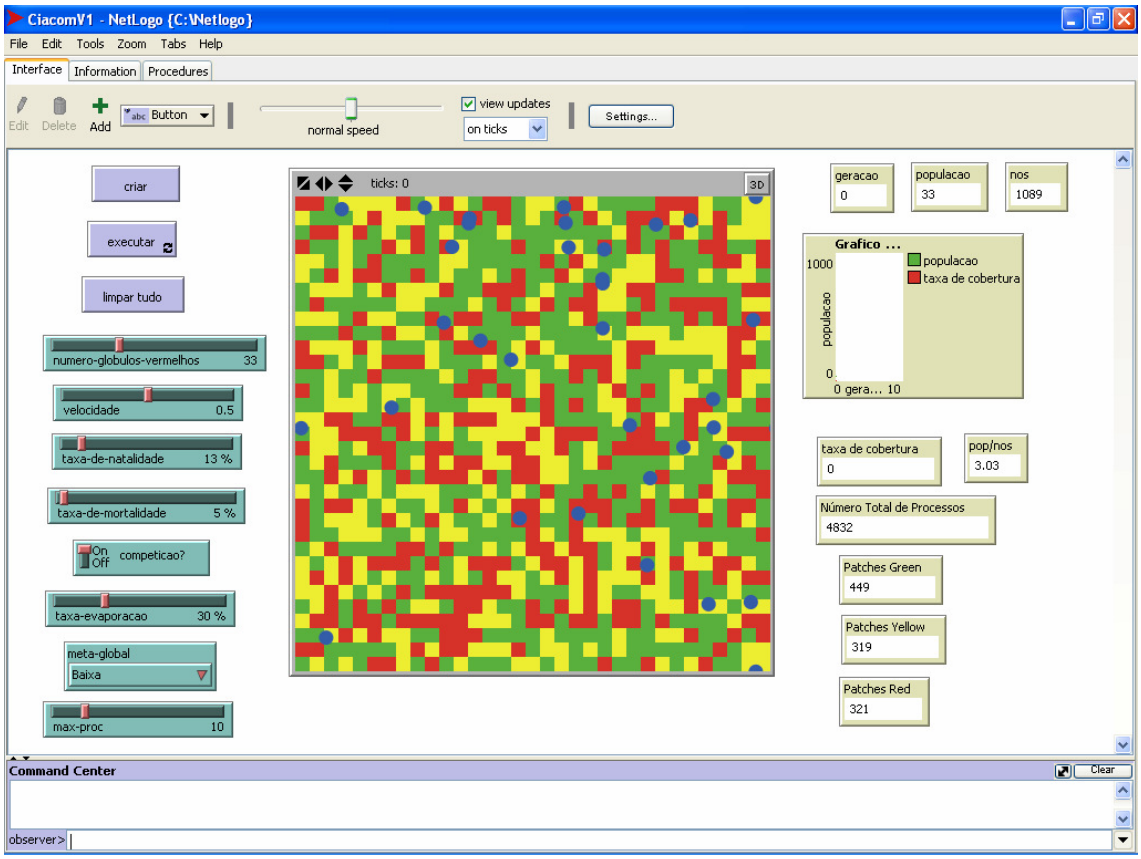

Figura 5.9 – Experimento com ambiente com mais carregamento.

A quantidade de *patches* de cor verde, vermelha e amarela influenciam na emersão do desejado balanceamento de carga. Por exemplo, a convergência (todos os *patches* verdes) é atingida quando 64,1% de *patches* são verdes, 32,3% de *patches* são amarelos e 12,5% de *patches* são vermelhos. Mas não há convergência, ou seja, o sistema oscila, para valores de 49,5%, 37,2% e 22,2% (Figura 5.9). Além disso, os glóbulos vermelhos sem processo (“vazios”) acabam sendo extintos, só sobrando glóbulos vermelhos com processo (cor magenta), causando um *deadlock* no balanceamento de carga, ou seja, glóbulos vermelhos que não têm onde depositar seus processos, pois a regra dita uma meta a ser cumprida que está acima da capacidade do sistema de gerenciamento para atendê-la (Figura 5.10).

Para resolver este problema foram feitas alterações e implementada nova versão do aplicativo (CIACOM_V2). Nesta versão foi introduzida a noção de “mico preto” inspirada em um jogo infantil em que os participantes tinham que fazer duplas de cartas com casais de animais. Quem tinha a carta com o “mico preto” queria se livrar dela, pois esta não possuía nenhum par (fêmea) correspondente. Então, caso um glóbulo vermelho com processo tente achar um nó para depositá-lo e não o consiga em mais que *n* tentativas, deposita o processo em qualquer nó para se “livrar do mico

preto” evitando assim, o *deadlock*, ou seja, glóbulos vermelhos eternamente acoplados a processos (Figura 5.11).

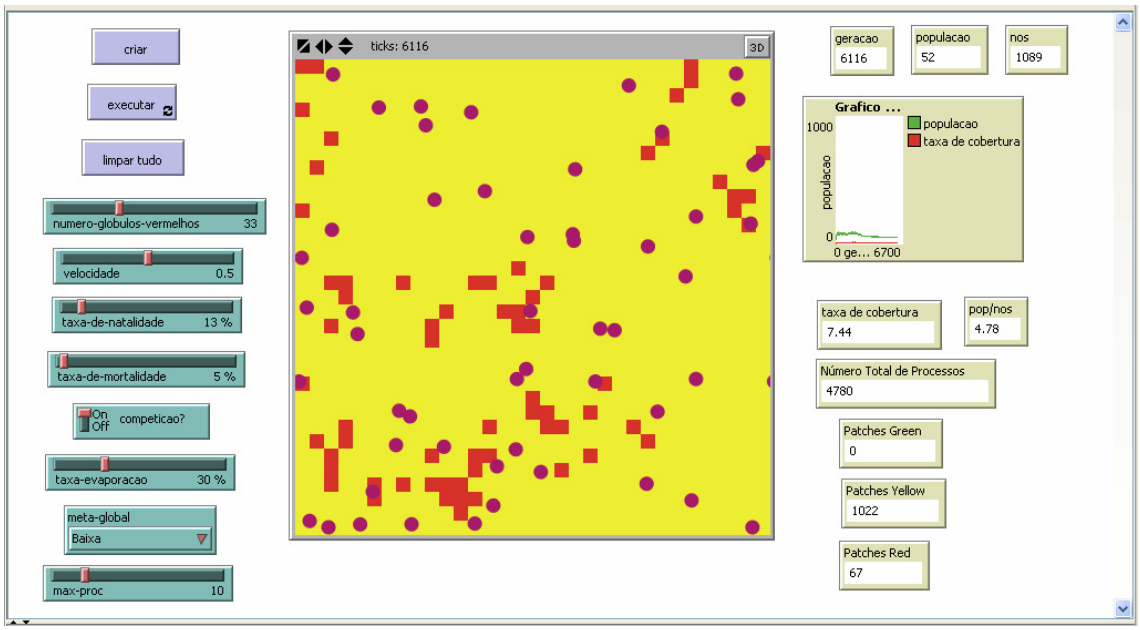

Figura 5.10 – *Deadlock* dos Glóbulos Vermelhos.

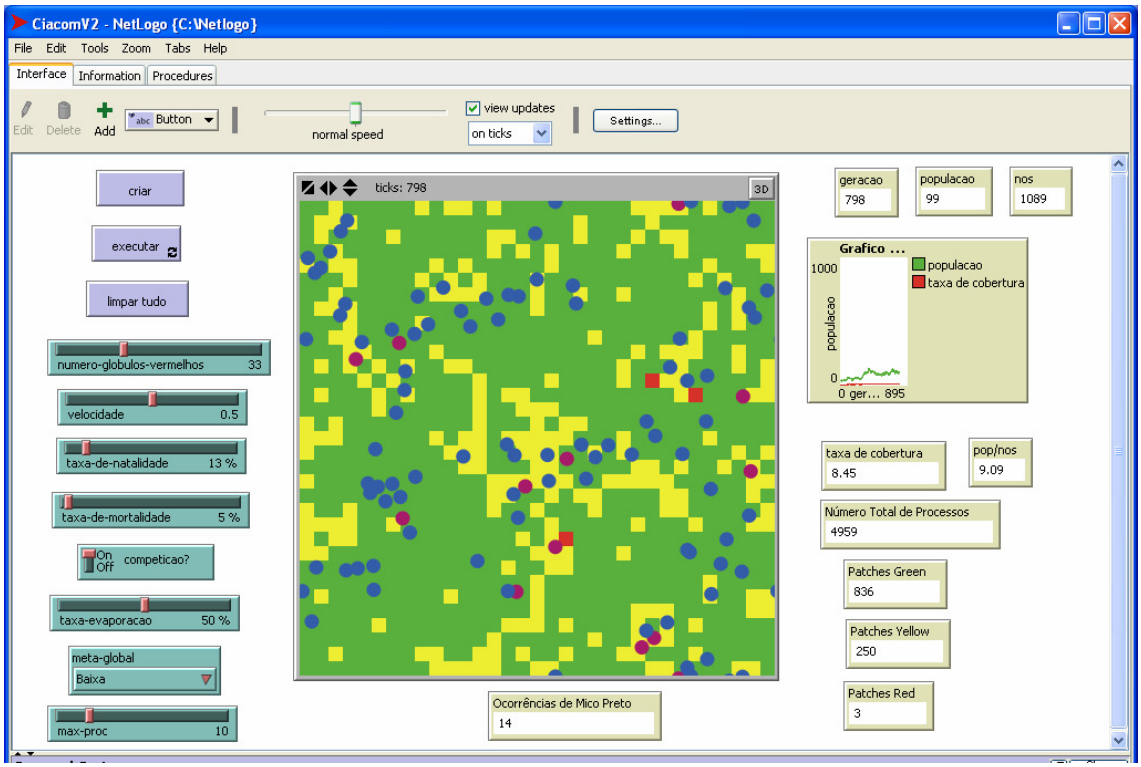

Figura 5.11 – CIACOM_V2 solução do *deadlock* com “mico preto”.

O operador deve perceber que a frequência de ocorrências de mico preto caracteriza um nível alto de carregamento do sistema, o que dificulta o balanceamento. Neste

caso pode optar por afrouxar a meta global (ex: passar de “baixa” para “média”) de forma a estabilizar o sistema ao obter uma boa solução e não uma ótima solução.

Na versão 3 do CIACOM (CIACOM_V3) foi introduzido o rastro (depósito químico) com evaporação para otimizar a migração dos glóbulos vermelhos. Eles percebem a presença do rastro e, se este for maior do que um determinado limiar, passam direto (são “repelidos”) para outro nó melhorando a cobertura dos agentes.

Na versão 4 do CIACOM (CIACOM_V4) foram introduzidos os glóbulos brancos e uma variável local boolena “infectado?” para indicar um nó que deve ser evitado pelos glóbulos vermelhos. Para criação desta nova “espécie” foi usada a primitiva “breed”, da linguagem de programação interpretada do NetLogo. Cerca de 20% dos nós infectados são aleatoriamente distribuídos e mostrados com as mesmas cores originais de carregamento, só que esmaecidas, ou seja, verde claro, amarelo claro e vermelho claro. Dividimos a execução por espécie, ou seja, dois botões: um para a execução dos glóbulos vermelhos e outro para os brancos (Figura 5.12).

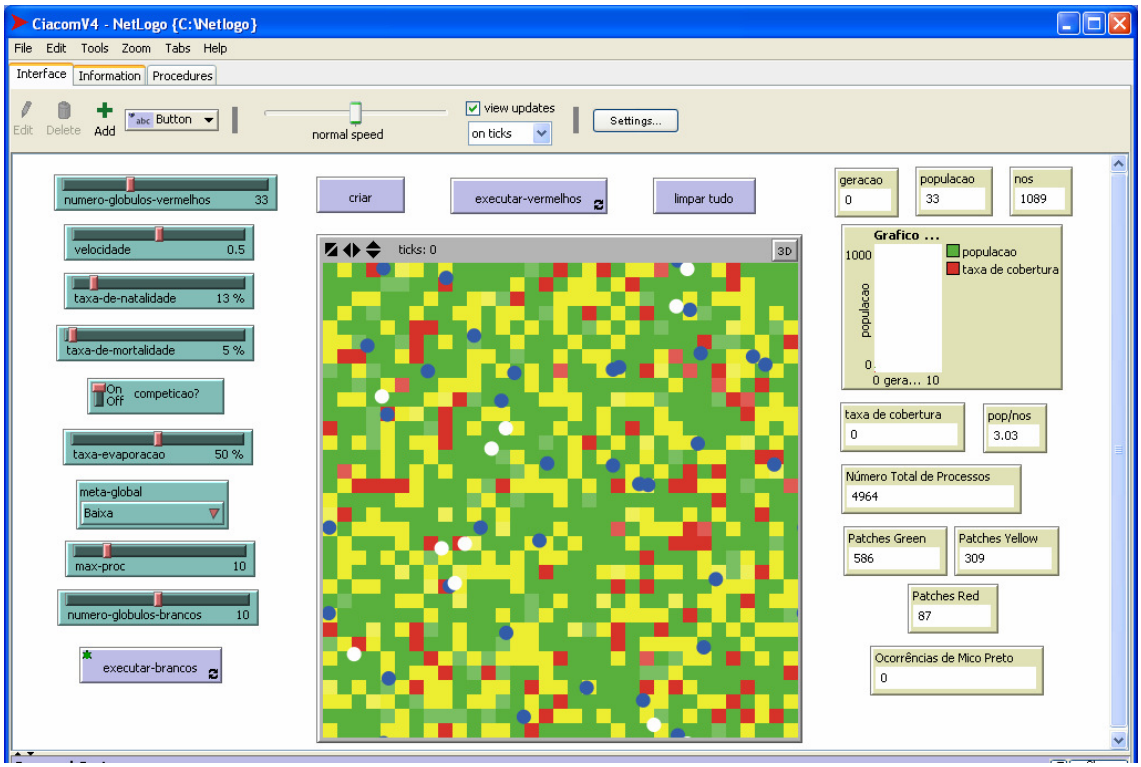

Figura 5.12 – CIACOM_V4 com glóbulos brancos e ambiente infeccionado.

Os glóbulos brancos devem identificar um nó como infectado e atrair (ou clonar) outros glóbulos brancos para cercá-lo e imunizá-lo (isolá-lo). Eles têm um comportamento aleatório de movimentação (busca) que passa a ser fixo quando identificam um nó

infectado. Este nó é então cercado para que seja desinfectado (curado). Então, o comportamento do glóbulo branco volta ao padrão original de movimentação (busca) e seu número é reduzido para caracterizar a cura, pois, assim como no corpo humano eles se multiplicam quando existe uma infecção, mas retornam ao seu número médio normal assim que cessa a infecção.

Assim como os glóbulos vermelhos, os glóbulos brancos são gerados periodicamente, proliferam em caso de infecção, posteriormente morrendo para controlar a sua população.

Para explorar melhor a atuação do glóbulo branco, na versão 6 do CIACOM (CIACOM_V6), optou-se pelo seu tratamento de forma isolada e independente do glóbulo vermelho. Assim como para o caso do balanceamento de carga, onde os nós apresentam um entre três estados possíveis (Alto – carregado, Médio – mediamente carregado e Baixo – pouco carregado), para o caso da desinfecção considerou-se que os nós podem assumir três estados no que diz respeito à infecção: suscetíveis, infectados e resistentes. Após clicar no botão Criar, assume-se que inicialmente todos os nós estão suscetíveis (Figura 5.13), ou seja, têm potencial de ser infectados, sendo que nenhum deles está resistente ao vírus. Aqueles em que há presença de vírus são considerados infectados (número igual ao de vírus). Associou-se a cor verde aos suscetíveis, vermelha aos infectados e cinza aos resistentes. Esta versão possui também agentes vírus que são representados graficamente como círculos pretos e os glóbulos brancos representados por círculos brancos.

Novos parâmetros foram criados: taxa de infecção, taxa de resistência e taxa de recuperação. Cada um deles está associado a um estado, respectivamente: Infectado, Resistente e Suscetível. Com isso, é possível simular a disseminação do vírus, assim como a desinfecção do ambiente pelos glóbulos brancos. Outros parâmetros, como velocidade do vírus e do glóbulo branco, permitem modificá-los em tempo de execução.

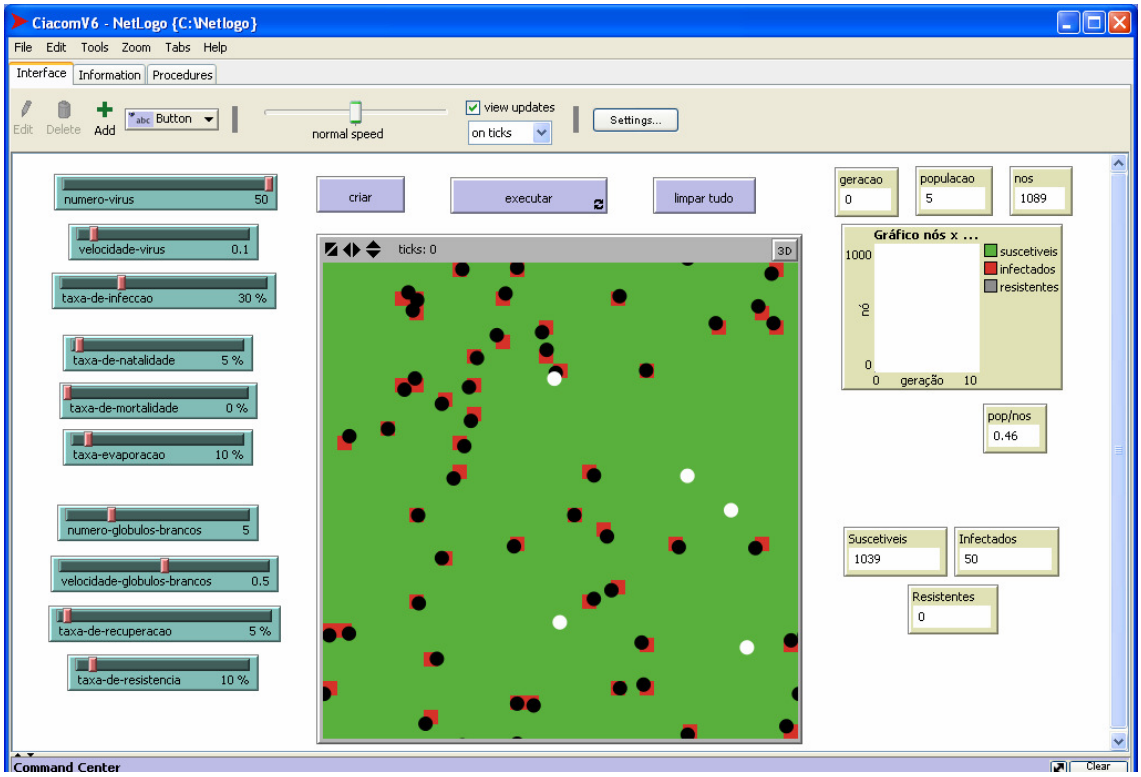

Figura 5.13 – CIACOM_V6 com glóbulos brancos, vírus e ambiente suscetível: modelo de infecção SIR (Suscetível, Infectado ou Resistente).

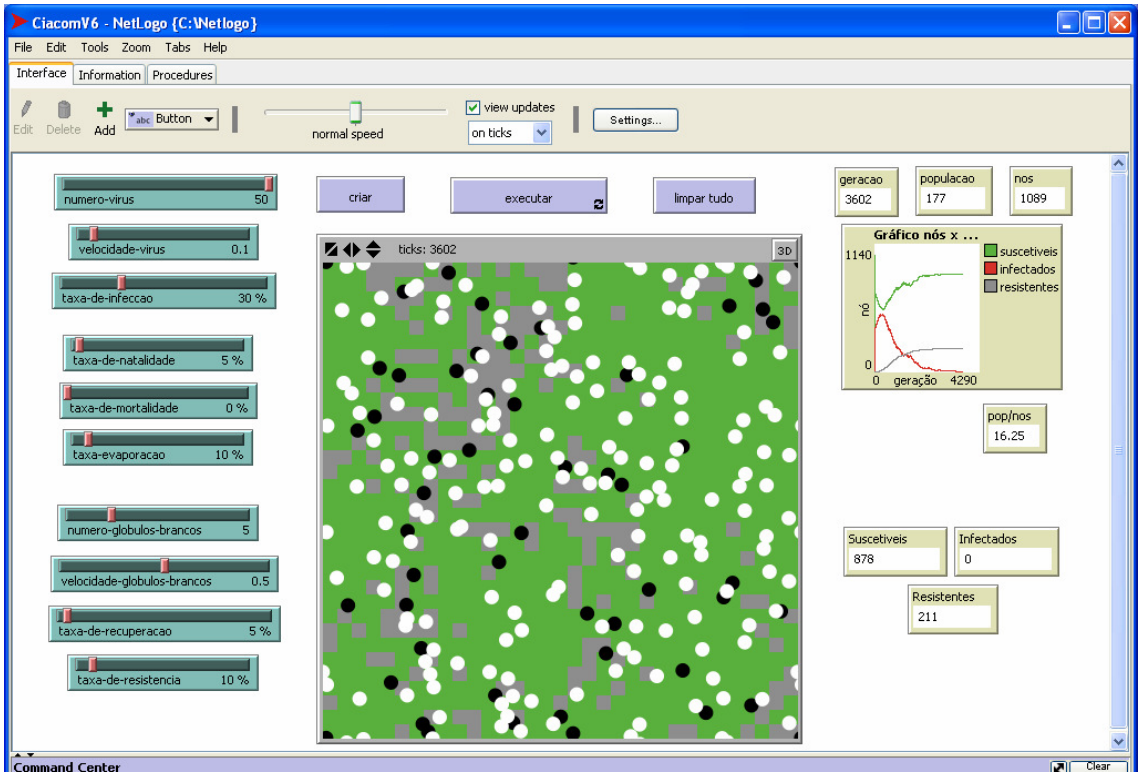

Figura 5.14 – CIACOM_V6 com glóbulos brancos desinfectando o ambiente.

Ao clicar no botão executar (Figura 5.14) os glóbulos brancos se movimentam pelo ambiente para detectar contaminação e, em caso afirmativo, com uma determinada

probabilidade (taxa-de-resistencia) tornam o local protegido ou limpo (taxa-de-recuperacao). Enquanto isso, o vírus com uma determinada probabilidade (taxa-de-infeccao) contamina o ambiente desde que este não esteja protegido, ou seja, esteja limpo.

A versão 9 do CIACOM (CIACOM_V9) foi baseado em um artigo sobre desinfecção [79] no qual também existem três estados: Limpo, Contaminado e Protegido. Entretanto, diferentemente da versão 6 assume-se inicialmente que todos os nós estão contaminados e não somente aqueles em que existe a presença do vírus (Figura 5.15). Os nós com presença de glóbulos brancos são considerados como protegidos e nenhum nó é considerado como limpo.

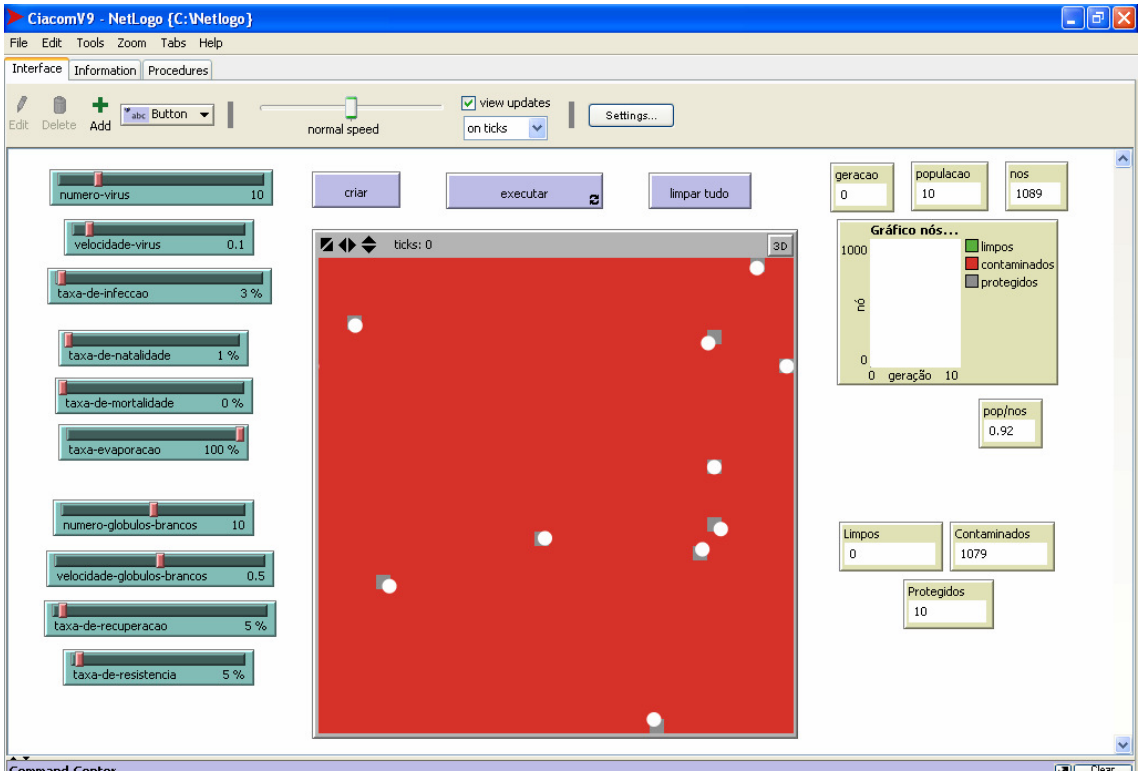

Figura 5.15 – CIACOM_V9 com glóbulos brancos e ambiente contaminado: modelo de infecção LCP (Limpo, Contaminado ou Protegido).

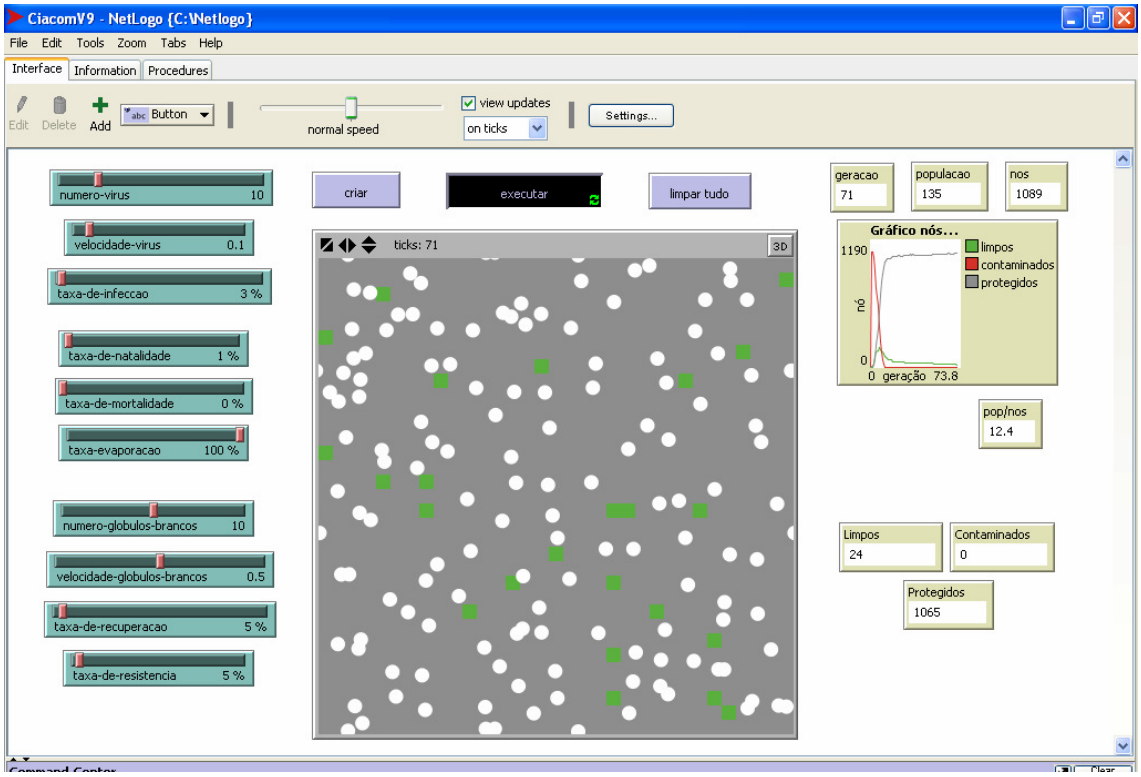

Figura 5.16 – CIACOM_V9 com glóbulos brancos e ambiente protegido: modelo de infecção LCP (Limpo, Contaminado ou Protegido).

Tabela 5.1 – Algoritmo de descontaminação baseado em [79].

```
Comportamento Limpar
Início
/* para um glóbulo branco b chegando em um nó x */
Se b está sozinho
    Executar Ação A
Senão
   Escolher um líder.
   Se b é líder
      Executar Ação A
   Senão
       Terminar.
Fim
Definição Ação A
Início
  Limpar x.
  Checar estados dos vizinhos.
  Seja |N_C (x)| o conjunto de vizinhos de x contaminados.
  Clonar |N_C (x)| glóbulos brancos.
  Enviar clones para os vizinhos contaminados.
Fim
```

Na Figura 5.16 é apresentado o ambiente protegido resultante da atuação dos glóbulos brancos segundo o algoritmo apresentado na Tabela 5.1, ou seja, de seu comportamento percebendo vizinhos contaminados e da ação de clonagem e envio dos clones para limpar os vizinhos.

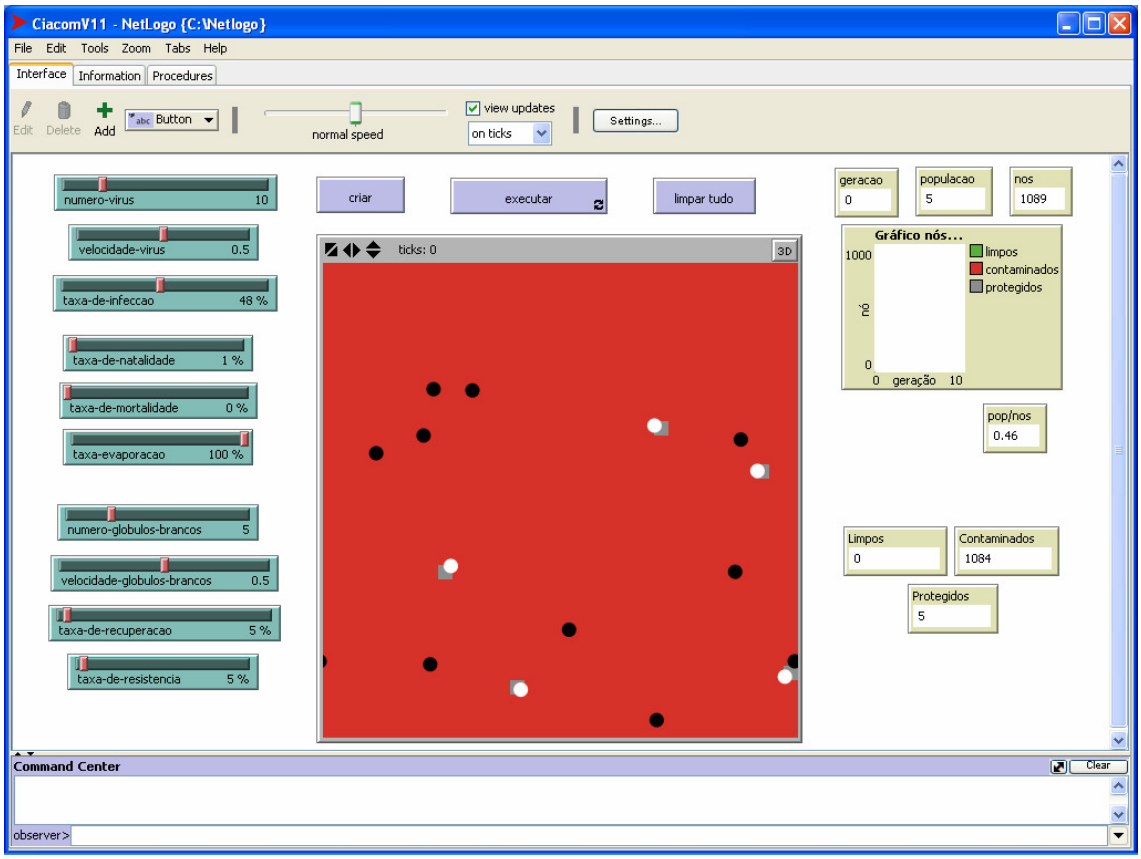

Figura 5.17 – CIACOM_V11 com glóbulos brancos, vírus e ambiente contaminado: modelo de infecção LCP (Limpo, Contaminado ou Protegido).

A versão 11 do CIACOM (CIACOM_V11) é semelhante à 9 com a introdução de agentes vírus que continuam contaminando o ambiente (Figura 5.17). Ou seja, após a limpeza o vírus continua contaminando desde que o local não esteja protegido. Nesta versão o vírus não se clona.

Na Figura 5.18 é mostrada a situação de ação contínua de descontaminação executada pelos glóbulos vermelhos segundo algoritmo da Tabela 5.1 e concomitantemente ação contínua dos vírus contaminando o ambiente.

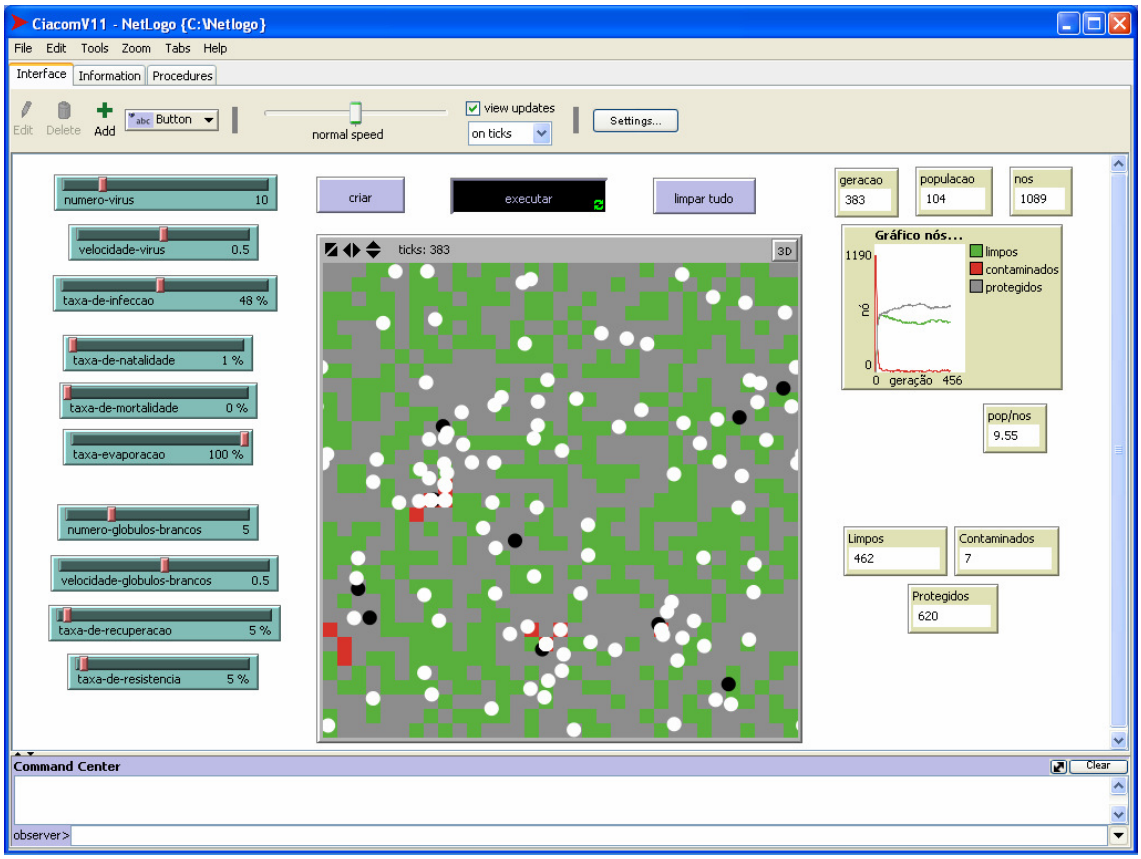

Figura 5.18 – CIACOM_V11 com glóbulos brancos limpando e vírus contaminando o ambiente.

5.1.7 Discussão dos Resultados

a) O efeito da competição – como pode ser observado na Figura 5.7, quando a competição é desabilitada, há o crescimento descontrolado da população de agentes. Ou seja, apesar de não ser natural no sistema circulatório este mecanismo foi introduzido e é necessário para fins de controle da população.

b) O efeito do número inicial de agentes – como pode ser observado nas Figuras 5.5 e 5.14 para diferentes números iniciais de agentes, ou seja, a população de agentes aumenta ou diminui e converge para um valor que depende das taxas de natalidade e mortalidade e do número total de nós (efeito competição).

c) O efeito da velocidade – a frequência de migração de um agente é ditada pelo parâmetro velocidade. Aumentá-la significa influenciar o total de agentes, causando sua diminuição. Diminuí-la significa influenciar a população de agentes fazendo-a aumentar.

d) A extinção – apesar de não ser sensível ao número inicial de agentes, a população é sensível às taxas de natalidade e mortalidade, sendo que obviamente a taxa de natalidade tem que ser maior que a de mortalidade. Após experimentos obtivemos as taxas de 13% (natalidade) e 5% (mortalidade) como taxas limites, pois caso abaixemos um ponto percentual a de natalidade ou aumentemos a de mortalidade, ocorre o efeito indesejável de extinção dos agentes. Além disso, com estas taxas obtivemos a relação aproximada de 10% entre o número de agentes (glóbulos) e o número total de células (nós), de acordo com o estabelecido pelo modelo circulatório.

e) A oscilação – no caso do balanceamento de carga, quando a distribuição de processos corresponde a um ambiente carregado com maior proporção inicial de nós vermelhos e amarelos do que verdes, para uma política de carga "Baixa", nota-se o esforço do sistema para distribuir os processos e tentar se adaptar à política definida. Isto gera uma oscilação, pois não há uma convergência para uma situação ideal. Esta situação é evidenciada pela ocorrência do "mico preto". O operador deve, neste caso, afrouxar a política para, por exemplo, uma política de "Média" carga.

f) O descontrole populacional – vide item a.

g) A convergência – a população de agentes aumenta ou diminui e converge para um valor que depende das taxas de natalidade e mortalidade e do número total de nós (efeito competição). Caso seja aumentado o número total de nós, a população de agentes também aumentará, assim como, se ela diminuir o número total também diminuirá. Ou seja, o número de agentes se adaptará ao número de nós.

h) Autocontrole populacional – parâmetros internos aos agentes, como taxa de natalidade e taxa de mortalidade, e externos, como ocorrência de colisão (dois agentes se encontrando no mesmo nó), dotam os agentes, e consequentemente o sistema, de um controle populacional.

i) O efeito dos feromônios – conjuntos de feromônios servem, no modelo circulatório, para diversos fins: repulsão de agentes para melhorar sua distribuição, atração de agentes para combate a intrusão, acúmulo de depósito químico com evaporação para direcionar comportamento de agentes do mesmo tipo ou diferentes. A informação local é a proposta do CIACOM para solucionar o problema de oscilação. Ações conflitantes devem ser arbitradas por meio de conjuntos de feromônios.

5.2 Estudo de Caso: Grid Oportunístico

O objetivo deste estudo de caso é avaliar a aplicação do modelo CIACOM ao autogerenciamento de um sistema distribuído: um ambiente de *grid* computacional. Escolheu-se como alvo o *grid* oportunístico OurGrid [80,81] que possibilita o processamento paralelo de trabalhos (*jobs*) constituídos por tarefas (*tasks*) classificadas como "saco de tarefas" (*Bag of Tags*), ou seja, aquelas que são independentes.

O OurGrid é uma grade computacional aberta e cooperativa. Os interessados se juntam à comunidade podendo, assim, executar aplicações paralelas. O poder computacional do OurGrid é obtido através de recursos ociosos dos participantes da comunidade.

Na Figura 5.19 é apresentada uma figura com os principais elementos que compõem o OurGrid.

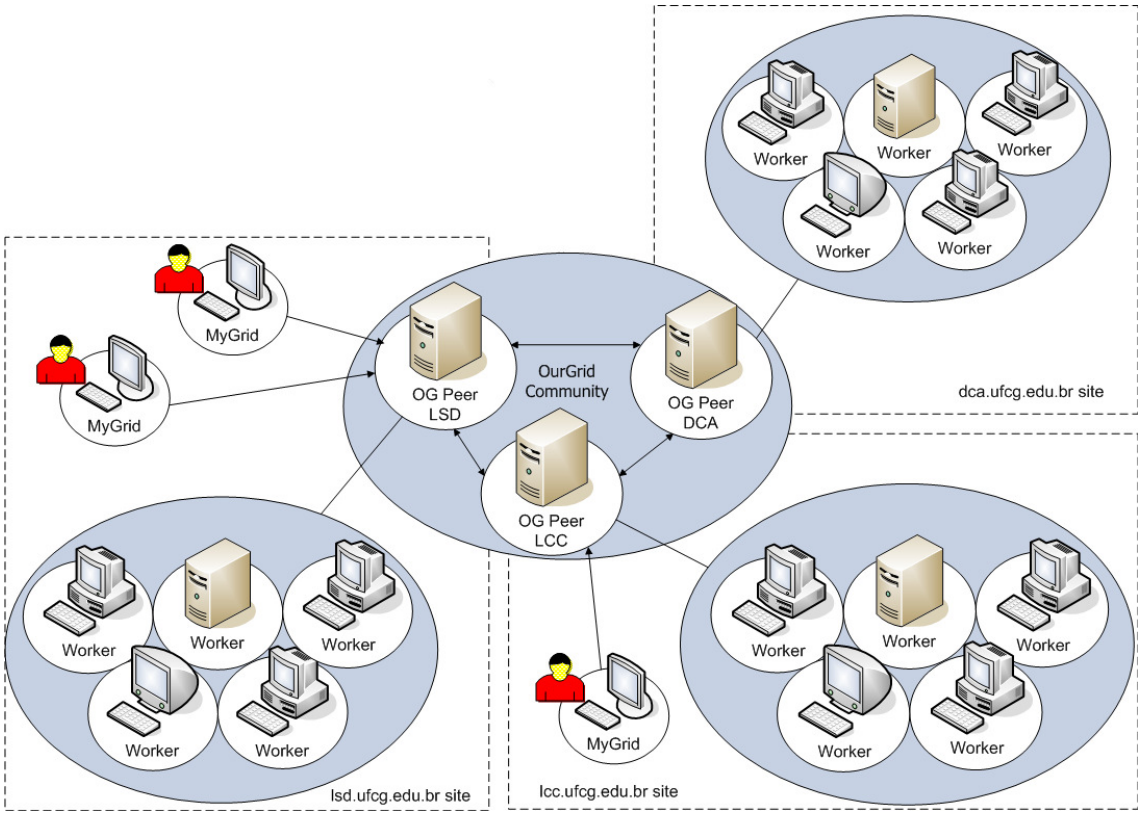

Figura 5.19 – Arquitetura do OurGrid.

O OurGrid é constituído pelos usuários, suas máquinas *brokers* rodando a interface de acesso (MyGrid) que solicita máquinas ao *peer* que junto com outros *peers* formam

uma comunidade disponibilizando máquinas *workers* (as máquinas do *grid*) para processar as tarefas dos trabalhos submetidos.

Um usuário elabora um trabalho constituído por um conjunto de tarefas. Ele deseja que este trabalho seja processado, para isso fornece as entradas necessárias e espera obter as saídas desejadas. No caso de um trabalho que necessite de muito poder de processamento podem ser necessárias horas ou mesmo dias para se obter um resultado, caso todas as tarefas rodassem sequencialmente em um único processador. Para acelerar este processamento foram criados os *grids* computacionais que possibilitam o processamento paralelo das tarefas de um trabalho. Este usuário deve submeter este trabalho para o *grid* e para isso usa o programa broker *MyGrid.* Este programa gerencia a execução das tarefas de um trabalho. Quando um usuário submete um trabalho a um *broker* este cuida de solicitar um conjunto de máquinas ao *peer* local. Este retorna as máquinas *workers* disponíveis localmente e caso estas não sejam suficientes solicita mais máquinas aos *peers* vizinhos que formam a comunidade. De posse destas máquinas o escalonador do *broker*, usando uma determinada estratégia, associa uma tarefa para cada máquina. Se o número de tarefas é maior do que o número de máquinas, assim que uma máquina ficar livre uma nova tarefa será alocada a ela. O *broker* MyGrid gerencia a execução do trabalho, envio de arquivos de entrada e executáveis para serem processados remotamente, bem como o retorno dos resultados (arquivos de saída).

Existem duas estratégias principais para escalonamento de tarefas [82,83]: *binpacking* e *workqueue*. Na estratégia de *binpacking*, para obter um escalonamento otimizado, é necessário ter informação *a priori* sobre as máquinas *workers* (memória, *cpu*, sistema operacional, etc) e sobre as tarefas (tamanho, tipo de processamento intensivo). Se associada a esta estratégia for utilizada a heurística *Fastest Processor to Largest Task First* (FPLTF), o "casamento" (*matchmaking*) entre tarefa e máquina é feito ordenando-se a lista de máquinas na ordem decrescente da velocidade do processador, ordenando-se a lista de tarefas na ordem decrescente de tamanho e alocando-se a máquina com processador mais rápido à maior tarefa e assim sucessivamente. Entretanto, na prática esta estratégia acaba não sendo ótima, pois é difícil ter conhecimento completo sobre as tarefas e as máquinas, o administrador do site pode ter cometido erros no cadastramento das características das máquinas *workers*, bem como as máquinas podem apresentar algum problema no processamento das tarefas. Para compensar a falta de informação completa sobre as tarefas e máquinas foi introduzida a estratégia de *workqueue*. Ela consiste em uma estratégia sem

informação para casamento entre tarefas e máquinas, ou seja, este é feito de forma aleatória. Para compensar um possível "mau casamento" uma ou mais cópias (réplicas) das tarefas submetidas são executadas, ou seja, é feita uma replicação de tarefas. Quando a última tarefa do trabalho (*job*) é processada é feita uma réplica da primeira tarefa que se encontra pendente de execução e esta é submetida à primeira máquina que for liberada. Se uma réplica de uma tarefa terminar sua execução antes da tarefa original, esta é cancelada. Assim, são necessárias mais máquinas onde serão executadas réplicas de tarefas, aumentando as chances de máquinas lentas ou com defeito não atrapalharem o desempenho global do trabalho (*job*) submetido. De forma resumida pode-se afirmar que esta estratégia consiste na troca de ciclos de máquina por informação.

Com isso, podem-se obter com ambas as estratégias tempos de *makespan*[8] semelhantes, com a vantagem da replicação não precisar de informação nem das tarefas, nem das máquinas e com a desvantagem de desperdiçar recursos se comparada à estratégia de *binpacking*.

A proposta deste estudo de caso do CIACOM utilizando o OurGrid consiste na execução de *probe tasks* (tarefas sonda), aleatoriamente nas máquinas, intercalados com as tarefas de um trabalho. A inspiração é baseada na função dos glóbulos vermelhos fornecendo recursos às células em uma proporção de poucos glóbulos vermelhos para muitas células. Estas tarefas "glóbulos vermelhos" são executadas remotamente nas máquinas do *grid* e retornam com informações sobre as máquinas *workers*. Além disso, elas podem "depositar" informações que podem ser utilizados por outros agentes. A informação a ser depositada pelas tarefas "glóbulos vermelhos" podem variar de aplicação para aplicação, pois dependendo do tipo de processamento requerido, podem ser necessários diferentes requisitos. Esta característica corresponde à propriedade de adaptabilidade do modelo CIACOM. Além disso, pode-se considerar que a apoptose do glóbulo vermelho se dá quando da finalização da execução de uma tarefa. As tarefas glóbulos vermelhos são executadas em máquinas das quais não se tem informação para delas obter um indicador de desempenho. Este indicador é usado pelo *broker* para ordenar a lista de *workers* (ordem decrescente).

Resumidamente a proposta é uma estratégia híbrida que começa totalmente com replicação e que, ao longo da execução das tarefas do trabalho (*job*), vai colhendo informações dos *workers,* permitindo que a estratégia de *binpacking* seja usada. Com

[8] Tempo decorrido entre o início do processamento da primeira tarefa e o fim do processamento da última tarefa.

isso, é promovida a melhora do *makespan* e consequentemente redução do desperdício de recursos computacionais inerentes à estratégia de *workqueue*.

O escalonador do programa *broker* do OurGrid foi modificado para permitir que a informação colhida pelo glóbulo vermelho seja utilizada. Com isso, pode-se experimentar a estratégia híbrida *binpacking* e compará-la a *workqueue* original do OurGrid.

5.2.1 Testes

Para executar os testes é necessário um *cluster* OurGrid, o *broker* modificado e um programa para executar tarefas independentes e usado para medir e comparar desempenho de grades computacionais (*grenchmark*). Para testar a nova estratégia do *broker* foi utilizado o *cluster* OurGrid do Laboratório de Sistemas Distribuídos (LSD) da UFCG e como *grenchmark* o problema das N Rainhas. Este problema consiste em posicionar N rainhas em um tabuleiro NxN sem que elas se ataquem e foi utilizado para comparar o *broker* original (*workqueue*) e o modificado (*binpacking*).

Utilizou-se uma implementacão em linguagem JAVA do programa NQueens [84] desenvolvida por Robert Sedgewick e Kevin Wayne da Universidade de Princeton (Anexo D.1). A versão sequencial do programa gera todas as tarefas e executa-as uma a uma. A paralelização do programa foi realizada com a separação da execução das tarefas conforme código do programa (Anexo D.2). As permutações consistentes nas duas primeiras linhas do tabuleiro são usadas como entrada para o programa *Queens* paralelo. Entende-se por consistente a configuração de rainhas em que uma não ataca a outra nas linhas, diagonais e colunas. Além disso, foram feitas modificações para permitir que o programa receba um identificador com o número da tarefa que se deseja executar. Exemplo de execução (neste caso, o programa irá processar a tarefa número 1 em um problema de tamanho 19): java Queens 19 0.

Uma vez feita a paralelização do programa, surge um problema: o programa sequencial para encontrar soluções para o problema das N Rainhas só precisa de uma única chamada; já o programa paralelo precisa de dezenas, centenas ou mesmo milhares de chamadas, dependendo do tamanho do problema (ex:19 Rainhas → 306). Então, foi criado um programa gerador de *jobs* (Anexo D.3) que automatiza este

processo, gerando arquivos formato JDF (Job Description File) para o OurGrid (ex: Anexo D.4). São gerados dois tipos de arquivos de saída: um (1) com extensão “gre” e outro (2) com extensão “ben”. O arquivo com extensão “gre” (1) retorna o número de soluções encontradas, sendo que seu nome contem uma *label* fixa “grenchmark”, seguida do número de rainhas do problema, o número da tarefa paralelizada (começando por zero), o nome da máquina que executou a tarefa, finalizando com o número do *job* e da tarefa do *broker*. Por exemplo, “grenchmark_6_0_urutu_2. lsd.ufcg.edu.br@xmpp.ourgrid.org.3.1.gre” significa que o problema de 6 rainhas, tarefa 0 foi executado na máquina “urutu_2.lsd.ufcg.edu.br@xmpp.ourgrid.org”, sendo o *job* 3 e a tarefa 1 do *broker.* O conteúdo “#sol: 0” significa que nenhuma solução foi encontrada para este caso. Se a tarefa fosse a 4 ou a 15 o conteúdo do arquivo gerado seria “#sol: 1”.

O arquivo com extensão “ben” (2) tem seu nome composto simplesmente pelo número do *job* e da tarefa do *broker.* Em seu conteúdo estão: o nome da máquina em que foi executada o *probe task* e um indicador agregado que reflete o desempenho da máquina, além de outras informações como qual é o sistema operacional, qual sua versão e indicadores parciais utilizados para cálculo do indicador agregado.

Conforme as *probe tasks* são executadas em máquinas remotas, as informações sobre o desempenho destas máquinas são salvas em arquivos com extensão “.ben” (Anexo D.5), retornam e são lidas pelo *broker* modificado, possibilitando a ordenação decrescente das máquinas por este indicador, implementando a estratégia de *binpacking*.

Este indicador é calculado usando a suíte de programas de *benchmark* do *National Institute of Standards and Technology* (NIST) [85] *scimark2*. Seus programas fonte estão disponíveis gratuitamente permitindo a sua modificação para a versão *scimark3* utilizada. Como estão escritos na linguagem JAVA, independem de sistema operacional, bastando que as máquinas na qual ele seja executado disponham de máquina virtual JAVA (JVM).

Em resumo, para aplicar a um caso real parte dos conceitos do CIACOM relativos à função de otimização dos Glóbulos Vermelhos e depósito de informações (feromônios sem evaporação) foi utilizada a plataforma OurGrid de execução de tarefas em paralelo. O *broker* do OurGrid foi modificado para levar em consideração o desempenho das máquinas nas quais as tarefas são submetidas. Para tanto foi

desenvolvido programa gerador de trabalho (*job*) que gera tarefas para resolver problema das N rainhas, intercaladas com cerca de 10% de *probe tasks.* Os arquivos de saída com os resultados das *probe tasks* são retornadas para o *broker* que gradativamente os lê e ordena a lista de máquinas na ordem decrescente de índice de desempenho. O *broker,*então, faz o casamento da próxima tarefa a ser executada com a máquina disponível com melhor desempenho.

Se estes arquivos permanecerem na máquina remota, outros *brokers* podem utilizá-los sem ter que executar as *probe task,* otimizando o processo.

Foram preparados três jobs Queens para 6, 13 e 19 rainhas, contendo, respectivamente, 20, 132 e 306 tarefas, para o teste da estratégia *workqueue* (Replicação) e 25, 155 e 360, para o teste *binpacking* (CIACOM). Lembrando que os testes *binpacking* apresentam sempre *overhead* de tarefas *probe tasks,* enquanto que os testes *workqueue* apresentam eventualmente *overhead* de tarefas devido à replicação. O resultado a ser avaliado é se o *overhead* de tarefas *probe tasks* é compensado pela informação do desempenho das máquinas, ficando menor que o *overhead* de tarefas devido à replicação.

Cada teste foi executado cinco vezes para cada estratégia. Como são duas estratégias e três testes, perfaz um total de 30 testes. Para cada teste é feita a média das cinco rodadas para minimizar efeitos de resultados fora do padrão, ou seja, resultados muito bons ou muito ruins.

A seguir são apresentados os resultados obtidos.

5.2.2 Resultados

Na tabela 5.2 são apresentados os resultados dos 20 testes realizados para comparação dos escalonadores (*broker* original e modificado). Os testes para 19 rainhas não foram realizados, pois haviam poucas máquinas disponíveis devido a transição do OurGrid da versão 4.1 para a versão 4.2.

Tabela 5.2 – Quadro comparativo do índice de desempenho das estratégias dos escalonadores.

nRainhas	*workqueue*				*binpacking*			
N	M	D	T	W	M	D	T	W
6	49247	13349	20	12	73663	14276	25	12
13	198712	40469	132	12	265744	24688	155	12

Onde,

N – Número de linhas e colunas do tabuleiro de xadrez para o problema das n-rainhas

M – *makespan* médio (Tempo Médio de Processamento em Paralelo – segundos)

D – Desvio padrão em relação a M.

T – Número de Tarefas do *job* submetido[9]

W – Número de *Workers* alocados pelo *Broker* no OurGrid

Pode-se observar pelos resultados obtidos que a estratégia de *workqueue* (Replicação) apresentou melhor resultado que a de *binpacking* (CIACOM), para o mesmo número de máquinas disponibilizadas. Basta observar as colunas “M” da Tabela 5.2: tempo de *makespan workqueue* 10% e 18% inferiores a *binpacking.* Observe também a coluna “T” que o número de tarefas da estratégia *binpacking* é, respectivamente, 25% e 18% maior, para 6 e 13 rainhas se comparada à estratégia *workqueue*.

As tarefas *probe tasks,* que executam o programa de *benchmark* científico *scimark3,* são muito custosas do ponto de vista de processamento, se comparadas ao *grenchmark* de 6 rainhas ou mesmo ao de 13 rainhas. Daí a importância do teste das 19 rainhas, pois suas tarefas são mais exigentes, minimizando o excedente de tempo das *probe tasks*. Para compensar, rodamos os testes de 6 e 13 rainhas com *probe tasks* também no broker original, para podermos comparar com o broker modificado. Na Tabela 5.3 são apresentados os resultados.

[9] O *job* submetido no *Broker* modificado apresenta entre 10% e 20% a mais de tarefas (*probe tasks*)

Tabela 5.3 – Quadro comparativo do índice de desempenho das estratégias dos escalonadores.

nRainhas	*workqueue*				*binpacking*			
N	M	D	T	W	M	D	T	W
6	65269	6960	25	12	73663	14276	25	12
13	268025	27189	155	12	265744	24688	155	12

Comparando o tempo médio do *broker* original com o *broker* modificado para o caso das 13 rainhas com *job* de 155 tarefas, constata-se pequena vantagem para o modificado. Deduzimos que o peso das *probe tasks* é grande se comparado ao tempo de execução das tarefas do problema das 13 rainhas. Mas conforme o problema vai ficando mais complexo (ex.: 19 rainhas) as *probe tasks* não pesarão tanto.

Devido aos maus resultados iniciais, propomos que seja feita modificação no algoritmo, de forma que seja mantida a memória do desempenho das rodadas anteriores. Ou seja, o *broker* modificado guardaria o indicador do desempenho das máquinas para aplicação nas rodadas posteriores. Na Tabela 5.4 é apresentado o indicador de desempenho (*benchmark*) dos *workers* colhido pelas tarefas “glóbulo vermelho”.

Tabela 5.4 – *Benchmark* dos *workers* obtido no teste de *binpacking* do CIACOM

worker	*benchmark*
urutu_1	707.462
urutu_2	706.733
krill_2	567.639
krill_1	558.975
worker2-lcc2-26	462.263
worker2-lcc2-28	461.146
worker1-lcc2-28	460.919
worker1-lcc2-26	459.629
worker2-lcc26	384.018
worker2-lcc28	382.043
worker1-lcc28	380.070
worker1-lcc26	377.799

A estratégia de replicação (*workqueue with replication*) é totalmente desinformada, mas apresenta um desempenho satisfatório se comparada a *binpacking*. Em compensação, como usa replicação, apresenta um índice de desperdício que pode ser elevado.

A estratégia híbrida CIACOM é parcialmente informada, pois conjuga as duas estratégias: a *binpacking* (informada) e a *workqueue* (não-informada). Com isso, espera-se obter um índice de desempenho próximo ao do *workqueue* com um índice de desperdício inferior, desde que o problema a ser tratado demande capacidade de processamento e o *cluster* disponibilizado seja heterogêneo.

5.3 Estudo de Caso: Operação do Setor Elétrico

O objetivo da operação do setor elétrico é o fornecimento de energia elétrica aos consumidores de forma ininterrupta com qualidade e segurança. Para tanto a geração, transmissão e distribuição de energia elétrica devem estar dimensionadas adequadamente à carga, ou seja, à demanda do consumidor. Esta demanda varia ao longo do ano e ao longo do dia. Serviços de manutenção programados, desligamentos indesejados, oscilações na tensão e frequência, e outras perturbações e falhas no sistema elétrico tornam difícil e complexa a tarefa de manutenção do serviço de fornecimento de energia elétrica com qualidade.

A atuação da operação pode ser resumida da seguinte forma:
a) Preventiva – deve monitorar os dispositivos do sistema elétrico de forma a prevenir problemas, ou seja, deve perceber perturbações para evitar falhas, agindo de forma pró-ativa;
b) Corretiva – caso as medidas preventivas não sejam suficientes para evitar as consequências de uma perturbação, medidas corretivas devem ser tomadas de forma a manter o sistema funcionando mesmo que de forma precária. Posteriormente medidas devem ser tomadas para sanar a causa do problema que gerou a falha e promover a restauração do sistema;
c) Organizacional – o sistema deve ser organizado de forma a manter sua estrutura funcionando permanentemente. O sistema deve ter uma configuração inicial bem definida, mas que se ajusta de acordo com:

- Tempo – ao longo do dia e ao longo do ano a demanda do consumidor altera, assim como a geração de energia de um sistema interligado com matriz energética preponderantemente hidráulica depende da frequencia de chuvas, linhas de transmissão disponíveis, etc;
- Perturbações – indisponibilidade temporária de equipamentos forçada por causas naturais, intrínsecas ao próprio equipamento, vandalismo, operacionais e do sistema;
- Falhas – indisponibilidade permanente de equipamentos forçada por causas naturais, intrínsecas ao próprio equipamento, vandalismo, operacionais e do sistema;
- Manutenções – indisponibilidade programada dos equipamentos.

d) Operativa – o sistema deve se manter funcionando de forma permanente com níveis de qualidade adequados, com margem de segurança e não sobrecarregado.

Além disso, aspectos econômicos, ambientais e de regulamentação devem ser considerados. Assim, o sistema deve operar de forma ótima levando em consideração o atendimento ininterrupto e com qualidade ao consumidor, sem, no entanto, esquecer aspectos de economia, evitando desperdícios e aspectos regulatórios, evitando multas.

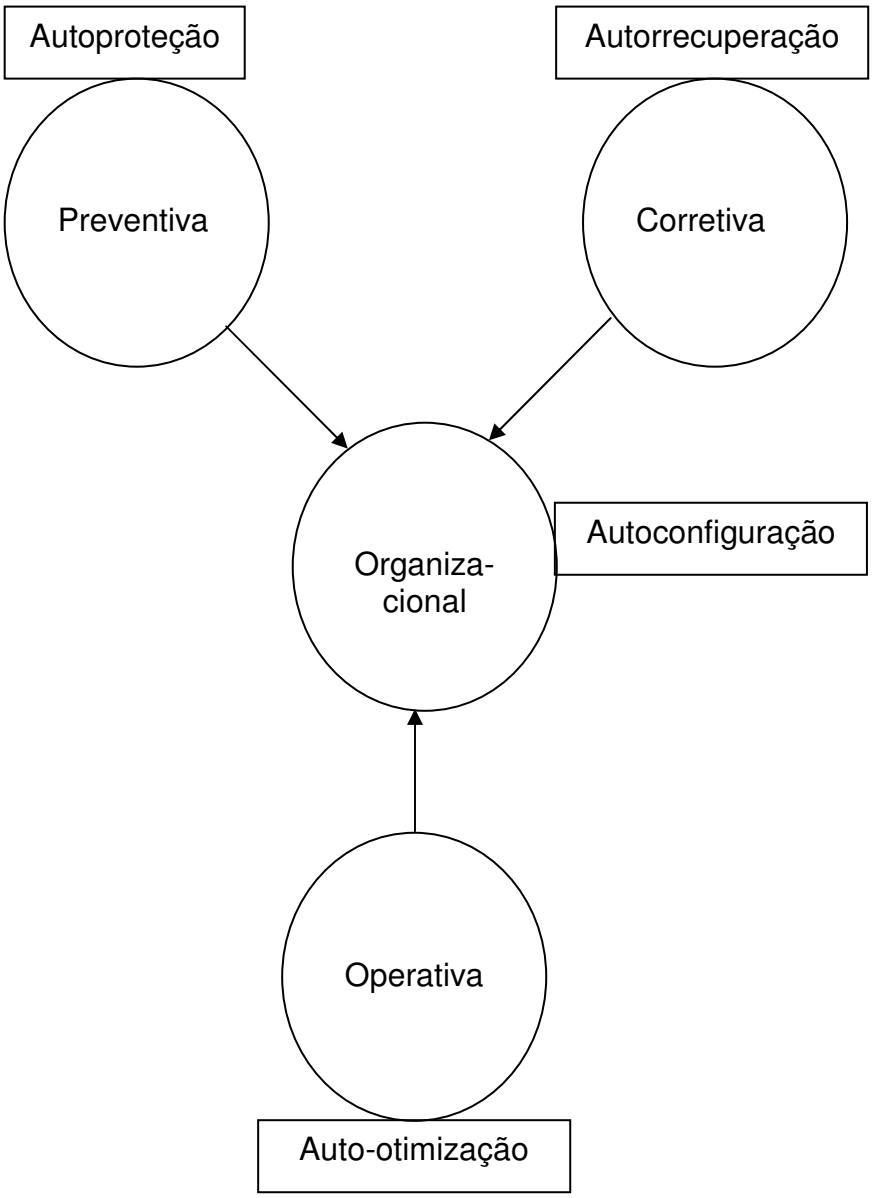

Figura 5.20 – Apectos da operação do sistema elétrico e associação com autopropriedades.

Fazendo uma associação com as autopropriedades definidas no CIACOM temos que: para o aspecto de prevenção a propriedade de *autoproteção*, para o aspecto de correção a propriedade de *autorrecuperação*, para o aspecto de operação a propriedade de *auto-otimização* e para o aspecto de organização a propriedade de *autoconfiguração*. Nota-se também que os aspectos de prevenção, correção e otimização agem diretamente sobre o aspecto de organização, tornando-o dependentes dos mesmos, condição representada pelas setas (Figura 5.20).

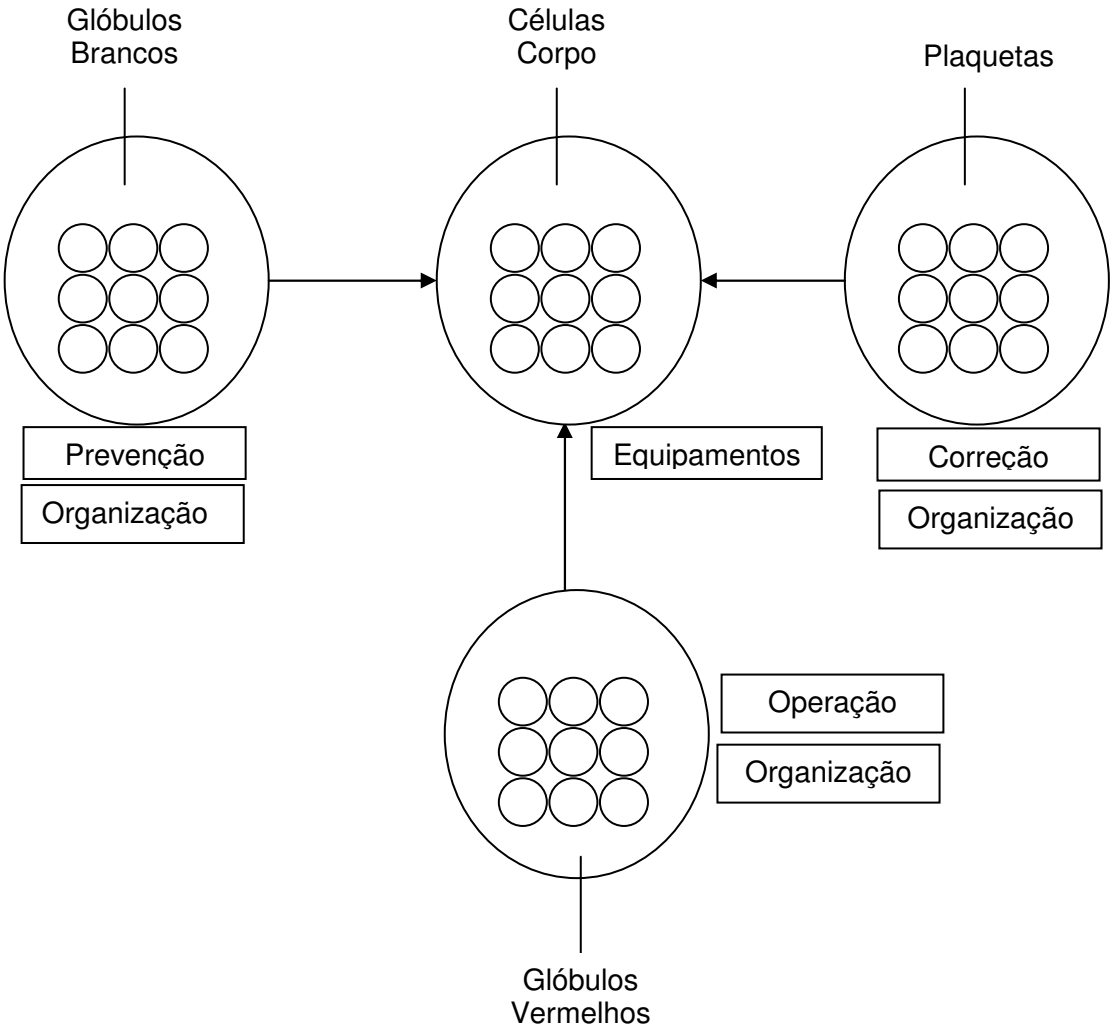

Figura 5.21 – Mapeamento dos conjuntos de agentes aos aspectos da operação do sistema elétrico.

A correspondência entre os aspectos da operação do sistema elétrico aos agentes biológicos é a seguinte: as células do corpo correspondem aos equipamentos a serem gerenciados, os glóbulos brancos correspondem aos gerenciadores de prevenção e organização, as plaquetas correspondem aos gerenciadores de correção e organização e os glóbulos vermelhos correspondem aos gerenciadores de operação e organização (Figura 5.21).

5.4 Conclusão

Neste Capítulo demonstramos a viabilidade do modelo CIACOM através de simulações e através de sua aplicação a um sistema distribuído real: um ambiente de grid computacional. As simulações permitiram que se estimassem parâmetros a serem utilizados para controle populacional. A aplicação do modelo CIACOM no ambiente de *grid* mostrou-se satisfatório e viável. Acreditamos que num ambiente de maior escala, com problemas de maior escala, onde o número de replicações possa tornar-se um problema na questão de disponibilidade de recursos para encontrar soluções, o CIACOM ultrapasse o desempenho do modelo de Replicação do OurGrid. Não podemos esquecer que o CIACOM tem como função primordial o autogerenciamento e neste aspecto, mostrou-se viável e capaz de exercer esta sua funcionalidade principal.

6 Conclusões

Implementamos um protótipo do modelo CIACOM e utilizamos o ambiente de simulação do NetLogo para estudar situações relacionadas com controle de população de células e as autopropriedades. Também fizemos uma implementação de um outro protótipo que foi integrado a um sistema distribuído real, um ambiente de grid computacional, para estudar características de desempenho e viabilidade do modelo num ambiente real. Os resultados indicam que o modelo CIACOM tem viabilidade e seu desempenho não é proibitivo para atingir o autogerenciamento.

A utilização do modelo CIACOM não é uma panaceia e, portanto, não se adequa a todos os problemas de gerenciamento autônomo de sistemas distribuídos. Entretanto, sua estruturação possibilita uma visão integrada servindo como um *checklist* (sistematização) dos atributos necessários e desejáveis para o funcionamento estável de um sistema distribuído.

A inspiração biológica deste trabalho segue a tradição da área de inteligência artificial de encontrar na natureza solução de problemas complexos da engenharia, em particular da engenharia de sistemas e computação.

Procurou-se desenvolver um modelo autônomo completo em que aspectos, como por exemplo, geração controlada de novos agentes, morte programada, adaptabilidade, distribuição, comunicação indireta, cooperação e competição favorecem o surgimento do desejado autogerenciamento.

A interação local de agentes com o ambiente, muitas vezes com informação parcial depositada pelos próprios agentes, e ações simples, mas eficazes, podem fazer emergir comportamentos globais desejáveis, de forma controlada, a exemplo da inteligência de enxames e a inteligência coletiva.

Conclui-se, portanto, que o modelo circulatório desenvolvido neste trabalho pode ser considerado também como um modelo de inteligência coletiva, onde enxames de glóbulo vermelhos, glóbulos brancos e plaquetas agem continuamente de maneira a manter funcionando o sistema distribuído equilibrado.

6.1 Considerações

Esta tese está inserida no contexto de gerenciamento autônomo de sistemas. Mas como se pode alcançar o desejado autogerenciamento? Muitos sistemas podem ser classificados como complexos, pois são compostos por muitos componentes distribuídos que excedem a capacidade de operadores os gerenciarem. Portanto, existe a necessidade de criação de automatismos para compensar estas limitações.
A IBM lançou a idéia da computação autonômica que busca inspiração biológica, no caso no sistema nervoso autônomo humano (SNH) e sua capacidade de atuação inconsciente para atingir uma meta desejada (voluntária). O autogerenciamento é alcançado a partir de quatro autopropriedades: autoconfiguração, auto-otimização, autoproteção e autorrecuperação. Porém a IBM apresenta a metáfora com inspiração biológica, mas depois se afasta dela. Ou seja, a inspiração biológica serve como motivação, mas não são utilizadas técnicas desenvolvidas com inspiração biológica para resolução do problema.

A proposta desta tese é aproximar o autogerenciamento de técnicas desenvolvidas com esta metáfora (SNH) e outras inspirações biológicas de forma a aplicá-las efetivamente na resolução do problema.

O sistema circulatório, as células sanguíneas, suas características de geração distribuída na medula óssea, fornecimento de recursos, proteção e recuperação, à proporção de milhões ou bilhões de células sanguíneas para trilhões de células do corpo, a apoptose das hemácias, a cooperação de células heterogêneas, como é o caso das plaquetas e hemácias para formar o tampão plaquetário na coagulação sanguínea, sua capacidade de chegar a quase todos os lugares do corpo pela rede de vasos[10] são explorados neste trabalho e subsidiam um novo modelo de gerenciamento computacional distribuído denominado CIACOM – Circulatory Autonomic COmputing Model.

A IBM e sua computação autonômica propõem que para um sistema alcançar o autogerenciamento precisa ser dotado de quatro autopropriedades. Um sistema multiagente composto por agentes especializados em cada autopropriedade, dotados de sensores e de atuadores, com capacidade de comunicação e comportamento

[10] A semelhança com uma rede de computadores onde as células do corpo são os computadores e as artérias e veias são suas vias de comunicação.

ditado por regras executadas dentro de um *loop* (laço) eterno pode alcançar o almejado gerenciamento autônomo. Porém esta atuação cooperativa pode se tornar competitiva gerando indesejadas oscilações, caso o sistema não seja dotado de mecanismos de autocontrole, como a comunicação indireta proposta pelo CIACOM.

Vejamos o seguinte exemplo de um gerenciador de banco de dados sobre um sistema operacional. Digamos que o operador defina uma política de acesso à base de dados priorizando a velocidade de acesso. Em um sistema baseado em computação autonômica, teríamos um *loop* eterno sendo executado pelo agente que alocaria memória para agilizar o acesso aos dados. Mas suponhamos que o sistema operacional, desenvolvido por outra equipe, esteja focado na utilização ótima de memória e desaloque a memória alocada pelo outro processo. Pode ser gerada uma oscilação, comprometendo o funcionamento global do sistema, ou seja, um desequilíbrio. Para solucionar este problema sugerimos o agrupamento do conjunto de agentes de acordo com sua especialização, semelhante às células sanguíneas, e a utilização de comunicação indireta para troca de informação local entre agentes do mesmo tipo ou de tipos diferentes, semelhante a depósitos químicos das plaquetas ou o depósito de feromônios pelas formigas. Ou seja, um conjunto de depósitos químicos que se evaporam com o passar do tempo serve como sinalização entre os agentes, possibilitando a identificação de ocorrências, que se não puderem ser automaticamente sanadas pelo próprio sistema, serão reportadas para o operador tomar as devidas providências.

Em outro caso em que o processo alocasse memória, mas não por pura necessidade, e sim por haver um mau funcionamento provocado por vírus, pode utilizar, antes de alocar mais memória, a informação do depósito químico (comunicação indireta) de um conjunto de células responsáveis pela proteção (glóbulos brancos).

6.2 Trabalhos Futuros

A complementação do estudo de caso “balanceamento de carga e desinfecção” é sugerida como continuação deste trabalho aplicado também à atuação das plaquetas (recuperação) com correspondente simulação no NetLogo. É sugerida também a incorporação das *probe tasks* ao OurGrid. Além disso, propõem-se uma aplicação efetiva do modelo CIACOM ao setor elétrico de forma a prover mecanismos de controle autônomo que auxiliem o operador a gerenciar o sistema elétrico, mantendo-o equilibrado. Finalmente é sugerida implementação em JADE dos comportamentos pendentes, inclusive utilizando dispositivos móveis (celulares).

Referências Bibliográficas

[1] STERRITT, R., PARASHAR, M., TIANFIELD, H., *et al.* "A Concise Introduction to Autonomic Computing", *Advanced Engineering Informatics*, Elsevier, v.19, pp. 181-187, ELSEVIER, 2005.

[2] MURCH, R., *Autonomic Computing,* 1 ed. New Jersey, IBM Press, Prentice Hall, 2004.

[3] IBM Corporation "Autonomic Computing: IBM's Perspective on the State of Information Technology", IBM, 2001.

[4] KEPHART, J. O., CHESS, D. M. "The Vision of Autonomic Computing", *IEEE Computer Society*, 2003.

[5] BANTZ, D. F., BISDIKIAN, C., CHALLENER, D., *et al.* "Autonomic Personal Computing", *IBM Systems Journal*, v. 42, n. 1, pp. 165-176, 2003.

[6] FORBES, N., *Imitation of Life: How Biology is Inspiring Computing*, Cambridge, MA, MIT Press, 2004.

[7] WOOLEY, J. C., LIN, H. S., *Catalyzing Inquiry at the Interface of Computing and Biology,* 1 ed., Washington, DC, USA, The National Academic Press, 2005.

[8] WHITE, T. "Swarm Intelligence: A Gentle Introduction with Applications", Disponível em: http://www.sce.carleton.ca/netmanage/tony/swarm-presentation/tsld001.htm, Acesso em: 30 mar 2009.

[9] GARLAN, D., SCHMERL, B. "Model-based Adaptation for Self-Healing Systems", *ACM WOSS '02*, November, 2002, Charleston, SC, USA.

[10] FORREST, S., HOFMEYER, S.A., SOMAYAJI, A. "Computer Immunology", *Communications of the ACM,* v. 40, n. 10, pp. 88-96, 1997.

[11] RUSSELL, S., NORVIG, P., *Artificial Intelligence – A Modern Approach.* 2 ed., New Jersey, Prentice Hall, 2003.

[12] ABELSON, H., ALLEN, D., COORE, D., *et al.* "Amorphous Computing", *Communications of the ACM,* v. 43, n. 5, pp. 74-82, May, 2000*,* Disponível em: http://groups.csail.mit.edu/mac/projects/amorphous/cacm-2000.html, Acesso em: 30 mar 2009.

[13] LINDEN, R. *Algoritmos Genéticos*, BRASPORT, 2006.

[14] *Neurociência Computacional*, Disponível em: http://www.themedicineprogram.com/category/showSubcategories/ Computational_Neuroscience, Acesso em: 30 mar. 2009.

[15] DORIGO, M., BLUM, C. "Ant colony optimization theory: A survey", *Theoretical Computer Science,* v. 344, pp. 243–278, Science Direct, 2005.

[16] WHITE, T. "Emergent Behaviour and Mobile Agents", 1999, Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.4368&rep=rep1&type=pdf , Acesso em: 20 Out. 2009.

[17] STERRITT, R., HINCHEY, M. "From Here to Autonomicity: Self-Managing Agents and the Biological Metaphors that Inspire Them", *Integrated Design and Process Technology*, IDPT-2005, 2005.

[18] TRUSZKOWSKI, W. F., HINCHEY, M., RASH, J. L., *et al.* "Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions", *IEEE Transactions on Man and Cybernetics* – Part C*: Applications and Reviews*, v. 36, n 3 (May), 2006.

[19] REHTANZ, C., *Autonomous Systems and Intelligent Agents in Power System Control and Operation*, 1 ed. Berlin, Springer-Verlag, 2003.

[20] BAKEWELL, S., *The Autonomic Nervous System*, Addenbrooke's Hospital, Cambridge, Disponível em: http://www.nda.ox.ac.uk/wfsa/html/u05/u05_010.htm#para, Acesso em: 9 jul. 2009.

[21] STREETEN, D.H.P., *The Autonomic Nervous System*, SUNY Health Science Center, Syracuse, NY 13210, Disponível em: http://www.ndrf.org/ans.html, Acesso em: 9 jul. 2009.

[22] *The Autonomic Nervous System*, Disponível em: http://staff.washington.edu/chudler/auto.html, Acesso em: 9 jul. 2009.

[23] *Sistema Nervoso*, Disponível em: http://cti.itc.virginia.edu/~psyc220/kalat/JK365.fig12.5.nervous_syste.jpg, Acesso em: 9 jul. 2009.

[24] GANEK, A. G., CORBI, T. A. "The Dawning of The Autonomic Computing Era", I*BM SYSTEMS JOURNAL*, VOL 42, NO 1, 2003.

[25] DIAO, Y., HELLERSTEIN, J. L., PAREKH, S., et al. "Self-Managing Systems: A Control Theory Foundation", *Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS 05)*, 2005.

[26] PARASHAR M., HARIRI, S. "Autonomic Computing: An Overview", J.-P. Banâtre et al. (Eds.): UPP 2004, LNCS 3566, pp. 247–259, Springer-Verlag, Berlin-Heidelberg, 2005.

[27] WOOLDRIDGE, M., *An Introduction to MultiAgent Systems*, 2nd edition, John Wiley & Sons, LTD, 2005.

[28] LANGE, D. B., OSHIMA, M. "Seven Good Reasons for Mobile Agents", *Communications of the ACM*, v. 42, n.3, pp. 88-89, Mar, 1999.

[29] BELLIFEMINE, F., CAIRE, G., POGGI, A., *et al.* "JADE – A White Paper", Telecom Italia Lab, *exp magazine*, v. 3, n.3 (Sep), pp. 6-19, 2003. Disponível em: <http://jade.telecomitalialab.com/papers/2003/WhitePaperJADEEXP.pdf>, Acesso em: 30 mar. 2009.

[30] LODDING, K.N., BREWSTER P. "Multi-Agent Organisms for Persistent Computing", *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, AAMAS, USA, 2004.

[31] TESAURO, G. *et al.* “A Multi-Agent Systems Approach to Autonomic Computing”, *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, AAMAS, USA, 2004.

[32] WHITE S. R., *et al.* “An Architectural Approach to Autonomic Computing”, *Proceedings of the International Conference on Autonomic Computing (ICAC’04)*, IEEE, 2004.

[33] PANAIT, L., LUKE, S. “A Pheromone-Based Utility Model for Collaborative Foraging”, *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, AAMAS 2004, USA, 2004.

[34] HELSINGER, A., KLEINMANN, K., BRINN M. “A Framework to Control Emergent Survivability of Multi Agent Systems”, *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, AAMAS 2004, USA, 2004.

[35] BIGUS, J.P., DIAO, Y., SCHLOSNAGLE, D. A. *et al.* “ABLE: A toolkit for Building Multiagent Autonomic Systems”, *IBM Systems Journal*, vol. 41, no 3, 2002.

[36] AMIN, M. “National Infrastructures as Complex Interactive Networks”, Automation, Control, and Complexity: An Integrated Approach”, Samad & Weyrauch (Eds.), John Wiley and Sons, pp. 263-286, 2000.

[37] VERISIGN *Intelligent Infrastructure for the 21st Century*, 2004.

[38] SMIRNOV, M. “Autonomic Communication Research Agenda for a New Communication Paradigm”, *Fraunhofer FOKUS White Paper*, November, 2004. Disponível em: http://www.fokus.fraunhofer.de/de/motion/_docs/aclab_200411_Autonomic-Communication.pdf, Acesso em: 30 mar. 2009.

[39] CARRERAS I., KIRALY, C., CHLAMTAC, I., *et al.* “A Biological Approach to Autonomic Communication Systems”*, Transactions on Computational Systems Biology IV,* pp.76-82, Springer-Verlag, Berlin-Heidelberg, 2006.

[40] WALDROP, M. M. “Autonomic Computing: The Technology of Self-Management”, *The Future of Computing at the Woodrow Wilson International Center of Scholars*, Foresight and Governance Project, 2004. Disponível em: http://mmwaldrop.com/Starclouds/wp-content/uploads/2007/09/wilson-92a.pdf, Acesso em: 30 mar. 2009.

[41] FORESIGHT OFFICE OF SCIENCE AND TECHNOLOGY *Intelligent Infrastructure Futures Project Overview,* January, 2006.

[42] LUKSZO Z. *Intelligent Infrastructures: The First Step Toward Next Generation Infrastructure Systems,* Next Generation Infrastructure Foundation, 2006.

[43] WEIJNEN, M.P.C. *et al. Next Generation Infrastructures,* Next Generation Infrastructure Foundation, 2006.

[44] FERGUSON, R., CHARRINGTON, S., *Building an Intelligent IT Infrastructure,* December, 2004. Disponível em:http://www.intelligententerprise.com/showArticle.jhtml?articleID=54200322, Acesso em: 5 mai. 2009.

[45] GOLDSZMIDT, G., YEMINI, Y. “Evaluating Management Decisions via Delegation”, *Proceedings of IFIP International Symposium on Network Management*, April, 1993.

[46] GOLDSZMIDT, G., YEMINI, Y. E “Distributed Management by Delegation”, IEEE, 1995.

[47] AZEVEDO, G. P., FEIJÓ, B. “Agents in Power System Control Center”, IEEE, 2005.

[48] AMIN, M. “Toward Self-Healing Energy Infrastructure Systems”, *IEEE Computer Applications in Power*, IEEE, 2001.

[49] STERRIT, R. “Autonomic Networks: Engineering the Self-Healing Property”, no. 17, pp. 727-739, Elsevier, Science Direct, 2004.

[50] KUMAR, S., COHEN, P., LEVESQUE, H. “The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams”, *Proceedings of the 4th International Conference on Autonomous Agents*, pp. 459–466, ACM Press, 2000.

[51] IBM Corporation “An Architectural Blueprint for Autonomic Computing”, IBM, 2004.

[52] IBM, “Practical Autonomic Computing: Roadmap to Self Managing Technology”*, A White Paper Prepared for IBM*, January 2006.

[53] HAMM, S. “Computer, Heal Thyself”, *Business Week magazine*, December 5, 2005.

[54] STERRITT, R., HINCHEY, M. “Biologically-Inspired Concepts for Self-Management of Complexity”, *Proceedings of the 11th IEEE International Conference on Engineering of Complex Systems (ICECCS’ 06)*, IEEE, 2006.

[55] STERRITT R., “Pulse Monitoring: Extending the Health-check for the Autonomic GRID”, *Proc. IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003) at INDIN 2003*, Banff, Alberta, Canada, Aug , 2003.

[56] STERRITT, R., BANTZ, D. F. “Personal Autonomic Computing Reflex Reactions and Self-Healing”, *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 36, no. 3, IEEE, May, 2006.

[57] STERRITT, R., BANTZ, D. F. “PAC-MEN: Personal Autonomic Computing Monitoring Environment”, *Proceedings of IEEE DEXA 2004 Workshops, 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04)*, Spain, 2004.

[58] STERRITT, R., *et al.* “PACT: Personal Autonomic Computing Tools”*, Proceedings of the 12th IEEE International Conference and Workshops on the Engeneering of Computer-Based Systems (ECBS 05)*, 2005.

[59] STERRITT, R., CHUNG S. “Personal Autonomic Computing Self-Healing Tool”*, Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS’04)*, IEEE, 2004.

[60] TESAURO, G., *et al.* “A Multi-Agent Systems Approach to Autonomic Computing”, *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, AAMAS, USA, 2004.

[61] STERRIT, R. “Autonomic Networks: Engineering the Self-Healing Property”, no. 17, pp. 727-739, Elsevier, Science Direct, 2004.

[62] WHITE S. R., *et al.* “An Architectural Approach to Autonomic Computing”, *Proceedings of the International Conference on Autonomic Computing (ICAC’04)*, IEEE, 2004.

[63] KOEHLER, J., *et al.* “On Autonomic Computing Architectures”, IBM Zurich Research Laboratory, Disponível em: www.zurich.ibm.com/pdf/ebizz/idd-ac.pdf, Acesso em: 5 mai. 2009.

[64] KOPILER, A. A., *Gerenciamento de Infra-estrutura de Sistemas Computacionais Inspirado em sistemas Biológicos*, Exame de Qualificação, PESC/COPPE, UFRJ, Rio de Janeiro, RJ, Brasil, Fevereiro, 2007.

[65] KOPILER, A. A., DUTRA, I. C. , FRANÇA, F. M. G. ”Personal Autonomic Desktop Manager with a Circulatory Computing Approach”, *5th IEEE Workshop on Engineering of Autonomic and Autonomous Systems*, Proceedings of EASE 2008, vol. 1. pp. 119-127, IEEE Computer Society Press, Belfast, Northern Ireland, UK, 2008.

[66] KOPILER, A. A., “Personal Autonomic Desktop Manager”, *The 2nd Latin American Autonomic Computing Symposium* (LAACS 2007), Petrópolis, Rio de Janeiro, Brasil, 2007.

[67] KOPILER, A. A., “Introdução à Computação Circulatória”, *Technical Report*, May, COPPE/UFRJ, 2007.

[68] *Sistema Circulatório*, Disponível em: http://www.whhi.com/circulatorysystem.htm, Acesso em: 9 jul. 2009.

[69] *Sangue e Células Sanguíneas*, Disponível em: http://www2.ufp.pt/~pedros/qfisio/sangue.htm, Acesso em: 9 jul. 2009.

[70] MCEWEN, B., KRAHN, D. *The Response to Stress*, November, 1999, Disponível em: http://www.thedoctorwillseeyounow.com/articles/behavior/stress_3/, Acesso em: 9 jul. 2009.

[71] BARABÁSI, A. L., BONABEAU, E. “Scale-free Networks”, *Scientific American*, pp. 50-59, 2003.

[72] SUZUKI, T., IZUMI, T., OOSHITA, F. et al “Self-Adaptation of Mobile Agent Population Inspired in Dynamic Networks: A Biologically Inspired Approach”, *Proceedings of the Second International Conference on Autonomic Computing* (ICAC’05), IEEE, 2005.

[73] AMIN, K. A., MIKLER, A. R. “Dynamic Agent Population in Agent-Based Distance Vector Routing”, *Second International Workshop on Intelligent Systems Design and Applications* (ISDA’02), 2002.

[74] BAKHOUYA, M., GABER, J. “Adaptive Approach for the Regulation of a Mobile Agent Population in a Distributed Network”, *Proceedings of The Fifth International Symposium on Parallel and Distributed Computing* (ISPDC’06), IEEE, 2006.

[75] TAHARA, Y., OHSUGA A., HONIDEN, S. “Agent System Development Method Based on Agent Patterns”, ICSE '99, 1999.

[76] LIMA, E.F.A., MACHADO, P. D. L., *et al.* “Implementing Mobile Agent Design Patterns in the JADE framework”, Special Issue on JADE of the TILAB Journal EXP, 2003.

[77] *Site do NetLogo*, Disponível em : http://ccl.northwestern.edu/netlogo/, Acesso em: 26 out. de 2009.

[78] SCHUT, M. C., *Scientific Handbook for Simulation of Collective Intelligence*, version 2, February, 2007, Disponível em: http://www.sci-sci.org/, Acesso em: 26 de out. de 2009.

[79] FLOCCHINI, P., NAYAK, A., SCHULZ, A. “Decontamination of Arbitrary Networks using a Team of Mobile Agents with Limited Visibility”, *ICIS International Conference on Computer and Information Science (ICIS 2007)*, IEEE, 2007.

[80] *Site do OurGrid*, Disponível em: http://www.ourgrid.org, Acesso em: 9 de out. de 2009.

[81] CIRNE, W., BRASILEIRO, F., COSTA, N. *et al.* "Labs of the World, Unite", *Journal of Grid Computing*, vol. 4, no. 3, pp. 225-246, 2006.

[82] NÓBREGA-JÚNIOR, N., ASSIS, L., BRASILEIRO, F. "Scheduling CPU-intensive GRID Applications Using Partial Information", *37th International Conference on Parallel Processing*, IEEE, 2008.

[83] CIRNE, W., BRASILEIRO, F., PARANHOS, D, *et al.* "On the efficacy, efficiency and emergent behavior of task replication in large distributed systems", *Parallel Computing*, no. 33, pp. 213-234, Science Direct, Elsevier, 2007.

[84] *Programa das N Rainhas*, Disponível em: http://www.cs.princeton.edu/introcs/23recursion/Queens.java.html, Acesso em: 26 out. de 2009.

[85] *Programa Benchmark Científico*, Disponível em: http://math.nist.gov/scimark2/, Acesso em: 26 out. de 2009.

[86] SOUZA E SILVA, E., MATTOSO, M., STAA, A., *et al., Grandes Desafios da Pesquisa em Computação no Brasil – 2006 - 2016*, SBC, 2006. Disponível em: <http://www.sbc.org.br/index.php?language=1&content=downloads&id=272> Acesso em: 26 mar. 2009.

[87] ADAMATZKY, A., *et al.*, Grand Challenges in Computing – GC7 *Journeys in Non-Classical Computation A Grand Challenge for Computing Research*, *The British Computer Society*, pp. 29-32, 2004. Disponível em: <http://www.ukcrc.org.uk/gcresearch.pdf> Acesso em: 26 mar. 2009.

[88] WEISER, M. "The Computer for the 21st Century", *Scientific American.*, Sept., pp. 94-104., 1991.

[89] CORREIA, L., *Vida Artificial*, *XXV Congresso da Sociedade Brasileira de Computação*, V ENIA, SBC, 2005. Disponível em: <http://www.sbc.org.br/bibliotecadigital/download.php?paper=414> Acesso em: 26 mar. 2009.

[90] HERRMANN, K., MÜHL G., GEIHS K. "Self Managemet: The Solution to Complexity or Just Another Problem?",*IEEE Distributed Systems Online*, v. 6, n. 1, Jan. 2005.

[91] WEISS, G., *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence.* 1 ed., Massachusetts, MIT Press, 1999.

[92] BIGUS, J. P., BIGUS, J., *Constructing Intelligent Agents Using Java.* 2 ed., New York, John Wiley & Sons, 2001.

[93] MITCHELL, T. M., *Machine Learning.* 1 ed., Ohio, McGraw-HillI, 1997.

[94] MINAR, N., *Designing an ecology of distributed agents.* M.Sc. Thesis, MIT, USA, 1988.

[95] KOTZ, D., GRAY, R., S. "Mobile Agents and the Future of the Internet", *ACM Operating Systems Review*, pp. 7-13., August 1999.

[96] SCHODER, D., EYMANN, T. "The Real Challenges of Mobile Agents", *Communications of the ACM*, 43(6), pp. 111-112, June 2000.

[97] JANSEN, W., MELL, P., KARYGIANNIS, T., *et al.* "Applying Mobile Agents to Intrusion Detection and Response", National Institute of Standards and Technology, NIST Interim Report (IR) # 6416, October 1999.

[98] FOSTER, I., KESSELMAN, C., TUECKE, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations"*, Intl J. Supercomputer Applications, 2001.*

[99] ASSIS, R., FERREIRA Jr., W. C. "Um Modelo Computacional de Recrutamento em Formigas", *Biomatemática*, IMECC, UNICAMP, vol. XIII, pp. 21-31, 2003.

[100] BRENNAND, C. SPOHN, M., COELHO, A. *et al*.,"*AutoMan: Gerência Automática no OurGrid*", *Sessão Técnica 2, Serviços de Infra-Estrutura*, 2007, Disponível em: http://www.inf.ufrgs.br/~frsantos/files/automan_wcga2007.pdf, Acesso em: 20 Out. 2009.

[101] WHITE, T., BIESZCZAD, A., PAGUREK, B "Distributed Fault Location in Networks Using Mobile Agents", *Proceedings of the second international workshop on Intelligent agents for telecommunication applications*, pp. 130-141, Springer-Verlag, 1999.

[102] ROWANHILL, J. C., KNIGHT, J. C. "ANDREA: Implementing Survivability in Large Distributed Systems", 2008. Disponível em: http://www.cs.virginia.edu/papers/rowanhill.knight.srds.2005.pdf, Acesso em: 20 Out. 2009.

[103] SUMPTER, D. J. T. "The principles of collective animal behaviour", *Philosophical Transactions of The. Royal Society. B*, vol. 361, pp. 5-22, 2006.

[104] LI, Z., SIM, C. H., LOW, M. Y. H. "A Survey of Emergent Behavior and Its Impacts in Agent-based Systems ", IEEE, 2006.

[105] YADGAR, Y., ORTIZ Jr., C., KRAUS, S., *et al.* "Cooperative Large Scale Mobile Agent Systems", Disponível em: http://www.ai.sri.com/pubs/files/970.pdf, Acesso em: 20 Out. 2009.

[106] HILKER, M. "Distributed Self Management for Distributed Security Systems", *Proceedings of the 2nd International Conference on Bio-Inspired Computing: Theories and Applications* (BIC-TA 2007), September 2007, Zhengzhou, China.

[107] HILKER, M., SCHOMMER, C. "AGNOSCO – Identification of Infected Nodes with Artificial Ant Colonies", *Proceedings of the 6th International Conference on Recent Advances in Soft Computing* (RASC2006), July 2006, Canterbury, United Kingdom.

[108] LI, Y. "A Bio-inspired Adaptive Job scheduling Mechanism on a Computacional Grid", *International Journal of computer Science and Network security*, vol. 6, no. 3b, march, 2006.

[109] HONG, J., LU, S., CHEN, D., *et al.* “Towards Bio-Inspired Self-Organization in Sensor Networks: Applying the Ant Colony Algorithm”, *22nd International Conference on Advanced Information Networking and Applications,* IEEE, 2008.

[110] KOK, C., ROMAN, G., LU, C. “Mobile Agent Middleware for Sensor Networks: An Application Case Study”, *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, Disponível em: http://www.cse.wustl.edu/~lu/papers/spots05.pdf, Acesso em: 26 Out. 2009.

[111] *Tutorial de NetLogo*, Disponível em: http://cftc.cii.fc.ul.pt/PRISMA/capitulos/netlogo/, Acesso em: 26 Out. 2009.

[112] VIDAL, J. M. *Fundamentals of Multiagent Systems with NetLogo Examples*, March, 2010. Disponível em: http://jmvidal.cse.sc.edu/papers/mas.pdf, Acesso em: 07 Mar. 2010.

Anexo A – Sistema Circulatório Humano

O sistema circulatório[11] [68] é composto pelo coração e pelos vasos sanguíneos que, em conjunto, mantêm um fluxo contínuo de sangue pelo organismo. O coração bombeia de forma involuntária sangue rico em oxigênio, proveniente dos pulmões, para o restante do corpo, através de uma rede de vasos chamados artérias, e ramos de menor calibre, as arteríolas. O sangue inicia seu retorno ao coração através de pequenos vasos, as vênulas, as quais vão desembocando em tubos cada vez maiores, as veias. Arteríolas e vênulas são ligadas por uma rede de finíssimos vasos, chamados capilares. Nos capilares ocorre a troca de oxigênio, gás carbônico, nutrientes, hormônios e outras substâncias de comunicação química, além de restos do metabolismo celular, entre o sangue e as células do corpo. O sangue é constituído por quatro elementos principais: glóbulos vermelhos (hemácias), glóbulos brancos (leucócitos), plaquetas e o plasma. Na Figura A.1, para reforçar estas características, é mostrada a divisão do sistema circulatório em sistema arterial (cor vermelha à direita) e venoso (cor azul à esquerda). Como um está ligado ao outro formando um ciclo que termina e começa no coração, o sistema circulatório é chamado de fechado.

O sistema composto pelas artérias e veias que levam o sangue aos pulmões e em seguida ao coração, é chamado de pequena circulação ou circulação pulmonar. Já no caso do sangue que parte do coração pelas artérias, seguindo em direção ao resto do corpo e retornando pelas veias, recebe o nome de grande circulação ou circulação sistêmica.

As células de todos os seres vivos necessitam de alimento (nutrientes) e também de oxigênio. No caso do ser humano, seu corpo possui órgãos especiais que possuem a função de digerir os alimentos a fim de absorver o oxigênio do ar (digestão e respiração). Contudo, é necessário que esse alimento seja levado para todas as células. Para isso existe o sistema circulatório que leva o alimento e o oxigênio para todas as partes do corpo. O sangue é vital para a sobrevivência das células, pois além de levar-lhes alimento e oxigênio, ele também retira delas as sobras das substâncias que já não lhe são úteis. Seu percurso por todo o corpo ocorre através das veias e artérias, que se subdividem até formar vasos extremamente finos, atingindo, desta

[11] Esta descrição do sistema circulatório humano foi simplificada e está focada nas principais características nas quais o modelo computacional foi inspirado.

forma, todas as células (exceto, os neurônios alimentados de sangue de forma indireta).

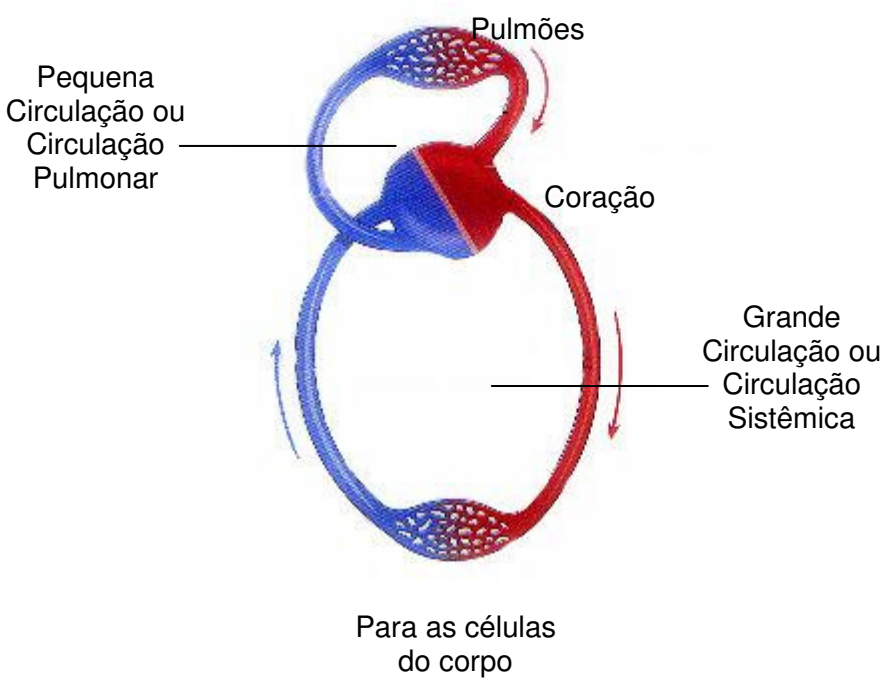

Figura A.1 – Esquema do Sistema Circulatório: Sistema arterial e Sistema Venoso.
Fonte: [68].

O papel do sangue é extremamente importante, pois ele retira os nutrientes dos órgãos de digestão e o oxigênio do pulmão para levar estas substâncias para as células. Para tanto, ele é impulsionado pelo coração e, assim, faz seu percurso pelas artérias, em sua forma boa e limpa.

O sangue, durante sua trajetória pelo corpo, é filtrado pelos rins que retiram muitos detritos deixados pelas células. Ao regressar, ele carrega gás carbônico que absorveu das células, uma vez que, em seu lugar, deixou o oxigênio.

Após este processo, o sangue retorna ao coração, através das veias, que o transportam em sua forma ruim e sem oxigênio. Para melhorar a qualidade sanguínea, o coração envia o sangue aos pulmões, para que, desta forma, o gás carbônico seja trocado pelo oxigênio, e, em seguida, o impulsiona de volta ao corpo.

A.1 Nervos do Coração

O coração é um órgão relativamente autônomo. Como possui uma circulação autônoma, também pulsa "por si só". O estímulo que faz bater o coração nasce, na verdade, no íntimo do músculo cardíaco. Isto é, o coração está em condições de bater sem a intervenção do sistema nervoso. No entanto, ao coração chegam nervos que proveem do nervo vago e do sistema simpático. Estes nervos regulam as batidas cardíacas: o simpático o acelera, enquanto o vago o deixa mais lento.

A.2 Sangue e Células Sanguíneas

O sangue é formado por três tipos de células sanguíneas suspensas em um líquido chamado de plasma. Os tipos de células sanguíneas são: os glóbulos vermelhos ou hemácias, os glóbulos brancos ou leucócitos e as plaquetas (Fig. A.2). As células sanguíneas são produzidas na medula óssea a partir de células-tronco que se diferenciam progressivamente sob controle de diversos fatores do sangue. Ela se encarrega de sua produção e constante renovação. O volume de sangue que circula em um ser humano adulto é de cerca de cinco litros.

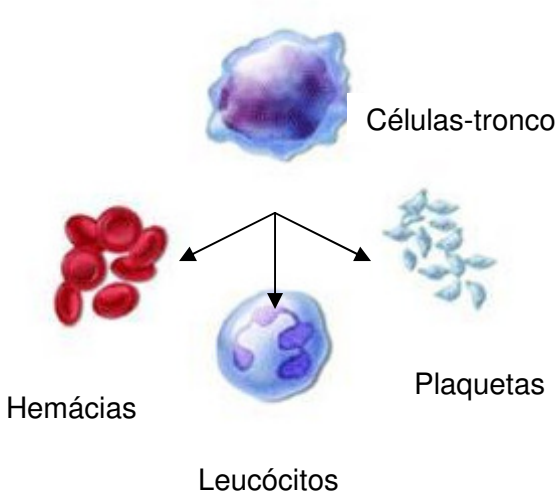

Figura A.2 – Geração de células sanguíneas a partir de células-tronco.

O sangue tem funções muito importantes e complexas entre as quais o transporte de oxigênio, nutrientes, hormônios, dióxido de carbono e outros resíduos do metabolismo. Os glóbulos vermelhos são responsáveis pelo transporte do oxigênio até as células e pela retirada do gás carbônico liberado por elas. Além disso, têm a particularidade de serem elásticos e deformáveis, permitindo assim um fluxo normal dentro dos vasos

sanguíneos. São as mais abundantes (>99%) células sanguíneas. São produzidos na medula óssea e destruídos (após cerca de 120 dias) no baço e fígado. A sua síntese é estimulada por um hormônio secretado pelos rins em resposta à diminuição do fornecimento de O_2.

Os glóbulos brancos podem ser classificados de acordo com as diferentes morfologias e funções – granulócitos, neutrófilos, eosinófilos, monócitos e linfócitos. Têm um papel essencial na defesa do organismo contra os vários agentes infecciosos. Os glóbulos brancos são responsáveis pela resposta imunológica do organismo, ou seja, por sua defesa contra ataques de vírus ou outros organismos considerados intrusos (*not self*).

As plaquetas são as menores células do sangue tendo cerca de 1/3 do diâmetro dos glóbulos vermelhos. São fragmentos celulares de megacariócitos (grandes células presentes na medula óssea em contato com os vasos sanguíneos). Elas têm um papel essencial na coagulação sanguínea, atuando imediatamente quando há uma hemorragia, formando o tampão plaquetário que faz a hemorragia parar.

A.3 Coagulação

A coagulação do sangue ocorre quando a lesão do vaso provoca a liberação (pelas células circundantes) de substâncias que se ativam mutuamente em uma sequência em cadeia fazendo com que o sangue “se solidifique” formando um trombo [69]. Em resumo, a coagulação sanguínea apresenta os seguintes mecanismos para evitar a perda de sangue:

a) Espasmo vascular: imediatamente após a ruptura ou o corte de um vaso sanguíneo ocorre vaso constrição (contração) do vaso sanguíneo lesado;
b) Formação de tampão plaquetário: acúmulo de plaquetas para formar um tampão plaquetário no vaso lesado (adesividade das plaquetas no local da lesão e aderência das plaquetas entre si – processo de ativação das plaquetas);
c) Coagulação sanguínea;
d) Regeneração: crescimento de tecidos fibrosos no coágulo sanguíneo para obturar o orifício do vaso.

A.4 Resposta Imunológica

A lesão dos tecidos por organismos patogênicos provoca a liberação de mensageiros químicos que provocam a vasodilatação em torno da região afetada, assim como o

aumento da permeabilidade protéica dos capilares e vênulas nessa região. Isto causa inchamento nesta região. À medida que o processo inflamatório avança, os neutrófilos circulantes aderem às células endoteliais da região afetada. Os neutrófilos acumulam-se, portanto, em torno da região afetada, em vez de serem arrastados pela corrente sanguínea. Seguidamente os neutrófilos deslocam-se através dos interstícios existentes entre as células endoteliais e migram para o fluido intersticial. Também os monócitos migram para o fluido intersticial e uma vez chegando lá, transformam-se em macrófagos. Começa então o processo de fagocitose do patógeno, iniciado pelo contato da célula fagocítica (neutrófilo ou macrófago) com os lipídios e carboidratos das paredes celulares bacterianas. O fagócito envolve a célula invasora, ataca-a e a destrói.

Os mecanismos descritos acima são não-específicos. A imunidade específica responsável, por exemplo, pela imunização, depende da atuação de moléculas (imunoglobulinas) capazes de reconhecer marcadores moleculares específicos da célula invasora (os antígenos). As imunoglobulinas são produzidas pelos linfócitos e contêm zonas constantes e extremidades variáveis, responsáveis pela ligação seletiva aos antígenos. O processo da sua síntese envolve o rearranjo aleatório dos genes das imunoglobulinas em cada linfócito. Cada linfócito produz, por isso, uma só imunoglobulina, diferente da dos outros linfócitos. Quando um linfócito reconhece um antígeno, é ativado e entra em divisão acelerada. Cada célula filha será específica para o mesmo antígeno reconhecido pela sua "célula-mãe". Após ativação, alguns linfócitos iniciam a resposta imunológica, e outros ficam de reserva como memória imunológica.

Em resumo, a resposta imunológica, disparada pelo processo infeccioso, pode se dar a partir de fatores não-específicos (ou genéricos) e específicos[12].

A.5 Resposta ao Estresse

O estresse ativa respostas adaptativas do organismo. O corpo concentra suas forças para confrontar as ameaças e nos protege no curto prazo. Mas por que o estresse pode ser tão danoso para nosso corpo e mente?

[12] Apesar da complexidade do processo, foram deixados detalhes que podem servir futuramente para algum algoritmo computacional específico relativo a sistemas imunológicos artificiais.

O estresse pode levar à fadiga crônica e à depressão. A exposição ao estresse abala o equilíbrio interno do corpo (homeostase), que pode levar a três estados: (1) o corpo pode recuperar seu equilíbrio uma vez que a situação de estresse cessou ou pode permanecer num estado de (2) sub ou (3) sobre excitação [70].

Os hormônios cortisol e adrenalina e a atividade do Sistema Nervoso Autônomo (SNA) atuam para deixar o corpo em equilíbrio. O SNA e a adrenalina nos mantêm alertas pelo aumento dos batimentos cardíacos e da pressão arterial e rapidamente concentrando as reservas de energia. De forma oposta, o cortisol atua de forma lenta, ajudando a recuperar o suprimento de energia e, ao mesmo tempo, nos auxiliando a memorizar coisas importantes.

Por exemplo, o cortisol prepara nosso sistema imunológico para enfrentar ameaças como vírus e bactérias ou um ferimento. Outro aspecto importante da ação do cortisol é chamado de contenção. Muitos sistemas fisiológicos são confrontados entre si de forma a que nenhum deles fique fora de controle. Todas estas respostas adaptativas são descritas pelo termo alostase que significa manter a estabilidade através de modificações. O corpo enfrenta constantemente desafios e gasta energia tentando "deixar cada coisa em seu lugar". Na maior parte das vezes ele tem sucesso, mas problemas reais surgem quando os sistemas envolvidos na alostase não desligam adequadamente, não são mais requeridos ou não são ativados quando necessários.

A.6 Considerações

O corpo humano tem em média cinco litros de sangue. Como cada milímetro cúbico (mm^3) de sangue possui milhares de células sanguíneas, o número total de células sanguíneas chega à ordem de bilhões. Os capilares e vênulas chegam a todas as células do corpo. Como existem trilhões de células no corpo humano, temos uma relação de uma célula sanguínea para cada mil células (exceção das plaquetas que estão em menor número). É uma abstração interessante para gerenciar a quantidade de computadores que existirão no futuro, todos ligados o tempo todo e com possibilidade de se comunicar a qualquer momento.

As ações das células sanguíneas de reparação, defesa e nutrição são uma boa analogia para autogerenciamento de sistemas computacionais, servindo para modelar sistemas computacionais e a Internet.

No sistema circulatório humano o sangue chega a todas as partes do corpo. Mais especificamente, a todas as células do corpo, através de vasos, que em última instância, na forma de capilares, "se comunicam" com as células. As células do sangue, hemácias, leucócitos e plaquetas, portanto, chegam a quase[13] todas as células, podendo exercer, respectivamente, a função de fornecimento de recursos, proteção imunológica e reparo (recuperação). Normalmente elas atuam em conjunto, ou seja, células sanguíneas do mesmo tipo atuam juntas para realizar sua função específica. Mas, também células sanguíneas diferentes podem atuar em conjunto para realizar a função especializada de uma delas somente. Um exemplo é o das plaquetas que para repararem um vaso danificado, podem usar as hemácias como "tijolo" para a formação do coágulo. Por outro lado, pode haver uma correlação de eventos em que o dano do vaso pode permitir a entrada de corpos estranhos, fazendo com que o sistema imunológico fique em alerta. Deve-se atentar, portanto, para a comunicação entre células sanguíneas do mesmo tipo, assim como de tipos diferentes, dadas possíveis cooperações ou atuações correlacionadas.

A.7 Função dos Elementos

É descrita a função dos elementos no modelo de computação circulatória.

O coração é responsável por emitir um sinal[14] de sobrevivência com uma frequência medida em batimentos por minuto (bpm). Em nosso modelo cada elemento autonômico possui um "coração" indicando se o elemento "está vivo" e uma pulsação indicando seu estado parassimpático (desacelerado) ou simpático (acelerado).

Os glóbulos vermelhos são responsáveis pelo fornecimento e monitoração do sistema quanto aos recursos e pela execução da função de otimização.

Os glóbulos brancos são responsáveis pela monitoração do sistema quanto à infecção por vírus de computador e executam a função de proteção.

As plaquetas são responsáveis pela função de recuperação do sistema, monitorando-o quanto à necessidade de sua manutenção.

[13] Exceto os neurônios que são supridos de sangue de uma forma diferente do que as outras células.
[14] Pode ser usado também como sinal de sincronismo.

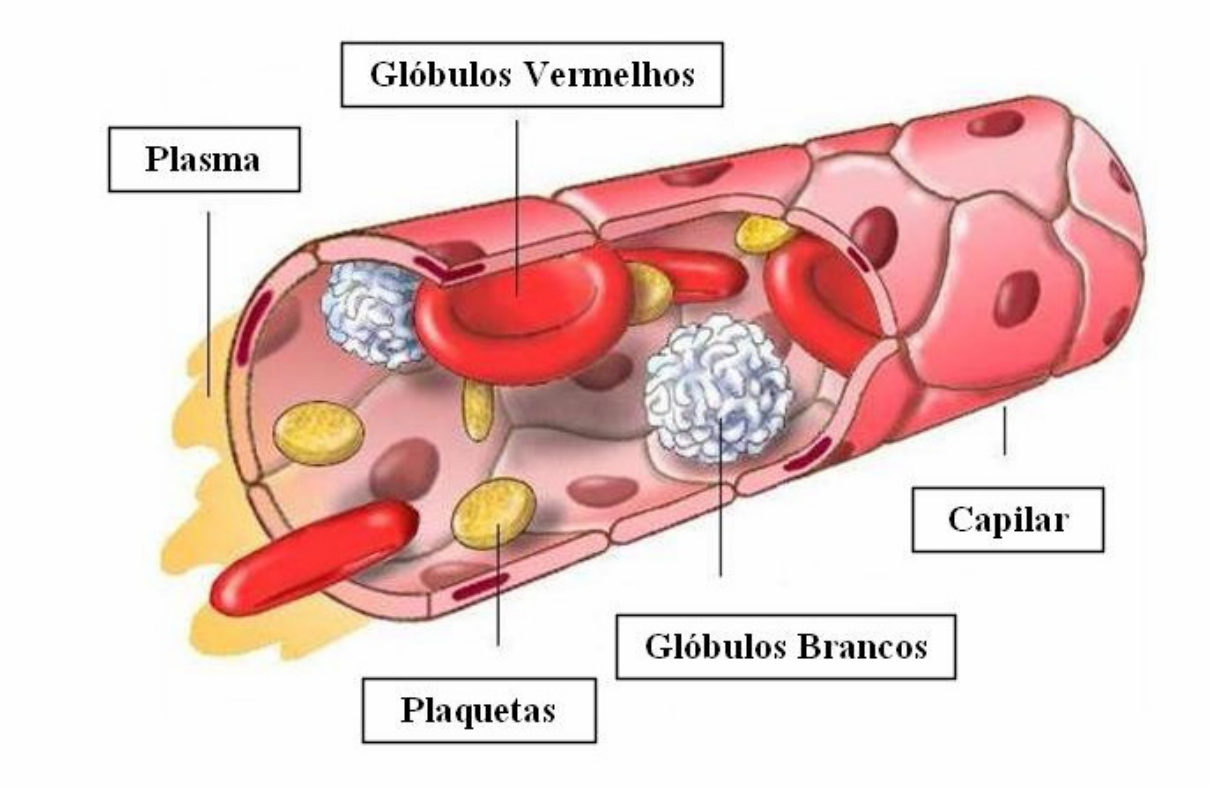

Figura A.3 – Alguns dos Elementos componentes da Computação Circulatória.

As artérias, arteríolas e capilares são as vias de comunicação do sistema. Assim como no sistema circulatório toda célula do sistema é irrigada pelo sangue, fazendo com o que os glóbulos brancos, vermelhos e plaquetas cheguem a elas, nosso sistema computacional deve ser capaz de ter esta característica.

As veias e vênulas também são responsáveis pela comunicação do sistema. Também se comunicam com todas as células e permitem o retorno do sangue para o coração. Esta característica de ciclo corresponde ao sistema vascular fechado, sendo interessante para a "leitura" das informações colhidas pelos glóbulos brancos, vermelhos e plaquetas para possibilitar a correlação entre elas.

Os órgãos são analogias com os componentes computacionais que compõem o sistema, exceto o coração, cérebro, baço e medula óssea que têm função específica.

O cérebro dita as políticas do que o sistema deve fazer de forma global e em alto nível. Como ele pode alterar sua estratégia em tempo–real, o sistema circulatório deve se encarregar de se adaptar às novas regras.

As células do corpo são “gerenciadas” pelas células sanguíneas. Na computação circulatória correspondem aos componentes de software (programas, *threads*, processo ou agente de software).

A medula óssea é o órgão responsável pela geração distribuída das células-tronco e por sua diferenciação dando origem às células sanguíneas: hemácias, leucócitos e plaquetas.

O baço é o órgão que gera novas células sanguíneas de forma centralizada.

As células-tronco são as células que podem ser diferenciadas para recompor qualquer “órgão” e criar novas células sanguíneas. Podem assumir a funcionalidade de qualquer outra célula do corpo. Em nosso modelo corresponde a uma nova versão de um programa, *threads*, processo ou agente de software.

Anexo B – Considerações Adicionais

B.1 Adaptabilidade e Evolução

O pressuposto que o sistema modelado é autoadaptável e autoconfigurável pode ser representado por diversas versões do sistema sendo geradas pelo próprio sistema. Por exemplo, digamos que o sistema seja criado em sua versão inicial, versão 1.0, e que cada elemento componente também possua esta versão. Após autoajustes e autoconfigurações, o sistema poderá "evoluir" para uma versão 1.1 ou 2.0, mas alguns de seus componentes poderão evoluir independentemente para versão 1.3 ou 2.2. Pode ser criado um mecanismo, semelhante ao CVS (*Concurrent Versions System*), para controle de versão de software. Isto vale para o software e para os arquivos de configuração. É como se um software "geneticamente" modificado fosse gerado automaticamente para ser executado da próxima vez, em substituição ao sistema anterior. Entretanto esta evolução só se dá caso o sistema atual mostre um desempenho melhor do que o anterior[15]. Pode haver uma atualização de um ou mais módulos, de acordo com suas dependências mútuas, gerando uma versão intermediária, mas melhor do que a original. O sistema evolui e quando é atualizado incorpora a evolução alcançada naquela geração.

Este controle de versões permitiria a auditoria do sistema e a análise contínua do que está ocorrendo com a evolução automática do software. Note-se que esta evolução é limitada, é como se sintonizássemos o sistema, ajustando os parâmetros.

B.2 Sistemas sob Estresse

Inspirado no funcionamento do sistema nervoso autônomo quando sujeito a situações de estresse é sugerida sua agregação ao modelo circulatório-autonômico em que se busca a homeostase, ou seja, o equilíbrio do sistema através da alostase, ou seja, modificações.

Os elementos circulatório-autonômicos que compõem um sistema circulatório-autonômico, de forma recursiva ou não, devem possuir regras antagônicas que se regulam: uma simpática e outra parassimpática.

[15] Critérios devem ser estabelecidos para este fim.

O sistema circulatório-autonômico está sempre em funcionamento, havendo comunicação entre suas partes, com sensores ativos monitorando o ambiente e inteligência focalizada na funcionalidade a ser atendida para escolher a próxima ação a ser tomada.

Assumindo que inicialmente o sistema se encontra em equilíbrio, mas sujeito a mudanças no ambiente (fatores extrínsecos), o que é percebido através dos sensores monitorando as propriedades *auto**, as regras (simpática e parassimpática) atuam de forma a dar suporte à ação a ser tomada pelo subsistema (alostase). Quando o meio externo se torna agressivo, havendo mudanças drásticas no ambiente ou simplesmente exigindo do subsistema mais recursos do que ele pode ofertar, este entra no estado de estresse, gerando uma contenção nas demandas. O sistema tenta manter o equilíbrio mesmo sujeito ao estresse. A parte simpática do sistema está em ação. Neste ponto o meio externo pode manter sua demanda, mas o novo equilíbrio alcançado pode ser considerado suportável e passa a ser o novo ponto de equilíbrio.

Caso a exigência do ambiente se reduza, as regras parassimpáticas entram em ação para fazer o sistema retornar ao estado de equilíbrio anterior.

Como o próprio sistema pode falhar (fatores intrínsecos), este deve ser tolerante a falhas e passível de autorrecuperação. O sistema deve diferenciar fatores extrínsecos “normais” daqueles causados por vírus ou funcionamento anormal de outros subsistemas. Para isso, as técnicas de segurança de informação embutidas na propriedade de autoproteção devem estar ativadas.

Mesmo que o sistema atinja um patamar de equilíbrio superior, quando cessada a demanda externa por mais do que um período pré-estabelecido pode liberar recursos e voltar a um ponto de equilíbrio anterior, ou mesmo a um ponto de equilíbrio inferior (auto-otimização).

Em casos extremos um subsistema pode “tirar do ar” outro subsistema no intuito de manter o sistema como um todo funcionando, mesmo que de forma precária. Os casos extremos devem ser reportados na forma de alarmes para o responsável pela manutenção do sistema tomar as providências cabíveis. O próprio sistema pode se antecipar e alertar na forma de avisos o operador responsável pela manutenção, sobre problemas em potencial.

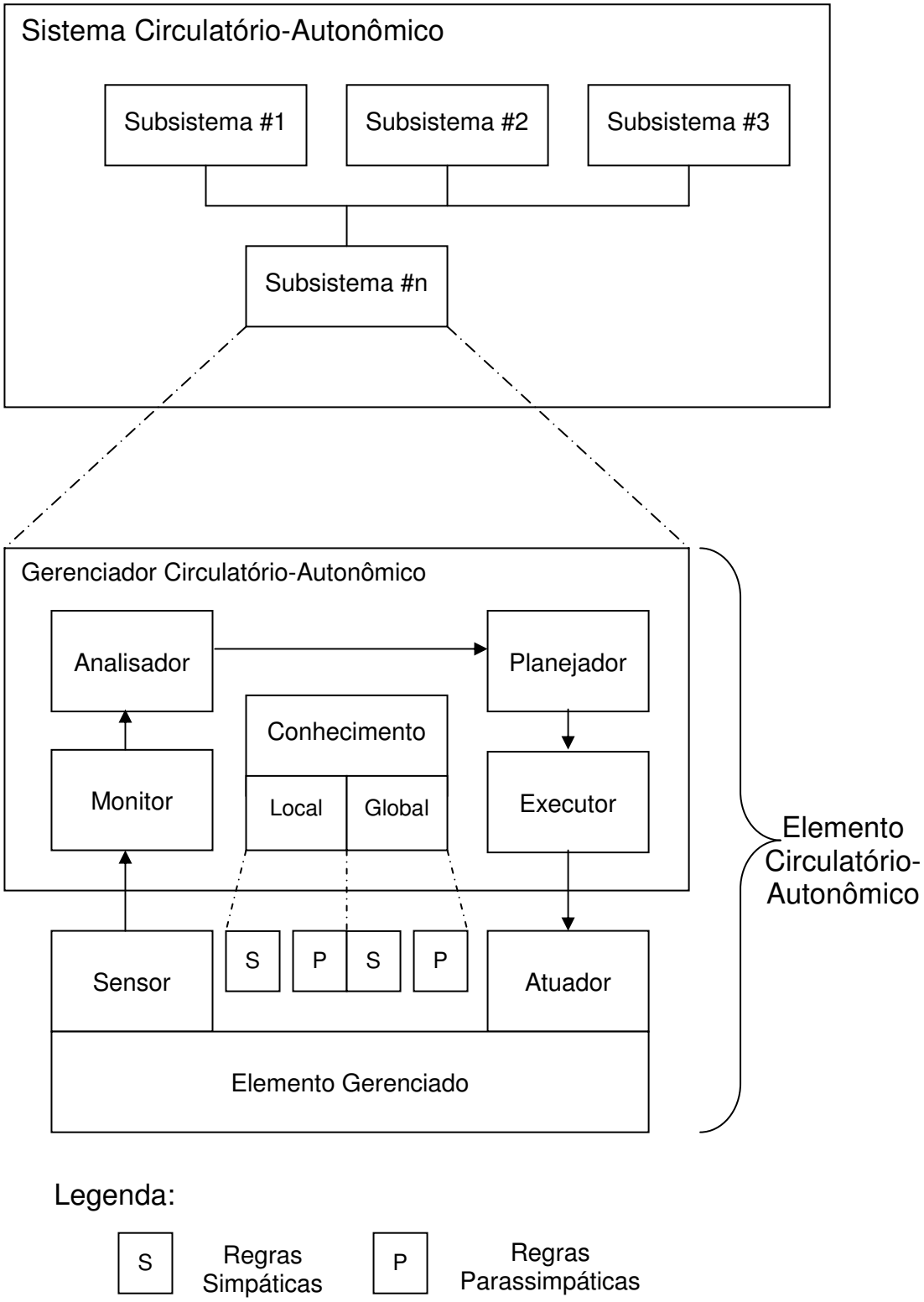

Figura B.1 – Regras Simpáticas e Parassimpáticas em Nível Local e Global para Elemento Circulatório-Autonômico.

Embora a meta final buscada seja a autonomia completa do sistema, um grande passo é dado quando as configurações repetitivas e padronizadas do sistema são realizadas autonomamente, de modo a liberar o operador para funções mais nobres e para permitir que ele lide com sistemas cada vez mais complexos, geograficamente distantes e que prescindem de profissionais capacitados.

Na Figura B.1 é mostrado um sistema circulatório-autonômico composto por outros subsistemas também circulatório-autonômicos. É sugerido, quando da confecção das regras locais e globais, um controle para que o sistema, originalmente em equilíbrio, retorne ao equilíbrio, em função de modificações de seu ambiente, mesmo em situações extremas (estresse). Um comportamento antagônico simpático ou parassimpático é disparado pela regra correspondente. De acordo com políticas estabelecidas, os elementos devem agir localmente para alcançar suas metas locais, mas ao mesmo tempo perceber o comportamento local dos outros elementos, de forma a alcançar em conjunto metas globais.

B.3 Considerações Adicionais sobre o Modelo

A propriedade de autoajuste se refere ao ajuste dos parâmetros do próprio elemento gerenciador. Esta propriedade está presente em todas as células sanguíneas artificiais individualmente (na forma de regras internas – "DNA"). Mudanças destas regras (autoadaptação) são possíveis, mas num processo lento após várias gerações. Para isso, esta informação deve ser armazenada para posterior análise. Este mecanismo de regras e adaptação pode ser estendido para todas as propriedades *auto**.

A propriedade de autoconfiguração depende de combinação entre o elemento gerenciador e o elemento gerenciado, sobre quais e como essas variáveis vão ser modificadas e seus limites. Dependendo das condições do ambiente, o sistema se autoconfigurará. É sugerido que se firme um "contrato" sobre as variáveis que podem ser alteradas, seus limites e como devem ser alteradas, possibilitando a configuração automática do elemento gerenciado pelo elemento gerenciador. Assim, a autoconfiguração de cada elemento gerenciado levará a autoconfiguração de todo o sistema.

O conceito de contrato para autoconfiguração pode ser estendido para as outras propriedades *auto**: recuperação, otimização e proteção. Devem ser estabelecidos contratos sobre quais variáveis devem ser monitoradas e seus limites, assim como

quais ações devem ser executadas e como o conhecimento *a priori* destes contratos possibilita à medula óssea criar células sanguíneas artificiais com regras e configurações locais que permitam exercer o papel de gerenciadores autonômicos.

No caso da propriedade de auto-otimização, os elementos a serem gerenciados devem ser mapeados com uma prioridade para funcionamento. Na condição limite algum órgão/células artificiais terá seu funcionamento suspenso para que o sistema continue funcionando minimamente. No caso, a otimização do sistema ocorre por ele continuar funcionando, mesmo que precariamente.

Assim como no sistema circulatório existe um fator de “localização”, isto é, uma célula do sangue (ex: glóbulo branco) ora está na grande circulação, ora está na pequena circulação. Dentro da grande circulação pode estar no estado arterial (GCA) ou venoso (GCV).

Existe também o fator “tempo”, pois a célula do sangue é criada, mas tem um tempo de vida limitado, ao final do qual é destruída.

Na grande circulação as células sanguíneas atuam de forma distribuída, analisando localmente, segundo regras locais, informações locais, mas também levando em consideração regras globais. Elas tomam decisões de forma distribuída, levando em consideração as informações coletadas diretamente nas células e rastros deixados nas proximidades por células do mesmo tipo ou diferentes.

Ou seja, estes rastros são informações indiretas, percebidas por outras células que estão próximas ao local da ocorrência. As células só se tornam executoras, ou seja, passam de monitoras a executoras, caso a correlação de todas estas informações, façam elas autodeliberarem assim.

Na pequena circulação há uma análise global centralizada com execução centralizada, que leva em consideração o número total de células sanguíneas, para proceder sua replicação ou não. Pode haver um mecanismo de prolongação da vida das células. A princípio cada célula está programada para se autodestruir. Caso haja uma mudança nas regras globais ou estratégia, neste momento é que esta alteração é efetivada. A princípio estas regras devem ser alteradas gradativamente com a passagem das células do sangue pela pequena circulação.

É também na pequena circulação, e depois de várias gerações, que podem ser reavaliados os parâmetros locais, de forma que novas células já sejam criadas adaptadas. Será considerada uma nova versão. Se as regras locais também precisarem ser atualizadas (ex.: novo antivírus), também será considerada uma nova versão. Assim como, as mudanças nas regras globais são consideradas uma nova versão.

O ciclo arterial (estados PCA e GCA) apresenta uma frequência igual ou mais rápida que o ciclo venoso (PCV e GCV). Caso o sistema seja mais exigido estas frequências aumentam, ou diminuem caso contrário. É como se o coração recebesse um estímulo elétrico do cérebro aumentando ou reduzindo seus batimentos. É importante notar que deve ser elaborado um mecanismo que permita que as células do sangue “leiam” através de sensores as informações das células e órgãos. Um órgão pode ser um componente de software, ou software legado que se comunicaria com as células de sangue através de um protocolo comum. Poderia haver um código de identificação de forma a um saber quem é o outro, e pegar informação na forma de índices de desempenho ou necessidades de acordo com a função específica da célula do sangue. Assim, seria identificado se o componente é próprio (*self*) ou não próprio (*not-self* ou intruso), se está funcionando adequadamente ou requer manutenção (plaqueta), proteção (leucócito) ou recurso (hemácia). As técnicas de vacinas antivírus, agentes estáticos e móveis devem ser avaliadas neste ponto.

Em caso de “ataque” os glóbulos brancos podem se transformar em “soldados”, ou seja, passam de monitores a executores.

Os glóbulos vermelhos servem para monitorar os recursos utilizados. Em caso de falta de recursos mais recursos devem ser alocados ou em caso de recursos em excesso, devem ser liberados.

As plaquetas ao perceber dano, devem atuar como reparadoras. Os glóbulos brancos, quando no papel de “soldado”, podem deixar um “rastro” químico para catalisar a atuação reparadora das plaquetas, assim como fazem as formigas na “inteligência de enxames”.

Este mecanismo é comum para as demais células sanguíneas: deixar “rastros químicos” para serem percebidos pelos outros, nas proximidades. Esta última tem uma componente (via) arterial e uma venosa. A via arterial leva os recursos novos ou

renovados para todo o “corpo” e coleta informações sobre os recursos, a proteção e a recuperação. A via venosa executa as ações necessárias baseadas nas informações coletadas. Estas informações podem ser internas, coletadas da célula pela própria célula sanguínea, externas diretas, provenientes de outras células ou ainda externas indiretas por “rastros”. Dependendo do nível hierárquico, do tipo de célula e da proximidade ao local de ocorrência, as informações coletadas têm uma influência maior ou menor na tomada de decisão. Isto permite tomar decisões distribuídas. O órgão central influencia a tomada de decisão distribuída através da adaptação de sua estratégia e regras globais. Isto é, de forma indireta e ao longo do tempo, a não ser que ocorra algum colapso.

A noção de ciclo de vida é aplicada às células sanguíneas que estão programadas para ter um tempo de vida, ao final do qual morrem (apoptose) [63]. Esta seria a morte natural (por tempo de vida). Caso sua morte não seja natural, deve ser disparado mecanismo de defesa, para averiguar se a morte foi acidental, proposital (vírus), por mau funcionamento de um “órgão” (componente computacional) ou por sobrecarga (estresse)[16].

As funções de reprodução ou replicação também existem, permitindo a “sobrevivência” do sistema e a “continuidade da espécie” e servindo também para auto-otimização, autoproteção e autorrecuperação. Quando o sistema é “atacado” deve haver a multiplicação dos glóbulos brancos. Quando o sistema necessita de “mais energia” deve haver a multiplicação de glóbulos vermelhos. Quando o sistema necessitar de mais manutenção deve haver a multiplicação das plaquetas.

A seguir é apresentado um detalhamento das principais ideias vistas até agora: ciclo correspondente à pequena circulação, estados venoso (PCV) e arterial (PCA) e ciclo correspondente a grande circulação, estados venoso (GCV) e arterial (GCA).

B.4 Detalhamento do Modelo

Nesta proposta os dois ciclos correspondentes à pequena e à grande circulação, assim como os estados do sangue arterial e venoso estão modelados por uma máquina de estados já apresentada anteriormente nas Figuras 3.21 e 3.22. O

[16] Na tese, por questões de simplificação, só será considerada a morte natural.

elemento gerenciador deve ser capaz de armazenar seu estado corrente e pode estar em qualquer dos estados possíveis. Os únicos estados válidos para o elemento medula óssea são o PCA ou o PCV. A seguir são mostradas figuras contendo fluxogramas de alto nível para especificar melhor os detalhes do modelo.

Nas Figuras B.2 e B.3 são apresentados de forma um pouco mais detalhada os fluxos de execução dos procedimentos realizados pelo elemento medula óssea nos estados PCV e PCA.

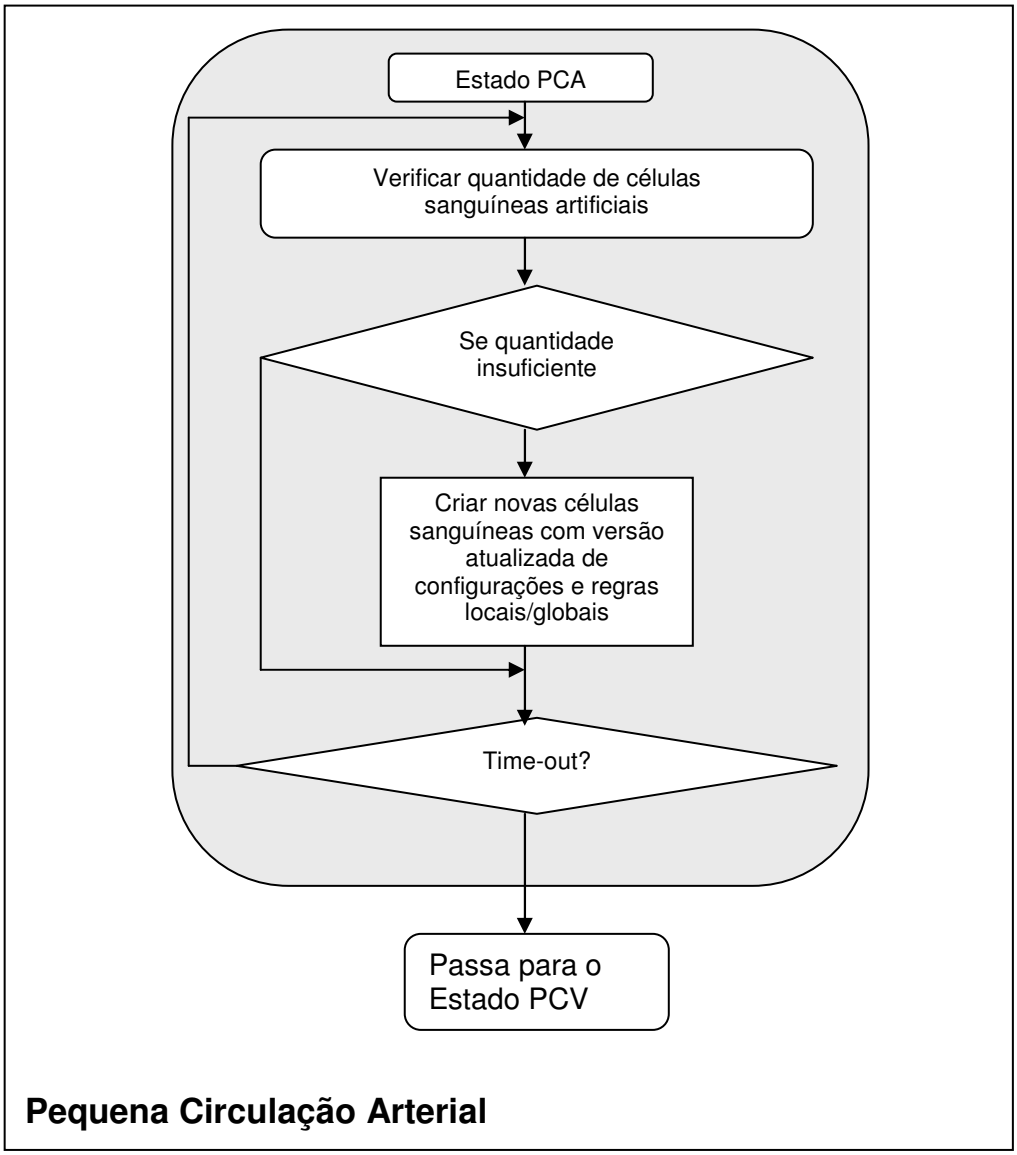

Figura B.2 – Detalhamento do Fluxograma do Elemento Medula Óssea para o Estado PCA.

Estes procedimentos correspondem à criação de novos elementos gerenciadores, gestão das versões de software e atualização de políticas globais.

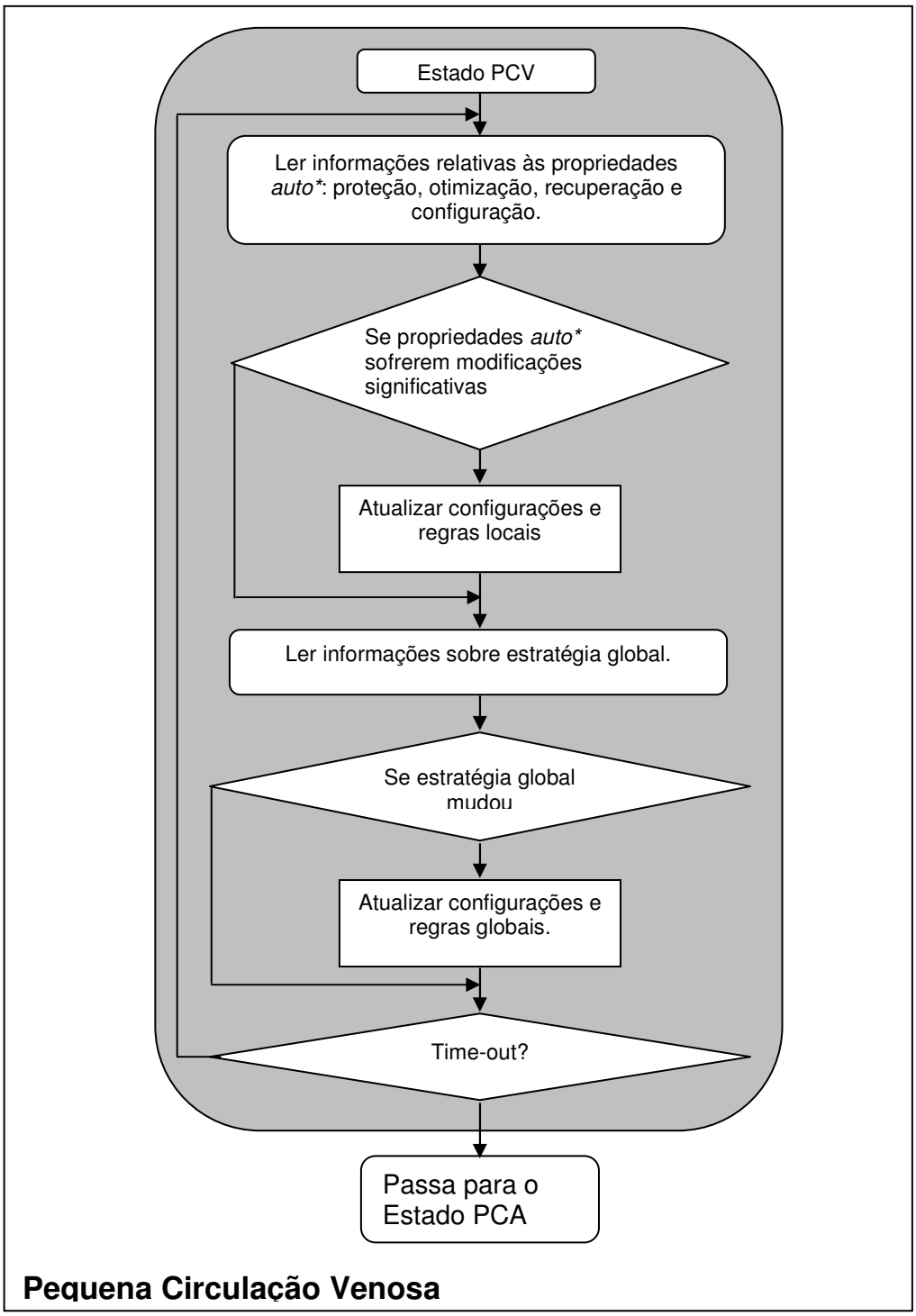

Figura B.3 – Detalhamento do Fluxograma do Elemento Medula Óssea para o Estado PCV.

Nas Figuras B.4 e B.5 são apresentados os fluxogramas de alto nível para os elementos gerenciadores (plaqueta, hemácia e leucócito) nos estados mais relevantes para eles: GCV e GCA. Por questões de simplificação não foram representados os testes (desvios) correspondentes ao *time-out*, acoplamento e vida-útil que podem manter o *loop* ou provocar a mudança de estado.

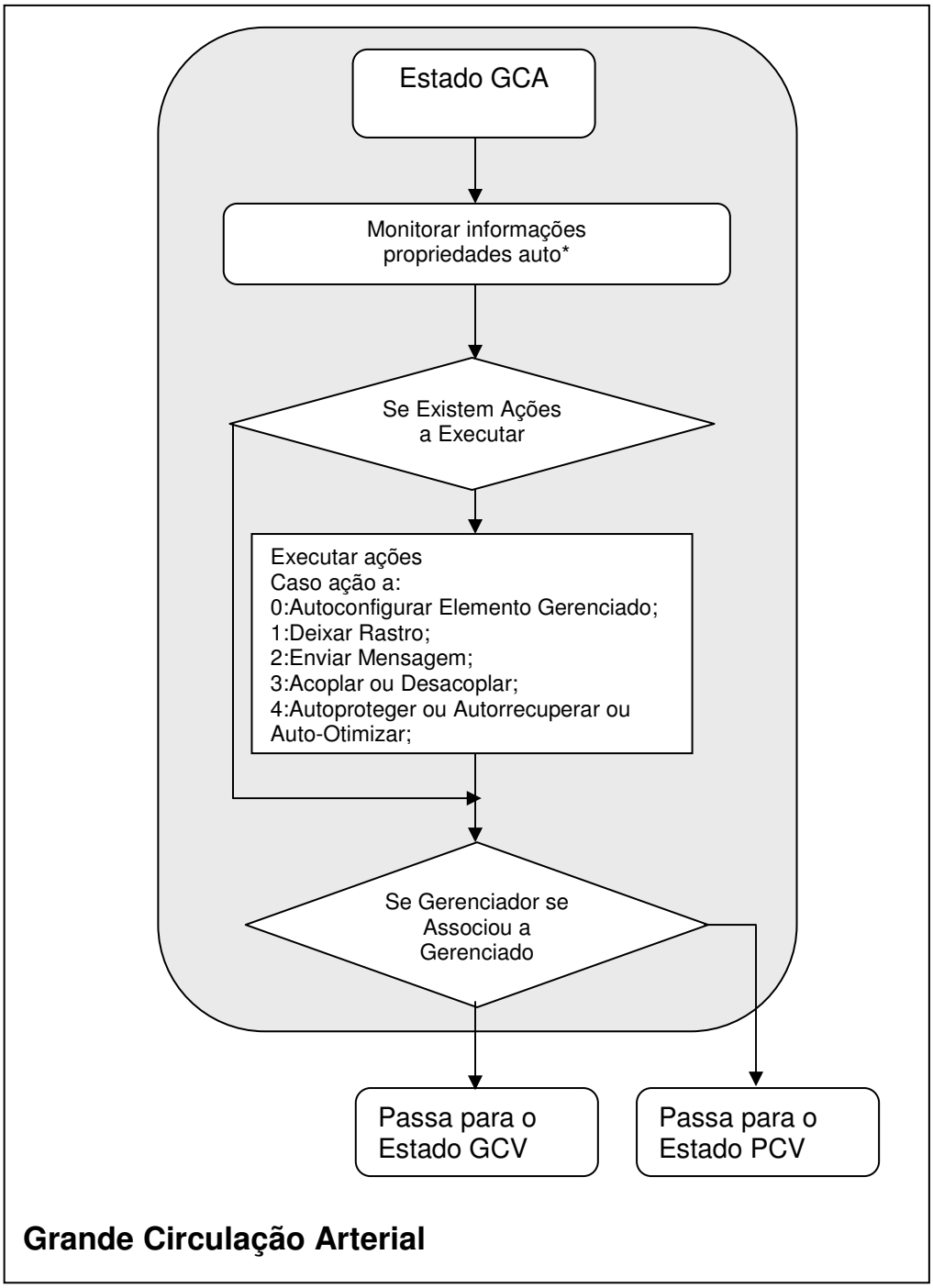

Figura B.4 – Detalhamento do Fluxograma do Elemento Gerenciador para o Estado GCA.

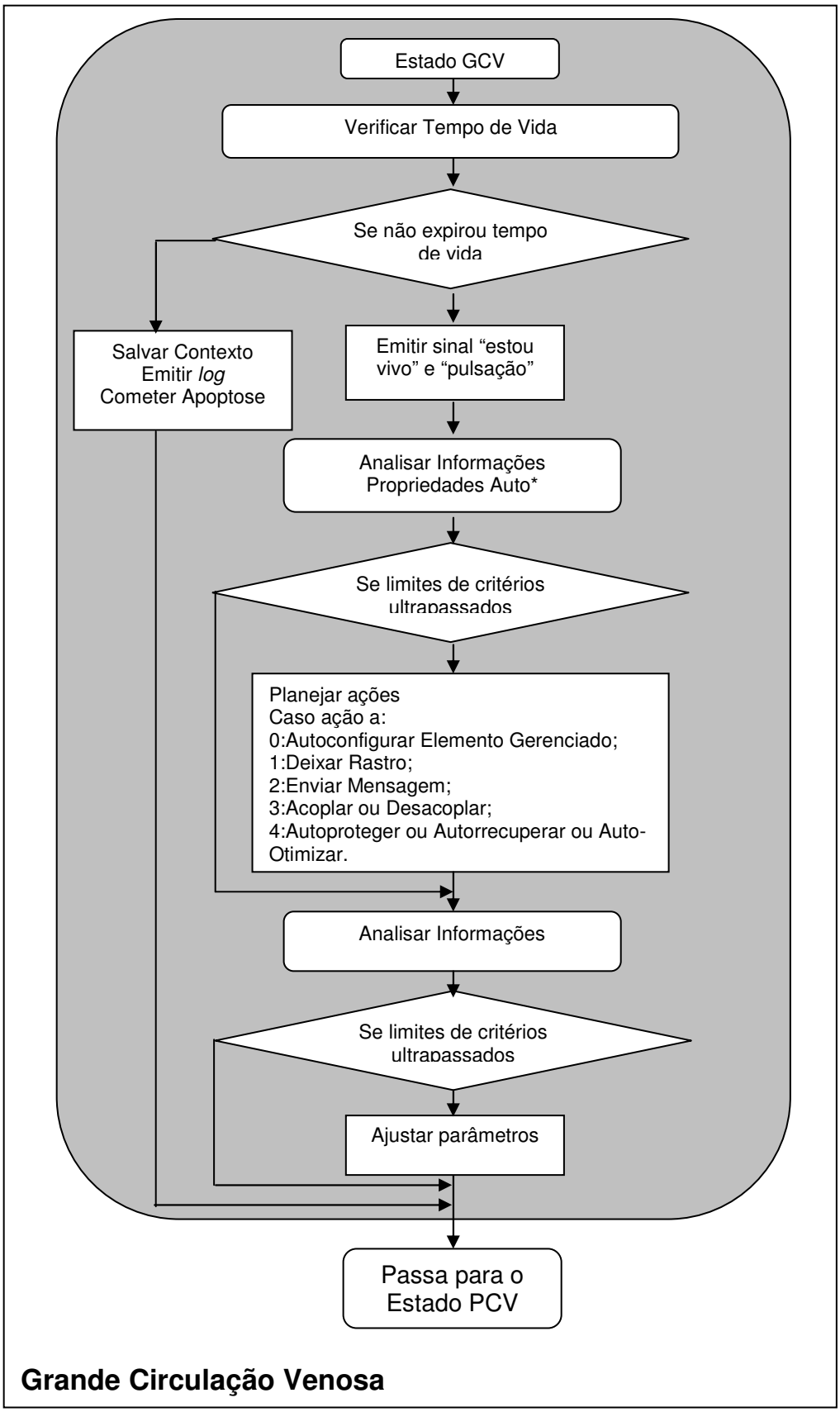

Figura B.5 – Detalhamento do Fluxograma do Elemento Gerenciador para o Estado GCV.

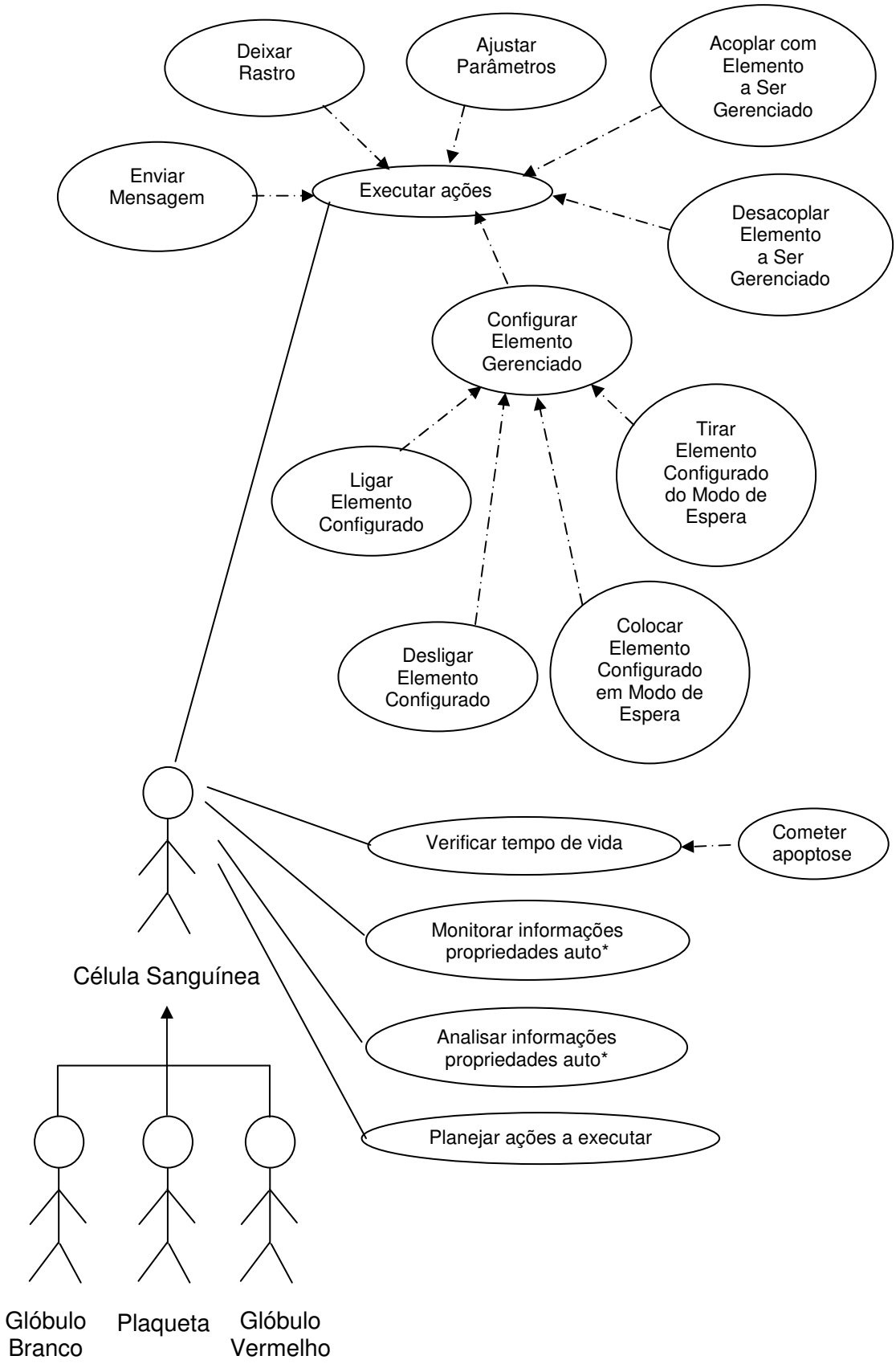

Figura B.6 – Diagrama de Casos de Uso das Células Sanguíneas no GCV (Comportamento Geral).

Estas mesmas funções estão representadas como casos de uso nas Figuras B.6 (comportamento geral) e B.7 (comportamento específico).

Estes casos de uso são relevantes para que o sistema como um todo permaneça funcionando ininterruptamente, mantendo sua estabilidade e alcançando o seu autogerenciamento através de ajustes frente às modificações do ambiente, analogamente a um organismo vivo. Esta estabilidade é alcançada através de *loops* de controle que cada elemento gerenciador circulatório-autonômico possui, e que no nosso modelo são representados pelas células sanguíneas artificiais, especializadas para o gerenciamento da recuperação (plaqueta), otimização (hemácias) e proteção (leucócitos).

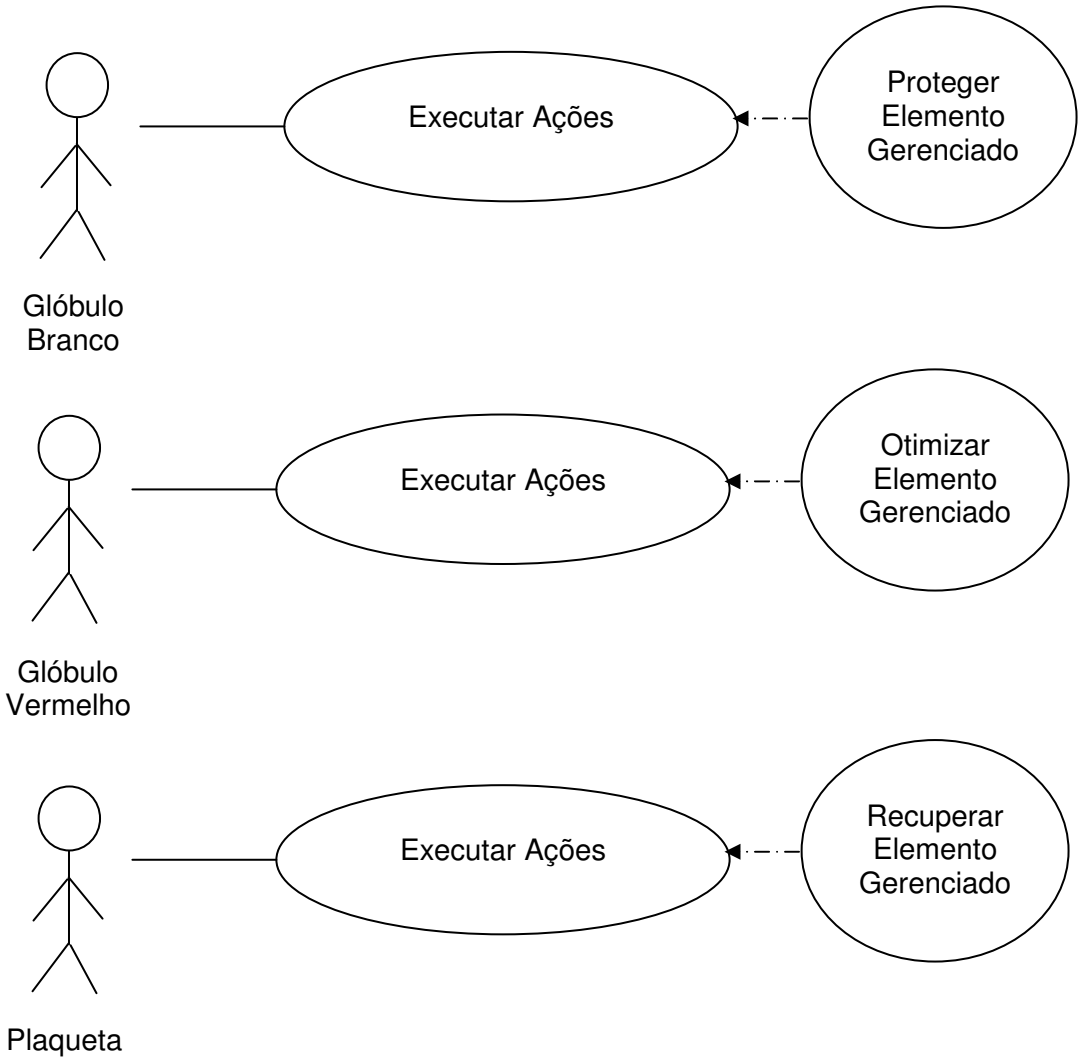

Figura B.7 – Diagrama de Casos de Uso das Células Sanguíneas no GCV (Comportamento Específico).

Na Figura B.8 são apresentados os casos de uso que podem ser executados pelas células do corpo ou órgãos. Representam componentes de software que para serem gerenciados pelos gerenciadores circulatório-autonômicos precisam se associar a eles (“acoplamento”).

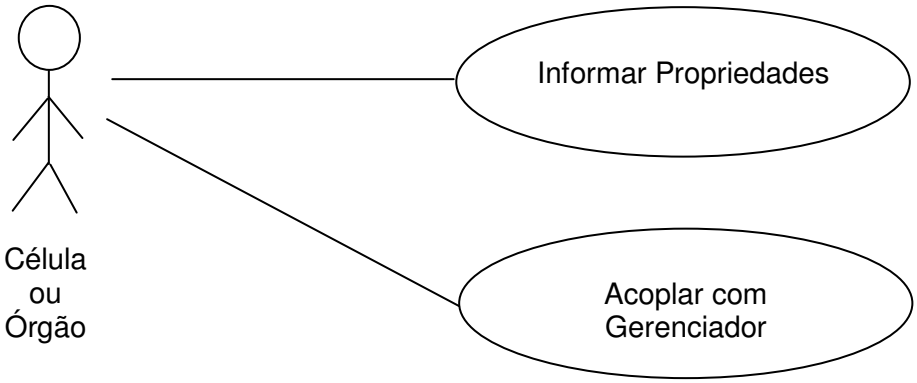

Figura B.8 – Diagrama de Casos de Uso das Células do Corpo e Órgãos na Grande Circulação.

Anexo C – Programas e Arquivos Auxiliares para *Jobs* OurGrid

C.1 Programa *Queens* Original

```
/***********************************************************************
 *  Compilation:  javac Queens.java
 *  Execution:    java Queens N
 *
 *  Solve the 8 queens problem using recursion and backtracing.
 *  Prints out all solutions.
 *
 *  Limitations: works for N <= 25, but slows down considerably
 *  for larger N.
 *
 *  Remark: this program implicitly enumerates all N^N possible
 *  placements (instead of N!), but the backtracing prunes off
 *  most of them, so it's not necessarily worth the extra
 *  complication of enumerating only permutations.
 *
 *
 *  % java Queens 3
 *
 *  % java Queens 4
 * * Q * *
 * * * * Q
 * Q * * *
 * * * Q *
 *
 * * * Q *
 * Q * * *
 * * * * Q
 * * Q * *
 *
 *  % java Queens 8
 * Q * * * * * *
 * * * * Q * * *
 * * * * * * * Q
 * * * * * Q * *
 * * Q * * * * *
 * * * * * * Q *
 * * Q * * * * *
 * * * Q * * * *
 *
 *  ...
 *
 ***********************************************************************/
```

```
public class Queens {

   /*************************************************************************
    * Return true if queen placement q[n] does not conflict with
    * other queens q[0] through q[n-1]
    *************************************************************************/
    public static boolean isConsistent(int[] q, int n) {
       for (int i = 0; i < n; i++) {
          if (q[i] == q[n])            return false;   // same column
          if ((q[i] - q[n]) == (n - i)) return false;   // same major diagonal
          if ((q[n] - q[i]) == (n - i)) return false;   // same minor diagonal
       }
       return true;
    }

   /*************************************************************************
    * Print out N-by-N placement of queens from permutation q in ASCII.
    *************************************************************************/
    public static void printQueens(int[] q) {
       int N = q.length;
       for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
             if (q[i] == j) System.out.print("Q ");
             else           System.out.print("* ");
          }
          System.out.println();
       }
       System.out.println();
    }


   /*************************************************************************
    *  Try all permutations using backtracking
    *************************************************************************/
    public static void enumerate(int N) {
       int[] a = new int[N];
       enumerate(a, 0);
    }

    public static void enumerate(int[] q, int n) {
       int N = q.length;
       if (n == N) printQueens(q);
       else {
          for (int i = 0; i < N; i++) {
             q[n] = i;
             if (isConsistent(q, n)) enumerate(q, n+1);
          }
       }
    }

    public static void main(String[] args) {
       int N = Integer.parseInt(args[0]);
       enumerate(N);
    }
}
```

C.2 Programa *Queens* Modificado

```
**************************************************************************/
public class Queens {

        int sol; /* attribute to store the number of solutions */

        /**************************************************************************
         *  Copyright © 2007, Robert Sedgewick and Kevin Wayne.
         *  Last updated: Tue Sep 29 16:17:41 EDT 2009.
         * Modified by Alberto Arkader Kopiler for parallelism purpose
         *  Compilation:  javac Queens.java
         *  Execution:    java Queens N T
         *  where N is the chess table size (NxN), and T corresponds to
         * the task starting by 0 (number of consistent permutations in the first two rows)
         * Solve the N queens problem using recursion and backtracing.
         *  Prints the number of solutions (not all the solutions!) to minimize
         * output file's size
         **************************************************************************/
         Queens () {
                 sol = 0;
         }

         public int getSol() {
                 return sol;
         }
          /**************************************************************************
           * Return true if queen placement q[n] does not conflict with
           * other queens q[0] through q[n-1]
           **************************************************************************/
           public static boolean isConsistent(int[] q, int n) {
              for (int i = 0; i < n; i++) {
                 if (q[i] == q[n])           return false;   // same column
                 if ((q[i] - q[n]) == (n - i)) return false;   // same major diagonal
                 if ((q[n] - q[i]) == (n - i)) return false;   // same minor diagonal
              }
              return true;
           }
          /**************************************************************************
           * Print out N-by-N placement of queens from permutation q in ASCII.
           **************************************************************************/
           public void printQueens(int[] q) {
              int N = q.length;
              for (int i = 0; i < N; i++) {
                 for (int j = 0; j < N; j++) {
                    if (q[i] == j) System.out.print("Q ");
                    else           System.out.print("* ");
                 }
                 System.out.println();
              }
              System.out.println();
           }
```

```
/**************************************************************************
 *  Try all permutations using backtracking
 **************************************************************************/
public void enumerate1(int N, int P) {
    int[] a = new int[N];
    int i, P1;
    boolean achou = false;
    i  = -1;
    P1 = 0;
    while (! achou){
       a[0]=Math.round(P1/N) % N;
       a[1]=Math.round(P1) % N;
       if (isConsistent(a, 1)) {i++;}
       if (i == P){achou = true;}
       P1++;
    }
    if (P1 > Math.pow(N,2)) System.out.println("Number of tasks exceeded!");
    else        enumerate(a, 2);
    }

public void enumerate(int[] q, int n) {
    int N = q.length;
    if (n == N) { sol++; /*printQueens(q);*/ }
    else {
       for (int i = 0; i < N; i++) {
          q[n] = i;
          if (isConsistent(q, n)) enumerate(q, n+1);
       }
    }
}

public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    int T = Integer.parseInt(args[1]);
    Queens queens = new Queens();
    queens.enumerate1(N,T);
    int sol=queens.getSol();
    System.out.println("#sol: "+sol);  /* print the number of solutions found */
}
```

C.3 Programa Gerador de *Jobs* com Tarefas *Queens* e *Probe Tasks* para Ourgrid

```
public class generator {

  public static void redCell()
  {
    System.out.println();
    System.out.println("task    :");
    System.out.println("init    : store scimark3.jar scimark3.jar");
    System.out.println("remote : java -cp $STORAGE/scimark3.jar commandline $PROC
>> $JOB_$TASK.ben ");
    System.out.println("final   : get $JOB_$TASK.ben $JOB_$TASK.ben");
    System.out.println();
  }

  public static void printJob()
  {
    System.out.println("job :");
    System.out.println("label    : grenchmark");
    System.out.println();
  }


  public static void printTask(int i, int N)
  {
    System.out.println();
    System.out.println("  task    :");
    System.out.println("  init    : put Queens.class Queens.class");
    System.out.print("  remote  : java Queens ");
    System.out.print(N);
    System.out.print(" ");
    System.out.print(i);
    System.out.print(" >> grenchmark_");
    System.out.print(N);
    System.out.print("_");
    System.out.print(i);
    System.out.println("_$PROC_$JOB_$TASK.gre");
    System.out.print("  final   : get grenchmark_");
    System.out.print(N);
    System.out.print("_");
    System.out.print(i);
    System.out.print("_$PROC_$JOB_$TASK.gre");
    System.out.print(" grenchmark_");
    System.out.print(N);
    System.out.print("_");
    System.out.print(i);
    System.out.print("_$PROC_$JOB_$TASK.gre");
    System.out.println();
  }
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		  int N = Integer.parseInt(args[0]);
		  int R = Integer.parseInt(args[1]);
		  if (N<2) System.out.println("N < 2!");
		  else {
			printJob();
			for (int i=0; i<((N-1)*(N-2)); i++){
				if (R != 0) {
					if (Math.random() < 0.15) redCell();
				}
				printTask(i,N);
			}
		  }
	}
}
```

C.4 Arquivos JDF para Ourgrid

Versão CIACOM com 6 rainhas

Esta versão de arquivo de definição de *jobs* para o *broker* modificado do OurGrid (CiaJob6.jdf) contem 20 tarefas Queens e cinco tarefas *probe* scimark3. A versão para o *broker* original (OurJob6,jdf) não contém as tarefas *probe*.

Linha de comando para execução de versão comum:

```
java generator 6 0 >> OurJob6.jdf
```

Linha de comando para execução de versão CIACOM:

```
java generator 6 1 >> CiaJob6.jdf
```

Conteúdo do arquivo CiaJob6.jdf:

```
job :
label    : CiaJob6
  task    :
  init    : store scimark3.jar scimark3.jar
  remote      : java -cp $STORAGE/scimark3.jar commandline $PROC >>
$JOB_$TASK.ben
  final   : get $JOB_$TASK.ben $JOB_$TASK.ben

  task    :
  init    : put Queens.class Queens.class
  remote  : java Queens 6 0 >> grenchmark_6_0_$PROC_$JOB_$TASK.gre
  final                 :       get       grenchmark_6_0_$PROC_$JOB_$TASK.gre
grenchmark_6_0_$PROC_$JOB_$TASK.gre

  task    :
  init    : store scimark3.jar scimark3.jar
  remote      : java -cp $STORAGE/scimark3.jar commandline $PROC >>
$JOB_$TASK.ben
  final   : get $JOB_$TASK.ben $JOB_$TASK.ben

...

  task    :
  init    : put Queens.class Queens.class
  remote  : java Queens 6 19 >> grenchmark_6_19_$PROC_$JOB_$TASK.gre
  final                 :       get       grenchmark_6_19_$PROC_$JOB_$TASK.gre
grenchmark_6_19_$PROC_$JOB_$TASK.gre
```

C.5 Arquivo .ben gerado pelo *scimark3*

leao_1.lsd.ufcg.edu.br@xmpp.ourgrid.org
721.0138324654954
FFT (1024): 445.52762630792597
SOR (100x100): 920.6665397539848
Monte Carlo : 295.7151780752678
Sparse matmult (N=1000, nz=5000): 514.6132843877856
LU (100x100): 1428.5465338025122

java.vendor: Sun Microsystems Inc.
java.version: 1.6.0_15
os.arch: i386
os.name: Linux
os.version: 2.6.31-14-vserver

Anexo D – Histórico e Tendências

O homem, desde os primórdios da humanidade, busca inventar dispositivos que lhe permitam imitar e controlar a natureza nas suas diversas formas. A invenção do barco, submarino e avião são alguns exemplos de criações humanas baseadas na observação da natureza.

A aplicação de modelos biológicos a sistemas computacionais também não é nova, aparecendo ao longo da história dos computadores ("cérebros eletrônicos") [89]. As primeiras tentativas de criar modelos e dispositivos artificiais, capazes de exibir propriedades semelhantes às que são encontradas nos seres vivos, datam do início da década de 50 do século passado, em paralelo com a origem dos próprios computadores.

Não foi por acaso que dois dos pesquisadores mais influentes na teoria da computação e na engenharia de computadores, Alan Turing e John Von Neumman, foram também os pioneiros na tentativa de criar modelos inspirados na biologia e aplicá-los a sistemas computacionais.

Em 1952, Turing publicou um trabalho sobre morfogênese, estudo da capacidade de uma entidade se modificar, se corrigir e obter novos e melhores resultados. Aproximadamente na mesma época, Von Neumann desenvolveu o conceito de autômato celular. Baseado neste conceito, Von Neumann propôs um modelo abstrato de autorreprodução semelhante ao modelo de reprodução celular baseado no DNA, descoberto pouco depois por Watson e Crick. Von Neumann também mostrou que a rede de autômatos poderia simular uma máquina de Turing, significando que era um sistema que poderia prover a computação universal.

Pouco antes, em 1943 foi publicado o trabalho de Warren McCulloch e Walter Pitts com o primeiro modelo de neurônio, que se considera ter dado origem às redes neurais artificiais.

A história das redes neurais artificiais mostra uma forte relação entre tentativas de simular a biologia e tentativas de construir uma nova ferramenta de software. Esta pesquisa é mais antiga do que a dos próprios computadores eletrônicos digitais modernos, uma vez que McCulloch e Pitts publicaram um modelo de neurônio que

incorporava pesos analógicos em um esquema binário em 1943. A pesquisa em redes neurais avançou nas décadas seguintes focando em arquiteturas, mecanismos de autoarranjo, reconhecimento de padrões e classificação. Entre estas pesquisas destaca-se o trabalho de Rosenblatt com os *perceptrons*. Entretanto, a falta de resultados satisfatórios naquela época causou uma "hibernação" nesta área até seu ressurgimento em 1982 com Hopfield. O progresso desde então firmou as redes neurais como ferramenta padrão para aprendizagem e reconhecimento de padrões.

No início da década de 50, o neurologista Grey Walter construiu robôs móveis, chamou-os de "tartarugas" e classificou-as como pertencendo à "espécie" *Machina Speculatrix*. Cada tartaruga possuía dois comportamentos elementares: após colidir com um obstáculo afastava-se dele e aproximava-se de fontes de luz, exceto se a fonte de luz fosse muito forte, caso em que também se afastava. Embora limitados a estes comportamentos, ainda assim os pequenos robôs eram capazes de exibir uma interação complexa com o ambiente e entre si.

Estes trabalhos e os conceitos que deles emergiram fazem parte da área de cibernética, que estuda a comunicação e o controle entre seres humanos e máquinas. Um dos exemplos típicos desta disciplina foi o homeostato, criado por Ross Ashby. Era um aparelho que, estabelecido um ponto de equilíbrio, se adaptava às perturbações induzidas externamente, de modo a recuperar esse ponto de equilíbrio inicial. Esta propriedade constitui também uma das principais características dos organismos vivos, que estão constantemente se adaptando às alterações ambientais.

O desenvolvimento da Inteligência Artificial, a partir da 2ª metade da década de 50, com uma perspectiva promissora mais orientada para aspectos cognitivos da mente, veio praticamente paralisar a pesquisa na área da cibernética. A pesquisa só foi retomada na década de 80, com a iniciativa de Christopher Langton, que organizou reunião de pesquisadores com interesse comum nos temas da Vida Artificial [89].

Alguns anos antes, John Holland introduziu, em 1975, a idéia de algoritmos genéticos, os quais utilizam processos genéticos simulados (seleção, cruzamento e mutação) para procurar em um vasto espaço de busca. De forma mais abrangente, um algoritmo genético é qualquer modelo baseado em população que utiliza operadores de seleção e recombinação para gerar novos pontos amostrais em um espaço de busca. Muitos algoritmos genéticos foram introduzidos por pesquisadores de uma perspectiva experimental. A maior parte deles tem interesse no algoritmo genético como

ferramenta de otimização. Mais recentemente John Koza, inspirado nestes algoritmos, descreveu a programação genética.

Outra iniciativa relacionada foi o surgimento do conceito de agentes [27] a partir do famoso *workshop* de Darmouth, realizado em 1956, no qual John MacCarthy cunhou o termo "inteligência artificial" (IA). A noção de agente pode ser exemplificada considerando o Teste de Turing, no qual uma pessoa testa e ao final decide se seu interlocutor é uma pessoa ou um programa de computador (agente). Ou seja, agente é um conceito inspirado no comportamento humano.

Como a exibição de um "comportamento inteligente" pelos agentes não correspondeu às expectativas, esta decepção causou a divisão da área de agentes de IA em duas vertentes: lógica e comportamental. Mas como a maioria dos pesquisadores desta área acha que para construir agentes capazes de realizar ações autônomas inteligentes são necessárias ambas as abordagens, até o presente momento dominam as arquiteturas híbridas.

O estudo de muliagentes começou em paralelo com a dos próprios agentes, onde questões como coordenação, cooperação, competição e interação entre agentes passaram a ser analisadas.

Datam da década de 1980 os sistemas multiagente usando *blackboard,* técnica de comunicação entre agentes no qual são compartilhadas as informações. Em 1980 foi cunhado o termo *Distributed Artificial Intelligence* (DAI) para melhor classificar os estudos de AI relacionados a multiagentes.

Em meados da década de 1990 o interesse da indústria na tecnologia de agentes aumentou, levando a criação da FIPA (Foundation for Intelligent Physical Agents) com o objetivo da especificação de padrões.

No final dos anos 90, pesquisadores em busca de aplicações realistas para sistemas multiagentes propuseram o desafio *RoboCup*: demonstrar que em 50 anos, um time de robôs jogadores de futebol poderá derrotar um time de excelentes jogadores de futebol humanos. Desde então são realizados torneios em várias partes do mundo com robôs reais ou virtuais.

O comércio eletrônico na Internet (*e-commerce*) passou a ser um campo fértil a ser explorado pela tecnologia de multiagentes. Além disso, desenvolvimento da tecnologia de agentes e da Internet levou à criação dos agentes móveis, como por exemplo, os *crawlers* do Google que visitam página na *web* contabilizando *links* para cálculo de indicadores que são utilizados para "ranquear" as respostas de seu mecanismo de busca.

Em 2001 a IBM tomou a iniciativa de desenvolver pesquisa em computação inspirada em sistemas biológicos, lançando a ideia da Computação Autônomica, baseada no sistema nervoso humano.

Técnicas inspiradas na incerteza dos seres humanos (Lógica Nebulosa o *Fuzzy*), no conhecimento humano de especialistas (*Expert Systems*), no sistema imunológico, na inteligência coletiva (*swarm intelligence*, *particle swarm optimization e ant colony optimization)* não podem deixar de ser citadas como outras iniciativas relevantes e inspiradas biologicamente, iniciadas nas décadas de 70, 80 e 90 do século passado.

Anexo E – Programa NetLogo CIACOM

O NetLogo [77] é um ambiente integrado de modelagem e programação de multiagentes criado por Uri Wilensky, no final da década de 90. Ele foi projetado mantendo o espírito da linguagem de programação Logo, ou seja, ser ao mesmo tempo simples para ser utilizado por novatos, e poderoso para satisfazer às necessidades de usuários avançados. O ambiente NetLogo facilita o estudo da emersão (surgimento) de fenômenos. Os modelos são programados na forma de instruções (ações) a serem executadas concorrentemente por centenas ou milhares de agentes independentes. Assim é possível acompanhar o comportamento dos agentes que compõem o modelo, desde o nível micro de comportamento individual até os padrões de nível macro que emergem pela interação de muitos indivíduos (agentes).

Portanto, o NetLogo é uma linguagem de programação simples, adaptada à modelagem e simulação de fenômenos naturais ou sociais. Além disso, outras características podem ser apresentadas, como por exemplo:

a) Adequação a modelagem de sistemas complexos que evoluem no tempo;
b) Possibilidade de modelar centenas ou milhares de indivíduos (pessoas, bactérias, insetos, nações, organizações, etc.) que interagem entre si e com o meio-ambiente;
c) Possibilidade de exploração das interações locais entre os indivíduos e os padrões macroscópicos (globais) emergentes dessas mesmas interações.

Figura E.1 – Tartarugas, que simbolizam os agentes do NetLogo, interagindo com o ambiente.

É também um ambiente de programação fácil de usar para criar ou testar tais modelos, pois:

a) Permite executar e experimentar simulações;
b) Permite criar modelos para testar hipóteses sobre sistemas descentralizados;
c) Vem munido de uma grande biblioteca contendo simulações em ciências naturais e sociais, que podem ser usadas e modificadas;
d) Possui uma linguagem simples para construção dos modelos, mesmo para aqueles que estão aprendendo sua primeira linguagem de programação;
e) Possui uma interface gráfica amigável.

O software de instalação do NetLogo está disponível em [77]. O software é compatível com vários sistemas operacionais nos quais esteja instalada a máquina virtual JAVA (JVM - *Java Virtual Machine*) versão 1.4.2 ou superior. Na versão para o Windows existe a opção para baixar a versão que inclui a JVM necessária, que é a opção recomendada para utilizadores menos experientes.

Neste anexo é apresentado um arquivo contendo o código compatível com o ambiente de programação e simulação de agentes NetLogo. Note que a primitiva “globals” serve para definir variáveis globais, ou seja, são “enxergadas” por todos os agentes, a primitiva “turtles-own” serve para definir variáveis locais, ou seja, só são enxergadas pelos próprios agentes e a primitiva “patches-own” serve para definir variáveis locais ao *patch* que só são enxergadas pelos agentes quando estes estão localizados sobre o *patch*, ou em suas vizinhanças.

A rotina “to criar” serve para inicializar o mundo do CIACOM, criando os agentes, inicializando variáveis e modificando atributos e cores dos *patches*. A rotina “to recolor-patch” serve para atualizar a cor dos *patches* de acordo com seu carregamento. A rotina “to executar” é a rotina principal e fica em *loop* eterno e está de acordo com a especificação na subseção 5.3.1. Em cada *tick* do *loop* são executadas instruções “em paralelo” para todos os agentes a partir da primitiva “ask” aplicada a “turtles”. Ao final de cada *loop* são feitas atualizações nos contadores, nos monitores e no *plot* do gráfico.

A cada *tick* cada agente analisa (percebe através de sensores) seu *patch*, pega um processo ou deposita um processo ou não faz nada (atua através de atuadores), deposita ou não feromônios, migra para outro lugar aleatoriamente (ação), se reproduz (ou não) e finalmente morre (ou não). Além disso, caso haja mais de um agente ocupando o mesmo patch, um é sorteado para morrer.

E.1 CIACOM_V1.nlogo

```
globals [geracao populacao taxa-cobertura num-tot-proc num-patches-red
num-patches-yellow num-patches-green ]

patches-own [rastro num-proc]

turtles-own [ temprocesso? meta-local]

to criar
   ca
   set geracao 0
   set populacao numero-globulos-vermelhos
   set num-tot-proc 0
   set num-patches-green 0
   set num-patches-red 0
   set num-patches-yellow 0
   set taxa-cobertura 0
   set-current-plot "Grafico pop. x ger."
   set-current-plot-pen "populacao"
   plotxy geracao populacao
   set-current-plot-pen "taxa de cobertura"
   plotxy geracao taxa-cobertura
   set-default-shape turtles "circle"
   crt populacao [
      setxy random-xcor random-ycor
      set color blue
      set meta-local meta-global
      set temprocesso? false
   ]
   ask patches [set pcolor white
            set num-proc random max-proc
            recolor-patch
            set num-tot-proc (num-proc + num-tot-proc)
            if pcolor = red [set num-patches-red num-patches-red + 1]
            if pcolor = green [set num-patches-green num-patches-green + 1]
            if pcolor = yellow [set num-patches-yellow num-patches-yellow + 1]
            ]

end

to recolor-patch
  ;;set pcolor scale-color green rastro 5 0.1
  ifelse (num-proc / max-proc) > 0.8 [
    set pcolor red
  ]
  [ ifelse ((num-proc / max-proc) >= 0.6 and (num-proc / max-proc) <= 0.8)
   [ set pcolor yellow
   ]
   [ set pcolor green
   ]
  ]
end
```

```
to executar
    ask turtles [
        analise
        deposito
        movimento velocidade
        reproducao
        morte
    ]
    if competicao? [ask turtles [ choque ] ]
    set geracao geracao + 1
    set populacao count turtles

    set-current-plot-pen "populacao"
    plotxy geracao populacao
    set-current-plot-pen "taxa de cobertura"
    plotxy geracao taxa-cobertura
    ;;diffuse rastro 0.5
    set num-patches-green 0
    set num-patches-red 0
    set num-patches-yellow 0
    set num-tot-proc 0
    ask patches
    [ set rastro rastro * (100 - taxa-evaporacao) / 100  ;; evapora rastro lentamente
    recolor-patch
    if pcolor = red [set num-patches-red num-patches-red + 1]
    if pcolor = green [set num-patches-green num-patches-green + 1]
    if pcolor = yellow [set num-patches-yellow num-patches-yellow + 1]
    set num-tot-proc (num-proc + num-tot-proc)
    ]
    set taxa-cobertura (count patches with [rastro > 30] / count patches) * 100
    tick
end

to-report max-proc?
  ifelse (num-proc = max-proc)
   [report true]
   [report false]
end

to-report min-proc?
  ifelse (num-proc = 0)
   [report true]
   [report false]
end
```

```
to-report carregada?
    ifelse (pcolor = red)
    [
     if (( meta-local = "Média") or (meta-local = "Baixa")) [report true]
    ]
    [if (pcolor = yellow)
      [if (meta-local = "Baixa")
        [report true]
      ]
    ]
    report false
end

to analise
   ifelse temprocesso?
   [ if (not carregada?)
     [if (not max-proc?)
       [set num-proc num-proc + 1
        set temprocesso? false
        set color blue
        recolor-patch
       ]
     ]
   ]
   [ if carregada?
     [if (not min-proc?)
       [set num-proc num-proc - 1
        set temprocesso? true
        set color magenta
        recolor-patch
       ]
     ]
   ]
  ; if rastro > 60 [
  ;      movimento velocidade
  ; ]
end

to movimento [ distancia ]
    rt random-float 360.0
    forward random-float distancia
end

to deposito
  set rastro rastro + 60
end
```

```
to reproducao
   if random 100 < taxa-de-natalidade [
      if not temprocesso? [hatch 1 [set meta-local meta-global set temprocesso? false
movimento 2] ]
   ]

end

to morte
   if random 100 < taxa-de-mortalidade [
      if not temprocesso? [die]
   ]
end

to choque
   without-interruption [
      while [ any? other turtles-here with [not temprocesso?]] [
         ask one-of turtles-here with [not temprocesso?] [die]
      ]
   ]
end
```

Anexo F – Programas JADE CIACOM

JADE [29] – Java Agent DEvelopment framework – é um ambiente de desenvolvimento para sistemas multiagente que está de acordo com a especificação FIPA (*Foundation for Intelligent Physical Agents*). Neste sistema estão incluídos:

a) Um ambiente de *runtime* onde os agentes JADE podem "viver" e que precisam estar ativos em um *host* antes que um ou mais agentes possam ser executados neste *host;*
b) Uma biblioteca de classes para os programadores desenvolverem seus agentes;
c) Um conjunto de ferramentas gráficas que permitem a administração e o monitoramento da atividade de agentes que estão ativos.

Cada instância do ambiente de *runtime* do JADE é chamado de *container*, pois pode conter diversos agentes. O conjunto de *containers* ativos é chamado de plataforma. Um *container* especial e único é o *main container* ou *container* principal. Ele precisa estar sempre ativo na plataforma e todos os outros *containers* devem ser registrados nele assim que são iniciados. Um outro *main container* iniciado em algum lugar na rede caracteriza outra plataforma.

Na figura F.1 é apresentado um esquema contendo duas plataformas, a primeira contendo três *containers* (sendo um principal) e a segunda apenas um.

No ambiente JADE os agentes são identificados por um nome único e podem se comunicar de forma transparente com outros agentes independendo de localização (*intracontainer*, *intercontainer* ou interplataforma).

Além de possibilitar o registro de outros *containers, o container* principal apresenta dois agentes especiais – o AMS e o DF – que são criados e iniciados automaticamente quando este é iniciado.

O AMS – Agent Management System é responsável pelo serviço de nomes (garante que o nome de um agente é único em uma plataforma) e representa a autoridade na plataforma (permite a criação e a destruição de agentes em *containers* remotos).

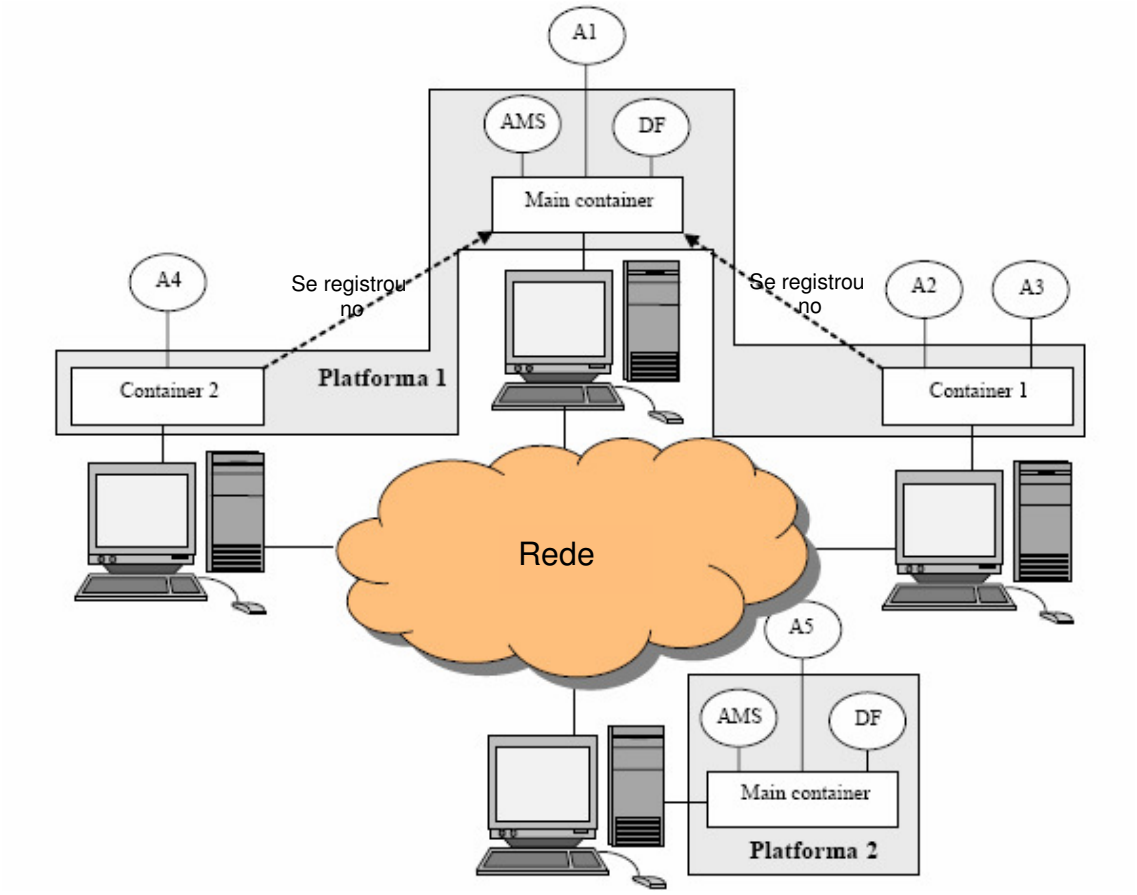

Figura F.1 – *Containers* e Plataformas JADE.

O DF – Directory Facilitator é responsável pelo serviço de “páginas amarelas”, ou seja, permite que um agente ache outros agentes que provêem os serviços por ele requisitados para alcançar seus objetivos.

Neste anexo são apresentadas as classes JAVA correspondentes a testes do CIACOM usando o ambiente JADE de desenvolvimento e execução de agentes, como especificado na subseção 5.2. Inicialmente são apresentadas as classes Criador, Celula e Corpo que estendem a classe Agent. Note que, a classe Criador cuida de criar todos outros agentes (Celula e Corpo), iniciá-los e movê-los. Para tanto, além do *container* principal é criado outro *container.* Esta implementação se baseou em exemplo, retirado da biblioteca do JADE, de agentes de “agradecimento” (*thanksagent*), para exercitar a comunicação direta entre agentes (CommunicationBehaviour). Toda classe Agent possui dois métodos principais: setup, que é chamado em sua inicialização correspondendo ao construtor na orientação a objetos e o takeDown, que é chamado em sua conclusão correspondendo ao destruidor na orientação a objetos. Na classe Célula note que o método setup é aproveitado para adição dos comportamentos (ConfigBehaviour) deste agente. Note

também o método afterMove que serve para execução de comandos após a migração do agente. Como a classe Corpo apresenta só o comportamento de comunicação, simplesmente responde o agradecimento da classe Celula.

Em seguida são apresentadas quatro classes correspondendo a comportamentos: (1) ConfigBehaviour que serve para concentrar a adição de todos os comportamentos de um agente, facilitando a depuração do código, pois isola as classes agente e comportamento, e é executada uma única vez (estende a classe OneShotBehaviour) (2) StateBehaviour na qual foi implementada a máquina de estados proposta (vide Figura 3.4), e que é executada para sempre com bloqueio de tempo de 2 segundos no final (estende a classe SimpleBehaviour), (3) ApoptosisBehaviour (estende a classe WakerBehaviour) que executa a apoptose do agente após ter decorrido seu tempo de vida e (4) TimeOutBehaviour (estende a classe *TickerBehaviour*) que é executada periodicamente, sempre que o contador de tempo é zerado, e serve para implementar o tempo de espera do agente para acoplamento.

Outras classes de comportamentos, como por exemplo, MobileBehaviour, ReflectionBehaviour, PulseBehaviour, IamAliveBehaviour, AnalisysBehaviour, foram criadas, mas não foram implementadas.

É importante notar que no ambiente JADE, assim como os comportamentos são adicionados inicialmente, podem ser alterados no decorrer da vida do agente, de acordo com o seu ambiente. Isto dá muita flexibilidade ao agente que pode mudar tanto seu comportamento para até se transformar em outro tipo de agente, bastando para isso ser dotado de todos comportamentos, pois estes podem coexistir e não estar ativos simultaneamente.

Nos testes realizados, o agente célula se movimentou levando consigo seu estado, ou seja, trata-se de um agente móvel que permite não só a migração de código, mas também do contexto da execução. Um teste que é deixado como trabalho futuro é a incorporação de um telefone celular ao sistema, fazendo o agente célula migrar para lá, uma vez que o JADE apresenta este recurso, ou seja, seu ambiente de execução atende tanto ao ambiente de execução JAVA para microcomputadores como para dispositivos móveis.

F.1 Criador.java

```
/**************************************************************
JADE - Java Agent DEvelopment Framework is a framework to develop
multi-agent systems in compliance with the FIPA specifications.
Copyright (C) 2000 CSELT S.p.A.
GNU Lesser General Public License

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation,
version 2.1 of the License.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA  02111-1307, USA.
 **************************************************************/

import jade.content.ContentElement;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.Result;
import jade.core.Agent;
import jade.core.AID;
import jade.core.Location;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.JADEAgentManagement.QueryPlatformLocationsAction;
import jade.domain.mobility.MobilityOntology;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.Runtime;
import jade.core.ProfileImpl;
import jade.util.leap.Iterator;
import jade.wrapper.*;

/**
 * This agent has the following functionality:
 * <ul>
 * <li> registers with the DF
 * <li> creates a list of agents
 * <li> each of this new agents registers with the DF
 * <li> the father agent sends a message of greeting to each of them
 * <li> it waits for an answer to the greeting
```

```
 * <li> it thanks the agents that have answered
 * </ul>
 * @author Fabio Bellifemine, TILab
 * @version $Date: 2007-03-12 11:07:03 +0100 (lun, 12 mar 2007) $ $Revision: 5944 $
 * @ Modified by Alberto Arkader Kopiler 2009-03
 **/
public class Criador extends Agent {

	private static final long serialVersionUID = 1L;
	// number of answer messages received.
	private int answersCnt = 0;

	public final static String GREETINGS = "GREETINGS";
	public final static String ANSWER = "ANSWER";
	public final static String THANKS = "THANKS";
	private AgentContainer ac1,ac2 = null;
	private AgentController cs1,cs2,corpo = null;
	private AID initiator = null;
	private Location l1,l2;

	protected void setup() {

		System.out.println(getLocalName()+" STARTED");
		Object[] args = getArguments();
		if (args != null && args.length > 0) {
			initiator = new AID((String) args[0], AID.ISLOCALNAME);
		}

		try {
			// create the agent description of itself
			DFAgentDescription dfd = new DFAgentDescription();
			dfd.setName(getAID());
			// register the description with the DF
			DFService.register(this, dfd);
			this.addBehaviour(new CommunicationBehaviour(this));
			System.out.println(getLocalName()+" REGISTERED WITH THE
DF");
		} catch (FIPAException e) {
			e.printStackTrace();
		}

		// create another two ThanksAgent
		String cs1AgentName = /*getLocalName()+*/"cs1";
		String cs2AgentName = /*getLocalName()+*/"cs2";
		String corpoAgentName = /*getLocalName()+*/"corpo";


		try {
			// create agent cs1 on the same container of the creator agent
			AgentContainer container =
(AgentContainer)getContainerController(); // get a container controller for creating new
agents
			cs1 = container.createNewAgent (cs1AgentName, "Celula", null);
			cs1.start();
```

```
                        System.out.println(getLocalName()+" CREATED AND STARTED
NEW THANKSAGENT:"+cs1AgentName + " ON CONTAINER
"+container.getContainerName());
                } catch (Exception any) {
                        any.printStackTrace();
                }

                try {
                        // create agent cs2 on the same container of the creator agent
                        AgentContainer container =
(AgentContainer)getContainerController(); // get a container controller for creating new
agents
                        cs2 = container.createNewAgent (cs2AgentName, "Celula", null);
                        cs2.start();
                        System.out.println(getLocalName()+" CREATED AND STARTED
NEW THANKSAGENT:"+cs2AgentName + " ON CONTAINER
"+container.getContainerName());
                } catch (Exception any) {
                        any.printStackTrace();
                }
                try {
                        // create agent corpo on the same container of the creator agent
                        AgentContainer container =
(AgentContainer)getContainerController(); // get a container controller for creating new
agents
                        corpo = container.createNewAgent (corpoAgentName, "Corpo",
null);
                        corpo.start();
                        System.out.println(getLocalName()+" CREATED AND STARTED
NEW THANKSAGENT:"+corpoAgentName + " ON CONTAINER
"+container.getContainerName());
                } catch (Exception any) {
                        any.printStackTrace();
                }

                // create agent cs1 on a new container
                // Get a hold on JADE runtime
                Runtime rt = Runtime.instance();
                // Create a default profile
                ProfileImpl p1 = new ProfileImpl(false);
                try {
                        // Create a new non-main container, connecting to the default
                        // main container (i.e. on this host, port 1099)
                        ac1 = rt.createAgentContainer(p1);
                        // create a new agent
                        ///*AgentController*/ cs1 =
ac1.createNewAgent(cs1AgentName,getClass().getName(),new Object[0]);
                        // fire-up the agent
                        //cs1.start();
                        //migra agente cs1 para container 1
                } catch (Exception e1) {
                        e1.printStackTrace();
                }
                // create agent cs2 on a new container
                ProfileImpl p2 = new ProfileImpl(false);
```

```
                try {
                        // Create a new non-main container, connecting to the default
                        // main container (i.e. on this host, port 1099)
                        ac2 = rt.createAgentContainer(p2);

//                      Register language and ontology
                        getContentManager().registerLanguage(new SLCodec());

        getContentManager().registerOntology(MobilityOntology.getInstance());
                        // create a new agent
                        ///*AgentController*/ cs2 =
ac2.createNewAgent(cs2AgentName,getClass().getName(),new Object[0]);
                        // fire-up the agent
                        //cs2.start();
                        //migra agente cs2 para container 1
//                      Get available locations with AMS
                        ACLMessage msg1 = new
ACLMessage(ACLMessage.REQUEST);
                        msg1.clearAllReceiver();
                        msg1.addReceiver(getAMS());
                        msg1.setLanguage(FIPANames.ContentLanguage.FIPA_SL);
                        msg1.setOntology(MobilityOntology.NAME);

        msg1.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
                        Action action = new Action();
                        action.setActor(getAMS());
                        action.setAction(new QueryPlatformLocationsAction());
                        getContentManager().fillContent(msg1, action);
                        send(msg1);

//                      Receive response from AMS
                        MessageTemplate mt = MessageTemplate.and(
                                        MessageTemplate.MatchSender(getAMS()),

        MessageTemplate.MatchPerformative(ACLMessage.INFORM));
                        ACLMessage resp = blockingReceive(mt);
                        ContentElement ce = getContentManager().extractContent(resp);
                        Result result = (Result) ce;
                        Iterator it = result.getItems().iterator();
                        while (it.hasNext()) {
                                Location loc = (Location)it.next();
                                if (ac1.getContainerName().equals(loc.getName()))
                                {
                                        l1=loc;
                                }
                                else if (ac2.getContainerName().equals(loc.getName()))
                                {
                                        l2=loc;
                                }
                        }
                        //migra agente cs2 para container 1
                        cs2.move (l1);
                        System.out.println(getLocalName()+" MIGRATED NEW
THANKSAGENT:"+cs2AgentName + " ON CONTAINER
"+l1.getName()/*ac1.getContainerName()*/);
```

```
                        //migra agente cs1 para container 2
                        cs1.move(l2);
                        System.out.println(getLocalName()+" MIGRATED NEW
THANKSAGENT:"+cs1AgentName + " ON CONTAINER
"+l2.getName()/*ac2.getContainerName()*/);

                } catch (Exception e2) {
                        e2.printStackTrace();
                }
                // send them a GREETINGS message
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.setContent(GREETINGS);

                msg.addReceiver(new AID(cs1AgentName, AID.ISLOCALNAME));
                msg.addReceiver(new AID(cs2AgentName, AID.ISLOCALNAME));

                send(msg);
                System.out.println(getLocalName()+" SENT GREETINGS MESSAGE
TO "+cs1AgentName+" AND "+cs2AgentName);

        /*}*/  /* IF YOU COMMENTED OUT THIS ELSE CLAUSE, THEN YOU WOULD
GENERATE
//              AN INTERESTING INFINITE LOOP WITH INFINTE AGENTS AND AGENT
                CONTAINERS BEING CREATED
                else {
                IAmTheCreator = true;
                doWait(2000); // wait two seconds
                }
         */

        }

        protected void takeDown() {
                // Deregister with the DF
                try {
                        DFService.deregister(this);
                        System.out.println(getLocalName()+" DEREGISTERED WITH
THE DF");
                } catch (FIPAException e) {
                        e.printStackTrace();
                }
        }
}
```

F.2 Celula.java

```
/*****************************************************************
JADE - Java Agent DEvelopment Framework is a framework to develop
multi-agent systems in compliance with the FIPA specifications.
Copyright (C) 2000 CSELT S.p.A.
@ Modified by Alberto Arkader Kopiler 2009-03
GNU Lesser General Public License

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation,
version 2.1 of the License.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA  02111-1307, USA.
 *****************************************************************/

import java.util.Random;

import jade.core.Agent;
import jade.core.AID;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;


/**
 * This agent has the following functionality:
 * <ul>
 * <li> registers with the DF
 * <li> creates a list of agents
 * <li> each of this new agents registers with the DF
 * <li> the father agent sends a message of greeting to each of them
 * <li> it waits for an answer to the greeting
 * <li> it thanks the agents that have answered
 * </ul>
 * @author Fabio Bellifemine, TILab
 * @version $Date: 2007-03-12 11:07:03 +0100 (lun, 12 mar 2007) $ $Revision: 5944 $
 **/
public class Celula extends Agent {

        private static final long serialVersionUID = 1L;

        /* ESTADOS POSSÍVEIS */
```

```
	public final static String CRIACAO = "CRIAÇÃO";
	public final static String ANALISE = "ANÁLISE";
	public final static String DISTRIBUICAO = "DISTRIBUIÇÃO";
	public final static String ACOPLAMENTO = "ACOPLAMENTO";
	public final static String UNKNOWN = "DESCONHECIDO";
	public final static int MAX_TENTATIVAS = 3;
	public final static int CRIACAO_EST=0;
	public final static int ANALISE_EST=1;
	public final static int DISTRIBUICAO_EST=2;
	public final static int ACOPLAMENTO_EST=3;
	public final static int UNKNOWN_EST=4;
	public enum  Estados {CRIACAO_EST, ANALISE_EST, DISTRIBUICAO_EST,
ACOPLAMENTO_EST, UNKNOWN_EST};

	public final static int ANALISE_NUM = 0;
	private AID initiator = null;

	private boolean  timeout, acoplado, fim_de_vida, criado, analisado, migra;
	private int nmigracoes;
	private Estados estado;

	public void setEstado(Estados estado) {
		this.estado=estado;
	}

	public void setEstado(int estado) {
		switch (estado) {
		case CRIACAO_EST: this.estado=Estados.CRIACAO_EST;
		break;
		case ANALISE_EST: this.estado=Estados.ANALISE_EST;
		break;
		case DISTRIBUICAO_EST: this.estado=Estados.DISTRIBUICAO_EST;
		break;
		case ACOPLAMENTO_EST:
this.estado=Estados.ACOPLAMENTO_EST;
		break;
		default:  this.estado=Estados.UNKNOWN_EST;
		break;
		}
	}

	public Estados getEstado() {
		return this.estado;
	}

	public String getNomeEstado () {
		switch (estado) {
		case CRIACAO_EST: return CRIACAO;
		case ANALISE_EST: return ANALISE;
		case DISTRIBUICAO_EST: return DISTRIBUICAO;
		case ACOPLAMENTO_EST: return ACOPLAMENTO;
		default:  return UNKNOWN;
		}
	}
```

```
	protected void setup() {

		System.out.println(getLocalName()+" STARTED");
		Object[] args = getArguments();
		estado=Estados.CRIACAO_EST; nmigracoes=0;
		acoplado = false; fim_de_vida = false;
		criado = true;  analisado = false;
		migra = false; timeout=false;
		if (args != null && args.length > 0) {
			initiator = new AID((String) args[0], AID.ISLOCALNAME);
		}
		try {
			// create the agent description of itself
			DFAgentDescription dfd = new DFAgentDescription();
			dfd.setName(getAID());
			// register the description with the DF
			DFService.register(this, dfd);
			System.out.println(getLocalName()+" REGISTERED WITH THE
DF");
			this.addBehaviour(new ConfigBehaviour(this));
		} catch (FIPAException e) {
			e.printStackTrace();
		}
	}

	protected void takeDown() {
		// Deregister with the DF
		try {
			DFService.deregister(this);
			System.out.println(getLocalName()+" DEREGISTERED WITH
THE DF");
		} catch (FIPAException e) {
			e.printStackTrace();
		}
	}

	protected void afterMove() {
//		 Se condição atendida incrementa o número de tentativas
		nmigracoes++;
		migra=false;
	}

	public boolean isAcoplado() {
		return acoplado;
	}

	public void setAcoplado(boolean acoplado) {
		this.acoplado = acoplado;
	}

	public boolean isFim_de_vida() {
		return fim_de_vida;
	}

	public void setFim_de_vida(boolean fim_de_vida) {
```

```
		this.fim_de_vida = fim_de_vida;
	}

	public boolean isCriado() {
		return criado;
	}

	public void setCriado(boolean criado) {
		this.criado = criado;
	}

	public boolean isAnalisado() {
		return analisado;
	}

	public void setAnalisado(boolean analisado) {
		this.analisado = analisado;
	}

	public int getNmigracoes() {
		return nmigracoes;
	}

	public void incNtentativas() {
		this.nmigracoes = nmigracoes++;
	}

	public boolean isMigra() {
		return migra;
	}

	public void setMigra(boolean migra) {
		this.migra = migra;
	}

	public boolean isTimeout() {
		return timeout;
	}

	public void setTimeout(boolean timeout) {
		this.timeout = timeout;
	}

}
```

F.3 Corpo.java

```
/**************************************************************
JADE - Java Agent DEvelopment Framework is a framework to develop
multi-agent systems in compliance with the FIPA specifications.
Copyright (C) 2000 CSELT S.p.A.
@ Modified by Alberto Arkader Kopiler 2009-03
GNU Lesser General Public License

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation,
version 2.1 of the License.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA  02111-1307, USA.
**************************************************************/


import java.util.Random;

import jade.core.Agent;
import jade.core.AID;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;


/**
 * This agent has the following functionality:
 * <ul>
 * <li> registers with the DF
 * <li> creates a list of agents
 * <li> each of this new agents registers with the DF
 * <li> the father agent sends a message of greeting to each of them
 * <li> it waits for an answer to the greeting
 * <li> it thanks the agents that have answered
 * </ul>
 * @author Fabio Bellifemine, TILab
 * @version $Date: 2007-03-12 11:07:03 +0100 (lun, 12 mar 2007) $ $Revision: 5944 $
 **/
public class Corpo extends Agent {

        private static final long serialVersionUID = 1L;
```

```
	private AID initiator = null;

	protected void setup() {

		System.out.println(getLocalName()+" STARTED");
		Object[] args = getArguments();
		if (args != null && args.length > 0) {
			initiator = new AID((String) args[0], AID.ISLOCALNAME);
		}
		try {
			// create the agent description of itself
			DFAgentDescription dfd = new DFAgentDescription();
			dfd.setName(getAID());
			// register the description with the DF
			DFService.register(this, dfd);
			System.out.println(getLocalName()+" REGISTERED WITH THE
DF");
			this.addBehaviour(new CommunicationBehaviour(this));
		} catch (FIPAException e) {
			e.printStackTrace();
		}
	}

	protected void takeDown() {
		// Deregister with the DF
		try {
			DFService.deregister(this);
			System.out.println(getLocalName()+" DEREGISTERED WITH
THE DF");
		} catch (FIPAException e) {
			e.printStackTrace();
		}
	}

}
```

F.4 ConfigBehaviour.java

```
import java.util.Random;

import jade.core.*;
import jade.core.behaviours.*;

/**
@author Giovanni Caire - CSELT S.p.A
@version $Date: 2002-07-16 11:20:11 +0200 (mar, 16 lug 2002) $ $Revision: 3271 $
@ Modified by Alberto Arkader Kopiler 2009-03
*/

class ConfigBehaviour extends OneShotBehaviour
{
	Random random;
	Agent agente;

	ConfigBehaviour(Agent a)
	{
		super(a);
		random=new Random();
		a.addBehaviour(new StateBehaviour(a));
		a.addBehaviour(new ApoptosisBehaviour
(a,random.nextInt(200)+20000));
		a.addBehaviour(new
TimeoutBehaviour(a,(100*random.nextInt(200)+20000)/2));
		a.addBehaviour(new CommunicationBehaviour(a));
		a.addBehaviour(new MobileBehaviour (a));
		a.addBehaviour(new MonitorBehaviour (a));
		a.addBehaviour(new IamAliveBehaviour (a));
		a.addBehaviour(new PulseBehaviour (a));
		a.addBehaviour(new ReflectionBehaviour (a));
	}

	public void action()
	{
		System.out.println(myAgent.getLocalName()+" ADICIONADOS
COMPORTAMENTOS");
	}

}
```

F.5 StateBehaviour.java

```
import jade.core.*;
import jade.core.behaviours.*;
/**
The behaviour uses two resources, in particular the counter cnt
and the flag cntEnabled, of the agent object.
It increments by one its value, displays it, blocks
for two seconds, and repeats forever.
@author Giovanni Caire - CSELT S.p.A
@version $Date: 2002-07-16 11:20:11 +0200 (mar, 16 lug 2002) $ $Revision: 3271 $
@ Modified by Alberto Arkader Kopiler 2009-03
*/

class StateBehaviour extends SimpleBehaviour
{
	StateBehaviour(Agent a)
	{
		super(a);
	}

	public boolean done()
	{
		return false;
	}

	public void action()
	{
		switch (((Celula)myAgent).getEstado()) {
			case CRIACAO_EST:trataCriacao();break;
			case ANALISE_EST:trataAnalise();break;
			case DISTRIBUICAO_EST:trataDistribuicao();break;
			case ACOPLAMENTO_EST:trataAcoplamento();break;
			default:;break;
		}
		// Bloqueia o comportamento por 2 segundos
		block(2000);
		return;
	}

	private void trataDistribuicao() {
		if(((Celula)myAgent).isAcoplado()) {

((Celula)myAgent).setEstado(((Celula)myAgent).ACOPLAMENTO_EST);
				System.out.println(myAgent.getLocalName()+" ESTADO
mudou para "+((Celula)myAgent).getNomeEstado());
			}
			else {
				if (((Celula)myAgent).isFim_de_vida()) {

	((Celula)myAgent).setEstado(((Celula)myAgent).ANALISE_EST);
					System.out.println(myAgent.getLocalName()+"
ESTADO mudou para "+((Celula)myAgent).getNomeEstado());
				}
```

```
                                    else if(((Celula)myAgent).getNmigracoes() <
                                ((Celula)myAgent).MAX_TENTATIVAS) {
                                                ((Celula)myAgent).setMigra(true);
                                    }
                                    else {

        ((Celula)myAgent).setEstado(((Celula)myAgent).ANALISE_EST);
                                                System.out.println(myAgent.getLocalName()+"
ESTADO mudou para "+((Celula)myAgent).getNomeEstado());
                                    }
        }
        }
        private void trataCriacao() {
                if (((Celula)myAgent).isCriado()) {

        ((Celula)myAgent).setEstado(((Celula)myAgent).DISTRIBUICAO_EST);
                        System.out.println(myAgent.getLocalName()+" ESTADO mudou
para "+((Celula)myAgent).getNomeEstado());
                };
        }
        private void trataAnalise() {
                if (((Celula)myAgent).isAnalisado()) {

        ((Celula)myAgent).setEstado(((Celula)myAgent).CRIACAO_EST);
                  System.out.println(myAgent.getLocalName()+" ESTADO mudou para
"+((Celula)myAgent).getNomeEstado());
                };
        }
        private void trataAcoplamento() {
                if (((Celula)myAgent).isFim_de_vida() || !((Celula)myAgent).isAcoplado()
) {
                        ((Celula)myAgent).setEstado(((Celula)myAgent).ANALISE_EST);
                          System.out.println(myAgent.getLocalName()+" ESTADO
mudou para "+((Celula)myAgent).getNomeEstado());
                        }
        }

}
```

F.6 ApoptosisBehaviour.java

```
import jade.core.*;
import jade.core.behaviours.*;

/**
The behaviour uses two resources, in particular the counter cnt
and the flag cntEnabled, of the agent object.
It increments by one its value, displays it, blocks
for two seconds, and repeats forever.
@author Giovanni Caire - CSELT S.p.A
@version $Date: 2002-07-16 11:20:11 +0200 (mar, 16 lug 2002) $ $Revision: 3271 $
@ Modified by Alberto Arkader Kopiler 2009-03
*/

class ApoptosisBehaviour extends WakerBehaviour
{

public ApoptosisBehaviour(Agent a, long timeout) {
                        super(a, timeout);
                        // TODO Auto-generated constructor stub
                }

protected void handleElapsedTimeout() {
        System.out.println(((Celula) myAgent).getLocalName()+" COMETEU
APOPTOSE");
        ((Celula)myAgent).setFim_de_vida(true);

}

}
```

F.7 TimeOutBehaviour.java

```
import jade.core.*;
import jade.core.behaviours.*;

/**
@author Giovanni Caire - CSELT S.p.A
@version $Date: 2002-07-16 11:20:11 +0200 (mar, 16 lug 2002) $ $Revision: 3271 $
@ Modified by Alberto Arkader Kopiler 2009-03
*/

class TimeoutBehaviour extends TickerBehaviour
{

public TimeoutBehaviour(Agent a, long period) {
                super(a, period);
                // TODO Auto-generated constructor stub
        }

@Override
protected void onTick() {
        // TODO Auto-generated method stub
        System.out.println(((Celula) myAgent).getLocalName()+" TIMEOUT");
        ((Celula)myAgent).setTimeout(true);
}

}
```

Anexo G – Glossário

.ben – extensão de arquivo de benchmark.

.gre – extensão de arquivo de grenchmark.

ABLE – Agent Building and Learning Environment. Ambiente desenvolvido pela IBM para construção de agentes inteligentes com possibilidade de aprendizagem utilizando técnicas de *machine learning* (redes neurais, classificadores bayesianos e árvores de decisão).

Acoplamento – característica do CIACOM em que os agentes de gerenciamento não ficam permanentemente conectados, necessitando do acoplamento para estabelecer uma conexão.

Agente Ação – agente do CIACOM responsável pela interface com o operador humano, criação e distribuição dos agentes criadores.

Agente Célula – agente móvel do CIACOM responsável pelo gerenciamento dos agentes corpo.

Agente Corpo – agente do CIACOM que faz o papel da aplicação.

Agente Criador – agente do CIACOM responsável pela criação e distribuição dos agentes Célula.

Agentes Móveis – são agentes de software capazes de se transmitir de uma máquina local (seu programa e seu estado) por uma rede de computadores e reiniciar a execução do código em uma máquina remota.

Alostase – manutenção da estabilidade através de modificações.

AMS – Agent Management System é responsável pelo serviço de nomes do (garante que o nome de um agente é único em uma plataforma) e representa a autoridade na plataforma (permite a criação e a destruição de agentes em *containers* remotos).

Análise Ativa – Análise reativa na qual um agente tem que tomar uma decisão rapidamente.

Análise Passiva – Análise deliberativa na qual um agente não precisa tomar uma decisão imediatamente.

Apoptose – ou “morte celular programada” é um tipo de autodestruição da célula. Constitui um processo essencial para a manutenção do desenvolvimento dos seres vivos, sendo importante para eliminar células supérfluas ou defeituosas.

Autoconfiguração – Autopropriedade da Computação Autonômica que se refere à capacidade do sistema em se adaptar automaticamente às mudanças do ambiente.

Autogerenciamento – Gerenciamento autônomo, ou seja, o sistema gerencia a sim mesmo e a outros sistemas de forma autônoma. Na Computação Autonômica diz-se dos sistemas que apresentam as quatro autopropriedades.

Autonômico – relativo à Computação Autonômica e ao sistema nervoso autônomo.

Autônomo – característica de um agente ter iniciativa própria, autonomia.

Auto-otimização – autopropriedade da Computação Autonômica que se refere à capacidade dos sistemas de procurarem melhorar continuamente.

Autopropriedades – ver Propiedades auto*.

Autoproteção – autopropriedade da Computação Autonômica que se refere à capacidade que um sistema tem de antecipar, detectar, identificar e se proteger de ataques mal intencionados externos (fatores extrínsecos)

Autorrecuperação – autopropriedade da Computação Autonômica que se refere à capacidade que um sistema tem de antecipar, detectar, identificar e se proteger do mau funcionamento do sistema devido a fatores internos (fatores intrínsecos), evitando, por exemplo, efeitos em cascata.

Bag of tags – saco de tarefas independentes, ou seja, podem ser executadas em paralelo sem que a entrada de uma dependa da saída de outra.

Benchmark – medida do índice de desempenho de um computador.

Binpacking – estratégia informada de escalonamento em que o matchmaking é feito a partir de listas de tarefas e máquinas ordenadas, por exemplo, por seu tamanho e capacidade de processamento.

Broker – programa que gerencia o escalonamento e a execução de tarefas em um *grid* computacional.

Carregamento – neste trabalho se refere ao percentual da capacidade de processamento e/ou armazenamento atingido por uma máquina da rede.

CIACOM – acrônimo para CIrculatory Autonomic COmputing Model. É como é chamado o modelo híbrido inspirado biologicamente, baseado nos sistemas circulatório e nervoso humano, desenvolvido meste trabalho.

Computação Amorfa – é o desenvolvimento de princípios organizacionais e linguagens de programação de forma a obter comportamento coerente a partir da contribuição de uma infinidade de partes não confiáveis interconectadas de forma desconhecida, irregular e variante no tempo.

Computação Autonômica – iniciativa da IBM de modelo inspirado no sistema nervoso cujo objetivo é o autogerenciamento de sistemas computacionais.

Computação Circulatória – complemento à Computação Autonômica desenvolvido neste trabalho.

Computação Edge – é a disponibilização de recursos computacionais o mais próximo possível de um ponto da rede de forma a trazer o maior benefício possível para o cliente e para o negócio.

Container – entidade do JADE que serve como entreposto para agentes.

Deadlock – beco sem saída. Situação em que um processo fica travado na espera de um evento que nunca ocorrerá.

DF – Directory Facilitator do ambiente JADE é responsável pelo serviço de “páginas amarelas”, ou seja, permite que um agente ache outros agentes que proveem os serviços por ele requisitados para alcançar seus objetivos

Escalonador – programa ou algoritmo para realizar o matchmaking entre a tarefa e a máquina que irá executá-la, colocando o par resultante em uma fila de execução.

Estigmergia – do inglês stigmergy. Colaboração através do meio físico utilizando comunicação indireta.

Feromônio – mensagem química volátil depositada por formigas ou emitida por seres humanos. Ou seja, se evapora certo tempo após sua emissão.

FIPA – Foundation for Intelligent Physical Agents. Associação internacional sem fins lucrativos criada para promover a utilização de agentes de software e desenvolver especificações de forma a suportar sua interoperabilidade.

GCA – acrônimo para Grande Circulação Arterial.

GCV – acrônimo para Grande Circulação Venosa.

Grande circulação – maior ciclo no sistema circulatório em que o sague sai do coração para (quase) todas as partes do corpo.
Grenchmark – medida do índice de desempenho de um *grid* computacional.

Grid – grade computacional. Conjunto (*cluster*) heterogêneo de computadores cujo objetivo é facilitar a a execução de tarefas em paralelo de forma a obter resultados mais rapidamente.

Homeostase – equilíbrio.

Inspiração Biológica – a partir da observação de organismos vivos, fazer analogias que podem se transformar em técnicas aplicadas à computação.

Inteligência Coletiva – inteligência distribuída por toda parte que competindo e cooperando leva ao estabelecimento de um equilíbrio.

Inteligência de enxames – interação de inúmeras partes (insetos) realizando tarefas simples que fazem emergir um comportamento complexo do conjunto como um todo.

JADE – Java Agent DEvelopment Framework. Ambiente de software aberto e descentralizado para construção de multiagentes implementado em JAVA respeitando padronização da FIPA e mantido pela Telecom Itália.

JDF – Job Description File ou Job Definition File. Extensão padrão para arquivo com formato específico para submissão de jobs no grid computcional OurGrid.

JVM – Java Virtual Machine. A partir de um programa fonte em Java é gerado um código padrão independente de máquina e sistema operacional. O JVM, desenvolvido para cada sistema operacional, permite a execução deste código

Limpo, Contaminado, Protegido – modelo para desinfecção de rede de computadores, no qual todos os nós da rede são assumidos inicialmente como contaminados Conforme vão sendo desinfectados ou vacinados passam, respectivamente para os estados limpo e protegido.

Loop de controle – laço de execução de comandos, geralmente ininterrupto, que serve para controlar um processo.

Loop global – laço de controle em que todos os agentes são executados.

Loop local – laço de controle executado por cada agente individualmente.

Makespan – período de tempo medido a partir do início da execução da primeira tarefa de um *job* e o término da execução da última tarefa deste mesmo *job*.

MAPE – Monitor, Analyse, Plan and Execute. Sequência completa de atuação de um agente (monitoração, análise, planejamento e execução) na arquitetura MAPE.

Matchmaking – termo usado para o casamento entre a tarefa e a máquina onde ela será executada, feito pelos escalonadores, quando da submissão de *jobs* em *grids* computacionais.

Mico-preto – Jogo de cartas para crianças cujo objetivo é formar casais de animais. O mote do jogo é o mico preto, carta que não possui par. Um jogador é considerado perdedor quando, depois de formados todos os pares, fica com o "mico preto" na mão.

MyGrid – broker com código aberto para controlar a execução de tarefas em um *grid* computacional.

NetLogo – é um ambiente integrado de modelagem e programação de multiagentes criado por Uri Wilensky, no final da década de 90. O ambiente NetLogo facilita o estudo da emersão (surgimento) de fenômenos. Os modelos são programados na forma de instruções (ações) a serem executadas concorrentemente por centenas ou milhares de agentes independentes.

NIST – National Institute of Standards and Technology. Agência federal norte-americana que promove o desenvolvimento de padrões e da tecnologia, a competitvidade e a inovação.

OurGrid – grade computacional oportunista, aberta e cooperativa para execução de tarefas *bag-of-tags* em paralelo. Os interessados se juntam à comunidade podendo, assim, executar aplicações paralelas. O poder computacional do OurGrid é obtido através de recursos ociosos dos participantes da comunidade.

PAC – Personal Autonomic Computing. É a Computação Autonônomica em nível pessoal.

Parassimpático – parte do sistema nervoso autônomo responsável de forma involuntária pela desaceleração funcional do organismo em resposta a estímulos de "descanso e digestão".

Patch – definição de um local (área) posicionado (coordenadas x,y) no mundo do ambiente NetLogo.

PCA – acrônimo para Pequena Circulação Arterial.

PCV – acrônimo para Pequena Circulação Venosa.

Peer – máquina componente de um *grid* computacional responsável pelo controle de um conjunto de workers e pela comunicação com os outros peers e os brokers.

Pequena Circulação – menor ciclo no sistema circulatório em que o sague sai do coração e vai para o pulmão para ser renovado.

Pervasivo – que se infiltra, que penetra; espalhado, difuso; penetrante.

Plataforma – no ambiente JADE uma plataforma é um conjunto composto pelo container principal (main container) e opcionalmente mais containers. Serve para definir conjuntos de máquinas em uma rede.

Probe tasks – são tarefas de prospecção que servem, por exemplo, para investigar e avaliar indicadores de desempenho.

Propriedades auto* – ou autopriedades são as quatro propriedades fundamentais definidas na Computação Autonômica para um sistema alcançar o autogerenciamento: autoproteção, autorrecuperação, autoconfiguração e auto-otimização.

Queens – problema das n rainhas. Em um tabuleiro de xadrez de tamanho n x n devem ser encontraradas todas as soluções possíveis para posicionamento de n rainhas, de forma que elas não fiquem dispostas simultaneamente na mesma coluna, linha e diagonal.

Rastro – mensagem indireta depositada localmente por um agente.

Replicação – estratégia que consiste na execução de uma réplica (duplicata) de uma tarefa no ambiente de *grid*, desde que a execução da tarefa original ainda não tenha concluído e haja máquina disponível. Se conjugada à estratégia de *workqueue*, serve para compensar em parte sua aleatoriedade, pelo provável mau casamento entre tarefa e máquina, mas em troca de ciclos de máquina (desperdício).

Scale free networks – a noção de redes sem escala que caracteriza uma variedade de sistemas complexos nos quais alguns nós apresentam um enorme número de conexões enquanto que outros apresentam poucas conexões.

Scimark – benchmark científico para medir o índice de desempenho de um computador para cálculos científicos.

Simpático – parte do sistema nervoso autônomo responsável de forma involuntária pela aceleração funcional do organismo em resposta a estímulos de “luta e fuga”.

Sistema ultraestável – idealizado por Ashby são constituídos por dois *loops* fechados: um *loop* controla as pequenas perturbações e um segundo *loop* é responsável pelas perturbações com maior duração.

Small world – conclusão do experimento de Stanley Milgram realizado em 1967 para identificação de redes sociais em que foram verificados “seis níveis de separação” de relacionamento entre as pessoas.

Suscetível, Infectado, Resistente – modelo para desinfecção de rede de computadores, no qual todos os nós da rede são considerados inicialmente como suscetíveis. Conforme vão sendo infectados ou vacinados passam, respectivamente para os estados infectado e resistente.

Tampão Plaquetário – etapa no processo de coagulação sanguínea em que os glóbulos vermelhos se juntam às plaquetas formando um coágulo no intuito de cessar o sangramento.

Turtle – definição *default* de agentes no ambiente NetLogo.

Ubíquo – que está ao mesmo tempo em toda a parte, onipresente.

Variáveis Essenciais – grandezas consideradas como fundamentais para o equilíbrio do organismo. São variáveis que são constantemente monitoradas, para, em caso de variação, medidas sejam tomadas para manter o organismo em equilíbrio.

Worker – denominação da máquina de trabalho no ambiente OurGrid.

Workqueue – estratégia não informada de escalonamento de tarefas no OurGrid na qual o casamento entre tarefa e máquina que irá executá-la é feita aleatoriamente formando uma fila de execução.