PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 10/09

Requisitos Funcionais para um Midleware Paralelo e Distribuído de Sistemas Multi-Agentes Auto-Organizáveis

Allan Alves Valeriano
Paulo Rogério da Motta Jr
Maíra Athanázio de Cerqueira Gatti
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 10/09 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Março, 2009

Requisitos Funcionais para um Middleware Paralelo e Distribuído de Sistemas Multi-Agentes Auto-Organizáveis

Allan Alves Valeriano, Paulo Rogério da Motta Jr, Maíra Athanázio de Cerqueira Gatti, Carlos José Pereira de Lucena

Laboratório de Engenharia de Software – LES
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil

{allvaleriano, pjunior, mgatti, lucena} @inf.puc-rio.br

Abstract. With the technological advance, some research areas turn itself more dependants to advanced computer system. Laboratory simulations become more expensive and take too long to be performed and computer simulations provide a more affordable way to make scientific experiences. Either way, the actual computers are still not so advanced to provide an efficient mechanism for those experiences to be executed in a satisfactory time. On this paper, it is described part of a framework's architecture with the goal to provide an infra-structure to run agent-based simulations on a parallel and/or distributed environment, with total transparency to the system developer**.**

Keywords: Multi-agent system, parallelism, distributed events, virtual space management.

Resumo. Com o avanço da tecnologia, algumas áreas de pesquisa tornam-se cada vez mais dependentes de sistemas de computação avançados. Simulações em laboratórios tornam-se cada vez mais caras e demoradas e simuladores computacionais proporcionam uma maneira menos custosa para que experiências científicas sejam feitas. Mesmo assim, os computadores atuais não se encontram tão avançados, a ponto de prover um mecanismo eficiente, para que estas experimentações sejam realizadas em tempo satisfatório. Neste artigo é apresentada parte de um framework que tem o propósito de proporcionar a sistemas baseados em agentes, uma infra-estrutura de paralelismo e distribuição de modo transparente para o programador.

Palavras-chave: Sistemas multi-agentes, paralelismo, eventos distribuídos, gerenciamento de espaço virtual

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

1 Introdução 1

2 Trabalhos Relacionados 1

3 Arquitetura Vexpa 2

4 Eventos Distribuídos 3

5 Gerenciamento de Espaço Virtual 4

6 Trabalhos Futuros 4

7 Conclusão 5

1 Introdução

Cada vez mais os experimentos científicos estão se tornando dependentes da computação. O computador fornece uma maneira prática, rápida e barata de fazer simulações científicas. Experimentos unindo a computação com a física, a biologia, a geologia, entre outras áreas, tem se tornado cada vez mais freqüentes e importantes. Os simuladores proporcionam um ambiente controlado e seguro para fazer experiências, evitando exposição à substâncias tóxicas, acelerando o processo da experiência ou mesmo evitando a compra de aparelhos e substâncias caras. Hoje em dia é possível simular condições em águas profundas, reações em lugares distantes do universo ou mesmo reações de moléculas invisíveis ao olho humano.

Com esta crescente demanda de simuladores científicos, os sistemas multi-agentes tornam-se ferramentas importantes no desenvolvimento de sistemas de simulação. Sua interface de execução independente de supervisão externa provê uma maneira inteligente de criar sistemas simuladores onde indivíduos relacionam entre si sem estímulos externos.

Com base nesta arquitetura, foi criado o Vexpa. Um framework de suporte a simulações em ambientes paralelos e distribuídos. Este framework foi criado com a intenção de prover uma base de apoio a sistemas que executam em uma arquitetura clusterizada, tendo um gerenciamento central do espaço onde os objetos são alocados. Este framework foi criado usando como base o Mason [1]. O Mason é uma biblioteca básica de simulação de sistemas multi-agentes, feita em Java. Ela serve como base para uma grande extensão de simulações como robótica, simuladores biológicos, *machine learning* de ambientes sociais complexos, entre outros.

Este artigo está estruturado da seguinte forma: seção 2 apresenta alguns trabalhos relacionados. A seção 3 contém detalhes da arquitetura do Vexpa. Na seção 4 são abordados os eventos distribuídos e na seção 5 é explicado como é feito o gerenciamento de espaço 3D. São mostrados alguns trabalhos futuros na seção 6 e por fim, na seção 7, é feita uma conclusão e dadas as referências usadas neste artigo.

2 Trabalhos Relacionados

Como base para o projeto desta implementação, alguns trabalhos se mostraram importantes.

O Octopus [6] é uma plataforma que facilita a construção e execução de aplicações baseadas em agentes móveis. Ele propõe um sistema computacional baseado em agentes com uma arquitetura de duas camadas que dividem claramente as funcionalidades do sistema. A camada superiora é composta pela aplicação a ser executada, enquanto a camada inferior é a plataforma que sustenta os agentes executados. A camada superior é formada por agentes, porém eles próprios são decompostos futuramente em duas subcamadas – uma casca e uma tarefa embutida. Uma tarefa é um problema particular com os dados requeridos para serem computados. Ela deve ser capaz de dividir-se, o que é uma propriedade natural da computação paralela. O agente exterior é o container de uma tarefa, ele requer seu próprio ambiente para providenciar a carga da informação a ser computada pela política de agendamento local e autonomia para tomar decisões de continuar uma tarefa interna ou migrar para encontrar recursos melhores.

A plataforma provê um set de funcionalidades mínimo, requerido claramente por grandes problemas computacionais, porém estes devem estar executando de maneira adequada. Esta plataforma deve estar bem escalonada em termos de utilização de máquinas e é requerida a habilidade de definir topologias virtuais.

O Flexible Large-scale Agent-Based Modelling Environment (FLAME) [7] é um framework que fornece uma abordagem *bottom-up* para modelagem de simulações em qualquer tipo de domínio de modo que após identificar os níveis mais baixos de agentes, as regras que governam sua interação podem ser geradas, resultando em modelos de interações moleculares, sociedades de células, etc. Desta maneira pode-se evoluir dados, estruturas e funções usando conceitos de emergência. Neste framework, as especificações são baseadas em *X-machines* [8] formais, que permitem o uso de analise de máquinas de estado e algoritmos de teste para a verificação e validação. O FLAME, ao invés de ajustar o modelo para adaptar-se às limitações de uma linguagem de programação particular, os sistemas podem ser especificados em termos de suas estruturas e processos constituintes. Assim que um modelo é especificado, o FLAME é capaz de, automaticamente, gerar simulações que incorporam comunicações eficientes entre agentes e pode ser executado em uma variedade de plataformas computacionais paralelas. A compatibilidade com uma grande variedade de arquiteturas é garantida usando o framework MPI (Message Passing Interface) para C, fazendo com que, quando os modelos sejam traduzidos para C, eles aderem a interface MPI. O FLAME, foi concebido para uso em grande escala e é capaz de realizar a simulação de modelos que envolvem milhões de agentes.

3 Arquitetura Vexpa

A arquitetura do Vexpa é caracterizada por uma arquitetura distribuída, centralizada. O controle do sistema é feito por um nó central, que gerencia o espaço virtual, fica responsável por fornecer o identificador único dos objetos instanciados e provê a comunicação entre os nós trabalhadores. Todo objeto criado no ambiente de execução, é marcado por um ID único. Este identificador evita conflitos no sistema, garantindo que os objetos são sempre classificados como diferentes.

Este tipo de arquitetura faz com que seja necessária uma quantidade maior de comunicação no ambiente, porém permite que o crescimento da infra-estrutura possa acontecer com qualquer quantidade de máquinas, ao contrário de uma arquitetura descentralizada, que força o crescimento de 8 em 8 máquinas. O paralelismo do sistema é totalmente transparente para o desenvolvedor da aplicação, que cria suas simulações exatamente como se estivesse implementando um sistema não paralelo/distribuído. O gerenciamento de espaço virtual é fornecido com um serviço, pois ele tem a possibilidade de ser aproveitado por simuladores em espaço 2D ou 3D.

O Vexpa utiliza do ParallelSequence provido pelo Mason, para permitir um paralelismo dentro de cada nó. Isso faz com que os sistemas possam ser paralelos e distribuídos ao mesmo tempo, aproveitando mais de uma infra-estrutura de computadores multi-processados dentro de uma rede clusterizada.

Os nós são interconectados através de uma infra-estrutura, usando RMI [2], para que seja alcançado um ambiente distribuído em um padrão Máster/Worker. O RMI possibilita uma comunicação JAVA distribuída, completamente transparente para o programador.

Cada nó possui um *broker*, usado para fazer a comunicação entre os nós distribuídos e os serviços locais. O *broker* é responsável por fazer todo o trabalho de comunicação na rede, independente de quem seja o receptor. Cada vez que um nó trabalhador precisa comunicar-se com outro nó, o *broker* envia a requisição para o *broker* do nó central, que a repassa para o receptor, no nó correspondente.

O controle do paralelismo é feito pelo Sequential Object Monitor (SOM) [3]. O SOM é uma alternativa fácil e pratica para trabalhar com monitores de Java. Ele promove uma separação do negócio, o desacoplando da lógica da aplicação.

4 Eventos Distribuídos

Na arquitetura do Vexpa, a simulação tem base para acontecer em um ambiente paralelizado ou não. Um grande desafio na implementação do framework foi a interceptação de eventos distribuídos. Os agentes podem mandar eventos diretamente para outros, sendo que o receptor pode não estar no mesmo espaço físico que o remetente. Os eventos também são guardados pelas posições no espaço virtual, o que faz com que seja necessário que o nó controlador consiga reconhecer os eventos guardados em um raio, a partir da posição do remetente. Como o sistema é controlado por um nó central, todas as comunicações de eventos são feitas através deste nó mestre.

Cada ponto no espaço virtual, seja ele 2D ou 3D, guarda uma referência para todos os eventos criados naquela posição. No exemplo das formigas, quando uma formiga passa por um lugar, ela libera feromônios, estes são deixados como estímulo para outras formigas encontrarem o caminho para chegar em algum lugar onde haja comida. Estes feromônios são modelados como eventos no Vexpa, e são guardados na posição do espaço virtual, para que, quando outros agentes passem por este ponto, possam consumi-lo. Estes eventos guardados na posição do espaço possuem um tempo de vida. Isto evita que eventos fiquem pairando no espaço virtual eternamente. A cada *step* realizado pelo agendador do Mason, o tempo de vida de um evento diminui. O evento também pode ser retirado do espaço virtual por um agente. Cabe a cada evento saber se ele, ao ser consumido, será eliminado ou não.

Cada vez que um agente entra em uma posição, ele executa o executa o método *updateEvents*. Este método faz com que ele consuma todos os eventos da posição atual. Outras possibilidades de consumir eventos também estão disponíveis, onde o agente os consome em um raio ao redor de si. Neste caso entra a estrutura de distribuição do Vexpa.

Como o espaço virtual é um elemento distribuído, a posição diretamente adjacente à ocupada por um agente, pode estar em outra máquina física. Para fazer a comunicação entre um agente e os eventos que não estão fisicamente na mesma máquina, o *broker* local (estação de trabalho) intercepta o pedido de consumo de um evento e verifica se a posição diretamente adjacente está mapeada na máquina onde ele se encontra. Caso ela não esteja no mesmo espaço físico, o *broker* local repassa a requisição para o nó controlador central, onde o broker *master* recepta o pedido e o repassa para o gerenciador de espaço virtual.

No nó controlador central, existe um mapeamento de todo o espaço virtual e onde cada posição se encontra fisicamente (qual nó trabalhador ela está fisicamente sendo alocada). Para repassar a requisição, este mapa é consultado e descobre-se onde a posição requisitada está fisicamente. É repassada então, a requisição, para o nó traba-

lhador respectivo, que possui os eventos a serem consumidos. Logo após é feito todo o caminho reverso, para que o evento possa ser entregue ao agente requisitor.

5 Gerenciamento de Espaço Virtual

O Gerenciamento do Espaço Virtual é feito através de um nó controlador central. Este gerenciamento é fornecido como um serviço para que se tenha uma flexibilidade maior em relação ao framework. Desta maneira é possível usá-lo em simulações de espaço 2D (simulação de formigas) ou simulações em espaço 3D, como é o caso do Biomabs [4].

O nó controlador mantém sempre uma referência para os agentes instanciados no espaço. Durante a criação destes objetos, é pedido um ID para o simulador. Este pedido é interceptado pelo *broker* local (nó trabalhador), que o repassa para o *broker* da controladora central. A requisição então segue para o gerenciador de espaço virtual, que cria um ID único para aquele agente, em todo o espaço virtual, permitindo o paralelismo da solução.

Um exemplo para o cálculo de gerenciamento de espaço virtual é a proliferação de células tronco em um corpo embrióide. Este cálculo é feito da seguinte maneira:

- É varrido o espaço virtual (seja ele 2D ou 3D), em todas as direções, da posição do objeto pai até a o tamanho total do espaço, procurando por posições livres.
- Pega-se uma posição aleatória dentre as livres encontradas
- É verificada se a posição possui vizinhos ocupados em um raio de cR, onde cR é o tamanho do raio do objeto.
- É verificado se existem objetos entre o objeto pai e a posição escolhida
- Caso existam objetos entre o objeto pai e a posição livre escolhida, estes são empurrados para fora e o elemento novo criado é alocado na posição exatamente adjacente ao pai.

Assim que terminar o mapeamento, o nó controlador central devolve para o objeto criado, seu ID único e seu posicionamento no espaço, via *broker*.

O nó controlador central só guarda referências para o tipo Object. Cada sistema usuário do Vexpa faz sua manipulação específica para a criação dos objetos e passa para o nó controlador apenas um objeto genérico.

Caso a execução não seja paralela/distribuída, a única modificação será que a comunicação não será via broker, não havendo interceptação do pedido de requisição de um novo ID, com o cálculo da posição no espaço virtual, pois tudo será feito na mesma máquina.

6 Trabalhos Futuros

Como futuros desafios, o projeto terá os seguintes trabalhos:

1) Avaliar a solução proposta durante esta etapa do desenvolvimento através de testes de desempenho, para ambos os espaços (2D e 3D). Desta maneira pode-se ter uma estatística das melhorias propostas, se foram realmente relevantes.

2) Para diferentes estratégias de propagação de eventos – que são essenciais para auto-organização [referencia os padrões], avaliar a solução proposta levando em conta os padrões existentes.

3) Tomando como base os resultados alcançados, melhorar a solução para alcançar uma simulação mais robusta e eficiente.

4) Aferir a ferramenta de verificação e otimização do comportamento emergente deve se adequar ao ambiente distribuído.

Avaliar uma integração do Vexpa ao Jadex ou outros frameworks BDI, para prover auto-organização paralela e distribuída com agentes inteligentes, por conta de o Vexpa, por enquanto, ser voltado para agentes reativos/pró-ativos com comportamentos simples.

7 Conclusão

O Vexpa fornece uma infra-estrutura prática e simples para o paralelismo em simulações com agentes. Ele permite que um sistema seja desenvolvido com transparência completa, por parte do programador, com respeito ao paralelismo e à distribuição. Sua arquitetura faz com que um sistema possa ser executado tanto em um ambiente clusterizado, como em uma única máquina, sem que muitas modificações sejam necessárias.

O gerenciamento de espaço virtual centralizado providencia a base para que não hajam conflitos em sistema sendo executado em várias máquinas paralelas. Desta maneira, pode-se ter uma maior performance na simulação, sem riscos de que redundância.

Os eventos distribuídos são responsáveis por prover uma base para que os agentes possam comunicar entre si, independente do fato de estarem em máquinas separadas ou no mesmo espaço físico. Por fim é feita uma otimização, através de mapeamento, para que seja minimizada a quantidade de tráfego desnecessário na rede, prezando a performance máxima da execução.

Referências Bibliográficas

[1] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan. MASON: A New Multi-Agent Simulation Toolkit. Proceedings of the Eighth Annual Swarm Users/Researchers Conference (SwarmFest 2004).

[2] Remote Method Invocation.
http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp, 2008

[3] D. Caromel, L. Mateu, E. Tanter. Sequential Object Monitors. In Proceedings of the 18th European Conference on Object-Oriented Programming (ECOOP 20004).

[4] Faustino, G.M. ; GATTI, M. A. C. ; Bispo, D. ; Lucena, C.J.P. de . A 3D Multi-Scale Agent-based Stem Cell Self-Organization. In: SEAS 2008 - Fourth Workshop on Software Engineering for Agent-Oriented Systems, 2008, Capinas. XXV SBES - Simpósio Brasileiro de Engenharia de Software, 2008.

[5] P. Uhruski, M. Grochowski, R. Schaefer, Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial, ISSN 1137-3601, Nº. 28, 2005 (Ejemplar dedi-cado a: Nuevas tendencias en Sistemas Multiagente y Soft Computing) , pags. 55-62.

[6] M. Kiran, S. Coakley, N. Wakinshaw, P. McMinn, M. Holcombe. International Conference "Collective Dynamics: Topics on Competition and Cooperation in the Biosciences - BIOCOMP2007.

[7] M. Holcombe and F. Ipate, Correct Systems: Building a Business Process Solution. Springer-Verlag 1998.