FAct: Um *Framework* para a Construção de Sistemas Multiatores

Allan D. S. Lima Patricia R. Tedesco Geber L. Ramalho

Universidade Federal de Pernambuco, Centro de Informática, Brasil

Keywords: synthetic agents, multi-agent systems, simulators, multiactors systems

Authors' contact:
{adsl, pcart, glr}@cin.ufpe.br

Resumo

Atualmente, existem diversos *frameworks* para a construção Sistemas Multiagentes (SMAs), possibilitando ao desenvolvedor definir apenas o comportamento dos agentes, não se preocupando com questões de infra-estrutura destes sistemas. Entretanto, estes projetos têm se mostrado bem menos aplicáveis para construção de sistemas baseados em atores sintéticos, tanto que no projeto SmartSim optou-se não utilizar nenhum destes para a implementação dos seus atores. Visando preencher esta lacuna, o FAct (*Framework for Actors*) foi desenvolvido como um framework *open source* para construção de Sistemas Multiatores. Desta fora, o presente pôster tem como objetivo relatar a experiência no desenvolvimento FAct além do estado atual da sua implementação.

1. Introdução

Há diversos projetos e grupos de pesquisa que trabalham na modelagem de atores sintéticos, agentes inteligentes credíveis, que apresentam características de personalidade humana [Silva 2000]. Estes projetos acabam gerando uma demanda por sistemas que possam auxiliar na implementação e simulação de seus atores.

Além disso, buscando auxiliar à construção de Sistemas Multiagentes ou SMAs, existem diversas ferramentas [Bittencourt e Osório 2002; Page et al 2000; Bellifemine et al. 1999; Sampaio 2004; Klügl e Puppe 1998]. Porém, durante o desenvolvimento do projeto SmartSim [SmartSim 2006] pôde-se constatar que nenhuma dessas ferramentas é suficientemente aplicável para modelagem de atores sintéticos. Tanto que seus atores sintéticos tiveram que ser implementados sem o suporte de qualquer uma dessas ferramentas.

Visando atender a esta demanda, o FAct (*Framework for Actors*), um *framework* para construção de Sistemas Multiatores ou SMActs, está sendo desenvolvido. Neste contexto, este pôster relata a experiência no desenvolvimento do FAct além do estado atual da sua implementação.

O presente pôster está organizado em cinco seções. A segunda apresenta um estudo comparativo entre cinco projetos para construção de SMAs; a terceira define o conceito de SMActs, destacando algumas diferenças entre SMActs e SMAs; a quarta descreve os requisitos do projeto, seus módulos e o estado atual de desenvolvimento do *framework*; finalmente, a quinta apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Durante o processo de desenvolvimento deste trabalho diversos projetos para construção de SMAs foram avaliados sob cinco critérios: (1) Objetivos; (2) Implementação e disponibilidade de código fonte; (3) Documentação do projeto; (4) Ferramentas de suporte ao desenvolvimento; e (5) Quantidade e domínio de suas aplicações. Sob estes critérios cinco projetos se destacaram, merecendo uma análise mais detalhada, são eles o GNU MAGES, JADE, CORMAS, FAMA e o SeSAm.

2.1 Análise

O *GNU Multi Agents Environment Simulator* (GNU MAGES) [Bittencourt e Osório 2002] tem como principal objetivo avaliar a desempenho de diferentes arquiteturas de agentes. Desenvolvido em Java [Deitel e Deitel 2005], ele tem o seu código fonte distribuído livremente sob a *General Public Licence* [GPL 2006]. Entretanto, apenas uma aplicação foi implementada sobre o *framework*, um jogo de guerra chamado *Capture the Flag*. Além disso, este projeto possui como ferramentas um gerenciador de simulação, um editor de *bots*, um editor de times e um editor de torneio.

Criado para simular interações entre um grupo de agentes e um ambiente que guarda recursos naturais, o *Common–pool Resources and Multiagent Systems* (CORMAS) [Page et al 2000] foi implementado em Smalltalk [Abdala e Wangenheim 2002], e seu código fonte é distribuído sob uma licença pública própria. A diversidade de aplicações desenvolvidas sobre o projeto é um dos seus diferenciais e uma conseqüência da sua utilização em diversas instituições de pesquisa ao redor do mundo, para simular desde a dinâmica de uma savana até o equilíbrio em florestas do Mediterrâneo. Dentre suas ferramentas se destacam seu construtor de modelos de simulação além do seu compacto gerenciador de simulação.

O *Java Agent DEvelopment Framework* (JADE) [Bellifemine et al. 1999] foi criado com o intuito de ser

um ambiente para desenvolvimento de aplicações baseadas em agentes inteligentes conforme as especificações da *Foundation for Intelligent Physical Agents* (FIPA) [FIPA 2006]. Implementado em Java, ele é distribuído sob a *Lesser General Public License* [LGPL 2006]. E, como o CORMAS, é utilizado em diversas aplicações (e.g. na construção de jogos multiagentes [Morais e Furtado 2003]). Além disso, sua principal ferramenta é um depurador para o comportamento dos agentes.

O *Framework para Aplicações Multimídia MultiAgentes* (FAMA) [Sampaio 2004] oferece uma biblioteca para aplicações multiagentes além de uma biblioteca musical. Seu código fonte, escrito em C++ [Eckel e Allison 2000], não é distribuído o que acabou por influenciar no número de aplicações do *framework*, tendo em vista que apenas uma foi construída.

Por fim, o SeSAm [Klügl e Puppe 1998] provê um ambiente genérico para modelagem e experimentação de sistemas baseados em agentes. Implementado em Java, este projeto tem o seu código fonte distribuído sob a licença LGPL, assim como o JADE. Um dos seus principais diferenciais é a quantidade de ferramentas disponíveis (e.g. ferramentas para construção de modelos de agentes, e modelagem de animações).

2.2 Discussão

Quatro dos projetos analisados foram criados visando auxiliar a construção de simuladores para tipos genéricos de agentes. A única exceção é o FAMA, que suporta principalmente a construção de simuladores com agentes musicais. O que o torna mais aplicável para o desenvolvimento deste tipo de sistema.

Quanto ao código fonte, quatro projetos foram implementados em linguagens interpretadas (Java e Smalltalk), exceto, novamente, pelo FAMA que foi desenvolvido em C++, por questões de desempenho.

Já em relação à forma de distribuição do código fonte dos projetos, o FAMA é o único que não dá acesso ao seu código enquanto que os demais o distribuem livremente. Este fato parece estar ligado diretamente ao número de aplicações que estes projetos possuem, tendo em vista que três destes projetos (CORMAS, JADE e SeSAm) que distribuem seu código, possuem diversas aplicações implementadas e uma grande comunidade de desenvolvimento. Outro fator que colabora para popularidade destes projetos é a variedade de ferramentas que eles possuem para auxiliar aos desenvolvedores.

Finalmente, dois dos projetos (FAMA e JADE) se mostraram preocupados em seguir os padrões da FIPA. O que pode facilitar consideravelmente a interoperabilidade entre agentes de sistemas distintos.

3. Sistemas Multiatores

Como visto na seção anterior, existe uma grande variedade de projetos para auxiliar na construção de SMAs. Porém, devido as características particulares dos atores sintéticos estes projetos não são aplicáveis para a construção de SMacts. Isto, aliado a grande aplicação de atores sintéticos em diversas áreas da Ciência da Computação, como Jogos eletrônicos [Silva 2000; SmartSim 2006] e *Chatterbots* [Galvão et al 2003], acaba cirando uma grande demanda por projetos que ajudem no desenvolvimento de SMActs.

Entretanto, antes do desenvolvimento de qualquer ferramenta para a construção de SMActs, se faz necessária uma definição precisa para o termo 'Sistema Multiatores'. Esta pode ser realizada com base na definição dada por Ferber [1999] para SMAs. Assim, é possível definir o termo 'Sistema MultiAtores', como aplicável a um sistema com os seguintes elementos:

1. Um ambiente, E, que é um espaço que geralmente tem um volume.
2. Um conjunto de objetos, O. Estes objetos são geograficamente situados e em qualquer dado momento é possível associar qualquer objeto com uma posição em E. Estes objetos podem ser percebidos, criados, destruídos e modificados pelos atores.
3. Um conjunto de atores sintéticos, A, que são objetos específicos (A está contido em O), representando as entidades ativas do sistema.
4. Um conjunto de representações gráficas, RG, para os atores A, tal que cada ator possua pelo menos uma representação.
5. Um conjunto de relações R, que liga objetos (e seus agentes) uns aos outros.
6. Um conjunto de operações, Op, tornando possível para os agentes de A perceber, produzir, consumir, transformar e manipular objetos de O.
7. Operadores com a tarefa de representar a aplicação destas operações e a reação do ambiente a elas.

Uma das principais diferenças entre SMAs e SMActs, vem do fato de que o segundo possui um conjunto de atores sintéticos. Isto faz com que os SMActs devam suportar todas as suas características dos atores como, por exemplo, modelos de personalidade. Outra importante diferença é a existência de um conjunto de representações gráficas, para os atores, dando suporte a mais uma particularidade deste tipo de agente.

4. O *Framework*

Propondo atender a lacuna existente no desenvolvimento de SMActs, foi elaborado o FAct (*Framework for Actors*), um *framework* que provê o suporte necessário ao desenvolvimento dos elementos apresentados na definição da seção anterior.

4.1 Requisitos Elicitados

Os seguintes requisitos foram identificados para a construção do *framework*.

1. Fornecer modelos genéricos e extensíveis tanto para atores sintéticos quanto para suas representações.
2. Controlar processo de simulação dos atores.
3. Controlar o ciclo de vida dos atores bem como o ciclo de vida das suas representações.
4. Prover mecanismos genéricos e extensíveis para controlar o processo de comunicação do atores.
5. Ser distribuído sobre a licença *open source* LGPL, visando a sua posterior modificação e extensão.
6. Ser implementado em C++ visando permitir a sua aplicação em mercados como o de jogos, onde C++ é predominante [Rolling e Morris 2004].

4.2 Módulos do *FAct*

Tendo como base os requisitos do *framework*, um conjunto de módulos foi elicitado para compor o *framework*. Ele visa dar suporte aos desenvolvedores, cada um dos módulos identificados será descrito a seguir.

- **Classes de simulação:** Coleção de classes que dá suporte à implementação de modelos de atores.
- **Simulador:** Conjunto de classes responsável provê toda a infra-estrutura para a simulação.
- **Gerador de Logs:** Visa gravar as mensagens e eventos gerados pela simulação que podem ser utilizados tanto para análise posterior da simulação, como por outras ferramentas.
- **Controlador/Visualizador:** Permitir ao usuário executar, pausar, ajustar a velocidade e visualizar os eventos da simulação.
- **Editor de Atores:** Possibilitar ao usuário do simulador configurar os atributos dos atores que irão interagir no ambiente de simulação.
- **Editor de Regras:** Fornecer uma interface gráfica para a edição das regras de simulação presentes no ambiente.
- **Ferramentas Estatísticas:** Exibir um conjunto de tabelas, gráficos e informações relevantes para o domínio da aplicação, para análise do modelo simulado.

4.3 Arquitetura

Composta por seis pacotes*, (Figura 1)* a arquitetura do FAct foi projetada e implementada não só para atender aos requisitos levantados, mas também para possibilitar futuras extensões. Atualmente ela contempla os módulos classes de simulação, simulador e gerador de logs. Toda via, os demais módulos poderão ser facilmente integrados ao *framework*, visto que o seu estado corrente já prevê tais modificações. A seguir, serão abordadas as funções de cada um destes pacotes.

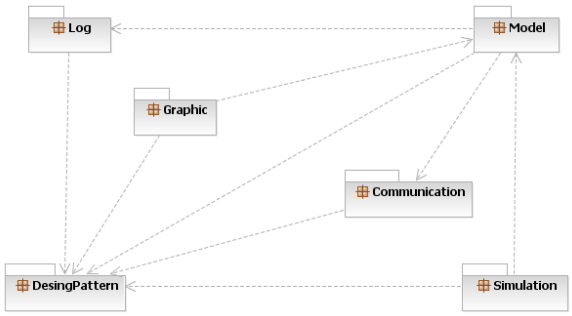

Figura 1: pacotes do *framework* e suas dependências.

Contendo o modelo genérico de ator sintético bem como o modelo genérico de mensagem para a comunicação dos atores, o pacote *Model* atende ao primeiro e ao quarto requisitos do projeto . Além disso, suas classes compõem a implementação das classes de simulação.

Para controlar o processo de comunicação dos agentes presentes no pacote *Model*, foi criado um gerenciador de comunicação que está contido no pacote *Communication*. Este pacote integra as classes de simulação e faz parte da implementação do quarto requisito do *framework*.

Como atores sintéticos devem ser representados graficamente por avatares, um pacote específico foi criado para isto. Chamado de *Graphic,* ele faz parte das classes de simulação e da implementação do terceiro requisito elicitado. Pois, além de possuir uma classe genérica e extensível para representar o avatar de um ator, ele também é responsável pelo controle do ciclo de vida destes avatares.

Uma das mais importantes peças do *framework*, o pacote *Simulation* contém classes que possuem três funções básicas: controlar o ciclo de vida dos atores, divulgar os serviços que cada ator possui e gerenciar o processo de simulação. Ele faz parte da implementação do segundo e do terceiro requisitos, além de compor o módulo simulador.

O gerador de logs do *framework* foi implementado através do pacote Log, que possui as classes responsáveis por representar e controlar as mensagens de Log geradas durante todo o processo de simulação.

Auxiliando os demais, o pacote *DesignPattern* possui a implementação genérica dos padrões de projeto *Singleton* e *Iterator* [Gamma et al 1998].

4.4 Testes

As classes implementadas no *framework* tiveram suas funcionalidades avaliadas através de testes unitários

[Luo et al 1995]. Para tal, a biblioteca de teste unitários *CPPUnit* [CPPUnit 2006] foi empregada. Devido ao grande número de novas classes geradas para os testes, elas foram agrupadas em um novo pacote no projeto que foi chamado de TestCases.

5. Conclusões

Com o crescimento da popularidade dos sistemas compostos por múltiplos atores sintéticos também se faz necessário o desenvolvimento de ferramentas que possam auxiliar na criação deste tipo de aplicação. Ao fazer uso de ferramentas no desenvolvimento destas aplicações é possível poupar um enorme esforço no desenvolvimento do projeto, diminuindo o período necessário para a modelagem e implementação dos atores.

Porém, há uma escassez deste tipo ferramenta disponibilizada gratuitamente e distribuída sobre uma licença *open source*. Visando preencher esta lacuna, foi construído o FAct, para prover facilidades para a construção de SMAct, permitindo que os desenvolvedores destas aplicações tivessem apenas que realizar a modelagem do comportamento dos seus atores.

Com o objetivo de avaliar e refinar o FAct, aplicações multiatores estão sendo desenvolvidas. A primeira aplicação implementada simula interações entre dois atores: um gerente de projetos e um desenvolvedor de software.

Finalmente, na tentativa de realizar mais um esforço de consolidação do *framework* como uma ferramenta de apoio ao desenvolvimento sistemas multiatores, pretende-se preparar o FAct para ser adicionado em um repositório de código aberto como o SourceForge [SourceForge 2006], bem como a criação de uma página para divulgação do projeto.

Referências

LUO, G., PROBERT, R. L. E URAL, H., 1995. *Approach to constructing software unit testing tools*. Department of Computer Science, University of Ottawa, Ottawa.

KLÜGL, F. E PUPPE, F., 1998. *The Multi-Agent Simulation Environment SeSAm*. Dep. for Artificial Intelligence, University of Würzburg.

FERBER, J., 1999. *Multi-agent Systems: Introduction to Distributed Artificial Intelligence,* first edition, Addison-Wesley Professional.

BELLIFEMINE, F., POGGI, A. E RIMASSA, G., 1999. CSELT internal technical report. *JADE – A FIPA-compliant agent framework*. London, April.

PAGE, L. C., BOUSQUET, F., BAKAM, I., BAH, A. E BARON C., 2000. *CORMAS : A multiagent simulation toolkit to model natural and social dynamics at multiplescales.* Workshop "The Ecology of Scales", Wageningen (Pays-Bas).

ECKEL, B. E ALLISON, C., 2000. *Thinking in C++, Volume 1: Introduction to Standard C++,* Second edition, Prentice Hall.

SILVA, D. R. D., 2000. *Atores Sintéticos em Jogos de Aventura Interativos: O Projeto Enigmas no Campus*. Dissertação – Mestrado em Ciências da Computação. CIn-UFPE.

BITTENCOURT, J. R. E OSÓRIO F. S., 2002. *Ambiente para Simulação de Múltiplos Agentes Autônomos, Cooperativos e Competitivos*. Monografia de Conclusão de Curso. UNISINOS, São Leopoldo.

ABDALA, D. D. E WANGENHEIM, VON A., 2002. *Conhecendo o SmallTalk,* Sexta edição, Visual Books.

GALVÃO, A. M., NEVES, A. M. M., BARROS, F. A., 2003. *Persona-AIML: Uma Arquitetura para Desenvolver Chatterbots com Personalidade.* CIn – UFPE.

MORAIS, L. A. S. E FURTADO, J. J. V., 2003. *Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development framework*. Monografia de conclusão de curso. UFCE, Fortaleza.

SAMPAIO P. A 2004. *Framework multiagente Para Aplicações Musicais*. Trabalho de Graduação. CIn-UFPE.

ROLLING A. E MORRIS A., 2004. *Game Architecture and Design: A New Edition*. New Riders Publishing.

DEITEL, H. E DEITEL, P., 2005. *Java Como Programar.* Sexta edição, Prentice Hall,

SMARTSIM, 2006. *Projeto SmartSim – Simulação em gerência de projetos com atores sintéticos.* [online] Disponível em: http://cin.ufpe.br/~smartsim [Acessado 11 de Agosto de 2006].

GPL, 2006. *General Public Licence* [online] Disponível em: http://www.gnu.org/copyleft/gpl.html [Acessado 11 de Agosto de 2006].

LGPL, 2006. *Lesser General Public Licence* [online] Disponível em: http://www.gnu.org/licenses/lgpl.html [Acessado 11 de Agosto de 2006].

FIPA, 2006. *Welcome to the Foundation for Intelligent Physical Agents* [online] http://www.fipa.org [Acessado 31 de Agosto de 2006].

GAMA, E., HELM, R., JOHNSON, R. E VILISSIDES J., 1995. *Design patterns: elements of reusable object-oriented software,* first edition. Addison-Wesley Professional.

CPPUNIT, 2006. *FrontPage - CppUnit Wiki* [online] Disponível em: http://cppunit.sourceforge.net/cppunit-wiki [Acessado 11 de Agosto de 2006].

SOURCERFORGE, 2006. *SourceForge.net: Welcome to SourceForge.net.* [online] Disponível em: http://sourceforge.net/index.php [Acessado 11 de Agosto de 2006].