1 Introdução

"Intelligent Agents and Artificial Intelligence:
Well, it has to do with smart programs,
so let's get on and write some."
Stuart Russell and Peter Norvig

"Em cinco anos, todas as empresas serão empresas da Internet ou simplesmente não serão empresas", presidente do conselho de administração da Intel Andy Grove (Revista Exame, n.16, 11/08/99, p 126). A revolução digital mudou complemente a gestão dos negócios, que no passado se baseava primordialmente em aquisição de bens físicos e átomos, e agora está se baseando em bits e idéias. As palavras chaves desta nova administração são informação, conhecimento e inteligência ao invés de prédios, fábricas, ou bens materiais.

Surgem a partir dessa revolução inúmeras oportunidades na área da tecnologia da informação. O comércio eletrônico é um exemplo de tema muito importante no mundo dos negócios relacionados com a Internet. De acordo com Peter Drucker (Drucker, 2000), o comércio eletrônico é para a Revolução da Informação o que a ferrovia foi para a Revolução Industrial.

As comunidades de Inteligência Artificial, *Business Intelligence*, Comércio Eletrônico, e Sistemas Multi-Agente estão contribuindo com o desenvolvimento desta área, e boa parte das tecnologias oferecidas já está sendo usada pelo mundo corporativo (IBM Business Intelligence, 2005; Computer Associates CleverPath, 2005; Dash Optimization, 2005; Living Systems & Whitestein Technologies, 2005). O objetivo principal destas contribuições é automatizar processos e maximizar a lucratividade das transações em meios eletrônicos.

1.1. Sistemas Multi-Agente

A tecnologia de agentes (Ferber, 1999; Wooldridge, 2000) permite criar simuladores e sistemas inteligentes que tomam decisões automaticamente. Muitos acreditam que a principal proposta desta tecnologia é bem semelhante a da inteligência artificial: programas capazes de substituir o ser humano em tarefas que exigem conhecimento, experiência e racionalidade. Esses programas podem ser utilizados para simulações ou em situações reais.

A base deste enfoque está em considerar os agentes como entidades inteligentes e isoladas. Porém, uma nova teoria que utiliza os agentes de software acredita que a inteligência não pode ser separada do contexto social (Ferber, 1999). Desta forma, começa a surgir um novo campo de interesse que estuda programas construídos com vários agentes – os Sistemas Multi-Agente.

Esses agentes dotados de autonomia podem executar tarefas simples ou complexas, só que agora inseridos dentro de uma organização e um meio ambiente. O importante nessa análise não é observar os objetivos de cada agente, e sim o objetivo global da sociedade formada. O agente utiliza algumas características encontradas nas sociedades para alcançar esse objetivo.

A primeira delas é Interação (Ferber, 1999). A interação ocorre quando dois ou mais agentes são submetidos a um relacionamento dinâmico através de um conjunto de ações recíprocas. A segunda é a Cooperação (Ferber, 1999), que é vista dentro de um sistema multi-agente como um método para atingir o objetivo global. Além disso, temos também as Organizações (Ferber, 1999), que são criadas para estruturar a cooperação e interação entre os agentes. Uma organização pode ser vista como um conjunto de relações entre agentes que produz uma unidade, com qualidades não encontradas nos agentes individualmente.

O desenvolvimento de um sistema multi-agente de larga escala (Lucena et al., 2003; Garcia et al., 2003) não é uma tarefa simples. Sistemas com muitos agentes em ambientes heterogêneos necessitam de soluções de engenharia de software para permitir reuso e um desenvolvimento eficiente.

Uma técnica interessante para modelar um problema computacionalmente difícil com o paradigma de agentes é dividir esse mesmo problema em subproblemas mais simples (Sardinha et al., 2004a). Assim, cada agente se torna

responsável por um ou mais desses subproblemas. Esses agentes ainda podem resolver esses subproblemas de forma assíncrona e em diferentes CPUs. Conseqüentemente, podemos listar algumas vantagens que surgem a partir dessa técnica de modelagem:

Escalabilidade – Problemas complexos são resolvidos de forma mais simples, e não há limite para o número de agentes no sistema;

Versatilidade – Como cada agente ataca um subproblema, podemos desenvolver agentes com técnicas diferentes de inteligência e combinar as soluções com simplicidade;

Flexibilidade – É simples evoluir o sistema para incluir novos agentes, e assim, incluir novas técnicas de inteligência.

1.2. A Engenharia de Software para Agentes Inteligentes

Muitas metodologias e linguagens de modelagem para agentes já foram propostas para construir sistemas multi-agente. Existem também muitas plataformas e *frameworks* de desenvolvimento para implementar os sistemas baseados em agentes. Porém, pouquíssimos trabalhos propõem um mapeamento entre modelos de *design* baseado em agentes e código. Esta tese propõe um mapeamento claro entre conceitos de agentes e código, e ainda apresenta várias aplicações complexas para demonstrar a robustez do método.

Metodologias como Gaia (Wooldridge et al., 2000; Zambonelli et al., 2003) e MaSE (Deloach, 1999), não fornecem um guia de como implementar os modelos propostos. Conseqüentemente, a geração de código não é um processo trivial. Gaia é uma metodologia orientada a agentes para as fases de análise e *design*. Os autores de Gaia afirmam que a metodologia não trata diretamente dos detalhes de implementação. Em MaSE, os autores afirmam que o foco principal da metodologia é auxiliar as fases de requisitos, análise, *design*, e implementação. Porém, a metodologia não descreve como os modelos devem ser implementados em qualquer plataforma.

Em (Do et al., 2003; Castro et al, 2001), os autores propõem um mapeamento de conceitos i*, usados na metodologia Tropos (Giunchiglia et al., 2002; Bresciani et al., 2004) para modelar um agente BDI (*Beliefs, Desires, and Intentions*), para um ambiente de desenvolvimento chamado Jack (Howden et al.,

2001). Cada conceito i* é mapeado para um BDI, e consecutivamente para uma abstração Jack. Apesar dos autores fornecerem um mapeamento, nenhum exemplo de aplicação é apresentado. Conseqüentemente, o mapeamento de conceitos i* para abstrações Jack não possui qualquer prova concreta. Os detalhes de como implementar os planos não é claro, o que evidencia uma pesquisa ainda em uma fase inicial.

A metodologia Prometheus (Padgham et al., 2002a) também propõe um suporte do "inicio-ao-fim" entre especificação e implementação. Os autores propõem o uso do ambiente Jack para implementar os modelos de Prometheus. Em (Padgham et al., 2002b), o *Jack Development Environment* (JDE) é apresentado como uma ferramenta para modelar detalhes de implementação dos modelos Prometheus e implementar sistemas baseados em agentes usando Jack. Porém, o mapeamento entre os artefatos detalhados de *design* e abstrações de implementação não é apresentado. Além disso, um exemplo não é apresentado no ambiente Jack.

Em (Huget, 2002), os autores demonstram a geração de código a partir de diagramas de seqüência AUML. O foco dos autores é exemplificar o mapeamento de interações de agentes para código Java. Entretanto, esse mapeamento não é baseado em nenhuma arquitetura ou *framework* de agentes. Essas tecnologias fornecem abstrações reutilizáveis para programadores implementarem sistemas baseados em agentes. Ao fazer uma extensão ou customização a esses *frameworks*, o processo de implementação se torna mais simples e mais rápido. Nos *frameworks* de implementação há muito código já escrito e compilado, e o re-uso é grande.

1.3.Aprendizado para Agentes Inteligentes

O processo de modelagem de um sistema multi-agente inteligente de larga escala não é uma tarefa simples. Principalmente quando o sistema deve atuar em ambientes onde há mudanças constantes nos cenários externos. Neste caso, o sistema deve possuir agentes dotados de aprendizado e inteligência. Os conceitos de aprendizado devem ser modelados já nas primeiras fases de desenvolvimento. Em sistemas complexos e abertos, agentes precisam de aprendizado para tomar

decisões e se adaptar a mudanças para poder atingir os seus objetivos e o objetivo global do sistema.

A modelagem de sistemas baseada em agentes inteligentes com aprendizado sempre apresenta questões semelhantes, dentre as quais destacamos:

(i) Como avaliar o objetivo do sistema como um todo?

(ii) Como definir e avaliar o objetivo individual de cada agente?

(iii) Como modelar o conhecimento de cada agente?

(iv) Como projetar o mecanismo de aquisição de conhecimento para cada agente?

(v) Como combinar múltiplas técnicas de aprendizado e distribuir essas técnicas para cada agente no sistema?

(vi) Como associar abstrações de agentes com abstrações de *machine learning*?

(vii) Como especificar abstrações de *machine learning* nas primeiras fases de *design* e permitir uma transição para a fase de implementação?

Infelizmente, os engenheiros de software ainda utilizam a sua experiência e intuição para resolver as questões acima. Muita pesquisa tem sido feita para criar metodologias e *frameworks* de implementação para sistemas multi-agente. Porém, nenhum desses trabalhos apresenta um guia para incluir técnicas de aprendizado já nas fases iniciais do desenvolvimento.

Os *frameworks* (Howden et al., 2001; Telecom Itália-Jade, 2003) de implementação disponibilizam muitas APIs para desenvolver sistemas multi-agente, mas não orientam a estruturação do *design* do aprendizado de uma maneira sistemática. Além disso, as várias metodologias orientadas a agentes (Zambonelli et al., 2003; Deloach, 1999; Bresciani et al., 2004; Padgham et al., 2002) são focadas em um nível muito alto de abstração, e não indicam como tratar aprendizado desde a fase de *design* até a implementação.

Um tema de pesquisa importante na área de aprendizado em sistemas multi-agente é denominado *MAS Learning* (Sem et al., 2000; Kudenko, 2003). O foco principal dos trabalhos é desenvolver algoritmos de *machine learning* distribuídos utilizando técnicas como *Reinforcement Learning* (Mitchell, 1997). Porém, os trabalhos desta área não apresentam um *design* ou metodologia, geral o suficiente

para qualquer algoritmo ou domínio, que permite incluir essas técnicas de *machine learning* em sistemas baseados em agentes.

1.4. A Solução Proposta

Esta tese apresenta o método MAS-School (Sardinha et al., 2004b; Sardinha et al., 2005b) para modelar e implementar agentes de software inteligentes desde as primeiras fases de desenvolvimento. Esse método também apresenta várias orientações de como incluir aprendizado na fase de *design* e implementação. O método apresenta no final uma estratégia incremental e iterativa de desenvolvimento para permitir a avaliação das técnicas de *machine learning*.

A linguagem de modelagem ANote (Choren, 2002; Choren et al., 2004a; Choren et al., 2004b) é utilizada para modelar os sistemas multi-agente desta tese, pois fornece todos os diagramas necessários para as várias fases de modelagem.

Todos os nossos projetos de software utilizam a técnica orientada a objetivos para a fase de requisitos. Esta fase utiliza o processo recursivo de decomposição de um objetivo do problema principal em vários objetivos de subproblemas. Essa técnica é interessante, pois permite dividir um problema computacionalmente difícil em subproblemas mais simples. O diagrama 1 exemplifica a decomposição.

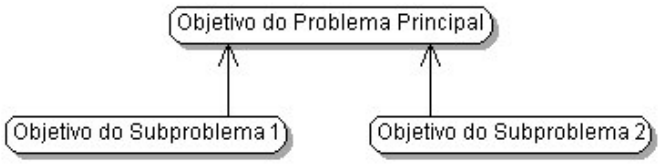

Figura 1: A decomposição dos objetivos relacionados com o objetivo do Problema Principal.

A abstração de objetivos é central neste método. Em uma segunda etapa, esse objetivo permite a definição de uma medida de performance para os agentes que necessitam de aprendizado. Conseqüentemente, o processo de decomposição dos objetivos em sub-objetivos é fundamental para o método proposto. É importante frisar que essa modelagem de hierarquia de objetivos é uma atividade comum nas metodologias orientada a agentes. Para esse método, os objetivos

funcionam como uma abstração para unificar conceitos de *machine learning* com conceitos básicos de sistemas multi-agente.

A segunda fase do processo de modelagem é criar os tipos de agentes que sejam capazes de conduzir os objetivos desses subproblemas encontrados na fase de requisitos. Essa criação de tipos de agentes começa com um processo de relacionamento de todos os objetivos de nível mais baixo com os tipos de agentes criados. A tabela 1 exemplifica um mapeamento entre esses objetivos e tipos de agentes. A figura 2 apresenta os vários tipos de agentes e seus relacionamentos. Esta figura permite uma visão estática do projeto do nosso sistema multi-agente.

Objetivo	**Agente**
Objetivo do Subproblema 1	Agente I
Objetivo do Subproblema 2	Agente II

Tabela 1: A mapeamento entre subproblemas e tipos de agentes.

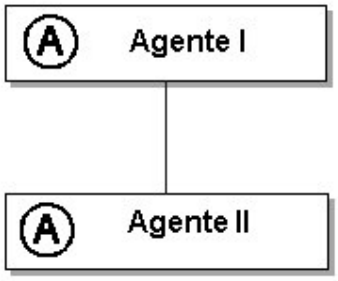

Figura 2: A arquitetura do Sistema Baseado em Agentes.

Após a criação dos vários tipos de agentes, cada agente deve possuir uma especificação de cenários. Esses cenários possuem informações sobre o contexto de como cada agente deverá atingir seus objetivos. Esses objetivos possuem ações que o agente deve seguir para alcançar os objetivos dos subproblemas, e conseqüentemente o objetivo do problema principal. O diagrama 3 apresenta o digrama de cenário do Agente I.

Cenário I	
Agente Principal	Agente I
Pré-Condições	Evento do ambiente externo
Plano Principal das Ações	Enquanto VERDADE Ação 1 Ação 2 Envia Mensagem para Agente II Fim Enquanto
Interação	Agente II
Variante do Plano	

Figura 3: Cenário do Agente I.

Um diagrama interessante que deriva dos diagramas de cenário é o diagrama de interação. Esse diagrama deixa explícita a comunicação entre os agentes, e permite uma visão dinâmica do sistema. O diagrama 4 apresenta a interação entre o Agent I e Agente II

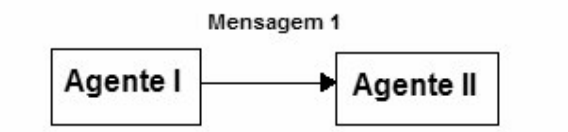

Figura 4: Diagrama de Interação do Agente I e Agente II.

Conceitos de Agentes	**Classes do ASYNC instanciadas**
O ambiente e os recursos	(i) Classe Principal para o ambiente; (ii) Classes para os recursos.
Agente	(i) Uma classe concreta que herda de *Agent*, e implementa *AgentInterface*; (ii) Uma classe concreta que herda de *InteractionProtocols*.
Ações internas do Agente	Métodos incluídos na classe concreta de *Agent*.
Protocolos de Interação	Métodos incluídos na classe concreta de *InteractionProtocols*.
Formato das Mensagens nos protocolos de interação	(i) Classe concreta que implementa *AgentMessage*, se a troca de mensagens é síncrona; (ii) Classe concreta que implementa *AgentBlackBoardInfo*, se a troca de mensagens é assíncrona.

Tabela 2: Mapeamento dos Conceitos de Agentes para as Classes OO.

O capítulo 3 apresenta um *framework* chamado ASYNC (Sardinha et al., 2003a), e um mapeamento das abstrações de agentes para abstrações OO (Sardinha et al., 2005c). A tabela 2 resume esse mapeamento, porém o detalhe de instanciação do *framework* será apresentado mais adiante nesta tese. Esse *framework* é fundamental para o método proposto, pois:

(i) Permite um mapeamento claro entre abstrações de agentes e código OO;
(ii) Permite re-uso de código;
(iii) Diminui o tempo de desenvolvimento;
(iv) Diminui a complexidade de implementar agentes de software.

Em conseqüência do uso do *framework* proposto, cada agente no sistema deve possuir duas classes concretas. A figura 5 exemplifica essas classes. As ações internas do agente, que não dependem da interação com outros agentes no sistema, são mapeados como métodos da classe *Agente I*. Os protocolos de interação, que definem como o agente deve se comunicar e interagir com outros agentes no sistema, são mapeados como métodos da classe *Agent I IP*.

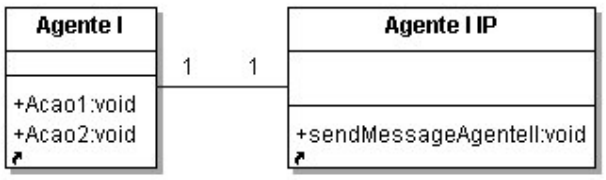

Figura 5: O diagrama de classes do Agente I.

Esse *framework* também possui uma camada de comunicação síncrona através de troca de mensagens, e uma comunicação assíncrona através do uso de um *tuple space* reflexivo e distribuído (Silva et al., 2001). Essa camada de comunicação fornece serviços básicos (ler, escrever, e retirar), e pode ser usado para implementar um *blackboard* associativo (Corkill et al., 1987). O uso dessa infra-estrutura de comunicação é fundamental para a nossa arquitetura de implementação. Os agentes devem executar as suas ações de forma assíncrona, e

compartilhar resultados através de uma base de conhecimento corporativa dentro do *tuple space*. Essa estratégia é utilizada também pelo A-Teams (Talukdar et al., 1997), um sistema que utiliza memória distribuída para resolver problemas de forma descentralizada. A figura 6 ilustra a arquitetura de implementação.

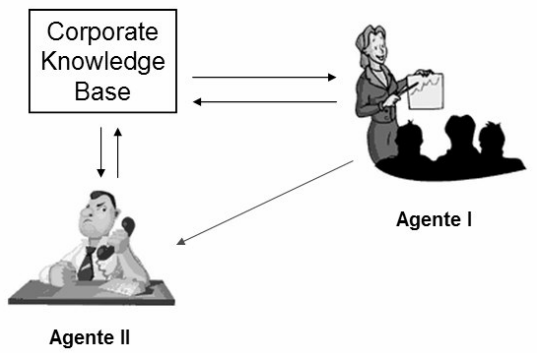

Figura 6: A arquitetura de implementação do Sistema Multi-Agente.

A arquitetura proposta permite uma alta escalabilidade para o sistema, pois não há limite para o número de agentes. Os resultados de ações dos agentes podem ser compartilhados nessa base de conhecimento corporativa, e acessado por qualquer outro agente no sistema, mesmo que este esteja fisicamente em outra CPU. A arquitetura também fornece uma grande versatilidade, pois cada agente ataca um objetivo do subproblema com técnicas diferentes de inteligência e aprendizado. A combinação dos resultados dos subproblemas é feita com simplicidade através dessa base de conhecimento corporativa. O sistema é simples de evoluir, pois novos agentes com técnicas de inteligência podem ser inseridos a qualquer momento. Isso fornece grande flexibilidade ao engenheiro do sistema.

Porém, ao invés de implementar o sistema multi-agente com todos os agentes inteligentes em uma única fase, o nosso método propõe uma primeira implementação com apenas agentes reativos (Ferber, 1999) sem nenhuma técnica de *machine learning* implementada. Essa primeira versão é importante para testar a comunicação entre os agentes e a interação com o ambiente externo.

A próxima etapa do método permite incluir disciplinadamente as técnicas de *machine learning* em sistemas multi-agente. Esse método também permite em sua fase de implementação, a integração de diferentes algoritmos de *machine learning* e a avaliação da performance do sistema. O método é apresentado no capítulo 6 e possui quatro fases distintas:

(i) *Objetivo Sistêmico & Seleção da Medida de Performance*, onde o objetivo do problema principal é associado a uma medida de performance para o sistema. Essa medida de performance permite mensurar se o objetivo sistêmico está sendo alcançado;

(ii) *Seleção do Agente & Definição do Objetivo do Aprendizado no Agente*, onde agentes com planos complexos são selecionados, e objetivos são atribuídos para o algoritmo de aprendizado;

(iii) *Design do Aprendizado no Agente*, onde o *refactoring* do *design* do agente é definido; e

(iv) *Implementação Incremental & Avaliação de Performance*, onde uma implementação incremental é proposta com treinamento, teste e avaliação.

O processo incremental e iterativo de implementação da fase 4 permite verificar o quanto o objetivo do problema principal do sistema esta sendo alcançado através da medida de performance selecionada na fase 1. Conseqüentemente, o engenheiro de software pode projetar técnicas de *machine learning* que maximizem essa medida de performance.

1.5. Avaliação Empírica

Esta tese apresenta várias aplicações desenvolvidas com agentes inteligentes. O método e *framework* proposto foram concebidos através de um desenvolvimento *bottom-up*. A primeira aplicação (Sardinha, 2001; Milidiú et al., 2001) apresenta um sistema baseado em agentes para criar promoções em um mercado varejista utilizando o conceito de agregação de produtos. A aplicação apresenta agentes que utilizam técnicas de *clustering* e algoritmos evolutivos para determinar "cestas" de produtos altamente lucrativos.

A segunda aplicação (Ribeiro, 2001; Sardinha et al., 2003a) apresenta um mercado virtual para compra e venda de bens. Os agentes no sistema também implementam uma negociação automática entre compradores e vendedores. Além disso, o sistema implementa um mecanismo de certificação dos vendedores, compradores e bens para garantir a segurança das transações.

As próximas duas aplicações implementadas utilizam integralmente o processo de generalização que concebeu o *framework* e o método propostos nessa

tese. As aplicações também foram extremamente importantes para garantir a robustez do processo desenvolvido, e testar a escalabilidade do método.

A terceira aplicação desenvolvida é o *LearnAgents* (Sardinha et al., 2004c; Sardinha et al., 2005a), um sistema multi-agente distribuído para um complexo cenário de *procurement* em leilões simultâneos e interdependentes. Os agentes cooperam para atingir um único objetivo que é maximizar a diferença entre a utilidade marginal dos pacotes de bens e os custos de obtenção dos bens nos leilões. Na arquitetura proposta, cada agente ataca um subproblema do objetivo global, e utilizam várias técnicas de inteligência e aprendizado para resolver esses subproblemas.

Além disso, o Trading Agent Competition (TAC) foi escolhido como ambiente para testar a arquitetura. Esse ambiente apresenta características de um mercado competitivo e permite avaliar a performance do *LearnAgents* em relação a outros sistemas para comércio eletrônico. O *LearnAgents* conseguiu o terceiro lugar na competição de 2004, e permitiu confirmar a robustez do método e *framework* desta tese.

A quarta aplicação é um sistema multi-agente para um *Supply Chain Management*. O ambiente do TAC também foi utilizado para avaliar a performance desse sistema. Nesse ambiente, os sistemas participantes competem por clientes e por peças de um número limitado de fornecedores. Os participantes também são limitados pela capacidade da fábrica, e tem que gerenciar todos os processos de uma cadeia de suprimentos.

1.6.Organização da Tese

O Capítulo 2 apresenta as principais tecnologias de engenharia de software e um resumo dos algoritmos utilizados para construir os sistemas multi-agente desta tese. Essas tecnologias são fundamentais para o desenvolvimento de sistemas multi-agente de larga escala.

O capítulo 3 apresenta o *framework* orientado a objetos ASYNC para facilitar e agilizar a implementação de agentes inteligentes em ambientes distribuídos. Este capítulo utiliza o estudo de caso *Bundles.com*, uma aplicação para determinar “cestas” de produtos altamente lucrativos, para exemplificar a utilização do *framework*.

O capítulo 4 apresenta o método MAS-School para modelar e implementar agentes de software inteligentes desde as primeiras fases de desenvolvimento. O capítulo 5 apresenta o *LearnAgents*, um sistema multi-agente distribuído para um complexo cenário de *procurement* em leilões simultâneos e interdependentes. O capítulo 6 apresenta o *LearnAgentsSCM*, um sistema multi-agente distribuído para um *Supply Chain Management*. O capítulo 7 apresenta as conclusões e trabalhos futuros.