# 2
# Conceitos Básicos

A engenharia de software baseada em agentes é uma área emergente cujo objetivo é oferecer suporte ao desenvolvimento de sistemas multi-agentes (Garcia et al., 2003; Jennings & Wooldridge, 2000; Jennings, 1999; Garcia, 2004). Tal engenharia de software tem sido proposta em acréscimo à engenharia de software orientada a objeto como um meio de dominar a complexidade associada ao desenvolvimento de sistemas distribuídos em larga escala (Silva et al., 2003).

Um agente pode ser visto como uma extensão de um objeto (Garcia, 2004). Objetos e agentes fornecem serviços a seus clientes. No entanto, os objetos são entidades não-autônomas que representam elementos de sistemas passivos. Já um agente é uma entidade interativa, adaptativa, pró-ativa, re-ativa e autônoma que age em um ambiente e manipula objetos (Garcia, 2004). Somente sistemas interativos, adaptativos e autônomos são considerados agentes (Garcia, 2004).

Um agente é composto de conhecimento e de um conjunto de propriedades, chamadas propriedades de agente. As propriedades de agente são características comportamentais que podem ser incorporadas a um agente.

A autonomia é a característica que um agente tem de não depender de outra classe ou agente. Para isto é necessário que os agentes instanciem seus objetivos. Eventos externos podem causar a instanciação de objetivos re-ativos, enquanto os eventos internos podem originar a criação de objetivos pró-ativos (Garcia, 2004).

O comportamento de interação consiste em receber e enviar mensagens para outros agentes. Este comportamento possibilita ao agente uma comunicação com o ambiente externo. Os *frameworks* de construção de agentes geralmente trazem uma família de mensagens pré-definidas. Este é um dos pontos que estimulam o uso de um *framework* de construção de agentes.

A adaptatividade é a característica que os agentes têm de modificar seu comportamento e conhecimento de acordo com estímulos internos e externos. A pró-atividade é a característica que um agente tem de agir ativamente, sem depender de um estímulo externo ou do sistema. A re-atividade é a característica

que um agente tem de reagir após um estímulo vindo do sistema ou do usuário (Garcia, 2004).

Num sistema multi-agente cada agente de software é definido segundo um objetivo de maneira que cada agente tenha um caminho a seguir dentro de um sistema. O objetivo de um agente pode ser visto como uma composição de planos e ações onde os planos são compostos de várias ações e as ações, por sua vez, podem ser compostas de mensagens e necessariamente executam funções para que o objetivo do agente seja consumado.

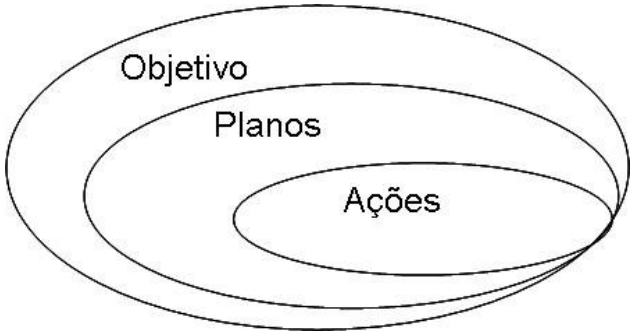


**Figura 1.** Estrutura do agente de software.

Os objetivos dos agentes estão sempre caminhando numa mesma direção, a fim de realizar o objetivo do sistema.

A arquitetura de um sistema multi-agente pode naturalmente ser vista como uma sociedade organizada de indivíduos (Zambonelli et al., 2001).



## 2.1. Agentes e Sistemas Multi-Agentes

Um sistema multi-agente (SMA) é composto por um conjunto de entidades. Essas entidades incorporam diferentes tipos de agentes e objetos que são inseridos em ambientes (Garcia, 2004).

Um SMA pode ser considerado um sistema orientado a objetos, desde que tais objetos sejam associados a outras propriedades específicas de agentes. Se os objetos pudessem adquirir essas propriedades de forma flexível, os agentes poderiam ser construídos explorando ferramentas e técnicas orientadas a objetos. Há várias formas de construir agentes de software a partir de objetos: uma delas seria definindo interfaces e comportamentos comuns em superclasses abstratas, envolvendo objetos com comportamento do agente usando técnicas de

composição etc. No entanto, essas formas apresentam problemas para transformar os objetos existentes em agentes e para desenvolver agentes a partir de um esboço, em especial caso sejam necessárias várias propriedades de agente (Garcia, 2004).

Os agentes de software são construídos a partir das propriedades já definidas na seção anterior, de modo que os *frameworks* que utilizam a abstração de agentes trazem embutidos mecanismos que possibilitam a reprodução de tais propriedades. Por isso não é observada dificuldade na construção de agentes como visto na construção a partir de objetos.

Os SMAs, como qualquer outro sistema, têm um objetivo a cumprir. Para que se cumpra o objetivo do SMA é necessário que os agentes tenham objetivos bem definidos e que o cumprimento destes objetivos, mesmo que de maneira isolada, impliquem o cumprimento do objetivo central.

Os agentes de um SMA não interagem todo o tempo de execução de um sistema, porém em grande parte dos SMAs os agentes exercem certa influência, uns sobre os outros. A influência que um agente exerce sobre o outro pode significar uma mudança de decisão. Um bom exemplo deste tipo de influência pode ser visto na própria sociedade humana, onde uma pessoa pode ter influência nas decisões de outras.

A figura 2 mostra uma sociedade de agentes e a influência que cada um sofre. Como numa sociedade humana, uns sofrem mais influência que outros, a figura mostra uma sociedade de agentes de um sistema que tem o objetivo de auxiliar o médico no desenvolvimento de uma prescrição medicamentosa (o sistema será explicado com mais detalhes nas seções subseqüentes).

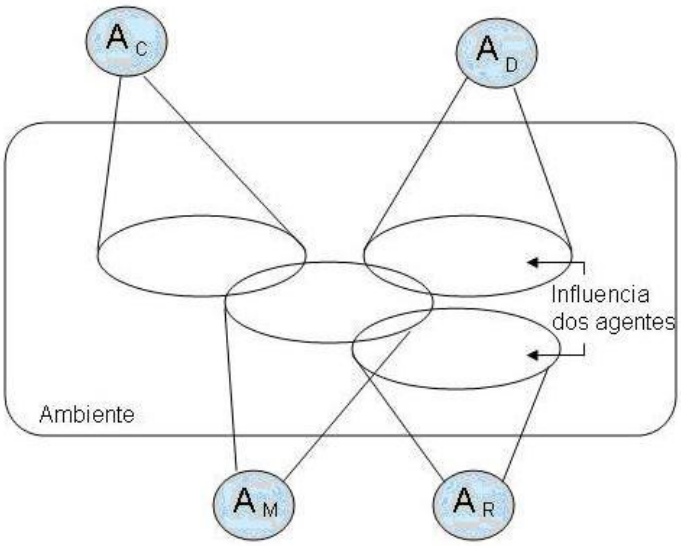


**Figura 2.** Interação entre os agentes de um SMA.

A comunicação entre os agentes de um SMA se dá através da troca de mensagens.

## 2.2. Variabilidade e Framework

Em orientação a objeto existem inúmeros frameworks e arquiteturas de linhas de produto. Tanto na indústria como na academia, tais conceitos são bastante consolidados em orientação a objeto.

A tão almejada reutilização é uma característica que pode ser alcançada através da utilização de componentes. Por isto, a definição de framework pode ser vista como uma coleção de componentes parcialmente implementados e pré-definidos com um padrão de cooperação entre eles (Fontoura, M. et al., 2002).

Um framework define uma arquitetura para aplicações com características similares e então, tal arquitetura é utilizada por meio de especialização através do código especifico da aplicação. Com isto grande parte da estrutura da aplicação é padronizada, o que possibilita uma significativa redução no tamanho e na complexidade do código que será implementado com a adaptação do framework. Alguns dos componentes do framework são designados para serem substituídos e estes componentes não denominados de hot-spots ou pontos de variação.

Os pontos de variação de um framework definem a variabilidade que tal framework possui. Isto é um framework só existe se existem duas ou mais

aplicações que utilizem à mesma base arquiteturada pelo framework, assim as aplicações que podem ser instanciadas mostram a variabilidade que um framework tem, ou seja, um framework só é construído se existe variabilidade.

A idéia de *framework* traz uma grande redução nos esforços por parte do desenvolvedor, além de embutir um alto grau de qualidade ao software (Fayad, 2000).

A idéia de uma arquitetura de linha de produto se adequa perfeitamente a abordagem de *framework*. Um dos pontos atraentes é que a linha de produto tem sido bastante utilizada na indústria, onde são relatados bastantes casos de sucesso no desenvolvimento de softwares de qualidade e ganho de produtividade (Bosch, 1998; Schmitt, 2000; Durscki, 2004). A arquitetura de linha de produto vem sendo conhecida como o caminho para projetar famílias de softwares relacionados (Schmitt, 2000) trazendo a noção de um sistema integrado que é composto por um conjunto de subsistemas que são acoplados de acordo com a necessidade (Bosch, 1998; Schmitt, 2000; Durscki, 2004). Um dos problemas com relação à utilização desta abordagem é que o desenvolvimento de software e a manutenção de seus produtos baseados na arquitetura de linha de produto requerem um bom conhecimento dos conceitos de linha de produto e uma boa estrutura para reutilização (Bosch, 1998; Durscki, 2004).



### 2.2.1. Representação de Frameworks

Nesta seção será discutida uma forma de representação de *framework* em orientação a objeto. A linguagem de modelagem utilizada como padrão em orientação a objeto é a *Unified Modeling Language* (UML). Tal linguagem serve para muitos propósitos, porém não fornece uma forma apropriada de construir modelos de *framework* e por isso a representação de *framework* em orientação a objeto pode ser projetada com uma extensão da UML denominada UML-F.

Na seção anterior foi visto os pontos de variação que podem existir em um *framework*. A UML-F visa representar de maneira clara os pontos de variação em um *framework* tais pontos podem surgir por meio de métodos, classes e numa modelagem tradicional com UML são apenas métodos ou classes.

Para tanto a UML-F traz uma extensão do meta-modelo da UML e por isto faz uso de alguns elementos já existentes na linguagem UML com uma semântica mais abrangente para o escopo de *framework*. A UML-F utiliza tags para determinar os pontos de variabilidade de um *framework*, dois bons exemplos são a tag {incomplete} que determina que uma classe com esta marca está pronta para receber tipos de ponto variação e que este serão feito através de generalização e as novas subclasses a variabilidade de instâncias do *framework* e a tag {appl-class} que determina que uma classe com esta marca esta pronta para ser adicionada ao *framework* (Fontoura, M. et al., 2002).

É extremamente alta a flexibilidade obtida através do uso de *framework* e muitas aplicações podem ser construídas desta forma na arquitetura de linha de produto, porém a documentação é um dos pontos cruciais para reutilização de *framework* (Schmitt, 2000).

Problemas semelhantes devem ocorrer em agentes de software, porém em agentes de software o conceito de framework e linha de produto ainda não estão tão disseminados como em orientação a objeto.



### 2.2.2. Mecanismos de Instanciação

Em OO existem diversos mecanismos para descrição e documentação de instâncias. Dentro deste escopo foram escolhidas técnicas mais difundidas como os *cookbooks e hooks* e uma linguagem mais recente denominada RDL (Oliveira et al., 2006). Com isto foi feito um estudo comparativo a fim de determinar uma base para a criação de um mecanismo de instanciação em agentes de software.

O *cookbook* é um tutorial estritamente baseado em linguagem natural onde seus leitores aprendem como instanciar uma aplicação. Este tutorial traz uma descrição com o objetivo geral do *framework*, e seus principais componentes além das partes interessantes do design (Mathias Filho, 2002). Os *cookbooks* tradicionalmente mostram exemplos de como utilizar os seus principais componentes. As seções do *cookbook* são chamadas de receitas onde são descritos os passos para a resolução de problemas específicos.

Um ponto que dificulta a automatização do processo de instanciação é o fato de que o *cookbook* é não-estruturado e baseado em linguagem natural, este é

também um ponto fraco no que diz respeito à escolha de uma abordagem de descrição e documentação.

*Hook* é considerado uma extensão de *cookbook* e trazem uma definição de pontos abertos numa dada localização de um *framework* (Normark, 1994), ou seja, os *hooks* oferecem a possibilidade de executar um ou mais fragmentos do programa em um local pré-definido no interior do programa. *Hooks* provem uma documentação base onde são descritas as possíveis mudanças que podem ocorrer no *framework* e as intenções deste.

Cada descrição de *hook* documenta um problema e um requisito (geralmente um requisito pequeno) para a construção do *framework*, antecipando o desenvolvimento da aplicação e assim provendo um guia para se obter todo o potencial do *hook* (Froehlich et al., 1998).

Diferente dos *cookbooks*, os *hooks* trazem um formato que ajuda a descrever e organizar as informações de modo que possam ser facilmente utilizadas por uma ferramenta, além disso, os *hooks* expõem apenas os detalhes necessários para solucionar um problema no *framework*, o que o torna fácil e simples de entender.

Apesar de todas as facilidades o *hook* é apenas uma ferramenta de documentação e não traz consigo ferramentas para automatizar a utilização dos *hooks*.

A linguagem RDL foi desenvolvida com o intuito de especificar os processos de instanciação de um *framework* orientado a objeto e introduzir a abordagem sistemática de re-uso. A utilização da linguagem possibilita aos desenvolvedores de *framework* a possibilidade de especificar a seqüência de ações as quais devem ser executadas como restrições que envolvem a instanciação de re-uso de propriedade, como um *framework* (Oliveira et al., 2006).

O uso da RDL possibilitará que o *framework* seja guiado através de um conjunto de ações de re-uso trazendo o foco para as partes relevantes do *framework* as quais deverão ser adaptadas durante os processos de instanciação.

Uma das grandes vantagens da RDL é que ela traz consigo a abordagem xFIT que tem por objetivo facilitar o processo de instanciação de um *framework* através de uma ferramenta que possui três tipos de documento como entrada, intitulados de *Features Model*, *Annotated Framework Design* e *Instantiation Script*. Com este tripé é possível obter uma visualização em alto nível de detalhes do *framework* disponibilizado pelo *Features Model*; através do *Annotated*

*Framework Design* é possível obter uma representação do *framework* com o uso da extensão da UML que se complementado com o RDL *Instantiation Script* o qual traz a especificação do processo de instanciação do *framework*. Com isto é possível se obter um guia o desenvolvimento de um *framework*, assim possibilitando que seus pontos de extensão possam incorporar aplicações específicas (Oliveira et al., 2006).

A RDL traz por si só uma linguagem mais complexa e coesa no que diz respeito à interação modelagem-código representando um caminho mais preciso para representar as atividades de instanciação, desde que não seja utilizada linguagem natural para representação (Oliveira et al., 2006).