4 Agentes de Software

4.1. Preliminares

Popularmente, os agentes são confundidos com programas "antropomorfizados", isto é, assemelhado à forma humana de resolver problemas. Tal não se confirma. Há um equívoco fundamental: comparar um software ao modo humano de resolução de problemas é, por um lado, reducionista quando se imagina que o humano atua de um modo tão previsível e algorítmico e, por outro lado pretensioso, quando se julga que o computador seja capaz de escrever um programa que pode criar e escolher caminhos autoconscientemente e por própria vontade.

Os agentes podem fazer muito, mas não fazem tudo pelas pessoas. Um agente é capaz de buscar informações na rede, regular agendas, negociar intenções simples, etc; por isso a sua grande popularidade. Contudo, o desenvolvimento de agentes requer um alto grau de complexidade e de conhecimento. Um projeto pode durar anos em desenvolvimento e não produzir resultados satisfatórios. Pela característica de autonomia relativa, um agente pode produzir erros e as conseqüências são, a princípio, imprevisíveis.

O que muda em relação a um software dito convencional é que um agente possui autonomia, é capaz de comunicar-se com outros agentes e com usuários humanos, reage a mudanças no seu ambiente e age sempre buscando atingir uma meta.

O paradigma mais usual de desenvolvimento de agentes é conhecido como modelo BDI – *Beliefs, Desires and Intentions Model*. Um agente, segundo este modelo possui crenças, desejos e intenções e opera sobre uma coleção de regras que define o seu mundo. Inspecionando esse seu mundo por tentativas de validação de suas regras, tudo se passa como se o agente, ao encontrar uma mudança, passe a "acreditar" que algo é verdade e, em reação a essa mudança, passe a "desejar" fazer algo segundo uma "intenção" projetada. Do ponto de vista físico, um agente é um programa de

computador, escrito em uma linguagem, e que possui uma coleção de regras a verificar e ações a serem tomadas segundo uma lógica de execução.

A idéia fundamental na concepção de um programa agente é a delegação. Delega-se ao software alguma autonomia de decisão. Um agente recebe de seu proprietário ou usuário uma ou mais tarefas a resolver, monitora o seu ambiente de execução e escolhe uma ou mais ações compatíveis com seu estado de mundo e as executa.

Dependendo do modo como foi projetado um agente pode funcionar "*stand-alone*" isolado na máquina do usuário ou em uma federação de agentes, neste caso o sistema é denominado "multiagente".

Quando um agente dispõe de algum grau de inteligência artificial para sua execução ele é denominado agente inteligente de software. Caso contrário, é dito agente de software. A diferença fundamental é que um agente de software opera de forma absolutamente previsível em reação às mudanças ambientais. Já o agente inteligente é capaz de raciocinar sobre os elementos percebidos e escolher a melhor forma de ação segundo as circunstâncias externas a ele.

Os agentes inteligentes surgiram na década de 90 como uma evolução da programação orientada a objetos. (Shoham, 1993) desenvolveu uma linguagem, a Agent-0 e desta surgiu a RADL – *Reticular Agent Development Language*, um ambiente de desenvolvimento de agentes inteligentes muito utilizado.

O conceito de agentes surgiu na comunidade de Inteligência Artificial (Wooldridge & Jennings, 1995) e apenas recentemente ganhou maturidade, quando pesquisadores de Engenharia de Software estabeleceram os sistemas multiagentes de larga escala (Lucena & col., 2003).

Neste capítulo dar-se-á uma breve noção da tecnologia de agentes e sua utilidade como paradigma de programação de agentes de interface.

4.2. Definição

Ainda não há um consenso sobre uma definição ideal para agentes. Também há discordância sobre a que campo ou domínio da informática pertence um agente. Pode-se entender vagamente que os agentes derivam da aplicação das técnicas da Inteligência Artificial no auxílio a um usuário na realização de uma tarefa especifica assistindo a este que detém a capacidade de execução principal da mesma e conta com o auxilio do agente na realização do trabalho. A outra corrente é estabelecida pela comunidade de Engenharia de Software. (Lucena & col., 2003).

(Russell & Nörvig, 1995) descrevem um *agente* como qualquer coisa que seja capaz de **perceber** o seu ambiente através dos seus **sensores** e **agir** sobre o mesmo com seus "**efetores**", em analogia a um agente humano que tem olhos, ouvidos e outros órgãos como sensores e mãos, braços, pernas, etc. como "efetores". Estes autores ainda caracterizam um *agente racional* como um agente que faz a coisa certa.

Algumas propriedades básicas podem ser encontradas nos agentes.

1. Agentes podem operar sem controle direto de humanos ou de outros agentes. São **autônomos.**
2. Agentes podem agir em sociedade com humanos e com outros agentes, eles se **comunicam.**
3. Agentes podem reagir a várias formas de estimulação dos domínios, eles são **reativos.**
4. Agentes podem por si mesmo tomar decisões para ajustar-se a metas definidas, eles são **pró-ativos.**

Agentes transformam o próprio ambiente, portanto é necessário considerar os estados do ambiente e os estados "mentais" do agente para entender o funcionamento do mesmo. Por serem relativamente autônomos, se torna relevante considerar a habilidade do agente em tomar decisões, no sentido de *quando* e *como* tomar decisões. Aí está a essência do agente e sua principal característica advinda da Inteligência Artificial.

Segundo (Shoham, 1993), os agentes diferem de objetos convencionais de software por que possuem estados mentais básicos e têm atos de fala nas suas mensagens (isto é,

há mensagens que simplesmente informam e há mensagens que solicitam informação). Pode-se entender, sem perda de generalidade, que as ações de um agente se dão conforme o ponto no tempo e conforme o estado mental atual. Basicamente os agentes transitam entre estados mentais diversos em pontos diversos no tempo. A seleção correta do estado mental, e a ação correta a ser realizada pelo agente se dá segundo essa dinâmica.

Tipicamente os agentes são entendidos como um modelo baseado em três atributos: crença, desejo e intenção. As ações de um agente são determinadas por suas *decisões* ou *escolhas*. As decisões são derivadas das *crenças* do agente. As crenças referem-se ao estado do mundo (passado, presente ou futuro), ao estado mental deste e de outros agentes e às *capacidades* dos agentes em interação. As decisões são dependentes de decisões anteriores.

(Shoham, 1993) ainda propõe uma nova visão derivada desses modelos. Ele mantém as propriedades da *crença* e decisão *(ou escolha)* e introduz como terceira categoria a *capacidade*. Apresenta ainda a noção de *obrigação* ou *comprometimento* e caracteriza a decisão por um agente como um comprometimento consigo mesmo.

Os agentes operam através de ações baseadas em crenças e capacidades, comprometendo-se a realizá-las imediatamente ou num tempo futuro. Jamais um agente é capaz de comprometer outro a realizar uma ação. Um agente informa ao outro e este decide comprometer-se ou não, conforme seu próprio estado mental, o estado do seu mundo e suas próprias crenças.

A cooperação entre agentes é não supervisionada, isto é, não existe um agente supervisor. Toda a ação deve atender às propriedades de autonomia, reatividade, pró-atividade e é originada pela comunicação entre eles e usuários humanos.

Dada a capacidade de comunicação e acesso a múltiplas bases de conhecimento, o domínio da Inteligência Artificial Distribuída se mostra um "*folding*" ideal para alocar a teoria dos agentes.

4.3. Uma taxonomia para agentes

Aqui é apresentada uma taxonomia de agentes proposta por Russell & Nörvig, (1995) com a finalidade de ilustrar as diferentes composições e complexidades que um agente pode assumir.

4.3.1. Agentes de reflexo simples

Esse é o tipo mais simples de agente. Um simples sistema de regras do tipo condição-ação é a base da tomada de decisão. Os sensores percebem o mundo, o agente, a partir dessa percepção usa essas regras e escolhe a ação mais ajustada para o estado do mundo e determina aos "efetores" a sua realização. A simplicidade desses agentes restringe a sua aplicabilidade. Por outro lado é muito fácil implementar esse tipo de agente, pois atuam em ambiente pouco abrangente e só realizam ações quando é possível escolher a ação certa, baseado na simples percepção do mundo.

Sua principal característica é a reatividade. Tais agentes se assemelham aos primeiros sistemas especialistas, diferindo basicamente por requererem baixa interatividade com o usuário e possuírem relativa autonomia.

4.3.2. Agentes que rastreiam o mundo

Um pouco mais complexo do que o de agente de simples reflexo, este modelo requer uma representação do mundo em estados possíveis. Isto requer que a abstração de mundo seja feita de modo que as mudanças sejam passíveis de serem finitamente representadas. A capacidade de analisar como o mundo evolui (para quais estados o mundo pode mudar) e a capacidade de "julgar" o que uma ação pode produzir de mudança no estado do mundo dá ao agente a funcionalidade de prever estados possíveis a alcançar. Com base nesta arquitetura o agente constrói uma crença acerca de qual a melhor ação a efetuar.

4.3.3. Agentes baseados em metas

Ter uma meta a alcançar não é suficiente para um bom desempenho de um agente. O processo para alcançar a meta depende da eficácia do agente em percorrer os passos de

sua execução. As técnicas básicas de IA para projetos de agentes que visam alcançar metas são a busca e o planejamento.

Um agente baseado em metas requer uma estrutura que divide o processo de perceber o mundo em duas fases. A representação de mundo em estados possíveis é a mesma utilizada para agentes de simples reflexo. O processo de "raciocínio" do agente implementa a capacidade de perceber como o mundo evolui e da capacidade de "avaliar" o que as ações do agente pode influenciar na evolução. A percepção se dá em dois passos: a) como está o mundo agora e b) como ficará o mundo se o agente executar a ação A.

O estado atual é relevante para o conhecimento do mundo presente. As capacidades de perceber como o mundo evolui e de avaliar o que acontece com o mundo se uma ação for executada interferem e modificam as situações nos passos *a* e *b*. Após essa etapa de análise o agente escolhe e determina qual ação executar. Para esse tipo de agente, a capacidade de análise implementa uma relativa autonomia que acrescida à reatividade e à pró-atividade dos modelos anteriormente descritos.

4.3.4. Agentes baseados na utilidade

A autonomia relativa pode gerar soluções de baixa qualidade. Há que se projetar um mecanismo que possa dar qualidade à seleção de metas. Existindo mais de um modo para se alcançar uma meta, torna-se necessário o "raciocínio" para decidir segundo uma melhor forma. O que um agente faz para selecionar a melhor ação quando está diante de metas em conflito, segundo Russell & Nörvig (1995), é constituída por quatro etapas: a) identificar como o mundo está agora; b) como o mundo evoluirá se for executada uma determinada ação; c) o quanto de sucesso se pode admitir (utilidade) se for escolhido um determinado estado do mundo e; d) que ação o agente pode executar agora. Nesta representação não se utilizam as regras de condição-ação como base fundamental para a escolha de ações. A utilidade é aqui implementada como um processo de julgamento e escolha de ações, e o processo de percepção do mundo se dá como no item 4.3.3 acima.

Este tipo de estrutura de agente é de fato de extrema complexidade e exigirá certamente amplo domínio da tecnologia de IA aos mais variados aspectos a ela correspondentes. A escolha de uma boa representação de todo os tipos de

conhecimentos envolvidos e a seleção dos processos de manipulação desse conhecimento é igualmente relevante. Uma estrutura sem dúvida nobre, de amplo espectro de aplicação, mas de difícil implementação.

4.4. A complexidade do processo de construção de um agente

Uma aplicação que projetada com base na tecnologia de agentes requer a definição da arquitetura global do processamento, do mecanismo de raciocínio do agente (inferenciação) e da tecnologia de "*pattern matching*". Deve possuir uma representação interna do conhecimento e dos dados e, ainda, ter definidos os protocolos de comunicação agente-a-agente e os formatos das mensagens. Se se torna necessária ao agente a capacidade de aprendizagem de máquina, então a complexidade fica ampliada. Agentes mais sofisticados requerem a capacidade de planejamento. No caso mais extremo, uma aplicação pode requerer comunicação entre múltiplos agentes. Para esse caso será preciso implementar um protocolo de comunicação robusto entre eles.

Assim, diferentemente das aplicações ditas convencionais, os agentes requerem do projetista habilidade específica de um engenheiro do conhecimento.

Além das habilidades exigidas para o registro, o processamento e o controle das informações, um agente requer modelagem comportamental. Isso é suficiente para mudar o cenário de projeto. Todo agente terá suas capacidades testadas continuadamente, requerendo do projetista um acompanhamento de longa duração. Um agente não se transforma num produto da noite para o dia, nem no mesmo tempo de uma aplicação dita convencional. A depuração é de difícil controle, a dinâmica do domínio da aplicação também pode sofrer alterações que influenciam diretamente na definição das capacidades do agente.

A solução para enfrentar essa complexidade reside no uso de uma ferramenta capaz de integrar funcionalidades convencionais e de IA. Para cada domínio de aplicação de um agente encontrar-se-á diferentes combinações de ações, comportamentos e crenças a selecionar e implementar. Considere-se ainda a necessidade de integração com outras aplicações e acesso a bases distribuídas de informações que podem alcançar

abrangência mundial. Cada caso deve ser tratado com um caso e a aplicação deve estar continuamente sendo acompanhada a fim de que se revisem as capacidades, se aprenda com os erros e acertos, e considerar o projeto com status de pioneiro, sujeito a evoluções de difícil previsão.

Deve-se, portanto, cogitar para uma tal aplicação o uso de ferramentas que garantam facilidade de manutenção, portabilidade, modularidade e parcimônia entre outras características de qualidade.

4.5. Ferramentas para construção de agentes

Existem várias opções de ambientes para construção de agentes, cada um com suas características e desvantagens. Neste trabalho serão focalizados três ambientes: o AgentBuilder, da Reticular Systems; os Aglets da IBM e o Zeus.

4.5.1. O modelo mentalístico de Shoham e a RADL da Reticular

(Shoham, 1993) descreve duas modalidades primitivas de estado mental: *crença* e *comprometimento*. O comprometimento é tratado como uma decisão para agir ao invés de buscar atingir uma meta. A crença é característica persistente que o agente possui de "acreditar piamente" em algo e isso só se modifica se ele aprender um fato contraditório. Isto quer dizer também que se um agente não acredita num fato num certo tempo (em oposição à crença na negação do fato) a única razão para ele vir a acreditar nesse fato é aprendê-lo.

Na linguagem AGENT-0 projetada por Shoham, estão definidos dois tipos diferentes de ação: *ações privativas* e *ações de comunicação.* As ações privativas são o método primário que os agentes usam para completar as tarefas que afetam seu ambiente. Um comportamento complexo de um agente possa é quase sempre resultado de operações simples.

Agentes recebem e enviam mensagens, realizam ações privativas e atualizam seus próprios modelos mentais de modo autônomo só dependendo do seu próprio programa. Um programa em AGENT-0 consiste de crenças iniciais, comprometimentos iniciais e das regras de comprometimento. As crenças e comprometimentos iniciais estão presentes no estado mental ao inicio da execução.

Os comprometimentos iniciais são instanciados no momento da sua execução, mas eles podem mudar ao longo do tempo de execução. As capacidades que definem as ações que o agente pode executar são fixas durante todo o tempo de vida do agente e constituem o seu *design*.

Tais capacidades não são modificáveis pelo agente e isso dá a condição de autonomia restrita. Somente quando o programador interferir no código reprogramando as capacidades é que o agente poderá contar com as novas capacidades implementadas. Já as regras de comprometimento definem as ações realizáveis e as mudanças mentais do agente em todas as situações.

Thomas desenvolveu a linguagem de comunicação PLACA (PLAnning Communicating Agents), uma linguagem de programação similar à AGENT-0 com recursos de planejamento (Thomas, 1993). A Reticular, pesquisou e estendeu o trabalho de Shoham e Thomas desenvolvendo uma nova linguagem de programação de agentes. Utilizando os conceitos de linguagem orientada a objetos essa empresa desenvolveu a Linguagem de Definição de Agentes da Reticular RADL (*Reticular Agent Definition Language*) e também o AgentBuilder um ambiente que disponibiliza ferramentas gráficas para criar programas em RADL que rodam no *runtime* da própria empresa.

No modelo da RADL, as crenças são parte fundamental do modelo mental do agente. Elas representam para o agente a percepção do estado corrente do seu mundo externo e interno e são atualizadas à medida que se obtém novas informações sobre ele.

A integridade do modelo mental foi aprimorada dado que as instâncias de crença e padrões variáveis são sempre de tipos conhecidos, de modo que a máquina de inferência do agente pode garantir que a integridade das comparações entre os elementos dos modelos mentais.

Os modelos de crença permitem que a máquina de inferência focalize suas atividades de busca enquanto procura por crenças que combinem com as condições específicas de uma regra. .

Na RADL um comprometimento é um acordo, usualmente comunicado a outro agente, para realizar uma ação particular a um certo e determinado tempo. Quando o tempo corrente for igual ao tempo comprometido, a ação prevista e combinada será executada ou descartada, se as condições do mundo (crenças) estiverem favoráveis ou desfavoráveis. Um agente não pode tentar executar uma ação se as precondições para isso falham, mesmo se o agente tiver a capacidade de fazê-lo.

No modelo de Shoham, todas as ações são executadas como resultados de comprometimentos. A Reticular estendeu a idéia de regra de comprometimento para incluir uma *regra comportamental* geral. As regras comportamentais determinam o curso de ação que o agente toma em todo o curso da execução.

As regras comportamentais podem ser vistas como declarações do tipo QUANDO-SE-ENTÃO. A parte do QUANDO comporta os novos eventos que ocorrem no ambiente do agente e inclui as novas mensagens recebidas de outros agentes. A parte SE compara o modelo mental corrente com as condições exigíveis para a regra ser aplicável. Os padrões na parte SE combinam com crenças, comprometimentos, capacidades e intenções. A parte ENTÃO define as ações do agente e as mudanças mentais realizadas em resposta ao evento corrente, modelo mental e ambiente externo. Isso pode incluir a atualização do modelo mental; a execução de ações comunicativas e/ou a execução de ações privativas.

Uma regra comportamental pode ter qualquer combinação de ações e mudanças mentais.

O formato de uma regra comportamental em RADL é como abaixo:

NOME {da regra}
QUANDO
    Condições do mundo
SE
    Condições mentais
ENTÃO
    Ações privativas
    Mudanças mentais
    Ações de mensagem ao mundo externo ao agente.

Uma *intenção* é um acordo, usualmente comunicado a outro agente, para ser levado a efeito em um certo estado do mundo do agente, em um certo tempo. As intenções são similares aos comprometimentos que um agente realiza em benefício de outro.

Entretanto, um comprometimento é um acordo para realizar uma única ação enquanto que uma intenção é um comprometimento para realizar quaisquer ações necessárias para alcançar um estado do mundo desejado.

Intenções são de implementação complexa. O agente deve ser capaz construir planos, monitorar o sucesso das ações realizadas e construir planos alternativos quando o plano original falha.

A linguagem RADL possui um interpretador de regras comportamentais e um módulo de "runtime" para sua execução. O interpretador de Agente da Reticular é bastante robusto e permite que um agente definido em seu ambiente possa ser executado de forma correta e eficiente.

A Figura 4.1 ilustra a arquitetura de agente da Reticular. Nela, um interpretador monitora continuamente as mensagens que chegam, atualiza o modelo mental e executa as ações apropriadas.

Inicialmente o agente é instanciado com as crenças, comprometimentos e intenções iniciais, e são carregadas as capacidades e as regras comportamentais. Um agente não-trivial requer pelo menos uma regra comportamental, os demais elementos são opcionais.

Os modelos mentais contêm as crenças correntes, comprometimentos, intenções, capacidades e regras do agente. Embora as regras e as capacidades sejam estáticas, as crenças, comprometimentos e intenções do agente são dinâmicas e podem mudar à medida do tempo de "vida" do agente.

O ciclo de execução do agente consiste em: processar novas mensagens; determinar quais as regras que são aplicáveis à situação corrente; executar as ações especificadas por essas regras; atualizar o modelo mental de acordo com essas regras e planejar.

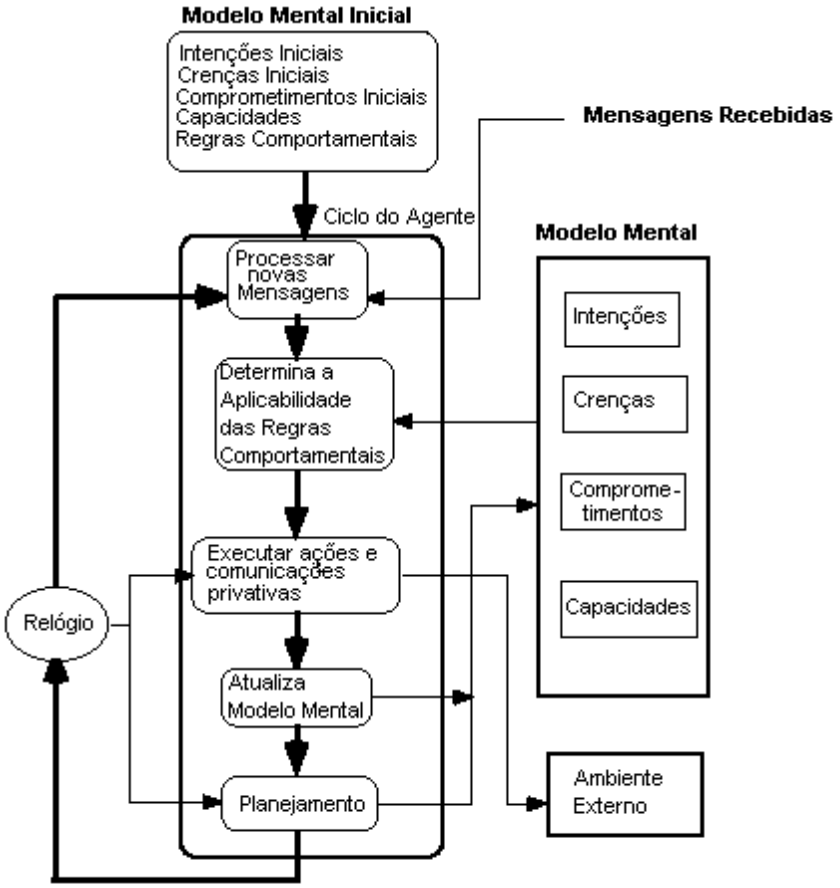

Figura 4.1 Ciclo de Execução do Agente

4.5.2. Os Aglets da IBM

Foram criados pelo Laboratório de Pesquisas da IBM do Japão. Seu desenvolvimento teve início em 1995, tornando-se um dos primeiros sistemas de agentes móveis baseados em Java. Seu modelo de agente é baseado no *applet* o que deu origem ao termo Aglet, que é uma fusão das palavras *agent* e *applet*. Seus executáveis sempre estiveram disponíveis na forma *freeware*, todavia, apenas recentemente, seu código tornou-se aberto. A versão *free* distribuída pela IBM é denominada Aglets Software Development Kit.

Aglets são objetos Java que, entre outras características possui, a capacidade de se mover de uma máquina para outra em uma rede levando consigo o código de programa e o estado dos objetos que o compõem. Uma migração se dá a partir de uma interrupção da execução do *aglet* na máquina origem, em seguida se dá o despacho para uma máquina remota e então se espera o reinicio da execução após sua chegada ao destino.

A biblioteca de classes de aglets foi concebida por pesquisadores da IBM, de modo a atender aos seguintes objetivos:

1. Fornecer um modelo compreensivo e simples de programação utilizando agentes móveis, sem, no entanto implicar em modificações na máquina virtual Java ou em código nativo;
2. Disponibilizar mecanismos de comunicação poderosos e dinâmicos que permitissem agentes se comunicarem outros agentes, fossem eles conhecidos ou não;
3. Projetar uma arquitetura de agentes móveis que permita extensibilidade e reusabilidade;
4. Obter uma arquitetura altamente coerente com o modelo tecnológico *Web*/Java.

Por concepção, os aglets não dispõem de *runtime* próprio. O que parece a princípio pouco vantajoso é na verdade uma grande vantagem, pois ao utilizar o *runtime* do próprio Java, os aglets podem rodar em todo ambiente onde esteja disponível esse recurso. Também se mostra ideal para projeto de agentes que utilizam plataformas distintas e o protocolo de comunicação TCP/IP com interfaces WWW para Internet e Intranets.

4.5.3. O ambiente de construção de agentes Zeus

Desenvolvido por Colins, Ndumu e van Buskirk, esse ambiente é concebido com característica de programação orientada a objetos, constituindo-se por um bem articulado conjunto de ferramentas para suporte à programação de agentes. O Zeus dispõe de um Toolkit para Construção de Agente, que utiliza GUI e é perfeitamente adaptável para ser executado em ambientes multiplataforma.

Conceitualmente o Zeus é uma entidade constituída por cinco camadas: Interface, Definição, Organização, Coordenação e Comunicação. Na camada de Interface estão os sensores e os efetores: Definição, Organização e Coordenação são camadas internas ao agente e a camada de Comunicação dá suporte à comunicação interagentes.

O Zeus permite integração com outras aplicações de forma organizada. Esse ambiente dispõe de uma ontologia robusta que dá bom suporte a processos de diversas naturezas e à manipulação de objetos de várias naturezas.

Na camada de Interface é possível a integração com outras aplicações. Nela estão definidas as capacidades para receber e devolver informações ao usuário. Na camada de Definição o agente é visto como uma entidade autônoma que raciocina. Na camada de Organização, estão definidas as regras de como o agente se relacionará com outros agentes. Na camada de Coordenação fica estabelecida a característica do agente como entidade social que interage conforme protocolos e estratégias pré-determinadas. Por fim, na camada de Comunicação estão implementados os protocolos e demais mecanismos de interação com outros agentes.

Apesar de bem construído, esse ambiente depende de sua biblioteca de componentes, como não se pode concluir se tal biblioteca tem características de software *opensource*, torna-se temerário optar por esse ambiente, em se tratando de projeto de larga escala.

Pode-se construir uma ontologia própria, uma federação de agentes, e construir cada agente dentro do próprio ambiente. A interface é bastante amigável, mas pouco flexível. Construir um agente nesse ambiente requer um conhecimento das facilidades do mesmo e isto exige um tempo de aprendizagem às vezes proibitivo. Tal exigência é pouco viável quando se dispõe de um outro ambiente que dá margem para se produzir aliando-se a cultura já adquirida em Java aos processos de programação de agentes segundo a conceituação atual.

Para a construção do modelo de avaliação a opção natural pelos *aglets* foi considerada a melhor possível, dadas as atuais condições e recursos. Ela dispensa aprendizado adicional e não é limitante para o escopo do projeto. Ainda, é multiplataforma, é flexível e robusta o suficiente. Assim, como não é parte fundamental, entrar amiúde em detalhes dos ambientes não selecionados, optou-se por dar tratos ao trabalho de programação via *aglets*.

4.6. Características dos Agentes Inteligentes

Um agente de software é visto como uma construção autônoma de software; isto é, algo capaz de atuar sem intervenção do usuário. Associou-se a esta característica duas outras condições: a capacidade de se comunicar com um humano ou com outro agente de software e a de monitorar e perceber o ambiente. A habilidade de comunicar na cooperação.

Pode-se afirmar que um agente de software é um componente de software com as seguintes características: execução autônoma; comunicam-se com outros agentes ou com o usuário e; monitora o estado do seu ambiente de execução.

Um agente *inteligente* de software precisa ser mais que isso. Aponta para a necessária definição do significado do termo *software inteligente*. Um tal software, possui as seguintes características: capacidade de explorar significativas quantidades de conhecimento de domínio; tolerância à entrada errada, ou inesperada ou totalmente estranha; capaz de usar símbolos e abstrações; capacidade de comportamento adaptativo e orientado para uma meta; habilidade de aprender com o ambiente; capaz de operar em tempo-real; capaz de comunicar usando linguagem natural.

Para (Hayes-Roth, 1995) os agentes inteligentes devem ser necessariamente capazes de realizar as seguintes funções: perceber condições dinâmicas no seu ambiente; tomar atitudes para modificar condições no seu ambiente; raciocinar para interpretar percepções, resolver problemas, traçar inferências e determinar ações.

Para alguns pesquisadores da IBM, agentes inteligentes são:

"*entidades de software que executam um conjunto de operações em beneficio de um usuário com alguma autonomia e empregam ou o conhecimento ou representações das metas e* desejos *do usuário*" Gilbert & col. (1995).

Agente	Executa autonomamente
	Comunica-se com outros agentes ou com o usuário
	Monitora o estado do seu ambiente de execução
Agente Inteligente	Capaz de usar símbolos e abstrações
	Capaz de explorar quantidades significativas de conhecimento do domínio
Agente Verdadeiramente Inteligente	Capaz de exibir comportamento adaptativo e orientado para metas
	Capaz de aprender com o ambiente
	Tolerante a entradas erradas, inesperadas ou completamente fora de contexto.
	Capaz de comunicar usando linguagem natural

Tabela 4.1 Atributos de um agente inteligente

Wooldridge & Jennings (1995) não apenas exige autonomia, percepção e reatividade como também expande a definição para incluir pró-atividade (isto é, agentes não precisam agir só em resposta ao ambiente, eles precisam ser capazes de exibir comportamento dirigido por metas tomando iniciativas).

Agentes *verdadeiramente* inteligentes devem possuir as capacidades de um agente de software (autonomia, comunicabilidade e percepção) com as capacidades de um agente inteligente (habilidade de manipular conhecimento e tolerar erros, raciocinar com símbolos, aprender e raciocinar em tempo real e comunicar numa linguagem apropriada).

Marvin Minsky afirma que redes com vários agentes simples comunicando e cooperando entre si e executando tarefas simples pode ser tudo o que é necessário para o processamento inteligente (Minsky, 1985).

4.6.1. O que não é um agente

Os agentes inteligentes de software têm sido alvo de exageros de interpretação. Pattie Maes observa que os agentes disponíveis comercialmente dificilmente justificam o nome *agente* (Maes, 1995).

(Foner, 1993) afirma:

> "Encontrei pouca justificativa para a maioria das ofertas comerciais que se autodenominam *agentes*. A maioria delas tendem a antropomorfizar excessivamente o software e conclui então que precisa ser um agente por causa dessa intensa antropomorfização, enquanto que falha simultaneamente em prover qualquer tipo de discurso ou contrato social entre o usuário e o agente. A maioria deles são apenas

> autônomos senão um conjunto de meros enumeradores regularmente programado em lotes. Muitos deles não tem execução tolerável e, portanto não inspiram suficiente confiança para justificar mais do que uma trivial delegação e seus riscos concomitantes".

Dizer que uma aplicação é um agente inteligente de software requer cautela e rigor. Não se trata de comparar a atuação do programa com o modo como um humano resolveria situação similar. Uma aplicação é inteligente quando tem competência que a distingue de forma indiscutível das demais aplicações. Visualizar pela competência, notadamente pela capacidade de aprendizagem, é um bom modo de estabelecer uma boa classificação.

4.7. Classificação de Agentes

Provavelmente existem tantos modos de classificar agentes inteligentes de software quanto existem pesquisadores nesse campo. (Nwana, 1996) estabelece uma tipologia definindo quatro tipos de agentes baseados nas suas habilidades de cooperar, aprender e agir autonomamente. Nwana os denomina por "*agentes espertos, agentes colaborativos, agentes de aprendizado colaborativo e agentes de interface"*. (Nwana, 1996). A figura 4.2 mostra como esses quatro tipos utilizam as capacidades descritas acima.

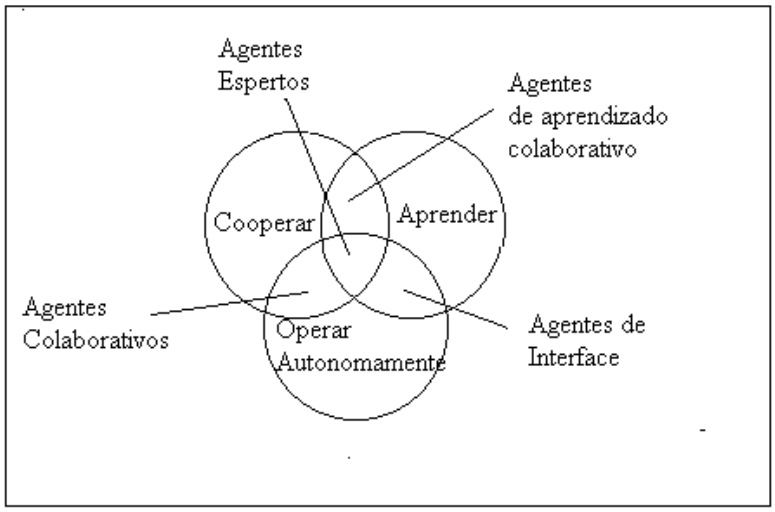

Figura 4.2 Tipologia de Agente (Fonte: H. Nwana, Software Agents: An Overview).

4.7.1. Agentes colaborativos

Agentes colaborativos enfatizam autonomia e cooperação para realizar tarefas por comunicação e possivelmente por negociação com outros agentes possivelmente para atender a mutuo acordo. Esses agentes são usados para problemas distribuídos

onde um agente único seria de aplicação praticamente impossível (por exemplo, controle de tráfego aéreo). É crucial para essa classe de agente a existência de uma linguagem de comunicação bem definida tal como a KQML descrita no Anexo C.

4.7.2. Agentes de Interface

Agentes de Interface são autônomos e usam aprendizado na realização de tarefas para seus usuários. A inspiração desta classe de agentes é o assistente pessoal que colabora com o usuário. Essa classe de agentes é usada para implementar assistentes, guias, auxiliares de memorização e filtros; casar e orientar ações ou comprar e vender em conforme o interesse do usuário.

4.7.3. Agentes Móveis

Agentes móveis são processos computacionais capazes de navegar numa rede (possivelmente na *WEB*) interagindo com "hosts" externos colhendo informações em auxilio ao usuário e a ele retornando quando finda a tarefa de sua responsabilidade. Agentes móveis são implementações de programas remotos (isto é, programas desenvolvidos em uma máquina e enviados para serem executados em outra). Muitos dos assuntos que podem ser abordados pela programação remota precisam ser abordados ao lidarmos com agentes móveis. Isso inclui:

1. *denominação de programas* - dar nomes para que os programas agentes possam distinguir uns dos outros;
2. *autenticação de programas* - autenticar o implementador de um programa agente;
3. *migração de programa* - mover um programa de uma máquina para outra;
4. *segurança de programa* - assegurar que o programa não causará danos à máquina que o executa.

A imprensa leiga, principalmente a imprensa comercial tem classificado os agentes móveis apenas como a única personificação de agentes de agentes inteligentes. Como Nwana aponta enfaticamente: "mobilidade não é condição nem necessária nem suficiente para caracterizar um agente" (Nwana, 1996).

4.7.4. Agentes de informação

Uma das mais populares aplicações de agentes está na busca, análise e recuperação de grandes quantidades de informação. *Agentes de informação* são ferramentas que ajudam a administrar a enorme gama de informações disponíveis em redes como a *WEB* e Internet (Cheong, 1996). Agentes de informação acessam a rede e procuram tipos particulares de informação, filtrando-as e retornando o achado a seus usuários. Agentes de informação são projetados para aliviar a sobrecarga de informação causada pela disponibilidade de uma grande quantidade de informações pobremente catalogadas. Esses agentes tipicamente utilizam protocolos HTTP para acessar informações, embora possam tirar vantagem das linguagens de comunicação interagentes tais como a KQML ou outras.

4.7.5. Sistemas Agentes Heterogêneos

Sistemas Agentes Heterogêneos se referem a coleções de dois ou mais agentes com diferentes arquiteturas de agente. Por causa da grande variedade de domínios de aplicação, é impraticável que seja possível a utilização de uma única arquitetura para atravessar todos os domínios; para cada domínio a melhor arquitetura será selecionada. Os agentes irão se comunicar, cooperar e interoperar cada um com o outro nesses sistemas heterogêneos. O requisito chave para essa interoperabilidade é uma linguagem de comunicação que permita que agentes de diferentes tipos possam se comunicar uns com os outros.

Para os propósitos deste documento vamos considerar como agente inteligente qualquer programa com execução autônoma (isto é, sem intervenção do usuário), que aprenda as preferências e hábitos do usuário, que receba instruções dele e se comunique com o usuário e/ou outros agentes e execute ações que um usuário ou outro agente a ele *design*e. Agentes inteligentes são projetados para oferecer um alto nível de competência e capacidade em domínios específicos. Um agente inteligente precisa ser capaz de desenvolver altos planos e metas e estar apto a satisfazer tais metas.

4.8. Requisitos para uma linguagem de comunicação entre agentes

A comunicação entre agentes requer precisão e assincronicidade. É ainda necessário levar em consideração a exigências diversas. Uma boa linguagem deve estar bem definida em pelo menos três camadas. Numa camada se estabelece o processo de comunicação entre plataformas físicas. Isto é, precisam estar definidos os protocolos de transferências de pacotes e de recebimentos, segundo um modelo universal, que transporte dados entre estações de trabalho em rede.

Numa segunda camada, são estabelecidos os processos de comunicação em *middleware*. E numa terceira, o conteúdo efetivo das mensagens e sua capacidade de interpretação e processamento dos conteúdos. Baseados nesses aspectos, pesquisadores do DARPA Knowledge Sharing propuseram a linguagem KQML. Dada a utilidade, a KQML concorre a ser um padrão de comunicação entre agentes. Veja no Anexo B, detalhes da KQML.

4.9. O Processo de Desenvolvimento de Agentes

Desenvolver um agente inteligente é similar às atividades de desenvolvimento de qualquer outro software nas quais o projetista tem de realizar os passos tradicionais de análise, projeto, implementação, teste e depuração, integração e manutenção. Em muitos aspectos o desenvolvimento de agentes é similar ao desenvolvimento de software orientado a objetos.

Desenvolver agentes inteligentes (comumente denominado processo de *programação orientada a agentes*) consiste em identificar as regras e funções dos vários agentes e especificá-las no comportamento de cada agente. A programação orientada a agentes é similar à programação orientada a objetos exceto que o projetista lida com entidades complexas (agentes) em nível muito mais elevado do que o normalmente feito em programação orientada a objetos.

Um projeto bem elaborado deve levar em consideração que o sistema agente operará num domínio de conhecimento específico. Assim, guiar as ações por vias de generalização e independência de conteúdo pode levar a soluções pouco interessantes

para um particular domínio. A tecnologia de agentes oferece ferramentas para operar em domínios de alto nível de complexidade, contudo um programa só pode fazer o que lhe for autorizado e no limite do que lhe for autorizado.

A Reticular Systems propõe um bom modelo para organizar e desenvolver um projeto de agente inteligente. Focaliza a importância do projetista conhecer e organizar o domínio e dele abstrair as funcionalidades para o programa agente.

Vê-se na figura 4.3, o fluxo proposto. A primeira fase é a organização do projeto. Estuda-se as possibilidades, a natureza do domínio, os recursos disponíveis, levanta-se as necessidades estruturais e operacionais para só então partir para a análise do domínio. Depois de analisado, pode-se obter uma descrição do que precisa ser o sistema. A isso se denomina ontologia. Esta se constitui por um documento que descreva claramente quantos componentes (agentes) vão operar em conjunto, quais os comportamentos destes e como é a comunicação destes entre si e com os usuários. Um programa agente é então concebido com base em três componentes: uma biblioteca de interface com o usuário, uma biblioteca de ações do agente e uma definição comportamental e estrutural do agente. Esse conjunto, ao ser programado já representa a definição completa do agente.

Para que este opere é preciso dar a ele a propriedade de ser executável em alguma plataforma. No caso da Reticular é proposta uma máquina agente "*runtime*" que interpreta a coleção de instruções e transforma o programa descrito em linguagem de alto nível, numa aplicação.

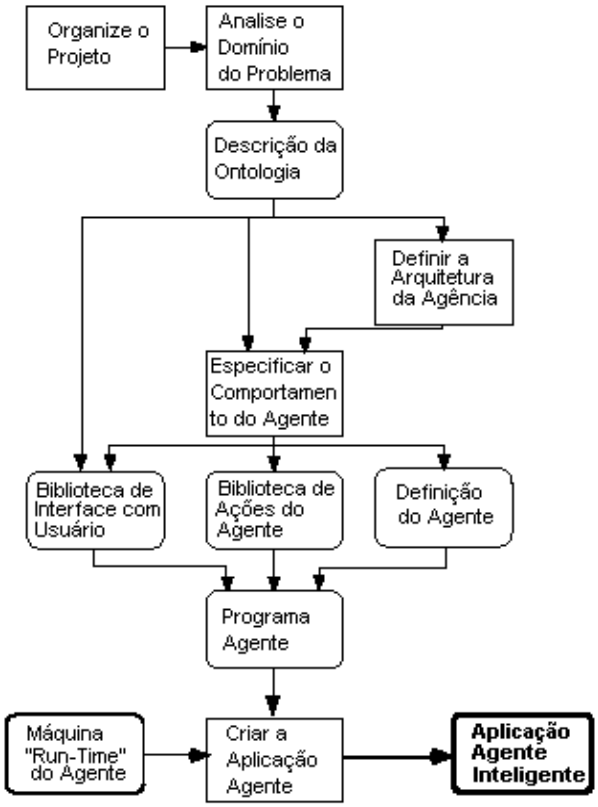

Figura 4.3 Construindo Agentes Inteligentes

4.10. A Organização do Projeto

Estabelecer os recursos é a principal meta dessa fase inicial de projeto. É feito um levantamento de recursos existentes e recursos a serem disponibilizados. Uma decisão de natureza gerencial é requerida para passar para a fase de trabalho analítico. Leva-se em consideração, a existência de ferramentas de software que podem auxiliar no progresso dos trabalhos. Inclui-se aqui a possibilidade de reuso de classes, bibliotecas de software já desenvolvidas, legado de software convencional a ser acessado, recursos humanos e materiais diversos.

Tomada a decisão de levar o projeto a efeito, uma metodologia deve ser escolhida para guiar os trabalhos. Um bom referencial é o PMI[1], para o caso de desenvolvimento de sistemas corporativos complexos. É um instrumento de gestão de fácil compreensão e uso acessível a usuários que possuam formação diversa da área de informática. Ajuda na divisão de responsabilidades e permite divisão de subgrupos por força tarefa, embora

[1] Vide www.pmi.org

não tenha sido concebido para esse fim. Para documentação, o padrão UML (Unified Modeling Language) é um bom recurso. Para desenvolvimentos no âmbito acadêmico, a Rational-IBM oferece condições convidativas para uso do RUP, um ambiente completo de recursos para definição de projetos orientados a objetos baseados na UML, com características de geração de código em diversas linguagens. Pode-se usar o recurso no todo ou apenas para a definição.

4.10.1. Analisar o Domínio do Problema

Um projetista precisa efetuar uma análise do domínio do problema para compreender os requisitos funcionais e de desempenho da sua aplicação e estabelecer uma solução baseada em agente para o problema. A análise do domínio fica facilitada ao se utilizar ferramentas de mapeamento conceitual e ferramentas de modelagem de objetos. O modelo de objeto especifica todos os objetos do domínio e as operações que eles podem realizar. Um outro produto da análise do domínio é uma ontologia para aquele particular domínio. Essa ontologia é uma descrição formal desse domínio de problema, pois confere significado aos símbolos e expressões usados para descrever tal domínio.

Essa organização em ontologia permite que cada programa agente tenha uma interpretação uniforme e não ambígua de cada elemento do programa. Aquilo que deve ser compartilhado deve estar descrito de maneira inequívoca para ser acessado e compreendido a qualquer tempo durante o ciclo de vida do agente. Assim, nada melhor do que especificar em superclasses às quais cada agente herda atributos e funcionalidades com facilidades de representação e de alteração de representação sem esforço adicional devido a definições independentes.

4.10.2. Definição da Arquitetura da Agência

Após completar a análise do domínio, o projetista via de regra decompõe o problema em funções que podem ser realizadas por um ou mais agentes. Identifica cada agente e sua função na solução global do problema. Cria um “esqueleto” do agente e define as características básicas de comunicação entre os agentes.

Identificados os agentes e seus encargos projeta-se a definição do protocolo de comunicação entre eles. São fornecidos editores de protocolos que tornam fácil a tarefa de especificar mensagens e "*handshaking*" (metáfora usada para indicar que um agente sabe o que o outro pediu e concorda em fazer o que foi pedido - assemelha-se a um "aperto de mão") requeridos entre os agentes.

4.10.3. Especificar o Comportamento do Agente

Após completar a definição da agência, o projetista especifica o comportamento de cada agente. O desenvolvimento de um agente é o processo de definir o comportamento do agente. Num modelo BDI isso deve ser representado pela especificação das regras comportamentais, crenças iniciais, comprometimentos, intenções e capacidades do agente.

A biblioteca de interface pode ser utilizada para construir a interface com o usuário do agente. Embora muitos agentes não requeiram quaisquer interações com usuários, outros irão precisar de informação dada pelo usuário e precisarão também oferecer informação de "feedback" para o usuário acompanhar o processamento.

O projetista especifica as ações do agente na biblioteca de ações do agente. Para definir cada ação do agente é preciso cumprir os seguintes passos:

1. Definir o nome da ação e a lista de parâmetros;
2. Associar a ação a um método de um objeto definido no modelo de objeto;
3. Importar bibliotecas de classes existentes ou desenvolver classes em Java que irão implementar a ação;
4. Armazenar essas classes na biblioteca de ações do agente.

O arquivo de definição do agente contém uma especificação detalhada do modelo mental inicial do agente e seu comportamento. Esse arquivo é usado com a biblioteca de ações do agente e a biblioteca de interface com o usuário para especificar completamente as atividades e o comportamento do agente. Esses três componentes englobam o *programa do agente.*

4.10.4. Criar a Aplicação do Agente

O passo final da construção do agente consiste em carregar o programa agente no ambiente de tempo de execução. A máquina do agente é um mecanismo de execução que interpreta o programa agente e executa as ações especificadas na biblioteca de interface com o usuário e na biblioteca de ações do agente. A criação da aplicação depende da plataforma e do ambiente utilizado como ferramenta de desenvolvimento. No caso de utilização de uma linguagem como o Java, a aplicação ganha portabilidade uma vez que há dispositivos *runtime* para várias plataformas físicas.

4.10.5. Depuração da Agência e do Agente

Na programação orientada a agentes não é necessária a depuração de baixo nível de nenhum código fonte porque o trabalho de desenvolvimento é feito em um alto nível de abstração - o agente inteligente. Entretanto, é necessária a capacidade de depuração em alto nível do modelo mental do agente.

Um depurador de agentes deve permitir ao projetista examinar o modelo mental do agente e verificar passo a passo, ciclo a ciclo, o progresso das operações do agente e examinar o modelo à medida que o agente é executado. É vantajoso poder contar com "*breakpoints*" e executar o sistema agente até cada um dos "*breakpoints*" estabelecidos examinando códigos de instrumentação e outros dispositivos de depuração tais como testes sobre casos conhecidos de atuação do agente.