SISTEMAS MULTIAGENTES NAS ENGENHARIAS

Lorí Viali, Dr. – viali@mat.pucrs.br
Professor Adjunto da FAMAT/PUCRS e do IM/UFRGS
Av. Ipiranga, 6681 - Caixa Postal 1429 - 90619-900-900 - PORTO ALEGRE - RS
Fone: (051) 320-3531 - Fax: (051) 320-3631 - htpp://www.mat.pucrs.br/~lori/

Resumo. *Nos últimos anos, os agentes estão sendo utilizados nas mais diversas áreas para solucionar problemas complexos, sendo especialmente freqüentes na área de Engenharia e do ensino via computador. A construção de sistemas multiagentes, assim como de qualquer outro sistema de software, depende hoje de linguagens, módulos, bibliotecas, para que possam ser efetivamente implementados. O paradigma objeto orientado permite o reaproveitamento de código e hoje é impraticável se implementar sistemas a partir do zero, como ocorria no início da década quando o sistema operacional para micros era o DOS. Com as interfaces gráficas cada vez mais sofisticadas, é inviável a construção de qualquer sistema sem o apoio de uma base sólida de componentes pré-construídos. A definição do caminho para se implementar um modelo multiagentes é um problema difícil devido à diversidade de opções oferecidas. Neste artigo são apresentadas as várias opções disponíveis no momento, com o objetivo de auxiliar a escolha da melhor opção.*
Palavras-chave: *Agentes, sistemas multiagentes, linguagens de agentes, plataformas de agentes*

1. INTRODUÇÃO

Nos últimos anos, os agentes estão sendo utilizados nas mais diversas áreas para solucionar problemas complexos, sendo especialmente freqüentes na área de Engenharia e no ensino via computador. Geralmente são sistemas grandes, pesados e pouco flexíveis. A utilização da hipermídia, flexibilizando o fornecimento dos conteúdos aliado a uma estrutura de multiagentes para coordenar e testar o aluno poderá ser uma alternativa promissora sobre o sistema tutorial tradicional.

As aplicações dos sistemas multiagentes já alcançam várias áreas das engenharias apesar de ser um assunto recente em termos de pesquisa.

2. DA INTELIGÊNCIA ARTIFICIAL AO AGENTE INTELIGENTE

Com a evolução da computação, as áreas consideradas de domínio da IA foram mudando. Uma nova fase iniciou com o reconhecimento de que muitos projetos foram bem sucedidos quando aplicados a domínios específicos e bem delimitados. Esta abordagem de modelar conhecimento específico resultou numa das mais bem sucedidas áreas da IA: os sistemas especialistas. A partir dos anos 80 pode ser identificado um novo rumo de trabalho, onde o objetivo agora está voltado para a solução de problemas como o processamento e tradução da linguagem natural, controle de robôs, visão e fala de máquinas e raciocínio por senso comum.

Um ramo da IA denominado de conexionismo ganhou popularidade e expandiu o leque de opções através do uso das redes neurais, modelagem e controle adaptativo. Algoritmos, como o genético, e sistemas alternativos, como os de lógica difusa, têm sido utilizados para dinamizar o campo da inteligência artificial. O crescimento explosivo da Internet e da computação distribuída levou a idéia de pequenos programas móveis que podem ser utilizados para executar tarefas úteis para os seus usuários, dando origem aos **A**gentes **I**nteligentes (AI).

2.1. Agentes

O termo agente é utilizado com vários significados tanto na IA quanto fora dela. O que pode ser percebido da literatura na área é que existem quase tantos conceitos quantos são as pessoas que trabalham no assunto. Assim, pode-se definir agente segundo tal ou qual autor. Alguns dos conceitos encontrados são: "Um **agente** é qualquer coisa que pode ser vista como **percebendo** seu ambiente através de **sensores** e **agindo** sobre este ambiente através de **modificadores**" [RUS95]. "**Agentes inteligentes** são **entidades de software** que executam um **conjunto de operações em benefício de um usuário** ou outro programa com algum grau de **independência** ou autonomia e, em fazendo isto, empregam algum **conhecimento** ou representação dos objetivos ou desejos do usuário" [GIL96].

A classificação de agentes também não é uma tarefa simples. Parece que existem tantas classificações quantas definições. Uma possível maneira de se classificarem os agentes é de acordo com suas características ou dimensões [NWA96]. Uma classificação divide os agentes de acordo com suas habilidades para trafegar em uma rede em **estáticos** ou **móveis**. Uma segunda maneira de classificar agentes é dividi-los quanto à forma de aquisição de inteligência entre **reativos** e **deliberativos**. Um agente **reativo** seria, segundo Goodwin [GOO93], aquele que responde rápida e apropriadamente a alterações em seu ambiente. Já um agente **deliberativo** seria aquele que possui um modelo de raciocínio e um modelo simbólico interno que é utilizado para planejar e negociar de forma coordenada com outros agentes. Nwana [NWA96] classifica os agentes de acordo com os atributos mais importantes que ele considera como sendo a autonomia, a aprendizagem e a cooperação. A **autonomia** seria o princípio de que os agentes podem operar por si só sem a necessidade de um instrutor humano. Os agentes teriam estados individuais internos e objetivos e agiriam de forma a alcançar estes objetivos. A **aprendizagem** seria caracterizada como um aumento do desempenho sobre o tempo. Por fim um agente será cooperativo se possuir habilidade social, isto é, habilidade para interagir com outros agentes e com o agente humano através de alguma linguagem de comunicação [WOO95].

2.2. Sistema multiagentes

Muitas vezes os agentes são mais úteis agindo em conjunto, como partes de um **S**istema **M**ulti**a**gentes (SMA). As razões para isto são inteiramente práticas e seguem a estratégia de fracionar um problema em subproblemas como uma opção para resolvê-lo.

Um SMA pode ser definido como "uma rede fracamente agrupada de solucionadores de problemas que trabalham de uma forma integrada para resolver um problema que estaria além de suas capacidades individuais". Os solucionadores de problemas, denominados "agentes", são autônomos e podem ser "heterogêneos por natureza", isto é, caracterizados por vários graus de capacidade para a resolução do problema [MOU96].

Os agentes de um sistema multiagentes podem ser caracterizados pela capacidade demonstrada na resolução de um problema. A distinção pode ser feita entre agentes reativos, intencionais e sociais. Um agente **reativo** é aquele que reage a alterações no seu ambiente ou a mensagens vindas de outros agentes. Ele não é capaz de “raciocinar” sobre suas intenções (manipulação de objetivos) e suas ações são executadas através do disparo de regras,

atualização da base de conhecimentos ou do envio de mensagens para outros agentes ou ambientes. Um agente **intencional** é capaz de "raciocinar" sobre suas intenções e crenças, para criar um plano de ações e executar este plano. Em um SMA os agentes intencionais se coordenam pela troca de informações sobre suas crenças, objetivos e ações. Esta informação é incorporada em seus esquemas. Um agente **social** possui modelos explícitos de outros agentes e é capaz de manter estes modelos (atualizando crenças, objetivos e eventualmente planos).

2.3. Arquiteturas de agentes

A estrutura de um sistema multiagentes pode ser classificada em função das ações que serão executadas pelos agentes. Se a ação é feita por deliberação explícita sobre um conjunto de diferentes opções, através da utilização de um modelo interno do ambiente, então a estrutura é denominada de arquitetura **deliberativa**. Se a ação a ser tomada está diretamente ligada a ocorrência de um conjunto de eventos no ambiente, isto é, a ação depende de um conjunto de pré-condições, então a arquitetura é dita **não-deliberativa**. Se, por outro lado, as ações forem tomadas de qualquer uma das duas formas acima, então a arquitetura é dita do tipo **híbrido**. Uma vez definido os agentes do modelo e a arquitetura do sistema é necessário determinar a forma de implementação do modelo. Esta talvez seja a etapa mais problemática para a viabilização de um modelo de multiagentes. Não pela falta de opções, mas sim pelo excesso delas. O número de alternativas disponíveis é tão grande e diversificada que a tomada de uma decisão nem sempre poderá ser feita sem uma análise prolongada e cuidadosa de todas as alternativas disponíveis.

As possíveis alternativas para a implementação de um modelo multiagentes envolvem considerações sobre três categorias:

❶ Utilização de uma camada intermediária de software;
❷ Utilização de uma linguagem especificamente projetada e
❸ Utilização de uma linguagem de propósito geral.

3. CAMADAS DE SOFTWARE

Existem diversas formas para implementar uma plataforma multiagentes. Uma opção é o desenvolvimento de agentes sobre camadas de software projetadas para facilitar o desenvolvimento e a implementação e que rodam sobre um sistema operacional, ou seja, são extensões do sistema operacional com o propósito específico de gerenciamento, desenvolvimento, coordenação e comunicação entre agentes. A "Tabela 1" apresenta algumas destas opções. A utilização de um destes ambientes pode permitir alguma flexibilidade na escolha da linguagem de programação ou então forçar o desenvolvimento em uma linguagem específica (a Java, por exemplo).

A grande virtude desta forma de implementação é a facilidade de criação e manutenção dos agentes, pois todo o trabalho miúdo e pesado de programação já foi realizado, sendo necessário apenas uma adaptação às características do ambiente sendo utilizado. Esta opção, por um lado viabiliza alguns projetos e reduz consideravelmente o ciclo de desenvolvimento de outros, mas por outro lado limita drasticamente a flexibilidade do projetista. A limitação decorre do fato de que a maioria destes ambientes exige a programação em uma linguagem específica e o sistema resultante só poderá rodar em plataformas que possuem esta camada adicional de software instalada ou nas quais este ambiente também rode.

O sistema denominado "Aglets" da empresa IBM é uma destas camadas de software que foi desenvolvida em Java e que requer que os agentes também sejam programados nesta linguagem. Situação semelhante ocorre com o Concordia da empresa Mitsubishi Electric. Menos mal já que a Java é uma linguagem que vem ganhando popularidade rapidamente. Já o ambiente Mozart desenvolvido por um consórcio de instituições requer que os agentes sejam

programados em uma linguagem própria denominada de Oz. Os exemplos citados acima são bastante restritivos, pois fornecem apenas uma linguagem como opção de programação, o mesmo não ocorre com o projeto TACOMA que oferece suporte para seis diferentes linguagens, entre elas o C++ e o Visual Basic e roda nas plataformas Windows 98 e NT e Unix. De qualquer forma se por um lado estes ambientes agilizam parte da programação, por outro lado será necessário estudá-los e entendê-los e possivelmente aprender a linguagem suportada para se poder fazer uso deles.

Tabela 1. Camadas de software para a implementação de agentes

Módulo	**Características**
Aglets	Módulo para o desenvolvimento de agentes da empresa IBM. Foi desenvolvido em Java [CHA96].
Concordia	Esforço do laboratório de sistemas Horizonte da empresa Mitsubishi Electric Information Center America. É desenvolvido inteiramente em Java e apresenta versões para rodar na plataforma Windows (95, 98 e NT) e para o sistema Solaris. Possui classes específicas de agentes permitindo o desenvolvimento dos mesmos de forma simples [WON97].
Inferno	Camada de software da Lucent Technologies, centro de pesquisas dos laboratórios Bell. É uma linguagem para aplicações distribuídas que incluí protocolos e APIs, isto é, um sistema completo [PAR98].
Mole	Módulo de desenvolvimento de agentes que utiliza a linguagem Java. Está sendo desenvolvido pelo grupo de Sistemas Distribuídos da Universidade de Sttugart (Alemanha). O módulo permite a escrita e execução de agentes móveis em Java [STR97].
Mozart	Sistema de suporte a computação distribuída aberta e a inferência baseada em restrições. Implementa a linguagem concorrente e objeto orientada Oz, para o desenvolvimento de sistemas multiagentes. O projeto é uma colaboração entre os departamentos de Ciências Computacionais das Universidades DFKI (Alemã) e Católica de Louvain (Bélgica) e o Laboratório de Sistemas Inteligentes do Instituto Sueco de Ciências Computacionais [MÜL95].
MSAgent	Camada de software desenvolvido pela empresa Microsoft que é integrada ao sistema operacional Windows (95, 98 e NT). Utiliza a tecnologia COM e ActiveX da empresa. Pode ser utilizado diretamente pelos navegadores através de uma marcação <OBJECT> ou então inserido como componente nas linguagens que suportam a tecnologia COM.
TACOMA (*Tronsø And Cornell Moving Agents*)	Camada de software para o suporte de processos (agentes), escrito em C e que utiliza o protocolo TCP. Suporta as linguagens C, C++, Perl, Python, Scheme e Visual Basic e as plataformas Unix e Windows (95 e NT). Está sendo desenvolvido como um esforço conjunto entre os departamentos de Ciências Computacionais das Universidades de Cornell (EUA), Tronso (Noruega) e da Califórnia (São Diego) [JOH96].

4. LINGUAGENS ESPECÍFICAS

Uma segunda alternativa para a implementação de sistemas multiagentes é a utilização de uma linguagem específica para este fim. Diversas opções já estão disponíveis. A "Tabela 2" apresenta algumas destas opções.

Tabela 2 - Linguagens específicas para a programação de agentes

Linguagem	Características
AKL (*Agents Kernel Language*)	É uma linguagem concorrente desenvolvida no Instituto Sueco de Ciências Computacionais. Possui bibliotecas para o suporte a escrita de agentes.
April (*Agent PRocess Interaction Language*)	Linguagem orientada a construção de sistemas multiagentes. É uma linguagem concorrente baseada em objetos, onde os objetos são processos. Oferece uma interface para a linguagem C [McA94].
Lalo	É uma linguagem orientada a agentes e uma estrutura para o desenvolvimento de sistemas multiagentes. Um programa escrito em Lalo é traduzido para o C++. Os agentes se comunicam através da KQML. A linguagem está disponível para as plataformas Unix e Windows (95 e NT).
Obliq	É uma linguagem não tipada e interpretada que suporta computação objeto orientada distribuída. A linguagem é uma especialização da Modula-3 que deve estar no computador do usuário para que possa rodar a Obliq [CAR95].
Python	É uma linguagem objeto orientada e interpretada, semelhante a Tcl, Perl, Scheme ou Java [CAM97]
Scheme	Linguagem de programação derivada do Lisp que incorpora o cálculo lambda, distinguindo procedimentos de expressões lambda e símbolos, utilizando um único ambiente léxico para todas as variáveis. A implementação do MIT é um ambiente completo que roda em plataformas Unix, Windows e IBM OS/2.
Tcl (*Tool Command Language*)	É a combinação de uma linguagem de script com uma biblioteca. A linguagem foi projetada com o objetivo de escrever programas interativos tais como editores de textos. A biblioteca pode ser embutida em aplicativos e consiste de um parser para a linguagem e rotinas para a implementação de comandos e procedimentos. A linguagem possui um ambiente gráfico de programação denominado de Tk (Tool Kit).
Telescript	É uma linguagem interpretada e objeto orientada que objetiva o desenvolvimento de aplicações distribuídas. A primeira classe da hierarquia de classes da linguagem é a classe "processo" [NWA96]. A linguagem foi desenvolvida pela empresa General Magic e é acompanhada de um ambiente de desenvolvimento.

5. LINGUAGENS GERAIS

Uma terceira e última opção é o desenvolvimento a partir de linguagens objeto orientadas que, apesar de não possuírem facilidades específicas para implementação de agentes, apresentam a vantagem da proximidade conceitual entre agentes e objetos. Entre as linguagens que podem ser utilizadas estão o C++, o Java e o Delphi. Destas, a que é atualmente mais utilizada nas implementações relatadas na literatura é a linguagem Java pelo fato de possuir uma camada de implementação, não de agentes em si, mas através de uma

máquina virtual Java que é utilizada pelos navegadores de rede e que permite a implementação de software em múltiplas plataformas, sem a necessidade de adaptações a cada hardware específico. Desta forma um agente escrito para uma plataforma Windows, poderá rodar em uma estação Unix.

Uma área de software de intensivos investimentos reside na criação de pacotes de desenvolvimento rápido - RAD (***R**apid **A**pplication **D**evelopment*). Um pacote deste tipo consiste de uma linguagem de programação, um modelo de componentes para a conexão de elementos de interação, denominados de "controles", e um ambiente personalizável (os editores de recursos). Um dos mais bem sucedidos ambientes RAD tem sido o Borland Delphi, entretanto o primeiro destes pacotes a ter ampla utilização foi o Microsoft Visual Basic [HUG97]. O produto da Borland utiliza uma versão proprietária do Pascal objeto, enquanto que o da Microsoft utiliza uma versão proprietária do Basic. O Delphi não é apenas um compilador, mas um ambiente completo que inclui um editor de texto com amplos recursos, um compilador, um depurador, bibliotecas de procedimentos, bibliotecas de classes e objetos, biblioteca encapsulando a API do Windows e outras ferramentas úteis, como o Winsight, um visualizador de classes.

Como o ambiente Java é tido como robusto e uma linguagem de programação segura ele está sendo visto como um inimigo natural dos ambientes RAD, mas a linguagem Java não suportava uma tecnologia considerada fundamental: a de componentes. Componentes de software são uma convenção para a escrita de controles de interface com o usuário, de forma que as ferramentas de desenvolvimento de aplicações e ambientes de execução possam manipulá-los. O JavaBeans é a resposta dos desenvolvedores do Java para o ingresso na arquitetura dos componentes. Para fazer frente a esta nova ameaça ao ambiente Visual Basic, a Microsoft estendeu a tecnologia COM para a Internet através dos controles ActiveX, oferendo suporte para que controles ActiveX sejam invocados a partir da linguagem Java e que a Java possa invocar controles ActiveX.

5.1. Os componentes de software

Os componentes de software estão se tornando tradicionais em todas as plataformas. Conforme a plataforma utilizada, eles podem receber diferentes denominações comerciais. Na plataforma Windows e Macintosh, os componentes foram denominados de tecnologia COM (***C**omponent **O**bject **M**odule*), na OS/2 da IBM de SOM (***S**ome **O**ther **M**odel*) e nas demais de CORBA (***C**ommon **O**bject **R**equest **B**roker **A**rquitecture*). O COM é a implementação do OLE (***O**bject **L**inking and **E**mbedding*) e este é uma extensão do modelo DDE (***D**ynamic **D**ata **E**xchange*). A utilização da área de transferência (*clipboard*) num PC permite a cópia de uma aplicação para outra. O DDE permite a conexão entre dois documentos e pode ser vista como uma extensão da área de transferência. O OLE permite a cópia dos dados de uma aplicação servidora para uma aplicação cliente. Os dados podem ser copiados junto com a conexão (*link*) formando a "inserção do objeto" ou deixados na aplicação original caracterizando a "conexão OLE".

A tecnologia OLE original foi atualizada para a OLE2 e recebeu novas características tais como a "automação" e "controles". Além disso, o sistema operacional Windows recebeu uma camada de software para suportar esta nova versão da tecnologia, que foi renomeada de "controles ActiveX", permitindo que controles mais "leves" sejam apropriados para a distribuição através da Internet.

Os termos OLE e COM são utilizados normalmente da mesma forma, uma vez que o COM especifica os detalhes técnicos do OLE e qualquer linguagem COM-compatível poderá ser utilizada para escrever objetos COM/OLE. A tecnologia COM incorporou o conceito de objetos distribuídos através do modelo DCOM. A empresa Microsoft e o OMG (***O**bject **M**anagment **G**roup*), responsável pelo modelo CORBA, concordaram em estabelecer uma

abordagem que permita que um objeto DCOM possa se comunicar com um CORBA e vice-versa.

O DCOM é um conjunto de conceitos e objetos de interface em que um programa cliente ou objeto pode solicitar serviços, utilizando RPC (***R**emote **P**rocedure **C**all*), a um programa servidor ou objeto, ou a outros computadores em uma rede. Já o COM fornece um conjunto de interfaces para a comunicação entre programas, módulos ou objetos em um mesmo computador. O DCOM faz parte do ambiente NT a partir da versão 4.0 e pode ser gratuitamente adicionada ao Windows 95. Estará em breve disponível para a plataforma Unix e IBM.

5.2. Controles ActiveX

A linguagem Visual Basic da Microsoft foi o primeiro ambiente de programação a utilizar a idéia de fornecer componentes de software, se bem que a idéia de componentes reutilizáveis seja tão antiga quanto o conceito de orientação a objetos.

Essencialmente, um controle deste tipo fornece condições para que componentes possam se comunicar uns com os outros não importando a linguagem de implementação ou a plataforma suportada. Isto não quer dizer que um controle ActiveX seja portável, mas sim que a comunicação pode ser realizada entre diferentes plataformas que suportam a tecnologia, da mesma forma que muitos outros ambientes exigem camadas intermediárias de software para serem portáveis. Por exemplo, agentes construídos sobre o sistema Concordia e as applets precisam de um interpretador Java rodando na máquina cliente.

Tabela 3 - Opções para utilizar controles ActiveX [COF96**]**

Linguagem	Vantagens	Desvantagens
C++	Melhor acesso a API. Código rápido e compacto. Grande número de implementações e de recursos testados.	Difícil de aprender, escrever e manter. Itens adicionais acrescidos na linguagem em diferentes épocas e por vendedores de compiladores diferentes.
Delphi	Produtiva, fácil de manter e com recursos altamente integrados.	Linguagem proprietária. Requer recursos de terceiros para gerar controles ActiveX.
Java	Altamente portável e segura. Possui muitas implementações, mas todas em conformidade com as especificações da empresa Sun.	O desempenho varia conforme a plataforma. É uma tecnologia ainda bastante jovem.
Visual Basic	Melhor conhecida do que qualquer outra ferramenta de desenvolvimento. Altamente portável para codificar aplicações GUI.	O desempenho em computação intensiva é inferior as demais linguagens. A geração de controles ActiveX só se tornou disponível na versão 5.

Um controle ActiveX incorpora os serviços COM e DCOM. Apesar de ser teoricamente compatível com outras plataformas, é necessário que o controle seja recompilado, antes, na plataforma que deverá ser utilizado. Cada controle OLE possui uma janela que ocupa espaço na memória, mesmo que esta janela nunca apareça. O uso de uma janela também significa que o controle, quando visível, deve ser retangular e não transparente, devendo suportar automação, edição visual e deve ser capaz, ainda, de manejar eventos entre ele e o local onde está inserido. Já um controle ActiveX não precisa ser uma janela e não precisa fornecer estas

habilidades. Ele deve satisfazer somente duas propriedades: deve poder responder questões sobre o que ele pode fazer e deve poder determinar quando não é mais necessário e liberar a memória por ele próprio. Isto é feito através da implementação do conceito de interface desconhecida (*IUnknown*). A Tabela 3 apresenta algumas vantagens e desvantagens das principais linguagens que utilizam controles ActiveX.

5.3. Outras facilidades (infra-estrutura)

A programação de agentes possui algumas particularidades que a programação convencional não apresenta. Algumas características dos agentes como a necessidade de comunicação entre eles, a mobilidade e a inteligência requerem esforços adicionais de programação. Como conseqüência algumas linguagens estão sendo criadas para atender a estas características.

Independente da forma e da arquitetura utilizada para implementar um conjunto de agentes, eles precisam trocar mensagens. Para que a comunicação possa ser mais efetivamente realizada, alguns padrões de linguagens de comunicação estão sendo propostos. A "Tabela 4" apresenta os principais projetos na área. A classificação utilizada nem sempre poderá ser isenta de críticas, pois o que para alguns autores é uma linguagem de programação para outros a linguagem serve mais a comunicação. Por isto é possível o encontro de divergências na literatura a respeito das classificações utilizadas neste trabalho.

Tabela 4 - Principais linguagens de comunicação de agentes

Linguagem	Características
ACL (*Agent Communication Language*)	Linguagem de comunicação de agentes desenvolvida através de uma iniciativa da DARPA. A linguagem é composta de um "vocabulário", um método de acesso externo através da KQML e uma linguagem interna de representação de conhecimentos a KIF [FIN94].
Agent Talk	É uma linguagem acrescida de um protocolo de coordenação. O protocolo pode ser definido de forma incremental e facilmente customizado através de um mecanismo de herança. Está sendo desenvolvida como um esforço conjunto do Laboratório de Ciências da Comunicação NTT e o Laboratório Ishida do departamento de Ciências da Informação da universidade de Kyoto.
KQML (*Knowledge Query and Manipulation Language*)	É uma linguagem mais um conjunto de protocolos que objetiva fornecer facilidades para que programas computacionais (agentes) se conectem e troquem informações entre si. É formada pelas camadas de conteúdo, de mensagem e de comunicação [FIN94].

Agentes inteligentes devem possuir uma base de conhecimentos. Atualmente não existe um padrão universal para a troca de conhecimentos. Assim, cada aplicação constrói sua própria base de conhecimentos e utiliza linguagens e metodologias particulares, acarretando a inviabilidade do aproveitamento do esforço de desenvolvimento em outras aplicações.

Para fornecer suporte ao reaproveitamento e o compartilhamento de conhecimentos formalmente representados entre sistemas multiagentes é útil à definição de um vocabulário comum em que o conhecimento compartilhado possa ser representado. Uma especificação de um vocabulário para um domínio de discurso compartilhado, incluindo definições de classes, relações, funções e termos, é denominado de **ontologia**. Alguns projetos estão realizados no sentido de formalizar um padrão para a troca de conhecimentos entre agentes de diferentes aplicações. A "Tabela 5" apresenta alguns resultados.

Tabela 5 - Linguagens de troca de conhecimentos

Linguagem	Características
KIF (*Knowledge Interchange Format*)	É uma versão do cálculo de predicados de primeira ordem com extensões para raciocínio não-monotönico e definições [FIN94]. A linguagem está sendo desenvolvida pelo Interlingua Group.
Ontolingua	O KSL (Knowledge Systems Laboratory) da universidade de Stanford está desenvolvendo ferramentas e serviços para suporte ao processo de obtenção de ontologias compartilhadas comuns por grupos geograficamente distribuídos [GRU93].
SRKB (*Shared, Reusable Knowledge Bases*)	O grupo SRKB está trabalhando no sentido de formalizar o conteúdo de bases de conhecimento compartilhadas. Um dos objetivos é compartilhar o conhecimento de áreas específicas, mas com metodologias e ferramentas independentes de conteúdo [FIN94].

6. CONCLUSÃO

Nos últimos anos, os agentes estão sendo utilizados nas mais diversas áreas para solucionar problemas complexos. O ensino por computador é um problema deste tipo. Geralmente são sistemas grandes, pesados e pouco flexíveis. A utilização da hipermídia, flexibilizando o fornecimento dos conteúdos aliado a uma estrutura de multiagentes para coordenar e testar o aluno poderá ser uma alternativa promissora sobre o sistema tutorial tradicional.

É inegável que a teoria dos agentes e dos sistemas multiagentes vem despertando bastante entusiasmo em todas as áreas. As aplicações em Engenharia crescem diariamente, assim como o número de recursos que pretendem facilitar a construção destes sistemas. Justamente em virtude do grande número de opções e da diversidade destas opções é que a decisão sobre a melhor alternativa não é uma decisão simples. Neste artigo foram apresentadas algumas das principais opções de caminho que se pode seguir para implementar um sistema de multiagentes.

7. REFERÊNCIAS

[**CAM97**] CAMERON, Laird, SORAIZ, Kathryn. Choosing a scripting language. *SunWorld*. USA, Oct. 1997.

[**CAR95**] CARDELLI, Luca. A Language with Distributed Scope. Palo Alto, CA: Digital Equipment Corporation, Systems Research Center, May 1995.

[**CHA96**] CHANG, D. T., LANGE, D. B. Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In: *Proceedings of the OOPSLA96 Workshop: Toward the Integration of WWW and Distributed Object Technology*. Oct. 1996.

[**COF96**] COFFEE, Peter. Feature. *Windows Source*. 1996, p. 1-8 [*Online*] http://www4.zdnet.com/wsources/content/0197/sub1.htm.

[**FIN94**] FININ, Tim, LABROU, Yannis, MAYFIELD, James. KQML as an agent communication language. Baltimore: University of Maryland, Technical Report, Computer Science Department. 1994.

[**GIL96**] GILBERT, Don, JANCA, Peter. IBM Intelligent Agents. Research Triangle Park, Raleigh, USA: IBM Corporation, 1996.

[**GOO93**] GOODWIN, Richard. Formalizing Properties of Agents. Pittsburgh: CMU, School of Computer Science. Technical Report, May 1993.

[**GRU93**] GRUBER, T. R. Towards principles for the design of ontologies used for knowledge sharing. In: *International Workshop on Formal Ontology*. Mar. 1993.

[**HUG97**] HUGHES, Merlin. JavaBeans and ActiveX go head to head. *JavaWorld*, March 97 [*online*] http://www.javaworld.com/javaworld/jw-03-1997/.

[**JOH96**] JOHANSEN, Dag, SCHNEIDER, Fred B., RENESSE, Robert van. What TACOMA Taught US. [*Online*] http://www.cs.uit.no/DOS/Tacoma/index.html.

[**McA94**] McCABE, F. G., CLARK, K. L. **April** - Agent PRocess Interaction Language. In: *Intelligent Agents*. LNCS, Berlin: Springer-Verlag, v. 890, 1995.

[**MOU96**] MOULIN, Bernard, CHAIB-DRAA, Brahim. An Overview of Distributed Artificial Intelligence. In: *Foundations of Distributed Artificial Intelligence*. New York: John Wiley & Sons, 1996, p. 3-55.

[**MÜL98**] MÜLLER, Martin, MÜLLER, Tobias, ROY, Peter Van. Multiparadigm Programming in OZ. In: *Visions for the Future of Logic Programming: Laying the Foundations for a Modern Successor of Prolog*. Portland, Oregon, Dec. 7.

[**NWA96**] NWANA, Hyacinth. Software Agents: An Overview. *Knowledge Engineering Review*. UK, v. 11, n. 3, p. 205-244, Oct./Nov. 1996.

[**PAR98**] PARK, Jooho. Implementation of SNMP Agent by Inferno. Murray Hill, NJ: Bell Labs Innovations, Lucent Technologies, 1998.

[**WON97**] WONG, David, PACIOREK, Noemi, WALSH, Tom. Concordia: An Infrastructure for Collaborating Mobile Agents. Waltham, MA: Horizon Systems Laboratory, Mitsubishi Electric ITA, 1997.

[**WOO95**] WOOLDRIDGE, Michael, JENNINGS, Nicholas R. Agents Theories, Architectures, and Languages: a Survey. In: "*Intelligent Agents (Wooldridge and Jennings Eds*)". Berlin: Springer-Verlag, 1995, p. 01-22.

[WOO95] WOOLDRIDGE, Michael, JENNINGS, Nicholas R. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*. UK, v. 10, n. 2, p. 115-152, 1995.

[**RUS95**] RUSSEL, Stuart J., NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, New Jersey: Prentice Hall, 1995, 932 p.

[**STR97**] STRABER, Markus, BAUMANN, Joachim, HOHL, Fritz. Mole - A Java Based Mobile Agent System. Stuttgart: IPVR (Institute for Parallel and Distributed Computer Systems), University of Stuttgart, 1997.