UCL – Uma Linguagem de Comunicação para Agentes de Software

Carlos Alberto Estombelo Montesco

Orientador
Prof. Dr. Dilvan de Abreu Moreira

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação, da Universidade de São Paulo - USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área de Ciências de Computação e Matemática Computacional.

São Carlos

Outubro de 2001

"Being a scientist means living on the borderline between your competence and your incompetence, if you always feel competent, you aren't doing your job.
-Carlos Bustamante"

AGRADECIMENTOS

A meus pais, Andrés e Eufemia por acreditar em mi todo este tempo, pelo apoio, carinho e amor oferecidos à distancia desde Arequipa.

A meu irmão, Richard pela força, apoio e motivação que sempre colocou nas minhas decisões em todo momento.

Ao professor Dilvan, que acreditou na minha capacidade para realizar meu mestrado. Obrigado pela ajuda nos momentos difíceis e pela amizade. Sem duvida aprendi muitas coisas valiosas que estarão presentes na minha vida.

Aos amigos que sempre estão por perto, Alex, Percy, Pastor, Eduardo, Ernesto, Govi, Miluska, Cesar, Ursula, Christian, Patricia, Guillermo, Juan Carlos, Jorgito. Sem eles seria muito mais difícil este caminho.

A todos os colegas que dividiram seu tempo comigo durante todo o mestrado. Em especial ao pessoal de Minas Gerais, Obrigado a vocês por compartir sua vida comigo, Débora, Iris, Elaine, Thiago, Stênio, Werley ... com vocês conheci e entendi as palavras "gente boa!".

À Marília, Beth e Laura, pela paciência e por estarem sempre prontas a me atender.

À CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), pelo apoio financeiro para a realização deste trabalho.

ÍNDICE

Figuras e Tabelas

RESUMO

Uma parte importante, da proposta dos agentes de software, é o princípio que esses agentes podem funcionar de forma mais eficiente quando trabalham em grupos. Para que a cooperação entre agentes tenha sucesso, é requerida comunicação entre eles. Para que essa comunicação seja possível precisa-se de uma Linguagem de Comunicação entre Agentes (em inglês Agent Communication Language, ou ACL). Dentro de uma ACL, torna-se importante a forma como são comunicadas as mensagens, isto é, se as mensagens expressam adequadamente seu propósito sob um ponto de vista semântico.

O objetivo deste trabalho é a especificação de uma nova ACL, chamada UCL – *Universal Communication Language*, que se preocupa com a descrição da estrutura das mensagens, com o modelo semântico e com suporte a protocolos para interação entre agentes (de software ou humanos). Além disso, é importante explorar, no contexto deste trabalho, a utilização do padrão XML (*Extensible Markup Language*), para atribuir à linguagem UCL meios para uma fácil integração à Internet. Por isso a linguagem UCL foi implementada no padrão XML.

Foi desenvolvido também um protótipo de enconverter-deconverter, que serviu como ferramenta para experimentação e teste da proposta de linguagem apresentada. Esse protótipo permite a conversão de inglês para UCL e de UCL para inglês, ele usou ferramentas e programas abertos, estando disponível para uso de todos sob a GPL (GNU Public License).

ABSTRACT

An important part of the software agents' theory is the idea that the agents should work more efficiently in groups. For the cooperation among software agents to be successful, it is required communication between them. To enable this communication an Agent Communication Language (ACL) is required. Messages coded in an ACL should adequately express their meaning from a semantic point of view.

The goal of this work is to specify a new ACL language, called Universal Communication Language (UCL). UCL design is concerned with the description of the messages' structures, the underlining semantic context and the support for protocols for agents interaction (software agents or humans agents). Also, it is important to explore the use of the Extended Markup Language (XML) to make UCL easier to integrate into the Internet. For this reason UCL was implemented using XML.

Also, an enconverter-deconverter software prototype was written to serve as a tool for testing and experimenting with the language specifications. This prototype allows the conversion from English to UCL and from UCL to English. It was written in Java using open programs and it is available free under the GPL (GNU Public License).

1. INTRODUÇÃO

1.1 Considerações iniciais.

À medida que cresce o interesse pela tecnologia de agentes de software, as ferramentas de software para projeto e construção de sistemas baseados nessa tecnologia começam a aparecer (Sierra et. al., 2000).

Em anos recentes, o interesse em sistemas multi-agentes (MAS) cresceu de forma tal que a abrangência das aplicações baseadas em MAS vai desde bibliotecas digitais até comércio eletrônico. Nessas aplicações, considera-se a existência de cooperação entre seus componentes para poder realizar suas tarefas, em conseqüência todas estas aplicações têm uma coisa em comum: os agentes que operam dentro destes sistemas têm que comunicar (Dignum, 2000).

Neste capítulo, são apresentados as motivações, um breve resumo dos trabalhos relacionados, os objetivos que levaram ao desenvolvimento deste trabalho e a forma como esta monografia está organizada.

1.2 Motivação

A tecnologia de agentes desperta o interesse para criar uma nova forma de sistemas de software complexos. No projeto de agentes, misturam-se muitas das técnicas tradicionais de inteligência artificial ("raciocínio" no nível de conhecimento, flexibilidade) com as experiências obtidas nas áreas de sistemas distribuídos, teorias sobre negociação e teorias de equipe de trabalho, como também das ciências sociais (Dignum, 2000).

Uma parte importante da proposta dos agentes de software é que esses agentes (similarmente aos humanos) podem funcionar de forma mais eficiente quando trabalham em grupos, tendo como características importantes a cooperação e a divisão do trabalho.

Para que a cooperação entre agentes tenha sucesso, é requerida comunicação entre eles. Uma coleção de agentes trabalhando juntos em cooperação pode ser vista como uma pequena sociedade, e o funcionamento de qualquer sociedade coerente precisa de uma linguagem comum e um meio de comunicação. Agentes que trabalham de forma isolada provavelmente serão menos eficazes do que aqueles que são capazes de interagir. (David, 1999).

É tão vasto o campo da comunicação que se torna difícil encaixá-lo numa definição exata. Em geral, a comunicação é o intercâmbio intencional de informação que se dá mediante a emissão e percepção de sinais que pertencem a um sistema convencional.

O que poderia motivar um agente a realizar atos de comunicação, enquanto ele poderia se concentrar apenas em realizar ações comuns? Em um caso hipotético, suponha que um grupo de agentes esteja pesquisando em um mundo de domínio específico de conhecimento, um labirinto, e os agentes precisem encontrar um objeto que se encontra em algum lugar do labirinto. Em função da tarefa encarregada aos agentes, o grupo se beneficia (em nível coletivo e individual) quando é capaz de fazer as seguintes ações:

- Trocar informações entre si a respeito da parte do mundo que cada um deles está pesquisando e, assim, poupar trabalho de pesquisa;
- consultar outros agentes sobre determinados aspectos do mundo;
- responder a perguntas de outros agentes;
- aceitar petições e propostas;
- compartilhar experiência entre si.

Para estes atos de comunicação e a cooperação entre os agentes serem possíveis precisa-se de uma Linguagem de Comunicação entre Agentes (em inglês *Agent Communication Language*, ou ACL). Dentro de uma ACL, torna-se importante a forma como são comunicadas estas mensagens, isto é, se as mensagens expressam adequadamente seu propósito sob um ponto

de vista semântico. Na continuação, será mostrado um breve resumo de trabalhos relacionados a este assunto.

1.3 Trabalhos relacionados

Uma primeira tentativa para padronizar uma linguagem de comunicação entre agentes veio do grupo *Knowledge Sharing Effort (KSE)*, que produziu a *Knowledge Query and Manipulation Language* (KQML). O objetivo da KQML foi desenvolver fundamentos para interação e interoperabilidade de sistemas de software (ARPA, 1993).

Recentemente, outros esforços estão sendo feitos para padronizar as ACLs. Estas iniciativas foram adotadas pela organização *Foundation for Intelligent Physical Agents* (FIPA). Em particular, o padrão FIPA para ACL tenta identificar os componentes práticos da comunicação e cooperação inter-agentes (FIPA, 1999).

Estes e outros trabalhos como programação orientada a agentes, comunicação entre agentes móveis, surgiram com propósitos similares aos já mencionados e serão revisados de forma mais clara no capítulo seguinte.

Embora estas iniciativas mostrem que alguns trabalhos foram feitos em relação às ACLs, de forma geral, percebe-se a falta de consenso nos fundamentos para a comunicação entre agentes. Não há um entendimento semântico claro das mensagens a serem transmitidas ou até mesmo os conceitos básicos que deveriam ser usados para definir uma semântica (Dignum, 2000).

Além disso, apesar de a Internet ser um repositório enorme de informação e que muitos sistemas fazem uso dela, percebe-se uma falta de integração das ACLs com as tecnologias da Internet (Grosof & Labrou, 1999). Em conseqüência, é razoável sugerir que as ACLs deveriam se integrar de uma forma mais fácil a Internet e serem capazes de trabalhar com as ferramentas e a infraestrutura que estão disponíveis nela. Isso motiva a se tentar utilizar a meta linguagem *Extensible Markup Language* (XML), projetada para uso na Internet, para codificação de mensagens em uma ACL como primeiro passo para esta integração.

1.4 Objetivos

O objetivo deste trabalho concentra-se na especificação de uma nova ACL, chamada UCL– *Universal Communication Language*, que se preocupa com a descrição da estrutura das mensagens, com o modelo semântico e com suporte a protocolos para interação entre agentes (de software ou humanos) na Internet.

Na comunicação entre agentes, é imprescindível um entendimento adequado do que vai ser comunicado através da troca de mensagens. Uma boa representação do domínio de conhecimento pode colaborar para um melhor entendimento do contexto em que a troca de mensagens acontece. Como conseqüência, é importante explorar as tentativas de classificar e estruturar conceitos e suas relações dentro de um domínio, focalizando-se no compartilhamento e reúso destes conceitos.

Além disso, é importante explorar, no contexto deste trabalho, a utilização do padrão XML (*Extensible Markup Language*), para atribuir à linguagem UCL meios para uma fácil integração à Internet. Por isso a linguagem UCL será implementada no padrão XML.

1.5 Estrutura da dissertação

Esta dissertação está organizada de forma a apresentar o contexto teórico no qual este trabalho está inserido, bem como os resultados obtidos no desenvolvimento do trabalho e as suas contribuições para a comunidade científica interessada.

No **Capítulo 2**, é mostrado o conceito de agentes (de software), destacando-se algumas das definições encontradas na literatura e algumas propriedades dos agentes. Além disto, é mostrada a contextualização deste projeto, relacionado-o a outros trabalhos sobre comunicação entre agentes de software, ressaltando-se os princípios básicos para a comunicação entre agentes.

No **Capítulo 3**, é apresentado um meio para representar o domínio de conhecimento, especificamente as ontologias e sua importância, como um meio de classificar e estruturar os conceitos e suas relações. Além disso, inclui-se uma metodologia para construir ontologias.

No **Capítulo 4**, é apresentada a descrição de várias tecnologias que auxiliaram na especificação da linguagem UCL e a construção de um protótipo que utiliza a linguagem UCL. Entre estas tecnologias, pode-se mencionar a meta-linguagem XML, a linguagem *Universal Networking Language* (UNL) (Uchida, 1999), e uma outra para representar sentenças em linguagem natural na forma de conceitos usando-se a ferramenta *Thought Treasure*.

No **Capítulo 5**, é apresentada a especificação da linguagem proposta para a comunicação envolvendo agentes na Internet, que será designada UCL– *Universal Communication Language*. Incluem-se as considerações sobre a linguagem e suas características. Além disso, a metodologia que orientou o seu desenvolvimento e, finalmente, uma descrição completa da estrutura das mensagens especificadas nesta linguagem é mostrada.

No **Capítulo 6**, é apresentada uma descrição da implementação da linguagem UCL, tomando como base a especificação mostrada no capítulo 5. Apresentam-se também diagramas para explicar os vários processos usados para a criação das mensagens utilizando-se os recursos tecnológicos mostrados no capítulo 3 e 4.

Finalmente, o **Capítulo 7** apresenta as conclusões deste trabalho, considerando as decisões de projeto, contribuições do trabalho e sugestões para pesquisas futuras.

2. COMUNICAÇÃO ENTRE AGENTES DE SOFTWARE

2.1. Considerações Iniciais

Assim como é necessária a cooperação entre pessoas na sociedade humana para executar alguma tarefa, existe uma necessidade semelhante num sistema multi-agentes. Esta cooperação se dá pela comunicação (troca de mensagens) entre agentes com o objetivo de executar tarefas determinadas. Atualmente nesta área, várias pesquisas estão sendo realizadas e linguagens específicas para esse tipo de comunicação estão sendo desenvolvidas.

Este capítulo contém a definição de agentes de software e suas características, os princípios para a comunicação entre esses agentes e algumas linguagens de comunicação que constituem resultado de pesquisas envolvendo a comunicação entre agentes. Será incluída a apresentação de algumas características dessas linguagens, focalizando suas vantagens e desvantagens. Além disso, será apresentada uma comparação entre elas e analisadas as tendências de desenvolvimento.

2.2. Agentes de Software

O termo "agente" existe na linguagem dos computadores desde a década de 80, porém a definição do termo é tão complexa para a comunidade científica quanto a definição do termo "inteligência artificial". Dessa forma, o que tem ocorrido é que cada autor adopta uma definição do termo "agente" que melhor se adapte ao seu contexto de trabalho.

2.2.1. Definição de Agente

Iniciando pela definição encontrada nos dicionários, agente é aquele que opera; é aquele que é encarregado dos negócios de terceiros. Tais definições consideram duas perspectivas diferentes: a primeira que associa um agente a uma entidade que é capaz de agir; e a segunda

em que um agente é considerado como um ajudante, ou alguém que atua por intermédio de outra pessoa.

No contexto da ciência da computação, um agente de software pode ser utilizado para facilitar a criação de software capaz de interoperar, ou seja, trocar informações e serviços com outros programas e, dessa forma resolver problemas complexos (Moreira & Walczowski, 1997). Porém, pode-se observar algumas outras definições feitas por outros pesquisadores da área:

- Agentes de *software* são componentes de *software* capazes de comunicar e cooperar com seus pares por meio da troca de mensagens, usando uma linguagem de comunicação (Ketchpel & Genesereth, 1994).

- Agentes são sistemas computacionais que habitam algum ambiente complexo dinâmico, percebem e agem de maneira autónoma nesse ambiente e, assim, atingem objetivos ou cumprem tarefas para as quais foram designados (Maes, 1994).

- Agente é um *hardware* ou (mais usualmente) um sistema computacional baseado em *software* que tem características como autonomia, capacidade social, reatividade e proatividade (Jennings & Wooldridge, 1995).

O termo "agente" não tem uma definição consensual, talvez por cobrir diversas áreas de investigação e desenvolvimento. As várias definições de agentes fornecem uma lista de atributos para eles, o que não significa que todos os agentes tenham que, necessariamente, conter todos os atributos, os quais dependem do tipo de aplicação que está sendo desenvolvida (Gouveia et al., 1998). Segundo (Franklin & Graesser, 1996), as características encontradas na maioria dos agentes são:

- Autonomia: um agente será tão mais autônomo, quanto mais controle tiver sobre as suas ações. Um agente pode ser considerado autônomo em relação ao ambiente ou em relação a outros agentes.

- Pró-atividade: um agente pró-ativo toma a iniciativa para atingir os seus objetivos, não se limitando a responder a estímulos do ambiente.

- Reatividade: um agente tem capacidade de reagir às mudanças que sente no ambiente (estímulos).

- Continuidade Temporal: um agente está continuamente ativo. Nota-se que grande parte do *software* existente não tem essa característica, já que esses programas executam uma ou mais tarefas e terminam.

- Capacidade Social: se um agente tem capacidade social, então ele pode se comunicar com outros agentes, o que poderá incluir humanos. Dessa comunicação poderá resultar uma cooperação. Para o caso específico da comunicação entre o agente de software e o usuário humano deverá ocorrer uma cooperação na construção do "contrato" sobre o que o agente deverá fazer e não uma simples ordem.

- Capacidade de Adaptação: um agente com capacidade de adaptação é capaz de alterar seu comportamento com base na experiência. Esse tipo de agente é o chamado "agente inteligente", pois possui a capacidade de aprendizagem. A adaptação pode ser relativa ao ambiente ou no sentido de melhorar a sua interação com outros agentes.

- Mobilidade: corresponde à capacidade de o agente se mover dentro do ambiente. Quando se trata de agentes de software, um agente móvel é aquele capaz de se transportar de uma máquina para outra durante a sua execução.

- Flexibilidade: um agente com flexibilidade é aquele que não executa ações pré-definidas em roteiros. Ou seja, possui a capacidade de escolher dinamicamente seqüências de ações em resposta a um estado do ambiente.

Para (Franklin & Graesser, 1996), autonomia, pró-atividade, reatividade e continuidade temporal são características essenciais em um agente.

2.3. Princípios para comunicação entre agentes

Na busca por uma linguagem de comunicação entre agentes (*Agent Communication Language* – ACL), houve várias iniciativas e trabalhos pioneiros dos grupos de pesquisa *Knowledge Sharing Effort* e da *Fundation for Intelligent Physical Agents* (FIPA) que têm formulado fundamentos

teóricos para que agentes possam interagir independentemente do ambiente de implementação (Mamadou et al., 2000).

O projeto das linguagens de comunicação para agentes (ACLs) evoluiu em torno de vários princípios, tais como heterogeneidade, cooperação e coordenação, separação, interoperabilidade, transparência, escalabilidade, estensibilidade, e desempenho. Uma linguagem de comunicação para agentes pode ser importante para um grande número de áreas de aplicação se estes princípios são observados. A generalização de um *framework* de uma ACL pode ser caracterizada pelos seguintes princípios (Mamadou, 2000):

- O princípio de heterogeneidade: Afirma que o agente deve ser capaz de se comunicar independentemente de seu ambiente de implementação. Nessa comunicação as mensagens trocadas devem refletir uma perspectiva global em lugar de uma perspectiva privada do agente emissor ou receptor.

- Os princípios de cooperação e coordenação: Afirmam que, para que uma cooperação seja efetiva e tenha o propósito de resolver uma tarefa complexa, é necessária uma linguagem de comunicação significativa. Geralmente, a comunicação entre agentes inclui transportar e trocar informação sobre o conhecimento dos agentes ou de seus ambientes. Neste sentido, uma ACL deve dar aos agentes os meios através dos quais eles possam se coordenar para alcançar seus objetivos.

- O princípio de separação: Afirma que o conteúdo, a estrutura, e o mecanismo de transporte de uma mensagem constituem entidades distintas que devem ser tratadas separadamente.

- O princípio de interoperabilidade: Afirma que uma ACL deve fornecer aos agentes meios para interoperar. A integração de software baseada em agentes foi concebida para permitir que componentes de software heterogêneos, modelados como agentes, possam interoperar usando uma linguagem apropriada.

- O princípio de transparência: Afirma que um sistema multi-agente deve ser protegido das complexidades existentes na especificação de uma ACL. Neste sentido, uma API (*Application Programming Interface*) adequada a uma ACL libera os agentes dos detalhes específicos e determina as interações num nível mais alto.

- Os princípios de escalabilidade e estensibilidade: Afirmam que os projetistas de ACLs podem adicionar novos atos comunicativos compatíveis com aqueles existentes. Estes novos atos podem implementar novos protocolos de interação. Além disso, o projeto de uma ACL deve considerar escalabilidade em relação ao crescimento do número de agentes em um sistema multi-agente.

- O princípio de desempenho: Afirma que uma implementação de ACL deve usar eficientemente os recursos do sistema (CPU, memória e largura de banda). Os atos primitivos de comunicação fornecidos pela linguagem devem ser compatíveis com a tecnologia de rede usada e exibir capacidades de conexão *unicast, multicast* , síncrona e assíncrona. Além disso, uma ACL deve suportar uma troca de mensagens confiável e segura entre agentes.

2.4. Especificações de uma linguagem de comunicação para agentes

A especificação de uma linguagem preocupa-se com a descrição da estrutura da mensagem, seu modelo semântico e os protocolos de interação em que se apóia (Mamadou, 2000):

- O formato da mensagem define os atos de comunicação (*communicative acts*) primitivos e os parâmetros da mensagem (como *sender*, *receiver*, etc.). O conteúdo da mensagem descreve fatos, ações, ou objetos em uma linguagem de conteúdo (KIF, Prolog, etc). Outros parâmetros podem cuidar do significado da mensagem (ontologia) e sua entrega.

- O modelo semântico de uma ACL estabelece os fundamentos para obter um significado conciso e não ambíguo das mensagens do agente e dos protocolos de interação.

- Os protocolos de interação são conjuntos de padrões bem definidos projetados para facilitar a comunicação entre agentes. Protocolos são opcionais, mas, caso sejam usados, a comunicação entre agentes deve ser consistente com o protocolo escolhido. Os seguintes protocolos de comunicação podem ser usados no projeto de ACLs:

- Protocolo de comunicação direta: Este tipo de protocolo é aplicado quando um agente emissor conhece o agente receptor e suas capacidades.

- Protocolo *Contract Net*: Projetado originalmente por Davies e Smith em (Davis & Smith, 1983), este protocolo define, em sua forma genérica, padrões de interação entre um agente (*the manager*) com outros agentes (*contractors*) para executar alguma tarefa complexa. Os agentes *Contractors* submetem propostas de tarefas ao agente *manager*, que as avalia e atribui seguindo algumas condições. Os *contractors,* que tiveram suas propostas aceitas, são responsáveis por executar a tarefa atribuída e enviar de volta o resultado ao *manager*.

- Protocolo de comunicação mediado: Este protocolo utiliza os serviços de agentes especiais (denominados *facilitators*) que atuam como *brokers* entre agentes necessitando de algum serviço e agentes que forneçam esse serviço. A mediação envolve agentes fazendo pedidos de serviço e facilitadores negociando, recrutado e recomendando agentes, que registraram suas identidades e capacidades, para executar esses serviços.

2.5. A linguagem de Comunicação KQML

Knowledge Query and Manipulation Language (KQML) é uma linguagem versátil de propósito geral que suporta a comunicação entre vários agentes com um conjunto de primitivas reservadas chamadas *performatives*. A linguagem KQML adotou o termo *performatives* para designar as primitivas das mensagens. A definição deste termo segundo (Labrou et. al., 1999) se refere a uma declaração feita pelo usuário emissor e que é executada pelo agente receptor, simplesmente por que o emissor a declara ou afirma.

A KQML, descrita por Tim Finin em (Finin et al., 1993), é o resultado de pesquisas feitas pelo grupo *Knowledge Sharing Effort (*KSE*)* (ARPA, 1993), cuja iniciativa teve por objetivo desenvolver fundamentos para interação e interoperabilidade em sistemas de software. Três subgrupos de trabalho com objetivos complementares compõem o KSE:

- O grupo de Interlíngua, que projetou o *Knowledge Interchange Format* (KIF) como uma linguagem comum para descrever o conteúdo de mensagens.

- O grupo *Shared and Reusable Knowledge Base* (SRKB), que se interessa pelo conteúdo de bases de conhecimento compartilhadas.

- O grupo *External Interface,* que produziu a linguagem KQML e considerou as interações dos componentes do sistema em tempo de execução.

Em KQML, as mensagens são expressas por meio de *performatives* e elas definem as operações que um agente pode conduzir. O conteúdo de uma mensagem KQML pode ser descrito em várias linguagens, inclusive KIF, desde que a linguagem escolhida possa expressar os *performatives* de maneira adequada. A lista completa dos *performatives* reservados e seus significados estão ilustrados na **Tabela 2.1**. Nesta tabela *S* representa o agente que envia uma mensagem de pedido (emissor) e *R* é o agente que recebe os pedidos de outros agentes (receptor).

Tabela 2.1. *Performatives* reservadas de *KQML*.

S: agente emissor, R: agente receptor.

Categoria	Nome	Significado
Discurso	Ask-if	S quer saber se um conteúdo está na base de conhecimento de R.
	Ask-one, ask-all	S quer perguntar a um ou a todos o conteúdo de algo.
	stream-all	Versão múltipla de ask-all.
	Eos	End-of-stream, marcador de fim de múltipla resposta (stream-all).
	Tell, untell	uer informar a R sobre uma sentença que está ou não na base de conhecimento.
	Deny	S quer informar a R que uma sentença não está na base de conhecimento.
	Insert, uninsert	S solicita que R adicione um conteúdo na base de conhecimento.
	Delete-one, Delete-all, undelete	S quer que R apague um ou mais sentenças da sua base de conhecimento.
	Achieve, unachieve	S quer de R alguma coisa.
	Advertive, Unadvertive	S informa a R de sua capacidade e disposição para processar conteúdo.

Categoria	Nome	Significado
Intervenção e Mecanismos	Subscribe	S quer atualizações regulares sobre *performatives* do facilitador.
	Error	S considera as mensagens prévias de R como mal formadas.
	Sorry	S entende as mensagens de R, mas não fornece ajuda.
	Standby	S quer anunciar a R que está pronto para fornecer respostas a suas mensagens
	Ready	S está pronto para responder as mensagens prévias de R
	Next	S quer a resposta seguinte de R à mensagem previamente enviada.
	Rest	S quer o que falta de R
	Discard	S ignora qualquer resposta restante de uma mensagem multi-resposta

Categoria	Nome	Significado
Facilitação e Rede	Register, unregister	S anuncia ao facilitador que quer fazer ou desfazer uma inscrição.
	Forward	S quer do facilitador uma transferência da mensagem a outro agente
	Broadcast	S quer do facilitador o envio de uma mensagem a todos os agentes conhecidos.
	Transport-address	S associa seu nome a um novo endereço de transporte.
	Broker-one, broker-all	S solicita ao facilitador para encontrar um agente disponível que forneça uma ou mais respostas a uma *performative*.
	Recomended-one, Recomended-all	S quer do facilitador recomendações sobre os agentes que estão disponíveis para executar uma *performative*.
	Recruit-one, recruit-all	S quer que facilitador encontre um agente adequado disponível para responder a uma *performative*.

Para mostrar como são utilizadas as *performatives* apresentadas na **Tabela** 2.1, é mostrado a seguir um exemplo de um simples cenário de conversação em *KQML* a respeito de um "*call for papers*" para uma conferência, no qual *C* é o agente que preside a conferência, *R* é o agente receptor e *A* é um agente relacionado com *R*. Ao presidir a conferência, *C* faz um *broadcast* para os agentes interessados no "*call for papers*", tais como *R*, que eventualmente transfere a mensagem para quem está relacionado com ele, como *A*:

```
(broadcast
      :sender C
      :receiver R
      :reply-with id0
      :language KQML
      :ontology kqml-ontology
      :content (tell
                  :sender C
                  :receiver A
                  :reply-with id0
                  :language Prolog
                  :ontology conference
                  :content "call(papers, conf)"
               )
)
```

Após o "*call for papers*" para a conferência, o agente *A* sugere seu *paper* para o agente *C*, que preside a conferência.

```
(tell
      :sender A
      :receiver C
      :in-reply-to id0
      :language Prolog
      :reply-with id1
      :ontology conference
      :content "submit(paper, conf)"
)
```

Finalmente, depois da revisão de todos os *papers* sugeridos, o agente *C* informa ao agente *A* o resultado.

```
(tell
     :sender C
     :receiver A
     :language Prolog
     :in-reply-to id1
     :ontology conference
     :content "accepted(paper, conf)"
)
```

Como pode ser observado na **Figura 2.1**, a linguagem KQML é composta por três camadas:

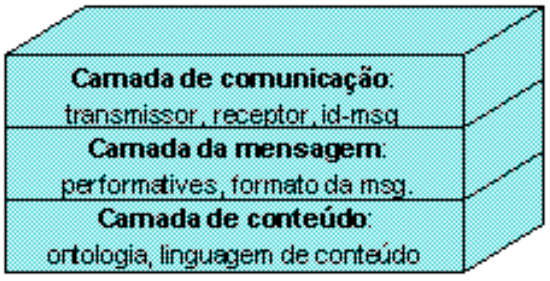


Figura 2.1. Estrutura das três camadas *KQML*

Camada de conteúdo: Esta camada abriga uma linguagem de conteúdo, tal como KIF ou Prolog. Para que dois ou mais agentes se comuniquem precisam concordar na utilização da mesma linguagem de conteúdo.

Camada de mensagem: esta camada é utilizada para codificar uma mensagem de uma aplicação para outra. Esta mensagem pode ser de dois tipos: *content message* ou *declaration message*. O primeiro tipo faz uma descrição do conhecimento que está representado na

camada de conteúdo. O segundo tipo serve para anunciar a presença de um agente no sistema, ou fornecer uma descrição geral da informação que o agente enviará ou receberá.

Camada de comunicação: é através desta camada que os agentes trocam pacotes. Estes pacotes são como um invólucro da mensagem, e especificam alguns atributos da comunicação tais como a especificação do emissor e do receptor.

Em KQML, há principalmente dois tipos de comunicação entre agentes: comunicação direta e mediada. No primeiro tipo, um agente conhece exatamente qual agente pode processar seu pedido. Assim, ele envia seu pedido diretamente a este agente. No segundo tipo, um agente faz pedidos para um agente especial chamado facilitador *(facilitator)*. A função deste é coordenar as interações dos agentes envolvidos em um problema particular, que pode ser resolvido através de:

- Transferência de pedidos aos agentes apropriados.
- *Brokering*, convocação e recomendação de um agente adequado para outro agente.
- Iniciar a resolução distribuída de um problema envolvendo agentes.

Presume-se que todos os agentes interessados informaram suas identidades, interesses e capacidades ao facilitador. Num ambiente de comunicação KQML, um *K-router* cuida da chegada das mensagens que serão futuramente processadas pela API *Router Interface Library* (KRIL) do KQML.

O número de aplicações da linguagem *KQML* varia da engenharia de *hardware* e sistemas de software a sistemas de banco de dados e experimentos na integração tecnológica. Uma das aplicações promissoras é a integração de software baseado em agentes (*Agent-Based software Integration* - ABSE) mostrado em (Ketchpel & Genesereth, 1994), em que a integração e interoperação entre componentes de software são alcançadas com agentes facilitadores.

2.5.1. Vantagens e limitações da linguagem KQML.

A principal vantagem da linguagem KQML é a sua capacidade para suportar várias arquiteturas diferentes de agentes com seu conjunto estensível de *performatives*. Como resultado, KQML tornou-se uma linguagem padrão para comunicação entre agentes em diferentes áreas de aplicação. No entanto, a sua primeira versão teve algumas críticas como

pode ser observado em (Cohen & Levesque, 1995), que apontam para uma confusão no uso destas *performatives*. De certa forma, devido a esta fraqueza, vários dialetos de KQML surgiram na indústria e foram usados em diferentes projetos de agentes de software que não podem interoperar. Felizmente, uma nova especificação não oficial (Labrou & Finin, 1997a) está melhorando a semântica e os conjuntos de *performatives*.

2.6. A linguagem AGENT-0

A linguagem AGENT-0 foi projetada por Torrance (Torrance, 1998), com o propósito de especificar agentes de software. Esta linguagem é baseada nos princípios definidos por Yoav Shoham (Shoham, 1993) que os introduziu como um novo paradigma de programação, chamada programação orientada a agente (*Agent Oriented Programming* - AOP). Este paradigma, descrito em (Shoham, 1993), é um *framework* computacional cujos conceitos são baseados em inteligência artificial (AI), *speech act theory* (Cohen & Levesque, 1990b), e programação orientada a objeto. De acordo com Shoham, neste *framework* um agente é uma entidade cujo estado é formado por componentes "mentais" tais como crenças, capacidades, escolhas e compromissos (Shoham, 1993). A definição de uma linguagem baseada em AOP deve ser formal e concisa, para que ela possa expressar satisfatoriamente os atos comunicativos e "estados mentais" do agente.

Nesta linguagem há somente três primitivas baseadas nos atos de comunicação: *inform*, *request*, e *unrequest*, que podem ser combinados para expressar uma variedade de ações.

2.6.1. Vantagens e limitações da linguagem AGENT-0

A idéia de atribuir atitudes mentais aos agentes de software na linguagem AGENT-0 é realmente único e atrativo. No entanto, o número pequeno (somente três) de primitivas de atos de comunicação disponíveis, faz com que esta linguagem se torne imprópria para a maioria das aplicações do mundo real. Conforme David Parks em (Parks, 1997), a base teórica da AOP ainda é uma área pouco desenvolvida porque não está direcionada a assuntos importantes como:

- Descrição formal dos fundamentos da teoria dos estados mentais.
- Segurança em redes heterogêneas.

Além disso, AGENT-0 precisa de uma linguagem de implementação mais expressiva para transformar-se em um paradigma de programação completo (Parks, 1997).

2.7. Linguagens de comunicação para Agentes Móveis

Um agente móvel é um programa que pode migrar de forma autônoma através de uma rede de computadores heterogênea. Em qualquer momento ele pode parar, mover-se juntamente com seu estado e dados para uma nova máquina, e reiniciar a execução no ponto em que parou. Agentes móveis são um novo tipo de abstração na comunicação cliente/servidor. O mecanismo de comunicação mais comum usado por agentes móveis é passar mensagens (pedidos ou consultas). Essa passagem de mensagens pode ser feita por vários métodos, como *Remote Procedural Calls (RPC)*, *Remoted Method Invocation (RMI)* ou através de *Object Request Brokers (ORBs)*. A tecnologia de agentes móveis tem encontrado muitas aplicações interessantes em sistemas multimídia distribuídos, sistemas de tempo real, computação móvel, automatização de fábricas e sistemas de missão crítica (Mamadou, 2000).

Ao contrário da proposta sobre os estados mentais, a tecnologia de agentes móveis não faz necessariamente uma suposição a respeito de estados mentais e nem da estrutura social dos agentes. Aqui, agentes são somente processos que são capazes de se mover de forma autônoma e executar tarefas em lugares remotos. Esta propriedade leva a uma diferença na forma como a comunicação entre agentes é definida, como mostrado nos dois exemplos seguintes.

2.7.1. O tipo de comunicação no sistema Telescript

Telescript é um sistema de agentes móveis concebido pela General Magic (White, 1994) para introduzir o paradigma de *Remote Programming (RP)*. A arquitetura do sistema Telescript, ilustrado na **Figura 2.2**, possibilita que os usuários deleguem tarefas a um agente e que este as executem em locais onde os serviços estão disponíveis. A tecnologia Telescript se apóia nos conceitos de lugar, agente, viagem e reunião.

Os agentes se reúnem em servidores ou clientes (lugares) que oferece serviços para os agentes móveis que estão chegando através da rede. A comunicação entre agentes acontece quando eles se reúnem num lugar ou por meio de instruções de conexão, quando estão em lugares diferentes. Esta comunicação pode ser implementada por meio de agentes que chamam

procedimentos um dos outros. Na parte central do sistema Telescript, está o *Telescript engine*, um interpretador que fornece um ambiente para execução para os agentes.

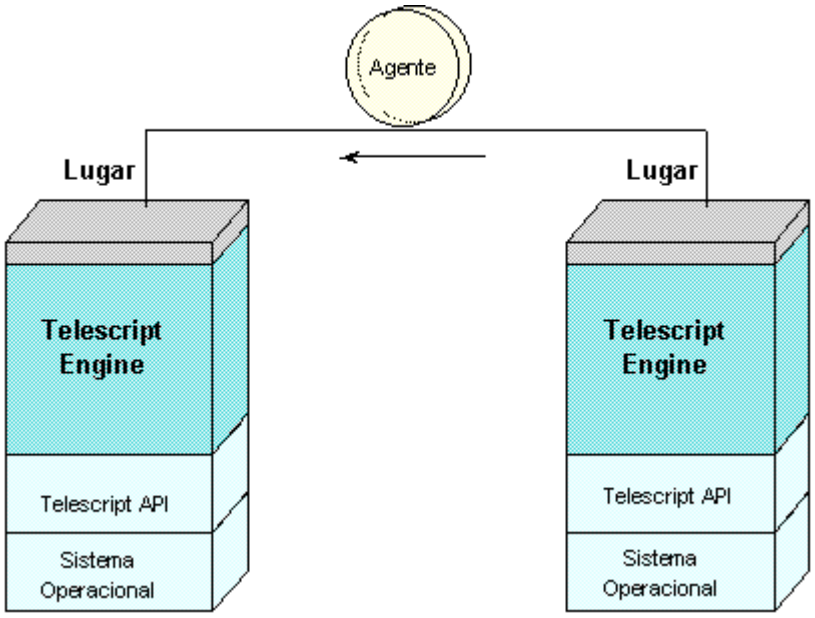


Figura 2.2. Arquitetura do sistema Telescript.

2.7.2. O tipo de comunicação do sistema Aglets

Os Aglets (*Agile applets*) são objetos Java projetados no centro de pesquisa da IBM em Tokyo (Mamadou et al., 2000), que podem migrar, junto com seu código e estado, de forma autônoma dentro de uma rede de computadores e que executar algum trabalho em benefício do seu proprietário. Os agentes se comunicam por meio de *message passing* (*local host*) ou *remote message passing* (*remote host*) como mostra a **Figura 2.3**.

Essas mensagens são objetos (*message objects*) que são serializados, organizados e enviados, para depois serem armazenados em uma fila de mensagens. Estas mensagens podem pertencer a um dos seguintes tipos:

- Mensagem *Now-type*, que é síncrona e requer processamento completo pelo receptor;
- Mensagem *Future-Type* , que é assíncrona;
- Mensagem *Oneway-Type*, que é também assíncrona, e tratada sempre como última na fila.

Uma mensagem recebida por um agente é tratada por um método Aglet (*Aglet.handleMessage*) antes do seu processamento. Além disso, mensagens suportam respostas de confirmação. Os princípios da programação de agentes móveis Java são explicados pelos projetistas desta tecnologia em (Lange & Mitsuru, 1998).

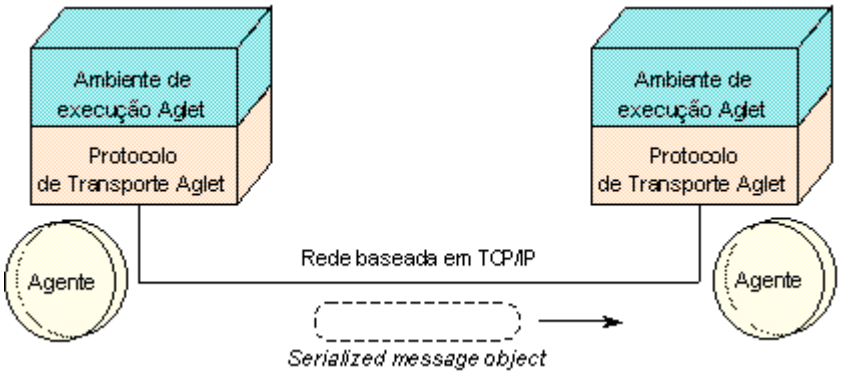


Figura 2.3. A Arquitetura Aglet e modelo de comunicação.

2.7.3. Vantagens e limitações da comunicação entre Agentes Móveis

Agentes que são capazes de migrar certamente oferecem uma alternativa atrativa ao estilo de comunicação tipo RPC entre *hosts*. No entanto, a comunicação interagente apresentada é limitada a pedidos simples. Devido às características dos agentes móveis, não há muito intercâmbio de informação nem conhecimento de domínio. O propósito da comunicação aqui não é buscar a cooperação de outros agentes para resolver uma tarefa, mas, de preferência, fazer uso dos serviços fornecidos no *host* destino. Como não existe o conceito de comunidade de agentes, pode-se questionar se os agentes móveis realmente se comunicam.

2.8. A linguagem FIPA-ACL

A *Foundation for Intelligent Physical Agents* (FIPA) é uma organização internacional que tem por objetivo desenvolver um conjunto de agentes padronizados e genéricos com a contribuição de todos os participantes envolvidos no desenvolvimento da tecnologia de agentes. Em particular, o padrão FIPA-ACL tenta identificar os componentes da comunicação e cooperação interagente, pela definição de uma linguagem com semântica concisa, formal e com suporte a protocolos de comunicação. De fato, a principal especificação padrão FIPA (FIPA, 1999) é composta por seis sub-especificações: administração de agentes; comunicação

entre agentes; interação entre agentes; assistência pessoal para viagem, entretenimento e *broadcasting* audiovisual; e administração.

O centro da especificação FIPA para comunicação entre agentes – que inclui primitivas de comunicação (*performatives*), um modelo formal, e uma linguagem de conteúdo (*Semantic Language* SL) – foi extraído da linguagem ARCOL (Sadek et al., 1997) que foi projetada pela France Télecom, enquanto que a *KQML* inspirou a estrutura de suas mensagens.

O cenário simples relatado na seção 2.4, *call for papers*, para chamada de *papers* para uma conferência é expresso da seguinte forma na linguagem FIPA-ACL:

Primeiro, o agente C *- que preside a conferência - envia (*cfp*) uma chamada de* papers *a um agente* R *interessado.*

```
(cfp
      :sender C
      :receiver R
      :reply-with call-proposal
      :language sl
      :ontology conference
      :protocol FIPA-Contract-Net
      :content ((action R (submit (paper, conf)) ) true)
)
```

O agente R *submete seu* paper *para revisão com o agente* C.

```
(propose
      : sender R
      :receiver C
      :in-reply-to call-proposal
      :reply-with proposal-R
      :language sl
      :ontology conference
      :content ( action ( submit(paper, conf)))
)
```

Finalmente, o agente C *informa (*accept-proposal*) ao agente* R *que seu* paper *foi aceito para a conferência.*

```
(accept-proposal
     :sender C
     :receiver R
     :in-reply-to proposal-R
     :language Prolog
     :ontology conference
     :content "(accepted(R, paper, conf))"
)
```

2.8.1. Vantagens e limitações da linguagem FIPA-ACL

A FIPA-ACL é uma linguagem de comunicação para agentes que envolve muitas das particularidades da indústria e da academia. FIPA-ACL se baseia em componentes práticos de comunicação e cooperação interagente com uma semântica formal definida.

Estas características a tornam um ponto de referência. No entanto, ao contrário de outros modelos ACL, a FIPA-ACL sofre da falta de aplicações práticas que poderiam apontar suas desvantagens ou limitações.

2.9. Uma proposta para uma ACL com ênfase no ambiente social.

A comunicação em sistemas convencionais multi-agentes baseados em atitudes mentais, sofre da falta de uma semântica formal concisa e universalmente aceita. Como resultado, a comunicação entre agentes está limitada a um domínio de ambiente restrito e os agentes heterogêneos não conseguem interagir adequadamente.

Como agentes de software são geralmente definidos em termos de autonomia, interoperabilidade, cooperação e capacidade para coordenar ações em direção a um objetivo comum, as atitudes mentais podem não ajudar a definir uma semântica formal e concisa. Várias soluções atrativas para estas desvantagens foram propostas por Singh (1998), que defende um modelo semântico formal que enfatiza a *social agency*.

Singh propõe uma teoria que fornece um fundamento conciso para o projeto de sistemas multi-agentes. Nesta teoria, ele mostra que o sucesso de um sistema multi-agente depende de quão bem a ACL suporta a interação entre agentes em um ambiente social. Na **Figura 2.4** apresenta-se o espaço de projeto das ACLs segundo (Singh, 1998). A região à esquerda

representa as ACLs existentes que seguem um modelo de agência mental (*mental agency model*). A região superior direita representa os objetivos a serem alcançados, os quais seguem o modelo de agencia social (*social agency model*) onde se preferem as seguintes características: uma alta cobertura (incluir todas as categorias significativas de atos de comunicação), contexto flexível, base semântica para o significado, tipo de significado convencional e uma perspectiva pública.

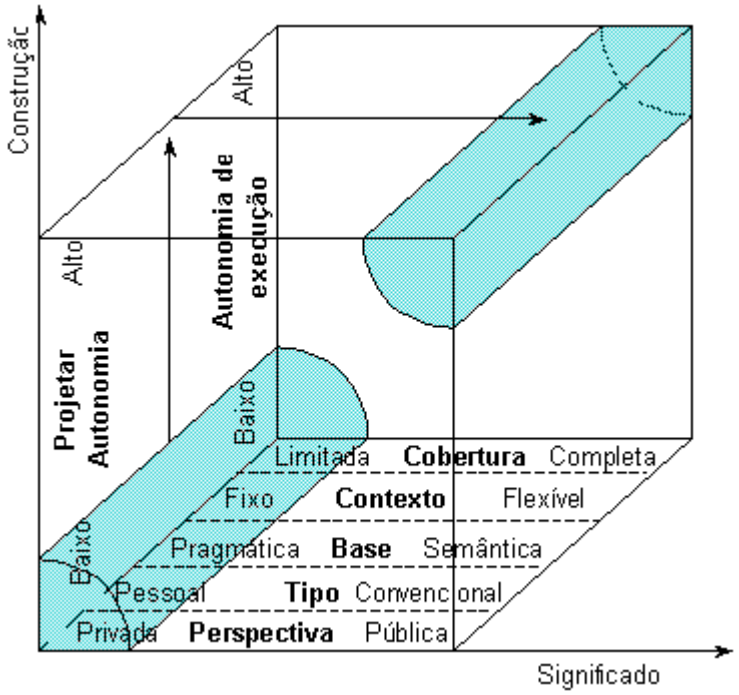


Figura 2.4. Espaço de projeto das linguagens de comunicação de agente(© 1998 IEEE). Procedência da figura: *"Agent Communication Languages: Rethinking the Principles" by Munindar P. Singh in IEEE Computer, volume 31, numer 12, December 1998, pages 40-47*.

Singh sugere que muitos elementos descritos na **Figura 2.4** contribuem para a significância da comunicação entre agentes (Singh, 1998) e elas podem ter as seguintes características:

- Perspectiva deve ser privada (se nos referimos ao transmissor) ou pública (se nos referimos a uma sociedade de agentes).

- Tipo de significado do conteúdo da linguagem é pessoal (estabelecido por um agente) ou convencional (estabelecido por um grupo de agentes).

- O Contexto deve ser flexível para tomar a comunicação mais significativa.

- Cobertura: Como os atos comunicativos representam o significado, quanto maior sua cobertura (incluindo todas as categorias do *Speech Act theory*) melhor será a interação em um sistema multi-agente (Cohen & Levesque, 1990b). Além disso, Singh introduz em (Singh, 1999) uma semântica social para ACL baseado em compromissos sociais da lógica temporal.

2.9.1. Vantagens e limitações da proposta.

A dimensão social dos sistemas multi-agentes é uma valiosa propriedade para o projeto e administração de grande parte destes sistemas. O fundamento teórico do modelo *social agency* estabelecido por Singh propõe o novo campo de integração de software baseado em agentes.

Esta proposta é uma macro teoria que pode ser difícil de implementar em ambientes do mundo real, porque os modelos computacionais impõem restrições que o modelo *social agency* não poderia definir (Mamadou, 2000).

2.10. Comparação entre as linguagens ACL

A falta de uma semântica formal e de um formato para as linguagens ACLs aceitos universalmente torna difícil fazer uma comparação em termos dos princípios mostrados na seção 2.1. No entanto, algumas similaridades e diferenças existem entre as ACLs, ilustradas na **Tabela 2.2**.

Por exemplo, FIPA-ACL e KQML têm sintaxe similar, formato de mensagem idêntico e compartilham vários parâmetros. Em KQML, as primitivas dos atos comunicativos (Communicative Acts - CAs) apresentam uma dificuldade na descrição das atitudes mentais reais na mensagem do transmissor, enquanto que a FIPA-ACL insere algumas dessas propriedades na expressão do conteúdo da mensagem.

Alguns críticos, tais como Cohen e Levesque, argumentam (Cohen & Levesque, 1995) que a semântica de KQML é mal definida, suas *performatives* (primitivas de comunicação) são ambíguas e não têm uma cobertura completa de todas as interações na comunicação entre agentes.

Por outro lado, os projetistas da KQML melhoraram sua semântica e forneceram uma fundamentação melhor para muitas aplicações que usam esta linguagem. Embora a linguagem FIPA-ACL esteja causando muito interesse, Yannis Labrou e Tim Finin declararam (Labrou et. al., 1999): "Quando surgiu o interesse pela integração dos sistemas multi-agentes por intermédio da comunicação, a linguagem KQML teve um papel pioneiro na definição do que seria uma linguagem de comunicação entre agentes, e de quais características seriam necessárias".

Tabela 2.2. Comparação das linguagens ACL (Mamadou, 2000).

Elementos ACL **Linguagens ACL**	**Formato da mensagem**	**Conteúdo Sintático e Linguagem**	**Conteúdo semântico**
KQML	Estrutura em camadas; separação entre conteúdo, mensagem e comunicação.	*Knowledge Interchange Format* (KIF) Linguagem de conteúdo não predefinida	Semântica informal
Agent-0	Atitudes mentais e regras de compromisso; lógica como base.	Linguagem não definida	Sem semântica
Agentes móveis	*Message object*	Linguagem não definida	Sem semântica
FIPA	Estrutura em camadas; Separação entre conteúdo, mensagem, e comunicação.	Linguagem SL	Semântica formal baseada em lógica
Proposta *Social Agency*	Estrutura social; agência social ou comunidade social.	Linguagem não predefinida	Semântica formal com perspectiva pública e crenças comuns

Por outro lado, a proposta teórica baseada na dimensão social da interação entre agentes, a *Social Agency* de Singh (1998), parece complementar as linguagens anteriores.

Em engenharia de software, agentes móveis têm necessidades diferentes a respeito da comunicação em sistemas multi-agentes. Em sistemas baseados em agentes móveis, a comunicação está intimamente relacionada ao protocolo de transporte, e principalmente, a

forma dos pedidos de informação. Nenhuma suposição é feita a respeito dos estados mentais no agente móvel.

Para que as linguagens de comunicação entre agentes sejam de importância tanto na indústria como na área acadêmica, os seguintes assuntos teóricos e práticos devem ser resolvidos (Mamadou, 2000):

- Consenso sobre a semântica parece ser o problema mais importante enfrentado pela comunidade de pesquisadores.

- Compartilhamento de ontologias e o conteúdo da linguagem são uma condição prévia para compartilhar conhecimento, porque a habilidade para trocar mensagens não pressupõe o entendimento sobre o conteúdo das mensagens.

- Suporte para a verdadeira heterogeneidade: agentes de diferentes ambientes ainda não se comunicam.

- O gerenciamento da conversação em sistemas multi-agentes deve incluir regras, semântica e protocolos de interação apropriados.

- Testes de conformidade são essenciais para que as ACLs sejam consideradas em concordância com algum padrão. Estes testes assegurarão que diferentes implementações de ACLs permitam aos agentes se comunicarem.

2.11 Considerações Finais

A pesquisa nas linguagens de comunicação para agentes (ACLs) está tendo um maior destaque devido às iniciativas dos grupos de pesquisa nesta área, como o *Knowledge Sharing Effort* (KSE) e a *Foundation for Intelligence Physical Agent* (FIPA). Sistemas multi-agentes e suas ACLs relacionadas são áreas de pesquisa que estão florescendo e novas perspectivas, como a interação social entre agentes, vão trazer ainda mais mudanças nessas áreas.

Neste capítulo foi apresentada uma visão global sobre as linguagens de comunicação entre agentes (ACL) que estão surgindo e esforços de padronização que estão acontecendo nesta

área. Além disso, foram apresentadas similaridades e diferenças entre estas linguagens e uma introdução sobre alguns assuntos atuais no projeto de uma ACL.

Neste contexto podem-se notar várias preocupações no momento de projetar uma linguagem de comunicação para agentes de software. Este trabalho focará especial atenção aos seguintes assuntos:

- Um consenso na semântica, que é um problema importante para o entendimento da mensagem a ser transmitida, com um significado conciso e não ambíguo.

- Compartilhamento do conhecimento sobre um assunto entre os agentes, e como pré-requisito para isto, uso de conceitos para representar senso comum (ontologia). A preocupação estará também em como é representado o conteúdo da linguagem, porque a habilidade para trocar mensagens não assume o entendimento do conteúdo da mensagem. No capítulo seguinte será mostrada mais informação sobre este assunto.

Neste ponto, é importante ressaltar que este trabalho leva em consideração os assuntos acima para projetar uma linguagem de comunicação para agentes de software.

3. ONTOLOGIAS PARA AGENTES

3.1 Considerações iniciais

Na comunicação entre agentes de software é imprescindível um entendimento adequado na troca de mensagens. É aí que uma representação do domínio de conhecimento pode colaborar para um melhor entendimento do contexto e, dessa maneira, coordenar as prováveis soluções de um determinado problema.

Este capítulo trata sobre ontologias, que são uma tentativa de classificar e estruturar conceitos e suas relações, focalizando-se aqui o compartilhamento e reúso destas estruturas, assim como critérios para projetá-las. Além disso, esse capítulo apresenta também a importância do uso desses conceitos, uma metodologia genérica para construir ontologias e as diferentes ferramentas existentes para o apoio a construção destas ontologias.

3.2 Definição e Aplicação de Ontologias

Pessoas, organizações e sistemas de software devem se comunicar. Porém, devido às necessidades e aos contextos diferentes, pode se notar vários pontos de vista e suposições relacionados a um mesmo assunto. A existência de conjuntos de palavras específicas para cada contexto ou assunto de uma área cria discrepâncias, justaposições e/ou conceitos mal compreendidos, em estruturas e métodos. Como conseqüência, surge a falta de um entendimento comum. Isto conduz a (Ushold & Gruninger, 1996):

1. Uma comunicação pobre, dentro e entre estas pessoas e/ou organizações.

2. Dificuldades na identificação de requisitos e deste modo na definição da especificação do sistema.

3. Restringem a interoperabilidade;

4. Restringem o potencial para reutilizar e compartilhar informação.

O segundo ponto refere-se ao contexto de construção de sistemas de tecnologia de informação (TI). O terceiro e quarto ponto se referem a discrepâncias no modelamento dos métodos, paradigmas, linguagens e ferramentas de software.

Um caminho para resolver estes problemas é reduzir ou eliminar as confusões conceituais e terminológicas e conseguir uma compreensão comum. Esta compreensão pode funcionar como um *framework* unificador para os diferentes pontos de vista e serve de base para (Ushold & Gruninger, 1996):

- **Comunicação** entre pessoas com diferentes necessidades e pontos de vista que surgem em seus diferentes contextos.

- **Inter-Operabilidade** entre sistemas que interagem com outros sistemas que usam diferentes métodos de modelamento, paradigmas, linguagens e ferramentas de software.

- **Benefícios na Engenharia de Sistemas:** em particular,

    - Reutilização: A compreensão compartilhada é a base para a especificação formal de entidades importantes, atributos, processos e suas inter-relações no domínio de interesse. Esta representação formal pode ser a alavanca para re-utilização de componentes compartilhados em um sistema de software.

    - Confiabilidade: Uma representação formal também torna possível uma automatização na verificação de resultados em um software confiável.

    - Especificação: a compreensão compartilhada pode ajudar ao processo de identificação de requisitos e definição de uma especificação para um sistema de tecnologia de informação. Isto é especialmente vantajoso quando os requisitos envolvem diferentes grupos usando diferentes terminologias no mesmo domínio, ou múltiplos domínios.

Alguns exemplos práticos, na unificação de áreas de pesquisa, na fabricação de semicondutores, e na missão de operações de astronaves, são descritos em (Ushold & Gruninger, 1996) como uma amostra das utilidades das ontologias.

3.2.2 Definição de Ontologia

'Ontologia' é um termo usado para se referir ao senso comum de algum domínio de interesse. A ontologia pode ser usada como um *framework* unificador para resolver os problemas apresentados anteriormente (Ushold & Gruninger, 1996).

Uma ontologia necessariamente vincula ou inclui algum tipo de "visão geral" referente a um domínio determinado. Esta "visão geral" é freqüentemente concebida como um conjunto conceitos (por exemplo: entidades, atributos, processos), suas definições e suas inter-relações. Isto é chamada uma conceitualização (*conceptualisation*).

Uma conceitualização pode estar concretamente implementada, como por exemplo, em um componente de software, ou pode ser abstrata, definida em linguagem natural de uma forma não formal. A palavra 'Ontologia' é algumas vezes usada para chamar isto de uma conceitualização implícita. Porém, o uso mais padronizado e que será adotado neste documento é que ontologia é uma idéia explícita, ou uma representação (de alguma parte) de uma conceitualização.

3.2.3 Características de uma ontologia

Uma ontologia explícita pode tomar uma variedade de formas, mas necessariamente ela incluirá um vocabulário de termos e alguma especificação dos seus significados (por exemplo: definições).

A maneira como estes vocabulários são criados e especificados tem graus de formalidade que variam consideravelmente. Esta variação pode ser mostrada nos seguintes quatro pontos de vista: (Ushold & Gruninger, 1996).

- Altamente informal: expresso livremente em linguagem natural.

- Semi-informal: expresso em uma forma restrita e estruturada em linguagem natural. Maior clareza pela redução de ambigüidade (Fraser et. al., 1995).

- Semiformal: expressa em uma linguagem artificial definida formalmente.

- Rigorosamente formal: termos meticulosamente definidos com uma semântica formal, teoremas e provas de tais propriedades como completitude e boa qualidade.

Neste ponto é importante ressaltar uma citação que têm como procedência a lista eletrônica SRKB (*Shared Re-usable Knowledge Bases*), e que resume sutilmente o que é uma ontologia e as várias formas e contextos em que uma surge: (Ushold & Gruninger, 1996).

"Ontologias são concordâncias sobre senso comum. Conceitualizações compartilhadas incluem: a) um *framework* conceitual para modelar conhecimentos sobre um domínio; b) protocolos de conteúdo específicos para comunicação entre agentes; c) e concordâncias sobre a representação de teorias de um domínio em particular. No contexto do compartilhamento de conhecimento, ontologias são especificações na forma de definições de um vocabulário representativo. Um simples caso seria um tipo de hierarquia, especificando as classes e suas relações. Um banco de dados relacional também serve como ontologia através das especificações das relações que podem existir nas tabelas de dados compartilhadas e nas regras de integridade que oferecem suporte".

Até este ponto, o enfoque foi sobre a identificação de alguns problemas reais e de propostas para resolvê-los. Tratou-se destes problemas de uma forma não controversa, refletindo-se sobre o termo 'ontologia' e como este é usado na comunidade de pesquisa. Infelizmente uma preocupação real é que não existe um consenso no significado do termo ontologia. Uma análise sobre esta situação pode ser vista em (Guarino & Giaretta, 1995).

Parte da confusão é devido ao fato que idéias e assuntos são orientados a vários contextos e áreas, freqüentemente usando diferentes terminologias, tais como: representação e aquisição de conhecimento, ontologias para entendimento da linguagem natural, modelamento de domínios na engenharia de software e empreendimentos de integração.

3.3 Usos de Ontologias

A literatura é atualmente rica sobre descrições de ontologias e seus propósitos pretendidos. Em um alto nível, a maioria parece ter pretensões para que, de algum modo, se faça sua reutilização. Alguns destes propósitos estão implícitos dentro das várias interpretações da palavra 'ontologia' que são encontradas na literatura, como por exemplo (Guarino &

Giaretta, 1995): um vocabulário para um processo de tradução (Gruber, 1993) ou uma especificação *meta-level* de teoria lógica (Schreiber et. al, 1995; Wielinga et. al, 1994).

Algumas definições sobre ontologias referem-se aos meios para estruturar uma base de conhecimento; outras concebem ontologias como sendo parte de uma base de conhecimento; e algumas se referem a ontologias como uma aplicação específica no domínio das *interlinguas* (Fuchs & Wheadon, 1995; Jones et. al, 1995).

Outra importante motivação para o uso de ontologias é para integrar modelos de diferentes domínios de conhecimento em *frameworks* coerentes. Ontologias estão também sendo usadas em processos da re-engenharia (onde se precisa de um modelo integrado da empresa, seus processos, sua organização, objetivos, e seus clientes) e em arquiteturas de multi-agentes distribuídos (onde diferentes agentes se comunicam e solucionam problemas).

Neste ponto, pode-se subdividir os espaços de uso das ontologias dentro das seguintes categorias:

- Comunicação,
- Interoperabilidade,
- Engenharia de Sistemas: especificação, confiabilidade e re-utilização.

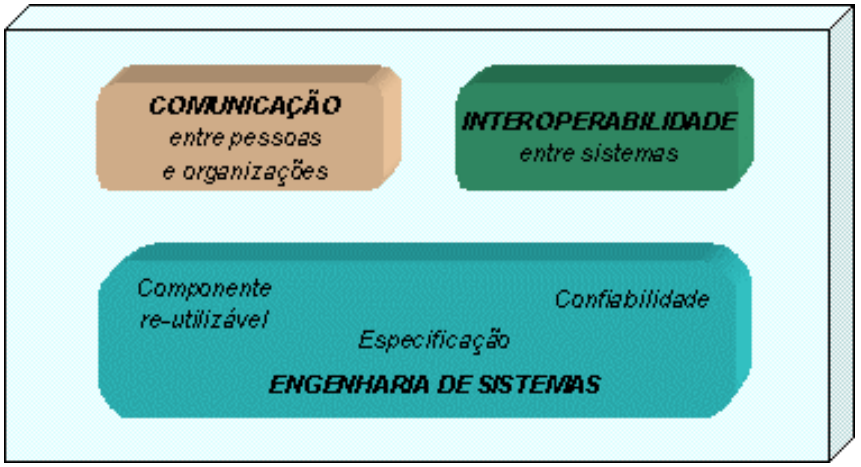


Figura 3.1 Usos de ontologias.

3.3.1 Comunicação

Ontologias reduzem confusões conceituais e terminológicas fornecendo um *framework* unificador dentro da organização. Desta forma, ontologias favorecem o entendimento compartilhado e a comunicação entre pessoas de contextos diferentes com necessidades e pontos de vista particulares. Nos parágrafos seguintes são considerados em detalhe vários aspectos sobre o uso de ontologias para facilitar a comunicação entre pessoas dentro de uma organização (Ushold & Gruninger, 1996):

a) Modelos normativos: dentro de qualquer sistema de software amplo e integrado, diferentes pessoas devem ter uma compreensão compartilhada sobre o sistema e seus objetivos. Usando ontologias, pode-se construir um modelo normativo do sistema. Este cria uma semântica para o sistema e um modelo extensível que posteriormente pode ser refinado.

b) Rede de relações: uma ontologia pode ser usada para criar uma rede de relações com o objetivo de manter um registro de dados que estejam sendo relacionados e para permitir a sua exploração pela navegação através desta rede. Considerando-se que pessoas podem ter diferentes perspectivas sobre um mesmo assunto, é necessário unificar essas perspectivas para forma um senso comum. Isto se torna particularmente importante em aplicações que exigem o uso de várias ontologias em diferentes domínios. É neste contexto que as ontologias servem para fazer com que todas estas perspectivas sejam explicitadas, identificando as conexões lógicas entre elas.

De forma geral, pretende-se que as ontologias ajudem no entendimento do impacto de possíveis mudanças num sistema. Por exemplo, pode-se usar uma ontologia para dar suporte a modelagem da descrição de uma organização permitindo capturar uma visão desta para possíveis modificações. Desta forma pode-se dar resposta às questões da modelagem organizacional, e saber quais cenários variam nas diferentes partes da organização durante uma operação de re-engenharia.

c) Consistência e não-ambigüidade: Um dos papéis mais importantes que uma ontologia representa num sistema é fornecer definições sem ambigüidade para termos usados na definição de suas partes (subsistemas) e na sua interface com o usuário. Qualquer subsistema deve ser capaz de manter a consistência com os outros subsistemas e suas ontologias. A ontologia usada pela interface tem de ser compatível com a ontologia entendida pelo usuário.

d) Integrando diferentes perspectivas de usuários: a integração de diferentes perspectivas através da compreensão do domínio de conhecimento compartilhado torna-se vital em um sistema onde vários agentes se comunicam. Existe o desafio de integrar diferentes perspectivas enquanto se identificam diferenças entre elas. Por exemplo, pessoas em diferentes posições em uma organização terão diferentes perspectivas sobre o que faz a organização, que objetivos alcançar, e como este objetivos devem ser alcançados. Uma ontologia estabelece os fundamentos para o desenvolvimento de padrões dentro de uma organização. Quando adotamos uma ontologia que possa ser compartilhada, todos os participantes usam uma terminologia padronizada em função dos objetivos e das relações da sua organização.

3.3.2 Inter-Operabilidade

A principal preocupação no uso de ontologias em domínios como o modelamento de organizações e arquiteturas multi-agentes é a criação de um ambiente integrador para diferentes ferramentas de software. Muitas dessas ferramentas usam ontologias para lidar com problemas de interoperabilidade (Ushold & Gruninger, 1996). Elas lidam com diferentes usuários que trocam dados ou que utilizam diferentes ferramentas de software.

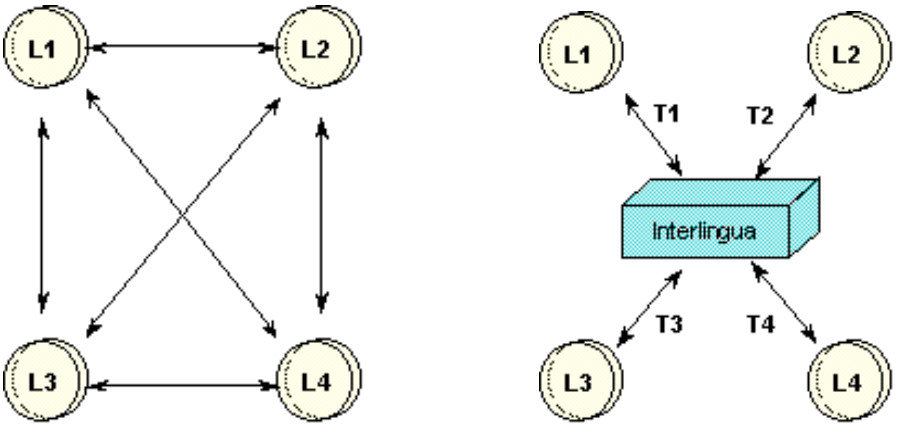
3.3.2.1 O uso de ontologias como InterLíngua

Qualquer ambiente de tecnologia de informação para a re-engenharia de processos de negócios ou para sistemas multi-agentes deve usar uma descrição integrada do seu modelo organizacional para assim poder especificar suas atividades, recursos, objetivos, produtos, e serviços. Esta descrição serve como um repositório comum e acessível para várias ferramentas de software do ambiente. Ela também pode servir para integrar repositórios de dados existentes, seja pela padronização dos termos usados nestes repositórios, ou pelo uso de tradutores que permitam a interoperabilidade entre eles e sua comunicação com os usuários.

Para facilitar essa interoperabilidade, ontologias devem ser usadas para suportar a tradução entre diferentes linguagens e representações. Uma possibilidade seria projetar um único tradutor para cada par de línguas que se quer fazer comunicar; porém, isto exigiria $O(n^2)$ traduções para n linguagens diferentes, como é mostrado na **Figura 3.1a**.

Usando uma ontologia como interlíngua para suportar a tradução, pode-se reduzir o número de traduções a $O(n)$ para n linguagens diferentes (Ushold & Gruninger, 1996), já que isto iria

requer somente traduções para uma ontologia nativa, como mostrado na **Figura 3.1b** (Lee et. al., 1995).



(a) **(b)**

Figura 3.1 : Ontologia como Interlíngua

Pode-se observar a tradução desde uma linguagem Li a Lj e viceversa, somente um tradutor é exigido entre Li e a interlíngua e outro entre a interlíngua e Lj. Deste modo, envolvendo n linguagens, somente são requeridas O(n) traduções, em comparação com as $O(n^2)$ do outro método.

3.3.2.2 Importância da Inter-Operabilidade

Deve-se considerar a natureza das relações entre os usuários que compartilham as ferramentas e os dados, pois ela é vital para que as ontologias e ferramentas que são usadas dentro da mesma organização sejam compartilháveis e re-utilizáveis (Guarino & Giaretta, 1995):

a) Inter-operabilidade interna: Esta característica exige que todos os subsistemas necessitem interoperar desta forma exista um controle direto da unidade organizacional.

b) Inter-operabilidade externa: Esta característica tem-se quando uma unidade organizacional está querendo se comunicar com outros sistemas, procedentes do ambiente externo.

c) Ontologias integradas entre os domínios: Outra diferença para interoperabilidade surge do assunto da integração de ontologias a partir de diferentes domínios para suportar tarefas. Por exemplo, uma ontologia para suportar sistemas *workflow management*, necessitariam de ontologias para os processos, recursos, produtos, serviços e a organização. Então, as diversas ferramentas *workflow* iriam usar este conjunto de ontologias integradas.

d) Ontologias que integram as ferramentas de software: De outro lado, pode-se necessitar diferentes ontologias no mesmo domínio devido à natureza do sistema. Por exemplo, diferentes ferramentas podem querer usar diferentes ontologias relacionadas a processos; para alcançar a interoperabilidade, precisa-se ter uma ontologia comum que ambas ferramentas possam usar. Este é o desafio mais dificultoso a enfrentar no uso de ontologias, visto que usualmente não é possível impor requisitos de integração sobre as ferramentas.

A **Figura 3.2** apresenta o uso de ontologias como uma interlíngua para integrar diferentes ferramentas de software. O termo `procedimento` é usado em uma ferramenta que é traduzido em outro termo, `MÉTODO` é usado por outras ferramentas por meio da ontologia, os termos são considerados os mesmos e dentro do conceito `processo`.

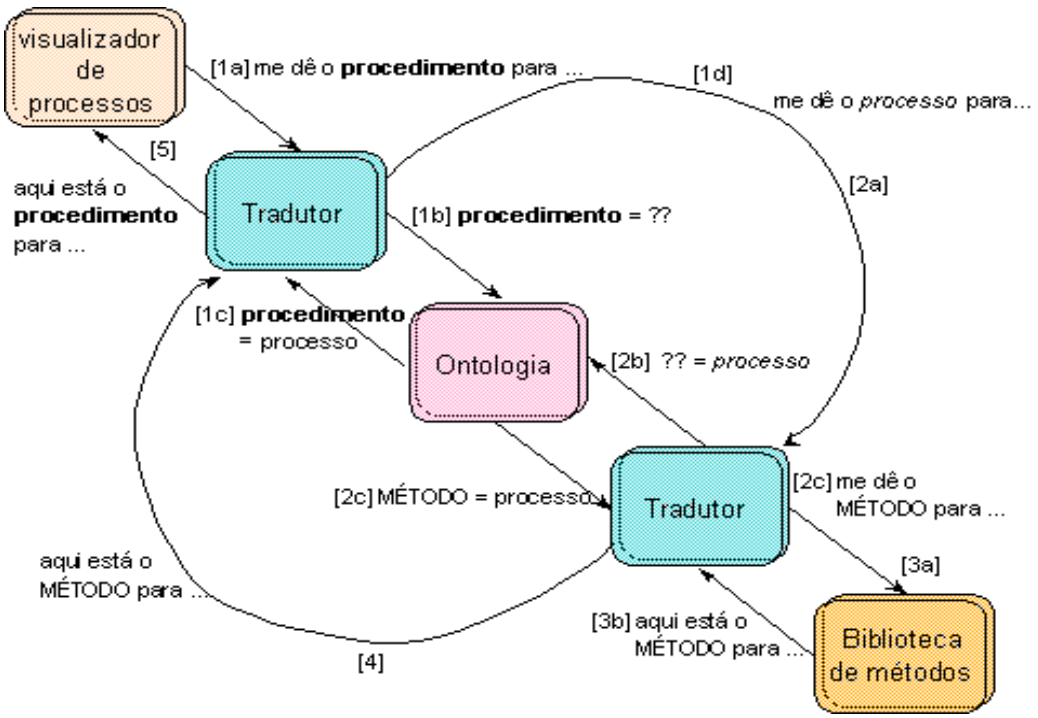


Figura 3.2 Exemplo de uso de ontologia como interlíngua.

3.4 Uma metodologia para construir ontologias

Não existem metodologias padrão para construir ontologias; nem foi possível encontrar muita informação publicada sobre este assunto. Em uma tentativa para começar a preencher esta brecha, uma proposta de metodologia para o desenvolvimento de ontologias foi apresentada por Uschold & Gruninger (1996), seus principais passos são:

- Identificar o propósito e o âmbito da ontologia;
- Construir a ontologia;
    - Escolher a ontologia,
    - Codificar a ontologia,
    - Integrar ontologias existentes;
- Avaliar;
- Documentar.

Na continuação, define-se brevemente cada passo e indica-se que tipo de trabalho pode ser feito e que poderia ser usado para desenvolver uma ontologia.

3.4.1 Identificar o propósito e o âmbito da ontologia

Esta etapa é importante para deixar claro o porquê da ontologia estar sendo construída e como se pretende usá-la. Na seção anterior, mostrou-se como uma ontologia pode ser utilizada; isto pode ser um ponto de início na identificação dos propósitos da ontologia a ser construída. Também pode ser útil para identificar e representar o alcance pretendido para a ontologia.

3.4.2 Construindo a ontologia

A identificação dos propósitos e do âmbito da ontologia, pelo menos em termos gerais, serve para fornecer um objetivo razoável para construção desta. Três aspectos são considerados nesta etapa: a captura, codificação, e integração de ontologias existentes.

3.4.2.1 Escolha

Entendesse por escolha ontológica:

1. Identificação de conceitos e relações principais envolvidos no domínio de interesse;
2. Produção de definições (textuais) sem ambigüidade para aqueles conceitos e relações;

3. Identificação de termos para poder referenciar os conceitos e relações (item 1), que estejam de acordo com as definições do item 2.

3.4.2.2 Codificação

Entende-se por codificação, uma representação explícita da conceitualização capturada na etapa anterior em alguma linguagem formal. Isto envolve as seguintes etapas:

- Considerar os termos básicos que serão usados para especificar a ontologia (por exemplo: *class, entity, relation*); freqüentemente isto é chamado de *meta-ontologia* por que esta é, em essência, uma subcamada ontológica de termos representativos que serão usados para expressar a ontologia principal.

- Escolher uma linguagem de representação (que seja capaz de suportar a *meta-ontologia*);

- Escrever o código.

3.4.2.3 Integrando ontologias existentes

Nas etapas de captura e/ou processo de codificação, existe a questão de se usar partes de ontologias já existentes, ou mesmo ontologias completas. De forma geral, este é um problema difícil. Alguns progressos importantes nesta área estão descritos em (Farquhar et. al., 1995) e (Skuce, 1995). O ponto principal de Skuce é que, para chegar a um acordo entre as ontologias que podem ser compartilhadas entre vários usuários da comunidade, deve-se trabalhar muito para compatibilizá-las.

De forma geral, fornecer orientações e ferramentas nesta área, pode ser um dos maiores desafios no momento de desenvolver metodologias abrangentes para construir ontologias. É fácil identificar sinônimos e estender uma ontologia quando não existe nenhum conceito prévio. Porém, quando existem conceitos similares já definidos em ontologias existentes, não fica claro "como" e "se" tais conceitos podem ser adaptados e re-utilizados.

3.4.3 Avaliação

Gómez-Pérez (Gómez-Pérez et. al., 1995) fornece uma boa definição de avaliação no contexto de compartilhamento de conhecimento:

> "Para fazer uma apreciação técnica sobre as ontologias, precisa-se que o software associado, e a sua documentação estejam em relação a um *framework* de referência. Este *framework* de referência pode conter especificações de requisitos, questões competentes, e/ou aspectos do mundo real."

Alguns detalhes sobre avaliação de ontologias que foram feitas, que poderiam contribuir para uma metodologia abrangente para construir ontologias, e outras informações podem ser encontradas em (Gómez-Pérez, 1995; Gómez-Pérez, 1996; Gruninger & Fox, 1995).

3.4.4 Documentação

Segundo Uschold e Gruninger (1996), pode ser desejável estabelecer diretrizes para documentar ontologias conforme o tipo e propósito da ontologia.

Como foi mencionada por Skuce (1995), uma das principais barreiras para efetivar o compartilhamento de conhecimento é a insuficiente documentação existente sobre as bases de conhecimentos e as ontologias. Para se resolver este problema, devem ser documentadas todas as premissas importantes sobre os conceitos principais definidos na ontologia, bem como as primitivas usadas para expressar as definições na ontologia.

3.4.5 Indicações iniciais para projetar ontologias

Uma metodologia abrangente para construir ontologias deve incluir também um conjunto de técnicas, métodos e princípios para cada um das quatro etapas apresentadas anteriormente, como um bom indicador das relações entre as etapas.

Uma tentativa de consolidar a experiência, que se ganhou em diversos grupos de pesquisas no desenvolvimento de ontologias, é descrita em (Gruber, 1995). Um breve resumo sobre estas experiências são apresentadas na continuação, como um conjunto de critérios de projeto. A ênfase está sobre o compartilhamento e a re-utilização, e são:

a) Clareza: Uma ontologia deve comunicar efetivamente as diferentes pretensões para os usuários que projetam agentes. Isto significa que a ambigüidade deve ser mínima, deve-sedar exemplos para ajudar o usuário a entender as definições que carecem de condições. Se for possível, as definições devem ser especificadas como premissas formais. Em todos os casos, definições devem ser documentadas em linguagem natural e deve-se dar exemplos para ajudar a esclarecer o processo.

b) Coerência: Uma ontologia deve ser consistente internamente. As premissas definidas devem ser pelo menos logicamente consistentes. Também a coerência deve se aplicar às partes das definições que não são colocadas como premissas (como a documentação em linguagem natural e seus exemplos).

c) Extensibilidade: Uma ontologia deve ser projetada antecipando o uso compartilhado do vocabulário. Este deve oferecer fundamentos conceituais para o conjunto de tarefas e a representação deve ser planificada de tal forma que posteriormente possa-se estender e especializar a ontologia de forma uniforme. O usuário deve ser capaz de definir novos termos baseado no vocabulário existente, de tal forma que não se precise fazer revisões de definições existentes.

3.5 Considerações finais

Neste capítulo foi apresentada a necessidade de compartilhamento de conhecimento para melhorar a comunicação entre pessoas, organizações e sistemas de software. Tal compartilhamento pode funcionar como um *framework* unificador, chamado de ontologia, que pode gerar uma serie de benefícios.

Para que exista uma comunicação entre agentes de software e esta seja possível, ela deve estar baseada numa linguagem de comunicação (discussão vista no capítulo 2), que permita a interoperação entre os agentes envolvidos. Essa linguagem deve ter como base uma ontologia comum que permita aos agentes compartilhar conhecimento através da troca de mensagens.

Uma linguagem de comunicação, a *Universal Communication Language*, e sua ontologia são as principais contribuições deste projeto de mestrado.

4. COMUNICAÇÃO ENTRE AGENTES DE SOFTWARE NA INTERNET

4.1 Considerações Iniciais

A Internet tem motivado o desenvolvimento de novas tecnologias voltadas para aplicações distribuídas. No contexto da comunicação entre agentes de software, várias tecnologias têm sido propostas e estudadas, no sentido de aprimorar as técnicas existentes e permitir o desenvolvimento de aplicações cada vez mais complexas e poderosas.

Este capítulo contém uma breve descrição dessas várias tecnologias, que podem ser aplicadas na comunicação entre agentes de software, especificamente para auxiliar a descrever a implementação da linguagem *Universal Communication Language* – UCL, que será vista com maior detalhe no capítulo 6. Dentro das tecnologias que serão descritas inclui-se algumas características da meta-linguagem XML e da linguagem *Universal Network Language* (UNL) (Uchida, 1999). Esta última é um projeto que esta sendo patrocinado pela Universidade das Nações Unidas (UNU). Além disso, é descrita a ontologia *ThoughtTreasure* como meio de representação conceitual.

4.2 A linguagem UNL – Universal Networking Language

A Universidade das Nações Unidas (UNU), sediada em Tóquio, resolveu patrocinar um projeto de longa duração – 10 anos – para o desenvolvimento de ferramentas de software, que usam uma interlíngua como representação intermediaria de mensagens, para vencer a barreira da língua para a comunicação entre os povos. Nesse projeto, ao invés da tradução de uma língua para outra se faz a codificação do conteúdo de um texto em uma dada língua natural para uma artificial, chamada *Universal Networking Language* (UNL) (Uchida, 1999), criada especificamente para textos escritos para o ambiente da Internet. O texto já codificado

em UNL pode então ser decodificado para a língua destino. O projeto UNL é de âmbito mundial, sendo coordenado pelo Instituto de Estudos Avançados, da Universidade das Nações Unidas. Nos três primeiros anos, a partir de 1997, começaram a ser desenvolvidos codificadores e decodificadores para cerca de 15 línguas, incluindo chinês, russo, alemão, francês, italiano, japonês, inglês, hindi, espanhol e português. Nos sete anos restantes previstos para o projeto UNL, espera-se atingir praticamente todas as línguas oficiais das Nações Unidas (Nunes, et.al., 1997).

O Projeto UNL busca uma integração comunicativa em um ambiente de processamento automático de linguagem natural (PALN) integrado em rede, que permitirá que usuários de qualquer parte do mundo possam se comunicar sem que, para isso, tenham que aprender uma linguagem especial de comunicação (Uchida, 1999).

4.2.1 O léxico segundo a perspectiva da UNL

Os componentes do léxico da UNL incluem conceitos padrões ou *Universal Words* (UWs) e um conjunto de relações conceituais e atributos que podem ser expressos estruturalmente, em termos de relações sentenciais. Essas relações são rotuladas adequadamente, por meio de "rótulos de relações" (*Relation Labels* - RLs) e "rótulos de atributos" (*Attribute Labels* - ALs) (Uchida, 1999). As UWs são baseadas principalmente em palavras da língua inglesa e são relacionadas segundo as relações hierárquicas da taxonomia conceitual ou segundo os rótulos de relacionamento sentencial fornecidos por um usuário especialista. Dessa forma, o léxico forma uma hiper-rede de relações entre UWs que abrange conceitos genéricos e específicos, indicando parte de seu inter-relacionamento semântico (Nunes et. al., 1999).

Assim, como se faz uso de RLs para se chegar ao significado pertinente e, logo, a uma UW particular, faz-se uso dos ALs para limitar o significado das UWs. Desse modo, o uso dos componentes sentenciais indica a semântica lexical incorporada ao léxico. Uma descrição completa dos componentes da linguagem UNL é feita no Apêndice A.

4.2.2 As palavras universais (UWs)

A função de uma UW é denotar um significado específico. Sua representação genérica é um rótulo simples (que indica o significado genérico de uma palavra em inglês). Cada UW é representada por uma palavra inglesa que contém o significado genérico na língua inglesa, acompanhada de restrições de maneira que essa mesma palavra inglesa possa gerar várias UWs. Consequentemente, a ambigüidade devido à homografia é eliminada. Por exemplo, a

palavra "book" tem várias UWs a ela associadas, para indicar os seus vários significados, como em :

book(icl>publication), ou seja, livro

ou book(obj>room), ou seja, reservar um quarto.

Para este exemplo deve-se considerar que a UW book esta associada com outras UWs como "publication" e "room", além disso para auxiliar na definição de uma UW se utilizam rótulos de relação como "icl" e "obj" que serão descritos no próximo tópico.

Por tanto, a UW "book" contém todos os significados possíveis, ao passo que as outras com restrições servem para desambigüização (Nunes, et.al., 1997). Observa-se no exemplo que a restrição "(icl>publication)" limita o significado da palavra book referindo esta a uma publicação. A outra restrição "(obj>room)" indica que a palavra book está relacionada com a reserva de um quarto.

4.2.3 Os rótulos de relação (RLs)

A UNL inclui atualmente cerca de 35 relações semânticas. Os RLs expressam relações binárias entre significados, ou seja, UWs. Sua representação genérica é um par do tipo *relation_label* (UW1, UW2), onde UW1 e UW2 são interrelacionadas através da relação semântica denotada pelo RL. Há varias classes de RLs que podem ser associadas a dois componentes da sentença, ou 2 UWs. Um exemplo é a relação (agente-objeto), em que o agente (agt) é um objeto animado que causa uma ação volitiva. Outros RLs incluem método (met), tempo (tim), beneficiário (ben), possessão (pos). Os RLs também são empregados para relacionar UWs na restrição de significados, como inclusão (icl) para representar hiperonímia, como em icl (dog, animal) ou sinônimo (equ) para representar significados equivalentes entre Uws (Nunes, et.al., 1997).

4.2.4 Os rótulos de atributos (ALs)

Os rótulos atributivos (ALs) são empregados para restringir o significado genérico de uma UW. Informações como tempo, aspecto e intenção também são representados como ALs. Um AL é representado por uma UW seguida de atributos identificados pelo símbolo "@". Por exemplo, uma UW com *n* atributos tem a forma:

UW.@attrib1.@attrib2....@attribn.

Se nenhum AL for associado a uma UW, esta terá o significado mais geral de sua classe. Assim como para os RLs, há diferentes tipos de ALs. Alguns tipos mais comuns são (Nunes, et.al., 1997):

- **os que restringem UWs :** que podem ser, por exemplo, @pl e @generic, que indicam, respectivamente, uma UW no plural e de caráter genérico. Este é o caso da sentença *Peter eats apples*, representada em UNL por:

  agt(eat.@present, Peter),

  obj(eat.@present, apple.@generic.@pl).

  Outros ALs deste tipo são @def, @indef e @not.

- **os que expressam tempo verbal :** que foram definidos de acordo com a gramática inglesa, incluindo @past, @present, e @future.

- **os que expressam aspecto, intenção, etc.,** são exemplos @begin-soon, "evento que irá começar", como em *The airplane is about to land*, representado por:

  agt(land@begin-soon, plane.@def).

  Outros exemplos de ALs que expressam aspecto são @begin-just para um evento que recém começou, @end-soon para um evento que está quase terminando, @end-just para um evento que recém terminou, @progress para eventos que estão em curso, @repeat para a repetição de um dado evento que geralmente envolve agente/objeto. Os ALs para expressarem intenção incluem: @emphasis, @topic, @intention, @recomendation, etc. Alguns são aplicados em componentes intra-sentenciais, enquanto outros se aplicam a uma sentença completa.

4.3 A meta-linguagem XML – Extensible MarkUp Language

Enquanto o conteúdo desta dissertação era criado, o processador de textos armazenava informações adicionais, além das palavras que estão sendo lidas agora. Essas informações adicionais são compostas de instruções para controlar o *layout* e a aparência das palavras

propriamente ditas. Tais informações são conhecidas em conjunto como *markup* (marcação) (McGrath, 1999). Existem várias linguagens de marcação patenteadas usadas em pacotes de processadores de texto como também existem linguagens de marcação abertas e sem patentes, como TeX, Troff, e a conhecida HTML (*HyperText Markup Language* – linguagem de marcação de hipertexto).

Nesse contexto surgiu a linguagem XML (*eXtensible Markup Language*) que também é uma linguagem de marcação com vantagens importantes sobre as já citadas. As linguagens de marcação, em sua maioria, são "fixas". O que equivale dizer que elas oferecem um certo conjunto de recursos em sua marcação, e esse conjunto é definido no modelo da linguagem. A XML, por outro lado, não define qualquer conjunto de recursos de marcação em particular. Ao invés disso, ela oferece uma estrutura padrão que possibilita a cada usuário criar sua própria estrutura ou usar outras definidas por terceiros (McGrath, 1999).

A linguagem XML surgiu como uma forma simplificada da *SGML (Standard Generalized Marked Language*). A SGML é um padrão muito poderoso e geral, mas muito complexo. A SGML foi proposta para solucionar um problema compartilhado por instituições, pesquisadores e empresas que necessitavam manipular e realizar o intercâmbio de documentos eletrônicos. A SGML foi desenvolvida de modo a permitir que os documentos pudessem ser escritos com sua própria gramática e formato, legíveis por seres humanos e manipulados de modo eficiente por aplicações computacionais. A XML é um subconjunto da SGML que pretende torná-la "leve" o suficiente para o uso na Web (McGrath, 1999) em muitos casos substituindo a HTML. Isso se dá porque, apesar da linguagem HTML ser muito simples e utilizada em diversas aplicações, existem algumas desvantagens relacionadas à sua flexibilidade como (Johnson, 1999; Mace et al., 1998):

- Não é extensível, ou seja, não é possível especificar e criar novos elementos e atributos para situações específicas e, desta forma, indicar o significado da informação;

- É uma linguagem boa para propósito de exibição, mas não para aplicação de dados, pois ela representa a informação em termos de seu *layout* e não através do seu significado;

- Possui pouca ou nenhuma estrutura semântica, pois seus elementos são agrupados sem seguir nenhuma estrutura pré-definida, o que torna difícil de encontrar o que se esteja procurando na Internet.

Devido a estas e outras desvantagens, torna-se evidente que a HTML atualmente não é suficiente para a especificação e o suporte à variedade de aplicações existentes na Internet. Já a SGML não possui essas desvantagens, mas é muito complexa. Assim, é necessário obter uma linguagem poderosa quanto a SGML e simples como a HTML. Foi com esse intuito que o grupo de trabalho SGML criou o novo padrão chamado XML (*Extensible Markup Language*) (Connolly, 2000).

4.3.1 XML como uma meta-linguagem

O objetivo da XML é fornecer muitos benefícios encontrados em SGML e não disponíveis em HTML. Além disso, ser fácil de aprender e usar se comparada com a complexa linguagem SGML (Mace et. al., 1998; MCGrath, 1999). A XML é definida como um subconjunto da SGML e oferece uma abordagem padrão para descrição, captura, processamento e publicação de informações.

Em XML, os desenvolvedores podem criar seus próprios elementos de acordo com a aplicação que está sendo modelada, dando importância ao conteúdo e à estrutura da informação, sem se preocupar com a apresentação. A linguagem descreve uma classe de objetos de dados (documentos XML) e descreve, parcialmente, o comportamento de programas que os processem (Bray et. al., 2000). Esse processamento é realizado através de dois módulos. O primeiro é chamado de processador XML e é usado para ler um documento XML e fornecer acesso ao seu conteúdo e estrutura. A tarefa fundamental do processador XML é interpretar (*parse*) os dados, ou seja, refere-se ao processo por meio do qual os caracteres que formam o documento XML são categorizados como marcação ou dados de caracteres. O processador envia essas informações para o segundo módulo (aplicação XML), ou seja, ele trabalha em benefício da aplicação, como mostra a **Figura 4.1** (McGrath, 1999)

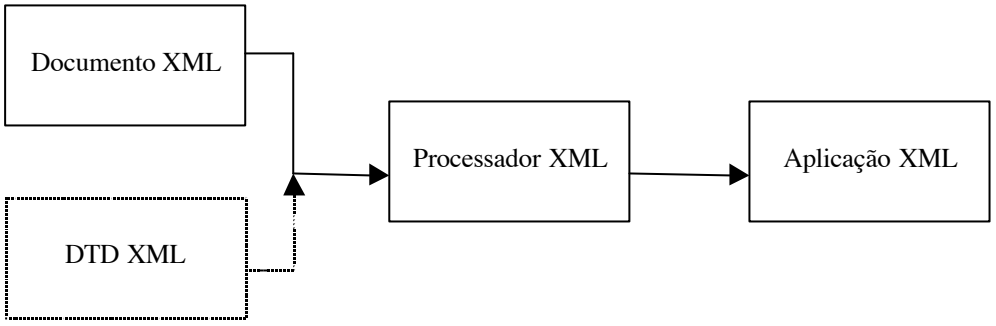


Figura 4.1 Sistema baseado em XML.

Um documento XML pode estar associado a um conjunto de regras que limitam o tipo do conteúdo das informações que podem estar contidas nesse documento. Essas regras encontram-se em um DTD (*Document Type Definition*) e o fato de ser mostrado por linhas pontilhadas na figura 4.1 indica que ele é opcional. A seguir, é detalhado o conceito de DTD e como se ele se relaciona com XML.

4.3.2 O Documento que define as regras para XML

Um DTD (*Document Type Definition*) define regras para a especificação de uma classe de documentos, tais como (Trindade, 1997):

- que tipos de elementos podem existir em um documento (por exemplo, Aluno, Professor, Curso);

- que atributos esses elementos podem ter (por exemplo, um Curso é formado pelos elementos Nome_Curso, Professor, Período, Alunos);

- como as instâncias desses elementos estão hierarquicamente relacionadas (como por exemplo, um Curso possui Alunos e cada Aluno possui um Id (Identificação_Curso), Nome, Email, Homepage e Notas, que por sua vez podem conter várias Notas e uma Nota final).

A estrutura especificada em um DTD, segundo sua definição no padrão XML, possui uma propriedade importante: apenas a estrutura lógica de um documento é descrita, não sendo fornecida informação sobre a apresentação dos elementos definidos (Brown, 1989).

Se o documento XML estiver associado a um DTD, o Interpretador deve verificar se o documento XML está correto (validação). Para isto, o Interpretador processa o DTD que corresponde ao documento XML e obtém sua estrutura. Posteriormente, o Interpretador obtém as informações do documento XML realizando a validação com a estrutura definida no DTD. Por outro lado, se o documento XML não está associado a um DTD, então apenas a sua estrutura sintática pode ser verificada (sem validação).

4.3.3 Acesso e Manipulação de documentos XML com Java

As APIs (*Aplication Programming Interface*) SAX (*Simple API for XML*) e DOM (*Document Object Model*) são recursos que podem ser utilizados por desenvolvedores Java que precisem manipular e apresentar documentos XML suportados por suas aplicações. Esses recursos, comentados a seguir, são os mais explorados na literatura para estes propósitos.

4.3.4 API SAX

A API SAX (*Simple* API *for* XML) foi criada para permitir o acesso à informação armazenada em documentos XML em qualquer linguagem de programação (Megginson, 1998). Ela provê esse acesso não como uma árvore de nós, mas como uma seqüência ordenada de eventos. A aplicação interpreta esses eventos de uma maneira significativa e cria o seu próprio modelo de objetos baseada nestes eventos. Assim, bibliotecas que implementam a API SAX tornam-se mais rápidas e simples, porém as aplicações que usam essa interface precisam atender das seguintes condições (Idris, 1999):

- criação de seu próprio modelo de objetos personalizado;
- criação de uma classe que capture os eventos SAX (gerados pelo Interpretador SAX conforme ele processa um documento XML) e crie adequadamente o modelo de objetos.

Uma vantagem de SAX é que o documento inteiro não fica residente na memória, o que é interessante em situações onde há processamento de documentos gigantes. O Interpretador SAX percorrem um documento XML ativando métodos que tratam de cada caso, como, por exemplo, início, conteúdo e fim de elemento.

A API SAX é mais adequada para propósitos onde é preciso ler um documento XML completo do início ao fim e executar alguma tarefa, tal como construir uma estrutura de

dados que represente um documento ou resumir a informação contida em um documento. SAX não é muito útil em casos onde se deseja modificar a estrutura do documento de alguma forma complicada que envolva troca de elementos que estão aninhados. Desse modo, a API SAX é recomendada em casos onde a informação é estruturada de tal maneira que seja mais fácil criar um mapeamento próprio que representar a informação como uma árvore (Idris, 1999).

4.3.5 API DOM

O poder da XML está na sua capacidade de representar a estrutura da informação baseada em como cada parte dessa informação se relaciona com as demais (Bray et al., 2000). Os documentos estruturados têm a propriedade de poder ser aninhados um dentro do outro, apresentando uma estrutura em forma de árvore.

A API DOM (*Document Object Model*) fornece uma interface independente de plataforma e linguagem para a estrutura e o conteúdo de documentos HTML e XML. Ela descreve uma linguagem neutra capaz de representar qualquer documento XML bem formado em forma de uma árvore e tratar a informação armazenada nesses documentos como um modelo de objetos hierárquicos (Idris, 1999). A API DOM cria uma árvore baseada na estrutura e na informação do documento, sendo que o acesso à informação pode ser feito através de interações com os nós dessa árvore.

A API DOM busca fornecer um modelo padrão único na forma como são organizados os diversos objetos que formam os documentos. Ela também busca padronizar uma interface desses objetos para facilitar a navegação em documentos e o processamento destes.

Os principais objetivos de projeto do modelo DOM são (McGrath, 1999):

- fornecer um conjunto de objetos e interfaces suficientes para representar o conteúdo e a estrutura dos documentos HTML e XML sem perda de informações significativas;

- realizar isso de uma forma independente de plataforma e linguagem;

- fornecer funcionalidade de criação de objeto poderosa o suficiente para permitir que documentos HTML e XML sejam criados completamente a partir do zero;

- fornecer uma base sólida e extensível na qual podem ser adicionadas novas camadas DOM no futuro.

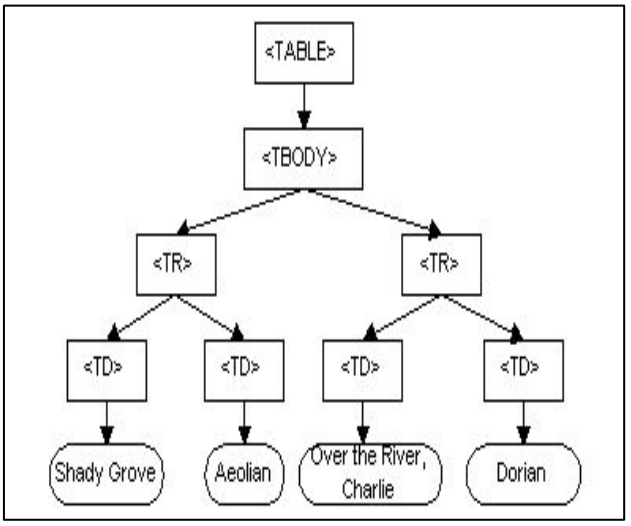
Diferentemente da API SAX, no modelo DOM o Interpretador faz quase tudo: lê o documento, cria um modelo de objeto em uma linguagem (nesse trabalho, em Java) e fornece uma referência para este modelo de objeto ser controlado (Idris, 1999).

Com o uso do modelo DOM, os usuários se beneficiam de uma interface homogênea tanto para HTML como para XML. E ainda é possível fazer a transformação de um documento XML para HTML, por exemplo, utilizando as especificações XSL (Extensible Stylesheet Language) (XSL, 2000).

Uma alternativa para manipular e apresentar documentos XML é utilizá-los como objetos ou tratá-los como uma seqüência de eventos. Isso porque, uma vez que os elementos de um documento estruturado estão aninhados, estes podem ser representados como uma estrutura em árvore (os objetos são os nós da árvore) ou uma seqüência ordenada de eventos. Exemplos: a **Figura 4.3(a)** apresenta a hierarquia do documento HTML da **Figura 4.2** em forma de nós em uma estrutura de árvore: o elemento <TBODY> é filho de <TABLE>, o elemento <Over the River, Charlie> é filho do elemento <TD>, que por sua vez, é filho à esquerda do elemento <TR> que, por sua vez, é filho à direita do elemento <TBODY>. Já a **Figura 4.3(b)** apresenta os elementos HTML da **Figura 4.2** como uma seqüência ordenada de eventos: 1) start table, 2) start element:tbody, 3) start element:tr, e assim sucessivamente.

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
 </TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
```

Figura 4.2 - Estrutura do HTML utilizado no exemplo (DOM, 1998)



start table
start element: tbody
start element: tr
start element: td
characters: Shady Grove
end element: td
start element: td
caracteres: Aeolian
end element: td
end element: tr
start element: tr
start element: td
caracteres: Over the River, Chaarlie
end element: td
start element: td
caracteres: Dorian
end element: td
end element: tr
end element: tbody
end elment: table

Figura 4.3 - **a**) Estrutura baseada em árvore (API DOM) (DOM, 1998) **b**) Estrutura baseada em evento (API SAX)

4.4 Representação de sentenças em linguagem natural.

No uso cotidiano da língua, a comunicação entre os falantes se dá através de textos embora esses mesmos falantes possuam uma consciência intuitiva das unidades mínimas de língua. Em termos de análise linguística, pode-se dizer que o texto é a unidade maior na estrutura de uma língua natural, pois reúne em si informações de diversas naturezas que, por sua vez constituem no objeto de estudo de alguns campos específicos na área da Linguística.

No processamento de linguagem natural (PLN) o material de processamento é um texto que deve ser analisado, ou seja, recortado em unidades menores para a compreensão completa dos mecanismos de operação envolvidos em cada dessas unidades. Assim, o PLN recorre a campos específicos da Linguística, procurando depreender da sua descrição as informações que irão fazer da máquina um instrumento sensível aos fenômenos da língua natural.

O processamento da linguagem natural tem seus inícios na semiótica, o estudo dos signos. A semiótica foi desenvolvida por Charles Sanders Peirce (logicista e filósofo) e Ferdinad de Saussure (lingüista). A semiótica é dividida em três ramificações: sintaxe, semântica, e pragmática.

Um processador de linguagem natural completo pode extrair o significado da linguagem em vários níveis. Quatro são os mais importantes para esse trabalho:

a) Morfológico: Palavras da língua também podem ser segmentadas em termos do seu conteúdo significativo. As unidades mínimas dotadas de significado (gramatical ou lexical) são denominadas morfemas e se constituem no objeto de estudo da Morfologia.

b) Sintático: A organização das palavras na sentença acarreta a definição desses itens lexicais em termos de suas funções gramaticais. Trata-se de reconhecer as regras pelas quais a distribuição das formas é determinada sendo esse exercício o objeto de estudo da Sintaxe.

c) Semântico: As relações envolvidas no plano do significado das palavras em busca de alcançarem certo sentido no escopo da sentença são a matéria de investigação da Semântica. O significado é inerente ao signo linguistico e está presente não só na palavra como uma unidade completa, mas nas suas unidades constitutivas. Da mesma forma, fala-se em significado de expressões, de sentenças, em fim, de unidades mais complexas da língua. Grande parte do esforço do tratamento semântico em PLN deve envolver, então, a depreensão das propriedades semânticas dos itens lexicais para a construção de sentenças semanticamente bem formadas da língua.

d) Pragmática: Nesse nível de análise linguística estão em foco as questões, consideradas por muitos estudiosos, do mundo extralinguístico. Essa noção é amparada pelo fato de que para além das formas e das estruturas, a língua recupera da situação comunicativa diversos fatores que implicam na determinação de certa compreensão das palavras e sentenças. Todo texto é produzido por certos interlocutores, em um tempo e um lugar determinado, o que significa dizer que nenhum texto existe independente dos indivíduos envolvidos na atividade comunicativa e nenhum texto existe sem uma situação de contexto. Quando se examina uma construção linguística procurando essas relações presentes no ato da fala, na verdade procura-se estudar aquilo que é objeto da Pragmática.

4.4.1 A ferramenta *Thought Treasure* (TT)

O programa Thought Treasure é uma poderosa ferramenta de processamento de linguagem natural (ferramenta de código aberto), desenvolvida por Erik T. Mueller.(1998). Esta ferramenta é capaz de interpretar linguagem natural, como também estender sua base de conhecimento que está baseado em ontologias. É um compilador para linguagem natural que permite extrair informação de sentenças ou frases.

O TT é formado por uma base de dados de 25,000 conceitos organizados hierarquicamente (Mueller, 1998). Por exemplo, *Evian* é um tipo de *flat-water*, o qual é um tipo de *drinking-water*, o qual é um tipo de *food* e assim por diante.

Cada conceito tem uma ou mais aproximações sinônimas, o programa chega a um total de 55,000 palavras e frases do idioma inglês e francês. Por exemplo, como se vê na **Figura 4.4**, associações com o conceito *food* no idioma inglês são as palavras *food* e *foodstuffs* e em francês *aliment* e *nourriture* (entre outras).

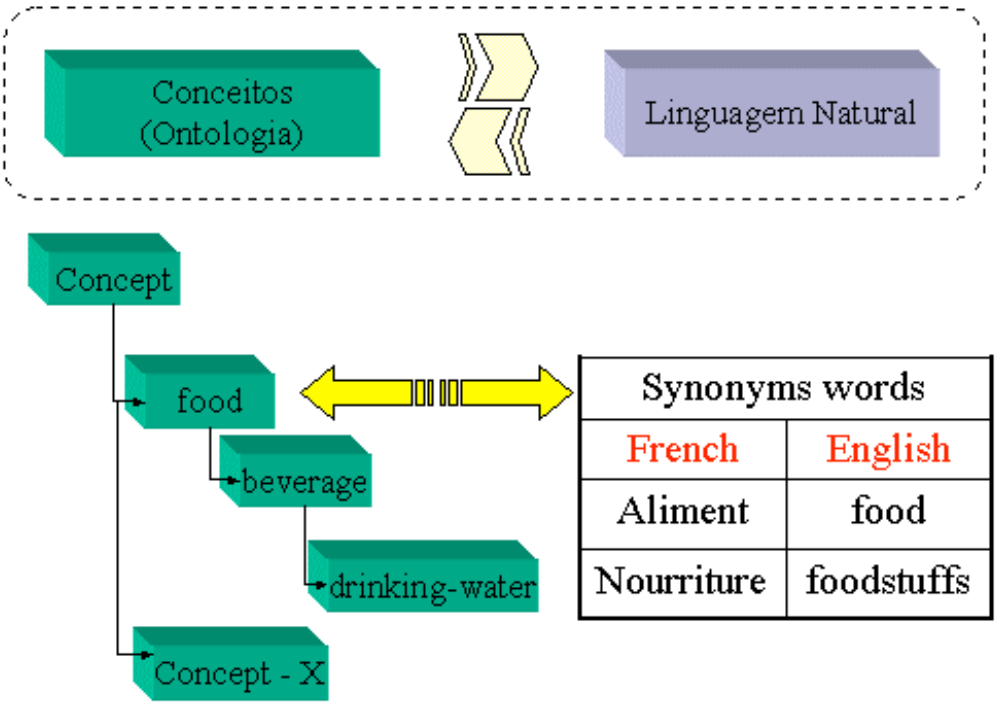


Figura 4.4 Associação da ontologia com a linguagem natural.

O programa *ThoughTreasure* tem aproximadamente 50,000 asserções relacionados a conceitos, assim como: a *green-pea* is a *seed-vegetable*, a *green-pea* is *green* , a *grean-pea* é parte de *pod-of-pea*, *pod-*

of-peas é encontrado geralmente em uma loja de comestíveis. *ThougTreasure* tem ao redor de 100 *scripts*, descrevendo atividades típicas, que permitem entender o mundo real. O programa *ThoughTreasure* compreende sete módulos, mas esse trabalho utilizou apenas cinco deles (Mueller, 1998):

Representação de agência

Esse módulo consiste do banco de dados, procedimentos básicos e estruturas para representar conceitos, espaço, tempo, atores e contextos. Esta é mais bem entendida através da **Figura 4.5**, que mostra o formato do banco de dados do *ThoughtTreasure:*

```
=media-object/information/
==advertisement//
==art//
==computer-program//
==dance//
==film//
===film-genre//
====comedy-film//
====documentary-film//
====drama-film//
====fantasy-film//
====horror-film//
====musical-film//
====mystery-film//
==genetic-code//
==opera//
==play//
==text//
===book//
===magazine//
```

Figura 4.5 Formato do banco de dados do *Thought Treasure*

O nível de distanciamento das margens pelo símbolo "=" identifica a hierarquia. Neste caso, por exemplo, *advertisement* é um tipo de *media-object*. A hierarquia consiste em conceitos concretos, os quais se subdividem em entidades, situações, estados, ações, relações, atributos e enumerações.

Objetos podem ter pais explícitos representados por uma lista de conceitos, separados pelo símbolo "/". Por exemplo, *media-objects* tem como pai o conceito *information*. Conceitos no mesmo nível de hierarquia são considerados equivalentes; por exemplo, *opera* é equivalente a *play* dentro do conceito *media-object*.

Um conceito amplo dentro da ontologia de *ThoughtTreasure* é a relação. Uma relação pode compreender subclasses, permitindo aos usuários adicionar relações de domínios específicos a suas ontologias proprietárias. A **Figura 4.6** apresenta um exemplo:

```
=media-object-relation/relation/
==author-of//
==composer-of//
==newscaster-of//
==viewer-of//
==actor-of//
==cinematographer-of//
==director-of//
==language-of//
==MPAA=rating-of//
==producer-of//
==writer-of//
```

Figura 4.6 Relações em *Thought Treasure*.

Na **Figura 4.6**, o usuário define o conceito *media-object-relation*, que é uma relação declarada explicitamente. Assim, o usuário pode definir várias relações para descrever o conceito *media-objects*.

O usuário pode fazer afirmações (*assertions*) sobre como um conceito se relaciona com outro. Por exemplo, na **Figura 4.7** pode-se observar que as afirmações são codificadas na forma `|relação = conceito|` e fazem parte de um conceito (neste caso, do conceito: filme *The Big Lebowski*)

```
==comedy-film//
==mystery-film//
===the-big-lebowski/|director-of=MALE:"Joel
                      Coen"|
                    |actor-of=MALE:"Jhon Goodman"|
                    |actor-of=MALE:"Jeff Bridges"|
                    |actor-of=MALE:"Steve Buscemi"|
```

Figura 4.7 Asserções em *ThoughtTreasure*.

Componente Léxico

Parte do formato da base de dados permite que palavras sejam marcadas (com atributos) para representar características especificas de um idioma. Por exemplo:

```
====médium#A-length# film*.z//
```

Esta entrada especifica que a frase *medium-length film* é composta por um adjetivo (A) e dois substantivos (por definição do idioma). Os símbolos "#" significam que a palavra é usada como é (outra forma gramatical não é permitida). O símbolo "*" significa que outra forma e conjugação da palavra podem ser usadas no lugar da que está especificada. Por exemplo, *médium-lenght films* e *médium-lenght film* são considerados conceitos equivalentes.

Text agency

Esse módulo é responsável por mapear ou examinar o texto de entrada em linguagem natural por palavras e frases predefinidas. Ele cria um nó para cada possível forma gramatical, coloca etiquetas nesses nós (com atributos característicos do idioma usado) e envia os resultados ao componente sintático.

Componente sintático

Este componente é responsável pela criação das árvores sintáticas a partir da saída (nós) gerada pelo módulo *Text agency*. Ele usa várias regras de produção e filtros para efectuar isto.

Componente semântico

Este é o mais importante de todos os módulos e consiste do *parser* semântico, o qual é responsável por transformar a árvore sintática em afirmações do *ThoughtTreasure*. Este usa várias construções gramaticais de alto nível para separar a árvore de informação em afirmações.

4.4.2 Comparação do *ThoughtTreasure* com outros sistemas

Na continuação, apresenta-se brevemente uma comparação do banco de dados do programa *ThoughtTreasure* (Mueller, 1998) em relação a outros como: *WordNet* (Miller, 1995b), *Cyc* (Lenat, 1995), *EDR* (Yokoi, 1995), and *LADL's electronic dictionary* (Courtois & Silberztein, 1990).

- Linguagens: Indica a quais linguagens o sistema é orientado (I = Inglês, F = Francês, J = Japonês).

- Le: Indica o número de entradas léxicas, incluindo palavras, frases, e nomes próprios. Para sistemas bilíngües tais como *ThoughtTreasure* e EDR, o número mostrado representa o total envolvendo as duas linguagens. 60 % das entradas léxicas no

ThoughTreasure são para o inglês e 37 % são para frases. 53 % das entradas léxicas para LADL são frases.

- Infl: Indica o número de inflexões validadas. *ThoughtTreasure* contém 127,448 inflexões geradas automaticamente das quais somente 32,802 foram conferidas, em relação a qualidade e precisão do seu significado, por usuários especialistas.

- Obj: Indica o número de conceitos atômicos.

- <lê, obj>: Indica o número de uniões (*links*) entre as entradas léxicas e objetos.

- Polissemia: Indica a porcentagem de entradas léxicas unidas a mais de um objeto.

- Asserções: Indica o número de asserções (fatos que contém conceitos atômicos) no sistema. As asserções do *ThoughtTreasure* que foram codificados na linguagem C não são incluídas na tabela de acima.

- Esforço: Indica o número de pessoas utilizadas por ano para o desenvolvimento do sistema.

Tabela 4.1 Comparação do ThoughtTrasure com outros sistemas (Mueller, 1998)

Características	TT	WordNet	Cyc	EDR	LADL
Linguagens	I,F	I	I	I,J	F
Le	50,133	118,000	Ind	600,000	170,000
Infl	32,802	Ind	Ind	Ind	600,000
Obj	21,521	90,000	100,000	400,000	Ind
<le, obj>	45,739	166,000	Ind	Ind	Ind
Polissemia	6.1%	17%	Ind	Ind	Ind
Asserções	37,162	116,000	1,000,000	Ind	Ind
Esforço	2	Ind	Ind	Ind	100

Existem campos de dados na tabela que não estavam disponíveis, em tal caso se colocou "Ind", indicando a indisponibilidade da informação.

4.5 Considerações finais

Neste capítulo foram apresentados os conceitos e as principais características das tecnologias a serem usadas no projeto. Como foi citada, a XML permite definir, criar e guardar documentos, além de recuperar hierarquias de dados. Estas são características importantes, que foram consideradas para a adoção de XML neste projeto, para representar de forma conveniente às mensagens numa linguagem de comunicação para agentes. Para definir os tipos de elementos que podem ser usados numa linguagem desse tipo, assim como suas possíveis relações, um documento DTD (*Document Type Definition*) será usado.

Neste ponto, é importante ressaltar que este projeto não fez uso da linguagem UNL, porque esta linguagem não disponibiliza toda sua informação sobre sua linguagem. Como todas as linguagens para uso na Web (HTML, XML, etc.) são derivadas da linguagem SGML, vai ser muito difícil que sejam escritos novos *parsers* para *browsers* (e outros softwares que compõem o WWW) que não sigam algum padrão derivado do SGML. Isso representa um grande obstáculo a adoção da linguagem UNL em browsers para o WWW, uma das metas do projeto UNL (e desse trabalho). Contudo serão usados os conceitos teóricos da linguagem UNL no desenvolvimento da linguagem de comunicação para agentes UCL, que será implementada em XML. A UCL também não usará a ontologia da UNL por esta não está disponível publicamente, será usada a ontologia disponível no programa *ThoughtTreasure* para a representação conceitual das mensagens em UCL.

5. A LINGUAGEM UCL

5.1 Considerações iniciais

Este capítulo contém a descrição da linguagem proposta neste trabalho para a comunicação entre agentes a UCL – *Universal Communication Language*. Inicialmente são situadas as características que foram consideradas para a elaboração da linguagem. Em seguida vem a metodologia que orientou o seu desenvolvimento e, por fim, a descrição completa da estrutura das mensagens especificadas nesta linguagem.

5.2 A abordagem da linguagem UCL

A linguagem UCL visa estabelecer uma comunicação de alto nível envolvendo agentes (de software ou humanos) com sentenças transmitidas representam de forma apropriada o domínio de conhecimento. Essa comunicação pode ocorrer:

- **Via Internet:** entre agentes de software e entre pessoas e agentes de software (através de uma interface). Ela também pode ocorrer entre pessoas (que falem diferentes línguas) e ser mediada por agentes de software usando a UCL.

- **Via Linguagem *Script*:** entre um programador, que escreve comandos em UCL, e um agente de software, que vai interpretar e executar esses comandos.

Muitos dos conceitos usados em UCL foram derivados da linguagem UNL (Uchida, 1999) e adaptados para serem usados num ambiente XML. Algumas das principais características que guiaram a definição da linguagem foram:

- Auxiliar a comunicação envolvendo agentes dando importância à semântica da mensagem;

- Ser de fácil utilização;

- Permitir sua integração aos padrões usados na Internet seguindo a compatibilidade com o padrão SGML (*Standard Generalized Markup Language*).

Para definir uma linguagem seguindo as características acima, foi considerada a abordagem de uma linguagem baseada em mensagens (comandos) e argumentos. O usuário expressa cada mensagem através de uma estrutura formada pelos conceitos relacionados ao domínio de conhecimento, esta estrutura contem relações que envolvem estes conceitos. Além disso, cada conceito contém uma estrutura de atributos que descreve o conceito de forma que se evite a ambigüidade com outros conceitos. A linguagem UCL apresenta a vantagem de permitir uma descrição formal e semântica das mensagens. A principal desvantagem desta abordagem é exigir do usuário um aprendizado anterior sobre alguns aspectos da linguagem tais como: a sintaxe da mensagem, palavras reservadas, notações especiais, etc.

Mesmo assim, considerou-se a utilização desta abordagem devido a importância que as mensagens têm para representar de forma mais apropriada o domínio de conhecimento. Por exemplo: um agente emissor que está em um domínio de conhecimento específico transmite uma mensagem a outro agente fazendo uma requisição de informação, o agente receptor deve saber que conceitos e características que formam a mensagem. Além do mais, considerou-se que o usuário desta linguagem já tem alguma experiência no uso de computadores, bem como no uso de uma linguagem de programação, tornando-se desnecessário direcionar o usuário durante a utilização da linguagem.

Para minimizar os inconvenientes desta abordagem, bem como atender as características da linguagem listadas anteriormente, foram estabelecidos alguns critérios que devem ser seguidos na definição da linguagem:

- As mensagens, que são compostas por conceitos, atributos e suas relações, devem ser representadas de forma organizada e estruturada para que sejam de fácil leitura para o usuário.

- Permitir a escalabilidade e extensibilidade da linguagem, de tal forma que se possa agregar novos conceitos relacionados ao domínio de conhecimento.

- Codificar a linguagem com uma meta-linguagem padrão que permita a comunicação entre sistemas heterogêneos e ao mesmo tempo que seja de fácil integração na Internet.

- Indiferença entre letras maiúsculas e minúsculas e minimização de pontuações especiais.

Incorporando estes aspectos à linguagem se pode oferecer uma representação mais legível.

5.3 Metodologia utilizada

O desenvolvimento da linguagem UCL pode ser dividido em duas etapas: definição da linguagem e implementação de um programa protótipo que converte inglês para UCL e UCL para inglês.

a) Definição da linguagem

Nesta primeira fase, procurou-se especificar a linguagem, tendo como ênfase abranger os aspectos conceituais de como representar o significado das mensagens. Para esta representação, utilizou-se a base teórica da linguagem *Universal Networking Language* (UNL) (Uchida, 1999). Como foi dito anteriormente, a UNL é uma linguagem de representação semântica baseada em relações envolvendo predicados (conceitos), desta forma, a linguagem UCL proporciona os meios para descrever as mensagens.

No item 5.5, será feita uma descrição mais detalhada desta fase.

b) Implementação do Protótipo

Utilizando a especificação inicial da linguagem, que se baseia nas características e critérios mostradas anteriormente, passou-se a preencher os aspectos gramaticais

necessários para a implementação de um *parser* para a linguagem. Para isso foi necessário:

- Estabelecer com rigor os critérios sintáticos da linguagem inicialmente definida, o que se traduziu em: definir palavras reservadas, separadores, estabelecer a gramática da linguagem;

- A utilização da meta-linguagem XML para definir a linguagem proposta. Esta meta-linguagem une a simplicidade da XML ao poder representativo da UNL e reduz as dificuldades de se escrever um interpretador para ela.

- Definição da gramática (regras) em um arquivo DTD (*Document Type Definition*), que representa a estrutura lógica da mensagem envolvendo elementos, atributos e instâncias.

Como foi dito na introdução deste trabalho, a linguagem UNL não é adequada à filosofia de trabalho do nosso grupo, por não ser um padrão aberto, além de não derivar do padrão SGML (*Standard Generalized Markup Language*), o que dificulta o seu uso em programas criados para a Internet. Mas muitos dos conceitos teóricos de UNL (Uchida, 1999) foram usados por seu poder de representar mensagens. Como resultado desta fase, obteve-se uma definição da linguagem para representar mensagens relacionadas a um determinado domínio de conhecimento e um programa protótipo para conversão inglês – UCL e UCL – inglês (capítulo 6).

5.4 Características da linguagem UCL

A linguagem UCL representa a informação em sentenças (também chamadas de mensagens) que envolvem uma estrutura sintática e um conjunto de conceitos, relações e atributos que são definidos a seguir:

- UW (*Univeral Word*), que representa o significado de uma palavra ou conceito.

- Rótulo de relação, que representa uma relação entre *Universal Words*.

- Rótulo de atributo, que contém alguma informação adicional ou definição que se acrescenta à *Universal Word* e que está presente na mensagem.

Considerações gerais sobre a linguagem:

- Para que a linguagem proposta cumpra com a característica relacionada à representação semântica da mensagem, considerou-se a utilização de uma ontologia que define formalmente o domínio de conhecimento.

- A ontologia usada foi a definida no programa *ThoughtTreasure* já que a linguagem UNL também não dispõe de uma ontologia explícita e aberta, O vocabulário da linguagem corresponde a todas as entradas léxicas da Ontologia que usa o programa *ThoughtTreasure*.

- Considerando-se que as mensagens devem evitar ambigüidade, cada *Universal Word* (representação de um conceito) é limitada na abrangência de seu significado por meio do uso de rótulos de relação. Desta forma evita-se que dois conceitos se justaponham.

- Definidos os conceitos que pertencem à mensagem, é necessário relacioná-los, utilizando rótulos de relação, para terminar a construção de uma mensagem.

5.4.1 A notação utilizada

Como foi visto anteriormente, a XML é uma meta-linguagem usada para definir outras linguagens. Para definir uma linguagem baseada em XML pode ser usado um arquivo contendo as regras de um DTD específico. A sintaxe de um DTD é essencialmente uma gramática de livre contexto, tal como, a forma BNF estendida (*Backus Naur Form*) usada para descrever linguagens de computador (Grosof & Labrou, 1999).

Considerando que a linguagen UCL é baseada na meta-linguagem XML, a gramática formal da UCL é especificada utilizando-se a notação definida para DTDs. A seguir descreve-se brevemente alguns elementos dessa notação (Pimentel et. al, 1999):

Element (elemento): Define os elementos do documento XML. Cada elemento tem um tipo, identificado por um nome e pode ter um conjunto de especificações de atributos a ele associado. Cada especificação de atributo compõe-se de um par nome/valor.

Attrlist (atributo): Um elemento pode ser classificado qualitativamente através de atributos. Um atributo é um par (nome, valor) presente no inicio de cada rótulo de elemento. Um atributo não pode aparecer mais de uma vez no mesmo elemento.

Entity (entidade): Em um documento, referências a outras entidades podem ser externas ou internas; uma referência a uma entidade acontece na forma &*nome_da_entidade;*. Entidades externas se referem a arquivos contendo documentos, imagens, etc. Entidades internas normalmente se referem a variáveis internas.

CDATA : Uma seção CDATA permite a inclusão de trechos que, contendo caracteres reservados para marcação, devem ter seu conteúdo ignorado pelo processador.

5.5 Especificação da linguagem

5.5.1 UW (Universal Word)

A *Universal Word* (UW) é a unidade mínima que representa um conceito, estas em conjunto denotam o significado específico de uma mensagem. Na sua representação são usadas palavras em inglês (algumas em japonês), que representem (o melhor possível) o significado ou conceito que a UW envolve.

Quando existe a necessidade de representar um conceito de forma mais exata, a linguagem possui métodos para que o conceito seja restrito no momento da definição de seu significado. Estes métodos são os rótulos de relação e rótulos de atributo que serão mostradas nos tópicos seguintes. Na **Figura 5.1** é mostrada a estrutura (sintaxe) geral de uma *Universal Word*:

```
<!ELEMENT uw ((%label;)*, (tense | aspect | reference | focus |
            attitudes | viewpoint | convention)*)>
<!ATTLIST uw
      id ID #IMPLIED
      head CDATA #REQUIRED>
```

Figura 5.1 Sintaxe geral de uma *Universal Word*

Na estrutura mostrada na **Figura 5.1**, pode se observar que cada `uw` definida deve ter um identificador `id`. Este identificador será formado por uma cadeia de caracteres alfanuméricos e servirá como único identificador do conceito em toda a mensagem. Este identificador pode ser referenciado em qualquer outra parte da mensagem, mas não pode ser redefinido.

O rótulo `head` corresponde ao lugar onde o conceito será incluído. Os conceitos utilizados sempre estarão relacionados à ontologia que se esteja usando, deste modo é neste ponto que a linguagem UCL se relaciona com a ontologia de um domínio especifico de conhecimento.

Na **Figura 5.2** é apresentado um exemplo de documento UCL com a definição de conceitos, utilizando a sintaxe da **Figura 5.1**:

```
<uw id="uw02" head="area">
   <icl direction="to">
      <uw head="place"/>
   </icl>
   <reference attribute="indef"/>
</uw>

<uw id="uw03" head="strategic"/>

<uw id="uw04" head="designate">
   <icl direction="to">
      <uw head="do"/>
   </icl>
   <focus attribute="entry"/>
   <viewpoint attribute="may"/>
</uw>

<uw id="uw05" head="read">
   <icl direction="to">
      <uw head="do"/>
   </icl>
</uw>
<uw id="uw06" head="home"/>
```

Figura 5.2 Definição de conceitos em um documento XML

Neste exemplo, pode-se observar a definição de cinco conceitos, cada um rotulado como uma `uw`. Cada conceito tem um identificador único no atributo `id, e um` conceito básico no atributo `head`. Os rótulos de relação e atributo mostrados na estrutura são explicados nos tópicos seguintes.

5.5.2 Rótulos de relação

A linguagem UCL representa mensagens que possuem um determinado significado envolvendo vários conceitos. Esta composição de conceitos é representada por um conjunto de relações binárias que permitem distinguir as diferentes relações envolvendo esses conceitos.

Em concordância com a teoria usada na definição da linguagem UNL para a representação de composição de conceitos, adotou-se os rótulos de relação definidos nessa linguagem. Estes rótulos estão formados com, no máximo, três caracteres de comprimento.

A sintaxe geral de um rótulo de relação é mostrada na **Figura 5.3**:

```
<!=========== Definição de Entidades ================-->
<!ENTITY % label "(icl | agt | aoj | bas | ben | cag | cao | cnt |
            cob | con | coo | dur | fmt | frm | gol | ins | man |
            met | mod | nam | obj | opl | or | per | pic | plf |
            plt | pof | pos | ptn | pur | qua | rsn | scn | seq |
            src | tim | tmf | tmt | to | via)">
<!ENTITY % extension "uw, %label;)?"
<!ENTITY % direction "direction ( to | from ) 'to'"
<!=========== Definição da sintaxe ================-->
<!ELEMENT uw ((%label;)*, (tense | aspect | reference | focus |
            attitudes | viewpoint | convention)*)>
<!ELEMENT icl (%extension;)+>
<!ATTLIST icl %direction;>
```

Figura 5.3 Sintaxe geral de um Rótulo de relação.

Os rótulos de relação estão representados por `%label;`. A seguir, mostra-se a lista e uma breve definição de cada uma das instancias de `%label;`.

- `Icl` (inclusão) representação de hiperonímia (super e subclasses) ou meronímia (relação parte-todo). Neste rótulo de relação se encontra o atributo "direction='to' que indica a relação parte-todo."
- `Agt` (agente), define um agente que causa uma ação volitiva, por exemplo, um objeto animado com intenções.

- `And` (união) define uma relação de conjunção envolvendo dois objetos ou conceitos.
- `Aoj` (objeto atributivo) define um objeto de um atributo.
- `Bas` (para expressar grau) define um objeto usado como a base para expressar um grau de comparação.
- `Ben` (beneficiário) define um beneficiário que está indiretamente relacionado com outro objeto ou evento.
- `Cag` (concomitância/co-agência) define um objeto (não evidente) que inicia implicitamente um evento de forma simultânea.
- `Cao` (concomitância/co-objeto com atributos) define um objeto (não evidente) dentro de um estado, efetuando-se de forma simultânea.
- `Cnt` (conteúdo) define um conceito equivalente.
- `Cob` (concomitância/co-objeto afetado) define um objeto que está diretamente afetado por um evento implícito de forma simultânea ou por uma condição.
- `Con` (condição) define uma condição que causa (voluntária ou involuntariamente) a ocorrência de um evento.
- `Coo` (co-ocorrência) define uma progressão simultânea de eventos.
- `Dur` (duração) define um período de tempo de existência de um estado ou evento.
- `Fmt` (alcance: Origem-destino) define uma abrangência de objetos ou eventos.
- `Frm` (origem) define a origem de um objeto.
- `Gol` (objetivo: estado final) define o local (físico ou lógico) de um agente/objeto relativo a um evento.
- `Ins` (instrumento) define o instrumento utilizado para levar a cabo um evento.
- `Man` (maneira, modo) define a maneira de levar a cabo um evento ou caracterizar um estado.
- `Met` (método) define um método para levar a cabo um evento.
- `Mod` (modificador) define um objeto evidenciando-o de forma restrita.
- `Nam` (nome) define o nome de um objeto.
- `Obj` (objeto afetado) define um objeto que está diretamente afetado por um evento ou estado.
- `Opl` (lugar objetivo) define um lugar onde um evento acontece.
- `Or` (disjunção) define uma relação de disjunção entre dois conceitos.

- `Per` (proporção, taxa ou distribuição) define uma base ou unidade de proporção, taxa ou distribuição.
- `Plc` (lugar) define o lugar onde acontece um evento ou está presente um objeto com um estado determinado.
- `Plf` (lugar inicial) define o lugar onde um evento começa ou define o lugar onde um estado começa a ser verdadeiro.
- `Plt` (lugar final) define o lugar onde um evento termina ou define o lugar onde um estado começa a ser falso.
- `Pof` (parte-de) define um conceito que forma parte de outro.
- `Pos` (possuidor) define o possuidor de um objeto.
- `Ptn` (companheiro) define um objeto de forma não evidente, mas imprescindível para que se comece uma ação.
- `Pur` (propósito ou objetivo) define o propósito ou objetivo de um agente relacionado a um evento ou o propósito de um objeto que existe.
- `Qua` (quantidade) define a unidade associada à quantidade de um objeto ou grau de mudança.
- `Rsn` (razão) define a razão pela qual um evento ou um determinado estado aconteceria.
- `Scn` (cenário) define um mundo virtual onde acontece um evento, ou um estado se torna verdadeiro, ou onde um objeto existe.
- `Seq` (seqüência) define um evento ou estado prévio ao evento ou estado que está sendo evidenciado.
- `Src` (estado inicial) define o local (físico ou lógico) ou estado de um agente/objeto relativo a um evento.
- `Tim` (tempo) define o tempo em que acontece um evento ou o tempo em que um estado é válido.
- `Tmf` (tempo inicial) define o tempo em que um evento começa ou o tempo inicial em que um estado começa a ser verdadeiro.
- `Tmt` (tempo final) define o tempo em que um evento termina ou o tempo em que um estado se torna falso.
- `To` (destino) define o destino de um objeto.

- `via` (lugar ou estado intermediário) define o lugar ou estado intermediário de um evento.

Esses são todas os rótulos de relação incluídos na linguagem que foram definidos até o momento. Um exemplo utilizando rótulos de relação mostra-se na **Figura 5.4** uma mensagem representada na linguagem UCL, construídas a partir das seguintes sentenças.

UNL is a common language that would be used for network communications

No exemplo apresentado na **Figura 5.4**, pode-se observar nas primeiras linhas a definição do arquivo Sentence.dtd, que validará a mensagem com a gramática definida. No seguinte tópico, mostrar-se-ão os rótulos de atributos utilizados na UCL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sentence SYSTEM "Sentence.dtd">
<sentence>
     <uw id="uw00" head="language">
        <icl direction="to">
           <uw head="abstract thing"/>
        </icl>
        <tense attribute="present"/>
        <focus attribute="entry"/>
     </uw>
     <uw id="uw01" head="UNL">
        <icl direction="to">
           <uw head="language"/>
        </icl>
        <focus attribute="topic"/>
     </uw>
     <uw id="uw02" head="common">
        <aoj direction="to">
           <uw head="thing"/>
        </aoj>
     </uw>
     <uw id="uw03" head="use">
        <icl direction="to">
           <uw head="do"/>
        </icl>
        <tense attribute="present"/>
     </uw>
     <uw id="uw04" head="language">
        < icl direction="to">
           <uw head="abstract thing"/>
        </icl>
        <tense attribute="present"/>
        <focus attribute="entry"/>
     </uw>
     <uw id="uw05" head="communication">
        < icl direction="to">
           <uw head="action">
        </icl>
```

```
        <convention attribute="pl"/>
      </uw>
      <uw id="uw06" head=”network”>
        <icl direction=”to”>
          <uw head=”thing”>
        </icl>
      </uw>
   <relation label="aoj" uw-id1="uw00" uw-id2=”uw01”/>
   <relation label="mod" uw-id1="uw00" uw-id2=”uw02”/>
   <relation label="obj" uw-id1="uw03" uw-id2=”uw04”/>
   <relation label="pur" uw-id1="uw03" uw-id2=”uw05”/>
   <relation label="mod" uw-id1="uw05" uw-id2=”uw06”/>
</sentence>
```

Figura 5.4 Definição de Rótulos de relação em um documento XML

5.5.3 Rótulos de atributo

Os rótulos de atributos foram introduzidos para limitar o significado dos conceitos definidos no tópico 5.5.1. Com os rótulos de atributos pode-se particularizar o significado de uma uw. As representações destes atributos estão agrupadas de forma tal que se possa identificar as informações de tempo verbal, o aspecto do conceito ou estrutura da mensagem, entre outros. Estes rótulos podem ser inseridos dentro do escopo da tag uw (<uw> ... </uw>) até se conseguir restringir seu significado como desejado. A sintaxe geral de um rótulo de atributo é mostrada na **Figura 5.5**.

```
<!=========== Definição da sintaxe ==================-->
<!ELEMENT uw ((%label;)*, (tense | aspect | reference | focus |
            attitudes | viewpoint | convention)*)>
<!=========== lista de atributos ==================-->
<!ELEMENT tense EMPTY>
<!ATTLIST tense attribute(past | present | future) #REQUIRED>
<!ELEMENT aspect EMPTY>
<!ATTLIST aspect attribute(begin-soon | begin-just | progress | end-
          soon | end-just | complete | state | repeat) #REQUIRED>
<!ELEMENT reference EMPTY>
<!ATTLIST reference attribute(generic | def | indef | not | order)
          #REQUIRED>
<!ELEMENT focus EMPTY>
<!ATTLIST focus attribute(emphasis | entry | qfocus | theme | title
          | topic) #REQUIRED>
<!ELEMENT attitudes EMPTY>
<!ATTLIST   attitudes   attribute(affirmative   |   confirmation   |
          exclamation | imperative | interrogative | invitation |
```

```
            politeness | respect | vocative) #REQUIRED>
    <!ELEMENT viewpoint EMPTY>
    <!ATTLIST viewpoint attribute(ability | ability-past | apodosis-real
            | apodosis-unreal | apodosis-cond | conclusion | custom |
            ... possibility | probability | should | unexpected-
            presumption | unexpected-consequence | will) #REQUIRED>
    <!ELEMENT convention EMPTY>
    <!ATTLIST convention attribute(angle_bracket | double_parenthesis |
            double_quotation | parenthesis | pl | single_quotation |
            square_bracket) #REQUIRED>
```

Figura 5.5 Sintaxe geral de um rótulo de atributo

Os rótulos de atributos estão agrupados em sete subgrupos, facilitando a escolha do atributo. A seguir, apresenta-se uma descrição deste grupo e os atributos envolvidos.

a) `Tense`, subgrupo que expressa tempo verbal. Estão agrupados os atributos que outorgam localização no tempo ao conceito. Temos três rótulos de atributo que estão envolvidos neste subgrupo:

- `Past`: refere-se ao evento que aconteceu no passado.
- `Present`: refere-se a um evento que está acontecendo.
- `Future`: refere-se ao evento que irá acontecer.

b) `Aspect`, subgrupo que se refere à noção de aspecto da língua inglesa. Está relacionado com o estado de um evento/objeto que forma parte da mensagem.

- `Begin-soon`: está relacionado com o evento que vai começar.
- `Begin-just`: está relacionado com o evento que recém começou.
- `Progress`: está relacionado com o evento que está em progressão.
- `End-soon`: está relacionado com o evento que está quase terminando.
- `End-just`: está relacionado com o evento que recém terminou.
- `Complete`: está relacionado com o evento já concluído.
- `State`: está relacionado com o estado final de um evento/objeto após a ocorrência de um outro evento.
- `Repeat`: está relacionado com a definição de um evento repetitivo, envolvendo o mesmo agente/objeto.

c) `Reference`, subgrupo que indica se o conceito está descrevendo um objeto, ou um pequeno grupo, ou a todos. Os seguintes rótulos de atributos são usados para explicitar a referência de um objeto.

- `Generic`: descreve que o conceito é genérico.
- `Def`: descreve que o conceito já foi referido.
- `Indef`: descreve que o conceito não pertence a uma classe específica.
- `Not`: corresponde à negação.
- `Order`: determina a ordem do conceito.

d) `Focus`, subgrupo que permite a quem constrói a mensagem poder evidenciar ou enfatizar um conceito ou parte da mensagem. Desta forma, pode-se indicar na mensagem a importância do que se pretende descrever. Geralmente está relacionado com a estrutura da mensagem.

- `Emphasis`: enfatiza um conceito.
- `Entry`: descreve o conceito como o princípio ou ponto de início de uma mensagem. É indispensável em uma mensagem UCL.
- `Qfocus`: informa o conceito que recebe o foco na mensagem.
- `Theme`: informação temática.
- `Title`: aponta um conceito correspondente ao título, como forma sentencial.

e) `Attitude`, refere-se às atitudes ou emoções que um agente pode adotar, seja direta ou indiretamente. Isto inclui respeito, cortesia relacionado ao que se transfere na mensagem.

- `Affirmative`: afirma um evento.
- `Confirmation`: requer confirmação do evento.
- `Exclamation`: representa o sentimento de exclamação sobre algum evento.
- `Imperative`: declaração imperativa.
- `Interrogative`: declaração interrogativa sobre um evento.
- `Invitation`: incentivar a fazer ou realizar algum evento.
- `Politeness`: representa o sentimento de cortesia.

- `Respect`: representa o sentimento de consideração ou respeito.
- `Vocative`: representação de expressões usuais.

f) `Viewpoint`, refere-se a uma variedade de possibilidades para representar grau de crença, enfatizar alguma coisa, e estender alguma explicação sobre o assunto que se está tratando. Isto vai depender do estado em que se encontre o agente transmissor. Os seguintes rótulos de atributos são usados para esclarecer pontos de vista de informação.

- `Ability`: representa a habilidade, ou capacidade de executar algo.
- `Ability-past`: representa alguma habilidade efetuada no passado.
- `Apodosis-real`: representa o apódose e refere-se a segunda parte de um período gramatical, em relação a primeira, de cujo sentido é complemento.
- `Apodosis-unreal`: representa o apódose.
- `Apodosis-cond`: representa o apódose de forma condicional.
- `Conclusion`: representa uma conclusão de uma primeira parte da mensagem.
- `Custom`: ação habitual, e corresponde ao passado.
- `Expectation`: representa a expectativa sobre outros objetos/agentes.
- `Grant`: conceder permissão para executar algo.
- `Grant-not`: não conceder permissão para a execução de algo.
- `Insistence`: insistência na execução de alguma ação.
- `Intention`: representa a intenção de fazer alguma ação no futuro.
- `Inevitability`: representa a suposição de que algum evento é inevitável.
- `May`: representa a suposição de uma atual possibilidade.
- `Obligation`: obrigar a alguém a executar alguma ação.
- `Obligation-not`: proibir a alguém a executar alguma ação.
- `Possibility`: representa uma razoável possibilidade de que aconteça um evento.
- `Probability`: representa uma razoável probabilidade sobre algo.
- `Should`: representa o sentimento de imposição.
- `Unexpected-presumption`: pressuposição contrária ao que se esperava.

- `Unexpected-consequence`: representa uma conseqüência contrária ao esperado.
- `Will`: representa o sentimento de querer executar algo.

g) `Convention`, este último subgrupo se refere à representação de símbolos convencionais.

- `Angle_bracket`: representa o uso dos símbolos < >.
- `Double_parenthesis`: representa o uso dos símbolos (( )).
- `Double_quotation`: representa o uso dos símbolos “ ”.
- `Parenthesis`: representa o uso dos símbolos ( ).
- `Pl`: representa o plural.
- `Single_quotation`: representa o uso dos símbolos ‘ ’.
- `Square_bracket`: representa o uso dos símbolos [ ].

5.6 Considerações finais

Neste capítulo foi definida a linguagem de comunicação UCL. Ela pode ser situada, de acordo com a descrição apresentada no capítulo 3, entre as linguagens de comunicação envolvendo agentes de software (ACL). Por UCL ser uma linguagem derivada do XML, sua integração com linguagens que suportam a comunicação entre agentes, como KQML e FIPA-ACL, derivadas do Lisp fica mais difícil (mas não impossível). Por outro lado, a sua integração com linguagens de suporte a comunicação, específicas para Internet, como a SOAP (*Simple Object Access Protocol*) (SOAP, 2000) derivada da XML, é muito facilitada. Essas linguagens, especialmente a SOAP, estão se tornando o padrão para comunicação de mensagens na Internet.

A notação utilizada para especificar a linguagem UCL foi definida em função da meta-linguagem XML: foi usado um documento no formato DTD para definir sua sintaxe. Na especificação da linguagem UCL tentou-se manter o poder representativo herdado da UNL na expressão de sentenças ao mesmo tempo que se adicionou a portabilidade e a compatibilidade com padrões da Internet da linguagem XML.

A linguagem UCL é compatível com a linguagem UNL, desta forma, no futuro, mensagens codificadas na linguagem UNL poderão ser representadas em UCL e vice-versa. Utilizando-se a especificação da linguagem UCL apresentada neste capítulo, implementou-se um protótipo que transforma sentenças do inglês para UCL e de UCL para o inglês, que será descrito no próximo capítulo.

6. UM ENCONVERTER-DECONVERTER UCL

6.1 Considerações iniciais

Neste capítulo é mostrada a implementação de um protótipo de um programa *Enconverter-Deconverter*. Esses termos, *enconverter* e *deconverter*, vieram do projeto da UNL (Uchida, 1999) e significam, respectivamente, um programa que converte sentenças de uma linguagem natural para uma interlíngua e um programa que converte de uma interlíngua para linguagem natural. No caso desse trabalho a interlíngua é a UCL e o protótipo trabalha com a língua inglesa como linguagem natural (ele também trabalha com o francês).

O detalhamento do desenvolvimento da linguagem está baseado na descrição da sua especificação e dos recursos apresentados nos capítulos anteriores. Também é mostrada uma arquitetura de um sistema que utilizaria a linguagem UCL.

6.2 Gramática

Para discutir a implementação do protótipo, será necessário situar alguns conceitos manipulados pela implementação.

A linguagem UCL é especificada através de regras que descrevem a estrutura da mensagem. O conjunto destas regras compõe a gramática da linguagem. O reconhecimento de uma mensagem UCL é feito pelo interpretador JAXP da Sun (Sun, 2001). A leitura de cada item, denominado "*token*", é feita pelo analisador léxico do JAXP.

A seguir, na **Tabela 6.1,** são mostradas as principais etiquetas que representam a estrutura de uma mensagem UCL que contem informações sobre a sentença, conceitos (UWs) e relações.

Tabela 6.1 Principais etiquetas para a construção de mensagens UCL

Etiqueta	Descrição
<sentence>	Inicio da mensagem
</sentence>	Fim da mensagem.
<uw>	Inicio de um conceito/objeto.
</uw>	Fim de um conceito/objeto.
<relation>	Definição de uma relação envolvendo dois conceitos previamente definidos.
</relation>	Fim de uma relação que envolve dois conceitos

Além das etiquetas mostradas acima, temos também: os rótulos de atributo e rótulos de relação que foram especificados no capítulo 5. O Apêndice B mostra a gramática completa da linguagem UCL definida em um documento DTD.

6.3 Implementando o protótipo

O protótipo é implementado seguindo as características mostradas no capítulo 5. Elas podem ser resumidas em: a) permitir que a linguagem seja de fácil integração à rede Internet; b) ser de fácil utilização; c) permitir sua escalabilidade e estensibilidade. A estrutura das mensagens UCL está baseada no DTD definida e descrita no capítulo 5. Para validar os documentos XML com o DTD, precisou-se de um interpretador de documentos XML, no caso o JAXP que usa a API DOM.

Para a representação semântica da linguagem, precisa-se de uma ontologia que represente os conceitos envolvidos na mensagem. Neste ponto, adotou-se a ontologia usada no programa *ThoughtTreasure*. Ela inclui a utilização de bibliotecas que permitem manipular os conceitos da ontologia, realizar consultas na rede de conceitos, e analisar a hierarquia.

Além disso, implementou-se em Java uma API UCL que permite gerar e manipular as mensagens UCL. Na **Figura 6.1** são mostradas as classes que foram utilizadas e a interface que foi implementada no protótipo.

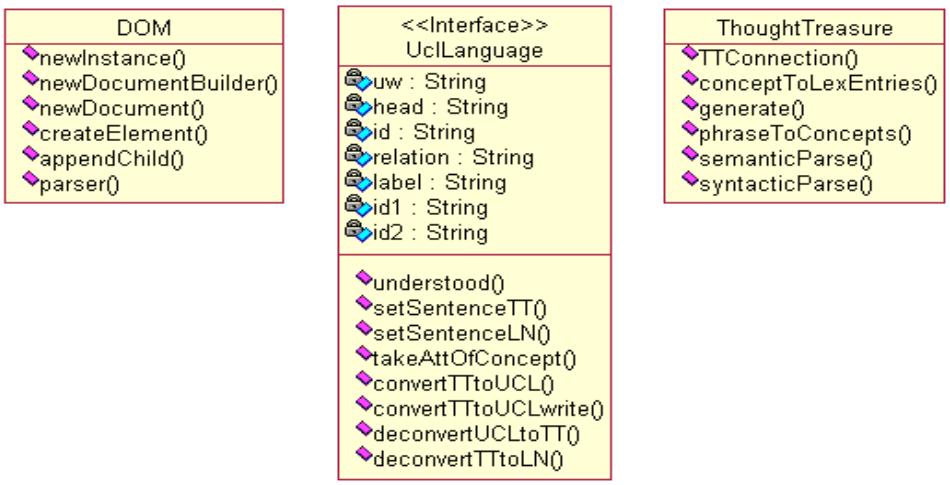


Figura 6.1 Classes e interface do protótipo.

Na **Figura 6.2** é mostrado o diagrama de classes da implementação do protótipo, este diagrama mostra quais são as relações entre a Interface *UclLanguage* com as Classes.

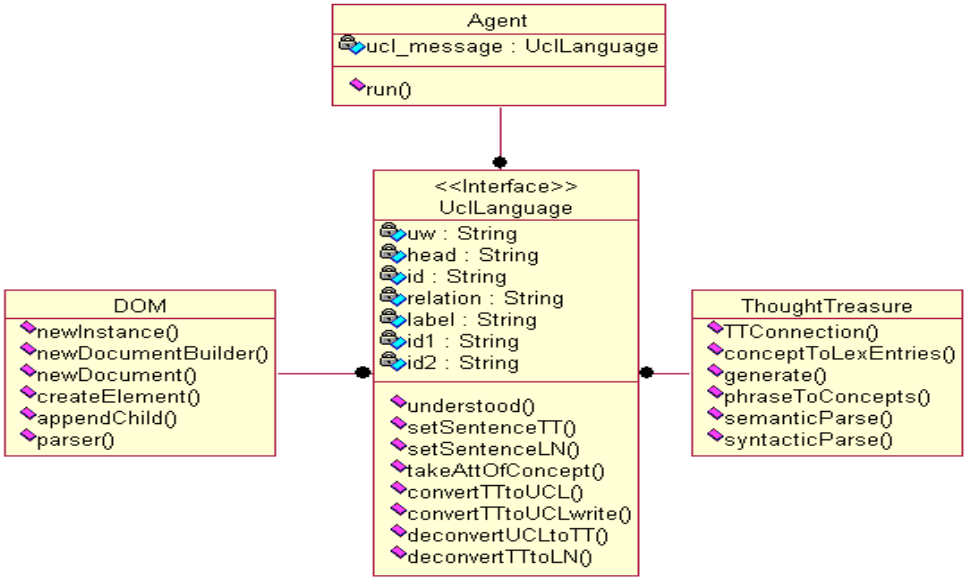


Figura 6.2 Diagrama de Classes.

6.3.1 O *enconverter*

O processo de codificação, feito pelo *enconverter*, segue a seqüência de eventos mostrada no diagrama da **Figura 6.3.** Nele o protótipo faz uso da interface *UclLanguage* para gerar mensagens UCL. O processo começa quando o *enconverter* chama o método *understood* da interface *UclLanguage,* desta forma se interpreta a sentença em linguagem natural e são retornadas várias interpretações desta sentença (dependendo de como a sentença foi entendidas pelo programa *ThoughtTreasure*). Na continuação, o usuário escolhe a interpretação mais adequada. A partir desta interpretação escolhida é realizado um processo de codificação para a linguagem UCL, resultando numa mensagem UCL. Esta mensagem pode ser manipulada e/ou enviada a outro agente.

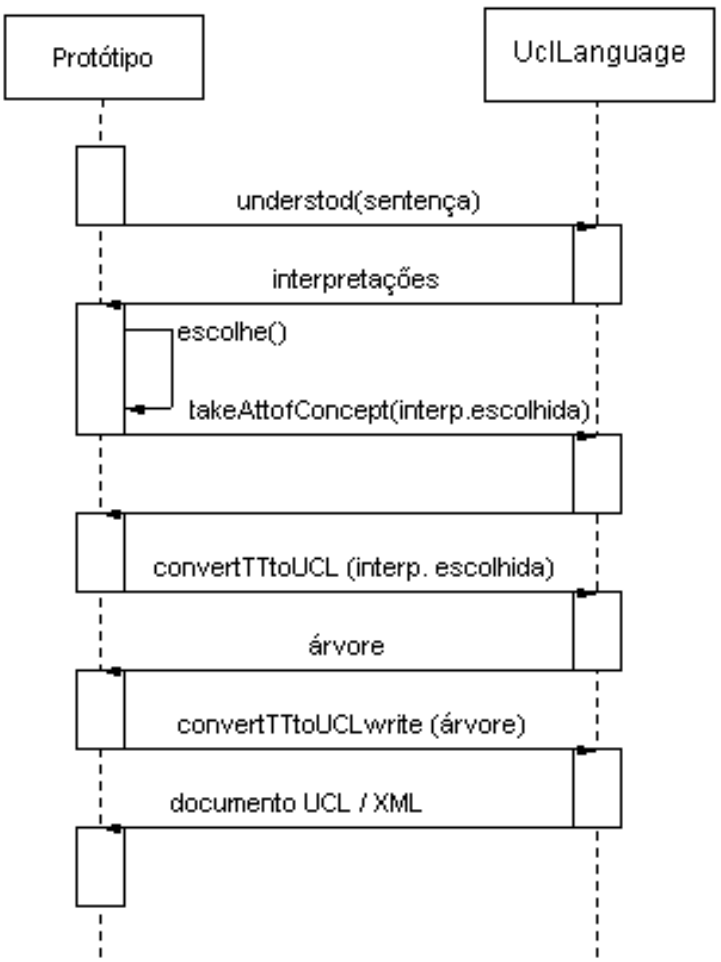


Figura 6.3 Diagrama de seqüência de eventos.

Segue uma descrição mais detalhada do processo de geração das mensagens UCL e uma descrição do funcionamento de cada método no processo.

Interpretando uma sentença em linguagem natural

O método *Understood* é chamado para realizar a interpretação de uma sentença em linguagem natural, e retorna uma lista de sentenças que expressam a mensagem inicial. Como resultado da aplicação deste método, pode-se obter dois tipos de resultados (**Figura 6.4)**. No primeiro caso, o resultado pode ser nulo, significando que alguma entrada léxica não está cadastrada na ontologia e, por isso, não foi conseguida nenhuma representação conceitual. No segundo caso, o resultado pode ser uma lista de alternativas, significando que a sentença que se deu como entrada têm várias representações semânticas (ou várias representações que podem ser entendidas). Neste último caso o sistema não pode optar por uma delas, pois isso depende muito do contexto em que se trabalha. Neste caso, deixa-se para o usuário a escolha de qual das interpretações é a mais apropriada.

Para poder avaliar sintaticamente e semanticamente a sentença, que está em linguagem natural, são criadas internamente duas estruturas. A primeira refere-se a uma árvore de nós representando a avaliação sintática e a segunda é outra árvore de nós representando a avaliação semântica.

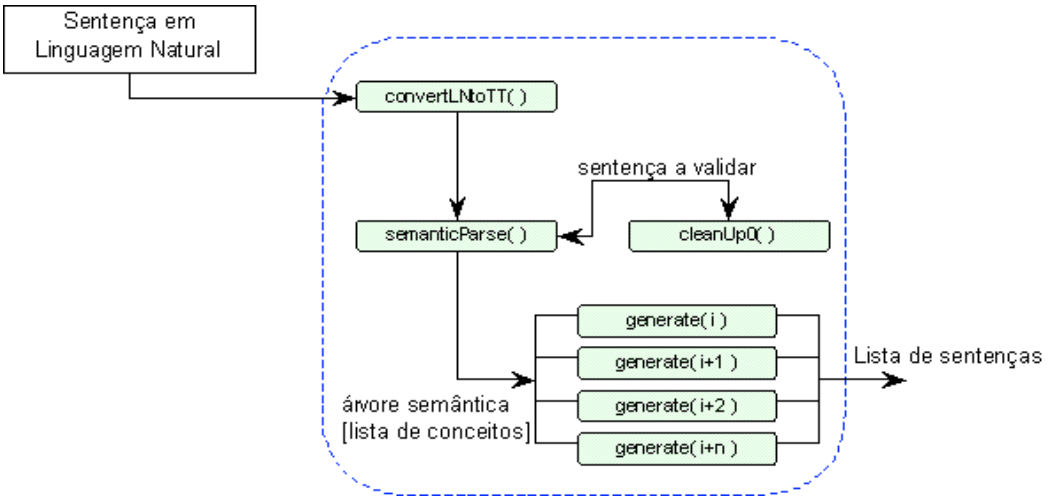


Figura 6.4 Interpretando uma sentença em linguagem natural

Cadastrar a lista de conceitos

Este método (*setSentenceTT*) cadastra a lista de conceitos dentro do *enconverter*. Esta lista de conceitos foi obtida do método *understood* apresentado anteriormente.

Obter os atributos de cada conceito

Este método (*takeAttOfConcept*) procura todos os atributos gramaticais das entradas léxicas que pertencem aos conceitos que foram cadastrados no *enconverter*. Estes atributos gramaticais são posteriormente inseridos em cada conceito da mensagem UCL. Em conseqüência, pode-se descrever com maior precisão o significado de cada conceito.

Considerando a árvore semântica, que contém os conceitos envolvidos na mensagem, extrai-se cada conceito para posteriormente procurar na ontologia as entradas léxicas e seus atributos gramaticais.

Transformar a lista de conceitos na mensagem UCL

Este método (*ConvertTTtoUCL*) cria a mensagem UCL baseado nos conceitos anteriormente cadastrados no *enconverter*. Este método utiliza a API DOM do JAXP para criar a árvore de nós que representará posteriormente a mensagem UCL. A construção desta árvore baseia-se na gramática da linguagem UCL especificada no capítulo 5. A **Figura 6.5** mostra que este processo se realiza em duas etapas.

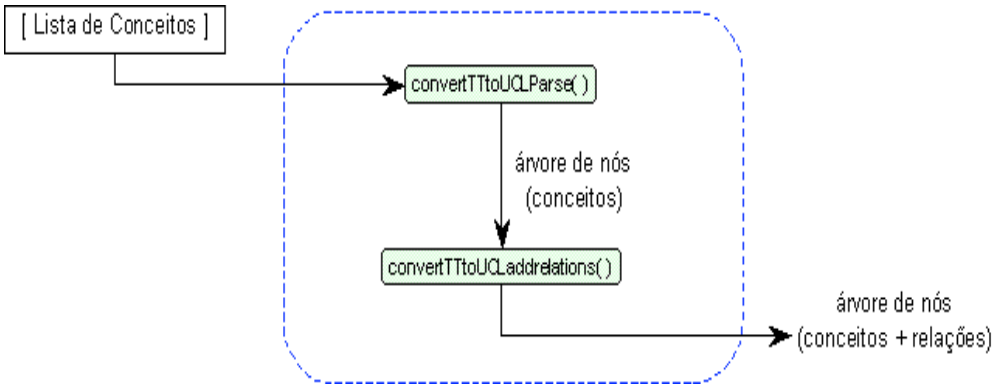


Figura 6.5 Transforma a lista de conceitos na mensagem UCL

Na primeira etapa são gerados os conceitos na forma de UWs**.** Além disso, é inserido um identificador único para cada conceito. Como é mostrado na **Figura 6.6,** a entrada inicial é a lista de conceitos, a partir dela se verifica a existência de cada conceito na ontologia e se obtém o conceito pai de cada um (desta forma se descobre a que hierarquia o conceito pertence). É usada a tag `icl` para indicar o conceito pai de cada um.

Na segunda etapa, as relações existentes entre os conceitos são determinadas e elas são explicitadas através da tag `relation`. Este processo dá forma e estrutura à mensagem. Para

explicitar estas relações, são utilizados os identificadores definidos na primeira etapa. Cada conceito definido anteriormente é referenciado.

O resultado da primeira etapa mais o resultado da segunda etapa formam uma única árvore de nós que representa a mensagem UCL.

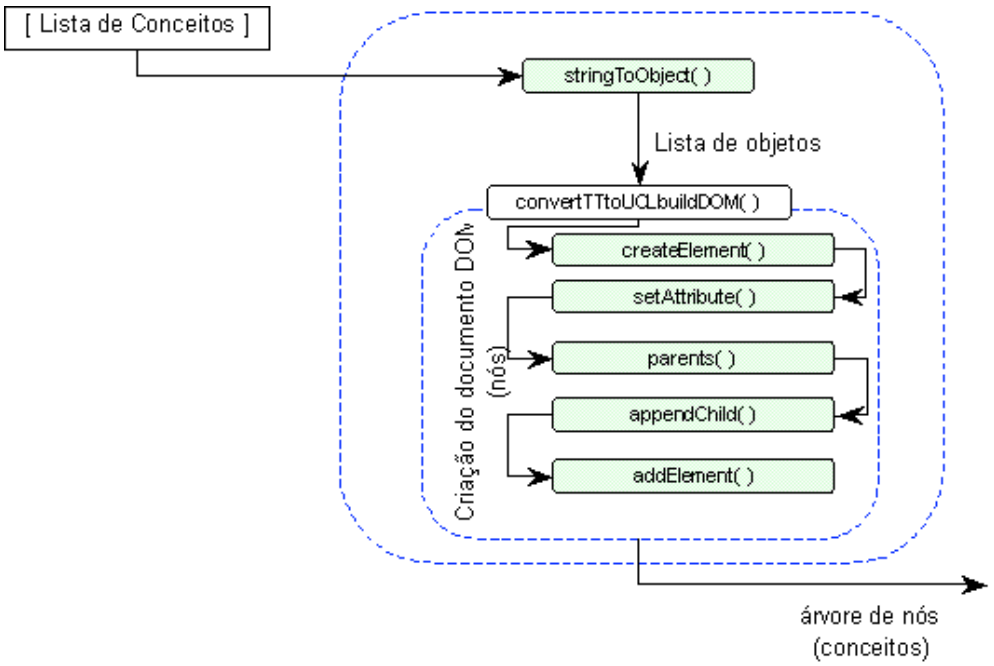


Figura 6.6 Criação de árvore de nós de um documento UCL.

Criar fisicamente a mensagem UCL como documento XML

Este método (*convertTTtoUCLwrite*) escreve fisicamente a mensagem UCL em um arquivo que pode ser enviado a outro agente. Inicialmente, tem-se a estrutura da mensagem em uma árvore de nós criada pela API DOM, onde estão embutidos os conceitos da mensagem e suas relações. Para criar o documento XML, a partir desta árvore, chama-se o método *write* da API DOM do JAXP, obtendo-se desta forma um documento contendo a mensagem na linguagem UCL.

6.3.2 O *deconverter*

O processo decodificação é iniciado com a recepção de uma mensagem codificada em UCL, para posteriormente ser transformada em uma sentença em linguagem natural equivalente.

Transformar a mensagem UCL na lista de conceitos

Este método (*deconvertUCLtoTT*), como mostrado na **Figura 6.7**, decodifica a mensagem UCL transformando-a em uma lista de conceitos. Para se obter esta lista de conceitos, primeiro precisa-se extrair da mensagem UCL as relações existentes, estas relações são as que indicam a seqüência e prioridade dos conceitos. Após a obtenção destas relações, se constrói a lista de conceitos percorrendo o conjunto de relações. Neste processo, utiliza-se o identificador (atributo `id`) de cada conceito para obter seus nomes.

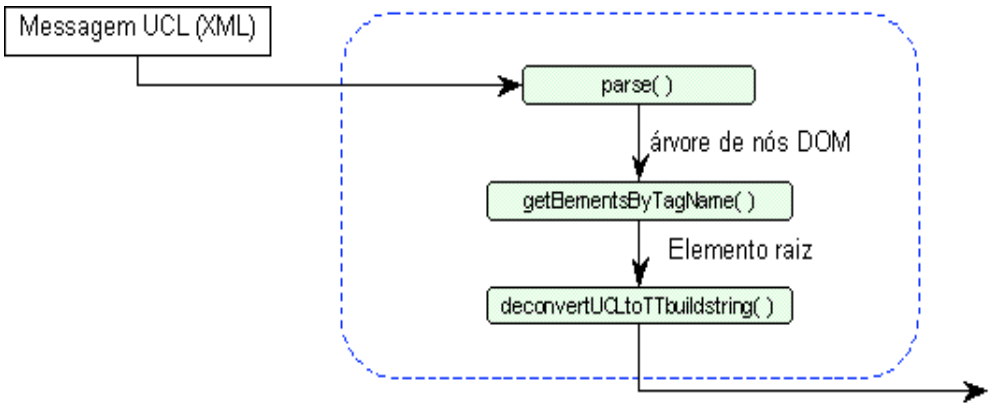


Figura 6.7 Transforma a mensagem UCL na lista de conceitos

Transformar a lista de conceitos em uma sentença em linguagem natural

Como pode ser observado na **Figura 6.8**, este método (*deconverterTTtoLN*) transforma a lista de conceitos em uma sentença em linguagem natural, que representa a sentença UCL original. A sentença em linguagem natural, obtida deste método, pode ser em Inglês ou em francês, isto porque a ontologia utilizada contém entradas léxicas para as duas línguas.

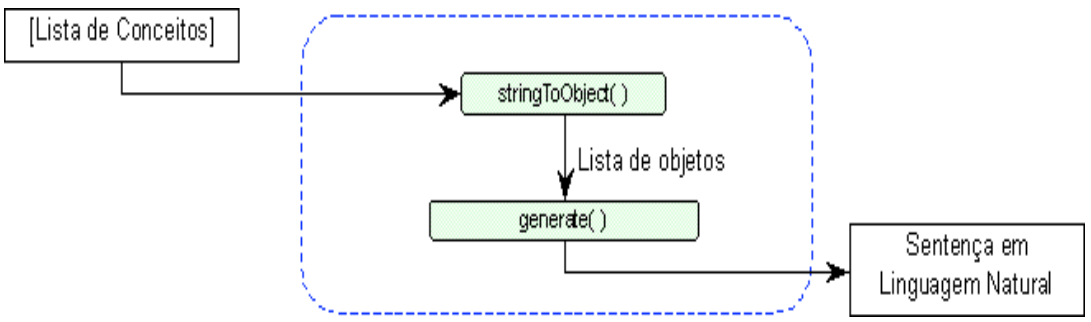


Figura 6.8 Transformar a lista de conceitos numa sentença em linguagem natural

6.4 Arquitetura de um sistema de comunicação em UCL

A utilização da linguagem UCL, entre dois agentes de software, pode ser implementada através da arquitetura mostrada na **Figura 6.9**.

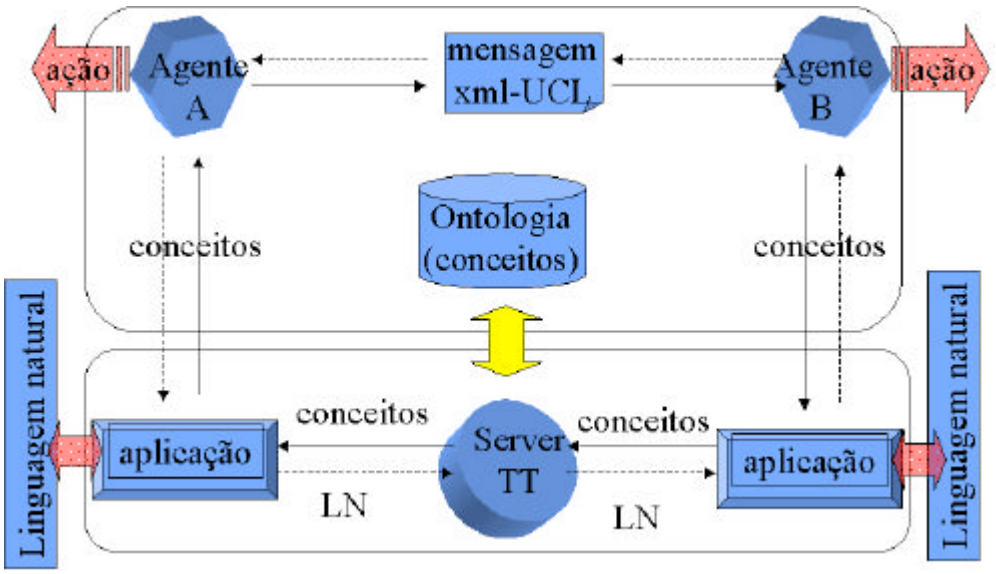


Figura 6.9 Arquitetura de um sistema que utiliza a linguagem UCL

No caso em que o agente A precisa se comunicar com o agente B, existem as seguintes possibilidades de comunicação:

- O usuário quer se comunicar com o Agente A: O fluxo inicia-se quando o usuário cria, em linguagem natural, uma mensagem que vai ser dirigida ao Agente A. Esta mensagem é a entrada para o *enconverter* (dentro da interface de usuário), que inicia a comunicação com o servidor TT (*ThoughtTreasure)* para efetivar o *parser* da sentença. A seguir, obtém-se do servidor TT a lista de conceitos da sentença. Com esta lista de conceitos, é construída a mensagem UCL, para finalmente ser enviada ao Agente A. O Agente A, em função da mensagem, pode efetuar alguma ação.

- O usuário quer se comunicar com o Agente B por meio do Agente A: De forma similar ao caso anterior, o fluxo inicia-se quando o usuário cria, em linguagem natural, uma mensagem que vai ser dirigida ao Agente B. A Interface realiza a comunicação com o servidor TT, para obter a lista de conceitos, e posteriormente gera a

mensagem UCL que será enviada ao Agente A. Este envia a mensagem ao Agente B que o processa.

- O usuário quer se comunicar com outro usuário: Neste caso, a entrada para a aplicação é uma sentença em linguagem natural sobre a qual o *enconverter* realiza o processo de codificação para UCL. Desta forma; o Agente A envia a mensagem codificada ao Agente B, este último realiza a comunicação com o servidor TT e por meio do *deconverter* gera a sentença em linguagem natural para o segundo usuário.

- O Agente A se quer comunicar com o Agente B: Neste caso, o Agente A, que é pro-ativo, inicia uma requisição ou consulta ao Agente B por meio de mensagens UCL. Não é preciso nenhuma codificação ou decodificação para linguagem natural, pois os agentes se comunicam com mensagens baseadas em UCL.

6.5 Considerações Finais

Este capítulo mostrou a implementação de um programa *enconverter-deconverter* usando UCL. A implementação considerou as características requeridas para ser de fácil integração na rede Internet, ser legível, também como permitir sua escalabilidade e estensibilidade.

Foi feito uma descrição da API UCL implementada em Java que auxilia a geração de mensagens UCL. Além disso, foi considerada uma arquitetura de sistema utilizando a linguagem UCL.

Ficou demonstrado que a linguagem UCL atende os requisitos exigidos para atuar como interlíngua e como linguagem de comunicação entre agentes de software. O protótipo de *enconverter-deconverter*, feito em Java e usando a versão de servidor do programa ThoughtTreasure, foi usado para testar a viabilidade da UCL como linguagem de comunicação.

7. CONCLUSÃO

7.1. Considerações iniciais

Neste capítulo é feita a síntese dos resultados deste trabalho. Ressaltam-se algumas decisões de projeto, contribuições e sugestões de futuras pesquisas que podem vir a ser desenvolvidas a partir deste trabalho.

7.2. Decisões de Projeto

Neste item são feitas as descrições de algumas das decisões de projeto assumidas durante este trabalho, de forma a ressaltar aquelas já descritas no texto e complementar outras que não foram anteriormente citadas.

Durante a definição da linguagem UCL (*Universal Communication Language)* procurou-se abranger todos os conceitos teóricos da linguagem modelo, que neste caso foi a UNL (*Universal Networking Language)* (Uchida, 1999). Desta forma se tentou preservar o poder representativo desta linguagem.

Os vários requisitos de uma linguagem para comunicação entre agentes, tais como: uma linguagem com ênfase na semântica da mensagem, facilidade de utilização, e compatibilidade com as linguagens usadas na Internet, dentre outros, levaram a optar por recursos como a linguagem XML, o uso de ontologias e a base teórica derivada da UNL.

Sendo a linguagem UCL definida para ser utilizada na comunicação entre agentes, foi importante ressaltar como as mensagens iriam representar conceitos do mundo real, neste caso optou-se por utilizar ontologias como meio de representação. Existiam algumas bases de dados ontológicas que poderiam ser usadas, tais como WordNet (Miller, 1995b), Cyc (Lenat, 1995), *ThoughtTreasure (TT)* (Mueller, 1998). Algumas destas bases ontológicas eram

proprietárias, o que dificultava a utilização das mesmas, em outras não existia uma forma clara de acessar os dados (por exemplo através de uma API). Neste sentido a única que cumpria os requisitos de disponibilidade (não proprietária) e oferecia uma API para acesso à base de dados ontológica foi a base do programa *ThoughtTreasure*.

Uma outra razão para a utilização da base ontológica do *ThoughtTreasure,* foi porque o programa continha, juntamente com os conceitos ontológicos, vários atributos para representar linguagem natural que pode ser processada através da sua API TT em Java. Desta forma foi possível não só a especificação da linguagem proposta (UCL), como a escrita de um programa *enconverter-deconverter* para sentenças em linguagem natural (em inglês e francês).

7.3. Contribuições

As pesquisas em linguagens de comunicação entre agentes vêm acompanhando a evolução dos sistemas multi-agentes, na medida que permitem aos agentes comunicar e cooperar entre si para resolver um problema.

Em tal sentido, este trabalho contribui com a especificação de uma linguagem, a UCL, que considera a importância da semântica nas mensagens a serem representadas. Esta linguagem facilita a representação de sentenças (mensagens) em linguagem natural, utilizando ontologias para reduzir confusões conceituais e terminológicas que por ventura existirem na construção da sentença. Além disso, a utilização de ontologias na linguagem auxilia na interoperabilidade entre diferentes aplicações que trocam mensagens. A UCL é uma concorrente direta da UNL, tendo as vantagens de:

1. Ser um software aberto, usando a *GNU Public License*.
2. Ser baseada numa linguagem padrão, a XML, já amplamente difundida na Internet, o que facilita sua integração em browsers (que já trabalham com XML ou HTML).
3. Ter como alvo não só o papel de interlíngua, mas também o de linguagem de comunicação e programação (*Scripting*) de agentes de software.

O protótipo de *enconverter-deconverter*, descrito no capítulo 6, serve como ferramenta para experimentação e teste da proposta de linguagem apresentada. Esse protótipo usou

ferramentas e programas abertos, estando disponível para uso de todos sob a *GPL (GNU Public License*) para futuros aprimoramentos.

7.4. Sugestões para Trabalhos Futuros

Neste item apresentam-se sugestões de outros estudos que podem ser motivados a partir desta pesquisa.

Uma extensão deste trabalho é implementar um interpretador da linguagem UCL para agentes de software. Desta forma a UCL poderá operar como uma linguagem de programação (*scripting*), permitindo aos agentes de software executar as semânticas das ações efetuando algum tipo de tarefa. Apesar da UCL ter sido especificada também para esse tipo de aplicação, o protótipo implementado não testou esta faceta da linguagem, deixando-a para trabalhos futuros dentro do grupo.

O protótipo do *enconverter-deconverter* pode ser melhorado para se tornar uma ferramenta para comunicação entre pessoas de línguas diferentes. Reusando material do projeto UNL da UNU (*United Nations University*), a medida que este se torne disponível publicamente, ou partindo para soluções próprias, baseadas ou não no *ThoughtTreasure*, no processo de desenvolvimento normal de um software aberto.

7.5. Considerações Finais

O objetivo deste trabalho foi a especificação de uma nova ACL, a *Universal Communication Language* (UCL), que se preocupa com a descrição da estrutura das mensagens, com o modelo semântico e com suporte a protocolos para interação entre agentes (de software ou humanos) na Internet.

Este capítulo apresentou as conclusões deste trabalho, bem como suas contribuições para a comunidade e sugestões para a sua continuação.

A principal contribuição deste trabalho foi a especificação da UCL e a demonstração de seu funcionamento através de um protótipo de *enconverter-deconverter*.

APÊNDICE A – DESCRIÇÃO DA LINGUAGEM UNIVERSAL NETWORKING LANGUAGE (UNL) (UCHIDA, 1999)

Notação utilizada para descrever a sintaxe da UNL:

```
< >             símbolos não terminais / variáveis
" "             cadeia de caracteres
::=             ... é definido como ...
|               disjunção, "ou".
[ ]             elemento opcional
{ }             elemento alternativo
{ }...          repete-se mais de zero vezes.
```

Estrutura Interna de relações binárias

```
<Binary relation>  ::=  <Relational Label>  [":"<Compound UW-ID>]
                    "(" {<UW1>|":"<Compound UW-ID1>} ","
                        {<UW2>|":"<Compound UW-ID2>} ")"
```

Formato de UNL

A estrutura de um documento UNL é expresso usando as seguintes etiquetas:

```
[D]   Início do documento
[/D]  Fim do documento
[P]   Inicio do parágrafo
[/P]  Fim do parágrafo
[S]   Inicio da sentença
[/S]  Fim da sentença
```

Um documento UNL é construído da seguinte forma:

```
[D] Início do documento
[P] Início do parágrafo
[S] Início da sentença
... expressão UNL
[/S] Fim da sentença
... Repetição de [S]...[/S]
[/P] Fim do parágrafo
```

```
... Repetição de [P]...[/P]
[/D] Fim do documento
... Repetição de [D]...[/D]
```

Uma expressão é identificada com as seguintes etiquetas:

{unl} Início da expressão UNL
{/unl} Fim da expressão UNL

Em uma expressão UNL existem três tipos de informação como, relações binárias, Uws, relações binárias codificadas. Na continuação, mostra-se as etiquetas que são usadas para diferenciar esta informação:

```
[W]    Inicio das UWs
[/W]   Fim das UWs t
[R]    Inicio das relações binárias
[/R]   Fim das relações binárias
```

A sintaxe das relações binárias, Uws e relações binárias codificadas são as seguintes:

```
<Binary relation> ::= <Relation Label> [":"<Compound UW-ID>]
                  "(" {<UW1>|":"<Compound UW-ID1>} ","
                      {<UW2>|":"<Compound UW-ID2>} ")"

<UW> ::= <Head Word> [<Constraint List>] [ ":" <UW-ID>]
                     [ "." <Attribute List>]

<Encoded Binary Relation> := {<UW-ID> | <Compound UW-ID>}

<Relation Label> ::= [":" <Compound UW-ID>] {<UW-ID> |
                     <Compound UW-ID>}
```

Exemplos de expressões UNL

Exemplo.1) Monkey eats bananas.

```
[S]
{unl}
[W]
eat(icl>do).@present.@entry:00
monkey(icl>animal).@generic:01
banana(icl>food).@generic:02
[/W]
[R]
00agt01
00obj02
[/R]
[/S]
```

Exemplo.2) UNL is a common language that would be used for network communications.

```
[S]
{unl}
[W]
language(icl>abstract thing).@present.@entry:00
UNL(icl>language).@topic:01
common(aoj>thing):02
use(icl>do).@present:03
language(icl>abstract thing).@present.@entry:04
communication(icl>action).@pl:05
network(icl>thing):06
[/W]
```

Universal Words (UWs)

Uma UW (Universal Word) representa um conceito simples ou um conceito composto. Os simples são conceitos unitários chamados de Uws (Universal Words). Na continuação se mostra a sintaxe de uma UW.

```
<UW> ::= <Head Word> [<Constraint List>] [ ":" <IUW-ID>]
         [ "." <Attribute List>]
<Head Word> ::= <character>...
<Constraint List> ::= "(" <Constraint> [ ","
                          <Constraint>]... ")"
<Attribute List> ::= <Attribute Label> [ "." ]...
<UW-ID> ::= {<upper case alphabetical character>
           |<digit>}
           {<upper case alphabetical character> |<digit>}
<Constraint> ::= <Relation Label> { ">" | "<" } <UW>
           [<Constraint List>] |
           <Relation Label> { "<" | ">" } <UW>
           [<Constraint List>]
           [ { ">" | "<" } <UW> [<Constraint List>] ] ...
<Attribute Label> ::= "@volitional" | "@reason" | "@past" | ....
<Relation Label> ::= "agt" | and" | "aoj" | "obj" | "icl" | ...
<digit> ::= 0 | 1 | 2 | ... | 9
<upper case alphabetical character> ::= "A" | ... | "Z"
<character> ::= "a" | ... | "z" | "_" | " " | "#" | "!" |
                "$" | "%" | "=" | "^" | "~" | "|" | "@" |
                "+" | "-" | "<" | ">" | "?" | "'"
```

Na continuação se mostra a sintaxe de uma UW composta.

```
<Compound UW> ::= ":" <Compound UW-ID> ["."<Attribute
                  List>]
<Compound UW-ID> ::= {<upper case alphabetical character>
                  |<digit>} {<upper case alphabetical
                  character> |<digit>}
<Attribute List> ::= <Attribute Label> ["." <Attribute
                  Label>]...
<Attribute Label> ::= "@imperative"|"@may"|"@past" | ...
<digit> ::= 0 | 1 | 2 | ... | 9
<upper case alphabetical character> ::= "A" | ... | "Z"
```

APÊNDICE B – DTD DA UNIVERSAL COMMUNICATION LANGUAGE

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ============================== Entities ==============================-->
<!ENTITY % label "(icl | agt | and | aoj | bas | ben | cag | cao | cnt | cob | con | coo | dur | fmt |
                        frm | gol | ins | man | met | mod | nam | obj | opl | or | per | plc | plf | plt | pof |
                        pos | ptn | pur | qua | rsn | scn | seq | src | tim | tmf | tmt | to | via)">
<!ENTITY % direction "direction (to | from) 'to' ">
<!ENTITY % extension "uw, (%label;)?">

<!-- ======================= Beginning of sentence ==========================-->
<!ELEMENT sentence (uw+, relation*)>

<!--Beginning of UW set-->
<!ELEMENT uw ((%label;)*, (tense | aspect | reference | focus | attitudes | viewpoint | convention)*)>
<!ATTLIST uw
    id ID #IMPLIED
    head CDATA #REQUIRED>

<!-- ====================== Beginning of binary relations ====================-->
<!ELEMENT relation EMPTY>
<!ATTLIST relation
    id ID #IMPLIED
    label %label; #REQUIRED
    uw-id1 IDREF #REQUIRED
    uw-id2 IDREF #REQUIRED>
<!-- ========================== relation label ==============================-->
<!ELEMENT icl (%extension;)+>
<!ATTLIST icl     %direction; >

<!ELEMENT agt (%extension;)>
<!ATTLIST agt    %direction; >

<!ELEMENT and (%extension;)>
<!ATTLIST and   %direction; >

<!ELEMENT aoj (%extension;)>
<!ATTLIST aoj    %direction; >

<!ELEMENT bas (%extension;)>
<!ATTLIST bas   %direction; >

<!ELEMENT ben (%extension;)>
<!ATTLIST bem %direction; >

<!ELEMENT cag (%extension;)>
<!ATTLIST cag   %direction; >

<!ELEMENT cao (%extension;)>
<!ATTLIST cao   %direction; >

<!ELEMENT cnt (%extension;)>
<!ATTLIST cnt    %direction; >

<!ELEMENT cob (%extension;)>
<!ATTLIST cob   %direction; >

<!ELEMENT con (%extension;)>
<!ATTLIST com %direction; >

<!ELEMENT coo (%extension;)>
<!ATTLIST coo   %direction; >
```

```
<!ELEMENT dur (%extension;)>
<!ATTLIST dur    %direction; >

<!ELEMENT fmt (%extension;)>
<!ATTLIST fmt    %direction; >

<!ELEMENT frm (%extension;)>
<!ATTLIST frm    %direction; >

<!ELEMENT gol (%extension;)>
<!ATTLIST gol    %direction; >

<!ELEMENT ins (%extension;)>
<!ATTLIST ins    %direction; >

<!ELEMENT man (%extension;)>
<!ATTLIST man %direction; >

<!ELEMENT met (%extension;)>
<!ATTLIST met    %direction; >

<!ELEMENT mod (%extension;)>
<!ATTLIST mod %direction; >

<!ELEMENT nam (%extension;)>
<!ATTLIST nam %direction; >

<!ELEMENT obj (%extension;)>
<!ATTLIST obj %direction; >

<!ELEMENT opl (%extension;)>
<!ATTLIST opl %direction; >

<!ELEMENT or (%extension;)>
<!ATTLIST or     %direction; >

<!ELEMENT per (%extension;)>
<!ATTLIST per    %direction;
>
<!ELEMENT plc (%extension;)>
<!ATTLIST plc %direction;
>
<!ELEMENT plf (%extension;)>
<!ATTLIST plf %direction; >

<!ELEMENT plt (%extension;)>
<!ATTLIST plt %direction; >

<!ELEMENT pof (%extension;)>
<!ATTLIST pof %direction; >

<!ELEMENT pos (%extension;)>
<!ATTLIST pos   %direction; >

<!ELEMENT ptn (%extension;)>
<!ATTLIST ptn %direction; >

<!ELEMENT pur (%extension;)>
<!ATTLIST pur %direction; >

<!ELEMENT qua (%extension;)>
<!ATTLIST qua   %direction; >

<!ELEMENT rsn (%extension;)>
<!ATTLIST rsn %direction; >

<!ELEMENT scn (%extension;)>
<!ATTLIST scn   %direction; >
```

```
<!ELEMENT seq (%extension;)>
<!ATTLIST seq   %direction; >

<!ELEMENT src (%extension;)>
<!ATTLIST src %direction; >

<!ELEMENT tim (%extension;)>
<!ATTLIST tim %direction; >

<!ELEMENT tmf (%extension;)>
<!ATTLIST tmf %direction; >

<!ELEMENT tmt (%extension;)>
<!ATTLIST tmt %direction; >

<!ELEMENT to (%extension;)>
<!ATTLIST to %direction; >

<!ELEMENT via (%extension;)>
<!ATTLIST via %direction; >

<!--========================= attribute list ==========================-->
<!ELEMENT tense EMPTY>
<!ATTLIST tense
    attribute (past | present | future) #REQUIRED>

<!ELEMENT aspect EMPTY>
<!ATTLIST aspect
    attribute (begin-soon | begin-just | progress | end-soon | end-just | complete |
            state | repeat) #REQUIRED>

<!ELEMENT reference EMPTY>
<!ATTLIST reference
    attribute (generic | def | indef | not | order) #REQUIRED>

<!ELEMENT focus EMPTY>
<!ATTLIST focus
    attribute (emphasis | entry | qfocus | theme | title | topic) #REQUIRED>

<!ELEMENT attitudes EMPTY>
<!ATTLIST attitudes
    attribute (affirmative | confirmation | exclamation | imperative | interrogative |
            invitation | politeness | respect | vocative) #REQUIRED>

<!ELEMENT viewpoint EMPTY>
<!ATTLIST viewpoint
    attribute (ability | ability-past | apodosis-real | apodosis-unreal | apodosis-cond |
            conclusion | custom | expectation | grant | grant-not | insistence | intention |
            inevitability | may | obligation | obligation-not | possibility | probability | should |
            unexpected-presumption | unexpected-consequence | will) #REQUIRED>

<!ELEMENT convention EMPTY>
<!ATTLIST convention
    attribute (angle_bracket | double_parenthesis | double_quotation | parenthesis | pl |
            single_quotation | square_bracket) #REQUIRED>
```

BIBLIOGRAFIA

(ARPA, 1993) Knowledge Sharing Initiative ARPA. *The Knowledge sharing effort approach*. UMBC agent Web, 1993. Disponível on-line: http://www.cs.umbc.edu/agents

(Bray et. al., 2000) Bray, T.; Paoli, J.; Sperberg-McQueen, C. M. *Extensible Markup Language (XML) 1.0*. (Second Edition) Outubro 2000. Disponível on-line: http://www.w3.org/TR/REC-xml

(Brown, 1989) Brown, H. *Standard for Structured Documents*. The Computer Journal, v.32, n.6, p.505-514, 1989.

(Cohen & Levesque, 1990b) Cohen, P. R., and Levesque, H. J. *Performatives in a rationally based speech act theory*. In Proceedings of the 28 th Annual Meeting of the Association for Computational Linguistics, Pittsburgh, Pennsylvania. 1990b.

(Cohen & Levesque, 1995) Cohen, Philip R.; Levesque Hector. *Communicative Actions for Artificial Agents*, p.419-436. AAAI press/The MIT press, Junho 1995. J. M.

(Connolly, 2000) Connolly, D. *Extensible Markup Language (XML)*. Fevereiro 2000. Disponível on-line: http://www.w3.org/XML/

(Courtois & Silberztein, 1990) Courtois, B., and Silberztein, M. (Eds.). *Dictionnaires électroniques du français*. Langue française, *87*, 1-127. 1990

(David, 1999) David Reilly, *Agent Communication*., Fevereiro 1999. Disponível on-line: http://www.webdevelopersjournal.com/articles/agent_commu ication.html

(Davis & Smith, 1983) Randal Davis and Smith. *Negotiation as a metaphor for distributed problem solving*. Artificial Intelligence, 20:63-109, 1983

(Dignum, 2000) Dignum, Frank; Greaves, Mark. (ed.). *Issues in agent communication.* – (Lecture notes in computer science; Vol 1916: Lecture notes in artificial intelligence) Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan; Paris; Singapore; Tokyo: Springer, 2000.

(Embury & Gray, 1995) Embury, S.; Gray, P. compiling a declarative high-level language for semantic integrity constraints. Technical Report

AUCS/TR9506, University of Aberden, 1995.

(Farquhar et. al., 1995) Farquhar, A; Fikes, R.; Pratt, W.; rice, J. *Collaborative ontology construction for information integration*. Technical report KSL-95-63, Stanford University Knowledge Systems Laboratory, 1995.

(Finin et al., 1993) Finin, Tim; Yannis Labrou; Mayfield, James. *KQML as an Agent Communication Language*, p.291-315. AAAI press/The MIT press, 1993.

(FIPA, 1999) Foundation for Intelligent Physical Agents FIPA. *Fipa spec 2 agent communication language*. Technical Report Draft, version 0.1, Foundation for Intelligent Physical Agents, 1999, Disponível on-line: http://www.fipa.org/spec/fipa99.html

(Franklin & Graesser, 1996) Franklin, S.; Graesser, A. *Is it an agent or Just a Program?: A Taxonomy for Autonomous Agents*. In Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, 1996. Disponível on-line: http://www.msci.memphis.edu/~franklin/AgentProg.html.

(Fraser et. al., 1995) Fraser. J., A. Tate, and M.Uschold. The enterprise toolset – an open enterprise architecture. In the Impact of Ontologies on Reuse, Interoperability and Distributed Processing, pages 42-50. Unicom Seminars, London, 1995. Further information about the Enterprise Project and Ontology is available on the World Wide Web from : http://www.aiai.ed.ac.uk/~enterprise/enterprise.

(Fuchs & Wheadon, 1995) Fuchs,J.; Wheadon, J. *Prospective applications of ontologies for future space missions*. In The Impact of Ontologies on Reuse, Interoperability and Distributed Processing, p83-96. Unicom Seminars, London, 1995.

(Gómez-Pérez et. al., 1995) Gómez-Pérez, a.; Juristo, N. Pazos, J. *Evaluation and assessment of knowledge sharing technology*. In N.J. Mars, editor, Towards Very Large Knowledge Bases – Knowledge Building and Knowledge sharing 1995, p289-296. IOS Press, Amsterdam, 1995.

(Gómez-Pérez, 1995) Gómez-Pérez, A. *Some ideas and examples to evaluate ontologies*. In Proceedings of the Eleventh conference on Artificial Intelligence Applications. IEEE computer society Press, 1995.

(Gómez-Pérez, 1996) Gómez-Pérez, A. *Guidelines to verify completeness and consistency in ontologies*. 1996. to appear at the Third World congress on Expert Systems.

(Gouveia et al., 1998) Gouveia, D; Neto, A. B.; Silva, M. J. *ACE:Um Agente de compras na Internet*. XII Simpósio Brasileiro de Engenharia de Software, Anais, p.281-296, 1998.

(Grosof & Labrou, 1999) Grosof, Benjamin N.; Labrou, Yannis. *An Approach to using XML and a Rule-based Content Language with an agent communication*

Language. IBM Research Report. RC 21491 (96965), 28 May 1999, Disponível on-line: http://www.research.ibm.com

(Gruber, 1993) Gruber, T. *A translation approach to portable ontology specifications*. Knowledge Acquisition, 5(2): 199-220, 1993.

(Gruber, 1995) Gruber, T. *Towards principles for the design of ontologies used for knowledge sharing*. International Journal of Human-Computer Studies, 43(5/6):907-928, 1995

(Gruninger & Fox, 1995) Gruninger M.; Fox, M. S. *Methodology for the design and evaluation of ontologies*. In Workshop on Basic Ontological Issues in Knowledge Sharing. International Joint Conference on Artificial Intelligence, 1995.

(Guarino & Giaretta, 1995) Guarino, P. Giaretta, P. *Ontologies and knowledge bases – towards a terminalogical clarification*. In N. J. Mars, editor, Towards Very Large Knowledge Bases – Knowledge Building and Knowledge Sharing 1995, p.25-35. IOS Press, Amsterdam. 1995.

(Idris, 1999) Idris, N. *Should I use SAX or DOM?* Maio 1999. http://developerlife.com/saxvsdom/default.htm

(Jennings & Wooldridge, 1995) Jennings, N. R.; Wooldrige, M. *Agent Theories, Architectures, and Languages: a Survey*. *Intelligent Agents*, p.55-67, 1995.

(Johnson, 1999) Johnson, M. *XML for the Absolute Beginner*. Disponível on-line: http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml.html. Abril 1999.

(Jones et. al, 1995) Jones, M.; Wheadon, J.; Whitgift, D.; Niezatte, M.; Timmermans, R.; Rodriguez, I.; Romero, R. An agent based approach to spacecraft mission operations. In N.J. Mars, editor, Towards Very Large Knowledges Bases – Knowledge Building and Knowledge sharing 1995, p259-269. IOS Press, Amsterdam, 1995.

(Ketchpel & Genesereth, 1994) Ketchpel, S.; Genesereth, M. *Software Agents*. Communications of ACM, Vol37, Nro 7, p.48-53, julho de 1994.

(Labrou & Finin, 1997a) Labrou, Y.; Finin, T. *A proposal for a new kqml specification*. Technical Report TR-CS-97-03, University of Maryland, Baltimore Country (UMBC), Disponível on-line: http://www.cs.umbc.edu/kqml/papers, 1997.

(Labrou et. al., 1999) Labrou; Yannis; Finin, Tim; Yun Peng. *Agent Communication Languages: The Current Landscape*. IEEE Intelligents Systems, p45-52, March/April 1999.

(Lange & Mitsuru, 1998) Lange, D.; Mitsuru, O. *Programming and Deploying Java Mobile Agents with Aglet*. Addison Wesley, August 1998. ISBN-0-201-

32582-9.

(Lee et. al., 1995) Lee J., Yost G., PIF Working Group. The pif process interchange format and framework. Technical Report 180, MIT Center for Coordination Science, 1995.

(Lenat, 1995) Lenat, D. B. *CYC: A large-scale investment in knowledge infrastructure*. Communications of the ACM, *38*(11), 33-48. 1995

(Mace et al., 1998) Mace, S.; Flohr, V.; Dobson, R.; Graham, T. *Weaving a Better Web*.Byte,. p58-68, Março 23, 1998.

(Maes, 1994) Maes, P. *Agents that Reduce Work and Information Overload*. Communication of the ACM, p.31-40, julho de 1994.

(Mamadou et al., 2000) Mamadou, T. K.; Shimazu, A.; Tatsuo, N. *The State of the Art in Agent Communication Languages*. Japan Advanced Institute of Science and Technology., Japan, 1999.

(McGrath, 1999) McGrath, S. *XML Aplicações Práticas - Como Desenvolver Aplicações de Comércio Eletrônico*. Editora Campus, 1999.

(Megginson, 1998) Megginson, D. SAX 1.0: *The Simple API for XML*. Maio 1998. http://www.megginson.com/SAX/SAX1/index.html

(Miller, 1995b) Miller, G. A. *WordNet: A lexical database for English*. Communications of the ACM, *38*(11), 39-48. 1995.

(Moreira & Walczowski, 1997) Moreira, D. A.; Walczowski, L. T. *Using Software Agents to Generate VLSI Layouts*. IEEE Expert Intelligent Systems, p.26-32. Nov/Dez. de 1997.

(Mueller, 1998) Mueller, Erik T. *Natural Language processing with ThughtTreasure*. New York: Signiform. Disponível on-line: http://www.signiform.com/tt/book/

(Nunes et. al., 1999) Nunes, M.G.V.; Dias da Silva, B.C.; Pino, L.H.M.; Novais de Oliveira Jr, O; Martins, R.T.; Montilla, Gisele. Introdução ao Processamento das Línguas Naturais, N38, Notas didácticas do ICMC, São Carlos, Junho 1999.

(Nunes, et.al., 1997) Nunes, M.G.V.; Sossolote, C.R.C.; Zavaglia, C.; rino, L.H.M. As Manifestações Morfosintácticas da Linguagem UNL no Portugês do Brasil. Nilc-97-TR-2, Série de Relatórios do Núcleo Interinstitucional de Linguistica Computacional. 28 de Outubro de 1997.

(Parks, 1997) Parks, D. *Agent-oriented programming: A practical evaluation*. Technical Report 94720, University of California Berkley, Disponível on-line: http://http.cs.berkley.edu/davidp/cs263/, 1997.

(Pimentel et. al, 1999) Pimentel, M.G.C., Teixeira, C.A.C., Pinto, C. C. – *Hiperdocumentos Estruturados na WWW: Teoria e Prática*. IN: XVIII Jornada de

Atualização em Informática. JAI99 –SBC. Julho de 1999. p. 367-424.

(Sadek et al., 1997) Sadek, D.; Bretier, P.; Panaget, F. *France telecom artimis technology, arcol agent communication language and mcp, cap and sap agent's cooperativeness protocols*. Technical report, FIPA, Disponível on-line: http://drogo.cselt.stet.it/fipa/cfp1/propos97.htm,1997

(Schreiber et. al, 1995) Schreiber, G; Wielinga, B.; Jansweijer, W. *The kactus view on the 'o'word*. In workshop on Basic Ontological Issues in Knowledge sharing. International Joint Conference on Artificial Intelligence, 1995

(Shoham, 1993) Shoham, Y. *Agent-oriented programming*. Artificial Intelligence, 60(1), p.51-92, 1993.

(Sierra et. al., 2000) Sierra, Carles; Wooldridge, Michael; Sadeh, Norman. *Agent Research and Development in Europe*. IEEE Internet computing, p81-83, September/October 2000.

(Singh, 1998) Singh, M. P. *Agent communication languages: Rethinking the principles*. IEEE Computer, 31(12), p.40-47, December 1998.

(Singh, 1999) Singh, M. P. *A Social Semantics for Agent Communication Languages*. ACM Press, August 1999. Stockholm, Sweden.

(Skuce, 1995) Skuce, D. *Conventions for reaching agreement on shared ontologies*. In Proceedings of the 9th Knowledge Acquisition for Knowledge Based Systems Workshop, 1995

(SOAP, 2000) *Technical Report : SOAP - Simple Object Access Protocol 1.1,* World Wide Web Consortium (W3C) disponível on-line: http://www.w3.org/TR/SOAP/

(Sun, 2001) *Specifications Java API for XML Processing Specification*, final Version 1.1. disponível on-line: http://java.sun.com/xml/download.html

(Torrance, 1998) Torrance, M. Agent-0: *A language for agent oriented programming*, 1998 Disponível on-line: http://www.scs.ryerson.ca/drimsha/courses/cps720/agent0.html

(Trindade, 1997) Trindade, C. C. *Minimal Hyperlink Documents: Especificação e Apresentação de Estruturas Clásicas de Hipertextos*. São Carlos, Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

(Uchida, 1999) Uchida, H.; Zhu Meiying; Della Senta, T. (1999). Universal Networking Language: a gift for amillenium. Tokyo: United Nations University. Disponível on-line: http://www.unl.ias.unu.edu/publications/index.html

(Ushold & Gruninger, 1996) Uschold, M. Gruninger, M. *Ontologies: Principles, Methods and Applications*. Knowledge Engineering Review, v.11, Number 2, June 1996. University of Edinburgh, 1996.

(White, 1994) White, E. J. *Telescript technology: The foundation for the electronic marketplace*. Technical report, General Magic, Inc., Disponível on-line: http://www.generalmagic.com/technology, 1994.

(Wielinga et. al, 1994) Wielinga, R; Schreiber, G; Jansweijer, W; Anjewierden, A; van Hamelen F. Framework and formalism for expressing ontologies. Technical report, University of Amsterdam, 1994. Esprit Project 8145 Deliverable DO1b1, Disponível on-line: http://www.swi.psy.uva.nl/projects/Kactus/Reports.html

(XSL, 2000) Learning XSL. [online]. Disponível na Internet em: www.w3.org/Style/XSL. 2000.

(Yokoi, 1995) Yokoi, T. *The EDR electronic dictionary*. Communications of the ACM, *38*(11), 42-48. 1995.