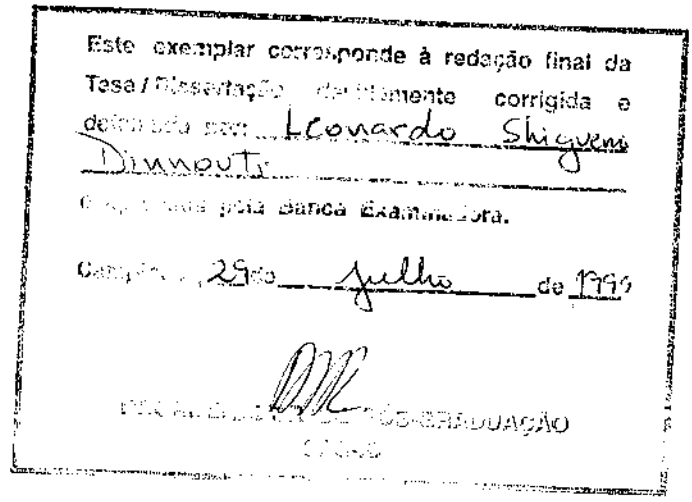


Utilização de um Sistema Multiagentes para Resolução de Referências em Língua Natural

Leonardo Shigueni Dinnouti

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Utilização de um Sistema Multiagentes para Resolução de Referências em Língua Natural

Leonardo Shiguemi Dinnouti[1]

abril de 1999

Banca Examinadora:

- Ariadne Maria B. Rizzoni Carvalho (Orientadora)
- Heloísa Vieira da Rocha
  Instituto de Computação - Unicamp
- Jorge Kinoshita
  Escola Politécnica da Universidade de São Paulo - USP
- Maria Cecília C. Baranauskas (Suplente)

[1]**Este trabalho foi financiado pelo CNPq.**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Dinnouti, Leonardo Shiguemi

D617u Utilização de um sistema multiagentes para resolução de referências em língua natural / Leonardo Shiguemi Dinnouti -- Campinas, [S.P. :s.n.], 1999.

Orientador : Ariadne Maria B. Rizzoni Carvalho

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Inteligência artificial. 2. Processamento da linguagem natural. I. Carvalho, Ariadne Maria Brito Rizzoni. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Utilização de um Sistema Multiagentes para Resolução de Referências em Língua Natural

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Leonardo Shiguemi Dinnouti e aprovada pela Banca Examinadora.

Campinas, 05 de abril de 1999.

Ariadne M. B. R. Carvalho
(Orientadora)

Jacques Wainer (Co-orientador)

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 05 de abril de 1999, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Jorge Kinoshita
POLI - USP

Profa. Dra. Heloísa Vieira da Rocha
IC- UNICAMP

Profa. Dra. Ariadne Maria Brito Rizzoni de Carvalho
IC - UNICAMP

Resumo

Esta tese contribui para a Área de Inteligência Artificial trazendo uma abordagem alternativa para o processamento de Língua Natural utilizando Sistemas Multiagentes.

O fenômeno lingüístico a ser analisado é a referência, e em particular a referência definida. O fenômeno é amplamente conhecido como anáfora definida e vem sendo trabalhado por diversos pesquisadores na área de lingüística computacional.

Nesta tese o problema é divido e tratado de forma distribuída, através de um sistema multiagentes projetado especificamente para este caso. Um protocolo de comunicação é proposto e alguns testes são feitos com agentes em funcionamento. A implementação do sistema utiliza bibliotecas de livre distribuição, como o SWI-Prolog e o PVM (*Parallel Virtual Machine*), tornando o sistema passível de ser distribuído livremente.

Este trabalho contribui com mais um tipo de abordagem para o Processamento de Língua Natural e abre perspectivas para novos tipos de tratamentos utilizando Sistemas Multiagentes.

Abstract

This thesis contributes to the field of Artificial Intelligence by providing an alternative approach to Natural Language Processing using a Multi-Agent System.

The linguistic phenomenon which is analyzed is reference, in particular, definite reference. This phenomena, which is also known as definite anaphor, has been studied by several researchers in the Computational Linguistics field.

In this thesis, the problem is divided and treated in a distributed form by a Multi-Agent system that was designed specifically to solve this problem. A communication protocol is developed and some tests are made while the agents are working. The implementation of the system uses free distribution libraries such as SWI-Prolog and PVM (Parallel Virtual Machine) and, therefore, it can be freely distributed.

This thesis contributes with a different approach to Natural Language Processing that opens up new perspectives through the use of Multi-Agent systems.

Agradecimentos

A Deus por todas as coisas que recebi sem merecer e por tudo o que Ele me ensinou durante o desenvolvimento deste trabalho.

Agradeço ainda pelo apoio psicológico e paciência em todos os momentos de dificuldades à minha esposa Gisane; ao meu irmão Luiz; aos meus colegas de república Fabiano Silva, Letícia Mara Peres e Achilles Delari Júnior.

Pela minha existência e pelo apoio durante toda a minha vida, agradeço aos meu pais, Genquiti e Lucinda Dinnouti.

Aos meus orientadores, Ariadne e Jacques, pela paciência e apoio durante todo o processo de desenvolvimento deste trabalho.

Pelas horas de trabalho que não trabalhei por causa da tese, agradeço aos meus colegas Juan Carlos Sotuyo, Habib Hanna El Khouri, João Alberto Fabro, Marcos Antônio Massao Hashisuca, Cláudio R. Marqueto Maurício, Luciano S. Cardoso, Déborah, Rosineide e Angelita.

Por terem ouvido minhas dúvidas e contribuído com idéias, agradeço ainda ao Reginaldo Hughes Carvalho, Michel Gagnon, Alexandre Direne e José Borges Neto.

À Igreja Batista Memorial do Centenário, ao Pr. Donaldo Carvalho da Paixão, à irmã Dilzete Carvalho da Paixão, à Igreja Batista Itaipu, ao Pr. Djoni Schallenberger e à irmã Áurea Schallenberger pelas suas orações.

Conteúdo

Resumo vi

Abstract vii

Agradecimentos viii

1 Introdução 1

2 Referência 4

2.1 Elementos envolvidos no uso de referências 5

2.2 Variações sobre o mesmo tema 7

2.2.1 Jiři Krámský 7

2.2.2 John R. Searle 9

2.2.3 Graeme Hirst 10

2.2.4 James Allen 11

2.2.5 Candace Sidner 12

2.3 Referência Definida 13

2.3.1 Uma Breve Discussão sobre o Artigo Definido 13

2.3.2 Classificação 14

2.4 Referência Não Definida 19

2.5 Conclusão 20

3 A representação lógica utilizada no sistema **22**

3.1 As fases de codificação 22

3.2 A representação escolhida 24

3.3 Intensionalidade e Extensionalidade das Sentenças 26

3.3.1 Sentenças "intensionais" 27

3.3.2 Sentenças Extensionais 28

3.4 A Gramática Utilizada 29

3.5 O Analisador Léxico e Sintático 29

3.6 Conclusão 30

4 Sistemas Multiagentes **32**

4.1 Um Agente 34

4.1.1 Memória 36

4.1.2 Percepção 38

4.1.3 Capacidade de Decisão 38

4.1.4 Capacidade de Negociação 38

4.1.5 Comunicação 39

4.2 Características de um sistema multiagentes 41

4.3 Exemplo de um agente 42

4.4 Conclusão 42

5 O Uso de um Sistema Multiagentes para o Problema de Referência Definida **43**

5.1 Granularidade do sistema 44

5.2 O funcionamento geral do sistema 44

5.3 A comunicação entre os agentes 47

5.4 Funcionamento de um agente 48

5.5 A arquitetura do Sistema Multiagentes 49

5.6 As heurísticas de cada agente 51

5.6.1 Heurística do agente 1 51
5.6.2 Heurística do agente 2 52
5.6.3 Heurística do agente 3 53
5.6.4 Heurística do agente 4 53
5.6.5 Heurística do agente 5 55
5.6.6 Heurística do agente 6 56
5.7 Conclusão 57

6 Conclusão **58**

A Descrição do Sistema Multiagentes **61**

Bibliografia **127**

Lista de Tabelas

2.1 Formas simples do artigo 14

2.2 Formas combinadas do artigo definido 14

2.3 Exemplos de regras para reduzir a ambigüidade de *a* 15

2.4 Exemplo de quatro tipos de referências definidas. 15

2.5 Quadro geral de classificação 21

5.1 Tipos de Mensagens 49

5.2 Exemplos de mensagens entre os agentes. 49

Lista de Figuras

2.1 Alguns elementos de uma referência. 6

3.1 Fases de processamento. 23
3.2 Parser de uma sentença. 25
3.3 Um exemplo de análise e tradução. 26
3.4 Gramática utilizada. 31

4.1 Agentes, um mundo e um problema. 35
4.2 A representação interna e a realidade do mundo. 36
4.3 Composição básica de um agente. 36
4.4 Um exemplo de dois agentes simples. 42

5.1 Funcionamento geral do sistema. 46
5.2 Visualização gráfica de uma execução do sistema. 48
5.3 Um agente dentro do sistema. 50
5.4 Arquitetura do sistema. 50

Capítulo 1

Introdução

Os estudos e pesquisas relacionados à área da computação conhecida como Inteligência Artificial (IA) buscam colocar no computador um pouco da "inteligência" humana. Considera-se que o objetivo principal da IA é a criação de modelos cognitivos que representem os processos mentais dos seres humanos e a implementação de sistemas computacionais baseados nestes modelos. Os modelos cognitivos, que visam o entendimento e a modelagem da inteligência humana, procuram representar aspectos diversos da inteligência sendo um destes a capacidade humana de comunicar-se com os outros seres da mesma espécie através da linguagem.

A Lingüística Computacional (LC), também conhecida como Processamento de Língua Natural (PLN), é o ramo de estudos em Inteligência Artificial que tem como objetivo específico o estudo da língua humana sob a perspectiva computacional. O seu objeto de estudo, a língua natural, é definida como a língua humana em forma escrita ou oral utilizada para comunicação, diferindo-se, portanto, das linguagens artificiais criadas pelos homens, como por exemplo, as linguagens de programação.

Por mais de trinta anos, estudiosos e pesquisadores têm procurado desenvolver teorias, métodos e técnicas com os quais os computadores possam ser programados para entender e gerar enunciados e textos. As vantagens preconizadas pelos estudos em LC são exemplificadas pelas numerosas aplicações que estão sendo utilizadas nos sistemas atuais. Interfaces em língua natural possibilitam ao usuário comunicar-se com o computador em português, inglês ou outra língua humana; algumas aplicações de tais interfaces são as consultas a banco de dados, compreensão de textos e os conhecidos sistemas especialistas. Outros avanços no reconhecimento da linguagem falada, como produção de software com corretores ortográficos e sintáticos e tradução automática também demonstram as possibilidades futuras da aplicação de estudos nesta área.

A língua natural, porém, oferece um desafio constante aos pesquisadores pela quantidade e a complexidade de fatores intra ou extra-lingüísticos que influenciam a significação de um enunciado. Os modelos computacionais têm fornecido representações mais precisas nas áreas da fonologia, morfologia e sintática; porém, as representações semânticas e a contextualização de enunciados na pragmática ainda constituem um terreno movediço. Todos estes fatores em conjunto ajudam a elucidar e compreender melhor os fenômenos lingüísticos que, por sua vez, são fundamentais para a compreensão e a modelagem da linguagem.

Dirigimos este trabalho para a compreensão da Língua Natural em sua modalidade escrita, com enfoque na parte sintática e semântica. O fenômeno lingüístico que será objeto de estudo neste trabalho é a questão da referência em sintagmas nominais introduzidos por artigo definido. O fenômeno a que nos referimos faz parte de um problema crucial para a modelagem da linguagem em termos computacionais que é: determinar como pensamentos e sentenças estão relacionados e fazem *referências* a entidades concretas e abstratas.

O tema "referência", ou anáfora, teve início como um tema de pesquisa em computação na década de 70, quando alguns pesquisadores publicaram diversos artigos sobre este tema. Podemos citar Wilks (1973) [43], Hobbs (1978) [19] e Sidner (1978) [38] como exemplo de trabalhos iniciais. A década de 80 não trouxe muitas novidades, mas na década de 90 foram apresentados inúmeros trabalhos relacionados à referência. Podemos citar o *Discourse Anaphora and Anaphora Resolution Colloquium* (DAARC), em 1996, e outros encontros onde diversos artigos sobre referência foram apresentados, como o *Anual Meeting of the Association for Computational Linguistics* (ACL), em 1997/98 e o *European Chapter of the Association for Computational Linguistics* (EACL), em 1998.

Um das dificuldades encontradas para tratar o problema da referência é a necessidade de se utilizar conhecimento semântico durante a interpretação das referências. É necessária uma consulta constante ao contexto, seja ele construído a partir do próprio texto sendo analisado ou adquirido previamente, fazendo parte do senso comum (ou conhecimento de mundo). No capítulo 3, comentaremos como o contexto é tratado neste trabalho.

Um aspecto importante da pesquisa que fizemos para abordar o problema de referência definida (também conhecida como anáfora definida) foi o uso de um Sistema Multiagentes. Este tipo de abordagem baseia-se em agentes inteligentes e autônomos que interagem entre si promovendo, através da distribuição de tarefas, a possibilidade de tratamento de problemas cuja complexidade pode aumentar gradativamente.

Sistemas Multiagentes (SMAs) é uma área de pesquisa emergente, com diversas aplicações

sendo desenvolvidas e testadas. A sua utilização prática, no entanto, é ainda limitada. Os resultados das pesquisas estão sendo divulgados e os maiores projetos se encontram em fase experimental. Quanto à sua utilização no Processamento de Língua Natural, existem poucos trabalhos relacionados, onde podemos citar Paiva [10], Silva [20] e Stefanini [40]. Paiva [10] utiliza este paradigma para tratar de problemas do tipo elipse; Silva [20] trata de ambigüidade léxica utilizando um paradigma multiagentes; Stefanini [40] propõe uma arquitetura para processamento de língua natural em várias fases (atribuídas a agentes): fase morfológica, léxica, sintática e semântica. O trabalho da Stefanini foi um dos pioneiros na utilização de Sistemas Multiagentes para o Processamento de Língua Natural.

Existem outros trabalhos na mesma linha, mas as técnicas empregadas são muito diversificadas para fazer uma comparação entre elas. Acreditamos que é prematuro dizer qual o grau de evolução do uso de sistemas multiagentes para o processamento de língua natural e o quanto esta abordagem será importante para a área.

Os conceitos de Sistemas Multiagentes utilizados neste trabalho são semelhantes aos introduzidos em Stefanini [40], Berthet [3], Wooldridge [44] e Coelho [7], dentre outros.

Para a análise sintática seguiremos a gramática gerativa introduzida por Chomsky [6] e a gramática do português fornecida por Luft [23]. Do ponto de vista semântico utilizaremos a definição de Searle [36] sobre referência como um ato da fala.

Como recursos computacionais utilizaremos linguagem C++ [41], Prolog (SWI-Prolog versão 3.1.2), Lex (analisador léxico), Yacc (analisador sintático) e PVM (*Parallel Virtual Machine*, uma biblioteca para implementação de algoritmos paralelos e distribuídos).

O restante desta dissertação está assim organizada: o capítulo 2 apresenta os conceitos e nomenclatura utilizados para tratar o fenômeno de referência definida assim como uma proposta de classificação baseada em um levantamento de trabalhos relacionados ao fenômeno; o capítulo 3 apresenta a análise sintática e a codificação das sentenças do português que serão utilizadas como exemplos; o capítulo 4 faz um levantamento sobre Sistemas Multiagentes direcionando a sua aplicação para o tratamento de Língua Natural, citando as características desejáveis para o sistema a ser desenvolvido; o capítulo 5 apresenta o sistema proposto nesta dissertação explicando como o sistema multiagentes é utilizado para resolver as referências de um texto; finalmente, o capítulo 6 apresenta a conclusão e o apêndice a documentação do sistema multiagentes desenvolvido nesta dissertação.

Capítulo 2

Referência

O problema da referência[1] é uma das grandes dificuldades encontradas na análise da Língua Natural. Mesmo entre humanos, é uma questão que pode suscitar dúvidas durante uma comunicação escrita ou falada.

Uma das dificuldades encontradas é a necessidade de utilizar conhecimento semântico durante a interpretação das referências. Sempre é necessário consultar um contexto anterior para fazer as ligações necessárias e, mesmo quando não há ligações anteriores, é necessário que as referências sejam anotadas para uma resolução posterior.

Como em ciência da computação ainda não há consenso quanto à representação de conhecimento semânticos e quanto à construção de contexto a partir da língua natural, o problema de referência continua sendo um tema de pesquisa com difícil resolução computacional.

O uso de referências na língua é comum quando estamos conversando com alguém ou escrevendo um texto. Por exemplo, o uso do nome de uma pessoa é uma forma de se fazer uma referência. Assim, em

(1) Ana viu Pedro.

o nome "Ana" é utilizado para identificar uma determinada pessoa. Existe, neste caso, uma referência simples e direta. Podemos fazer também referências entre objetos ou pessoas mencionados no texto, como na sentença abaixo:

(2) Ela não o reconheceu.

[1]Conhecido também como anáfora (*anaphora* em inglês)

Neste exemplo temos referências pronominais à Ana, através do "Ela" e a Pedro, através do pronome oblíquo "o".

Já no exemplo

> *(3) Encontrei uma caneta azul e um lápis vermelho.* **A caneta** *não funcionava.*

a referência **"a caneta"** é também um caso simples onde se menciona um dos objetos apresentados na primeira frase. Este é um uso freqüente da referência. Existem inúmeros outros casos e outras formas de se fazer referências.

Neste trabalho nos restringiremos aos casos de referências aos sintagmas nominais introduzidos por artigo *(o, a, os, as, um, uns, uma)*. Neste capítulo iremos expor os casos que pretendemos tratar baseados em alguns trabalhos relacionados com este problema. A notação e as definições não são comuns a todos os autores e no decorrer desta seção redefiniremos alguns termos e conceitos utilizados para descrever o problema da referência. Apresentaremos também uma classificação para os diferentes tipos de referência e um breve comentário sobre os casos de referência indefinida.

2.1 Elementos envolvidos no uso de referências

O fenômeno lingüístico que chamamos de referência é um mecanismo utilizado para relacionar objetos ou para apresentar novos objetos em um discurso. Relacionar objetos consiste em identificar quando um mesmo objeto (ou entidade) é mencionado novamente para que sejamos capazes de atribuir a referência ao objeto correto. Quando um objeto não está sendo relacionado com outro ele normalmente está sendo introduzido no texto pela primeira vez, comumente através de um artigo indefinido, como veremos na seção 2.4.

Uma referência bem sucedida é aquela que permite que, no processo de compreensão do discurso, seja possível identificar os objetos envolvidos (ou co-relacionados) de maneira a não criar dúvidas. O exemplo (4) ilustra uma falha ao se referir a um elemento anterior.

> *(4) Em uma feira Ana ganhou um preá e Pedro ganhou uma cobaia.* **O roedor** *não parava de comer.*

Este exemplo não tem sucesso em sua referência porque tanto a cobaia quanto o preá são animais considerados roedores; portanto, o substantivo "roedor" não consegue identificar de maneira não ambígüa qual o referente da sentença anterior: o preá e a

cobaia. O ser humano pode assumir uma leitura preferencial, considerando que apenas o preá é "mais" roedor do que a cobaia, ou vice-versa.

A partir das referências é possível relacionar objetos dentro do discurso ou voltar a mencioná-los de forma que, sempre que se estiver falando de um mesmo objeto, seja possível identificá-lo de forma não ambígua. Uma referência bem sucedida é aquela que permite que no processo de compreensão do discurso seja possível identificar os objetos envolvidos (ou co-relacionados) de forma a não criar casos de ambigüidade.

A seguir definiremos os seguinte termos que serão utilizados no decorrer do texto: **sentença**, **discurso**, **referência**, **referente** e **antecedente**.

Uma **sentença** pode ser vista como uma seqüência de palavras escritas (ou faladas). A sentença deve possuir, dentro do contexto em que se encontra, uma significação própria e será considerada como uma unidade mínima de comunicação, com o mesmo sentido de "frase". Uma sentença, por sua vez, pode ser composta por mais de uma oração e organizada em períodos (orações principais e subordinadas).

Chamaremos de **discurso** uma seqüência de sentenças com um propósito informativo, ou seja, espera-se que o discurso seja coerente e que possua uma seqüência lógica. Não estamos considerando seqüências de frases soltas que não façam sentido.

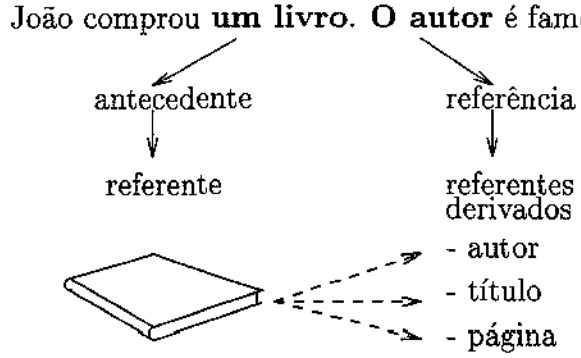


Figura 2.1: Alguns elementos de uma referência.

Para haver **referência** precisamos identificar a **referência** e o **referente**. O **antecedente** é o sintagma nominal que concorda com a referência feita, ou seja, que satisfaz todos as propriedades mencionadas na referência. Um antecedente é considerado um candidato a referente ou conjunto de referentes. O **referente**, por sua vez, deve ser um antecedente que concorda com a referência. A figura 2.1 ilustra um exemplo em que a referência é o sintagma "o autor", o antecedente é o sintagma "um livro" e o referente é o autor do objeto livro sendo referido no contexto, pois este livro não é um livro qualquer; existem fatores que o tornam único dentro do discurso, apontado pelo fato de que

foi o João quem o comprou. Existe um conhecimento adicional que não está claro nas sentenças, isto é, a relação de autor com livro. Sabe-se, pelo conhecimento geral, que um livro possui pelo menos um autor.

O exemplo (5) é uma continuação do exemplo da figura 2.1 onde se utiliza um artigo definido para indicar que se trata do mesmo objeto introduzido anteriormente. Neste caso a referência é "(d)o livro", o antecedente é "um livro" e o referente é o próprio objeto livro indicado na sentença "Pedro comprou um livro".

(5) João comprou um livro. O autor é famoso. João gostou muito **do livro**.

Utilizaremos as definições desta seção no restante deste trabalho.

2.2 Variações sobre o mesmo tema

Como já foi dito, existem diversos conceitos e classificações diferentes para o problema de referência. As subseções a seguir buscam trazer as idéias de alguns autores que mencionam o problema de referência em seus trabalhos, procurando destacar os pontos relacionados à referência definida. Nem todos os trabalhos concordam entre si mas cada um contribui com uma maneira diferente de ver o mesmo problema.

2.2.1 Jiři Krámský

Este autor [32] faz um estudo sobre o uso do artigo definido em diferentes linguagens e chega a fazer uma classificação utilizando o conceito de familiaridade. Este conceito está relacionado com o fato de uma referência ser indefinida (não familiar), parcialmente definida (aproximadamente familiar) e definida (familiar).

Segundo Krámský [32]:

> *... o artigo definido é posto antes de um substantivo para mostrar que a idéia expressa pelo substantivo já foi dita anteriormente e para se referir novamente àquela expressão.*

> *O artigo definido indica que a pessoa ou objeto do qual se está falando é de alguma maneira conhecida ou familiar ... e é exatamente esta familiaridade ou sentido geral da palavra que o artigo indica.*

Completa Familiaridade: existe uma necessidade em se distinguir uma referência como definida (familiar) ou não definida (não familiar). Normalmente, uma referência é feita a um objeto ou a uma classe a qual pertence o objeto. Na sentença 6.1, do exemplo abaixo, "um livro" se refere a uma classe de livros, ou seja, Pedro pode ter comprado qualquer livro. Porém, na sentença 6.2, temos a referência a um livro em específico, ou seja, exatamente aquele que Pedro comprou.

(6.1) Pedro comprou um livro.
(6.2) O livro que Pedro comprou estava em promoção.

Os casos de familiaridade completa (ou referência direta) se estendem para nomes próprios e palavras com sentido bem determinado, como por exemplo *Deus, Pedro, Ana, Sol, Lua.*

Familiaridade Aproximada: o uso do artigo definido, porém, não é suficiente para se fazer a distinção entre uma referência definida e uma não definida. Por exemplo, na sentença

(7) O mamute é um mamífero.

a referência está sendo feita a toda uma classe de animais, mesmo usando-se o artigo definido. Pode-se dizer que o artigo é utilizado para restringir ou delimitar a classe dos mamutes como sendo única.

Não-familiaridade: estes casos são aqueles em que não se quer ou não se consegue definir o objeto mencionado. Por exemplo, na sentença

(8) Ana comeu um chocolate.

pode-se introduzir uma nova entidade no contexto para representar "um chocolate". Uma característica que o torna único é o fato de que "Ana (o) comeu". Da próxima vez que esta entidade for mencionada deverá se reportar à mesma.

É difícil seguir a classificação sugerida por Krámský porque no português não há uma regra clara dizendo o que é familiar (definido) ou não-familiar. O estudo de Krámský se baseia na função do artigo definido para determinar a familiaridade de uma entidade. Na nossa língua o uso do artigo definido é muito variado e depende muitas vezes do estilo do texto sendo escrito. Os exemplos abaixo são válidos em português:

(9.1) Leite é melhor do que cerveja.

(9.2) O leite é melhor do que a cerveja.

(9.3) Todo homem busca felicidade.

(9.3) A felicidade é buscada por todos.

As frases (9.1) e (9.2) e as frases (9.3) e (9.4) têm a mesma intenção mas são escritas de forma diferente quanto ao uso do artigo definido. Este tipo de "indefinibilidade" faz com que seja difícil chegar a alguma conclusão quanto ao sintagma sendo analisado ("o leite" ou simplesmente "leite", qual deles é mais definido que o outro?).

2.2.2 John R. Searle

Segundo Searle [36] uma referência é uma abreviação para "expressões definidas utilizadas para se referir a entidades particulares" e seguem um determinado modelo. Ele deixa claro que o conceito de referência não é preciso. Uma referência, a princípio, é composta por:

- nomes próprios;
- sintagmas nominais iniciados por artigo definido ou pronome possessivo e seguidos por algum substantivo; e
- pronomes.

Uma referência segue os seguintes axiomas:

1. Qualquer objeto que seja referido deve existir (princípio da existência).
2. Se uma propriedade é verdadeira para um objeto então a mesma propriedade é verdadeira para qualquer objeto que seja idêntico a ele independentemente da expressão que está sendo utilizada para se referir a ele (princípio da identidade).
3. Se o narrador se refere a um objeto então ele identifica, ou é capaz de identificar no decorrer do discurso, este objeto para o ouvinte de maneira a diferenciá-lo de todos os outros objetos.
4. Uma condição necessária para a execução correta de uma referência definida na declaração de uma expressão é que:
    a. a declaração desta expressão precisa informar ao ouvinte uma descrição verdadeira ou um fato sobre um e somente um objeto;

b. no caso em que a declaração não expressa a informação desejada o narrador deve ser capaz de substituir a expressão que ele usou.

Podemos ainda parafrasear os axiomas (1) e (2) da seguinte forma:

1a'. é necessária a existência de pelo menos um objeto sobre o qual a expressão do narrador se aplique.

1b'. não deve existir mais do que um objeto sobre o qual a expressão do narrador se aplique.

2'. O ouvinte deve possuir conhecimento suficiente para identificar o objeto a partir da declaração do narrador.

Os axiomas (1a') e (1b') afirmam a idéia de existência e unicidade do referente. A princípio, não se pode referir a um objeto que não existe, ou seja, que não possua nenhum tipo de representação no contexto conhecido. A necessidade de existir um único referente é discutível, pois estes axiomas não valem para casos em que se faz referência a conjuntos de objetos. Estes conjuntos podem ser caracterizados de maneira a diferenciar cada elemento deles de outros objetos no contexto, ou seja, se considerarmos que cada elemento é definido podemos dizer que o conjunto é também definido. Por exemplo, a sentença

(10) Pedro comprou algumas revistas. As revistas mais velhas estavam com um preço menor.

trata de conjuntos de objetos de forma muito pouco precisa. A representação dessas sentenças em proposições nem sempre consegue representar aquilo que o narrador quis dizer com a sentença. O conjunto formado pelas "revistas mais velhas" não é bem delimitado.

O axioma (2') exige que o ouvinte seja capaz de interpretar o discurso do narrador. Isso é possível se o ouvinte e o narrador utilizam as mesmas convenções da língua. Searle [36] defende a existência de regras e convenções que governam a língua e que são estas que determinam o significado das sentenças. Supor a existência destas regras facilita o processo de análise da comunicação entre duas pessoas, mesmo que as pessoas envolvidas não estejam conscientes disto.

2.2.3 Graeme Hirst

Segundo Hirst [18] o uso da referência tem como objetivo fazer uma menção abreviada a algum termo utilizado anteriormente, uma função normalmente associada a pronomes.

No exemplo abaixo o pronome *"Ele"* tem a função de identificar *"Pedro"* na segunda sentença.

(11) Pedro ganhou um livro. **Ele** *gostou do autor.*

A relações entre as referências e seus antecedentes (o uso de antecedente aqui se refere ao objeto identificado por ele) podem ser variadas, como relações de *"faz parte de"*, *"é subconjunto de"*, *"é um aspecto de"*, *"é um atributo de"*, *"é idêntico a"*, e outros tipos de relações que podem servir para identificar um antecedente válido para a referência. Exemplo:

(12) O departamento graduou cinco estudantes este ano. **Os doutorandos** *eram todos de IA.*

Este tipo de identificação de referente, ou recuperação de um referente identificado por um antecedente, se limita a considerar o relacionamento entre sintagmas nominais e seus significados dentro de uma mesma sentença (intrasentencial) ou entre sentenças diferentes (intersentencial). As duas classes principais que Hirst apresenta são a *Indentity of Sense Anaphora* (ISA) e *Identity of Reference Anaphora* (IRA). A primeira engloba os casos em que a referência pronominal denota uma entidade similar à identificada pelo antecedente e a segunda quando a entidade identificada pelo antecedente é a mesma. Exemplos:

(13.1) Pedro fez um sanduíche de pepino e **o** *comeu.*

(13.2) O homem que dedica o seu salário para a sua esposa é mais sábio do que o homem que **o** *dedica para sua amante.*

Na sentença (13.1) o pronome oblíquo "**o**" se refere ao mesmo sanduíche feito por Pedro mas na sentença (13.2) o salário do segundo homem é diferente do salário do primeiro, pois considera-se que cada homem possui o seu próprio salário.

Hirst não considera os casos em que a referência não possui antecedentes que possam ser relacionados no texto.

2.2.4 James Allen

Allen [1] faz distinção entre *Referências Anafóricas* e *Não-anafóricas*, que são referências a objetos já mencionados anteriormente ou não mencionados no contexto, respectivamente. As referências podem ainda ser *definidas* ou *indefinidas*, *singulares* ou *plurais*. As referências não-anafóricas ocorrem quando um novo objeto é introduzido no contexto. Por exemplo:

(14) Encontrei uma caneta azul.

A caneta do exemplo acima é um caso de referência indefinida, não-anafórica e singular. Para ser uma referência no plural basta que ela se refira a vários referentes ou a um conjunto de referentes. No exemplo abaixo, ocorre na segunda sentença um caso de referência definida e anafórica. A caneta agora mencionada já foi introduzida na sentença anterior.

(15) Encontrei uma caneta azul e um lápis vermelho. **A caneta** *não funcionava.*

Podemos ter casos de referência não-anafórica, porém definidas, como é o caso de nomes próprios e sintagmas nominais introduzidos por artigo definido. Segundo Allen, o tratamento deste caso é complexo e depende do sintagma nominal sendo analisado. No caso de nomes próprios podemos apresentar uma nova constante para representar o novo indivíduo no contexto (caso ainda não exista uma para representá-lo), mas no caso de sintagma introduzido por artigo definido é necessário um estudo de cada caso em particular, como veremos no decorrer deste texto.

2.2.5 Candace Sidner

Sidner [38] distingue os casos de *Co-referência* (intra e intersentenciais) dos casos de *Referência Interna* (que são muito parecidos com os casos de referência anafórica e não-anafórica citados por Allen [1]). Os casos de referência interna são aqueles que não necessitam de um antecedente para serem identificados por serem de conhecimento mútuo do narrador e do ouvinte ou por existirem por alguma convenção da própria língua ou do contexto de mundo que está sendo utilizado. Por exemplo, a sentença:

(16)O presidente viajou.

tem sentido completo em um determinado contexto e em um determinado país. O ouvinte pode identificar "presidente" como sendo o atual presidente de seu país sem nenhum problema maior; basta que ele conheça as convenções utilizadas em seu contexto cultural (que o sistema é presidencialista, que o país possui um presidente e que ele gosta de viajar).

2.3 Referência Definida

Após estes conceitos gerais sobre o problema de referência, nos aprofundaremos no problema de referência definida que é um subconjunto dos casos de referência. O problema de referência definida não é visto como um problema separado dos problemas de referência, mas neste trabalho identificaremos os casos mais simples. Nossa proposta inicial é analisar os casos de sintagmas nominais introduzidos por **artigo definido**. Este casos são similares aos casos de referência por pronomes demonstrativos, mas não se referem exatamente ao mesmo problema.

Tentaremos abordar também os casos de referência definida envolvendo conjuntos de objetos, como é o caso de sintagmas nominais no plural ou referência a conjuntos, contrariando a idéia de referência definida dada por Searle [36].

2.3.1 Uma Breve Discussão sobre o Artigo Definido

Como o artigo definido será um elemento marcante na escolha dos sintagmas nominais a serem estudados, faremos um breve apanhado sobre o uso do artigo definido em português.

As seguintes definições para artigo definido e artigo indefinido foram retiradas de Cunha [9]:

> Dá-se o nome de **artigos** às palavras *o* (com as variações *o, a, os, as*) e *um* (com as variações *um, uma, uns, umas*), que se antepõem aos substantivos para indicar:
>
> 1. (**artigo definido**) que se trata de um ser já conhecido do leitor ou ouvinte, seja por ter sido mencionado antes, seja por ser objeto de um conhecimento de experiência, como nestes exemplos:
>
> > ...
> >
> > *Atravessaram* **o** *pátio, deixaram* **na** *escuridão* **o** *chiqueiro e* **o** *curral, vazios, de porteiras abertas,* **o** *carro de bois que apodrecia, os juazeiros.* (G. Ramos, *Vidas Secas*. 2ª edição, Rio de Janeiro, José Olympio, 1947)
>
> 2. (**artigo indefinido**) que se trata de um simples representante de uma dada espécie ao qual não se fez menção anterior:
>
> > *Era* **uma** *casinha nova, a meia encosta, com trepadeira pela varanda. Tinha* **um** *pomar pequeno de laranjeiras e marmeleiros*

e mais **uma** *hortazinha, ao longo do rego que descia do morro.* (R.M.F. de Andrade, *Velórios*, Belo Horizonte, Os Amigos do Livro, s.d.)

As formas do artigo são resumidas na tabela 2.1.

	Artigo Definido		**Artigo Indefinido**	
	Singular	Plural	Singular	Plural
Masculino	o	os	um	uns
Feminino	a	as	uma	umas

Tabela 2.1: Formas simples do artigo

Podemos ainda ter as formas combinadas do artigo expressas na tabela 2.2.

	Artigo Definido			
Preposições	o	a	os	as
a	ao	à	aos	às
de	do	da	dos	das
em	no	na	nos	nas
por (per)	pelo	pela	pelos	pelas

Tabela 2.2: Formas combinadas do artigo definido

Existe também o problema de ambigüidade léxica na identificação do artigo. Uma possibilidade para diminuir as ambigüidades é aplicar regras que possibilitem determinar aproximadamente se a palavra é um artigo, um pronome ou uma preposição. As regras seguintes, sugeridas por Pacheco [27], trazem uma boa solução para o problema. A idéia é propor uma classificação baseada no contexto direito (na classe gramatical a qual pertence a palavra à direita) do artigo, preposição ou pronome. A tabela 2.3 mostra alguns casos para a partícula "a".

2.3.2 Classificação

Iremos considerar basicamente quatro tipos de referências: co-referência e referência interna, que por sua vez, pode ser específica ou genérica. Denominamos **co-referência** os casos em que a referência e o antecedente estão presentes no mesmo texto, ou seja, existe uma introdução anterior de um possível referente no texto sendo analisado. A **referência**

Dupla de ocorrência no texto		Decisão sobre a classificação
palavra alvo	contexto direito	
a	artigo	**a** é preposição
a	pronome (possessivo)	**a** é artigo
a	pronome (relativo)	**a** é pronome
a	verbo (infinitivo)	**a** é preposição
a	verbo (não infinitivo)	**a** é pronome

Tabela 2.3: Exemplos de regras para reduzir a ambigüidade de *a*

interna contempla os casos em que não existe a introdução no texto de um possível referente, sendo necessário buscar no contexto conhecido pelo leitor (ou pelo computador). Essa classificação segue regras subjetivas e não possui um limite bem definido, servindo apenas para delimitar melhor nossa área de trabalho. Na tabela 2.4 vemos exemplos de alguns casos. Iremos ignorar os casos de referência não definida no momento, mas elas serão representadas de maneira análoga à referência definida.

	interna	**co-referente**
específica	**O presidente** viajou.	Ganhei um gato ontem. **O gato** não gosta de leite.
genérica	**O gato** é um mamífero.	Elefantes e mamutes são mamíferos. **Os mamutes** estão extintos.

Tabela 2.4: Exemplo de quatro tipos de referências definidas.

Os termos **genérica** e **específica** são baseados nos trabalhos de Bond [5] e seguem uma linha similar aos trabalhos de Hawkins [17] e de Krámský [32] sobre as funções do artigo definido e o conceito sobre o que é definido ou não (*definiteness*).

A **referência específica** é aquela que consegue identificar de maneira não ambígua um único objeto do domínio. Kadmon [21] comenta em seu trabalho alguns casos em que as referências são únicas e chega a citar condições para que as referências sejam realmente únicas.

A **referência genérica** não se preocupa em desfazer ambigüidades mas apenas em citar uma classe ou um grupo de entidades. Podemos dizer que é uma referência definida, pois ela consegue identificar de maneira única essa classe (como no exemplo (7): "Os mamutes são mamíferos").

A **referência específica** é aquela que se dirige a uma entidade (ou várias entidades) bem determinada e a **referência genérica** é aquela que se refere a uma classe de en-

tidades. Kadmon [21] comenta os casos em que ocorrem referências únicas, no sentido de não serem ambíguas, que levam sempre a uma mesma entidade em um determinado discurso e em um determinado mundo; apresenta uma breve análise para os casos em que as referências não são únicas. Uma referência determinada normalmente deve ser única, porém, uma referência não-única nem sempre será genérica. Por exemplo, em

(17) Pedro pôs a mão **no joelho** *de Ana.*

as referências são bem definidas, mas não são únicas (podem se referir a qualquer uma das mãos ou a qualquer joelho) no sentido apresentado por Kadmon [21]. Pode-se ainda dizer que a referência não é bem definida, pois a sentença poderia ser reescrita como:

(18) Pedro pôs uma mão em um joelho de Ana.

Na maioria dos casos reais não é necessário fazer a distinção exata entre um referente ou outro (no exemplo (18) entre o joelho esquerdo e direito).

Casos de Co-referência

A co-referência é caracterizada pela referência a algum referente já introduzido no texto. A busca por este referente é feita considerando-se apenas o texto sendo analisado. Algum contexto externo pode ser utilizado no processo de identificar possíveis relações entre referência e referente, mas nenhum objeto novo é criado para satisfazer esta relação. Os objetos considerados serão apenas aqueles mencionados no texto.

1. **direta:** Utiliza-se o artigo definido para indicar que o objeto sendo referido já foi mencionado no contexto. O narrador introduz um novo objeto (ou novo conceito) e o leitor precisa apenas fazer uma ligação do tipo **um-para-um** entre a referência e o objeto mencionado e concluir que se referem a um mesmo objeto. Por exemplo, na sentença

   (19) Há um livro com capa vermelha sobre a mesa. Passe-me **o livro** *por favor.*

   podemos ter co-referências que se refiram a mais de um objeto no texto, em um tipo de relação de **um-para-muitos** (também conhecidas como referência a conjuntos e que estamos classificando como referências genéricas), mas de maneira similar às co-referências um-para-um (referência específica). Podemos chamar estes casos de referência definida (contrariando as idéias de Searle) ainda se considerarmos que a

função principal da referência é identificar objetos dentro de um discurso entre um narrador e um ouvinte. Se os objetos podem ser identificados de maneira "definida", ou seja, o ouvinte pode identificar os objetos de maneira a diferenciá-los de outros objetos dentro do contexto (ou o narrador pode identificar a referência no decorrer do discurso), podemos dizer então que a referência é definida.

(20.1) Quando mergulhávamos perto do velho navio naufragado, nós encontramos uma moeda de ouro e um canivete enferrujado. Entregamos **os objetos** *à guarda costeira quando retornamos.*

(20.2) Quando mergulhávamos perto de um velho navio naufragado, nós encontramos uma moeda de ouro e um canivete enferrujado. Informamos para a guarda-costeira a posição **do navio**.

(20.3) Ana e Pedro saíram de casa. **O casal** *parecia feliz.*

No exemplo (20.1) acima a resolução da referência **os objetos** se dá pela busca de antecedentes na sentença anterior. Uma coisa importante a considerar é que a busca é feita primeiramente na oração principal que diz que encontraram uma moeda de ouro e um canivete enferrujado. Como estes objetos satisfazem a referência a "objetos" podemos incluí-los em um conjunto solução para esta referência.

No exemplo (20.2) existe uma referência direta de "navio" na segunda sentença para "navio" na primeira sentença.

O exemplo (20.3) tem uma referência a "casal", que é satisfeita por um conjunto de duas pessoas. O conjunto que satisfaz a referência é composto por Ana e Pedro.

2. **Associação:** Existe algum tipo de relação entre os dois sintagmas nominais, como uma relação de sinonímia, pertinência, estar-contido-em, fazer-parte-de, ser-constituinte-de, compra-e-venda e outras. Estes tipos de associações podem ser variadas e não previsíveis, sendo necessário fazer novas associações durante a interpretação de um discurso, o que é comum no uso da língua. Alguns dos exemplos abaixo foram baseados em Scheler [35].

(21.1) Ana viajou para Munique. **A jornada** *foi longa e cansativa.*

(21.2) O carteiro trouxe-nos uma caixa esta manhã. **O vaso** *estava quebrado.*

(21.3) Ana pegou as coisas do piquenique. **A cerveja** *estava quente.*

(21.4) Quando trouxeram a escrivaninha para a minha sala **a gaveta** *já estava quebrada.*

(21.5) No dia seguinte ao que vendemos o carro o comprador voltou e pediu **o dinheiro** *de volta.*

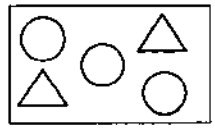
No exemplo (21.5) temos uma associação entre comprador, compra e venda; existe um encadeamento de idéias que é necessário para deduzir (ou induzir) que o dinheiro referido origina-se do ato de compra-e-venda efetuado entre as duas partes envolvidas (comprador e vendedor).

Podem existir casos em que a associação não é bem conhecida, como por exemplo em:

Ana pegou as coisas do piquenique. O **sushi** *estava aberto.*

Mesmo que não seja comum levar *sushi* num piquenique, o leitor pode fazer uma associação utilizando seu próprio raciocínio. Talvez fosse necessário fazer uma associação entre *sushi*, comida e piquenique. O mesmo pode valer para o caso da cerveja.

3. **Restrição:** É uma referência baseada na dependência mútua entre os objetos. É necessário mais de um objeto ao mesmo tempo para se conseguir identificar um deles (ou o conjunto formado por eles).

(22)  **O círculo sobre o triângulo** *será pintado.*

O exemplo (22) mostra um caso onde não é suficiente identificar um objeto do tipo círculo e um objeto do tipo triângulo separadamente, mas é necessário identificar o relacionamento entre eles. Neste exemplo o relacionamento se refere à posição relativa entre os objetos.

Casos de referência interna

O nome **referência interna** refere-se ao fato de que o leitor (ou o computador) cria uma representação interna para o referente, quando não é possível identificá-lo. Esta representação existe apenas na sua interpretação e provê significado à sentença lida, mesmo sem identificar o referente.

1. **Unicidade:** o artigo indica a unicidade da entidade. Se a entidade não é única por natureza então o artigo serve para afirmar que esta entidade referida é diferente de qualquer outra entidade pertencente ao contexto.

(23.1) **O rei da França** *é calvo.*
(23.2) **O presidente** *viajou.*

O leitor entende que a pessoa referida é única e não precisa de apresentações. O simples fato de ser rei da França permite que se encontre (ou não) a pessoa correspondente na época atual.

Existem casos mais simples em que o artigo é utilizado apenas para confirmar a unicidade já dita na sentença:

(24) **A primeira pessoa a navegar na América** *foi um Islandês.*

2. **Definição:** Pode-se fazer uma referência a alguma coisa que ainda está sendo definida ou a um novo tema sendo introduzido. Searle [36] classificaria este tipo de referência como sendo apenas predicativa e não referencial. Por exemplo:

específica:
(25.1) **O fato** *é que existe muita vida na Terra.*
(25.2) Podemos considerar a **fermata**, *um recurso de notação musical, como uma maneira de indicar variação na duração de uma nota.*
genérica:
(25.3) **Os mamutes** *são animais parecidos com elefantes.*

No exemplo (25.1) primeiro se apresenta "o fato" e logo em seguida este fato é caracterizado. Esta caracterização confirma a existência do fato e o torna único. Podemos ver o "fato" como uma entidade e uma das propriedades dessa entidade é "existe muita vida na Terra".

O exemplo (25.2) é um caso similar, apenas com uma sintaxe mais complexa. A propriedade de "fermata" seria "uma maneira de indicar variação na duração de uma nota" e isto é suficiente para dar um sentido de existência e unicidade ao termo.

2.4 Referência Não Definida

Iremos utilizar este termo para indicar as referências introduzidas por artigo indefinido. Isto não quer dizer que a referência não seja possível de se definir, mas indica apenas que a referência não tem o objetivo de explicitar o objeto sendo referido.

Segundo Cunha [9], a função do artigo indefinido é antagônica a do artigo definido:

> "O **artigo definido** é, essencialmente, um sinal de notoriedade, de conhecimento prévio por parte dos interlocutores, do ser ou do objeto mencionado: o **artigo indefinido**, ao contrário, é por excelência um sinal da falta de notoriedade, de desconhecimento individualizado, por parte de um dos interlocutores (o ouvinte), do ser ou do objeto em causa."

A diferença entre o artigo indefinido e o artigo definido considerando-se suas funções é bem clara quando consideramos o nível de determinação que cada um traz.

No exemplo abaixo vemos uma mesma sentença reescrita com o uso de artigo definido e pronome demonstrativo:

> *Foi chegando* **um** *caboclinho magro, com* **uma** *taquara na mão.* (A. Amoroso Lima, *Afonso Arinos*, Rio de Janeiro, Lisboa-Porto, 1922)

Reescrevendo a mesma sentença:

> *Foi chegando* **o** *caboclinho magro, com* **a** *taquara na mão.*
>
> *Foi chegando* **este** *caboclinho magro, com* **esta** *taquara na mão.*

Note que o nível de determinação do objeto tratado aumenta de acordo com o uso de artigo definido e do pronome demostrativo.

A importância das referências indefinidas torna-se clara quando consideramos que é ela que apresenta um objeto novo no discurso. Este objeto será forte candidato para ser um referente numa situação posterior.

Revendo o exemplo (15)

> *(15) Encontrei* **uma** *caneta azul e* **um** *lápis vermelho.* **A caneta** *não funcionava.*

nota-se o responsável pela apresentação da caneta na primeira frase foi o artigo **uma**. Esta será uma das características exploradas para se resolver os casos mais diretos de referência definida, na qual basta relacionar os sintagmas "uma caneta" e "a caneta".

2.5 Conclusão

Neste capítulo fizemos uma apresentação e classificação do problema de acordo com o ponto de vista de alguns autores e acrescentamos algumas definições próprias. O que temos

é um problema extenso e com muitas características que podem variar de acordo com a abrangência e profundidade com a qual se pretende tratá-lo. Procuramos, então, limitar os casos de referência definida e agrupá-los dentro de uma classificação genérica. Nem todos os casos de referência definida são classificados aqui, mas o principais problemas de referência que pretendemos tratar foram apresentados. Considerando o tratamento dos casos aqui apresentados, estaremos limitados à representação computacional utilizada, descrita no capítulo 3, utilizando a abordagem multiagentes (capítulo 4).

Uma classificação resumida pode ser vista na tabela 2.5.

Referências		
Definida		Não-definida
Co-referência	Interna	
- direta ("Ana comprou um vaso. **O vaso** quebrou.") - por associação ("Ana viajou para Munique. **A jornada** foi longa e cansativa.") - por restrição "**O círculo** sobre o triângulo será pintado.")	- indicando unicidade ("O Rei da França é calvo.") - indicando definição ("**O fato** é que existe muita vida na Terra.")	- apresenta uma nova entidade no contexto ("Encontrei **uma caneta** azul.")

Tabela 2.5: Quadro geral de classificação

Capítulo 3

A representação lógica utilizada no sistema

Este capítulo descreve o tipo de análise sintática[1] utilizada, seguindo normas da gramática brasileira e a codificação para cláusulas de Horn (cláusulas do Prolog). O analisador sintático tem como entrada sentenças em língua natural e como saída proposições lógicas em forma de cláusula. Em uma segunda fase, estas proposições são expandidas em mais proposições para explicitar as relações entre as referências e os referentes em um texto.

A primeira codificação servirá como base para uma fase seguinte que receberá como entradas as sentenças já codificadas em forma de cláusulas de Horn. A representação é importante para o problema porque é nesta fase que serão marcadas as referências que devem ser associadas a seus referentes. O sistema multiagentes proposto no capítulo 5 se utilizará dessa representação.

Serão descritos, ainda, o fragmento de gramática utilizado, a fundamentação lingüística e o método utilizado na codificação das sentenças.

3.1 As fases de codificação

Trabalharemos com proposições lógicas conforme a seguinte definição:

Proposição: Uma proposição é uma sentença lógica escrita em uma linguagem lógica que pode ser avaliada e associada a um determinado valor lógico verdadeiro ou falso.

[1] Em inglês: *parsing*

A conversão de sentenças para proposições é um processo que procura sintetizar ao máximo o significado de uma sentença. Estamos com isso limitando a representatividade da Língua Natural, pois estamos trabalhando com uma linguagem lógica e formal, com suas vantagens do ponto.de vista computacional e desvantagens do ponto de vista lingüístico e comunicativo. Rayner [31], em sua tese, defende a utilização da codificação em Cláusulas de Horn dada a facilidade com que se pode fazer inferências e deduções sobre uma base de dados codificada desta forma. Scha [34], apesar de defender o uso desse tipo de cláusulas, também diz que este é um método incompleto para representar a língua natural, pois é difícil chegar a alguma conclusão quando há falta de dados para testar uma proposição. Não é possível saber se a proposição é falsa por falta de dados ou porque ela realmente contraria alguma outra proposição dentro da mesma base de dados. Não podemos dizer que estas são limitações a este trabalho, pois algumas vezes queremos trabalhar apenas com o valor lógico das proposições e não com todo o seu conteúdo semântico; assim, alguns autores também utilizam este tipo de codificação em seus trabalhos, como McCord [25] e Pereira e Shieber [29, 30], convivendo com as vantagens e desvantagens desta abordagem.

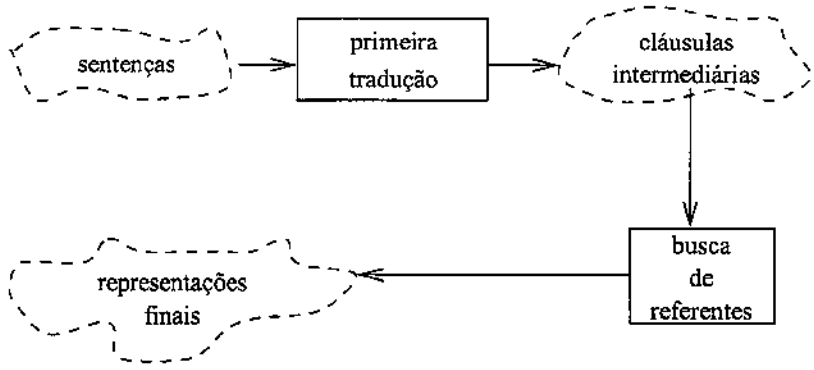


Figura 3.1: Fases de processamento.

A conversão das sentenças passa basicamente por duas fases: a primeira busca levantar os problemas de referência, que chamamos de **primeira tradução**, e a segunda faz a **busca de referentes**. A figura 3.1 mostra em termos gerais o processamento efetuado pelo sistema.

Primeira tradução: são indicados os objetos que fazem referência ou que são possíveis candidatos a referentes. Essas indicações são feitas através de cláusulas que permanecem livres para serem unificadas[2] posteriormente.

[2]Considera-se, neste trabalho, que uma cláusula livre é aquela que possui pelo menos uma variável livre e a unificação ocorre quando uma variável livre é trocada por uma constante.

Na primeira tradução é feita a análise sintática para efetuar conversão de sentenças em língua natural para cláusulas. O sistema de análise sintática é baseado na gramática gerativa, proposta inicialmente por Chomsky, utilizando um fragmento da gramática do português com base em Luft [23]. Durante a análise sintática é feita a categorização das palavras e o co-relacionamento das entidades em uma mesma sentença. Esta fase é bem distinta dentro do sistema e utiliza o Lex (um gerador de analisadores léxicos para linguagem C) e Yacc (um gerador de analisadores sintáticos para linguagem C). Uma referência inicial sobre estas ferramentas pode ser vista em Levine [22]. Uma das saídas do analisador sintático é um conjunto de cláusulas do Prolog para cada sentença em português fornecida como entrada. Este tipo de abordagem possui um bom grau de representatividade, mantendo a simplicidade e o formalismo necessários, segundo van Noord [42].

Busca de referentes: nesta fase o sistema faz as possíveis unificações, substituindo variáveis livres por constantes que representam os referentes. Quando mais de um referente satisfaz a referência, várias cláusulas são geradas para representar a ambigüidade da referência. Cláusulas repetidas, ou equivalentes entre si, não são geradas. Veremos no capítulo 4 como o sistema multiagentes faz o tratamento dessas cláusulas.

Ao final do processo espera-se ter uma base de dados com várias cláusulas representando as sentenças iniciais. Estas cláusulas ficam à disposição do sistema para servir de base para a análise das sentenças seguintes. Conforme novas sentenças são analisadas a verificação de quais referências foram feitas ocorre sobre estas cláusulas.

3.2 A representação escolhida

Em uma primeira instância, a sentença é representada a partir de seus elementos mais básicos (os sintagmas) que contribuem para a execução de uma ação (o verbo) ou a determinação de um estado. Seria o emprego do princípio de Frege: "o significado de uma sentença como um todo é função do significado de suas partes" (citado em Peirce [28]). O modelo básico é dado por:

$$SN_1, SN_2, \ldots, SN_n \longrightarrow \text{Verbo}$$

Seguindo este modelo, uma seqüência de n sintagmas nominais têm como resultado um estado ou uma ação. Por exemplo, a sentença

(1) Pedro corre.

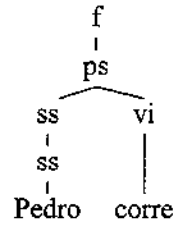


Figura 3.2: Parser de uma sentença.

utilizando-se a notação do Prolog torna-se uma proposição do tipo:

```
(1.1)  pedro('Pedro').
(1.2)  corre(X) :- pedro(X).
```

A cláusula (1) faz parte do conhecimento global do sistema. Podemos ler a cláusula acima como **"se X é Pedro então X corre"**, ou ainda, se existe algum objeto `X` que satisfaz a propriedade de ser Pedro, então `X` corre. Do nosso ponto de vista lógico tem o mesmo significado da sentença original **"Pedro corre"**.

Outras sentenças também poderiam ser representadas dessa forma, podendo-se ainda encadear as idéias (ou fazer ligações entre os objetos envolvidos), que é a base para se compreender e representar a coerência do texto. As relações entre os elementos das sentenças criam uma cadeia que possibilita a confirmação de uma ação ou de um sintagma.

O exemplo

(2) Pedro comprou um livro.

é codificado em:

```
(2.1) comprou(X,Y) :- pedro(X), undef(Y, livro(Y)).
(2.2) undef(a, livro(a)).
```

A idéia permanece a mesma da sentença anterior, mas utiliza-se mais uma cláusula para representar a sentença.

A proposição "`undef(Y, livro(Y))`" da cláusula (2.1) quer dizer que existe um livro `Y` mas ele não é definido. Na cláusula (2.2) a expressão "`undef(a, livro(a))`" é apresentada uma constante "a" para se ter uma representação imaginária de "a", ou seja, se existe um livro, então ele pode ser representado por "a".

Acrescentando mais uma sentença:

(3) O livro é famoso.

obtemos a seguinte codificação:

```
(3.1) famoso(X) :- def(X, livro(X)).
(3.2) def(a, livro(a)).
(3.3) famoso(a) :- livro(a).
(3.4) livro(a).
```

Quando se codifica "o livro é famoso" é necessário respeitar a sentença anterior e fazer a instanciação da variável livre `X` na cláusula (3.1) com a constante "a", da cláusula (2.2), que representa o referente "livro". As cláusulas seguintes (3.2,3.3 e 3.4) utilizam o mesmo referente

Na figura 3.3 vemos a codificação da sentença do exemplo (4).

(4) Ana pegou as coisas do piquenique.

```
(4.1)        pegou(X,Y):-
                  ana(X),
                  def(Y,coisas(Y)),
                  def(Z,piquenique(Z)),
                  do(Y,Z).
(4.2)        do(X,Y):-
                  def(X,coisas(X)),
                  def(Y,piquenique(Y)).
```

Figura 3.3: Um exemplo de análise e tradução.

Os exemplos apresentados neste capítulo procuram mostrar o mecanismo básico utilizado no sistema. A partir deste código inicial são desenvolvidas as heurísticas descritas no capítulo 5.

3.3 Intensionalidade e Extensionalidade das Sentenças

Esta seção mostra a dificuldade que existe na representação de sentenças da Língua Natural, pois nem todas possuem uma relação direta com a linguagem lógica. Estaremos justificando porque não é possível representar todas as sentenças corretamente.

Segundo Gallin [14], quando falamos em significado de uma sentença existe uma distinção entre a extensão e **intensão**[3] de seu valor real. A língua natural é muito rica em transmitir idéias sem estarem diretamente ligadas a entidades que possuem um valor lógico. Este fator a diferencia da linguagem lógica ou matemática, que trabalha essencialmente com símbolos. A intensão não está ligada diretamente aos símbolos, mas às funções que estes símbolos representam para quem os utiliza. A intensão de uma sentença existe independentemente das entidades por ela referidas. O nosso trabalho limita-se a identificar apenas as entidades que são referidas em algum ponto do discurso ou que são inseridas propositalmente para dar sentido aos exemplos[4].

Nesta tese não tratamos a representação intensional das sentenças, considerando apenas o universo extensional da representação semântica. Como o nosso objetivo é estudar a utilização de um sistema multiagentes para o problema da referência, é interessante utilizar uma abordagem mais simples e eficiente, sem se preocupar com esse nível de complexidade lingüística, deixando-a para estudos posteriores.

Descrevemos a seguir os motivos que nos levaram a não tratar os exemplos considerados intensionais.

3.3.1 Sentenças "intensionais"

Algumas sentenças não possuem um referente específico, mas um conceito ou uma "intensão". Do ponto de vista computacional não é simples representar uma intensão, pois a lógica mais utilizada é a extensional. A lógica intensional, conforme Halvorsen [16], é capaz de representar alguns fenômenos de linguagem ditos intensionais, mas o tratamento não se demonstra viável.

Vemos abaixo um exemplo utilizado para demonstrar este tipo de interpretação.

(5) A primeira mulher a pisar na lua casou-se na semana passada.

Note que para determinar quem é a primeira mulher a pisar na lua seria necessário um bom conhecimento de mundo, ou seja, um contexto muito maior do que o exemplo citado. Não queremos entrar em detalhes de contexto e construção de conhecimento de mundo. Segundo alguns autores, seria errado interpretar a sentença acima somente com o sentido

[3]Observe que esta palavra não é do português (intenção). Podemos dizer que ela possui a idéia contrária de extensão, ou seja, não possui uma extensão no mundo real que apresente um objeto como referente.

[4]o discurso é construído e interpretado com base apenas naquilo que está sendo escrito nele mesmo ou em informações explicitamente construídas para o seu contexto.

extensional, pois não conseguiremos encontrar em nosso mundo a primeira mulher a pisar na lua, uma vez que, até o presente momento nenhuma mulher pisou na lua. Talvez passe a existir em um futuro próximo mas, enquanto isto, apenas a intensão da sentença faz sentido.

Outros exemplos clássicos:

(6) O presidente viajou.

(7) O presidente é eleito a cada cinco anos."

A sentença (6) é extensional e podemos considerar que um único indivíduo, que satisfaz a propriedade de ser presidente, viajou. Uma possível representação seria:

```
(6') viajou(X) :- presidente(X).
```

Na cláusula (6') `presidente(X)` poderia unificar com algo como `presidente(''Fernando Henrique Cardoso'')`.

A sentença (7) não funciona da mesma maneira, pois quem será eleito a cada cinco anos é algum candidato ao cargo. Por outro lado, alguém poderia estar fazendo uma observação, dizendo que *o mesmo* presidente é eleito a cada 5 anos, então a sentença seria classificada como extensional. Concluimos que não é possível fazer uma classificação arbitrária do que pode ser representado extensional ou intensionalmente.

3.3.2 Sentenças Extensionais

Estaremos representando, neste trabalho, as sentenças extensionais utilizando apenas a lógica de primeira ordem, sem recorrer às outras lógicas de mais alta ordem (ou lógicas modais). Alguns detalhes da língua natural se perdem neste tipo de representação, mas a consideramos suficiente para o tipo de trabalho que estamos desenvolvendo. A crítica a este tipo de abordagem seria o fato de que praticamente toda sentença possui um intensão, além de uma extensão, que vai depender muito do contexto de utilização da mesma. Entretanto, este tipo de conhecimento vai além do que conseguimos representar computacionalmente.

Vemos abaixo alguns exemplos extensionais e sua representação em cláusulas de Horn.

(8) Pedro comprou um livro.

(9) O autor é José de Alencar.

codificação:

```
(8.1) comprou(X,Y):-pedro(X),undef(Y,livro(Y)).
(8.2) undef(a,livro(a)).
(9')  def(X,autor(X)):-josedeAlencar(X).
```

Esta representação traduz apenas os principais pontos da sentença. Não existe uma interpretação da língua natural. A associação do autor com o livro será feita posteriormente quando os agentes receberem estas sentenças.

3.4 A Gramática Utilizada

A gramática utilizada pretende representar apenas um pequeno fragmento da língua portuguesa, limitando-se aos exemplos utilizados nesta tese. Nada impede que esta gramática seja expandida para trabalhar com um fragmento maior. Ela está baseada na obra de Luft [23].

Na figura 3.4 vemos o fragmento utilizado nesta tese em forma de gramática gerativa.

3.5 O Analisador Léxico e Sintático

O analisador sintático converte sentenças em língua natural para a forma de cláusulas. As cláusulas são geradas arbitrariamente conforme regras programadas no analisador léxico e sintático. O dicionário léxico contém apenas as palavras utilizadas nos exemplos desta tese, pois não era um objetivo inicial desenvolver um analisador sintático. O analisador sintático foi desenvolvido porque o método de tradução para fórmulas é muito particular para os casos sendo tratados e algumas regras foram incluídas arbitrariamente para se chegar às fórmulas lógicas desejadas para cada sentença.

O dicionário léxico classifica as palavras somente quanto à sua classe gramatical (as classes gramaticais podem ser vistas na legenda da figura 3.4). O analisador sintático é capaz de gerar uma árvore sintática de forma parentetizada, como nos mostra o exemplo abaixo, mas esta estrutura não é utilizada no restante do trabalho. Serve apenas para visualizar como foi feita a análise de uma determinada sentença.

A sentença do exemplo (4).

(4) Ana pegou as coisas do piquenique.

A árvore sintática em forma parentetizada mostra o léxico reconhecido no nível mais interno de parênteses e, nos níveis mais externos, os tipos reconhecidos.

f(ps(oa(ss(sp(Ana)), vtd(pegou), ss(artdef(as), sc(coisas)), snp(prepart(do), sc(piquenique)))))

As clásulas resultantes do analisador sintático procuram manter alguma informação semântica da sentença original. Estas cláusulas atendem apenas às necessidades deste trabalho.

```
pegou(G43,G44):- ana(G43), def(G44, coisas(G44)),
   def(G53,piquenique(G53)),do(G44,G53).
do(G73,G74):-def(G73, coisas(G73)), def(G74,piquenique(G74)).
```

Existem, ainda, muitos detalhes a serem implementados no parser. Um item a ser tratado é a ambigüidade das sentenças. Atualmente apenas a primeira análise é considerada e nenhuma ambigüidade léxica ou sintática é representada.

3.6 Conclusão

Descrevemos neste capítulo o uso de gramática gerativa no analisador sintático das sentenças e a sua codificação para lógica de primeira ordem. Vemos que nem todos os tipos de sentenças são tratados e codificados, nos limitando às sentenças extensionais, mas concluímos que este tipo de codificação é suficiente para o propósito deste trabalho.

f ⟶ pc PERIODO
f ⟶ ps PERIODO
f ⟶ oai QUESTIONMARK
pc ⟶ pcs
pc ⟶ pcc
pcs ⟶ os ss VLIG sadj
pcs ⟶ ss VTD os
pcs ⟶ oa os
pcc ⟶ oa CONJ oa
os ⟶ SB o
o ⟶ oa
ps ⟶ oa
oa ⟶ ss snp VLIG sadj
oa ⟶ ss snp VLIG ss
oa ⟶ ss VLIG sadj
oa ⟶ ss VLIG ss
oa ⟶ ss VI snp
oa ⟶ ss VI
oa ⟶ ss VTD ss snp
oa ⟶ ss VTI snp
oa ⟶ ss VTD ss
oa ⟶ VIMP
ss ⟶ ARTDEF SC
ss ⟶ ARTUNDEF SC
ss ⟶ PROINDADJ SC
ss ⟶ SP CONJ SP
ss ⟶ SP
ss ⟶ PRPESS
snp ⟶ PREP VI
snp ⟶ PREPART SC
snp ⟶ PREPART SP
sadj ⟶ ADJ
oai ⟶ ARTDEF SB VI
oai ⟶ ARTDEF SB ss VTD
oai ⟶ PROINT VLIG sadj
oai ⟶ PROINT VLIG ss snp
oai ⟶ PROINT VLIG ss
oai ⟶ PROINT VTD ss
oai ⟶ PROINT VTI snp
snp ⟶ PREP SC

Legenda:
ADJ - adjetivo
ARTDEF - artigo definido
ARTUNDEF - artigo indefinido
CONJ - conjunção
PERIODO - ponto final
PREPART - contração de preposição com artigo
PROINDADJ - pronome indefinido adjetivo
PROINT - pronome interrogativo
QUESTIONMARK - ponto de interrogação
SB - subordinador
SC - substantivo comum
SP - substantivo próprio
VI - verbo intransitivo
VIMP - verbo impessoal
VTD - verbo transitivo direto
f - frase
o - oração
oa - oração absoluta
oai - oração absoluta interrogativa
os - oração simples
pc - período composto
pcc - período composto por coordenação
pcs - período composto por subordinação
ps - período simples
sadj - sintagma adjetivo
snp - sintagma nominal preposicionado
ss - sintagma substantivo
PRPESS - pronome pessoal do caso reto

Figura 3.4: Gramática utilizada.

Capítulo 4

Sistemas Multiagentes

Apresentaremos neste capítulo a notação utilizada no SMA e suas características. Descreveremos, também, o agente isoladamente e sua função dentro do sistema.

A pesquisa sobre Multiagentes em IA é uma área emergente e ainda pouco utilizada em PLN, portanto, ainda com poucos resultados. O tema *Multiagentes* está relacionado à IADz (Inteligência Artificial Descentralizada), na qual a busca de soluções para um problema é feita de maneira não linear e distribuída através de unidades autônomas e inteligentes, as quais denominamos **agentes**.

Vale salientar aqui a diferença entre Inteligência Artificial Distribuída e Inteligência Artificial Descentralizada, conforme Demazeau [12]:

Inteligência Artificial Distribuída (IAD): se baseia em uma "solução global de problemas por um grupo de entidades distribuídas". Estas **entidades** podem ser simples ou complexas, com capacidade de comportamento racional ou não. A solução do problema é feita de maneira sempre cooperativa e depende de um alto grau de compartilhamento de informações para que se possa atingir uma solução única (global). Para que um sistema seja chamado de distribuído basta que as unidades de controle (os algoritmos) e os dados estejam fisicamente ou logicamente distribuídos.

Inteligência Artificial Descentralizada (IADz): está relacionada com a atividade de agentes autônomos em um ambiente multiagentes, portanto, mais próxima da idéia de sistemas multiagentes. Numa sociedade multiagentes é desejável que não exista um controle centralizado do sistema, preferindo-se um controle distribuído, pois isto torna o sistema mais flexível a alterações e modular, garantindo independência aos agentes. Quanto mais flexível for o sistema melhor será seu desempenho e modularidade.

Durfee e Rosenschein [13] fazem uma comparação entre Solução Distribuída de Problemas (*Distributed Problem Solving* - DPS) e Sistemas Multiagentes (SMA). Estas duas linhas de trabalho existentes em Inteligência Artificial são basicamente complementares, pois possuem muitas características em comum e mesmo assim não são consideradas a mesma coisa. Um SMA se preocupa mais com a cooperação entre os agentes que devem mostrar um comportamento "inteligente" quando em conjunto. Um sistema que utiliza DPS como abordagem, por sua vez, busca resolver um determinado problema de maneira eficiente e segura e, para tanto, é necessário que a comunicação e a cooperação entre os agentes seja confiável. Podemos focalizar a diferença por este ponto de vista: o DPS é orientado ao problema e o SMA é orientado ao agente. Enquanto o primeiro necessita de um estudo preciso do problema para propor uma solução eficiente, o segundo se preocupa em coordenar diversos agentes em uma comunidade para chegar a uma solução, independente de um conhecimento mais profundo do problema. O SMA busca distribuir a complexidade de um problema e não apenas analisá-lo e posteriormente quebrá-lo em partes como faria um sistema distribuído.

Na realidade não existe uma conclusão precisa quanto as diferenças entre DPS e SMA. Considerando este fato, classificamos nosso sistema como tendo caracteríticas de DPS e SMA, pois na verdade partimos de um problema único, o subdividimos em partes menores (característica de DPS) e adotamos uma abordagem orientada a agentes para resolvê-lo (característica de SMA). Os agentes desenvolvidos são especializados em resolver um determinado tipo de problema, pois quando eles são projetados e desenvolvidos, devemos ter em mente o seu objetivo como colaborador para resolver o problema maior que é a referência definida.

Consideraremos apenas que o sistema que estamos propondo é um SMA e iremos trabalhar dentro dos limites e possibilidades desta abordagem.

A idéia inicial de se utilizar um sistema multiagentes surge quando consideramos a interpretação da língua como um processo não linear e que depende de um forte interelacionamento com diversas áreas de conhecimento. Note-se que este não é um problema que pode ser facilmente distribuído, mas um problema complexo e que pode ser decomposto em subproblemas. Dependemos de muitas "ilhas" de conhecimento para sermos capazes de entender uma simples sentença. Para termos uma compreensão global de uma sentença é preciso que estas "ilhas" tenham um meio de comunicação entre si com o objetivo de compor o sentido da sentença.

Um Sistema Multiagentes pode ser utilizado para representar uma seqüência de sentenças em língua natural que compõem um texto. Nossa proposta para tal sistema é que todos os agentes recebam a mesma sentença e que cada um tente fazer uma representação lógica diferente para aquela sentença. As representações lógicas feitas por cada agente

também podem ser relacionadas, ainda que divergentes. As ligações ou relações entre os objetos das sentenças seriam o meio pelo qual se explicitam as referências entre os objetos de um texto. Se o "significado de uma sentença é função do significado de suas partes" (Peirce [28]), então podemos imaginar que o "significado de um texto é uma função do significado de suas sentenças", considerando que uma sentença isolada pode ter um significado muito diferente do que se considerarmos a mesma sentença dentro de um contexto maior.

Neste capítulo estaremos apresentando um modelo geral para o sistema multiagentes que iremos utilizar na seção 5. Neste modelo apresentamos a arquitetura básica de um agente e seu modo de interação com o ambiente, fazemos a descrição de um agente e mostramos um exemplo.

4.1 Um Agente

Existe uma distinção entre agentes cognitivos[1] e agentes reativos. Um **agente cognitivo** é visto como uma entidade inteligente e autônoma, que possui uma representação de mundo e um tipo de comportamento baseado em planejamento e negociações, com o objetivo de obter uma coordenação com outros agentes. O fato de ser autônomo garante que um agente exista independentemente de outros agentes. Ele possui uma base de conhecimento (representação simbólica do ambiente), capacidade de decisão, capacidade de inferência e capacidade de comunicação (Berthet [3], Demazeau [11], Shoham [37]). Podemos ainda dizer que um agente deve ser dinâmico e capaz de se adaptar às alterações do ambiente um comportamento comum a um ser que consideramos "inteligente" (Beer [2], Boissier [4], Cohen [8]). Para um agente ser **reativo** não é necessário que tenha uma base de conhecimento própria e nem um raciocínio simbólico muito complexo. O seu comportamento baseia-se em estímulo/resposta, respondendo apenas ao estado atual do ambiente em que se encontra. É comum encontrarmos características cognitivas e reativas em agentes, não existindo um limite claro entre estes tipos.

Um agente deve ser capaz de fazer representações do mundo em que se encontra, utilizar um processo de inferência para derivar novas representações e utilizá-las para deduzir o que fazer. Normalmente o agente está limitado pelo mundo em que se encontra e sua visão não ultrapassa estes limites. Dentro deste contexto, ele tenta atingir os objetivos ou resolver um problema proposto, como mostra a figura 4.1.

No projeto de um sistema multiagentes é importante considerar o quão abrangente é o domínio de um agente, ou seja, em que nível de granularidade um agente do sistema

[1] ou deliberativos.

irá contribuir para o tratamento do problema. No nosso caso, um agente tem a visão do problema em nível sentencial e é capaz de armazenar e trabalhar com mais de uma sentença.

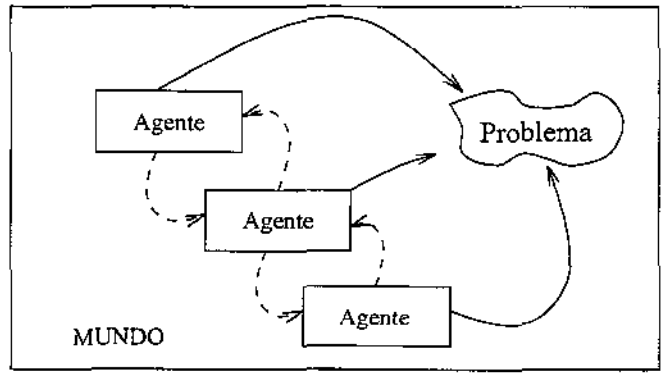


Figura 4.1: Agentes, um mundo e um problema.

Um componente central de um agente é sua base de conhecimento. Uma base de conhecimento é um conjunto de representações de fatos e objetos do mundo. No nosso caso o mundo é o contexto de Processamento de Língua Natural, com a representação do mundo em forma de cláusulas de Horn e com implementação em Prolog.

Um agente pode ser visto em três camadas lógicas diferentes, conforme Russell [33]:

1) **Nível de Conhecimento:** é o nível mais abstrato onde são feitas as negociações de mais alto nível. Em um sistema de PLN este seria o nível da própria língua natural, onde é feita a comunicação com o resto do mundo.

2) **Nível lógico:** nível onde o conhecimento é codificado em sentenças (proposições) lógicas, como descrito no capítulo 3. Existe neste nível a responsabilidade de converter a língua utilizada no mundo real em uma linguagem mais acessível e com menos complexidade que a original (troca-se a língua natural pela linguagem formal das cláusulas de Horn). A figura 4.2 mostra a separação entre o mundo "real" e a representação interna utilizada pelo agente.

3) **Nível de implementação:** é o nível no qual as informações são codificadas computacionalmente, utilizando-se alguma linguagem de programação. No nosso caso a linguagem utilizada é o Prolog e o C++.

Estes três níveis de abstração permitem diferentes tipos de visão dentro do projeto de um sistema.

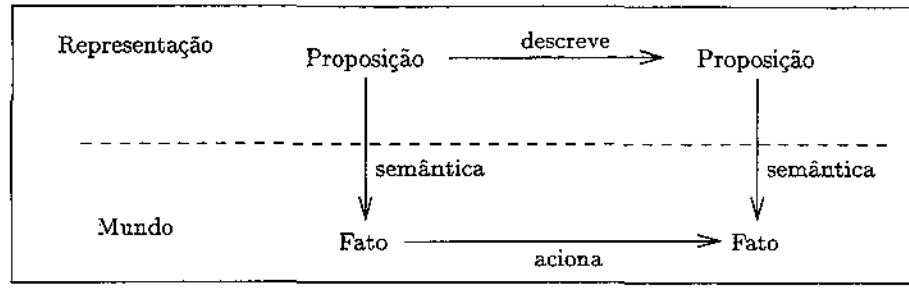


Figura 4.2: A representação interna e a realidade do mundo.

A semântica a qual se refere a figura 4.2 é a relação entre a representação interna dos agentes com os fatos do mundo. Seriam interpretações para as proposições.

Além dos componentes básicos, mostrados na figura 4.3, os agentes nascem com um comportamento pré-determinado, que os capacitam a se comunicar, negociar e tomar decisões. Esse comportamento deve ser descrito ainda na fase de projeto do agente. Isto quer dizer que um agente já deve nascer com sua capacidade de raciocínio.

Componentes do agente	descrição
Memória	base de dados local
Percepção	capacidade de ouvir outros agentes
Capacidade de decisão	através de regras e uma máquina de inferência
Capacidade de negociação	através de um protocolo de negociação
Capacidade de comunicação	através de um protocolo de comunicação

Figura 4.3: Composição básica de um agente.

Nas subseções seguintes são descritos os componentes dos agentes em detalhes.

4.1.1 Memória

A memória é onde o agente guarda representações de fatos e objetos do seu mundo. Estes dados são obtidos a partir de sua percepção. Uma vez que eles se encontram em seu poder um agente pode decidir se vai compartilhar os dados com outros agentes ou se vai apenas guardar para si mesmo, selecionando as possíveis consultas vindas de outros agentes que irá responder.

O agente pode ser "benevolente" ou "egoísta", compartilhando todos os dados com os seus companheiros ou não compartilhando nada. O agente benevolente não se preocupa em limitar o acesso às suas informações e responderá a todas as consultas que forem feitas a ele, enquanto que o agente "egoísta" não se preocupa em responder a todas as

consultas feitas a ele. A questão das vantagens ou desvantagens de cada comportamento dependerá muito do tipo de aplicação a ser desenvolvida. Em alguns casos, é vantajoso não compartilhar dado algum porque um agente pode tirar conclusões erradas sobre um determinado problema e especular soluções baseadas em observações próprias. Neste caso, uma possível desvantagem seria que esta abordagem limita a diversidade de informações geradas no sistema, mesmo que contraditórias. A vantagem do agente que compartilha todas as informações está no fato de que não existe nenhum tipo de seleção destas informações. A desvantagem é a geração de informações que podem ser conflitantes entre si, resultando uma dificuldade adicional, que seria administrar estes conflitos. Em nosso sistema, os agentes não fazem uma seleção de respostas, atendendo a todas as consultas que lhes são feitas, caracterizando-os como agentes benevolentes.

Se coordenarmos todas as memórias de cada agente em uma única base de dados teremos uma base de dados distribuída que estaria sofrendo alterações ao mesmo tempo, de acordo com a percepção de diferentes agentes no sistema. As representações e as observações feitas pelos agentes nem sempre são compatíveis entre si e isto traria uma diversidade de pontos de vista e talvez até incompatibilidades entre as informações coletadas pelos agentes. O sistema consegue garantir a consistência dos dados apenas em nível local.

Uma memória consistente é aquela que não possui informações conflitantes do tipo $A \wedge \neg A$ (A e não A). Um agente deve acreditar em apenas uma coisa de cada vez. Ele não pode acreditar em duas coisas contraditórias ao mesmo tempo. Por outro lado, o sistema como um todo pode ter dados conflitantes, sendo necessário uma negociação quanto ao tipo de verdade que estaria sendo buscada.

A distribuição da base de dados dificulta muito a manutenção de uma coerência maior. O objetivo final do sistema é ter uma representação bem diversificada das informações, com "pontos de vista" diferentes para cada agente. Este tipo de diversidade enriquece a base de dados e permite buscar soluções de forma variada, pois torna o conhecimento do sistema dinâmico e eclético.

Em nosso sistema o tipo de código a ser utilizado será o mesmo utilizado na comunicação, ou seja, o mesmo código que um agente utiliza para armazenar os dados também será utilizado para fazer a troca de informações (ou dados) entre os agentes.

O comportamento quanto ao compartilhamento de memória em nosso sistema classifica-se como benevolente, uma vez que os agentes sempre respondem quando consultados.

4.1.2 Percepção

Cada agente tem sua própria percepção do mundo. Ele deve ser capaz de receber, armazenar e processar informações do ambiente. As formas de percepção podem variar muito de um sistema para outro. Imagine o caso de robôs que trabalham em uma fábrica. Podem existir robôs que se locomovem pelo ambiente de trabalho sendo necessário sensores de movimento e temperatura, visão, etc. Neste caso de interação com um mundo físico a percepção do ambiente é essencial. No caso de um sistema de PLN o agente deve ser capaz de "ouvir" as sentenças e perguntas lançadas no sistema. Cada agente "ouviria" aquilo que lhe fosse permitido e com isso processaria suas sentenças, consultaria outros agentes, tiraria suas conclusões e aumentaria sua base de conhecimento. A percepção é o único meio pelo qual um agente aumenta sua compreensão sobre o ambiente em que se encontra.

Em nosso sistema os agentes se comunicam através de mensagens em *broadcast* contendo tipo da mensagem, destinatário e remetente. O destinatário pode ser inclusive um grupo de agentes ou todos os agentes do sistema.

4.1.3 Capacidade de Decisão

Um ponto importante para que alguém seja independente é a capacidade de tomar suas próprias decisões. Da mesma maneira, a autonomia de um agente é possibilitada pela sua capacidade em chegar à decisões próprias.

A base para as suas decisões são as informações que ele mesmo tem armazenadas e as consultas que pode fazer a outros agentes. As inferências de cada agente serão feitas de acordo com a sua especialidade dentro do sistema. O ideal é que cada agente possua autonomia para decidir.

4.1.4 Capacidade de Negociação

Os agentes não são totalmente independentes quando consideramos que a existência deles é dirigida para um trabalho cooperativo. Para alcançar os objetivos individuais é necessária uma forte interação com outros agentes. Nem sempre as conclusões de todos os agentes são equivalentes e pode ser necessário escolher uma resposta que seja melhor que as outras; nestes casos existem políticas de negociação. Uma forma de avaliar as respostas seria fazer um sistema de pontuação para elas e considerar aquelas que possuem mais votos dos agentes. Dependendo dos casos analisados outras políticas poderiam ser utilizadas. Apesar da diversidade de opiniões, sempre será necessário que alguma resposta prevaleça

perante as outras e parte da inteligência do sistema estará na capacidade de escolha da melhor resposta.

Durante a análise de um texto podem existir conclusões contraditórias, fazendo com que algumas respostas tenham que ser descartadas ou consideradas apenas na memória local do agente. É interessante guardar mesmo as respostas não utilizadas caso tenha que se recorrer[2] a elas quando uma conclusão não satisfizer os outros agentes. Esta é a maneira utilizada para explorar os caminhos possíveis para se chegar ao objetivo final.

Note-se que a colaboração entre os agentes nem sempre é positiva, ou seja, nem sempre um agente estará colaborando com o trabalho do outro, mas pode estar indo contra as conclusões do outro agente. A diversidade de respostas irá depender da diversidade dos algoritmos escolhidos para cada agente. Podemos dizer, então, que há colaboração positiva e negativa entre os agentes. Considera-se que este é um aspecto positivo, pois contribui para que o sistema seja mais dinâmico, enfatizando a idéia de que múltiplas abordagens podem trazer boas soluções.

4.1.5 Comunicação

A comunicação nada mais é do que a capacidade de receber e emitir mensagens de e para outros agentes. Pode-se utilizar os mecanismos normais de comunicação entre processos (memória compartilhada, RPC, Sockets, etc.) ou alguma biblioteca já desenvolvida especificamente para comunicação entre agentes.

Existem linguagens e padrões para a comunicação de mais alto nível entre agentes, como a KQML (*Knowledge Query Manipulation Language*, Mayfield [24]), uma linguagem bem completa que cobre muitas das necessidades de comunicação entre agentes. Esta linguagem é suportada pela *ARPA Knowledge Sharing Effort* e possui diversas implementações em diferentes grupos de pesquisa. Existe também uma especificação sobre um padrão que utiliza lógica de primeira ordem como base para a comunicação, suportando inclusive raciocínio não-monotônico, que é a *KIF - Knowledge Interchange Formalism* [15].

Neste trabalho foi desenvolvido um protocolo próprio e específico para a abordagem multiagentes adotada neste trabalho. Existe alguma similaridade com as linguagens mencionadas, uma vez que a base do protocolo aqui desenvolvido são cláusulas do Prolog, um subconjunto da lógica de primeira ordem. A implementação de uma linguagem de

[2]Este é um comportamento característico dos algoritmos de prova baseados em árvores de prova, onde os possíveis caminhos são armazenados em forma de árvore. Nestes algoritmos é comum utilizar o *backtracking* para percorrer todos os caminhos possíveis.

comunicação entre agentes mais completa ocuparia muito tempo de implementação. Este protocolo será discutido no próximo capítulo.

Uma boa capacidade de comunicação irá contribuir e facilitar a negociação entre os agentes bem como aumentar a eficiência na troca de informações. Alcançar este objetivo não é muito simples, visto que é muito difícil capacitar um agente a "falar" com outro agente da mesma maneira que uma pessoa consegue conversar com outra. Temos limitações na própria linguagem, na capacidade de raciocínio de cada agente e na coordenação da comunicação em uma comunidade de agentes.

Quando existem muitas opiniões sendo analisadas e consultadas ao mesmo tempo pode ocorrer uma explosão de mensagens entre os agentes. Há necessidade de colocar ordem na confusão, da mesma forma que em uma assembléia ou em uma reunião com muitas pessoas participando ao mesmo tempo. Em nosso sistema temos algumas regras quanto a isso:

1. Não podem haver ciclos entre as perguntas dos agentes. Um ciclo ocorre quando um agente A pergunta uma coisa ao agente B que por sua vez pergunta a mesma coisa ao agente C e essa pergunta retorna ao agente A.
2. Perguntas e respostas repetidas devem ser ignoradas.
3. Respostas que excedem um determinado tempo para retornarem a quem consultou devem ser ignoradas.

Mesmo com essas limitações, mostraremos que é possível utilizar LPO (Lógica de Primeira Ordem) para fazer com que os dados necessários sejam passados de um agente para outro de forma inteligível.

O protocolo de comunicação não é muito complexo. A base é um "tell" e um "ask", como utilizado em Russell [33].

tell: informa um novo dado ao agente. Funciona com um aviso pedindo ao agente que incorpore a informação em sua memória. O agente alvo pode (e deve) fazer uma seleção deste tipo de informação antes de aceitá-la. Este tipo de mensagem corresponde ao tipo SENTENCE utilizado no nosso sistema e será descrito no capítulo 5.

ask: é uma consulta a outro agente qualquer, a um agente específico ou a toda comunidade. O agente deve esperar pelo menos uma resposta para a sua pergunta antes de continuar o processamento. Nunca deve tentar responder a uma nova pergunta que ele mesmo tenha feito. Conclui-se que devem existir pelo menos dois agentes

para que haja comunicação. Este tipo de mensagem corresponde ao tipo QUERY do nosso sistema e será descrito na seção 5.

O conteúdo das mensagens entre os agentes será em forma de cláusulas, utilizando-se a mesma sintaxe do Prolog. Uma mensagem poderia ser enviada da seguinte forma:

SENTENCE	comprou(X,Y):-pedro(X),undef(Y,livro(Y)))

A mensagem teria apenas um identificador para um determinado agente ou ainda um identificador geral do tipo ''ALL'' para uma mensagem em *broadcast* para todos os agentes ativos no sistema.

4.2 Características de um sistema multiagentes

Existem vantagens e desvantagens na utilização do conceito de multiagentes em relação ao processamento seqüencial, segundo Mouta [26] e Demazeau [12]:

- Vantagens:
  - coordenando diferentes algoritmos é possível atingir uma solução de maneira mais eficiente do que utilizando cada algoritmo separadamente;
  - existe maior facilidade para introduzir novos agentes no sistema, possibilitando o teste de novas abordagens sem muitas dificuldades;
  - existe um melhor aproveitamento dos recursos computacionais considerando uma rede de computadores, como ocorre em sistemas distribuídos convencionais;
  - a modularidade dos agentes facilita também a manutenibilidade visto que é possível acrescentar e retirar módulos do sistema sem afetar o sistema como um todo; e
  - pelo mesmo motivo, o sistema é também tolerante a falhas, pois o não funcionamento de alguns dos módulos implica em uma degradação apenas parcial do sistema.
- Desvantagens:
  - o projeto do sistema como um todo e seu funcionamento são difíceis de descrever, pois além de ser um sistema distribuído, é comum que o sistema cresça gradualmente e em módulos (agentes); e

- um agente separadamente possui visão incompleta do problema.
- não há uma garantia de que o trabalho conjunto dos agentes encontrará uma solução ótima, ou mesmo boa, para o problema.

4.3 Exemplo de um agente

A figura 4.4 ilustra dois agentes que utilizam a mesma linguagem para representação do conhecimento (*Knowledge Base* - KB) e um protocolo de comunicação simples baseado em "tell" e "ask".

O resultado que se espera de um sistema desses é o comportamento emergente do grupo todo. O objetivo do sistema é atingido pelo grupo e não pelos agentes individualmente. Cada agente pode receber um estímulo do sistema (**percepção**) e através da sua base de conhecimento e regras de inferência **consultar** outros agentes, alcançar o seu objetivo e **informar** sobre suas atividades a um agente ou a um grupo de agentes.

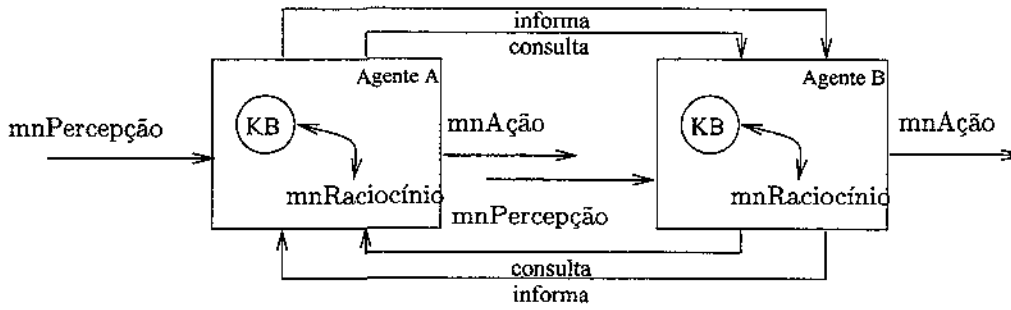


Figura 4.4: Um exemplo de dois agentes simples.

4.4 Conclusão

Este capítulo descreveu o que é um SMA e apresentou as características desejáveis para o nosso sistema, situando-o dentro do contexto da IA. Como exemplo, utilizamos a descrição do funcionamento e comportamento do SMA desenvolvido para o Processamento de Língua Natural.

Um agente foi decomposto em memória, percepção, capacidade de decisão, capacidade de negociação e comunicação. Seguiremos este modelo para desenvolver o sistema apresentado no capítulo seguinte, dirigido a resolução de referências, dentro do contexto da Lingüística Computacional.

Capítulo 5

O Uso de um Sistema Multiagentes para o Problema de Referência Definida

O objetivo deste capítulo é demonstrar como um Sistema Multiagentes pode resolver os casos de referências propostos no capítulo 2, sendo estas: co-referência direta, referência não definida, referência por associação, referência a conjuntos e, por último, um caso de referência pronominal. Para tanto, estaremos utilizando como base a codificação proposta no capítulo 3 e a definição de SMA apresentada no capítulo 4.

Duas grandes vantagens a serem exploradas no SMA são a sua multiplicidade de representações e sua modularidade. A multiplicidade de interpretações para uma sentença em Língua Natural é tratada de maneira natural pelo SMA, uma vez que toda a base de dados é distribuída em módulos independentes. O consenso entre as interpretações é obtido reunindo-se as informações e conclusões de cada agente do sistema. A modularidade permite que novas heurísticas sejam acrescentadas sem influenciar o funcionamento das heurísticas anteriores, ou seja, podemos desenvolver novos agentes conforme novos fenômenos lingüísticos são identificados.

Nesta seção apresentaremos a arquitetura do sistema descrevendo como um agente age isoladamente e como eles interagem no sistema; quais são as heurísticas de cada agente e como é feito o processamento das sentenças.

Os detalhes de implementação sobre os mecanismos de comunicação utilizados e outros recursos específicos das linguagens utilizadas são descritos no apêndice A.

5.1 Granularidade do sistema

Um dos ajustes iniciais a serem feitos é determinar o quanto cada agente deve conhecer do problema, isto é, definir a granularidade do sistema. Normalmente, um agente não tem a visão completa do problema de referência contido em um enunciado e a sua percepção alcança apenas algumas de suas características. Dizemos que quanto maior é a abrangência de visão que o agente tem do problema, menor é a granularidade do sistema e, por sua vez, quando a visão do agente é muito limitada dizemos que o granularidade é maior.

Devemos analisar primeiramente quais foram as abordagens propostas, as perspectivas de cada agente para determinar a melhor granularidade do sistema. Podemos citar algumas abordagens já utilizadas no PLN como a proposta por Paiva [10], na qual cada agente é associado a uma palavra da sentença. A proposta de Stefanini [40] separa os agentes por nível de processamento, no qual os agentes do sistema cuidam da análise morfológica, léxica, sintática, semântica e de fenômenos da língua, cada um conforme a sua especificação.

O nosso sistema propõe que a visão dos agentes seja determinada por tipos referências e problemas relacionados, ou seja, serão implementados agentes para tratar dos diferentes casos de referência. Os agentes propostos tratam dos casos de referência mencionados no início do capítulo 2, comprovando que o sistema é flexível o suficiente para tratar de diversos fenômenos lingüísticos relacionados à referência. A fase de análise léxica e sintática é feita por um dos agentes e fornece uma representação em forma de cláusulas de Horn que será utilizada por todos os outros agentes, como mostrado no capítulo 3.

5.2 O funcionamento geral do sistema

A arquitetura proposta surgiu de forma incremental, na qual os agentes que tratam os problemas relacionados à referência definida foram os primeiros elementos a serem projetados.

Um agente deve, inicialmente, responder às consultas vindas do ambiente formado por uma comunidade de agentes. Uma consulta deve produzir uma resposta, que pode ser vazia ou não. Um problema encontrado inicialmente foi o fato de que um agente pode gerar outras consultas quando recebe uma consulta do ambiente. Quando um agente faz uma consulta é necessário que este fique esperando uma resposta, mas esta espera impediria que ele respondesse às outras consultas que são feitas simultaneamente por outros agentes. A solução foi separar a funcionalidade de um agente em dois agentes mais simples: um

agente que se encarrega de receber e responder às consultas, que denominamos *agente mestre* e um agente que faz o processamento e gera outras consultas, chamado de *agente escravo*. Com esta abordagem, consegue-se atingir o paralelismo necessário para executar as duas tarefas simultaneamente.

Cada *agente mestre* possui um *agente escravo* correspondente. Estes dois agentes possuem a mesma base de conhecimento, tanto em algoritmos, como em dados obtidos durante o processamento. O *agente mestre* é responsável por controlar as requisições, administrando as respostas e passando o processamento (execução do algoritmo de busca de referentes) para o seu *agente escravo*. A função do escravo restringe-se ao processamento básico baseado na máquina de inferência do Prolog.

Existe a necessidade de uma interface para nos comunicarmos com os agentes, enviando-lhes sentenças em língua natural e coletando as respostas para uma avaliação posterior. Esta função foi dada a um agente chamado de *mestre geral*. Este *mestre geral* faz a interface com o usuário, que fornece um texto em língua natural e recebe como resposta uma lista de possíveis traduções para estas sentenças em forma de cláusulas. Este agente permite também a consulta, em forma de cláusulas, a todos os agentes simultaneamente. O *mestre geral* faz uma análise sintática das sentenças em língua natural e gera mensagens de consulta em forma de cláusulas para os agentes. Atualmente, toda a análise sintática do sistema encontra-se no *mestre geral*. Uma possível extensão seria colocar esta análise nos próprios agentes, possibilitando diversos de tipos de análise para uma mesma sentença, ou ainda, criar agentes especializados em análise sintática. O sistema não trata, atualmente, de nenhum tipo de ambigüidade sintática, como vimos no capítulo 3, existindo apenas uma árvore de derivação sintática para cada sentença.

Um outro tipo de agente criado foi o *guru*, cuja função seria fazer uma seleção das melhores respostas do sistema e ser capaz de responder às consultas do *mestre geral*. Atualmente, ele ainda não é capaz de fazer isto e apenas acumula todas as respostas do sistema, sem selecioná-las.

O sistema como um todo funciona através de intensa troca de mensagens entre os agentes. Existem consultas que aguardam respostas imediatas (bloqueadas) e consultas não bloqueadas. As consultas bloqueadas ocorrem de um agente escravo para um agente mestre no mesmo grupo e as consultas não bloqueadas ocorrem de um agente escravo para o grupo de mestres. Este último tipo de mensagem utiliza-se do mecanismo de *multicast* enviando uma mesma mensagem para um grupo de agentes. A implementação utiliza recursos nativos da biblioteca PVM para fazer isto (os detalhes de implementação e uso desta biblioteca podem ser vistos no apêndice A).

A figura 5.1 mostra uma mensagem originada no mestre geral. Esta mensagem é

enviada ao grupo de agentes. Uma vez recebida a mensagem todos os escravos começam a interpretá-la simultaneamente. Quando há necessidade de uma consulta externa[1] esta é feita primeiro ao grupo de mestres que, por sua vez, se encarrega de repassá-la aos outros agentes escravos. Quando um agente escravo faz uma consulta externa, ele fica esperando até que a resposta seja retornada. Por este motivo, não é permitido que a consulta original ou uma consulta derivada da original retorne ao mesmo agente que a fez inicialmente. O controle do fluxo de mensagens é feito pelo grupo de agentes mestres.

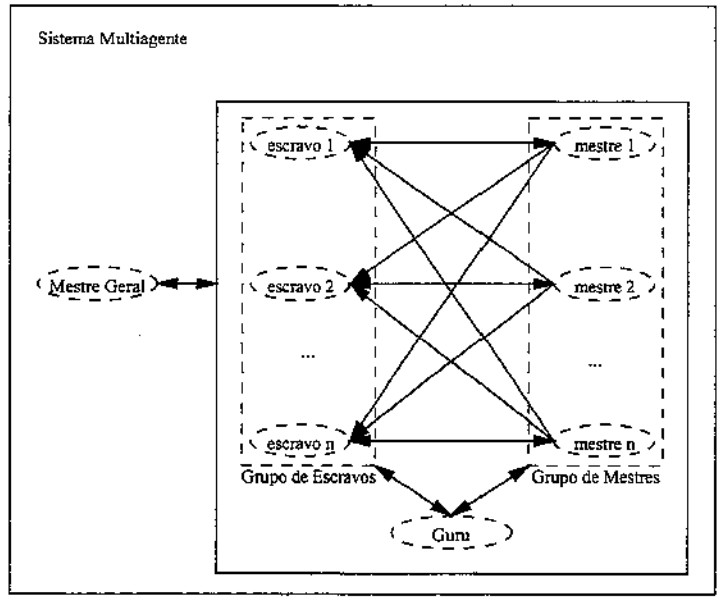


Figura 5.1: Funcionamento geral do sistema.

Note que uma mensagem enviada para o grupo de agentes pode gerar até n mensagens para o grupo de mestres, onde n é o número de agentes mestres, totalizando um máximo de $(1+n)$ mensagens. Da mesma forma, uma mensagem enviada para o grupo de mestres gera n consultas e n respostas (totalizando $(1+n(n+n))$). Cada agente seleciona uma das respostas vindas do grupo dos mestres e envia para o mestre geral (totalizando $(1+n(n+n)+n)$). Resumindo, uma sentença inicial pode gerar $(2n^2+n+1)$, ou seja, é da ordem $O(n^2)$ em função do número de heurísticas implementadas (agentes ativos no sistema). Temos então um limite computacional em termos da quantidade de agentes ativos simultaneamente.

A figura 5.2 mostra uma execução do programa em uma tela do xpvm (um visualizador utilizado para a biblioteca PVM). A execução ocorreu em três máquinas[2] utilizando como

[1]Uma consulta externa ocorre quando há necessidade de se obter uma informação que esteja fora da base de dados do agente em execução.

[2]Os nomes das máquinas são *besouro*, *joaninha* e *abelha*.

entrada as sentenças (1) e (2). Nesta figura a execução de cada agente no decorrer do tempo é representada por uma barra horizontal, identificado à esquerda com o nome da máquina e o tipo de agente (como por exemplo "`besouro:master`" e "`joaninha:slave`"). As linhas, muitas delas sobrepostas, representam as mensagens que trafegam endereçadas de um agente para outro. As mensagens em *broadcast* não são representadas no gráfico. A primeira metade da execução caracteriza-se por várias mensagens partindo do mestre para os outros agentes. Isto ocorre porque o mestre está enviando as sentenças iniciais para os outros agentes. Na segunda metade da execução existe uma interação maior entre os agentes e ao final todos enviam uma resposta para o mestre. A execução encerra quando o mestre envia uma mensagem solicitando que todos os agentes terminem[3] sua execução.

Percebe-se, através da figura 5.2, que a quantidade de mensagens entre os agentes é reduzida. A execução e a troca de mensagens ocorre de maneira assíncrona mas os próprios agentes controlam a sincronização das tarefas, não permitindo que uma nova consulta seja iniciada até que a última tenha sido totalmente processada.

(1) Pedro comprou um livro.

(2) O autor eh famoso.

O sistema funciona de maneira similar a um sistema distribuído, mas não possui um controle estritamente centralizado, aproximando-se mais da abordagem da Inteligência Artificial Descentralizada. Os agentes são autônomos e controlam suas próprias atividades através de regras pré-estabelecidas, direcionando suas tarefas de acordo com as necessidades e requisições do ambiente. Pode-se dizer que o sistema possui as características básicas de um sistema multiagentes.

5.3 A comunicação entre os agentes

A comunicação entre os agentes é feita através de mensagens que possuem um *tipo* e um *conteúdo*. O *conteúdo* de uma mensagem é sempre uma seqüência de caracteres que pode conter uma proposição ou um argumento para um comando a ser executado. No caso geral, os agentes trocam entre si proposições em forma de cláusulas. Os tipos de mensagens utilizadas pelos agentes estão descritos na tabela 5.1 e alguns exemplos são mostrados na tabela 5.2.

[3]Corresponde à mensagem KILL do protocolo de comunicação dos agentes.

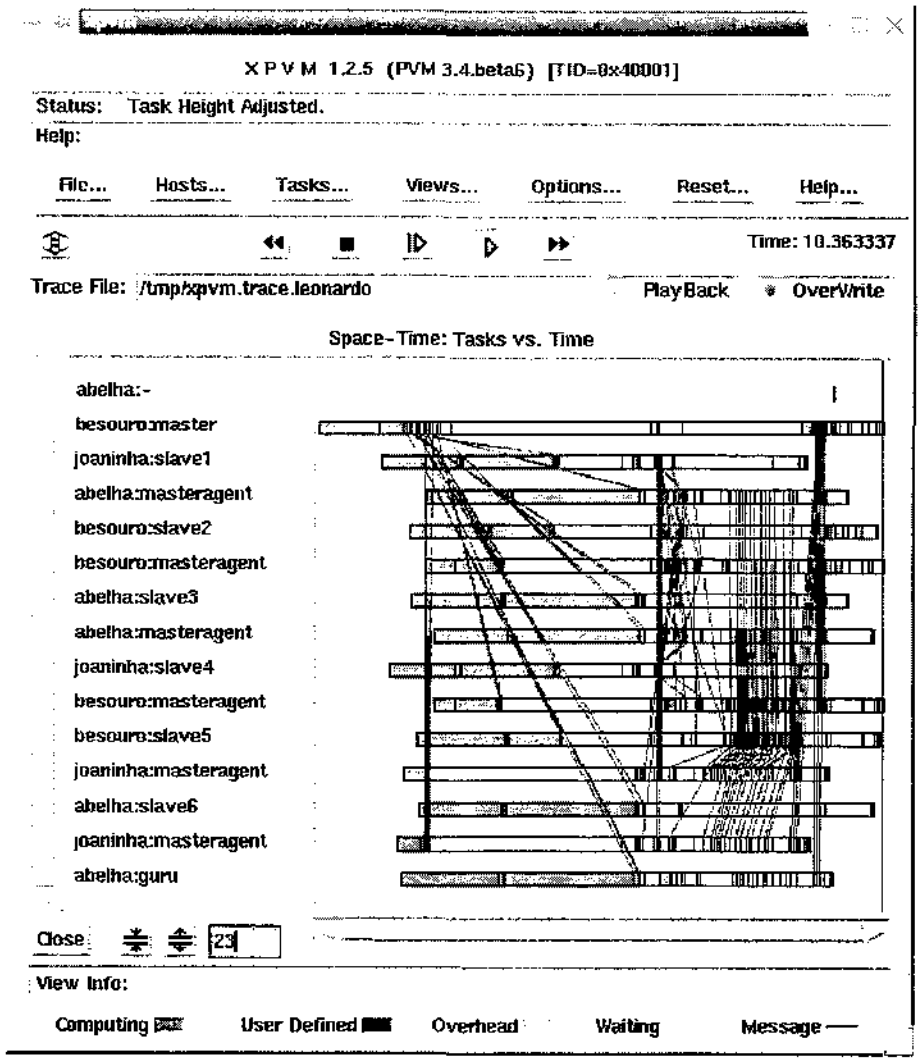


Figura 5.2: Visualização gráfica de uma execução do sistema.

5.4 Funcionamento de um agente

Os agentes (mestres e escravos) utilizam-se dos mesmos mecanismos de comunicação mas alguns estão aptos a responder mensagens que outros não estão. O seu comportamento depende das heurísticas que implementa e de seu tipo dentro do sistema.

Um *escravo* aguarda mensagens do tipo SENTENCE enviadas pelo *mestre geral* e atualiza as informações dos agentes mestres. Os *mestres* respondem consultas do tipo ASK vindas do *escravos* e armazenam informações que chegam a eles através de mensagens do tipo ASSERT. A figura 5.3 mostra a comunicação entre estes tipos de agentes.

Tipo	Descrição
SENTENCE	pede ao agente ou grupo de agentes que processe o conteúdo da mensagem.
QUERY	faz uma consulta a um determinado agente ou grupo de agentes.
ASK	pede ao agente que responda a uma consulta do usuário.
ANSWER	informa a resposta a uma consulta do tipo QUERY.
ASSERT	pede ao agente que atualize sua base de dados com o conteúdo da mensagem.
GROUP	pede ao agente que se cadastre em um grupo.
BARRIER	pede ao agente que espere até que os outros agentes cheguem ao mesmo ponto.
KILL	pede ao agente para abandonar o sistema (encerrar sua execução).

Tabela 5.1: Tipos de Mensagens

Tipo	Conteúdo	Significado
SENTENCE	corre(X):-pedro(X)	processe "Pedro corre".
QUERY	def(X,autor(X))	pergunta: "Existe um autor definido?"
ASK	corre(X)	Quem corre?
ANSWER	corre('Pedro')	'Pedro' corre.
ASSERT	def(a,autor(a))	adicione o autor "a" ao sistema
GROUP	grupo1	cadastre-se no grupo1
BARRIER		aguarde todos chegarem a este ponto
KILL		abandone o sistema

Tabela 5.2: Exemplos de mensagens entre os agentes.

Todas as sentenças são geradas a partir do mestre geral e todas as informações são replicadas para o guru.

5.5 A arquitetura do Sistema Multiagentes

O grupo dos *escravos* recebe as consultas iniciais (mensagens do tipo SENTENCE) e processa as sentenças para resolver as referências. Durante o processo os *escravos* podem fazer consultas aos agentes do grupo dos *mestres*. Os *mestres* acumulam as informações geradas pelos agentes de acordo com seu subgrupo, pois cada agente está vinculado a um grupo numerado de 1 a n, onde n é o número de fenômenos lingüísticos tratados no

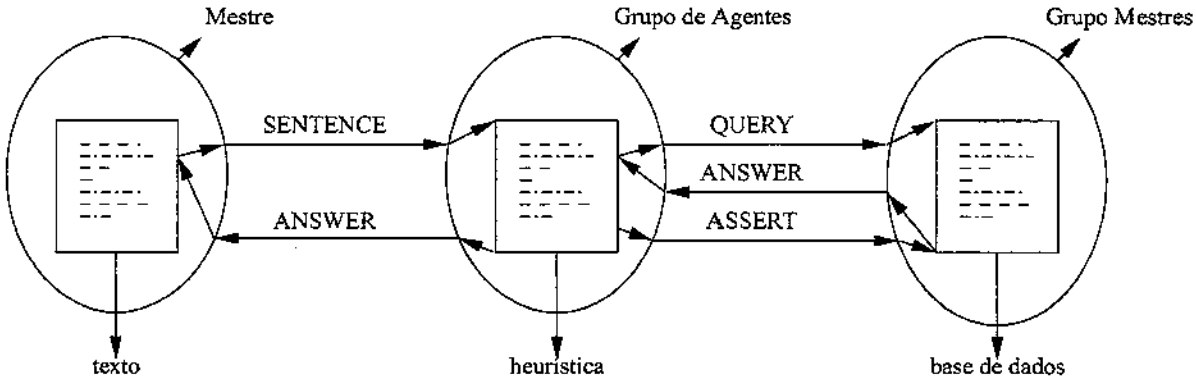


Figura 5.3: Um agente dentro do sistema.

sistema. Quem responde às consultas do sistema e faz a atualização das bases de dados são os mestres. Os agentes (na posição de escravos) apenas executam o processamento. A separação das tarefas evita que sejam feitas consultas de um *escravo* para ele mesmo e permite um processamento paralelo. Desta forma, possibilita-se que *escravos* de outros subgrupos também façam consultas aos *mestres* de qualquer grupo.

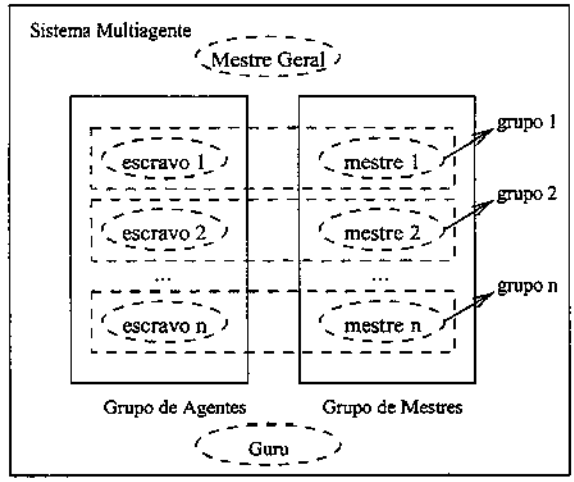


Figura 5.4: Arquitetura do sistema.

Cada agente possui um algoritmo próprio para tratar as referências e os mestres são todos iguais, com exceção do "mestre" que faz as consultas iniciais com as sentenças a serem processadas. Analisando cada subgrupo individualmente podemos ver que eles se comportam como se fossem um agente com suas tarefas separadas: processar sentenças e manter a base de dados.

5.6 As heurísticas de cada agente

Colocando em termos não somente lingüísticos mas também computacionais, os agentes que compõem o sistema estão divididos de acordo com a sua funcionalidade e de acordo com as heurísticas que implementam.

As heurísticas dos agentes procuram imitar o comportamento humano na representação e compreensão das referências. Queremos fazer com que o sistema satisfaça os requisitos esperados na compreensão de um texto. As heurísticas são responsáveis por caracterizar o perfil "inteligente" dos agentes. Utilizamos os conceitos propostos no capítulo 2 para fazer com que os agentes sejam capazes de compreender os fenômenos de referência.

As subseções seguintes mostram como funcionam as heurísticas dos agentes.

5.6.1 Heurística do agente 1

A heurística do agente 1 trata do caso de co-referência direta. Esta heurística faz a unificação entre o referente de um sintagma nominal definido com o último sintagma nominal não-definido mencionado anteriormente que combine com o atual. Este agente procura por sintagmas que possuem o mesmo nome, seja ele definido ou indefinido. Considere o exemplo:

(3) Ana comprou um vaso. O vaso quebrou.

Este exemplo é transformado na seguinte representação em forma de proposições pelo *mestre geral*:

```
(3.1) comprou(X,Y):-ana(X),undef(Y,vaso(Y)).
(3.2) quebrou(X):-def(X,vaso(X)).
```

No exemplo acima a referência ao "o vaso" é resolvida unificando-se os núcleos dos dois sintagmas envolvendo o vaso. A unificação é feita consultando todos os agentes envolvidos com uma mensagem do tipo QUERY contendo a seguinte proposição:

```
def(X,vaso(X)).
```

onde X é uma variável livre representando o núcleo do sintagma. Quando nenhum agente responde, uma segunda consulta idêntica é feita buscando por um sintagma indefinido:

```
undef(X,vaso(X)).
```

A resposta, se houver uma, terá o formato da proposição "`def(aa40002,vaso(aa40002))`", onde aa40002 representa um índice para o vaso citado. Este índice é único em todo o sistema. Quando uma entidade passa a existir ela deve ser única e caracterizada pelas suas propriedades. O vaso que quebrou deve ser o mesmo que Ana comprou na sentença anterior. Nenhum outro vaso deve satisfazer a consulta.

Quando existe uma resposta a sentença é codificada em duas proposições:

```
quebrou(a):-def(a,vaso(a)).
def(a,vaso(a)).
```

Quando não se tem resposta este agente conclui que nenhum vaso foi citado anteriormente e, portanto, esta é a sua primeira ocorrência no discurso. O agente cria uma nova entidade, atribuindo um novo índice a ela. A sentença é similar a anterior, com o detalhe de receber um índice pertencente a este agente.

O agente escravo envia as proposições processadas ao seu respectivo mestre e permanece aguardando novas sentenças. O mestre correspondente está livre para responder consultas mesmo quando o escravo correspondente está processando.

5.6.2 Heurística do agente 2

Este agente trata das referências não-definidas (sintagmas nominais introduzidos por artigo indefinido), instanciando novas entidades. Este agente considera que um sintagma nominal indefinido, quando utilizado no texto, se refere sempre a uma nova entidade. Seria uma das funções do artigo indefinido mencionada no capítulo 2. A instanciação é feita atribuindo-se uma nova constante à variável livre do sintagma.

Utilizando o mesmo exemplo do agente 1, as proposições codificadas são:

```
comprou(X,a):-ana(X),undef(a,vaso(a)).
undef(a,vaso(a)).
```

As proposições são enviadas para o mestre acrescentar à sua base de conhecimento. Uma cópia é enviada para o guru e uma instanciação das proposições é feita no próprio agente para evitar futuras consultas ao mestre.

5.6.3 Heurística do agente 3

Este agente completa a codificação de sujeitos compostos. Quando a análise sintática identifica um sujeito composto, os agentes o representam através de um conjunto de entidades que compõem o sujeito.

O exemplo abaixo mostra a codificação da sentença:

(4) Pedro parecia feliz. Ele e Ana saíram para passear.

A codificação do exemplo acima é mostrada abaixo:

```
feliz(X):-pedro(X).
sairam([X,Y]) :- ele(X), ana(Y), undef([X,Y], conjunto([X,Y])).
passear([X,Y]) :-ele (X), ana(Y), undef([X,Y], conjunto([X,Y])).
undef([X,Y], conjunto([X,Y])) :- ele(X), ana(Y).
```

Esta codificação não é completada na fase de análise sintática porque podem ocorrer casos em que o sujeito, além de ser composto, não é determinado dentro da mesma sentença, recorrendo-se a pronomes pessoais que remetem a outras entidades. Um agente poderia ser implementado para resolver o pronome "ele" buscando a ocorrência mais próxima de um substantivo masculino no texto.

5.6.4 Heurística do agente 4

Este agente constrói relações entre o referente e a referência. Utiliza relações de pertinência, como faz-parte-de, é-membro-de, possui-um, e outras que podem ser descritas explicitamente.

Considere o exemplo:

(5) Ana e Pedro saíram para passear. O casal parecia feliz.

Existe uma definição global dizendo que o casal é um conjunto formado por duas entidades, normalmente uma entidade masculina e uma entidade feminina. O agente utiliza esta informação para criar uma relação entre "Ana e Pedro" e "o casal". A codificação das sentenças é a seguinte:

```
(5.1) sairam([X,Y]):-ana(X),pedro(Y),undef([X,Y], conjunto([X,Y])).
```

```
(5.2) passear([X,Y]):-ana(X),pedro(Y),undef([X,Y], conjunto([X,Y])).
(5.3) undef([X,Y], conjunto([X,Y])):-ana(X),pedro(Y).
(5.4) feliz(X):-def(X,casal(X)).
```

O mesmo conjunto acima se transforma após o processamento pelo sistema nas seguintes proposições:

```
(a) sairam([G9,G11]):-
      ana(G9),pedro(G11),undef([G9,G11], conjunto([G9,G11])).
(b) passear([G56,G58]):-
      ana(G56),pedro(G58),undef([G56,G58], conjunto([G56,G58])).
(c) undef([G112,G114], conjunto([G112,G114])):-ana(G112),pedro(G114).
(d) feliz(G151):-def(G151,casal(G151)).
(e) feliz(['Ana','Pedro']):-def(['Ana','Pedro'],casal(['Ana','Pedro'])).
```

As proposições de (a) a (d) continuam representando a tradução original, mas a proposição (e) faz a ligação com os referentes 'Ana' e 'Pedro' da sentença anterior. Para recuperar estes referentes o sistema utiliza os conhecimentos globais adicionados inicialmente a todos os agentes do sistema. O fragmento utilizado é citado abaixo:

```
sp('Ana',singular,feminino).
sp('Pedro',singular,masculino).

def([X,Y],casal([X,Y])) :-
   sp(X,singular,masculino),
   sp(Y,singular,feminino);
   sp(X,singular,feminino),
   sp(Y,singular,masculino).

casal([X,Y]) :-
   sp(X,singular,masculino),
   sp(Y,singular,feminino);
   sp(X,singular,feminino),
   sp(Y,singular,masculino).

conjunto(X) :- casal(X).
```

Este trecho de código em Prolog garante que um casal é formado por dois substantivos próprios (sp), singulares, um do gênero masculino e outro do gênero feminino, sendo eles

definidos com o def ou não. Com base nestas informações o sistema conclui que o casal que parecia feliz é definido por Ana e Pedro, mencionados na sentença imediatamente anterior. Acrescenta ainda que um casal pode ser visto como um conjunto.

O agente 3 aplicou regras de inferência na proposição (d) para obter a proposição (e), de forma a apresentar os referentes encontrados. A inferência baseia-se na idéia de que um conjunto citado recentemente deveria satisfazer a condição de casal da proposição (d). Como o conjunto das proposições (a) e (b) possuem como elementos dois substantivos próprios que satifazem a condição para formar um casal, o agente deduz que estes elementos são os referentes da proposição (d) e acrescenta a proposição (e) para confirmar isto.

5.6.5 Heurística do agente 5

Este agente faz o relacionamento entre sintagmas que não têm relação aparente utilizando a relação de proximidade entre os sintagmas. Vamos considerar o exemplo (21.3), utilizado na seção 2, repetido aqui por conveniência:

(6) Ana pegou as coisas do piquenique. A cerveja estava quente.

Vemos que existe uma referência à cerveja, mas não há uma relação explícita entre esta bebida e "as coisas do piquenique". Este agente adiciona uma informação dizendo que o último sintagma mencionado é o antecedente do sintagma "a cerveja". Esta relação de antecedência será somada às cláusulas geradas por outros agentes e pode ser utilizada, futuramente, por um agente que seja capaz de utilizar esta informação para uma seleção do melhor candidato a referente.

A saída do processamento deste exemplo é mostrada abaixo:

```
(6.1) pegou(G1,G2):-ana(G1),def(G2,coisas(G2)),
          def(G11,piquenique(G11)),do(G2,G11).
(6.2) def(aa40014,coisas(aa40014)).
(6.3) def(ab40014,piquenique(ab40014)).
(6.4) do(aa40014,ab40014):-def(aa40014,coisas(aa40014)),
          def(ab40014,piquenique(ab40014)).
(6.5) quente(G68):-def(G68,cerveja(G68)).
(6.6) antecedente(G106,ab40014):-def(G106,cerveja(G106)),
          def(ab40014,piquenique(ab40014)).
(6.7) def(ac40014,cerveja(ac40014)).
```

No exemplo (6) vemos que a proposição (6.6) marca a possível relação entre a cerveja e o piquenique. Este relação poderá ser utilizada pelo mesmo agente, ou por um outro, para identificar a cerveja como um componente das coisas do piquenique ou apenas para identificar a relação de ordem entre os sintagmas sendo processados.

5.6.6 Heurística do agente 6

Trata de referência pronominal para o pronome "eles". Este agente mostra que o sistema pode ser utilizado para tratar de outros fenômenos lingüísticos que não estão relacionados apenas a sintagmas definidos e não definidos. O agente busca a ocorrência de entidades que formam conjuntos para unificar com as entidades referenciadas por eles. Quando não encontra nenhum conjunto anterior instancia uma nova entidade para representar os referentes não encontrados. A partir deste agente seria possível tratar também outros casos de referência pronominal.

O exemplo abaixo trata de duas sentenças, sendo que a segunda utiliza uma referência pronominal:

(7) Ana e Pedro saíram para passear.

(8) Eles pareciam felizes.

Após a tradução inicial as sentenças tornam-se:

```
(7.1) sairam([G3,G0]):-ana(G0),pedro(G3),undef([G3,G0], conjunto([G3,G0])).
(7.2) passear([G3,G0]):-ana(G0),pedro(G3),undef([G3,G0], conjunto([G3,G0]))
(7.3) undef([G3,G0], conjunto([G3,G0])):-ana(G0),pedro(G3).
(8.1) felizes(G38):-eles(G38).
```

Com a colaboração do agente 4, o agente 6 consegue unificar os seguintes predicados:

```
felizes(['Pedro','Ana']).
eles(['Pedro','Ana']).
```

Podemos dizer que os agentes conseguiram fazer a ligação do pronome pessoal "eles" como os núcleos do sujeito da primeira sentença: "Ana" e "Pedro". Outras referências similares poderiam ser tratadas da mesma maneira, considerando-se outros pronomes.

5.7 Conclusão

Esta seção descreveu o funcionamento do sistema de um ponto de vista geral e do ponto de vista específico de cada agente.

Vemos que cada agente isoladamente não tem conhecimento suficiente para processar e representar um pequeno texto com suas referências, mas que o sistema como um todo fornece diferentes tipos de informações que possibilitam relacionar todas as entidades referenciadas.

Foi possível observar que a cooperação e coordenação das heurísticas de cada agente possibilita resultados interessantes do ponto de vista "multiagentes". Seria muito difícil implementar todas as heurísticas de maneira seqüencial considerando um único programa e obter resultados semelhantes.

Capítulo 6

Conclusão

Este trabalho teve como objetivo a utilização de um Sistema Multiagentes para a resolução de Referência Definida em Língua Natural. O nosso primeiro passo foi delimitar o problema de referência definida tendo como ponto de partida autores como Krámský [32], Searle [36], Hirst [18], Allen [1] e Sidner [39].

O desenvolvimento de trabalhos utilizando multiagentes para o processamento de língua natural ainda se encontra em uma fase experimental, com poucos resultados aplicados. Esta tese vem contribuir como um experimento, mostrando que é possível utilizar o paradigma multiagentes como uma abordagem viável. Neste experimento podemos verificar que, uma vez tendo a arquitetura e os mecanismo de coordenação e comunicação definidos, é simples acrescentar uma nova heurística ao sistema. O sistema pode, desta forma, crescer gradualmente, conforme novas heurísticas sejam desenvolvidas.

Neste trabalho, as maiores dificuldades foram a definição da arquitetura, a representação do conhecimento dos agentes, a forma de racicínio simbólico, coordenação dos agentes e seleção de respostas. Para se desenvolver um trabalho semelhante é importante observar estes itens. Descrevemos, a seguir, linhas gerais para o desenvolvimento de um sistema multiagentes similar ao que foi desenvolvido.

Na definição da arquitetura é necessário observar se o modelo é centralizado ou descentralizado. Teoricamente, quanto mais descentralizado o sistema, mais próximo ele está da paradigma multiagentes. A dificuldade no modelo descentralizado é definir os mecanismos de controle de cada agente para se alcançar uma coordenação cooperativa.

Considerando um modelo cognitivo, é necessário apresentar o conhecimento dos agentes em forma simbólica; escolher uma linguagem de comunicação e um modelo de representação interna do conhecimento de mundo (memória). Em um modelo reativo seria

necessário especificar regras, através de uma linguagem de regras. Estas regras seriam ativadas de acordo com o estado do ambiente.

A coordenação dos agentes entre si pode ser feita através de um coordenador, ou árbitro, ou ainda, através de regras colocadas em cada agente, especificando claramente qual o comportamento do agente quanto à recepção, tratamento e envio de respostas.

Após todo os sitema estar montado é necessário criar um mecanismo de extração das informação geradas pelo sistema.

Apresentamos uma classificação para o fenômeno da referência definida e propusemos um tratamento computacional para alguns tipos de referências. A classificação inicial nos permitiu distribuir a resolução de referência em partes menores, ou seja, a partir de um problema aparentemente monolítico partimos para uma solução distribuída. A partir desta classificação, apresentada no capítulo 2, o nosso sistema tratou dos seguintes fenômenos lingüísticos:

- co-referência direta;
- co-referência por associação;
- referência indefinida.

A utilização da tecnologia multiagentes trouxe duas grandes vantagens para o tratamento do fenômeno de referência: a multiplicidade de representações e a modularidade. A multiplicidade de representações permitiu uma abordagem distribuída na qual as características de cada agente podem ser somadas de acordo com novas heurísticas para o tratamento do problema. Nesta mesma linha de raciocínio, podemos dizer que novos problemas podem ser tratados com a inclusão de novos agentes, reaproveitando todo o sistema atual, graças a sua modularidade.

A implementação dos agentes atendeu às expectativas iniciais, pois eles são capazes de acumular informações durante a troca de mensagens, tanto em consultas quanto em pedidos de processamento. São capazes, também, de fazer novas consultas quando necessário e alterar suas decisões à medida que outras informações, além das iniciais, são adquiridas.

A distribuição do problema requereu a criação de um protocolo de comunicação entre os agentes utilizando cláusulas de Horn. Foi então proposto um protocolo específico para o processamento de língua natural utilizando um sistema multiagentes. Este protocolo demonstrou-se suficiente para o tratamento de referências, sendo possível aplicá-lo ainda a outros tipos de problemas tais como referências pronominais.

O uso de PVM permitiu uma fácil distribuição do sistema e uma ótima portabilidade, permitindo que o sistema execute em praticamente qualquer plataforma que implemente o PVM (a maioria dos sistemas baseados em Unix). Nenhuma parte do programa possui código proprietário, ou seja, todo o código utilizado baseia-se em bibliotecas de livre distribuição.

Como trabalhos futuros podemos citar a extensão do sistema para tratar de outros problemas de referências, tais como referência pronominal e referência temporal, compreensão de textos maiores, implementação do tratamento de outros fenômenos lingüísticos e implementação de uma interface que permita consulta em língua natural das sentenças já processadas.

Apêndice A

Descrição do Sistema Multiagentes

Esta seção descreve a implementação do SMA com o objetivo de possibilitar uma continuidade do trabalho reaproveitando o sistema já implementado.

O sistema foi desenvolvido em Linux 2.0.34, utilizando PVM[1] e SWI-Prolog[2]. Foi testado em Linux e em Solaris[3].

Para executar o sistema é necessário que uma máquina PVM (Parallel Virtual Machine) esteja ativa no sistema.

O agente inicial do sistema espera como entrada um arquivo texto com sentenças em Língua Natural (limitadas ao fragmento gramatical compreendido pelo sistema) e pode ser utilizado para consulta ao agentes do sistema. Outra forma de consultar os resultados é utilizando as cláusulas que são geradas pelo sistema no arquivo clauses.pl e as cláusulas de conhecimento global no arquivo global.pl, através do utilitário "consulta".

Os programas que integram o sistema após a compilação são:

master : é o agente inicial que controla todos os outros agentes.

slave$_1$,...,**slave**$_n$: são os agentes que respondem às consultas.

master$_1$,...,**master**$_n$: são os agentes que controlam as consultas e acumulam os dados processados.

[1]A versão utilizada neste trabalho é a PVM 3.3, que pode ser obtida através de ftp anônimo no endereço: ftp://netlib2.CS.UTK.EDU/pvm3/pvm3.3.11.tgz.

[2]Esta distribuição do Prolog é mantida por Jan Wielemaker, e-mail jan@swi.psy.uva.nl, na University of Amsterdam.

[3]Solaris é uma marca registrada da Sun.

A execução do sistema inicia-se pelo comando master:

```
master [-f <filename>] [-q <filename>] [-?]
 -f input com as sentencas
 -q input com as consultas
 -i consulta interativa
 -? este help
```

As sentenças estão em língua natural e as consultas devem ser feitas em forma de cláusulas. No exemplo abaixo, as cláusulas entradas pelo usuário iniciam com a palavra "Consulta:".

```
pinky$ master -f testes/teste3.txt -i
Arquivo de entrada:
  Ana e Pedro sairam para passear.
  o casal parecia feliz.

Welcome to SWI-Prolog (Version 3.1.2)
Copyright (c) 1993-1998 University of Amsterdam.  All rights reserved.

For help, use ?- help(Topic). or ?- apropos(Word).

Sentence: sairam([G3,G0]):-ana(G0),pedro(G3),undef([G3,G0],conjunto([G3,G0]))
Sentence: passear([G3,G0]):-ana(G0),pedro(G3),undef([G3,G0],conjunto([G3,G0]))
Sentence: undef([G3,G0],conjunto([G3,G0])):-ana(G0),pedro(G3)
Sentence: feliz(G38):-def(G38,casal(G38))
Iniciando consulta ...
sairam(X),passear(X)
Consulta: sairam(X),passear(X)
  guru: ------------------------
ANSWER sairam(['Pedro','Ana']),passear(['Pedro','Ana'])
   agente1: ------------------------
ANSWER
   agente2: ------------------------
ANSWER
   agente3: ------------------------
ANSWER
   agente4: ------------------------
```

```
ANSWER
   agente5: ------------------------
ANSWER
   agente6: ------------------------
ANSWER
def(X,casal(X))
Consulta: def(X,casal(X))
  guru: ------------------------
ANSWER def(['Pedro','Ana'],casal(['Pedro','Ana']))
   agente1: ------------------------
ANSWER
   agente2: ------------------------
ANSWER
   agente3: ------------------------
ANSWER def(['Pedro','Ana'],casal(['Pedro','Ana']))
   agente4: ------------------------
ANSWER def(['Pedro','Ana'],casal(['Pedro','Ana']))
   agente5: ------------------------
ANSWER def(['Ana','Pedro'],casal(['Ana','Pedro']))
   agente6: ------------------------
ANSWER
shutting down ...
pinky$
```

As cláusulas geradas por cada agente são gravadas em arquivos separados. Uma listagem destas cláusulas pode ser vista abaixo:

Agente 1 :

```
sairam([G9, G11]):-
        ana(G9), pedro(G11), undef([G9, G11], conjunto([G9, G11])).
passear([G56, G58]):-
        ana(G56), pedro(G58), undef([G56, G58], conjunto([G56, G58])).
undef([G112, G114], conjunto([G112, G114])):-
        ana(G112), pedro(G114).
feliz([Pedro, Ana]):-
        def([Pedro, Ana], casal([Pedro, Ana])).
def([Pedro, Ana], casal([Pedro, Ana])).
```

Agente 2 :

```
sairam([G9, G11]):-
        ana(G9), pedro(G11), undef([G9, G11], conjunto([G9, G11])).
passear([G56, G58]):-
        ana(G56), pedro(G58), undef([G56, G58], conjunto([G56, G58])).
undef([G135, G137], conjunto([G135, G137])):-
        ana(G135), pedro(G137).
feliz(G151):-
        def(G151, casal(G151)).
```

Agente 3 :

```
sairam([G9, G11]):-
        ana(G9), pedro(G11), undef([G9, G11], conjunto([G9, G11])).
passear([G56, G58]):-
        ana(G56), pedro(G58), undef([G56, G58], conjunto([G56, G58])).
undef([G112, G114], conjunto([G112, G114])):-
        ana(G112), pedro(G114).
feliz([Ana, Pedro]):-
        def([Ana, Pedro], casal([Ana, Pedro])).
def([Ana, Pedro], casal([Ana, Pedro])).
```

Agente 4 :

```
sairam([G9, G11]):-
        ana(G9), pedro(G11), undef([G9, G11], conjunto([G9, G11])).
passear([G56, G58]):-
        ana(G56), pedro(G58), undef([G56, G58], conjunto([G56, G58])).
undef([G112, G114], conjunto([G112, G114])):-
        ana(G112), pedro(G114).
feliz(G128):-def(G128, casal(G128)).
```

Agente 5 :

```
feliz([Pedro, Ana]):-
        def([Pedro, Ana], casal([Pedro, Ana])).
def([Pedro, Ana], casal([Pedro, Ana])).
```

Agente 6 :

```
sairam([G9, G11]):-
        ana(G9), pedro(G11), undef([G9, G11], conjunto([G9, G11])).
passear([G56, G58]):-
        ana(G56), pedro(G58), undef([G56, G58], conjunto([G56, G58])).
undef([G112, G114], conjunto([G112, G114])):-
        ana(G112), pedro(G114).
feliz(G128):-def(G128, casal(G128)).
```

No restante desta seção apresentamos a documentação gerada a partir do código fonte através do utilitário DOC++[4]

[4]Pode ser encontrado em: http://www.zib.de/Visual/software/doc++/index.html.

Conteúdo

1 **rrd1.pl — Algoritmo do agente 1** 67
2 **rrd2.pl — Algoritmo do agente 2** 70
3 **rrd3.pl — Algoritmo do agente 3** 72
4 **rrd4.pl — Algoritmo do agente 4** 76
5 **rrd5.pl — Algoritmo do agente 5** 79
6 **rrd6.pl — Algoritmo do agente 6** 83
7 **Agent — Classe** *Agent* 86
8 **Message — Classe** *Message* 92
9 **MasterAgent — Classe** *MasterAgent* 96
10 **Guru — Classe** *guru* 99
11 **PrologAgent — Classe** *PrologAgent* 102
12 **Prolog — Classe** *Prolog* 105
13 **Parser — Classe** *Parser* 110
14 **result_s — Estrutura utilizada pelo** *Lex* 114
15 **Anasint — Classe** *Anasint* 116
16 **Symbol — Classe** *Symbol* 118
17 **SymbolTable — *Classe* *SymbolTable*** 123
Class Graph 125

1

rrd1.pl

Algoritmo do agente 1

Algoritmo do agente 1. Este algoritmo faz uma busca pela ocorrência de antecedentes indicados por sintagmas nominais com a mesma forma que a referência utilizada no texto. Exemplo: "Ana comprou um vaso. O vaso quebrou."

O predicado "`processSent(Clause,P)`" recebe cláusulas do tipo "`famoso(X):-autor(X)`" e inicia o processo de busca por referentes. Se encontrar, unifica a cláusula passada como parâmetro P com uma cláusula do tipo "`famoso(aa40001) :- autor(aa40001)`", identificando um referente único com a constante "aa40001".

```
% rrd1.pl
% busca de antecedentes identicos aa referencia

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente1: '), write(P), nl.

% inicia o processo de resolucao de referencias
processProposition(Clause,NewClause) :-
  lookForDef(Clause,NewClause).

lookForDef((HeadA:-F1a),(HeadA:-F1b)) :-
  lookForDef(F1a,F1b).

lookForDef((HeadA:-F1a,F2a),(HeadA:-F1b,F2b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% conjunto de quatro clausulas
lookForDef((F1a,F2a,F3a,F4a),(F1b,F2b,F3b,F4b)) :-
```

```
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b),
  lookForDef(F4a,F4b).

% conjunto de tres clausulas
lookForDef((F1a,F2a,F3a),(F1b,F2b,F3b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b).

% conjunto de duas clausulas
lookForDef((F1a,F2a),(F1b,F2b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% procura por def's aninhados
lookForDef(def(X,def(Y,F)),(G,H)) :-
  lookForDef(def(X,F),G),
  lookForDef(def(Y,F),H).

% ignora se existir uma instanciacao
lookForDef(def(X,F),def(X,F)) :-
  clause(def(X,F),_), !,
  def(X,F).

% procura referencia a um def (o objeto)
lookForDef(def(X,F),def(X,F)) :-
  externalQuery(def(X,F)), !,
  asserta(def(X,F)).

% procura referencia a um undef (um objeto)
lookForDef(def(X,F),def(X,F)) :-
  externalQuery(undef(X,F)), !,
  asserta(def(X,F)),
  externalAssert(def(X,F)),
  write('Agente1 asserta: '), write(def(X,F)), nl.
```

```
% cria um novo objeto quando nao encontra  nenhuma referencia anterior
lookForDef(def(X,F),def(X,F)) :-
  newReferent(X),
  asserta(def(X,F)),
  externalAssert(def(X,F)).

% tentativa de responder sempre corretamente
lookForDef(X,X) :- !.
```

2

rrd2.pl

Algoritmo do agente 2

Algoritmo do agente 2. Faz o tratamento das referências não definidas de maneira similar ao agente 4.

Busca referências do tipo "undef", ou seja, para sentenças do tipo *"Pedro comprou um vaso. O vaso quebrou."* a primeira ocorrência do "um vaso" deve ser tratada. A regra abaixo recebe uma cláusula do tipo "`comprou(X,Y) :- pedro(X), undef(Y,vaso(Y))`". Se nenhuma entidade anterior satisfizer a referência uma nova entidade será criada para representar o vaso.

```
% rrd2.pl
% Trata referencias nao definidas

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente2: '), write(P), nl.

% inicia o processo de resolucao de referencias
processProposition(Clause,NewClause) :-
  assertUndef(Clause,NewClause).

assertUndef((HeadA:-F1a,F2a),(HeadA:-F1b,F2b)) :-
  assertUndef(F1a,F1b),
  assertUndef(F2a,F2b).

assertUndef((F1a,F2a),(F1b,F2b)) :-
  assertUndef(F1a,F1b),
  assertUndef(F2a,F2b).

% procura por undef's aninhados
```

```
assertUndef(undef(X,undef(Y,F)),(G,H)) :-
  newReferent(X),
  assertUndef(undef(X,F),G),
  assertUndef(undef(Y,F),H).

% procura internamente algum undef anterior
assertUndef(undef(X,F),undef(X,F)) :-
  clause(undef(X,F),_), !,
  undef(X,F).

% procura algum undef anterior internamente
assertUndef(undef(X,F),undef(X,F)) :-
  externalQuery(undef(X,F)), !,
  asserta(undef(X,F)).

% instancia o objeto quando nao encontra nenhuma
% relacao anterior.
assertUndef(undef(X,F),undef(X,F)) :-
  newReferent(X),
  asserta(undef(X,F)),
  externalAssert(undef(X,F)).

assertUndef(X,X) :- !.
```

3

rrd3.pl

Algoritmo do agente 3

Algoritmo do agente 3. Responsável pela representação de sujeitos compostos. Quando encontra uma referência a um sujeito composto procura por alguma referência anterior para fazer a relação ou unificar os núcleos de tais sujeitos. Caso não encontre um conjunto anterior que satisfaça a referência instancia um novo conjunto com os núcleos que satisfaçam a referência.

Exemplo:

Ana e Pedro sairam para passear =>

```
saiu(X) :- ana('Ana'), pedro('Pedro'), conjunto(X,['Ana','Pedro']).
```

```
% rrd3.pl
% Trata sujeitos compostos.

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente3: '), write(P), nl.

% inicia o processo de resolucao de referencias
processProposition(Clause,NewClause) :-
  lookForConjunto(Clause,NewClause).

lookForConjunto((HeadA:-F1a),(HeadA:-F1b)) :-
  lookForConjunto(F1a,F1b).

lookForConjunto((HeadA:-F1a,F2a),(HeadA:-F1b,F2b)) :-
  lookForConjunto(F1a,F1b),
  lookForConjunto(F2a,F2b).

% conjunto de quatro clausulas
```

```
lookForConjunto((F1a,F2a,F3a,F4a),(F1b,F2b,F3b,F4b)) :-
  lookForConjunto(F1a,F1b),
  lookForConjunto(F2a,F2b),
  lookForConjunto(F3a,F3b),
  lookForConjunto(F4a,F4b).

% conjunto de tres clausulas
lookForConjunto((F1a,F2a,F3a),(F1b,F2b,F3b)) :-
  lookForConjunto(F1a,F1b),
  lookForConjunto(F2a,F2b),
  lookForConjunto(F3a,F3b).

% conjunto de duas clausulas
lookForConjunto((F1a,F2a),(F1b,F2b)) :-
  lookForConjunto(F1a,F1b),
  lookForConjunto(F2a,F2b).

% conjunto: ignora se existir uma instanciacao
lookForConjunto(conjunto(X),conjunto(X)) :-
  clause(conjunto(X),_), !,
  undef(X,conjunto(X)),
  write('Agente3: jah definida internamente '),write(conjunto(X)),nl.

% conjunto: procura referencia a um conjunto jah definido (o objeto)
% se encontrar instancia localmente
lookForConjunto(conjunto(X),conjunto(X)) :-
  externalQuery(conjunto(X)), !,
  asserta(conjunto(X)),
  write('Agente3: jah definida externamente'),write(conjunto(X)),nl.

% conjunto: ignora se existir uma instanciacao com undef
lookForConjunto(conjunto(X),conjunto(X)) :-
  clause(undef(X,conjunto(X)),_), !,
  undef(X,conjunto(X)),
  write('Agente3: jah definida internamente '),write(undef(X,conjunto(X))),nl.

% conjunto: procura referencia a um conjunto jah definido com undef (o objeto)
% se encontrar instancia localmente
```

```
lookForConjunto(conjunto(X),conjunto(X)) :-
  externalQuery(undef(X,conjunto(X))), !,
  asserta(undef(X,conjunto(X))),
  write('Agente3: jah definida externamente'),write(undef(X,conjunto(X))),nl.

% conjunto: procura referencia a um conjunto jah definido com undef (o objeto)
% se encontrar instancia localmente
lookForConjunto(conjunto(X),conjunto(X)) :-
  externalQuery(def(X,conjunto(X))), !,
  asserta(def(X,conjunto(X))),
  write('Agente3: jah definida externamente'),write(undef(X,conjunto(X))),nl.

% casal: ignora se existir uma instanciacao
lookForConjunto(def(X,casal(X)),def(X,casal(X))) :-
  clause(conjunto(X),_), !,
  undef(X,conjunto(X)),
  write('Agente3: jah definida internamente '),write(conjunto(X)),nl.

% casal: procura referencia a um conjunto jah definido (o objeto)
% se encontrar instancia localmente
lookForConjunto(def(X,casal(X)),def(X,casal(X))) :-
  externalQuery(conjunto(X)), !,
  asserta(conjunto(X)),
  write('Agente3: jah definida externamente'),write(conjunto(X)),nl.

% casal: ignora se existir uma instanciacao com undef
lookForConjunto(def(X,casal(X)),def(X,casal(X))) :-
  clause(undef(X,conjunto(X)),_), !,
  undef(X,conjunto(X)),
  write('Agente3: jah definida internamente '),write(undef(X,conjunto(X))),nl.

% casal: procura referencia a um conjunto jah definido com undef (o objeto)
% se encontrar instancia localmente
lookForConjunto(def(X,casal(X)),def(X,casal(X))) :-
  externalQuery(undef(X,conjunto(X))), !,
  asserta(undef(X,conjunto(X))),
  write('Agente3: jah definida externamente'),write(undef(X,conjunto(X))),nl.
```

```
% casal: procura referencia a um conjunto jah definido com undef (o objeto)
% se encontrar instancia localmente
lookForConjunto(def(X,casal(X)),def(X,casal(X))) :-
  externalQuery(def(X,conjunto(X))), !,
  asserta(def(X,conjunto(X))),
  write('Agente3: jah definida externamente'),write(undef(X,conjunto(X))),nl.

% cria um novo conjunto quando nao encontra  nenhuma referencia anterior
lookForConjunto(conjunto(X),conjunto(X)) :-
  externalAssert(undef(X,conjunto(X))),
  asserta(undef(X,conjunto(X))),
  write('Agente3: nova definicao'),write(undef(X,conjunto(X))),nl.

% tentativa de responder sempre corretamente
lookForConjunto(X,X) :-
  write('Agente3: falharam todas'),nl.
```

4

rrd4.pl

Algoritmo do agente 4

Algoritmo do agente 4. Procura resolver referências que envolvem relações normalmente conhecidas, como quando uma entidade faz parte de um todo (escrivaninha/gaveta), tem algum tipo de relacionamento já estabelecido. Por exemplo, na sentença "Pedro comprou um livro. O autor é famoso" o agente procura uma regra que relacione o autor com o livro e instancia uma regra que faz a ligação entre as duas entidades mencionadas.

```
% rrd4.pl
% Procura antecedentes atraves de relacoes: possui, conjunto,
% e-parte-de, etc.

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente4: '), write(P), nl.

% inicia o processo de resolu,c~ao de refer~encias
processProposition(Clause,NewClause) :-
  lookForDef(Clause,NewClause).

lookForDef((HeadA:-F1a),(HeadB:-F1b)) :-
  lookForDef(HeadA,HeadB),
  lookForDef(F1a,F1b).

lookForDef((HeadA:-F1a,F2a),(HeadB:-F1b,F2b)) :-
  lookForDef(HeadA,HeadB),
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% conjunto de quatro cl'ausulas
lookForDef((F1a,F2a,F3a,F4a),(F1b,F2b,F3b,F4b)) :-
```

```
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b),
  lookForDef(F4a,F4b).

% conjunto de tr~es cl'ausulas
lookForDef((F1a,F2a,F3a),(F1b,F2b,F3b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b).

% conjunto de duas cl'ausulas
lookForDef((F1a,F2a),(F1b,F2b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% procura um possui como definido mais adiante e se encontrar envia um
% assert externo.
lookForDef(def(X,F),def(X,F)) :-
  write('Agente4: tentativa 1'),nl,
  possui(X,Y), !,
  asserta(def(X,F)),
  externalAssert(possui(X,Y)),
  externalAssert(def(X,F)),
  write('Agente4: (possui interno) externalAssert('),
  write('def('), write(X), write(','), write(Y),write(')'), nl.

% Consulta 2. Instancia uma das entidades sendo testadas.
lookForDef(def(X,F),def(X,F)) :-
  write('Agente4: tentativa 2'),nl,
  newReferent(X),
  write('Agente4: criando novo referente '),write(X),nl,
  assertz(def(X,F)),
  write('Agente4: asserting def('),write(X),write(','),write(F),write(')'),nl,
  possui(X,Y), !,
  externalAssert(possui(X,Y)),
  externalAssert(def(X,F)),
  write('Agente4: (possui interno) externalAssert('),
```

```
  write('def('), write(X), write(','), write(Y),write(')'), nl.

% o algoritmo falhou em todos os tratamentos.
lookForDef(X,X) :-
  write('Agente4: Falharam todas.'),nl.

% Conhecimentos especificos deste agente.
%  - um livro possui um autor; o livro possui um autor.
possui(X,Y):-
  adef(X,livro(X)), adef(Y,autor(Y)).

adef(X,F) :-
  def(X,F);
  undef(X,F).

def(X,F) :-
  write('Agente4: externalQuery(def('),write(X),write(','),write(F),write('))'
  externalQuery(def(X,F));
  clause(def(X,F),_).

undef(X,F) :-
  write('Agente4: externalQuery(undef('),write(X),write(','),write(F),write('))
  externalQuery(undef(X,F));
  clause(undef(X,F),_).
```

5

rrd5.pl

Algoritmo do agente 5

Algoritmo do agente 5. Quando não é possível encontrar relações entre dois objetos dentro do texto, este agente cria uma nova relação entre os objetos mais próximos, considerando a concordância em número e gênero.

As relações criadas por este agente se baseiam no predicado "antecedente", significando que o sintagma anterior é um possível antecedente da referência sendo analisada.

```
% rrd5.pl
% Faz uma anotacao para saber se uma entidade antecede a outra no
% discurso, mantendo um ordem entre objetos referencidados.

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente5: '), write(P), nl.

% inicia o processo de resolucao de referencias
processProposition(Clause,NewClause) :-
  lookForDef(Clause,NewClause).

lookForDef((HeadA:-F1a),(HeadA:-F1b)) :-
  lookForDef(F1a,F1b).

lookForDef((HeadA:-F1a,F2a),(HeadA:-F1b,F2b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% conjunto de quatro clausulas
lookForDef((F1a,F2a,F3a,F4a),(F1b,F2b,F3b,F4b)) :-
  lookForDef(F1a,F1b),
```

```
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b),
  lookForDef(F4a,F4b).

% conjunto de tres clausulas
lookForDef((F1a,F2a,F3a),(F1b,F2b,F3b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b),
  lookForDef(F3a,F3b).

% conjunto de duas clausulas
lookForDef((F1a,F2a),(F1b,F2b)) :-
  lookForDef(F1a,F1b),
  lookForDef(F2a,F2b).

% procura por def's aninhados
lookForDef(def(X,def(Y,F)),(G,H)) :-
  lookForDef(def(X,F),G),
  lookForDef(def(Y,F),H).

% ignora se existir uma instanciacao
lookForDef(def(X,F),def(X,F)) :-
  clause(def(X,F),_), !,
  def(X,F).

% procura referencia a um def (o objeto)
lookForDef(def(X,F),def(X,F)) :-
  externalQuery(def(X,F)), !,
  asserta(def(X,F)).

% procura referencia a um undef (um objeto)
%lookForDef(def(X,F),def(X,F)) :-
%  externalQuery(undef(X,F)), !,
%  asserta(def(X,F)),
%  externalAssert(def(X,F)),
%  write('Agente5 asserta: '), write(def(X,F)), nl.

% inicia uma busca por antecedentes, dando preferencia
```

```
% ao mais recentes. O limite esta fixado no predicado
% "buscaAntecedente".
lookForDef(def(X,F),def(X,F)) :-
  buscaAntecedenteDef(def(Y,F),def(Y,F),2);
  buscaAntecedenteUndef(def(Y,F),def(Y,F),2).

% tentativa de responder sempre corretamente
%lookForDef(X,X) :- !.

% busca o i-esimo antecedente num limite de 5 sintagmas anteriores
% e cria uma relacao de possivel antecedente. Da preferencia a
% sintagmas definidos.
buscaAntecedenteDef(def(X,F),def(X,F),N) :-
  menor(N,5),
  write(buscaAntecedenteDef(def(X,F),def(X,F),N)), nl,
  externalQueryn(def(Y,F2),N),   % o penultimo sintagma
  externalAssert((antecedente(X,Y):-def(X,F),def(Y,F2))),
  asserta((antecedente(X,Y):-def(X,F),def(Y,F2))),
  write('Agente5: '),write((antecedente(X,Y):-def(X,F),def(Y,F2))),
  plus(N,1,N1),
  buscaAntecedenteDef(def(X,F),def(X,F),N1);
  menor(N,5),
  plus(N,1,N1),
  buscaAntecedenteDef(def(X,F),def(X,F),N1).

% o mesmo caso que o anterior, mas da preferencia a sintagmas nominais
% nao definidos.
buscaAntecedenteUndef(def(X,F),def(X,F),N) :-
  menor(N,5),
  write(buscaAntecedenteUndef(def(X,F),def(X,F),N)), nl,
  externalQueryn(undef(Y,F2),N), % o penultimo sintagma
  externalAssert((antecedente(X,Y):-def(X,F),undef(Y,F2))),
  asserta((antecedente(X,Y):-def(X,F),undef(Y,F2))),
  write('Agente5: '),write((antecedente(X,Y):-def(X,F),undef(Y,F2))),nl,
  plus(N,1,N1),
  buscaAntecedenteUndef(def(X,F),def(X,F),N1);
  menor(N,5),
  plus(N,1,N1),
```

```
  buscaAntecedenteUndef(def(X,F),def(X,F),N1).


% funcao para comparar dois numeros.
menor(X,Y) :-
  X < Y.
```

6

rrd6.pl

Algoritmo do agente 6

Algoritmo do agente 6. Este agente ilustra a possibilidade de se tratar problemas de referência a pronomes pessoais e outros tipos de referências não relacionados diretamente a casos de sintagmas nominais definidos. O código deste agente trata o pronome pessoal "eles".

Exemplo:

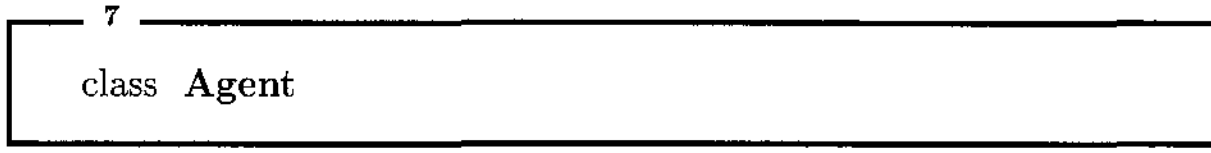
Ana e Pedro sairam para passear $\longrightarrow$

```
saiu(X) :- ana('Ana'), pedro('Pedro'), conjunto(X,['Ana','Pedro']).
```

```
% rrd6.pl
% Ilustra o tratamento do pronome pessoal ''eles''

processSent(Clause,P) :-
  processProposition(Clause,P),
  asserta(P),
  externalAssert(P),
  write('Agente6: '), write(P), nl.

% inicia o processo de resolucao de referencias
processProposition(Clause,NewClause) :-
  lookForEles(Clause,NewClause).

lookForEles((HeadA:-F1a),(HeadA:-F1b)) :-
  lookForEles(F1a,F1b).

lookForEles((HeadA:-F1a,F2a),(HeadA:-F1b,F2b)) :-
  lookForEles(F1a,F1b),
  lookForEles(F2a,F2b).

% conjunto de quatro clausulas
lookForEles((F1a,F2a,F3a,F4a),(F1b,F2b,F3b,F4b)) :-
```

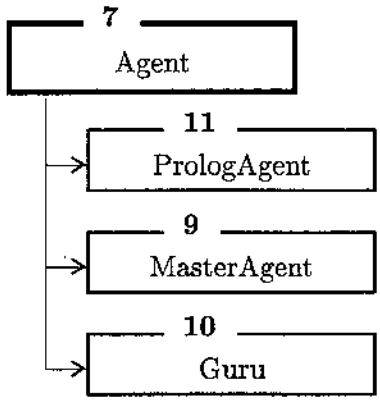
```
  lookForEles(F1a,F1b),
  lookForEles(F2a,F2b),
  lookForEles(F3a,F3b),
  lookForEles(F4a,F4b).

% conjunto de tres clausulas
lookForEles((F1a,F2a,F3a),(F1b,F2b,F3b)) :-
  lookForEles(F1a,F1b),
  lookForEles(F2a,F2b),
  lookForEles(F3a,F3b).

% conjunto de duas clausulas
lookForEles((F1a,F2a),(F1b,F2b)) :-
  lookForEles(F1a,F1b),
  lookForEles(F2a,F2b).

% ignora se existir uma instanciacao d'eles
lookForEles(eles(X),eles(X)) :-
  clause(eles(X),_), !,
  eles(X),
  write('Agente6: jah definida internamente '),write(eles(X)),nl.

% ignora se existir uma instanciacao d'eles
lookForEles(eles(X),eles(X)) :-
  externalQuery(eles(X)), !,
  write('Agente6: jah definida externamente'),write(eles(X)),nl.

% procura por um conjunto definido internamente (definido)
lookForEles(eles(X),eles(X)) :-
  clause(conjunto(X),_), !,
  conjunto(X),
  asserta(eles(X)),
  externalAssert(eles(X)),
  write('Agente6: definicao baseada em conjunto interno'),write(conjunto(X)),n

% procura por um conjunto definido internamente (indefinido)
lookForEles(eles(X),eles(X)) :-
  clause(undef(X,conjunto(X)),_), !,
```

```
  undef(X,conjunto(X)),
  asserta(eles(X)),
  externalAssert(eles(X)),
  write('Agente6: definicao baseada em conjunto interno'),write(undef(X,conjunto

% procura referencia a um conjunto jah definido (o objeto)
% se encontrar instancia localmente
lookForEles(eles(X),eles(X)) :-
  externalQuery(conjunto(X)), !,
  asserta(eles(X)),
  externalAssert(eles(X)),
  write('Agente6: definicao baseada em conjunto externo'),write(eles(X)),nl.

% procura referencia a um conjunto jah definido com undef (o objeto)
% se encontrar instancia localmente
lookForEles(eles(X),eles(X)) :-
  externalQuery(undef(X,conjunto(X))), !,
  asserta(eles(X)),
  externalAssert(eles(X)),
  write('Agente6: definicao baseada em conjunto externo (undef)'),write(eles(X))

% cria um novo conjunto quando nao encontra  nenhuma referencia anterior
lookForEles(eles(X),eles(X)) :-
  newReferent(X),
  externalAssert(eles(X)),
  asserta(eles(X)),
  write('Agente6: nova definicao'),write(eles(X)),nl.

% tentativa de responder sempre corretamente
lookForEles(X,X) :-
  write('Agente6: falharam todas'),nl.
```



Classe *Agent*

Inheritance



Membros Públicos

7.3		**Agent** (int agentNumber)	
		Construtor	87
		Agent () *Construtor. Atribui automaticamente um identificador único para o agente. Não recebe parâmetros.*	
	char*	**retGroup** () *Grupo. Retorna uma seqüência de caracteres com o nome do último grupo ao qual o agente se filiou.*	
7.4	void	**setGroup** (char *group)	
		Novo grupo	88
7.5	void	**waitMessage** (Message *msg)	
		Recebe uma mensagem	88
7.6	void	**waitMessage** (Message *msg, int msgtag)	

			Recebe uma mensagem	88
7.7	int	**waitnMessage** (Message *msg)		
			Recebe uma mensagem	89
7.8	int	**waitnMessage** (Message *msg, int msgtag)		
			Recebe uma mensagem	89
7.9	void	**sendMessage** (int id, Message *msg)		
			Envia uma mensagem	90
7.10	void	**sendBcast** (char *grupo, Message *msg)		
			Envia uma mensagem	90

Membros Protegidos

7.1	int	**agentId**	**Identificador**	91
7.2	char*	**myGroupName**	**Nome de grupo**	91

Classe *Agent*. Define os atributos e comportamento comuns a todos os agentes do sistema. Estas definições permitem a criação de diferentes agentes baseados em uma mesma arquitetura, incluindo identificadores únicos para cada um e mecanismos de comunicação similares, baseados em *send* e *wait* (*receive*). Esta classe determina a homogeneidade necessária para se desenvolver um sistema multiagentes que se comunica utilizando o mesmo protocolo de comunicação. Os agentes mestres e escravos devem herdar da classe Agent.

7.3

Agent (int agentNumber)

Construtor

Construtor. Este construtor pode receber como parâmetro o seu identificador. Não é necessário utilizá-lo normalmente, pois se nenhum identificador for passado o PVM atribui automaticamente um identificador único para ele.

Parâmetros: `agentNumber` — identificador único para o agente.

7.4

void **setGroup** (char *group)

Novo grupo

Novo grupo. Configura o grupo ao qual o agente pertence. Não o remove de grupos anteriores, mas o acrescenta numa lista de grupos.

7.5

void **waitMessage** (Message *msg)

Recebe uma mensagem

Recebe uma mensagem. A mensagem deve ser um objeto da classe *Message* e será preenchido pelo método *unpack* desta classe. O identificador de quem enviou a mensagem constará como um atributo do próprio objeto.

Parâmetros: `group` — seqüência de caracteres determinando o nome do grupo ao qual o agente deve se filiar.

7.6

void **waitMessage** (Message *msg, int msgtag)

Recebe uma mensagem

Recebe uma mensagem. Similar ao método waitMessage de aridade 1, com a vantagem de selecionar somente um determinado tipo de mensagem. Outras mensagens que estejam na fila de tipo diferente do especificado serão ignoradas. Os tipos de mensagens podem ser vistos na definição da classe *Message*.

Parâmetros: `msg` — objeto do tipo Message
`msgtag` — tipo da mensagem esperada

7.7

int **waitnMessage** (Message *msg)

Recebe uma mensagem

Recebe uma mensagem. Este método faz a mesma coisa que o método *waitMessage* mas não aguarda, apenas retorna a mensagem que estiver na fila. Retorna 1 quando recebe uma mensagem com sucesso e 0 (zero) caso contrário.

Parâmetros: `msg` — objeto do tipo Message

7.8

int **waitnMessage** (Message *msg, int msgtag)

Recebe uma mensagem

Recebe uma mensagem. Este método é similar ao método *waitMessage* mas não aguarda resposta, apenas retorna a mensagem que estiver na fila. Retorna 1 quando recebe uma mensagem com sucesso e 0 (zero) caso contrário (quando não há mensagem na fila).

Parâmetros: `msg` — objeto do tipo Message
`msgtag` — tipo da mensagem esperada

7.9

void **sendMessage** (int id, Message *msg)

Envia uma mensagem

Envia uma mensagem. O conteúdo da mensagem pode variar conforme o seu tipo e o destinatário deve ser especificado de acordo com seu identificador único. O identificador pode ser obtido na própria mensagem que um agente recebe. A rotina garante que a entrega será feita, mas não garante o tempo que isso pode levar.

Parâmetros: `id` — identificador único do agente
`msg` — conteúdo da mensagem

7.10

void **sendBcast** (char *grupo, Message *msg)

Envia uma mensagem

Envia uma mensagem. Apesar do nome sugerir um *broadcast* o envio é feito na forma de *multicast*, ou seja, a mensagem é replicada para cada agente que se filiou ao grupo. O grupo é identificado por uma seqüência de caracteres.

Parâmetros: `grupo` — nome do grupo receptor
`msg` — conteúdo da mensagem

7.1

int **agentId**

Identificador

Identificador. Este é um identificador único para um agente dentro do sistema. Atualmente este identificador é do mesmo tipo utilizado pelo PVM e garante que uma determinada mensagem seja enviada para um único agente.

7.2

char* **myGroupName**

Nome de grupo

Nome de grupo. Contém o nome do grupo ao qual o agente pertence. Na realidade um agente pode pertencer a vários grupos, mas apenas o nome do último grupo ao qual ele foi registrado é que fica guardado nesta variável. Pode-se dar qualquer nome para um grupo de agentes.

8

class Message

Classe *Message*

Membros Públicos

	void	**setSender** (int s)	*Remetente. Configura o identificador do remetente.*
	int	**retSender** ()	*Remetente. Retorna o identificador do remetente da mensagem.*
	void	**setType** (int t)	*Tipo. Configura o tipo da mensagem sendo enviada. Pode ser um dos tipos citados em Type.*
8.2	int	**retType** ()	**Tipo** 94
	void	**setSeq** (int s)	*Número seqüencial. Configura o número de seqüência da mensagem. O número seqüencial não é único e começa com zero em cada agente do sistema.*
	int	**retSeq** ()	*Número seqüencial. Retorna o número de seqüência da mensagem.*
	Message*	**retNext** ()	*Próxima. Retorna a próxima mensagem da fila.*
	void	**setGroup** (char *name)	*Mensagem. Coloca no conteúdo da mensagem o nome do grupo o que a mesma é do tipo GROUP, ou seja, serve para configurar o grupo ao qual pertence o agente que receberá a mensagem.*
	char*	**retGroupName** ()	

		Mensagem. Retorna o conteúdo da mensagem (o mesmo que setMessage).
char*	**retMessage** ()	*Mensagem. Retorna o conteúdo da mensagem (o mesmo que retGroupName)*
void	**setMessage** (char *m)	*Mensagem. Configura o conteúdo da mensagem.*
void	**pack** ()	*Empacota. Prepara a mensagem para ser enviada pelo PVM.*
void	**unpack** ()	*Desempacota. Retira os dados do buffer do PVM.*
void	**show** (FILE *log)	*Visualização. Mostra o conteúdo da mensagem fazendo a formatação de acordo com o seu tipo.*
void	**clear** ()	*Limpeza. Força uma limpeza dos dados*
	Message (messageType, int, char*)	*Construtor. Configura o tipo, número de seqüência da mensagem e conteúdo.*
	Message ()	*Construtor. Inicializa todos os dados com vazio. O contéudo da mensagem é nula, o número de seqüência é -1 e o tipo é desconhecido (-1).*
	~Message ()	*Destrutor. Remove o conteúdo da mensagem e de todas as mensagens que estejam na fila do objeto.*

Membros Privados

	int	**sender**	*Identificador do agente que envia a mensagem.*	
	int	**Type**	*Tipo da mensagem. Veja o método setType para saber quais são os tipo possíveis.*	
	int	**seq**	*Número seqüencial. Especifica um número de seqüência da mensagem. Utilizado apenas para controle interno*	
	char*	**message**	*Mensagem. É o apontador para o buffer da mensagem. Pode ser uma seqüência qualquer de caracteres.*	
8.1	Message*	**next**	**Lista de mensagens**	95
	int	**insert** (char *m)	*Inserção. Insere uma nova mensagem na lista, fazendo o encadeamento de mensagens.*	

Classe *Message*. Esta classe é a responsável pelo empacotamento de mensagens pela rede. Não faz nenhum serviço de envio ou recepção. Todo o serviço de envio ou recepção está a cargo da classe Agente. A funcionalidade desta classe resume-se ao encapsulamento de uma mensagem e a garantia de que não ocorrerá erros no empacotamento e desempacotamento das mensagens. Note que os métodos *pack* e *unpack* são simétricos do ponto de vista de empilhamento e desempilhamento de dados. Os dados que são codificados na transmissão, são decodificados na mesma ordem na recepção. O limite do tamanho de uma mensagem é imposto pela versão do PVM que estiver em uso. Atualmente na versão do PVM 3.4 não se menciona um limite específico.

8.2

int retType ()

Tipo

Tipo. Retorna o tipo da mensagem, que pode ser um dos tipos abaixo:

- ANSWER: resposta oriunda de um agente
- ANSWERALL: resposta ao ASKALL
- ASK: consulta de um agente para outro
- ASKALL: consulta de várias perguntas
- ASSERT: avisa para um agente que deve atualizar sua base
- BARRIER: para ativar uma barreira em um determinado grupo
- DUMMY: utilizado para debug
- GROUP: mensagem inicial para criação de grupo
- KILL: avisa para o agente se desativar
- QUERY: consulta genérica do mestre a um agente
- SENTENCE: sentença original

8.1

Message* **next**

Lista de mensagens

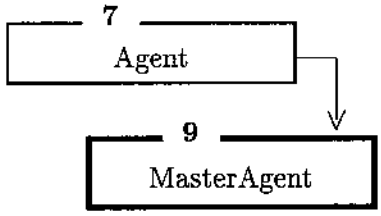
Lista de mensagens. Ponteiro utilizado para fazer o encadeamento de várias mensagens, como numa lista. Quando se utiliza o método de inserção de mensagem e quando a mensagem é do tipo ANSWERALL ou ASKALL o objeto empacota todos os dados referentes a todas as mensagens constantes da lista.

9

class **MasterAgent** : public Agent

Classe *MasterAgent*

Inheritance



Membros Públicos

9.2 **MasterAgent** (char *fileName)
Construtor 97
9.3 ~**MasterAgent** () **Destrutor** 97

Membros Privados

9.1 void **start** (char *type, char *group)
Início 98

Classe *MasterAgent.* Esta classe define um agente mestre capaz de inicializar todo o sistema. Um objeto desta classe recebe como entrada um arquivo de sentenças representando o discurso a ser processado. O construtor da classe coloca todos os outros agentes em funcionamento e envia mensagens do tipo SENTENCE para cada um deles com as sentenças a serem processadas. Por ser descendente de um agente possui todos os recursos de comunicação comum a todos os agentes do sistema.

O destrutor da classe se encarrega de desativar todos os agentes que ativou com mensagens do tipo KILL.

9.2

MasterAgent (char *fileName)

Construtor

Construtor. Este construtor inicia todo o sistema utilizando o seu método privado *start*. Após ativar todos os grupos do sistema envia uma mensagem de barreira para fazer a sincronização inicial e começa a ler as sentenças do discurso que devem ser processadas. Envia cada sentença (mensagens do tipo SENTENCE) e aguarda até que todos os agentes as tenham recebido e processado. Quando terminam as sentenças este agente não é mais utilizado. A sua existência se limita a fazer consultas sobre o texto processado com base nas informações que o sistema gerou e armazenou durante o processamento das sentenças.

Parâmetros: `fileName` — Nome do arquivo que contém as sentenças a serem processadas. Se for vazio é considerado a entrada padrão, caso contrário lê-se de um arquivo.

9.3

~MasterAgent ()

Destrutor

Destrutor. O destrutor aguarda até que todos os agentes tenham terminado o seu processamento e então encerra o sistema enviando uma mensagem do tipo KILL para cada um deles.

9.1

void start (char *type, char *group)

Início

Início. Inicializa um grupo de agentes. Um grupo de agentes pode ser do tipo:

- *guru*: agente que é capaz de "monitorar" todas as comunicações feitas no sistema. Este agente é útil quando se quer saber em detalhes o que o sistema acumulou em termos de informações.
- *slave*: um escravo é um tipo de agente que recebe consultas e as processa. O seu comportamento está descrito na classe *PrologAgent*. Podem existir vários objetos desta classe, mas cada um com sua própria heurística. Para cada "slave" existe um "master" associado.
- *master*: é um outro agente genérico do sistema e a sua função é apenas receber consultas do sistema

Parâmetros: `type` — seqüência de caracteres contendo o tipo do agente, que pode ser "master" ou "slave".
`group` — determina o grupo que está sendo inicializado, que pode ser "guru", "agent" ou "master".

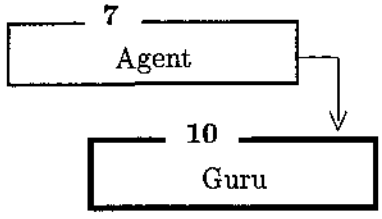
Veja Também: PrologAgent

10

class **Guru** : public Agent

Classe *guru*

Inheritance



Membros Públicos

	Guru ()	*Construtor. O construtor abre o arquivo de cláusulas para gravação. Não verifica se já existe um arquivo anterior, apenas sobrepõe ou cria um novo arquivo a cada nova sessão. O arquivo possui um nome fixo "clauses.pl" e pode ser carregado por um Prolog comum. Atualmente utiliza o SWI-Prolog 3.0.*
	~**Guru** ()	*Destrutor. O destrutor fecha o arquivo de cláusulas e abandona o sistema.*
10.1 void	**init** (int argc, char **argv)	**Inicialização** 100

Membros Privados

FILE*	**clausesFile**	*Arquivo. Possui um descritor de arquivo onde pode armazenar em formato texto as cláulas geradas pelo sistema. Tem utilidade para se fazer testes posteriores utilizando um Prolog independente do sistema multiagentes.*
Prolog	**prolog**	*Prolog. Possui um objeto do tipo Prolog capaz de armazenar dados, responder consultas e fazer inferências. Pelas regras do sistema não faz consultas externas, pois gerariam consultas não úteis para resolver o problema.*

Classe *guru*. Este classe foi criada apenas para monitorar os "eventos" dentro do sistema. Como é derivada classe agente faz as mesmas comunicações que os outros agentes. Um objeto desta classe recebe todas as informações geradas no sistema e armazena utilizando um objeto do tipo Prolog. É possível fazer consultas a este objeto assim como a qualquer outro agente do sistema.

Veja Também: PrologAgent
MasterAgent

10.1

void **init** (int argc, char **argv)

Inicialização

Inicialização. É similar à inicialização de qualquer agente especializado do sistema. Logo após as inicializações necessárias ao Prolog entra em um laço de tratamento de mensagens. Este irá terminar apenas quando o agente receber uma mensagem do tipo KILL. As mensagens que ele trata são:

- BARRIER: entra em modo de espera até que todos os agentes tenham recebido uma cópia da mesma mensagem.
- QUERY, ANSWER, ASK: faz uma consulta ao Prolog local e envia uma única resposta, quando houver.
- ASKALL: faz uma consulta ao Prolog local e envia todas as respostas possíveis, descartando as repetições e cláusulas equivalentes.
- GROUP: filia-se ao grupo indicado no conteúdo da mensagem.
- KILL: abandona o sistema.
- ASSERT: acrescenta uma nova cláusula à sua base de dados local executando um *assert* do seu Prolog.

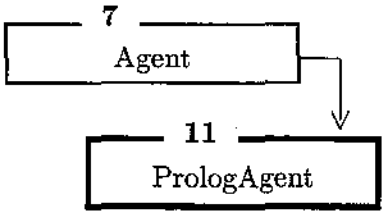
Veja Também: Message

11

class **PrologAgent** : public Agent

Classe *PrologAgent*

Inheritance



Membros Públicos

11.1 **PrologAgent** () **Construtor** 102
11.2 void **init** (int argc, char **argv)
Inicialização 103

Classe *PrologAgent*. Esta classe herda todas as características de um agente do sistema e possui como atributo um objeto do tipo *Prolog*, o que possibilita a ele a interpretação de código em Prolog.

Veja Também: Prolog

11.1

PrologAgent ()

Construtor

Construtor. Inicializa o objeto Prolog, inicializando suas variáveis de ambiente e instalando as rotinas em C utilizadas pelo Prolog para fazer a comunicação com outros agentes.

11.2

void **init** (int argc, char **argv)

Inicialização

Inicialização. Este método é o que inicializa o agente efetivamente, colocando-o em um loop de tratamento de mensagens. Um agente do tipo Prolog possui dois modos distintos de atuação. Ele pode ser um *master* ou um *slave*.

O *master* processa mensagens do tipo:

- SENTENCE: inclui a sentença contida na mensagem em sua base de conhecimento.
- ASK: responde a uma consulta de acordo com o seu conteúdo na base de dados. Utiliza para isto o Prolog deste objeto.
- ASSERT: inclui a nova cláusula em sua base de dados. Não insere cláusulas que sejam idênticas ou unificáveis entre si, ou seja, não repete inserção.

O *slave* processa mensagens do tipo:

- SENTENCE: processa a sentença de acordo com a heurística para a qual foi programado. A sua programação depende de um módulo em Prolog que é ligado no momento da compilação. Todos os agentes do sistema utilizam esta classe como base para a implementação das heurísticas, sendo estas escritas diretamente em Prolog. O processamento de uma sentença envolve a aplicação da heurística sobre a mesma e o preenchimento de uma mensagem de resposta do tipo ANSWER com possíveis soluções ou refinamentos na sentença inicial. Os refinamentos consistem na eliminação de ambigüidades ou sugestões para eliminá-las. As heurísticas podem gerar também consultas a outros agentes antes de responder a esta chamada. Um mesmo agente nunca responde a uma consulta que seja feita por ele mesmo por uma imposição da própria arquitetura do sistema.
- QUERY: responde às consultas do sistema do tipo QUERY e envia todas as respostas que conseguiu ao sistema novamente. Como todos os agentes também recebem a mesma sentença, cada um deles aguarda até que os outros agentes também tenham processado a mesma sentença.

Todos os agentes (mestres e escravos) processam as seguintes mensagens:

- BARRIER: quando um agente recebe este tipo de mensagem entra em estado de espera até que todos os outros agentes do sistema, incluindo mestres e escravos, tenham recebido esta mensagem. Utiliza-se o recurso de barreira do PVM (*pvm_barrier*).
- GROUP: esta mensagem solicita que o agente se filie a um grupo.
- KILL: avisa para o agente abandonar o sistema.
- DUMMY: o agente não faz processamento, apenas envia a mesma mensage como resposta, invertendo o remetente com o destinatário. Serve para testar se o agente está respondendo ou não.

12

class Prolog

Classe *Prolog*

Membros Públicos

		Prolog ()	*Construtor. Não tem nenhuma funcionalidade por enquanto.*
12.1	void	**init** (int argc, char **argv)	
		Inicializa	106
12.2	int	**process** (char *sentence)	
		Processa	107
12.3	int	**process** (char *in, char **out)	
		Processa	107
12.4		**askFor** (char *clause, Message *msg)	
		Consulta	108
12.5		**askForAll** (char *clause, Message *msg)	
		Consulta	108
12.6		**assert** (char *clause)	
		Atualiza	109

Membros Privados

Parser	**parser**	*Analisador. Este objeto é capaz de analisar cláusula do Prolog e traduzí-las para a estrutura interna do Prolog e vice-versa, ou seja, traduz cláusulas do Prolog para uma seqüência de caracteres. Esta tradução é necessária para se fazer a comunicação entre os agentes, que interpretam apenas cláusulas Prolog em formato ASCII.*

module_t	**user_module**	*Módulo. Um Prolog pode ter vários módulos ao mesmo tempo permitindo a montagem de vários contextos. Atualmente somente um módulo é utilizado.*

Classe *Prolog.* Esta classe é responsável por toda interface feita entre os programas em C++ e Prolog. Os métodos do tipo *process* e *ask* fazem chamadas ao Prolog e retornam o resultado em forma de seqüência de caracteres. O *assert* faz a mesma coisa que o comando *assert* do próprio Prolog.

12.1

void **init** (int argc, char **argv)

Inicializa

Inicializa. Cria um novo módulo para trabalhar com os dados do agente e registra as rotinas em C que serão chamadas pelo Prolog. O prolog pode utilizar as chamadas *externalQuery*, *externalQueryn*, *newReferent* e *externalAssert*. Estão descritas abaixo as funcionalidades de cada uma delas:

- *externalQuery(term_t t)*: recebe um termo do Prolog para efetuar uma consulta ao sistema. Esta consulta utiliza a rotina *externalQueryn* descrita abaixo com o segundo parâmetro sendo 1.
- *externalQueryn(term_t t, term_t tn)*: recebe um termo do Prolog para efetuar uma consulta ao sistema. O segundo argumento da rotina indica que se quer como retorno a n-ésima resposta à consulta. Quando não existe tal resposta retorna falha e um termo vazio como resposta.
- *newReferent*: aloca uma nova constante para o sistema. Esta constante é única, pois se baseia no identificador único de cada agente do sistema. Um referente tem a forma de *aa80004*, onde as duas primeiras letras variam de "aa" a "zz" e os digitos seguintes são hexadecimais contendo o identificador do agente, que já é gerado de forma única.

- *externalAssert*: coloca no sistema uma nova informação. Os agentes que recebem a mensagem podem escolher se querem acrescentar a nova informação à sua base de dados ou não. É um forma de publicação global de informações.

Os argumentos argc e argv passados como parâmetros não possuem significado especial. Servem apenas para manter a compatibilidade com versões anteriores.

12.2

int **process** (char *sentence)

Processa

Processa. Recebe uma cláusula em forma de cláusula de Horn e procura prová-la utilizando a base de dados que o agente possui. Utiliza as heurísticas programadas em cada agente para provar a cláusula, instanciando novas informações quando necessário e fazendo consultas ao sistema. Quando consegue provar retorna verdadeiro, caso contrário retorna falso.

Parâmetros: `sentence` — Sentença em cláusula de Horn a ser processada.

12.3

int **process** (char *in, char **out)

Processa

Processa. Faz o mesmo que o método *process*, com a diferença de que retorna a cláusula instanciada no segundo parâmetro quando consegue provar a primeira.

Parâmetros: `in` — Senten,ca em cláusula de Horn a ser processada.
`out` — Retorno da sentença instanciada.

12.4

askFor (char *clause, Message *msg)

Consulta

Consulta. Apenas tenta provar a cláusula passada como parâmetro com os dados já existentes. Preenche a mensagem passada como parâmetro, sendo que o conteúdo contém a cláusula instanciada.

Parâmetros: `clause` — Cláusula sendo consultada.
`msg` — Mensagem de resposta à consulta. Pode ser inclusive uma mensagem vazia.

12.5

askForAll (char *clause, Message *msg)

Consulta

Consulta. A consulta feita é similar ao da *askFor*, com a diferença de que obtém todas as respostas possíveis do Prolog. Como muitas respostas podem ser equivalentes umas às outras (da ordem de milhares) é feita uma pré-seleção onde as repetidas ou unificáveis são retiradas das respostas. Dá-se preferência às respostas sem variáveis livres.

Parâmetros: `clause` — Cláusula sendo consultada.
`msg` — Mensagem contendo todas as respostas. O formato da mensagem é uma lista de mensagens, cada uma com o endereço do remetente e destinatário.

12.6

assert (char *clause)

Atualiza

Atualiza. Inclui uma nova cláusula na base de dados.

Parâmetros: `clause` — Nova cláusula de Horn em formato ASCII. Exemplo: "`P(X):-H(X),Velho(X)`".

13

class **Parser**

Classe *Parser*

Membros Públicos

int **TermToString** (term_t t, char **str)
Conversão. Faz a conversão inversa de StringToTerm, ou seja, de uma representação interna do Prolog para o formato ASCII. O retorno da seqüência de caracteres é feita no segundo parâmetro. A alocação é feita utilizando-se malloc e pode ser desalocada pela rotina que o chamou.

term_t **StringToTerm** (char *buffer)
Conversão. Converte uma seqüência de caracteres em um termo do Prolog. A seqüência de caracteres deve estar na forma de cláusulas de Horn. Exemplo:
`p(X) :- q(X), r(X).`
`r(a).`

int **nconst** (char *clause)
Contagem. Conta o número de constantes de uma cláusula em forma de string.

int **nconst** (term_t t) *Contagem. Conta o número de constantes de uma cláusula em forma de termo Prolog.*

int **isFreeVar** (char *clause)

Contagem. Retorna diferente de zero quando encontra uma variável livre em uma cláusula em forma de string.

int **isFreeVar** (term_t t)

Contagem. Retorna diferente de zero quando encontra uma variável livre em uma cláusula em forma de termo Prolog.

int **isImplication** (char *clause)

Implicação. Retorna diferente de zero quando a cláusula contém uma implicação.

int **isImplication** (term_t t)

Implicação. Retorna diferente de zero quando o termo contém uma implicação.

Membros Privados

TK_NULL SymbolTable

table

Tabela de símbolos. Esta tabela é utilizada na tradução de sentenças para garantir que as variáveis sejam referidas por um mesmo termo em Prolog quando possuem um mesmo nome.

13.1 term_t **StringToTerm** (char **str)

Conversão 112

int **returnArgs** (char **str, term_t *args)

Argumentos. Retorna os argumentos de um predicado qualquer em forma de um termo composto. Retorna diferente de zero quando consegue sucesso.

int **initSymbolTable** ()

Tabela de Símbolos. Inicializa a tabela de símbolos.

int **clearSymbolTable** ()

Tabela de Símbolos. Elimina todos os símbolos armazenados na tabela. Este procedimento é feito após cada conversão de uma cláusula.

13.2 int **nextToken** (char **str, char *name)

Parsing 113

term_t **insertSymbol** (char *variable)

Tabela de Símbolos. Insere o nome de uma nova variável na tabela. Caso a variável já exista retorna o mesmo termo que se refere a ela; senão cria um novo termo.

Classe *Parser*. Esta classe implementa métodos para traduzir sentenças do Prolog em ASCII para estruturas do tipo termo (*term_t*) e vice-versa.

13.1

term_t **StringToTerm** (char **str)

Conversão

Conversão. Este método tem a mesma funcionalidade do método StringToTerm público, com a diferenca da dupla indireção utilizada no parâmetro. Como a seqüência de caracteres passada como parâmetro pode ser alterada este método deve ser utilizado com muito cuidado. Na realidade o que se altera é o próprio endereço da string, que avança a cada novo termo lido dentro da cláusula.

Parâmetros: `str` — seqüência de caracteres contendo a cláusula a ser analisada.

13.2

int **nextToken** (char **str, char *name)

Parsing

Parsing. Retorna o próximo elemento dentro de uma seqüência de caracteres. Os elementos podem ser do tipo:

- TK_PREDICATE: identificador de predicados. Deve ser uma letra minúscula seguida por um abre parênteses. Exemplo, na cláusula "`p(X)`", "`p`" é o TK_PREDICATE.
- TK_VARIABLE: qualquer identificador iniciando com letra maiúscula ou *underscore* (_) entre os argumentos de um predicado.
- TK_CONSTANT: qualquer identificador iniciando com letra minúscula.
- TK_OPEN: abre parênteses.
- TK_CLOSE: fecha parênteses.
- TK_OPENLIST: abre colchete.
- TK_CLOSELIST: fecha colchete.
- TK_ERROR: símbolo não identicado.
- TK_END: final de arquivo.
- TK_COMMA: vírgula.
- TK_SEMICOLON: ponto e vírgula.
- TK_PERIOD: ponto final.
- TK_THEN: símbolo de implicação do Prolog (:-)
- TK_NULL: caracter zero representando final de string.

14

struct **result_s**

Estrutura utilizada pelo *Lex*

Members

char*	**tree**	*tree é uma seqüência de caracteres parentetizada representando a árvore sintática sendo lida. Por exemplo sn(artdef(o), sc(livro)) é um possível conteúdo para tree.*
char*	**lex**	*lex é a seqüência reconhecida como um léxico da linguagem. Se a palavra reconhecida é livro, o conteúdo de lex é "livro" também.*
term_t	**clause**	*clause é um índice para a pilha do Prolog correspondente à cláusula que representa o léxico lido. Por exemplo, quando o léxico lido é livro, clause deve apontar para uma cláusula do tipo "livro(X)", onde "X" é uma variável livre.*
term_t	**core**	*core corresponde ao núcleo da cláusula para a qual o léxico foi traduzido.*
int	**nclause**	*nclause indica o número de cláusulas que foram geradas a partir do léxico. Geralmente o número de cláusulas não passa de 1.*

Estrutura utilizada pelo *Lex.* Esta estrutura é utilizada como interface de comunicação

do *yyparse()* com o *yylex()*. Toda vez que a rotina *yylex* é chamada uma estrutura *result_s* é preenchida

15

class Anasint

Classe *Anasint*

Membros Públicos

15.1 int **readSentence** (int *n, char **sentence, istream& in)
Análise sintática 116

Classe *Anasint.* Esta classe é responsável por toda a análise sintática das sentenças em português. Utiliza como ferramentas um gerador automático de analisadores léxicos, o flex (*Fast Lexical Analizer*), e um gerador de analisadores sintáticos, o Bison (um descendente do Yacc). Estas duas ferramentas permitem que seja feita a análise de maneira simples e eficiente de estrutura sintáticas baseadas em gramáticas regulares. O fragmento do português que o sistema reconhece está definido no arquivo "anasint.y" e as palavras, ou léxicos, estão definidas no arquivo "analex.lex". Cada um deles segue uma sintaxe rígida das ferramentas utilizadas.

Uma vez que o código para fazer tanto a análise léxica quanto sintática esteja pronto, a interface utilizada é baseada em duas funções em C: o *yylex()* e o *yyparse()*. O primeiro lê uma palavra ou símbolo do arquivo de entrada e diz a qual categoria sintática ele pertence. Os casos de ambigüidade estão sendo ignorados por enquanto. O *yyparse()* analisa um sentença por inteiro e gera como resultado uma estrutura do tipo cláusula de Horn, em ASCII, que pode ser passada diretamente para o Prolog.

15.1

int **readSentence** (int *n, char **sentence, istream& in)

Análise sintática

Análise sintática. Este método lê uma sentença do arquivo especificado no parâmetro *in* e retorna a sua representação em cláusula no segundo parâmetro (*sentence*). O primeiro

parâmetro indica o número de cláusulas que foram geradas a partir da sentença em português que foi lida.

Parâmetros:

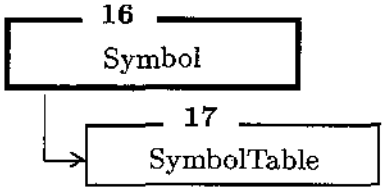
`n` — número de cláusulas geradas a partir da sentença lida.
`sentence` — um vetor alocado dinâmicamente contendo as cláusulas geradas.
`in` — ponteiro do tipo FILE contendo as sentenças que precisam ser traduzidas e analisadas.

16

class Symbol

Classe *Symbol*

Inheritance



Membros Públicos

16.3		**Symbol** (char *name, term_t term)	**Construtor**	119
16.4		**Symbol** ()	**Construtor**	119
16.5		**equals** (Symbol *symbol)	**Comparação**	120
16.6	void	**clear** ()	**Limpa**	120
16.7	void	**setName** (char *s)	**Altera**	120
	char*	**retName** ()	*retorna o nome do símbolo.*	
16.8	void	**setTerm** (term_t t)	**Atribui**	121
	term_t	**retTerm** ()	*retorna o índice na pilha do Prolog para o símbolo.*	
16.9		**~Symbol** ()	**Destrutor**	121

Membros Privados

16.1	char*	**name**	**Identificador**	121
16.2	term_t	**termReference**	**Identificador interno**	122

Classe *Symbol*. Associa um nome de uma variável livre do Prolog a um índice na pilha do módulo Prolog em execução. Esta classe é utilizada somente no objeto SymbolTable, que gerencia os símbolos na ocasião da conversão de uma sentença representada em forma de seqüência de caracteres para a forma interna do Prolog.

16.3

Symbol (char *name, term_t term)

Construtor

Construtor. Recebe o nome e um índice para o nome do símbolo.

Parâmetros: `name` — seqüência de caracteres representando o nome do símbolo.
`term` — índice na pilha do Prolog para o símbolo sendo representado.

16.4

Symbol ()

Construtor

Construtor. Apenas zera a área de memória utilizada pelo símbolo e deixa o objeto a espera de uma nova configuração.

16.5

equals (Symbol *symbol)

Comparação

Comparação. Compara dois objetos do tipo *Symbol*, retornando 1 (um) caso sejam iguais em nome e 0 (zero) caso contrário.

Parâmetros: `symbol` — objeto do tipo *Symbol* representando o símbolo a ser comparado.

16.6

void **clear** ()

Limpa

Limpa. Libera a memória ocupada pelo símbolo. O novo estado do símbolo é o mesmo no momento da sua criação.

16.7

void **setName** (char *s)

Altera

Altera. Este método pode ser utilizado tanto para alterar o nome do símbolo atual, quanto para atribuir um novo nome.

Parâmetros: s — seqüência de caracteres contendo o nome do símbolo.

16.8

void **setTerm** (term_t t)

Atribui

Atribui. Atribui um novo índice Prolog para o símbolo.

Parâmetros: `t` — índice válido para uma posição na pilha do Prolog representando o símbolo corrente.

16.9

~Symbol ()

Destrutor

Destrutor. Libera a área de memória ocupada pelo símbolo. Não altera nada na pilha do Prolog, uma vez que o gerenciamento de memória é próprio dele.

16.1

char* **name**

Identificador

Identificador. Seqüência de caracteres representando o símbolo. Exemplos: `X`, `XYZ`, `_GG66`

16.2

term_t **termReference**

Identificador interno

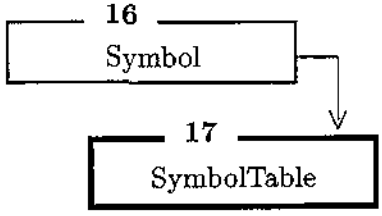
Identificador interno. Índice na pilha do módulo Prolog representando a mesma variável internamente com o nome apontado por *name*

17

class **SymbolTable** : public Symbol

Classe SymbolTable

Inheritance



Membros Públicos

SymbolTable () *construtor. Inicializa a tabela de símbolos com uma lista vazia de símbolos.*

SymbolTable (Symbol *s)
construtor. Inicializa a tabela de símbolos com o símbolo apontado por s. Parâmetros:

s: novo símbolo utilizado para criar um novo nó da tabela.

SymbolTable*

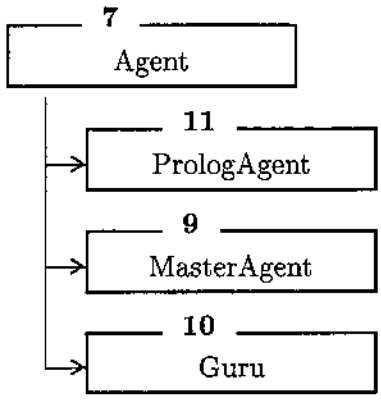
	insert (SymbolTable*)	*insere um novo nó na tabela ou retorna um apontador para o nó que contém o símbolo sendo inserido. Parâmetros:* *SymbolTable: novo nó da tabela para ser inserido*
void	**clear** ()	*elimina todos os símbolos correntes da tabela de símbolos. O seu estado a será de uma tabela que acabou de ser criada.*
	~**SymbolTable** ()	*destrutor. Libera a área de memória ocupada pela tabela e pelos símbolos que ela contém.*

Membros Privados

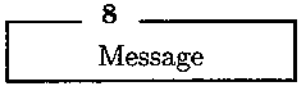
SymbolTable*	**next**	*ponteiro para fazer o encadeamento da lista*

Classe *SymbolTable*. Implementa uma tabela de símbolos utilizando uma lista simplesmente encadeada. O seu funcionamento é recursivo e não se preocupa com desempenho. Poderia ser facilmente melhorada se utilizasse uma implementação utilizando *hash* ou técnicas similares.

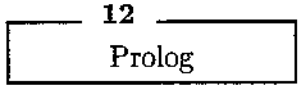
Gráfico das Classes



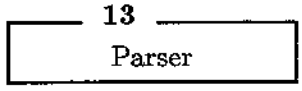
Agent 86
PrologAgent 102
MasterAgent 96
Guru 99



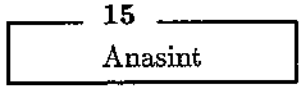
Message 92



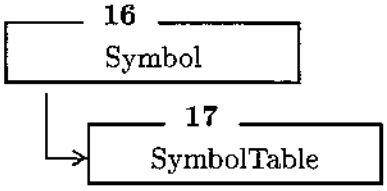
Prolog 105



Parser 110



Anasint 116



Symbol 118
SymbolTable 123

Bibliografia

[1] James Allen. *Natural Language Understanding.* Benjamin Cummings, 1988.

[2] Randall D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72(1–2):173–215, 1995.

[3] Sabine Berthet, Yves Demazeau, and Olivier Boissier. Knowing each other better. *11^{th} International Workshop on Distributed Artificial Intelligence*, Feb 1992.

[4] Olivier Boissier and Yves Demazeau. Asic: An architecture for social and individual control and its application to computer vision. *6^{th} European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Aug 1994.

[5] Francis Bond, Kentaro Ogura, and Tsukasa Kawaoka. Noun phrase reference in japanese-to-english machine translation. *Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'95)*, pages 1–14, july 1995.

[6] Noam Chomsky. *Syntactic Structures.* The Hague, Mounton, 1957.

[7] Helder Coelho, Luis Antunes, and Luis Moniz. *On Agent Design Rationale.* Technical report, INESC, 1994.

[8] Philip R. Cohen, Adam Cheyer, and Soon Cheol Baeg. An open agent architecture. Technical report, SRI International, Stanford University, ETRI, 1994.

[9] Celso Cunha and Cintra Lindley. *Nova Gramática do Português Contemporâneo.* Editora Nova Fronteira, 1985.

[10] Daniel da Silva de Paiva. Um sistema multi-agentes para processamento distribuído de linguagem natural. Master's thesis, Instituto de Computação, Unicamp, 1996.

[11] Yves Demazeau. From interactions to collective behaviour in agent-based systems. MAGMA, 1995.

[12] Yves Demazeau and Jean Pierre Müller, editors. *Decentralized A. I.: Proceedings of the First European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Cambridge, England, Aug 16-18 1989. North Holland.

[13] Edmund H. Durfee and Jeffrey S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. Technical report, EECS Department, University of Michigan; Computer Science Department, Hebrew University, 1994.

[14] Daniel Gallin. *Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics*. North-Holand Publishing Company, 1975.

[15] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format: version 3.0*, 1992. Reference Manual.

[16] Per-Kristian Halvorsen. Natural language understanding and montague grammar. *Computational Intelligence*, 2(1):54–62, 1986.

[17] John A. Hawkins. *Definiteness and Indefiniteness: A Study in Reference and Gramaticality Prediction*. Croom Helm, London, 1978.

[18] Graeme Hirst. *Anaphora in Natural Language Understanding: A Survey*. Springer-Verlag, 1981.

[19] Jerry R. Hobbs. Resolving pronoun references. In Karen Sparck-Jones Barbara J. Grosz and Bonnie Lynn Webber, editors, *Readings in Natural Language Processing*, pages 339–352. Morgan Kaufmann, Los Altos, 1978.

[20] João Luís Tavares da Silva. Utilização do paradigma multi-agentes no processamento da linguagem natural: um modelo voltado à resolução da ambigüidade léxica categorial na língua portuguesa. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 1997.

[21] Nirit Kadmon. *On Unique and Non-Unique Reference and Asymetric Quantification*. Garland, New York, 1992.

[22] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc: unix programming tools*. O'Reilly & Associates, Inc., 1992.

[23] Celso Pedro Luft. *Moderna Gramática Brasileira*. Editora Globo, 1994. 12ª edição.

[24] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of kqml as an agente comunication language. University of Maryland Blatimore Country, USA.

[25] M.C. McCord. *Natural Language Processing in Prolog.* Addison Wesley, MA, 1987. in Knowledge Systems and Prolog, A. Walker.

[26] Fernando Mouta and Eugénio Oliveira. Reasoning methodologies for decision making in intelligent multi-agent systems.

[27] Hellen C. da F. Pacheco and Mike Dillinger. Uma Nova Abordagem para a Análise Sintática do Português. In *Simpósio Brasileiro de Inteligência Artificial, Encontro para o Processamento Computacional de Português Escrito e Falado (2)*, pages 51–60, CEFET Curitiba, PR, Outubro 1996.

[28] Charles Sanders Peirce. *Semiótica.* Editora Perspectiva, 1990.

[29] Fernando C. N. Pereira and Shieber. *Prolog and Natural-Language Analysis.* Center for the Study of language and Information, Leland Stanford Junior University, 1987.

[30] Fernando C.N. Pereira and David H.D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with aumengted transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[31] Manny Rayner. *Abductive Equivalential Translation and its application to Natural Language Database Interfacing.* PhD thesis, SRI Internation Cambridge, Sep 1993.

[32] Jiří Kramský. *The Article and the concept of defineteness in language.* Mount, Paris, 1972.

[33] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[34] R.J.II. Scha. *Logical Foundations for Question Answering.* PhD thesis, University of Groningen, Netherlands, 1983.

[35] Gabriele Scheler. *36 Problems for Semantic Interpretation.* Technical report, Institut für Informatik, Technische Universität München, 1989.

[36] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, Cambridge, 1969.

[37] Yoav Shoham and Moshe Tenneholtz. On social laws for artificial agent socities. *Artificial Intelligence*, 72(1–2):231–252, 1995.

[38] Candace L. Sidner. The use of focus as a tool for disambiguatin of definite noun phrase. In *Proceedings of Second Conference on Theoretical Issues in Natural Language Processing - TINLAP.* University of Illinois at Alabama-Champaign, 1978.

[39] Candace L. Sidner. Focusing in the comprehension of definite anaphora. In Karen Sparck-Jones Barbara J. Grosz and Bonnie Lynn Webber, editors, *Readings in Natural Language Processing*, pages 363–394. Morgan Kaufmann, Los Altos, 1983.

[40] Marie-Hélène Stefanini and Yves Demazeau. *Talisman: A Multi-Agent System for Natural Language Processing. XII Simpósio Brasileiro em Inteligência Artificial*, 1995.

[41] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[42] Gertjan van Noord. The intersection of finite state automata and definite clause grammar. Technical report, Vakgroep Alfa-Informática & BCN, Tijksuniversiteit Groningen, s.d.

[43] Yorick Wilks and A. Herskovits. An intelligent analyser and generator for natural language. In *Proc. International Conference on Computational Linguistics*, Pisa, Italy, 1973.

[44] Michael J. Wooldridge and Nicholas R. Jennings. *Agent Theories, Architecture, and Languages: A Survey. ECAI*, 1994.