EDUARDO CARLOS HAMERSKI JÚNIOR

UM SISTEMA INTELIGENTE PARA A ALOCAÇÃO DISTRIBUÍDA DE TAREFAS E BALANCEAMENTO DE CARGA

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: *Sistemas Inteligentes*

Orientador: Prof. Dr. Edson Emílio Scalabrin
Co-orientador: Prof. Dr. Bráulio Coelho Ávila

CURITIBA
2003

Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia

ATA DA SESSÃO PÚBLICA DE DEFESA DE DISSERTAÇÃO DE MESTRADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

DEFESA DE DISSERTAÇÃO Nº 074

Aos 19 dias do mês de fevereiro de 2003 realizou-se a sessão pública de defesa da dissertação **"Um Sistema Inteligente para Alocação Distribuída de Tarefas e Balanceamento de Carga"**, apresentada por **Eduardo Carlos Hamerski Jr.,** como requisito parcial para a obtenção do título de **Mestre em Informática Aplicada**, perante uma Banca Examinadora composta pelos seguintes membros:

	assinatura	parecer (aprov/ reprov.)
Prof. Dr. Edson Emílio Scalabrin PUCPR (Orientador)		Aprovado
Prof. Dr. Bráulio Coelho Ávila PUCPR		Aprovado
Prof. Dr. Hilton José S. de de Azevedo CEFET-PR		APROVADO.

Conforme as normas regimentais do PPGIA e da PUCPR, o trabalho apresentado foi considerado Aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora, conforme registrado no Livro de Defesas do programa.

Prof. Dr. Carlos Maziero
Diretor do PPGIA PUCPR 30/5/03
Data e assinatura, após homologação da defesa pelo colegiado.

Programa de Pós-Graduação em Informática Aplicada
PPGIA
PUCPR

Rua Imaculada Conceição, 1155 - Prado Velho - CEP 80215-901 - Cx. Postal 16.210 - CEP 81611-970
Telefone: (41) 330-1513 - Fax (41) 330-1620 - http://www.pucpr.br - Curitiba - Paraná - Brasil

À minha esposa e filho,

pelo amor, carinho e compreensão.

Aos meus pais e irmãos, pelos mesmos motivos.

Agradecimentos

Ao orientador professor Edson Emílio Scalabrin, pelo apoio e incentivo constantes e principalmente por sua paciência e amizade sincera.

Ao amigo professor Marcos A. H. Shmeil, pelo apoio e incentivo constantes.

Aos professores Bráulio Coelho Ávila e Hilton J. de Azevedo por terem aceitado fazer parte da banca de avaliação e defesa deste trabalho.

Aos professores Marco Cândido e Robert Carlisle Burnett pela oportunidade de participar dos projetos SCI e HP e principalmente pela grande ajuda que me prestaram. Agradeço também à Lourdes, secretária do ICET, pela paciência e ajuda. Aos amigos que fiz durante este período: Aron, Leonardo e Matias (SCI); Thiago e Leonardo (HP); e ao professor Marcos Paludo, pela constante preocupação e apoio. Ao amigo Paulo Vinicius por ter me indicado para participar do projeto HP e também por sua preocupação, apoio e incentivo constantes.

À amiga Elaini, pela oportunidade de dar continuidade ao seu trabalho.

Aos professores Cinthia e Luiz Eduardo pela ajuda e conhecimentos compartilhados.

Aos amigos Fernanda, Denes, Thieme, Attílio, Paulo Vinicius, Fabrício, Emerson, Airo, Elio, Cláudio Oliveira, Cláudio Carvilhe, Orlando e tantos outros, pela companhia, apoio e incentivo. Pelas vezes que alguns me socorreram e tanto mais.

À minha família, que me tolerou durante os momentos de nervosismo e impaciência. Pelo que representam em minha vida e pelo orgulho e satisfação que me trazem.

Minhas mais sinceras desculpas a quem aqui, esqueci de citar. Todos os que colaboraram com este trabalho, direta ou indiretamente, sabem que serei sempre grato.

Ao PPGIA e a PUCPR pela oportunidade e por parte do apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Motivado pela proposta *Multicheck* [SCA98] que rendeu também outros trabalhos relacionados às áreas de Inteligência Artificial Distribuída e Processamento de Imagens, este trabalho também define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. Partindo da escolha de uma arquitetura distribuída e aberta, com critérios que facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, este trabalho implementa um ambiente de software que facilita a criação, manutenção (mobilidade, entrada e saída) e persistência de agentes cognitivos, assim como implementa uma aplicação teste para o processamento de imagens reais empregando a Lógica Paraconsistente para representar e raciocinar sobre informações contraditórias trocadas entre agentes. O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

Palavras-chave: Alocação Distribuída de Tarefas, Balanceamento de Carga, Sistemas Multi-Agente, Lógica Paraconsistente.

Resumo

Motivado pela proposta *Multicheck* [SCA98] que rendeu também outros trabalhos relacionados às áreas de Inteligência Artificial Distribuída e Processamento de Imagens, este trabalho também define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. Partindo da escolha de uma arquitetura distribuída e aberta, com critérios que facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, este trabalho implementa um ambiente de software que facilita a criação, manutenção (mobilidade, entrada e saída) e persistência de agentes cognitivos, assim como implementa uma aplicação teste para o processamento de imagens reais empregando a Lógica Paraconsistente para representar e raciocinar sobre informações contraditórias trocadas entre agentes. O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

Palavras-chave: Alocação Distribuída de Tarefas, Balanceamento de Carga, Sistemas Multi-Agente, Lógica Paraconsistente.

Abstract

Motivado pela proposta *Multicheck* [SCA98] que rendeu também outros trabalhos relacionados às áreas de Inteligência Artificial Distribuída e Processamento de Imagens, este trabalho também define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. Partindo da escolha de uma arquitetura distribuída e aberta, com critérios que facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, este trabalho implementa um ambiente de software que facilita a criação, manutenção (mobilidade, entrada e saída) e persistência de agentes cognitivos, assim como implementa uma aplicação teste para o processamento de imagens reais empregando a Lógica Paraconsistente para representar e raciocinar sobre informações contraditórias trocadas entre agentes. O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

Keywords: Alocação Distribuída de Tarefas, Balanceamento de Carga, Sistemas Multi-Agente, Lógica Paraconsistente.

Abstract

Motivado pela proposta *Multicheck* [SCA98] que rendeu também outros trabalhos relacionados às áreas de Inteligência Artificial Distribuída e Processamento de Imagens, este trabalho também define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. Partindo da escolha de uma arquitetura distribuída e aberta, com critérios que facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, este trabalho implementa um ambiente de software que facilita a criação, manutenção (mobilidade, entrada e saída) e persistência de agentes cognitivos, assim como implementa uma aplicação teste para o processamento de imagens reais empregando a Lógica Paraconsistente para representar e raciocinar sobre informações contraditórias trocadas entre agentes. O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

Keywords: Alocação Distribuída de Tarefas, Balanceamento de Carga, Sistemas Multi-Agente, Lógica Paraconsistente.

Capítulo 1

Introdução

Este trabalho insere-se no contexto do Projeto *Multicheck*[1], o qual define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques brasileiros manuscritos. Deve-se destacar que este trabalho é uma continuação do já realizado em [ANG01]. No contexto deste projeto a escolha de uma arquitetura distribuída e aberta baseou-se nos seguintes critérios: (i) trata-se de um problema cuja decomposição em sub-problemas é natural, à medida que cada sub-problema (reconhecimento dos montantes numérico e literal, reconhecimento da data e verificação da assinatura) pode ser representado e implementado como um agente autônomo e independente; (ii) a interação entre os agentes pode tornar o sistema de compensação de cheques mais confiável, à medida que as trocas de informações entre os agentes podem resolver situações difíceis se considerarmos também as informações não locais, por exemplo, os agentes que reconhecem os valores numéricos e extensos dos cheques trocam suas crenças para confirmar ou não suas hipóteses de reconhecimento; e (iii) o paralelismo natural dos sistemas multi-agente pode contribuir de forma considerável na implementação de um sistema de alta performance. Deve-se enfatizar que tais critérios, no contexto desta aplicação, facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, que são os princípios básicos para se construir sistemas eficientes e de fácil manutenção.

[1] O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

A arquitetura definida para o Projeto *Multicheck* inclui basicamente cinco grandes tarefas, as quais serão distribuídas/implementadas sobre agentes diferentes. Tais tarefas são [ANG01]:

- Localizar/extrair a estrutura lógica de um cheque, ou seja, obter os campos de identificação do banco, da agência, do emissor, dos montantes numérico e literal, da data e da assinatura;
- Interpretar o montante numérico;
- Interpretar o montante por extenso;
- Reconhecer a data; e
- Verificar a assinatura.

As competências associadas à interpretação de um campo lógico foram implementadas em duas camadas, que são:

- A primeira camada corresponde aos algoritmos de reconhecimento de padrões (redes neurais, cadeias de *Markov*), que são aplicados diretamente sobre segmentos de imagens de um cheque;
- A segunda camada corresponde aos mecanismos de inferência aplicados para interpretar os dados gerados pela primeira camada no intuito de validá-los. Tal mecanismo implementa um conjunto de operadores da lógica paraconsistente.

Do ponto de vista de aplicação, o esforço aqui será calcado sobre a segunda camada[2] e tem por objetivo montar um protótipo de um sistema aberto que executará uma base de dados real para validação dos algoritmos.

A proposta tem sua fundamentação nos sistemas distribuídos, na lógica paraconsistente e no conceito de agente desenvolvido pela comunidade de inteligência artificial distribuída. "Esta união permite a execução de tarefas interativas de forma cooperativa, mesmo na presença de informações inconsistentes, na tentativa de alcançar um objetivo comum (e.g., interpretar as informações presentes em um cheque)" [ANG01].

[2] O trabalho de Elaini S. Angelotti concentrou-se também sobre a segunda camada, porém os dados utilizados eram fruto de uma simulação.

1.1. Motivação

O processamento de imagens, além de requerer um cuidado especial para superar a complexidade natural de uma aplicação real, é um grande consumidor de tempo de CPU. Tal complexidade se apresenta na forma de uma grande diversidade de conhecimentos e dificuldade em mantê-los. Ela é em si mesma uma área de grande atratividade no tocante a concepção e implementação de algoritmos para tratamento de grandes volumes de dados. Por exemplo, a automação do processo de compensação de cheques manuscritos requer um grande número de tarefas bem definidas, porém de níveis elevados de complexidade. As principais tarefas são [ANG01]: (i) aquisição de imagens; (ii) eliminação de informações irrelevantes presentes no cheque (fundo artístico, sobreposição de linhas, etc); (iii) localização e extração de informações relevantes (identificação do banco, da agência, do emissor, dos montantes numérico e literal, da dada e da assinatura); (iv) obtenção da estrutura lógica do documento; (v) separação do pré-impresso em relação ao manuscrito; (vi) segmentação de cada um dos campos lógicos; (vii) interpretação dos dados lógicos (reconhecimento dos montantes numérico e literal, reconhecimento da data e verificação da assinatura); e (viii) análise para aceitação ou rejeição de um determinado cheque. Claramente, estas diferentes tarefas são bem definidas, porém a solução computacional requer uma grande quantidade de recursos de processamento. Este grande volume de conhecimento e recursos necessários indicam que uma solução monolítica tem poucas chances de sucesso. Uma abordagem verdadeiramente distribuída tem maiores chances de sucesso por levar em conta três fatores: projeto modular, implementação distribuída e paralelismo entre tarefas independentes.

As dificuldades naturais de uma aplicação da área de processamento de imagens nos permitem acreditar que uma abordagem modular é a solução mais apropriada para vencer as tarefas já enumeradas. Estas tarefas são bem definidas e independentes, porém a troca de informações entre elas durante suas resoluções pode aumentar sobremaneira a eficiência do sistema de reconhecimento global. O conceito de agente cognitivo oferece um campo fértil para definir uma arquitetura de sistema distribuída, onde suas entidades cooperam visando o cumprimento de objetivos comuns (reconhecer o montante numérico, reconhecer o montante literal).

A existência de tarefas independentes e cooperativas pode facilmente gerar informações inconsistentes. Semelhante situação requer, além de um modelo de agente sofisticado para decidir como, quando e com quem comunicar, mecanismos para representar e

raciocinar sobre informações contraditórias. Tais informações podem ser tratadas de forma natural e simples pela Lógica Paraconsistente, notadamente onde as lógicas clássicas não apresentam bons resultados [AVI97], [COS90], [COS63], [SUB87b].

Validar um cheque requer um conjunto de iterações e interações. A seqüência de operações para verificação de um cheque pode ser a seguinte: reconhecimento e validação dos montantes numérico e literal, reconhecimento e validação da data e verificação da assinatura. Esta forma de proceder nos parece bem adequada supondo que as operações e os fluxos das mesmas são semelhantes os realizados por um ser humano. Do ponto de vista computacional, tais tarefas podem ser cumpridas em paralelo e cooperativamente pelas capacidades e competências de um sistema de agentes desenvolvidos para este fim. A cooperação é colocada em prática por meio da troca de informações e compromissos. Tal iniciativa fará confrontar diferentes visões sobre um mesmo objeto, gerando contradições. A representação e raciocínio sobre tais contradições são implementados através da Lógica Paraconsistente.

1.2. Proposta

O objetivo deste trabalho é duplo. O primeiro consiste em projetar e implementar um ambiente de software que facilite a criação, a manutenção (mobilidade, entrada e saída de agentes) e a persistência de agentes cognitivos. O segundo é implementar uma aplicação teste para o processamento de imagens reais, empregando a Lógica Paraconsistente para representar e raciocinar sobre informações contraditórias trocadas entre os agentes. Para alcançar estes dois grandes objetivos, definiu-se um conjunto de objetivos específicos:

- Estudar diferentes formas de organizar uma rede de agentes que deverão distribuir dinamicamente tarefas entre si;
- Definir e implementar procedimentos para a gestão da inclusão e remoção dinâmica de agentes de um sistema, ou seja, permitir a criação de sistemas abertos;
- Definir e implementar procedimentos que permitam ao sistema gerenciar automaticamente o balanceamento de carga;
- Mostrar o uso da Lógica Paraconsistente no contexto de uma aplicação na área de imagens com dados reais;
- Criar um ambiente de software para administrar os agentes. As funções principais serão: instalar, destruir, suspender, executar e mover um agente sobre uma rede de computadores.

O cumprimento destes objetivos será alcançado por meio da construção de um protótipo de um sistema multi-agente. Um ponto a enfatizar é a definição de um agente especial que auxiliará no processo de balanceamento de carga do sistema. Tal agente desempenhará esta tarefa acrescentando e removendo agentes automaticamente em função da necessidade de uma competência. Deve-se destacar que poderão ser adicionados vários agentes com a mesma competência simplesmente no intuito de aumentar o poder cálculo.

1.3. Organização

Este documento está organizado em quatro capítulos e um apêndice:

- O capítulo 2 apresenta uma visão geral sobre a alocação de tarefas em um universo multi-agente, notadamente as formas de alocação por delegação, rede relacionamentos (*acquaintances*), chamada de ofertas ou rede contratual.
- O capítulo 3 examina a Lógica Evidencial Paraconsistente, sua sintaxe e alguns operadores que foram utilizados na realização deste trabalho.
- O capítulo 4 apresenta o processamento de imagens de cheques e os resultados obtidos respectivamente.
- Na seqüência, são apresentados as conclusões e trabalhos futuros.
- Finalmente, é apresentada a descrição da arquitetura do ambiente de suporte proposto.

Capítulo 2

Alocação de Tarefas em um Universo Multi-Agente

A cooperação fornece vantagens em termos de eficiência quantitativa e de emergência qualitativa [FER95], capítulo 2, porém traz alguns problemas difíceis de distribuição do trabalho entre os agentes. Tal dificuldade surge devido à presença de vários parâmetros, tais como capacidades cognitivas, capacidades de comprometimento, competências individuais, natureza das tarefas, eficiência, custo de transmissão de uma informação, estruturas sociais, dentre outros.

A distribuição de tarefas e de recursos constitui ao mesmo tempo um dos domínios mais importantes dos sistemas multi-agente e uma de suas principais contribuições à informática em geral [FER95]. Elevando-se o nível de importância sobre a alocação distribuída de tarefas e sobre as noções de contrato e comprometimento, os sistemas multi-agente definem o problema da atividade em termos social e computacional. Isto que os distingue de formas mais clássicas[3].

Quando o problema é distribuir tarefas, informações e/ou recursos à questão clássica é: quem deve fazer o que e com quais recursos. Esta atividade é feita em função dos objetivos e competências dos agentes e das restrições contextuais (tipos e qualidades de recursos). Trata-se aqui de um dos pontos essenciais da função organizacional de uma sociedade de agentes juntamente com a coordenação de ações, segundo o autor supracitado.

Serão apresentados, na seqüência, alguns dos princípios para a colaboração da alocação dinâmica de tarefas em um ambiente distribuído. Os temas abordados serão:

[3] Problemas, por exemplo, de distribuição de processos sobre processadores em uma máquina. Muitas técnicas de alocação de tarefas são, em geral, fundamentadas em resultados obtidos no domínio da pesquisa operacional e dos sistemas distribuídos.

(i) formas de alocação de tarefas; (ii) alocação centralizada de tarefas; (iii) alocação distribuída de tarefas; e (iv) alocação de tarefas e tratamento de informações inconsistentes.

As representações gráficas necessárias para a compreensão e formalização de papeis (e.g., cliente, mediador, fornecedor) serão emprestadas de [FER95]. Este último desenvolveu uma forma de representação chamada BRIC (módulo) que facilita a definição dos componentes de um sistema complexo. Ela tem, na sua forma gráfica de representação, grandes similaridades no tocante a representação de componentes e suas interfaces com aquelas utilizadas na área de eletrônica. Combinado a esta representação é utilizado uma rede de *Petri* para formalizar o comportamento interno de cada componente. É importante destacar que os diferentes protocolos apresentados neste capítulo foram formalizados pelo autor supracitado e, no contexto deste trabalho, tentar-se-á retomá-los e representá-los da forma mais fiel possível.

2.1. Alocação de Tarefas

A alocação de tarefas é a forma clássica de organização do trabalho colaborativo. Ela passa pela definição de um conjunto de mecanismos organizacionais que os agentes utilizam para articular suas competências no intuito de realizar um trabalho coletivo. Segundo [FER95], o problema em questão é: como descrever a forma de alocar as tarefas sabendo que as capacidades de um agente dependem ao mesmo tempo de suas aptidões intrínsecas (arquitetura, capacidade cognitiva, tipo de comunicação), dos meios energéticos que ele dispõe (tempo de cálculo e autonomia energética), de recursos externos (ferramentas, fontes enérgicas) e restrições ambientais.

Um ponto importante sobre a alocação de tarefas é a decomposição de uma tarefa complexa T em sub-tarefas {T1, ... , Tn}. Uma tarefa complexa T caracteriza-se aqui por uma situação em que se requer mais recurso, trabalho ou expertise que um único agente pode fornecer. Assim, tal tarefa deve ser, em primeiro momento, decomposta em varias sub-tarefas e depois alocadas a diferentes agentes. Estas duas operações estão fortemente ligadas. Semelhante afirmação baseia-se no fato de que a decomposição de uma tarefa deve levar em conta as competências dos agentes para facilitar a alocação que se segue.

2.1.1. Decomposição de Tarefas

Pode-se encontrar, na literatura da área, diferentes critérios para decompor um problema complexo em subproblemas mais simples. A decomposição é particularmente

indicada para problemas que podem ser naturalmente analisados em níveis de abstração (do mais geral ao mais específico). Esta atividade requer também que se leve em conta restrições tais como: controle, dados e/ou recursos [FOX88]. Em outras palavras, busca-se tornar os problemas o mais independentes possível uns dos outros de modo a reduzir ao máximo à coordenação. Ou seja, tenta-se minimizar a quantidade de informações que as tarefas devem comunicar e fazer com que tais tarefas utilizem os recursos locais para diminuir os conflitos sobre os mesmos. Em resumo, o que se procura é minimizar o acoplamento e maximizar a coesão entre tarefas.

A atividade de decomposição de um problema em subproblemas é tão importante quanto à alocação dos mesmos sobre um conjunto de máquinas diferentes. Em geral, a decomposição é uma tarefa exclusivamente dos seres humanos. Segundo [FER95] ainda não existe nenhum sistema de computação autônomo de decomposição de tarefas em sub-tarefas. Tratando-se de pesquisa, os grandes esforços concentram-se sobre a alocação automática de tarefas. Em [FER95], capítulo 3, pode-se encontrar diferentes maneiras de abordar a decomposição de tarefas baseando-se nas perspectivas objeto e funcional.

2.1.2. Principais Papéis: Cliente, Mediador e Fornecedor

Em um sistema de alocação automática de tarefas, os papeis que os agentes podem assumir, em geral, são dois: cliente e fornecedor. O agente que necessita de uma informação ou deseja que uma tarefa seja realizada assume o papel de cliente ou requerente. O agente que fornece um serviço assume o papel de fornecedor ou servidor. Complementarmente, o sistema de alocação automática de tarefas tem por função e capacidade colocar em relação clientes e fornecedores. Deve-se notar que um agente pode ao mesmo instante assumir os papeis de requerente e fornecedor. Em geral, estes papeis são determinados de maneira dinâmica durante a execução do sistema multi-agente. Dizemos então que o agente assume o papel de cliente e/ou fornecedor, sem que o papel seja uma de suas características intrínsecas.

2.1.3. Formas de Alocação

Existem duas formas de gerenciar a distribuição de tarefas. A primeira seria centralizando o processo de alocação e a segunda, mais complexa, consistiria na distribuição de tal processo sobre um conjunto de agentes específicos.

2.2. Alocação Centralizada de Tarefas

Neste modo de alocação há dois casos a serem examinados:

Caso 1: Se a estrutura de subordinação é hierárquica então é o superior que solicita de forma direta a um subordinado para fazer sua tarefa.

Caso 2: Se a estrutura é igualitária então a alocação passa pela definição de agentes especiais, os mediadores, que gerenciam todo processo de alocação.

No caso 1, tem-se uma alocação rígida ou bem definida. Este modo de alocação é característico na chamada de procedimento da programação clássica de computadores. Um dos inconvenientes deste tipo de alocação é que ele é adequado apenas para organizações fixas ou não dinâmicas. No caso 2, tem-se uma alocação mais dinâmica. O agente mediador centraliza as solicitações dos clientes e as ofertas de serviços dos servidores no intuito de colocá-los em correspondência de forma dinâmica. Pode-se assim aplicar formas centralizadas de alocação de tarefas a organizações variáveis.

2.2.1. Alocação Centralizada de Tarefas por Mediador

O caso mais simples envolve apenas um agente mediador. Ele dispõe de uma tabela de relacionamentos. Esta tabela indica o conjunto de agentes capazes de executar uma tarefa T. A atualização desta tabela pode ser feita de forma direta pelos fornecedores que entram no sistema e informam suas competências.

A Figura 2.1 ilustra o funcionamento do processo envolvendo os papeis cliente, mediador e fornecedor, assim como os atos de linguagem comunicados. Se um agente A possui uma tarefa T para executar e ele não pode fazê-la por uma dada razão, então ele deve solicitar ao agente mediador M para encontrar um outro agente para realizá-la.

O mediador M se endereçará então ao conjunto de agentes que ele conhece e que têm a capacidade requerida para fazer a tarefa T. Se um agente dentre o conjunto de relacionamentos aceita o trabalho, então M enviará a informação de aceite ao cliente, senão M informa ao agente cliente A que não foi possível encontrar alguém para executar T.

Supõe-se aqui que um agente fornecedor B que aceita executar uma tarefa T compromete-se efetivamente em fazê-la. Faz-se também a hipótese que o mediador M possui uma tabela contendo dados que descrevem as competências dos agentes do sistema e que tal

conteúdo é completo e coerente. Assim, todos os agentes que possuem uma competência são referenciados nesta tabela e tais referências existem efetivamente.

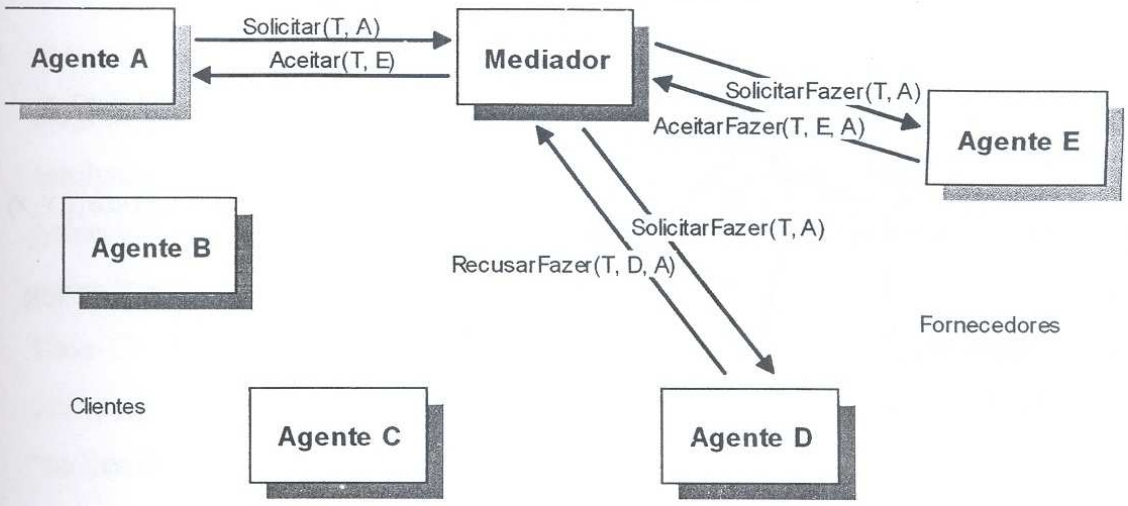

Figura 2.1: Alocação centralizada de tarefas.

Para efeito de simplicidade, apresentar-se-á apenas o modelo do mediador M. Este módulo toma como entrada as solicitações dos clientes e produz como saídas: requisições diretas aos fornecedores capazes de atender tais solicitações e as respectivas respostas aos clientes. Em [FER95], as comunicações entre o mediador, os clientes e os fornecedores são expressas por meio de um conjunto de mensagens que correspondem aos seguintes atos de linguagem:

- **Solicitar(T, X)** é uma solicitação do cliente **X** ao mediador **M** para realizar a tarefa **T**.
- **Aceitar(T, Y)** é a resposta do mediador **M** ao agente cliente **X** para informar que o agente fornecedor **Y** compromete-se em fazer a tarefa **T**.
- **Impossível(T)** é uma resposta do mediador **M** ao agente cliente **X** para informar que não há nenhum agente que aceita fazer a tarefa **T**.
- **SolicitarFazer(T, X)** é uma solicitação do mediador **M** para um fornecedor **Y** fazer a tarefa **T** para um cliente **X**.
- **AceitarFazer(T, Y, X)** é a resposta positiva do fornecedor **Y** à solicitação do mediador **M**. O fornecedor **Y** informa ao cliente **X** que ele aceita fazer a tarefa **T**.

- **RecusarFazer(T, Y, X)** é a resposta negativa do fornecedor **Y** à solicitação do mediador **M**. O fornecedor **Y** informa ao cliente **X** que ele não aceita fazer **T**.

A Figura 2.2 mostra a representação do mediador centralizada em BRIC.

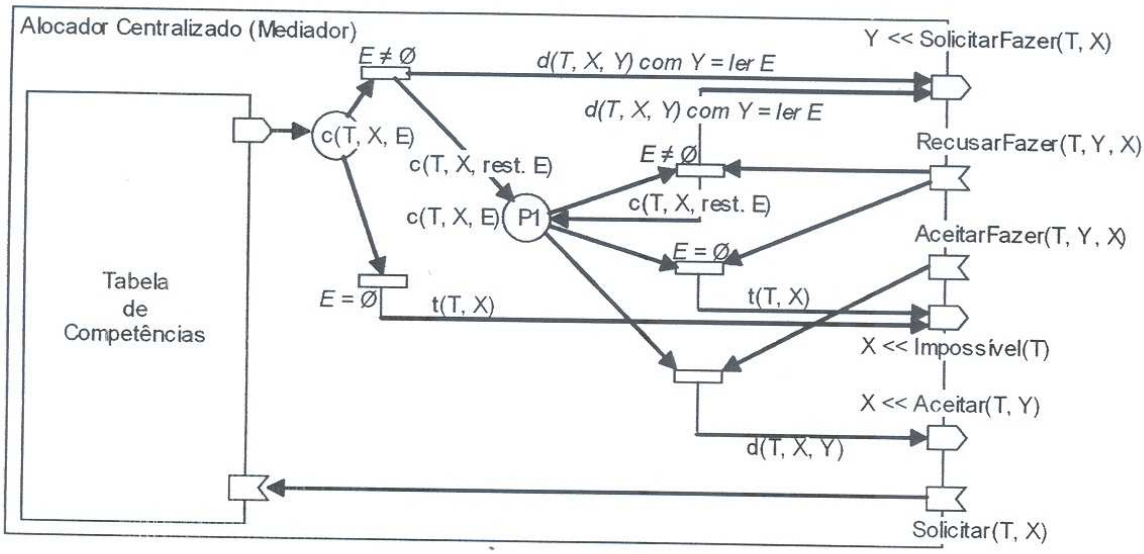

Figura 2.2: Descrição de um mediador.

Quando o mediador M recebe de um cliente X uma solicitação para fazer uma tarefa T ele lança o seguinte processo: recuperar a lista de agentes capazes de efetuar a tarefa T. Se o conjunto de agentes E é vazio, então o mediador M informa o fracasso da tentativa ao cliente X por meio da mensagem Impossível(T). Senão, o mediador M solicita de forma iterativa a todos os agentes de E se um deles pode fazer a tarefa T. Para isto, utiliza-se uma memória P1 que registra todos os fornecedores a serem consultados. Se um dos agentes aceita, então P1 é liberado e o mediador M responde ao cliente X informando que o fornecedor Y aceita fazer a tarefa T. As estruturas de dados utilizadas são as seguintes:

- **c(T, X, E)** retorna a lista dos agentes candidatos E que possui a competência para fazer a tarefa T para o cliente X. 1er E retorna o primeiro elemento da lista e rest.E retorna a lista E privada de seu primeiro elemento.
- **d(T, X, Y)** é uma solicitação do mediador **M** ao fornecedor **Y** para ele fazer a tarefa **T** para o cliente **X**.
- **t(T, X)** tarefa **T** solicitada pelo cliente **X**.

Pode-se melhorar tal procedimento otimizando o processo de seleção, o qual buscará prioritariamente os agentes mais competentes ou que minimizam uma função de custo. Dois casos podem ser vislumbrados:

Caso 1: se o mediador M conhece as funções de avaliação utilizadas pelos fornecedores, então ele pode selecionar os agentes competentes em função dos resultados da avaliação. O protocolo não é modificado; supõe-se apenas que o gerenciador de relacionamentos do mediador M retorna o conjunto de fornecedores potenciais por ordem de preferência.

Caso 2: se o mediador M não conhece as funções de avaliação, então ele deve primeiramente solicitar para cada agente fornecer sua proposta antes escolher a "melhor"[4].

O interesse da alocação centralizada de tarefas é poder manter atualizado o conjunto de agentes e suas respectivas competências, bem como favorecer a coerência do sistema. Além disso, as necessidades em otimização são mais facilmente satisfeitas, ou seja, pode-se determinar de forma relativamente simples o melhor agente para uma determinada de tarefa.

O maior inconveniente de um sistema de alocação centralizada de tarefas é que ele constitui um gargalo de estrangulamento e diminui consideravelmente a performance do sistema quando o número de agentes e solicitações aumenta. Em [FER95], analisa-se tal situação e conclui-se que o número de mensagens que um mediador M deve gerir é considerável. Por exemplo, para 50 agentes o número de mensagens processadas pelo mediador M é da ordem de 500, para 100 agentes são 2.000. Isto indica que a alocação centralizada não é aconselhada para um grande número de agentes. Além disso, se o sistema é distribuído, todas as mensagens devem passar pelo mediador sem levar em conta a topologia da rede.

Finalmente, um sistema centralizado não é tolerante a falha. Por exemplo, se o mediador M falha todo o sistema está comprometido. Deve-se então prever mecanismos complexos de tolerância à falha. É possível, evidentemente, melhorar esta situação utilizando vários mediadores. Porém deve-se gerir a coerência entre os diferentes mediadores, o que se torna algo complicado. Sendo assim, quando se deseja que o sistema possa distribuir seus

[4] Esta técnica assemelha-se a um mecanismo de chamada de ofertas.

mecanismos de alocação de tarefas, deve-se passar para um modo de alocação verdadeiramente distribuído de tarefas.

2.3. Alocação Distribuída de Tarefas

Neste modo de alocação, cada agente deve individualmente obter os serviços dos fornecedores que serão úteis para a realização de seus projetos. Distinguem-se dois mecanismos distribuídos nas organizações pré-definidas, que são: as redes de relacionamentos (ou *acquaintances*) e as redes contratuais. Uma rede de relacionamentos supõe que os clientes possuem uma representação dos outros agentes e de suas capacidades. Nós veremos que não é necessário que os agentes conheçam as capacidades de todos os outros agentes para poder resolver seu problema, porém que a rede formada pelo conjunto dos relacionamentos seja fortemente conexa (ou seja, que exista um caminho indo de qualquer agente para qualquer outro agente) e evidentemente coerente (ou seja, que as representações sobre as competências dos relacionamentos de um agente corresponda de forma precisa às competências efetivas dos agentes). Uma rede contratual[5] ou de alocação por chamada de oferta apresenta a vantagem de oferecer uma grande dinamicidade e fácil implementação. Estes modos se aplicam eficazmente a organizações variáveis, onde se supõe que as ligações entre os clientes e fornecedores podem evoluir no tempo, sem destruir a estrutura geral da organização.

Como já foi apresentado, existem vários modos de alocação de tarefas, porém pode-se vislumbrar inúmeras variantes destas abordagens e, em particular, abordagens híbridas que tentam sintetizar as vantagens de diferentes abordagens.

2.3.1. Alocação por Rede de Relacionamentos

Supõe-se aqui que cada agente dispõe de uma tabela de competências dos agentes que ele conhece. Esta tabela é representada na forma de um dicionário {C1 : [A11,...,A1k],..., Cn : [An1,...,Anl]}, onde os Ci são as competências requisitadas para realizar uma tarefa Ti e os Aij são os agentes que sabem fazer tais tarefas. [FER95] representa esta tabela na forma de uma matriz local para cada agente, onde as linhas armazenam as competências e as colunas seus relacionamentos, i.e., os agentes que ele conhece. Cada entrada da matriz pode receber 0 ou 1, onde 0 indica ausência e 1 presença de uma competência. Por exemplo, o agente A, que

[5] As redes contratuais são bastante conhecidas em inteligência artificial distribuída.

sabe que ele possui a competência C3, que a competência C1 está disponível em B e C e que D sabe fazer C3, possui a seguinte tabela de competências:

Tabela 2.1: Tabela de Competências do Agente A.

	A	B	C	D
C1	0	1	1	0
C2	0	0	1	0
C3	1	0	0	1

Cada agente conhece apenas um conjunto das competências dos outros agentes. Ou seja, deve-se respeitar a hipótese da representação parcial dos sistemas multi-agente. A não observância de tal hipótese restringiria a realização e a atualização de tais sistemas. Supõe-se apenas que as tabelas de relacionamentos são corretas, i.e., que suas indicações correspondem efetivamente às reais competências dos agentes. Faz-se também a hipótese que as tabelas de relacionamentos de cada um dos agentes são definidas e não são atualizadas. A atualização das tabelas será visto mais adiante.

Uma rede de relacionamentos pode ser representada sob a forma de um grafo. Os agentes são os vértices e os arcos representam as competências que os agentes conhecem dos outros agentes (Figura 2.3). A tabela acima corresponde apenas o vértice que representa o agente A.

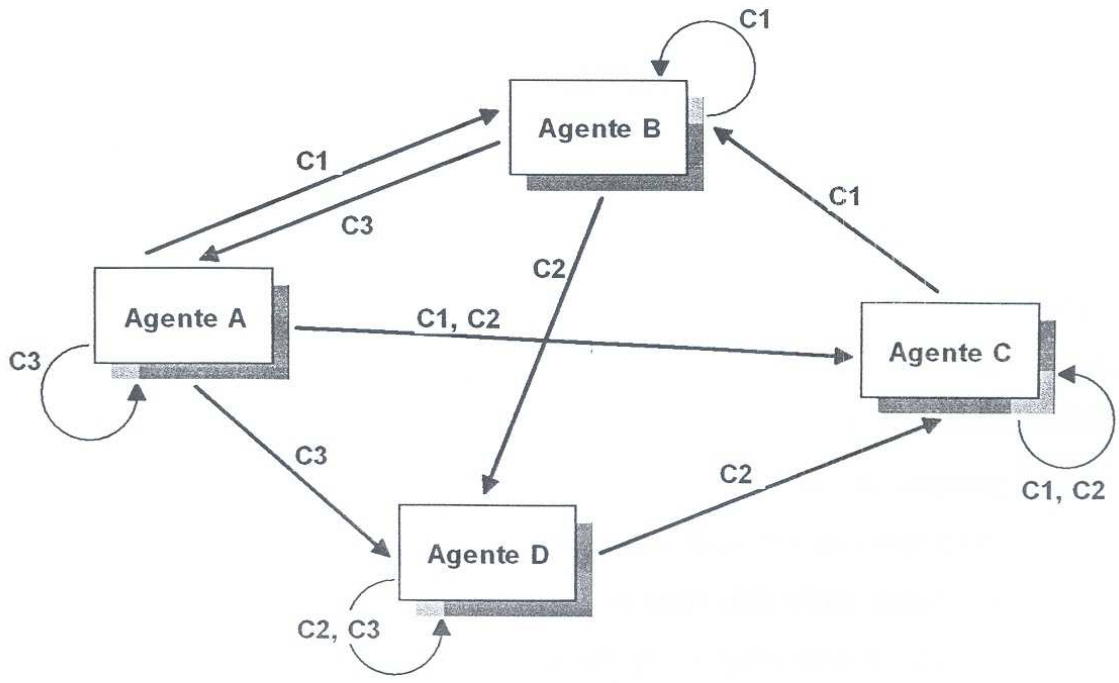

Figura 2.3: Um rede de relacionamentos representada na forma de um grafo.

Há, essencialmente, dois modos de alocação por redes de relacionamentos: as redes que autorizam os agentes a delegarem suas solicitações a outros agentes e as que não autorizam. Fala-se então de alocação por delegação ou direta.

Alocação Direta

Aqui um agente cliente A pode fazer um agente fornecedor B executar uma tarefa T se A tem acesso ou conhece diretamente B. O mecanismo de distribuição é bastante simples. Ele assume um comportamento idêntico do mediador M no caso das alocações centralizadas. Um agente cliente A que deseja encontrar um agente fornecedor B para fazê-lo executar uma tarefa T verifica cada um de seus relacionamentos que possuem a competência para fazer T até que algum agente fornecedor aceite o trabalho. Caso nenhum agente fornecedor aceite fazer T, pode-se então considerar que se trata de um defeito do sistema, porém:

> **Caso 1:** deve-se assegurar que em todas as situações existirá pelo menos um agente que aceitará efetuar a tarefa T; ou
>
> **Caso 2:** deve-se adotar medidas para direcionar a tarefa T a outro agente, atribuindo autoritariamente esta tarefa a um agente fornecedor ou relançando as operações por um mecanismo de chamada de ofertas, permitindo escolher um agente fornecedor B que não tem o por quê recusar-se a fazer a tarefa. Se tal procedimento falhar, pode-se ainda fazer uma chamada a um sistema de alocação centralizado de tarefas para tentar resolver a questão.

Em [FER95] enfatiza-se, para o Caso 2 supracitado, que as reservas associadas aos sistemas centralizados de alocação de tarefas não são importantes extrema, porque apenas um pequeno número de solicitações serão enviadas ao sistema centralizado. Este último desempenha assim, mais um papel de agente de manutenção e alocação do que efetivamente um mediador. Ele revela-se muito útil porque ele pode servir para atualizar as tabelas de relacionamentos dos agentes e assim auxiliar na reorganização da rede de relacionamentos para acelerar o processo de alocação.

Aqui o mecanismo de alocação é integrado diretamente no sistema organizacional dos clientes potenciais. Existe então dois tipos de conexões, que são: c1 e c2. c1 liga o módulo de alocação aos sistemas motivacional e organizacional. c2 representa as mensagens que um

cliente A troca com os outros agentes. As conexões internas são definidas pelas seguintes mensagens:

- **Alocar(T)** é uma solicitação do módulo motivacional para o módulo de alocação para que este último encontre alguém para fazer a tarefa **T**.
- **ImpossívelAll(T)** é uma resposta do gerenciador de alocações para o sistema motivacional do agente informando que não há nenhum agente fornecedor que aceite fazer a tarefa **T**.
- **ComprometidoFazer(Y,T)** é uma resposta do módulo de alocação para o módulo organizacional informando que o agente fornecedor **Y** se comprometeu em fazer a tarefa **T**.

As mensagens externas são definidas pelos seguintes atos de linguagem:

- **Solicitar(T, self)** é uma solicitação do agente cliente **self** para um agente fornecedor potencial requerendo a realização da tarefa **T**.
- **Aceitar(T, Y)** é a resposta positiva do agente fornecedor **Y** à solicitação de um cliente.
- **Recusar(T, Y)** é a resposta negativa do agente fornecedor Y à solicitação de alocação de tarefa **T**.

A Figura 2.4 apresenta a estrutura do módulo de alocação direta. Ele difere, de forma geral da alocação centralizada, sobre dois pontos:

- O parâmetro X que representa o agente cliente desapareceu das estruturas dos dados internos do mediador M.
- O aceite ou a impossibilidade de encontrar um agente fornecedor Y é passada aos sistemas motivacional e organizacional do agente cliente.

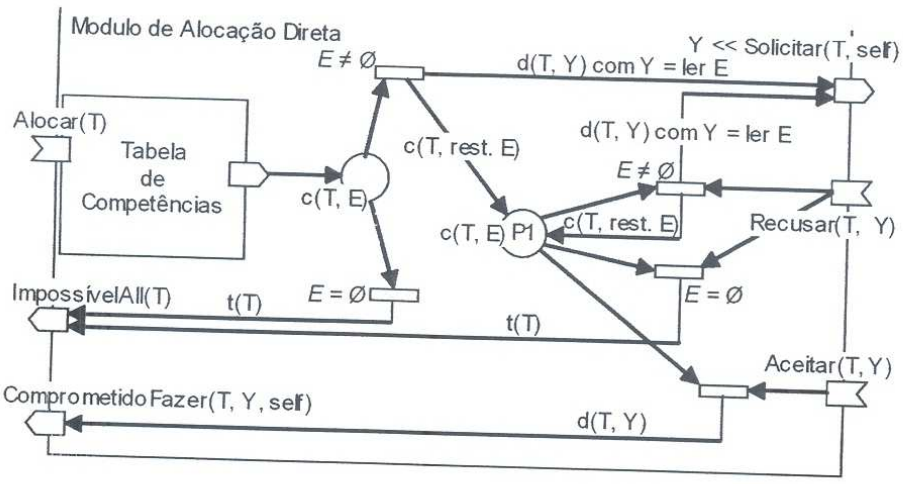

Figura 2.4: Representação do módulo de solicitação do mediador M para relacionamentos diretos.

A ordem na qual os agentes são questionados não é necessariamente neutra para encontrar quem fará a tarefa **T**. Classicamente, duas possibilidades se apresentam:

- Usar a noção de prioridade para optar por um agente fornecedor **Y** ao invés de outro;
- Efetuar um cálculo que leve em conta a carga de processamento de um agente e as características da tarefa **T**.

Supõe-se apenas que um agente cliente A possui a necessidade de conhecer um número suficiente de especialistas. Desta forma, se A tem uma tarefa T que ele não pode efetuar então poderá ser realizada por um destes especialistas. Este mecanismo permite colocar em contato apenas os fornecedores que estariam em contato direto com os clientes. Isto é um fator limitante. Tem-se então a necessidade introduzir um mecanismo que permita a comunicação indireta entre os agentes.

Alocação por Delegação

A técnica de alocação por delegação resolver o fator limitante supracitado. De fato, ela permite conectar um agente cliente A e um agente fornecedor B que não se conhecem diretamente. Este modo de alocação permite que um agente fornecedor B, incapaz de realizar a tarefa T que lhe foi confiada, repasse-a a um outro agente fornecedor C capaz de fazê-la. Por exemplo, o agente cliente A não conhece nenhum agente fornecedor capaz de executar uma

tarefa T que requer a competência C2. Assim, o agente A solicita a todos os agentes que ele conhece para que estes solicitem "recursivamente" aos seus relacionamentos quem possui a competência C2 para fazer a tarefa T. Este processo se repete até encontrar um agente que aceite fazer a tarefa T ou até que a rede de agentes seja percorrida por completo.

Este processo repousa sobre algoritmos de busca em largura/paralela ou profundidade. A técnica geral da busca em paralelo repousa sobre algoritmos de difusão que propagam recursivamente as solicitações para todos os agentes conhecidos. Deve-se, no entanto, levar em conta duas restrições: (i) verificar que a execução da tarefa T seja solicitada apenas uma vez para um dado agente; e (ii) assegurar que o processo de busca termine. Se tal processo terminou e nenhum agente aceitou fazer a tarefa T, então atribuir a tarefa T de maneira autoritária ou passar para o modo de chamada de oferta. Classicamente, a primeira dificuldade é superada marcando os agentes consultados no momento da busca e a segunda utilizando uma mensagem Reconhecer. Esta mensagem serve para retornar para os agentes envolvidos no processo o fato de que todas as solicitações que se encontram em avaliação foram efetivamente efetuadas.

Deve-se também gerenciar a possibilidade de haver várias aceitações simultâneas. Devido ao paralelismo, dois agentes podem aceitar o comando e se comprometer vis-à-vis a um dado agente. Sendo assim, o fornecedor que aceita não deve se comprometer, mas simplesmente reservar seu comprometimento até o estabelecimento do contrato com o cliente. Além disso, o cliente deve levar em conta o fato que se pode receber várias propostas ou considerar também que ele deve aceitar apenas a primeira.

[FER95] ilustra tal mecanismo por meio da seguinte situação. Supondo que o agente cliente A necessita que alguém realize a tarefa T. O cliente A solicita a todos os seus relacionamentos, que são B e C para realizar T. Como B e C não podem fazê-la, eles delegam a solicitação, onde B reenvia a D e E, e C reenvia a B e F. Assim, B recebe uma nova solicitação e ele se contenta em enviar uma mensagem Reconhecer ao agente C. Como D não pode fazer a tarefa T e ele não tem nenhum conhecido, então ele devolve apenas a mensagem Reconhecer ao agente B. E por sua vez delega a solicitação a F. Considerando que F sabe fazer a tarefa T e ele recebeu a primeira solicitação de B. F envia para A seu aceite em fazer a tarefa T e para B uma mensagem de reconhecimento. Quando F recebe uma nova solicitação de C, ele responde apenas com uma mensagem de reconhecimento. Se várias mensagens de aceite chegam até A, ele leva em conta apenas a primeira e marca que a tarefa T que ele

solicitou está contratada. Os atos de linguagem necessários para as comunicações entre agentes são os seguintes:

- **Solicitar(P, X, Y)** é uma solicitação do cliente **X** ao fornecedor potencial para que a tarefa **P** seja realizada. Tal solicitação advém do solicitante **Y**.
- **Propor(T, Y)** é a resposta positiva do fornecedor **Y** à solicitação do cliente. Ele informa que ele fez uma reserva para a tarefa **T** e aguarda uma mensagem de acordo ou uma mensagem de recusa.
- **Reconhecer(T, Y)** informa que o agente **Y** aceita fazer **P** ou que ele recebeu uma mensagem de reconhecimento de todos os agentes para quem ele solicitou para fazer **T**.
- **OfertaAceita(T, X)** é uma resposta positiva do cliente **X** à proposta de aceite do fornecedor. O fornecedor pode então comprometer-se em fazer **P** para **X**.
- **OfertaRecusada(T,X)** é uma resposta negativa do cliente **X** à proposta de aceite do fornecedor, o qual pode retirar **P** de seus comprometimentos em reserva.

Para estas mensagens, deve-se acrescentar as conexões com os sistemas organizacional e motivacional do agente.Tais conexões são estabelecidas através das seguintes mensagens:

- **Alocar(T)** é uma solicitação do sistema motivacional para tentar encontrar alguém para alocar a tarefa **T**.
- **ImpossívelFazer(T)** é uma resposta negativa ao sistema motivacional informando que a tarefa **T** não pode ser alocada.
- **ComprometidoFazer(T,Y,X)** é um comprometimento do agente **X** em fazer a tarefa **T** para **Y**. Cada vez que existe um contrato firmado entre um cliente e um fornecedor, ambos agentes memorizam este comprometimento de forma simétrica.

O mecanismo de alocação por delegação pode ser representado na forma de um conjunto de quatro módulos (Figura 2.5), que gerenciam tanto à parte do cliente e do fornecedor de cada agente:

- O módulo de avaliação de solicitações pode propor uma oferta ou transmitir a solicitação do módulo de delegação. Se o agente já recebeu uma solicitação, ele se contenta em enviar uma mensagem de reconhecimento ao agente emissor da solicitação (Figura 2.7).
- O módulo de delegação se encarrega de enviar uma mensagem de solicitação à todos os seus relacionamentos e receber todos os reconhecimentos. Este módulo é ativado por um comando do módulo precedente ou por uma solicitação de alocação de tarefa proveniente do sistema motivacional do agente (Figura 2.6).
- O módulo de avaliação de propostas se encarrega de aceitar a primeira oferta recebida de um fornecedor e recusar todas as outras (Figura 2.8b).
- O módulo de recepção de decisões registra o compromisso do agente em executar uma tarefa (se a oferta foi aceita) e colocar no lixo as ofertas recusadas em esperas (Figura 2.8a).

O módulo de delegação (Figura 2.6) se limita a enviar a mensagem SolicitarFazer a todos seus relacionamentos e a esperar seus reconhecimentos. Ele conta os reconhecimentos recebidos graças ao contador P1. Quando ele recebeu o mesmo número de reconhecimentos que ele enviou de solicitações, ele reenvia-lhe também uma mensagem de reconhecimento (exceto se ele é o cliente, ou seja, o agente que iniciou a difusão). Neste caso, ele reenvia a mensagem Fim ao módulo de avaliação de propostas.

O módulo de avaliação de solicitações (Figura 2.7) se encarrega de tratar uma solicitação. Se ele pode (e ele deseja) fazer a tarefa T, ele propõe seus serviços e se coloca em espera. Se não ele delega essa solicitação a todos seus relacionamentos, exceto se ele já recebeu uma solicitação semelhante, neste caso, ele reenvia simplesmente uma mensagem de reconhecimento. A variável P2 serve apenas para memorizar o fato que ele já respondeu a solicitação para fazer T para um cliente X.

A Figura 2.8a mostra o módulo que recebe as decisões provenientes de um cliente. A Figura 2.8b mostra por sua vez o módulo que avalia as propostas. Quando o módulo de avaliação recebe uma oferta O, ele informa o sistema organizacional que um agente fornecedor se compromete efetivamente em atender a solicitação do cliente. Para isto, tal módulo registra a oferta O utilizando o espaço P5 e recusa todas as demais ofertas. O primeiro fornecedor recebe aceite ou recusa e envia ao agente solicitante um reconhecimento para lhe sinalizar que ele recebeu uma solicitação. Finalmente, se a oferta foi aceita, o fornecedor se

compromete com o seu cliente. O espaço P4 memoriza as propostas em espera de uma confirmação positiva ou negativa, e o espaço P3 consome as marcas se a oferta é recusada.

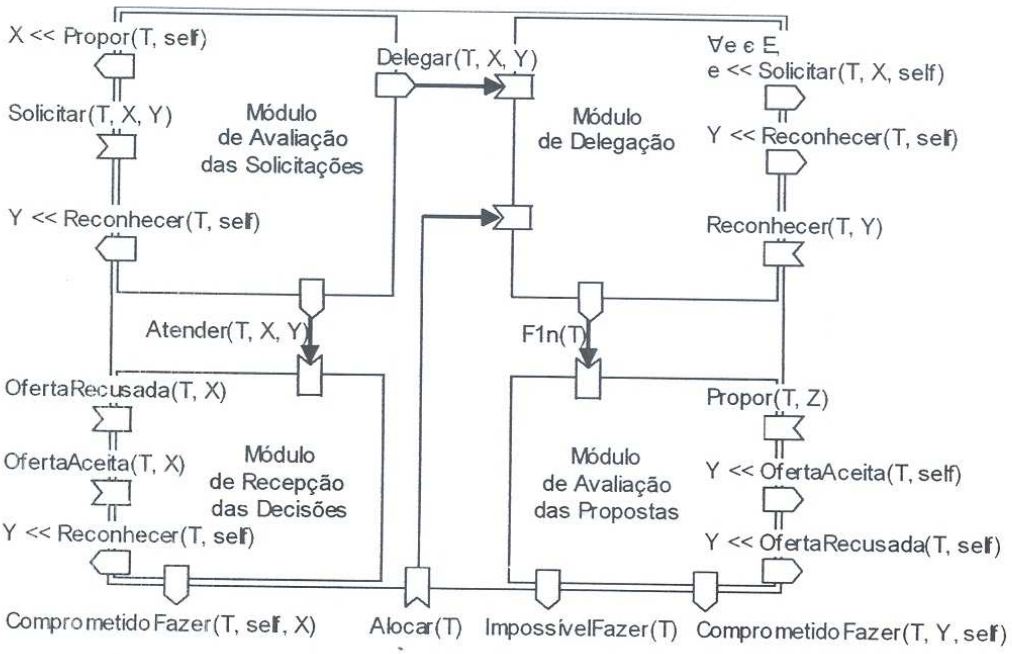

Figura 2.5: O sistema de alocação por delegação decomposto em quatro módulos.

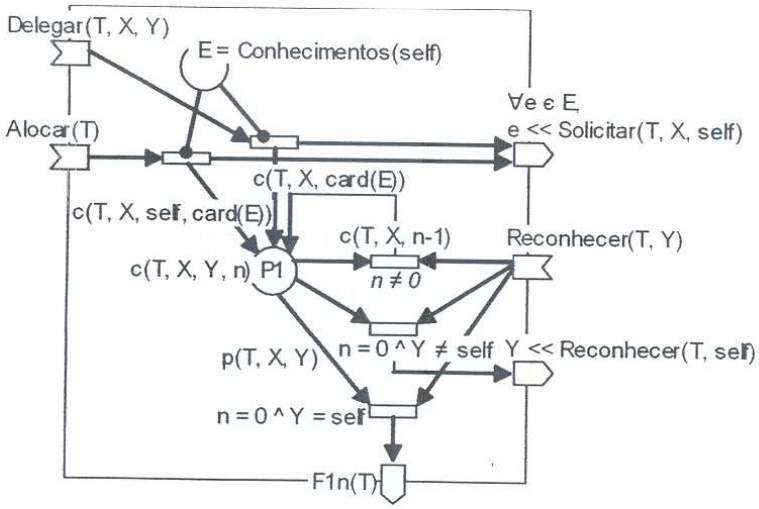

Figura 2.6: Módulo de delegação.

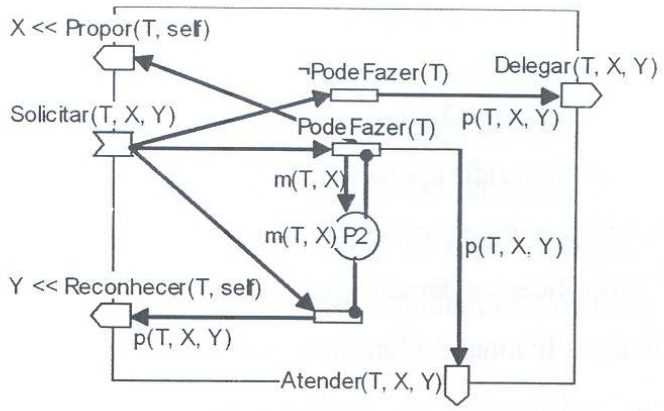

Figura 2.7: Módulo de avaliação de solicitações.

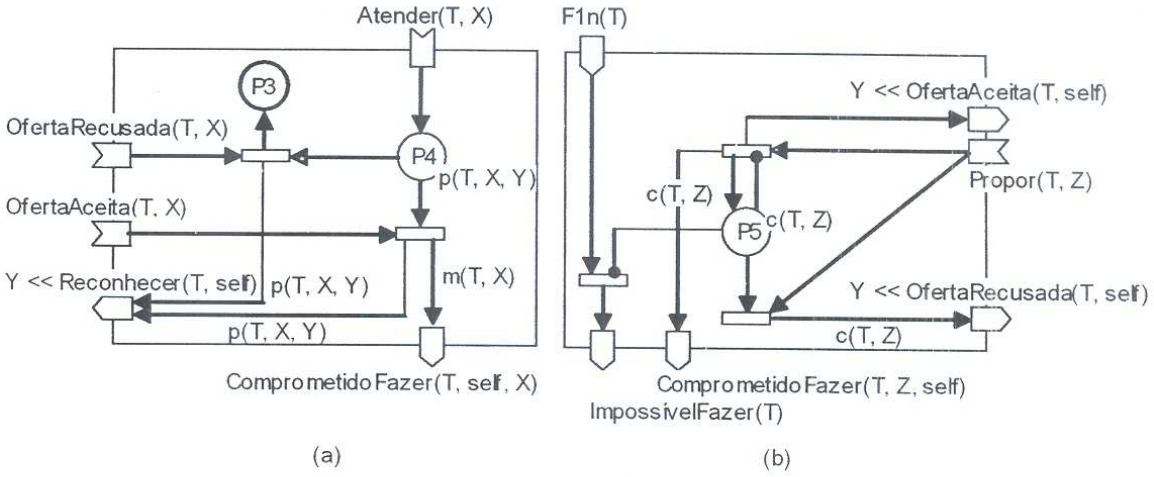

Figura 2.8: Módulo de recepção das decisões (a) e de avaliação de propostas (b).

Este algoritmo apresenta certas imperfeições:

- Em primeiro lugar, não foi levado em conta as informações sobre as competências dos relacionamentos para otimizar a busca. Cada agente envia as solicitações a todos os seus relacionamentos, quando ele poderia enviar a solicitação em prioridade aos agentes que ele sabe que dispõe desta competência.
- Quando um agente informa que ele aceita uma tarefa. É suficiente então que o agente que aceita envie uma mensagem que **ReconhecerSucesso** para o solicitante e que este solicitante envie uma mensagem Parar a todos os agentes para quem ele enviou uma solicitação e que não tenham ainda devolvido uma resposta. Esta modificação, que pode aumentar o número de mensagens entre os agentes, é

eficiente apenas se as mensagens Parar forem mais rápidas e mais prioritárias que as outras.

- Este algoritmo também não leva em conta os diferentes níveis de competências dos agentes, considerando apenas a primeira aceitação. Para mudar tal processo, é suficiente memorizar os agentes que aceitaram fazer o trabalho e contratar apenas os "melhor" e notificar os demais que eles não foram aceitos.
- Uma dificuldade adicional é identificar cada tarefa de forma única. Para isto pode-se rotular as solicitações de tarefas concatenando a descrição da tarefa, o identificador do agente solicitante inicial e um número de ordem às solicitações.

Reorganização de uma Rede de Relacionamentos

Os relacionamentos dos agentes podem ser considerados como um tipo de memória "cache" para os quais um agente é a mesma coisa que conhecer diretamente as características dos outros agentes que ele pode necessitar. Deste fato, encontramos nos sistemas de relacionamentos os problemas clássicos das memórias "cache" e em particular a dificuldade em manter uma coerência entre os relacionamentos e o estado dos agentes que elas representam, notadamente quando os agentes mudam de especificidade, quando um novo agente entra no sistema, ou quando um dentre eles deixa o sistema. Neste caso é necessário atualizar a base dos relacionamentos dos agentes de tal forma que eles se tornem coerentes, o que põe o difícil problema da reorganização de uma rede de relacionamentos.

Existe evidentemente um grande número de variantes possíveis da rede de relacionamentos, por exemplo, pode se querer reorganizar a rede de maneira dinâmica a fim que os agentes que não se conhecem inicialmente sejam colocados em contato. Pode-se também melhorar as performances do sistema, gerindo de maneira estatística os relacionamentos dos agentes de maneira a otimizar as solicitações. Os agentes com os quais um agente está freqüentemente em contato se encontrarão em sua tabela de relacionamentos. É então possível fazer algo para que a rede aprenda a alocar as tarefas que ela deve executar durante seu funcionamento. Por exemplo, dando "boas" notas aos agentes que dão satisfação e notas ruins para aqueles que respondem insatisfatoriamente as solicitações. O sistema CASCADE realizado por [PAR90] que controla o sistema de gestão de manutenção funciona *sobre este principio.*

O que fazer quando novos agentes são colocados em serviço em um universo multi-agente? Pode-se utilizar a técnica do *check-in*, os novos agentes se apresentam a um "gestor".

Pode-se eliminar quase totalmente a figura do "gestor central e único". Para isto pode-se definir um sistema hierárquico de gestores como nas redes de computador, onde cada gestor controla sua região e seu conjunto de agentes. Pode-se também fazer algo para que os novos agentes se enderecem apenas a alguns agentes do sistema, depois, pela utilização de relacionamentos, a informação se propaga ao longo da rede. Se o grafo dos relacionamentos é fortemente conectado, todos os agentes do sistema podem se beneficiar das características dos novos agentes.

2.3.2. Alocação por Chamada de Oferta: Redes de Contrato

A rede contratual é um mecanismo de alocação de tarefas baseado sobre a negociação de chamada de oferta [SMI79]. É uma estrutura de controle bastante simples para compreender e utilizar. "A rede contratual é uma técnica da computação onde o aspecto conceitual se associa à implementação para apresentar mecanismos utilizáveis, fáceis a aprender e a programar, cujos resultados correspondem relativamente as melhores expectativas" [FER95].

A rede contratual baseia-se no protocolo de elaboração de contratos do tipo mercado público. A relação entre o cliente ou gestor e os fornecedores ou respondentes passa por uma chamada de oferta e uma avaliação de propostas. A chamada de oferta se efetua em quatro etapas:

- Lança-se a chamada de oferta. O gestor envia uma descrição da tarefa que ele gostaria que fosse realizada. Tal mensagem vai para todos os agentes que o gestor estima ser capaz de responder ou em broadcast.
- Cada respondente elabora, a partir dessa descrição, uma proposta e retorna-a ao gestor.
- O gestor recebe e avalia cada proposta, atribui o contrato C ao melhor respondente.
- O respondente, que recebeu contrato C torna-se o contratado. Ele envia uma mensagem ao gestor informando que está de acordo em executar a tarefa requerida comprometendo-se. Se o contrato C não for cumprido, relança a avaliação das ofertas e a atribuição do contrato a um outro agente.

A questão ligada a re-alocação de um contrato não cumprido é um problema tratado amplamente no sistema B-Contract proposto por [SCA96].

Algoritmo

[FER95] considera que são necessários seis tipos de mensagens para realizar a alocação de tarefas utilizando o conceito de redes contratuais:

- **ChamadaOferta(T,X)** é uma mensagem do gestor **X** para um respondente potencial solicitando que uma proposta seja retornada sobre a tarefa **T** (se ele está interessado).
- **Proposta(T,O)** é a resposta positiva de um respondente **Y** para uma chamada de oferta sobre a tarefa **T**. Ele retorna então uma oferta **O**, incluindo também informações como o nome do respondente.
- **NãoInteressado(T,Y)** é a resposta negativa de um respondente ao gestor informando-o que **Y** não está interessado na execução de **T**.
- **Atribuir(T,C,X)** é a mensagem do gestor **X** para um respondente sinalizando-o que foi contemplado com o contrato **C** sobre a tarefa **T**.
- **Aceitar(T,Y)** é a resposta positiva do respondente **Y** à atribuição do contrato pelo gestor.
- **Recusar(T,Y)** é uma resposta negativa do respondente **Y** à atribuição do contrato pelo gestor. Este evento relança a avaliação e o processo de atribuição.

Aqui o gestor apresenta também as conexões clássicas com seus sistemas motivacional e organizacional: a solicitação de alocação Alocar, as respostas **ImpossívelFazer** e **ComprometidoFazer**.

Um agente pode ser ao mesmo tempo gestor e respondente. Trata-se de dois papeis diferentes que um agente pode assumir indiferentemente. A Figura 2.9 apresenta a arquitetura geral dos módulos necessários ao estabelecimento de um protocolo de rede contratual.

O gestor compreende dois módulos (Figura 2.10): o módulo de definição da chamada de oferta e recepção das propostas (A). Estas propostas em seguida selecionadas pelo módulo (B) que se encarrega de atribuir os contratos e receber as aceitações ou recusas dos respondentes. O módulo A comporta três espaços de memória. O primeiro, P1, contém apenas o identificador de todos os agentes do sistema. O espaço P2 serve para contar as respostas à chamada de oferta no intuito de saber quando ele poderá passar para a seguinte e P3 memoriza todos as propostas recebidas. O módulo B comporta apenas um único espaço que é utilizado para a definição da atribuição interativa dos contratos aos respondentes. Em

[FER95], assim como no contexto deste trabalho, foi suposto que na implementação do protocolo da rede contratual, que os respondentes podem recusar o contrato enviado pelo gestor. Os respondentes possuem também dois módulos (Figura 2.11). O primeiro (C) é consagrado à avaliação da chamada de oferta e à elaboração de uma proposta, o segundo a avaliação dos contratos atribuídos pelo gestor.

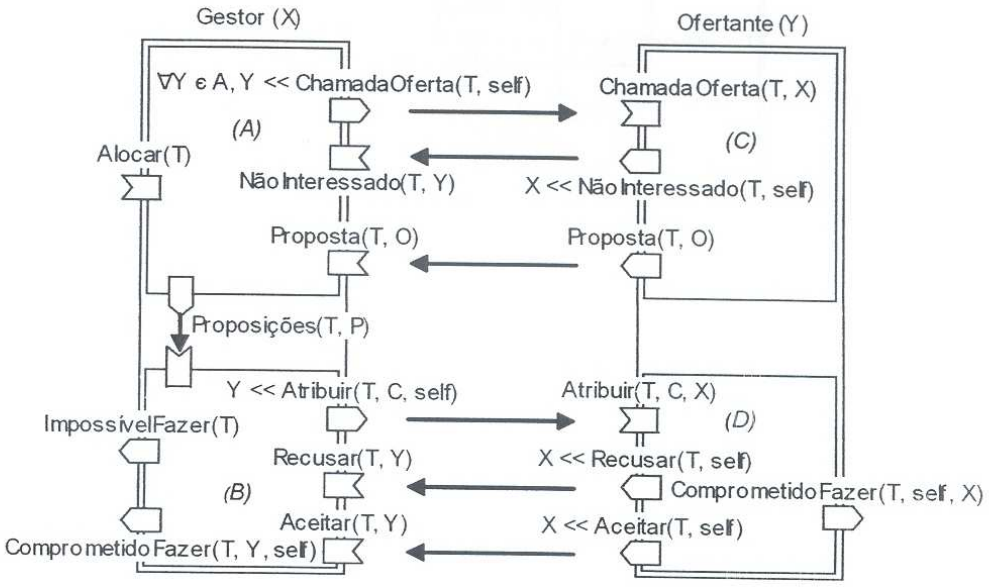

Figura 2.9: diagrama esquemático de um protocolo de rede contratual.

O gestor se encarrega em definir uma chamada de oferta e da recepção das propostas (módulo A), depois seleciona as propostas e atribui um contrato ao "melhor" (módulo B). O respondente elabora uma proposta em função da chamada de oferta (módulo C) e decide ou não em aceitar o contrato elaborado pelo gestor (módulo D).

Linguagem de Elaboração de Contratos

R. Smith propõe uma linguagem completa que se situa a um nível de abstração elevado [SMI80], no qual o anuncio da chamada de oferta comporta, além da descrição da tarefa e o número do contrato, uma definição das qualidades requeridas para a tarefa, o formato da proposta e uma data de expiração. A definição das qualidades requeridas permite eliminar rapidamente os agentes que não tem essas qualidades sem ter a necessidade de efetuar uma análise minuciosa da descrição da tarefa. Além disso, reduz o número de mensagens nos canais de comunicação.

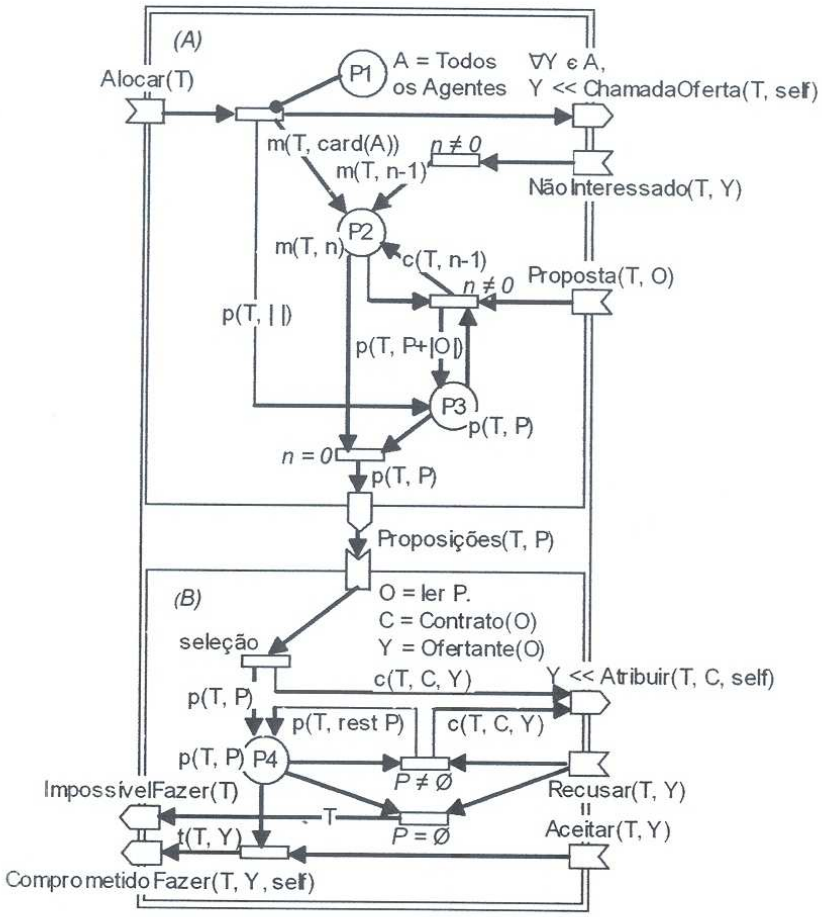

Figura 2.10: Diagrama do comportamento de um gestor de uma chamada de oferta.

O módulo (A) serve para a emissão das chamadas de oferta e recepção das propostas que são em seguida transmitidas ao módulo (B) de seleção de propostas e atribuição de contrato.

O formato da proposição serve para indicar ao respondente quais são os critérios que serão levados em conta no momento da avaliação e assim facilitar em seguida o trabalho de avaliação do gestor. Em fim, a data de expiração dá a data limite após a qual as propostas não serão mais retidas.

Segue um exemplo de mensagem do tipo chamada de oferta baseado no formato definido por R. Smith:

Mensagem: Chamada de Oferta

*Para: **

De: Explorador-25

Contrato: Perfurar-Explorar-35-234

Descrição de tarefa:

Tipo de tarefa: Perfurar

Qualidades necessárias:

Deve ter : Sistema de perfuração

Deve poder: Perfurar solo mole

Formato de proposta:

Posição: <lat, long>

Qualidade perfuração: {tipo de solo: performance}

Data de expiração:

20 Junho 2193 13:06:46

Fim de chamada de ofertas

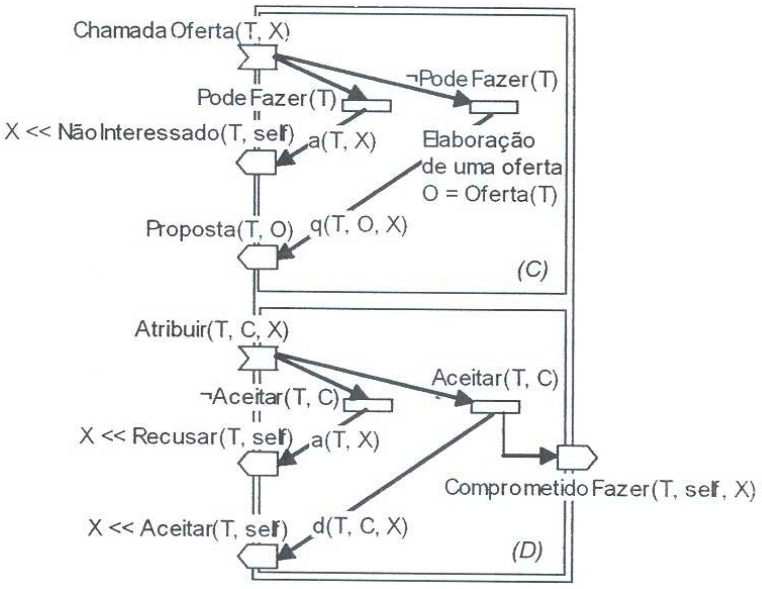

Figura 2.11: diagrama de comportamento de um respondente.

O módulo (C) serve para a avaliação das propostas e a elaboração de uma oferta. O módulo (D) se encarrega da aceitação ou não de um contrato tal como ele foi elaborado pelo gestor.

Segue um exemplo de mensagem do tipo proposta também baseado no formato definido por R. Smith:

Mensagem: Proposta

Para: Explorador-25

De: Perfurador-18

Contrato: Perfurar-Explorar-35-234

Descrição da proposta:

Posição: <47N, 17W>

Qualidade perfuração: {Área: 0,5 Argila: 0,3 Xisto: 0,8}

Fim de proposta

O gestor atribuirá o "contrato" àquele que ele estima ser a melhor proposta. A simplicidade aparente do algoritmo apresentado acima não deve mascarar um certo número de dificuldades, que são:

- A avaliação da proposta;
- O acesso aos respondentes;
- A consideração de gestores múltiplos e dos compromissos pessoais;
- A decomposição das tarefas em sub-tarefas; e
- A qualidade da alocação obtida.

Avaliação da Proposta

A avaliação de uma proposta pode ser na forma de uma simples medida numérica, ou de uma descrição que inclui vários critérios. Quem faz tal avaliação? Trata-se de um respondente que se interessa pela tarefa T. A resposta consiste de uma proposta que contem uma avaliação de sua capacidade para realizar tal tarefa. Classicamente em um sistema multiprocessado, onde uma aplicação busca apenas melhorar seus tempos de resposta, é possível facilmente medir a carga de um processador na forma de um simples valor e conseqüentemente o interesse sobre o mesmo. Destaca-se em [SCA96] que parece mais difícil caracterizar o interesse de um agente para uma tarefa, assim como medir suas capacidades para oferecer um "bom" serviço. Neste caso, é tarefa do projetista do sistema em definir uma função econômica adequada. A performance de uma rede contratual está fortemente ligada à função econômica empregada.

Endereçamento dos Respondentes

Parte-se, em geral, que o endereçamento dos agentes situados em uma rede não constitui um problema. Ou seja, um gestor sempre se endereça a todos os agentes. Trata-se

evidentemente de uma concepção idealista. Ela não trata dos problemas de implementação computacional em contextos realmente distribuídos.

Que mecanismo deve-se lançar mão para acessar os agentes que enviam suas propostas? Algoritmos de busca em rede paralela podem ser utilizados para acessar o conjunto de agentes da rede. Eles assemelham-se essencialmente ao algoritmo utilizado para alocação por delegação de uma rede de relacionamentos, porém ao invés de solicitar aos agentes se eles aceitam executar uma tarefa T, requer-se destes agentes a elaboração e envio de uma proposta ao gestor. Quando o processo encerrou, o gestor começa sua avaliação.

[FER95] enfatiza que tal processo é bastante custoso em tempo e em quantidade de mensagens. Por partir-se do principio que todos os agentes do sistema devem ser endereçados de forma individual. Destaca-se também o problema da constituição de uma rede de relacionamentos, mesmo se está é mais simples que a rede de alocação por delegação, onde não se requer aos agentes conhecerem as competências de seus vizinhos, mais apenas saber da existência dos vizinhos. Uma outra solução, também destacada, consiste em que todos os agentes sejam memorizados em uma estrutura de dados global conservada por um único agente, porém recai-se sobre um gargalo de estrangulamento e sensibilidade às falhas características dos sistemas centralizados.

Classicamente, os projetistas de uma rede contratual adotam uma solução que consiste em colocar os agentes sobre um "barramento" (Figura 2.12).

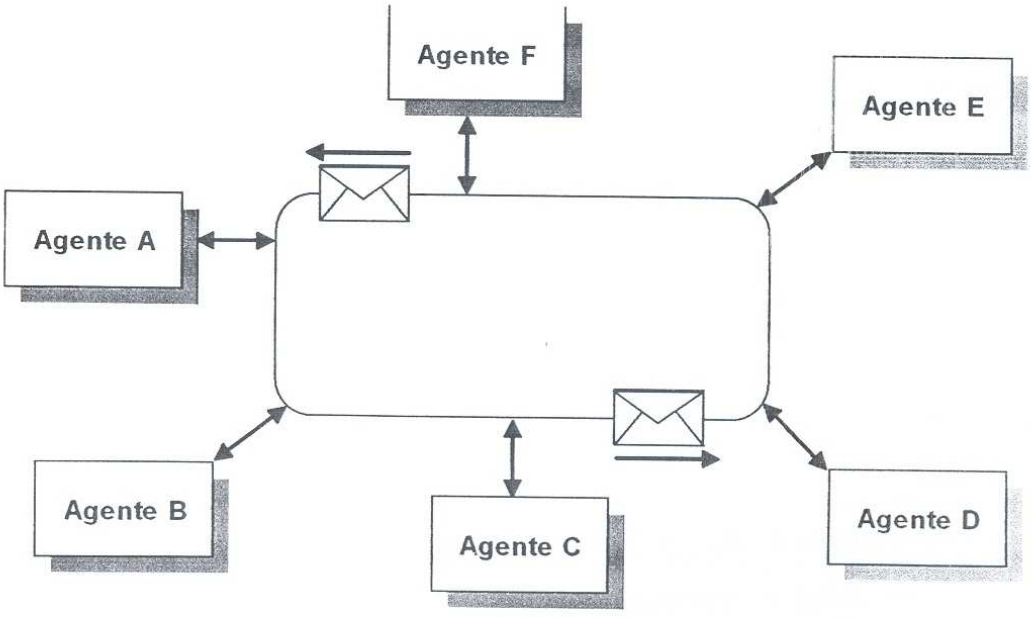

Figura 2.12: Implantação de uma rede contratual pela implementação de agentes sobre um barramento em anel.

O funcionamento da veiculação das mensagens assemelha-se a uma rede token ring. Aqui as mensagens (chamada de oferta ou proposta) trafegam sempre no mesmo sentido e passam de vizinho em vizinho. Uma solução consiste em considerar que as chamadas de oferta são colocadas em uma bandeja. Os respondentes lêem de forma exclusiva as chamadas de oferta, avaliam-nas e colocam sua proposta na bandeja. Quando a bandeja retorna ao gestor responsável desta chamada, ele sabe que todos responderam e ele pode escolher entre o conjunto das propostas. Deve-se destacar que o processo é seqüencial e desconsidera o paralelismo potencial da rede contratual. Por outro lado, esta forma de veicular as mensagens permite definir alocações de tarefas mais ótimas [SCA96].

Para melhorar a performance (tempo de calculo e número mensagens), tem-se verificado na literatura o seguinte caso: quando uma rede é pouco "volátil" (identidade, quantidade e competências dos agentes variam pouco), é possível então aplicar técnicas de memórias cache. Tal técnica recai em gerenciar de forma local tabelas de relacionamentos dos gestores e indexar os agentes pelas tarefas que eles se habilitam. Esta técnica foi desenvolvida por [PAR90] para gerenciar a alocação automática de tarefas em ateliês flexíveis.

Supõe-se também que todos os agentes que sabem efetuar as tarefas de um certo tipo respondem necessariamente à oferta retornando uma proposta. Quando um gestor deve efetuar uma chamada de oferta para um novo tipo de tarefa, ele difunde esta chamada na rede e memoriza em sua tabela de relacionamentos os agentes que retornaram uma proposta para este tipo de tarefa. Na presença de uma tarefa do mesmo tipo, ele apenas percorrerá tal lista para obter a mesma eficiência. A tabela de relacionamentos desempenha assim o papel de memória cache que otimiza os acessos à rede de agentes. Pode-se ter uma lista com um único indivíduo. Neste caso, o gestor pode firmar contratos diretamente sem passar por um mecanismo de chamada de ofertas. O protocolo B-Contract, desenvolvido por [SCA96], implementa tal mecanismo. A alocação direta é feita pela mensagem Execute, emitida pelo gestor. O respondente agraciado retorna uma mensagem Accept ou Reject caso ele aceite ou rejeite respectivamente o contrato.

A manutenção da rede se faz necessária basicamente quando:

- Um novo agente é acrescentado ao sistema;
- Um agente deixa o sistema; e
- Uma competência foi atualizada/modificada.

Os agentes informam as mudanças supracitadas de forma pró-ativa a rede difundindo a informação a todos os gestores. Tal informação é veiculada a uma mensagem **Inform**. Ele enfatiza que tal técnica reduz os tempos de resposta e os números de mensagens trocadas, porém as vantagens são perceptíveis se as modificações são em número reduzido frente o número de chamada de ofertas.

Data de Expiração

Considerou-se até o presente que o gestor aguarda todas propostas provenientes dos fornecedores potenciais antes de iniciar o processo de avaliação e seleção das mesmas. Este procedimento tem, de um lado, a vantagem de melhor levar em conta o conjunto das potencialidades do sistema sem deixar nenhuma oferta de lado, do outro lado, dois consideráveis inconvenientes se apresentam, que são: (i) todos os agentes devem responder, mesmo aqueles que não estejam a priori interessados pela chama de oferta. Tal procedimento pode comprometer os canais de comunicação, saturando-os com uma enorme quantidade de mensagens passando apenas a informação "não interessado"; e (ii) se um agente falha ou está em funcionamento precário pode comprometer todo o sistema, de forma a fazer o gestor aguardar uma resposta indefinidamente.

A solução é bastante simples. Ela consiste em desobrigar os agentes desinteressados ou incapazes de fazer uma tarefa T retornar uma mensagem comunicando tal informação. Desta forma, cada chamada de oferta inclui uma data de expiração (ou limite de espera) para a recepção das propostas. O gestor descartará todas as propostas que chegarem após a expiração da data de limite.

Em um sistema que busca aprender com as comunicações trocadas entre os diferentes agentes não deve simplesmente descartar as mensagens que chegam com atraso. Elas podem servir para atualizar a tabela de relacionamento do gestor. O sistema OSACA (*Open System for Asynchronous Cognitive Agents*) desenvolvido por [SCA96], gerencia duas filas de recepção de mensagens, que são: W1 e W2. Em W1 são inserida as mensagens endereçadas especificamente a um agente gestor e dentro da data limite estipulada. Todas as outras mensagens que não foram endereças diretamente ao agente gestor ou chegaram com atraso são armazenadas em W2. Os elementos de W2 são avaliados por um processo que busca aprender a partir das informações contidas nestas mensagens. Deve-se destacar que as mensagens do tipo broadcast recebem um tratamento especial, em particular, porque elas não

são endereças diretamente ao agente gestor, porém pode ser de interesse do gestor (e.g., chamada de oferta, normalmente enviada em broadcast).

Evidentemente, esta solução traz outros problemas: deve-se calibrar muito bem este tempo de reação de maneira que a maior parte das propostas tenham tempo para chegar até o gestor. Uma data limite muito próxima pode ter como conseqüência à não avaliação de inúmeras propostas, por outro lado, uma data limite muito longa poderá fazer o gestor aguardar inutilmente instantes preciosos antes começar a avaliação. Evidentemente, pode-se definir um sistema adaptativo.

Caso 1: se muitas solicitações são recebidas próximas à data limite, então aumenta-se o tempo de reação.

Caso 2: se o gestor aguardar muito tempo após a recepção das ultimas respostas, então diminui-se o tempo de reação.

O sistema será bem adaptado se todos os agentes respondem com o mesmo tempo de resposta.

Múltiplos Gestores e Otimização

Pode-se, em uma aplicação real, ter vários gestores trabalhando ao mesmo tempo e emitindo suas chamadas de oferta simultaneamente. Isto implicará em dotar os gestores de mecanismos capazes de gerenciar chamadas de ofertas simultâneas e concorrentes, que podem influenciar de forma perniciosa o comportamento global do sistema. É importante notar que as atribuições de contratos, em uma rede contratual, são geralmente feitas apenas a partir de conhecimentos puramente local. Ou seja, um gestor A não leva em conta o que os demais gestores de um sistema necessitam. [PAR87] considera que por causa da localidade do processo os agentes são submetidos a três tipos de falta de informação, que são a falta de informação temporal, espacial e carga de um agente.

Falta de informação temporal significa que os respondentes conhecem apenas as chamadas de ofertas já recebidas e não dispõem de nenhuma informação sobre as chamadas de ofertas que poderão chegar na próxima unidade de tempo. Deste fato, um agente pode comprometer-se a fazer uma tarefa T para qual que ele não é o mais adaptado, porém deixar de trabalhar sobre contratos que corresponderiam mais efetivamente as suas competências.

Falta de informação espacial significa que os gestores são incapazes de saber quais são as outras chamadas de ofertas que se encontram em andamento em paralelo, assim como os respondentes terem informações sobre as outras propostas também em andamento. Tal problema é ilustrado por [SMI83] por meio do seguinte exemplo: existem dois gestores, A e B que enviam para dois respondentes potenciais X e Y duas chamadas de ofertas para duas tarefas TA e TB, onde as propostas foram as seguintes: X propôs sobre TA uma eficiência de 90% e 80% para TB, Y propôs sobre TA uma eficiência de 80%e 20% para TB. Após a recepção das propostas, o gestor A escolherá normalmente X, porque sua proposta parece mais interessante, assim como B também escolherá X. Desta forma, X ficará enormemente carregado enquanto que Y permanecerá inativo, enquanto que o ideal seria atribuir a tarefa TB a X e TA a Y. Se X conhecesse a proposta de Y ele poderia diminuir sua proposta em favor de TA, da mesma forma, que se Y conhecesse a proposta de X ele poderia aumentar sua proposta em favor de TB. Existem sistemas onde todos os respondentes enviam suas propostas a todos os outros agentes. Isto permite obter propostas mais apropriadas, levando em conta o conjunto de propostas dos agentes.

Falta informação sobre as cargas de trabalhos dos outros agentes não conhecidas pelos os respondentes.

A Influência de sub-Contratantes

Quando um respondente X executa uma tarefa T sobre contrato com um gestor A, ele pode ser levado a terceirizar partes da tarefa T, que são: {T1, ..., Tn}. Neste caso, X se comporta como um gestor de {T1, ..., Tn}. Esta situação pode levar um respondente emitir as chamadas de oferta apenas após ter recebido o contrato, ou solicitar primeiramente aos terceiros em comprometer-se efetuar essas tarefas antes que ele devolva uma proposta ao gestor A. Trata-se de fato de um problema recursivo, e os terceiros podem ser levados a terceirizar partes de suas tarefas Ti e assim por diante.

Para um respondente evitar encontrar-se sem sub-contratante pode emitir uma sub-chamada de oferta antes de reenviar uma proposta ao gestor. Se não foi possível encontrar sub-contratantes devolve-se uma proposta pouco interessante. No caso contrário, emite uma proposta sem preocupação. Diz-se então que o respondente compromete-se tardiamente visto que ele solicita primeiramente a seus sub-contratantes para comprometer-se com ele antes que ele se comprometa efetivamente com o gestor. Tal abordagem traz alguns problemas, o que requer a definição de mecanismos eficientes para:

- Gerir os comprometimentos dos sub-contratantes;
- Liberar os sub-contratantes caso o gestor não retenha a proposta;
- Fazer face a um problema clássico em gestão de recursos em sistemas distribuídos, que é o *dead lock* entre vários processos que devem acessar vários recursos de forma simultânea.

Pontos Importantes sobre Redes de Contrato

A rede contratual é um mecanismo conceitualmente falando muito simples. Ela oferece de fato um mecanismo de alocação de tarefa que aporta uma grade flexibilidade na forme de gerenciar o acordo entre clientes e fornecedores. Deve-se destacar que trata-se unicamente de um protocolo de organização dinâmica do trabalho e não de um modelo de resolução de problemas. Tal protocolo não afirma nada sobre o conteúdo do que é trocado. Em particular, supõe-se que existe uma ontologia compartilhada para a descrição das tarefas e informações a serem transmitidas.

Vantagens

As vantagens da rede contratual são:

- Trata-se de um mecanismo de alocação dinâmica de tarefas simples a implementar, que se pode implementar sobre uma arquitetura distribuída.
- Oferece uma grande dinamicidade visto que ela não faz nenhuma invocação a gestão de quaisquer relacionamentos. Por exemplo, para que a rede leve em conta a entrada de novos agentes e a saída de outros se requer apenas que cada agente tenha as capacidades de *check-in* e *check-out*.
- Permite uma seleção exclusiva ou acordo bilateral entre cliente e fornecedor. Tem-se assim como resultado um grande "leque" de parâmetros que podem ser levados facilmente em conta, que são: as competências dos agentes, a carga de trabalho, o tipo da tarefa a efetuar, o tipo de dados a fornecer, a duração de espera máxima autorizada, o custo, a urgência, etc.

Desvantagens

A rede contratual apresenta os seguintes inconvenientes:

- Se a implementação é simples, a execução é pesada e produz um grande numero de mensagens.
- A performance está ligada a existência de um número pequeno de chamadas de oferta;
- Requer-se a implementação de uma estrutura complexa para gerir problemas ligados ao paralelismo do processo;
- Necessita-se de estratégias especificais para gerir a terceirização de sub-tarefas; e
- Alto custo na re-alocação de um contrato que foi quebrado por alguma razão. O protocolo *B-Contract* foi desenvolvido especificamente para remediar tal problema na rede contratual original [SCA96].

A rede contratual revela-se eficiente apenas à alocação de tarefas de granularidade alta. Sua simplicidade e dinamicidade garantem uma considerável vantagem com relação às abordagens vista anteriormente. Pode-se, em particular, referir-se a [PAR90] para uma apresentação do sistema YAMS de gestão de ateliê flexível e do *Flavor Paint Shop* descrito brevemente em [FER95].

Rede de Contratos Dirigida por Ofertas

A forma clássica da rede contratual é dirigida por solicitações. Ou seja, são os gestores que solicitam aos respondentes potenciais para fazerem propostas relacionadas às tarefas que os gestores gostariam de fossem executadas. Pode-se fazer que os fornecedores tomem a iniciativa, onde eles enviam suas capacidades aos clientes potenciais. Assim, o sistema passa a ser dirigido por ofertas. Esta abordagem é utilizada nos sistemas operacionais distribuídos.

Estados de um Contrato

Um contrato passa por uma série de estados, que são em geral:

- Em alocação, que significa que tanto em uma rede contratual como em sistema de alocação por relacionamentos a tarefa T não foi ainda atribuída.
- Fechado, significa que o fornecedor aceita efetuar uma tarefa comprometendo-se realizar T.
- Executado, significa que a tarefa T foi executada normalmente e o fornecedor retorna o fato que a tarefa se terminou corretamente.

- Rompido, significa que o fornecedor não pode honrar seus compromissos.

O que fazer quando um contrato é rompido? O cliente pode tomar as seguintes decisões:

Caso 1: se a tarefa T é não importante, então o cliente desta tarefa pode decidir simplesmente em abandoná-la.

Caso 2: se a tarefa T é importante e participa da realização de uma tarefa U mais importante que T, então o cliente de T pode: (a) tenta encontrar um outro fornecedor ou (b) informar o cliente de T que ele não poderá mais executar T. Isto pode provocar um efeito cascata de ruptura de contratos.

A ruptura de contrato pode também provocar penalidades. Isso significa de fato que no caso de uma rede de relacionamentos, o cliente de T pode eliminar tal fornecedor de seus relacionamentos de competências necessárias para realização de T. Em uma rede contratual, estas penalidades levaram o fornecedor diminuir suas propostas futuras para as tarefas do tipo T.

Quando ocorre a ruptura de um contrato de uma tarefa importante, conforme supracitado no Caso 2, requer-se um mecanismo eficiente de re-alocação de contrato. O protocolo B-Contract já referenciado tem um duplo objetivo: O primeiro consiste em tentar utilizar o melhor possível os recursos de um sistema multiprocessado. O segundo é dispor de um conjunto de informações e "pré-contratos" que permitam re-alocar uma tarefa T o mais rapidamente possível.

Tais objetivos definem uma alternativa à alocação de tarefas. Tal alternativa baseia-se na constatação que as máquinas de uma rede são em geral pouco utilizadas. Por exemplo, [BER91] em suas experiências constatou que as máquinas são freqüentemente sub-utilizadas, com um tempo médio de inatividade variando entre 33% e 90%, em certos momentos do dia. Estes valores reaparecem como limites inferiores e superiores das cifras indicadas por outros autores (37% em [ZHO88], 33% em período de ponta em [THE89], 70% em período normal em [MUT87]). Considerando esses números, uma maneira de melhor utilizar as máquinas seria alocar a mesma tarefa a vários agentes (máquinas) e recuperar em seguida a primeira resposta disponível. Se esta abordagem possui o inconveniente de despender mais recursos, ela tem a vantagem de reduzir consideravelmente o tempo de resposta quando uma re-

alocação de uma tarefa T se faz necessária. Levando esse raciocínio ao extremo, poder-se-ia alocar uma mesma tarefa T a todos os agentes que responderam a uma chamada de ofertas. Neste caso, é evidente que a escolha de um dentre eles não tem muito sentido. Todavia, no caso onde os agentes têm competências bastante amplas, esta política prevê a linearizar a execução das tarefas, e pode assim conduzir a bloqueios. A proposta baseia-se em uma iniciativa intermediária que consiste em alocar uma determinada tarefa T a todos os agentes que submeteram propostas, porém especificando dois níveis de prioridades. Assim, certos agentes, os respondentes principais, serão oficialmente encarregados de resolver/executar a tarefa T (prioridade alta), e os outros, os respondentes secundários, poderão igualmente trabalhar sobre tal tarefa (prioridade baixa), porém ficando disponível para tratar eventualmente outras tarefas. Esta abordagem define o protocolo B-Contract ("*Backed up*[6] *Contract*").

2.4. Considerações Finais

A alocação de tarefas tanto por uma rede contratual como por uma rede de relacionamentos desencadeia dois problemas:

- Como fazer um agente competente realizar uma tarefa T?
- Como fazer que as transformações de uma organização descritas pela entrada ou saída de agentes, ou pelas modificações de suas capacidades possam ser levadas em conta sem que o projetista do sistema tenha que explicitar quem deve fazer o que. Este problema é discutido em profundidade em [SCA96b], no contexto do desenvolvimento de um sistema de agentes aberto.

É importante relembrar que em uma rede de relacionamentos as informações que descrevem as competências ou o estado de um agente são diretamente acessíveis pelos clientes. Deste fato, os clientes podem decidir-se sem a necessidade de difusão anúncios. Por outro lado, é também relevante notar que em uma rede contratual pura supõe-se que agentes não possuem nenhum conhecimento a priori sobre os outros agentes. Desta forma, revela-se necessário que um gestor solicite a todos os outros agentes que estão interessados informarem o que eles sabem fazer por meio de uma proposta.

[6] Backed up: solution de secours.

Uma rede de relacionamentos permite que tarefas sejam alocadas de forma mais eficiente que em uma rede contratual, em particular, se a alocação é direta. Tal eficiência baseia-se no fato que a solicitação de realização de uma tarefa é feita diretamente aos agentes conhecidos sem passar necessariamente pelo lançamento de uma chamada de oferta. O grande problema de uma rede de relacionamentos é a falta dinamismo, onde toda alteração passa por uma fase de reorganização da rede. Diferentes abordagens são possíveis para fusioná-las no intuito de obter o melhor de cada um, a saber: (i) a primeira consiste em utilizar o sistema de relacionamentos para as "pequenas" tarefas e a rede contratual para as "grandes" tarefas [FER95]. Deve-se relembrar que o grande problema da rede contratual é o número de mensagens trocas entre os diferentes papeis (gestor, respondente, sub-respondente, ...), bem como o atraso Δ para reunir as propostas e em seguida avaliá-las e atribuir o contrato. Entende-se, no entanto, que tratando-se de uma tarefa importante (granularidade alta) o atraso Δ é suportável. Todavia, deve-se frisar que o atraso Δ. é insuportável para uma tarefa elementar cuja execução toma um tempo que corresponde as vezes uma fração do atraso Δ. Jacques Ferber recomenda utilizar, de um lado, uma rede de relacionamentos para todas as solicitações que não necessitam muito esforço e em particular se ocorrer uma falha de alocação tal fato não é tão relevante considerando que as tarefas gastam pouco tempo e, do outro, uma rede contratual para as tarefas mais importantes. (ii) a segunda consiste em fazer com que a aprendizagem de uma rede de relacionamentos passe por uma chamada de oferta cada vez que esta seja necessária. Tal mecanismo foi implementado no sistema OSACA [SCA96]. Este recurso foi utilizado para auxiliar na resolução do seguinte problema em uma rede de relacionamentos: como encontrar fornecedor para uma determinada tarefa? Como integrar facilmente novos agentes que desejam entrar na rede?

Capítulo 3

Lógica Paraconsistente

A cooperação no contexto deste trabalho apoiará a troca de informações entre os agentes para realizar uma tarefa, ou apenas para solicitar uma informação sobre um determinado objeto. Como apresentado na seção anterior, a cooperação fornece vantagens em termos de eficiência quantitativa e de emergência qualitativa, porém traz alguns problemas difíceis de distribuição do trabalho entre os agentes e tratamento das informações trocadas. Tal dificuldade surge devido à presença de vários parâmetros tais como: capacidades cognitivas, capacidades de comprometimento, competências individuais, natureza das tarefas, eficiência, custo de transmissão de uma informação, presença de informações contraditórias, estruturas sociais, dentre outros. O interesse da Lógica Paraconsistente no contexto deste trabalho porta sobre a representação e raciocínio sobre informações contraditórias ou inconsistentes sem eliminá-las do processo.

A Lógica Paraconsistente tem objetivo aplicar lógicas alternativas à lógica clássica, chamadas de Lógicas Não-Clássicas. Esta necessidade apresenta-se porque tais lógicas, notadamente a Lógica Paraconsistente, investigam a existência de outros valores-verdade além de "verdadeiro" e "falso". O resultado destas pesquisas é primordial para viabilizar a representação do conhecimento de especialistas humanos, os quais raciocinam quase sempre com valores-verdade diferentes de "verdadeiro" e "falso", ou 0 e 1. Em outras palavras, a Lógica Paraconsistente, é uma extensão da Lógica Clássica e desenvolvida com o propósito de conceber ferramentas que permitam um tratamento não-trivial para as informações contraditórias. Segundo Newton da Costa, as contradições ou inconsistências possuem como origem as condições do

ambiente em que se desenvolvem as tarefas. Como e quando estas situações contraditórias aparecem é, na maioria das vezes, independente da vontade dos dispositivos envolvidos.

O objetivo aqui será examinar a Lógica Paraconsistente intuito de definir um conjunto de operadores implementáveis computacionalmente, os quais deve facilitar a interpretação de informações contraditórias ou não. Em outras palavras, para representar o que um agente acredita sobre um objeto qualquer.

Claramente, não se deve admitir – como grande avanço da computação – a pura e simples eliminação de informações contraditórias de um processo de raciocínio. A Lógica Paraconsistente nos parece ser uma ferramenta apropriada para abordar tal problema. Neste sentido, diferentemente das metodologias de raciocínio quantitativo ([BLA87], [VAN86], [SUB87a], [SUB87b]) um sistema lógico paraconsistente estendido ([BLA88], [COS90], [COS63]) é capaz de fornecer interpretações para informações inconsistentes, tal como, p e ¬p. [ENE99], enfatiza que a presença de tais interpretações pode apresentar, de forma explícita, a inconsistência nos dados e expressar informações importantes para a tomada de decisão.

O nosso estudo portará sobre a Lógica Evidencial Paraconsistente que é um tipo de Lógica Paraconsistente que se fundamenta nos resultados da Lógica Evidencial. Neste escopo, será examinado a Lógica Evidencial Paraconsistente no intuito de formarmos uma base conceitual para a representação explícita e processamento de dados inconsistentes, assim como fornecer interpretações a informações que poderão também portar inconsistências. Tal iniciativa pode também ser encontrada em [ANG01].

3.1. Lógica Evidencial Paraconsistente

Nós seres humanos, em geral, tomamos nossas decisões palpadas em evidências. Estas evidências desempenham um papel fundamental em nossas vidas e em nossos negócios. Empiricamente, quando um indivíduo precisa tomar uma decisão e se encontra na presença de informações imprecisas/inconsistentes, então ele pode pensar em todas as possibilidades, analisar seus resultados e optar por aquilo lhe parece mais verdadeiro.

Deve-se enfatizar que as crenças humanas nem sempre são verdadeiras [HIN63]. Tal imprecisão pode ser conseqüência da incapacidade dos seres humanos em abstrair e

relacionar uma informação e sua interpretação em um universo complexo. A complexidade se caracteriza aqui em função do número de variáveis a ser levado em conta para interpretar uma situação. Quando este número é elevado (acima de 8), requer-se então o uso de ferramentas apropriadas para vencer o número de combinações possíveis entre as variáveis/dimensões analisadas. Além disso, deve-se também dispor de procedimentos para levar em conta os dados inconsistentes e raciocinar sobre evidências.

Em [SUB87b] são apresentados os fundamentos para o uso do raciocínio evidencial em programação lógica. Semelhante raciocínio porta sobre valores-verdade compostos por dois fatos evidenciais pertencentes a um reticulado $\{x\in\Re|0\leq x\leq 1\}x\{x\in\Re|0\leq x\leq 1\}$ (Figura 3.1). Cada um destes fatores são associados a uma proposição p e podem ser interpretados como a evidência favorável a proposição p e o outro a evidência contrária a proposição p. Deve-se notar que a única restrição porta sobre o fato que os valores devem pertencer ao intervalo [0,1]. Ou seja, infinitos valores podem ser associados às premissas de um sistema. Isto permite usar a teoria dos átomos anotados para raciocinar sobre evidências ([SUB87a], [SUB87b], [BLA88]).

Definição 1: Se ρ é um átomo e $\mu 1,\nu 1 \in [0,1]$, então dizemos que $\rho:[\mu 1,\nu 1]$ é um átomo anotado evidencialmente. Nestè caso, ρ chama-se parte atômica de $\rho:[\mu 1,\nu 1]$ enquanto $[\mu 1,\nu 1]$ chama-se anotação evidencial de $\rho:[\mu 1,\nu 1]$. $\mu 1$ denomina-se evidência favorável de ρ e $\nu 1$ a evidência contrária de ρ.

Pode-se então definir formalmente um reticulado infinito $\tau = <|\tau|, \leq>$, onde $\tau = \{x\in\Re|0\leq x\leq 1\}x\{x\in\Re|0\leq x\leq 1\}$.

[ABE91] ilustra o reticulado τ na forma de uma diagrama de Hasse (Figura 3.1).

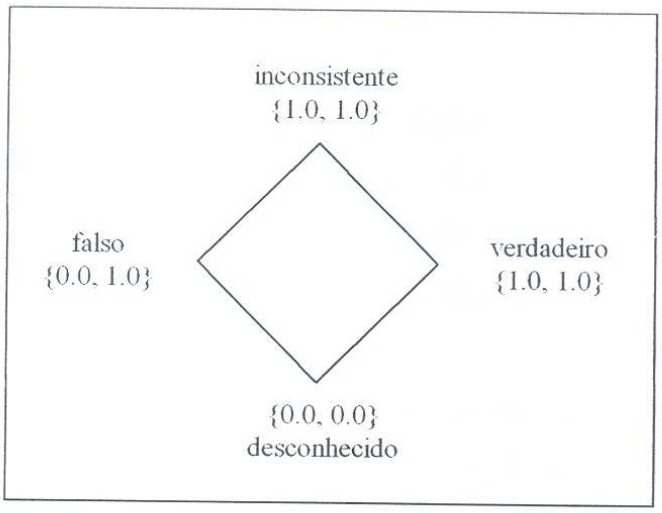

Figura 3.1: Reticulado Infinitamente Valorado τ.

Os valores [1.0,0.0], [0.0,1.0], [0.0,0.0], [1.0,1.0] do reticulado τ da Figura 3.1 correspondem respectivamente a: verdadeiro, falso, desconhecimento e inconsistente. Deve-se destacar, além das interpretações de falso, verdadeiro e inconsistente, a interpretação de desconhecido ([0.0,0.0]).

Diferentes autores trabalharam sobre a construção e inferência de programas lógicos evidenciais, dentre eles podemos citar ([PRA96], [AVI96], [ENE99], [SUB87b], [BLA88]). Mais recentemente o trabalho de Elaini S. Angelotti [ANG01] tem aplicado a Programação Lógica Evidencial Paraconsistente na implementação do Sistema Pandora. Este último é um sistema multi-agente voltado ao tratamento automática de imagens de cheques manuscritos brasileiros. Ela mostrou, em particular, a eficiência da Programação Lógica Evidencial Paraconsistente no tratamento de informações inconsistentes geradas e trocas pelos agentes: Numérico e Extenso.

As definições são:

Negação: O operador de negação $\sim:|\ \tau\ | \rightarrow |\ \tau\ |$ é definido como: $\sim:[\mu 1,\nu 1] = [\nu 1,\mu 1]$.

Outros valores do reticulado são:

$\perp$ indica o mínimo de $\tau = (0.0, 0.0)$;

T indica o máximo de $\tau = (1.0, 1.0)$;

sup indica a operação supremo; e

inf indica a operação de ínfimo.

Conjunção: A conjunção de $[\mu 1,\nu 1] \wedge [\mu 2,\nu 2] \in \tau$ é definida como: $[\mu 1,\nu 1] \wedge [\mu 2,\nu 2] = [min(\mu 1, \mu 2) \wedge Max(\nu 1,\nu 2)]$

Disjunção: A conjunção de $[\mu 1,\nu 1] \vee [\mu 2,\nu 2] \in \tau$ é definida como: $[\mu 1,\nu 1] \vee [\mu 2,\nu 2] = [max(\mu 1, \mu 2) \wedge min(\nu 1,\nu 2)]$

Grau de Certeza: O grau de certeza $c:|\tau| \rightarrow |\tau|$ é dado por: $c([\mu j,\nu j]) = \mu j - \nu j$

3.1.1. Sintaxe

Literal Evidencial: Se p é uma fórmula básica $\mu,\nu \in \Re|0 \leq x \leq 1\}$, então dizemos $p:[\mu,\nu]$ é um literal bem anotado e que $[\mu,\nu]$ é a anotação de p.

Cláusula Evidencial: Se $p0:[\mu 0,\nu 0]$, ..., $pn:[\mu n,\nu n]$ são literais bem anotados então:

$p0:[\mu 0,\nu 0] \Leftarrow p1:[\mu 1,\nu 1] \wedge,..., \wedge pn:[\mu n,\nu n]$

chama-se cláusula evidencial, onde $p0:[\mu 0,\nu 0]$ é a cabeça, enquanto $p1:[\mu 1,\nu 1] \wedge,..., \wedge pn:[\mu n,\nu n]$ é o corpo da cláusula.

Unificação: Se $p1:[\mu 1,\nu 1]$ e $q2:[\mu 2,\nu 2]$ são literais então dizemos que : $p1:[\mu 1,\nu 1]$ e $q2:[\mu 2,\nu 2]$ são unificáveis se p e q são unificáveis.

Programa Lógico Evidencial: é qualquer conjunto finito não-vázio de cláusulas evidenciais.

[FAB99] destaca que, diferentemente de outras teorias de raciocino quantitativo como na teoria da probabilidade, os fatores evidenciais em programas lógicos

evidenciais não estão relacionados, ou seja, os referidos fatos evidenciais operam de forma independente.

3.1.2. Interpretações

[VAN86] e [SUB87a] mostram que todas as interpretações têm como domínio à base de Herbrand, o que significa afirmar: o universo de indivíduos da interpretação são os termos básicos da linguagem que está sendo interpretada. E uma interpretação é uma função $I:BE \rightarrow \tau$, onde BE é a base Herbrand e τ é o respectivo reticulado.

> **Interpretação:** uma interpretação I satisfaz um programa lógico evidencial se ela satisfaz todas as cláusulas evidenciais de E. Portanto, I é um modelo de E.

A ordenação $\leq$ para interpretações é semelhante à ordenação estabelecida sobre as constantes anotacionais, e é descrita da seguinte forma: $I1 \leq I2$ sse $(\forall p \in BE)\ I1(p) \leq I2(p)$, onde BE é a base Herbrand do programa lógico evidencial.

Se E é um programa lógico evidencial, define-se TE como uma aplicação do conjunto de interpretações Herbrand de E em interpretações Herbrand de E, onde $TE = \sup \{[\mu,\nu] \mid p:[\mu,\nu] \Leftarrow q1:[\mu1,\nu1] \wedge ... \wedge qk:[\mu k,\nu k]$ é uma instância básica de uma cláusula evidencial em E e $I \models q1:[\mu1,\nu1] \wedge ... \wedge qk:[\mu k,\nu k]$. Um programa lógico evidencial é bem-comportado se as cláusulas evidenciais de E satisfazem a seguinte condição: Se C1 C2 são duas cláusulas evidenciais em E, sendo suas cabeças $p1:[\mu1,\nu1]$ e $p2:[\mu2,\nu2]$ são unificáveis, então $\max(\mu1, \mu2) + \max(\nu1, \nu2) < 1$.

Um modelo I que atribui valores-verdade $[\mu,\nu]$ ao átomo p, sendo $\mu + \nu > 1$ é dito sobredeterminado. Um modelo I do programa lógico evidencia é dito:

- Correto em relação ao átomo $p \in BE$, se $I(p) = [\mu,\nu]$ e $\mu + \nu \leq 1$.
- Completo em relação ao átomo $p \in BE$, se $I(p) = [\mu,\nu]$ e $\mu + \nu \geq 1$.
- Completo e/ou correto, se é completo e/ou correto em relação a átomo $p \in BE$.

Um programa lógico evidencial E é dito:

- (sobre τ) inconsistente por negação se existe alguma cláusula evidencial p:[μ,ν], p $\in$ BE, tal (1) TE $\uparrow$ w |= p:[μ,ν] e (2) TE $\uparrow$ w |= p:[ν, μ]. Isto quer dizer que a iteração TE sobre um limite ordinal w satisfaz (|=) tanto p:[μ, ν] quanto p:[ν, μ];
- Não trivial se existe alguma cláusula evidencial p:[μ, ν] tal que TE $\uparrow$ w não satisfaz p:[μ,ν];
- Paraconsistente se E for inconsistente por negação e não trivial;

3.1.3. Grau de Inconsistência e Subdeterminação

O Cálculo do grau de I/S, definido em [SUB87a], permite mapear os fatores evidenciais favoráveis e contrários em apenas um único valor a inconsistência ou subdeterminação da informação analisada. Para ilustrar, podemos transferir o reticulado da Figura 3.1 para o plano cartesiano (Figura 3.2).

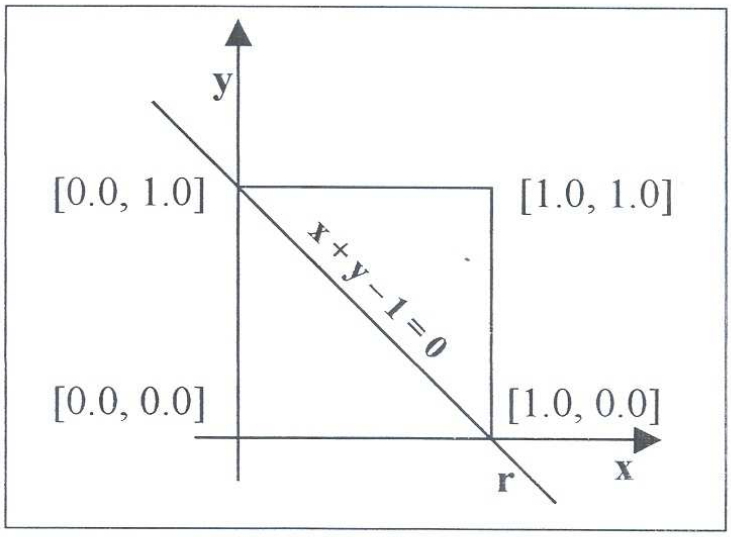

Figura 3.2: Reticula no plano cartesiano.

Uma cláusula evidencial p:[$\mu 1$,$\nu 1$] é dita:

- Perfeitamente definida se $\mu 1+\nu 1 = 1$;
- Subdeterminada (ou subdefinida) se $\mu 1+\nu 1 < 1$; e
- Sobredeterminada (ou inconsistente/sobredefinida) se $\mu 1+\nu 1 > 1$.

O grau de Inconsistência/subdeterminação (I/S) de uma cláusula evidencial é definido através de $\tau \rightarrow [-100, 100]$, onde $I/S = |(\mu 1+\nu 1 - 1)| * 100$. Este grau é importante, como já foi supracitado, para mapear os fatores evidenciais favoráveis e contrários em apenas um único valor a inconsistência ou subdeterminação da informação analisada.

3.2. Considerações Finais

A Lógica Evidencial Paraconsistente é capaz de fornecer interpretações para informações contraditórias. Tais interpretações podem apresentar, de forma explícita, a inconsistência nos dados e expressar informações importantes para a tomada de decisão ([AVI96], [ENE99]). Deve-se enfatizar o fato que se pode obter infinitos valores-verdade possíveis para uma premissa, além de permitir quantificar a inconsistência e avaliar a qualidade das informações por meio de um cálculo simples (I/S).

Como em [ANG01], a Lógica Evidencial Paraconsistente será aplicada na definição e implementação de mecanismos para a avaliação/interpretação das informações locais e não locais. As não locais representam o que os agentes de uma aplicação trocarão no intuito de verificar suas hipóteses para interpretação de um objeto.

Capítulo 4

Estudo de Caso

Este sistema (protótipo) implementa a arquitetura *Multicheck*. Tal arquitetura consiste de agentes cognitivos voltados à análise e tratamento de imagens de cheques bancários brasileiros manuscrito [SCA98]. Ela define quatro tipos de agentes:

- Agente segmentação, que identifica, extrai e cria um modelo lógico de um cheque (e.g., data, assinatura, montante extenso e montante numérico);
- Agente de reconhecimento, que reconhece os diferentes campos lógicos extraídos de um cheque (numérico, literal/extenso, data e assinatura);
- Agente analisador, que aceita ou rejeita um cheque. A tarefa executada consiste em verificar se todos os agentes de reconhecimento obtiveram uma interpretação positiva ou não sobre um mesmo cheque. A informação é armazenada na base de dados de cheques aceitos ou rejeitados.
- Agente gerenciador, que monitora a evolução da rede de agentes no intuito de balancear a carga do sistema.

A capacidade de reconhecer padrões (sobre segmentos de imagens) está presente apenas nos agentes: data, assinatura, numérico e extenso. Os conhecimentos para interpretar os padrões aparecem em todos os agentes, exceto nos agentes segmentação e gerenciador. A aceitação ou rejeição de um cheque é feita pelo agente analisador, que valida as informações recebidas dos agentes de reconhecimento.

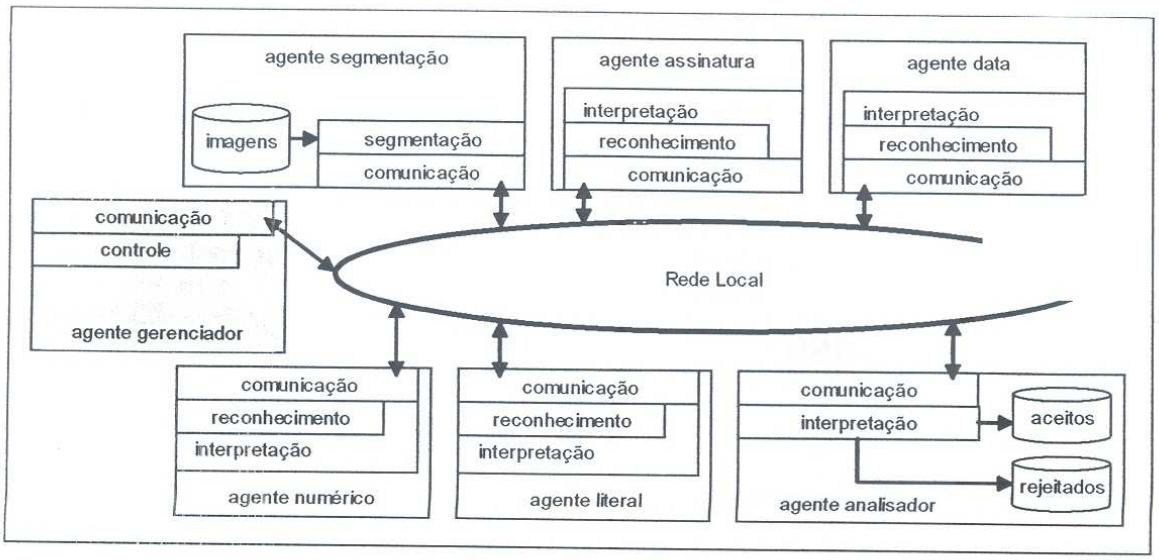

Figura 4.1: Arquitetura *Multicheck*.

Deve-se destacar a função do agente gerenciador:

- inserir um ou mais agentes para aumentar o poder de processamento; ou
- retirar um ou mais agentes para liberar recursos em excesso.

Estas decisões são tomadas levando em conta o tempo médio gasto pelos agentes para concluir seus cálculos sobre uma determinada tarefa de reconhecimento e considerando o tamanho médio da fila de tarefas dos agentes de segmentação.

Neste trabalho é proposto um novo agente denominado Agente Interface que serve como ponto de entrada de dados e gerador de demanda para o sistema. Ele é quem recebe os cheques que devem ser validados e os repassa ao sistema. E é ele também quem recolhe as respostas e às armazena e/ou às devolve ao usuário. A Figura 4.2 mostra a nova visão da arquitetura do sistema.

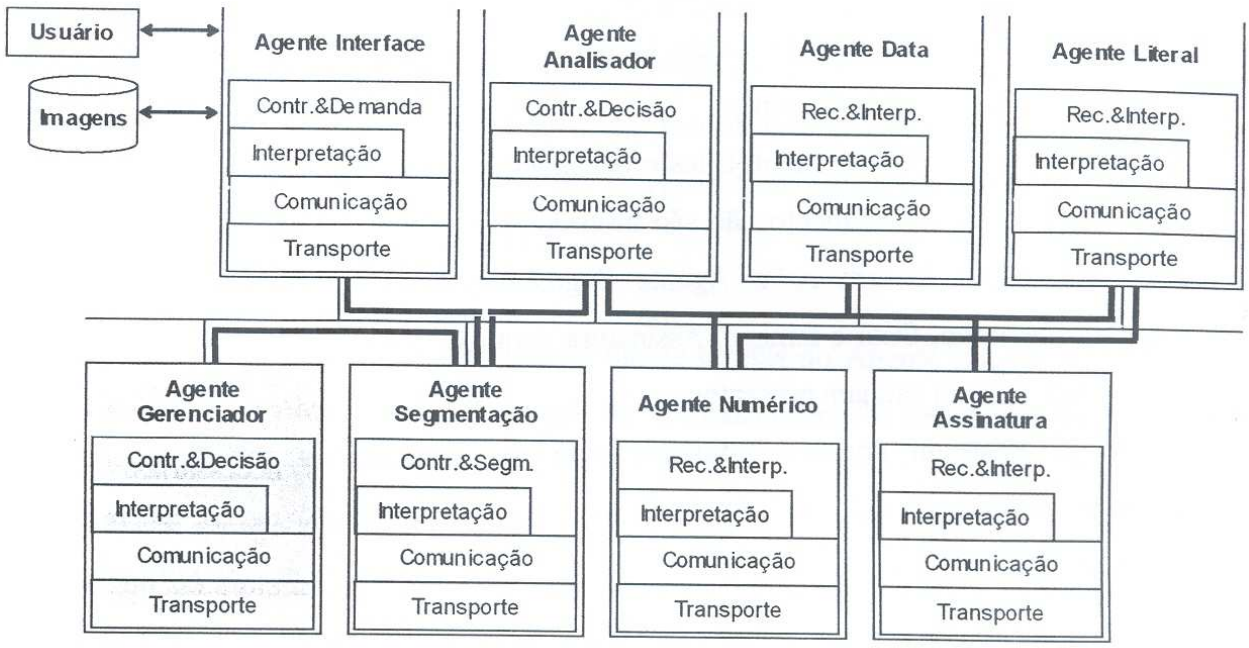

Figura 4.2: Nova Arquitetura Proposta. A camada de interpretação apresentada é utilizada para interpretar as mensagens que chegam ao agente. A interpretação dos resultados é implementada na 1ª. Camada.

O Agente Interface interage com vários agentes segmentação. Cada Agente Segmentação interage com vários agentes analisadores e cada Agente Analisador comunica-se com um único Agente Numérico, Literal/Extenso, Data e Assinatura, conforme mostra a Figura 4.3

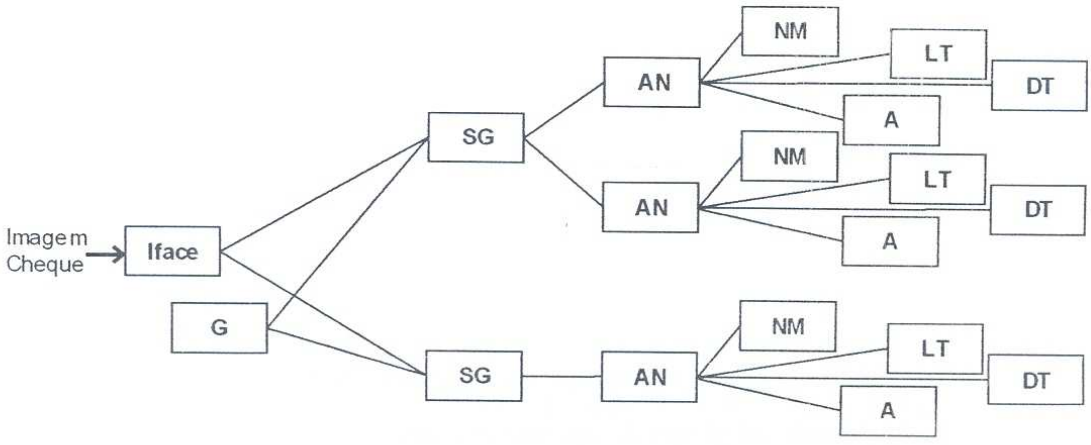

Figura 4.3: Relação de quantidades e dependências entre os agentes do sistema. Iface, agente interface; G, agente gerenciador; SG, agente segmentação; AN, agente analisador; NM, agente numérico; LT, agente literal; DT, agente data e A, agente assinatura.

O Agente Gerenciador representa as informações para suas tomadas de decisões na forma de pares ordenados: < i, t >. Onde, i é um agente e t é o tempo gasto pelo mesmo para concluir sua tarefa de reconhecimento.

Segundo a relação de quantidades e dependências expressas na Figura 4.3, os agentes de segmentação e reconhecimento não são inseridos individualmente no sistema, mas sim em grupos. A inserção de um novo Agente Segmentação exigirá a inserção dos agentes Analisador, Numérico, Data e também Assinatura para que todas as competências necessárias para o reconhecimento estejam presentes.

Se já existirem agentes instalados com as competências necessárias, o agente gerenciador monta uma rede de reconhecimento e a recomenda ao agente interface. Caso contrário, o Agente Gerenciador pode solicitar ao gerenciador da arquitetura em que o sistema multi-agente está implementado, para que sejam instalados mais agentes com as competências necessárias.

Como podem existir vários grupos de agentes participando do sistema, o agente gerenciador leva em consideração a média dos tempos gastos por eles.

Exemplo:

i/tg1 = {<numerico-1, 80s>, <literal-1, 90s>, <data-1, 30s>, <assinatura-1, 32s>}
i/tg2 = {<numerico-2, 50s>, <literal-2, 75s>, <data-2, 40s>, <assinatura-2, 55s>}

As decisões são baseadas no valor de βi e na regra A.

$T = \{ (80 + 90 + 30 + 32) / 4, (50 + 75 + 40 + 55) / 4 \}$

$\rightarrow T = \{ 58s, 55s \}$

$TM = 56.5s$

Calcula-se βi e aplica-se a regra A:

$\beta i = (TM / Min(T)) - 1$

$\beta = 0.02$

Onde, TM é o tempo médio de execução dos grupos de agentes, e Min(T) é o menor tempo médio com que um grupo de agentes executa.

Regra A: SE <βi é maior que 0> ENTÂO <inserir βi redes de reconhecimento e recomendá-las ao Agente Interface>

Uma rede de reconhecimento é composta pelos agentes Segmentação, Analisador, Numérico, Literal/Extenso, Data e Assinatura; e está associada ao Agente Interface através do Agente Segmentação. Já, uma sub-rede de reconhecimento é comporta pelos agente Analisador, Numérico, Literal/Extenso, Data e Assinatura; e está associada a um Agente Segmentação através do Agente Analisador.

Regra B: SE <número de cheques na fila de tarefas do Agente Interface é maior que 50> ENTÃO <inserir uma nova rede de reconhecimento e recomendá-la ao Agente Interface>

Regra C: SE <número médio de cheques na fila de tarefas dos agentes Segementação é maior que 50> ENTÃO <inserir uma nova sub-rede de reconhecimento e recomendá-la ao Agente Segmentação que possui a maior fila de tarefas>

Regra D: SE <(TM < min(T))> ENTÃO <retirar a sub-rede de reconhecimento que leva mais tempo para realizar uma tarefa>

A principal vantagem da arquitetura Multicheck reside no fato que os agentes são independentes (possuem objetivos compatíveis, recursos e competências suficientes). Estes agentes são entidades capazes de satisfazer seus objetivos, interagindo entre si e raciocinando sobre crenças [CAS90], tornando o processo de interpretação de um cheque mais robusto, além de permitir que as etapas do tratamento possam se repetir.

4.1. Reconhecimentos dos campos lógicos de um cheque

O reconhecimento de um campo lógico de um cheque segue um fluxo bem determinado. O módulo de reconhecimento de um dado agente recebe um segmento de imagem σi que corresponde à um campo lógico de um determinado cheque. Tal segmento é decomposto em átomos σij. Estes átomos são classificados por meio de classificadores altamente especializados e suas saídas são no seguinte formato: $< \sigma ij \in Nk : [\mu j, \nu j] >$, onde $\mu j, \nu j \in [0,1]$, e representam respectivamente os coeficientes de evidência favorável e contrária em relação a classe que pertence um dado σij. Nk são as classes possíveis.

Algoritmo de reconhecimento

Para efeito de simplicidade, consideraremos apenas os agentes: Numérico, Extenso e Analisador.

Seja $\sigma 1$ a imagem do campo lógico referente o montante numérico de um cheque C. $\sigma 1j$ os valores de evidência favorável e contrária para cada átomo de $\sigma 1$. $\chi 1j$ os graus de certeza dos átomos de $\sigma 1j$. Destacamos que os dados que serão utilizados, em toda esta sub-seção, para descrever o algoritmo foram retirados de [ANG01].

Tabela 4.1 : Segmento de imagem, graus de evidências favoráveis e contrárias, e graus de certeza.

$\sigma 1j$	$\chi 1j$
< 1 : [0.96, 0.01] >	0.95
< 7 : [0.86, 0.01] >	0.85
< , : [0.70, 0.25] >	0.45
< 3 : [0.85, 0.36] >	0.49
< 1 : [0.70, 0.30] > .	0.40

A leitura dos dados presentes na Tabela 4.1 se faz da seguinte maneira para $\sigma 11$: existe uma evidência favorável de no máximo 96% que o primeiro átomo seja o dígito "1" e uma evidência contrária de no máximo 1% que este primeiro átomo não seja um "1". $\chi 11$ é o grau de certeza que o primeiro átomo analisado seja um "1".

Um grau de certeza é dado pela seguinte formula:

$$c([\mu j, \nu j]) = \mu j - \nu j = \chi ij$$

Tem-se, portanto, um grau de certeza χij associado a cada segmento σij classificado. Os graus de certeza serão utilizados para decidir, por exemplo, quando um agente deve comunicar com os demais agentes do sistema. Em [ANG01] definiu quatro regras aplicáveis aos agentes de reconhecimento, em particular, aos agentes Numérico e Extenso.

Regra 1: Se $< \chi ij \in [0, 50] >$ então solicitar uma nova segmentação.

Regra 2: Se < solicitação de nova segmentação fracassar > então concluir que o montante não pode ser reconhecido e enviar tal resultado a todos os agentes, em particular, ao agente analisador

Regra 3: Se < $\chi ij \in (50, 90]$ >então solicitar informações ao agente literal ou numérico para aumentar χij

Regra 4: Se < $\min(\chi ij) \in (90, 100]$ > então enviar o resultado ao analisador e demais interessados

O agente analisador implementa uma regra especifica para classificar um cheque como reconhecido ou não.

Regra 5: Se < um ou mais campos lógicos não puderam ser interpretados corretamente > então rejeitar o cheque senão aceitar o cheque

A expressão "não pôde ser interpretado corretamente" será inferida a partir do cálculo do coeficiente de I/S (Inconsistência/Subdeterminação) que será mostrado no final desta sub-seção.

Um agente busca interagir quando ele não consegue reconhecer o campo lógico de sua competência. As ações que ele pode tomar são as seguintes:

- Solicitar ao agente segmentação uma nova extração do campo lógico em questão;
- Tentar validar uma crença solicitando informações a um outro agente; e
- Notificar todos os agentes do sistema que o campo lógico de sua competência não pôde ser reconhecido.

A cooperação entre os agentes será no sentido de obter uma informação, a qual poderá gerar novos coeficientes evidenciais. Estes serão obtidos por sucessivas combinações de coeficientes evidenciais. É importante destacar que, primeiramente, a combinação ocorre apenas com dados gerados pelo próprio agente, ou seja, dados locais.

Aplicando a Regra 1 sobre o exemplo da Tabela 4.1, observa-se que o terceiro, quarto e quinto componente de $\sigma 1$ foram reconhecidos com graus de certeza menores de 50%. A

conclusão da regra indica a solicitação de uma nova segmentação do campo lógico em questão. A Tabela 4.2 apresenta uma nova segmentação $\sigma 2$ para o campo lógico numérico.

Tabela 4.2 : Segunda segmentação do montante numérico, graus de evidências favoráveis e contrárias, e graus de certeza.

$\sigma 2j$	$\chi 2j$
< 1 : [0.99, 0.02] >	0.97
< 1 : [0.98, 0.01] >	0.97
<, : [0.90, 0.11] >	0.79
< 3 : [0.80, 0.23] >	0.57
< 1 : [0.99, 0.40] >	0.59

Combinação de valores de crença.

Regra 6: Se < $\sigma 1j$, $\sigma 2j \in (90, 100]$ > então aplicar o operador supremo (sup) sobre $\chi 1j$ e $\chi 2j$.

Regra 7: Se < $\sigma 1j$, $\sigma 2j \in (50, 90]$ > então aplicar o operador supremo (sup) sobre $\chi 1j$ e $\chi 2j$.

Regra 8: Se < $\sigma 1j$, $\sigma 2j \in (0, 50]$ > então aplicar o operador supremo (sup) sobre $\chi 1j$ e $\chi 2j$.

Regra 9: Se < $\sigma 1j$, $\sigma 2j$ não pertence uma mesma classe > então iniciar processo de troca de informações entre os agentes Numérico e Extenso.

Aplicando as Regras 6, 7 e 8 se obtém como resultado os valores da Tabela 4.3.

Tabela 4.3 : Segmento de imagem, graus de evidências favoráveis e contrárias, graus de certeza.

(σ1j, σ2j)	(χ3j)
< 1 : [0.99, 0.02] >	0.97
< 7 : [0.86, 0.01] >	0.85
< , : [0.90, 0.11] >	0.79
< 3 : [0.80, 0.23] >	0.57
< 1 : [0.99, 0.40] >	0.59

A Regra 9 desencadeia uma busca de informação para descobrir qual classificação é a mais correta. Isto ocorrer porque σ1j, σ2j não pertencem a uma mesma classe, ou seja, σ12, σ22 pertencem respectivamente as classes (90, 100] e (50, 90]. Desta forma, as informações da Tabela 4.3 serão objeto de validação, rejeição ou combinação de acordo com os resultados obtidos, por exemplo, pelo agente Extenso.

No contexto de uma aplicação voltada à compensação automática de cheques requer-se a interação entre os agentes Numérico e Extenso para obter-se uma mesma interpretação da informação. Em outras palavras, tais agentes devem obter exatamente a mesma informação sobre campos lógicos distintos (codificados em formatos diferentes). Pode-se obter resultados conflitantes e serem levados a interagir para obter uma interpretação consistente e aumentar seus graus de certezas [ANG00].

Considerando que os agentes Numérico e Extenso já concluíram isoladamente suas tarefas de reconhecimento, pode-se aplicar a Regra 3.

Tabela 4.4 : Graus de evidências favoráveis e contrárias, graus de certeza para os montantes numérico e extenso.

Informações obtidas pelo agente Numérico (σ1j, σ2j)	(χ3j)
< 1 : [0.99, 0.02] >	0.97
< 7 : [0.96, 0.01] >	0.95
< , : [0.90, 0.11] >	0.79
< 3 : [0.80, 0.23] >	0.57
< 1 : [0.99, 0.40] >	0.59

Informações obtidas pelo agente Extenso (σ1j, σ2j)	(χ3j)
< onze : [0.89, 0.04] >	0.85
< reais : [0.90, 0.04] >	0.86
< trinta : [0.93, 0.06] >	0.87
< um : [0.91, 0.04] >	0.87
< centavos : [0.88, 0.06] >	0.82

A avaliação da qualidade das informações produzidas pelos agentes será feita por meio da aplicação dos seguintes operadores:

- Disjunção, que permite combinar valores para aumentar um determinado grau de crença;
- Conjunção, que permite avaliar um conjunto de valores sobre um dado campo lógico e gerar um único valor.
- Grau de certeza, que permite estudar individualmente cada parte segmentada de um montante; e
- Grau de inconsistência/subdeterminação, que permite mapear, em apenas um único valor, a inconsistência ou subdeterminação da informação.

Para aumentar os graus de certeza dos três últimos valores do campo numérico (Tabela 4.4), aplica-se o operador disjunção:

$[0.90,0.11]\vee[0.89,0.04]\vee[0.90,0.04]\vee[0.93,0.06]\vee[0.91,0.04]\vee[0.88,0.06] = [0.93,0.04]$

$[0.80,0.23]\vee[0.89,0.04]\vee[0.90,0.04]\vee[0.93,0.06]\vee[0.91,0.04]\vee[0.88,0.06] = [0.93,0.04]$

$[0.99,0.40]\vee[0.89,0.04]\vee[0.90,0.04]\vee[0.93,0.06]\vee[0.91,0.04]\vee[0.88,0.06] = [0.99\ 0.04]$

Tabela 4.5 : Graus de evidências favoráveis e contrárias, graus de certeza obtidos após aplicação do operador de disjunção.

Informações obtidas pelo agente Numérico ($\sigma 1j$, $\sigma 2j$)	($\chi 3j$)
< 1 : [0.99, 0.02] >	0.97
< 7 : [0.96, 0.04] >	0.95
< , : [0.93, 0.04] >	0.89
< 3 : [0.93, 0.04] >	0.89
< 1 : [0.99, 0.04] >	0.95

Informações obtidas pelo agente Extenso ($\sigma 1j$, $\sigma 2j$)	($\chi 3j$)
< onze : [0.89, 0.04] >	0.85
< reais : [0.90, 0.04] >	0.86
< trinta : [0.93, 0.06] >	0.87
< um : [0.91, 0.04] >	0.87
< centavos : [0.88, 0.06] >	0.82

Para obter um fechamento sobre um montante, aplica-se pura e simplesmente o operador conjunção.

Agente Numérico:

$[0.99, 0.02]\wedge[0.96, 0.01]\wedge[0.93, 0.04]\wedge[0.93, 0.04]\wedge[0.99, 0.04] = [0.93, 0.04]$

Agente Extenso:

$$[0.89, 0.04] \wedge [0.90, 0.04] \wedge [0.93, 0.06] \wedge [0.91, 0.04] \wedge [0.88, 0.06] = [0.88, 0.06]$$

A aplicação do operador de conjunção nos permite descobrir inconsistências. Este processo seleciona o mínimo das evidências favoráveis e o máximo das evidências contrárias. Para o caso acima, tem-se os seguintes reconhecimentos: o agente numérico reconhece <17,31 : [0.93, 0.04]> e <dezessete reais e trinta e um centavos: [0.88, 0.3]>.

Os resultados obtidos pelos agentes numérico e extenso devem ser enviados ao agente analisador, o qual deverá calcular um grau de inconsistência/subdeterminação no intuito de decidir se o cheque será aceito ou rejeitado.

Estes cálculos são efetuados em duas etapas:

- Primeiramente aplica-se o operador de conjunção para obter um único valor de inconsistência ou subdeterminação; e
- Aplica-se a formula do cálculo de I/S.

Obtém-se então respectivamente:

$$[0.93, 0.04] \wedge [0.88, 0.06] = [0.88, 0.06]$$

$$|0.88 + 0.06 - 1| * 100 = 6\%$$

Tal resultado significa que as informações de um dado cheque possui 6% de I/S.

Finalmente, podemos aplicar a Regra 9 abaixo.

Regra 9: Se <I/S $\in$ [0%, 5%]> então aceitar o cheque senão rejeitar o cheque.

4.2. Definição de Limiares

A Figura 4.4 apresenta um reticulado no plano cartesiano com 20 regiões. A interpretação de cada uma das regiões é descrita na Tabela 4.6.

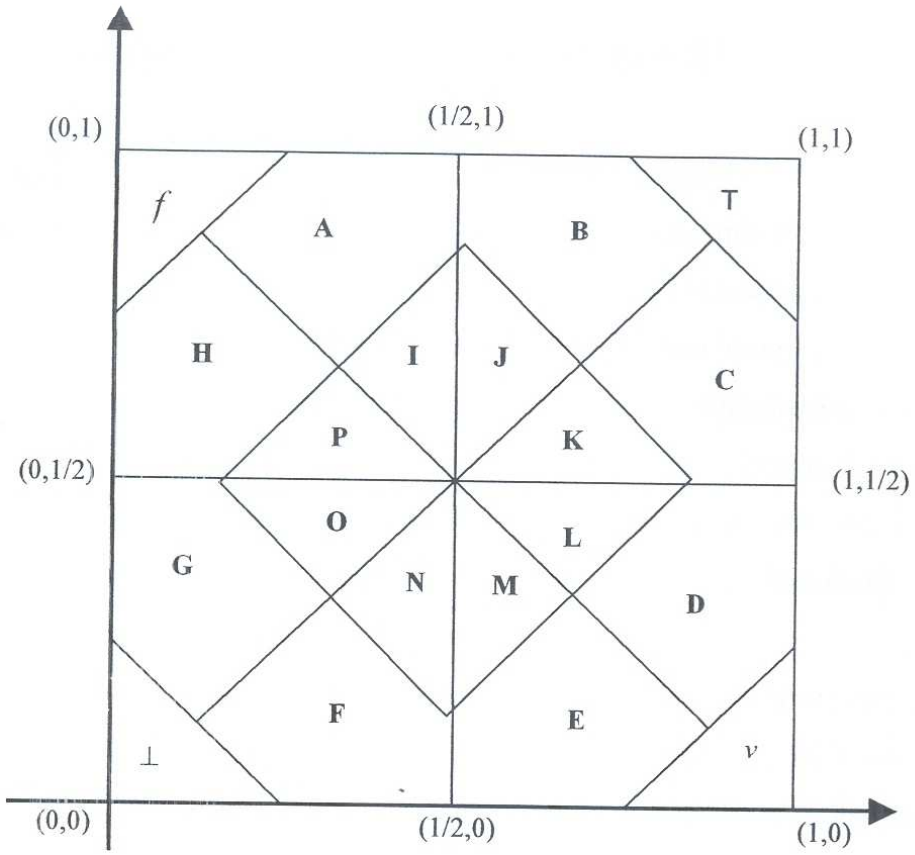

Figura 4.4 : Reticulado no plano cartesiano com um total de 20 regiões.

Tabela 4.6: Reticulado no plano cartesiano com um total de 20 regiões.

Regiões	**Estados resultantes**
A	Quase falso tendendo ao inconsistente
B	Inconsistente tendendo ao falso
C	Inconsistente tendendo ao verdadeiro
D	Quase verdadeiro tendendo ao inconsistente
E	Quase verdadeiro tendendo ao indeterminado
F	Indeterminado tendendo ao verdadeiro
G	Indeterminado tendendo ao falso
H	Quase falso tendendo ao indeterminado
I	Quase quase falso tendendo ao inconsistente
J	Inconsistente tendendo ao quase falso
K	Inconsistente tendendo ao quase verdadeiro
L	Quase quase verdadeiro tendendo ao inconsistente
M	Quase quase verdadeiro tendendo ao indeterminado
N	Indeterminado tendendo ao quase verdadeiro
O	Indeterminado tendendo ao quase falso
P	Quase quase falso tendendo ao indeterminado
F	Falso
V	Verdadeiro

$\perp$	Indeterminado
T	Inconsistente

O interesse porta sobre três estados, que são:

- verdadeiro;
- quase verdadeiro tendendo ao indeterminado e inconsistente (vice-versa); e
- quase falso ao indeterminado e inconsistente (vice-versa).

Quando o reconhecimento de um campo se configura com um grau de certeza:

- Entre (90%, 100%] considera-se que a interpretação é verdadeira. O agente de reconhecimento envia o resultado ao agente Analisador.
- Entre (50%, 90%] considera-se que a interpretação é quase verdadeira tendendo ao inconsistente e indeterminado (vice-versa). O agente de reconhecimento deve trocar informações com os outros agentes de reconhecimento no intuito de elevar o grau de certeza para um valor superior a 90%.
- Entre [0%, 50%] considera-se que a interpretação é quase falsa tendendo ao inconsistente e indeterminado (vice-versa). O agente de reconhecimento deve solicitar ao agente segmentação uma nova extração do campo em questão no intuito de elevar o grau de certeza para um valor (pelo menos) superior a 50%.

Estes três limiares foram introduzidos na base de conhecimento dos agentes de reconhecimento dos campos numérico e extenso por meio das regras 1, 3 e 4 apresentadas anteriormente.

4.3. Resultados

Para o teste da aplicação foi utilizada uma base com 470 imagens. As probabilidades de reconhecimento, tanto do campo numérico quanto do campo literal de cada cheque, foram obtidas a priori em uma etapa de pré-processamento.

A principal dificuldade encontrada foi gerar os graus de crença e descrença a partir das probabilidades retornadas pelos algoritmos de reconhecimento de imagens utilizados ([FRE01], [OLI00], [OLI02] e [OJR02]), principalmente para o extenso.

O algoritmo de reconhecimento do campo numérico ([OLI00], [OLI02]), retorna probabilidades de reconhecimento para cada digito encontrado na imagem:

Tabela 4.7: Probabilidades de reconhecimento geradas para os campos numéricos.

Imagem: te0001 [5 8 . 9 4 0 , 1 0]		
Dígitos	**Maior Probabilidade**	**Crença/Descrença**
5	0.997010	< 5, 1.00, 0.00 >
8	0.992128	< 8, 0.99, 0.01 >
.	0.997864	< ., 1.00, 0.00 >
9	0.991901	< 9, 0.99, 0.01 >
4	0.989628	< 4, 0.99, 0.01 >
0	0.990148	< 0, 0.99, 0.01 >
,	0.721407	< ,, 0.72, 0.28 >
10	0.984567	< 10, 0.98, 0.02 >

O algoritmo de reconhecimento do campo literal ([FRE01], [OJR02]), retorna 39 probabilidades de reconhecimento para cada palavra do extenso encontrado na imagem:

Tabela 4.8: Probabilidades de reconhecimento geradas para os campos extensos.

Imagem: te0001 [5 8 . 9 4 0 , 1 0]	
Palavras	**Probabilidades**
cinqüenta	39 valores de probabilidade
oito	39 valores de probabilidade
mil	39 valores de probabilidade
novecentos	39 valores de probabilidade
quarenta	39 valores de probabilidade
reais	39 valores de probabilidade
dez	39 valores de probabilidade
centavos	39 valores de probabilidade

Cada palavra a ser reconhecida passa por 39 modelos (cadeias de *Markov*) que retornam a probabilidade daquela palavra ter sido reconhecida por aquele modelo. Esses modelos são treinados para reconhecer palavras isoladas: 1°. Modelo → um, 2°. Modelo → dois, etc. O principal problema está na composição dos graus de crença e descrença para cada palavra do extenso do cheque.

Dentre as estratégias utilizadas para combinar as probabilidades dos 39 modelos e seus respectivos resultados, obtemos:

- 1a. maior probabilidade e o complemento:
  - Sem troca de informações: 0 aceito /470 rejeitados;
  - Com troca de informações: 3 aceitos/467 rejeitados;
- 1a. maior probabilidade e a 2a. maior probabilidade:
  - Sem troca de informações: 0 aceito /470 rejeitados;
  - Com troca de informações: 7 aceitos/463 rejeitados;
- 1a. maior probabilidade e a média do restante:
  - Sem troca de informações: 2 aceitos/468 rejeitados;
  - Com troca de informações: 40 aceitos/430 rejeitados;
- 1a. maior probabilidade e a 1a. menor:
  - Sem troca de informações: 2 aceitos/468 rejeitados;
  - Com troca de informações: 42 aceitos/428 rejeitados;
- 1a. maior probabilidade e 0.00:
  - Sem troca de informações: 2 aceitos/468 rejeitados;
  - Com troca de informações: 42 aceitos/428 rejeitados;

O que fica nítido nestes resultados é que sempre que há troca de informações entre os agentes numérico e literal, o número de cheques reconhecidos aumenta. Ao mesmo tempo, estes resultados questionam a eficiência da lógica paraconsistente neste contexto e frente às estratégias de composição utilizadas. As duas últimas estratégias apresentam resultados iguais e descartam a utilização do fator de descrença proposto.

Nos artigos referentes ao reconhecimento dos campos numérico e extenso, fala-se de resultados em torno de 98% e 72% respectivamente. Estes resultados não foram alcançados neste trabalho e nenhuma avaliação extra foi efetuada.

Quanto às diferenças nos resultados encontrados em [ANG01] frente a este trabalho, podem ser explicadas pelo fato de que no primeiro existe a simulação de novas segmentações que alteram os valores dos pesos dos cheques não reconhecidos. No contexto deste trabalho não há novas segmentações já que cada algoritmo de reconhecimento propõe e utiliza a sua própria forma de segmentação, única.

4.4. Considerações Finais

A Lógica Paraconsistente é uma alternativa ao tratamento de informações contraditórias sem excluí-las. A implementação é simples, sendo a maior dificuldade a obtenção das evidências favoráveis e contrárias para cada proposição a ser analisada.

No caso de um sistema multi-agente, em particular, no Projeto *Multicheck* a Lógica Paraconsistente permitiu implementar algoritmos para combinar informações de fontes diferentes, porém sobre o mesmo objeto (e.g., campo numérico e extenso de um cheque).

Conclusão e Trabalhos Futuros

Lembramos que o trabalho insere-se no contexto do Projeto *Multicheck*[7]. Tal projeto define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques brasileiros manuscritos. Deve-se destacar que este trabalho é uma continuação do já realizado em [ANG01]. No contexto deste projeto a escolha de uma arquitetura distribuída e aberta baseou-se nos seguintes critérios: (i) trata-se de um problema cuja decomposição em sub-problemas é natural, à medida que cada sub-problema (reconhecimento dos montantes numérico e literal, reconhecimento da data e verificação da assinatura) pode ser representado e implementado como um agente autônomo e independente; (ii) a interação entre os agentes pode tornar o sistema de compensação de cheques mais confiável, à medida que as trocas de informações entre os agentes podem resolver situações difíceis se considerarmos também as informações não locais, por exemplo, os agentes que reconhecem os valores numéricos e extensos dos cheques trocam suas crenças para confirmar ou não suas hipóteses de reconhecimento; e (iii) o paralelismo natural dos sistemas multi-agente pode contribuir de forma considerável na implementação de um sistema de alta performance. Deve-se enfatizar que tais critérios, no contexto desta aplicação, facilitam a decomposição do problema em entidades fracamente acopladas e fortemente coesas, que são os princípios básicos para se construir sistemas eficientes e de fácil manutenção.

Arquitetura Gare

Foi implementada uma plataforma para a construção e à gerência de agentes e sistemas multi-agente (Apêndice A). Tal plataforma dispõe:

- Agente *GareSleep* que deve estar instalado em cada máquina da rede e tem a finalidade de receber e inicializar agentes recebidos de outras máquinas. Fornece

[7] O Projeto *Multicheck* está sendo realizado pela Pontifícia Universidade Católica do Paraná (PUCPR/Brasil), com o apoio financeiro do governo brasileiro (CNPq), e com a colaboração internacional entre *l'École de Technologie Supériieure* (ETS/Canadá) e a PUCPR.

serviços de páginas brancas e amarelas e disponibiliza informações sobre os agentes e também sobre a carga da máquina;

- Agente *GareManager* que é o gerenciado principal da plataforma. Possui funcionalidades que permitem instalar, movimentar, interagir e remover agentes em uma rede (por intermédio do agente *GareSleep*). Possui serviços de páginas brancas e amarelas e serviço de subscrição e notificação;
- Agente básico (*AgB*) a partir do qual novos agentes podem ser criados. Sua implementação básica provê as camadas de comunicação e transporte, camadas de interpretação de mensagens (páginas brancas e amarelas), *check-in* e *check-out* automáticos da plataforma *Gare*. Os novos agentes criados possuem ainda, as seguintes características:
    - Baixo acoplamento entre os módulos que compõem o agente;
    - Podem ou não estender o agente básico;
    - Podem implementar novas camadas proprietárias de interpretação de mensagens;
    - Podem estender o suporte para mobilidade.

Estudo de Caso

Foi implementado um sistema multi-agente para o reconhecimento e validação de cheques utilizando lógica paraconsistente e dados reais. Este sistema foi implementado sobre a arquitetura *Gare* e implementa parte dos agentes descritos pelo projeto *Multicheck*. Foi implementado o agente Gerenciador proposto em [ANG01].

No contexto da aplicação, a arquitetura multi-agente tem seu grande interesse na construção de um sistema efetivamente modular. Cada módulo é implementado na forma de um agente. Os agentes nos permitem projetar sistemas abertos, onde um agente pode entrar e sair do sistema perturbando-o minimamente. A organização de sistema multi-agente na forma de uma rede contratual facilita o acréscimo e remoção de um agente sem grandes custos. Pode-se também utilizar a figura de um mediador para possibilitar os agentes se descobrirem. Este mecanismo tem o problema de ser um centralizador, e deste fato, tornar-se um potencial gargalo de estrangulamento, em particular, se existe um grande número de agentes no sistema. Um mediador pode ser utilizado quando o número de agentes clientes e fornecedores são pequenos (menos de 50).

A existência de tarefas independentes e cooperativas gera informações inconsistentes. Cada agente gera informações locais que são compartilhadas. Tal compartilhamento, em particular, entre os agentes numérico e extenso, pode requerer mecanismos para tratar informações inconsistentes. A Lógica Paraconsistente é uma alternativa ao tratamento de informações contraditórias sem excluí-las. A implementação é simples, sendo a maior dificuldade a obtenção das evidências favoráveis e contrárias para cada proposição a ser analisada.

A gestão de agentes distribuídos requer um ambiente de software que facilite: a instalação, remoção, movimentação de código. *GARE* implementa tais funcionalidades.

Como conclusões gerais sobre o trabalho, temos:

- Discrepância entre dados simulados e dados reais. Neste trabalho foram utilizadas as mesmas regras de validação de cheques utilizadas em [ANG01] porém, foram utilizados dados reais de reconhecimento. O que pôde ser constatado é que tais pesos foram ajustados de acordo com o conjunto de dados simulados utilizados. Os bons resultados obtidos em [ANG01] também se devem à simulação de novas segmentações o que, no contexto deste trabalho, não é possível devido às técnicas de segmentação e extração de dados empregados pelos algoritmos de reconhecimentos ([FRE01], [OLI00], [OLI02] e [OJR02]). Parece-nos razoável que isso venha a acontecer já que as probabilidades de reconhecimento tendem a variar de acordo com os algoritmos de reconhecimento utilizados. Neste caso, a adoção de uma forma para se calibrar o sistema multi-agente, automática ou não, para o ajuste dos pesos utilizados pela lógica paraconsiste, parece aceitável.
- Forte dependência entre as camadas de reconhecimento e interpretação dos resultados, devido ao fato dos pesos utilizados com a lógica paraconsistente dependerem muito das probabilidades de reconhecimento retornadas pelos algoritmos de reconhecimento de imagens.
- Baixo desempenho e alta complexidade da camada de reconhecimento reforçando a idéia de um ambiente distribuído e paralelo.
- Interação entre os agentes torna o processo de reconhecimento mais robusto. Isto foi apresentado por meio da cooperação entre os agentes Numérico e Extenso;

- Agentes independentes geram informações contraditórias, em particular, quando se trabalha sobre um mesmo objeto. Por exemplo, os agentes Numérico e Extenso reconhecem montantes diferentes para um mesmo cheque.
- Informações contraditórias podem ser tratadas através da Lógica Paraconsistente. Para que este tratamento seja efetivo, é necessário um profundo conhecimento sobre a natureza das informações e suas reais implicações.

Trabalhos futuros

Os trabalhos futuros que nos parecem adequados a realizar são os seguintes:

- Disponibilizar outras formas de alocação para a plataforma *Gare* (direta, delegação, redes de contrato); transformar os agentes *GareSleep* em mediadores locais; desenvolver uma ferramenta gráfica que auxilie na definição e criação de agentes.
- Estudar uma nova forma de combinar as saídas dos reconhecedores para gerar as evidências favoráveis e contrarias.
- Implementar diferentes agentes com a mesma competência para testar diferentes protocolos distribuição de tarefas.
- Introduzir procedimentos de aprendizagem nos agentes, visando dotá-los de um comportamento pró-ativo e adaptável, em particular, para tentar antecipar uma situação onde um determinado cheque será rejeitado.
- Dotar os agentes de mecanismos de aprendizagem no intuito possibilitar os agentes realizarem as tarefas de reconhecimento de forma mais eficiente.

Referências Bibliográficas

[ANG00] ANGELOTTI, E. S.; SCALABRIN, E. E.; ÁVILA, B. C.; BORTOLOZZI, F. *Concepção de um sistema de agentes independentes paraconsistente para o tratamento de cheques bancários brasileiros*. 1o. Congresso de Lógica Aplicada à Tecnologia – Laptec'2000, Faculdades SENAC, São Paulo, setembro de 2000.

[ANG01] ANGELOTTI, E. S. *Utilização da lógica paraconsistente na implementação de um sistema multi-agente*. Dissertação de Mestrado. PUCPR, 2001.

[AVI96] ÁVILA, B. C. *Uma Abordagem Baseada em Lógica Evidencial para tratar Exceções em Sistemas de Frames com Múltipla Herança*. Tese de Doutorado. Escola Politécnica, Universidade de São Paulo, São Paulo, 1996.

[AVI97] ÁVILA, B. C.; ABE, J. M.; PRADO, J. P. A. *Paralog_e: A Paraconsistent Evidencial Logic Programming Language*. Chile/Valparaiso, XVII International Conference of the Chilean Computer Society, IEEE Computer Society Press. P. 2-8, novembro de 1997.

[ABE91] ABE, J. M.; PAPAVERO, N. *Teoria Intuitiva dos Conjuntos*. São Paulo: Makron Books/McGraw Hill, 1991.

[BER91] BARNARD G., STÈVE D., SIMATIC M. *Placement et Migration de Processus dans lês Systèmes Repartis Faiblement Couplés*. In: T.S.I. – Technique et Science Informatiques, vol. 10, no. 5.

[BLA87] BLAIR, H. A.; SUBRAHMANIAN, V. S. *Paraconsistent Logic Programming*. Proc. 7th Conference on Foundations of Software Tecnology and Theoretical Computer Science. Lecture Notes in Computer Science, Springer-Verlag. Vol. 287, p. 340-260. 1987.

[BLA88] BLAIR, H. A.; SUBRAHMANIAN, V. S. *Paraconsistent Foundations for Logic Programming*. Journal of Non-Classical Logic. 1988.

[CAS90] CASTELFRANCHI, C. *A Pont Missed in Multi-Agent*. DAI & HCI. Decentralized AI. Y. Demazeau & J-P Muller (Eds.), Elsevier Science Publisher B. V. (North-Holland), p. 49-62, 1990.

[COS63] DA COSTA, N. C. A. *Calculs Propositionnels pour lês Systèmes Formales Inconsistants*. Compte Rendu Acad. Des Sciences (Paris), 257, p. 3790-3792, França, 1963.

[COS90] DA COSTA, N. C. A.; HENSCHEN, L. J.; SUBRAHMANIAN, V. S. *Automatic Theorem Proving in Paraconsistent Logic: Logics and Implementation*. In: 10th International Conference on Automatic Deduction, Lecture Notes in Computer Science, vol 449, p. 72-86, 1990.

[DAV83] DAVIS, R.; SMITH, R. G. *Negociation as a Metaphor for Distributed Problem Solving*. Artificial Intelligence, 20 (1), p. 63-109, 1983.

[ENE99] ENEMBRECK, F.; ÁVILA, B. C.; SABOURIN, R. *Decision Tree-Based Paraconsistent Learning*. In: Proccedings of XIX International Conference of the Chilean Computer Science Society, p. 32-44, IEEE Computer Society Press, Talca, Chile, 1999.

[FER95] FERBER, J. *Les systèmes Multi-agents: vers une intelligence collective*. InterEditions, Paris, 1995.

[FOX88] FOX, M. S. *An Organizational View of Distributed Systems*. In : Reading Distributed Artificial Intelligence, A.H..Bond & L. Gasser (editors), Morgan Kaufman Publishers, San Mateo, Califórnia, 1988.

[FRE01] FREITAS, C. O. A.; BORTOLOZZI, F.; SABOURIN, R.; *Handwritten isolated word recognition: an approach based on Mutual Information for feature set validation.* Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on, 2001 Page(s): 665 -669.

[HAM98] HAMERSKI JR., E. C. *Gerenciador de agentes reativos no âmbito da gerência de redes.* Bacharelado em Análise de Sistemas - Projeto final de curso. Projeto GIR (Gerência inteligente de redes) [PAR98]. PUCPR, 1998.

[HIN63] HINTIKKA, J. *Knowledge and Bilief.* Cornell University Press, 1963.

[IBM98] IBM Research. *A KQML implementation in Java.* 1998. See web site at: http://www.alphaworks.ibm.com.

[LAB97] LABROU, Y.; FININ, T. *A Proposal for a new KQML Specification.* TR CS-97-03, February 1997, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250, 1997.

[MUT87] MUTKA M. W., LIVNY M. *Profiling Workstations' Available Capacity for Remote Execution.* In: Proc. Performance'87, 12th IFIP WG7.3 International Symposium on Computer Performance, Brusels, December.

[OLI00] OLIVEIRA, L. S.; SABOURIN, R.; BORTOLOZZI, F.; SUEN, C. Y.; *A new segmentation approach for handwritten digits.* Pattern Recognition, 2000. Proceedings. 15th International Conference on, Volume: 2, 2000 Page(s): 323 -326 vol. 2.

[OLI02] OLIVEIRA, L. S.; SABOURIN, R.; BORTOLOZZI, F.; SUEN, C. Y.; *Automatic recognition of handwritten numerical strings: a recognition and verification strategy.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, Volume: 24 Issue: 11, Nov 2002 Page(s): 1438-1454.

[OJR02] DE OLIVEIRA JUNIOR, J. J.; DE CARVALHO, J. M.; FREITAS, C. O. D. A.; SABOURIN, R.; *Evaluating NN and HMM classifiers for handwritten word recognition.* Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on, 2002 Page(s): 210 -217.

[PAR98] PARAISO, E. C.; et al. *Intelligent Management of Computer Networks: The GIR Proposal.* XVIII International Conference of the Chilean Computer Science Society - Antofagasta, Chile, 1998.

[PAR90] PARUNAK, H. V. D. *Distributed AI and Manufacturing Control: Some Issues and Insigthts.* In: Decentralized AI, Vol 1, Y. Demazeau e J-P Muller (Ed.), p. 81-104, North-Holland, 1990.

[PRA96] PRADO, J. P. A. *Uma Arquitetura para Inteligência Artificial Distribuída Baseada em Lógica Paraconsistente Anotada.* Tese de Doutorado. Universidade de São Paulo, São Paulo, 1996.

[SCA96] SCALABRIN, E. E. *Conception et Réalisation d'environnement de développement de systèmes d'agents cognitifs.* Thèse de Doctorat. Université de Technologie de Compiègne, France, 1996.

[SCA96b] SCALABRIN, E. E.; VANDENBERGUE L., DE AZEVEDO H., BARTHÈS J-P. A. *A Generic Model of Cognitive Agent to Develop Open Systems.* In: 13th Brazilian Symposium on Artificial Intelligence , SBIA'96, Díbio L. Borges and Celso A. A. Kaestner (eds.), (Lecture Notes in Artificial Intelligence 1159, Springer), Curitiba, Brazil, October.

[SCA98] SCALABRIN, E. E.; BORTOLOZZI, F.; SABOURIN, R. *Multicheck: Une Architecture d'agents cognitifs independents pour le tratement automatique des cheques bancaires Brésiliens.* In: CIFED'98, Canadá, maio de 1998.

[SHM99] SHMEIL, M. A. H. *Sistemas Multiagentes na Modelação da Estrutura e Relação de Contratação de Organizações*. Tese de Doutorado. Faculdade de Engenharia, Universidade do Porto, Portugal, 1999.

[SMI79] SMITH, R. G. *A framework for Distributed Problem Solving*. In: Proceedings of IJCAI'79, 1979.

[SMI80] SMITH, R. G. *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solving*. IEEE Transaction on Computers, 29(12), p. 1104-1113, 1980.

[SUB87a] SUBRAHMANIAN, V. S. *Towards a Theory of Evidential Reasoning*. Logic Colloquium'87. The European Summer Meeting of the Association for Symbolic Logic, Granada Soain, July of 1987.

[SUB87b] SUBRAHMANIAN, V. S. *On the Semantics of Quantitative Logic Programming*. Proceedings of 4th IEEE Symposium on Logic Programming, San Francisco, September of 1987.

[THE89] THEIMER M. M., LANTZ K. A. *Finding Idle Machines in a Workstation-Based Distributed System*. In: IEEE Trans. on Software Engineering, 15(11), November.

[VAN86] VAN EMDEN, M. H. *Quantitative Deduction and its Fixpoint Theory*. The Journal of Logic Programming, Vol. 1, New York, 1986.

[ZHO88] ZHOU S. *A Trace-Driven Simulation Study of Dinamic Load Balancing*. In: IEEE Trans. on Software Engineering, 14(9). September.

Apêndice A

Arquitetura GARE

GARE (Gerenciador de Agentes Reativos no Âmbito da Gerência de Redes) nasceu como uma aplicação no contexto do Projeto *GIR* (Gerência Inteligente de Redes – convênio PUCPR/SIEMENS) [HAM98] [PAR98]. GARE fornece um conjunto de serviços voltados ao gerenciamento de uma rede de agentes. As novas versões desenvolvidas e ampliadas neste trabalho não se limitam apenas ao gerenciamento de agentes reativos. *GARE* agora se desdobra em uma plataforma mais geral voltada à construção e à gerência de agentes e sistemas multi-agente.

Este apêndice apresenta a arquitetura da plataforma *GARE*, seus componentes e alguns detalhes de implementação. Destacamos que os agentes do protótipo/estudo de caso foram implementados sobre esta arquitetura (Capítulo 4).

A.1. Visão Geral

GARE define um conjunto de bibliotecas de objetos e ferramentas que facilitam a criação e gerência de agentes e sistemas multi-agente. Neste contexto, os agentes não são classificados como reativos ou cognitivos, comunicantes ou não comunicantes, etc., visto que a responsabilidade de desenvolver os mecanismos necessários para os agentes e sistemas que irão caracterizar esta ou aquela categoria de agente é exclusiva do desenvolvedor, cliente das bibliotecas e ferramentas.

A arquitetura *GARE* ainda define:

- Um agente básico, denominado simplesmente AgB, com suporte à comunicação, gerência remota e gerência de mobilidade. Ele pode ser facilmente estendido para a criação de novos agentes;
- Dois agentes complexos criados a partir deste agente básico, chamados *GareSleep* e *GareManager*. ***GareSleep*** possui funcionalidades para receber e iniciar agentes, fornecer a lista de agentes em execução a um agente solicitante, fornecer informações sobre o estado atual da máquina (somente o número de agentes – versão atual) a um agente solicitante; e serviços de *WhitePages* (registro e informação local de agentes), *YellowPages* (registro e informação local de competências), e *subscribe/notify* (notificação de agentes e competências que entram no sistemas aos agentes inscritos). ***GareManager*** possui as funcionalidades para cadastrar novos agentes, cadastrar segmentos de rede, instalar agentes, movimentar agentes, gerenciar e monitorar agentes; e serviços de *WhitePages* (registro e informação global de agentes), *YellowPages* (registro e informação global de competências), *subscribe/notify*. Os agentes diferem basicamente quando a sua amplitude de atuação. O primeiro fornece informações locais (em uma máquina específica) enquanto o segundo abrange todo o sistema.

Estes agentes constituem a base da plataforma e permitem a distribuição e o gerenciamento de agentes em uma rede de computadores.

O agente *GareManager* (Figura A.1), permite visualizar os nós da rede na forma de uma árvore. A visualização dos agentes que residem (em operação) em uma determinada máquina é feita selecionando tal máquina por meio de um click de mouse. Os agentes criados com base nesta arquitetura podem ser cadastrados e mantidos em um depósito de agentes. Estes últimos podem, posteriormente, ser instalados em qualquer máquina da rede que possua um agente *GareSleep*.

A Figura A.2 mostra, por meio de uma janela, as competências de um determinado agente que está em execução. É possível utilizando-se desta janela enviar *queries* ao agente, as quais são imediatamente executadas pelo agente.

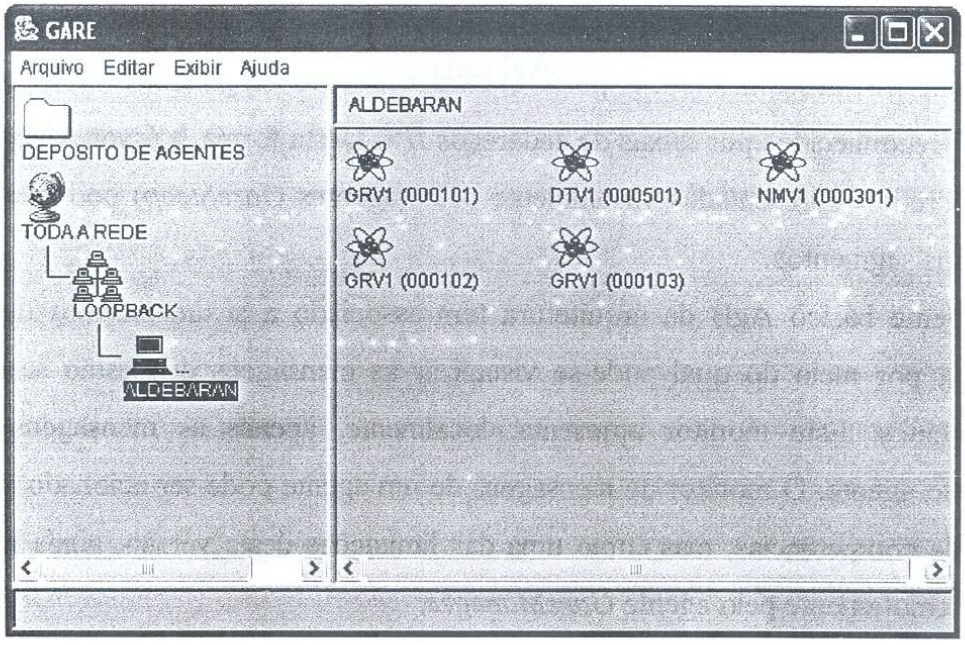

Figura A.1: Interface principal do agente *GareManager*.

Para instalar um novo agente na rede, acessa-se o depósito de agentes, seleciona-se o agente desejado e arrasta-se o ícone do agente sobre a máquina em que se deseja fazer a instalação; as opções copiar e colar do menu editar produzem o mesmo efeito. Esta é a forma padrão, em *GARE*, para proceder à instalação e distribuição de agentes em uma rede de computadores.

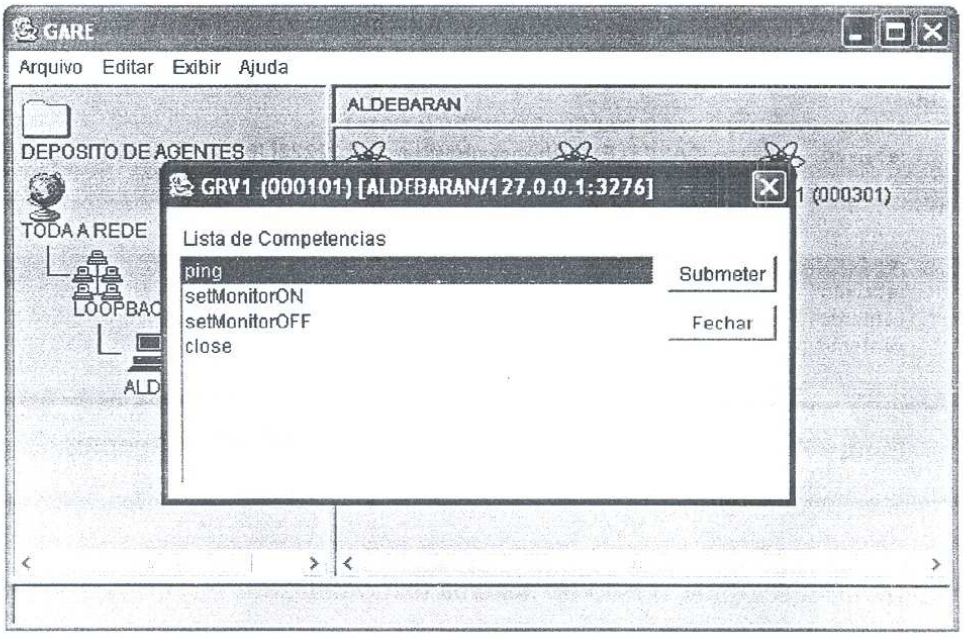

Figura A.2: Lista de Competências de um agente. Por esta interface é possível enviar uma query ao agente para execução imediata.

Quando não há segmentos de rede cadastrados, todos os agentes aparecem sob um segmento genérico, desconhecido. É possível cadastrar novos segmentos (arbitrariamente) de rede que são reconhecidos por faixas de endereços *IPs*. Desta forma, a árvore que apresenta as máquinas da rede (representadas virtualmente pelos agentes *GareSleep*) pode ser visualizada em seus vários segmentos.

O agente básico AgB da arquitetura tem associado a si um monitor de mensagens (Figura A.3), por meio do qual pode-se visualizar as mensagens que estão sendo trocadas entre os agentes. Este monitor apresenta, localmente, apenas as mensagens enviadas e recebidas pelo agente. O monitor de mensagens de um agente pode ser acionado remotamente pela janela de competências, mas como uma das limitações desta versão, ainda não pode ser consultado remotamente pelo agente *GareManeger*.

O agente *GareManager* também oferece outros serviços aos demais agentes por meio de troca de mensagens. Ele funciona como um servidor de nomes já que é ele que atribui um nome único pelo qual determinado agente será reconhecido na rede. Não se trata propriamente de um facilitador ou mediador, pois as mensagens não são enviadas por meio dele, mas pode-se visualizá-lo como tal. Fornece também os serviços de busca de agentes por nome (serviço de *WhitePages*) e por competências (serviço de *YellowPages*), mobilidade de agentes, solicitação de instalação remota, dentre outros.

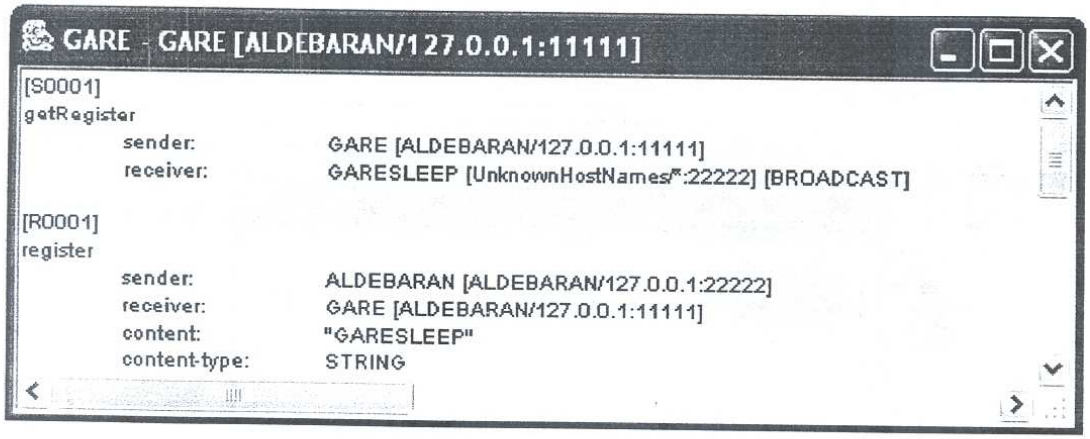

Figura A.3: Monitor de Mensagens do Gerenciador de Agentes.

Em resumo esta segunda versão do sistema disponibiliza:

- Uma API para a criação dos agentes independentes dos agentes criados para a arquitetura;

- Um conjunto de serviços administrativos/operacionais. Dentre os mais importantes citamos: servidor de nomes, *WhitePages/YellowPages*, gerência de mobilidade e instalação.

Deve-se notar que a arquitetura é aberta, e neste sentido, é perfeitamente possível acrescentar outros agentes que venham a integrar e tornar o ambiente mais robusto. Ou seja, pode-se incluir alternativas de gerenciadores e *daemons* sem grandes esforços.

O agente *GareSleep* (Figura A.4) comporta-se como um *daemon*, i.e., ele espera, em *background*, por requisições de serviços de outros agentes. De fato, todos os agentes nesta arquitetura funcionam em parte como um *daemon*. Toda à parte de recepção de mensagens e delegação para tratamento é efetuada por uma *thread* principal e um conjunto de *threads* auxiliares.

Figura A.4: Agente *GareSleep*.

É o agente *GareSleep* que responde pelos nós de rede que aparecem na janela do agente *GareManager*. O agente *GareSleep* pode ser visto, apesar de sua pouca sofisticação, como um facilitador ou mediador. É importante notar que cada máquina possui um agente *GareSleep* e cada agente *GareSleep* pode associar-se a vários agentes da aplicações. Esta hierarquia nos permite comparar a arquitetura *GARE* a um sistema formado por um conjunto de mediadores distribuídos[8]. Tal forma de organização traz a vantagem de comportar um número elevado de agentes em uma rede de computadores, aproximando-se da organização da Internet. O agente *GareSleep* é também responsável pela recepção e instalação de agentes,

[8] O agente GareSleep guarda uma lista de agentes em execução e compartilha tal lista com qualquer solicitante. Deve-se notar que para que se cumprisse integralmente o papel de um mediador, seria necessário implementar uma camada de delegação e controle das competências individuais.

considerando que a mobilidade não é implementada como elemento nativo no agente básico. Ou seja, os agentes são capazes de gerir a sua mobilidade, mas não de se moverem.

Este agente não permite grandes interações com o usuário por meio de uma interface gráfica, considerando que o mesmo desempenha um papel mais operacional; a sua forma de comunicação é essencialmente via troca de mensagens.

A.2. Comunicação e Transporte

A arquitetura divide-se em camadas como pode ser visualizado na Figura A.5.

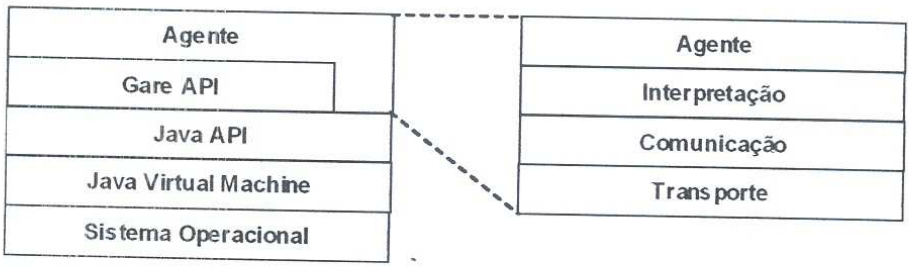

Figura A.5: Arquitetura *GARE*.

A camada de transporte da arquitetura é independente da camada de comunicação. Novas implementações para esta camada podem ser criadas (TCP, UDP, RMI, CORBA, JMS, JINI, etc.) e podem ser instaladas facilmente na camada de comunicação. A arquitetura disponibiliza como base uma implementação para o transporte em UDP. Figura A.6.

A localização e identificação dos agentes foi proposta da forma mais genérica possível e já largamente utilizada para que pudesse servir às mais diversas implementações possíveis para esta camada. Utilizou-se o formato *protocol://host:port/agentname* que na linguagem de programação JAVA é implementado pela classe *java.net.URI* (*Uniform Resource Identifier*).

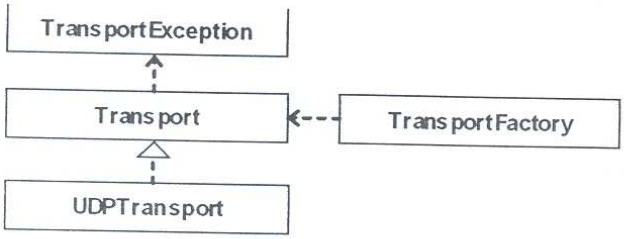

Figura A.6: Camada de transporte.

A camada de comunicação recebe os dados (*bytes*) da camada de transporte e os converte para uma mensagem de alto nível. Estas mensagens são encaminhadas para as camadas de interpretação correspondentes onde serão tratadas.

A linguagem de comunicação entre os agentes é baseada na estrutura da linguagem KQML [LAB97] e as formas de recepção e tratamento das mensagens, totalmente baseada na estrutura proposta pelo JKQML (camada de interpretação e ontologia) [IBM98].

De forma simplificada, tal linguagem define uma performativa, que no sistema é tratado como um evento. Cada mensagem tem um conjunto de campos:

- Performativa: ato da fala. Possui um significado especifico e geralmente é imperativo;
- Emissor: aquele que envia a mensagem;
- Receptor: aquele que recebe a mensagem;
- Rótulo de envio: identificação da mensagem;
- Rótulo em resposta: relaciona-se com um rótulo de envio. Indica que é uma resposta específica para uma mensagem com esta identificação;
- Linguagem: tipo de linguagem utilizada para codificar o conteúdo da mensagem;
- Ontologia: conjunto específico de termos utilizados no conteúdo da linguagem e da mensagem. Seleciona um contexto;
- Conteúdo: conteúdo da mensagem.

Um campo protocolo foi acrescentado como campo extra para a mensagem. Ele informa o protocolo em que a mensagem está codificada.

Na arquitetura não há uma implementação formal para KQML. O que há é uma implementação de uma LPL (*Lisp Property List*) onde são acrescentados os campos necessários para a mensagem.

Por exemplo, uma mensagem para registro:

```
LPL reply = new LPL( );
reply.put( KQML.PERFORMATIVE , KQMLPerf.REGISTER );
reply.put( KQML.SENDER, "udp://broadcast:8027/GareManager");
...
... send( reply );
```

As mensagens implementadas nesta arquitetura possuem o seguinte formato:

(:performative register :sender upd://localhost:2186/Segmentação :receiver udp:/broadcast:8027/GareManager :protocol KQML :language N&F :ontology WhitePages)

As classes com atributos estáticos *KQML* e *KQMLPerf* definem, respectivamente, os campos e as performativas de um mensagem. Outras classes deste tipo podem ser criadas para representar os campos e as performativas de uma outra linguagem – de comunicação ou negociação – *Contract-Net, B-Contract*, etc.

A mensagem é convertida em *String* e depois em *bytes* e só então delegadas à camada de transporte. Isto permite que agentes criados em outras linguagens de programação enviem mensagem na forma de *String* convertidas em *bytes* e também às recebam.

As mensagens podem ser síncronas ou assíncronas. Mensagens assíncronas são tratadas pelas camadas de interpretação. Já as mensagens síncronas são controladas por um objeto da classe *Conversation*. Devem ter rótulo de envio (*reply-with*) e apenas um destinatário. Ao enviar a mensagem, recebe-se um objeto *Conversation* que será atualizado quando a resposta à mensagem chegar. É possível controlar o tempo de espera para a resposta. A resposta também pode ser postada por uma das camadas de interpretação do agente para onde a mensagem de resposta foi encaminhada (Figura A.7).

As mensagens podem ser expedidas em *broadcast* ou a múltiplos destinatários adicionados como *receiver* da mensagem (uma forma de *multicast*).

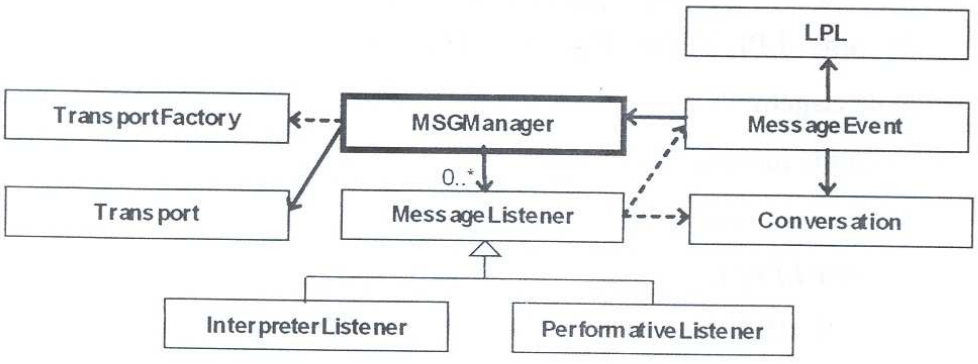

Figura A.7: Camada de Comunicação/Interpretação.

A.3. Interpretação e Ontologia

A camada de interpretação recebe as mensagens e às trata. Algumas são pré-definidas pela arquitetura *GARE* (*WhitePages, YellowPages* e *Subscribe*) e outras podem ser definidas pelo desenvolvedor. Caso não possam tratá-las isoladamente, repassam para a camada de tratamento genérica do agente.

Existem três formas de redirecionar as mensagens para as implementações de tratamento ou módulos específicos dos agentes, no contexto deste trabalho:

- Todas as mensagens (*MessageListener*). A aplicação define uma ou mais classes que implementam a interface *MessageListener* cujas instâncias são adicionadas à lista de *listeners* na camada de comunicação (*MSGManager*). Quando uma mensagem chega, ela é encaminhada para as instâncias cadastradas para ser tratada. A interpretação das mensagens deve ser implementada nestas classes e as interações com outras camadas do agente devem ser lavadas em consideração.
- Apenas mensagens com performativas específicas são redirecionadas (*PerformativeListener*). A aplicação define uma ou mais classes que implementam a interface *PerformativeListener* cujas instâncias são adicionadas à lista de *listeners* na camada de comunicação. Quando uma mensagem chega verifica-se se algum dos *listeners* espera por uma performativa igual a da mensagem. Caso afirmativo, a mensagem é redirecionada para esta instância.
- Apenas mensagens com protocolo, linguagem e ontologia especificadas são redirecionadas (*InterpreterListener*). A aplicação define uma ou mais classes que implementam a interface *InterpreterListener* cujas instâncias são adicionadas à lista de *listeners* na camada de comunicação. Quando uma mensagem chega verifica-se se algum dos *listeners* espera por uma mensagem com protocolo, linguagem e ontologia informada.

Não é aconselhável que o agente receba todas as mensagens e tenha que interpretá-las e tratá-las diretamente em seu código. Se for assim, o que se verá é um aumento espantoso na complexidade e tamanho do código, principalmente quando o número de performativas diferentes é grande.

A implementação de camadas de interpretação cria pontos específicos para o tratamento de mensagens, separando a implementação dos módulos dos agentes.

Estes módulos podem conversar entre si através de associações unidirecionais ou bidirecionais, MVC (*Model/View/Controller*), modelos de dados. As mensagens também podem ser redirecionadas para outros módulos do agente utilizando-se as mesmas classes da camada de interpretação.

Acreditamos que a forma mais importante de redirecionamento de mensagens é a terceira forma supracitada. Ela permite selecionar uma camada de interpretação pelos campos protocolo e/ou linguagem e/ou ontologia (pelo menos uma delas deve ser informada) da mensagem. Neste contexto, a ontologia se mostra muito importante. Com ela, pode-se selecionar um contexto, um assunto específico que se quer conversar. Permite também reutilizar performativas da linguagem aumentando o poder de expressão da mesma.

A arquitetura *GARE* define duas ontologias básicas que podem ser utilizadas por qualquer agente. *WhitePages*, *YellowPages* e *Subscribe*. As funcionalidades do serviço de *YellowPages* foram implementadas juntamente com os de *WhitePages*.

WhitePages (protocolo: KQML, linguagem: N&F – Networking & Facilitation, ontologia: WhitePages) é utilizado para tratar mensagens de registro, busca e remoção de registro de agentes. Ela depende de um modelo de dados (*WhitePagesModel*), onde serão armazenados os dados dos agentes e que fornecerá os métodos para a inclusão, procura e remoção dos mesmos. Além disso, este modelo de dados implementa mecanismos de *subscribe/notify* para a implementação do *design pattern* MVC para a notificação de interfaces gráficas e componentes quando o modelo de dados é alterado (Figura A.8).

As performativas aceitas pela camada *WhitePages/YellowPages* são:

- ***register***. Solicita registro do agente emissor da mensagem ao serviço de *WhitePages*. O nome do agente é retirado do campo *sender*. Esta mensagem é do tipo Sincronizada;

  (:performative register :sender protocol://host:port/AgentName :receiver ... :reply-with X :protocol KQML :language N&F :ontology WhitePages)

- ***reply***. Resposta a um *register*. Retorna o número de registro do agente e um nome único para ser utilizado na rede. Este nome único retorna com o campo *receiver* atualizado. O tipo da mensagem é *One-Way*;

  (:performative reply :sender ... :receiver protocol://host:port/AgentName-9 :in-reply-to X :protocol KQML :language N&F :ontology WhitePages :content (:reg-num 9))

- **unregister**. Solicita a remoção do registro do agente ao serviço de *WhitePages*. O tipo da mensagem é *One-Way*;
  (:performative unregister :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:reg-num 9))
- **transport-address**. Solicita a atualiza do endereço do agente ao serviço de *WhitePages*. Esta mensagem é enviada após a movimentação do agente para informar o seu novo endereço. O tipo da mensagem é *One-Way*;
  (:performative transport-address :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:uri protocol://new-host:new-port/AgentName-9))
- **ask-one**. Solicita informação de localização do agente informado no conteúdo da mensagem ao serviço de *WhitePages*. O tipo da mensagem é Sincronizada;
  (:performative ask-one :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:reg-name AgentName-9))
- **ask-all**. Solicita informação de localização de todos os agentes com o mesmo nome (*realname*), por exemplo, ASSINATURA, informado no conteúdo da mensagem ao serviço de *YellowPages*. O tipo da mensagem é Sincronizada;
  (:performative ask-one :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:realname AgentName))
- **tell**. Resposta a um *ask-one* ou *ask-all*. O tipo da mensagem é *One-Way*. Respectivamente:
  (:performative tell :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:uri protocol://host:port/AgentName-9))

  (:performative tell :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:addresses protocol://host:port/AgentName-9 protocol://host:port/AgentName-9 protocol://host:port/AgentName-9 ...))
- **sorry**. Resposta a um *ask-one* ou *ask-all*. O tipo da mensagem é *One-Way*;
  (:performative sorry :sender ... :receiver ... :protocol KQML :language N&F :ontology WhitePages :content (:msg-error message))

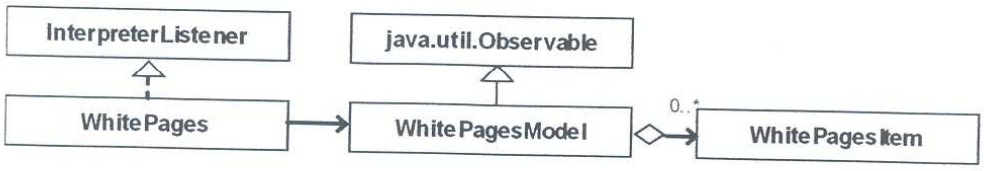

Figura A.8: Interação com uma *WhitePages*.

Subscribe (protocolo: KQML, linguagem: Subscribe/Notify, ontologia: Subscribe) é utilizado para armazenar pedidos de notificação de registro de agentes, remover pedidos e notificar o registro de novos agentes. Ela depende de um modelo de dados (*SubscribeModel*) onde serão armazenados os dados dos pedidos de notificação (Figura A.9). Sempre que um novo agente se registra, este modelo é notificado e então verifica se alguém na rede deseja ser notificado da presença do novo agente e notifica-o. Quando um agente é notificado, a mensagem e redirecionada para um instância da classe que implementa *SubscribeListener*.

As performativas aceitas são:

- ***subscribe***. Solicita receber informações sobre um determinado agente ao serviço de *Subscribe/Notify*. O nome do agente é passado no conteúdo da mensagem. O tipo da mensagem é *One-Way*;
  (:performative subscribe :sender ... :receiver ... :protocol KQML :language Subscribre/Notify :ontology Subscribe :content (:realname AgentName))
- ***discard***. Remove a solicitação de receber informações sobre um determinado agente ao serviço de *Subscribe/Notify*. O nome do agente é passado no conteúdo da mensagem. O tipo da mensagem é *One-Way*;
  (:performative discard :sender ... :receiver ... :protocol KQML :language Subscribre/Notify :ontology Subscribe :content (:realname AgentName))
- ***tell***. O serviço de *Subscribe/Notify* notifica que um novo agente registrou-se no serviço de *WhitePages*. *One-Way*;
  (:performative tell :sender ... :receiver ... :protocol KQML :language Subscribre/Notify :ontology Subscribe :content (:realname AgentName :uri protocol://host:port/AgentName-9))

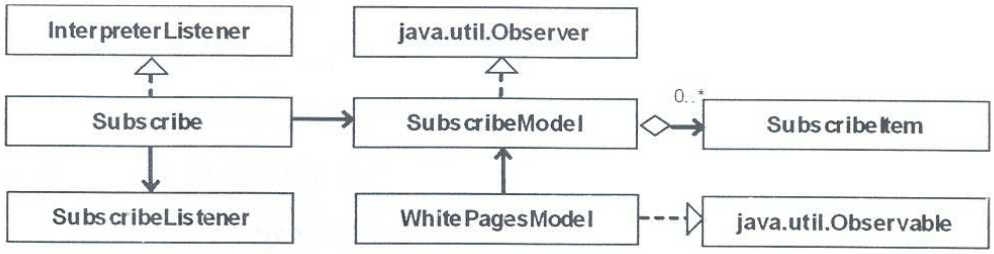

Figura A.9: Interação com um *Subscribe*.

GareManager e *GareSleep* possuem serviços de *WhitePages*. O agente *GareManager* possui também serviço de *Subscribe*. A Figura A.10 mostra o fluxo das mensagens para *WhitePages* e *Subscribe*.

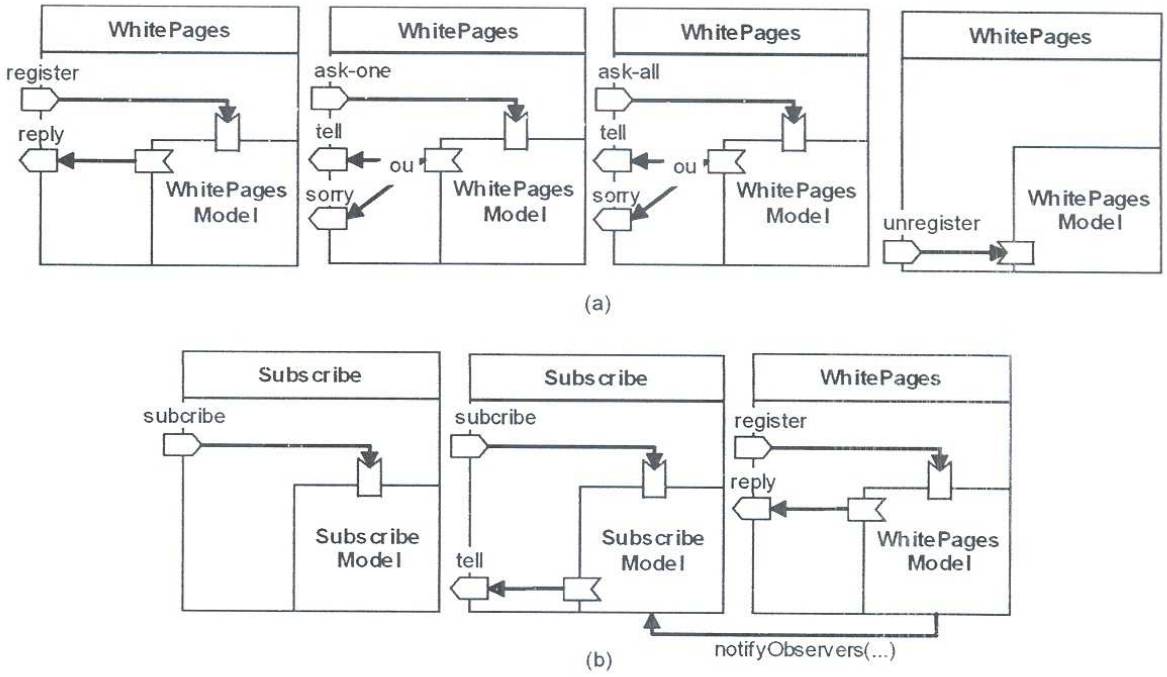

Figura A.10: (a) Mensagens *WhitePages*. (b) Mensagens *Subscribe*.

A.4. Mobilidade e Instalação

A mobilidade dos agentes pela rede não é implementada de forma nativa nos agentes da plataforma *GARE*. Ela depende dos agentes *GareManager* e *GareSleep*, assim como do processo de instalação remota. Nesta versão, as mensagens de movimentação ainda não foram reunidas em uma camada de interpretação e também ainda não foram normalizadas de acordo com as performativas originais do KQML.

Um agente só poderá ser instalado nas máquinas que possuem um agente *GareSleep* ativo. Em outras palavras, um agente **X** só poderá ser instalado nas máquinas que aparecem na interface gráfica do agente *GareManager* (Figura A.1). Assim, quando o ícone de um agente **X** é arrastado sobre o ícone da máquina **H** destino, uma mensagem ***send-file*** é enviada ao agente *GareSleep* situado na máquina **H**. Ambos *GareManager* e *GareSleep* iniciam um processo de transferência de arquivos por meio do serviço de FTP. O arquivo transportado contem o código do agente **X**, que sai da origem — agente *GareManager* — para o destino — agente *GareSleep* da máquina **H**. Encerrado a transferência, o agente *GareSleep* em questão encarrega-se de lançar o agente **X**. Deste ponto em diante, o agente **X** tem vida própria.

O atraso no processo de instalação ocorre apenas na primeira vez em que o código do agente **X** é transportado para máquina hospedeira **H**. Se houver uma nova solicitação para instalar na máquina **H** um novo agente do mesmo tipo do agente **X**, não será necessário fazer a transferência de código. O agente *GareSleep* da máquina **H** lançará um novo agente a partir do código recebido anteriormente. O novo agente segue os procedimentos de inicialiazação já mencionados.

Salientamos que as mensagens de instalação e mobilidade pertencem aos agentes *GareManager* e *GareSleep*. Ou seja, elas não estão presentes no agente básico AgB.

Destacamos também que o ambiente suporta a solicitação de instalação remota e não assistida. O gerenciador decide em qual máquina um agente **X** será instalado. Ele toma tal decisão em função do número de agentes que cada máquina possui. A máquina que possui o menor número de agentes recebe o agente **X**. Este mecanismo é utilizado para mover agentes de um local para outro visando o balanceamento de carga. Evidentemente, tal procedimento torna-se interessante à medida que existe um número considerado de máquinas, ou seja, o número de máquinas é superior o número de agentes.

Na mobilidade não assistida, um agente pode ser movido por meio da interface do gerenciador (Figura A.1). Aqui, o usuário seleciona um agente ativo **X** de uma determinada máquina origem **H1** e o arrasta para uma máquina destino **H2**. O agente **X** recebe a mensagem ***move-out***. Ele responderá com um ***accept-move*** ou ***cancel-move***. A implementação destas mensagens é feita pelo desenvolvedor do agente. Caso o agente **X** não as suporte, um ***timeout*** ocorrerá para a mensagem ***move-out*** e o usuário será notificado do fato. A mensagem ***cancel-move*** informa ao usuário que o agente não pode ser movido.

A mobilidade na plataforma *GARE* é dita fraca. Quando um agente **X** aceita ser movido, ele envia juntamente com a mensagem ***accept-move*** os dados que representam o seu

estado interno **E1**. Do ponto de vista técnico, o que ocorre é o seguinte: um agente do tipo **X** é lançado sobre a máquina destino e quando ele se registra, ele recebe os dados referentes ao estado **E1**. Este estado incorpora o registro do agente. Isto é feito por meio da mensagem ***remember***.

Esta versão da plataforma ainda não suporta a persistência dos agentes. Destacamos, no entanto, que se pensássemos apenas no uso da linguagem de programação Java, a persistência seria algo resolvido. Java facilita, na forma nativa, a persistência por meio de mecanismos de serialização de objetos. Estes mecanismos, proprietário, reduzem sobremaneira os esforços para viabilizar a mobilidade e facilitar a instalação de agentes em uma rede de máquinas, assim como o ciclo de vida destes agentes. Não foi utilizado a serialização do Java visando à possibilidade de termos um mesmo agente implementado em linguagens diferentes. Cada implementação mantendo a especificidade da linguagem utilizada, porém permitindo a transferência de estado. Por exemplo, a implementação em Prolog do agente **X** transferindo seu estado a uma outra implementação de **X**, porém codificada em Lisp, Java, etc.

Na mobilidade forte, os dados do contexto de execução e CPU são salvos na origem e restaurados no destino. Transferir agentes serializados entre máquinas e iniciá-los não se trata de um exemplo de mobilidade forte. Ainda tem-se a mobilidade fraca já que os dados do contexto de execução e CPU não são salvos. Neste caso, faz-se necessário um mecanismo de reinicialização do agente.

A.5. Considerações Finais

Uma das grandes dificuldades da área de inteligência artificial distribuída é dispor de um ambiente de desenvolvimento que facilite tanto a criação como a manutenção/gestão dos agentes de uma aplicação. As particularidades deste ambiente são os mecanismos que nos permitem testar a mobilidade. Destacamos também o desejo de dispor de uma plataforma que facilite a troca de estado entre agentes idênticos, porém escritos em linguagens diferentes.