ELAINI SIMONI ANGELOTTI

UTILIZAÇÃO DA LÓGICA PARACONSISTENTE NA IMPLEMENTAÇÃO DE UM SISTEMA MULTI-AGENTE

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

2001

Angelotti, Elaini Simoni
Utilização da Lógica Paraconsistente na Implementação de um Sistema Multi-Agente. Curitiba, 2001. 75p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1.Inteligência Artificial 2.Agente Autônomo 3.Lógica Paraconsistente 4.Distribuição de Tarefa. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada.

Pontifícia Universidade Católica do Paraná

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática Aplicada

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 036

Aos 12 dias do mês de setembro de 2001 realizou-se a sessão pública de defesa de dissertação "**Utilização da Lógica Paraconsistente na Implementação de um Sistema Multi-Agentes**", apresentada por **Elaine Simoni Angelotti** como requisito parcial para a obtenção do título de **Mestre em Ciências**, perante uma Banca Examinadora composta pelos seguintes membros:

	assinatura	parecer
Prof. Dr. Edson Scalabrin PUCPR (Presidente)		aprovado
Prof. Dr. Bráulio Ávila PUCPR		aprovado
Prof. Dr. Marco Shmeil PUCPR		Aprovado
Profa. Dra. Cinthia Freitas PUCPR		aprovado
Prof. Dr. Hilton de Azevedo CEFET-PR		Aprovado

Conforme as normas regimentais do PPGIA e da PUCPR, o trabalho apresentado foi considerado aprovado sem restrições (aprovado sem restrições, aprovado com exigências ou reprovado), segundo a avaliação da maioria dos membros da Banca Examinadora acima indicada.

Observações, exigências e/ou restrições da Banca Examinadora, quando houver:

continuar no verso, se necessário

Prof. Dr. Carlos Alberto Maziero, Diretor
Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná

PPGIA PUCPR

Aos meus pais e irmãs,
pelo apoio, incentivo e paciência
nos momentos que não passamos juntos.

Ao Gustavo, pela sua existência,
pelo seu amor e incentivo.

Agradecimentos

Ao Prof. Edson Emílio Scalabrin por sua dedicação, amizade e apoio constante no desenvolvimento dessa dissertação.

Ao Prof. Bráulio Coelho Ávila pelos comentários realizados durante o desenvolvimento deste trabalho e pela amizade demonstrada.

Ao Prof. Marcos A. H. Shmeil pelas sugestões e comentários realizados durante o desenvolvimento deste trabalho.

À Prof. Cinthia O. de Almendra Freitas por ter aceitado fazer parte da banca e pelos comentários e sugestões realizados sobre este trabalho.

Ao Prof. Hilton J. de Azevedo por ter aceitado fazer parte da banca e pelas sugestões e discussões realizadas sobre esta dissertação.

Ao Prof. Alceu de Souza Britto Jr. pelos comentários e sugestões feitas sobre este trabalho.

Aos amigos Katiana, Regiane, Francislene, Josiane, Júnior, Adilson, Luciene, Simone, Marcio, Alessandro, Elcio, Fabrício e todos que sempre estiveram ao meu lado, que souberam entender a minha ausência, que sempre me apoiaram e proporcionando-me muitos momentos de alegria.

Ao PPGIA pela oportunidade concedida, assim como pela disponibilidade do uso do laboratório para o desenvolvimento e teste do proposto trabalho.

À CAPES pelo apoio financeiro.

Aos colegas de laboratório que contribuíram nesta pesquisa, especialmente ao amigo Fabrício pelo seu constante apoio e sugestões durante o desenvolvimento dessa dissertação.

Finalmente, agradeço a todos que, direta ou indiretamente, prestaram alguma contribuição e apoio ao desenvolvimento dessa dissertação.

Sumário

Lista de Figuras

Resumo

Este trabalho faz parte do Projeto *Multicheck*[1], que define uma arquitetura de agentes autônomos para o tratamento automático de cheques bancários brasileiros manuscritos. A competência desses agentes é implementada em duas camadas. *A primeira*, corresponde aos algoritmos de reconhecimento de padrões aplicados diretamente sobre os segmentos de imagens. *A segunda*, corresponde aos mecanismos de raciocínio aplicados sobre as informações provenientes da primeira camada para validá-las ou interpretá-las. Esta interpretação envolve também informações provenientes de outros agentes. Portanto, estas informações podem apresentar inconsistências. Este problema é tratado adequadamente e naturalmente através de conceitos e operadores da lógica evidencial paraconsistente. Este trabalho é consagrado à segunda camada supracitada e sobre os problemas de distribuição de tarefas e comunicação entre agentes. As informações referentes à primeira camada foram obtidas por meio de uma base de dados simulada.

Palavras-Chave: Inteligência Artificial, Agentes Autônomos, Lógica Paraconsistente, Distribuição de Tarefas.

[1] O projeto Multicheck está sendo realizado na Pontifícia Universidade Católica do Paraná, Brasil (PUCPR), com o apoio financeiro do governo brasileiro (CNPq), no quadro de uma colaboração internacional entre l'École de Technologie Supérieure (ETS)/Canadá e a PUCPR

Resumo

Este trabalho faz parte do Projeto *Multicheck*[1], que define uma arquitetura de agentes autônomos para o tratamento automático de cheques bancários brasileiros manuscritos. A competência desses agentes é implementada em duas camadas. *A primeira*, corresponde aos algoritmos de reconhecimento de padrões aplicados diretamente sobre os segmentos de imagens. *A segunda*, corresponde aos mecanismos de raciocínio aplicados sobre as informações provenientes da primeira camada para validá-las ou interpretá-las. Esta interpretação envolve também informações provenientes de outros agentes. Portanto, estas informações podem apresentar inconsistências. Este problema é tratado adequadamente e naturalmente através de conceitos e operadores da lógica evidencial paraconsistente. Este trabalho é consagrado à segunda camada supracitada e sobre os problemas de distribuição de tarefas e comunicação entre agentes. As informações referentes à primeira camada foram obtidas por meio de uma base de dados simulada.

Palavras-Chave: Inteligência Artificial, Agentes Autônomos, Lógica Paraconsistente, Distribuição de Tarefas.

[1] O projeto Multicheck está sendo realizado na Pontifícia Universidade Católica do Paraná, Brasil (PUCPR), com o apoio financeiro do governo brasileiro (CNPq), no quadro de uma colaboração internacional entre l'École de Technologie Supérieure (ETS)/Canadá e a PUCPR

Abstract

This work is part of the *Multicheck* Project[2] that defines architecture of autonomous agents for the automatic treatment of handwritten Brazilian bank checks. The competence of these agents is implemented in two layers. The first one corresponds to algorithms of patterns recognition which are applied directly on the image segment. The second one corresponds to reasoning mechanisms applied to the information from the first layer either to validate or to interpret it. The interpretation process involves as well information obtained from other agents. Therefore, information can present inconsistencies. This problem is treated properly and naturally through concepts and operators of paraconsistent logic. This work focuses on the second layer, on task distribution problems and on communication between agents. The first layer information was obtained through a simulated database.

Keywords: Artificial Intelligence, Autonomous Agents, Paraconsistent Logic, Task Distribution.

2 The Multicheck Project is being developed by Pontifical Catholic University of Paraná, Brazil (PUCPR), with the financial support of the Brazilian Government (CNPq), in an international cooperation between l'École de Technologie Supérieure (ETS)/ Canada and PUCPR

Abstract

This work is part of the *Multicheck* Project[2] that defines architecture of autonomous agents for the automatic treatment of handwritten Brazilian bank checks. The competence of these agents is implemented in two layers. The first one corresponds to algorithms of patterns recognition which are applied directly on the image segment. The second one corresponds to reasoning mechanisms applied to the information from the first layer either to validate or to interpret it. The interpretation process involves as well information obtained from other agents. Therefore, information can present inconsistencies. This problem is treated properly and naturally through concepts and operators of paraconsistent logic. This work focuses on the second layer, on task distribution problems and on communication between agents. The first layer information was obtained through a simulated database.

Keywords: Artificial Intelligence, Autonomous Agents, Paraconsistent Logic, Task Distribution.

2 The Multicheck Project is being developed by Pontifical Catholic University of Paraná, Brazil (PUCPR), with the financial support of the Brazilian Government (CNPq), in an international cooperation between l'École de Technologie Supérieure (ETS)/ Canada and PUCPR

I - Introdução

Este trabalho insere-se no escopo do *Projeto Multicheck*, o qual define uma arquitetura de agentes cognitivos independentes para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. Deve-se destacar que a escolha do uso de agentes no contexto deste projeto apóia-se em uma tripla hipótese: *primeira*, trata-se de um problema que se subdivide naturalmente, à medida que cada tarefa (e.g., verificação da assinatura, o reconhecimento da data) pode ser modelada como sendo um agente autônomo independente e fracamente acoplado; *segunda*, a interação entre estes agentes pode tornar o processo de tratamento mais robusto, à medida que as trocas entre os agentes podem resolver situações aparentemente difíceis *ou mesmo* impossíveis de serem solucionadas por um único especialista (e.g., a validação do montante literal e do montante numérico); *terceira,* o paralelismo natural entre os agentes pode contribuir de forma significativa para implementar uma aplicação de alto desempenho/performance.

Cada agente da arquitetura supracitada implementa uma competência especifica que corresponde a uma das seguintes tarefas:

1. localizar/extrair a estrutura lógica de um cheque; i.e., obter os campos de identificação do banco, da agência, do emissor, dos montantes numérico e literal, data, assinatura;
2. interpretar/validar o montante literal;
3. interpretar/validar o montante numérico;
4. reconhecer a data;
5. verificar a assinatura.

As competências ligadas *a interpretação de um campo lógico* (montante literal e numérico, data e assinatura) são implementadas em duas camadas:

- a primeira, corresponde aos algoritmos de reconhecimento de padrões aplicados diretamente sobre segmentos de imagens de um cheque;

- a segunda, corresponde aos mecanismos de raciocínio aplicados sobre as informações provenientes da primeira camada para validá-las ou interpretá-las. Estes mecanismos são implementados apoiando-se na lógica paraconsistente.

Neste contexto, o presente trabalho será consagrado à segunda camada e tem por objetivo definir e implementar um protótipo para a arquitetura *Multicheck*. Deve-se, porém, notar que as informações referentes à primeira camada serão obtidas por meio de uma base de dados simulada.

De forma resumida, a proposta apoia-se, em particular, na *lógica paraconsistente* e no conceito de agente cognitivo desenvolvido na 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).

1.1. Justificativa e Motivação

Em um ambiente bancário, a verificação de cheques realizada manualmente por funcionários, apesar de ser uma tarefa trivial, pode gerar alguns problemas como: imperícia técnica, idoneidade do responsável, demora na realização da tarefa, etc. A automação de tal processo é uma forma rápida e confiável de realizar esta tarefa, proporcionando o decréscimo dos custos e redução do tempo de compensação. No entanto, o tratamento automático de cheques manuscritos é um problema bastante complexo. Como descrito em Scalabrin *et al.* (1998), isto ocorre devido a diversidade e dificuldade de manutenção dos conhecimentos envolvidos, o custo dos cálculos, a necessidade de reconfigurar dinamicamente um processo de tratamento e a interação entre os especialistas.

Este processo de automação envolve um grande número de operações a serem implementadas:

- aquisição da imagem;
- eliminação de informações irrelevantes presentes no cheque, como por exemplo fundo artístico, sobreposição de linhas, etc.;
- localização e extração de informações relevantes, como por exemplo identificação do banco, da agência, do emissor, dos montantes numérico e literal, data, assinatura;

- obtenção da estrutura lógica do documento;
- separação do pré-impresso em relação ao manuscrito;
- segmentação de cada um dos campos lógicos;
- interpretação dos dados lógicos (reconhecimento da data, dos montantes literal e numérico e a verificação da assinatura),
- análise para aceitação ou rejeição de um cheque.

Trata-se, claramente, de um problema cujas tarefas são bem definidas. A implementação de cada uma delas exige grandes recursos computacionais e o compartilhamento dos resultados parciais de algumas destas tarefas, pode ser decisivo na obtenção de uma interpretação correta das informações presentes em um cheque.

Diante destes fatos, decide-se informatizar o processo de compensação de cheques bancários, apoiando-se no conceito de agente cognitivo. Este último permite organizar os conhecimentos da aplicação e obter vários benefícios próprios à abordagem. Tal abordagem foi escolhida pelas seguintes motivações:

- a natureza do problema em questão permite uma decomposição em tarefas bem definidas e cada uma delas pode ser encapsulada em um agente autônomo e independente;
- a capacidade natural de interação dos agentes torna o processo de tratamento dos cheques mais robusto, em particular, à medida em que as trocas entre eles resolvem situações aparentemente difíceis;
- a possibilidade de introduzir mecanismos de raciocínio e aprendizagem nos agentes, permite dotá-los de um comportamento pró-ativo e adaptável; e
- a modularidade aportada pelos agentes permite lutar contra a complexidade do domínio, bem como desenvolver um sistema de maneira incremental, ou seja, um sistema aberto de agentes [Scalabrin 96a].

No entanto, em um sistema de *inteligência artificial distribuída*, os agentes podem facilmente obter informações inconsistentes, em particular, porque as soluções de determinados subproblemas podem ser produzidas por agentes diferentes, e em algum dado momento, um desses agentes pode ser levado a combinar estas diferentes soluções para produzir uma outra. Desta forma, alguns agentes devem ser sofisticados o bastante para decidir *como*, *quando* e *com quem* interagir, assim como comportar-se adequadamente na presença de *informações*

contraditórias. O mecanismo desenvolvido para implementar esta sofisticação baseia-se em alguns conceitos e operadores da *lógica paraconsistente*. A *lógica paraconsistente* integra de forma natural o tratamento de informações inconsistentes, as quais não podem ser tratadas por meio da lógica clássica [Avila *et al.* 97], [da Costa *et al.* 90], [da Costa 63], [Subrahmanian 87b].

Empiricamente, a verificação de um cheque, quando realizada por um ser humano, passa pela interpretação de modo interativo e aproximativo do montante numérico, do montante literal e, em seguida, pela verificação da data e da assinatura. Dessa forma e intuitivamente, a imagem do cheque bancário da Figura 1, requer conhecimentos específicos para verificar/tratar cada um dos campos lógicos relevantes e presentes neste documento, a saber: *data, assinatura, montantes numérico e literal.*

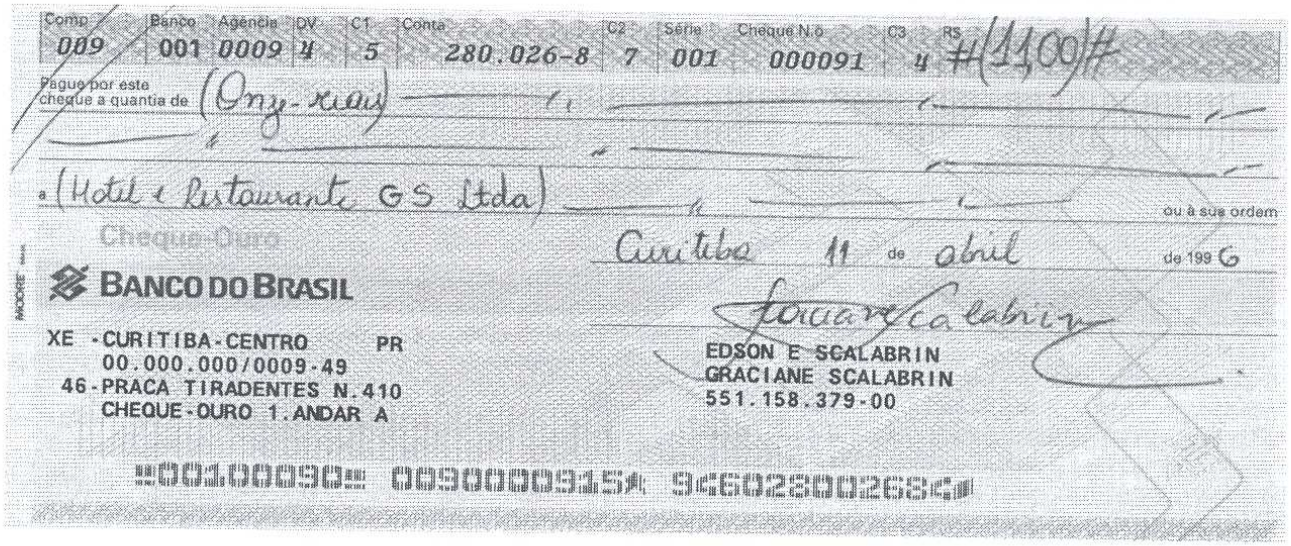

Figura 1. Imagem de um cheque bancário brasileiro.

A interatividade entre as tarefas de reconhecimento/interpretação pode ser implementada naturalmente através das capacidades dos agentes cognitivos, e a forma aproximativa de tratar as informações locais e não locais[3], trocadas entre os agentes, pode ser implementada por meio da lógica paraconsistente.

[3] Informações locais são informações referentes ao campo lógico que um agente está reconhecendo. As informações não locais são as informações que estão sendo reconhecidas pelos outros agentes do sistema.

1.2. Objetivos e Metodologia

Como citado anteriormente, o objetivo deste trabalho é definir e implementar um protótipo de uma arquitetura baseada em agentes cognitivos, que possibilitará o tratamento automático de cheques bancários brasileiros manuscritos. No entanto, para se alcançar este objetivo, será necessário o cumprimento dos seguintes objetivos específicos:

- criar uma base de dados para testar e validar o protótipo. Os dados desta base serão gerados artificialmente por métodos estatísticos;
- definir e implementar o protótipo, que se chamará doravante Pandora4;
- definir e implementar procedimentos para a gestão da inclusão e remoção de agentes do sistema de forma dinâmica, i.e., o sistema deve ser aberto;
- definir e implementar a arquitetura interna dos agentes que terão, ao mesmo tempo, as camadas de reconhecimento e interpretação/validação de padrões;
- definir e implementar procedimentos de alto nível para controlar a interação entre estes agentes (e.g., na comunicação de informação);
- verificar o uso da lógica paraconsistente no tratamento de informações inconsistentes oriundas de processamentos de outros agentes;

A realização destes objetivos será obtida através da construção de três versões do protótipo *Pandora*. A construção destas versões é parte integrante da metodologia adotada para alcançar os objetivos fixados da pesquisa. É importante salientar que cada nova fase ou versão inclui de forma integral todas as funcionalidades das fases anteriores. As versões são as seguintes:

- *v-1*, consiste de uma arquitetura multi-agente onde os agentes são "completamente" independentes, i.e., cada agente recebe uma entrada de dados, processa esses dados e gera uma resposta de forma independente e sem nenhum tipo de interação com os demais agentes;
- *v-2*, consiste de uma arquitetura multi-agente onde os agentes de reconhecimento interagem com o agente segmentação durante a análise dos cheques, por exemplo, para solicitar uma nova segmentação; e
- *v-3*, consiste na versão final do protótipo onde todos os agentes interagem, se necessário.

---

[4] O nome Pandora vem da mitologia grega e foi dado a primeira mulher criada por Júpiter.

1.3. Sumário dos Capítulos

Este documento está organizado em cinco capítulos.

O segundo capítulo consiste na fundamentação teórica para este trabalho e compreende duas seções:

1. apresenta uma visão geral dos sistemas multi-agente, assim como, alguns de seus componentes, arquiteturas, definições, etc; e
2. examina a lógica evidencial paraconsistente, sua sintaxe e semântica, além de alguns operadores que foram utilizados no desenvolvimento desse trabalho.

O terceiro capítulo consiste na descrição do funcionamento do sistema *Pandora* e compreende duas seções:

1. apresenta a arquitetura do Pandora, a arquitetura interna dos agentes, assim como a forma com que a lógica paraconsistente foi aplicada na implementação das competências de alguns dos agentes; e
2. descreve alguns cenários onde agentes trocam mensagens utilizando o protocolo de comunicação JKQML.

O quarto capítulo apresenta o tratamento de 1.600 cheques bancários. Deve-se salientar que todos estes cheques foram submetidos às três versões do sistema e os resultados obtidos para cada uma das versões são exibidos de forma gráfica e comparativa.

Finalmente, apresentamos as conclusões, relatando as vantagens do uso de um sistema multi-agente em particular no contexto dessa aplicação, assim como, os benefícios que a lógica evidencial paraconsistente proporcionou a este trabalho.

II. Sistema Multi-Agente e Lógica Paraconsistente

A construção de sistemas baseado em recursos computacionais, e envolvendo diversas fontes de conhecimentos e algoritmos de reconhecimento de padrões, constitui um problema complexo. A complexidade ocorre devido a dificuldade de manutenção dos conhecimentos, o custo dos processamentos, a necessidade de reconfigurar dinamicamente um processo de tratamento e a interação entre especialistas que executam tarefas interagentes (e.g. tratamento dos montantes numérico e literal). A abordagem multi-agente oferece um certo número de possibilidades para melhorar a estruturação, a modularidade e a evolução de tais sistemas. Porém, o desenvolvimento efetivo de um sistema multi-agente apresenta algumas dificuldades, em particular a ausência de um protocolo de interação que seja *simples* de compreender e implementar para o problema de distribuição de tarefas e o compartilhamento de resultados, assim como a integração de forma natural do tratamento de informações inconsistentes.

O objetivo do trabalho é mostrar que é possível conceber um sistema que permita tratar alguns dos problemas supracitados.

Esta parte é dividida em duas seções:

- *a seção 1*, relembra sumariamente o conceito de um sistema multi-agente e alguns de seus componentes.
- *a seção 2*, examina a lógica evidencial paraconsistente, em particular, os trabalhos de [da Costa 99], [da Costa 90], [Abe et al. 98], [Blair 88] e [Subrahmanian 87a e b].

2. O que é um sistema multi-agente?

Os sistemas multi-agente podem ser divididos basicamente em duas classes:

- sistemas reativos; e
- sistemas cognitivos.

Os sistemas de agentes cognitivos são geralmente constituídos de um pequeno número de agentes à *granularidade alta* — tipicamente menor que 50. Estes agentes são *inteligentes*, ou seja, possuem uma representação parcial e explícita de seu ambiente, capacidade local de decisão e podem negociar uma informação ou um serviço. Eles são em geral dotados de conhecimentos, competências, intenções e planos, o que possibilita coordenar suas ações na resolução de um problema [Scalabrin 96a].

Os sistemas de agentes reativos são constituídos por um grande número de agentes à *granularidade fina*, bastante simples, sem inteligência e sem representação de seu ambiente. Eles podem modelar, por exemplo, uma sociedade de formigas, ou ainda os clientes e os servidores na abordagem "OMG CORBA" [Scalabrin 96a]. Estes agentes são fortemente acoplados e interagem utilizando um comportamento do tipo estímulo/resposta [Ferber & Drogoul 92]. Neste caso, um comportamento inteligente emerge a partir das interações entre estes agentes e seu ambiente [Brooks 86], i.e., os agentes não são individualmente inteligentes, mas seu comportamento global o é.

No contexto desta dissertação, os agentes são cognitivos, i.e., cada agente é uma entidade que tem capacidade de raciocínio, capacidade de tratamento de informações relativas a seu domínio de aplicação e à gestão das interações com os outros agentes e o ambiente, assim como levam em conta a inconsistência das informações que podem ocorrer durante as trocas com outros agentes. Finalmente, o objetivo de cada agente será atingir seu objetivo, colaborando se necessário.

Deve-se salientar, que as arquiteturas de sistemas desenvolvidos em *inteligência artificial distribuída* inspiram-se originalmente na forma como um comitê de especialistas resolve um problema, i.e. a decomposição do problema é desconhecida *a priori* e os especialistas são capazes de atribuir-se subproblemas tomando como base suas próprias competências. Eles podem também desenvolver atividades cooperativas e coordenadas para resolver um problema. Este comportamento define um *sistema* onde os agentes trabalham independentemente e comunicam apenas quando necessitam de uma informação para alcançar um objetivo comum. Desta forma, um *sistema multi-agente* pode ser definido em função da autonomia de cada agente

e dos meios que os mesmos dispõem para gerar suas interações, por exemplo, na auto atribuição de subproblemas através de procedimentos de negociação e coordenação.

De modo geral, os agentes buscam simplesmente satisfazer seus objetivos individuais e a interação entre eles passa a existir apenas quando surgem conflitos, que podem decorrer da falta de um recurso local ou simplesmente da necessidade de articulação de ações individuais. A ausência de controle global e de dados globalmente acessíveis e coerentes é compensada por procedimentos locais de coordenação definidos sobre o modelo dos outros agentes, de seus objetivos, de suas intenções e sobre os procedimentos de cooperação. Finalmente, no que concerne a pesquisa neste domínio, ela é centrada basicamente sobre os agentes e sobre os meios que eles dispõem para comunicar [Scalabrin 96a].

Freqüentemente, distinguem-se dois tipos de organizações nos *sistemas cooperativos*: os sistemas *multi-especialista* (baseado na estrutura *quadro-negro*) e os sistemas *multi-agente*.

2.1. Multi-Especialista vs. Multi-Agente (Shen & Barthès 96, Scalabrin 96a)

Em um sistema multi-especialista, cada especialista (freqüentemente chamado de fonte de conhecimento) tem acesso a uma zona de dados comum — ou *quadro-negro* — onde as informações são inteiramente armazenadas e deixadas a disposição de todos. Em um sistema multi-agente, cada agente é uma entidade independente e possui sua própria representação da situação. No primeiro caso, um agente depende fortemente de uma memória compartilhada, já no segundo, ele pode migrar livremente sobre diferentes máquinas sabendo-se que o mesmo pode comunicar por troca de mensagens assíncronas. O exemplo a seguir ilustrará tais organizações.

Exemplo 01:

> Um conjunto de entidades trabalhando de maneira cooperativa visando a construção de um artefato ou reconhecimento de um cheque.

Multi-Especialista

A troca (ou compartilhando) de informações, por meio de um *quadro-negro* pode ser relativa ao artefato ou ao controle da solução de um problema. Os fatos iniciais e os diferentes resultados parciais representam as versões sucessivas do artefato (objeto em concepção) que é armazenado neste tipo de estrutura de dados. O *quadro-negro* é o único meio de troca de informações. Esta arquitetura possui a vantagem de assegurar a coerência das trocas, mas tem o inconveniente em ser potencialmente um gargalo de estrangulamento no fluxo de informações. Este modelo foi

utilizado em inúmeras aplicações, notadamente em concepção na engenharia. Shen & Barthès (1996) apresentam uma longa lista de projetos que utilizam a arquitetura *quadro-negro* no domínio da concepção. Pode-se também encontrar uma discussão detalhada sobre a abordagem *quadro-negro* em Hayes-Roth (1995) e em Corkill *et al.* (1986).

Multi-Agente

Cada agente constrói seu próprio modelo da solução em andamento para um problema por meio da aquisição de informações provenientes de outros agentes. Este modelo de troca requer a especificação de um protocolo e de um formato de mensagem (linguagem comum) para expressar demandas e fornecer resultados. Cada agente armazena localmente a solução em andamento do problema em questão, ou uma parte desta solução, em sua base de fatos. Geralmente, em um sistema multi-agente, seus componentes são heterogêneos e não existe controle global; a comunicação pode ser síncrona ou assíncrona; ela pode ser *ponto-a-ponto*, *broadcast* ou *multicast*; e finalmente a semântica das mensagens trocadas entre os agentes é de alto nível. Um certo número de projetos, tais como: MARS [Abriat 91], PACTO [Cutkosky *et al.* 93], DIDE [Shen & Barthès 95] utilizam ou têm utilizado este tipo de arquitetura. Shen & Barthès (1996) apresentam igualmente uma longa lista de projetos no domínio da engenharia simultânea. Eles indicam em particular que a comunicação assíncrona, presente nos sistemas multi-agente, convém à um ambiente distribuído no contexto de grandes projetos. Já a comunicação síncrona (estilo quadro-negro), é bastante utilizada entre os agentes que devem trabalhar simultaneamente, porém não é apropriada aos problemas encontrados nos projetos de grande porte, para os quais o período de concepção é freqüentemente longo [Shen & Barthès 95].

- a comunicação assíncrona nos sistemas Multi-Agente convém perfeitamente à um ambiente distribuído no contexto de projetos de grande porte, e
- a comunicação síncrona (estilo quadro-negro) convém aos agentes que devem trabalhar simultaneamente.

Figura 2: Pontos retidos na análise de Shen & Barthès e Scalabrin.

No presente trabalho, há um maior interesse pelos sistemas multi-agente porque eles oferecem a possibilidade de distribuir os conhecimentos e o controle de uma aplicação mais facilmente que nos sistemas multi-especialistas.

Mais precisamente e segundo Ferber (1995), um *sistema multi-agente* é constituído por:

- um ambiente E, ou seja um espaço dispondo em geral de uma métrica;
- um conjunto de objetos O. Estes objetos são situados, ou seja, para todo objeto, é possível, em um dado momento, associar uma posição em E. Estes objetos são passivos, ou seja, eles podem ser percebidos, criados, destruídos e modificados pelos agentes;
- um conjunto A de agentes, que são objetos particulares ($A \subseteq O$), os quais representam as entidades ativas do sistema;
- um conjunto de relações R unindo os objetos (e agentes) entre eles,
- um conjunto de operações Op permitindo aos agentes de A perceber, produzir, consumir, transformar e manipular objetos de O; e
- operadores encarregados de representar a aplicação destas operações e a reação do mundo à esta tentativa de modificação, que oportunamente chamar-se-á leis do universo.

Em uma arquitetura multi-agente, os principais conceitos são: ambiente, agente, organização (reconfiguração dinâmica), e as vezes a noção de tempo real. Na seqüência, serão apresentados apenas alguns destes conceitos. Porém, o leitor poderá, em particular, encontrar discussões mais completas sobre o assunto em [Ferber 95], [Shmeil 99], [Scalabrin 96a].

2.2. Existem agentes

Em um sistema multi-agente existem agentes ...

> *"Um agente é uma entidade computacional permanente que pode perceber seu ambiente, raciocinar e agir tanto sozinho quanto com outros agentes." [Singh 98].*
>
> *"Um agente é uma entidade física ou virtual que é capaz de agir num ambiente, que pode comunicar diretamente com outros agentes, que é munida por um conjunto de tendências (sob a forma de objetivos individuais), que possui recursos próprios, que é capaz de perceber (de maneira limitada) seu ambiente, que dispõe de uma representação parcial desse ambiente, que possui competências e oferece serviços, que pode eventualmente se reproduzir, cujo comportamento é satisfazer seus objetivos — levando em conta seus recursos e suas competências — em função de sua percepção, de suas representações e de suas comunicações." [Ferber 95].*
>
> *"Agentes são entidades computacionais, dotadas de capacidades cognitivas (percepção, raciocínio e memória) ou reativas, que agem ou reagem no domínio para as quais foram concebidas." [Shmeil 99]*

> *"Um agente é uma entidade autônoma inteligente que age racionalmente e intencionalmente conforme seus objetivos e conhecimento." [Prado 96].*
>
> *"Um agente é uma entidade autônoma (software) que tem conhecimentos especializados, possui uma representação desses conhecimentos, possui uma representação do problema a tratar, possui uma representação parcial do ambiente, possui protocolos de comunicação especializados e as vezes um modelo de suas intenções." [Scalabrin 96a].*

De forma geral, o principal componente que caracteriza um agente é *a autonomia*, a qual possui uma relação direta com a arquitetura interna dos agentes.

2.2.1. Entidade ativa e autônoma

Um agente autônomo age sem a intervenção direta do ser humano ou de qualquer outro agente, e possui certos tipos de controle sobre suas ações. Ele também possui um estado interno [Wooldridge & Jennings 94]. Para Beer (1992), um agente autônomo é um sistema que foi concebido para satisfazer objetivos de maneira automática interagindo com o ambiente onde ele está situado. Para Demazeau & Müller (1990), um agente autônomo é um agente cuja existência justifica-se independentemente da existência de outros agentes. Para Castelfranchi (1990), um agente autônomo deve tomar decisões e ter preferências. "Ele deve igualmente possuir seus próprios objetivos, ser capaz de tomar decisões a respeito de seus objetivos (e então resolver seus conflitos internos), adotar os objetivos dos outros agentes, porém aplicando um critério de escolha sobre eles, ver a adoção como um meio que o permite alcançar um de seus objetivos e de controlar a aquisição das crenças. Um agente autônomo age sem a intervenção dos seres humanos ou de qualquer outro agente, e detém o controle de suas ações e seu estado interno".

Jennings & Wooldridge (1996) definem um agente autônomo como sendo um sistema de computação físico ou mais freqüentemente lógico que possui as seguintes propriedades:

- *autonomia:* o agente opera sem intervenção direta do ser humano ou de uma outra entidade, e possui certos tipos de controle sobre suas ações e sobre seu estado interno;
- *comportamento social:* o agente interage com outros agentes (as vezes seres humanos) via certos tipos de linguagens de comunicação;
- *reatividade:* o agente percebe seu ambiente e responde de maneira oportuna as mudanças de seu ambiente;
- *pró-atividade:* o agente age em resposta a eventos externos, porém é capaz de exibir um comportamento guiado por objetivos, tomando iniciativa.

Para Etzioni (1995), um agente autônomo é capaz de tomar a iniciativa e de exercer um grau de controle *não trivial* sobre suas próprias ações:

- um agente aceita requisições de alto nível, indicando o que um ser humano deseja, e é responsável pela decisão e pelo modo como vai satisfazer estas requisições;
- um agente não obedece comandos cegamente, mas tem a capacidade de modificar as requisições, de solicitar explicações ou mesmo recusar-se a satisfazer certas requisições;
- um agente possui ações não determinísticas; ele é capaz de escolher dinamicamente, em resposta ao estado de seu ambiente externo, qual ação executar e em que ordem;
- um agente pode perceber certas mudanças em seu ambiente e decidir quando agir, em oposição aos programas clássicos que são diretamente invocados por seu usuário.

Deste modo, a noção de autonomia é sinônimo de auto controle e de assincronismo. Os problemas práticos decorrentes desta autonomia são:

- a comunicação (linguagem e vocabulário comum);
- a noção de ambiente "aberto".

Para ser autônomo, cada agente deve ser dotado de um mecanismo de controle a fim de poder gerenciar suas diferentes atividades em função de seu estado interno e do estado do mundo exterior.

É importante notar que a redundância de informações e de capacidades em um domínio especifico fornece, à um grupo de agentes, um conjunto de soluções alternativas a uma mesma situação. Para Gouveia (1992), a alocação dinâmica de recursos, um dos principais objetivos deste trabalho, é uma característica essencial destes sistemas, visto que é impossível *a priori* prever tanto a evolução da sua composição (população), quanto a evolução do seu ambiente — e logo de suas necessidades.

O conjunto de mecanismos de controle de um agente é uma das partes integrantes da sua arquitetura interna.

2.2.2. Arquiteturas

Os principais tipos de arquiteturas são:

- a reativa e
- a cognitiva.

Neste trabalho foi escolhido o modelo de agente cognitivo; ele é cognitivo à medida que ele pode negociar uma informação, uma nova segmentação de um campo de um cheque e/ou a execução de *subtarefas* especificas para resolver um problema.

Arquitetura reativa

Os agentes reativos possuem um comportamento bastante simples e eles são freqüentemente numerosos em um sistema multi-agente. O comportamento inteligente do sistema resulta da coexistência e da combinação de comportamentos simples; os agentes reativos utilizam constantemente o ambiente para comunicar. Eles dispõem de um protocolo de comunicação e de uma linguagem de comunicação reduzida. Por exemplo, o modelo cliente-servidor utiliza um protocolo de comunicação e uma linguagem de comunicação bastante simples, à medida que os servidores reagem apenas quando eles recebem um sinal do tipo *questionamento*; igualmente para os clientes que reagem apenas quando eles recebem um sinal do tipo *resposta*. Na literatura, as aplicações mais presentes concernem os robôs. Por exemplo, o comportamento complexo de um robô pode ser decomposto em comportamentos individuais simples. Esses comportamentos são colocados em contato com o mundo real; o robô capta continuamente informações sobre o ambiente que o cerca. Deste modo, o robô faz parte do ambiente e reage baseado em um comportamento simples, ditado pelos estímulos/respostas que podem chegar. Dentre os trabalhos sobre os agentes reativos pode-se citar:

- Brooks (1986) realizou os primeiros estudos sobre os agentes reativos no MIT. A idéia central é que a concepção de um robô inteligente e autônomo se traduz na criação de um conjunto de camadas que agem como pequenos robôs reativos.
- Ferber & Jacopin (1990); Ferber & Drogoul (1992) consideram que a resolução distribuída de um problema é uma série de interações bastante simples dentro de uma população de agentes. A solução do problema emerge das interações entre os agentes.
- Steels (1989) tentou resolver o problema da coleta de minério por robôs em ambiente desconhecido.

Arquitetura cognitiva

Por que um modelo cognitivo? Pelo gênero de aplicação que estamos interessados, este modelo parece operacionalmente preferível por aproximar-se do modelo de uma *sociedade de especialistas*. Neste modelo, encontram-se as noções dos modelos de si; dos outros; de crenças; de intenções; de atos da fala; etc. Estes modelos permitem implementar métodos complexos de cooperação e de negociação.

Representação dos outros agentes

Nos sistemas de agentes, a interação é o elemento fundamental. É a partir da interação que a solução de um problema pode surgir. Assim, uma das formas que um agente possui para assegurar uma boa coordenação de suas ações, é possuir uma representação (a mais fiel possível) dos outros agentes. Por exemplo, para alocar uma tarefa, um agente deve conhecer o agente que é capaz de executá-la mediante certas preferências. Da mesma forma, para formar um plano coletivo, os agentes possuem a necessidade de conhecer as capacidades e os conhecimentos dos outros agentes [Conry *et al.* 88, 91]. Para obter uma interação razoável entre os agentes, cada um deles deve dispor de um modelo dos outros [Gasser *et al.* 88a; Gasser *et al.* 88b; Wesson *et al.* 81]. Enfim, um agente deve também raciocinar sobre suas próprias ações para avaliar seus efeitos sobre o processo de coordenação.

O modelo que um agente tem dos outros é o meio pelo qual ele pode representar o que ele conhece dos outros agentes, chamado *acquaintance* [Gasser *et al.* 88b]. Este modelo pode definir as crenças, as competências, os objetivos ou planos dos outros agentes, do ponto de vista do agente que os modela. Esse modelo deveria permitir ao agente compreender, explicar, ou mesmo predizer as ações dos outros assim como as suas [Khoualdi 94]. As redes contratuais [Smith 80; Smith & Davis 81; Davis & Smith 83] são exemplos que evidenciam a importância desse tipo de modelo. Porque, através de seu modelo, um agente pode conhecer os agentes que são potencialmente capazes de responder à uma chamada de oferta. A modelagem permite então reduzir o número de mensagens enviadas inutilmente. Entretanto, em certos casos o modelo que um agente possui dos outros tem pouca importância, em especial à medida que as condições ambientais mudam freqüentemente.

De uma maneira geral, segundo Gasser (1992), a representação que um agente faz dos outros agentes apresenta as seguintes vantagens:

- auxilia-o a planejar e a coordenar suas ações;
- permite-lhe raciocinar sobre as crenças e as intenções dos outros agentes, e por conseqüência saber o que comunicar aos outros;
- permite-lhe conhecer "quem faz o que" e a quem endereçar-se para obter uma informação [Smith & Davis 81].
- permite aos agentes coordenar suas ações sem comunicar [Genesereth et al. 86];
- permite reduzir a comunicação, visto que apenas as informações úteis são comunicadas (e.g., as listas de interesses em KQML [Finin 93])
- permite aos agentes tomar decisões locais devido a descentralização do controle;

A respeito da representação de si, em Scalabrin *et al.* (1996b), os agentes modelam essencialmente suas próprias competências e suas necessidades. As necessidades de um agente são modeladas para servir de regulador dos conhecimentos à armazenar sobre os outros; *i.e.,* o modelo dos outros agentes irá conter apenas as informações sobre os agentes que são potencialmente capazes de satisfazer suas necessidades.

As crenças

Claramente a representação dos outros agentes é fundamental nos sistemas multi-agente. Entretanto, os conhecimentos contidos neste modelo não são necessariamente exatos. Fala-se então de crenças. A distinção entre crença e conhecimento é que a primeira pode ser verdadeira ou falsa, enquanto que a segunda é sempre verdadeira. Pode-se citar duas maneiras de definir as crenças:

- em Scalabrin et al. (1996b), elas representam fatos e portam sobre os valores a priori que podem existir nas representações dos outros. Essas crenças são atualizadas (inserção, supressão) no momento da recepção de mensagens.
- Moses & Shoham (1993) propuseram uma definição de conhecimento e de crença, que é o inverso da definição clássica, que considera que "o conhecimento é uma crença verdadeira", i.e., "crença é um conhecimento que pode ser anulado".
- Rao & Georgeff (1991) definem que as crenças de um agente correspondem a informação que o agente possui sobre o mundo, e estas crenças podem ser completas ou incompletas.

As intenções

As intenções correspondem a um outro estado mental de um agente. Um agente autônomo deve agir em função de suas intenções. Cohen & Levesque (1988) propõem (em detalhe) um formalismo que permite descrever as intenções de um agente. Este último é construído em função das crenças que o agente possui sobre o mundo. O agente efetua uma ação somente se ele possui a intenção para tal e se esta permite satisfazer um objetivo que ele se deu. Entretanto, as intenções são as representações das ações possíveis que o sistema pode efetuar para alcançar seu objetivo; pode ocorrer que elas jamais sejam executadas.

Atos da fala

Os atos da fala [Cohen 78; Cohen & Perrault 79; Chaib-draa & Vanderveken 95], resultam da esperança de obter uma ferramenta robusta para expressar as crenças dos agentes. Em um universo multi-agente, o ato da comunicação não é uma simples troca de mensagens entre

agentes. Ele consiste em saber *o que, quando* e *com quem* comunicar. Todo ato de comunicação é um ato intencional que se traduz pela modificação das crenças dos outros. Ele pode ser utilizado pelo emissor para comunicar suas competências, suas intenções, suas preocupações aos demais agentes. O envio de mensagem pode ser igualmente um *questionamento* ou uma *ordem*. Na seção 2.3 será detalhado a linguagem de comunicação entre agentes, *denominada KQML* ("**K**nowledge **Q**uery for **M**anipulation **L**anguage"), que é uma implementação dos atos da fala. *KQML* [Finin *et al.* 93 e 96; Mayfield *et al.* 95; e Finin & Labrou 97] será utilizada nesta dissertação para definir a comunicação entre os agentes do sistema *Pandora*.

A cooperação

Em um sistema multi-agente, a solução de um problema é em geral distribuída entre diferentes agentes. *I.e.,* nenhum agente possui os conhecimentos suficientes para resolver individualmente um problema. Assim, os agentes devem organizar suas atividades a fim de otimizar seus recursos permitindo ações coletivas. Em outras palavras, os agentes são levados a cooperar. A similaridade dos comportamentos dos agentes com os dos seres humanos, conduz a comportamentos sociais bastante variados. Por exemplo, o comportamento determinado pela metáfora da cooperação pode ser complexo como: o estabelecimento de contratos, a delegação e a associação. Os comportamentos correspondem à diferentes modelos de organizações. Estas organizações podem ser dinamicamente reconfiguráveis e evoluir para uma hierarquia complexa como sendo o resultado da otimização do fluxo de informações entre os indivíduos (os diferentes custos — da coordenação, da transferência de informação, da utilização/compartilhamento de diferentes recursos — em alguns tipos de organizações são analisados por Malone 1987).

A cooperação entre os agentes passa em geral pela troca de informações. Estas trocas se dão pelo compartilhamento de tarefas ou de resultados parciais [Smith & Davis 81]. O compartilhamento de tarefas é realizado com o objetivo de balancear a carga computacional de um sistema, à medida que um problema global é dividido em subproblemas e cada um destes é alocado à um agente específico do sistema. Por exemplo, no trabalho aqui desenvolvido, os agentes compartilham tarefas quando um novo cheque entra no sistema e este cheque é subdividido entre os agentes *data*, *assinatura*, *numérico* e *literal*, com o intuito de que cada agente específico efetue o reconhecimento de cada parte do cheque que lhe é confiada. Os resultados parciais são compartilhados baseando-se nas diferentes perspectivas que cada agente específico possui sobre o problema global, com o intuito de, por exemplo, confirmarem uma determinada hipótese. Pode-se perceber claramente este tipo de compartilhamento entre os agentes *numérico* e *literal,* que reconhecem a mesma informação (codificada em formatos

diferentes) e trocam resultados a fim de aumentar uma crença ou confirmar se os campos lógicos são consistentes.

Deve-se também salientar que, no compartilhamento de tarefas, o controle é tipicamente guiado por objetivos, *i.e.,* o processamento efetuado por um agente visa realizar um objetivo, cujo resultado pode ser utilizado para resolver um problema global. Enquanto que, no compartilhamento de resultados parciais, o controle é tipicamente guiado por dados, *i.e.,* o processamento efetuado em dado instante por um agente depende dos dados que ele tem disponível localmente ou externamente.

Segundo Durfee *et al.* (1987), a cooperação entre agentes deve ser feita levando em conta os objetivos visados, por exemplo:

- acelerar a solução de um problema, privilegiando o trabalho paralelo dos agentes;
- obter várias soluções locais, utilizando as capacidades dos outros agentes para obter uma solução local;
- melhorar a confiabilidade dos resultados, utilizando o fato que os agentes podem verificar os resultados;
- reduzir a possibilidade de duplicação de processamento, atribuindo um subproblema a um número limitado de agentes, por exemplo a um único;
- reduzir o volume de comunicação, trocando apenas as informações necessárias.

Um ponto importante vem também do fato que para implementar esses modos de cooperação, existe a necessidade de protocolos de comunicação relativamente elaborados. Davis & Smith (1983), utilizam a metáfora da negociação de contrato para propor um protocolo de alto nível. A negociação é uma discussão na qual os agentes interessados trocam informações a fim de chegar a um acordo sobre um dado serviço. Deve-se salientar que os conhecimentos para a atribuição de um contrato variam de uma aplicação para outra.

Modelos de negociação

Nos sistemas multi-agente, distingue-se basicamente dois tipos de negociação:

- centralizada; e
- distribuída.

A negociação centralizada pressupõe a existência de uma visão global do plano. Um agente central raciocina sobre o conjunto de ações dos diferentes agentes. Ele trata os conflitos entre os agentes estabelecendo um plano para os mesmos (uma discussão é apresentada em Cammarata *et*

al. 1983). *Na negociação distribuída*, tal agente não existe. Cada agente cria seu próprio plano[1]. Os agentes trocam seus planos parciais para detectar e evitar conflitos. Como exemplo de negociação distribuída, pode-se citar os trabalhos de Smith (1980), Davis & Smith (1983), Scalabrin (1996c) sobre a alocação dinâmica de tarefas, de Conry *et al.* (1988 e 1991) sobre a resolução de conflitos de recursos e de objetivos, de Sandholm (1993) sobre o problema de transporte de cargas, de Shmeil (1999) sobre a negociação da compra/venda de produtos ou serviços.

Outros trabalhos realizados sobre a negociação como mecanismo de cooperação são, por exemplo, os sistemas de preço [Mullen & Wellman 95; Fischer *et al.* 96] e de compartilhamento de recursos [Rosenschein & Zlotkin 94].

2.3. Comunicação entre agentes

A comunicação entre os agentes cognitivos é fundamental e requer uma linguagem de comunicação apropriada. Existem várias linguagens de comunicação. ACL ("**A**gent **C**ommunication **L**anguage") de Mayfield *et al.* (1995) representa o maior esforço até o momento no sentido de propor uma linguagem padrão para comunicação entre agentes. Ela permite que os agentes troquem conhecimentos utilizando interfaces declarativas. Uma mensagem ACL é uma mensagem KQML[5], composta de uma diretiva de comunicação (ou ato da fala) e de um conteúdo semântico expresso em termos de um vocabulário codificado em um formato de troca. Cada mensagem ACL tem três componentes:

- um vocabulário ligado a semântica própria de um domínio (ontologia),
- uma linguagem denominada KIF[6] ("**K**nowledge **I**nterchange **F**ormat") para codificar o conteúdo da mensagem, e
- uma linguagem de comunicação denominada KQML.

A linguagem KQML possui, por sua vez, três níveis representados na Figura 3.

Conteúdo		(fator-certeza 0.89)
Comunicação	Emissor, receptor, id-msg	(fator-certeza 0.89)
Mensagem	ask-one Emissor, receptor, id-msg	(fator-certeza 0.89)

Figura 3: Níveis da linguagem de comunicação KQML.

[1] O planejamento é discutido em profundidade na tese de doutoramento de Gouveia (1992).
[5] Desenvolvida pelo ARPA no Programa de Compartilhamento de Conhecimento (*Knowledge Sharing Effort*)
[6] KIF foi adotada como padrão pela ANSI e está sendo estudado pela ISO. KQML está sendo avaliado pela OMG.

O nível conteúdo diz respeito ao conteúdo real de uma mensagem. Este conteúdo é expresso em qualquer linguagem de representação (e.g., KIF), utilizando os termos de um domínio definido sob a forma de uma ontologia.

O nível comunicação codifica o conjunto de características da mensagem e descreve os parâmetros de mais baixo nível, tais como: identificador do emissor e receptor. Trata-se de identificadores únicos associados à mensagem.

O nível mensagem codifica a mensagem que um agente deseja transmitir. Este nível forma o núcleo do sistema KQML. Ele determina o tipo de interação que um agente pode ter com seu correspondente e ocorre independentemente da sintaxe e do contexto ontológico. A função principal desse nível é identificar o protocolo a ser utilizado para expedir uma mensagem e fornecer uma *performativa* (e.g., ask-one), que será anexada ao conteúdo. Este nível inclui também características operacionais que descrevem a linguagem do conteúdo e a ontologia. Uma vez que a mensagem é codificada, ela é passada ao sistema de transporte para ser expedida. O conteúdo não é visível pela linguagem de comunicação do agente. Deve-se salientar que KQML não é uma linguagem homogênea, porque ela fornece uma linguagem para o conteúdo, porém não fornece para os demais parâmetros da mensagem.

Os trabalhos a seguir tentam suprir algumas lacunas do KQML que permitem, em particular, a codificação de tipos relativos aos comportamentos dos agentes:

- a linguagem COOL de Barbuceanu & Fox (1995) fornece ao KQML uma estrutura que implementa mecanismos de interação ("define-conversation") que facilitam a descrição de diálogos entre os agentes.
- Shmeil (1999) define um protocolo de alto nível que permite efetivamente implementar a negociação. Ele introduz, no processo de conversação, as noções de estratégia, tática e critérios, os quais fornecem um mecanismo que permite fazer evoluir uma discussão.

Segundo Mayfield *et al.* (1995), uma linguagem de comunicação entre agentes deve possuir os seguintes aspectos:

- *forma*: uma linguagem de comunicação de agentes razoável deve ser declarativa, sintaticamente simples e legível. Além disso, sua sintaxe deve ser extensível;
- *conteúdo*: uma linguagem de comunicação deve ser estendida de modo que adapte-se facilmente com outros sistemas, e deve fornecer um conjunto pré-definido de primitivas;

- *semântica*: a descrição semântica de uma linguagem é feita através da linguagem natural, contudo, uma descrição formal é necessária para satisfazer algumas propriedades, e.g., ser baseada em uma teoria, não ser ambígua e etc;
- *implementação*: a implementação deve ser eficiente. Ela deve possuir uma interface fácil de utilizar e deve, ainda, esconder dos usuários detalhes das camadas de rede. A linguagem deve ser apropriada para que agentes inteligentes possam manusear apenas um subconjunto de primitivas;
- *rede*: uma linguagem de comunicação entre agentes deve suportar conexões síncrona e assíncrona, além de todos os modos básicos conexões: ponto-a-ponto, multicast e broadcast. A linguagem deve conter um rico conjunto de primitivas que podem servir como um substrato sobre as quais linguagens de alto-nível e protocolos possam ser desenvolvidos. Estes protocolos de alto-nível devem ser independentes dos mecanismos de transporte (e.g., TCP/IP, email, http, etc.) utilizados;
- *ambiente*: o ambiente deve ser altamente distribuído, heterogêneo e extremamente dinâmico. Ele deve suportar interoperabilidade com outras linguagens e protocolos;
- *confiabilidade*: a linguagem de comunicação deve suportar uma comunicação confiável e segura. Devem ser oferecidos recursos para trocas privadas e seguras.

Deve-se salientar, que para que os agentes KQML cooperem, existe a necessidade de requerer os serviços de um agente especializado, que funciona como um *mediador/facilitador* entre os agentes clientes e os agentes provedores de serviços. Pode-se citar, como exemplo facilitador, o sistema de serviço *Matchmaker* desenvolvido por Kuokka & Harada (1995), no contexto do projeto SHADE.

O conjunto de mensagens KQML pode ser estendido desde que as novas performativas criadas obedeçam a mesma forma da especificação original da linguagem. Os agentes que estão de acordo com KQML não precisam reconhecer todas as mensagens, de forma que um pequeno subconjunto pode ser suficiente para um determinado sistema, *i.e.,* dependendo da necessidade, pode-se escolher apenas algumas performativas a serem utilizadas na comunicação.

2.4. Sistema aberto *"à la Hewitt"*

Os agentes que compõem um sistema multi-agente são entidades dotadas de uma certa autonomia e inteligência. Pode-se dizer que o tipo de sociedade que define um sistema pode ser dividida em dois grupos: sistema fechado e sistema aberto [Faraco 98]. No primeiro caso, o número de agentes sempre é o mesmo, *i.e.,* não entram e não saem agentes da sociedade. Por

outro lado, em um *sistema aberto*, os agentes podem entrar e sair da sociedade e o número inicial de agentes pode variar durante a execução do sistema. A escolha de qual abordagem utilizar depende muito do problema que pretende-se resolver. Para o presente trabalho a atenção está voltada ao segundo tipo de sistema, i.e., um sistema aberto.

Pode-se caracterizar um *sistema aberto* da seguinte forma [Hewitt 1988]:

- *concorrência*: sistemas abertos são compostos por numerosos componentes e, para tratar o fluxo simultâneo de informações, estes componentes devem processar as informações de forma concorrente.
- *assincronismo*: existem duas fontes de assincronismo em sistemas abertos: (i) novas informações podem entrar no sistema a qualquer momento, requerendo que o sistema opere assincronamente e (ii) os componentes são fisicamente separados, impossibilitados de agirem sincronamente.
- *controle descentralizado*: o controle deve ser distribuído por todo o sistema, tal que as decisões locais possam ser tomadas onde são necessárias.
- *informação inconsistente*: informações externas ou de diferentes partes do sistema podem tornar-se inconsistentes. Portanto, decisões devem ser tomadas pelos componentes de um sistema aberto considerando qualquer evidência concorrentemente disponível.
- *independência*: a operação interna, a organização, e o estado de um agente é desconhecido e indisponível para o outro agente por razões de privacidade ou congestionamento da rede. Dessa forma, a comunicação deve ser explícita a fim de conservar energia e manter a segurança. Isto garante que cada agente seja mantido de forma simples.
- *continuidade*: sistemas abertos devem ser projetados de tal modo que a falha de um componente individual possa ser acomodada pelos componentes operantes, enquanto que o componente que falhou possa ser reparado ou substituído.

Deve-se observar, que o termo *sistema aberto*, é empregado freqüentemente para designar diferentes camadas de hardware e software com propósitos bem definidos [Scalabrin 96a]:

- a interconexão entre arquiteturas de máquinas diferentes (e.g., o modelo OSI).
- a ligação dinâmica entre cliente-servidor (e.g., o modelo OMG CORBA).
- a reconfiguração seletiva e dinâmica (e.g., modelo agente).
- a integração dinâmica de um agente dentro de um determinado contexto de trabalho.

Este trabalho enquadra-se principalmente no modelo agente, onde a preocupação é a inserção/remoção de agentes do sistema sob demanda.

2.5. Considerações finais

Nesta seção foi apresentado o conceito de sistema multi-agente e seus componentes, destacando o uso de agentes cognitivos como a abordagem escolhida no contexto da aplicação.

O modelo agente cognitivo deve permitir a implementação de um *sistema aberto*, onde enfatizamos a reconfiguração seletiva e dinâmica, que possibilita a inserção/remoção de agentes do sistema *sob demanda*.

Um outro ponto importante, trazido pela abordagem adotada, é a interação entre os agentes. Porém, esta interação exige a criação de um mecanismo simples e eficiente para decidir *quando* e *como* um agente deve interagir com os demais agentes do sistema. A interação se dará através da troca de mensagens. Esta troca de mensagens, propriamente dita, será implementada utilizando a linguagem de comunicação KQML. Durante as trocas de mensagens os agentes poderão compartilhar resultados no intuito de confirmar uma hipótese ou aumentar uma crença. Em outras palavras, os agentes são levados a cooperar, o que permite obter resultados mais confiáveis ou mesmo impossíveis em uma abordagem distribuída tradicional.

3. Programação Lógica Paraconsistente

Devido a necessidade de se projetar sistemas mais eficientes e capazes de considerar situações reais envolvendo informações inconsistentes ou contraditórias, várias pesquisas estão sendo realizadas. Elas têm por objetivo aplicar lógicas alternativas à lógica clássica, denominadas *lógicas não-clássicas*. Esta necessidade apresenta-se, porque as *lógicas não-clássicas* tal como a *lógica paraconsistente*, investigam a existência de outros valores-verdade além de "verdadeiro" e "falso".

A *lógica paraconsistente* é uma extensão da *lógica clássica* e foi desenvolvida com o intuito de encontrar meios de permitir um tratamento não trivial às informações contraditórias. Segundo Newton da Costa, as contradições ou inconsistências têm como origem as condições do ambiente em que desenvolvem-se as tarefas. *Como* e *quando* estas situações contraditórias aparecem é, na maioria das vezes, independente da vontade dos dispositivos envolvidos.

> O objetivo aqui será estudar a *lógica paraconsistente* visando implementar mecanismos simples e eficazes, que facilitem o compartilhamento de resultados parciais entre agentes, em particular para levar em conta o tratamento de informações inconsistentes, o que deverá ser freqüente entre os agentes que reconhecem os montantes numérico e literal de um cheque.

Com o avanço tecnológico é impossível admitir a resolução de problemas que apresentem inconsistências simplesmente ignorando-as ou, ainda, refutando-as como falsas ou confirmando-as como verdadeira. Dessa forma, a *lógica paraconsistente* surge como uma ferramenta para modelar o comportamento humano de forma mais completa e adequada.

Diferentemente das metodologias de raciocínio quantitativo (Blair 87; Van Emden 86; Subrahmanian 87a; Subrahmanian 87b) foi desenvolvido um sistema lógico paraconsistente estendido (Blair 88; da Costa 90; da Costa 63) capaz de fornecer interpretações para informações que são contraditórias, isto é, p e $\neg p$. A presença de tais interpretações pode apresentar, de forma explícita, a inconsistência nos dados e expressar informações importantes para a tomada de decisão [Enembreck *et al.* 99].

A *lógica evidencial paraconsistente é* uma classe de *lógicas paraconsistentes*, cujos estudos consideram os resultados da Lógica Evidencial.

A seguir faremos uma apresentação da programação *lógica evidencial paraconsistente* que será base para o desenvolvimento deste trabalho.

3.1. Programação Lógica Evidencial Paraconsistente

Em vários casos do mundo real, *evidências* possuem papéis chave nas tomadas de decisões. Quando um indivíduo precisa tomar uma certa decisão e possui muitas informações imprecisas, ele pensa em todas as possibilidades, analisando todas as evidências e finalmente realiza uma ação baseando-se nas mesmas.

Um exemplo que ilustra o que foi supracitado é dado por Subrahmanian (1987b). Neste exemplo, considera-se um médico frente a um paciente cujos sintomas fazem ele acreditar, segundo suas experiências anteriores, que existe 95% de evidência de que o paciente morrerá dolorosamente se tratado com uma droga *D*, 2% de evidência de que o paciente sobreviverá (mas ficará em coma) se tratado com a droga *D*, e 3% de evidência indeterminada. O médico pode muito bem tomar a decisão de não administrar a droga em questão no paciente. Mas como saber se esta é a melhor decisão a ser tomada?

De um ponto de vista mais técnico, crenças humanas nem sempre são verdadeiras [Hintikka 63], e portanto quando não existem informações suficientes para tomar uma decisão é necessário buscar outras evidências a fim de se chegar a uma decisão mais confiável.

Em Subrahmanian (1987b) são apresentados os fundamentos para o uso do raciocínio evidencial em programação lógica. No raciocínio evidencial, os valores-verdade são compostos por dois fatores evidenciais pertencentes a um reticulado $\{x \in \Re \mid 0 \leq x \leq 1\} \times \{x \in \Re \mid 0 \leq x \leq 1\}$ (Figura 4). No entanto, cada um desses fatores são associados a uma proposição p e podem ser pensados como a evidência favorável a proposição p e o outro a evidência contrária a proposição p. A única restrição colocada a esses valores é que eles devem pertencer ao intervalo [0,1], *i.e.*, infinitos valores podem estar associados às premissas do sistema. Isto permite usar a teoria dos átomos anotados para raciocinar sobre evidências (Subrahmanian 87a; Subrahmanian 87b; Blair 88).

Definição 1 : Se ρ é um átomo e $\mu_1, \upsilon_1 \in [0,1]$, então diz-se que $\rho:[\mu_1, \upsilon_1]$ é um *átomo anotado evidencialmente*. Neste caso, ρ chama-se parte atômica de $\rho:[\mu_1, \upsilon_1]$ enquanto $[\mu_1, \upsilon_1]$ chama-se *anotação evidencial* de $\rho:[\mu_1, \upsilon_1]$. μ_1, denomina-se *evidência favorável de* ρ e υ_1 a *evidência contrária de* ρ.

Podemos definir formalmente um reticulado infinito $\tau = < | \tau |, \leq >$, onde

$$\tau = \{ x \in \Re \mid 0 \leq x \leq 1 \} \times \{ x \in \Re \mid 0 \leq x \leq 1 \}$$

O reticulado τ ilustrado pelo diagrama de Hasse [Abe 91] é mostrado na Figura 4:

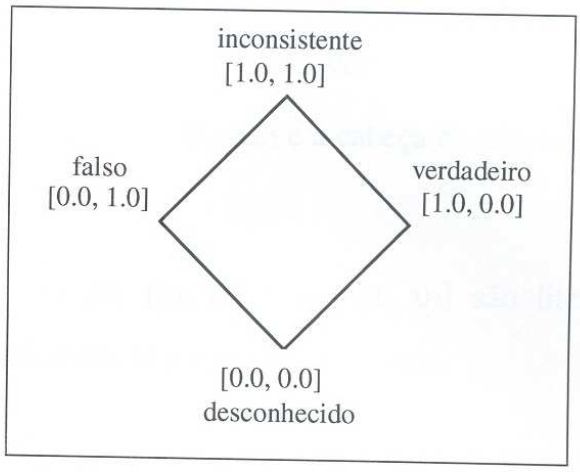

Figura 4: Reticulado Infinitamente Valorado τ.

No reticulado τ da Figura 4, os valores [1.0, 0.0], [0.0, 1.0], [0.0, 0.0], [1.0, 1.0] correspondem respectivamente a: *verdadeiro*, *falso*, *desconhecido* e *inconsistente*. Além da interpretação da inconsistência, um outro conceito introduzido pela *programação lógica evidencial paraconsistente*, em relação a *lógica clássica*, é a interpretação do desconhecido ([0.0, 0.0]). Neste caso, não se tem informação nem sobre a verdade e nem sobre a falsidade de uma premissa.

A seguir serão apresentados algumas definições utilizadas na construção e na inferência de *programas lógicos evidenciais* [Prado 96; Ávila 96; Enembreck *et al.* 99; Subrahmanian 87b; Blair 88].

Definição 2 [Negação] : O operador de negação ~: $|\tau| \rightarrow |\tau|$ é definido como:

$\sim : [\mu_1, \upsilon_1] = [\upsilon_1, \mu_1]$

Definição 3 [Conjunção] : A conjunção de $[\mu_1, \upsilon_1] \wedge [\mu_2, \upsilon_2] \in \tau$ é definida como:

$[\mu_1, \upsilon_1] \wedge [\mu_2, \upsilon_2] = [\min(\mu_1, \mu_2), \max(\upsilon_1, \upsilon_2)]$

Definição 4 [Disjunção] : A disjunção de $[\mu_1, \upsilon_1] \vee [\mu_2, \upsilon_2] \in \tau$ é definida como:

$[\mu_1, \upsilon_1] \vee [\mu_2, \upsilon_2] = [\max(\mu_1, \mu_2), \min(\upsilon_1, \upsilon_2)]$

Definição 5 [Grau de Certeza]: O grau de certeza c: $|\tau| \rightarrow |\tau|$ é dado por:

$c([\mu_j, \upsilon_j]) = \mu_j - \upsilon_j$

3.1.1. Sintaxe

Definição 6 [Literal Evidencial] : Se p é uma fórmula básica e $\mu, \upsilon \in \{x \in \Re \mid 0 \leq x \leq 1\}$, então diz-se que p: $[\mu, \upsilon]$ é um literal bem anotado e que $[\mu, \upsilon]$ é a anotação de p.

Definição 7 [Cláusula Evidencial] : Se p_0: $[\mu_0, \upsilon_0]$, ..., p_n: $[\mu_n, \upsilon_n]$ são literais bem anotados, então:

$$p_0: [\mu_0, \upsilon_0] \Leftarrow p_1: [\mu_1, \upsilon_1] \wedge ... \wedge p_n: [\mu_n, \upsilon_n]$$

chama-se cláusula evidencial, onde, p_0: $[\mu_0, \upsilon_0]$ é a cabeça da cláusula, enquanto p_1: $[\mu_1, \upsilon_1] \wedge ... \wedge p_n$: $[\mu_n, \upsilon_n]$ é o corpo da cláusula.

Definição 8 [Unificação] : Se p_1: $[\mu_1, \upsilon_1]$ e q_2: $[\mu_2, \upsilon_2]$ são literais, então diz-se que p_1: $[\mu_1, \upsilon_1]$ e q_2: $[\mu_2, \upsilon_2]$ são unificáveis se p e q são unificáveis.

Definição 9 [Programa Lógico Evidencial] : é qualquer conjunto finito não-vazio de cláusulas evidenciais.

É importante salientar que, diferentemente de outras teorias de raciocínio quantitativo como na *teoria da probabilidade*, os fatores evidenciais em *programas lógicos evidenciais* não estão relacionados [Enembreck *et al.* 99]. Isto significa que os referidos fatores evidenciais atuam de forma independentes.

3.1.2. Semântica

Todas as interpretações possuem como domínio a base de Herbrand [Van Emden 86; Subrahmanian 87a], isto significa dizer que o universo de indivíduos da interpretação consiste dos termos básicos da linguagem que está sendo interpretada. Uma interpretação é uma função $I : B_E \rightarrow \tau$, onde B_E é a base Herbrand e τ é o respectivo reticulado.

Definição 10 : 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:

$$I_1 \leq I_2 \text{ sse } (\forall p \in B_E)\ I_1(p) \leq I_2(p)$$

onde, B_E é a base Herbrand do *programa lógico evidencial*.

Definição 11 : Se E é um *programa lógico evidencial*, defini-se T_E como uma aplicação do conjunto de interpretações Herbrand de E em interpretações Herbrand de E, onde

$T_E = \sup \{ [\mu, \upsilon] | p:[\mu, \upsilon] \} \Leftarrow q_1: [\mu_1, \upsilon_1] \wedge ... \wedge q_k: [\mu_k, \upsilon_k]$ é a instância básica de uma cláusula evidencial em E e $I \models q_1: [\mu_1, \upsilon_1] \wedge ... \wedge q_k: [\mu_k, \upsilon_k] \}$

Definição 12 : Um modelo I que atribui valores-verdade $[\mu, \upsilon]$ ao átomo p, sendo $\mu + \upsilon > 1$ é dito *sobredeterminado*.

Definição 13 : Um modelo I do *programa lógico evidencial* E é dito *correto* em relação ao átomo $p \in B_E$, se $I(p) = [\mu, \upsilon]$ e $\mu + \upsilon \leq 1$.

Definição 14 : Um modelo I do *programa lógico evidencial* E é dito *completo* em relação ao átomo $p \in B_E$, se $I(p) = [\mu, \upsilon]$ e $\mu + \upsilon \geq 1$.

Definição 15 : Um modelo I do *programa lógico evidencial* E é *completo e/ou correto*, se é *completo e/ou correto* em relação a todo átomo $p \in B_E$.

Exemplo 01:

Considere o *programa lógico evidencial* E_1 descrito em Subrahmanian (1987b)

p(a): [0.0, 1.0]
p(a): [1.0, 0.0]

E_1 possui apenas um modelo I que atribui o valor verdade [1.0, 1.0] para p(a).

Definição 16 : Um *programa lógico evidencial* é *bem-comportado* se as cláusulas evidenciais de E satisfazem a seguinte condição:

Se C_1 e C_2 são duas cláusulas evidenciais em E, sendo suas cabeças $p_1: [\mu_1, \upsilon_1]$ e $p_2: [\mu_2, \upsilon_2]$ são unificáveis, então $max(\mu_1, \mu_2) + max(\upsilon_1, \upsilon_2) < 1$

3.1.3. Semântica Operacional

A semântica operacional dos *programas lógicos evidenciais* é dada em termos de uma busca em árvores E/OU.

Definição 17 : Se E é um *programa lógico evidencial*, p é um átomo na linguagem de E e $[\mu, \upsilon]$ uma anotação, defini-se uma árvore E/OU $T(E, p[\mu, \upsilon])$ da seguinte forma:

- a raiz de T(E, p:[μ, υ]) é um nó "OU" rotulado p:[μ, υ].
- se N é um nó "OU", então ele é rotulado por um literal anotado simples.

- cada nó "E" é rotulado por uma cláusula evidencial em E e por uma substituição θ.
- descendentes de nós "OU" são todos nós "E" e vice-versa.
- se N é um nó "OU" rotulado por p:[μ, υ] e se Cθ é uma instância de uma cláusula evidencial C em E da seguinte forma: $p:[\mu, \upsilon] \Leftarrow q_1:[\rho_1, \varphi_1] \wedge \ldots \wedge q_k:[\rho_k, \varphi_k]$, onde $[\mu_1, \upsilon_1] \geq [\mu, \upsilon]$, então existe um descendente N rotulado por C e θ. Um nó "OU" sem descendentes é chamado de nó não-informativo.
- se N é um nó "E" rotulado por uma cláusula evidencial C e a substituição θ, então para todo literal anotado p:[μ, υ] no corpo de C, existe um nó "OU" descendente rotulado por pθ:[μ, υ]. Um nó "E" sem descendentes é chamado de nó sucesso.

Associado a cada nó N da árvore E/OU definida anteriormente, há uma constante anotacional *v*(N) chamada *valor do nó*. Defini-se *v* assim:

- se N é um nó sucesso rotulado p:[μ, υ], então v(N) = [μ, υ].
- se N é um nó não-informativo, então v(N) = [0,0].
- se N é um nó "OU" que não é não informativo e seus descendentes são $N_1, \ldots, N_m$, então $v(N) = \sup\{v(N_1), \ldots, v(N_m)\}$.
- se N é um nó "E" não-terminal rotulado pela cláusula evidencial $p:[\mu, \upsilon] \Leftarrow q_1:[\rho_1, \varphi_1] \wedge \ldots \wedge q_k:[\rho_k, \varphi_k]$ e se o valor $v(N_i)$ de cada um dos nós descendentes de N_i rotulados q_i é tal que $v(N_i) \geq [\mu_i, \upsilon_i]$ para todo $1 \leq i \leq m$, então $v(N) = [\mu, \upsilon]$, senão v(N) = [0,0].

A seguir é apresentado um exemplo de um *programa lógico evidencial* e a árvore E/OU gerada. Este exemplo foi apresentado por Blair (1988).

Exemplo 02:

Seja τ o conjunto dos valores-verdade ordenados da Figura 4. Considere o *programa lógico evidencial* E sobre τ definido abaixo:

C_1: $p(X)$: [1.0, 0.0] ⇐ $q(X)$: [1.0, 0.0] ∧ $r(X)$: [1.0, 0.0]
C_2: $q(a)$: [1.0, 0.0]
C_3: $q(b)$: [0.0, 1.0]
C_4: $r(a)$: [1.0, 0.0]
C_5: $r(b)$: [0.0, 1.0]

Dado o questionamento $p(a)$: [1.0, 0.0] sobre a base de fatos acima, obtém-se a árvore E/OU a seguir. Deve-se salientar que a árvore gerada é trivial. Isto ocorre porque o *programa lógico evidencial* possui apenas anotações perfeitamente definidas — [1.0, 0.0] e [0.0, 1.0] — e consequentemente não possui informações conflitantes.

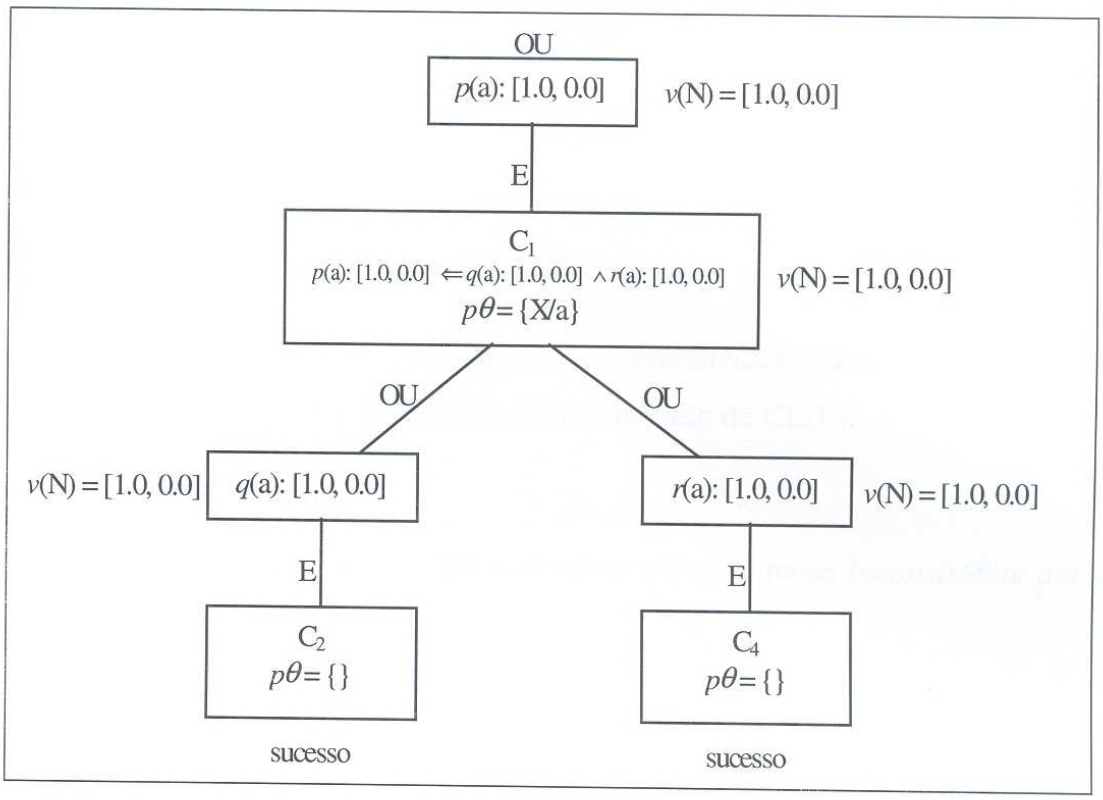

Figura 5: Árvore "E/OU"

Não é possível aplicar diretamente um procedimento de resolução padrão — *i.e.,* uma regra de inferência aplicável a fórmulas em forma clausal — em um *programa lógico evidencial* [Ávila 96]. Dessa forma, foi proposto em Subrahmanian (1987b) um procedimento de resolução chamado *resolução-SLDe* — *i.e.,* um procedimento de inferência seletiva linear aplicável a cláusulas definidas, no sentido em que deriva-se a cláusula vazia se e somente se o conjunto de cláusulas inicial é não satisfeito. Porém, este procedimento não pode ser aplicado diretamente a um *programa lógico evidencial* que não seja *bem-comportado* [Ávila 96]. Devido a essa restrição, em Blair (1988) é proposto um outro procedimento chamado *fechamento* [Enembreck *et al.* 99], dado a seguir:

Definição 18 : Um *programa lógico evidencial* é dito *fechado* se para quaisquer duas cláusulas evidenciais C_1 e C_2 pertencentes a E na forma:

$$p_1:[\mu_1,\nu_1] \Leftarrow q_1:[\rho_1,\varphi_1] \wedge \ldots \wedge q_k:[\rho_k, \varphi_k] \quad k \geq 0$$
$$p_2:[\mu_2,\nu_2] \Leftarrow q_1:[\upsilon_1,\varpi_1] \wedge \ldots \wedge q_k:[\upsilon_n,\varpi_n] \quad n \geq 0$$

tais que p_1 e p_2 são unificáveis — através de uma unificação mais geral — e $[\mu_1,\nu_1]$ e $[\mu_2,\nu_2]$ são não comparáveis, há uma cláusula evidencial

$$p_1\theta : \mathbf{sup}\{[\mu_1,\nu_1],[\mu_2,\nu_2]\} \Leftarrow q_1:[\rho_1,\varphi_1] \wedge \ldots \wedge q_k:[\rho_k, \varphi_k] \wedge q_1:[\upsilon_1,\varpi_1] \wedge \ldots \wedge q_k:[\upsilon_n,\varpi_n]$$

representada por λ (C_1, C_2), onde *sup* representa o maior valor existente entre os valores evidenciais das cláusulas C_1 e C_2.

Definição 19 : Seja E um *programa lógico evidencial* e

$$A_1(E) = E$$
$$A_{n+1}(E) = A_n(E) \cup \{\lambda\ (C_1, C_2) \mid C_1, C_2 \in A_n(E)\}\ (n \geq 1).$$

Para todo *programa lógico evidencial* E existe um inteiro *n*, tal que $A_n(E) = A_{n+1}(E)$. $A_n(E)$ é chamado *fechamento de E* e é denotado por *CL(E).*

Teorema 20 : Suponha que E é um *programa lógico evidencial* fechado, então:

(1) *I* é um modelo de E sse *I* é um modelo de CL(E).

(2) $T_E = T_{CL(E)}$.

Definição 21 : Um *programa lógico evidencial* sobre τ denomina-se *inconsistente por negação* se existe alguma cláusula evidencial $p:[\mu, \upsilon]$, $p \in B_E$, tal que:

(1) $T_E \uparrow w \models p:[\mu,\upsilon]$
(2) $T_E \uparrow w \models p:[\upsilon,\mu]$

Isto significa dizer que a iteração de T_E sobre um limite ordinal *w* satisfaz ($\models$) tanto $p:[\mu,\upsilon]$ quanto $p:[\upsilon,\mu]$.

Definição 22 : Um *programa lógico evidencial* E é dito *não trivial* se existe alguma cláusula evidencial $p:[\mu,\upsilon]$ tal que $T_E \uparrow w$ não satisfaz $p:[\mu,\upsilon]$.

Definição 23 : Um *programa lógico evidencial* E é dito *paraconsistente* se E for *inconsistente por negação* e *não trivial.*

3.1.4. Grau de Inconsistência e Subdeterminação

O reticulado da Figura 4 pode ser transferido para o plano cartesiano como mostra a Figura 6.

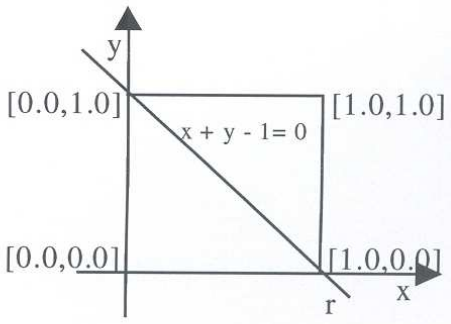

Figura 6: Reticulado no Plano Cartesiano.

Definição 24 : Uma cláusula evidencial $p{:}[\mu_1, \upsilon_1]$, é dita *perfeitamente definida* se $\mu_1+\upsilon_1= 1$. De maneira semelhante, a cláusula $p{:}[\mu_1, \upsilon_1]$ diz-se *subdeterminada* (ou subdefinida) se $\mu_1+\upsilon_1 < 1$. Por outro lado, a cláusula $p{:}[\mu_1, \upsilon_1]$ diz-se *sobredeterminada* (ou inconsistente, ou ainda sobredefinida) se $\mu_1+\upsilon_1 > 1$.

Definição 25 : O grau de *Inconsistência/Subdeterminação (I/S)* de uma cláusula evidencial é definido através de $\tau \rightarrow$ [-100, 100], onde:

$$I/S = |(\mu_1+\upsilon_1-1)| * 100$$

O cálculo do grau de I/S, permite mapear os fatores evidenciais favoráveis e contrários em apenas um único valor de inconsistência ou subdeterminação da informação analisada.

3.2. Considerações Finais

Nesta seção foram apresentados os principais conceitos e definições da *lógica evidencial paraconsistente*. Esta última fornece um modelo de raciocínio que opera mesmo com a presença de informações contraditórias (Ávila 96; Enembreck *et al.* 99), e permite também a presença de infinitos valores-verdade possíveis para uma premissa (cf. Figura 4). Além disso, é possível quantificar a inconsistência e avaliar a qualidade das informações envolvidas.

Foi apresentada a semântica operacional da *programação lógica evidencial* (Blair 87; Subrahmanian 87b), assim como o procedimento de fechamento [Blair 88] e o grau de inconsistência e subdeterminação.

A definição e implementação dos mecanismos de avaliação/interpretação das informações locais e não locais (trocadas pelos agentes) desenvolvidos a seguir neste trabalho apoiaram-se na *lógica evidencial paraconsistente*.

III - Sistema Pandora e Comunicação entre Agentes via JKQML

Pandora[7] é um sistema multi-agente. Cada agente é uma entidade cognitiva e implementa uma competência específica. A comunicação entre os agentes dá-se através de troca de mensagens. A estrutura mental é a mesma para todos os agentes. A implementação das competências dos agentes é em uma ou duas camadas, dependendo do grau de interação possível ou exigida entre os agentes. Por exemplo, a interação entre os agentes envolvidos no reconhecimento dos campos numérico e literal de um cheque, pode ser necessária para verificar se ambos os montantes são os mesmos ou para reforçar uma crença sobre uma informação.

Na seqüência, será apresentada a arquitetura do sistema, a arquitetura interna dos agentes, assim como a forma com que *lógica paraconsistente* foi aplicada na definição e implementação dos mecanismos que:

- interpretam/validam as informações produzidas localmente, assim como as informações não locais, provenientes de outros agentes;
- decidem quando um agente deve comunicar-se com os demais agente do sistema.

Para finalizar, será apresentado também como a comunicação entre os agentes foi implementada utilizando a linguagem KQML. Deve-se lembrar que a base de cheques utilizada nos testes é produto de uma geração simulada de dados a partir de informações estatísticas.

[7] Na mitologia grega, Pandora, foi a primeira mulher criada por Júpiter. Como presente de casamento, Pandora recebeu uma caixa, denominada *Caixa de Pandora*, onde cada deus colocara um bem. "Pandora abriu a caixa, inadvertidamente, e todos os bens escaparam, exceto a esperança. Assim, mesmo que tudo nos escape a esperança não nos deixa inteiramente."

4. Arquitetura Multi-Agente

No contexto deste trabalho, um sistema multi-agente configura-se pela reunião de um conjunto de agentes autônomos e altamente especializados. Eles cooperam para alcançar um objetivo comum, resolver conflitos, articular ações coordenativas ou simplesmente trocar informações para interpretar/validar uma situação. Neste sentido, a comunicação entre estes agentes é fundamental.

4.1. Multicheck

A arquitetura *Multicheck* consiste de um conjunto de agentes relativamente complexos voltados à análise e tratamento de imagens de cheques bancários brasileiros manuscritos [Scalabrin *et al.* 98]. Nesta arquitetura foram definidos três tipos de agentes:

- *agente de segmentação*, que identifica, extrai e cria um modelo lógico de um cheque (e.g., data, assinatura, montante literal e montante numérico).
- *agente de reconhecimento*, que reconhece os diferentes campos lógicos extraídos de um cheque.
- *agente de análise ou 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 de gerência ou gerenciador*, que monitora a evolução da rede de agentes, no intuito de balancear a carga do sistema.

A Figura 7 mostra uma visão simplificada da arquitetura do sistema *Multicheck*, bem como da arquitetura de cada um de seus agentes [Angelotti *et al.* 01a]. A capacidade em reconhecer padrões — sobre segmentos de imagens — está presente apenas nos agentes: *data, assinatura, numérico e literal*. A *expertise* para interpretar/validar os padrões aparece em todos os agentes, exceto no agente segmentação. A aceitação ou rejeição de um cheque é feita pelo *agente analisador*, que valida as informações fornecidas por todos os agentes de reconhecimento.

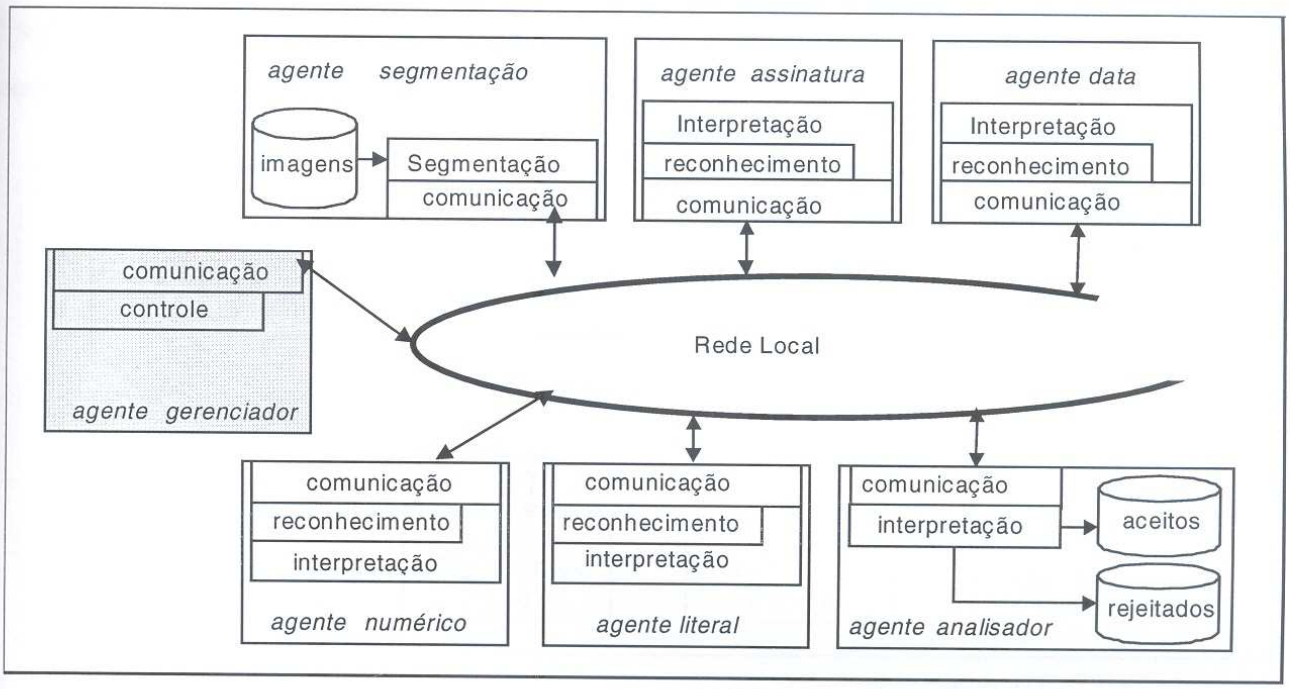

Figura 7: Arquitetura Multicheck.

A capacidade de comunicação está presente em todos os agentes e é implementada pelo *módulo comunicação*. Este último é responsável pela troca de mensagens assíncronas entre os agentes, e implementa algumas tarefas básicas, tais como: *receber/expedir uma performativa, extrair o conteúdo de uma mensagem e repassar este conteúdo aos módulos especializados*. Por exemplo, se uma mensagem contém informações sobre *graus de certeza* e *fatores evidencias*, estas informações devem ser repassadas ao módulo de interpretação. Se, por outro lado, uma mensagem contém a imagem de um campo lógico, esta imagem deve ser repassada ao módulo de reconhecimento.

É importante salientar, que na implementação dessa arquitetura, podem existir vários agentes implementando a mesma competência. Esta redundância permite almejar vários tratamentos em paralelo e o balanceamento da carga do sistema. No entanto, a arquitetura propõe um mínimo de seis competências/agentes (um(a) de cada tipo, exceto o agente gerenciador) para interpretar um cheque.

4.2. Balanceamento da carga

Para gerenciar o balanceamento da carga do sistema foi introduzido um *agente gerenciador* [Angelotti *et al.* 00c], cuja principal tarefa, quando necessário, é:

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

Estas ações/decisões são tomadas levando em conta o tempo médio gasto por um agente para concluir seus cálculos sobre uma dada tarefa de reconhecimento.

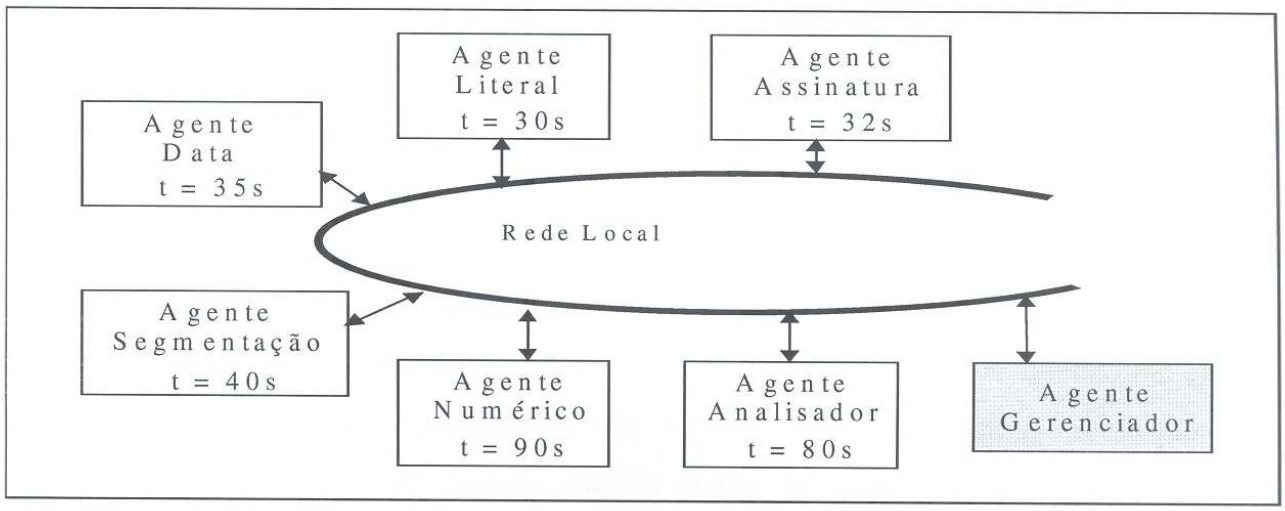

Figura 8: Inserção ou remoção de agentes no sistema Pandora.

O par ordenado $< i, t >$ corresponde a informação utilizada pelo *gerenciador* para suas tomadas de decisões, onde i é um agente qualquer e t é o tempo médio gasto pelo mesmo para concluir sua tarefa de reconhecimento. O conjunto *i/t* resume as informações da Figura 8.

$$i/t = \{< Assinatura, 32s >, < Data, 30s >, < Numérico, 80s >, < Literal, 90s >\}$$

Por exemplo, a decisão de *inserir ou retirar* um agente de reconhecimento é tomada pelo agente *gerenciador* baseando-se no valor β_i e na *regra 01* abaixo, ambos definidos a seguir:

T = {32s, 30s, 80s, 90s}

Para cada elemento de T faça:

$$\beta_i = \left(\frac{\frac{T_i}{N_i}}{Min(T)} \right) - 1$$

aplica *regra 01*

onde N_i é o número de agentes de um mesmo tipo, T_i é o tempo médio que um agente leva para realizar uma tarefa e *Min(T)* é o menor tempo para realizar uma tarefa.

De uma forma geral, o *gerenciador* toma suas decisões, quanto a inserção de agentes de reconhecimento, análise ou segmentação, assim como a remoção de um agente qualquer do sistema, avaliando as seguintes regras:

```
Regra 01: insere um novo agente de reconhecimento no sistema
   Se < (βi é maior que 0) >
   então < inserir βi agentes >
```

Regra 02: *insere um novo* agente analisador *ou* segmentação *no sistema*

Se < (números de cheques na fila *é maior que* 50) >

então < inserir um novo agente no sistema >

Regra 03: *retira um agente do sistema*

Se < $(\frac{T_i}{N_i} < Min(T))$ >

então <retirar o agente que leva mais tempo para realizar uma tarefa>

A principal vantagem da arquitetura *Multicheck* reside sobre os agentes independentes[8] ou cognitivos. Este agentes são entidades capazes de satisfazer seus objetivos, interagindo entre si e raciocinando sobre crenças [Castelfranchi 90], tornando o processo de interpretação de um cheque mais robusto, além de permitir que as etapas do tratamento possam se repetir (se necessário).

Por outro lado, o maior inconveniente é a complexidade de implementação desses agentes, em particular no que concerne a gestão e tratamento de suas comunicações. Por exemplo: *quando* e *como* um agente deve comunicar uma informação? *Quando* e *como* um agente deve solicitar uma informação? *Quando* e *como* os agentes devem organizar-se para atingir um objetivo comum? *Como* um agente deve tratar uma crença?

4.3. Interação entre agentes: um cenário

Os agentes *numérico* e *literal* representam o caso mais interessante deste trabalho, em particular, porque a interpretação dos campos lógicos *numérico* e *literal* pode ser efetuada de maneira interativa e aproximativa. Dessa forma, os agentes são levados a trocar crenças e raciocinar sobre as mesmas. A Figura 9, mostra sumariamente o funcionamento/interação destes dois agentes.

[8] Os agentes são independentes porque possuem objetivos compatíveis, recursos e competências suficientes.

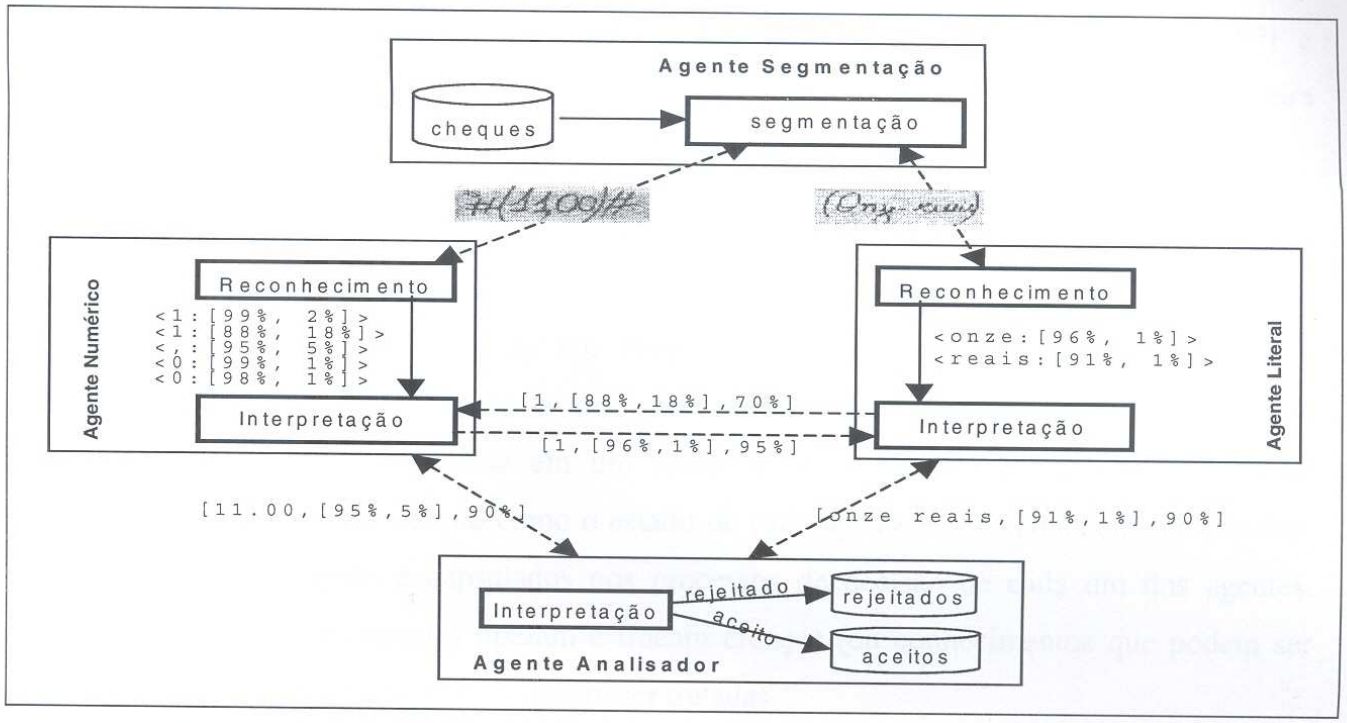

Figura 9: Segmentação, reconhecimento e validação dos campos lógicos numérico e literal.

Cada *processo de reconhecimento* corresponde aos diversos algoritmos de classificação aplicados sobre um determinado campo lógico. A entrada destes processos são imagens e as saídas são *tuplas* $<n,[\mu, \upsilon]>$, onde μ exprime a *evidência favorável* e υ a *evidência contrária*[9] de que *n* seja um determinado dígito no caso do *agente numérico* ou uma determinada palavra no caso do *agente literal*. Cada conjunto de padrões, obtido em um *processo de reconhecimento*, é a entrada para um *processo de interpretação/validação*. A interpretação de cada conjunto de padrões é realizada de maneira interativa, onde, por exemplo, os agentes *numérico* e *literal* trocam informações/crenças para resolver certos conflitos internos e chegar a um consenso sobre o valor do cheque. Estes agentes comunicam suas conclusões ao agente analisador, o qual aceita ou rejeita estas conclusões ou interpretações. Esta decisão é tomada considerando unicamente os valores *evidenciais favoráveis* e *contrários* sobre as informações fornecidos pelos agentes de reconhecimento dos campos lógicos. O resultado é obtido aplicando alguns operadores da lógica evidencial paraconsistente sobre esses valores, assim como o uso de algumas heurísticas do domínio.

É importante relembrar que este trabalho limita-se unicamente à validação ou interpretação dos padrões obtidos nos processos de reconhecimento, ou seja, somente a implementação dos módulos de interpretação de cada um dos agentes do sistema. Deve-se observar também, que os *valores evidencias* (crença e descrença) associados aos montantes *numérico* e *literal* foram obtidos utilizando um gerador de dados aleatórios. Os diversos módulos

de reconhecimento fazem parte dos seguintes trabalhos: assinatura [Santos *et al.* 97] e [Enembreck *et al.* 99], data [Morita *et al.* 99], montante numérico [Brito *et al.* 2000] e [Oliveira *et al.* 2000], montante literal [Freitas *et al.* 98].

4.4. Interpretação de Padrões

A interpretação das informações de um cheque é uma tarefa interativa, aproximativa e distribuída, ou seja, não limita-se à um processamento unicamente local. Cada agente implementa esta tarefa apoiando-se em um protocolo de comunicação de alto nível. Este protocolo entra em ação de acordo como o estado de cada agente e seus conhecimentos locais. Estes conhecimentos estão encapsulados nos processos de decisão de cada um dos agentes. Deve-se observar que os agentes operam e trocam crenças (ou conhecimentos que podem ser cancelados) e, como tal, essas crenças devem ser tratadas.

O tratamento de crenças — informações locais ou informações trocadas entre os agentes — baseia-se no raciocínio lógico *evidencial*. Neste tipo de raciocínio, descrito por Subrahmanian (1987a), dois valores são associados a uma proposição. Deve-se relembrar que na lógica *evidencial*:

- esses valores podem ser pensados como a evidência favorável à proposição p e o outro a evidência contrária ¬ p ([Ávila 96 e 97], [Prado 96]).
- nenhuma restrição é colocada nesses valores, exceto que pertençam ao intervalo [0,1].
- os fatores de evidência favorável e contrária não são diretamente relacionados como na teoria da probabilidade [Enembreck et al. 99].

Dessa forma, o funcionamento do tratamento de um campo lógico de um cheque, segue o seguinte fluxo: o módulo de reconhecimento de um dado agente recebe um segmento de imagem σ_i — que corresponde à um campo lógico de um determinado cheque e decompõe σ_i em vários partes σ_{ij}. Estas partes são classificadas por meio de classificadores altamente especializados. As saídas destes classificadores são no seguinte formato: $< \sigma_{ij} \in N_k : [\mu_j; \upsilon_j] >$, onde $\mu_j, \upsilon_j \in [0, 1]$, e representam respectivamente coeficientes de *evidência favorável* e *contrária* em relação a classe que pertence um dado σ_{ij}. N_k são as possíveis classes.

Considerando-se que σ_1 seja o campo lógico do *montante numérico* (segmentado dígito a dígito), σ_{1j} os valores de *evidência favorável* e *contrária* para cada dígito, e χ_{1j} os *graus de certeza* (cf. Figura 10).

---

[9] Conforme Subrahmanian (1987a), na lógica evidencial a utilização de duas evidências associadas à uma mesma proposição *p*, pode aumentar o poder expressivo da mesma.

σ_1	σ_{1j}	χ_{1j}
17,31	< 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

Figura 10: Segmento de imagem, graus de evidências favoráveis e contrárias, e graus de certeza.

Por exemplo, σ_{11} pode ser lido da seguinte maneira: existe uma *evidência favorável*, de no máximo 96%, que o primeiro dígito seja "1" e uma *evidência contrária*, de no máximo 1%, que este primeiro dígito não seja "1". É importante salientar que estes coeficientes são independentes.

A interpretação dos valores evidenciais é feita através de operadores e conceitos da lógica evidencial paraconsistente, onde as evidências são mapeadas na forma de *graus de certeza* por meio da seguinte função:

$$c([\mu_j, \upsilon_j]) = \mu_j - \upsilon_j = \chi_{ij}$$

onde μ_j representa a evidência favorável a uma proposição *p* e υ_j representa a evidência contrária à preposição *p*.

Tem-se, portanto, um *grau de certeza* χ_{ij} associado para cada segmento σ_{ij} classificado. χ_{ij} será utilizado em diversas situações, como por exemplo, para definir quando um agente deve comunicar-se com os demais agentes.

A decisão de *quando* um agente vai interagir e *com quem* é tomada baseada no grau de certeza que o agente possui sobre as informações do campo lógico que está sendo reconhecido. A seguir apresentaremos algumas regras utilizadas pelos agentes e sua relação com o grau de certeza.

4.4.1. Regras dos Agentes Numérico e Literal

As regras a seguir representam algumas das regras utilizadas pelos agentes para decidir quando enviar uma determinada informação, enviar um resultado, solicitar uma nova segmentação, etc.

Regra 04:

Se < $\chi_{ij} \in (50, 90]$ >

então < solicitar informações ao *agente literal ou numérico* para aumentar χ_{ij} >

Regra 05:

Se < min(χ_{ij}) $\in (90, 100]$ >

então < enviar o resultado ao *analisador* e demais interessados >

Regra 06:

Se < $\chi_{ij} \in [0, 50]$ >

então < solicitar uma nova segmentação σ_{i+1} >

Regra 07:

Se < solicitação de nova segmentação é rejeitada >

então < concluir que o montante numérico não pode ser reconhecido > *e*
< enviar este resultado a todos os demais agentes >

4.4.2. Regras do Agente Analisador

A regra a seguir representa, de forma bastante simplificada, o conhecimento utilizado pelo agente analisador para classificar um cheque como aceito ou rejeitado.

Regra 08:

Se < um dos campos lógicos não pôde ser interpretado corretamente >
então < rejeitar cheque >
senão < aceitar cheque >

Um agente busca interagir, em particular, quando ele não consegue reconhecer o campo lógico de sua competência. Ele pode, então, decidir recorrer a:

- *um agente de segmentação* para solicitar uma nova extração do campo lógico em questão;
- *um agente de reconhecimento* para tentar validar uma crença;
- *todos os agentes do sistema* para avisar que o campo lógico de sua competência não pôde ser reconhecido.

As informações trocadas entre os agentes podem resultar em novos coeficientes evidencias, em especial por meio de combinações sucessivas.

4.5. Combinação de Informações

As combinações sucessivas de informações podem ocorrer em dois momentos distintos:

- durante a segmentação local de um determinado campo lógico — combinação simples;
- durante a interpretação de dois ou mais campos lógicos que interagem entre si — combinação compartilhada.

As combinações de informações serão detalhadas nas subseções a seguir.

4.5.1. Fase I - Combinação Simples

> A *Fase I* consiste na combinação de diferentes segmentações e classificações sobre o mesmo campo lógico.

O *agente segmentação* identifica, extrai e cria a estrutura lógica de um cheque (data, assinatura, montante literal e montante numérico). Esta tarefa é realizada em um primeiro instante pela segmentação global do cheque, e em um segundo instante por uma segmentação local, *i.e.*, a extração de diferentes componentes, tais como: a assinatura, a data, etc. Isto permite que um dado agente solicite, explicitamente, ao *agente segmentação* uma nova extração de um determinado campo lógico. Os algoritmos de reconhecimento serão aplicados sobre esta nova extração, obtendo-se novos *valores evidenciais* e *graus de certeza*, que serão consequentemente combinados.

Na Figura 10, deve-se observar, que o terceiro, o quarto e o quinto componente de σ_1 foram reconhecidos com *graus de certeza* menores que 50%. Assim, aplicando-se a *Regra 06*, toma-se a decisão de solicitar uma nova segmentação.

Considerando-se que σ_2 é uma nova segmentação para o campo lógico *montante numérico*, σ_{2j} são valores de *evidência favorável* e *contrária* para cada dígito, e χ_{2j} os *graus de certeza* (cf. Figura 11).

σ_2	σ_{2j}	χ_{2j}
17,31	< 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

Figura 11: Segunda segmentação do montante numérico, graus de evidências favoráveis e contrários, e graus de certeza.

Cada valor σ_{1j}, da primeira segmentação (Figura 10), é comparado com cada valor σ_{2j}, da segunda segmentação (Figura 11). Se, por exemplo, σ_{11} e σ_{21} pertencerem a mesma classe, então aplica-se o operador *supremo* (*sup*) sobre χ_{11} e χ_{21}. A tupla σ_{ij} que possui o maior *grau de certeza* é selecionada. Dessa forma, para σ_{11} e σ_{21} seleciona-se < 1:[0.99 0.02], 97% >. O operador *supremo* é utilizado porque retorna o maior grau de certeza no processo de seleção. Porém, se σ_{11} e σ_{21} não pertencerem a mesma classe, então é necessário iniciar um processo de trocar informações entre os *agentes numérico* e *literal*, no intuito de descobrir qual classificação é a

mais correta. É importante notar, que mesmo quando, o *grau de certeza* de σ_{22} é maior que o *grau de certeza* de σ_{12}, σ_{12} será selecionado. Isto ocorre porque, na tomada de decisão, o *montante literal*[10] tem um peso/importância maior que o *montante numérico*. Neste caso, a combinação das atuais informações resulta nas tuplas de valores de *evidência favorável* e *contrário*, e, nos *graus de certeza*, mostrados na Figura 12.

	$(\sigma_{1j}, \sigma_{2j})$	(χ_{3j})
< 1 :	[0 . 9 9 ; 0 . 0 2] >	0 . 9 7
< 7 :	[0 . 8 6 ; 0 . 0 1] >	0 . 8 5
< , :	[0 . 9 0 ; 0 . 1 1] >	0 . 7 9
< 3 :	[0 . 8 0 ; 0 . 2 3] >	0 . 5 7
< 1 :	[0 . 9 9 ; 0 . 4 0] >	0 . 5 9

Figura 12: Segmento de imagem, graus de evidências favoráveis e contrários, graus de certeza.

Estas informações serão objeto de validação, rejeição ou combinação de acordo com os resultados obtidos, por exemplo, pelo *agente literal*.

4.5.2. Fase II - Combinação Compartilhada

> A *Fase II* consiste no compartilhamento/combinação de resultados parciais de diferentes campos lógicos.

O compartilhamento de resultados parciais é fundamental entre os *agentes literal* e *numérico*, em particular, porque eles devem obter exatamente a mesma informação sobre campos lógicos distintos (codificados em formatos diferentes). Eles podem também, claramente, obter resultados conflitantes e serem levados a interagir para obter uma interpretação consistente e aumentar seus graus de certezas [Angelotti *et al.* 00b].

Assumindo que, os *agentes literal* e *numérico* já concluíram independentemente a *Fase 1* e reconheceram a mesma informação (Figura 13), então o conseqüente da *Regra 04*, de ambos os agentes, pode ser avaliada.

[10] No caso da legislação brasileira, para cheques bancários, vale o montante que esta descrito por extenso.

Informações obtidas pelo *agente numérico* às segmentações: σ_1 e σ_2	$c([\mu_j, \upsilon_j])$
$(\sigma_{1j}, \sigma_{2j})$	χ_{3j}
< 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 literal* às segmentações: σ_1, σ_2 e σ_3	$c([\mu_j, \upsilon_j])$
$(\sigma_{1j}, \sigma_{2j}, \sigma_{3j})$	χ_{4j}
< onze : [0.89 ; 0.04] >	0.85
< reais[11] : [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

Figura 13: Graus de evidências favorável e contrária, e graus de certeza para os montantes numérico e literal.

Os mecanismos aqui utilizados (cf. [Blair 87], [Subrahmanian 87a], [Prado 96]), para avaliar a qualidade das informações que os agentes possuem, são:

- *a disjunção*, que permite combinar valores para aumentar um determinado grau de crença;
- *a conjunção*, que permite avaliar um conjunto de valores (σ_{ij} e χ_{ij}) sobre um dado campo lógico e gerar um único valor para σ_i e χ_i;
- *o grau de certeza*, que permite estudar individualmente cada parte segmentada (σ_{ij}) de um montante;
- *o grau de inconsistência/subdeterminação*, que permite mapear, em apenas um único valor, a inconsistência ou subdeterminação da informação analisada.

4.6. Operadores

4.6.1. Disjunção

Aplica-se o operador de disjunção ($\vee$), definido em [Prado 96], quando um agente necessita confirmar uma hipótese ou aumentar suas crenças sobre um determinado componente.

$$[\mu_1, \upsilon_1] \vee [\mu_2, \upsilon_2] = [\max(\mu_1, \mu_2), \min(\upsilon_1, \upsilon_2)]$$

onde, as anotações evidenciais são: $[\mu_1, \mu_2], [\upsilon_1, \upsilon_2] \in [0,1]$.

[11] Unidade da moeda brasileira.

No exemplo da Figura 13, os *graus de certeza* que precisam ser aumentados são os três últimos valores do campo numérico, porque são menores que os graus de certeza obtidos sobre o campo literal correspondente. Assim, o *agente numérico* aplica o *operador disjunção* sobre as informações calculadas localmente e as informações recebidas do *agente literal*, obtendo-se desta forma as seguintes expressões:

$[0.90\ 0.11] \vee [0.89\ 0.04] \vee [0.90\ 0.04] \vee [0.93\ 0.06] \vee [0.91\ 0.04] \vee [0.88\ 0.06] = [0.93\ 0.04]$
$[0.80\ 0.23] \vee [0.89\ 0.04] \vee [0.90\ 0.04] \vee [0.93\ 0.06] \vee [0.91\ 0.04] \vee [0.88\ 0.06] = [0.93\ 0.04]$
$[0.99\ 0.40] \vee [0.89\ 0.04] \vee [0.90\ 0.04] \vee [0.93\ 0.06] \vee [0.91\ 0.04] \vee [0.88\ 0.06] = [0.99\ 0.04]$

Informações obtidas após a aplicação do *operador disjunção* sobre as informações locais do *agente numérico* e as informações recebidas do *agente literal*.	c([μ, υ])
	χ_{4i}
< 1 : [0.99 ; 0.02]>	0.97
< 7 : [0.96 ; 0.01]>	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 literal* às segmentações: σ_1, σ_2 e σ_3	c([μ, υ])
$(\sigma_{1i}, \sigma_{2i}, \sigma_{3i})$	χ_{4i}
< onze : [0.89 ; 0.04] >	0.85
< reais[12] : [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

Figura 14: Graus de evidências favoráveis e contrárias, e graus de certeza após a aplicação do operador de disjunção.

4.6.2. Conjunção

Aplica-se o *operador de conjunção* (∧), definido em [Prado 96], quando um agente necessita obter um valor de fechamento sobre cada um dos montantes.

$$[\mu_1, \upsilon_1] \wedge [\mu_2, \upsilon_2] = [\min(\mu_1, \mu_2), \max(\upsilon_1, \upsilon_2)]$$

onde, as anotações evidenciais são: $[\mu_1, \mu_2], [\upsilon_1, \upsilon_2] \in [0,1]$.

O *operador de conjunção* permite a partir de vários valores de σ_{ij} e χ_{ij}, gerar um único valor para σ_i e χ_i. Em outras palavras, é possível associar um único valor de evidência à descrição do mundo combinando as várias *anotações evidenciais* dos átomos através do operador ∧. Isto permite descobrir inconsistências, porque seleciona-se o mínimo entre os *valores evidenciais favoráveis* e o máximo entre os *valores evidencias contrários*. Por exemplo, aplicando o operador ∧ sobre as informações locais dos *agentes numérico e literal*, obtém-se respectivamente: <17,31 : [0.93 0.04]> e <dezessete reais e trinta e um centavos : [0.88 0.03]>.

Agente numérico:

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

[12] Unidade da moeda brasileira.

Agente literal:

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

Essas informações serão enviadas ao *agente analisador* que interpretará os *fatores evidenciais* obtidos sobre cada um dos montantes.

4.6.3. Grau de Inconsistência/Subdeterminação (I/S)

O cálculo do *grau de I/S*, definido em [Subrahmanian 87a], permite mapear os *fatores evidenciais favoráveis* e *contrários* em apenas um único valor a inconsistência ou subdeterminação da informação analisada.

$$I/S = |\mu_1 + \upsilon_1 - 1| * 100$$

Estes cálculos são efetuados — pelo *agente analisador* — em duas etapas:

- aplicação do operador $\wedge$ sobre as informações recebidas pelos agentes de reconhecimento, onde obtém-se para o caso em questão: $[0.93\ \ 0.04] \wedge [0.88\ \ 0.06] = [0.88\ \ 0.06]$
- cálculo do I/S é: $|0.88 + 0.06 - 1| * 100 = 6\%$

Isto significa que as informações obtidas — sobre um dado cheque — possuem 6% de I/S. Pode-se então decidir, a partir desta informação, se o cheque será aceito ou rejeitado. Para isto, foi definido a *Regra 06*.

```
Regra 09:

   Se < I/S ∈ [0%, 5%] >
   então < aceitar cheque >
   senão < rejeitar cheque >
```

Na arquitetura proposta, o *grau de inconsistência* é empregado para analisar a qualidade de uma informação. Uma proposição com alto grau de *sobredeterminação* ou de *subdeterminação* é pouco confiável e o sistema deve, portanto, ser capaz de reconhecer tais situações.

4.7. Definição de Limiares

O *grau de certeza* define um ponto no reticulado da Figura 4 da Seção 3, que é composto por dois valores: *valor de grau de crença* e um *valor de grau de descrença* (μ_1 - ν_1). Em Newton da Costa (1999), é apresentado um reticulado no plano cartesiano com um total de 12 regiões, como mostra a Figura 15. O aumento do número de regiões delimitadas, permite maior resolução, trazendo resultado mais precisos.

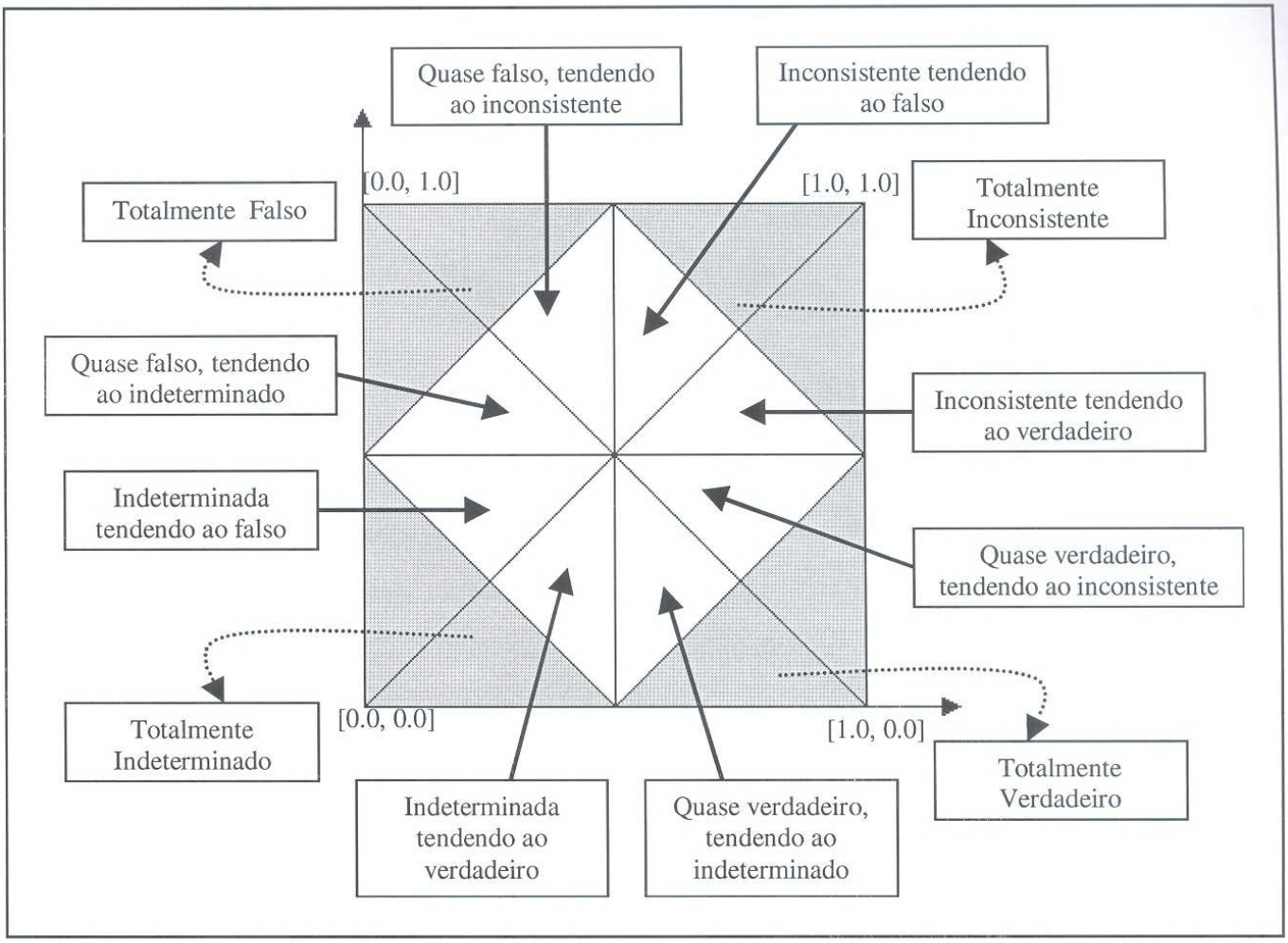

Figura 15: Reticulado no plano cartesiano com um total de 12 regiões.

Conforme o *reticulado representativo* da lógica evidencial paraconsistente, os pontos extremos do reticulado têm os valores: *falso, verdadeiro, inconsistente* e *indeterminado*. Conforme podemos observar, quando o grau de crença e descrença são altos — valores próximos ou iguais a 1 — está configurado um estado de inconsistência. Por outro lado, quando o grau de crença e descrença são baixo — valores próximos ou iguais a 0 — o estado é de indeterminação. Ainda segundo a lógica evidencial paraconsistente, o resultado é falso quando o *grau de crença* é baixo e o *grau de descrença* é alto simultaneamente. O resultado será verdadeiro quando o *grau crença* é alto e o *grau de descrença* é baixo.

Quando os valores do *grau de crença* e *descrença* são iguais, isso resultará em valores pertencentes a uma linha perfeitamente indefinida (cf. Figura 16), expressa por: $\mu_1 - \upsilon_1 = 0$.

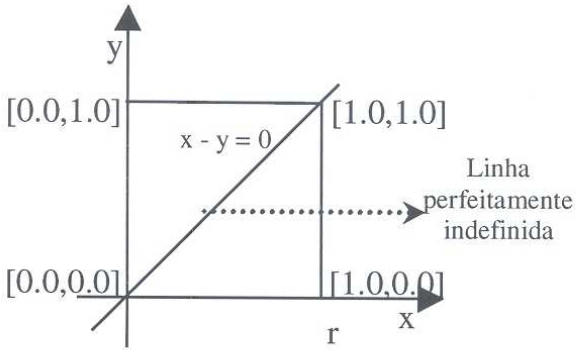

Figura 16: Reticulado no plano cartesiano com linha perfeitamente indefinida.

Estamos interessados no caso em que o resultado é *totalmente verdadeiro*. Para isso, como mostra a Figura 15, os valores de crença devem ser altos e os de descrença devem ser baixos. Isso ocorrerá quando o *grau de certeza* for maior ou igual (≥) a 0.50. Como podemos observar, quanto maior o *grau de crença* e quanto menor o *grau de descrença* simultaneamente, mais perto estaremos de encontrar o *valor verdadeiro* ([1.0, 0.0]).

Por este motivo, estabelecemos que quando o *grau de certeza* for menor (<) 0.50, será necessário solicitar uma nova segmentação, pois o valor obtido não pertence a região de *totalmente verdadeira* do reticulado da Figura 15. Da mesma forma, quando o valor do *grau de certeza* estiver entre 0.50 e 0.90 ($0.50 \geq$ *grau de certeza* < 0.90), perto do valor *verdadeiro*, ainda não fornece muita certeza, é necessário então que os agentes troquem informações a fim de confirmar uma hipótese ou aumentar o valor referente a um dado grau de certeza. Quando o valor do grau de certeza for maior ou igual (≥) a 0.90 não será necessário tomar qualquer atitude em relação a esse valor, pois ele já está bastante próximo do valor de *verdadeiro*. Estas diversas considerações foram expressas anteriormente através das regas: *Regra 04, Regra 05,* etc.

4.8. Considerações Finais

A implementação de uma arquitetura multi-agente para o tratamento de cheques bancários brasileiros manuscritos parece permitir, que os agentes alcancem seus objetivos interagindo entre si e raciocinando sobre suas crenças, o que torna o processo de interpretação de um cheque mais robusto, à medida que é reduzido o número de cheques rejeitados indevidamente.

A presença de informações inconsistentes é freqüente na interação entre os agentes *literal* e *numérico*, porque eles devem reconhecer a mesma informação, codificada em formatos diferentes. A *lógica paraconsistente* foi utilizada no contexto dessa aplicação para levar em conta, de forma natural, as inconsistências entre informações — sem excluí-las —, e acima de

tudo, permitir a implementação de mecanismos relativamente simples no mundo dos agentes cognitivos.

5. Comunicação entre agentes via JKQML

É importante salientar, que os diversos raciocínios apresentados anteriormente, são efetuados localmente no interior de cada agente. Isto implica que os agentes devem ser dotados de mecanismos de comunicação para permitir a troca de informações. Sumariamente, estes mecanismos incluem três fases distintas:

- o estabelecimento de conexão entre os agentes;
- a solicitação e comunicação de determinadas informações;
- o encerramento de conexão.

A comunicação em um sistema multi-agente é fundamental e requer uma linguagem de comunicação comum entre eles, em particular, para codificar as suas intenções durante um diálogo. Para esse fim, adotou-se a linguagem KQML [Finin 93, 96 e 97], onde cada mensagem representa, intuitivamente, uma parte do diálogo entre dois ou mais agentes.

Nesta implementação, a cooperação começa pelo estabelecimento de conexões entre os agentes detentores de tarefas a serem executas e os demais agentes competentes a executá-las [Angelotti *et al.* 00a]. Por exemplo, o *agente segmentação az1* recebe um cheque, segmenta-o e repassa-o ao *agente analisador aa1*, que possui a competência requerida (analisar cheque). A conexão entre estes agentes é feita por meio da performativa *recruit-one* (Figura 17).

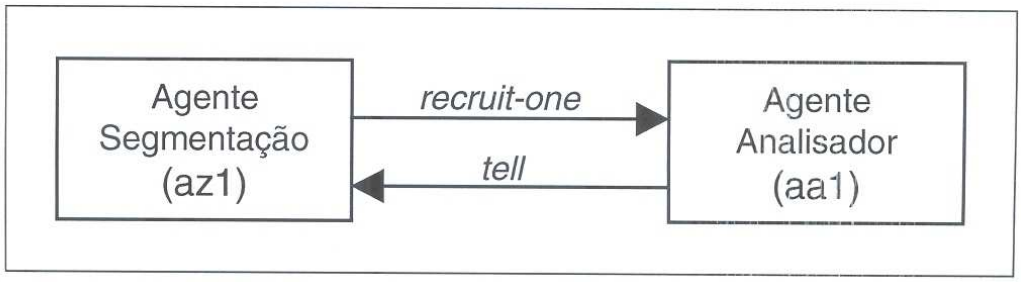

Figura 17: Conexão entre os agentes: Segmentação e Analisador.

O agente *aa1*, emissor do *tell*, assume a responsabilidade em analisar o cheque em questão. Esta tarefa de análise será compartilhada com os demais agentes do sistema. Para isto, um processo de recrutamento de competências específicas — verificar assinatura, verificar data, reconhecer montante literal, reconhecer montante numérico — é executado pelo agente *aa1*. Este processo cria várias outras conexões entre *aa1* e os agentes de reconhecimento.

Como mostra a Figura 18, cada emissor de um *tell* assume a responsabilidade de tratar o campo lógico de sua competência. Neste processo, os agentes começam trabalhando de maneira

individual e à medida que alguns resultados parciais começam a ser obtidos, eles passam a compartilhá-los. Isto ocorre, em particular, entre os agentes que assumem a responsabilidade de interpretar os *montantes numérico* e *literal*.

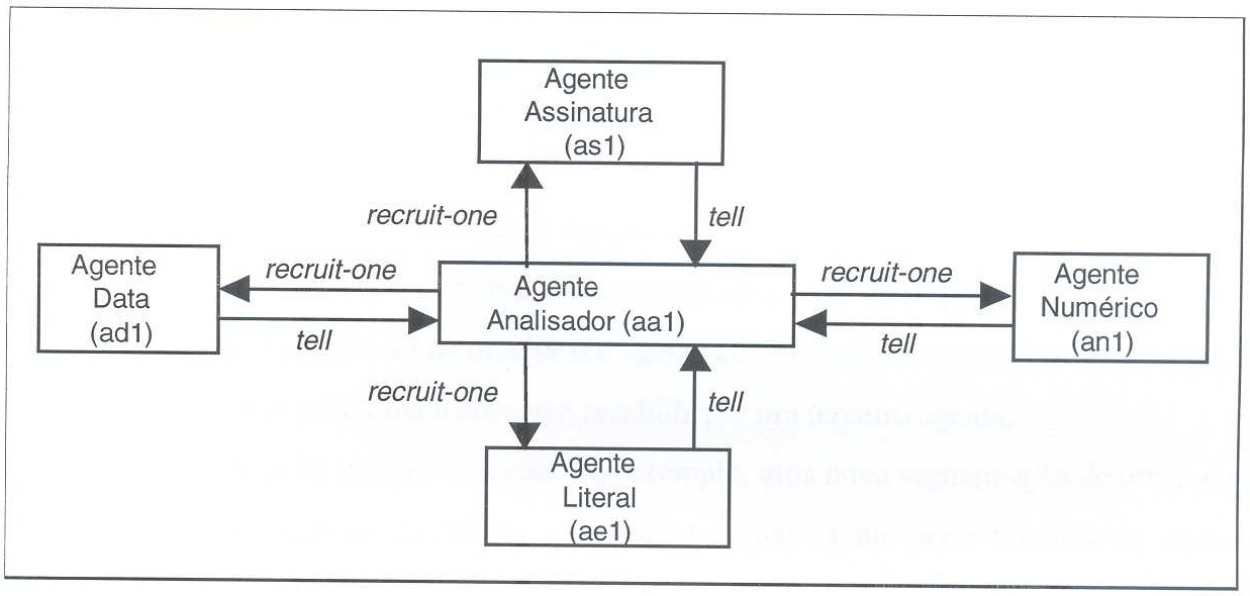

Figura 18: Conexões para a distribuição dos campos lógicos a tratar.

O encerramento destas diversas conexões é efetuado unicamente após a finalização dos cálculos realizados pelos agentes de reconhecimento, e a comunicação dos respectivos resultados ao agente analisador *aa1*. Este último decide (baseado nos resultados recebidos) se o cheque em questão será classificado como *aceito* ou *rejeitado*. É importante salientar que os agentes analisadores são *monotarefa*.

5.1. Performativas Utilizadas no Sistema Pandora

A linguagem KQML possui um grande conjunto de mensagens pré-definidas (performativas), as quais definem as operações possíveis de serem executadas pelos agentes. Dessa forma, uma performativa pode ser uma declaração, uma consulta, um comando ou um conjunto de performativas conhecidas.

O protótipo *Pandora* foi implementado em Java™, utilizando uma classe do próprio Java™ denominada JKQML (**J**ava **K**nowledge **Q**uery and **M**anipulation **L**anguage) que implementa KQML.

JKQML suporta 23 das 36 performativas propostas na especificação original de KQML. Além disso, o JKQML possui um agente denominado *facilitador,* que é responsável por estabelecer as conexões e fazer intermediações entre todos os agentes do sistema. Todos os

agentes conhecem o endereço físico do *facilitador*, mas podem comunicar-se uns com os outros utilizando apenas nomes simbólicos.

As principais mensagens utilizadas na implementação das interações entre dois ou mais agentes são:

- *register*: registra um agente no facilitador;
- *advertise*: avisa o facilitador de que um dado agente é o responsável por uma dada tarefa;
- *tell*: informa um dado a um outro agente;
- *reply*: responde a uma performativa, caso esta necessite de resposta;
- *recruit-one*: solicita ao facilitador um agente competente em uma determinada tarefa;
- *forward*: reexpede uma mensagem recebida por um terceiro agente;
- *ask-one*: solicita uma informação. Por exemplo, uma nova segmentação de um cheque;
- *broke-one*: pede ao facilitador que encontre qual agente pode fornecer os dados já computados sobre uma determinada tarefa;
- *broadcast*: envia uma mensagem a todos os agentes presentes no sistema.

5.1.1. Cenário utilizando as performativas básicas

Como dito anteriormente é o *facilitador* que intermedia todas as trocas de mensagens entre os agentes. Dessa forma, é necessário que todos os agentes registrem seu endereço físico, logo que o programa inicia, junto ao *facilitador*. Além de registrarem-se junto ao *facilitador,* os agentes também devem informar quais são as suas habilidades, *i.e.,* que tipo de tarefa eles realizam utilizando a performativa *advertise* (cf. Figura 19).

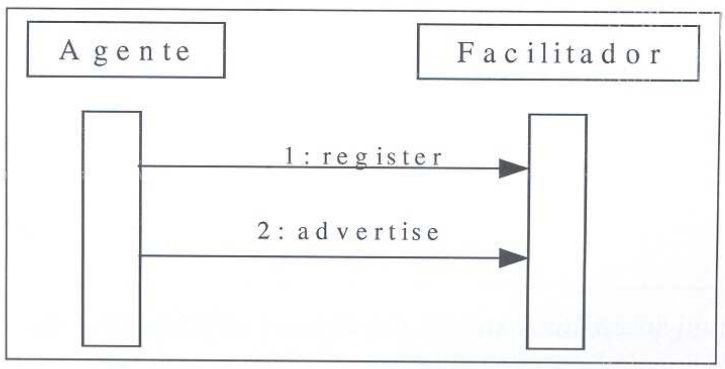

Figura 19: Agente registrando-se no facilitador e informando suas habilidades.

A Figura 20 mostra uma "janela" do sistema *Pandora*, que representa o agente data. Nesta "janela" estão presentes as seguintes informações: o nome do agente e o seu número de registro, o nome da performativa comunicada, o agente que recebeu a mensagem, a tarefa que está sendo

executada naquele instante, um histórico das tarefas que agente executou durante a execução do programa, etc.

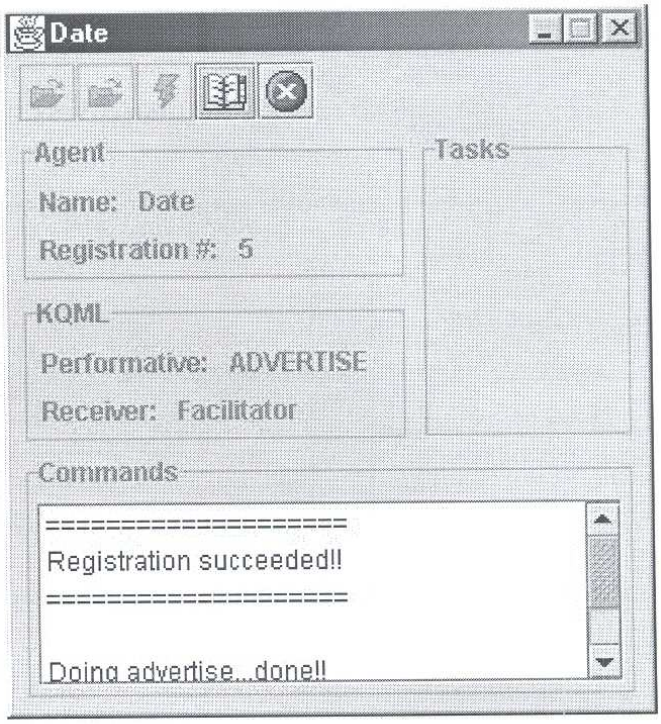

Figura 20: O Agente Data informa ao facilitador suas habilidades através da performativa advertise.

Dessa forma, sempre que um agente necessita comunicar-se com um outro agente, será o *facilitador* quem encontrará o agente solicitado.

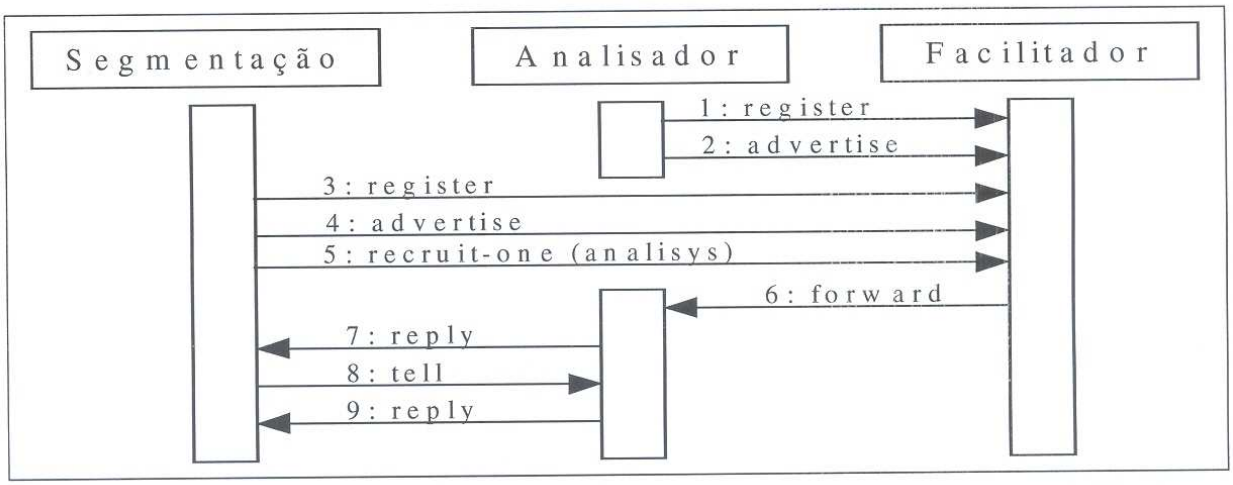

Figura 21: O Agente Segmentação recruta um Agente Analisador junto ao facilitador.

Quando os agentes terminam seu processamento, eles enviam todas as informações que obtiveram ao *agente analisador,* que é o responsável por decidir se o cheque vai ser aceito ou não. No cenário abaixo, o *agente analisador* recruta um agente capaz de interpretar o montante *numérico* de um cheque. Quando o *agente numérico* recebe o *forward* do *agente facilitador*, ele

envia um *reply* informando se ele está disponível ou não para trabalhar sobre aquele cheque. Se a resposta for "*disponível*" então o *agente analisador* envia os dados a serem processados usando um *tell*. Quando o *agente numérico* termina o seu processamento, ele devolve um *reply* com os dados computados para o *agente analisador*. É importante salientar que esta seqüência de mensagens sempre ocorrerá entre todos os agentes do sistema.

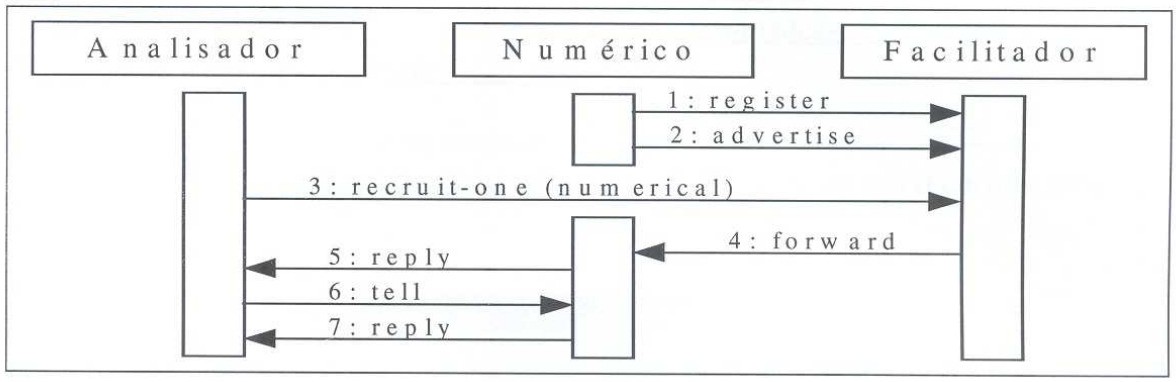

Figura 22: O Agente Analisador recruta um agente capaz de interpretar o montante numérico de um cheque.

Relembrando, o *agente segmentação* é quem possui todas as informações sobre um cheque. Este agente recupera um cheque de uma base, segmenta-o e repassa ao *agente analisador* todos os campos lógicos por ele segmentados. O *analisador* torna-se assim o responsável pelo recrutamento dos demais agentes capazes de interpretar os campos lógicos deste novo cheque.

Como é o *agente analisador* que recruta todos os agentes de reconhecimento, ocorre a criação dinâmica de canais de comunicação entre o agente analisador e os agentes de reconhecimento. Da mesma forma, uma conexão especial entre o *agente segmentação* e *analisador* é estabelecida. Assim, quando um agente solicita uma nova segmentação, essa solicitação é feita ao *agente analisador* — utilizando a performativa *ask-one* — e este repassa a solicitação ao *agente segmentação*, como mostra a Figura 23.

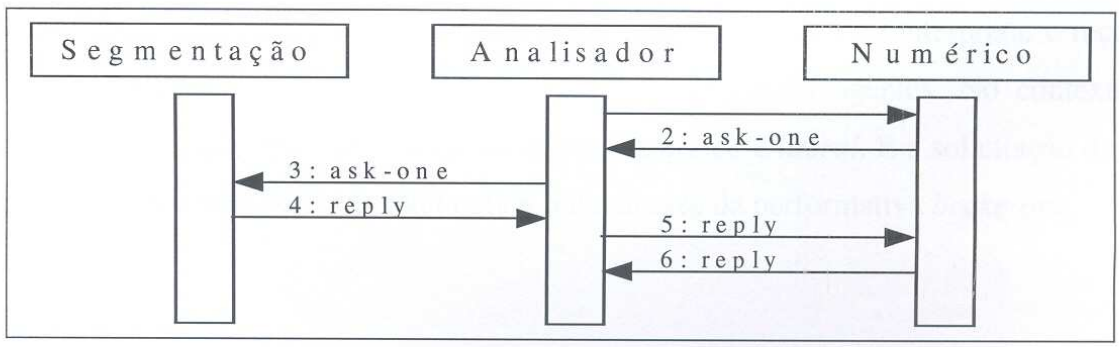

Figura 23: O Agente Numérico solicita uma nova segmentação.

Durante o processamento de um campo lógico, se um determinado agente não consegue reconhecer aquele campo, então uma mensagem é enviada a todos os agentes do sistema, através

da performativa *broadcast*, avisando que o cheque será rejeitado e, portanto, todos os agentes que estão processando este cheque podem abandonar suas tarefas (cf. Figura 24).

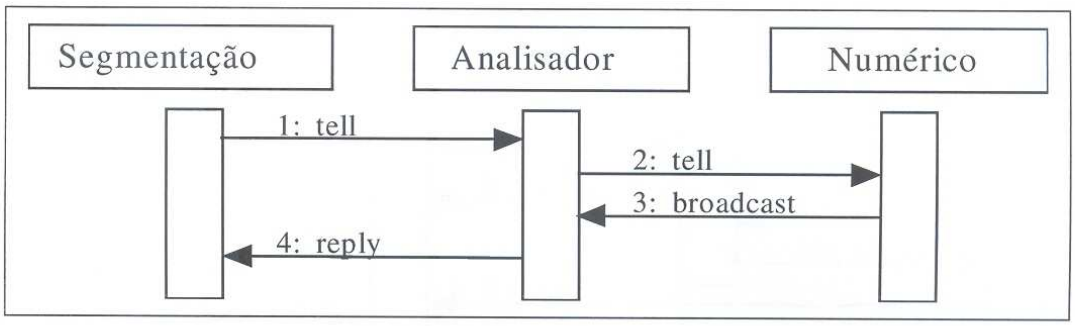

Figura 24: O Agente Numérico avisa aos outros agentes do sistema que o cheque será rejeitado.

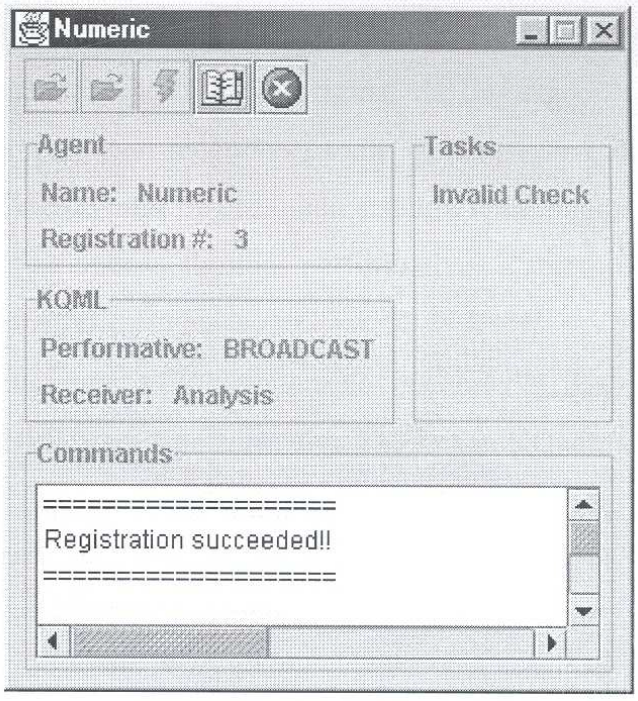

Figura 25: O Agente Numérico envia uma performativa broadcast para avisar que o cheque é inválido.

Finalmente, quando um agente não possui informações suficientes para interpretar e reconhecer um determinado campo lógico, ele pode solicitá-las aos outros agentes. No contexto deste trabalho, isto ocorre principalmente entre os *agente numérico* e *literal*. E a solicitação dos dados já computados sobre uma determinada tarefa é feita através da performativa *broke-one*.

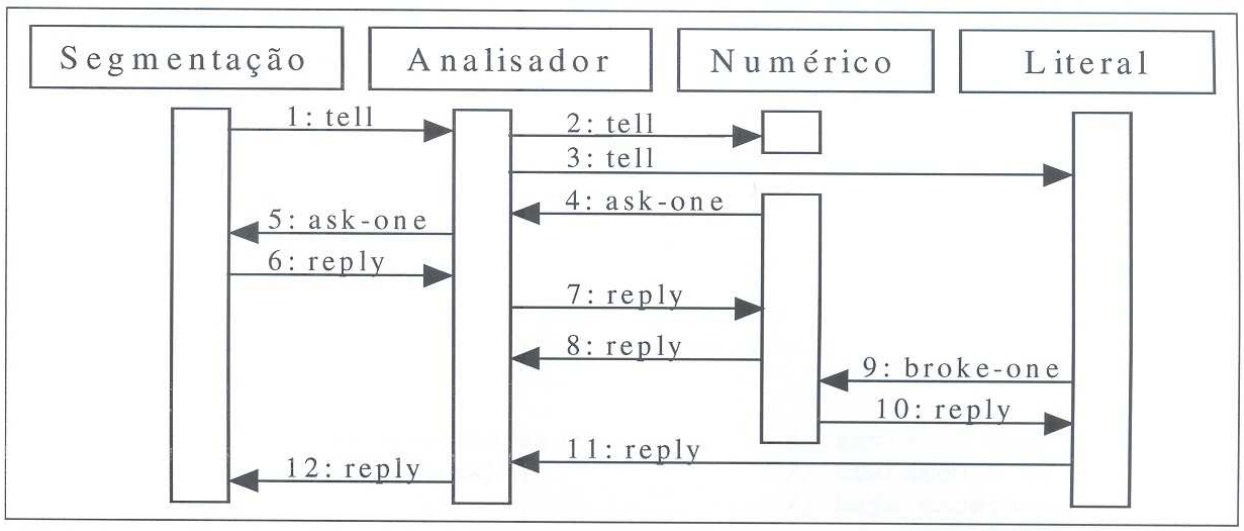

Figura 26: O Agente Literal solicita os dados já computados pelo agente numérico.

O conjunto de mensagens KQML pode ser estendido desde que as novas performativas criadas obedeçam a mesma forma da especificação original da linguagem. Os agentes que estão de acordo com a KQML não precisam reconhecer todas as mensagens, de forma que um pequeno subconjunto pode ser suficiente num determinado sistema multi-agente, ou seja, dependendo da necessidade pode-se escolher apenas algumas performativas a serem utilizadas na comunicação — como fizemos nesse trabalho.

A seguir apresentaremos a implementação em Java™ das mensagens que usam as funções da biblioteca JKQML para a performativa *TELL*. Em KQML a mensagem seria formada da seguinte maneira:

```
(tell
    :sender A
    :receiver B
    :reply-with id0
    :language multicheck
    :ontology multicheck
    :content cont)
```

Em Java a mesma mensagem para a performativa *tell* é escrita da seguinte forma:

```
KQML message = new KQML();          // cria um novo objeto da classe KQML que
                                    //transportará a mensagem entre os agentes.
try {
     message.setPerformative( KQML.TELL );      // define qual a performativa
                                                // a ser transmitida

     message.setSender( agent.agentName );      // define o nome do remetente
     message.setReceiver( "analysisAgent" );    // define o nome do
                                                // destinatário

     message.setRW( replyWith );                // define argumento para
                                                // resposta

     message.setLanguage( "multicheck" );       // define a linguagem
```

```
     message.setOntology( "multicheck" );        // define a ontologia
     message.setContent( cont );                 // define o conteúdo,
                                                 // contido em cont

      } catch (NoSuchParameterException e) {     // tratamento de erro
        System.out.println("erro");               // caso haja exceção por
      }                                          // falta de parâmetro


Conversation conv = null;          // cria o objeto que transporta a mensagem
                                   //contida em message  para o outro agente.

try {
      conv = km.sendMessage( message )          // envia  a mensagem
    } catch (JKQMLException ex) {               // tratamento de erro. Caso
                                                // haja exceção...

    System.err.println(ex.getMessage());        // imprime na tela o erro
                                                // ocorrido

    System.err.println("tell failed, exit.");   // imprime tell failed na
                                                // tela
    System.exit(1);                             // sai do programa
    }
```

Caso haja resposta para a performativa anterior, a seguinte performativa é enviada:

```
// em KQML
(reply
     :sender B
     :receiver  A
     :in-reply-to id0
     :reply-with id1
     :language multicheck
     :ontology multicheck :content content2)
```

```
// em JKQML
ResponseSet resp = new ResponseSet();      // cria um novo objeto da classe
ResponseSet                                // responsável pelo transporte da
                                           // mensagem de resposta a outro
                                           // agente
KQML message_reply = new KQML();           // cria um novo objeto da classe
                                           // KQML que transportará a mensagem
                                           // entre os agentes.

Try {
     message_reply.setPerformative(KQML.REPLY);     // define qual a
                                                    //performativa a ser
                                                    // transmitida
     message_reply.setSender(msg.getReceiver());    // define o remetente
     message_reply.setReceiver(msg.getSender());    // define o destinatário



     message_reply.setIRT(msg.getRW());         // define que esta mensagem é
                                                // uma resposta a outra
                                                // mensagem recebida
                                                // anteriormente

     message_reply.setRW(km.getInitialID());    // define argumento caso haja
```

```
                                                    // resposta

    message_reply.setLanguage("multicheck");        // define linguagem
    message_reply.setOntology("multicheck");        // define ontologia
    message_reply.setContent(data);                 // o conteúdo da resposta está
                                                    // em data

    resp.addElement(message_reply);                 // adiciona a resposta ao
                                                    // objeto resp
    return resp;                                    // retorna o objeto

    } catch (NoSuchParameterException e) {          // tratamento de erro caso
                                                    // haja exceção por falta
      System.out.println("erro");                   // de parâmetro
    }
```

5.2. Considerações finais

O processo de comunicação entre agentes é essencial para a cooperação na busca de uma solução para um problema. Este processo é guiado por protocolos de alto nível. Estes recursos permitem aos agentes trocarem informações entre si de forma inteligente.

O uso da linguagem de comunicação KQML possibilita que o processo de interação entre os agentes seja um processo intuitivo, uma vez que esta linguagem foi projetada para prover uma comunicação entre agentes de alto nível.

IV - Resultados

Os testes realizados, para demonstrar o interesse em particular da interação entre os agentes do sistema, foram realizados sobre três diferentes versões do sistema e analisados sobre uma base de 1.600 cheques para cada versão.

Versão I

Esta primeira versão corresponde a um processamento de cheques simplificado, em particular porque não existe nenhuma interação entre os agentes. Isto quer dizer que os agentes executam suas tarefas independentemente uns dos outros, sem trocar qualquer informação.

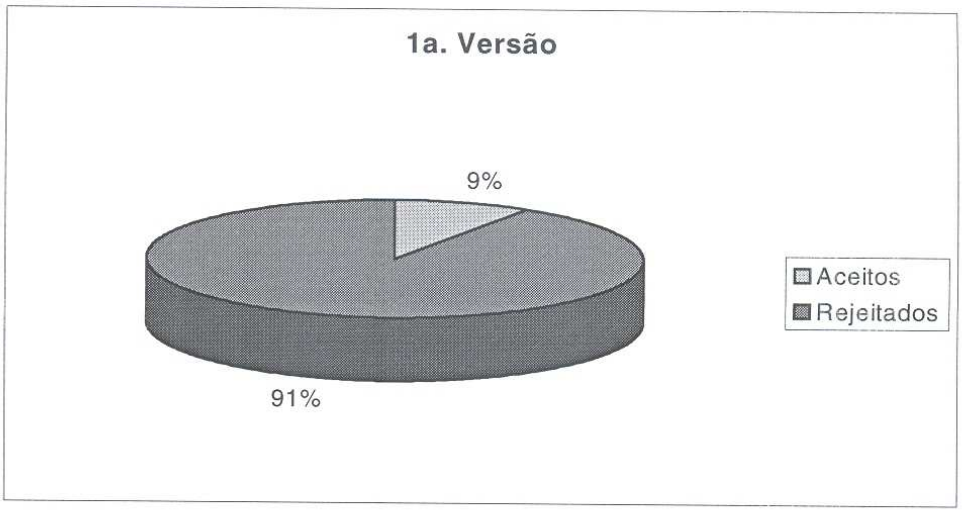

Figura 27: Resultados obtidos na implementação da primeira versão do sistema.

O gráfico da Figura 27 mostra claramente um índice elevadíssimo de cheques rejeitados.

Versão II

Na segunda versão do *Pandora*, os *agentes de reconhecimento* interagem com o *agente segmentação* durante a análise dos cheques, por exemplo, solicitando uma nova segmentação. Dessa forma, se um agente não consegue, por algum motivo, reconhecer o campo lógico de sua competência, ele pode pedir uma nova segmentação do mesmo; esperando que a nova imagem seja de melhor qualidade.

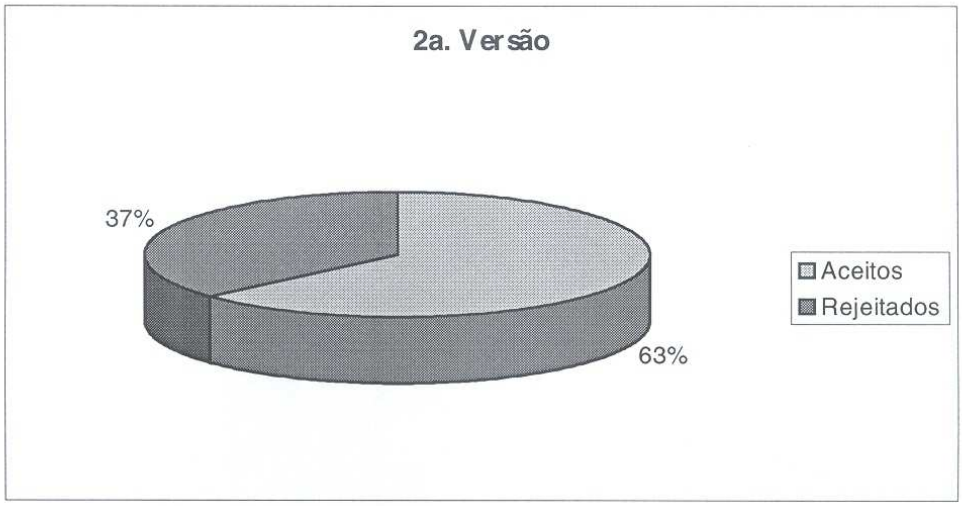

Figura 28: Resultados obtidos na implementação da segunda versão do sistema.

O gráfico da Figura 28 mostra, claramente, que a interação entre os agentes de reconhecimento e o agente de segmentação resulta em um melhor índice de cheques aceitos.

Versão III

Finalmente, a última versão do *Pandora,* representa o caso onde todos os agentes podem interagir. Nesta fase foi implementada a arquitetura proposta na Seção 4.1.

Pode-se perceber que o ganho desta última versão em relação à segunda versão é de 7%. No entanto, é este ganho que diferencia e que torna este sistema mais robusto.

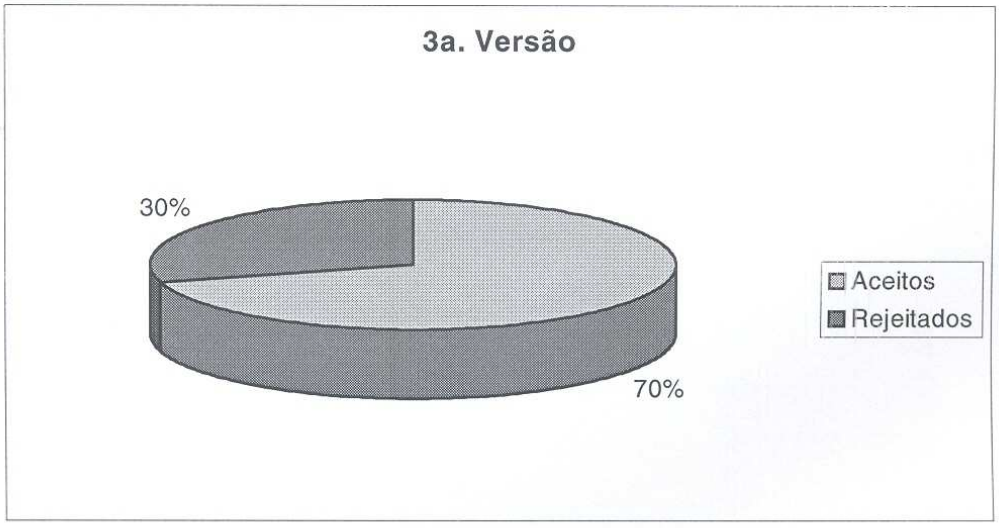

Figura 29: Resultados obtidos na implementação da terceira versão do sistema.

O gráfico da Figura 29 mostra uma ligeira melhora no índice de cheques aceitos.

O gráfico da Figura 30 mostra de forma comparativa os resultados obtidos nas três versões do sistema e foi apresentado em Angelotti *et al.* (2001b). Uma primeira conclusão é que a interação entre os agentes torna o processo de tratamento mais robusto, à medida que as trocas

entre os agentes podem resolver situações aparentemente difíceis, ou mesmo impossíveis de solucionar com um único especialista (como a validação do montante literal e do montante numérico), e reduzindo de forma considerável o número de cheques rejeitados.

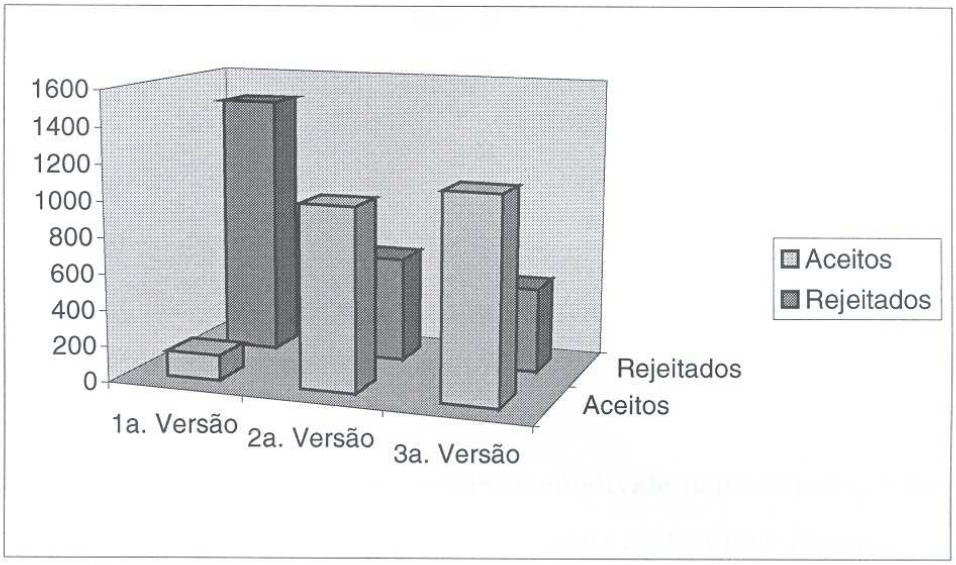

Figura 30: Gráfico que compara as três versões implementadas.

Do ponto de vista pragmático, a melhoria dos resultados vem da possibilidade, em particular, dos agentes poderem solicitar novas segmentações de um campo lógico. De fato, na área de processamento de imagens, existe uma grande gama de algoritmos projetados para realizar a segmentação de uma imagem, cuja eficiência depende das características da imagem. No caso de um cheque, o fundo artístico representa bem a dimensão do problema. Do ponto de vista de projeto, o agente segmentação implementa diferentes algoritmos/métodos de segmentação, o que possibilita resegmentar um campo lógico de um dado cheque, utilizando um algoritmo diferente.

V – Considerações Finais e Trabalhos Futuros

Primeiramente, relembremos que este trabalho insere-se no escopo do *Projeto Multicheck*. Este projeto define uma arquitetura de agentes autônomos para o tratamento inteligente e automático de cheques bancários brasileiros manuscritos. E foi adotado o paradigma agente baseando-se em uma tripla hipótese:

- *primeira*, trata-se de um problema que se subdivide naturalmente, à medida que cada tarefa pode ser modelada como sendo um agente autônomo e fracamente acoplado;
- *segunda*, a interação entre estes agentes pode tornar o processo de tratamento mais robusto, à medida que as trocas entre os agentes podem resolver situações aparentemente difíceis ou mesmo impossíveis de solucionar com um único especialista;
- *terceira*, o paralelismo natural entre os agentes pode contribuir de forma significativa para implementar uma aplicação de alto desempenho/performance.

No contexto da primeira hipótese: mostrou-se, empiricamente e experimentalmente, que o modelo agente aplica-se de forma natural a problemas complexos, cuja decomposição funcional é bem definida; o que pode resultar em módulos fracamente acoplados e altamente coesos. Por exemplo, reconhecer um montante literal, reconhecer um montante numérico, reconhecer uma data ou verificar uma assinatura são funções bem definidas, à medida que elas não são apenas aplicáveis ao problema de tratamento de cheques. Deve-se portanto destacar que a decomposição de uma aplicação em módulos fracamente acoplados e altamente coesos não resulta necessariamente em um sistema mais robusto. No entanto, a interação entre os agentes torna o processo de tratamento de cheques bancários mais robusto, uma vez que reduz o número de cheques rejeitados.

No contexto da segunda hipótese: mostrou-se, também experimentalmente, que a interação entre os agentes pode reduzir o problema da inconsistência dos dados de um agente e da troca dinâmica de informações entre módulos funcionalmente independentes. Por exemplo, os módulos que reconhecem, em paralelo, os montantes numérico e literal de um cheque,

necessitam trocar informações para aumentar suas crenças sobre o que está sendo reconhecido. A importância destas trocas fica clara a partir dos resultados obtidos em cada uma das três versões do Sistema *Pandora*, onde os percentuais de cheques aceitos foram de 9% quando não havia nenhuma troca de informação entre os agentes e 70% no caso contrário.

No contexto da terceira hipótese: mostrou-se, também experimentalmente, que a capacidade de um sistema multi-agente em levar em conta situações de conflito, por exemplo, escassez de recurso é importante. Tal capacidade permite em particular a concepção de sistemas abertos, à medida que pode-se inserir ou remover agentes do sistema de forma dinâmica, perturbando minimamente o funcionamento do ambiente. Para este fim e também para balancear a carga do sistema, foi implementado o *agente gerenciador*.

Sabe-se que este agente toma suas decisões levando em conta o tempo médio gasto por cada um dos outros agentes para concluir seus cálculos sobre uma dada tarefa de reconhecimento, como mostrado na seção 4.2. Desta forma, todos os mecanismos para o funcionamento do agente gerenciador foram implementados. No entanto, não foi possível avaliar a performance deste agente, uma vez que os dados de reconhecimento foram simulados e o restante do processamento é apenas cálculo, o que torna o tempo de execução das tarefas de cada um dos agentes praticamente iguais.

Deve-se destacar, no caso do reconhecimento dos *montantes numérico e literal*, a freqüente presença de informações inconsistentes, em particular, porque os respectivos agentes devem reconhecer a mesma informação, porém codificada em formatos diferentes. Desta forma, foi necessário lançar mão de mecanismos capazes de tratar tais situações de forma simples e adequada. Utilizou-se então e com sucesso, a lógica evidencial paraconsistente, *tanto* para decidir quando um agente deveria solicitar uma informação, *como* para decidir se um cheque deve ser aceito ou não.

Como *trabalhos futuros* pode-se citar as seguintes evoluções:

- introduzir mecanismos/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;
- implementar os mecanismos/procedimentos desenvolvidos em uma aplicação real, testando em particular a interação entre as camadas de reconhecimento e interpretação no interior de cada agente;
- testar outras formas de organização dos agentes que não regida pela figura de um facilitador.

VI - Referências

Abe, J. M., Papavero N., (1991), **Teoria Intuitiva dos Conjuntos**, São Paulo: Makron Books, McGraw Hill.

Abe, J. M.; Ávila, B. C.; Prado, J. P. A., (1998), **Multi-Agents and Inconsistence**, In: ICCIMA'98, p.131-42, Traralgon, Austrália.

Abriat P., (1991), **Conception et Réalisation d'un Système Multi-Agents de Robotique Permettant de Récupérer les Erreurs dans les Cellules Flexibles**, Thèse de Doctorat, Division Informatique, U.R.A. C.N.R.S. n° 817 HEUDIASYC, Université de Technologie de Compiègne , Compiègne.

Angelotti E. S.; Scalabrin E. E.; Ávila B. C.; Bortolozzi F., (2000a), **A Paraconsistent System of Autonomous Agents for Brazilian Bank Check Treatment**, In: Proceedings of XX International Conference of the Chilean Computer Science Society, IEEE Computer Society Press, Santiago, Chile, Novembro.

Angelotti E. S.; Scalabrin E. E.; Ávila B. C.; Bortolozzi F., (2000b), **Concepção de um Sistema de Agentes Independentes Paraconsistente para o Tratamento de Cheques Bancários Brasileiros**, I Congresso de Lógica Aplicada à Tecnologia – Laptec'2000, Faculdades SENAC, São Paulo, Setembro.

Angelotti E. S.; Scalabrin E. E.; Ávila B. C.; Bortolozzi F., (2000c), **An Open System of Cognitive Agents for Brazilian Bank Check Treatment**, In: Third Iberoamerican Workshop On Distributed Artificial Intelligence And Multi-Agent Systems, Atibaia, São Paulo, Novembro.

Angelotti E. S.; Scalabrin E. E.; Ávila B. C., (2001a), **Using Paraconsistent Logic in a Multi-Agent System**, In: Proceedings of International Workshop Computational Models of Scientific Reasoning and Applications (CMSRA - 2001) and International Conference on Artificial Intelligence, Las Vegas, USA, Junho.

Angelotti E. S.; Scalabrin E. E.; Ávila B. C., (2001b), **PANDORA: A Multi-Agent System Using Paraconsistent Logic**, In: Proceedings of International Conference onComputational Intelligence and Multimedia Applications (ICCIMA'01), YokosukaCity, Japan, Outubro.

Ávila, B. C., (1996), **Uma Abordagem Paraconsistente 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.

Ávila, B. C.; Abe, J. M.; Prado, J. P. A., (1997), **Paralog_e: A Paraconsistent Evidential Logic Programming Language**, XVII International Conference of the Chilean Computer Science Society, IEEE Computer Society Press, pp.2-8, Valparaiso, Chile, November.

Barbuceanu M.; Fox M. S., (1995), **A Language for Describing Coordination in Multi Agent Systems**, In: Proceedings First International Conference on Multi-Agent Systems – ICMAS'95.

Beer R. D., (1992), **A Dynamical Systems Perpective on Autonomous Agents**, In : Special Issue of the AI Journal on Computational Theories of Interaction and Agency.

Blair, H. A.; Subrahmanian V. S., (1987), **Paraconsistent Logic Programming**, Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Springer-Verlag, Vol. 287, pp. 340-360.

Blair, H. A.; Subrahmanian V. S., (1988), **Paraconsistent Foundations for Logic Programming**, Journal of Non-Classical Logic.

Brito Jr. A. S.; Bortolozzi F.; Lethelier E.; Sabourin R., (2000), **Numeral Slant Normalization Based on Contextual Informational**, In: 7th International Workshop on Frontiers in Handwritten Recognition, Amsterdam, Netherlands, September.

Brooks R. A., (1986), **A Robust Layered Control System for a Mobile Robot**, In : IEEE Journal of Robotics and Automation, 2(1), pp. 14-33, March.

Cammarata, S., Mc Arthur D., Steeb R., (1983), **Strategies of Cooperation in Distributed Problem Solving**, Proceedings of the Eighth IJCAI, vol. 2, pp. 767-770, Karlsrube, August 8-12. Also in : Readings in Distributed Artificial Intelligence, A.H. Bond & L. Gasser, editors, pp. 102-105, Morgan Kaufmann Publishers, San Mateo, California, 1988.

Castelfranchi C., (1990), **A Pont Missed in Multi-Agent, DAI and HCI**. Decentralized AI, Y. Demazeau & J-P. Müller (Eds.), Elsevier Science Publisher B.V. (North-Holland), pp. 49-62.

Chaib-draa B., Vanderveken D., (1995), **On the Success and Satisfaction of Speeck Acts for Computational Agents**, In : Essays in speech act theory, Vanderveken D. and Kubo S. (eds.), http ://www.ift.ulaval.ca/publications/chaib/pub-95/pub-95.html

Cohen P. R., (1978), **On Knowing what to say : Planning speech acts**, Ph.D. Thesis ; Technical Report No. 118, Department of Computer Science, University of Toronto, January.

Cohen P.R., C.R. Perrault, (1979), **Elements of Plan-Based Theory of Speech Acts**, Cognitive Science, 3(3), pp.177-212. Also in : Readings in Distribued Artificial Intelligence, A.H. Bond & L. Gasser, editors, pp. 169-186, Morgan Kaufmann Publishers, San Mateo, California, 1988.

Cohen P.R., Levesque H.J., (1988), **Intention Is Choice with Commitment**, In : Artificial Intelligent, 42(2-3), pp. 213-261

Conry S. E., Mayer R. A., Lesser V. R., (1988), **Multistage Negotiation in Distributed Planning**, In : Readings in Distribued Artificial Intelligence, Bond and Gasser (eds) Morgan Kaufman, page 367-383.

Conry S. E., K. Kuwabara, V. R. Lesser, R. A. Mayer, (1991), **Multistage Negociation for Distributed Constraint Satisfaction**, IEEE Transaction on Systems, Man and Cybernetics, pp. 1462-1477, SMC-21(6), November-December.

Corkill D. D., Gallagher K. Q., Murray K. E., (1986), **GBB : A Generic Blackboard Development System**, In AAAI, 2, pp. 1008-1014.

Cutkosky M. R., Engelmore R. S., Fikes R. E., Genesereth M. R., Gruber T. R., Mark W. S., Tenenbaum J. M., Weber J. C., (1993), **PACT : An experiment in Integrating Concurrent Enginnering Systems**, IEEE Computer, January, pp. 23-37.

da Costa, N. C. A., (1963), **Calculs Propositionnels pour les Systèsmes Formales Inconsistants**, Compte Rendu Acad. des Sciencis (Paris), 257, pp. 3790-3792, in French.

da Costa, N. C. A.; Henschen, L. J.; Lu J. J.; Subrahmanian, V. S., (1990), **Automatic Theorem Proving in Paraconsistent Logics: Logics and Implementation**, In: 10th International Conference on Automatic Deduction, Lecture Notes in Computer Science, Vol 449, pp.72-86.

da Costa, N. C. A., Abe J. M., Murolo A. C., Leite, C. F. S.; Silva Filho J. I., (1999), **Lógica Paraconsistente Aplicada**, São Paulo : Atlas.

Davis R., Smith R. G., (1983), **Negociation as a Metaphor for Distributed Problem Solving**, Artificial Intelligence, 20(1) : 63-109.

Demazeau Y., Müller J-P., (1990), **Decentralized Artificial Intelligence**, In : Demazeau Y., Müller J-P., Decentralized AI, Elsevier Science Publishers.

Durfee E., Lesser V.R., Corkill D. D. (1987), **Cooperation through Communication in a Distributed Problem Solving Network**, IEEE Transactions on Computers, vol 36 (11).

Enembreck, F.; Ávila B. C.; Sabourin, R., (1999), **Decision Tree-Based Paraconsistent Learning**, In: Proceedings of XIX International Conference of the Chilean Computer Science Society, pp. 32-44, IEEE Computer Society Press, Talca, Chile.

Etzioni O., (1995), **What is an agent ?,** In : http ://www.cs.huji.ac./labs/dai/wkshp/defag.html

Faraco, R. A., (1998), **Uma Arquitetura De Agentes Para Negociação Dentro Do Domínio Do Comércio Eletrônico**, Florianópolis : UFSC, 1998.

Ferber J., Jacopin E., (1990), **The Framework of Eco-Problem Solving**, In : Proceedings of the second MAAMAW, Saint-Quentin en Yvelines, August, 13-16.

Ferber J., Drogoul A., (1992), **Using Reactive Multi-Agent Systems in Simulation and Problem Solving**, In : Avouris N.M., Gasser L., eds., Distributed Artificial Intelligence : Theory and Praxis, Kluwer Academic Publishers, pp. 53-80.

Ferber J., (1995), **Les Systèmes Multi-Agents**. Interéditions, Paris.

Finin, T., (1996), **UMBC KQML Web**, Lab for Advanced Information Technology. In: http://www.cs.umbc.edu/kqml

Finin T.; Weber J.; Wiederhold G.; Genesereth M.; Fritzson R.; McKay D.; McGuire J.; Pelavin R., Shapiro S.; Beck C., (1993), **Specification of the KQML Agent-Communication Language**, DARPA Knowledge Sharing Initiative, External Interfaces Working Group, June.

Finin T.; Labrou Y., (1997), **A Proposal for a new KQML Specification**, TR CS-97-03.

Fischer K., Müller J. P., Pischel M., (1996), **Cooperative Transportation Scheduling : An Application domain for DAI**, In : Applied Artificial Intelligence, 10 :1-33.

Freitas, C.; Bortolozzi, F.; Sabourin, R., (1998), **Reconhecimento do Extenso Manuscrito em Cheques Bancários Brasileiros**, ISDM98, Curitiba, Brasil, pp 149-167, Novembro.

Gasser L., Braganza C., Herman N., (1988a), **Implementing Distributed AI System using MACE**, In : Readings in Distributed Artificial Intelligence, A.H. Bond & L. Gasser, editors, pp. 445-450, Morgan Kaufmann Publishers, San Mateo California, 1988.

Gasser L., Braganza C., Herman N., (1988b), **MACE : a Flexible Testbed for Distributed AI Research**, In : Readings in Distribued Artificial Intelligence, Bond and Gasser (eds) Morgan Kaufman, pp 119-152.

Gasser L., (1992), **An Overview of DAI**, In : Distributed Artificial Intelligence : Theory and Praxis, N.M. Avouris & L. Gasser, editors, pp. 9-30, Kluwer Academic Publishers.

Genesereth M. R., Ginsberg M. L., Rosenschein J. S., (1986), **Cooperation without Communication**, In : Readings in Distributed Artificial Intelligence, A.H. & L. Gasser, editors, pp. 220-226, Morgan Kaufmann Publishers, San Mateo, California, 1988.

Gouveia F. R., (1992), **Contribution a l'étude des mécanismes de coordination dans les systèmes d'agents autonomes**, Thèse de Doctorat, Division Informatique, U.R.A. C.N.R.S. n° 817 HEUDIASYC, Université de Technologie de Compiègne , Compiègne.

Hayes-Roth B., (1995), **An Architecture for Adaptative Intelligent Systems**, In : Artificial Intelligence, vol 72, pp. 329-365.

Hewitt C., (1988), **Office are Open Systems**, In : Bond A., Gasser L., eds., Readings in Distributed Artificial Intelligence, Morgan Kaufman.

Hintikka J., (1963), **Knowledge and Bilief**, Cornell University Press.

Jennings N. R., Wooldridge M., (1996), **Software Agents**, In : IEE Review, pp. 17-20, January.

Kuokka D., Harada L., (1995), **Matchmaking for information agents**, Proceedings of the Joint Conference on Artificial Intelligence.

Khoualdi K., (1994), **Filtrage d'alarmes pour un système automatise par une approche multi-agents**, Thèse de Doctorat de l'Université Paris 6, Paris.

Malone T. W, (1987), **Modeling Coordination in Organizations and Markets**, In : Proc. Performance'87, 12th IFIP WG7.3 International Symposium on Computer Performance, Brusels, December.

Mayfield J., Labrou Y., Finin T., (1995), **Evaluation of KQML as an Agent Communication Langage**, In : Intelligent Agents Volume II—Proc. of the 1995 Workshop on Agents Theories, Architecture, and Langages, M. Wooldridge, J-P. Müller and M. Tambe (eds). Lecture Notes in Articicial Intelligence. Springer-Verlag

Morita M. E.; Bortolozzi F.; Facon J.; Sabourin R., (1999), **Mathematic Morphology and Weighted Least Square to Correcting Handwritten**, Proceedings of IEEE — ICDAR99 — The 50th International Conference on Document Analysis an Recognition, Vol.I, pp. 430-433, Bangalore, India, September.

Moses Y., Shoham Y., (1993), **Belif as Defeasible Knowledge**, In : Artificial Intelligence, 64, pp. 299-321.

Mullen T., Wellman M. P., (1995), **A Simple Computational Market for Network Information Services**, In : Proceedings of the First International Conference on Multiagent Systems (ICMAS'95), San Francisco, CA, USA, June

Oliveira, L.E.; Bortolozzi F.; Lethelier E.; Sabourin R., (2000), **A New Segmentation Approach for Handwritten Digits**, 15th International Conference on Pattern Recognition, Barcelona, Spain, September.

Prado J.P.A., (1996), **Uma Arquitetura para Inteligência Artificial Distribuída Baseada em Lógica Paraconsistente Anotada**, Tese de Doutorado, Universidade de São Paulo, São Paulo.

Rao, A., S., Georgeff, M. P., Modeling Rational Agents within a BDI-Architecture, Technical Notes 14, Autralian Artificial Intelligence Institute, 1991.

Rosenschein J.S., Zlotkin G., (1994), **Designing Conventions for Automated Negotiation**, In : AI Magazine, vol. 15, n° 3, FALL 1994.

Sandholm T., (1993), **An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations**, In : Proceedings of the Eleventh National Conference on AI, July II-15, Washington, DC, pp :256-262.

Santos J.E.B; Bortolozzi F.; Sabourin R., (1997), **A Simple Methodology to Bank Check Segmentation**, Published in the Lecture Notes in Computer Science – on Advances in Document Image Analysis, number 1339, Springer-Verlag, Curitiba, Brazil.

Scalabrin, E. E., (1996a), **Conception et Réalisation d'environnement de développement de systèmes d'agents cognitifs**, Thèse de Doctorat, Université de Technologie de Compiègne, France.

Scalabrin E. E., Vandenberghe L., De Azevedo H., Barthès J-P. A., (1996b), **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 Notres in Artificial Intelligence 1159, Springer), Curitiba, Brazil, October.

Scalabrin E. E., Barthès J-P. A., (1996c), **OSACA 1.0 : A Primer**, Rapport technique, Memo UTC/GI/DI/N 121, January.

Scalabrin E.; Bortolozzi F.; Kaestner C.; Sabourin R., (1998), **Multicheck: Une Architecture d'agents cognitifs indépendants pour le traitement automatique des chèsques bancaires Brésiliens**, In: CIFED'98, Canada, Maio.

Shen W., Barthès J-P., (1995), **DIDE : A Multi-agent environnement for Enginnering Design**, In : Proceedings First International Conference on Multi-Agent Systems, AAAI Press/The MIT Press, june 12-14, San Francisco, California, pp. 344-351

Shen W., Barthès J-P., (1996), **An Experimental Multi-Agent Environnement for Engineering Design**, In : International Journal of Cooperative Information Systems, Ed. World Scientific Publishing Company, Vol. 5, Nos. 2 & 3, pp. 131-151.

Shmeil M. A. H., (1999), **Sistemas Multiagente na Modelação da Estrutura e Relaçao de Contratação de Organizações**, Tese de Doutorado, Faculdade de Engenharia, Universidade do Porto, Portugual.

Singh M. P., (1998), **Agente Communication Languages: Rethinking the Principles**, IEEE Computer, pp. 40-47, dezembro.

Smith R. G., (1980), **The Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solver**, IEEE Transations on Computers, C-29(12), pp 1104-1113.

Smith R. G., Davis R., (1981), **Frameworks for Coorperation in Distributed Problem Solving**, In : IEEE Transations on Systems, Man Cybernetics, A.H. Bond & L. Gasser, editos, pp. 61-70. Also in : Readings in Distributed Artificial Intelligence, A.H. Bond & L. Gasser, editors, pp. 61-70, Morgan Kaufmann Publishers, San Mateo, California, 1988.

Steels L., (1989), **Cooperation between Distributed Agents Through Self-Organization**, In : Proceedings of the First MAAMAW, Cambridge, England, August, pp. 16-18.

Subrahmanian V., S.; (1987a), **Towards a Theory of Evidential Reasoning**, Logic Colloquium'87, The European Summer Meeting of the Association for Symbolic Logic, Granada, Spain, July.

Subrahmanian V., S.; (1987b), **On the Semantics of Quantitative Logic Programming**, Proceedings of 4th IEEE Symposium on Logic Programming,,San Francisco, September.

Van Emden, M. H., (1986), **Quantitative Deduction and Its Fixpoint Theory**, The Journal of Logic Programming, vol 1, new York.

Wesson, R., Hayes-Roth F., Burge J. W., Stasz C., Sunshine C. A., (1981), **Network structures for Distributed Situation Assessment**, In : IEEE Transactions on Systems, Man and Cybernetics, SMC-11(1), pp. 5-23. Also in : Readings in Distributed Artificial Intelligence, A.H. Bond & L. Gasser, editors, pp. 71-89, Morgan Kaufmann Publishers, San Mateo, California, 1988.

Wooldridge M. J., Jennings N. R., (1994), **Agent Theories, Architectures, and Languages : A Survey**, In : ECAI'94, Workshop on Agent Theories, Architectures and Languages, Amsterdam, The Netherlands, pp. 1-32.