UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

FRANCISCO CALAÇA XAVIER

Cognare

Um sistema para alocação dinâmica de recursos baseado em técnicas de Inteligência Artificial

Goiânia
2012

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Cognare – Um sistema para alocação dinâmica de recursos baseado em técnicas de Inteligência Artificial

Autor(a): Francisco Calaça Xavier

Goiânia, 21 de junho de 2012.

_____

Francisco Calaça Xavier – Autor

_____

Cedric Luiz de Carvalho – Orientador

FRANCISCO CALAÇA XAVIER

Cognare

Um sistema para alocação dinâmica de recursos baseado em técnicas de Inteligência Artificial

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em CIÊNCIA DA COMPUTAÇÃO.

Área de concentração: Inteligência Artificial.

Orientador: Prof. Cedric Luiz de Carvalho

Goiânia
2012

FRANCISCO CALAÇA XAVIER

Cognare

Um sistema para alocação dinâmica de recursos baseado em técnicas de Inteligência Artificial

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em CIÊNCIA DA COMPUTAÇÃO, aprovada em 21 de junho de 2012, pela Banca Examinadora constituída pelos professores:

Prof. Cedric Luiz de Carvalho
Instituto de Informática – UFG
Presidente da Banca

Prof. Keiji Yamanaka
Universidade Federal de Uberlândia – UFU

Prof. Celso Gonçalves Camilo Júnior
Universidade Federal de Goiás – UFG

Francisco Calaça Xavier

Estudou Matemática na Universidade Federal de Goiás - UFG. Graduou-se em Redes de Comunicação no Centro Federal de Educação Tecnológica de Goiás. Durante a Graduação participou de um programa de iniciação científica onde desenvolveu um CAD para análise de transitórios em circuitos RL e RC. Durante o Mestrado na UFG foi bolsista da Fundação de Amparo à Pesquisa do Estado de Goiás e desenvolveu um trabalho prático no desenvolvimento de um sistema que utiliza técnicas de Inteligência Artificial na alocação dinâmica de recursos. Este sistema foi implantado no projeto Nota Fiscal Eletrônica, da Secretaria da Fazenda de Goiás e no COD (Centro de Operação e Distribuição) da Celg (Centrais Elétricas de Goiás).
Atualmente estuda aplicações das técnicas de Inteligência Artificial na Computação Ubíqua e na Robótica.

Dedico este trabalho primeiramente a Deus.

Aos meus pais, Waldir e Vitória, por terem sempre me apoiado no meu projeto mais fascinante: a busca pelo conhecimento.

À minha esposa, Luciana, pelo companheirismo, compreensão e carinho.

Ao meu filho, Pedro, que mesmo ainda no útero da minha esposa, já é capaz de me inspirar e motivar.

Agradecimentos

Agradeço primeiramente meu orientador Cedric, pelos ensinamentos, sugestões, conselhos, paciência, atenção e todas as críticas construtivas.

À Fundação de Amparo à pesquisa do Estado de Goiás pela bolsa que muito me ajudou.

Ao meu irmão Otávio, pelas conversas que tanto me inspiraram novas idéias.

Ao meu amigo Max, que compartilhou comigo os desafios que existem em fazer um Mestrado.

Talento é 1 porcento inspiração e 99 porcento transpiração.

Thomas Alva Edison,
Foi um inventor e empresário dos Estados Unidos que desenvolveu muitos dispositivos importantes e de grande interesse industrial.

Resumo

Xavier, Francisco Calaça. **Cognare**. Goiânia, 2012. 106p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

O problema da tomada de decisão quanto a alocação de recursos está presente em diversas áreas da sociedade. A alocação de ambulâncias à ocorrências de acidentes com vítimas e a alocação de equipes para solução de problemas no fornecimento de energia elétrica são exemplos de situações onde são necessárias tomadas de decisão. Os problemas que ocorrem na alocação de recursos de *hardware* quando um sistema é executado de forma distribuída também requerem decisões. Neste contexto, este trabalho apresenta o sistema COGNARE, que reúne a utililização de técnicas como Algoritmos Genéticos, Lógica *Fuzzy* e Sistemas Multiagentes com o objetivo de alocar dinamicaminte tarefas a recursos. O COGNARE foi utilizado em duas situações distintas. Na primeira, o problema consistia em alocar dinamicamente viaturas de uma empresa de distribuição de energia elétrica a ocorrências de falhas no fornecimento. Na segunda situação, o problema consistia em alocar dinamicamente recursos de *hardware* em um sistema distribuído. Nestes dois casos, o COGNARE apresentou-se como um sistema de alocação de recursos eficiente.

Palavras–chave

Alocação dinâmica de Recursos, Lógica Fuzzy, Algoritmos Genéticos, Aprendizagem Evolutiva, Sistemas Multiagentes

Abstract

Xavier, Francisco Calaça. **Cognare**. Goiânia, 2012. 106p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The problem of decision making about the allocation of resources is present in many areas of society. The allocation of ambulances to the occurrence of accidents with victims and the allocation of teams to solve problems in the supply of electricity are examples of situations where it is necessary to make this decision. We can also mention the problems that occur in the allocation of hardware resources when a system is running in a distributed form. In this context, this paper presents the system COGNARE, which brings together techniques such as Genetic Algorithms, Fuzzy Logic and Multiagent Systems in order to allocate tasks to resources dynamically. The COGNARE was used in two different situations. At first, the problem was to allocate vehicles to a distributor of electricity to occurrences of failures in supply. In the second situation, the problem was to allocate hardware resources in a distributed system. In both cases, the COGNARE presented as a system of allocating resources efficiently.

Keywords

Fuzzy Logic, Genetic Algorithms, Evolutionary Learning, Multiagent Systems

Lista de Abreviaturas e Siglas

AG - Algoritmo Genético

BDI - *Belief-Desire-Intention*

Celg - Centrais Elétricas de Goiás

COD - Centro de Operação e Distribuição

FCL - *Fuzzy Control Language*

FIS - *Fuzzy Inference System*

FRBS - *Fuzzy Rule Based Systems*

GCCL - *Genetic Cooperative-Competitive learning*

HIDER - *Natural Encoding for Hierarchical Decision Rules*

IA - Inteligência Artificial

IAD - Inteligência Artificial Distribuída

ICP-Brasil - Infra-estrutura de chaves públicas do Brasil

JADE - *Java Agent Development Framework*

JVM - Máquina Virtual Java

KB - Base de Conhecimento

NF-e - Nota Fiscal Eletrônica

NSGA-II - *Non-dominated Sorting Genetic Algorithm II*

PDT - *Prometheus Design Tool*

QoS - *Quality of Service*

SGBD - Sistema Gerenciador de Banco de Dados

SGERD - *Steady-State Genetic Algorithm for Extracting Fuzzy Classification Rules From Data*

SMA - Sistemas Multiagentes

TARGET - *Tree Analysis with Randomly Generated and Evolved Trees*

UFG - Universidade Federal de Goiás

UTI - Unidade de Tratamento Intensivo

VM - Máquina Virtual

XML - *eXtensible Markup Language*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

CAPÍTULO 1

Introdução

O problema da alocação dinâmica de recursos, aparece em todos os cenários onde a quantidade de tarefas é sempre maior que a quantidade de recursos disponíveis para resolvê-las. Neste caso, é necessário tomar a decisão sobre qual recurso será alocado a uma determinada tarefa. É possível citar diversos exemplos onde este problema aparece.

Por exemplo, se em um dia ocorrerem 100 acidentes de trânsito com vítimas, mas somente 20 ambulâncias estiverem disponíveis, será necessário realizar uma tomada de decisão quanto à vítima que será atendida primeiro, ou seja, alocar as ambulâncias para atenderem aos acidentes.

Em um outro exemplo, após uma tempestade é comum ocorrerem falhas no fornecimento de energia elétrica em diversas regiões. Normalmente, as empresas de distribuição de energia dispõem de uma quantidade de equipes capazes de resolver estas falhas. Em um dia de pico, a quantidade de equipes é bem menor que a quantidade total de falhas. Neste caso, torna-se necessário decidir qual região será atendida primeiro, ou seja, alocar as equipes para os atendimentos.

Para citar mais um exemplo, atualmente os sistemas de computadores necesistam cada vez de mais de recursos de *hardware* para serem executados. Normalmente, nestes casos, utiliza-se uma *grid*[1] de computadores, que tem o objetivo de aumentar a performance do sistema. Neste contexto, é necessário um balanceador de carga, que pode ser considerado um alocador de recursos de *hardware* [33].

Além destes exemplos, ainda é possível verificar outras áreas da sociedade onde o problema da alocação dinâmica de recursos ocorre, tais como, fábricas de software, indústria em geral, etc.

Neste trabalho, serão apresentadas algumas técnicas da Inteligência Artificial que podem ser empregadas no tratamento do problema da alocação dinâmica de recursos.

[1] Computação em grid é um modelo computacional capaz de alcançar uma alta taxa de processamento. Isto acontece pois são divididas as tarefas entre diversas máquinas, ligadas em rede [6].

1.1 Motivação

A primeira motivação deste trabalho, é que normalmente a alocação de recursos é feita de forma manual. Esta abordagem manual toma tempo de especialistas e torna a tomada de decisão mais subjetiva. Desta forma, surge a necessidade de se criar uma solução computacional capaz de realizar esta alocação de forma automática. A segunda motivação é que, nos casos onde já existem aplicações capazes de realizar esta alocação de forma automática, existe a necessidade de que os critérios utilizados nesta alocação possam ser aprendidos a partir de decisões históricas e, desta forma, melhorar a qualidade das decisões. Esta melhoria acontece pois, com o aprendizado, é possível obter o padrão de tomada de decisão dos melhores especialistas, o que também tornam melhores as decisões do sistema.

Para testar a proposta, este trabalho trata dois tipos de casos onde este problema ocorre. No primeiro caso, apresentado na Seção 5.1, é apresentado o problema de alocação dinâmica de viaturas de uma empresa de distribuição de energia elétrica. No segundo caso, apresentado na Seção 5.2, é apresentado o problema da alocação dinâmica de servidores utilizados no atendimento da Nota Fiscal Eletrônica de Goiás.

1.2 Objetivo

O objetivo deste trabalho é desenvolver um sistema computacional capaz de:

1. Aprender a tomar decisões quanto à alocação dinâmica de recursos a partir de uma base de decisões históricas;
2. Alocar dinamicamente recursos.

1.3 Metodologia

A metodologia utilizada no desenvolvimento deste projeto compreendeu as seguintes etapas:

- **Estudo e Fundamentação Teórica**. Nesta etapa, foi realizada a pesquisa relacionada com os fundamentos teóricos deste trabalho. Foram estudadas três das áreas da Inteligência Artificial: Lógica *Fuzzy*, Algoritmos Genéticos e Sistemas Multiagentes. Com isso, foram selecionadas as referências bibliográficas para obtenção de informações relacionadas com a pesquisa.
- **Análise de Trabalhos Correlatos**. Nessa etapa, foi feito um estudo do estado da arte para a concepção do COGNARE. Foram levantados alguns trabalhos com objetivos semelhantes ao desta pesquisa.

- **Definição da Arquitetura do COGNARE e sua implementação**. Nesta etapa, foi definida a arquitetura do COGNARE. Após esta definição, o COGNARE foi implementado.
- **Aplicação do COGNARE em estudos de caso**. Nesta etapa, o COGNARE foi utilizado em dois estudos de caso. No primeiro, o objetivo foi alocar dinamicamente viaturas de uma empresa distribuidora de energia elétrica. No segundo, o objetivo foi o de alocar dinamicamente servidores que recebem Nota Fiscal Eletrônica.

1.4 Organização da Dissertação

O restante do trabalho está divido em cinco capítulos, além deste. O Capítulo 2 apresenta a fundamentação teórica de todo este trabalho. O Capítulo 3 apresenta os trabalhos relacionados com os problemas da alocação dinâmica de recursos e os problemas do aprendizado de regras. O Capítulo 4 apresenta o projeto COGNARE. O Capítulo 5 apresenta dois estudos de caso onde a aplicação do COGNARE resultou em uma melhoria no processo de tomada de decisão. Por fim, o Capítulo 6 apresenta a conclusão e alguns trabalhos futuros.

CAPÍTULO 2

Fundamentos Teóricos

Este trabalho foi baseado em três das áreas da Inteligência Artificial:

- Lógica *Fuzzy*;
- Algoritmos Genéticos;
- Agentes e Sistemas Multiagentes.

Este capítulo fornece uma visão geral dos conceitos relacionados com estas três áreas. Inicialmente, é apresentado, na Seção 2.1, os conceitos relacionados com a Lógica *Fuzzy* e os Sistemas Baseados em Regras *Fuzzy*. A Seção 2.2 apresenta os Algoritmos Genéticos, utilizados no processo de aprendizagem das regras. A Seção 2.3 descreve os agentes e sistemas multiagentes. No final, a Seção 2.4 apresenta uma discussão quanto à utilização destes conceitos no projeto COGNARE.

2.1 Lógica Fuzzy

A Lógica *Fuzzy*[23], também chamada de Difusa ou Nebulosa, é um tipo de lógica com múltiplos valores que, por sua vez, é uma extensão da lógica clássica ou lógica de Aristóteles [7]. Enquanto que na lógica clássica, uma proposição pode resultar em apenas dois possíveis valores, verdadeiro ou falso, na Lógica *Fuzzy* uma proposição pode assumir *n* valores.

Segundo Zadeh [41], as pessoas possuem a capacidade de tomarem decisões a partir de incertezas, de um conhecimento parcial, impreciso ou incompleto do ambiente. A Lógica *Fuzzy* tem como principal objetivo a mecanização dessa capacidade. Para tal, são definidos os conceitos de conjuntos *Fuzzy*, fuzzificação e defuzzificação. Estes conceitos são empregados em Sistemas Baseados em Regras *Fuzzy* (*Fuzzy Rule Based Systems* - FRBS).

A seguir serão apresentados mais detalhes sobre estes conceitos e como eles se interagem em um FRBS.

2.1.1 Conjuntos Fuzzy

Segundo Zadeh [23], os elementos de um Conjunto *Fuzzy* possuem um grau de pertinência. Sendo assim, é possível que um elemento pertença "muito" a um determinado Conjunto *Fuzzy* ao mesmo tempo em que um outro elemento pertença "pouco" a este mesmo Conjunto *Fuzzy*.

Desta forma, os Conjuntos *Fuzzy* não possuem fronteiras bem definidas e, por este motivo, são adequados à resolução de problemas em que a transição entre classes necessita ocorrer de forma suave.

Para exemplificar, seja um sistema que classifica uma pessoa como "Baixa" ou "Alta". Neste exemplo, o valor de transição entre "Baixa" e "Alta" pode ser definido como $1,80m$, ou seja, para este sistema, alguém com menos de $1,80m$ de altura será considerada "Baixa" enquanto que uma pessoa com altura igual ou acima de $1,80m$, será considerada "Alta".

Na teoria dos conjuntos clássica, uma pessoa com $1,79m$ será classificada como "Baixa", e uma pessoa com $1,80m$ será "Alta". Apesar da diferença ser de apenas $0,01m$, a transição entre as classes "Baixa" e "Alta" não é suave, conforme ilustrado na Figura 2.1. Na Lógica Fuzzy, uma pessoa pode ser classificada mais "Alta" do que "Baixa".

A figura 2.2 ilustra esta mesma classificação realizada com a Lógica *Fuzzy*. Nesta Figura, é possível notar que uma pessoa com $1,80$ é mais "Alta" do que "Baixa", ou seja, a pessoa com $1,80$ ainda pertence à classe "Baixa", mas com um grau de pertinência menor do que o seu grau de pertinência à classe "Alta". Desta forma, com a Lógica *Fuzzy*, uma pessoa com $1,80$ é mais "Alta" do que "Baixa". Os termos "Alta" e "Baixa" são denominados, na Lógica *Fuzzy*, "Termos Linguísticos".

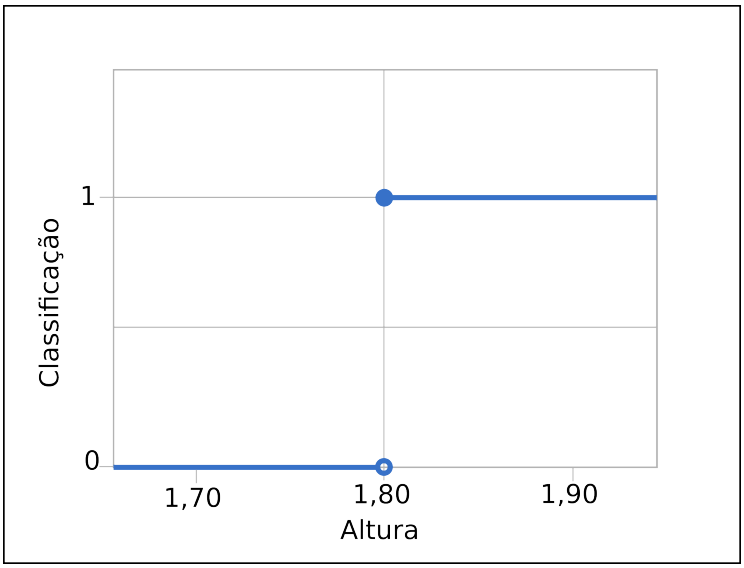


Figura 2.1: *Classificação de uma pessoa utilizando a lógica clássica*

Para cálculo do grau de pertinência de um elemento a um conjunto *Fuzzy*, é utilizada uma função, chamada de Função de Pertinência. Na Figura 2.2 é possível notar duas curvas. A curva tracejada, que expressa o quanto uma pessoa é “Baixa” em função de sua altura e a curva contínua que expressa o quanto uma pessoa é “Alta” em função de sua altura. Estas curvas são as Funções de Pertinência, para as classes “Baixa” e “Alta”, respectivamente.

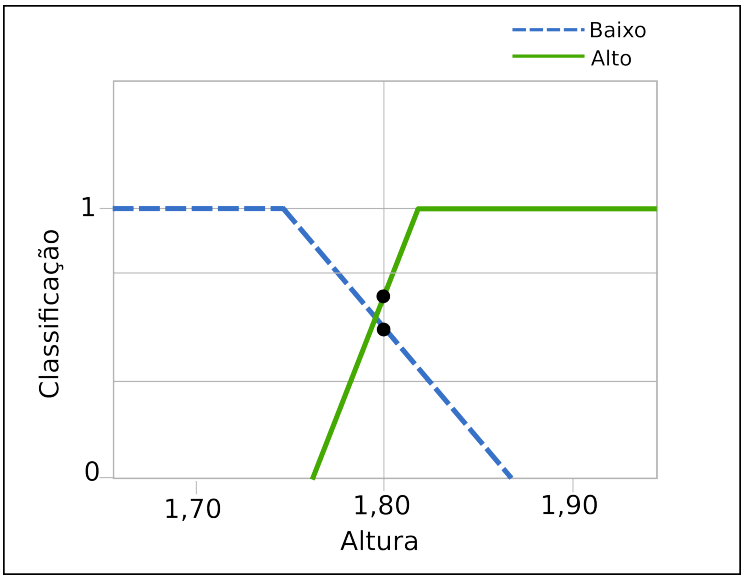


Figura 2.2: *Classificação de uma pessoa utilizando a lógica Fuzzy*

2.1.2 Fuzzificação e Defuzzificação

Fuzzificação é o processo que converte valores numéricos de uma determinada variável de entrada em valores *Fuzzy*. Este processo utiliza funções de pertinência que retornam o grau de pertinência dos valores numéricos em cada um dos conjuntos *Fuzzy*.

Defuzzificação é o processo que converte os valores *Fuzzy* de uma determinada variável de saída em valores numéricos. Este processo também utiliza funções de pertinência e existe uma para cada conjunto *Fuzzy* de saída. Além disto, é utilizado também um método de defuzzificação. Este método é necessário para combinar os valores calculados em cada regra *Fuzzy* em um único valor de saída.

2.1.3 Sistemas Baseados em Regras Fuzzy

Um Sistema Baseado em Regras *Fuzzy* [17] (FRBS) é composto por uma Base de Conhecimento (KB), que contém informações na forma de regras *Fuzzy if-then*, a camada de entrada, que contém a interface de fuzzificação, o sistema de inferência *Fuzzy* que, junto com a Base de Conhecimento realiza a inferência a partir da entrada e a camada de saída, que contém a interface de defuzzificação. Este processo é ilustrado na Figura 2.3.

As regras *Fuzzy*, contidas na Base de Conhecimento, possuem a seguinte estrutura: "*IF* um conjunto de condições são satisfeitas (antecedente da regra) *THEN* um conjunto de consequências podem ser inferidas (consequência da regra)".

Além das regras, a base de conhecimento também possui as funções de pertinência utilizadas nos processos de fuzzificação e defuzzificação dos valores de entrada e saída respectivamente.

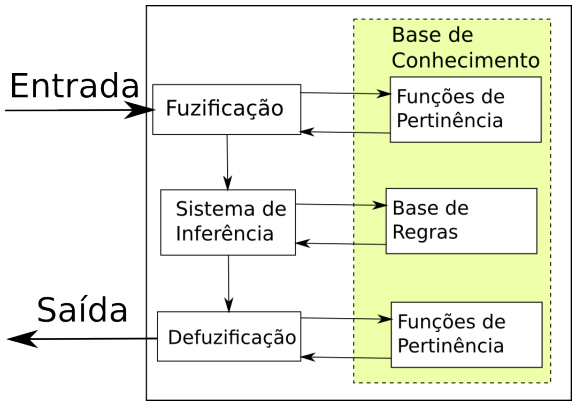


Figura 2.3: *Estrutura de um Sistema Baseado em Regras Fuzzy*

Um FRBS pode seguir a estrutura formalizada pelo *International Electrotechnical Commission* na IEC 1131-7 [19] que padroniza a *Fuzzy Control Language (FCL)*, apresentada no Apêndice B. Esta linguagem permite descrever todas as partes de um FRBS, ilustradas na Figura 2.3.

2.2 Algoritmos Genéticos

Desenvolvido por John Holland [18], o Algoritmo Genético (AG) [26] é um tipo de Algoritmo Evolutivo [3] que busca resolver problemas através da utilização de mecanismos encontrados na genética.

Os Algoritmos Evolutivos, foram baseados na Teoria da Evolução de Charles Darwin [11] onde é afirmado que a vida na Terra é o resultado de um processo de seleção em que os indivíduos mais aptos e adaptados possuirão mais chances de sobreviver e reproduzir.

Os sistemas baseados em AG lidam com indivíduos[1] de uma população de soluções onde são realizadas operações genéticas tais como seleção, reprodução, mutação, dentre outros.

O diagrama da Figura 2.4 ilustra os passos executados por um AG. O fluxo começa com a inicialização de uma população de indivíduos *P*. Geralmente este passo

[1] Os indivíduos são soluções candidatas para um problema. O conjunto destes indivíduos é chamado de População.

é aleatório. Após esta inicialização, cada indivíduo de P é avaliado com uma função de *fitness*. Esta função retorna um valor que representa a medida de adaptação do indivíduo. A seleção é realizada em função desta medida de adaptação e pode ocorrer utilizando diversos mecanismos tais como: roleta, *ranking*, por diversidade, dentre outros.

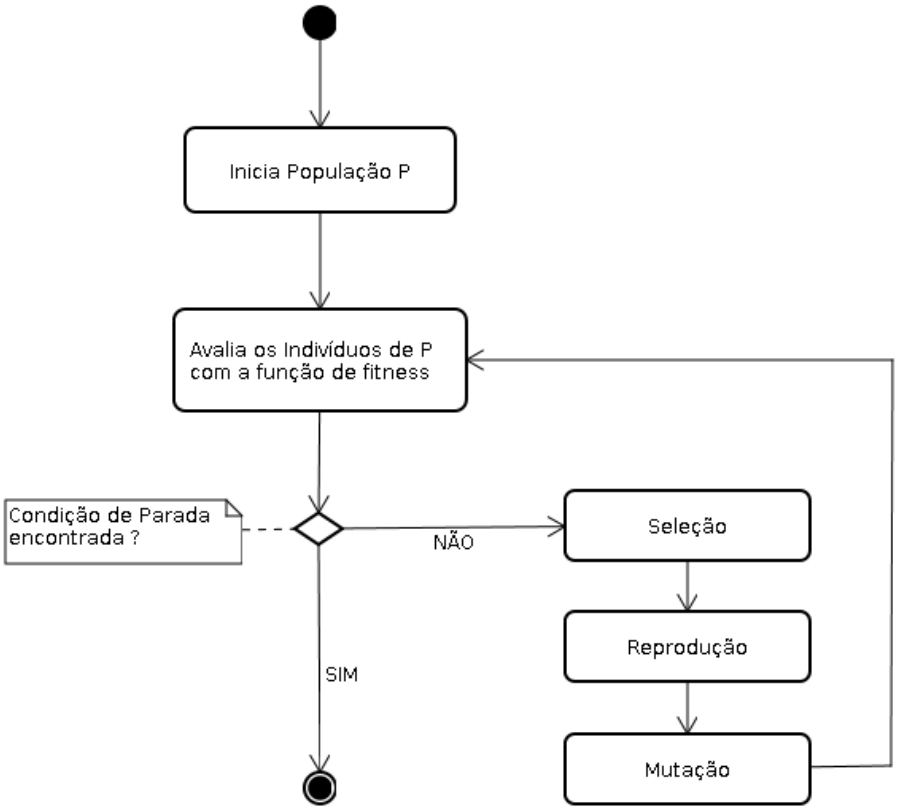


Figura 2.4: *Diagrama de fluxo de um Algoritmo Genético*

Após a seleção, diversas operações genéticas podem ser executadas, a fim de obter melhores respostas. As operações mais clássicas são:

- reprodução - é uma operação que combina dois ou mais indivíduos de uma população para criar um novo indivíduo. O objetivo principal desta operação é a troca de informações entre diferentes indivíduos.
- mutação - é uma operação que modifica aleatoriamente genes de um indivíduo. Esta operação tem como objetivo diminuir a probabilidade do Algoritmo Genético convergir para uma solução considerada um ótimo local;
- elitismo - é uma operação que copia os melhores indivíduos de uma geração de população para a próxima geração. Esta operação visa principalmente manter as melhores soluções entre gerações de uma população;

Além destas operações clássicas, recentemente, diversos pesquisadores têm apresentado abordagens, também relacionadas com conceitos da genética tais como reprodução *in vitro* [8].

Finalmente, um Algoritmo Genético pode utilizar algumas formas distintas para avaliar a condição de parada. Entre elas, a primeira forma utilizada para a parada é o prévio estabelecimento de uma quantidade fixa de iterações para a execução. Quando esta quantidade é alcançada, o algoritmo para e retorna a população atual como resultado. A segunda forma utilizada para a parada é a avaliação dos melhores indivíduos de cada iteração. Se a medida de adaptação deste indivíduos não variar muito, com base em um parâmetro previamente estabelecido, diz-se que que a população convergiu para uma solução. Desta forma, o algoritmo para, retornando esta população.

2.3 Agentes e Sistemas Multiagentes

Um Agente de *software* é um sistema autônomo capaz de perceber o seu ambiente por meio de sensores e, por meio de atuadores, agir sobre ele [5]. Desta forma, o agente é um sistema independente que toma suas próprias decisões. Esta é uma propriedade que distingue um sistema como "agente" [36].

Segundo Wooldridge [39], um agente é autônomo, porque opera sem a intervenção direta de seres humanos ou outros elementos, ou seja, tem controle sobre suas ações e seu estado interno. Um agente é social, porque coopera com seres humanos ou outros agentes a fim de alcançar as suas tarefas. Um agente é reativo, porque ele percebe seu ambiente e responde em tempo hábil às mudanças que ocorrem nele. Um agente é pró-ativo, porque não simplesmente age em resposta ao seu ambiente, mas é capaz tomar a iniciativa em realizar determinadas ações relacionadas com o cumprimento do seu objetivo.

Neste sentido, Sistemas Multiagentes (SMA) são sistemas computacionais compostos por uma comunidade de agentes[2]. Constituem uma subárea da Inteligência Artificial Distribuída[3][36] onde o comportamento social é a base para a inteligência do sistema.

2.3.1 Arquiteturas utilizadas para agentes inteligentes

De acordo com a arquitetura, um agente pode ser classificado em quatro classes [36]:

- Arquitetura baseada em lógica - utiliza técnicas de representação e manipulação simbólicas; A vantagem deste tipo de arquitetura se dá pelo fato de serem fáceis

[2] Comunidade de agentes são vários agentes de *software* que interagem entre si.

[3] A Inteligência Artificial Distribuída (IAD) foi criada em meados da década de 1970 e busca resolver problemas através da aplicação de conceitos relacionados com Inteligência Artificial (IA) e Computação Distribuída [36].

de programar pois os seres humanos entendem facilmente a lógica. A desvantagem principal é que é difícil traduzir o mundo real em uma descrição simbólica precisa e adequada, além de demandar um tempo considerável para a execução de seus resultados;

- Arquitetura Reativa - implementa a tomada de decisões como um mapeamento direto da situação para a ação e são baseados em um mecanismo de estímulo-resposta desencadeada por dados do sensor;
- Arquitetura *belief-desire-intention* (BDI) - Baseada nas quatro camadas: crenças (*beliefs*), desejos (*desires*), intenções (*intentions*) e planos (*plans*) ligadas a um interpretador, conforme ilustrado na Figura 2.5. As crenças representam a informação que um agente tem sobre seu ambiente. Esta informação pode ser incompleta ou incorreta. Desejos representam as tarefas atribuídas ao agente e correspondem aos objetivos que ele deve realizar. Intenções representam desejos que o agente se comprometeu a atingir. Finalmente, os planos são um conjunto de ações que podem ser executadas por um agente a fim de atingir as suas intenções. Estas quatro estruturas de dados são gerenciados pelo interpretador do agente que é responsável por atualizar as crenças a partir de observações feitas do meio ambiente, gerando novos desejos com base em novas crenças. Finalmente, o interpretador seleciona uma ação a ser executada com base em intenções atuais do agente e do conhecimento processual;
- Arquitetura em camadas - é uma arquitetura híbrida e flexível pois permite aos agentes um comportamento reativo e pró-ativo simultaneamente. Para fornecer tal flexibilidade, utilizam um subsistema hierárquico em camadas.;

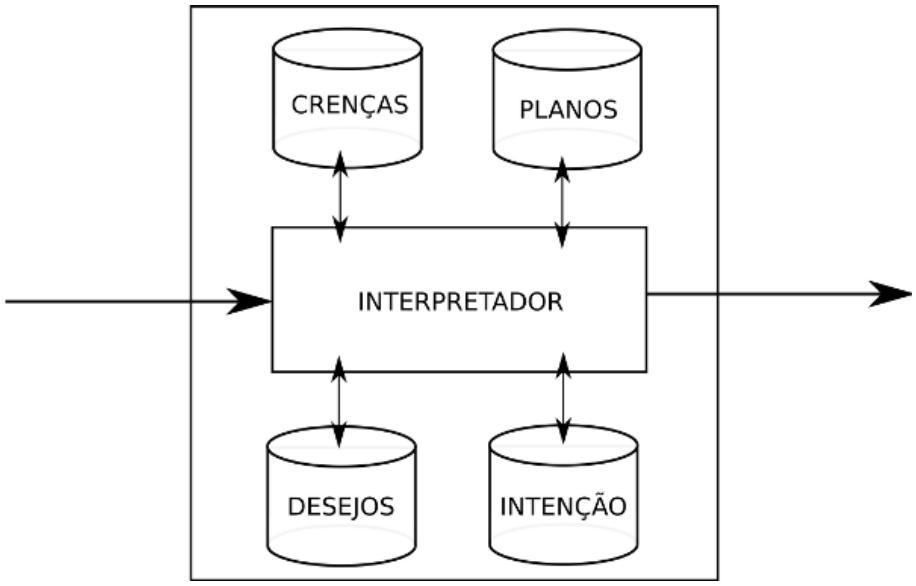


Figura 2.5: *Arquitetura BDI de um agente (retirado de [5]).*

2.4 Discussão

Este Capítulo apresentou uma visão geral da teoria por trás dos conceitos Lógica *Fuzzy*, Aprendizagem Evolutiva e Sistemas Multiagentes, utilizados neste trabalho com o objetivo de auxiliar na alocação dinâmica de recursos.

A principal vantagem da utilização de Lógica *Fuzzy* neste trabalho é a sua capacidade de lidar com regras cujos termos antecedentes e consequentes são elementos linguísticos, de fácil entendimento por seres humanos [41]. Outra grande vantagem, é a capacidade que sistemas *Fuzzy* tem para lidar com problemas imprecisos. Esta característica se mostra bastante útil na solução do problema da alocação dinâmica de recursos pois pequenas mudanças não implicam em grandes alterações no comportamento do sistema, como um todo.

A Aprendizagem Evolutiva tem sido utilizada na extração de conhecimento, a partir de dados históricos. Este conhecimento pode ser utilizado na geração de regras *Fuzzy* facilitando, desta forma, a utilização de sistemas de inferência *Fuzzy*. Atualmente, diversos algoritmos relacionados com a Aprendizagem Evolutiva [2] [3] [16] [24] [28], tem cumprido este objetivo de forma eficaz.

A Figura 2.6 ilustra uma forma de relacionamento desses três conceitos apresentados neste capítulo. Nesta Figura, é possível notar que a Aprendizagem Evolutiva pode ser utilizada com o objetivo de gerar regras *Fuzzy* para o sistema de inferência *Fuzzy* (na Figura, ilustrado pela caixa "Lógica *Fuzzy*"). Os agentes realizam a comunicação dos dados externos com o mundo interno. Finalmente, o sistema de inferência *Fuzzy* pode tomar decisões em função destes dados.

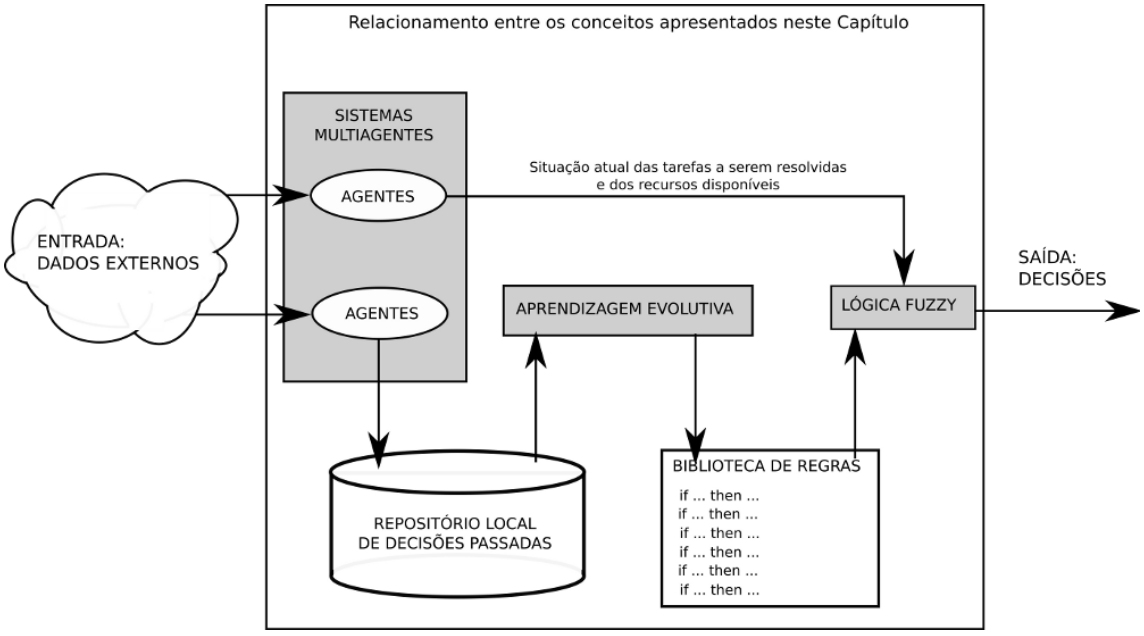


Figura 2.6: *Exemplo do relacionamento entre os conceitos apresentados neste capítulo.*

CAPÍTULO 3

Trabalhos Relacionados

Foram encontrados, na literatura científica, diversos trabalhos relacionados com o problema da alocação dinâmica de recursos e extração do conhecimento na forma de regras *Fuzzy*[1]. Este capítulo apresenta os trabalhos mais significativos, em relação à proposta desta dissertação e foi dividido da seguinte forma.

As Seções 3.1, 3.2 e 3.3 apresentam alguns trabalhos cujo objetivo foi a concepção de algoritmos capazes de extrair conhecimento, a partir de uma base de dados. Nestes trabalhos, o conhecimento extraído é representado através de regras do tipo *if ... then*.

As Seções 3.4 e 3.5 apresentam trabalhos cujo objetivo foi a utilização de técnicas da Inteligência Artificial na alocação dinâmica de recursos, propriamente dita. O trabalho apresentado na Seção 3.4 apresenta a alocação, realizada com o auxílio de AG, de equipes de trabalho a setores de uma empresa. O trabalho apresentado na Seção 3.5 apresenta a alocação dinâmica de recursos de uma máquina virtual hospedeira de um Sistema Gerenciador de Banco de Dados (SGBD).

No final, a Seção 3.6 faz uma discussão sobre esses trabalhos.

3.1 Algoritmo TARGET

Tree Analysis with Randomly Generated and Evolved Trees (TARGET) [16] utiliza AG para construir Árvores de Decisão em que cada cromossomo representa uma árvore de decisão completa. A população é chamada de floresta. Neste algoritmo, a população inicial é obtida aleatoriamente. A função de *fitness*[2] utiliza uma medida baseada nas classificações corretas realizadas e no número de condições por nó em

[1] O problema da extração do conhecimento é um problema secundário neste trabalho. Devido ao fato da proposta central estar relacionada com a utilização de um sistema de inferência *Fuzzy* no auxílio à tomada de decisão, viu-se a necessidade de pesquisar sobre técnicas utilizadas para extração do conhecimento na forma de regras *Fuzzy*.

[2] *Fitness* é uma função utilizada para avaliar cada indivídulo de uma população.

cada árvore. A evolução da floresta utiliza os operadores genéticos reprodução, mutação, clonagem e transplante.

Na reprodução, duas árvores da floresta são selecionadas aleatoriamente. A probabilidade de seleção de cada árvore é proporcional ao valor retornado pela função de *fitness*. TARGET utiliza dois tipos de reprodução: troca de nós e troca de subárvores. Na troca de nós, um nó é identificado aleatoriamente em cada uma das árvores selecionadas e, após esta escolha, estes nós são trocados, mantendo o restante das características da árvore. Na troca de subárvores, duas subárvores são selecionadas aleatoriamente, uma de cada participante do processo de reprodução. Cada subárvore é cortada da árvore original e colada na outra árvore.

Na mutação, uma árvore da floresta é escolhida aleatoriamente. A probabilidade desta escolha é proporcional ao *fitness* de cada árvore. Após a mutação ser realizada, a nova árvore é adicionada na floresta.

Além das operações de reprodução e mutação, o TARGET também realiza a operação de clonagem. A clonagem é uma espécie de elitismo[3], onde as melhores árvores são mantidas no processo evolutivo através de cópia das mesmas para a próxima geração.

O transplante é a adição de novas árvores na floresta geradas aleatoriamente. O objetivo é aumentar o "material genético" e assim, diminuir a probabilidade do Algoritmo Genético convergir para ótimos locais. A quantidade de novas árvores transplantadas é um dos parâmetros de entrada do Algoritmo TARGET.

3.2 Algoritmo HIDER*

Natural Encoding for Hierarchical Decision Rules (HIDER*) [2] é uma melhoria do algoritmo evolutivo HIDER [3] ao qual foi acrescentada a idéia chamada de *"Natural Encoding"*.

"Natural Encoding" é uma abordagem que utiliza somente números naturais para representar um intervalo de dados, quando o atributo for contínuo, ou um conjunto de valores, quando o atributo for discreto. A Figura 3.1 apresenta os dados utilizados para exemplificar esta abordagem. Trata-se de um exemplo de dados que associa o peso e a cor dos olhos com o fato da pessoa ter ou não um trabalho aceito em uma conferência relevante. Este exemplo foi retirado de [2].

O algoritmo HIDER* utiliza a codificação híbrida de um cromossomo, proposta por Aguilar[3]. Esta codificação representa um cromossomo como a união de dois tipos de genes: genes que representam atributos cujos valores são contínuos e genes que

[3]Elitismo é a cópia de elementos de uma geração para outra. Normalmente, o elitismo ocorre dentre os melhores elementos de uma geração.

representam atributos cujos valores são discretos. Um atributo contínuo é associado a dois números reais, que são os limites de um intervalo que contém o valor deste atributo. Um atributo discreto é associado a um número binário com tamanho igual à quantidade de valores que este atributo pode assumir.

peso,	**cor,**	**classe**
(55,	verde,	sim)
(57,	verde,	não)
(59,	preto,	não)
(60,	verde,	não)
(62,	preto,	sim)
(64,	azul,	não)
(68,	preto,	sim)
(70,	preto,	sim)

Figura 3.1: *Exemplo de dados utilizados na exemplificação do algoritmo HIDER [2].*

A Figura 3.2 ilustra o comprimento de um cromossomo codificado de forma híbrida[4] e o comprimento de um cromossomo codificado utilizando "Natural Encoding". Este novo método de codificação reduz consideravelmente o tamanho do cromossomo.

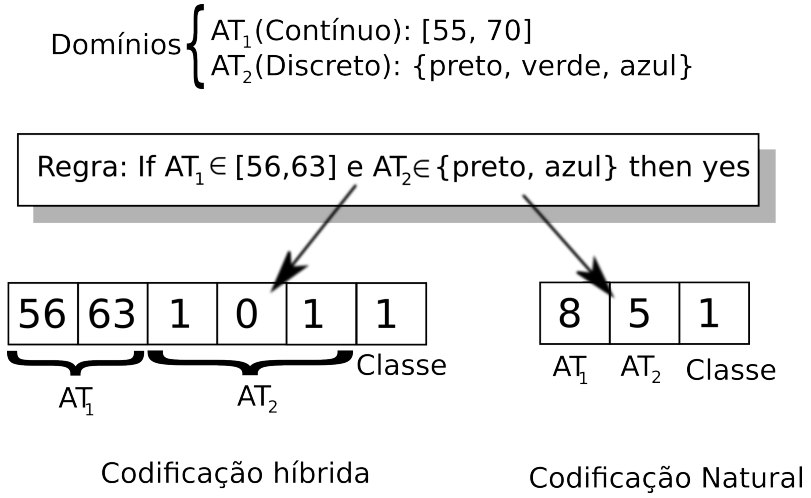


Figura 3.2: *Cromossomo híbrido vs. cromossomo natural[2]. O atributo AT_1 é o peso enquanto que o atributo AT_2 é a cor dos olhos.*

A codificação de valores discretos é feita associando-se cada combinação de elementos do domínio dos valores discretos em um número natural, conforme ilustrado na Tabela 3.2. Nesta tabela, o domínio da variável cor *{preto, verde, azul}*, que foi representado na codificação híbrida com valores binários, passa a ser representado com valores decimais, no exemplo, números do intervalo $[1,7]$.

[4]Na codificação híbrida de um cromossomo são utilizados valores reais e valores binários. Neste exemplo, os valores reais $\{56,63\}$ representam um intervalo do atributo contínuo peso e os valores binários $\{1,0,1\}$ representam os valores do atributo cor.

A codificação de valores contínuos é realizada associando subintervalos com números naturais, conforme Tabela 3.3. Os subintervalos podem ser obtidos por meio de vários algoritmos de discretização de dados [2]. A Tabela 3.3 ilustra a distribuição de números naturais por subintervalos dos dados apresentados na Figura 3.1 e utilizando $L = \{55, 56, 61, 63, 66, 70\}$ como limites dos subintervalos.

Neste sentido, o algoritmo HIDER* define os seguintes termos:

- Mutação natural discreta;
- Conjunto de mutação discreta;
- Mutação $\delta - order$;
- Reprodução natural discreta;
- Transições linha e coluna;
- Mutação natural contínua
- Reprodução natural contínua.

O algoritmo HIDER* [2] define a operação **mutação natural discreta**, utilizada para modificar os indivíduos da população[5], da seguinte forma: Seja n o valor de um gene de um indivíduo, a mutação natural do $k - simo$ bit, representado por $mut_k(n)$ é o número natural produzido utilizando a seguinte Equação 3.1:

$$mut_k(n) = (n + 2^{k-1})\%2^k + 2^k \left\lfloor \frac{n}{2^k} \right\rfloor \quad (3.1)$$

onde $k \in \{1, 2, \ldots, |\Omega|\}$, $|\Omega|$ é o número de atributos, $\%$ é o resto da divisão inteira e $\lfloor . \rfloor$ é a parte inteira da divisão. Para exemplificar, ao aplicar o valor 3 na Equação 3.1 é obtido:

$$mut_1(3) = (3 + 2^0)\%2^1 + 2^1 \left\lfloor \frac{3}{2^1} \right\rfloor = 2\ (010)$$
$$mut_2(3) = (3 + 2^1)\%2^2 + 2^2 \left\lfloor \frac{3}{2^2} \right\rfloor = 1\ (001)$$
$$mut_3(3) = (3 + 2^2)\%2^3 + 2^3 \left\lfloor \frac{3}{2^3} \right\rfloor = 7\ (111)$$

o resultado deste exemplo, assim como demais resultados são apresentados na Tabela 3.1.

Conjunto de mutação discreta é dado pela Equação 3.2:

$$Mut(n) = \bigcup_{k=1}^{|\Omega|} mut_k(n) \quad (3.2)$$

[5] Para exemplificar, seja a operação e mutação do o subconjunto *{verde, azul}*. Este subconjunto é representado pelo número 3 (011) e possui três opções de mutação: *111, 001 e 010*, respectivamente 7, 1 e 2. Desta forma o subconjunto *{verde, azul}* pode mutar para *{preto verde, azul}*, *{azul}* e *{verde}*.

	0	1	2	3	4	5	6	7
- significativo	1	0	3	2	5	4	7	6
	2	3	0	1	6	7	4	5
+ significativo	4	5	6	7	0	1	2	3

Tabela 3.1: *Valores de mutação para atributos discretos [2].*

onde $|\Omega|$ é o número de atributos e $mut_k(n)$ é calculado pela Equação 3.1. Para exemplificar, $Mut(3) = \{2,1,7\}$.

Mutação $\delta - order$ é definido pela seguinte Equação 3.3:

$$[Mut(Z)]^{\delta} = \bigcup_{z \in [Mut(X)]^{\delta-1}} (z \cup Mut(z)) \quad (3.3)$$

para exemplificar,

$[Mut(3)]^0 = \{3\}$, pois $[Mut(Z)]^{\delta} = Z$

$[Mut(3)]^1 = \{1,2,3,7\}$, união do resultado anterior com o conjunto de mutação.

$[Mut(3)]^2 = \{0,1,2,3,5,6,7\}$, união do conjunto anterior com os valores de mutação discreta de cada um dos elementos do conjunto anterior.

A **reprodução natural discreta** é calculado utilizando:

$$Cross(n_i, n_j) = \{z \in Q^t | \forall s \geq 0, s < t, Q^t \neq \emptyset, Q^s = \emptyset\} \quad (3.4)$$

onde $Q^t = [Mut(n_i)]^t \cap [Mut(n_j)]^t$, para todo $t \geq 0$.

Para exemplificar, sejam 1 e 6 os valores que participarão da reprodução. O cálculo segue abaixo:

$[Mut(1)]^0 = \{1\}$
$[Mut(6)]^0 = \{6\}$
$Logo, Q^0 = \emptyset$
$[Mut(1)]^1 = \{0,1,3,5\}$
$[Mut(6)]^1 = \{2,4,6,7\}$
$Logo, Q^1 = \emptyset$
$[Mut(1)]^2 = \{0,1,2,3,4,5,7\}$
$[Mut(6)]^2 = \{0,2,3,4,5,6,7\}$
$Logo, Q^2 \neq \emptyset$
$Cross(1,6) = \{0,2,3,4,5,7\}$

Desta forma, esta operação de reprodução gera seis descendentes.

Valores discretos			Codificação natural
preto	verde	azul	
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
⋮	⋮	⋮	**⋮**
1	1	0	**6**
1	1	1	**7**

Tabela 3.2: *Codificação de valores discretos em números naturais*

Limite	56	61	63	66	70
55	**1**$\equiv [55,56]$	**2**$\equiv [55,61]$	**3**$\equiv [55,63]$	**4**$\equiv [55,66]$	**5**$\equiv [55,70]$
56	-	**7**$\equiv [56,61]$	**8**$\equiv [56,63]$	**9**$\equiv [56,66]$	**10**$\equiv [56,70]$
61	-	-	**13**$\equiv [61,63]$	**14**$\equiv [61,66]$	**15**$\equiv [61,70]$
63	-	-	-	**19**$\equiv [63,66]$	**20**$\equiv [63,70]$
66	-	-	-	-	**25**$\equiv [66,70]$

Tabela 3.3: *Números naturais associados com intervalos de dados*

As operações de **transições linha e coluna** são necessárias para o cálculo da mutação natural contínua e da reprodução natural contínuo. São definidas através das equações abaixo:

$$\begin{aligned} left(n) &= max(leftb(n), n-1) \\ right(n) &= min(rightb(n), n+1) \\ upper(n) &= max(upperb(n), n-k+1) \\ lower(n) &= min(lowerb(n), n+k-1) \end{aligned} \tag{3.5}$$

(3.6)

onde $leftb(n)$, $rightb(n)$, $upperb(n)$ e $lowerb(n)$ são os limites à esquerda, à direita, acima e abaixo respectivamente. Por exemplo, na Tabela 3.3, o número 9 possui os seguintes limites:

$$\begin{aligned} leftb(9) &= 7 \\ rightb(9) &= 10 \\ upperb(9) &= 4 \\ lowerb(9) &= 19 \end{aligned}$$

Ainda são definidos os conjuntos $hor(n)$, que representa todos os números da linha que contém n na Tabela 3.3 e $ver(n)$, que são todos os números da coluna que contém n. As equações são definidas da seguinte forma.

$$\begin{aligned} hor(n) &= \bigcup_{i=1}^{k-1} max(leftb(n), (k-1)(row(n)-1)+i) \\ ver(n) &= \bigcup_{i=1}^{k-1} min(lowerb(n), (k-1)(i-1)+col(n)) \end{aligned} \quad (3.7)$$

para exemplificar, $hor(9) = \{7,8,9,10\}$ e $ver(9) = \{4,9,14,19\}$.

Mutação natural contínua é definida com a seguinte Equação:

$$Mut(n) \in \{x | x \in (mov(n) - \{n\})\} \quad (3.8)$$

onde $mov(n) = left(n) \cup right(n) \cup upper(n) \cup lower(n)$. Para exemplificar, $Mut(14) = \{9,13,15,19\}$, ou seja todos os valores vizinhos de 14 na Tabela 3.3.

O *Crossover* **natural contínuo**:

$$Cross(n_i, n_j) \in ((hor(n_i) \cap ver(n_j)) \cup (hor(n_j) \cap ver(n_i))) \quad (3.9)$$

A Figura 3.3 ilustra as operações de mutação e reprodução em atributos contínuos da Tabela 3.3. Por exemplo, os descendentes possíveis da operação de reprodução entre 5 e 14, calculado pela Equação 3.9, são 4 (resultado da intersecção da linha que contém 5 com a coluna que contém 14) e 15 (resultado da intersecção da coluna que contém 5 com a linha que contém 14).

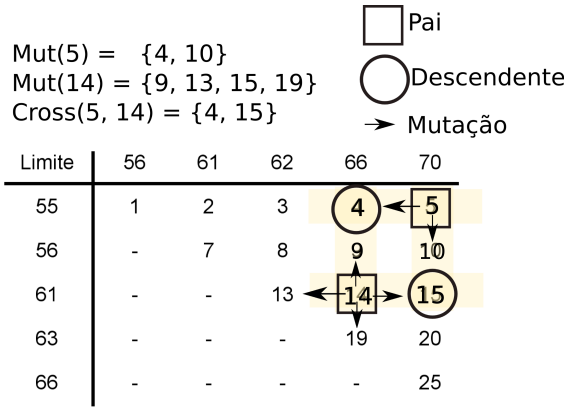


Figura 3.3: *Mutação e reprodução em atributos com valores contínuos*

O algoritmo HIDER* foi comparado[2] com outro algoritmo, o C4.5 [22] e o resultado foi que as regras extraídas com o HIDER* apresentaram as mesmas taxas de

erro que o equivalente C4.5. A vantagem do HIDER* [2] se mostrou no número de regras, que foi significativamente menor que o C4.5. Esta é uma grande vantagem pois quanto menor a quantidade de regras em um FRBS, mais rápida é sua execução, o que favorece sua utilização em sistemas de tempo real.

3.3 Algoritmo NSGA-II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [12] é um AG multiobjetivo que implementa o conceito de dominância. Neste conceito, os melhores indivíduos são os dominantes. É dito que uma solução domina outra quando esta possui, pelo menos, um aspecto melhor e os demais aspectos não são piores que os da outra. Utilizando este conceito de dominância, o NSGA-II classifica os indivíduos em frentes de dominância de tal forma que, os indivíduos da frente n possuem dominância em relação aos indivíduos da frente n + 1. Uma frente que não é dominada por nenhuma outra, ou seja, possui as melhores soluções, é chamada de frente de Pareto [12].

Inicialmente, o NSGA-II cria, aleatoriamente uma população P_0 de tamanho N. Esta população é ordenada seguindo o conceito de dominância. Sendo assim, para cada indivíduo, é atribuído um ranking igual ao seu nível de dominância (o nível n é melhor que o nível n + 1).

Após este passo, é criada a população Q_0, de tamanho N, através de mutações e *crossover's* sobre P_0. Depois, R_0 é criado tal que $R_0 = P_0 \cup Q_0$. Desta forma, a população R_0 tem tamanho 2N. Esta nova população é ordenada, seguindo o conceito de dominância. Uma nova população P_1 é gerada e populada a partir de todos os indivíduos das frentes $F_1, F_2, \ldots, F_j$, nesta ordem, enquanto o tamanho desta nova população for menor que N.

A Figura 3.4 ilustra esta operação. Para completar P_0 até o tamanho N, a última frente, que na Figura é representada como F_3, é ordenada pela distância entre as aglomerações, ou seja, indivíduos mais isolados são melhores. Uma nova população P_1 é formada. Este processo continua até a convergência do algoritmo.

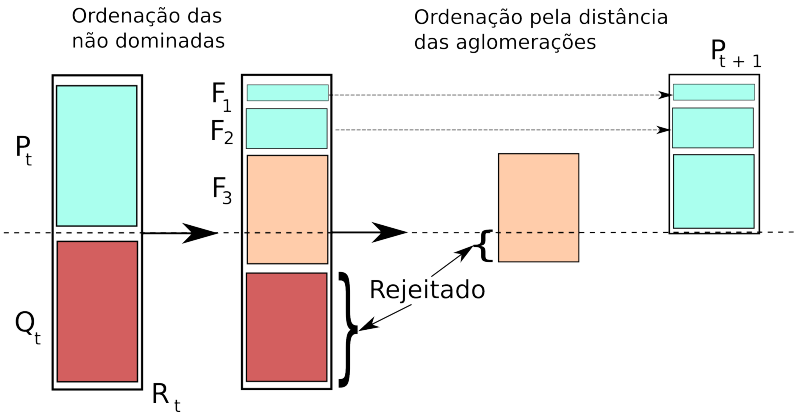


Figura 3.4: *Algoritmo NSGA-II [12]*

Pulkkinen e Koivisto [28] propuseram um método de classificação híbrido que utiliza o NSGA-II em conjunto com um outro algoritmo, o C4.5 [22]. O objetivo é extrair Regras *Fuzzy* a partir de dados. Conforme ilustrado na Figura 3.5, inicialmente o método utiliza o algoritmo C4.5 para extrair Regras *Fuzzy* a partir de uma base de dados. Após este passo, outras regras são criadas apenas alterando, aleatoriamente, alguns parâmetros das regras extraídas no passo anterior. Logo em seguida, é aplicado o algoritmo NSGA-II com o objetivo de escolher as melhores regras que cumpram dois objetivos. Primeiro, minimizar o número de regras. Segundo, minimizar o número de condições de cada regra. Por se tratar de objetivos concorrentes, o autor propôs a utilização do NSGA-II, que é um algoritmo multiobjetivo capaz de lidar com esta situação.

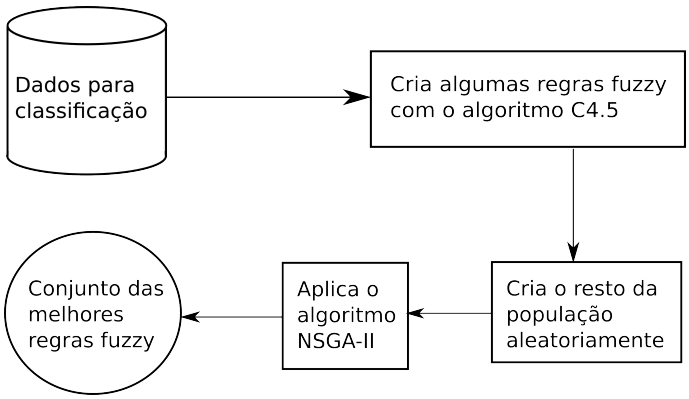


Figura 3.5: *Método de classificação híbrido proposto por Pulkkinen e Koivisto[28]*

3.4 Aplicação de Algoritmos Genéticos na otimização de recursos humanos

Segundo Yang et Al [40] a alocação de recursos humanos é um processo de decisão que consiste em combinar as habilidades de cada pessoa com os diferentes tipos de trabalhos existentes em uma determinada empresa. Neste processo de tomada de decisão, devem ser considerados tanto os interesses da pessoa, que ocupará a vaga, como os interesses da empresa.

O quadro de funcionários das empresas é composto por equipes de diferentes habilidades, qualidades e posição. Yang et Al. [40] apresentam um modelo de escolha de “dois sentidos”. Neste modelo, são consideradas duas avaliações. A primeira é feita a partir do ponto de vista da empresa e por isto leva em consideração os objetivos organizacionais. A segunda é feita do ponto de vista do funcionário levando em consideração seus objetivos pessoais. A equação 3.10 apresenta o modelo matemático, proposto por Yang et Al. [40], que representa este problema.

$$maxz(x) = \sum_{i=1}^{n} \sum_{j=0}^{m} (E_i * p_{ij} * q_{ij} * x_{ij}) \quad (3.10)$$

onde:

- i - indice do posto de trabalho, $i = 1, 2, \ldots, n$;
- j - indice do empregado, $i = 1, 2, \ldots, m$;
- E_i - grau de importância do $i - simo$ posto;
- n - quantidade total de postos de trabalho;
- m - quantidade total de empregados ;
- p_{ij} - pontuação referente à avaliação do posto de trabalho pelo empregado quando este conhece o mesmo;
- q_{ij} - pontuação referente à avaliação do empregado para a ocupação de determinado cargo na empresa.
- x_{ij} - é calculado da seguinte forma: $\begin{cases} 1, & \text{se o empregado j for alocado ao posto i} \\ 0, & \text{se não houver atribuição} \end{cases}$

O cromossomo do algoritmo genético proposto por Yang et Al [40] é estruturado da seguinte forma. Cada gene representa a alocação de um empregado a um determinado posto de trabalho. Os seus valores são binários, 0 ou 1. O valor 0 indica empregado não alocado enquanto que o valor 1 indica empregado alocado. Por exemplo, se a empresa espera alocar 10 funcionários a 3 postos de trabalho, um cromossomo válido é $C = [0101001010|1010100000|0000010101]$. Nota-se uma divisão em três partes com 10 genes cada. Cada uma destas partes representam um posto de trabalho. Desta forma, os empregados 2, 4, 7 e 9 estão alocados no primeiro posto de trabalho. Os empregados 1, 3 e 5 no segundo posto e os empregados 6, 8 e 10 no terceiro.

No trabalho de Yang et Al [40], a população inicial foi gerada aleatoriamente, a reprodução foi realizado em apenas um ponto do cromossomo e foi empregada uma mutação básica, onde apenas um gene aleatório é alterado. Após vários experimentos, os parâmetros de entrada que apresentaram melhor resultado foram:

- Tamanho da população: 60;
- Probabilidade de reprodução: $0,7$;
- Probabilidade de mutação: $0,05$.

Os experimentos foram realizados com base em uma situação real encontrada em uma grande empresa. Os resultados destes experimentos foram avaliados por um especialista, no caso o gerente desta empresa, e foram considerados muito satisfatórios [40].

3.5 Modelagem Fuzzy para gerenciamento de recursos em Sistemas de Banco de Dados Virtualizados

Segundo Wang et Al [35], a hospedagem de bancos de dados em máquinas virtuais (VMs) tem um grande potencial para melhorar a eficiência de utilização dos recursos e a facilidade de implantação de sistemas de banco de dados.

O trabalho de Wang et Al [35] considera o problema de alocação de recursos sob demanda para uma máquina virtual rodando um banco de dados que serve dados para uma grande quantidade de consultas dinâmicas e complexas e ainda, atendendo aos requisitos de QoS (*Quality of Service*[6]).

Para resolver este problema, foi proposta uma abordagem de gerenciamento de recursos autônoma. Esta abordagem utiliza um modelagem *Fuzzy* para capturar e avaliar o comportamento de uma VM que hospeda um banco de dados. O objetivo é tomar decisões referentes às necessidades de recursos para cada máquina virtual, prevendo suas necessidades e alterando dinamicamente as cargas de trabalho. A solução proposta foi implementada no Xen[7]e os resultados demonstraram que a CPU e o I/O de disco podem ser eficientemente alocados para as VMs que executam os banco de dados com a finalidade de cumprir os objetivos de QoS.

A Figura 3.6 ilustra a arquitetura do sistema de gerenciamento de recursos proposta para bancos de dados virtualizados com base na modelagem *Fuzzy* acima. Esta arquitetura consiste em quatro módulos principais. Os Sensores monitoram a carga de trabalho da VM, representada por $w(t)$, o seu desempenho, representado por $p(t)$, e o uso de recursos da VM $r(t)$.

[6] A qualidade do serviço (QoS) refere-se a vários aspectos que permitem avaliar a qualidade do transporte de informações em redes de computadores e em sistemas telefônicos.

[7] Xen é um software livre utilizado para virtualização das arquiteturas x86, x86-64, IA-32, IA-64 e PowerPC. O Xen permite a execução de vários sistemas operacionais, simultaneamente, sobre um mesmo hardware.

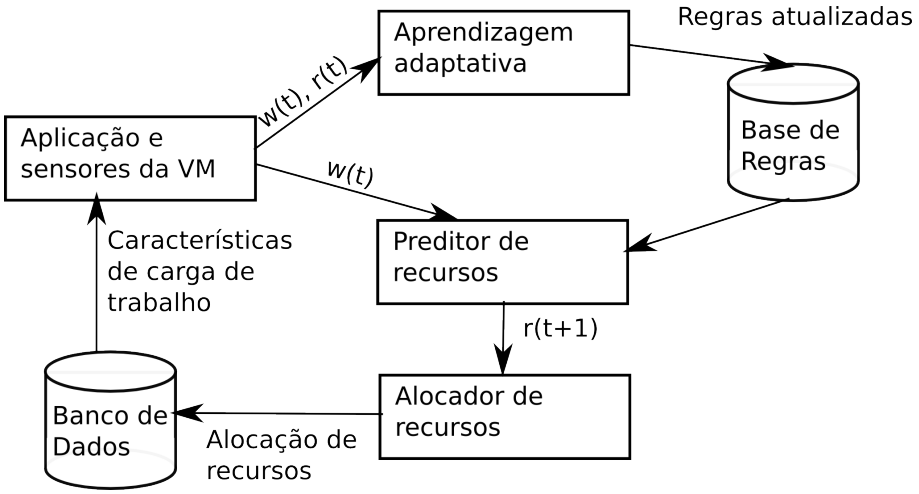


Figura 3.6: *Arquitetura do sistema de gerenciamento de recursos [35].*

O módulo de Aprendizagem Adaptativa gera regras Fuzzy do tipo *if ... then* a partir dos dados agrupados para a modelagem da VM de banco de dados. Para exemplificar, suponha que a carga de trabalho para a entrada $w(t)$ é descrita por N diferentes características, $[C1, C2, ..., CN]$ e para a saída, dois tipos de recursos, CPU e I/O, $[R_{CPU}, R_{IO}]$, são consumidos. Se os clusters K são formados a partir de todos os pares de dados, então as regras K são produzidos para este modelo Fuzzy. A base de regras é construído da seguinte forma:

R_i: *If input* $[C1, C2, ..., CN]$ *is in cluster i, then output* $[R_{CPU}, R_{IO}]^T = A_i[C1, C2, ..., CN]^T + b_i$

As regras são armazenadas na Base de Regras. Após isto, o Preditor de Recursos realiza inferência *Fuzzy* para gerar uma estimativa precisa do recurso r, dada a carga de trabalho de entrada w. É utilizada uma função de pertinência Gaussiana para a fuzzificação da entrada w. Após o processamentos das regras e a defuzzificação, o resultado final é obtido. Este resultado é então enviado ao Alocador de Recursos para orientar a alocação de recursos da VM.

Wang et Al [35] realizaram alguns experimentos para verificar a abordagem sugerida. Em um destes experimentos, a carga de trabalho foi alterada conforme apresenta a Figura 3.7.

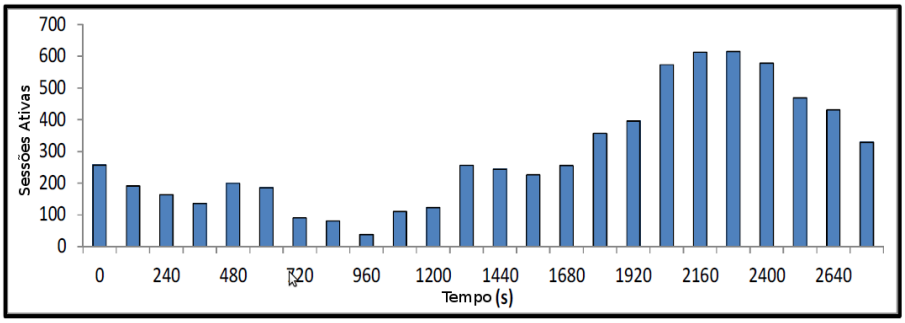


Figura 3.7: *Alteração da carga realizada nos experimentos [35].*

As Figuras 3.8 e 3.9 mostram a utilização de CPU da VM que hospeda o banco de dados. Nestas Figuras, é possível notar aumento de carga nos tempos $480s, 1450s e 1930s$ e que, no tempo $2160s$ ococorre um pico na demanda por CPU. Neste momento ocorre uma alocação automática de recursos. A Figura 3.8 ilustra como a CPU ficou estável neste momento. A Figura 3.9 mostra o tempo médio de resposta a cada consulta realizada no banco de dados e que, apesar do pico existente no tempo $2100s$, o tempo médio ultrapassou o QoS apenas sete vezes.

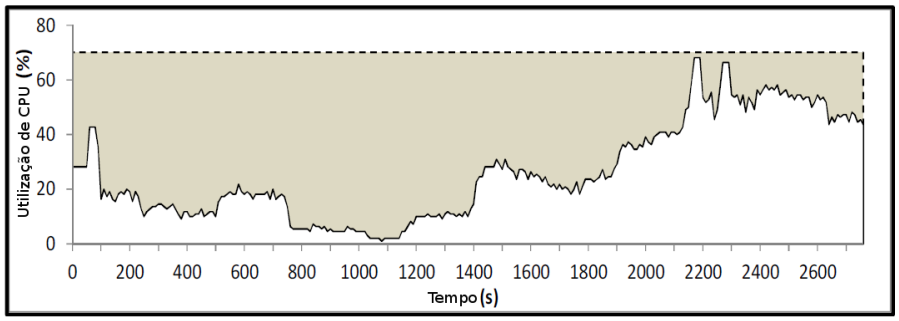


Figura 3.8: *Utilização de CPU em função do tempo [35].*

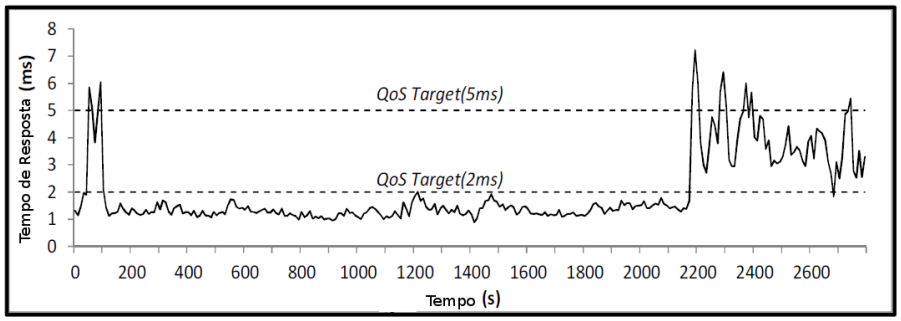


Figura 3.9: *Tempo médio de resposta em função do tempo [35].*

3.6 Discussão

Atualmente existem diversos estudos relacionados com a extração do conhecimento, a partir de uma base de dados, com o objetivo de gerar regras na forma

if ... then. Este capítulo apresentou alguns destes estudos cujo objetivo principal foi o desenvolvimento de algoritmos capazes de realizar tal tarefa. Neste sentido, foram apresentados os algoritimos TARGET na Seção3.1, HIDER na Seção3.2 e NSGA II na Seção3.3. Apesar de diversos outros algoritmos terem sido encontrados na literatura, estes apresentaram-se mais semelhantes com a proposta deste trabalho.

Além dos algoritmos utilizados na extração do conhecimento, hoje existem também, diversas abordagens que empregam técnicas da Inteligência Artificial na alocação dinâmica de recursos. Duas foram citadas neste capítulo. A abordagem apresentada na Seção 3.4 utiliza Algoritmos Genéticos para alocar dinamicamente pessoas a cargos em uma empresa. A abordagem da Seção 3.5 utiliza Lógica *Fuzzy* para alocar dinamicamente recursos de memória e CPU para uma máquina virtual que hospeda um banco de dados. Conforme será visto no próximo capítulo, este projeto empregará algumas destas técnicas, combinadas com SMA, com o objetivo de alocar dinamicamente recursos de qualquer natureza, desde recursos humanos até recursos de máquina.

CAPÍTULO 4

O projeto COGNARE

Este Capítulo apresenta o projeto COGNARE. Trata-se de um projeto cujo objetivo é realizar a alocação dinâmica de recursos[1]. O problema da alocação dinâmica de recursos pode ser enunciado da seguinte forma: "Dada uma lista de recursos disponíveis R e uma lista de tarefas a serem executadas T, retornar as duplas <T, R>[2] que possam maximizar parâmetros previamente informados."

Com o objetivo de obter uma melhor organização das funcionalidades, o projeto COGNARE foi dividido em três módulos. O primeiro utiliza os SMAs, apresentados na Seção 2.3 para a obtenção de informações do ambiente externo[3]. O segundo módulo utiliza AG, apresentados na Seção 2.2, para realizar a aprendizagem de regras *Fuzzy*, utilizadas no processo de alocação de recursos. O terceiro, e último módulo, utiliza um FRBS, apresentado na Seção 2.1, para realizar a alocação dinâmica das tarefas a recursos, oriundos do ambiente exterior.

Este Capítulo está organizado da seguinte forma. Na Seção 4.1 é apresentada a arquitetura empregada no projeto, os módulos e como cada um destes módulos se relacionam. As Seções 4.2, 4.3 e 4.4 descrevem cada um dos módulos apresentados na arquitetura. No final de cada uma destas seções são descritos os detalhes de utilização destes módulos. Para encerrar este capítulo, a Seção 4.5 tece uma discussão a respeito da arquitetura e dos módulos propostos para o projeto COGNARE.

O COGNARE não possui interfaces gráficas. A sua utilização se dá como um módulo de outros sistemas que necessitam de alocar dinamicamente recursos. Para isto foram disponibilizados métodos públicos capazes de executarem as funções propostas neste projeto. A utilização do módulo de aprendizagem é descrita na Seção 4.3.4 e a utilização do módulo de alocação dinâmica de recursos é apresentada na Seção 4.4.1.

[1] Dependendo da situação, esta alocação pode ser apenas uma sugestão para que um usuário possa avaliar o seu efetivo cumprimento. Em outras situações, esta alocação é feita diretamente, sem a necessidade da avaliação deste usuário. Estes dois casos foram verificados no Capítulo 5.

[2] A dupla <T, R> representa uma alocação da tarefa T ao recurso R.

[3] É chamado de ambiente externo qualquer outro sistema gerador de tarefas e recursos. Estas tarefas e recursos são as entradas utilizadas pelo COGNARE para a sugestão de alocações.

4.1 Arquitetura do COGNARE

O objetivo da Arquitetura de Software é definir os componentes de um sistema, suas propriedades externas e seu relacionamento com seres humanos, demais sistemas e sensores [32]. A arquitetura utilizada no projeto COGNARE foi ilustrada na Figura 4.1. Nesta Figura, é possível visualizar os três módulos principais do projeto:

- Agentes Sensores - possuem dois objetivos. O primeiro é captar os eventos externos, variáveis e demais informações necessárias e armazená-las em um repositório local. O segundo objetivo é captar as situações dos recursos e tarefas a serem alocados. Estes dados são provenientes do ambiente exterior e são enviados para o Alocador *Fuzzy* fazer a sugestão de alocação de recursos;
- Aprendizagem de Regras - Lê as informações armazenadas no repositório local e gera um conjunto de regras *Fuzzy*. Estas regras são armazenadas na biblioteca de regras;
- Alocador *Fuzzy* - Utiliza as regras armazenadas na biblioteca de regras e os valores do ambiente exterior oriundos dos agentes sensores com o objetivo de produzir sugestões de alocação de recursos.

Esta arquitetura visa atender a dois objetivos centrais. O primeiro objetivo é gerar regras *Fuzzy* a partir de dados obtidos do ambiente exterior. O segundo é realizar a sugestão de alocação de tarefas a recursos a partir de variáveis obtidas do ambiente exterior. Cada um destes módulos da Figura 4.1 estão descritos de forma detalhada nas próximas Seções.

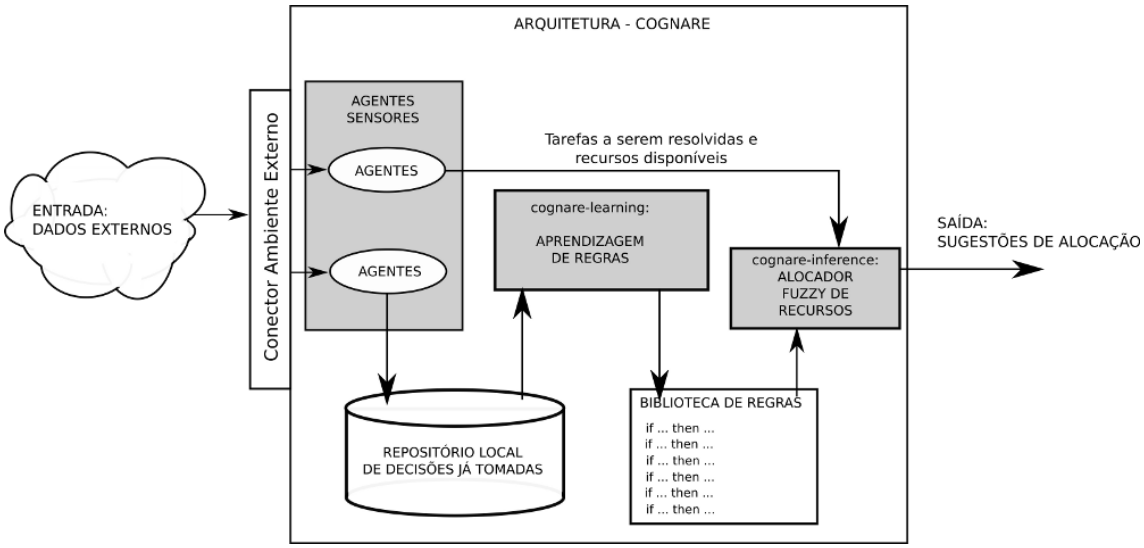


Figura 4.1: *Arquitetura do sistema COGNARE.*

Na Figura 4.1, também é possível notar um elemento denominado “Conector Ambiente Externo”. Este conector é uma interface, modelada com o nome

ExternalEnviromentConnector, responsável por converter os dados do ambiente externos em informações úteis para o COGNARE. A Figura 4.2 apresenta o diagrama desta interface com os seguintes métodos:

- *getResources()* - Retorna a lista de recursos disponíveis a realizarem tarefas;
- *getTasks()* - Retorna a lista de tarefas que necessitam serem realizadas;
- *getTaskAssigned(Resource resource)* - Retorna a tarefa atribuída a um recurso passado como parâmetro para este método.

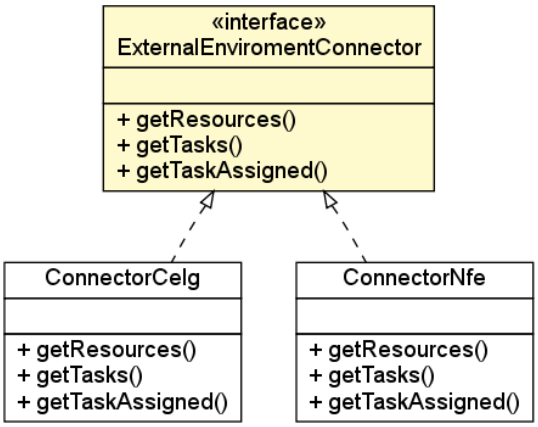


Figura 4.2: *Interface ExternalEnviromentConnector e seu relacionamento com as classes ConnectorCelg e ConnectorNfe.*

Nota-se também, na Figura 4.2 as classes *ConnectorCelg* e *ConnectorNfe*. Estas, são classes concretas, implementadas a partir da interface *ExternalEnviromentConnector*. Estas classes serão utilizadas nos estudos de caso apresentados no capítulo 5. Desta forma, o COGNARE consegue abstrair os tipos de representação das informações provenientes de fontes de dados distintas, uma para cada estudo de caso. Por isso é possível integrar o COGNARE com outras fontes de dados, bastando para isto apenas implementar os métodos da interface *ExternalEnviromentConnector*. Estes métodos deverão conter o código responsável por converter esta nova fonte de dados externa para instâncias das classes *Resource* e *Task*, conhecidas pelo COGNARE.

4.2 Módulo: Agentes Sensores

Este módulo é responsável por integrar os dados externos com o COGNARE e executar as funções implementadas nos demais módulos com o objetivo de tomar decisões quanto à alocação de recursos. Para isto é utilizado um Sistema Multiagente.

O SMA foi modelado utilizando a metodologia Prometheus [38]. Esta metodologia consiste em três fases, que estão apresentadas na Seção A.3.

Os diagramas da metodologia Prometheus foram criados utilizando a ferramenta *Prometheus Design Tool*(PDT)[4] [34]. A implementação dos agentes foi feita utilizando a plataforma JADE[5] (*Java Agent Development Framework*) [5]. Esta plataforma utiliza a arquitetura BDI (*Belief-Desire-Intention*), apresentada na Seção 2.3.1.

A seguir, serão detalhadas essas três fases da metodologia Prometheus, referente à implementação dos agentes utilizados no COGNARE.

4.2.1 Especificação do Sistema

Os agentes do COGNARE cumprem os três objetivos:

- O_1: Obter as listas de tarefas e recursos a serem atribuídos;
- O_2: Decidir quanto à atribuição de uma tarefa a um recurso;
- O_3: Descobrir qual foi a decisão tomada por um usuário quanto à alocação de uma tarefa a um recurso. Este objetivo é necessário somente quando o COGNARE atuar como ferramenta auxiliar na alocação. Neste caso será realizada apenas uma sugestão, que poderá ser acatada ou não por um usuário. Esta informação será utilizada em uma futura avaliação da qualidade regras e, consequentemente, das sugestões fornecidas.

O objetivo O_1, referente à obtenção das listas, consiste na execução dos métodos *getResources()* e *getTasks()* da *interface ExternalEnviromentConnector*, ilustrada na Figura 4.2. Após a obtenção destas listas, é verificado se estas sofreram alguma alteração. Esta alteração significa que surgiram novas tarefas ou novos recursos, ou ainda mais, alguma tarefa ou recurso deixou de existir. Toma-se este cuidado para diminuir a demanda para o módulo que faz a alocação e, desta forma, aumentar a performance do sistema.

O objetivo O_2, referente à realização da sugestão, consiste na invocação do módulo alocador. Este módulo será descrito na Seção 4.4. Desta forma, este objetivo consiste em obter a lista de tarefas e recursos, descrita no parágrafo anterior e retornar pares *Tarefa x Recursos* que são as efetivas alocações realizadas pelo sistema.

O objetivo O_3, referente à descoberta das decisões tomadas pelo usuário, consiste na execução do método *getTaskAssigned(Resource resource)* da *interface ExternalEnviromentConnector*, passando como parâmetro um recurso. A função deste método é retornar a tarefa que foi atribuída pelo usuário a este recurso. A necessidade deste objetivo se dá somente quando o COGNARE for utilizado como sistema auxiliar, ou seja, quando este apenas fornecer sugestões de alocação para um usuário.

[4]Esta ferramenta é descrita na Seção A.3.
[5]Esta plataforma é descrita na Seção A.2

Após a definição dos objetivos, que o SMA deverá cumprir, o próximo passo, na metodologia Prometheus, consiste na definição do Projeto Arquitetural[38]. Esta etapa será descrita a seguir.

4.2.2 Projeto Arquitetural

Nesta etapa é realizado o agrupamento dos objetivos similares com a finalidade de definir as funcionalidades. Estas funcionalidades podem ser avaliadas em função dos critérios de acoplamento e coesão definidos na Engenharia de Software[38]:

- Acoplamento: é apresentado principalmente na comunicação entre agentes, mas também pode ser apresentado no uso compartilhado de dados;
- Coesão: capacidade de controlar várias ações que ocorrem ao mesmo tempo e a forma como objetivos do agente são relacionados.

Após o agrupamento dos objetivos, são gerados dois diagramas. O primeiro é o Diagrama de Acoplamento de Dados, conforme ilustrado na Figura 4.3. Este diagrama é uma ferramenta utilizada para avaliar o acoplamento e a coesão entre as funcionalidades do SMA. O segundo é o diagrama de agrupamento de funcionalidades, conforme ilustrado na Figura 4.4. Este diagrama tem como objetivo relacionar os objetivos esperados pelo SMA com os agentes propriamente ditos.

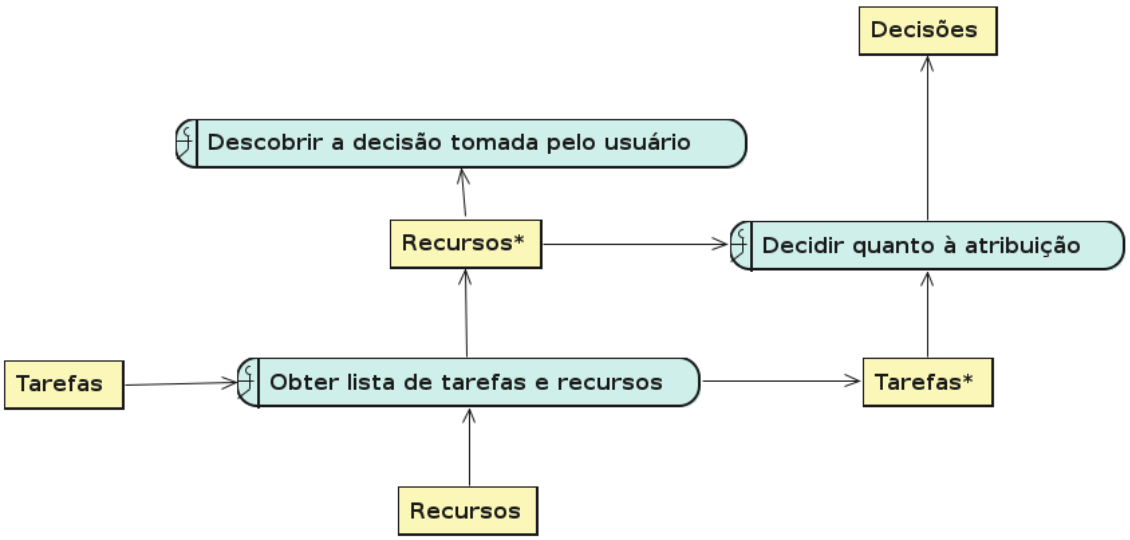


Figura 4.3: *Diagrama de Acoplamento de Dados.*

Na Figura 4.3 está ilustrada as duas fontes de dados externas: *Tarefas* e *Recursos*[6], e como estas se relacionam com as funcionalidades dos agentes. A

[6]Estes dados são obtidos através da execução dos métodos *getResources()* e *getTasks()* da *interface ExternalEnviromentConnector*.

funcionalidade “Obter lista de tarefas e recursos” é responsável pela criação dos dados *Tarefas** e *Recursos**[7]. Estes dados são as listas que participarão do processo de alocação. Este processo de alocação é representado na Figura 4.3 pela funcionalidade “Decidir quanto à atribuição” e é responsável pela criação do dado *Decisões*. Este dado consiste na lista de decisões propriamente dita.

Também nota-se na Figura 4.3 que o dado *Recursos** é compartilhado pelas funcionalidades “Decidir quanto à atribuição” e “Descobrir a decisão tomada pelo usuário”. Isto é necessário para orientar a funcionalidade “Descobrir a decisão tomada pelo usuário” na descoberta das decisões tomadas pelo usuário. Neste caso, esta funcionalidade “aguarda um tempo”[8], referente à espera pela tomada de decisão do usuário e, após este tempo, percorre a lista de *Recursos** com o objetivo de consultar as reais atribuições realizadas por este usuário[9].

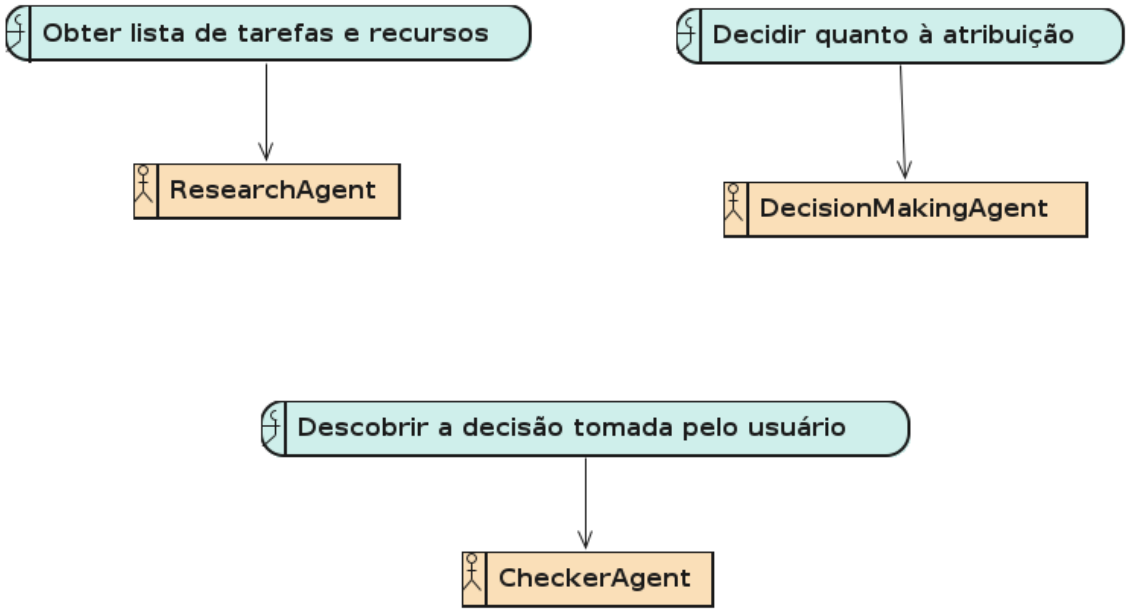


Figura 4.4: *Diagrama de Agrupamento de Funcionalidades*

Na Figura 4.4 são apresentados os mapeamentos entre as funcionalidades do SMA com os agentes. Conforme ilustrado nesta Figura, o agente “*ResearchAgent*” é responsável pela funcionalidade “Obter lista de tarefas e recursos”, o agente “*DecisionMakingAgent*” é responsável pela funcionalidade “Decidir quanto à atribuição”

[7]Nota-se um asterisco nas fontes de dados *Tarefas** e *Recursos**. Isto indica que esta lista foi alterada, em relação à lista original, sem o asterisco. Esta alteração é realizada pela funcionalidade “Obter lista de tarefas e recursos”.

[8]Este tempo é um dos parâmetros de configuração do sistema. Durante os testes foi considerado 5 minutos.

[9]Para realizar esta consulta é executado o método *getTaskAssigned(Resource resource)* da *interface ExternalEnviromentConnector*

e o agente "*CheckerAgent*" é responsável pela funcionalidade "Descobrir decisão tomada pelo usuário".

Uma vez tendo identificados os agentes, a próxima etapa, definida pela metodologia Prometheus, consiste na criação do Projeto Detalhado[38]. Esta etapa será descrita a seguir.

4.2.3 Projeto Detalhado

É nesta etapa que o diagrama de visão geral do SMA é gerado. Este diagrama, conforme ilustrado na Figura 4.5, apresenta o relacionamento entre os agentes, juntamente com suas ações, papéis, dados, mensagens, cenários e objetivos[38].

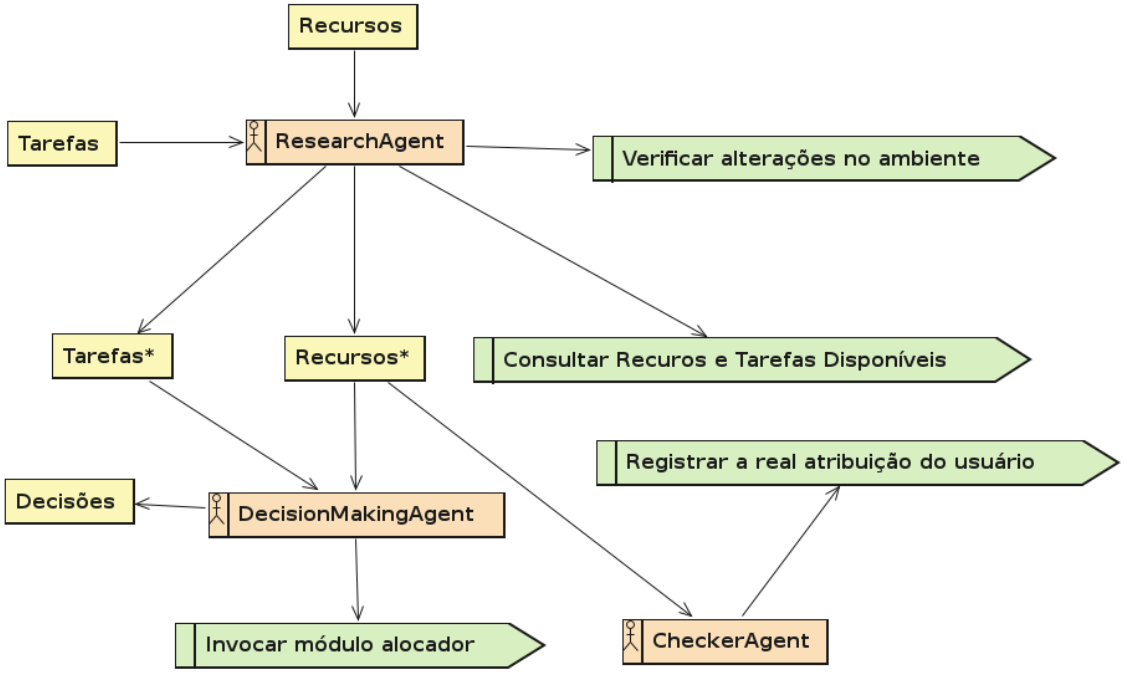


Figura 4.5: *Diagrama da visão geral do SMA.*

Nesta Figura, é possível visualizar como os agentes do SMA interagem e trocam dados. Também é possível identificar as ações executadas por cada agente.

4.2.4 Implementação do SMA

Uma vez definido o projeto do SMA, o próximo passo é implementá-lo. Neste projeto foi utilizada a plataforma JADE [5] para a criação dos agentes.

A plataforma JADE possui a classe *Agent* que deve ser estendida por todos os agentes, conforme ilustrado no diagrama da Figura 4.6. Neste diagrama os agentes estão no pacote destacado com a cor cinza. Os demais elementos foram inseridos para contextualizar as classes dos agentes dentro do sistema e serão detalhados posteriormente.

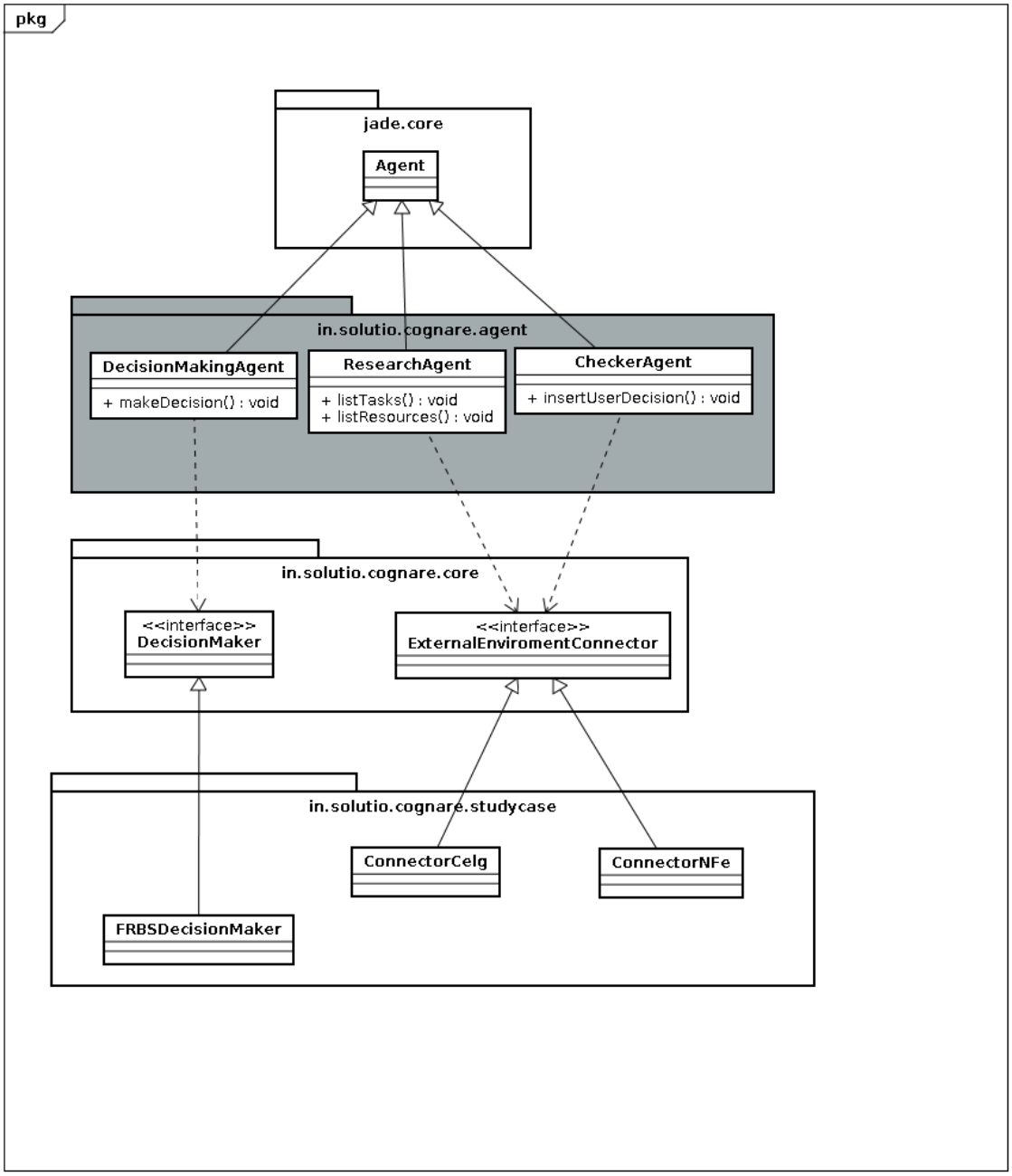


Figura 4.6: *Diagrama de classe dos agentes implementados.*

4.2.5 Utilização do módulo Agentes e Sensores

Para utilizar este módulo, inicialmente é necessário colocar o arquivo *cognare-agent.jar* nos computadores que executarão os agentes. É necessário ter a Máquina Virtual Java (JVM) instalada. Nestes computadores, devem ser criados, no diretório do usuário[10], um arquivo de propriedades, com o nome *cognare-agent.properties*

[10]O sistema operacional (SO) possui um diretório específico para cada usuário. No caso do SO Linux, este diretório fica em */home/<usuario>* enquanto que no Windows XP fica em *C:/Document and*

cujo conteúdo está exemplificado na Listagem 4.1.

A função de cada uma destas propriedades é a seguinte. A propriedade *enviromentConnector* deve referenciar uma instância da interface *ExternalEnviromentConnector*, ilustrada na Figura 4.2. A propriedade *hostLearner* deve referenciar o endereço onde se encontra instalado o módulo de Aprendizagem de Regras, apresentado na Seção 4.3. A propriedade *hostInference* deve referenciar o endereço onde se encontra instalado o módulo Alocador *Fuzzy*, apresentado na Seção 4.4.

Listagem 4.1: *Propriedades configuradas para o módulo "Agentes e Sensores"*

```
1 enviromentConnector=in.solutio.cognare.core.util.ConnectorCelg
2 hostLearner=localhost
3 hostInference=localhost
```

A inicialização deste módulo acontece após a execução do seguinte comando:

```
java -cp cognare-agent.jar in.solutio.cognare.agent.Start
```

Após a execução deste comando os agentes são inicializados e começam a enviar informações para os demais módulos.

4.3 Módulo: Aprendizagem de Regras

Este módulo é responsável por processar os dados contidos no repositório local, conforme ilustrado na Figura 4.1, e convertê-los em regras *Fuzzy*. Com este objetivo, foi criada a interface *Learner*, que possui o objetivo de abstrair a técnica utilizada no aprendizado. Esta interface, ilustrada na Figura 4.7, possui o método *getKnowledge(List<Instance> instances)*, responsável por realizar o aprendizado das regras a partir de uma lista de instâncias. Este método recebe como parâmetro de entrada uma lista de objetos do tipo da classe *Instance*.

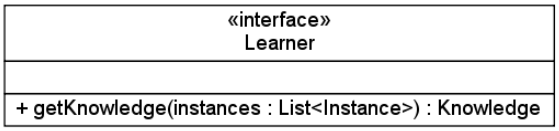


Figura 4.7: *Interface Learner*

A classe *Instance*, ilustrada na Figura 4.8, representa uma instância de decisão e possui o relacionamento "um para muitos" com a classe *Attribute*. Desta forma, cada objeto do tipo *Instance* contém vários atributos. Cada atributo possui um nome, um valor

Settings/<usuario>.

e um tipo, que pode ser entrada ou saída, conforme apresentado no diagrama da Figura 4.8.

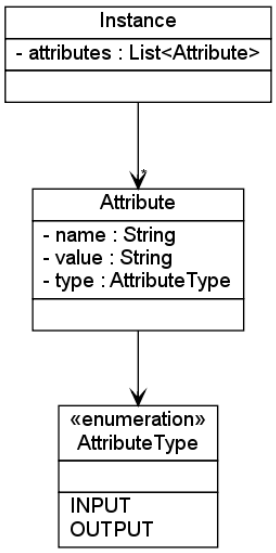


Figura 4.8: *Classes Instance, Attribute e AttributeType com seus relacionamentos.*

Para exemplificar, seja a Tabela 4.1[11]. Serão consideradas neste exemplo apenas as variáveis de entrada: "Distância" e "Prioridade". A variável de saída será a última coluna, o "Coeficiente de Decisão".

Nesse exemplo, cada linha da Tabela 4.1 será convertida em um objeto do tipo *Instance* e armazenada em uma lista. Esta lista será passada como parâmetro ao método *getKnowledge(List<Instance> instances)* da interface 4.7 que iniciará o aprendizado das regras.

Distância	Prioridade	...	Saída: Coeficiente de Decisão
3	0,7	...	altissimo
2	0,8	...	alto
10	0,5	...	normal
15	0,4	...	baixo

Tabela 4.1: *Tabela D gerada à partir do exemplo da Figura 4.21.*

O método *getKnowledge(List<Instance> instances)* retorna um objeto do tipo da classe *Knowledge*, após o processamento do aprendizado das regras. O diagrama desta classe está ilustrado na Figura 4.9. Neste diagrama, é possível identificar que *Knowledge* se relaciona com as classes *Rule* e *FuzzyTerm*.

[11] Nesta tabela, as reticências ilustram que é possível trabalhar com uma quantidade flexível de variáveis de entrada.

A classe *Rule* representa uma regra *Fuzzy*. A classe *FuzzyTerm* representa um conjunto Fuzzy. Conforme descrito na Seção 2.1.1. Neste projeto, foram utilizadas somente funções de pertinência triangulares[12]. Neste sentido, a classe *FuzzyTerm* possui os atributos "*min*", "*middle*" e "*max*" que representam os vértices do triângulo da função de pertinência.

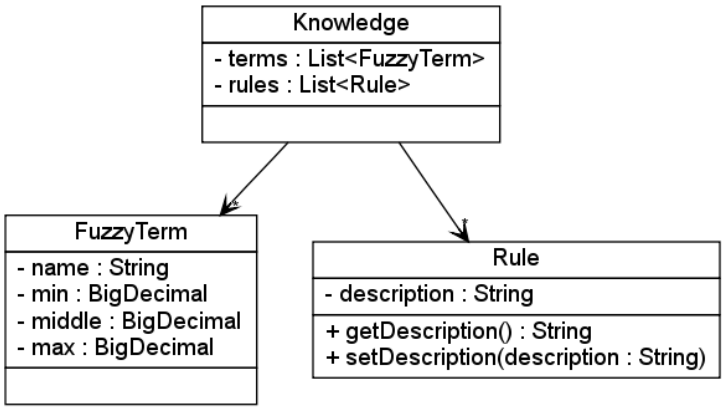


Figura 4.9: *Classes Knowledge, FuzzyTerm e Rule com seus relacionamentos*

Foi implementada a classe *SGERDLearner*, conforme ilustrado na Figura 4.10. Esta classe é uma implementação da *interface Learner*, capaz de gerar regras *Fuzzy* utilizando o algoritmo de aprendizagem evolutiva SGERD[24], apresentado na Seção 4.3.1. Portanto, para utilização deste módulo de aprendizagem com outros tipos de algoritmos de aprendizagem, basta implementar, para cada um deles, a interface *Learner* com o código necessário para a execução adequada desse algoritmo.

[12]O motivo da utilização de funções de pertinência triangulares se dá pela limitação do escopo deste projeto. Pode ser considerado um trabalho futuro estender os testes para os demais tipos funções de pertinência contidos na teoria dos conjuntos *Fuzzy*.

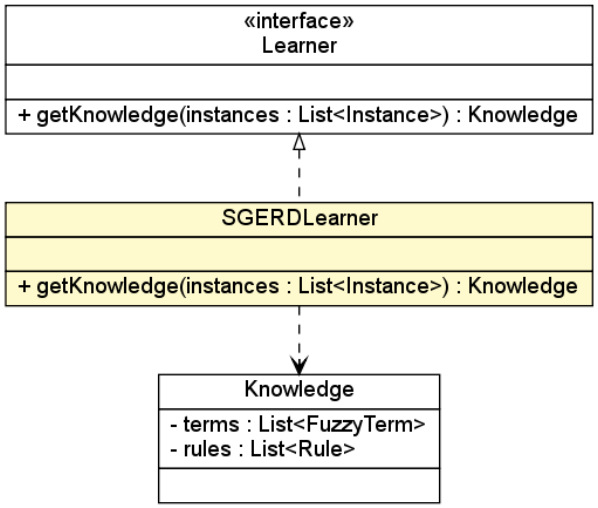


Figura 4.10: *Classes Decision, Task e Resource e seus relacionamentos*

Finalmente, foi implementada a classe *LearnerConverter*, capaz de converter um objeto do tipo *Knowledge* em uma *string* cujo conteúdo segue a sintaxe definida na linguagem FCL. Posteriormente, esta *string* será gravada em um arquivo e este arquivo será armazenado na biblioteca de regras, ilustrada na Figura 4.1. Esta biblioteca de regras será utilizada no módulo seguinte, para realização da tomada de decisão através de inferência *Fuzzy*.

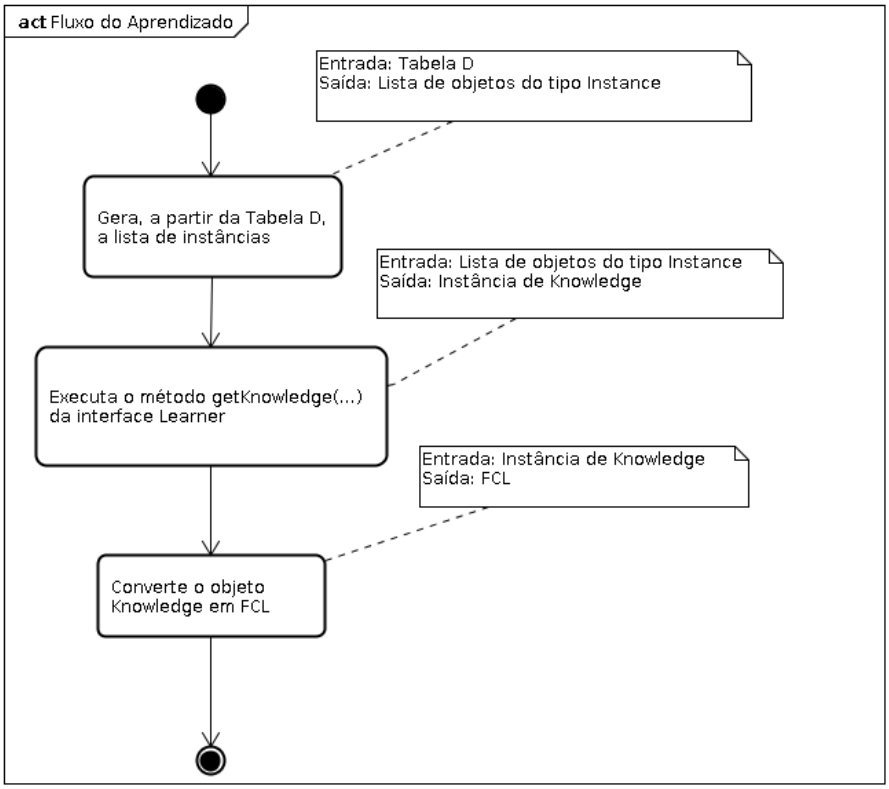


Figura 4.11: *Diagrama que ilustra o fluxo do aprendizado de regras no COGNARE*

O diagrama da Figura 4.11, ilustra todo o fluxo do aprendizado, descrito nesta Seção. Resumindo, este módulo recebe uma tabela (chamada de "Tabela D") contendo exemplos de classificações e retorna um arquivo, na linguagem FCL, com os conjuntos *Fuzzy*, as funções de pertinência e as regras *Fuzzy*. Este arquivo é armazenado na biblioteca de regras do COGNARE.

A seguir é apresentado, de forma mais detalhada, o processo de aprendizado executado pelo COGNARE.

4.3.1 Algoritmo SGERD

Este projeto utiliza o algoritmo *Steady-State Genetic Algorithm for Extracting Fuzzy Classification Rules From Data* (SGERD) [24] na aprendizagem das regras. Conforme apresentado na Tabela 4.2, o algoritmo SGERD apresentou-se eficiente em uma comparação com outros algoritmos capazes de realizar a classificação de dados através de regras. Conforme apresentado nesta tabela, o SGERD obteve uma menor taxa de erro na maioria das bases de dados testadas[13].

[13]Estas e outras bases de dados estão disponíveis no site da Universidade da Califórnia (http://archive.ics.uci.edu/ml/index.html), onde é hospedado o "Repositório de Aprendizagem de Máquina". Este repositório é largamente utilizado em diversos estudos relacionados com a aprendizagem de máquina.

Abordagem de Classificação			Base de Dados					
Referência		Abordagem	Glass	Wine	Cancer	Sonar	Iris	Pima
Quinlan	[22][29]	C4.5	32,50	-	5,26	25,60	4,80	25,40
Elomaa	[13]	C4.5	**27,70**	5,60	5,30	25,10	6,60	25,70
Sanchez	[30]	GP+SA	42,10	-	4,65	-	-	-
Ishibuchi	[20]	GBML	40,38	6,90	3,25	24,06	-	-
Ishibuchi	[21]	Hybrid GBML	34,63	4,94	3,32	23,70	5,33	**24,17**
Guan	[21]	Nonfuzzy+GA	-	8,33	4,70	-	4,40	-
Abonyi	[1]	DT+Fuzzy	33,97	8,78	3,18	-	3,89	26,95
Mansoori	[24]	SGERD	36,62	**3,81**	**2,98**	**22,80**	**3,07**	25,36

Tabela 4.2: *Comparação entre as taxas de erros do SGERD e outras abordagens utilizadas para classificação de dados [24].*

O SGERD é um AG de estado estacionário[14]que tem como objetivo a geração de um número Q de regras por classe, seguindo a abordagem *Genetic Cooperative-Competitive learning* (GCCL) [17]. As gerações, ou filhas, são finitas e limitadas à dimensão do problema. O algoritmo busca manter a mesma quantidade de Q regras para cada classe. As regras *Fuzzy if-then* que classificam um problema específico com n atributos são representadas na forma:

$$\textit{Regra } R_j: \text{ If } x_1 \text{ is } A_{j1} \text{ and } \ldots \text{and } x_n \textit{ is } A_{jn}, \text{ then class } C_j \qquad (4.1)$$

$$\text{para j = 1, 2, 3, } \ldots \text{N}$$

onde $X = [x_1, x_2, \ldots x_n]$ é o vetor de padrões de entrada, $A_{ji}(i = 1, 2, \ldots, n)$ são os termos linguísticos utilizados e representam os antecedentes da regra R_j. C_j é a classe consequente de R_j e N é o número fixo de regras *Fuzzy* necessárias para classificar os padrões de entrada. Esta regra de classificação *Fuzzy* R_j também pode ser expressa na forma $A_j \Rightarrow C_j$, onde A_j são todos os termos linguísticos antecedentes da regra e C_j é a classe que esta regra classifica.

O SGERD particiona os padrões de entrada em conjuntos *Fuzzy*, também chamados de termos linguísticos antecedentes da regra. A Figura 4.12 apresenta exemplos de diferentes particionamentos. Nesta figura, K representa a quantidade de termos

[14]A seleção tipo Estado Estacionário dos AG's funciona do seguinte modo: Em cada nova geração uns bons e poucos cromossomos são selecionados para a criação da descendência. A seguir, alguns maus cromossomos são removidos e novos descendentes são colocados em seus lugares. Todo o resto da população sobrevive para as próximas gerações.

linguísticos de cada partição. Se K for 3, por exemplo, L_3, L_4 e L_5 podem ser interpretados com os valores linguísticos “pequeno”, “médio” e “grande”, respectivamente.

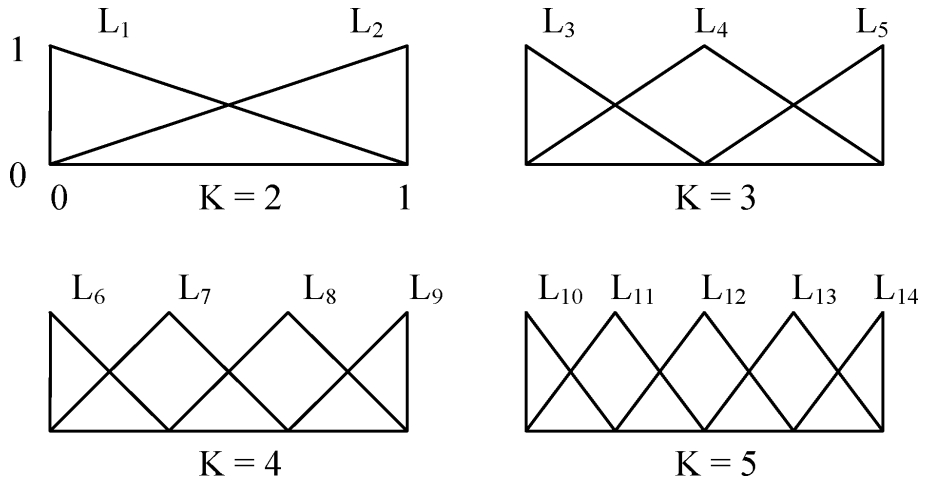


Figura 4.12: *Diferentes particionamentos utilizados em regras Fuzzy [24].*

A população inicial do SGERD é obtida a partir de todas as regras *Fuzzy* que contenham apenas um valor linguístico antecedente. Se a quantidade total de termos linguísticos for 14, então o número de regras iniciais será $14 * n$. Para cada regra inicial, é calculada a sua respectiva classe consequente. Este cálculo é apresentado a seguir.

Seja $X_1, X_2, \ldots, X_p, \ldots, X_m$ os padrões utilizados para extração das regras. O grau de compatibilidade de cada padrão X_p com uma determinada regra R_j, é dado pela equação 4.2:

$$\mu_j(X_p) = \prod_{i=1}^{n} \mu_{ji}(x_{pi}) \qquad (4.2)$$

onde n é a quantidade de variáveis linguísticas, x_{pi} é o valor de cada variável linguística em cada um dos padrões de entrada tal que $X_p = [x_{p1}, x_{p2}, \ldots, x_{pi}, \ldots, x_{pn}]$. $\mu_{ji}(x_{pi})$ é a função de pertinência de cada uma das A_{ji} variáveis linguísticas existentes na regra R_j aplicada ao padrão de entrada x_{pi}.

Após o cálculo do grau de compatibilidade de cada padrão de entrada, é calculada a eficiência de classificação de cada regra R_j em cada classe $Class\ T$, ou seja, o quanto uma determinada regra é capaz de classificar padrões em uma determinada classe. Este cálculo é realizado pela Equação 4.3:

$$Conf(A_j \Rightarrow Class\ T) = \frac{\sum_{X_p \in Class\ T} \mu_j(X_p)}{\sum_{p=1}^{m} \mu_j(X_p)} \qquad (4.3)$$

onde $\sum_{X_p \in Class\ T} \mu_j(X_p)$ realiza a soma apenas dos graus de compatibilidade de padrões de entrada que tem T como classe consequente.

Quanto maior for o valor de $Conf(A_j \Rightarrow Class\ T)$, maior será a eficiência de classificação da regra R_j para a classe $Class\ T$. Desta forma, a classe consequente C_j da regra R_j é obtida através da seguinte equação 4.4:

$$C_j = arg\ max\{Conf(A_j \Rightarrow ClassT)|T = 1,2,\ldots,M\} \quad (4.4)$$

Após conhecer a classe consequente de cada uma das regras R_j da população do algoritmo SGERD, o próximo passo é calcular o *fitness* de cada regra. Este cálculo inicia-se com a medida da qualidade de cada regra ao classificar padrões em uma determinada classe. Esta medida de avaliação é dada por:

$$f_F(A_j \Rightarrow Class\ C_j) = \sum_{X_p \in ClassC_j} \mu_j(X_p) - \sum_{X_p \notin ClassC_j} \mu_j(X_p) \quad (4.5)$$

onde $\mu_j(X_p)$ é calculado através da Equação 4.2.

Outro valor importante, para utilização no cálculo da função *fitness* é o tamanho do subespaço ρ_j coberto por cada valor linguístico A_{ji} de R_j. O valor de ρ_j é dado por:

$$\rho_j = \prod_{i=j}^{k_j} \xi(A_{ji}) \quad (4.6)$$

onde $\xi(A_{ji})$ é o grau de cobertura que cada valor linguístico possui, conforme apresentado na Tabela 4.3. Para exemplificar, o valor L_7 da Figura 4.12 possui grau de cobertura $\frac{2}{3}$, $\xi_{L_7} = \frac{2}{3}$, porque, ao observar esta Figura, a base do triângulo definido pela função de pertinência de L_7 toma $\frac{2}{3}$ espaço total.

Valor linguísticos, A	Grau de cobertura de atributos $\xi(A)$
L_1, L_2, L_4	1
L_7, L_8	2/3
$L_3, L_5, L_{11}, L_{12}, L_{13}$	1/2
L_6, L_9	1/3
L_{10}, L_{14}	1/4

Tabela 4.3: *Grau de cobertura de cada valor linguístico da Figura 4.12 [24]*

O maior inconveniente observado em sistemas baseados em regras *Fuzzy* é o *overfitting* [24]. Isto ocorre quando as regras geradas são muito específicas. Regras muito específicas possuem muitas variáveis antecedentes, o que diminui significativamente a performance do classificador. Outro problema relacionado com regras muito específicas está relacionado com a quantidade necessária de regras para que haja uma cobertura

adequada de classificação. Para evitar este inconveniente, a taxa de sobreposição das classes χ_j é utilizada na função *fitness*. O cálculo de χ_j é realizado através da Equação 4.7:

$$\chi_j = \left(1 - o(C_j)v(C_j)\right)^{k_j} \quad (4.7)$$

onde C_j é a classe consequente da regra R_j, k_j é o comprimento da regra R_j, $v(C_j)$ é o número de padrões classificados como C_j, $o(C_j)$ é o número de padrões atípicos classificados como C_j. Um padrão atípico pode ser um ruído ou um padrão com ocorrência muito rara.

Finalmente, o valor do *fitness* é dado por:

$$fitness\,(R_j) = \begin{cases} f_j, & se \quad k_j \leq n2 \\ \\ \rho_j \chi_j f_j, & se \quad k_j > n2 \end{cases} \quad (4.8)$$

onde f_j é o valor da medida da avaliação da regra R_j, calculada através da Equação 4.5, k_j é o comprimento da regra R_j, ρ_j é calculado com a Equação 4.6, χ_j é calculado com a Equação 4.7. Desta forma, a função *fitness* usada no SGERD objetiva caracterizar como melhores regras dentre toda uma população, as regras que possuírem melhor medida de avaliação, maior cobertura de seus termos linguísticos e menor comprimento.

As seleções individuais no algoritmo não são aleatórias e somente o melhor indivíduo sobreviverá. Cada pai produz uma quantidade finita de descendentes através da reprodução. As operações de mutação ocorrem com uma frequência muito baixa e, desta forma, poucos *crossovers* devem ser substituídos por mutações. O mecanismo de seleção de regras do SGERD utiliza competição entre estas, considerando a capacidade de aproximação a um resultado esperado como critério de qualidade de cada regra. No entanto, é considerada a cooperação entre estas regras, de modo a aumentar o poder de generalização do classificador. Para isto, o SGERD propõe uma abordagem heurística, que seleciona somente as regras mais cooperativas dentre a população.

Para exemplificar, serão utilizadas as seguintes melhores regras com apenas um termo linguístico antecedente, onde R1, R2 e R3 são as primeiras melhores regras para cada classe e R4, R5 e R6 são as segundas melhores regras para cada classe:

R1 : If x_4 is L_3, then class 1.
R2 : If x_4 is L_{12}, then class 2.
R3 : If x_4 is L_5, then class 3.
R4 : If x_3 is L_3, then class 1.
R5 : If x_3 is L_{12}, then class 2.
R6 : If x_3 is L_{13}, then class 3.

Distância	Distância Outra Viatura	Prioridade	Classificação
5	3,5	0,390	**alto**
6	3	0,415	**altíssimo**
7	2	0,125	**normal**
8	1	0,039	**baixo**
...	...	...	

Tabela 4.4: *Padrões utilizados para exemplificar o aprendizado de regras apresentado nesta Seção.*

Dentre a população de regras, é escolhido o melhor indivíduo por classe para participar da reprodução. No exemplo citado, R1 é o melhor indivíduo para a classe 1. Após esta escolha, o outro indivíduo que participará do processo de reprodução será escolhido de forma aleatória dentre os indivíduos da classe 1, a classe de R1. Se nesta escolha, R1 for selecionado novamente, como segundo indivíduo a participar da reprodução, será realizada a mutação neste e este novo indivíduo participará do processo de reprodução.

Neste exemplo, R4 foi escolhido como segundo indivíduo para a reprodução. O termo linguístico antecedente de R4 é x_3. Neste caso, serão produzidas 14 novos indivíduos, ou regras, do tipo "If x_4 is L_3 and x_3 is L_i, then class 1, for i = 1 ... 14". Estes novos indivíduos são avaliados quanto à aptidão através da função de *fitness*.

Os indivíduos que participaram do processo de reprodução R_P juntamente com os novos indivíduos aptos gerados R_O são candidatos a participar novamente deste processo. No máximo serão $C = R_P + R_O * 14$ indivíduos para a próxima etapa.

Este processo se repete enquanto novos indivíduos aptos estiverem sendo gerados e para quando, no processo de reprodução, nenhum novo indivíduo seja mais apto que seus pais.

Desta forma, o algoritmo retorna as melhores Q regras para cada uma das M classes, ou seja, $M * Q$ regras são retornadas.

4.3.2 Aplicação do SGERD no COGNARE

Esta Seção descreve o aprendizado de regras *Fuzzy* a partir de padrões, similares aos apresentados na Tabela 4.4. Nesta tabela, as colunas "Distância", "Distância Outra Viatura" e "Prioridade" são padrões de entrada enquanto que a coluna "Classificação" é o padrão de saída. Nota-se que, neste exemplo, os padrões de entrada (ou antecedentes da regra) são números reais e o padrão de saída (ou classe consequente) é um elemento do conjunto $S = \{baixo, normal, alto, altissimo\}$. O objetivo é gerar regras que, a partir de novos padrões, consigam realizar classificações similares às classificações da Tabela 4.4.

Neste exemplo, foram definidos os 14 termos linguísticos *Fuzzy* ilustrados na Figura 4.12. Tradicionalmente são utilizadas funções de pertinência triangulares [24] pois estas são de mais fácil entendimento por seres humanos.

Na codificação, cada gene do cromossomo é representado pelos antecedentes da regra e pela classe consequente, conforme ilustrado na Figura 4.13. O termo linguístico L_0 é utilizado para representar as variáveis que estão inativas na regra. Desta forma, na Figura 4.13, o cromossomo representa a regra: "*if distância is pouco then classificação is alto*" pois, somente o gene Distância possui um valor diferente de L_0 (no caso, L_3 é interpretado como "pouco"). Os genes que representam os demais antecedentes possuem valor igual a L_0.

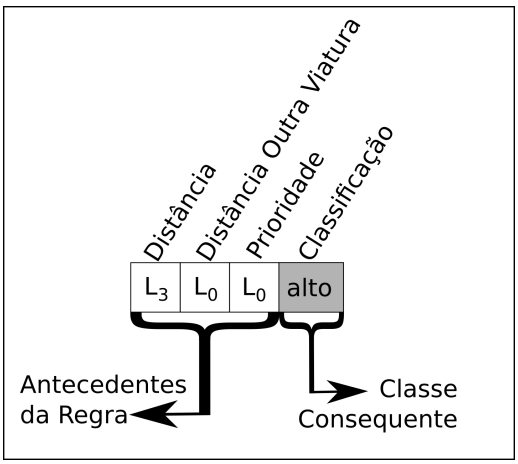


Figura 4.13: *Ilustração da codificação de um cromossomo.*

A população inicial é composta por todas as regras *Fuzzy* com apenas um termo linguístico antecedente. Neste exemplo, a população inicial contém 42 indivíduos[15](que neste caso são chamados de regras *Fuzzy* candidatas). Para cada uma destas regras candidatas, foi determinada a sua respectiva classe consequente através da Equação 4.4.

A seleção é realizada da seguinte forma. Cada uma das regras, geradas no passo anterior, são separadas em M conjuntos, um para cada classe consequente. Após esta separação é calculado o valor do *fitness* de cada uma destas regras candidatas, através da Equação 4.8. Cada um dos M conjuntos são ranqueados, em ordem decrescente do valor do *fitness* de cada uma das regras candidatas contidas nele. Após isto, são escolhidas as Q[16]melhores regras de cada conjunto.

Na reprodução, inicialmente são escolhidos os dois melhores indivíduos de cada um dos M conjuntos. Após isto, o descendente é criado a partir da união dos genes

[15]Este valor é o resultado da multiplicação da quantidade de termos linguísticos gerada (no caso 14) pela quantidade de variáveis de entrada (no caso 3 - "Distância", "Distância Outra Viatura" e "Prioridade").

[16]Q é um dos parâmetros de entrada do algoritmo. Neste caso $Q = 4$. Esta foi uma escolha empírica, após a análise do resultado final.

ativos (diferentes de L_0) de cada um dos cromossomos. Por exemplo, seja R_1 e R_2 os cromossomos escolhidos para participar do processo de reprodução. Conforme ilustrado na Figura 4.14, R_1 possui o gene "Distância Outra Viatura" ativo com o valor L_5 (muito) enquanto que R_2 possui o gene "Distância" ativo com o valor L_3 (pouco). Neste caso, o descendente terá os genes ativos de R_1 e R_2 com seus respectivos valores.

Após sua geração, o cromossomo descendente é submetido a nova avaliação da sua classe consequente, através da Equação 4.4. Isto é necessário pois, a alteração dos valores linguísticos de cada gene pode implicar na mudança da classe consequente da regra. Na ilustração da reprodução, apresentada na Figura 4.14, é exemplificado este fato. Nesta figura, os reprodutores fazem parte do grupo que classifica para a classe "alto" e gerou um descendente que classifica para "altíssimo".

Após a geração do novo cromossomo, o próximo passo é calcular o seu valor de *fitness*, através da Equação 4.8. Se este valor for maior que o *fitness* de R_1, então este cromossomo descendente substituirá R_1. Caso contrário, o descendente será descartado.

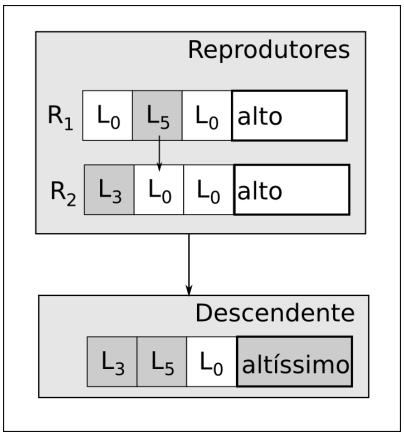


Figura 4.14: *Reprodução a partir de dois cromossomos.*

Este algoritmo para quando uma das duas condições seguintes forem alcançadas. Primeiro, quando os descentes gerados não forem melhores que seus pais. Esta é uma situação de convergência, ou seja, mesmo que ocorram novas reproduções, estas não implicarão em indivíduos melhores. A segunda condição de parada ocorre quando a quantidade de iterações ultrapassar $n * M$, onde n é a quantidade de padrões de entrada (antecedentes das regras) e M é a quantidade de classes consequentes. Neste exemplo este valor limite de iterações é 12 ($3 * 4$). No final, as regras obtidas são similares às da Tabela 4.5.

4.3.3 Geração do arquivo FCL

A geração do arquivo, na linguagem FCL, a partir das regras geradas na Seção anterior, ocorre da seguinte forma. Inicialmente são identificados os termos linguísticos de

Número	Regra
1	IF distancia IS L_5 THEN classificacao is baixo
2	IF distancia IS L_7 AND prioridade IS L_3 THEN classificacao is normal
3	IF distanciaViatura IS L_1 AND prioridade IS $L_1$1 THEN classificacao is alto
4	IF distanciaViatura IS L_2 AND prioridade IS L_4 THEN classificacao is altissimo
5	IF distancia IS $L_1$1 AND prioridade IS L_3 THEN classificacao is normal

Tabela 4.5: *Regras geradas após o término do algoritmo SGERD.*

Variável	Quantidade	Termos linguísticos	Conversão através da Tabela 4.7
distancia	3	L_5, L_7, L_{11}	pouco, médio, muito
distanciaViatura	2	L_1, L_2	pouco, muito
prioridade	3	L_3, L_4, L_{11}	pouco, médio, muito

Tabela 4.6: *Identificação dos termos semânticos e valores linguísticos utilizadas por cada uma das variáveis das regras aprendidas na Seção 4.3.2.*

Quantidade	Termos Linguísticos
2	pouco, muito
3	pouco, médio, muito
4	muitoPouco, pouco, médio, muito
5	muitoPouco, pouco, médio, muito, muitíssimo

Tabela 4.7: *Relação dos valores semânticos com a quantidade de termos.*

cada variável destas regras geradas. Para exemplificar, os termos linguísticos, das regras geradas na Seção anterior, estão identificados na Tabela 4.6. Esta tabela relaciona cada um dos termos com valores com um significado semântico, tal como “pouco”, “médio”, etc.

Este relacionamento é realizado com auxílio da Tabela 4.7. Esta tabela relaciona a quantidade de valores linguísticos diferentes, utilizados para uma determinada variável, a valores com um significado semântico. Por exemplo, se determinada variável, gerada na Seção anterior, fizer uso de três termos linguísticos, os seus valores semânticos serão: “pouco”, “médio” e “muito”. Desta forma, a tabela de regras, geradas na Seção anterior, pode ser reescrita conforme apresentado na Tabela 4.8.

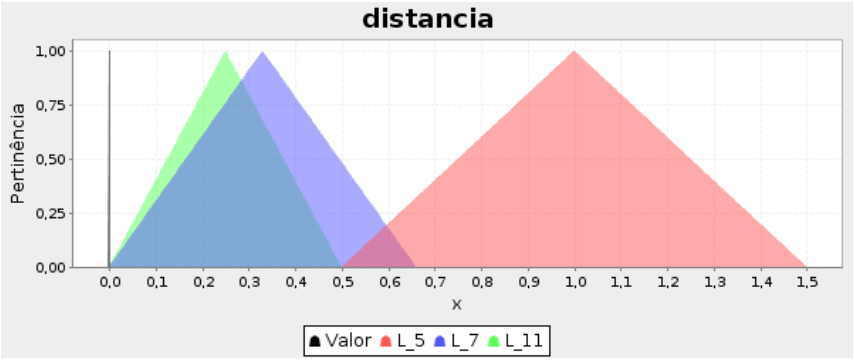
O próximo passo, do processo de criação do arquivo FCL [19], consiste em gerar os conjuntos *Fuzzy*, com suas respectivas funções de pertinência. Para isto, é utilizada a Tabela 4.9. Nesta tabela, estão relacionados os valores (normalizados) de cada uma das funções de pertinência triangulares com seu respectivo termo linguístico. Por exemplo, de acordo com a Tabela 4.6, os termos linguísticos da variável distância são: L_5, L_7, L_{11}. Desta forma, a Figura 4.15 ilustra os conjuntos *Fuzzy* desta variável com suas respectivas

Número	Regra
1	IF distancia IS **pouco** THEN classificacao is baixo
2	IF distancia IS **médio** AND prioridade IS **pouco** THEN classificacao is normal
3	IF distanciaViatura IS **pouco** AND prioridade IS **muito** THEN classificacao is alto
4	IF distanciaViatura IS **muito** AND prioridade IS **médio** THEN classificacao is altissimo
5	IF distancia IS **muito** AND prioridade IS **pouco** THEN classificacao is normal

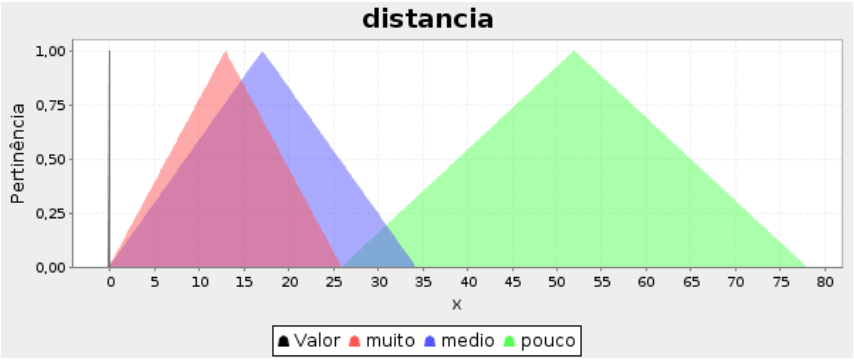
Tabela 4.8: *Regras geradas após a substituição para termos com significado semântico.*

Número	Função		
	min	médio	max
L_1	-1,0	0,0	1,0
L_2	0,0	1,0	2,0
L_3	-0,5	0,0	0,5
L_4	0,0	0,5	1,0
L_5	0,5	1,0	1,5
L_6	-0,33	0,0	0,33
L_7	0,0	0,33	0,66
L_8	0,33	0,66	1,0
L_9	0,66	1,0	1,33
L_{10}	-0,25	0,0	0,25
L_{11}	0,0	0,25	0,5
L_{12}	0,25	0,5	0,75
L_{13}	0,5	0,75	1,0
L_{14}	0,75	1,0	1,25

Tabela 4.9: *Relação dos termos linguísticos com suas respectivas funções de pertinência. Nesta tabela, estes valores estão normalizados.*



(a) *Valores normalizados.*



(b) *Valores finais depois da conversão.*

Figura 4.15: *Ilustração das funções de pertinência geradas no exemplo.*

funções de pertinência. A parte (b) da Figura 4.15, apresenta os valores após a conversão para a escala real. Para esta conversão, é obtido o valor máximo que a variável distância pode ter, no caso, o valor máximo desta variável é 52 (medidos na unidade Km). Este valor é multiplicado pelos valores normalizados, resultando nos valores finais, considerados na construção do arquivo FCL.

Finalmente, são adicionadas mais regras no conjunto de regras geradas até agora. O objetivo é que seja obtida uma maior cobertura por parte destas regras. Para isto, utiliza-se os indivíduos, da população inicial, do algoritmo genético apresentado na Seção 4.3.2. Estes indivíduos, são separados em grupos. Cada um destes grupos contém todos os indivíduos com a mesma variável antecedente. Depois calcula-se, por meio da equação 4.8, o valor do *fitness* para cada um destes indivíduos. Posteriormente, são escolhidos os indivíduos com melhor valor de *fitness* de cada um dos grupos para preencher a lista final de regras. Desta forma, a quantidade final de regras, geradas por intermédio deste processo, é quantidade de variáveis de entrada, acrescida com a quantidade de regras geradas pelo algoritmo genético.

4.3.4 Utilização do módulo de Aprendizagem

Para utilizar este módulo, inicialmente, é necessário adicionar o arquivo *cognare-learner.jar* no *classpath* da aplicação que se deseja integrar o COGNARE. Após isto, deve ser criado, no diretório do usuário, o arquivo *cognare-learner.properties* cujo conteúdo foi exemplificado na Listagem 4.2.

A propriedade *learner* referencia a classe implementada a partir da interface *Learner*, conforme ilustrado na Figura 4.10. As propriedades *database.driver, database.url, database.username* e *database.password* devem ser configuradas com os dados necessários para a conexão com o banco de dados. O esquema de criação das tabelas deste banco de dados é apresentado no Apêndice C.

Listagem 4.2: *Propriedades configuradas para o módulo "Aprendizagem"*

```
1 learner=in.solutio.cognare.learning.impl.SGERDLearner
2
3 database.driver=org.postgresql.Driver
4 database.url=jdbc:postgresql://localhost/cognare
5 database.username=postgres
6 database.password=123456
```

A inicialização deste módulo deve ser feita através de código java executado na aplicação integrada ao COGNARE. Para isto, é necessário a execução do seguinte método:

```
CognareLearner.learn()
```

Após a execução deste método, o módulo de aprendizagem lê o repositório de decisões já tomadas, contido no banco de dados configurado. Após esta leitura são geradas, na mesma base de dados os conjuntos e as regras *Fuzzy* utilizadas pelo módulo "Alocador *Fuzzy*", descrito na Seção 4.4.

4.4 Módulo: Alocador Fuzzy

Este módulo é um FRBS, responsável por realizar a alocação de recursos. Conforme descrito na Seção 2.1, uma das partes de um FRBS é um *Fuzzy Inference System* (FIS). Atualmente existem diversas implementações de FIS disponíveis para utilização. Um dos cuidados tomados, no momento da concepção deste projeto, foi abstrair a utilização de um FIS específico. Com este objetivo, foi criada a interface *FisExecutor*.

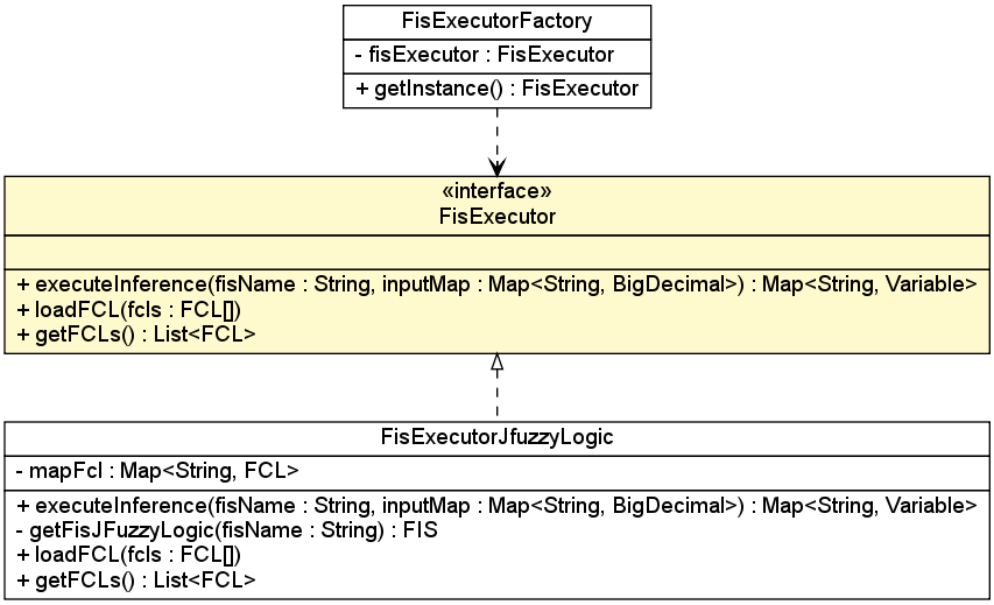


Figura 4.16: *Diagrama de classes que ilustra os relacionamentos da interface FisExecutor.*

A Figura 4.16 apresenta o diagrama da interface *FisExecutor* com seus relacionamentos. Conforme pode ser visto, esta interface possui os métodos *executeInference(...)*, *loadFCL(...)* e *getFCLs()*. A função do método *loadFCL(...)* é carregar os parâmetros do FIS. Nota-se que o método *loadFCL(...)* recebe um vetor de objetos do tipo FCL, cuja classe foi ilustrada na Figura 4.17. O valor do atributo *fclDefinition* deve ser uma *String* com o conteúdo na linguagem FCL.

O método *executeInference(...)*, da *interface FisExecutor*, é responsável por processar os parâmetros de entrada e retornar as variáveis de saída do FIS. Este processamento consiste em fuzzificar as variáveis de entrada, processar as regras *Fuzzy* declaradas no objeto do tipo FCL e defuzzificar as variáveis de saída.

A Figura 4.16, apresenta também, a classe *FisExecutorJfuzzyLogic*. Esta é uma classe que implementa a interface *FisExecutor*. Neste caso, esta classe utiliza a implementação do *framework* JFuzzyLogic[17] para realizar as operações *Fuzzy*, utilizado como implementação de referência neste trabalho. Devido ao fato de ter sido abstraída a camada de inferência, é possível substituir esta implementação. Isto se dá por intermédio da classe *FisExecutorFactory* que implementa o padrão de projeto "*Factory*"[18][15] e é responsável pela instanciação adequada do *FisExecutor*. Neste caso, as instâncias de *FisExecutor* são obtidas através da invocação do método *getInstance()* da classe *FisExecutorFactory*.

[17] O *framework JFuzzyLogic* é descrito na Seção A.1.

[18] O padrão Factory é um padrão de projeto orientado a objetos que implementa o conceito de fábricas. Este padrão lida com o problema da criação de objetos, sem especificar a classe exata do objeto que será

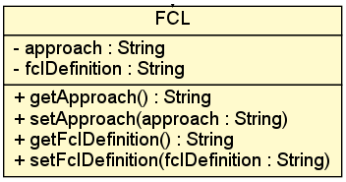


Figura 4.17: *Classe FCL*

Uma vez calculadas as variáveis de saída, através da execução do método *executeInference(...)*, da *interface FisExecutor*, inicia-se o processo para a geração decisões, retornadas por este módulo. Uma decisão é uma instância da classe *Decision*, apresentada na Figura 4.18. Cada decisão relaciona-se com um recurso e uma tarefa. O atributo *score* da decisão contém o valor do Coeficiente de Decisão. Este é um coeficiente utilizado para avaliar a qualidade das decisões. Desta forma, quanto maior for este coeficiente, melhor será a decisão.

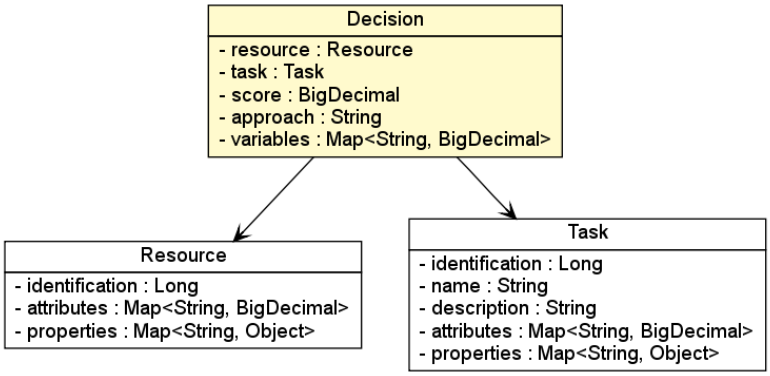


Figura 4.18: *Classes Decision, Task e Resource e seus relacionamentos*

A decisão é tomada pelo método *makeDecision(String approach, List<Resource> resources, List<Task> tasks)* da classe *DecisionMaker*, ilustrada na Figura 4.19. Este método recebe as listas, contendo os recursos e as tarefas a serem alocados, e retorna uma lista de decisões. O fluxo executado por este método foi ilustrado no diagrama da Figura 4.20. Inicialmente, é gerada a lista *D* com todas as possíveis decisões. Esta lista é criada por meio da combinação de todos os elementos da lista de tarefas com os elementos da lista de recursos.

instanciado. A criação de um objeto muitas vezes exige processos complexos, não apropriados para inclusão dentro de um objeto de composição ou ainda que podem levar a uma duplicação significativa de código. Neste sentido, o padrão de projeto Factory lida com esses problemas através da definição de um método separado para criar os objetos, o que pode, então, substituir subclasses para especificar o tipo derivado de produto que será criado [15].

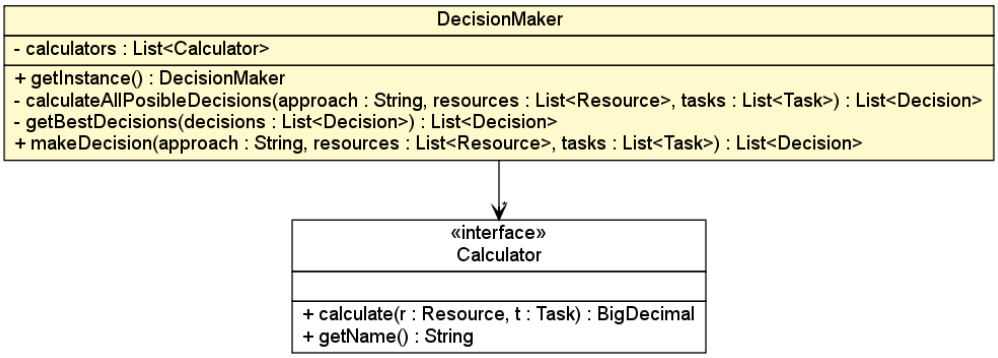


Figura 4.19: *Classes DecisionMaker, Task e Resource e seus relacionamentos*

Para cada elemento de *D*, são calculadas as demais variáveis. Este cálculo é realizado invocando o método *calculate()* da interface *Calculator*, ilustrada na Figura 4.19. Este método recebe uma tarefa e um recurso e calcula as variáveis que não estão disponíveis no ambiente externo. Este valor calculado é adicionado na lista das possíveis decisões *D*. Um exemplo de variável, que pode ser calculada neste processo, é a distância entre uma tarefa e um recurso. Esta é uma das variáveis discutidas no estudo de caso apresentado na Seção 5.1. Neste caso, a distância não está disponível no banco de dados externo. Devido a este fato, naquele estudo de caso foi criada a classe *EuclidianDistance*, que implementa *Calculator*, cujo objetivo é calcular a distância euclidiana entre uma tarefa e um recurso.

Uma vez tendo sido preenchidos os demais valores das variáveis de *D*, esta lista será enviada ao FIS. Já no FIS, é gerado um mapa de variáveis para cada elemento da lista de possíveis decisões *D*. Este mapa de variáveis é passado como parâmetro para o método *executeInference(...)* da interface *FisExecutor*. Este método fuzzifica estas variáveis de entrada, depois processa as regras *Fuzzy* previamente carregadas e, logo após, defuzzifica as variáveis de saída. Estas variáveis de saída são retornadas pelo FIS, sendo uma destas o "Coeficiente de Decisão".

A escolha da melhor decisão é feita utilizando uma estratégia gulosa. Neste tipo de estratégia, o objetivo é buscar pelas soluções que parecem melhores localmente. Desta forma, a lista *D* é ordenada de forma decrescente por "Coeficiente de Decisão". O *loop* percorre do primeiro elemento até o último. Para cada iteração é lido um elemento de *D*. Os outros elementos, cujas tarefas e recursos estejam relacionados com a decisão lida nesta iteração, são excluídos. Este passo visa impedir que existam decisões concorrentes. Duas decisões são consideradas concorrentes quando elas referenciam o mesmo recurso ou a mesma tarefa. A lista contendo as decisões finais é chamada de D_f

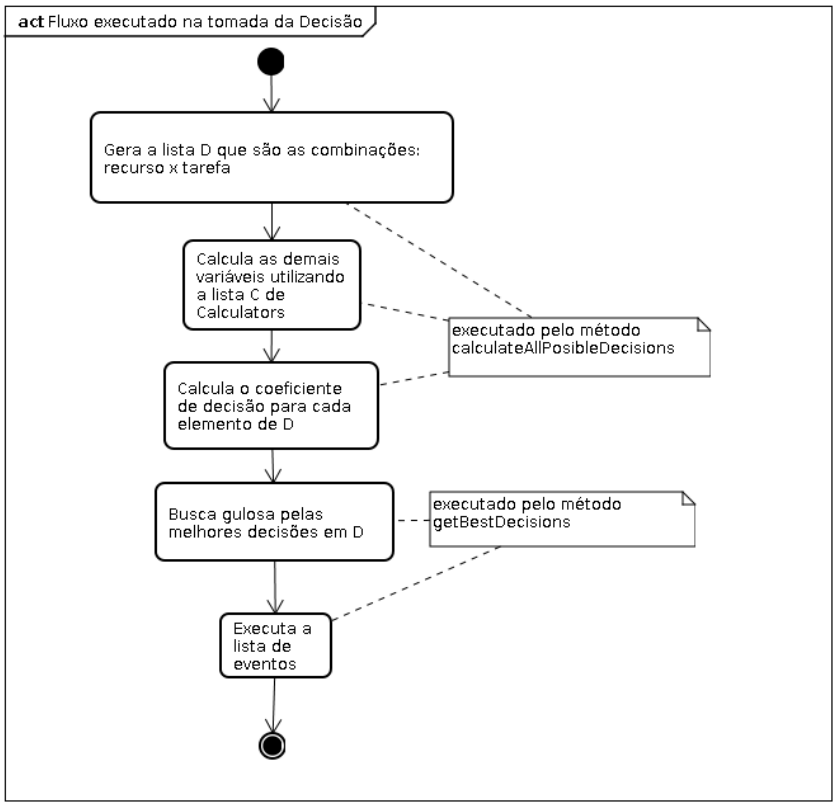


Figura 4.20: *Fluxo executado na tomada de decisão executado pelo método makeDecision da classe DecisionMaker.*

Para exemplificar este processo, seja a situação ilustrada na Figura 4.21. Esta situação é similar à apresentada na Seção 5.1 onde cada tarefa é um ponto de defeito na rede elétrica da CELG[19]. Naquele estudo de caso as tarefas são as ocorrência abertas no sistema da CELG. Os recursos são as equipes, denominadas viaturas, responsáveis por corrigir estes defeitos. Na Figura 4.21, as viaturas são ilustradas com o desenho de um caminhão enquanto que as ocorrências são ilustradas com sinais de exclamação. Próximo a cada viatura existem algumas informações como a denominação ("Recurso 1" e "Recurso 2") e a localização (coordenadas x e y). Próximo a cada ocorrência, além dos elementos denominação e localização, existe também a informação da prioridade. Por tratar-se de um exemplo, todas estas informações são fictícias, as coordenadas podem estar em qualquer escala, sem prejudicar o sentido do exemplo.

[19]CELG é a empresa estatal responsável pela distribuição de energia elétrica no estado de Goiás.

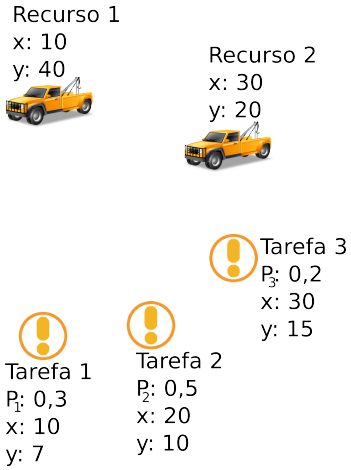


Figura 4.21: *Situação utilizada para exemplificar o processo da tomada de decisão.*

Seguindo o fluxo de decisão explicado até aqui e aplicado ao exemplo apresentado, será criada, inicialmente, a lista D com todas as possíveis decisões. Após este passo, serão calculadas, para cada elemento de D as demais variáveis. Este cálculo é realizado através da *interface Calculator*, da Figura 4.19. Neste exemplo, a única variável que necessita ser calculada, é a distância euclidiana entre uma viatura (Recurso) e uma ocorrência (Tarefa). Portanto, as variáveis que serão consideradas na tomada de decisão serão a "Prioridade da Tarefa" e a "Distância". Estas variáveis são passadas como parâmetro para o método *executeInference(...)*, que calcula o "Coeficiente de Decisão" de cada possível decisão em D. A Tabela 4.10 apresenta a situação destas possíveis decisões após esses passos serem executados. Esta tabela já está ordenada de forma decrescente pela coluna "Coeficiente de Decisão".

Decisão	Tarefa	Recurso	Prioridade da Tarefa	Distância	Coeficiente de Decisão
D_6	$Tarefa_3$	$Recurso_2$	$0,2$	5	$0,9$
D_4	$Tarefa_2$	$Recurso_2$	$0,5$	$14,14$	$0,8$
D_3	$Tarefa_2$	$Recurso_1$	$0,5$	$31,62$	$0,7$
D_2	$Tarefa_1$	$Recurso_2$	$0,3$	$23,85$	$0,6$
D_1	$Tarefa_1$	$Recurso_1$	$0,3$	33	$0,5$
D_5	$Tarefa_3$	$Recurso_1$	$0,2$	$32,01$	$0,4$

Tabela 4.10: *Tabela D gerada à partir do exemplo da Figura 4.21.*

A estratégia de escolha das melhores decisões será feita de forma gulosa. Seguindo esta estratégia, o primeiro elemento da Tabela 4.10 é escolhido, no caso D_6. Como esta decisão referencia a $Tarefa_3$ e o $Recurso_2$, as decisões D_2, D_4, D_5 serão excluídas, pois referenciam o mesmo recurso ou a mesma tarefa de D_6. Após

esta exclusão, restam as decisões D_6, D_3, D_1. A segunda decisão escolhida é D_3. Esta referencia a $Tarefa_2$ e o $Recurso_1$ e, por isto, a decisão D_1 será excluída.

Após a execução da estratégia gulosa, para a obtenção das melhores decisões, o processo retorna as decisões D_6, D_3, ou seja, a $Tarefa_3$ será atribuída ao $Recurso_2$ e a $Tarefa_2$ será atribuída ao $Recurso_1$, conforme ilustrado na Figura 4.22. Desta forma, tem-se $D_f = \{D_3, D_6\}$.

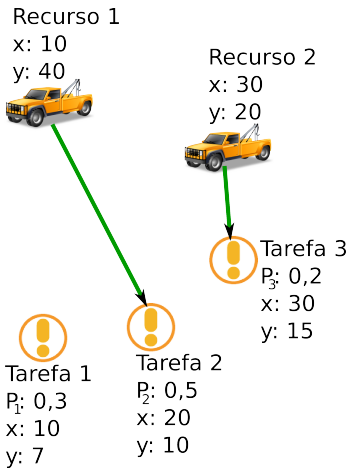


Figura 4.22: *Decisões retornadas após a execução do processo de tomada de decisão.*

Finalmente serão invocados os métodos *executePosibleDecisions(List<Decision> decision)* e *executeBestDecisions(List<Decision> decision)* da classe *UtilDecisionListener*, ilustrada na Figura 4.23. Este é o passo final do fluxo apresentado na Figura 4.20. O objetivo deste passo é permitir a personalização de ações que devem ser executadas quando o COGNARE retornar uma decisão. Desta forma, sempre que o COGNARE terminar o processo de tomada de decisão, o método *executePosibleDecisions(List<Decision> decision)* será invocado tendo a lista D como parâmetro de entrada. O método *executeBestDecisions(List<Decision> decision)* será executado tendo como parâmetro de entrada a lista D_f.

Portanto, para definir ações customizadas, que serão executadas no final do processo de tomada de decisão, basta implementar uma classe a partir da *interface DecisionListener* e configurá-la no COGNARE. A classe *UtilDecisionListener* se encarregará de instanciar os objetos destas classes de customização e executar os métodos da interface. A Figura 4.23 também apresenta o diagrama da *interface DecisionListener* com os métodos que devem ser implementados.

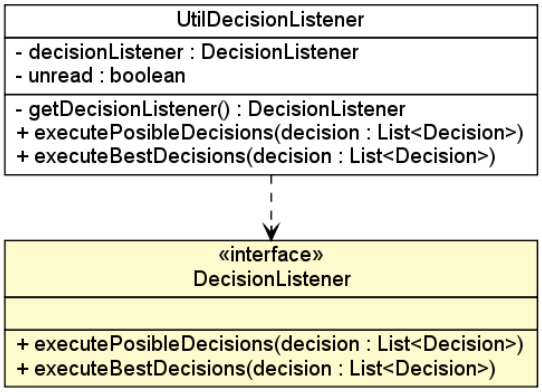


Figura 4.23: *Classes Decision, Task e Resource e seus relacionamentos*

4.4.1 Utilização do módulo Alocador Fuzzy

Para utilizar este módulo, inicialmente é necessário adicionar o arquivo *cognare-inference.jar* no *classpath* da aplicação que se deseja integrar o COGNARE. Após isto, deve ser criado, no diretório do usuário, o arquivo *cognare-inference.properties* cujo conteúdo foi exemplificado na Listagem 4.3.

A propriedade *fis* referencia a classe implementada a partir da interface *FisExecutor*, conforme ilustrado na Figura 4.16. A propriedade *timeWaitDecision* deve ser configurada com o tempo, em milisegundos, que o sistema irá esperar entre cada tomada de decisão. A propriedade *calculator.1* deve ser configurada com a classe que implementa a interface *Calculator*, apresentada na Figura 4.19. As propriedades *database.driver, database.url, database.username* e *database.password* devem ser configuradas com os dados necessários para a conexão com o banco de dados. O esquema de criação das tabelas deste banco de dados é apresentado no Apêndice C.

Listagem 4.3: *Propriedades configuradas para o módulo "Aprendizagem"*

```
1  fis=in.solutio.cognare.inference.jfuzzylogic.FisExecutorJfuzzyLogic
2
3  timeWaitDecision = 15000
4
5  calculator.1=in.solutio.cognare.celg.calculator.EuclidianDistance
6
7  database.driver=org.postgresql.Driver
8  database.url=jdbc:postgresql://localhost/cognare
9  database.username=postgres
10 database.password=123456
```

A inicialização deste módulo deve ser feita através de código java, executado na aplicação integrada ao COGNARE. Para isto, é necessário a execução do código

da Listagem 4.4. Nesta listagem estão exemplificados os métodos necessários para o carregamento na memória das regras *Fuzzy* (linha 2) e para a tomada de decisão a partir das listas de recursos e tarefas (linha 6).

Listagem 4.4: *Propriedades configuradas para o módulo "Aprendizagem"*

```
1 FisExecutor exec = FisExecutorFactory.getInstance();
2 exec.loadFCL();
3
4 DecisionMaker decisionMaker = DecisionMaker.getInstance();
5
6 List<Decision> decisions=decisionMaker.makeDecision(resources, tasks);
```

4.5 Discussão

Este Capítulo apresentou o projeto COGNARE e sua arquitetura. A principal vantagem desta arquitetura proposta é a sua flexibilidade, pois esta abstraiu em *interfaces* os *frameworks* utilizados. Isto possibilita a criação de diversas implementações sem alterar o núcleo do projeto.

Outra preocupação tomada, foi a utilização de *frameworks* cujas licenças são livres, o que permite a utilização deste sem restrições.

A metodologia Prometheus[38], utilizada na concepção do SMA, foi de grande auxílio pois norteou o desenvolvimento dos agentes e a troca de dados entre eles. Neste sentido, a ferramenta PDT, foi utilizada para desenhar os diagramas, necessários para a modelagem do SMA.

Nesta implementação, foi utilizado somente o algoritmo SGERD[24] para geração das regras *Fuzzy*, utilizadas na tomada de decisão. Segundo Mansoori [24], este algoritmo produz bons resultados quando comparado com outros algoritmos, também utilizados na extração de regras *Fuzzy*. Conforme apresentado na Seção 4.3, é possível utilizar outros algoritmos de aprendizagem de regras para este fim. Para isto, basta implementar a *interface Learner* com seu método *getKnowledge(...)*.

Foi utilizado o *framework* JFuzzyLogic na implementação do módulo alocador pois este, além de possuir licença LGPL[20], foi escrito em Java, o que tornou mais fácil sua integração com o COGNARE. Apesar disto, conforme apresentado na Seção 4.4, é possível a integração de outros *frameworks* similares, mesmo aqueles escritos em outras linguagens, ao projeto COGNARE.

[20] http://www.gnu.org/copyleft/lesser.html

CAPÍTULO 5

Estudos de caso na aplicação do Projeto do COGNARE

Este Capítulo apresenta dois estudos de caso que utilizaram o COGNARE na alocação dinâmica de recursos.

O primeiro estudo de caso, apresentado na Seção 5.1, empregou o COGNARE na alocação de ocorrências, referentes às falhas no fornecimento de energia elétrica, à viaturas da CELG, responsáveis por corrigirem estas falhas. A apresentação deste estudo de caso foi dividida em duas etapas. Na primeira etapa, foi descrita a metodologia utilizada no aprendizado das regras *Fuzzy* junto a um especialista da CELG. Na segunda etapa, foi descrita a utilização do COGNARE como ferramenta capaz de fornecer sugestões quanto à alocação de ocorrências a viaturas. No final da Seção 5.1.2, é apresentado o resultado deste estudo de caso.

O segundo estudo de caso, apresentado na Seção 5.2, associou o COGNARE a um balanceador de carga cujo objetivo foi o de melhorar a performance do sistema receptor da Nota Fiscal Eletrônica de Goiás. A apresentação deste estudo de caso se dividiu da seguinte forma. Inicialmente, foi descrita toda a integração do COGNARE com um balanceador de carga e como este foi configurado a fim de ser capaz de auxiliar na recepção das Notas Fiscais Eletrônicas. Finalmente, na Seção 5.2.1 foram apresentados os resultados obtidos. Nesta Seção, foi possível notar o quanto o COGNARE é capaz de aumentar a performance do sistema receptor de Notas Fiscais Eletrônicas.

Em ambos os estudos de caso foram obtidos resultados que justificam a utilização das técnicas de Inteligência Artificial, empregadas no COGNARE, na solução dos problemas referente à alocação dinâmica de recursos aqui apresentados.

5.1 Estudo de Caso 1 - Alocação de equipes no COD da CELG

Este estudo de caso utilizou o COGNARE para auxiliar na tomada de decisão dos operadores da CELG. Quando ocorrem problemas na rede elétrica, os consumidores ligam para a CELG informando o caso. Estas ligações, dão origem a reclamações. Posteriormente, estas reclamações são enviadas para um processo de triagem. O objetivo desta triagem é descobrir o ponto da rede elétrica em que está o defeito. Esta triagem dá origem a ocorrências, que são inseridas no sistema OPER[1].

A CELG possui um departamento denominado Centro de Operação da Distribuição (COD)[2] cuja responsabilidade é decidir sobre qual viatura deve atender uma determinada ocorrência. Para realizar esta decisão, um operador do COD leva em consideração diversas variáveis, tais como, distância, tipo da ocorrência, etc. É neste ponto que o COGNARE foi empregado, para auxiliar um operador do COD nessa tomada de decisão.

A Figura 5.1 ilustra como o COGNARE foi integrado ao sistema OPER, da CELG. Nesta figura, é possível notar que a interface entre o OPER e o COGNARE é o "Conector CELG". Este conector é responsável por disponibilizar os dados do OPER diretamente para o COGNARE.

Após o processamento, o COGNARE armazena as decisões de alocação em um banco de dados, também ilustrado na Figura 5.1. Neste momento, as decisões do COGNARE passam a ser sugestões para os operadores do COD.

Finalmente, o sistema SATE 01[3] lê estas sugestões e as apresenta para os operadores do COD auxiliando-os, desta forma, na tomada de decisão quanto a alocação de ocorrências a viaturas.

---

[1] O OPER é um sistema desenvolvido pela CELG. O objetivo deste sistema é possibilitar o gerenciamento das ocorrências abertas e das equipes disponíveis para atendê-las.

[2] Existem vários COD's, um para cada região do estado. Cada COD é dividido em mesas e cada mesa é gerenciada por um operador que possui uma certa quantidade de viaturas. A tarefa deste operador é decidir qual ocorrência cada viatura deve atender primeiramente.

[3] O SATE 01 é o protótipo de um sistema que será desenvolvido para apresentar, ao operador do COD, as sugestões fornecidas pelo COGNARE.

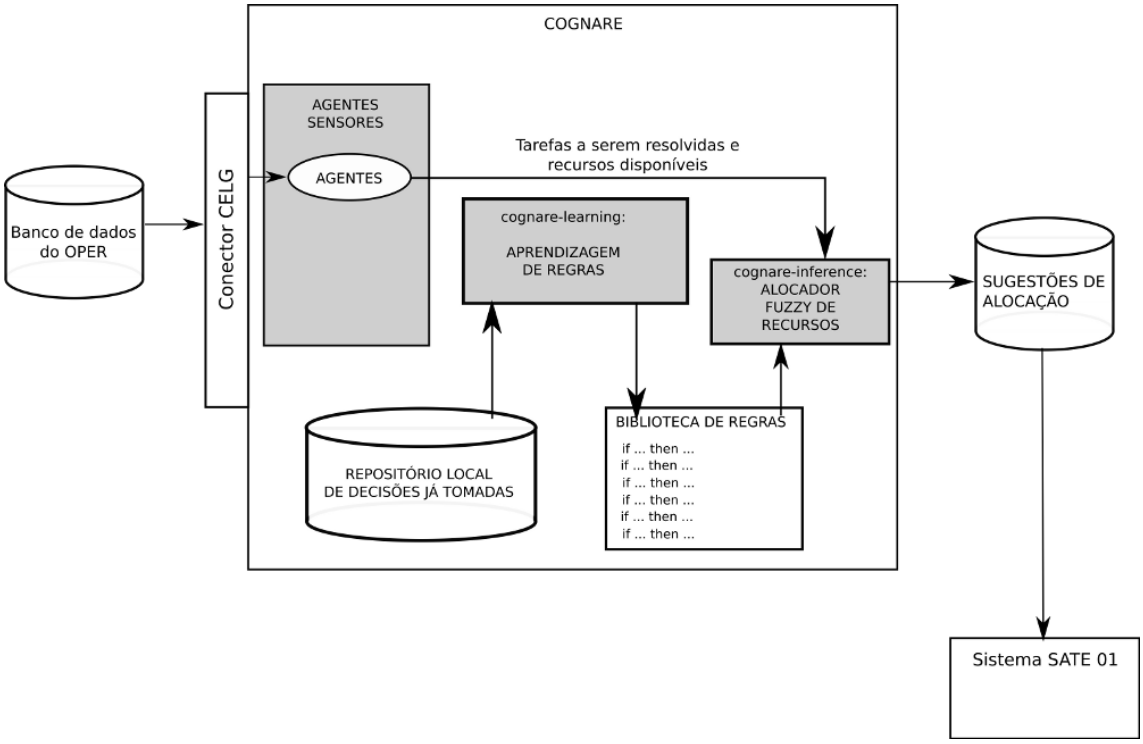


Figura 5.1: *Integração do COGNARE na CELG.*

Conforme apresentado no Capítulo 4, o COGNARE toma as decisões por meio de regras *Fuzzy* previamente aprendidas. Uma dificuldade encontrada, neste estudo de caso, foi a obtenção dos dados necessários no processo de aprendizagem destas regras. Inicialmente, acreditava-se na possibilidade de utilização dos dados contidos no banco de dados do OPER. Após algumas análises neste banco de dados e reuniões com a equipe do COD, foi constatado que os exemplos de atribuições, contidos na base de dados do OPER, não eram bons o suficiente para servirem como exemplo no aprendizado das regras.

Neste sentido, foi criado um aplicativo, capaz de gerar exemplos de cenários que ocorrem tipicamente em um COD. Posteriormente, estes cenários foram submetidos à avaliação de um especialista do próprio COD. A partir do resultado desta avaliação foi criado o repositório local de decisões já tomadas, ilustrado na Figura 5.1. Após isto, o módulo de aprendizagem de regras, descrito na Seção 4.3 do Capítulo 4, foi capaz de realizar a aprendizagem das regras e armazená-las na “Biblioteca de Regras”.

Posteriormente, estas regras foram utilizadas, pelo COGNARE, para decidir, de forma satisfatória e automática, a alocação de tarefas a recursos. Neste caso, as tarefas são as ocorrências abertas pelo OPER e os recursos as viaturas disponíveis.

A seguir são detalhadas as etapas deste processo. Na primeira etapa, apresentada na Seção 5.1.1, é descrita a metodologia utilizada e o aplicativo desenvolvido para obtenção dos dados junto ao especialista do COD da CELG. Na segunda etapa, apresentada na Seção 5.1.2, é descrito o processo de aprendizagem das regras *Fuzzy*. A terceira etapa, apresentada na Seção 5.1.3, descreve o processo de tomada de decisão,

Distância	Prioridade OPER	Risco Vida	Distância da Viatura mais Próxima	Coeficiente de Decisão
5	0,390	NÃO	6	**alto**
6	0,415	SIM	7	**altissimo**
7	0,125	NÃO	5	**normal**
8	0,039	NÃO	3	**baixo**
...	...	...	...	...

Tabela 5.1: *Amostra da tabela de avaliações.*

realizado pelo COGNARE. Finalmente, a Seção 5.1.4 apresenta os resultados obtidos na utilização do COGNARE para auxiliar nas tomadas de decisões em um COD da CELG.

5.1.1 Etapa 1: Obtenção dos dados junto a um especialista

Nesta etapa, foi empregada a seguinte metodologia. Inicialmente, foram gerados alguns cenários de decisão que ocorrem frequentemente nos COD's da CELG. Para isto, foi desenvolvida a interface da Figura 5.2. Conforme pode ser visto nesta figura, os parâmetros de entrada são a quantidade de tarefas e de recursos para cada cenário, bem como a quantidade de cenários que será gerada.

Estes cenários, após terem sido gerados, foram apresentados para um especialista com o objetivo que ele os avaliasse. Esta apresentação, foi realizada com o auxílio da interface da Figura 5.3, desenvolvida para este fim. Nesta interface, é apresentado o mapa da cidade, algumas ocorrências e viaturas, nas quantidades informadas na interface da Figura 5.2. As ocorrências estão representadas com um ponto de exclamação enquanto que as viaturas são representadas por caminhões. A ocorrência com uma cruz ao seu lado, indica que o problema relacionado com ela oferece risco à vida humana. Isto significa que esta ocorrência pode representar um fio partido, padrão dando choque, um poste caído, Unidade de Tratamento Intensivo (UTI), etc.

Na Figura 5.3, a linha que liga a ocorrência a uma viatura representa a sugestão proposta neste cenário, ou seja, a atribuição da ocorrência para uma viatura. Nesta interface, o especialista deve avaliar esta sugestão, bastando, para isto, escolher uma das opções fornecidas na caixa de sugestão apresentada na tabela, ao lado do mapa da Figura 5.3.

Ao todo foram gerados 100 cenários para avaliação do especialista.

O especialista levou em consideração as seguintes variáveis na sua avaliação:

- A distância, em quilômetros, entre a ocorrência e a viatura;
- Um valor, previamente calculado pelo sistema OPER, que representa a prioridade de atendimento da ocorrência;

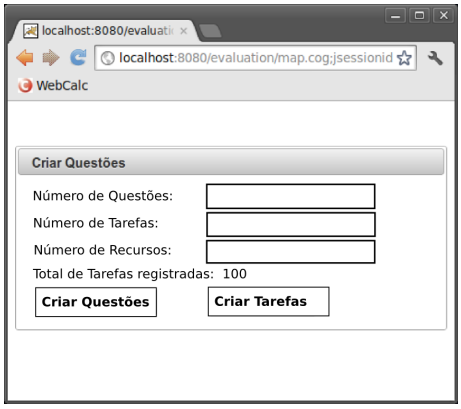


Figura 5.2: *Interface utilizada para criação dos cenários utilizados na aprendizagem do sistema.*

- Um valor que indica se a ocorrência representa um risco à vida humana;
- Proximidade entre as viaturas pois, na tomada de decisão, objetiva-se ao máximo, gerenciar as viaturas de tal forma que a distância entre elas seja a maior possível.

Para obter mais informações, referentes à viaturas ou à ocorrências, o avaliador pode clicar nestes elementos. A Figura 5.4 ilustra esta ação. Conforme pode ser visto, são apresentados mais detalhes sobre o cenário avaliado.

O resultado final desta avaliação foi expresso na Tabela 5.1, onde está uma amostra destes resultados.

5.1.2 Etapa 2: Aprendizagem das Regras

A aprendizagem das regras ocorreu da seguinte forma. Inicialmente, o COGNARE foi configurado para utilizar a classe *SGERDLearner*, implementada para realizar o aprendizado conforme apresentado na Seção 4.3. Detalhes desta configuração é apresentado na Seção 4.3.4.

Foram utilizados os dados, obtidos na Seção 5.1.1, para gerar as regras *Fuzzy*.

Ao todo, para os dados da avaliação realizada com o especialista, o COGNARE gerou 23 regras. As funções de pertinência estão apresentadas na Figura 5.5. Algumas das regras geradas estão apresentadas na Tabela 5.2.

Nr.	Regra
1	*if distancia is longe then coeficienteDecisao is muitoBaixo*
2	*if distanciaViatura is muitoPerto then coeficienteDecisao is muitoBaixo*
3	*if distancia is muitoPerto then coeficienteDecisao is muitoBaixo*
...	...

Tabela 5.2: *Exemplo de regras geradas pelo COGNARE a partir dos dados da Tabela 5.1.*

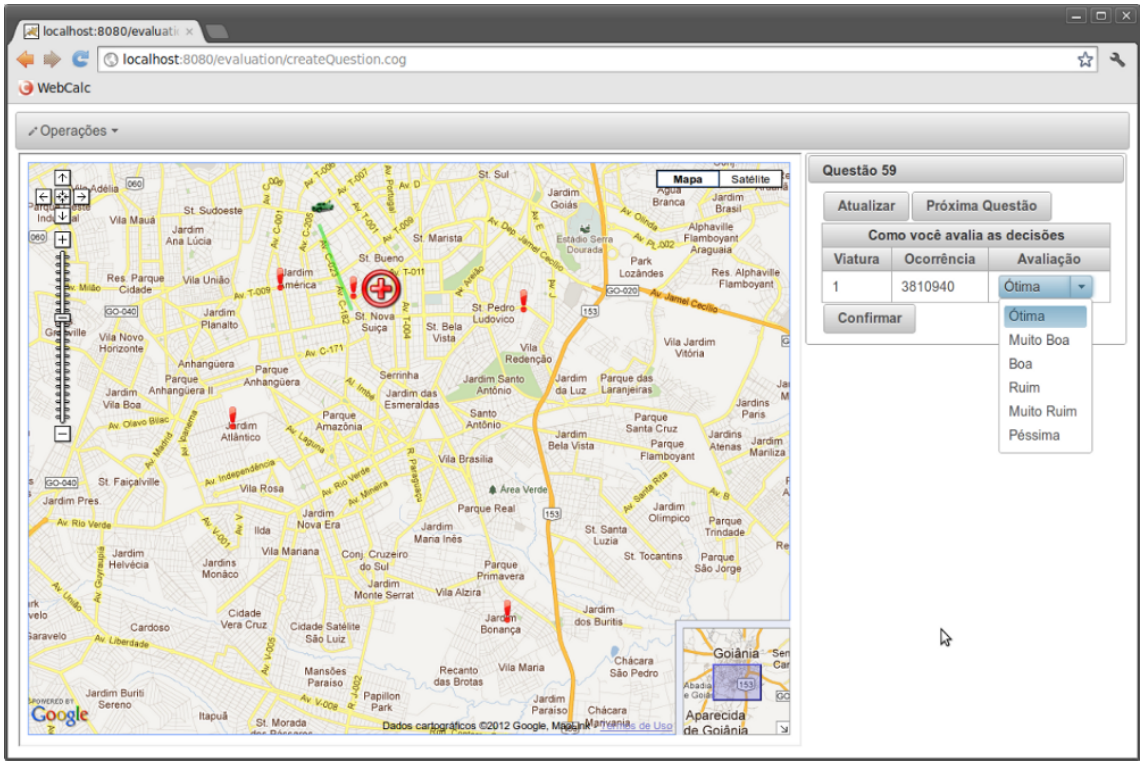


Figura 5.3: *Exemplo de um cenário com cinco ocorrências e uma viatura.*

5.1.3 Etapa 3: Tomada de decisão e apresentação das sugestões

Após a obtenção das regras *Fuzzy* na etapa anterior, o próximo passo consistiu na utilização dos módulos "Agentes Sensores" e "Alocador *Fuzzy*" (ambos apresentados no Capítulo 4) para gerar sugestões quanto à alocação de ocorrências a viaturas.

O COGNARE foi configurado para obter os dados, referentes à viaturas livres e ocorrências abertas, contidos no banco de dados do sistema OPER. Detalhes desta configuração foram descritos na Seção 4.2.5. Outra configuração realizada, e também apresentada naquela Seção, foi a definição do *Calculator EuclidianDistance* para cálculo da distância euclidiana entre uma viatura e um recurso. Após as configurações, os agentes do COGNARE foram inicializados e, desta forma, foram capazes de gerar sugestões para os operadores do COD.

Nesta etapa, foram desenvolvidas as duas interfaces do sistema SATE 01. A primeira, apresentada na Figura 5.6, possui o objetivo de mostrar as sugestões, fornecidas pelo COGNARE aos operadores do COD. A utilização desta interface é bastante simples. O operador escolhe de qual COD e de qual mesa deseja visualizar as sugestões. Após esta escolha, o sistema apresenta as sugestões, automaticamente, na tabela "Sugestões".

A segunda interface, apresentada na Figura 5.7, mostra graficamente as mesmas sugestões da interface anterior. O objetivo desta interface é simplificar a visualização das sugestões, realizadas pelo COGNARE. Desta forma, o especialista pode avaliá-las melhor

(a) *Informações da ocorrência.* (b) *Informações da viatura.*

Figura 5.4: *Exemplo de detalhamento de informações da ocorrência e da viatura.*

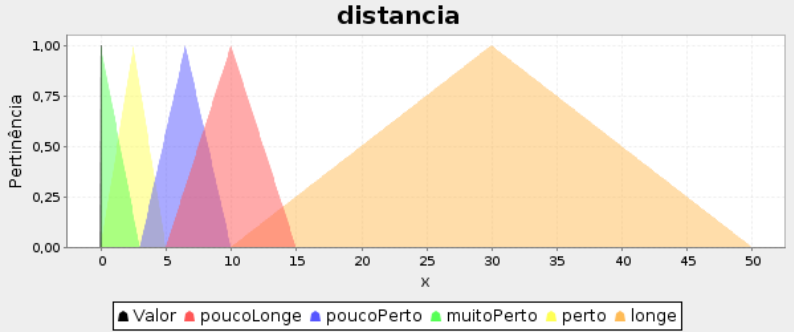
pois, através do mapa, é possível ter uma melhor noção da distância, localização e outras variáveis utilizadas na tomada da decisão.

5.1.4 Resultados deste estudo de caso

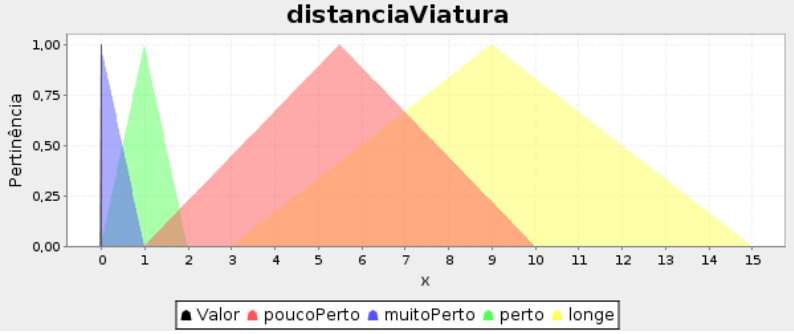
A avaliação dos resultados deste estudo de caso ocorreu da seguinte forma. O COGNARE foi inicializado sem que os operadores tomassem conhecimento de suas sugestões. O objetivo foi impedir que as sugestões oferecidas pelo COGNARE interferissem nas decisões dos operadores. Desta forma, tornou-se possível comparar as sugestões do COGNARE com as decisões dos operadores do COD da CELG.

Os dados utilizados para comparar a eficiência do COGNARE em auxiliar na tomada de decisão da CELG, foram as sugestões apresentadas pelo COGNARE, juntamente com as decisões tomadas pelos operadores. Foram colhidas todas as decisões tomadas em uma única mesa de operação e em dois período de 24 horas. Cada um destes períodos se referiam a duas situações diferentes. A primeira situação se referia às decisões tomadas em um dia chuvoso. Nestes dias, nota-se um aumento significativo na quantidade de ocorrências. Na segunda situação, os dados se referiam às decisões que foram tomadas em um dia ensolarado.

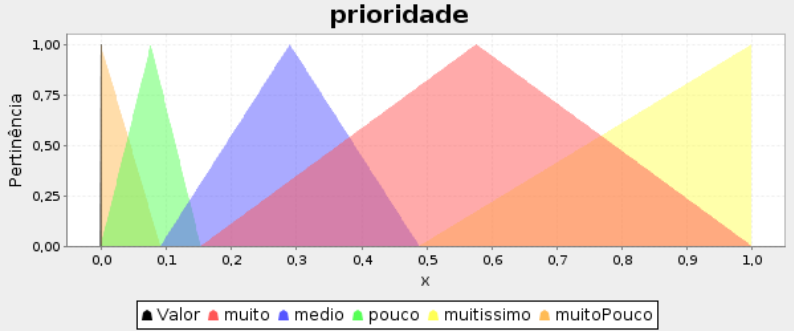
Estes dados foram apresentadas a um especialista, para que ele os avaliasse. O resultado desta avaliação está apresentado na Tabela 5.3. Nesta tabela, é exibida uma comparação da taxa de acertos que o COGNARE obteve com a taxa de acertos dos operadores do COD da CELG. A coluna “Qtd. de Decisões em 24h” se refere à quantidade de decisões, tomadas pelos operadores do COD de uma mesma mesa de operação e em um período de 24 horas.



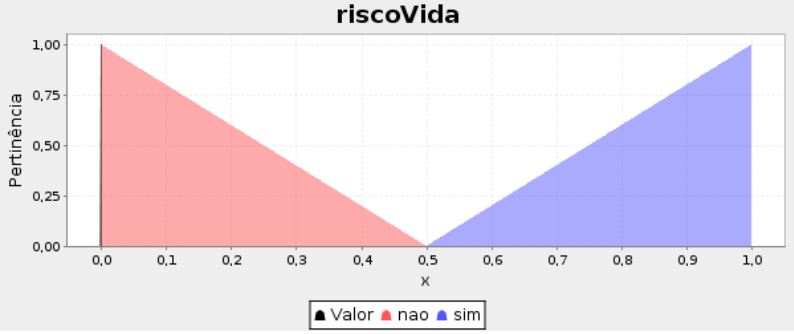
(a) *Variável distancia*



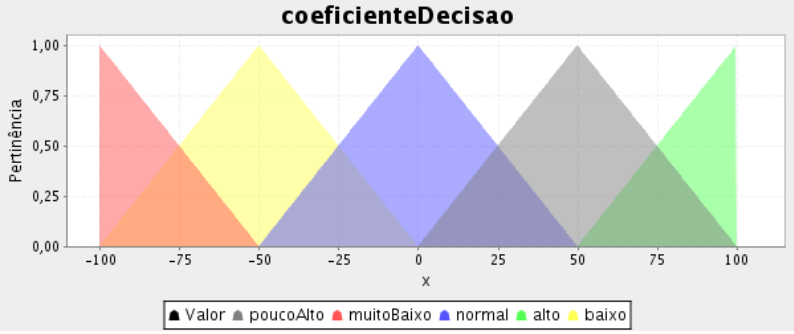
(b) *Variável distanciaViatura*



(c) *Variável prioridade*



(d) *Variável riscoVida*



(e) *Variável coeficienteDecisao*

Figura 5.6: *Interface utilizada para apresentar ao operador do COD sugestões de atribuição.*

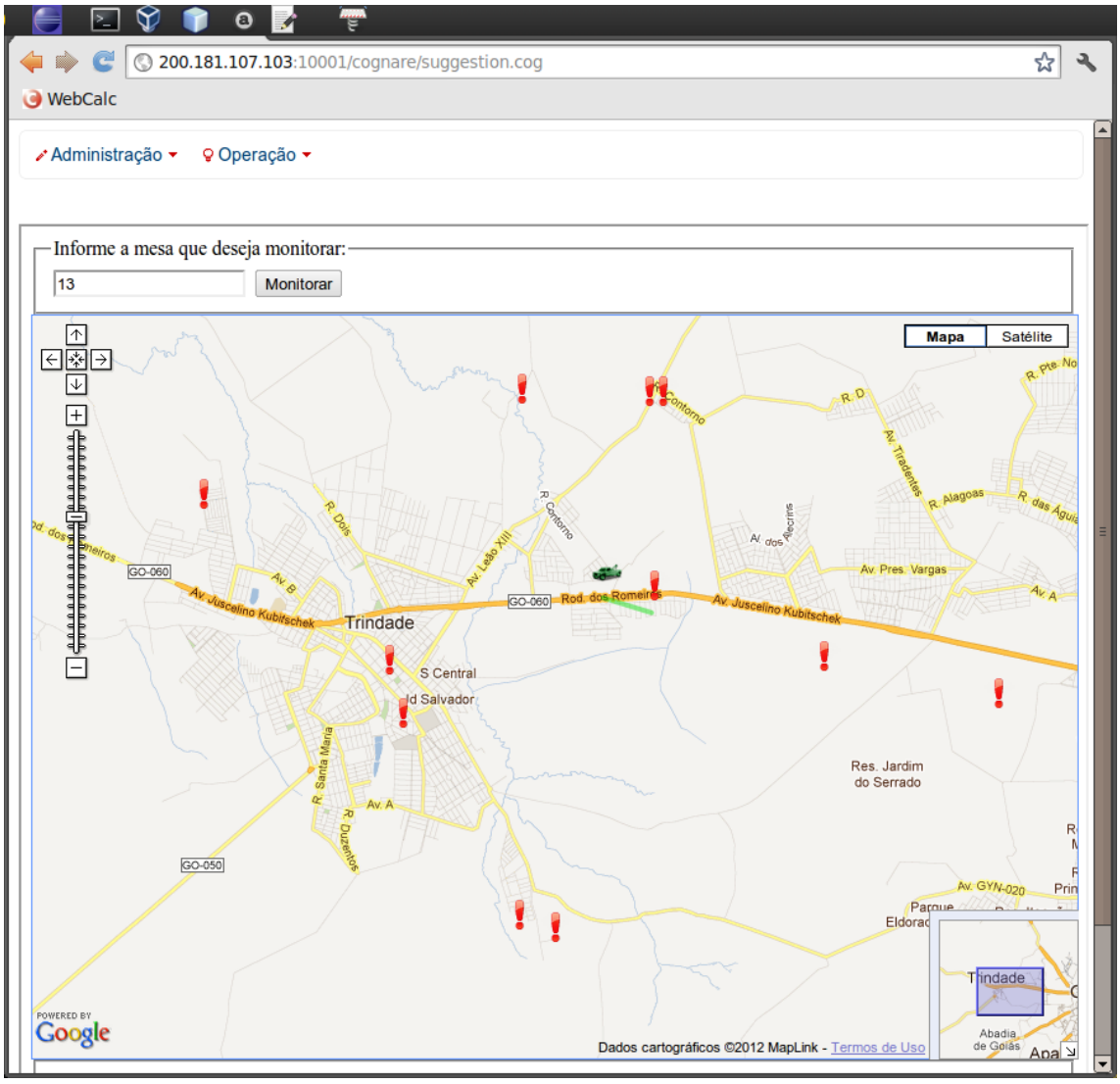


Figura 5.7: *Apresentação, no mapa, das sugestões apresentadas pelo COGNARE para avaliação de um especialista.*

Situação do Tempo	Qtd de Ocorrências Abertas	Qtd. de Decisões em 24h	Acertos COGNARE(%)	Acertos Operador (%)
Chuvoso	1230	125	86	67
Ensolarado	109	112	81	73

Tabela 5.3: *Comparação da taxa de acertos do COGNARE com a taxa de acertos do operador da CELG, segundo a avaliação do especialista.*

Nota-se, na Tabela 5.3, que o COGNARE obteve melhores taxas de acertos, tanto em dias chuvosos como em dias ensolarados. Este fato confirma a utilização da Aprendizagem Evolutiva e da Lógica *Fuzzy* no auxílio à resolução de problemas do tipo apresentado neste estudo de caso.

5.2 Estudo de Caso 2 - Alocação dinâmica de recursos em um balanceador de carga para o sistema receptor da Nota Fiscal Eletrônica (NF-e) do Estado de Goiás

Com o advento do projeto da Nota Fiscal Eletrônica (NF-e) em todo o Brasil, os estados e o Distrito Federal são responsáveis por recepcionar arquivos digitais que representam as Notas Fiscais Eletrônicas. Desta forma, um contribuinte, ao invés de imprimir uma Nota Fiscal em papel, deve gerar este arquivo XML, assiná-lo com certificado um digital ICP-Brasil[4] válido e transmiti-lo à receita estadual.

Com o passar do tempo, a quantidade de Notas Fiscais recepcionadas tem aumentado significativamente. Isto ocorre devido a dois fatores principais. O primeiro é que, a cada dia, novas empresas são obrigadas a deixarem de emitir Notas Fiscais em papel para começarem a emiti-las no seu formato digital. O segundo, é o crescimento do volume de negócios das empresas que já emitem NF-e o que implica em um aumento nas emissões de NF-e's.

Neste contexto, surge a necessidade de escalonar os recursos de *hardware*, utilizados para recepcionar tais arquivos. Uma das forma de realizar este escalonamento é com o balanceamento de carga.

[4]ICP-Brasil: Infraestrutura de chaves públicas do Brasil. Trata-se do Sistema Nacional de Certificação digital e é uma estrutura composta de um ou mais certificadores denominados de Autoridades Certificadoras (AC) que, através de um conjunto de técnicas e procedimentos de suporte a um sistema criptográfico, baseando-se em certificados digitais, consegue assegurar a identidade de um usuário de mídia eletrônica ou assegurar a autenticidade de um documento suportado ou conservado em mídia eletrônica.

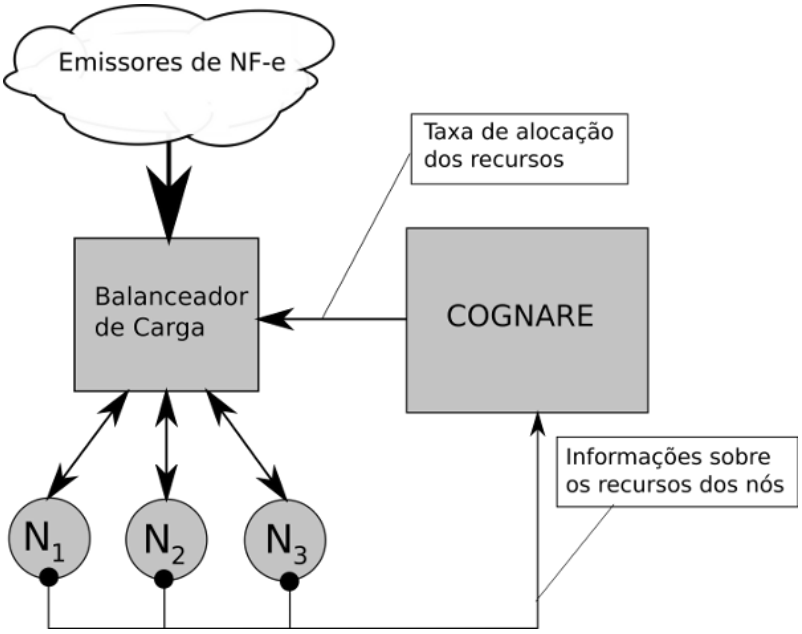


Figura 5.8: *COGNARE utilizado com um balanceador de carga na NF-e.*

Este estudo de caso utilizou o COGNARE com o objetivo de auxiliar um balanceador de carga a alocar dinamicamente recursos de *hardware* para o sistema receptor da NF-e[5] de Goiás. A Figura 5.8 ilustra esta integração. Nesta Figura, os elementos N_1, N_2 e N_3 são os servidores, que recepcionam Notas Fiscais Eletrônicas.

O balanceamento de carga[9] é uma técnica utilizada para distribuir a carga de trabalho uniformemente entre dois ou mais recursos, tais como rede, CPUs, discos rígidos, etc. O objetivo é otimizar a utilização destes recursos, maximizar seu desempenho, minimizar o tempo de resposta e evitar sobrecarga. Conforme ilustrado na Figura 5.8, o balanceador é capaz de dividir as requisições, originadas dos emissores de NF-e (neste caso, os contribuintes) para uma quantidade de nós que seja necessária. Nesta Figura, foram ilustrados três nós.

Geralmente, os balanceadores de carga utilizam um escalonador de tarefas do tipo *"Round-Robin"* [31] [33]. No escalonamento de tarefas *"Round-Robin"*, as tarefas são armazenadas em uma fila circular. O escalonador percorre esta fila e aloca o recurso para cada uma destas tarefas, conforme ilustrado na Figura 5.9. Desta forma, cada tarefa tem uma probabilidade de ser executada igual a $\frac{1}{n}$, onde n é o número total de tarefas a serem alocadas.

Portanto, podem ocorrer sobrecargas em um dos nós, devido a diversos fatores. Um destes fatores, que ocorre frequentemente, é quando uma determinada requisição ocasiona um consumo excessivo de recursos. Por exemplo, quando um arquivo muito

[5] http://www.nfe.fazenda.gov.br

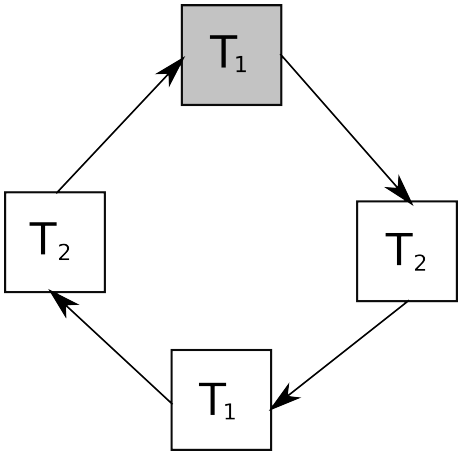


Figura 5.9: *Escalonamento Round-Robin.*

CPU N_1 (%)	CPU N_2 (%)	Taxa (%) N_1|N_2
64	21	79|21
53	47	63|37
7	89	11|89
87	13	81|19

Tabela 5.4: *Tabela utilizada no aprendizado das regras. Aqui estão apenas alguns dos valores obtidos.*

grande é transmitido. O nó que receber este arquivo sofrerá um aumento significativo em sua carga o que, inevitavelmente, provocará um desbalanceamento. Outro fator, que também pode ocorrer, é quando os nós não são idênticos, ou seja, existem *hardwares* diferentes recepcionando arquivos. Neste caso, se a mesma quantidade de requisições for enviada a cada um dos nós, os que possuírem menores quantidades de recursos, como processamento e memória, trabalharão em seus limites de funcionamento, ocasionando uma significativa lentidão ou até mesmo podendo falhar.

O COGNARE é capaz de auxiliar o balanceador através da alteração dinâmica da porcentagem de requisições enviadas a cada um dos nós deste balanceador. Desta forma, as entradas devem ser os estados atuais do *hardware*, como memória, CPU, disco, rede, etc. Neste estudo de caso, somente a carga da CPU foi utilizada como entrada, mas nada impede a utilização de mais variáveis.

O processo de aprendizado das regras, necessárias para o módulo alocador *Fuzzy* foi realizado da seguinte forma. Inicialmente, foram enviadas várias requisições ao balanceador, sem que este estivesse ligado ao COGNARE. Durante o envio destas requisições, foram alteradas, de forma aleatória, as porcentagens de carga enviada para cada um dos nós. Posteriormente foram medidas as variações de velocidade de processamento do sistema e os valores deram origem à Tabela 5.4.

CPU N_1	CPU N_2	Taxa N_1
H	L	L
M	M	H
L	H	VH

Tabela 5.5: *Valores da Tabela 5.4 fuzzificados. Saída referente ao comportamento do nó 1.*

CPU N_1	CPU N_2	Taxa N_2
H	L	H
M	M	L
L	H	VL

Tabela 5.6: *Valores da Tabela 5.4 fuzzificados. Saída referente ao comportamento do nó 2.*

No.	Regra
1	if CPU_1 and CPU_2 is MEDIUM then $RATE_1$ is MEDIUM
2	if CPU_1 is LOW then $RATE_1$ is HIGHT

Tabela 5.7: *Algumas das regras obtidas após o processo de aprendizagem.*

As funções de pertinência, ilustradas na Figura 5.10, foram utilizadas para fuzzificar os valores da Tabela 5.4. Nesta Figura, VL, L, M, H e VH significam *Very Low, Low, Medium, High e Very High* respectivamente. Após este passo, foram geradas as Tabelas 5.5 e 5.6. A partir destas tabelas foram geradas as regras *Fuzzy*. Algumas destas regras estão representadas na Tabela 5.7.

O agente *"ResearchAgent"* do COGNARE deve ser instalado em cada um dos nós. Sua função é obter o valor da carga da CPU, de cada um destes nós, e enviá-la ao COGNARE, conforme ilustrado na Figura 5.8. Nesta Figura, esses agentes são representados por circunferências pretas, fixadas em cada um dos nós.

5.2.1 Resultados deste estudo de caso

Com o objetivo de verificar a eficiência do COGNARE, integrado a um balanceador de carga, foram realizados três tipos de testes diferentes:

- T_1: Nós sem carga: Os nós apenas recebiam requisições e não executavam nenhuma tarefa, ou seja, não tinham uma carga significativa para cada requisição;
- T_2: Nós com carga: A cada requisição, os nós realizavam um processamento que provocava o aumento da carga de trabalho das CPUs;

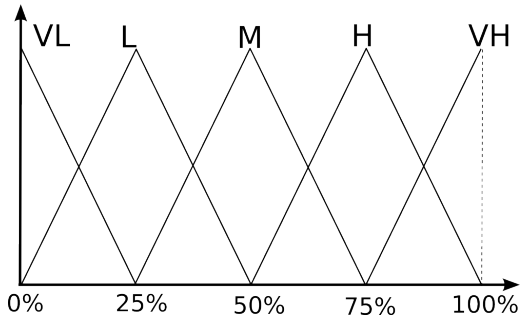


Figura 5.10: *Funções de pertinência utilizadas para fuzzificar os valores da Tabela 5.4.*

- T_3: Carga extra em um dos nós: Além da carga adicionada no teste T_2, em um dos nós foi executado um processo que aumentava significativamente a carga de trabalho da CPU.

Foram utilizados três computadores em rede, com configurações diferentes, conforme apresentado na Tabela 5.8 e ilustrado na Figura 5.11. O computador com denominação C_3 é o que possui menos recursos dos três. Este fato auxiliou na verificação da eficiência do balanceador na situação em que os *hardwares* são diferentes.

Denominação	Utilização	Qtd. núcleos	CPU	Memória RAM
C_1	Cognare e Balanceador	4	800 MHz	4 GB
C_2	Nó 1	4	800 MHz	6 GB
C_3	Nó 2	2	1200 MHz	2 GB

Tabela 5.8: *Configuração dos computadores utilizados nos testes.*

Conforme ilustrado na Figura 5.11, o computador C_1 foi responsável por executar a geração de requisições para o teste de carga, o balanceador de carga e o COGNARE. Os computadores C_2 e C_3 ficaram com os nós do balanceador.

Na Tabela 5.9 foram descritos os modelos de cada um dos computadores utilizados nestes testes.

Denominação	Configuração
C_1	AMD Phenom(tm) II X4 955 Processor - 4 GB RAM
C_2	Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz - 6 GB RAM
C_3	AMD Turion(tm) 64 X2 Mobile Technology TL-50 1.2GHZ - 2GB RAM

Tabela 5.9: *Modelos dos computadores utilizados nos testes.*

Os testes consistiram em enviar 2000 requisições para o balanceador e medir o tempo gasto para processá-las. A partir deste tempo, foi calculada a velocidade do

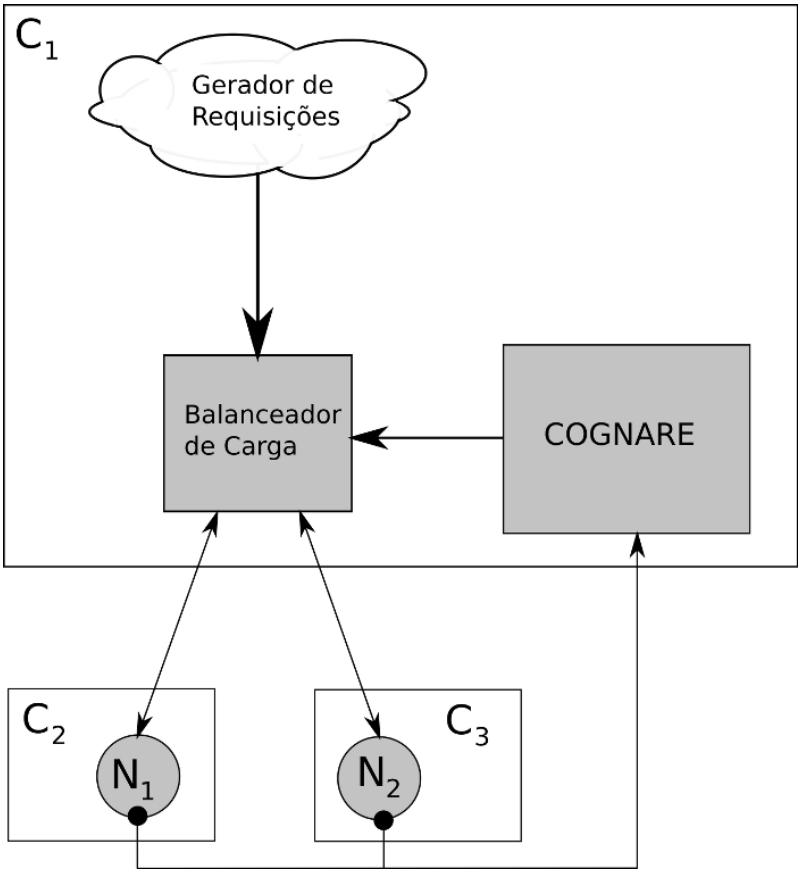


Figura 5.11: *Esquema de utilização e ligação dos computadores utilizados nos testes.*

processamento de cada uma das requisições. Esta velocidade foi medida em requisições por minuto. Desta forma, quanto maior for esta velocidade mais eficiente o será o sistema. Cada um dos três testes foram repetidos 20 vezes, sendo 10 vezes com o COGNARE alterando dinamicamente a carga dos nós e 10 vezes utilizando o algoritmo *Round-Robin*, cuja proporção é sempre a mesma e, neste caso, igual a 50% para cada nó.

Após a sequência de execução dos testes, foi calculada a quantidade de requisições antendidas por minuto. Estes valores foram apresentados na Tabela 5.10.

Teste	COGNARE	*Round-Robin*
T_1	1484 req/min	804 req/min
T_2	96 req/min	68 req/min
T_3	67 req/min	19 req/min

Tabela 5.10: *Quantidade de requisições atendidas por minuto pelo balanceador em cada um dos três testes realizados.*

Conforme pode ser verificado na Tabela 5.10, a utilização do COGNARE, como auxiliar no balanceamento de carga, aumentou significativamente a capacidade de

processamento de todo o sistema. Um destaque deve ser dado ao teste T_3. Neste teste nota-se que a utilização do COGNARE provocou um aumento de $3,5$ vezes quando comparado com o *Round Robin*. Isto se justifica pois, com uma carga desigual em um dos nós, este respondeu bem mais lentamente. Neste caso a utilização de uma alocação dinâmica provocou um considerável aumento na eficiência do sistema como um todo.

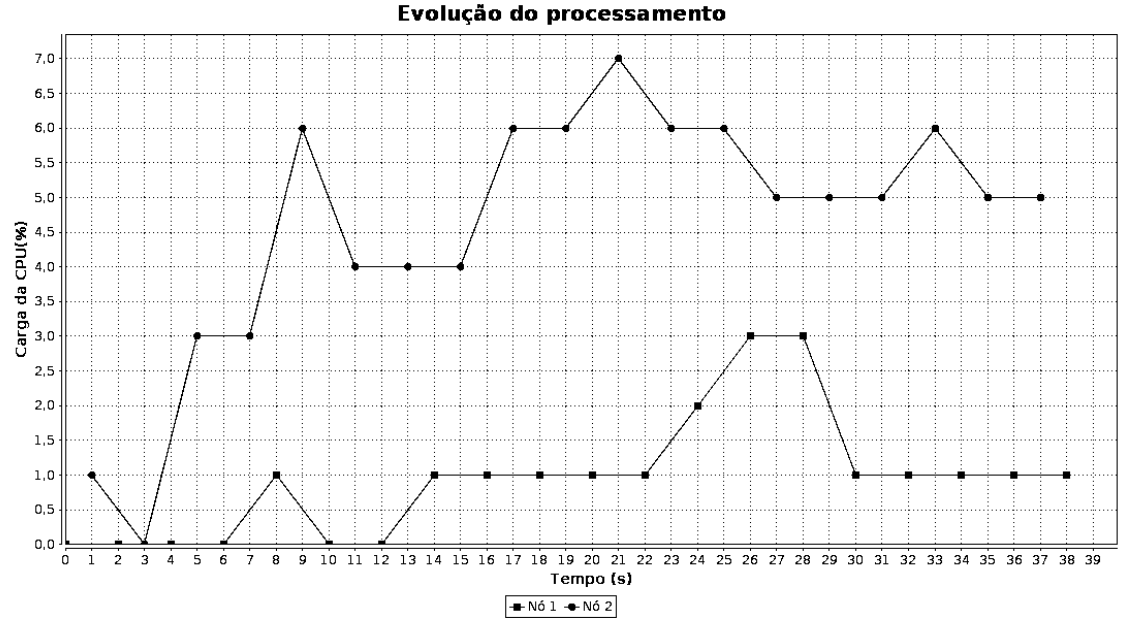
Após a obtenção destes dados, foram executados os três testes mais uma vez. Desta vez o objetivo foi gerar gráficos que apresentassem o comportamento de cada um dos nós. Estes gráficos estão nas Figuras 5.12, 5.13, 5.14.

O gráfico da Figura 5.12 apresenta o comportamento dos nós no teste T_1. Neste teste, os nós não possuíam uma carga significativa, o que justifica a baixa utilização da CPU. Nota-se que em *(b)* houve um melhor balanceamento da carga do que em *(a)* pois, apesar da diferença significativa entre os *hardwares* do nó 1 e do nó 2 o gráfico em *(b)* mostra as linhas mais próximas entre si do que em *(a)*.

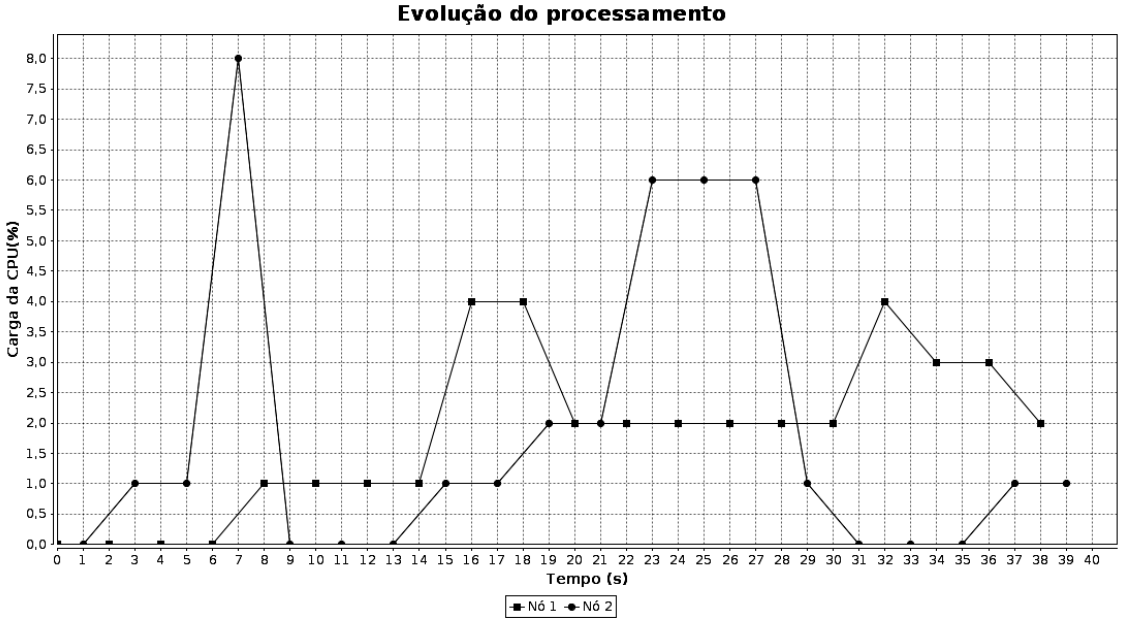
O gráfico da Figura 5.13 apresenta o comportamento dos nós no teste T_2. Neste teste, os nós possuíam uma carga, o que elevou significativamente a utilização da CPU, em relação ao teste anterior. Nota-se que a utilização do COGNARE melhorou o desempenho do balanceador de carga pois o pico de utilização da CPU foi 42% (situação apresentada em *(b)*) contra $62,5\%$ referente à utilização do *Round-Robin* (situação apresentada em *(a)*). Outra vantagem, verificada na utilização do COGNARE, é que este apresentou um balanceamento mais eficiente pois, mesmo com as diferenças nos *hardwares* dos nós, estes se comportaram de forma similar, diferentemente da situação apresentada em *(a)*, onde ficou evidente que o nó 1 trabalhou menos que o nó 2.

O gráfico da Figura 5.14 apresenta o comportamento dos nós no teste T_3. Neste teste, os nós possuíam uma carga, similar à carga do teste anterior. Além disto, ao nó 2 foi adicionada uma carga extra, oriunda de um outro processo. O objetivo foi testar o balanceador em uma situação em que um dos nós sofre uma sobrecarga em relação ao outro. Conforme apresentado na Tabela 5.8, o computador que executa o nó 2 possuía apenas dois núcleos. Portanto a carga extra foi tal que um destes núcleos chegou ao seu limite. Este cuidado foi tomado para certificar de que a carga extra adicionada a este nó foi suficientemente alta para provocar um desbalanceamento. Conforme pode ser verificado, ao comparar *(a)* com *(b)*, a utilização do COGNARE evitou que o nó 1 chegasse a extremos de utilização de CPU. Outro fato constatado, após a visualização dos gráficos, é que, também neste caso, a utilização do COGNARE provocou um melhor balanceamento da carga.

Desta forma, os resultados deste estudo de caso foram satisfatórios e mostraram que o COGNARE é capaz de elevar a capacidade de um sistema balanceador de carga. Devido a estes resultados, o COGNARE, integrado ao balanceador de carga, já se encontra em produção, auxiliando na recepção das Notas Fiscais Eletrônicas do Estado de Goiás.

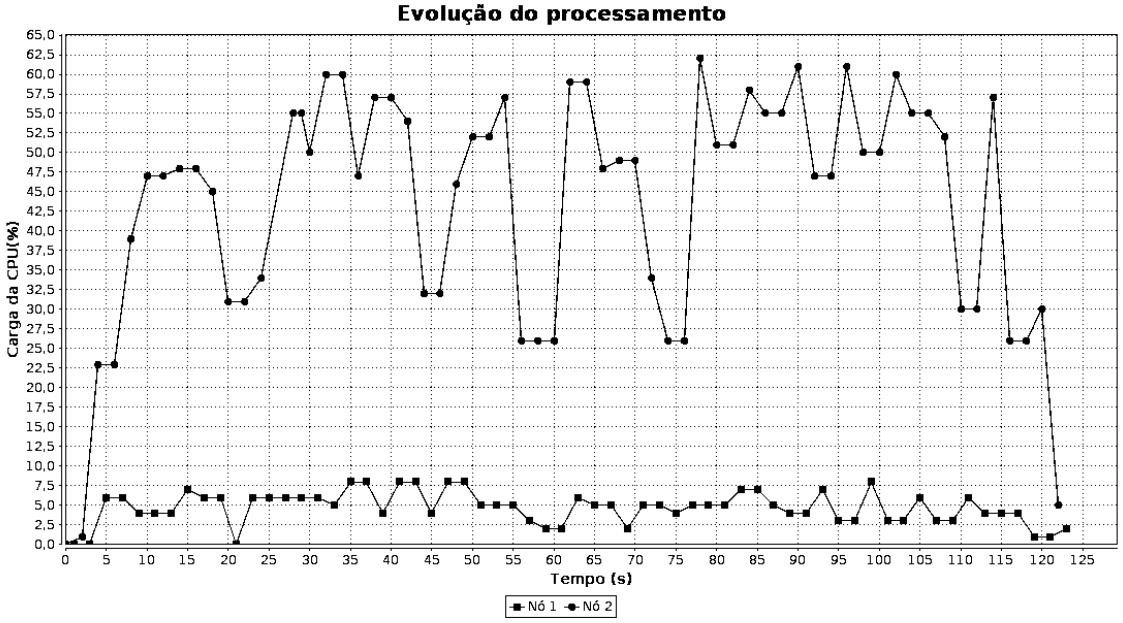


(a) *Balanceador de carga com "Round Robin".*

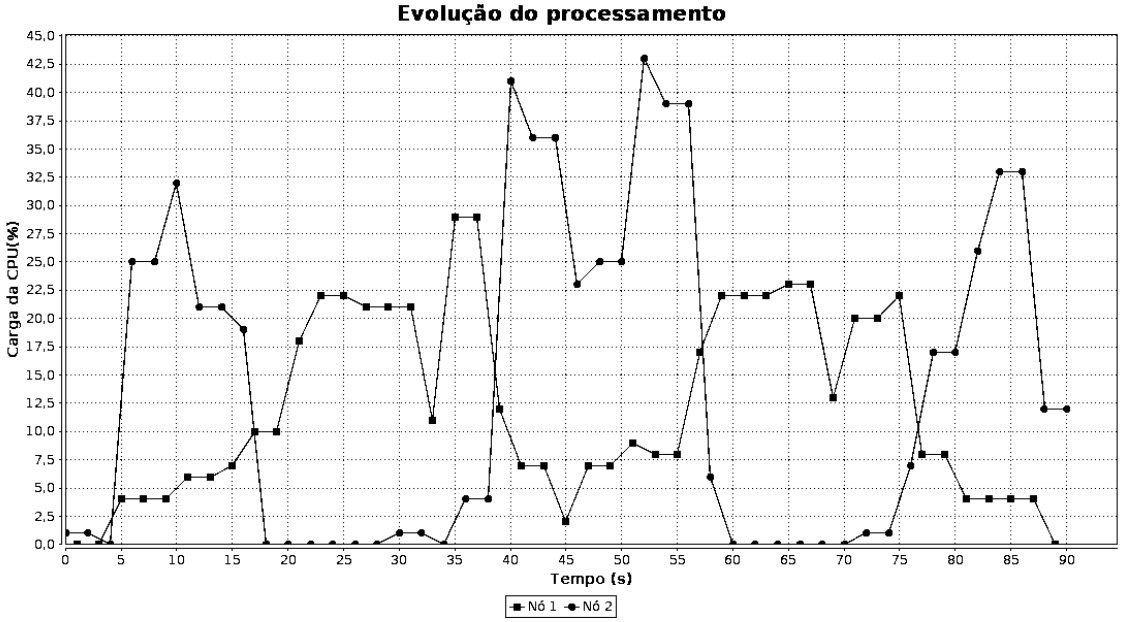


(b) *Balanceador de carga com COGNARE.*

Figura 5.12: *Teste 1: Nós sem carga.*

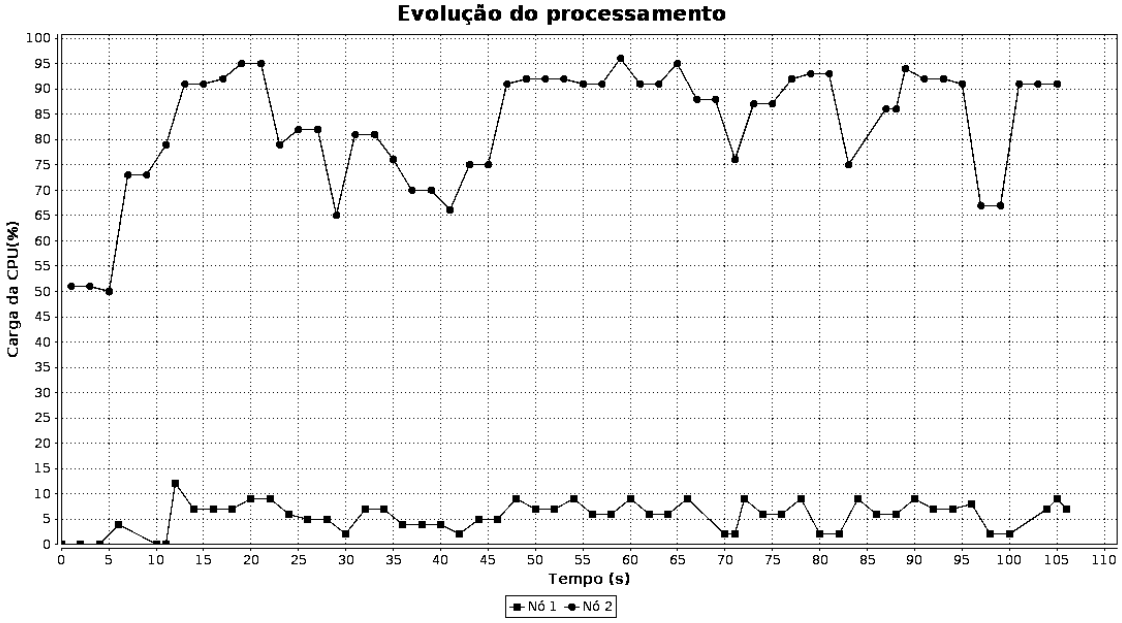


(a) *Balanceador de carga com "Round Robin".*

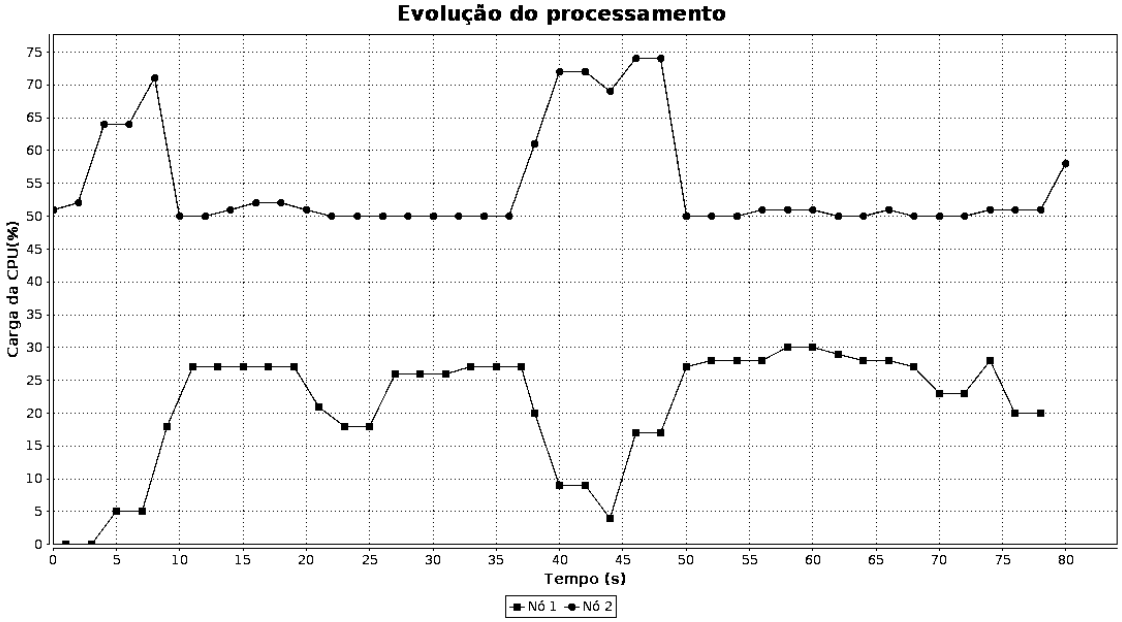


(b) *Balanceador de carga com COGNARE.*

Figura 5.13: *Teste 2: Nós com carga.*



(a) *Balanceador de carga com "Round Robin".*



(b) *Balanceador de carga com COGNARE.*

Figura 5.14: *Teste 3: Carga em apenas um dos nós.*

CAPÍTULO 6

Considerações Finais

Após a apresentação deste trabalho e dos resultados alcançados nos estudos de caso, a Seção 6.1 apresenta algumas conclusões e a Seção 6.3 direciona os trabalhos futuros com sugestões de aprimoramento e aplicações em novas áreas.

6.1 Conclusão

O problema da alocação dinâmica de recursos está presente em diversas áreas da sociedade. Em alguns casos, esta alocação é realizada por seres humanos, que estão sugeitos a falhas. Em outros, esta alocação é realizada de forma automática, mas estática [31] [33]. Isto quer dizer que, se alguma condição for alterada, estes sistemas não são capazes de acompanhá-las. Este trabalho apresentou uma proposta de emprego dos Algoritmos Genéticos, Lógica *Fuzzy* e Sistemas Multiagentes na resolução deste problema, a alocação dinâmica de recursos. O resultado foi a concepção do COGNARE.

No COGNARE, os Algoritmos Genéticos foram empregados com o objetivo de gerar regras *Fuzzy*. Especificamente foi empregado o SGERD [24], um algoritmo capaz de gerar regras. Posteriormente foi concebido um Sistema de Inferência *Fuzzy* cujo objetivo foi o de realizar a inferência em função das variáveis de entrada, utilizadas na alocação de recursos. Agentes de *software* foram adicionados com o objetivo de integrar o COGNARE com o mundo exterior, que contém as variáveis necessárias no processo de decisão quanto a alocação.

Este trabalho avaliou a utilização do COGNARE em duas situações diferentes. A primeira situação foi apresentada no estudo de caso da Seção 5.1. Neste caso, o cognare auxiliou na alocação dinâmica de viaturas no COD da CELG. Estas viaturas são responsáveis por resolver problemas relacionados com falhas no fornecimento de energia elétrica. Conforme apresentado naquela Seção o COGNARE mostrou-se eficiente no auxílio na tomada de decisão quanto à alocação das viaturas responsáveis por atenderem as falha no fornecimento de energia elétrica. A segunda situação, onde o COGNARE pôde ser empregado com sucesso, foi na alocação dinâmica de recursos de *hardware*, utilizados na Nota Fiscal Eletrônica do Estado de Goiás. Este tipo de emprego, é também

denominado de balanceamento de carga [9]. Neste ponto, o COGNARE se mostrou capaz de balancear a carga dos servidores que recepcionam a NF-e.

Com isto, conclui-se que o algoritmo SGERD é eficiente para a geração de regras *Fuzzy*, úteis para um Sistema de Inferência *Fuzzy* ser capaz de realizar inferência, tanto na alocação de viaturas na CELG, quanto na alocação de recursos da NF-e.

6.2 Contribuições

A contribuição do COGNARE se dá em problemas relacionados com a alocação de recursos. Principalmente nos casos em que estes problemas possuem muitas variáveis. Estes casos são muito difícieis de serem resolvidos pois demandam um grande esforço humano na descoberta do grau de importância de cada uma destas variáveis.

O sistema proposto por este trabalho busca resolver esta questão com a utilização de Algoritmos Genéticos no aprendizado do grau de importância de cada uma das variáveis utilizadas na tomada de decisão. Este aprendizado pode ser realizado a partir de uma base de dados histórica.

Desta forma, a partir de uma base de dados histórica que contenha as decisões realizadas quanto à alocação de recursos, é possível empregar o COGNARE em diversas áreas da sociedade, como por exemplo:

- Alocação de equipes a problemas que ocorrem na rede elétrica;
- Alocação de ambulâncias a casos de acidentes;
- Alocação de casos de usos a programadores em uma fábrica de software.

Na etapa final deste trabalho, foi escrito o artigo *"Evolutionary Learning and Fuzzy Logic applied to a Load Balancer"* que apresenta o Estudo de Caso 5.2. Este artigo foi aceito na *The 24th International Conference on Software Engineering and Knowledge Engineering*.

6.3 Trabalhos Futuros

Como trabalhos futuros, é sugerida a melhoria no algoritmos de escolha da melhor decisão, apresentado na Seção 4.4. Conforme apresentado naquela Seção, este trabalho utiliza uma estratégia gulosa na escolha da melhor decisão. Sugere-se a utilização de programação dinâmica para resolver este caso.

Outra sugestão, está na melhoria do algoritmo de cálculo da distância, empregado na Seção 5.1. Este trabalho utilizou o cálculo da distância euclidiana. Nota-se que a taxa de acertos do COGNARE, apresentada na Tabela 5.3, pode ser melhorada se empregado o cálculo da distância considerando as vias de acesso, trânsito e horários do dia.

Sugere-se também tornar o acionamento do aprendizado das regras automático e constante. Conforme pôde ser visto nos estudos de caso das Seções 5.1 e 5.2, o aprendizado foi acionado de forma manual. A proposta é tornar este aprendizado constante.

Finalmente, é sugerido o emprego de outros algoritmos/técnicas, capazes de realizar o aprendizado de regras a partir de uma base de dados, tais como HIDER* [2], FARC-HD [4], NSGA-II [12] [28], TARGET [16], dentre outros.

Referências Bibliográficas

[1] ABONYI, J.; ROUBOS, J. A.; SZEIFERT, F. **Data-driven generation of compact, accurate, and linguistically-sound fuzzy classifiers based on a decision-tree initialization**. *International Journal of Approximate Reasoning*, 32:1–21, 2002.

[2] AGUILAR-RUIZ, J. S.; GIRÁLDEZ, R.; RIQUELME, J. C. **Natural encoding for evolutionary supervised learning**. *IEEE Transactions on Evolutionary Computation, 11(4)*, p. 466–479, 2007.

[3] AGUILAR-RUIZ, J. S.; RIQUELME, J. C.; TORO, M. **Evolutionary learning of hierarchical decision rules**. *IEEE Transactions on systems, man, and cybernetics-part B: Cybernetics, VOl. 33, No. 2, April*, p. 324–331, 2003.

[4] ALCALÁ-FDÉZ, J.; ALCALÁ, R.; HERRERA, F. **A fuzzy associative classification system with genetic rule selection for high-dimensional problems**. *International Workshop on Genetic and Evolutionary Fuzzy Systems*, 2010.

[5] BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. **Developing Multi-Agent Systems with JADE**. Wiley, 2007.

[6] BERMAN, F.; FOX, G.; HEY, A. J. G. **Grid Computing: Making the Global Infrastructure a Reality**. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[7] BEZIAU, J.-Y. **What is many-valued logic?** In: *Multiple-Valued Logic, 1997. Proceedings., 1997 27th International Symposium on*, p. 117 –121, may 1997.

[8] CAMILO-JUNIOR, C. G.; YAMANAKA, K. **In Vitro Fertilization Genetic Algorithm, Evolutionary Algorithms**. Eisuke Kita (Ed.), 2011.

[9] CARDELLINI, V.; I, R.; COLAJANNI, M.; YU, P. S. **Dynamic load balancing on web-server systems**. *IEEE Internet Computing*, 3:28–39, 1999.

[10] COSTAGLIOLA, G.; FERRUCCI, F.; FUCCELLA, V. **A web-based e-testing system supporting test quality improvement**. In: Leung, H.; Li, F.; Lau, R.; Li, Q., editors, *Advances in Web Based Learning - ICWL 2007*, volume 4823 de **Lecture Notes in Computer Science**, p. 264–275. Springer Berlin / Heidelberg, 2008.

[11] DARWIN, C. **The Origin of Species**. Gramercy, May 1995.

[12] DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. **A fast and elitist multiobjective genetic algorithm: Nsga-ii**. *IEEE Transactions on Evolutionary Computation, 6(2)*, p. 182–197, 2002.

[13] ELOMAA, T.; ROUSU, J. **General and efficient multisplitting of numerical attributes**. In: *Machine Learning*, p. 201–244, 1999.

[14] FOWLER, A.; TEREDESAI, A.; DE COCK, M. **An evolved fuzzy logic system for fire size prediction**. In: *Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American*, p. 1 –6, june 2009.

[15] GAMMA, E.; HELM, R.; JOHNSON, R. E.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, Reading, MA, 1995.

[16] GRAY, J. B.; FAN, G. **Classification tree analysis using target**. *Computational Statistics Data Analysis, 52(3)*, p. 1362–1372, 2008.

[17] HERRERA, F. **Genetic fuzzy systems: taxonomy, current research trends and prospects**. *Evolutionary Intelligence*, 1:27–46, 2008. 10.1007/s12065-007-0001-5.

[18] HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. MIT Press, Cambridge, MA, USA, 1992.

[19] International Electrotechnical Commission (IEC). **IEC 1131 Programable Controllers. Part 7: Fuzzy Control Programming**, 1997.

[20] ISHIBUCHI, H.; ISHIBUCHI, H.; YAMAMOTO, T.; YAMAMOTO, T.; NAKASHIMA, T.; NAKASHIMA, T. **Hybridization of fuzzy gbml approaches for pattern classification problems**. *IEEE Trans. on Systems, Man, and Cybernetics - Part B*, 35:359–365, 2005.

[21] ISHIBUCHI, H.; YAMAMOTO, T.; ISHIBUCHI, P. H. **Comparison of heuristic criteria for fuzzy rule selection in classification problems**, 2004.

[22] J.R.QUINLAN. **C4.5: Programs for machine learning**. *Morgan Kaufman Publishers, 2929 Campus Drive, Suite 260 San Mateo, CA 94403*, 1993.

[23] L.A..; ZADEH. **Fuzzy sets**. *Information and Control*, 8(3):338 – 353, 1965.

[24] MANSOORI, E.; ZOLGHADRI, M.; .; KATEBI, S. **Sgerd: A steady-state genetic algorithm for extracting fuzzy classification rules from data**. *IEEE Transactions on Fuzzy Systems, 16(4)*, p. 1061–1071, 2008.

[25] MARQUES, V.; GOMIDE, F. **Fuzzy coordination of genetic algorithms for vehicle routing problems with time windows**. In: *Genetic and Evolutionary Fuzzy Systems (GEFS), 2010 4th International Workshop on*, p. 39 –44, march 2010.

[26] MITCHELL, T. M. **Machine Learning**. McGraw-Hill, New York, 1997.

[27] MOON, S. Y.; CHO, T. H. **Intrusion detection scheme against sinkhole attacks in directed diffusion based sensor networks**. *JCSNS International Journal of Computer Science and Network Security, VOL.9 No.7, July*, p. 118–122, 2009.

[28] PULKKINEN, P.; KOIVISTO, H. **Fuzzy classifier identification using decision tree and multiobjective evolutionary algorithms**. *International Journal of Approximate Reasoning 48*, p. 526–543, 2007.

[29] QUINLAN, J. R. **Improved use of continuous attributes in C4.5**. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[30] SANCHEZ, L.; COUSO, I.; CORRALES, J. A. **Combining GP operators with SA search to evolve fuzzy rule based classifiers**. *Information Sciences*, 136(1-4):175–191, 2001.

[31] SHREEDHAR, M.; VARGHESE, G. **Efficient fair queuing using deficit round robin**. *IEEE Trans. Net*, 1996.

[32] SILVEIRA, S. L. . P. S. . G. **Introdução à Arquitetura e Design de Software - Uma Visão Sobre a Plataforma Java**. ELSEVIER - CAMPUS, São Paulo, 2011.

[33] SOTOMAYOR, B.; MONTERO, R. S.; LLORENTE, I. M.; FOSTER, I. **Virtual Infrastructure Management in Private and Hybrid Clouds**. *Internet Computing, IEEE*, 13(5):14–22, Sept. 2009.

[34] SUN, H.; T HANGARAJAH, J. P. L. **Eclipse-based prometheus design tool**. In: AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, p. 1769-1770, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems., 2010.

[35] WANG, L.; XU, J.; ZHAO, M.; TU, Y.; FORTES, J. **Fuzzy modeling based resource management for virtualized database systems**. In: *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, p. 32 –42, july 2011.

[36] Weiss, G., editor. **Multiagent systems: a modern approach to distributed artificial intelligence**. MIT Press, Cambridge, MA, USA, 1999.

[37] WINIKOFF, M.; PADGHAM, L. **The prometheus methodology**. The prometheus methodology. In: Weiss, G.; Bergenti, F.; Gleizes, M.-P.; Zambonelli, F., editors, Methodologies and Software Engineering for Agent Systems, volume 11 de Multiagent Systems, Artificial Societies, And Simulated Organizations, p. 217-234. Springer US, 2004. 10.1007/1-4020-8058-$1_1$4., 2004.

[38] WINIKOFF, L. P. M. **Developing Intelligent Agent Systems: A Practical Guide**. John Wiley Sons Ltd, 2004.

[39] WOOLDRIDGE, M. **Agent-based computing**. *Interoperable Communication Networks*, 1:71–97, 1997.

[40] YANG, Q.; HE, G.; LI, L. **Application of genetic algorithm on human resources optimization**. In: *Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference On*, volume 3, p. 160 –163, june 2010.

[41] ZADEH, L. **Is there a need for fuzzy logic?** In: *Fuzzy Information Processing Society, 2008. NAFIPS 2008. Annual Meeting of the North American*, p. 1 –3, may 2008.

APÊNDICE A

Ferramentas utilizadas

A.1 JFuzzyLogic

JFuzzyLogic[1] é um utilitário escrito em Java e utilizado amplamente como mecanismo de inferência *Fuzzy* [10], [14], [25], [27]. Implementa a epecificação IEC 61131 parte 7 [19] que define a *Fuzzy Control Language* (FCL)[2].

As características do JFuzzyLogic são:

- Possui integração com a ferramenta eclipse por meio de *plugin*.
- Oferece suporte a diversos tipos de funções de pertinência, tais como: Triangular, Trapezoidal, Gauss, Sino Generalizada, Sigmoide e *Singleton*.
- Implementa a FCL.
- Oferece suporte a diversos tipos de defuzzificadores, tais como: Centro de Gravidade, Máximo mais à Direita, Centro de Área, Máximo mais à Esquerda e Máximo Médio.
- Agregação de regras, ou seja, como as regras são agregadas ou acumuladas.
- Conectores *AND* e *OR*.

A.2 Plataforma Jade

A plataforma JADE[3] (*Java Agent DEvelopment*) constitui uma ferramenta de desenvolvimento de agentes inteiramente desenvolvida em Java. Funciona em ambiente distribuído e facilita a implementação de Sistemas Multiagentes através de um *Middleware* que segue as especificações da FIPA (*Foundation for Intelligent Physical Agents*)[4]. Esta organização define uma série de protocolos e padrões necessários para interação entre agentes. Também promove tecnologias baseadas em agentes e

[1] http://jfuzzylogic.sourceforge.net/
[2] O apêndice B apresenta esta linguagem em detalhes.
[3] http://jade.tilab.com/
[4] http://www.fipa.org/

interoperabilidade destes padrões com outras tecnologias. Atualmente, as especificações FIPA representam a maior atividade de padronização conduzida na área de tecnologias de agentes [5].

O fato da plataforma ser desenvolvida em Java, implica que esta pode ser executada em qualquer sistema operacional, desde que o mesmo possua uma Máquina Virtual Java. Por ser distribuída permite o seu funcionamento entre várias máquinas, mesmo que possuam diferentes arquiteturas.

A.3 *Prometheus Design Tool* (PDT)

A ferramenta de projeto Prometheus Design Tool[5] [34] foi desenvolvida para auxiliar na elaboração e no desenvolvimento de Sistemas Multiagentes, utilizando a metodologia Prometheus [37]. As três fases de um projeto na metodologia Prometheus e que são incluídas no PDT são:

- Especificação do sistema: descrição dos objetivos, cenários e funcionalidades; a descrição das interfaces do sistema para o ambiente é feita em termos de ações, percepções e dados externos.
- Projeto arquitetural: identificação dos tipos de agentes; a estrutura do sistema é capturada em diagramas de alto nível; cenários são desenvolvidos em protocolos de interação.
- Projeto detalhado: os detalhes internos de cada agente são especificados e definidos em termos de habilidades, dados, eventos e planos; diagramas de processos são utilizados para sair dos protocolos de interação e chegar nos planos.

A ferramenta PDT é capaz de gerar diagramas automaticamente, a partir do projeto dos agentes. Estes diagramas são úteis no entendimento do SMA e auxiliam na documentação do mesmo.

[5] http://www.cs.rmit.edu.au/agents/pdt/pdt.shtml

APÊNDICE B

Fuzzy Control Language

Fuzzy Control Language é uma linguagem que implementa controladores *Fuzzy* e é padronizada pela IEC 61131-7 [19].

Nesta linguagem é possível especificar conjuntos *Fuzzy*, funções de pertinência e regras *IF-THEN*, por exemplo:

RULE 0: IF (Temperature IS Cold) THEN (Output IS High).

B.1 Elementos da *Fuzzy Control Language*

A especificação completa da FCL pode ser obtida em [19]. Esta seção limita-se à apresentação dos principais elementos desta linguagem, necessários ao entendimento do restante deste trabalho. Estes elementos são:

- Bloco de função;
- Bloco para Fuzificação e Defuzificação;
- Bloco de regras.

B.2 Bloco de Função

Neste bloco, é possível descrever todo o FRBS. É delimitado pelas expressões *FUNCTION_BLOCK* e *END_FUNCTION_BLOCK*, conforme pode ser visto nas linhas 2 e 65 da Listagem B.1. A Figura B.1 apresenta um exemplo da utilização da expressão *FUNCTION_BLOCK*.

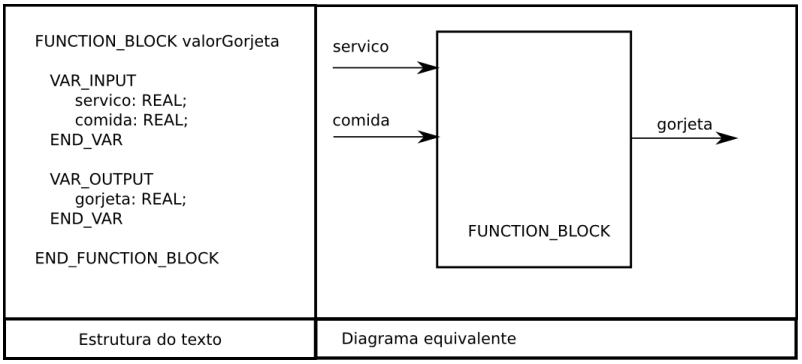


Figura B.1: *Exemplo da declaração FUNCTION_BLOCK e diagrama equivalente para representá-la. Retirado de [19].*

B.3 Blocos para Fuzificação e Defuzificação

Os valores de entrada devem ser convertidos em valores *Fuzzy*. Esta conversão se chama Fuzificação e é feita utilizando funções de pertinência definidas para cada variável de entrada. São descritas na FCL pelas expressões *FUZZIFY* e *END_FUZZIFY*. Utiliza-se a expressão *TERM* para descrever os conjuntos *Fuzzy* de cada variável de entrada. A Figura B.2 apresenta os conjuntos *Fuzzy* da variável "comida" da Listagem B.1 com sua respectiva codificação.

Listagem B.1: *Arquivo FCL com as regras*

```
1  // Definição de um bloco (pode haver mais de um bloco por arquivo)
2  FUNCTION_BLOCK prioridadeAtendimento
3
4  // Variáveis de entrada
5  VAR_INPUT
6      comida : REAL;
7      atendimento : REAL;
8  END_VAR
9
10 // Variáveis de saída
11 VAR_OUTPUT
12     valor : REAL;
13 END_VAR
14
15 FUZZIFY comida
16     TERM ruim := trian 0 0 5;
17     TERM boa := trian 2.5 5 7.5;
18     TERM excelente := trian 5 10 10;
19 END_FUZZIFY
20
21 FUZZIFY atendimento
```

```
22      TERM ruim := trian 0 0 5;
23      TERM bom := trian 2.5 5 7.5;
24      TERM excelente := trian 5 10 10;
25  END_FUZZIFY
26
27
28  // Defuzzificação da variável de saída
29  DEFUZZIFY valor
30      TERM barato := trian -6 0 6;
31      TERM normal := trian 3 5 8;
32      TERM caro := trian 4 10 16;
33      // Uso do método de defuzzificação 'Centro de Gravidade'
34      //METHOD : COG;
35
36      // Uso do método de defuzzificação 'Maximo mais a direita'
37      //METHOD : RM;
38
39      // Valor padrão é 0 (se não existir regra ativa na defuzzificação)
40      DEFAULT := -1;
41  END_DEFUZZIFY
42
43  RULEBLOCK No1
44      // Use 'min' para 'and' (e implicitamente use 'max'
45      // para 'or' ao cumprir o teorema de  DeMorgan)
46      AND : MIN;
47      // Use o metodo de ativacao 'min'
48      ACT : MIN;
49      // Use o metodo de ativacao 'max'
50      ACCU : MAX;
51
52      RULE 11  : IF comida is ruim THEN valor IS barato;
53      RULE 12  : IF comida is boa THEN valor IS normal;
54      RULE 13  : IF comida is excelente THEN valor IS caro;
55
56      RULE 21  : IF atendimento is ruim THEN valor IS barato;
57      RULE 22  : IF atendimento is bom THEN valor IS normal;
58      RULE 23  : IF atendimento is excelente THEN valor IS caro;
59
60      RULE 31  : IF comida is boa and atendimento is bom THEN valor IS caro;
61      RULE 32  : IF comida is boa and atendimento is ruim THEN valor IS normal;
62
63  END_RULEBLOCK
64
65  END_FUNCTION_BLOCK
```

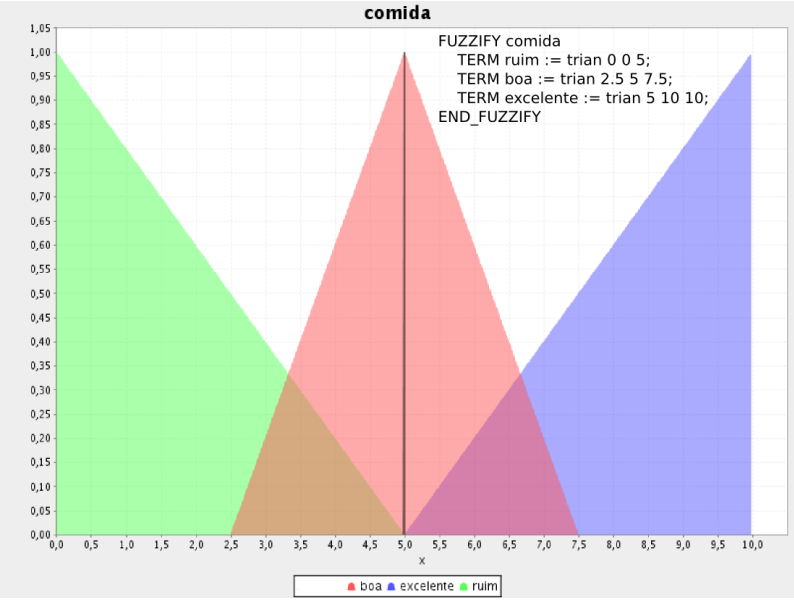


Figura B.2: *Representação da fuzificação da variavel de entrada "comida".*

A etapa final, ilustrada na Figura 2.3, é a etapa da defuzificação. Nesta etapa, os valores de saída devem ser convertidos para valores numéricos. A representação deste processo na FCL é análoga à representação do processo de fuzificação, diferindo apenas nos termos *DEFUZZIFY* e *END_DEFUZZIFY*. Também é necessário informar o método de defuzificação, que pode ser: Centro de Gravidade, Centro de Gravidade para *Singletons*, Centro de Área, Máximo mais a esquerda e Máximo mais a direita [19].

B.4 Bloco de regras

O bloco de regras deve ser delimitado pelas expressões *RULEBLOCK* e *END_RULEBLOCK*. É neste bloco que ficam as regras do tipo *if-then*. São exemplos de regras:

```
RULE 31  : IF comida is boa and atendimento is bom
                                    THEN valor IS caro;

RULE 32  : IF comida is boa and atendimento is ruim
                                    THEN valor IS normal;
```

APÊNDICE C

Esquemas do Banco de Dados utilizado

Aqui é apresentado o esquema do banco de dados utilizado pelo COGNARE. O objetivo é o de auxiliar no entendimento do funcionamento do projeto.

Ao todo o COGNARE utiliza três tabelas:

- **instance** (Listagem C.1): contém as instâncias utilizadas no aprendizado;
- **attribute** (Listagem C.1): contém os atributos das instâncias utilizadas no aprendizado;
- **decision** Listagem C.2: contém as decisões tomadas pelo cognare, para posterior utilização.

Listagem C.1: *Esquema das tabelas "instance" e "attribute".*

```
1  CREATE TABLE instance
2  (
3    id bigserial NOT NULL,
4    CONSTRAINT instance_pk PRIMARY KEY (id)
5  );
6
7  CREATE TABLE attribute
8  (
9    id bigserial NOT NULL,
10   instance_id bigint,
11   attribute_name character varying,
12   attribute_value character varying,
13   attribute_type character varying,
14   CONSTRAINT attribute_pk PRIMARY KEY (id)
15   CONSTRAINT instance_attribute_fk FOREIGN KEY (instance_id)
16       REFERENCES instance (id) MATCH SIMPLE
17       ON UPDATE NO ACTION ON DELETE NO ACTION
18 );
```

Listagem C.2: *Esquema da tabela "decision".*

```
1  CREATE TABLE decision
```

```
2 (
3   approach character varying NOT NULL,
4   fcl oid,
5   CONSTRAINT decision_pk PRIMARY KEY (approach)
6 );
```