RT-MLR: UM FRAMEWORK PARA DESENVOLVIMENTO DE SISTEMAS SENSÍVEIS AO CONTEXTO E DE TEMPO REAL

Pablo Rangel

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Jano Moreira de Souza

Rio de Janeiro
Dezembro de 2010

RT-MLR: UM FRAMEWORK PARA DESENVOLVIMENTO DE SISTEMAS SENSÍVEIS AO CONTEXTO E DE TEMPO REAL

Pablo Rangel

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Jano Moreira de Souza, Ph.D.

_____
Prof. Milton Ramos Ramirez, D.Sc.

_____
Prof. Geraldo Bonorino Xexéo, D.Sc.

_____
Prof. Leonardo Gresta Paulino Murta, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2010

Rangel, Pablo

RT-MLR: Um framework para desenvolvimento de sistemas sensíveis ao contexto e de tempo real / Pablo Rangel. – Rio de Janeiro: UFRJ/COPPE, 2010.

XIV, 110 p.: il.; 29,7 cm.

Orientador: Jano Moreira de Souza

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referencias Bibliográficas: p. 93-97.

1. Sistemas sensíveis ao contexto. 2. Múltiplas lógicas. 3. Tempo real. I. Souza, Jano Moreira de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Titulo.

A história será gentil para mim, pois pretendo escrevê-la.

Winston Churchill

Este trabalho é dedicado aos meus pais Paulo e Vera, a minha irmã Paula e a minha esposa Raquel, pelo amor, dedicação e sacrifício.

AGRADECIMENTOS

Agradeço a Deus por Sua presença. É Dele minha força. És tu, Senhor, que me sustenta, ama, cuida, prova, corrige e perdoa. Faz de mim prova de Sua existência. Peleja comigo, sara minhas feridas e me enaltece nas batalhas.

Ao meu pai, Paulo Rangel, pelo seu exemplo de amor, honestidade, dedicação e trabalho. Sua conduta à frente da família me ensinou a valorizar o esforço, a lealdade e o comprometimento com seus ideais.

A minha mãe, Vera Lúcia, pela sua dedicação, altruísmo e valentia. Sua coragem e determinação são as bases da minha educação e existência.

Agradeço a minha irmã Paula, por seu amor, incentivo e ajudas incontáveis.

A minha amada esposa Raquel por seu amor incondicional, sua ajuda constante, presença motivadora, companheirismo e sua interminável paciência.

Meus sinceros agradecimentos ao meu amigo Jeremy Johnson e a minha sogra Jorgenete, pela amizade, orações e torcida.

Meus agradecimentos a Drª. Tânia Vergara e ao Dr. Mauro da Costa pelo amor e dedicação na execução de seus trabalhos ao longo da minha vida.

Aos meus amigos de graduação pela torcida e apoio.

A todos os mestres da minha vida. Em especial ao Prof. Fabiano Ramos, Profª. Vera Prudência, Prof. Adriano Caminha e a Profª. Claudia.

Ao Prof. José Gomes de Carvalho Júnior, por sua importante presença em minha vida. Agradeço a oportunidade de poder não só testemunhar a sua notória capacidade profissional e acadêmica, mas também aprender com seus ricos valores culturais e humanos. Além da tutoria diária, sua contribuição se fez presente neste trabalho com elementos que serviram de base para o desenvolvimento de sua tese de doutorado.

Ao Capitão-de-Mar-e-Guerra João Luís Marins pelo vasto incentivo, compreensão e oportunidades concedidas a mim durante minha carreira profissional.

Meus sinceros agradecimentos ao Prof. Jano Moreira de Souza e ao Prof. Milton Ramos Ramirez, pela orientação, apoio e oportunidade de ingresso na carreira acadêmica. Obrigado pelo empenho na orientação da minha dissertação e pelo privilégio de poder conviver e estar exposto aos seus conhecimentos e experiências.

À Marinha do Brasil por permitir o meu aperfeiçoamento profissional.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RT-MLR: UM FRAMEWORK PARA DESENVOLVIMENTO DE SISTEMAS SENSÍVEIS AO CONTEXTO E DE TEMPO REAL

Pablo Rangel

Dezembro/2010

Orientador: Jano Moreira de Souza

Programa: Engenharia de Sistemas e Computação.

Este trabalho apresenta a implementação de um *framework* de *software* chamado Real-Time Multiple Logic Reasoner (RT-MLR), cuja finalidade é servir de ferramenta para o desenvolvimento de sistemas sensíveis ao contexto e de tempo real. A implementação do *framework* concilia a abordagem Orientada a Objetos com a abordagem de sistemas baseado em regras, fornecendo um motor de inferência capaz de investigar o contexto em tempo real. Na solução adotada neste trabalho, as regras são escritas com os objetos do projeto de *software* e expressas por meio de três tipos de lógicas: lógica de primeira ordem, lógica *fuzzy* e lógica temporal. Um ambiente de teste foi criado visando demonstrar a aplicabilidade do *framework* no desenvolvimento de um sistema tático sensível ao contexto.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

RT-MLR: A FRAMEWORK TO DEVELOP CONTEXT-AWARE AND REAL-TIME SYSTEMS

Pablo Rangel

December/2010

Advisor: Jano Moreira de Souza

Department: System Engineering

This work presents the implementation of a software framework called Real-Time Multiple Logic Reasoner (RT-MLR), which aims to support the development of context-aware and real-time systems. The framework implementation reconciles the Object-Oriented approach with the rule-based systems approach, providing an inference engine to investigate the context in real-time. In the solution adopted, the rules can be written with the objects of software project and expressed in three logic types: first order logic, fuzzy logic and temporal logic. A test environment has been created aiming to show the framework applicability on the development of a C4ISR context–aware system.

ÍNDICE

ÍNDICE DE FIGURAS

INDÍCE DE TABELAS

INDÍCE DE EXEMPLOS

1. INTRODUÇÃO

1.1 Preâmbulo

A evolução tecnológica ao longo da história aponta que os sistemas computacionais necessitam fazer muito mais do que o tradicional processo de conversão "dado-informação". O desenvolvimento de um sistema não se restringe à criação e ao uso de códigos e dispositivos capazes de produzir respostas de acordo com situações pré-estabelecidas. Um sistema não é um produto imutável cujo uso, aprimoramento e respostas são sempre bem definidos. É necessário entender a sua aplicabilidade, o seu público alvo, aderência aos requisitos e expectativas ao longo de seu ciclo de vida.

Segundo Grudin [1], durante a década de 60, os sistemas desenvolvidos tinham por objetivo apenas servir de interface para entrada de dados e resposta do processamento do hardware existente. Posteriormente, durante a década de 70, os sistemas desenvolvidos carregavam consigo a finalidade de suportar a integração entre aplicativos de escritório. Em meados da década de 80, o termo cooperação é introduzido, permitindo maior eficiência entre grupos de trabalho. Em 1992, os autores Want *et al*. [2] apresentaram um sistema chamado "Active Badge Location System", que foi considerado a primeira aplicação que produzia respostas de acordo com o contexto de utilização. O sistema tinha como finalidade transferir ligações para o telefone que estivesse mais próximo ao usuário, cuja realização era possível graças aos sensores que indicavam a localização geográfica dos utilizadores.

Como parte desse processo de evolução, muitos sistemas têm sido construídos com o propósito de oferecer melhores serviços aos usuários, considerando não só as regras de negócios, mas também fatores externos ao sistema. A combinação de diversas informações tem permitido que respostas sejam adaptadas de acordo com as necessidades de indivíduos, de equipamentos e até de fatores externos à interação direta entre computador e usuário. Considerar a situação corrente do sistema, isto é, o contexto de utilização, trouxe novas perspectivas, paradigmas e serviços aos usuários e profissionais de tecnologia.

Assim como o "Active Badge Location System", sistemas que possuem a capacidade de adaptar suas respostas em função contexto são chamados de sistemas sensíveis ao contexto.

1.2 Problema

A questão tecnológica tem papel fundamental no desenvolvimento de sistemas sensíveis ao contexto. A melhora contínua nos processos de desenvolvimento desses sistemas constitui um processo natural de evolução.

Um sistema sensível ao contexto deve ser capaz de obter informação de diversas fontes, tal como um sensor, um satélite, uma câmera ou uma informação introduzida manualmente, utilizando-as de forma combinada e conveniente.

Um dos aspectos a ser considerado no processo de desenvolvimento é referente às questões tecnológicas diretamente envolvidas, como por exemplo, a conexão a um determinado banco de dados, a utilização de hardware adequado, protocolos, *software*s de comunicação e integrações de módulos de terceiros com o sistema principal [3].

Superada esta etapa, outro aspecto importante no processo de desenvolvimento de um sistema sensível ao contexto se refere à gestão da vasta informação obtida com o suporte tecnológico, representando-a em conhecimento e traduzindo-a em efetiva eficiência no uso do sistema. Para os autores Baader *et al.* [4], o desafio de integrar informações heterogêneas do ambiente em uma única visão, cujo nível semântico seja compreensível aos seres humanos, continua em aberto.

Sistemas sensíveis ao contexto são desenvolvidos para atuarem de acordo com o raciocínio efetuado em cima do contexto atual, além das regras de domínio pré-estabelecidas. Em certas ocasiões, a essência de muitos sistemas está ligada a sua capacidade de realizar inferências, em razão dos processos decisórios envolvidos necessitarem ser entendidos e justificados. Raciocinar em cima de um contexto tem um caráter lógico e formal, o que o torna um campo de aplicação para análise de decisão, métodos baseados em regras, sistemas especialistas, lógica de programação, etc. [3]. Nestes casos, o contexto pode ser adequadamente representado por regras expressas em lógicas formais, que permitem uma representação do conhecimento inteligível aos seres humanos.

Infelizmente, adequar o uso de regras em sistemas sensíveis ao contexto traz à tona problemas de integração com projetos de *software*.

Sistemas computacionais são normalmente projetados e desenvolvidos por meio da abordagem de Orientação a Objetos (OO) e de linguagens aderentes a essa metodologia, que permitem melhor entendimento do domínio do problema e melhor codificação. Na abordagem OO, os “objetos” do sistema representam instâncias das

entidades existentes no mundo real, o domínio do sistema, com atributos, operações e relacionamentos baseados na natureza do sistema e das entidades em específico.

Assim sendo, a própria arquitetura de um sistema OO representa a modelagem e a definição do contexto de utilização, ainda que limitada pelos seus recursos. Por outro lado, a definição de uma regra de domínio pode não estar suficientemente explícita por meio de seus objetos e relações. A adoção de métodos baseados em regras pode resolver essa deficiência, mas necessita de um mecanismo de ligação entre a modelagem do contexto já descrita com a OO e as regras de domínio. Além disso, em certas ocasiões as mesmas entidades que estão dispostas nas estruturas OO podem vir a ser parte integrante de proposições de regras de domínio. Este inerente relacionamento entre os objetos dos sistemas OO e as regras de domínio pode potencialmente melhorar a coesão do projeto.

Outra questão pertinente é a de que sistemas sensíveis ao contexto possuem, usualmente, requisitos de tempo real. Nestes casos, o uso do conhecimento do domínio deve ser feito de forma precisa e imediata, o que significa, para estes sistemas, que raciocinar em cima do contexto tem grau crítico de importância, pois o próprio contexto está em constante mudança. Sistemas sensíveis ao contexto baseados em regras necessitam investigar as regras de domínio sempre que o contexto se alterar, o que resulta em um esforço computacional custoso para monitoração, principalmente no que tange à comparação a uma abordagem não orientada a regras.

Algumas soluções já existentes na literatura e no mercado de *software* utilizam regras com objetos OO. Contudo, estas soluções são usualmente oferecidas através de uma sintaxe proprietária, não aproveitando diretamente os objetos do sistema como parte de suas proposições. Estas mesmas soluções não contemplam lógicas de diferentes naturezas para expressar regras e conceitos heterogêneos – a expressão do conhecimento é, em geral, limitada a predicados de primeira ordem. Mesmo quando se beneficiam da própria linguagem de programação e conseqüente uso dos objetos OO, os mecanismos para monitoração do contexto são totalmente ignorados ou tratados com pouca importância.

1.3 Motivação

O Instituto de Pesquisas da Marinha (IPqM) tem desenvolvido inúmeros projetos ao longo dos últimos anos, em diferentes áreas vinculadas à operação de equipamentos

de defesa. São exemplos desses sistemas os simuladores de operações táticas utilizados em treinamento de pessoal operativo, os sistemas táticos operativos utilizados em navios e helicópteros da Marinha do Brasil (MB) e da Marinha do Uruguai, e os sistemas de controle e monitoração de propulsão e avarias utilizados em embarcações e em instalações terrestres da MB. Os sistemas desenvolvidos pelo IPqM encontram-se atualmente instalados e operativos em 26 diferentes plataformas (navios e helicópteros), em 3 centros de ensino e em um hospital da MB.

Atualmente em curso no IPqM, o projeto do novo Terminal Tático Inteligente (TTI) visa criar um sistema capaz de apoiar as operações táticas navais que, entre outras tarefas, analisa e classifica entidades próximas ao navio, apóia as operações de gerência e combate, além de colaborar com outros navios da frota aliada em missões.

Em sistemas militares como o TTI, qualquer decisão de comando deve estar em consonância com doutrinas específicas para diferentes situações de combate. Tais decisões baseiam-se em dados advindos de diversos equipamentos, como sensores, radares, informações de gps, etc. Nestes sistemas, as informações heterogêneas, o conhecimento do militar e a colaboração provenientes de outros navios são fatores de extrema importância na tomada de decisões. Este contexto situacional é prioritariamente fundido em diversos níveis de granularidade, para diversos tipos de fins, desde o refinamento e precisão do processamento de sinais até decisões de comando de alto nível.

A capacidade de interpretação do operador do sistema também é relevante para este domínio de aplicação. As interpretações promovem a correlação e classificação de entidades detectadas, bem como a sua avaliação comportamental, que podem indicar ações ou contramedidas necessárias como parte de suas ações de combate e defesa. Assim, a transformação do conhecimento tácito do operador para o explícito torna-se não só desejável, mas sim imprescindível, à medida que as decisões se dão em razão de um raciocínio aproximado do contexto naval efetuado pelo operador. Além disso, a definição das regras que regem um contexto militar devem ser adequadamente expressas para futura manutenção, pesquisa e auditoria.

Nesse cenário, surgiu a necessidade de implementação de um *framework* de *software* na linguagem Orientada a Objetos Java (linguagem adotada no projeto) capaz de apoiar o desenvolvimento do TTI, permitindo que os especialistas possam expressar as regras do domínio de uma maneira mais simples e integrada com o projeto de *software*.

1.4 Objetivo

Em consonância ao problema e a motivação apresentada, este trabalho apresenta o estudo, a implementação e a aplicação de um *framework* de *software* chamado Real Time Multiple Logic Reasoner (RT-MLR).

O objetivo do *framework* desenvolvido é apoiar o desenvolvimento de sistemas sensíveis ao contexto Orientado a Objetos que adotem a abordagem baseada em regras, aproveitando-se da inerente interseção entre o conhecimento explicitado por meio da arquitetura OO e as regras de domínio. O desenvolvimento do RT-MLR procura oferecer um melhor modo para expressão do conhecimento, bem como uma maneira de monitorar o contexto de forma computacionalmente adequada.

Do ponto de vista conceitual, este trabalho parte do pressuposto de que a expressão de regras pelos próprios objetos permite a criação de sistemas sensíveis ao contexto cuja investigação se dará de uma forma mais precisa e acurada, em razão da capacidade de expressão mais próxima da linguagem natural e da combinação de três tipos de lógicas: lógica de primeira ordem, lógica *fuzzy* e lógica temporal.

Do ponto de vista computacional, este trabalho parte do pressuposto de que uma investigação das regras por meio de uma abordagem orientada a eventos permite a obtenção de um custo computacional adequado para uma monitoração do contexto em tempo real. O RT-MLR procura tirar proveito da associação das regras aos objetos, investigando as regras que estão diretamente associadas aos objetos que sofreram mudança de contexto.

O RT-MLR é exemplificado frente a sua capacidade de apoiar a investigação e raciocínio sobre um contexto de um sistema militar naval. Sistemas militares são bons exemplos de sistemas sensíveis ao contexto e de tempo real, pois grande parte do conhecimento do domínio está embutida no próprio sistema, permitindo aos usuários tomar decisões, estimar eventos futuros e obter respostas precisas.

Neste trabalho, todos os exemplos são apresentados conforme conhecimentos de um especialista naval. A proficiência do *framework* é apresentada em relação à capacidade de expressão de regras de correlação e classificação de alvos simulados da Marinha Brasileira (MB). De uma forma não imperativa, os resultados práticos dos testes serão utilizados como evidências da capacidade do RT-MLR de apoiar o desenvolvimento do TTI.

1.5 Organização

Este trabalho possui seis capítulos além da introdução.

O capítulo dois apresenta as definições formais de contexto e de sistemas sensíveis ao contexto encontrados na literatura. Neste capítulo são discutidos os principais conceitos, além de exemplos de sistemas que se enquadrem nesta categoria de *software*.

O capítulo três discute questões a respeito do desenvolvimento de sistemas sensíveis ao contexto e o uso do conhecimento em tais sistemas. Este capítulo discute os principais problemas na adoção de sistemas baseados em regras integrados com projetos de *software* orientados a objetos, quer seja no âmbito conceitual, quer seja no âmbito computacional.

O capítulo quatro apresenta a implementação completa do *framework* sob o ponto de vista conceitual e computacional, suas implicações, os detalhes da implementação, além de apresentar os operadores lógicos e seus exemplos.

O capítulo cinco apresenta alguns produtos similares ao *framework* RT-MLR. Esta amostra de produtos é avaliada conforme critérios estabelecidos em relação à capacidade de expressão do conhecimento e à capacidade de monitoração do contexto.

O capítulo seis apresenta os resultados obtidos com o RT-MLR referentes à correlação e à classificação de alvos do contexto militar naval.

O capítulo sete apresenta as conclusões e os possíveis trabalhos futuros.

2. DEFINIÇÕES DE CONTEXTO E SISTEMAS SENSÍVEIS AO CONTEXTO

2.1 Contexto em sistemas computacionais

Existem diversas definições de contexto na literatura de computação, descritas por sinônimos ou exemplos, que dependem do tipo de área que está sendo pesquisada. Em alguns casos, contexto está estritamente relacionado a recursos de computação e comunicação, enquanto outros trabalhos são referentes a conceitos de alto nível de abstração. Para o autor Thanos Demiris, ambos os significados podem ser considerados [5].

Denotativamente, o significado de contexto está descrito como "... condições inter-relacionadas que fazem algo existir ou ocorrer..." [6]. Intuitivamente, no âmbito de sistemas computacionais, o termo contexto remete à idéia na qual um sistema age em função de uma variável (aleatória ou não) ou interpreta significados de acordo com um objetivo pré-determinado.

Na literatura de computação existem diversas definições para o termo "contexto" [1]. Em geral, o termo "contexto" é utilizado de forma agregada ao termo "sensibilidade", resultando em um termo composto chamado "sensibilidade ao contexto" [2]. Em sistemas computacionais, o termo sensibilidade ao contexto é por vezes mencionado como "consciência situacional" [3], que como a própria expressão sugere, refere-se à capacidade de um sistema possuir grande parte dos conhecimentos, procedimentos e respostas aos fatores influenciáveis nas tomadas de decisão. Um sistema é considerado sensível ao contexto se puder extrair, interpretar e usar a informação contextualizada, adaptando suas funcionalidades ao contexto de uso corrente [7].

Para Schilit *et al.* [8], sistemas sensíveis ao contexto são aqueles que se adaptam conforme a sua localização. Para os autores, esta categoria de sistema se caracteriza por se adaptar conforme a proximidade (proximidade do dispositivo no qual o sistema está rodando) com outros atores (dispositivos móveis, servidores, etc.). O autor ainda define tais sistemas como "sistemas capazes de examinar o ambiente computacional e reagir às mudanças deste ambiente".

[1] Tradução da palavra "context" do idioma inglês.
[2] Tradução livre da expressão "context awareness" do idioma inglês.
[3] Tradução livre da expressão "situational awareness" do idioma inglês.

De acordo com os autores Abowd *et al*. [9], contexto é qualquer informação que pode ser usada para caracterizar uma situação de uma entidade. Para os autores, uma entidade é uma pessoa, um objeto ou lugar a ser considerado, sendo relevante para interação entre os usuários e o sistema, bem como para eles mesmos. Ainda de acordo com estes autores, um sistema sensível ao contexto é aquele para qual o contexto provê informações ou serviços relevantes para o usuário, sendo a relevância determinada pela tarefa a ser executada.

Para Chen e Kotz [10], contexto é um conjunto de estados e parâmetros que determinam o comportamento ou o acontecimento de um evento no sistema que é interessante para o usuário. Indo ao encontro destes pesquisadores, os autores Hull *et al.* [11] explicam que contexto nada mais é que os aspectos da atual situação em que o sistema se encontra.

Para encontrar trabalhos que façam um levantamento sobre sistemas sensíveis ao contexto, consulte o trabalho dos autores Demiris [5], Chen e Kotz [10] e Badaulf *et al.* [12].

Para consultar uma revisão e classificação recente da literatura correlata, consulte o trabalho dos autores Hong *et al.* [3].

Para consultar exemplos de sistemas que são referidos como sensíveis ao contexto, consulte os trabalhos dos autores Strang e Linnhoff-Popien [13], Gkekas [14], Fujinami *et al.* [15], Scherp e Boll [16], Brodt e Nicklas [17] e Dey e Abowd [18].

2.2 Tipos de contexto

Do ponto de vista funcional, existem três tipos gerais de contexto: contexto temporal, contexto espacial e o contexto do usuário.

2.2.1 Contexto Temporal

O contexto temporal está relacionado ao momento de execução de um sistema. Isto significa que sistemas sensíveis ao contexto temporal mudam seu comportamento em função de variáveis que estão em constante mudança, bem como mudam suas respostas baseadas na hora de ocorrência de um determinado evento. Por exemplo, um sistema pode mudar suas respostas em função da hora atual; ou então, este mesmo sistema pode aplicar um peso diferente aos seus parâmetros internos em função da estação do ano.

2.2.2 Contexto Espacial

O contexto espacial se refere às informações pertinentes ao lugar físico no qual o sistema - por meio de seu dispositivo - se encontra em execução. Sendo assim, os sistemas que são sensíveis ao contexto espacial alteram seu comportamento em função da sua localização geográfica. Por exemplo, tais sistemas podem diferenciar suas respostas por estarem sendo executadas no escritório e não em casa; ou ainda, mudar a interface homem-máquina em razão de uma transição de cidade.

2.2.3 Contexto do Usuário

Por fim, os sistemas sensíveis ao contexto do usuário mudam seu estado conforme uma parametrização, assumindo papéis diferentes em função do ambiente externo. Conforme explica Pichler *et al.* [19], tal parametrização pode ocorrer de forma automática, por meio de mecanismos de inteligência artificial ou por meio de uma entrada explícita para o sistema. Por exemplo, uma livraria *web* pode alterar a disposição dos ponteiros para os livros à venda em função do perfil do usuário. Tal perfil pode ter sido “aprendido” pelo sistema (baseado em compras anteriores) ou ajustado manualmente pelo usuário, por meio de um formulário para preenchimento de preferências.

3. MODELAGEM E MONITORAÇÃO DE CONTEXTO

O objetivo geral de uma modelagem de contexto é o desenvolvimento de sistemas sensíveis ao contexto cujo formalismo das informações contextuais permita eficientes adaptações em tempo de execução, expressividade adequada de conhecimento, além de futuras manutenções de *software* mais simples.

Conforme os autores Bettini *et al.* [20], o desenvolvimento de sistemas sensíveis ao contexto pode ser viabilizado por meio de métodos de engenharia de *software*. Uma modelagem de contexto adequada permite reduzir a complexidade de tais sistemas, bem como realizar raciocínios adequados sobre o contexto atual. Ainda segundo estes autores, uma modelagem de contexto deve ser capaz de atender seis requisitos: heterogeneidade da informação, relacionamentos e dependências, temporalidade, qualidade da informação, capacidade de inferência, formalismo e obtenção da informação.

Existem várias técnicas para auxiliar o desenvolvimento e a modelagem de contexto. Segundo os autores Strang e Linnhoff-Popien [13], uma maneira de modelar o contexto é ter por base os dados que dele advém. Entre as técnicas analisadas por estes autores, destaca-se a modelagem de contexto baseada em lógica (ou baseada em regras), a modelagem de contexto por Orientação a Objetos e a modelagem com base nas Ontologias.

3.1 Modelagem de contexto por meio de linguagens computacionais

3.1.1 Linguagens Orientadas a Objetos

Desde a década de 70, linguagens de desenvolvimento Orientadas a Objetos (OO) têm sido desenvolvidas e utilizadas em diversos sistemas. Através do paradigma e princípios OO, sistemas computacionais cada vez mais complexos têm sido construídos visando à melhora da produtividade, o aumento da qualidade e a elevação da manutenibilidade. São exemplos de linguagens Orientados a Objetos as linguagens C++, Smalltalk, Delphi, PHP e Java.

Como já mencionado, na abordagem OO, o cerne do desenvolvimento está nos “objetos” do sistema, a representação de instâncias de entidades existentes no mundo

real, o domínio do sistema. Tais objetos são modelados de acordo com seus atributos, operações e relacionamentos, baseados na natureza do sistema e das entidades em específico. Em sistemas OO, o domínio e a representação do contexto e sua complexidade estão intrínsecos ao processo de definição, por meio de classes, objetos e mensagens.

Segundo os autores Coad e Yourdon [21], a abordagem OO segue cinco princípios básicos de administração da complexidade e modelagem de contexto:

- Abstração: É relativo à diminuição da atenção em aspectos menos importantes e ao aumento da percepção do que é realmente relevante em um determinado aspecto do mundo real. Abstração é também definido como um conceito que englobe atributos e serviços referentes a um determinado propósito.
- Encapsulamento: a idéia do encapsulamento está ligada à ocultação dos detalhes de implementação de um determinado aspecto do projeto. Segundo Coad e Yourdon [21], o encapsulamento ajuda a minimizar o trabalho de desenvolvimento de um novo sistema, pois torna-se mais fácil utilizar partes do programa de forma isolada. Estas partes refletem a modelagem de requisitos específicos do projeto, com interfaces para serviços de interesse de outras partes do projeto.
- Herança: o conceito de herança está centrado na idéia de organizar uma taxonomia entre as classes do domínio, evidenciando as semelhanças entre classes em relação a atributos e serviços.
- Associação: as associações presentes na abordagem OO têm como princípio explicitar a relação entre abstrações do mundo real. Pode-se dizer, inclusive, que a hierarquia é um tipo de associação, comumente chamada de generalização ou especialização.
- Comunicação com mensagens: é a maneira pela qual os objetos do sistema OO se comunicam, já que de acordo com os princípios gerais da abordagem OO, os objetos podem realizar ou obter serviços de outros objetos que encapsulam aspectos complexos do sistema.

A principal idéia em modelar sistemas sensíveis ao contexto por meio da abordagem OO está em obter os benefícios do encapsulamento e reutilização [13]. Um dos conceitos atrelados à adoção da abordagem OO está em prover serviços referentes a aspectos do contexto por meio de interfaces, de maneira que possam ser acessados e inter-relacionados por outras partes do sistema. Para os autores Bouzy e Cazenave [22], a modelagem do contexto por meio da abordagem OO se justifica em função da inerente capacidade de reutilização, que permite o desenvolvimento de sistemas através da definição de um pequeno número de propriedades, funções e regras. Os autores ainda ressaltam que tal paradigma permite "simplificar a representação do conhecimento de sistemas de domínios complexos".

O conhecimento acerca do domínio, bem como qualquer ação a ser tomada, está previamente programada por meio de objetos, relacionamentos e instruções de código. Simples sistemas sensíveis ao contexto utilizam as linguagens OO para modelagem e desenvolvimento por meio de seus próprios recursos.

3.1.2 Linguagens Ontológicas

No âmbito computacional, as linguagens ontológicas (ou simplesmente ontologias) são modelos de representação de conhecimento de um domínio específico de problema, suas relações e particularidades. O objetivo principal de uma ontologia é definir um esquema formal para troca de conhecimento entre agentes, de modo que seja explícito, formal, consistente e coerente. São exemplos de linguagens ontológicas as linguagens OIL, DAML, DAML+OIL e OWL.

Segundo os autores Corcho e Gómes-Pérez [23], os principais recursos que uma linguagem ontológica possui para representação do conhecimento são:

- Conceitos ou classes: possuem a mesma idéia de representação das classes na OO. Nas ontologias, as classes permitem a definição de atributos e modificadores, como polimorfismo, facetas, relações, cardinalidades, etc.
- Taxonomias: recurso que descreve a idéia de classificação segundo uma hierarquia, tais quais as relações de especialização e generalização. A definição de taxonomia permite ainda a definição de um mecanismo de herança múltipla ou simples, a critério da necessidade de representação que for adotada.

- Relações: representam o tipo de interação entre os diversos conceitos do domínio.
- Axiomas: modelo de sentença que permite, entre outras coisas, verificar a corretude e a capacidade de imposição de definição.
- Regras de produção: expressão na forma “Se..Então” de um conjunto de condições, que se verdadeiro, resultam em uma ação específica.

As Ontologias são normalmente prioridade de escolha quando se é necessário estabelecer conhecimentos mais aprofundados sobre o domínio e suas restrições, pois possuem mais recursos sintáticos e semânticos, como por exemplo, a transitividade. As Ontologias são particularmente úteis em sistemas de domínio aberto, como os desenvolvidos para a internet e os sistemas que trabalham com mineração de texto. Geralmente possuem, além disso, um motor de inferência que apresenta respostas baseadas na hierarquia de classes e restrições impostas pelas relações. Estas inferências podem começar através de uma consulta explícita à base de dados ou através de processo de investigação automática, disparadas quando ocorrem modificações na base de fatos. Infelizmente, em domínios complexos, a investigação do contexto não está restrita a objetos de sistema em uma hierarquia, sendo altamente recomendado ter mecanismos que façam inferências por meio de regras de domínio [24].

Para obter acesso a uma comparação entre várias linguagens ontológicas existentes, no que diz respeito aos recursos existentes e a sua capacidade de representação do conhecimento, consulte o trabalho dos autores Corcho e Gómes-Pérez [23]. Para obter acesso a uma comparação entre as linguagens orientada a objetos e às linguagens ontológicas, bem como um possível mapeamento entre as abordagens, consulte o trabalho dos autores Baclawski *et al.* [25].

3.2 Modelagem de contexto por meio de regras

Sistemas que usam regras são habitualmente chamados de sistemas baseados em regras, mas também podem ser referenciados como sistemas baseados em conhecimento, uma vez que o núcleo base destes sistemas é o conhecimento de especialistas. O principal conceito atrelado ao uso de regras na representação do conhecimento é o de que é possível realizar o mapeamento completo dos problemas do

domínio por meio de um número adequado de regras e lógicas que representem o raciocínio de um especialista.

Em sistemas que usam regras, uma lógica define as condições para conclusão de uma expressão ou derivação de um fato, a partir de um conjunto de outras expressões ou fatos, cujo processo é conhecido como inferência ou raciocínio[1]. Em uma modelagem de contexto baseado em lógicas, o contexto é definido em termos de fatos, expressões e regras e sua principal característica é o alto grau de formalidade [13].

Existem três componentes básicos em um sistema baseado em regras: a base de conhecimento (ou regras), a base de fatos e o mecanismo de inferência.

I. A base de conhecimento guarda as regras do sistema (banco de regras), que são constituídas por uma condição e uma ação (conclusão).
II. A base de fatos (ou memória de trabalho) é um banco de dados com as conclusões efetuadas pelas regras, além de dados iniciais que dão suporte às primeiras decisões.
III. O mecanismo de inferência é responsável por fazer uma correspondência entre os fatos relacionados ao contexto e as regras que fazem uso destes fatos.

3.2.1 Base de regras e base de fatos

Um sistema tradicional baseado em regras possui um componente chamado “banco de regras” e outro componente chamado “base de fatos”, que como o próprio nome sugere, guarda as regras do sistema e os fatos a respeito do contexto, respectivamente.

A noção de banco ou base não é bem definida no âmbito da literatura de Inteligência Artificial, mas subentende-se por meio da visão prática de implementação, de que se trata de um banco de dados que guarda as regras e fatos, e que possuem um mecanismo para adição, exclusão e leitura de regras em tempo de execução, tal qual um Sistema Gerenciador de Banco de Dados (SGBD) faz para dados do domínio do sistema.

Assim sendo, para compreensão posterior, entenda banco ou base como uma abstração de um componente que possui interfaces para adição, exclusão e consulta de informações, mais precisamente aplicadas às regras e aos fatos.

[1] Tradução da palavra “reasoning” do idioma inglês.

3.2.2 Inferência

Um modelo de inferência de um sistema sensível ao contexto utiliza as bases de fatos e regras no intuito de inferir e adaptar suas funcionalidades. Ao examinar as regras e confrontá-las com os fatos, o modelo de inferência deve adotar algum método de investigação, pois a forma como ele lida com as informações altera o resultado do raciocínio e o resultado da adaptação do sistema. Assim, em sistemas baseados em regras, uma característica importante a se ressaltar é o modo como as inferências são realizadas.

A investigação das regras pode ser realizada por dois métodos: “encadeamento para frente” e o “encadeamento para trás”.

No método de “encadeamento para frente”, ou orientado a dados, as regras são investigadas a partir das condições estabelecidas nas regras. Isto significa que, dada uma condição verdadeira, uma ação será executada como conclusão da regra - esta abordagem parte dos fatos para encontrar soluções. Quando uma regra é verdadeira, então um novo fato é adicionado à base de fatos. São exemplos de motores de inferência que utilizam o “encadeamento para frente”: Jess [26], Clips [27], Drools [28] e Hammurapi Rules [29].

Em geral, sistemas baseados em regras com “encadeamento para frente” utilizam o algoritmo RETE [30] para organização, casamento dos fatos e investigação das regras. O algoritmo RETE examina as regras e cria um grafo acíclico, no qual cada nó representa uma operação unária ou binária. O grafo representa uma árvore contendo as operações comuns às regras, conectadas desde a raiz até as folhas. Enquanto a raiz e os demais nós da árvore representam as operações, as folhas representam as conclusões das regras.

O principal objetivo atrelado à organização hierárquica do RETE é o de eliminação de busca a fatos de forma desnecessária. Com o compartilhamento de operações, têm-se, conseqüentemente, fatos do sistema compartilhados. O RETE determina que uma investigação reutilize os fatos e valores das operações já executadas, a menos que um novo fato tenha sido adicionado à memória de trabalho. Assim, sistemas baseados em regras executam somente as operações cujos fatos tenham sido alterados e que possam, de alguma maneira, alterar o resultado de uma operação relacionada. Com isso, sistemas baseados em regras que utilizam o algoritmo RETE

possuem tempo de resposta eficiente, uma vez que as operações não são investigadas de forma constante, apenas se um fato na base de fatos for alterado.

Para ilustrar melhor o funcionamento do RETE, tome como exemplo duas regras que tratam de um alvo detectado em um radar:

Regra 1: Se o tamanho do alvo é menor que 8 metros e a velocidade do alvo é maior que 300 nós, então o alvo é um míssil.
Regra 2: *Se o tamanho do alvo é menor que 8 metros e a velocidade do alvo é menor que 10 nós, então o alvo é uma embarcação desconhecida.*

Exemplo 3.1: funcionamento do algoritmo RETE

A montagem de grafo do RETE procuraria otimizar a investigação das regras do exemplo 3.1 por meio de um compartilhamento da operação comum às duas regras, que neste caso está relacionado ao tamanho do alvo. Seja, portanto, as seguintes condições e conclusões:

p: Tamanho do alvo < 8 metros;
q: velocidade > 300 nós;
r: velocidade < 10 nós;
X: alvo = míssil;
Y: alvo = embarcação desconhecida;

A figura 3.1 ilustra como o RETE organiza estas duas simples sentenças.

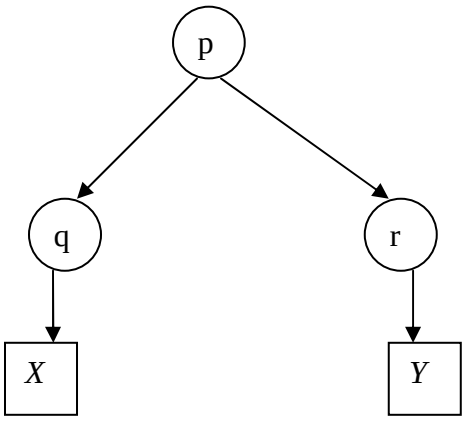

Figura 3.1: organização hierárquica do RETE.

Os sistemas baseados em regras com “encadeamento para frente” necessitam ainda adotar algum tipo de critério para escolha de qual regra investigar frente a um possível conflito existente. Para tornar mais claro esta questão, tome como exemplo as seguintes regras:

Fato: velocidade do alvo igual a 320 nós.

Regra 1: Se a velocidade de um alvo é maior que 300 nós, então o alvo é um míssil.
Regra 2: Se a velocidade de um alvo é maior que 300 nós, então o alvo é um foguete.

Exemplo 3.2: “encadeamento para frente” - metodologias de resolução de conflitos

Ao avaliar o exemplo 3.2, pode-se perceber que as duas regras pretendem classificar uma mesma entidade do domínio, um alvo ainda não identificado. Um motor de inferência com “encadeamento para frente” deve decidir qual regra utilizar, já que ambas as regras podem ser satisfeitas de uma forma não mutuamente excludente. Os métodos de resolução de conflitos mais utilizados podem decidir que regras selecionar de acordo com os seguintes critérios:

- Ordenação por especificidade: uma regra será preterida em relação à outra quando as condições e as lógicas referentes a esta regra estiverem englobadas em outra.
- Ordenação por prioridade: consiste na escolha da primeira (ou da última) regra do banco de regras.
- Ordenação por data: ordenação dos dados por prioridade. A escolha das regras é aplicada aos dados com maior prioridade.
- Ordenação por tamanho da regra: escolha da regra com base no número de condicionais na regra.
- Ordenação por uso recente: ordenar as regras com base no seu uso recente. As regras que foram usadas recentemente têm maior prioridade sobre as outras.
- Agrupamento por afinidade: redução do conflito por meio de agrupamento de regras que tenham afinidade ou objetivo comum. Este método por si só não garante a exclusão do conflito.
- Seleção ao acaso: escolha da regra com base em uma seleção aleatória.

Além destas tradicionais formas de resolução de conflito, outros mecanismos já foram propostos e adotados em diversos trabalhos da literatura. Por exemplo, o autor Marc Dörflinger propôs um motor de inferência para a lógica cortês [31] que utiliza "etiquetas" [1] para determinar a precedência das regras.

A desvantagem na adoção da abordagem orientada a dados está relacionada principalmente à possível adição de novos fatos inúteis ao sistema, além de aumentar o consumo de memória e processamento no casamento entre os fatos e as operações.

Diferentemente da primeira abordagem, o "encadeamento para trás" estabelece uma relação de objetivo com a investigação. Neste caso, de acordo com uma meta traçada, somente aquelas regras que tenham relação com esta meta serão investigadas. Isto é, as conclusões da regras direcionam a investigação.

A investigação com "encadeamento para trás" em um sistema baseado em regras começa pelo estabelecimento de um objetivo inicial. Uma pilha de objetivos é construída e preenchida com base nas investigações das regras e confrontação com novos objetivos. À medida que os objetivos vão sendo resolvidos, essa pilha vai se esvaziando até que o principal objetivo (estabelecido no início) da investigação seja solucionado (exemplo 3.3). O problema intrínseco à abordagem do "encadeamento para trás" está concentrado na possível combinatória entre o número de objetivos que vão surgindo durante a investigação e o número de regras. São exemplos de motores de inferência que utilizam o "encadeamento para trás" o Hammurapi Rules [29] e o Prolog [32].

[1] Tradução da palavra "labels" do idioma inglês

Objetivo: descobrir a classificação de um alvo → alvo (?).

Fato 1: a velocidade do alvo é igual a 320 nós.
Fato 2: a quantidade de vezes que o alvo foi iluminado é igual a 4 vezes.

Regra 1: Se a velocidade de um alvo é maior que 300 nós e o tamanho do alvo é pequeno, então o alvo é um míssil.
Regra 2: Se o número de vezes que o alvo foi iluminado for menor ou igual a 10 vezes, então o tamanho é pequeno.

Resolução:

Pilha de objetivos: classificação do alvo;
Após a investigação da Regra 1, pilha de objetivos atualizada;
Pilha de objetivos: classificação do alvo, tamanho do alvo;
Após a investigação da Regra 2, pilha de objetivos atualizada;
Pilha de objetivos: classificação do alvo
Após a investigação da Regra 1, pilha de objetivos atualizada;
Inferência terminada.

Exemplo 3.3: "encadeamento para trás" – pilha de objetivos

O modo como o processo de inferência é iniciado trata-se de um problema de implementação específico para cada sistema. Neste trabalho, este assunto é tratado no tópico "Monitoração de contexto em tempo real".

3.2.3 Lógicas

Quando se trata da capacidade de expressão do conhecimento, sistemas baseados em regras devem aderir a um padrão de descrição do raciocínio do especialista em função do seu objetivo ou compromisso com a natureza do conhecimento. Russell e Norvig [33] descrevem quatro linguagens formais: a lógica proposicional, a lógica de primeira ordem, a lógica temporal e a lógica *fuzzy*.

3.2.3.1 Regras com lógica proposicional

A lógica proposicional é ideal para representação mais simples de sentenças, cujo resultado só pode assumir dois tipos de estados: verdadeiro ou falso. A expressividade por meio de conectivos lógicos, cujos símbolos, com exceção da

"implicação" (→), possuem muita semelhança com o significado de certas palavras no nosso idioma. Por exemplo, o "e" lógico significa que uma sentença será verdadeira se ambas as proposições forem verdadeiras. O exemplo 3.4 apresenta um exemplo de regra proposicional.

*Regra 1: Se a velocidade de um alvo é maior que 300 nós **E** o tamanho do alvo é pequeno, então o alvo é um míssil..*

Exemplo 3.4: definição de uma regra proposicional

3.2.3.2 Regras com lógica de primeira ordem

A lógica de primeira ordem ou predicados de primeira ordem é uma linguagem de representação mais completa que a lógica proposicional e suficientemente expressiva para o conhecimento comum. A lógica de primeira ordem apresenta-se como um complemento da lógica proposicional, permitindo que relações entre entidades do mundo real possam ser estabelecidas e representadas através de três tipos de quantificadores: "quantificador universal" (∀), "quantificador existencial" (∃) e Igualdade (=). Para os autores Russell e Norvig [33], a lógica de primeira ordem não se apresenta como uma solução viável para resolução de problemas cujos sistemas necessitem de lógicas de ordem superior, incerteza ou outros tipos de lógicas não clássicas. Entretanto, para estes mesmos autores, a expressividade da lógica de primeira ordem é o suficiente para representar solução para diversos tipos de problema. O exemplo 3.5 apresenta um exemplo de regra de primeira ordem.

*Regra: Se **existe** um alvo com velocidade maior que 300 nós, então a prioridade é combater.*

Exemplo 3.5: definição de uma regra de primeira ordem.

3.2.3.3 Regras com lógica temporal

No âmbito de lógicas formais, a lógica temporal é a linguagem de representação de regras e sentenças cujo raciocínio está relacionado ao tempo. A lógica temporal é útil para os comportamentos que só podem ser identificados se considerarmos os eventos

relacionados com o momento em que ocorreram, sendo particularmente importantes para comportamentos que devem ser considerados de natureza evolutiva.

Segundo Chittaro e Montanari [34], a lógica temporal pode ser dividida em dois tipos: “projeção temporal para frente” [1] e “projeção temporal para trás” [2].

Na abordagem de “projeção temporal para frente”, a preocupação está em expressar uma condição de estado no futuro, avaliado de acordo com o contexto atual. Uma regra deste tipo especifica uma condição em um tempo *t* igual ou maior que o presente, e sua ativação será realizada apenas se a condição for satisfeita em termos de fato e momento.

Diferente da abordagem de “projeção para frente”, a “projeção temporal para trás” está relacionada a tempos passados. Ao especificar uma regra nesta abordagem, as condições devem tratar de tempos *t* menores ou iguais ao presente. Tal qual a abordagem de projeção para trás, a ativação da regra só será feita caso a condição lógica seja satisfeita em termos de fato e momento. O exemplo 3.6 apresenta um exemplo de regra temporal.

*Regra: Se a velocidade de um alvo é maior que 300 nós **há pelo menos 2 segundos**, então o alvo é um míssil.*

Exemplo 3.6: definição de uma regra temporal.

Em se tratando do formalismo e da expressividade da lógica temporal, não se pode dizer que exista um consenso ou uma sintaxe amplamente difundida. Muitos trabalhos propõem modelos de expressão que em geral especificam qualitativamente as regras, como por exemplo, o modelo proposto por Allen [35], bem como o modelo apresentado pelos autores Bennett e Galton [36]. Outros trabalhos se preocupam em expressar as condições de forma quantitativa, como por exemplo, o modelo Pearl’s Distance Álgebra [37]. Para consultar uma avaliação destas e outras lógicas temporais, consulte o trabalho de Chittaro e Montanari [34].

3.2.3.4 Regras com lógica fuzzy

[1] Tradução livre do termo técnico “forward temporal projection” do idioma inglês.
[2] Tradução livre do termo técnico “backward temporal projection” do idioma inglês.

A lógica *fuzzy* ou lógica nebulosa, desenvolvida por Lofti Zadeh [38], é a representação geralmente utilizada frente à necessidade de se representar um conhecimento cujo grau de expressividade dado pelo especialista é incerto.

Segundo Almeida e Evsukoff [39], um ser humano que está resolvendo um problema complexo tenta estruturar o conhecimento sobre este problema em conceitos gerais, observando posteriormente as relações essenciais entre esses conceitos. Ainda segundo esses autores, o conceito de representação *fuzzy* está relacionado à perspectiva essencialmente humana de encarar um problema, pois, em determinadas situações, a definição de uma solução não pode ser dada de forma exata, mas pode ser encontrada em termos gerais ou imprecisos. O exemplo 3.7 apresenta um exemplo de regra fuzzy.

*Regra: Se a velocidade de um alvo é **muito alta**, então o alvo é um míssil.*

Exemplo 3.7: definição de uma regra fuzzy.

Uma explicação mais prática é dada pelos autores Guimarães e Lapa [40]. Segundos estes autores, um sistema baseado em regras *fuzzy* consiste em uma coleção de regras "se-então" nebulosas cujas operações obtêm quantificações de um universo do discurso de uma variável de entrada, que por sua vez, baseada nos princípios da lógica *fuzzy*, mapeiam o resultado em um universo do discurso de uma variável de saída. O resultado final de um modelo *fuzzy* retorna um valor escalar em relação a todo processo de raciocínio.

Um sistema de regras *fuzzy* utiliza habitualmente o modelo de Mamdani (ou modelo MIN-MAX) [39] como método de inferência. O modelo de Mamdani (figura 3.1) possui quatro fases, a saber:

1- Primeiramente, os dados escalares de entrada são transformados em graus de verdade utilizando funções *fuzzy* que expressam a pertinência (μ) desses valores escalares aos conceitos lingüísticos modelados (l). A esta fase nomeia-se "*Fuzzyficação"* dos valores de entrada.

2- As regras que expressam o conhecimento são escritas utilizando apenas os conceitos lingüísticos que estejam modelados por alguma função *fuzzy*. Já as inferências expressas nas regras *fuzzy* são realizadas utilizando os graus de pertinência das entradas

aos conceitos lingüísticos. O grau de pertinência da premissa de cada regra é fator limitante para conclusão da regra. A esta fase chama-se “Fase de Inferência”.

3- Como o resultado da Fase de Inferências pode ser a produção de vários conjuntos de saída (um para cada regra acionada), é necessário produzir um único conjunto *fuzzy* de saída, resultante da composição dessas regras. Existem diversos modelos para realizar esta “Fase de Composição das Regras”, sendo um dos mais utilizados a composição pelo valor máximo.

4- Por fim, de posse desse conjunto único, um processo de “*Defuzzyficação”* é iniciado, que consiste em transformar o conjunto *fuzzy* de saída em uma saída escalar. Um dos processos mais utilizados nessa fase é o de Média dos Máximos, onde são utilizados os valores de máximo de cada conjunto que originariamente foi usado para a composição do conjunto de saída. O valor final de saída é calculado como a média desses valores de máximo ponderados pelos respectivos graus pertinência. Isto é, $\frac{\sum(\mu_i * avg_i)}{\sum \mu_l}$.

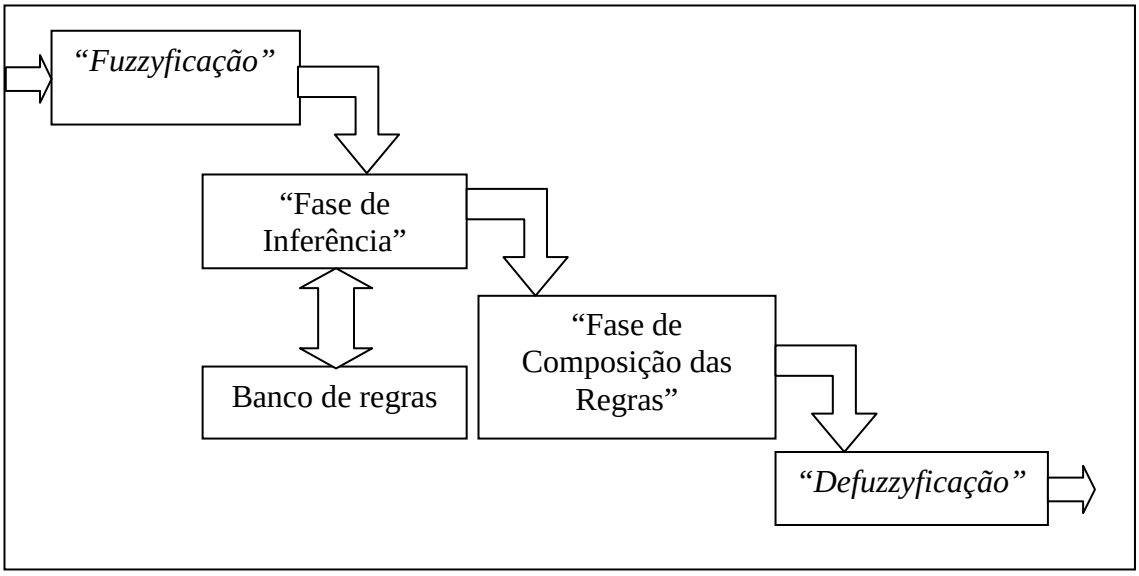

Figura 3.1: Modelo de inferência Mamdani.

Além do modelo de Mamdani, outros modelos de inferência são encontrados, como por exemplo, o modelo de “Takagi-Sugeno-Kang” [39]. Para o processo de *defuzzificação*, existem, ainda, outros métodos que transformam uma variável de saída lingüística em um valor escalar, como por exemplo, o método “centróide” e o “método do primeiro máximo”.

3.3 Considerações sobre uso das abordagens

Regras em sistemas sensíveis ao contexto de diferentes propósitos têm sido usadas desde 1993 por McCarthy [41-42], bem como no início do século 21 com Dey e Abowd [18] e em sistemas recentes com Rangel *et al.* [43-44]. Apesar de alguns autores já desenvolverem sistemas OO e Ontologias com regras explícitas de domínio, existem dois aspectos que ainda necessitam serem discutidos e entendidos nesta integração: a relação entre a representação do conhecimento e a integração tecnológica.

3.3.1 Diferenças na representação do conhecimento

A OO e as Ontologias são poderosas justamente por fornecer mecanismos hierárquicos de representação, além dos relacionamentos entre entidades do domínio. Regras de domínio apresentam habitualmente as mesmas entidades presentes na Ontologia e na OO, porém sem a capacidade de representar as relações senão por lógica de predicados [8]. Os operadores lógicos já presentes em linguagens OO, como por exemplo, o “maior (>)”, “menor (<)” e “igual (=)” permitem definir estas relações, mas não de forma intuitiva. Portanto, fazem-se necessários mais recursos para definições de regras, atribuindo maiores valores semânticos a elas. A lógica de primeira ordem e a lógica *fuzzy*, por exemplo, permitem uma partição mais intuitiva do domínio do sistema, expressando de forma mais concisa o conhecimento de especialistas humanos. As duas lógicas combinadas habilitam o desenvolvimento de um sistema sensível aos contextos espacial, temporal e de usuário, cujo conhecimento não esteja restrito à hierarquia de classes.

Ademais, segundo os autores Baclawski *et al.* [25], uma importante diferença na representação do conhecimento está na noção de monotonicidade. Em sistemas monotônicos, a adição de um novo fato não invalida um fato pré-existente. Tais sistemas assumem a hipótese de “mundo aberto”, cujo desconhecimento de um fato não torna falsa a afirmativa. Por outro lado, em sistemas não monotônicos, a hipótese assumida é a de “mundo fechado”. A hipótese de mundo fechado significa que se um fato não é verdadeiro, então ele é necessariamente falso. O comportamento em sistemas OO é tipicamente de sistemas não monotônicos.

3.3.2 Integração entre regras e lógicas computacionais

O último ponto a se discutir está relacionado à integração tecnológica de um sistema OO a um sistema baseado em regras. Não é o objetivo das linguagens ontológicas permitir o desenvolvimento de sistemas em todos os aspectos tecnológicos, estando restrito à representação do conhecimento. Assim, linguagens ontológicas não serão consideradas neste tópico.

Um sistema baseado em regras tipicamente possui um banco de regras e uma base de fatos. Sempre que uma nova regra deva ou não ser considerada na investigação, esta deve ser inserida ou removida da base de regras. Portanto, por possuírem uma base separada para as regras, tais operações de inclusão ou exclusão podem ser feitas em tempo de execução. Por outro lado, ao explicitar regras diretamente com os objetos do sistema OO, o manuseio de regras fica inviabilizado em tempo de execução, visto que as regras são compiladas juntamente com o código fonte do sistema. Utilizar os objetos do sistema OO para expressar as regras introduz um problema de manutenibilidade, já que todas as vezes que for necessário introduzir mais regras o programa deverá ser re-compilado. Entretanto, este mesmo programa apresentará melhor desempenho, pois não precisará realizar buscas prévias em banco de dados a fim de saber sobre novas atualizações.

A integração de conceitos ocorre de forma menos traumática para a base de fatos. Não há nada que impeça que um sistema que expresse regras por meio de objetos OO manuseie os fatos através de uma base separada. A única restrição pode estar relacionada a sistemas de tempo real, pois a inserção e remoção de fatos em contextos de constante mudança podem acarretar em um desperdício computacional.

3.4 Monitoração de contexto em tempo real baseado em regras

Os autores Maler *et al.* [45] definem “monitoração” como o processo de teste a fim de saber se um determinado comportamento ξ satisfaz a uma determinada condição φ. Esses mesmos autores explicam que o ideal de monitoração é aquele capaz de detectar a violação ou a satisfação de uma condição assim que elas acontecem, pois, para ser útil, o fato deve estar em consonância com o contexto atual, além de poder lidar com dados que se alteram concorrentemente.

Do ponto de vista computacional, é muito importante que sistemas cujo contexto esteja em constante mudança possam monitorar a mudança dos fatos relacionados de

forma automática e contínua. A aderência à monitoração contínua implica que sistemas sensíveis ao contexto se tornam, de fato, sistemas de tempo real. Do ponto de vista conceitual, as regras implementadas devem ser sempre investigadas frente a novos fatos advindos do contexto.

Para ilustrar melhor estes problemas, tome como exemplo a seguinte regra que trata da classificação de um alvo detectado em um radar qualquer que atualiza dados de 1 em 1 segundo:

*Regra: Se a **velocidade** de um alvo é maior que 300 nós, então o alvo é um míssil.*

Exemplo 3.8: monitoração do contexto em tempo real

Ao examinar a regra do exemplo 3.8, pode-se perceber que o fato relacionado a esta regra possui uma natureza de alteração elevadíssima, já que se trata de um alvo cuja velocidade é atualizada a cada 1 segundo. Então, nesse caso, a cada 1 segundo a base de fatos será atualizada e a regra será investigada. Isto poderia resultar em um custo computacional alto, podendo tornar inviável a execução do sistema.

Para sistemas sensíveis ao contexto e de tempo real, torna-se essencial possuir um mecanismo que possa lidar adequadamente com a monitoração contínua. Os autores Murayama *et al.* [46] explicam que existem basicamente dois tipos de modo de monitoração contínua: através do processo de monitoração periódica ou monitoração baseado em eventos.

3.4.1 Processo de monitoração periódica

Na primeira abordagem, a monitoração periódica, as regras de domínio definidas são avaliadas ciclicamente. Ao adotar esta abordagem, o desenvolvedor necessita especificar um intervalo de tempo entre as investigações das regras. O tempo de investigação varia conforme o algoritmo de investigação (o RETE, por exemplo) e a quantidade de regras. Assim sendo, o custo computacional associado em função do tempo para cada iteração é dado por:

$$Ti = O(\delta(N)) + C \text{, onde:}$$

Ti é o tempo de cada iteração, que é dado pela soma do pior caso de um algoritmo de investigação $O(\delta(N))$ para um número total de *N* regras mais o tempo constante *C* pré-determinado pelo desenvolvedor.

A constante *C* não determina a duração da monitoração, mas sim o intervalo entre elas. Além de tornar o processador ocioso durante o tempo *C*, a abordagem por monitoração periódica também resulta em problema sob o ponto de vista conceitual, pois, para garantir que nenhuma relação contextual deixe de ser avaliada, todas as regras deverão ser investigadas. O que ocorre é que nem sempre todos os fatos são novos e tampouco todas as regras têm necessariamente relação com o novo fato. O uso do algoritmo RETE certamente minimizaria esse problema, mas não o livraria completamente de um custo de avaliação.

Um outro problema está relacionado à capacidade de se manter em harmonia com o contexto atual. Se o intervalo *C* for superior à taxa de atualização dos fatos, o sistema não apresentará as informações mais atualizadas existentes, além de apresentar "saltos" de informação, descompassadas em razão da perda.

Suponha, por exemplo, que existam duas regras de domínio, regra 1 e regra 2 com um tempo *C* de intervalo de investigação de 2 segundos. Um determinado fato *A* é utilizado na premissa da regra 2 e o fato *B* é utilizado na premissa da regra 1 (figura 3.2). Mesmo que apenas o fato *B* seja novo, as duas regras 1 e 2 serão investigadas. Além disso, se o fato *B* se altera a cada 1 segundo, então, haverá 50% de perda de informação ao longo do tempo.

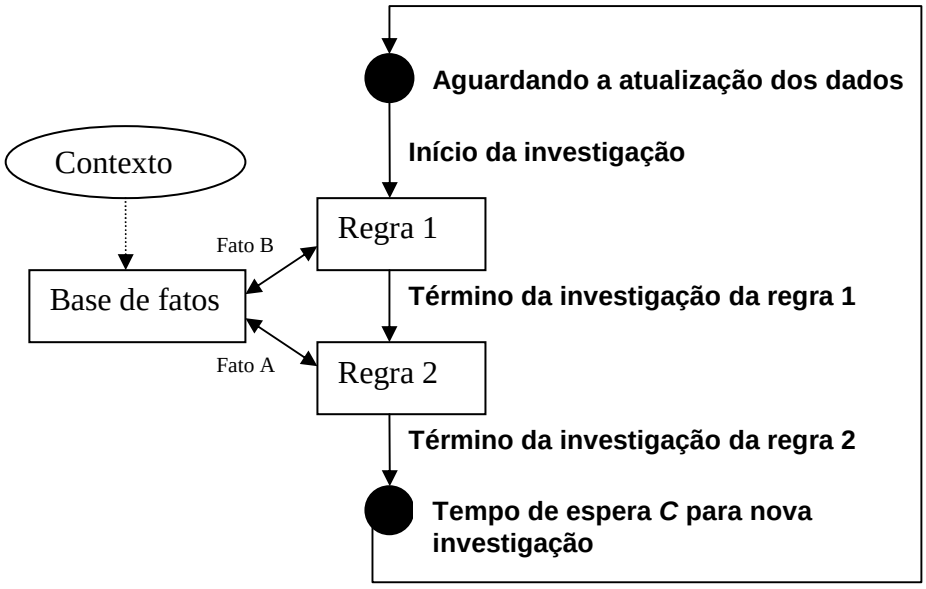

Figura 3.2: Método de monitoração periódica

No que se refere à concorrência de dados, o processo de monitoração periódica não apresenta problemas, pois o processo é iterativo. Cada regra é avaliada uma a uma, a cada ciclo, não importando a relação entre a mudança de contexto e as proposições. Contudo, nesta abordagem se faz necessário manter coerência entre todos os fatos e a investigação das regras, pois não se devem investigar as regras com apenas parte dos fatos atualizados – investigação das regras só pode ser continuada caso a atualização dos fatos esteja terminada.

3.4.2 Processo de monitoração por eventos

A abordagem de monitoração por eventos apresenta melhor desempenho computacional na maioria dos casos, pois o processo consiste em realizar uma monitoração do contexto de acordo com um evento pré-determinado, como por exemplo, a modificação de um registro no banco de dados. A idéia básica gira em torno da investigação das regras associadas imediatamente após a detecção de ocorrência de um evento.

Nesta abordagem, o custo computacional não pode ser determinado por iteração, pois não é determinístico. A vantagem desta abordagem está em associar somente as regras de interesse a determinados eventos, pois, com um número menor de regras para ser investigadas, menor será o custo computacional. Entretanto, determinar os eventos traz maior esforço para desenvolvedores e especialistas do domínio, pois terão de se preocupar com mais fatores relativos à integração, com a associação de códigos a abstrações conceituais do contexto (figura 3.3).

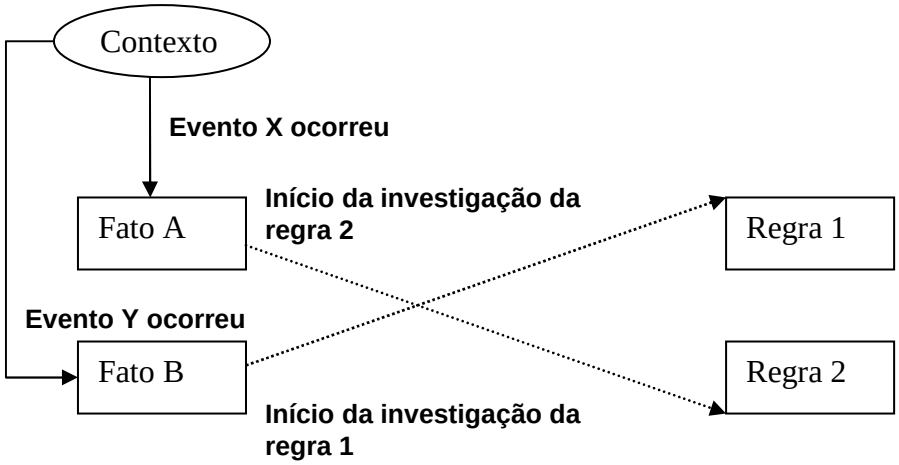

Figura 3.3: Método de monitoração por eventos

A ocorrência constante de um evento em um intervalo temporal muito pequeno poderia também acarretar em problemas de desempenho. Nesse caso, as regras de domínio seriam investigadas a todo o momento, o que poderia gerar um custo computacional até pior do que a abordagem por monitoração periódica.

Para entender a complexidade algorítmica deste caso, suponha *N* o número total de regras e *M* uma parcela das regras *N*, tal que *M* seja necessariamente menor que *N* e represente o número de regras associadas a um evento *E*. Suponha, ainda, o pior caso de um algoritmo de investigação $O(\delta(M))$ para uma parcela de regras *M*. Então, o custo computacional de tempo associado para cada monitoração e investigação é somente dado pelo algoritmo de investigação, isto é:

$$E = O(\delta(M))$$

Em relação à concorrência, a abordagem de monitoração por eventos necessita prover algum mecanismo para exclusão mútua para os fatos, pois estes podem estar sendo utilizados e atualizados de forma concorrente por uma ou mais regras, já que os próprios eventos podem ocorrer concorrentemente.

3.4.3 Monitoração periódica X Monitoração por eventos

No que diz respeito à complexidade algorítmica, pode-se considerar a abordagem orientada a eventos idêntica à abordagem por monitoração periódica, já que nesta última, a indicação geral é de que a constante é desprezível, que o torna, por sua vez, igual à complexidade de pior caso do algoritmo de investigação. Neste caso, o que é determinante é o número de regras aplicadas aos algoritmos de investigação.

Considere, por exemplo, um mesmo algoritmo de investigação para ambas as abordagens de monitoração, com um número total de regras igual 100, isto é, $N = 100$. Considere, ainda, um número de regras *M* relacionadas a um evento igual a 10, isto é, $M=10$. Neste caso, na prática, o custo computacional da abordagem periódica será da ordem N^2 em relação à abordagem orientada a evento. Entretanto, se *M* for igual *N*, a diferença entre as abordagens está no intervalo entre as investigações. Se o tempo *C* definido na abordagem de monitoração periódica for equivalente ao intervalo de tempo de ocorrência do evento *E*, então não haverá diferença na adoção entre uma abordagem e outra.

Torna-se explicito que o diferencial entre as abordagens está no número de regras que é aplicado ao motor de inferência. Assim, uma avaliação de caso médio torna-se fundamental para cada domínio de problema.

No que diz respeito ao mecanismo de concorrência, as abordagens diferem em relação à granularidade: enquanto a abordagem de monitoração periódica necessita prover exclusão mútua para toda a base de regras e para toda base de fatos, a abordagem de monitoração por eventos necessita prover exclusão mútua apenas para os fatos isoladamente.

3.4.4 Processo de monitoração combinada

A combinação de ambas as abordagens é conceitualmente válida. Hipoteticamente, a abordagem combinada é passível de ser aplicada a contextos cujos vários eventos de pouca interseção ocorrem concomitantemente.

A adoção desta abordagem tem como principal vantagem a diminuição do custo computacional da investigação das regras por meio de um tempo pré-determinado (idéia advinda da abordagem de monitoração periódica) e a diminuição do número de regras através de subgrupos (idéia advinda da abordagem orientada a eventos).

A desvantagem da abordagem de monitoração combinada está justamente na pouca generalidade e no alto acoplamento entre as regras de negócio e o projeto de *software* – abordagem requererá sempre ajustes finos por parte dos desenvolvedores e especialistas do domínio.

Entre as três abordagens citadas, esta é a menos encontrada em soluções de *software* devido à necessidade de manutenção contínua.

4. O FRAMEWORK

4.1 Visão geral

O principal objetivo do *framework* Real Time Multiple Logic Reasoner (RT-MLR) é o apoio ao desenvolvimento de sistemas sensíveis ao contexto e de tempo real.

Por meio da combinação da abordagem OO e da abordagem baseada em regras, um sistema computacional pode ser desenvolvido para três tipos de contexto: contexto temporal, contexto espacial e o contexto do usuário, viabilizados por meio das lógicas de primeira ordem, lógica temporal, lógica *fuzzy* ou mesmo a combinação delas. A expressividade do conhecimento através da combinação de conceitos heterogêneos é similar à abordagem híbrida do *framework* "CARE middleware" [47-48].

Basicamente, um especialista do domínio pode expressar seu conhecimento por meio de regras, através da própria linguagem de programação orientada a objetos, se utilizando dos próprios objetos e relações existentes no domínio do sistema. A abordagem OO é fundamental para os sistemas que o RT-MLR deseja apoiar, já que as escritas das regras podem se beneficiar da inerente capacidade de reusabilidade, utilizando os princípios OO de herança, associação e encapsulamento. Além disso, no modelo adotado, os objetos do sistema servem como referência para investigação das regras, se tornando, na realidade, fatos representativos do contexto.

Em razão da popularidade da linguagem e do seu objetivo final que é dar suporte a um projeto de sistema tático (ver o tópico "Motivação"), a linguagem de desenvolvimento Java foi escolhida. Entretanto, a abordagem adotada é extensível à maioria das linguagens comerciais de programação, desde que elas respeitem os princípios OO [20], como por exemplo, a linguagem Delphi e a linguagem C++.

4.2 Solução adotada

4.2.1 Arquitetura

A arquitetura do RT-MLR procura seguir os padrões de projeto de *software* existentes. O RT-MLR é uma ferramenta reutilizável de desenvolvimento do tipo "caixa-cinza", pois os desenvolvedores de sistemas terão de entender um pouco das funções de determinadas classes, além de necessitarem escrever classes do domínio que

estabeleçam uma relação de compromisso com algumas outras classes do RT-MLR. Mesmo que o RT-MLR não possua alguma característica particular, como por exemplo, outro operador, o desenvolvedor pode estender as funcionalidades existentes, acrescentando operações, classes e algoritmos de seu interesse.

Ainda que necessite estabelecer uma relação direta entre o projeto do sistema e as classes do RT-MLR, ressalta-se que não há restrições no desenvolvimento de sistemas no mapeamento conceitual – o desenvolvedor pode fazê-lo da maneira que achar conveniente. Torna-se apenas recomendável que os desenvolvedores sigam, posteriormente as etapas tradicionais de desenvolvimento de sistemas, os seguintes procedimentos:

1. Definir as lógicas em linguagem natural, documentadas da maneira que desejar.
2. Definir os objetos observáveis, isto é, aqueles que serão usados em alguma regra e que serão alvos de monitoração para investigação das regras.
3. Definir os tipos *fuzzy* existentes, isto é, aqueles que serão usados como variáveis de entrada ou saída em regras *fuzzy*.
4. Definir as regras em arquivos de códigos separados, de maneira genérica, para futuras reutilizações.

A arquitetura do RT-MLR especifica 7 entidades principais: as operações (classe *Operator*), as regras (classe *Rule*) compostas de operações recursivamente montadas com uma conclusão (classe *Conclusion*), os fatos (classe *ObservableData*), as expressões *fuzzy* (*Fuzzy*Data), o conjunto de regras (classe *RuleSet*) e um motor de inferência (classe *Reasoner*). A figura 4.1 apresenta uma visão conceitual da arquitetura do *framework.*

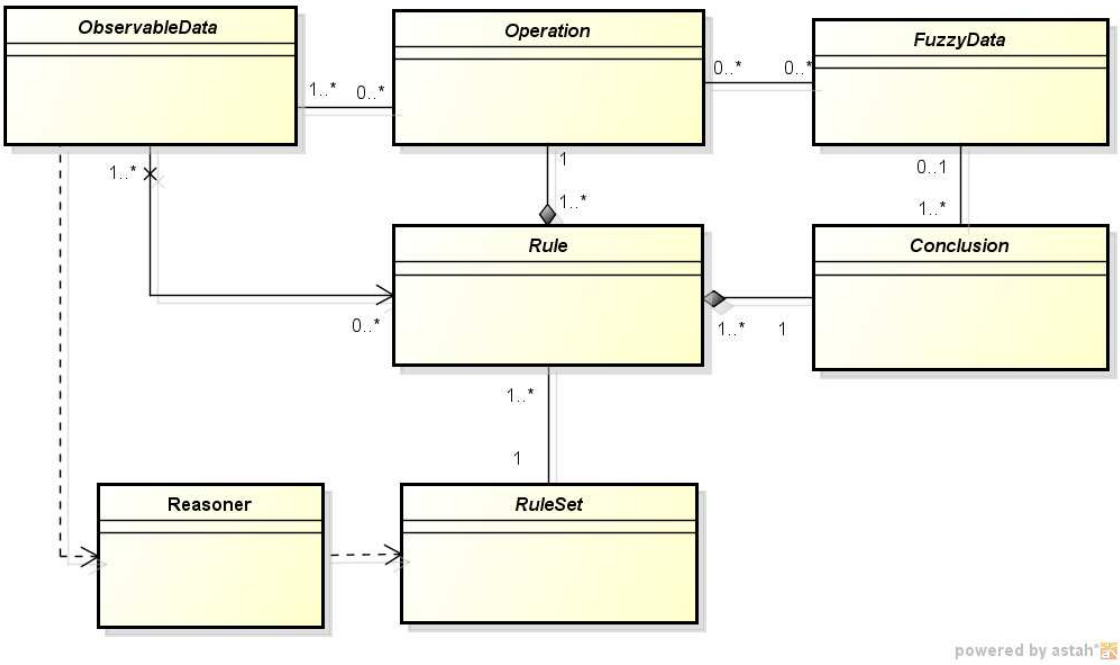

Figura 4.1: diagrama conceitual do RT-MLR

4.2.2 Definição de tipos observáveis

A classe *ObservableData* é a representante das entidades cujos valores atribuídos estão condicionados ao contexto em uma regra qualquer, sendo, conseqüentemente, fatos a respeito do contexto atual. A classe *ObservableData* também é responsável por encapsular os tipos básicos do Java, além de prover mecanismos que associam automaticamente um objeto às regras. Ao definir um atributo de uma classe do domínio do sistema como um tipo derivado de *ObservableData*, os objetos desta classe serão observáveis, e a alteração de qualquer valor deste objeto implica na investigação das regras associadas a este objeto. Os objetos *ObservableData* são os únicos que podem ser usados para expressar regras conectadas diretamente aos fatos, e a escolha da classe derivada implica diretamente no tipo de dado que será utilizado (figura 4.2).

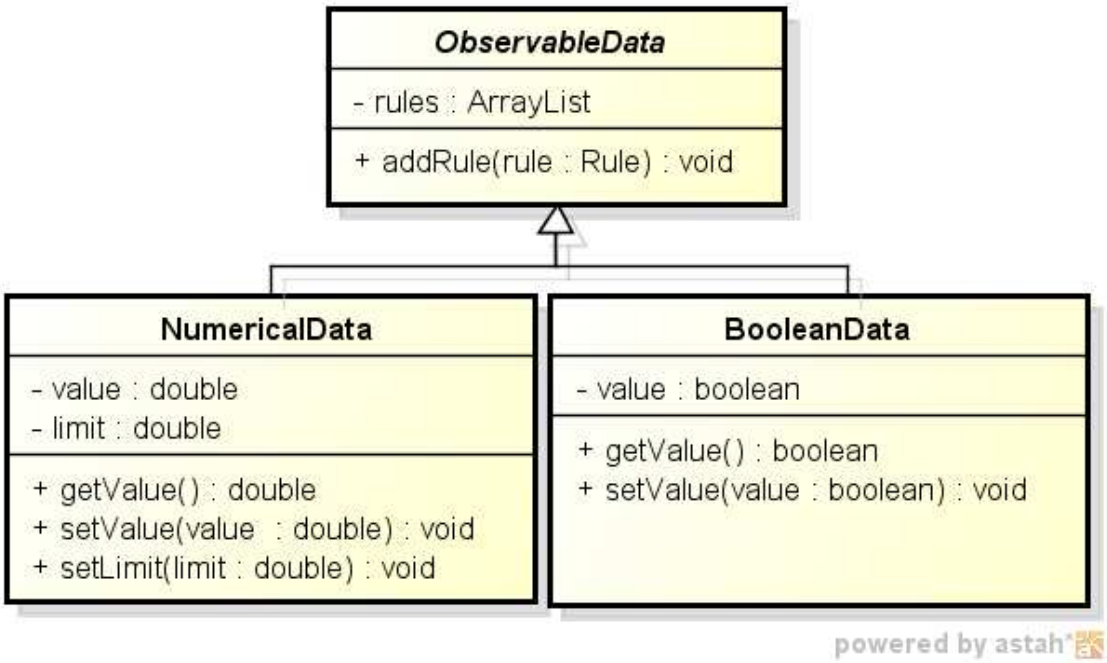

Figura 4.2: tipos observáveis

Como o próprio nome da classe sugere, as classes derivadas *NumericalData* e *BooleanData* representam os tipos básicos *double* e *boolean*, respectivamente. Nestas classes, os métodos *getValue* e *setValue* são os responsáveis por obter e alterar dados, além de proporcionar mecanismos para tratá-los em relação à possíveis concorrências de acesso. Mais do que isso, os métodos *setValue* destas classes também são os responsáveis por notificar atualizações, que por conseqüente investiga as regras associadas – objetos da classe *Rule.*

Por meio da operação *setLimit*, a classe *NumericalData* permite ainda o estabelecimento de um intervalo de tolerância para notificação. O valor do atributo do objeto somente será considerado relevante se extrapolar o limite pré-definido. Por exemplo, se o limite estabelecido for o valor de 3 unidades, então o novo valor só poderá ser considerado se for menor ou maior em 3 unidades do que o último valor previamente considerado.

O exemplo a seguir representa uma simplificação da etapa 2 (definição de objetos observáveis) do processo de desenvolvimento com o RT-MLR. Tome como exemplo a definição de uma classe de domínio chamada "*Target*" (Alvo). Esta classe possui dois atributos: *speed* e *lost*, que indicam, respectivamente, a velocidade do alvo e a indicação se o alvo está perdido ou não.

```
import RTMLR.types;
public class Target {
	NumericalData speed = new NumericalData();
	BooleanData lost = new BooleanData();
}
public class Main{
	public static void main(){
		Target target = new Target();
		target.speed.setValue(100.0);
		target.speed.setLimit(3.0);
		target.lost.setValue(true);
	}
}
```

Exemplo 4.1: definição de um objeto observável

4.2.3 Definição de tipos fuzzy

A classe *FuzzyData* é a responsável por encapsular valores escalares referentes a conjuntos de entrada e saída *fuzzy*, provendo ainda mecanismos de definição de funções de pertinência. Diferentemente dos objetos *ObservableData*, os objetos *FuzzyData* não representam fatos do contexto, mas sim variáveis lingüísticas que possam ser usadas para representar regras *fuzzy*. As classes derivadas de *FuzzyData* (figura 4.3) representam algumas funções comuns em definições desse tipo, cada qual com um construtor com os parâmetros indispensáveis à definição. Tais parâmetros representam apenas os valores da abscissa das funções, já que o valor máximo da ordenada é igual a 1 (valor máximo de pertinência).

Todas as classes derivadas de *FuzzyData* implementam um método *getAverage*, que tem por finalidade retornar o valor médio do conjunto, sendo projetado para uso no processo de *defuzzyficação.*

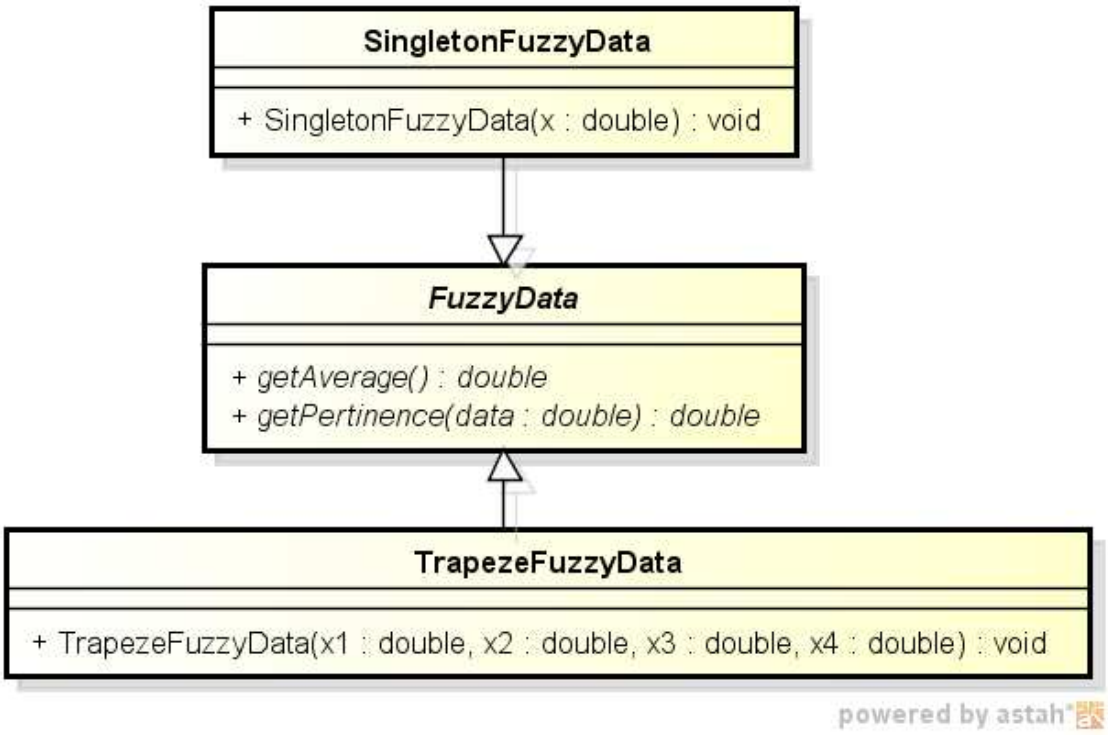

Figura 4.3: tipos fuzzy

O exemplo a seguir representa uma simplificação da etapa 3 (definição de conjuntos *fuzzy*) do processo de desenvolvimento com o RT-MLR. Tome como exemplo a definição uma variável lingüística chamada “tamanho pequeno” (*small size*). Esta variável corresponde a uma definição do termo “pequeno” para alvos segundo um especialista do domínio. Para este especialista, um alvo é considerado pequeno se ele estiver dentro dos limites de uma função trapezoidal cujos pontos de semi-retas são: 0, 2, 6, 8.

```
import RTMLR.types;
public class Main{
	public static void main(){
		TrapezeFuzzyData smallSize = new TrapezeFuzzyData(0.0, 2.0, 6.0, 8.0);
	}
}
```

Exemplo 4.2: definição de uma variável fuzzy

4.2.4 Definição de regras

No RT-MLR, uma regra é uma entidade representada pela classe *Rule*. As classes derivadas de *Rule* determinam a natureza da regra. A classe *FirstOrderRule*

representa uma regra de lógica de primeira ordem, tal qual a classe *FuzzyRule* representa uma regra *fuzzy*. Na solução adotada neste trabalho, optou-se por representar as regras temporais como regras de primeira ordem, com um tempo associado às condições estabelecidas.

Todas as classes derivadas de *Rule* são compostas por duas partes: uma premissa e uma conclusão. A premissa e a conclusão da regra são definidas e obtidas pelos métodos *getOperation*, *setOperation*, *getConclusion* e *setConclusion* (figura 4.4).

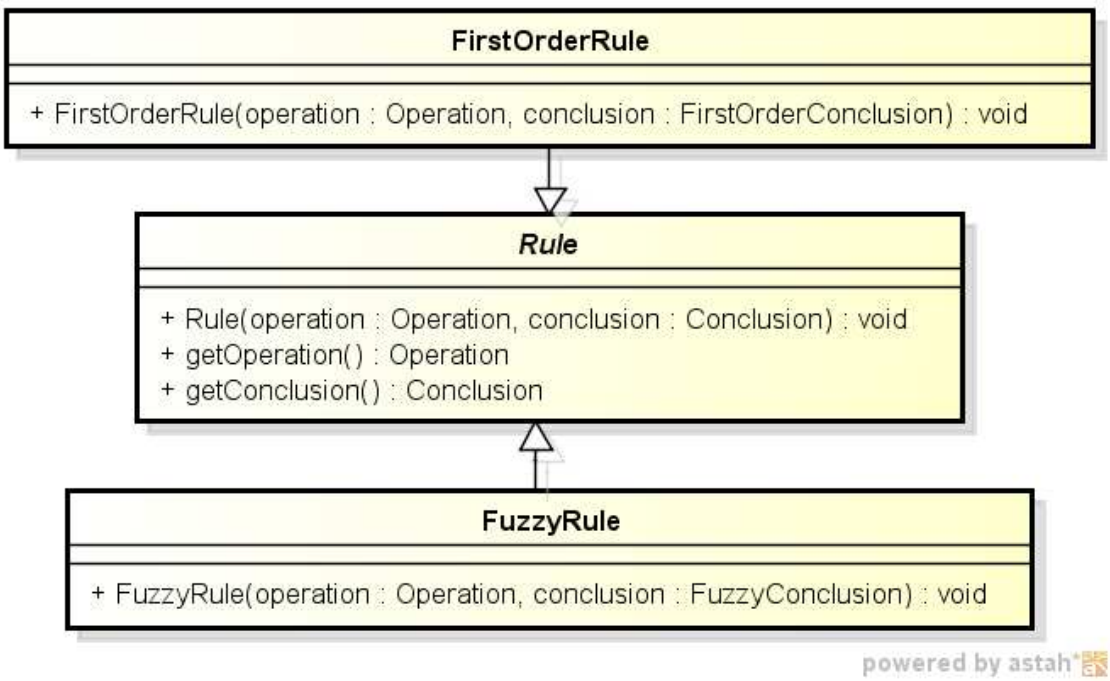

Figura 4.4: tipos de regras

Tanto as regras de primeira ordem, quanto as regras temporais e as regras *fuzzy* são dispostas como uma lista simples, guardadas em uma estrutura de *arraylist* na classe *RuleSet e* igualmente armazenadas no objeto *ObservableData*.

4.2.4.1 Definição da premissa da regra

O desenvolvimento da premissa é basicamente a combinação dos operadores existentes. Os operadores de uma premissa são combinados recursivamente, ao longo de uma árvore binária de busca.

A organização das operações é inspirada no algoritmo RETE, mas é diferente no sentido de não utilizar uma árvore para compartilhamento de operações comuns para todas as regras existentes. A diferença crucial na organização dos operadores no RT-

MLR para o algoritmo tradicional RETE está no particionamento das árvores de operadores, justamente para evitar a ligação de regras não relacionadas à alteração do contexto. O custo da adoção desta metodologia é um consumo maior de memória. Todavia, a eficiência prática da organização do RT-MLR é similar a uma organização RETE tradicional, pois um número menor de regras de interesse será avaliado.

No RT-MLR, cada operador é uma classe do *framework* derivada da classe de operação unária (*UnaryOperation*) ou da classe de operação binária (*BinaryOperation*) que implementam o método *getOperationResult* de acordo com a sua operação. As classes *UnaryOperation* e *BinaryOperation* são definidas no intuito de se estabelecer uma taxonomia de tipos de operação. Portanto, o uso dos operadores, bem como a combinação dos tipos existentes no RT-MLR estão restritos por meio dos construtores destas classes (figura 4.5). Assim, a única necessidade é a de instanciação dos objetos derivados da classe de operação que se deseja.

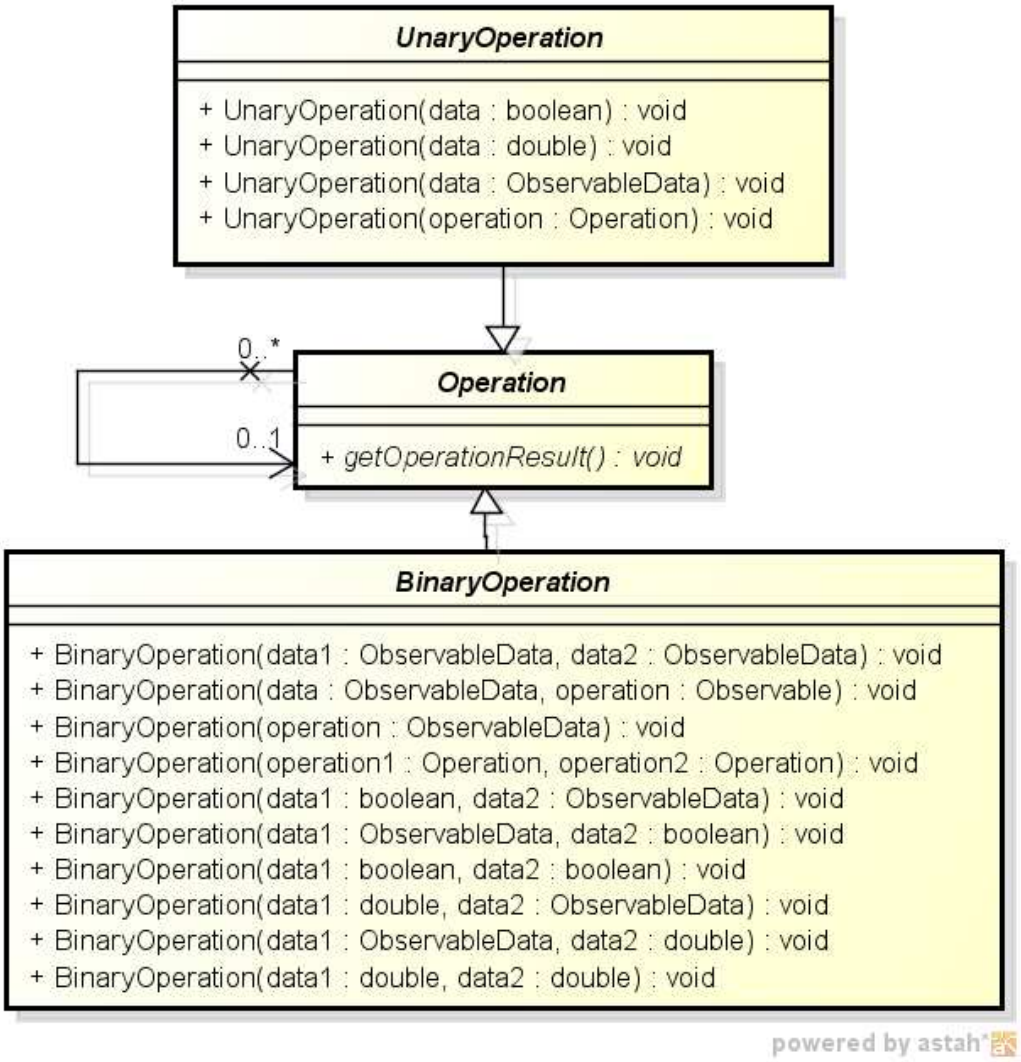

Figura 4.5: tipos de operadores

Os construtores das classes derivadas de *UnaryOperation* só permitem a passagem de um parâmetro, enquanto que as classes derivadas da classe *BinaryOperation* permitem a passagem de 2 parâmetros. A instanciação dos operadores unários ou binários pode ser realizada de quatro maneiras:

1) Por meio de combinação de um par de objetos *ObservableData* (ou apenas um objeto para as operações unárias) para operadores de primeira ordem ou temporais;
2) Pela obtenção de uma pertinência de um objeto *ObservableData* em relação a um conjunto *fuzzy* para operadores *fuzzy*.
3) Por passagem de um outro objeto operador compatível.
4) Combinado com instruções da linguagem Java que retornem ou um valor discreto (para operadores de primeira ordem ou temporais) ou um valor de precisão dupla (para operadores *fuzzy*).

Todos os operadores de primeira ordem mencionados por Russell e Norvig [33] e que são considerados essenciais para resolução de diversos problemas estão devidamente implementados. A tabela 4.1 apresenta os operadores de primeira ordem e o significado de suas operações.

Operador	**Significado**	**Resultado**
Eq	X é igual Y	Verdade se $x = y$, falso se $x \neq y$.
Not	Negação de X	Verdade se x = falso, falso se x = verdade.
Ht	X é maior que Y	Verdade se $x > y$, falso se $x \leq y$.
He	X é maior ou igual a Y	Verdade se $x \geq y$, falso se $x < y$.
Lt	X é menor que Y	Verdade se $x < y$, falso se $x \geq y$.
Le	X é menor ou igual a Y	Verdade se $x \leq y$, falso se $x > y$.
And	X E Y (E booleano)	Verdade se x e y forem verdadeiros. Senão é falso.
Or	X OU Y (OU booleano)	Verdade se x ou y forem verdadeiros. Senão é falso.
XOr	X OU EXCLUSIVO Y (XOR BOOLEANO)	Verdade se x e y forem iguais. Senão é falso.
Exits	$\exists$ (φ (L)).	Verdade se um ou mais elementos da lista l forem aplicados ao operador φ e retornarem verdade. Senão é falso.
All	$\forall$ (φ (L)).	Verdade se todos os elementos da lista l forem aplicados ao operator φ e retornarem verdade. Senão é falso.

Tabela 4.1: operadores de primeira ordem

Em razão de ter sido projetada como uma extensão dos operadores de primeira ordem, os operadores temporais (tabela 4.2) se comportam como uma operação auxiliar

a outras operação de primeira ordem, com condições associadas a um tempo. Portanto, os operadores temporais são necessariamente associados a outro objeto operador por meio da passagem de parâmetros. Além disso, para cada objeto *ObservableData* usado em um operação temporal, uma lista de valores associados ao seu tempo de obtenção (*timestamp*) é criada e mantida automaticamente no próprio objeto. Porém, esses valores só são guardados se satisfizerem à condição estabelecida pela operação principal.

Operador	Significado	Resultado
ThereWas	∃ (T2, φ , T1). * T1 < T2.	Verdade se o operador φ retornar verdadeiro uma ou mais vezes entre os tempos T2 e T1. Senão é falso.
Count	∑ (T2, φ , T1). * T1 < T2.	O número de vezes que o operador φ retorna verdadeiro entre os tempos T2 e T1.
Average	$\frac{\sum_{j=T2}^{T1} X_{Tj}}{n}$. * T1 < T2. * X_{Tj} = valor de X no tempo Tj que a mudança ocorreu * n= somatório dos tempos que X teve seu valor mudado.	O valor médio de X entre os tempos T1 e T2.
Occurs	∃ (*MODE*, φ, T). * MODE = {before, after}	Verdadeiro se operator φ for verdadeiro antes ou depois do tempo T.

Tabela 4.2: operadores temporais

Os operadores *fuzzy* implementados (tabela 4.3) no RT-MLR seguem a lógica de Zadeh [38]. Todos os operadores *fuzzy* implementados possuem uma letra ao seu final no intuito de não permitir ambigüidade no processo de escolha, pois alguns operadores de primeira ordem possuem nomes semelhantes.

Operador	Significado	Resultado
IsF	μS(X).	Pertinência do valor X de acordo com um conjunto *fuzzy* S.
AndF	Min(μ(X), μ(Y))	Pertinência mínima entre X e Y.
OrF	Max(μ(X), μ(Y))	Pertinência máxima entre X e Y.
NotF	1 - μ(X)	O complemento da pertinência X.

Tabela 4.3: operadores fuzzy

4.2.4.2 Definição da conclusão da regra

A conclusão de uma regra deve ser feita por meio da extensão da classe abstrata *Conclusion* (figura 4.6). O processo de criação da classe de conclusão do domínio prossegue pela necessidade de se sobrescrever o método *conclusion* de acordo com o tipo de resultado que se deseja. Caso a regra seja de primeira ordem ou temporal, o método *conclusion* a se sobrecarregar é o que possui o parâmetro booleano, proveniente da classe *FirstOrderConclusion*. Caso contrário, para o caso de uma regra *fuzzy*, o método *conclusion* que deverá ser implementado é aquele com um parâmetro de precisão dupla (*double*), proveniente da classe *FuzzyConclusion*.

Adicionalmente, para as conclusões *fuzzy*, é também necessário especificar um conjunto *fuzzy* de saída, uma variável lingüística que esteja em consonância com as condições estabelecidas na premissa da regra.

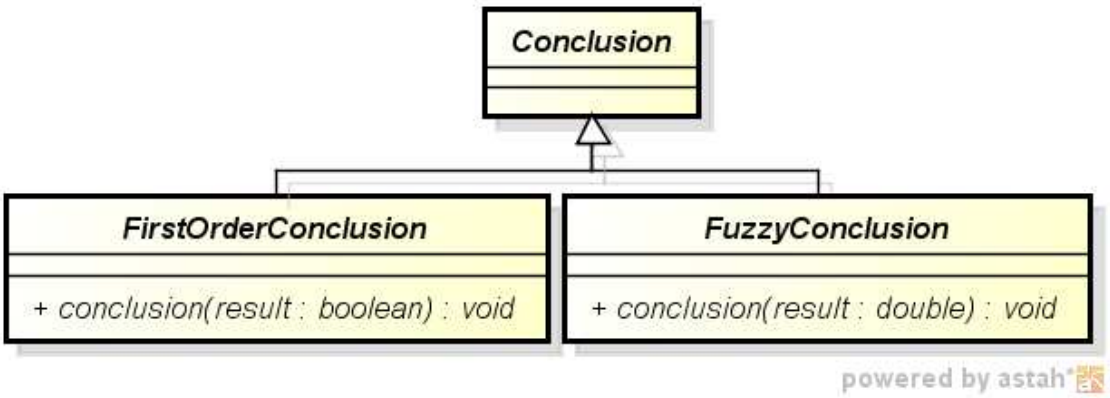

Figura 4.6: tipos de conclusões

Como poderá ser visto adiante no modelo de inferência, tanto para as regras de primeira ordem, quanto para as regras *fuzzy*, o respectivo método *conclusion* é invocado ao término da investigação da regra. Ao implementar a conclusão desejada, torna-se necessário avaliar o parâmetro passado ao método *conclusion* a fim de saber se o resultado da premissa é verdadeiro ou falso (para lógica temporal e de primeira ordem) ou o valor da pertinência da regra (para lógica *fuzzy*). Deste modo, a critério do desenvolvedor, todas as instruções contidas no método *conclusion* podem ser executadas em função do resultado obtido.

Por fim, vale salientar que não é possível a criação de conclusões que não estejam em consonância com a natureza da regra, representada por uma das classes

derivadas da classe *Rule*. Assim, não é possível criar uma conclusão *fuzzy* para uma regra de primeira ordem e *vice-versa*.

4.2.4.3 Agrupamento de regras e fatos

Uma abordagem tradicional para sistemas baseados em regras utiliza uma base de regras e uma base de fatos, para manuseio de regras e fatos respectivamente. Em razão do objetivo principal do RT-MLR de apoiar o desenvolvimento de sistemas de contexto com constantes alterações, pressupõe-se que os fatos só têm validade momentaneamente, com exceção de algumas operações temporais, uma característica que também o difere de um motor de inferência que utilize o algoritmo RETE. Isto significa que o RT-MLR não mantém efetivamente uma base de fatos de forma separada. Os objetos são os próprios fatos, e seus valores só importam no momento em que a regra é investigada (figura 4.7). Além disso, como as condições das regras são definidas em tempo de desenvolvimento, isto é, pré-compiladas, se torna igualmente desnecessária a utilização de um banco de regras tradicional.

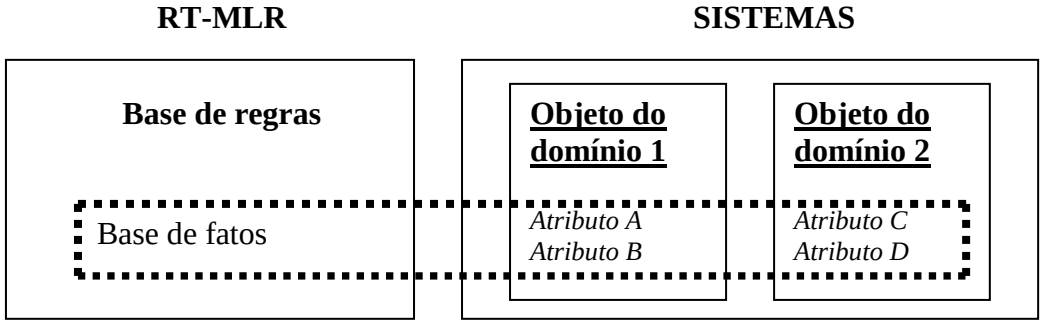

Figura 4.7: base de regras e fatos do RT-MLR. Adaptado de Carvalho et al. [44].

Independentemente do tipo de regra que está se desenvolvendo, toda regra deve fazer parte de um objeto representativo de um conjunto de regras de classe derivados da classe *RuleSet*, por meio da chamada do método *addRule*. O objeto *FirstOrderRuleSet* é o responsável por agrupar as regras de primeira ordem e temporais, bem como o objeto *FuzzyRuleSet* é o responsável por agrupar as regras *fuzzy*.

Para o caso das regras *fuzzy*, é especialmente necessário que uma classe do sistema estenda a classe abstrata *FuzzyRuleSet*, implementando o método *defuzzyfication*. Todas as vezes que o processo de inferência terminar, este método será

chamado com a passagem de um valor do tipo *double* como resultado do processo de *defuzzyficação*.

Assim, pode-se dizer que o *RuleSet* é a base de regras do RT-MLR, exceto pelo fato de que não é possível adicionar novas regras em tempo de execução (figura 4.8).

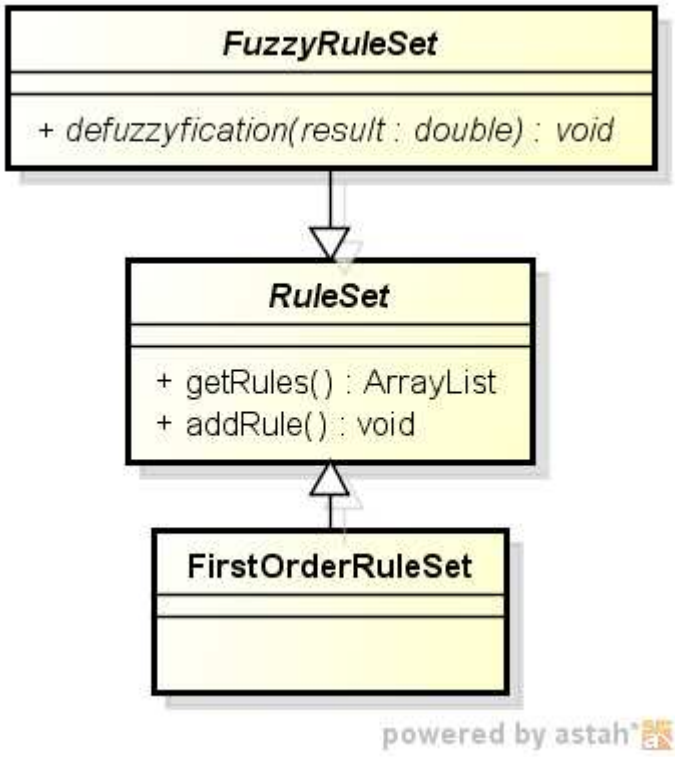

Figura 4.8: agrupamento de regras no RT-MLR

O exemplo 4.3 mostra como definir uma regra de primeira ordem e uma regra temporal por meio da abordagem adotada, tomando como classe de domínio a mesma apresentada no exemplo 4.1 (classe Target). A regra exemplificada trata da classificação de um alvo com base na velocidade.

Já o exemplo 4.4 mostra como definir uma regra *fuzzy* com base na mesma classe de domínio, utilizando um conjunto de entrada “velocidade alta” e um conjunto de saída “risco alto”, ambos fornecidos por um especialista. A regra exemplificada neste exemplo trata da atribuição de um grau de risco a um alvo com base no seu tamanho.

```
/**
** REGRA 1:
*** Se a velocidade de um alvo é maior que 300 nós há pelo menos 2 segundos,
*** Então o alvo é um míssil.
** REGRA 2:
*** Se o tamanho do alvo é maior do que 30 metros,
*** Então o risco é baixo.
**/
import RTMLR.rules;
import RTMLR.operations;
import RTMLR.conclusions;
public class Main{
        public class MissileConclusion extends FirstOrderConclusion{
                public void conclusion (boolean result){
                        if(result) System.out.println("It's a missile!");
                }
        }
        public class LowRiskConclusion extends FirstOrderConclusion{
                public void conclusion (boolean result){
                        if(result) System.out.println("Low risk!");
                }
        }
        public static void main(){
                Target target = Target.getTarget("A");
                Operation missilePremise = new ThereWas (new HE (target.speed, 300), 2000);
                Conclusion missileConclusion = new MissileConclusion ();
                FirstOrderRule rule1 = new FirstOrderRule (missilePremise, missileConclusion);
                Operation lowRiskPremise = new HE (target.size, 30);
                Conclusion lowRiskConclusion = new LowRiskConclusion ();
                FirstOrderRule rule2 = new FirstOrderRule (lowRiskPremise, lowRiskConclusion);
                FirstOrderRuleSet fset = new FirstOrderRuleSet();
                fset.addRule(rule1);
                fset.addRule(rule2);
        }
}
```

Exemplo 4.3: definição de uma regra de primeira ordem e uma regra temporal com o RT-MLR.

```
/**
** REGRA:
*** Se a velocidade de um alvo é alta,
*** Então o risco é alto.
**/
import RTMLR.rules;
import RTMLR.operations;
import RTMLR.conclusions;
public class Main{
        public class HighRiskConclusion extends FuzzyConclusion{
                public HighRiskConclusion (FuzzyData fdata){
                        super(fdata);
                }
                public void conclusion (double result){
                        System.out.println("Belief on the rule: " + result);
                }
        }
        public class ClassificationFuzzyRuleSet extends FuzzyRuleSet{
                public void defuzzyfication (double result){
                        System.out.println("Risk: " + result ) ;
                }
        }
        public static void main(){
                TrapezeFuzzyData highSpeed = new TrapezeFuzzyData (100,200,300,300);
                TrapezeFuzzyData highRisk = new TrapezeFuzzyData (60,80,100,100);
                Target target = Target.getTarget("A");
                Operation premise = new IsF(target.speed, highSpeed);
                Conclusion conclusion = new HighRiskConclusion (highRisk);
                FuzzyRule rule = new FuzzyRule (premise, conclusion);
                FuzzyRuleSet fset = new ClassificationFuzzyRuleSet();
                fset.addRule(rule);
        }
}
```

Exemplo 4.4: definição de uma regra fuzzy com o RT-MLR.

4.2.4.4 Combinação de operações lógicas

A criação de regras com operadores de diferentes tipos de lógica é uma das características fundamentais no RT-MLR para expressão do conhecimento acerca do contexto.

Na solução adotada neste trabalho, não existe nenhum tipo de verificação quanto à compatibilidade entre a natureza dos operadores e a natureza das regras em tempo de compilação. O que determina a natureza das regras é o tipo de classe derivada de *Rule* que foi utilizada, cuja resolução é feita em tempo de execução. Porém, mesmo que seja permitido mesclar operações distintas em uma regra, não é possível misturar regras *fuzzy* com regras de primeira ordem ou regras temporais em único *RuleSet.* Esta verificação é realizada em tempo de desenvolvimento, sendo a mescla de regras vetada sintaticamente.

Em conformidade a estas observações, este trabalho adota uma metodologia para compatibilização e adoção de operadores não naturais à lógica.

Se uma regra de primeira ordem for escrita utilizando operadores *fuzzy*, o resultado obtido de tais operações é encarado como um conjunto discreto, no qual o valor de pertinência igual a 1 retorna verdadeiro e, caso contrário, falso. Contudo, mesmo usando operadores *fuzzy*, o processo de *defuzzyficação* não será efetuado, pois não é possível usar o objeto *FuzzyRuleSet* – a regra originalmente é de primeira ordem.

De forma semelhante, em um determinado conjunto de regras *fuzzy*, o uso de um operador de primeira ordem ou temporal não altera a natureza do processo. Neste caso, o retorno de uma operação verdadeira é encarado como valor 1 e falso como valor 0. Assim, mesmo usando operadores de primeira ordem ou operadores temporais, o processo de *defuzzyficação* será efetuado.

O exemplo 4.5 mostra como definir uma regra utilizando operadores de diferentes naturezas.

```
/**
** REGRA:
*** Se a velocidade de um alvo é alta,
*** Então o risco é alto.
**/
public class Main{
        public class HighRiskConclusion extends FirstOrderConclusion{
                public void conclusion(boolean result){
                        if (result) System.out.println("Resultado: " + result);
                }
        }
        public static void main(){
                TrapezeFuzzyData highSpeed = new TrapezeFuzzyData (100,200,300,300);
                Target target = Target.getTarget("A");
                Operation premise = new IsF(target.speed, highSpeed);
                Conclusion conclusion = new HighRiskConclusion ();
                FirstOrderRule rule = new FirstOrderRule (premise, conclusion);
                FirstOrderRuleSet fset = new FirstOrderRuleSet();
                fset.addRule(rule);
        }
}
```

Exemplo 4.5: definição de uma regra de múltiplas lógicas com o RT-MLR.

4.2.5 Modelo de inferência

Para inferência de regras de primeira ordem e regras temporais, o RT-MLR trabalha com a hipótese de mundo fechado, com uma lógica não monotônica. As inferências temporais seguem a idéia de validação em um determinado intervalo de tempo, assim como os trabalhos dos autores Allen [35] e dos autores Bennett e Galton [36]. Por fim, o modelo de inferência de Mamdani [39] é o adotado para lógica *fuzzy*.

O método de investigação adotado é o de "encadeamento para frente", com uma pequena flexibilidade em relação à abordagem tradicional, pois instruções condicionais podem ser admitidas nas conclusões das regras, mesmo se elas forem falsas.

4.2.5.1 Responsabilidades

As classes derivadas de *RuleSet* desempenham um papel importante no modelo de inferência adotado no *framework*. Além de determinar a afinidade entre as regras, estas classes também carregam consigo informações que determinam a maneira pela qual a inferência do contexto é processada.

A precedência das regras é definida por meio da seqüência de adição das regras nos objetos derivados de *RuleSet* – as regras que são adicionadas anteriormente têm precedência sobre as regras posteriores. Com isso, a conclusão final da inferência para um conjunto de lógicas de primeira ordem ou temporal é baseada na conclusão da regra com maior prioridade adicionada ao *FirstOrderRuleSet*. Isto é, a inferência para regras de primeira ordem e regras temporais não produzem um resultado final baseado nos resultados individuais das regras. Assim, na maioria dos sistemas, será provavelmente necessário estabelecer apenas uma única base para todas as regras de primeira ordem e regras temporais. O que ocorre, de fato, é que a base de regras apenas serve de mecanismo para estabelecimento de prioridades.

Para o caso das regras *fuzzy*, a base de regras, isto é, o objeto *FuzzyRuleSet*, é responsável por agregar as regras que estão inter-relacionadas com o mesmo objetivo (variáveis de saída que aparecem na mesma conclusão). Como mencionado, todas as vezes que o processo de inferência terminar, o método *defuzzyfication* será invocado com um valor do tipo *double* como resultado do processo. No processo de *defuzzyficação*, só serão consideradas as regras cujas pertinências sejam maiores que zero.

O principal objeto do modelo de inferência é o objeto *Reasoner*. Ele é basicamente o responsável por mediar a interação e a chamada de métodos essenciais para resolução do raciocínio, como poderá ser observado no próximo tópico.

4.2.5.2 Interação

A figura 4.9 apresenta o diagrama de colaboração do motor de inferência do RT-MLR. Os eventos significativos do modelo de inferência estão numerados seqüencialmente, e serão utilizados durante a explicação.

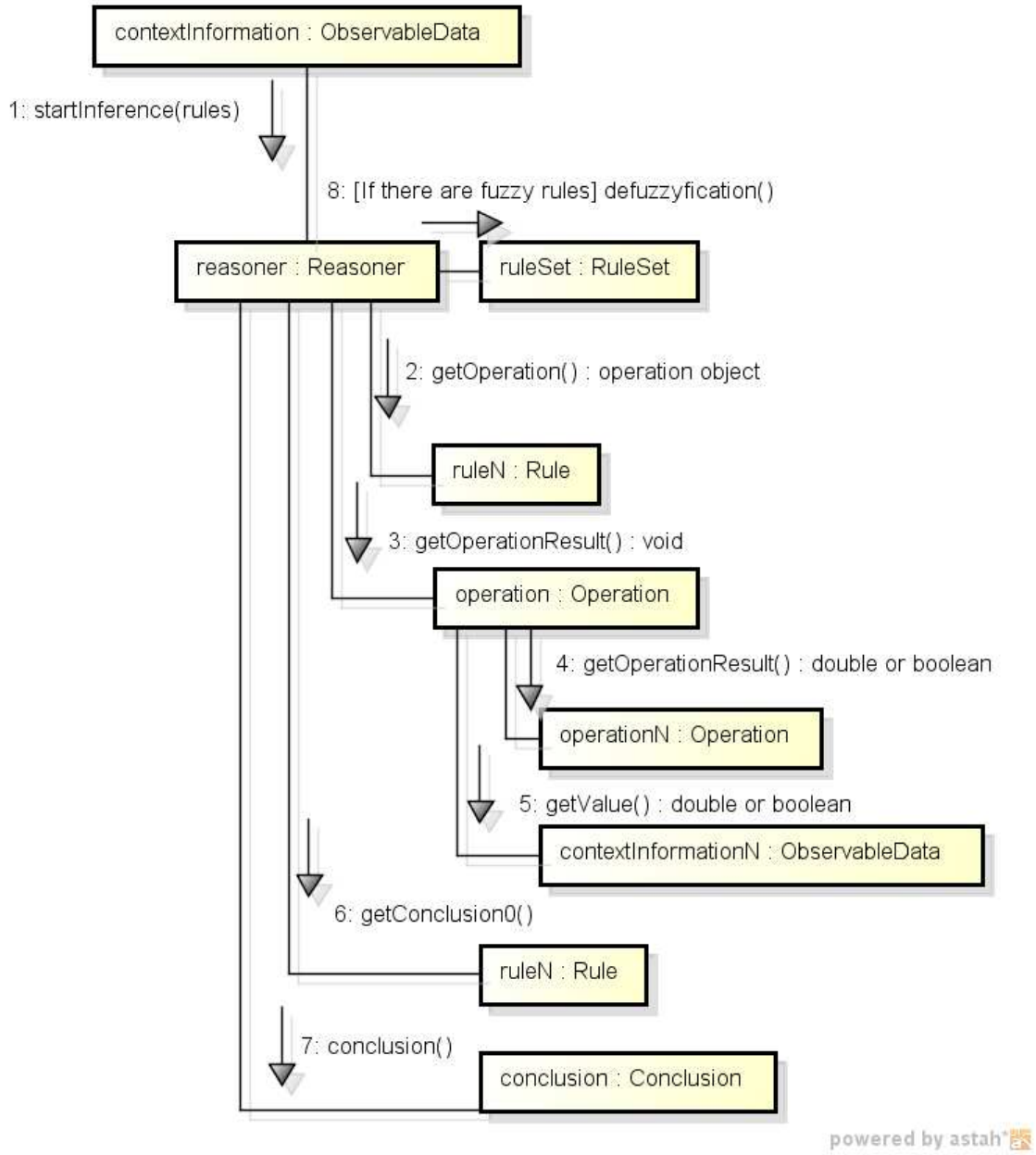

Figura 4.9: colaboração dos objetos no motor de inferência

Sempre que há um evento que gere a necessidade de investigar o contexto, o método *startInference* do objeto *Reasoner* é invocado com a passagem de um parâmetro *arraylist* que contém as regras de interesse do objeto observável (#1).

Após receber o conjunto de regras que serão investigadas, o método *startInference* do objeto *Reasoner* invoca o método *getOperation* da primeira regra do conjunto de regras passado (#2).

Após obter a operação raiz da árvore de operações, o método *getOperationResult* de cada operação é chamado igualmente de forma recursiva. A

condição de parada estabelecida é obtenção de um tipo primitivo de dado, como por exemplo, um dado de precisão dupla (*double*) (#3).

Posteriormente à investigação recursiva das operações, o objeto *Conclusion* da primeira regra do conjunto de regras é obtido pelo método *getConclusion* (#6). O método *conclusion* implementado pelo desenvolvedor da regra então é chamado (#7).

O motor de inferência agora invoca a segunda regra da lista de regras passada. O mesmo procedimento é executado tanto para premissa quanto para conclusão da regra. Este mesmo procedimento é então repetido para todas as regras passadas como parâmetro para o método *startInference*, até que a lista chegue ao seu final.

Caso existam regras *fuzzy*, o objeto *FuzzyRuleSet* apropriado é obtido e o método *defuzzyfication* é disparado (#8).

4.2.6 Monitoração de contexto em tempo real

O RT-MLR trabalha com a abordagem baseada em eventos, cuja idéia é investigar as regras somente quando eventos significativos ocorrem no contexto, quer seja o contexto temporal, o contexto espacial ou contexto do usuário.

Primeiro, é importante destacar a concepção e o significado de eventos no RT-MLR. Neste caso, o conceito de eventos está relacionado fixamente à alteração dos estados dos objetos do sistema, em razão destes serem partes da modelagem do contexto e representarem entidades ou atributos relevantes para investigação das regras.

Para que o contexto seja monitorado de forma contínua, os sistemas desenvolvidos com este *framework* devem descrever cada regra por meio dos próprios objetos do sistema. Assim, a única imposição é que tais objetos sejam instâncias de classes derivadas de *ObservableData.* Estes objetos encapsulam os tipos primitivos do Java e, por conseqüência, seus valores só podem ser acessados e alterados por meio dos métodos *getValue* e *setValue* respectivamente.

A chave para monitoração em tempo real do RT-MLR está na possibilidade de investigação de um número menor de regras, bem como na disponibilidade imediata dos fatos, pois somente as regras associadas a um determinado objeto *ObservableData* serão preponderantes no processo.

Neste *framework*, a implementação da abordagem baseada em eventos se torna possível graças ao uso de padrões de projeto descritos por Gamma *et al.* [49], em especial, o padrão de projeto “Observer”, cuja idéia principal é definir uma relação de

notificação, na qual os objetos dependentes (observadores) são avisados quando um evento para qual eles são dependentes ocorre.

No RT-MLR, os objetos observadores são os objetos da classe *Rule*. Ao descrever uma regra com os objetos domínio que herdam da classe *ObservableData*, o desenvolvedor do sistema faz com que os objetos da classe *Rule* se cadastrem automaticamente em uma lista única de observadores dos objetos *ObservableData* (figura 4.10). Mesmo que a regra utilize o mesmo objeto *ObservableData* duas vezes, apenas uma vez essa regra será cadastrada. Tal processo ocorre quando uma regra é adicionada a algum *RuleSet.*

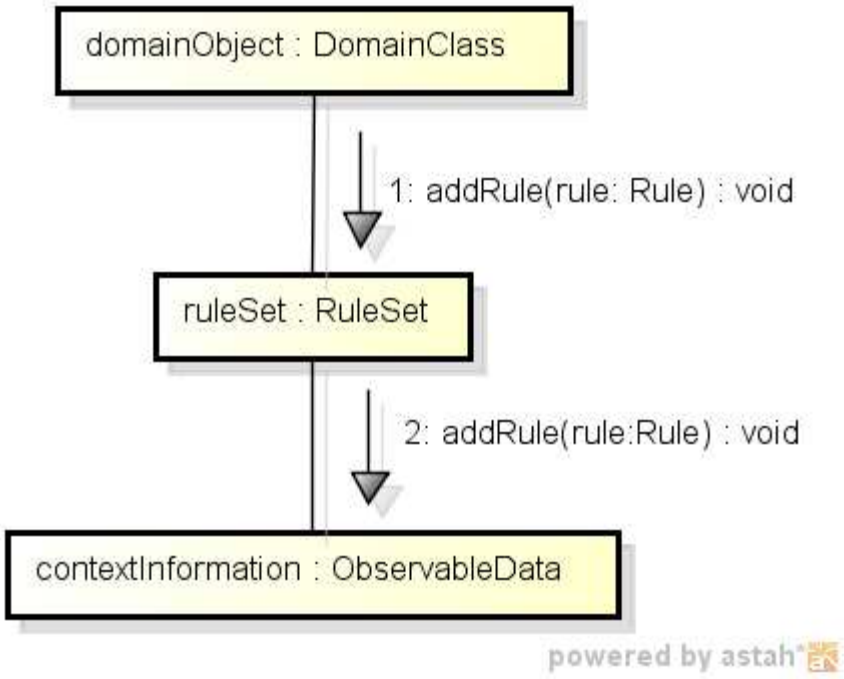

Figura 4.10: cadastro de observadores

A ocorrência do evento de mudança do contexto está ligada à invocação do método *setValue* do objeto *ObservableData*. Quando este método é chamado, necessariamente um novo valor é passado e atualizado, dando início ao procedimento de investigação das regras conforme explicado no tópico “Modelo de inferência”. Assim, quando o método *setValue* é chamado, um valor booleano ou de precisão dupla (dependendo do objeto *ObservableData*) é atualizado, e, automaticamente, o método *startInference* do objeto Reasoner é invocado. Como pode ser percebido, a inferência do modelo de monitoração do RT-MLR está diretamente ligada ao método de inferência adotado. Com isso, se o sistema que estiver sendo desenvolvido puder ter suas regras

relacionadas a eventos particulares em tempo de desenvolvimento, melhor será a sua eficiência.

O efeito prático desta abordagem é o de que todas as vezes que um valor de um objeto *ObservableData* for alterado, todas as regras associadas a ele serão investigadas. A figura 4.11 mostra o processo de monitoração no RT-MLR.

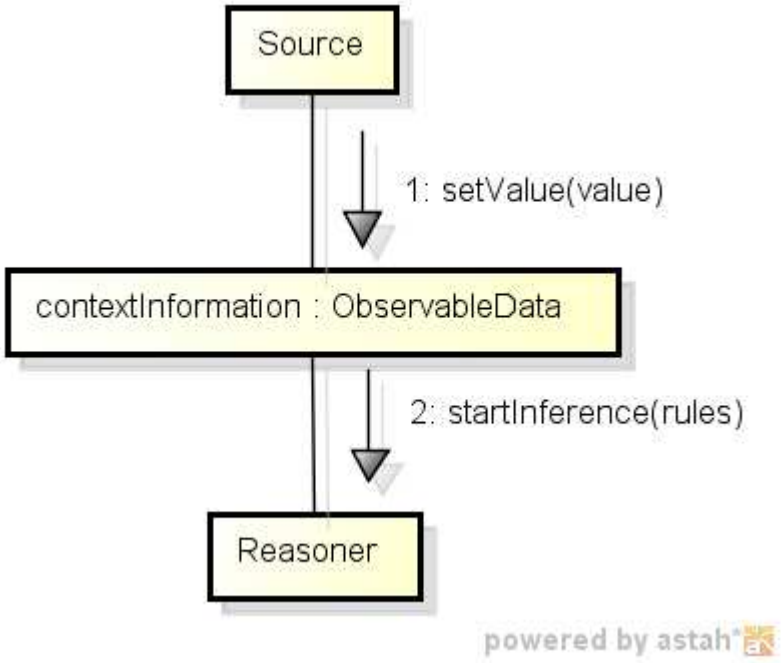

Figura 4.11: início da inferência no RT-MLR

4.2.7 Associação de novos objetos do sistema às regras já definidas

A abordagem adotada para escrita de regras é definida com os objetos do sistema e não com classes. Como se sabe, esses objetos são instâncias de classes, e representam materialização dos conceitos, atributos e operações definidas para uma determinada entidade do mundo real. Assim, a princípio, as regras só são definidas apenas para aquelas instâncias utilizadas na definição das regras.

O efeito positivo desta abordagem é a melhor expressão do conhecimento com os objetos do domínio. Já o efeito negativo é o baixo nível de manutenibilidade de *software*, pois o uso da mesma regra para um objeto diferente só pode ser realizada mediante a reescrita da regra. Para tornar mais claro o problema da manutenibilidade, suponha o exemplo a seguir (exemplo 4.6).

Seja a classe já definida em exemplos anteriores chamada *Target* (alvo). Sejam, ainda, dois objetos alvos (instâncias da classe *Target*) e uma mesma regra. A escrita da

regra para estes dois alvos incidirá em uma repetição de código, um retrabalho desnecessário por parte do desenvolvedor. Se esta mesma regra tiver que ser mantida por algum motivo, poderá ocorrer inconsistência devido à falha humana de repetição.

```
/**
** REGRA:
*** Se a velocidade de um alvo é maior que 300 nós,
*** Então o alvo é um míssil.
**/
import RTMLR.rules;
import RTMLR.operations;
import RTMLR.conclusions;
public class Main{
        public class MissileConclusion extends FirstOrderConclusion{
                public void conclusion (boolean result){
                        if(result) System.out.println("It's a missile!");
                }
        }
        public static void main(){
                Target target1 = Target.getTarget("A");
                Operation premise1 = new He(target1.speed, 300);
                Conclusion conclusion1 = new MissileConclusion ();
                FirstOrderRule rule2 = new FirstOrderRule (premise1, conclusion1);
                Target target2 = Target.getTarget("B");
                Operation premise2 = new He(target1.speed, 300);
                Conclusion conclusion2 = new MissileConclusion ();
                FirstOrderRule rule2 = new FirstOrderRule (premise2, conclusion2);
                FirstOrderRuleSet fset = new FirstOrderRuleSet();
                fset.addRule(rule1);
                fset.addRule(rule2);
        }
}
```

Exemplo 4.6: problema na definição de uma mesma regra para dois objetos de uma mesma classe.

A solução deste problema é inerente à modelagem Orientada a Objetos. Uma solução possível para este problema é a criação de uma classe no domínio do sistema que encapsule a escrita da regra. Assim, se o problema é relacionado à classificação, uma classe *Classification* pode ser criada visando o encapsulamento desta regra. O

exemplo 4.7 mostra como realizar esta tarefa, o que equivale à etapa 4 de desenvolvimento de um sistema sensível ao contexto com o RT-MLR.

```
/**
** REGRA:
*** Se a velocidade de um alvo é maior que 300 nós,
*** Então o alvo é um míssil.
**/
import RTMLR.rules;
import RTMLR.operations;
import RTMLR.conclusions;
public class Classification{
        private static FirstOrderRuleSet fset = new FirstOrderRuleSet();
        public MissileConclusion extends FirstOrderConclusion{
                public conclusion (boolean result){
                        if(result) System.out.println("It's a missile!");
                }
        }
        public Classification (Target target){
                Operation premise = new He(target.speed, 300);
                Conclusion conclusion = new MissileConclusion ();
                FirstOrderRule rule = new FirstOrderRule (premise, conclusion);
                fset.addRule(rule);
        }
}
public class Main{
        public static void main(){
                Target target1 = Target.getTarget("A");
                Target target2 = Target.getTarget("B");
                new Classification (target1);
                new Classification (target2);
        }
}
```

Exemplo 4.7: solução para definição de uma mesma regra para dois objetos de uma mesma classe.

A limitação desta solução continua sendo o consumo de memória. As definições das regras não são replicadas no código fonte, mas são replicadas na memória da aplicação. A diferença do exemplo 4.7 para o exemplo 4.6 está na impossibilidade de

gerar inconsistências em razão de uma falha de modelagem. A solução do RT-MLR sacrifica a memória em função do objetivo que é o de usar os próprios objetos do sistema como entidades das regras de domínio.

5. TRABALHOS RELACIONADOS

5.1 Considerações

O desenvolvimento deste *framework* utiliza idéias de diversas áreas da computação, como por exemplo, conceitos da Engenharia de *Software*, métodos de Inteligência Artificial e algoritmos matemáticos.

Muitos trabalhos utilizam diferentes tipos de lógicas aplicadas a diferentes contextos. Lógica *fuzzy*, por exemplo, é amplamente utilizada quando se é necessário expressar conhecimento incerto de domínios militares [50-51]. Alguns trabalhos estão mais preocupados com o formalismo e implementação da lógica temporal [34, 45] e outros trabalhos estão preocupados em prover uma maneira de utilizar lógica *fuzzy* integrada com projetos de *software*s OO [52]. Existem ainda outros trabalhos concentrados em adicionar ou melhorar algum recurso de algoritmos existentes, como por exemplo, adicionar expressão de incerteza no algoritmo RETE [53].

A integração entre regras e objetos para representação adequada do contexto já foi proposta por alguns autores. Em 1994, por exemplo, os autores Bento *et al.* [54] propuseram o "XLog+", uma extensão de um ambiente de integração já existente chamado "XLog". O trabalho dos autores propunha uma maneira de integração eficiente entre a lógica de primeira ordem e objetos OO. Este mesmo trabalho foi explanado em paralelo na dissertação de mestrado de Prates [55].

De uma forma um pouco mais recente e em consonância à evolução tecnológica, a combinação e implementação de conceitos tem sido o principal objetivo perseguido por alguns outros autores. Um exemplo deste tipo de pesquisa com viés mais prático é o software "JEOPS", que foi desenvolvido e explanado na dissertação de mestrado de Santos [56].

Nesse cenário, salvo as questões conceituais relacionadas à Engenharia de *Software*, a resultante deste trabalho acadêmico é um produto, e como tal, deve ser comparado a outros produtos com a mesma finalidade. O RT-MLR é uma aplicação prática dos diversos conceitos relacionados. Não há um profundo paralelo a ser traçado entre os conceitos do RT-MLR e outros trabalhos com enfoques mais conceituais, como é o caso do "XLog+". Em contrapartida, pode-se afirmar que o RT-MLR compartilha

das preocupações e objetivos do JEOPS e suas características são afins ou complementares (apesar da descontinuação do JEOPS[1]).

5.2 Critérios para amostra e comparação

Posto as considerações e contextualização deste trabalho, torna-se necessário estabelecer critérios para amostra e comparação com produtos semelhantes.

Além da afinidade entre os objetivos, uma natural escolha da amostra pode ser efetuada sob dois aspectos tecnológicos: a linguagem de programação e a freqüência de atualização.

Comparar produtos desenvolvidos em diferentes linguagens introduz dificuldades no estabelecimento de um paralelo entre as funcionalidades existentes. Por exemplo, a linguagem de programação Delphi e a linguagem Java possuem mecanismos para declaração de variáveis associadas a métodos de classes e eventos. No Delphi, essas variáveis são chamadas *properties*, e possuem comportamento semelhante ao mecanismo *JavaBeans* da linguagem Java. No entanto, não é possível apontar mecanismos semelhantes em outras linguagens OO, como por exemplo, a linguagem de Scripts PHP. Por esta razão, a implementação de determinadas funcionalidades em alguns produtos de *software* pode variar conforme a linguagem de programação adotada.

É igualmente necessário atentar para a freqüência de atualização corretiva ou evolutiva do produto de *software* que está se comparando. Trivialmente, não faz sentido efetuar comparações com produtos descontinuados, ou mesmo levar em consideração produtos com intervalos muito grandes de produção.

Dados os critérios para a amostra, é também necessário estabelecer os critérios para comparação. Descritos logo abaixo, tais critérios adotados são referentes às características e recursos presentes na sintaxe e implementação do RT-MLR:

1. Os produtos que apóiam a criação de sistemas baseados em regras são avaliados em relação à presença e à capacidade de expressão do conhecimento por meio de regras *fuzzy*, regras temporais e regras de primeira ordem.

[1] A última atualização do JEOPS é de Abril de 2007.

2. Estes mesmos produtos são avaliados quanto a sua capacidade de integração com a linguagem de programação OO Java paralelamente à solução adotada no RT-MLR – seus prós e contras;

3. Por fim, um comparativo é traçado entre o RT-MLR e os produtos semelhantes quanto à capacidade de executar uma monitoração do contexto de forma automática.

5.3 Amostra de produtos semelhantes

5.3.1 JESS

O *software* JESS (*Java Expert System Shell*) [26] foi criado em 1997 pelo pesquisador Ernest Friedman-Hill. O JESS é considerado o primeiro produto de *software* cujo objetivo é integrar os objetos de um sistema OO com regras de domínio. A história de criação do JESS [26] passa pelo desenvolvimento do CLIPS [27], um *software* para criação de sistemas baseados em regras criado pela NASA, utilizado em sistemas construídos com as linguagens de programação C/C++. O CLIPS tem sido amplamente utilizado em sistemas de diversos propósitos, inclusive em sistemas militares para cálculo de rota e extração de fumaça de navios pertencentes à Marinha do Brasil.

Com um motor de inferência com algoritmo RETE similar ao CLIPS, o JESS é na verdade uma interface para comandos – uma *shell*, cujos comandos são utilizados em uma linguagem própria, que além de permitir a criação de scripts de regras de interpretação, possibilita que os objetos Java sejam utilizados como ligações para os fatos do contexto.

Por meio do JESS, a monitoração do contexto pode ser feita de forma automática através da definição de eventos pré-determinados. Assim como no RT-MLR, os objetos Java são os meios de relacionamento com o contexto, e a alteração destes objetos resulta na investigação de todas as regras.

Para um objeto de um sistema poder ser usado nas regras escritas no JESS, este deve ser configurado de acordo com o padrão de nomenclatura de componentes, o JavaBeans. A diferença do RT-MLR para o JESS é a de que no JESS existe uma base de fatos separada para os objetos Java utilizados nas condições das regras, enquanto que no RT-MLR todos os fatos do contexto são objetos *ObservableData*.

Apesar de usar os objetos Java do sistema, a efetiva integração do JESS com a linguagem de programação é muito prejudicada, uma vez que a linguagem de definição de regras utiliza uma sintaxe particular, que exige do desenvolvedor, além de um período de adaptação e aprendizado, a integração por meio de mecanismos intermediários. Com isso, se por um lado a abordagem do JESS permite maior flexibilidade na definição/adição/remoção de regras em tempo de execução, por outro lado a abordagem adotada necessita de um maior trabalho na integração com os objetos do domínio. As regras expressas e guardadas em uma base separada resultam, conseqüentemente, em um custo computacional mais alto para casamento entre fatos e regras.

Em relação à expressividade do conhecimento, o JESS permite apenas a definição de regras de primeira ordem.

5.3.2 Hammurapi Rules

Hammurapi Rules [29] é um *software* desenvolvido em Java, cuja principal idéia, segundo seus desenvolvedores (Hammurapi Group), é a de apoiar o desenvolvimento de sistemas baseados em regras de uma maneira mais simples e rápida.

O motor de inferência do Hammurapi Rules [29] oferece a possibilidade de utilização dos dois métodos tradicionais de investigação, o “encadeamento para frente” e o “encadeamento para trás”, com a implementação do tradicional algoritmo de inferência RETE.

O modelo de escrita de regras do Hammurapi Rules é semelhante ao método utilizado neste trabalho. Tanto o Hammurapi Rules quanto o RT-MLR permitem utilizar a própria linguagem Java para escrita das regras e, conseqüentemente, os próprios objetos do domínio. A diferença crucial entre as abordagens é a de que o Hammurapi Rules não especifica objetos especiais de operações, uma vez que só a lógica de primeira ordem está disponível. No Hammurapi Rules os operadores presentes na linguagem Java são os responsáveis por descrever as relações e condições presentes nas regras.

Nesse sentido, apesar da forte integração com a linguagem de programação, a capacidade de expressão do conhecimento fica prejudicada, pois está intrinsecamente limitada aos recursos da linguagem de programação. O RT-MLR define objetos especiais para descrição das relações, que por um lado torna a curva de aprendizado do

produto maior do que a do Hammurapi Rules. Contudo, no RT-MLR, a capacidade de expressão não está limitada aos recursos da linguagem de programação. Ressalta-se, ainda, que o RT-MLR permite a extensão das lógicas existentes por mecanismos de herança OO, apresentando-se com relativa flexibilidade para futuras manutenções e melhorias.

Como o Hammurapi Rules adota o mesmo mecanismo para expressão de regras que o RT-MLR – regras compiladas –, o problema na administração das regras é similar.

Nas documentações do Hammurapi Rules [29] não foi possível observar nenhum recurso especial em relação à monitoração do contexto de forma automática. Deste modo, presume-se que a única maneira de associar o início da investigação das regras com a alteração do contexto é de maneira manual, isto é, através de instruções de código desenvolvidas particularmente para um determinado sistema.

5.3.3 Drools

Mantido pela empresa RedHat, o Drools [28] é uma família de projetos escritos em Java, que visa apoiar a criação de regras de domínio integradas com códigos de *software*. Como um produto de *software*, o Drools se mostra bem completo, com ferramentas de apoio e interfaces gráficas para desenvolvimento de regras de domínio.

Basicamente, o motor de inferência do Drools oferece a possibilidade de utilização do método de “encadeamento para frente” e do método de “encadeamento para trás” como abordagem de investigação. Tal qual o *framework* Hammurapi Rules, o algoritmo de inferência presente no Drools é o RETE [30], adaptado para sistemas orientados a objetos.

Assim como o JESS e o Hammurapi Rules, o Drools oferece a possibilidade de escrever as regras de negócio utilizando os objetos Java do domínio, que, no entanto, de forma diferente ao Hammurapi Rules a ao próprio RT-MLR, necessita que as regras sejam definidas por meio de uma sintaxe proprietária.

Tal qual o produto JESS, o Drools tem uma integração deficitária com a linguagem de programação Java, uma vez que a linguagem de definição de regras utiliza uma sintaxe particular, que exige do desenvolvedor, além de um período de adaptação e aprendizado, a integração por meio de mecanismos intermediários. O Drools necessita que o desenvolvedor especifique as regras por meio de um arquivo separado do código fonte do sistema, além de necessitar que o desenvolvedor especifique um arquivo XML

que contemple a interpretação e o casamento das regras de domínio com os objetos Java. Isto é, as regras são compiladas exatamente como o RT-MLR faz, contudo, casadas com os fatos pelo motor de inferência.

O problema na administração das regras é evidente assim como no RT-MLR. Entretanto, o Drools não se vale da pré-compilação das regras para uma integração completa com a linguagem de programação, não havendo, sob esta ótica, vantagem na necessidade de compilação das regras.

Sistemas desenvolvidos com o Drools podem monitorar o contexto de forma manual ou automática. O modo de monitoração manual consiste na explicitação de instruções de código, com chamadas de métodos para o início do processo de investigação do contexto. Já o processo de monitoração automática é baseado na abordagem de investigação periódica, que requer que o desenvolvedor estipule um tempo para verificação de novos fatos.

Já acerca da expressividade do conhecimento, o Drools permite a definição de regras em dois tipos de lógicas: lógica de primeira ordem e lógica temporal. A lógica de primeira ordem está presente desde as primeiras versões do Drools, enquanto que a lógica temporal só foi acrescentada há pouco tempo. No Drools, a abordagem para implementação da lógica temporal é similar à abordagem adotada no RT-MLR, onde os operadores são expressos de forma qualitativa.

5.4 Tabela comparativa

Os critérios estabelecidos para análise são colocados de forma a definir um grau de adesão aos requisitos já implementados no RT-MLR. Os graus de adesão são definidos em termos de:

- ausência do recurso (AUSENTE);
- presença parcial do recurso (PARCIALMENTE);
- presença completa do recurso (PRESENTE);

Posto isso, neste trabalho é considerado para integração com a linguagem de programação:

- Como presente (PRESENTE) caso o produto ofereça uma maneira de implementar as regras por meio dos próprios objetos JAVA, sem scripts intermediários;

- Como parcial (PARCIALMENTE) caso o produto ofereça uma maneira de implementar as regras por meio dos próprios objetos JAVA, com scripts intermediários;

- Como ausente (AUSENTE) caso o produto não ofereça nenhuma maneira de implementar as regras por meio dos próprios objetos JAVA.

Além disso, é considerado para a monitoração do contexto:

- Como presente (PRESENTE) caso o produto ofereça as duas maneiras mencionadas de monitoração automática - por monitoração periódica e por monitoração por eventos;

- Como parcial (PARCIALMENTE) caso o produto ofereça ao menos uma das duas maneiras mencionadas de monitoração automática - por monitoração periódica ou por monitoração por eventos;

- Como ausente (AUSENTE) caso o produto não ofereça ao menos uma das duas maneiras mencionadas de monitoração automática - por monitoração periódica ou por monitoração por eventos;

Finalmente, são considerados para a expressão de regras de primeira ordem, regras temporais e regras *fuzzy*:

- Como presente (PRESENTE) caso o produto ofereça ao menos uma operação referente à lógica;

- Como ausente (AUSENTE) caso o produto não ofereça ao menos uma operação referente à lógica;

Assim, a tabela 5.1 apresenta a comparação de três produtos brevemente descritos com base nas características do RT-MLR:

Produto / Critérios	JESS	Hammurapi Rules	Drools
Integração com a linguagem de programação	Parcialmente	Presente	Parcialmente
Monitoração automática do contexto	Parcialmente	Ausente	Parcialmente
Expressão de regras de primeira ordem	Presente	Presente	Presente
Expressão de regras temporais	Ausente	Ausente	Presente
Expressão de regras de *fuzzy*	Ausente	Ausente	Ausente

Tabela 5.1: comparação entre produtos semelhantes ao RT-MLR

Em relação à integração com a linguagem de programação, os três produtos apresentam algum tipo de relação com a linguagem Java. É passível de uma explicação razoável o fato de que nem o Drools e nem o JESS tenham obtido a classificação máxima (“presente”). Ambos têm propósitos um pouco mais gerais do que o Hammurapi Rules e o próprio RT-MLR. Em vista disso, o Drools e o JESS abdicam de um compromisso maior com a integração, buscando uma maior abrangência de apoio no desenvolvimento. O JESS e o Drools auxiliam a criação de sistemas baseados em regras em Java, mas só a experiência na linguagem de programação não é suficiente para o processo de desenvolvimento.

A monitoração do contexto está presente no JESS da mesma forma que o RT-MLR, por meio da relação entre os objetos do domínio e o contexto. A monitoração automática oferecida pelo JESS baseia-se na abordagem de “monitoração por eventos”, na qual sempre que os objetos do domínio alteram, as regras são investigadas. Apenas uma ressalva deve ser feita a este respeito: no RT-MLR, as regras são associadas diretamente a objetos e, somente essas regras serão investigadas. No caso do JESS, todas as regras são investigadas sempre que um dos objetos do domínio sofre alteração. A vantagem computacional associada à monitoração por eventos parece ser desprezível, pelo menos sob este aspecto. O Drools, que também possui um mecanismo de monitoração automática, pode vir a ter melhor desempenho que o JESS, mesmo adotando a abordagem de “monitoração periódica”. O número de regras e o domínio do sistema serão determinantes para uma possível medida de desempenho entre as duas abordagens.

Desconsiderando o número de operadores, pode-se afirmar que a expressão do conhecimento por meio de regras de primeira ordem está presente nos três produtos,

enquanto que as regras temporais só estão presentes no Drools. Por fim, nenhum dos três produtos apresentou alguma maneira para expressão de conhecimento por incerteza, quer seja através da lógica *fuzzy*, quer seja por qualquer outra lógica não convencional.

6. APLICAÇÃO DO RT-MLR A UM CONTEXTO MILITAR

6.1 Contexto de um sistema militar naval

Os trabalhos atuais que lidam com as diversas informações de cunho militar procuram fundi-las em diversos níveis de granularidade, para diversos tipos de fins, desde o refinamento e precisão do processamento de sinais até decisões de comando de alto nível [57, 58, 59]. Na literatura, o raciocínio em cima de um contexto militar por meio da combinação de diversos dados de diferentes fontes é usualmente chamado de fusão de dados.

Os autores Carvalho *et al.* [44] explicam que o processo de fusão de dados é largamente estudado e que inclui diferentes técnicas e métodos, cujo planejamento é estabelecido de acordo com um modelo chamado JDL. Segundo estes autores, o nível 0 do modelo JDL estabelece a conversão, o filtro e o gerenciamento de dados de modo que se possa fornecer informação o mais rápido possível. Já no nível 1, o objetivo principal é prover uma estimativa de posição, velocidade e aceleração de alvos, bem como correlacioná-los. Por fim, os níveis 2 e 3 estão preocupados em realizar uma avaliação estratégica alinhada com a capacidade de detecção e prevenção de possíveis ameaças.

Em um contexto militar naval (figura 6.1), a detecção e a monitoração das características cinemáticas de uma entidade qualquer em um determinado espaço de monitoração é chamado de “acompanhamento”. Um acompanhamento carrega consigo informações detectadas por radares, tal qual a velocidade, a aceleração, a posição atual e a posição futura estimada de possíveis alvos. Em um contexto militar naval, são exemplos de entidades sujeitas a acompanhamentos: navios, aviões, helicópteros, submarinos, etc. (figura 6.1).

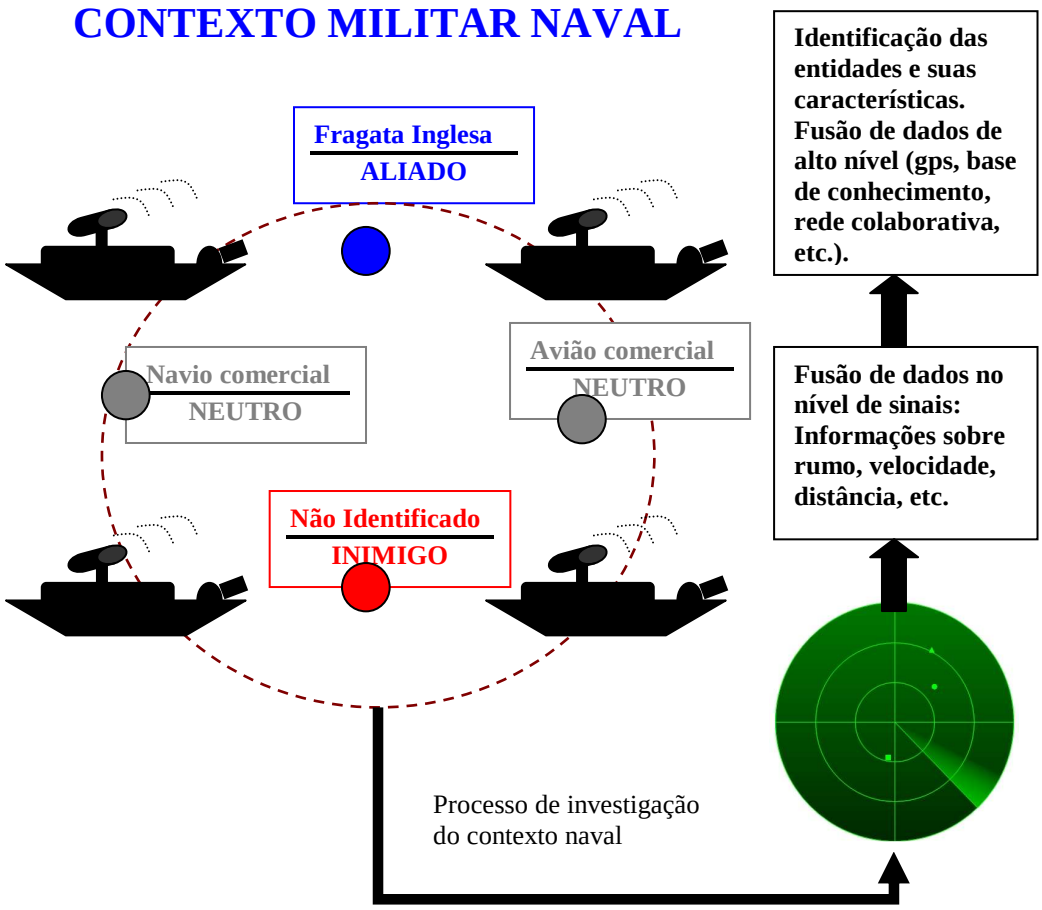

Figura 6.1: contexto militar naval

Fundir dados oriundos de diversos sensores é um dos principais requisitos do Terminal Tático Inteligente (TTI), *software* em desenvolvimento pelo Instituto de Pesquisas da Marinha (IPqM). O RT-MLR foi pensado e desenvolvido para ser aplicado ao contexto naval, de forma a ter uso imediato nesse tipo de contexto nos níveis do modelo JDL.

O IPqM já possui um *software* chamado Extrator Radar (EXR) que provê a fusão de dados no nível 0, o nível dos sensores. O EXR adota uma solução amplamente aceita para este procedimento no nível 0, o filtro de Kalman [57], que tem por característica uma avaliação probabilística de previsão. Assim, torna-se desnecessário realizar testes neste nível, pois já existe uma solução amplamente testada e aceita.

Na verdade, os níveis do modelo JDL não importam conceitualmente para os testes realizados, já que não fazem diferença nas implementações nem fazem diferença na avaliação dos resultados. Os testes serão efetivamente apresentados visando à correlação e classificação de acompanhamentos em um cenário tático naval.

6.2 Correlação e classificação de acompanhamentos

Os navios de guerra usualmente possuem diversos radares, cada qual com uma posição particular, diferente alcance e propósito específico. Em contextos militares, uma das grandes necessidades está relacionada à descoberta de similaridades entre diferentes acompanhamentos, uma vez que os diferentes radares presentes no navio podem eventualmente se referir a uma mesma entidade. Aliado a isso, existe a possibilidade de redundância de informações entre os próprios navios da frota, já que estes realizam intercâmbio de informações.

Em fusão de dados, o procedimento cujo objetivo é saber se dois ou mais acompanhamentos se referem a uma mesma entidade é normalmente chamado de correlação [57]. Os sistemas militares que realizam fusão de dados prioritariamente devem realizar uma investigação do contexto, no intuito de correlacionar dois ou mais acompanhamentos, provendo, assim, a diminuição da redundância de informações, o aumento da legibilidade do contexto, e o suporte à decisão de forma mais segura e precisa (figura 6.3).

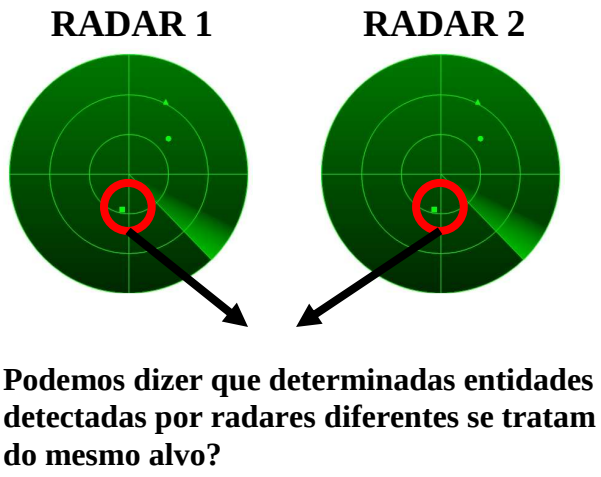

Figura 6.2: o problema da correlação em um contexto militar naval

Posteriormente à correlação de acompanhamentos, permanece ainda a questão relacionada à interpretação dos acompanhamentos por parte dos tripulantes dos navios.

Nos atuais sistemas militares navais brasileiros as informações de alto nível dos acompanhamentos estão intrinsecamente ligadas à capacidade de interpretação do operador do sistema, que o faz com base em sua experiência e com base na combinação de informações de diferentes sensores e radares. Tais interpretações promovem a

classificação dos acompanhamentos detectados, bem como a sua avaliação comportamental, que podem indicar ações ou contramedidas necessárias como parte de suas ações de combate e defesa.

Reconhecer comportamentos e classificar acompanhamentos são os princípios básicos dos níveis 1, 2 e 3 do modelo JDL de fusão de dados.

Segundo um especialista da Marinha do Brasil, os principais fatos do contexto que servem como base para correlação e classificação de entidades residem nas características cinemáticas dos acompanhamentos. Portanto, para este especialista, uma fusão de dados eficiente envolve a investigação dos contextos temporal e espacial dos acompanhamentos de forma contínua, pois, em razão da sua constante movimentação, estes acompanhamentos têm seus contextos atualizados em função do tempo e espaço continuamente.

Nesse cenário, a transformação do conhecimento tácito do operador para o explícito torna-se não só desejável, mas sim imprescindível, à medida que as decisões se dão em razão de um raciocínio aproximado do contexto naval efetuado pelo operador. Como mencionado anteriormente, expressões e regras formais são usualmente a prioridade de escolha para representação formal de conhecimento [13].

6.3 Ambiente de simulação

A capacidade do RT-MLR de apoiar a investigação de um contexto militar naval foi avaliada por meio de um ambiente de simulação, capaz de auxiliar a correlação e classificação dos acompanhamentos. Este ambiente conta com aplicativos e equipamentos pertencentes ao Instituto de Pesquisas da Marinha. Neste ambiente, todos os dados cinemáticos dos acompanhamentos que foram artificialmente introduzidos são extratos de exercícios efetuados pela Marinha do Brasil que, por questões de sigilo, não permitiu a identificação precisa dos navios, datas e contexto de aplicação destes exercícios.

6.3.1 Componentes

O ambiente de simulação criado é composto por três componentes: um Visualizador Radar (VR) para simulação de medidas e interface de visualização do contexto, um Extrator Radar (EXR) para criação de acompanhamentos baseados nas

medidas simuladas dos alvos pelo VR e um módulo de fusão de dados (MFD), desenvolvido com o apoio do RT-MLR.

O Visualizador Radar permite a simulação de dois radares diferentes, um radar DECCA modelo 1228 e um radar Thompson modelo DRBV 15 de superfície – exemplares de radares encontrados em navios da frota da Marinha do Brasil. O VR permite, além da inserção artificial de medidas, a apresentação do resultado das investigações do contexto (fusões de dados), as correlações e classificações dos acompanhamentos (figura 6.3).

Figura 6.3: Visualizador Radar

Os alvos simulados pelo VR são enviados para o Extrator Radar (EXR) em termos de posição e velocidade em nós a cada 1 segundo, que por sua vez, efetua um monitoramento referente a estas simulações de medida (alvos). Tais monitorações dos alvos são relativas aos acompanhamentos e correspondem ao nível 0 de fusão de dados do modelo JDL. Com isso, para cada alvo, um acompanhamento é criado pelo EXR, que é retornado para o VR com informações cinemáticas agregadas, como o rumo, a aceleração, a velocidade e a posição do alvo simulado, que são efetuados pelo algoritmo "Filtro de Kalman". Tais medidas simuladas dos alvos podem sofrer variação tal qual

ocorre em exercícios reais, tornando possível, de forma separada para cada radar, a adição de ruídos de processo[1] e de medida[2] por meio de uma curva gaussiana.

Por fim, o módulo de fusão de dados (MFD) desenvolvido tem por finalidade capturar, correlacionar e classificar os acompanhamentos efetuados pelo EXR, bem como enviá-los de volta para o VR a fim de apresentar o resultado da fusão de dados.

A figura 6.4 apresenta a interação entre os três módulos presentes no ambiente de simulação.

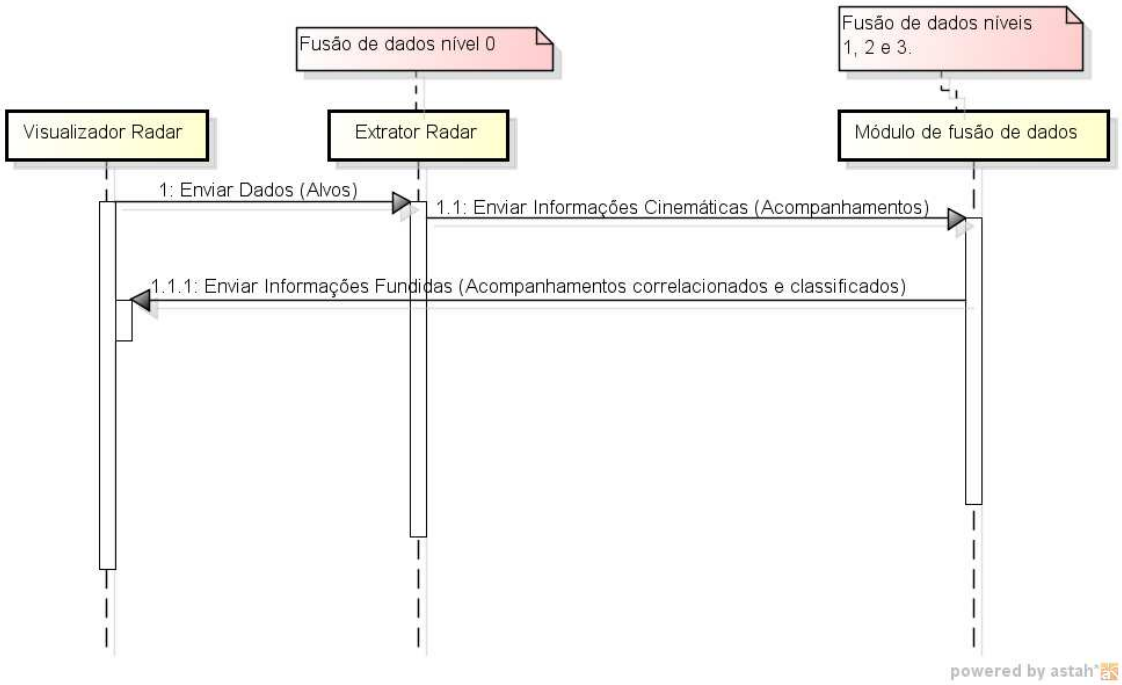

Figura 6.4: ambiente de simulação do contexto militar naval

Efetivamente aplicado ao contexto naval, o RT-MLR apoiará as fusões de dados de nível 1, 2 e 3 no TTI, por meio de correlações e classificações que são realizadas obedecendo a critérios definidos por meio de regras escritas por um especialista do domínio e avaliadas sempre que o contexto se alterar. O evento que demarca a alteração do contexto está relacionado a estas três situações:

1) Todas as vezes que um novo acompanhamento é criado, uma regra de correlação é instanciada e associada para o novo acompanhamento e para outro acompanhamento

[1] O erro de processo se refere às interferências obtidas durantes a emissão e recepção de sinais, como por exemplo, o balanço do mar.
[2] O erro de medida se refere à precisão e à acurácia do instrumento.

existente na lista de acompanhamentos do MFD. Com *N* acompanhamentos já existentes, mais *N* regras serão instanciadas.

2) Todas as vezes que um novo acompanhamento é criado, uma regra de classificação é associada a este acompanhamento.

3) Todas as vezes que um acompanhamento é destruído, todas as regras associadas ao acompanhamento são destruídas.

4) Sempre que um fato se altera no contexto, as regras associadas ao acompanhamento são investigadas no MFD.

6.3.2 Modelagem

Visando contemplar a modelagem do contexto naval, uma classe chamada *Track* (figura 6.5) foi criada para representar os acompanhamentos recebidos do EXR, cujos objetos são instanciados ou destruídos conforme os alvos são simulados e acompanhados pelo módulo VR e pelo módulo EXR. Além disso, no MFD, uma classe chamada *Route* representa possíveis rotas simuladas por acompanhamentos. A classe *Route* é especialmente utilizada no procedimento de classificação de acompanhamentos.

A classe *Track* possui os seguintes atributos: número identificador único (atributo *nt*), velocidade em nós (atributo *speed*), posição nas coordenadas X e Y (atributos *positionX e positionY*), tamanho (atributo *size*) e radar de origem (atributo *source*). Os atributos *speed*, *positionX*, *positionY* e *size* são do tipo *NumericalData*, tipo estabelecido pelo *framework* para definição de tipos observáveis de precisão dupla. Com isso, todas as vezes que ou a velocidade, ou a posição ou mesmo o entendimento sobre o tamanho dos alvos providos pelo EXR for modificado, as regras criadas para correlação e classificação serão investigadas.

Além das operações tradicionais de encapsulamento *get* e *set*, a classe *Track* possui operações pertinentes ao desenvolvimento das regras, que retornam informações necessárias à tomada de decisão, correlação e classificação.

A operação *getCourse* retorna o rumo do alvo baseado em sua velocidade e posição. Já a operação *getEstimatedPosition* retorna a posição estimada do alvo em um tempo futuro, baseada também na atual posição e na velocidade, enquanto que a operação *getDistance* retorna a distância do alvo para o navio. As operações de diferença (*getDifferencePosition*, *getDifferenceSpeed*, *getDifferenceSize*, *getDifferenceCourse*) retornam, como os próprios nomes dos métodos sugerem, a diferença de posição, velocidade, rumo e tamanho entre dois acompanhamentos.

As operações *getIff* e *setIff* retornam e alteram, respectivamente, os dados simulados do radar IFF (*Identification of a Friend or Foo*). O IFF é um radar presente em navios de diversos propósitos, cujo objetivo é difundir, em *broadcast*, um código que identifica o navio, avião, etc. no cenário tático militar, sendo classificado, conforme a interpretação desse código e a conveniência de quem o interpreta, como "amigo" ("*friend*") – sinal 0, "inimigo" ("*foo*") – sinal 1, "neutro" ("*neutral*") – sinal 2 ou "desconhecido" ("*unknown*") – sinal 3.

As operações *setHostility*, *getHostility*, *setLabel*, *setLabelBelief* e *getLabelBelief*, inserem e obtêm, respectivamente, a hostilidade e os rótulos atribuídos durante o processo de classificação do acompanhamento.

Por fim, a classe *Route* tem por objetivo principal contemplar a modelagem de rotas que são utilizadas no procedimento de investigação do contexto. A classe *Route* possui, além dos atributos referentes aos vetores de posições, duas operações para cálculo de distância e diferença de rumos (figura 6.15) – *getDistanceToRoute e getDifferenceCourseToRoute* .

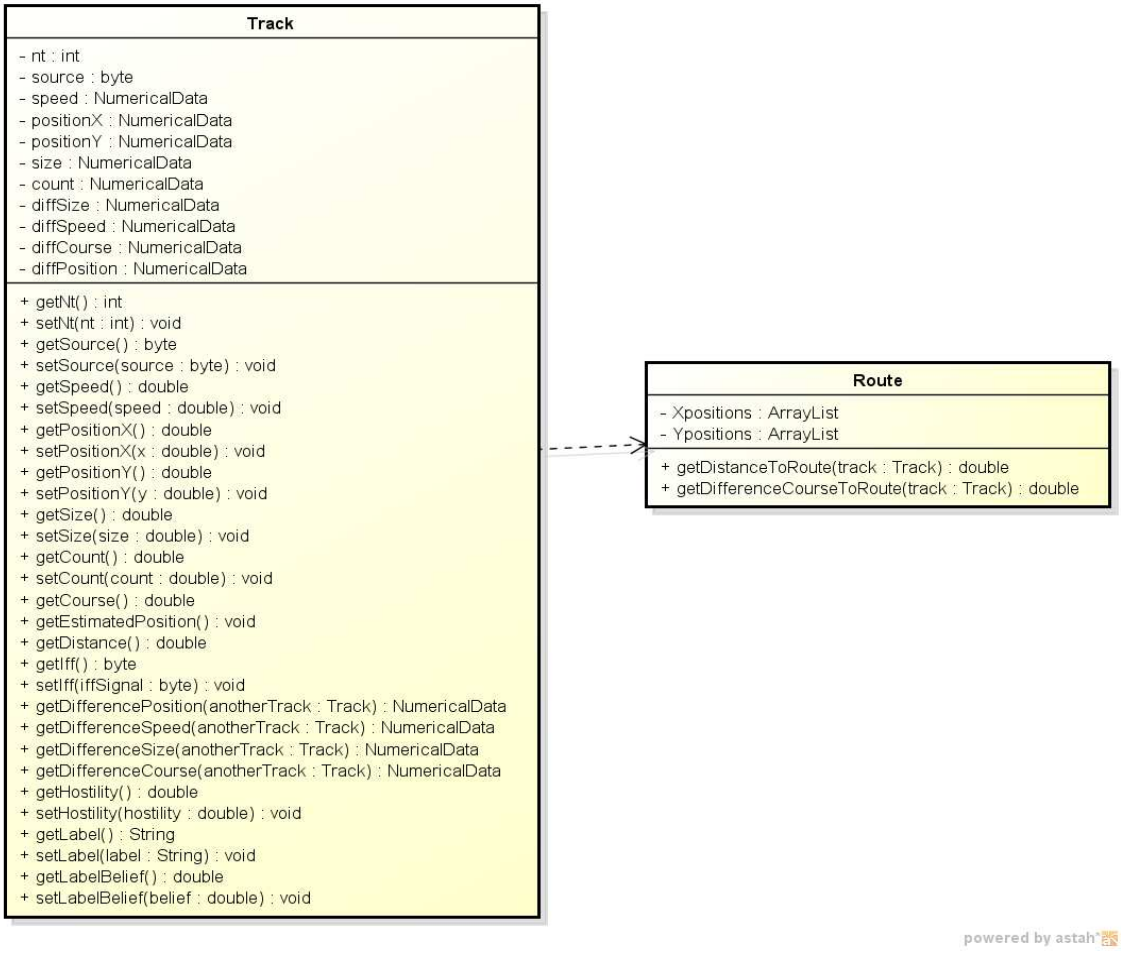

Figura 6.5: modelagem dos dados de acompanhamento no módulo de fusão de dados

A seguir, a descrição das regras para correlação e classificação de acompanhamentos, bem como os conjuntos *fuzzy*, os atributos dos objetos e os casos de teste envolvidos no processo de aplicação do RT-MLR no contexto militar naval.

6.4 Regras para investigação do contexto militar naval

6.4.1 Regra para correlação de acompanhamentos

O especialista do domínio optou por definir a regra para correlação como uma regra de natureza de primeira ordem, mas com algumas regras *fuzzy* que dessem suporte à obtenção do tamanho dos acompanhamentos. Mais do que isso, a regra de correlação é avaliada com base no contexto temporal.

Segundo o especialista, a decisão de correlação só pode ser considerada caso haja uma validação contínua ao longo de 5 segundos. Se por acaso, em algum momento, para este determinado intervalo de tempo, a condição para correlação não for verdadeira, então a correlação não pode ser considerada. Assim, como os dados dos radares são enviados a cada 1 segundo, então a regra deve contemplar uma condição cuja sentença seja verdadeira por 5 vezes consecutivas.

A correlação é condicionada à medida de semelhanças entre o rumo, a velocidade, a posição e o tamanho de dois acompanhamentos. Como no ambiente de simulação não há uma garantia de atualização homogênea para dois acompanhamentos distintos, a regra de correlação se utiliza da operação de estimativa de cinemática futura presente na classe *Track*. A medida de correlação então é obtida pela diferença (Δ) entre os atributos dos dois objetos *tracks* confrontada com um limiar de tolerância definido.

A relação entre o limiar e a diferença absoluta presente entre os atributos dos dois acompanhamentos se refere à confiabilidade das medidas efetuadas pelos radares. Para a velocidade, a posição e o rumo, um limiar mais baixo indica um modelo de correlação mais exigente e, com um limiar mais alto, o modelo de correlação será menos exigente. De forma contrária para um tamanho de acompanhamento, um limiar mais alto indica um modelo de correlação mais exigente e, com um limiar mais baixo, o modelo de correlação será menos exigente. Em trabalhos de outros autores [60], este limiar reflete diretamente a precisão dos sensores.

Os limiares podem ser alterados pelas ferramentas presentes no MFD.

Dentre os atributos utilizados dos objetos *tracks* para construção da regra, a única exceção em relação à manipulação dos dados fica por conta do tamanho dos acompanhamentos. O tamanho de um acompanhamento não é fornecido diretamente pelo VR nem pelo EXR, mas inseridos manualmente por uma interface de alteração. Na verdade, os alvos simulados possuem como informação a quantidade de vezes que o alvo foi iluminado, cujo valor permite obter o tamanho do alvo baseado na distância que se encontra o alvo em relação ao navio (figura 6.6).

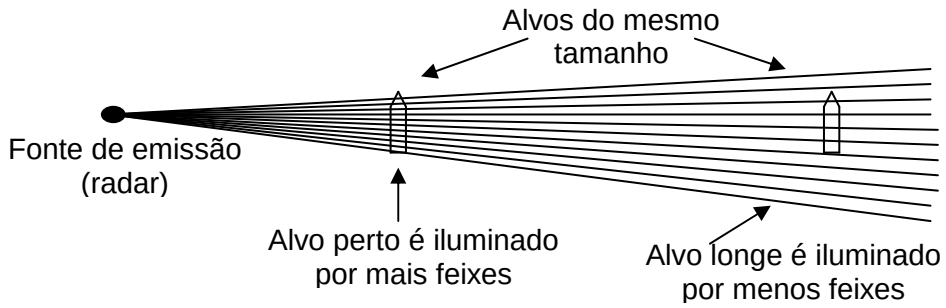

Figura 6.6: raciocínio para obtenção do tamanho do alvo. Adaptado de Carvalho et al. [44].

Nesse cenário, para se obter o tamanho dos alvos em metros, o especialista do domínio optou por definir três variáveis *fuzzy* e nove regras. A idéia geral é a de que quanto maior forem as vezes que um alvo tenha sido "iluminado" e maior seja a distância do radar para o alvo (posição do navio – a posição do alvo), maior será o tamanho estimado deste alvo. A figura 6.7 apresenta as variáveis *fuzzy* utilizadas nas premissas e nas conclusões das regras. Já a tabela 6.1 apresenta as regras para obtenção do tamanho do alvo em metros.

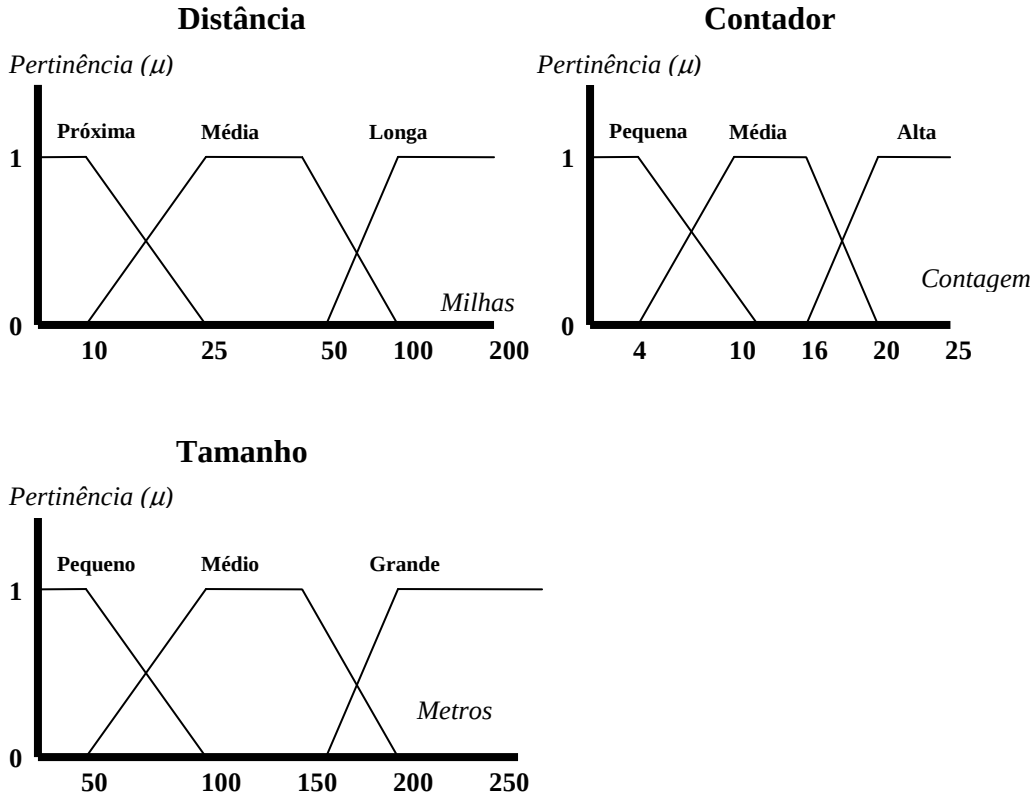

Figura 6.7: Conjuntos fuzzy para obtenção do tamanho do alvo. Adaptado de Rangel et al. [43].

Número da regra (#)	Premissa	Conclusão
1	Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é próxima	Então o tamanho do alvo é pequeno
2	Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é média	Então o tamanho do alvo é pequeno
3	Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é longa	Então o tamanho do alvo é médio
4	Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é próxima	Então o tamanho do alvo é médio
5	Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é média	Então o tamanho do alvo é médio
6	Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é longa	Então o tamanho do alvo é grande
7	Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é próxima	Então o tamanho do alvo é médio
8	Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é média	Então o tamanho do alvo é grande
9	Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é longa	Então o tamanho do alvo é grande

Tabela 6.1: regras fuzzy para obtenção do tamanho do alvo.

Seja, portanto, um acompanhamento denominado A1 e um acompanhamento denominado A2 para correlação. A regra de correlação é estabelecida em termos de:

Número da regra (#)	Premissa	Conclusão
1	Se nos últimos 5 segundos, a seguinte condição for verdadeira: Se a diferença entre as posições de A1 e A2 for menor ou igual ao limiar E a diferença entre as velocidades de A1 e A2 for menor ou igual ao limiar E a diferença entre os rumos de A1 e A2 for menor ou igual ao limiar E a diferença entre os tamanhos de A1 e A2 for maior ou igual ao limiar	Então os acompanhamentos tratam do mesmo alvo

Tabela 6.2: regra temporal para correlação.

Esta regra é equivalente a:

$|P_{A1} - P_{A2}| \leq \lambda_p$ E $|V_{A1} - V_{A2}| \leq \lambda_v$ E $|R_{A1} - R_{A2}| \leq \lambda_r$ E $|T_{A1} - T_{A2}| \geq \lambda_t \rightarrow$ A1 = A2,

onde:

$P_{Ax:}$ posição do acompanhamento X no momento da correlação.

$V_{Ax:}$ velocidade do acompanhamento X no momento da correlação.

$R_{Ax:}$ rumo do acompanhamento X no momento da correlação.

$T_{Ax:}$ tamanho do acompanhamento X no momento da correlação.

λ_y: limiar de validação Y [posição (p), velocidade (v), rumo (r), tamanho (t)].

As definições da regra de correlação, bem como as definições da regra para obtenção do tamanho do alvo estão encapsuladas em classes e arquivos definidos com este propósito. A classe *TrackSize* e seu respectivo arquivo (apêndice I) contêm as três variáveis *fuzzy* e as nove regras *fuzzy* descritas na tabela 6.1. Já a classe *TrackCorrelation* e seu respectivo arquivo (apêndice II) contêm a regra de correlação para dois acompanhamentos.

Em referência à avaliação da capacidade do RT-MLR de correlacionar alvos, o especialista do domínio estabeleceu três casos de testes, variando a qualidade da estimação dos dados cinemáticos dos acompanhamentos providos pelo EXR. Estes mesmos testes foram apresentados recentemente nos trabalhos dos autores Rangel e*t al.* [43] e também na publicação dos autores Carvalho *et al.*[44].

O primeiro caso de teste (figura 6.8) retrata um contexto onde dois navios (Acompanhamento 1 e Acompanhamento 2) estão separados inicialmente por uma distância de 5 milhas (T_0). Estes dois navios se movem em uma velocidade constante de

20 nós, com rumos de 210° para o acompanhamento A1 e 135° para o acompanhamento A2. O movimento uniforme dos dois navios resulta em uma sobreposição de detecção por parte dos radares (T_1), conforme a progressiva diminuição de distância entre estes acompanhamentos. Posteriormente à sobreposição de detecção, estes acompanhamentos se afastam e prosseguem se deslocando em seus rumos, até que não estejam mais sob o alcance dos radares (T_2).

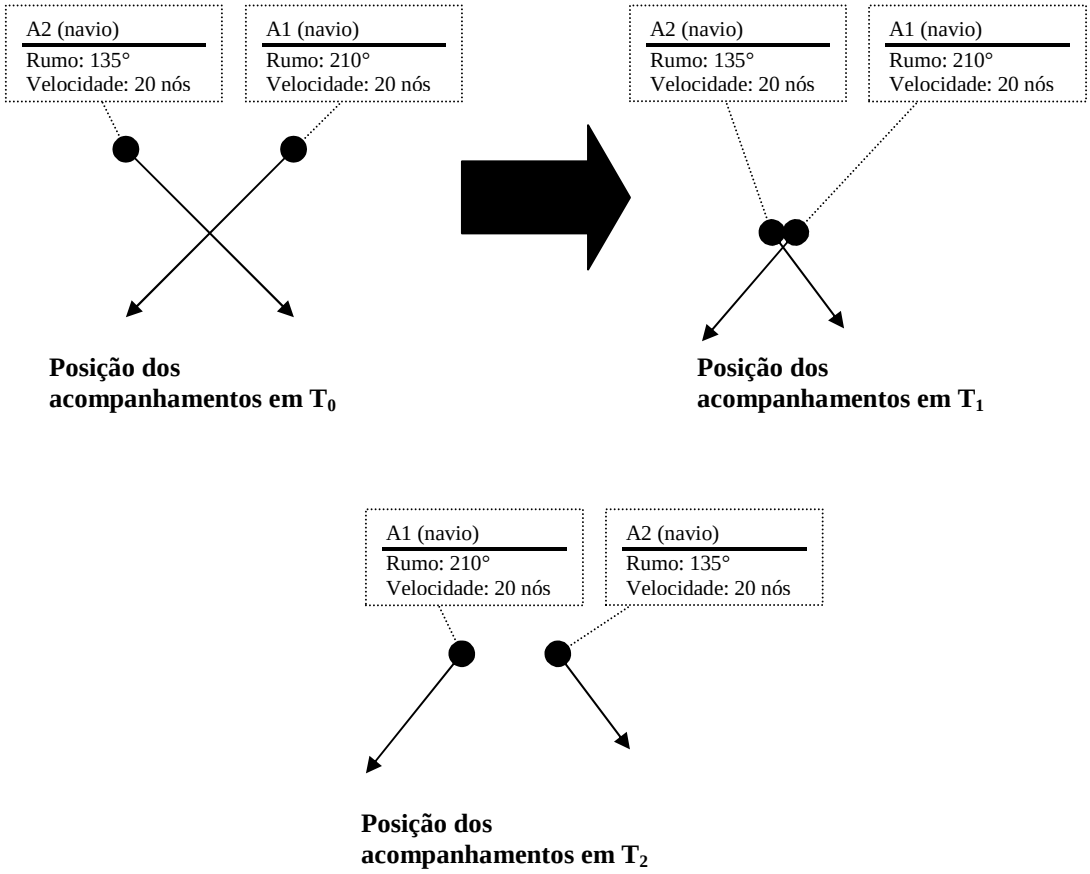

Figura 6.8: caso de teste 1 do problema de correlação.

O segundo caso de teste (figura 6.9) se refere a um contexto onde um pequeno navio e um helicóptero são detectados pelos radares (T_0) com o mesmo rumo (0°), mas com velocidades diferentes (50 nós para o helicóptero e 10 nós para o navio) a uma distância de 3 milhas. O helicóptero, por possuir maior velocidade, ultrapassa o navio em um momento futuro. O helicóptero diminui paulatinamente a distância entre a sua posição e a posição do pequeno navio, sobrepondo, momentaneamente, as imagens de detecção nos radares (T_1). Posteriormente à sobreposição, os acompanhamentos detectados se afastam completamente, prosseguindo em seus rumos até que os radares não consigam mais detectá-los (T_2).

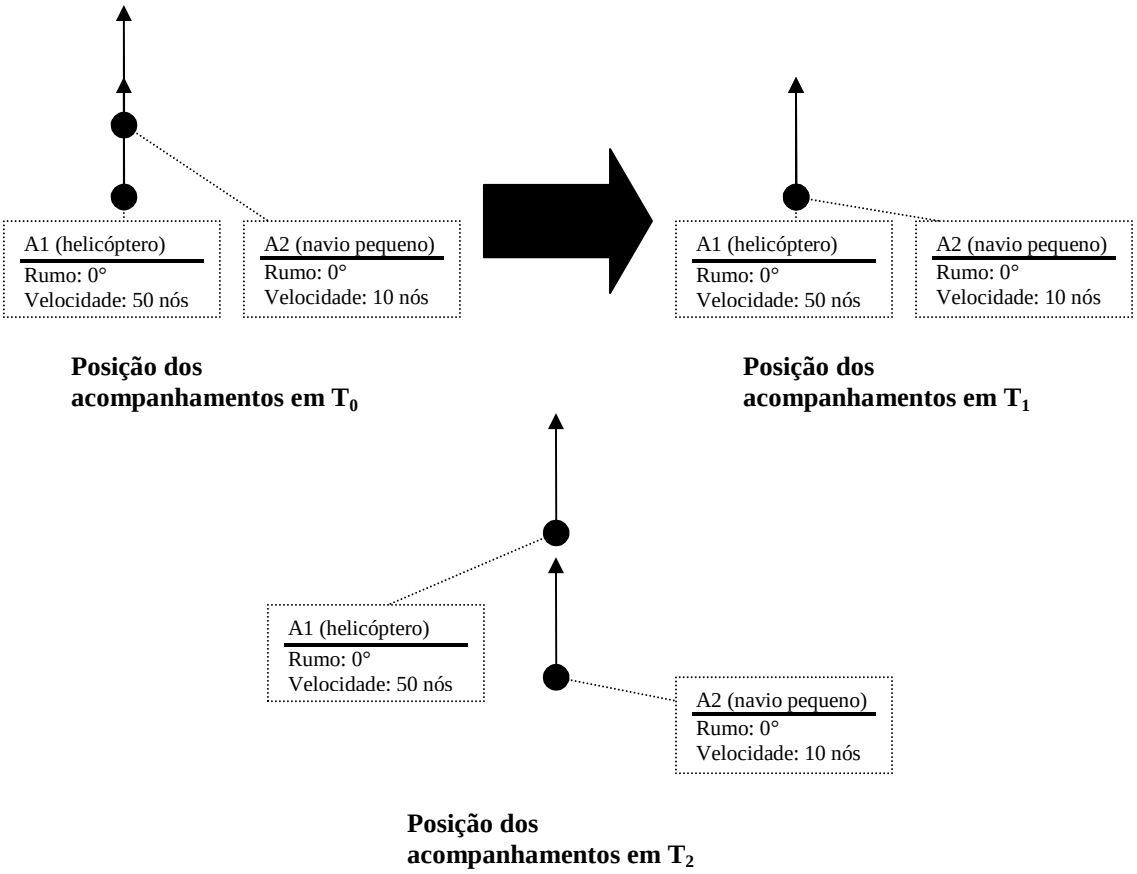

Figura 6.9: caso de teste 2 do problema de correlação.

Por fim, o terceiro caso de teste (figura 6.10) se refere a dois navios em situações de manobra, alterando seu rumo em 100° a cada minuto. Como os dois casos de testes anteriores, os dois navios terão suas distâncias diminuídas conforme o tempo passa e seus rumos convergem (T_0). Devido ao procedimento de manobra e à proximidade dos acompanhamentos, os dois navios se sobrepõem durante a detecção do radar (T_1). Por fim, os dois navios ficam alinhados e com velocidade e rumo constante em 180° (T_2).

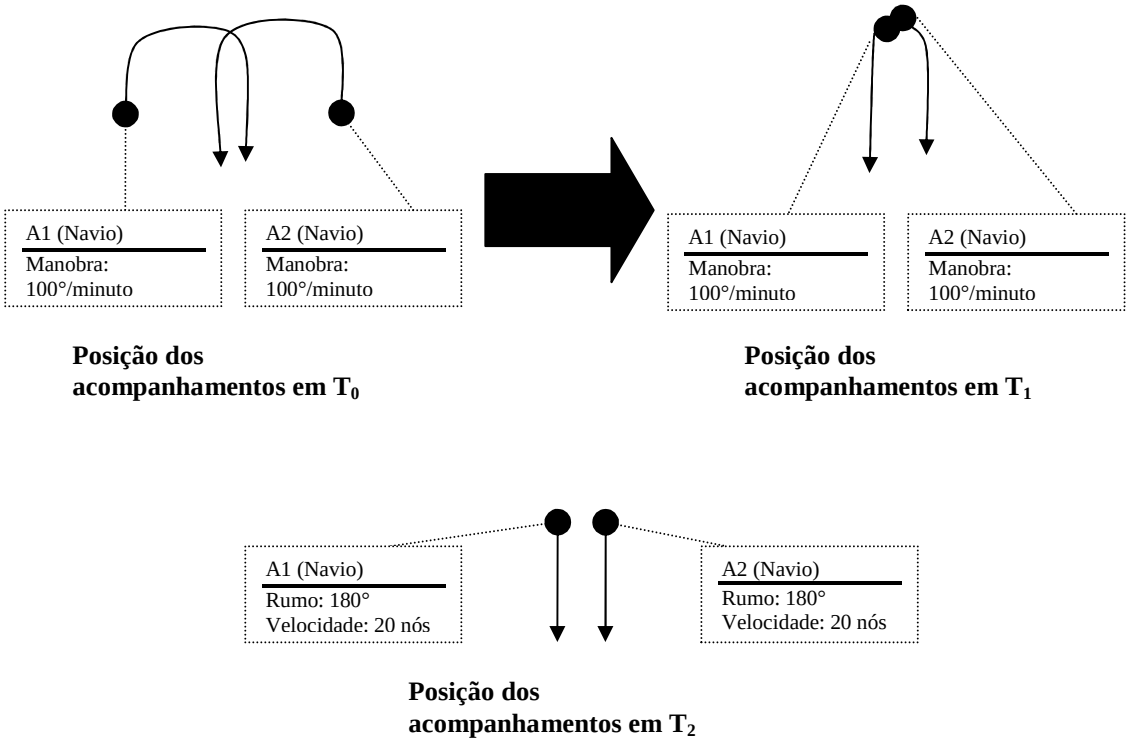

Figura 6.10: caso de teste 3 do problema de correlação.

Para cada caso de teste, duas baterias foram executadas, cada uma com um erro estimado de filtragem. A primeira bateria tem erro estimado de filtragem em 5 metros (s=5) e a segunda bateria tem erro estimado de filtragem em 20 metros (s=20). Além disso, para cada caso de teste, dois acompanhamentos são criados para cada alvo, já que existem dois radares (DECCA e DRBV-15) no ambiente simulado. Então, 2 acompanhamentos serão gerados para cada radar, resultando em um total de quatro acompanhamentos (4 objetos *Track*).

A regra de associação estabelecida pelo especialista do domínio tem como objetivo associar os dois acompanhamentos que se referem à mesma entidade e que vieram de radares diferentes, bem como não associar os dois acompanhamentos que não se referem à mesma entidade e são providos por radares diferentes (figura 6.11).

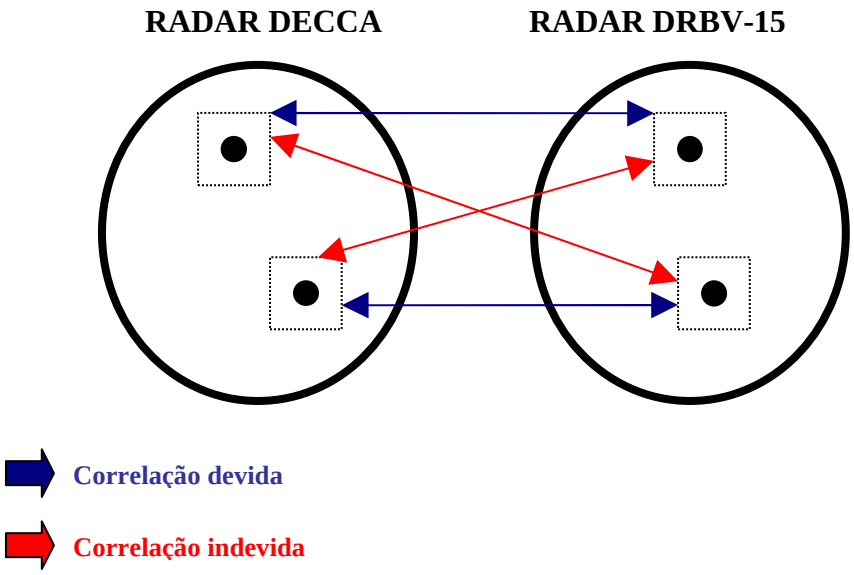

Figura 6.11: comportamento esperado para o problema da correlação.

6.4.2 Regra para classificação de acompanhamentos

A idéia geral da classificação é atribuir um rótulo ao acompanhamento, bem como definir um grau de hostilidade.

O cálculo da hostilidade é definido com base na velocidade de um acompanhamento, o tamanho do alvo e a reposta de um radar IFF.

O tamanho do acompanhamento é obtido e utilizado de forma igual à solução proposta para o problema da correlação. Entretanto, de forma diferenciada, a velocidade do acompanhamento é expressa em uma variável *fuzzy* (figura 6.12), pois o objetivo neste caso é avaliar a velocidade em termos gerais. Ademais, o problema da classificação exige um sinal de um radar especial, o IFF, que neste caso é inserido como fato para cada acompanhamento no sistema tático. Assim, como não há a simulação deste sinal, os dados a respeito de identificação são inseridos manualmente no módulo de fusão.

O especialista do domínio optou por definir 4 regras de natureza *fuzzy* (apêndice III) que dessem suporte à avaliação da hostilidade dos acompanhamentos no contexto do usuário no cenário tático (tabela 6.3). A hostilidade de um acompanhamento é então definida em termos de variáveis *fuzzy* de hostilidade, descritas na figura 6.13 como baixa hostilidade, média hostilidade, alta hostilidade e hostilidade crítica.

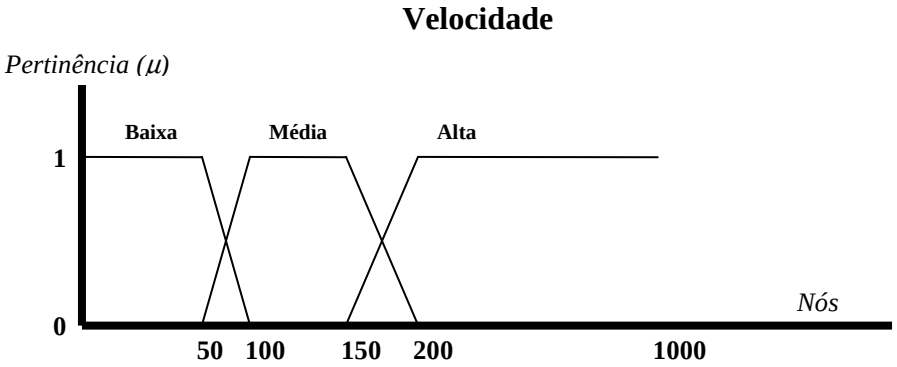

Figura 6.12: variável fuzzy de entrada "velocidade".

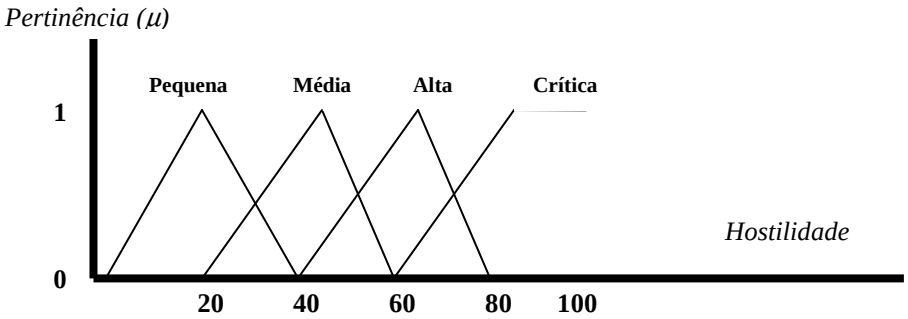

Figura 6.13: variável fuzzy de saída "hostilidade".

Número da regra (#)	Premissa	Conclusão
1	Se o acompanhamento é "desconhecido" OU "amigo" OU "neutro" E o tamanho do acompanhamento é grande E a velocidade do acompanhamento é pequena	Então a hostilidade é pequena
2	Se o acompanhamento é "desconhecido" E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é média	Então a hostilidade é média
3	Se o acompanhamento é "desconhecido" E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é alta	Então a hostilidade é alta
4	Se o acompanhamento é "inimigo" ou o acompanhamento é "desconhecido" E a velocidade do acompanhamento é alta	Então a hostilidade é crítica

Tabela 6.3: regras fuzzy para obtenção da hostilidade

As regras de classificação dos acompanhamentos são também definidas com base nos dados cinemáticos dos acompanhamentos, além dos tamanhos dos alvos e o graus de hostilidade obtidos pelo conjunto das 4 regras *fuzzy* que acabaram de ser

descritas. De forma semelhante, a velocidade do acompanhamento é descrita em termos do conjunto *fuzzy* "Velocidade" (figura 6.12), bem como o tamanho do alvo é descrito em termos do conjunto *fuzzy* "Tamanho" (figura 6.7). O único termo diferente que é utilizado nas regras de classificação é o que se refere à proximidade de um acompanhamento a uma rota comercial. Segundo o especialista do domínio, o comportamento de um acompanhamento em relação a uma rota comercial pode indicar a natureza e o tipo de acompanhamento ao longo de uma avaliação temporal do contexto. Assim, dada uma rota comercial introduzida no ambiente de simulação e duas variáveis *fuzzy* chamadas "Distância para Rota" e "Diferença de Rumo para Rota" (figura 6.14), um acompanhamento é classificado não só em termos de seus dados físicos e cinemáticos, mas também em termos de sua proximidade e semelhança de rumos em relação a uma determinada rota (figura 6.15).

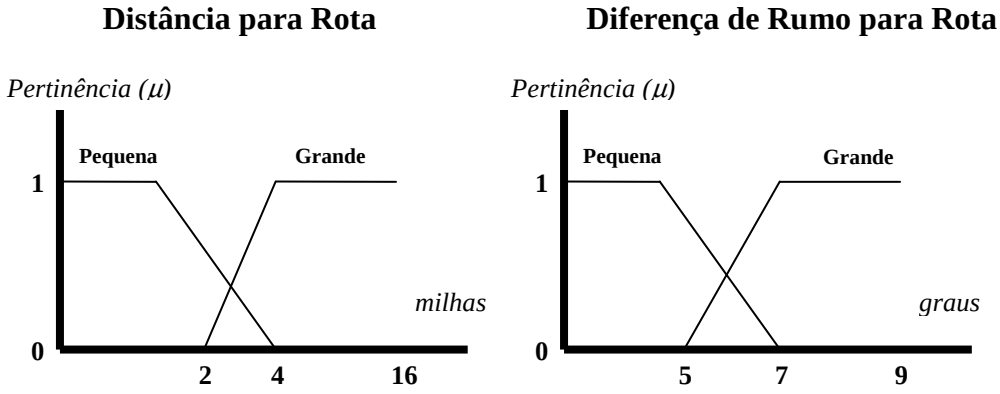

Figura 6.14: variáveis fuzzy de entrada para proximidade e semelhança de rumo com uma rota.

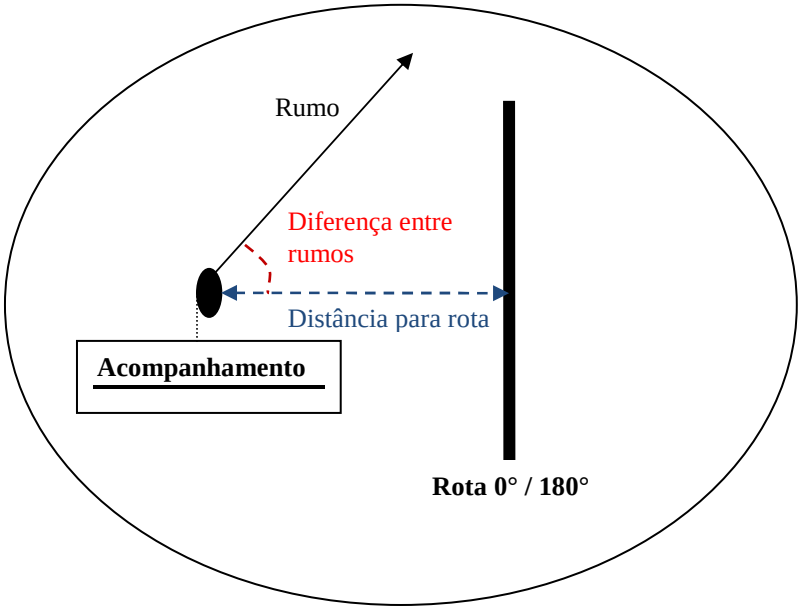

Figura 6.15: Comportamento de um acompanhamento em relação a uma rota.

O número de entidade presentes em um cenário tático naval é bem grande e, como o trabalho apenas trata o contexto naval como um estudo de caso, três casos de teste foram escolhidos pelo especialista do domínio para o problema da classificação (figura 6.16). Ainda em razão do vasto número de entidades presentes no contexto naval, torna-se razoável atribuir graus de pertinência para classificação, e não decidi-las de uma forma mutuamente excludente.

Nesse sentido, o especialista do domínio definiu uma regra *fuzzy* de exemplo para cada caso de teste. Posterior à avaliação da hostilidade, a regra *fuzzy* de classificação atribui um rótulo a um acompanhamento, bem como um grau de pertinência a esta avaliação, variando de 0 a 1, isto é, 0 a 100 %. Como há uma regra para cada caso de teste, o resultado da *defuzzyficação* é exatamente igual à pertinência obtida na regra.

Contudo, como o número de regras para classificação de um mesmo acompanhamento pode variar, torna-se necessário, portanto, uma base de regras *fuzzy* para cada tipo de classificação, permitindo que o especialista do domínio defina mais regras para cada caso de teste futuramente.

O primeiro caso de teste se refere a um acompanhamento com baixo grau de hostilidade, cuja classificação se refere a um cargueiro a 5 nós de velocidade navegando adequadamente em uma rota comercial. O segundo caso de teste se refere a um acompanhamento com médio grau de hostilidade, cuja classificação se refere a um navio comercial a 25 nós de velocidade. O terceiro caso de teste se refere a um acompanhamento com alto grau de hostilidade, cuja classificação se refere a um míssil a 300 nós de velocidade.

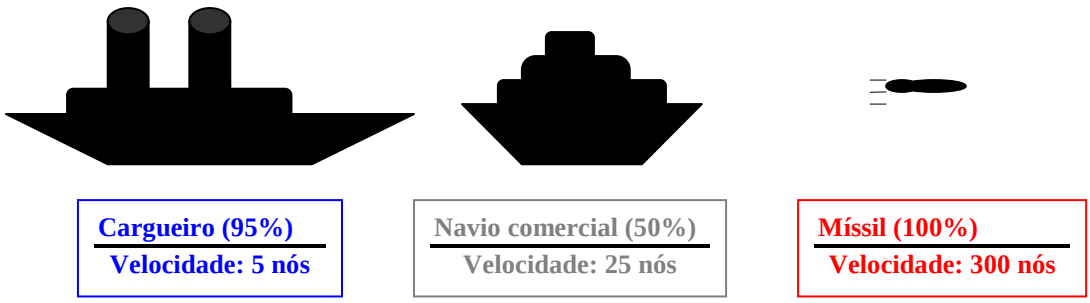

Figura 6.16: Casos de teste para classificação

As regras para estes três casos de teste estão descritos na tabela 6.4. A implementação destas regras pode ser encontrada no apêndice IV.

Número da regra (#)	Premissa	Conclusão
1	Se o acompanhamento não é “inimigo” E a distância do acompanhamento para rota é pequena E a diferença entre o rumo do acompanhamento e a rota for pequena E o tamanho do acompanhamento é grande E a velocidade do acompanhamento é baixa	Então o acompanhamento é um cargueiro
2	Se o acompanhamento é “neutro” E a hostilidade do acompanhamento é baixa E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é média	Então o acompanhamento é um navio comercial
3	Se o acompanhamento é “desconhecido” E a hostilidade do do acompanhamento é crítica E o tamanho do acompanhamento é pequeno E a velocidade do acompanhamento é alta	Então o acompanhamento é um míssil

Tabela 6.4: Regras para classificação de acompanhamentos.

6.5 Resultados

6.5.1 Avaliação objetiva do RT-MLR para correlação de acompanhamentos na investigação de contexto militar naval

Uma maneira amplamente aceita de avaliação de resultados e classificação se dá através de uma matriz de confusão [61], que tem por idéia principal apresentar medidas de qualidade relacionadas a classes obtidas. Uma matriz de confusão deve gerar, de forma qualitativa, a relação de acertos de classificação confrontada com as expectativas geradas em torno de um determinado resultado. No caso da correlação, as medidas da matriz de confusão devem prover informações que dêem uma noção da capacidade da regra em atender às exigências de decisão na correlação de acompanhamentos. A regra de correlação dever decidir por correlacionar aqueles acompanhamentos que sabidamente devem ser correlacionados e não correlacionar àqueles acompanhamentos que sabidamente não devem ser correlacionados.

Nesse cenário, visando apresentar os resultados obtidos com a aplicação do RT-MLR em um contexto militar naval, uma matriz de confusão foi modelada com base na hipótese de acerto na correlação dos acompanhamentos. Seja *n* o número de vezes que o contexto foi investigado, isto é, o número de vezes que a regra de correlação foi investigada. Seja, ainda, *h* a hipótese de correlação com h=1 (tabela 6.5):

	H=1	H=0
Os mesmos acompanhamentos	Verdadeiro positivo (Vp)	Falso negativo (Fn)
Acompanhamentos diferentes	Falso positivo (Fp)	Verdadeiro negativo (Vn)

Tabela 6.5: modelo de avaliação de correlação de acompanhamentos.

Baseada nesta modelagem, a avaliação é expressa (tabela 6.6) em 20 execuções, por meio da média dos seguintes critérios:

1. O erro em relação à decisão de correlação ("erro"). $Erro = \frac{F_p + F_n}{n}$;
2. A precisão do acerto na decisão de correlação ("precisão"). $\Pr ecisão = \frac{V_p + V_n}{n}$;
3. O percentual de confiabilidade para correlações que deveriam ser efetuadas ("confiabilidade positiva"). $ConfiabilidadePositiva = \frac{V_p}{V_p + F_p}$;
4. O percentual de confiabilidade para correlações que não deveriam ser efetuadas ("confiabilidade negativa"). $ConfiabilidadeNegativa = \frac{V_n}{V_n + F_n}$;
5. A medida de qualidade de escolha nos casos de teste ("cobertura"). $Cobertura = \frac{V_p + V_n}{n} ou \frac{F_p + F_n}{n}$;
6. O percentual de correlações efetuadas bem sucedidas em relação à quantidade de testes ("suporte"). $Suporte = \frac{V_n}{n}$;

S=5	Erro	Precisão	Confiabilidade Positiva	Confiabilidade Negativa	Suporte	Cobertura
Caso de Teste 1	0,01	0,99	1	0,99	0,49	0,49
Caso de Teste 2	0,02	0,98	1	0,96	0,48	0,48
Caso de Teste 3	0,01	0,99	1	0,99	0,50	0,50
S=20	**Erro**	**Precisão**	**Confiabilidade Positiva**	**Confiabilidade Negativa**	**Suporte**	**Cobertura**
Caso de Teste 1	0,04	0,96	0,99	0,92	0,46	0,46
Caso de Teste 2	0,04	0,96	1	0,97	0,46	0,46
Caso de Teste 3	0,02	0,98	1	0,96	0,48	0,48

Tabela 6.6: resultado da correlação de acompanhamentos

Os baixos valores de erro e os altos valores de precisão obtidos mostram que a regra de correlação adotada e implementada por meio do *framework* RT-MLR está correta. Destaca-se, ainda, o alto valor de confiabilidade positiva, uma vez que todos os acompanhamentos que deveriam ser correlacionados foram correlacionados de fato. De maneira geral, apesar de apresentar menor eficiência, a confiabilidade negativa apresenta também um bom resultado, uma vez que a maioria dos acompanhamentos que não foram correlacionados não deveriam de fato ter sido correlacionados.

O valor de cobertura obtido (por volta de 0,5), combinado com o alto valor de confiabilidade obtido, apresenta uma correta escolha dos casos dos testes escritos pelo especialista do domínio e a amostra de dados obtida com exercícios da Marinha Brasileira. Há, portanto, um balanceamento adequado entre os acompanhamentos correlacionados e os acompanhamentos não correlacionados.

6.5.2 Avaliação computacional utilizando o RT-MLR para correlação de acompanhamentos na investigação de contexto militar naval

No que tange à avaliação computacional, torna-se necessário lembrar que os objetos referentes aos acompanhamentos (*tracks*) são instanciados ou destruídos à medida que as demandas no ambiente de simulação ocorrem. Assim, uma regra de correlação é instanciada para comparação entre dois acompanhamentos de diferentes radares a cada vez que um novo acompanhamento é criado e a cada vez que um

acompanhamento é destruído. Como a associação é feita para dois acompanhamentos de diferentes radares, o módulo de fusão de dados o faz com base no byte de identificação, que é obtido por meio do método *getSource* da classe *Track*.

Seja, portanto, a relação de *N* acompanhamentos simulados do radar DECCA, *M* acompanhamentos simulados do radar DRBV-15 e 1 regra de correlação. Portanto, o número total de regras de correlação instanciadas é da ordem *N*x*M* objetos de regras de correlação. Assim, a complexidade algorítmica computacional, quer seja em relação ao consumo de memória, quer seja em relação ao consumo do processador, cresce linearmente ao número de acompanhamentos existentes apenas no melhor caso, no qual apenas o número de acompanhamentos de um radar cresce. Quando *N* é igual a *M*, então o pior caso é dado pela ordem O(N^2) ou O(M^2).

Já a associação da regra para obtenção do tamanho do alvo é feita para cada acompanhamento individualmente. Seja, portanto, o número de regras de obtenção do tamanho do alvo igual a 9. Seja, ainda, a relação de *N* acompanhamentos simulados do radar DECCA e *M* acompanhamentos simulados do radar DRBV-15. Então, o número total de regras para obtenção do tamanho do alvo instanciadas é da ordem 9x*N*x*M* objetos de regras (objetos *rule*) para obtenção do tamanho do alvo. A complexidade algorítmica computacional, quer seja em relação ao consumo de memória, quer seja em relação ao consumo do processador, cresce linearmente ao número de acompanhamentos existentes. Isto é, o pior caso para esta associação é da ordem O(*N*).

6.5.3 Avaliação objetiva do RT-MLR para classificação de acompanhamentos na investigação de contexto militar naval

A métrica de avaliação adotada no problema da correlação não pode ser adotada neste caso. Ao contrário da correlação, os resultados da investigação do contexto para classificação não resultam em uma decisão do tipo “verdadeiro” ou “falso”, isto é, classificar ou não classificar dois acompanhamentos. A regra de correlação, por possuir uma natureza lógica de primeira ordem, pode ser avaliada por meio da matriz de confusão adequadamente.

Por outro lado, a obtenção do grau de hostilidade é definida em termos de um conhecimento prévio de um especialista, confrontada com variáveis *fuzzy* representativas de graus de hostilidade. Com isso, em função da regra de obtenção de hostilidade utilizar variáveis *fuzzy* para representação do conhecimento, a resposta final da hostilidade não está em uma decisão, mas sim na obtenção de um valor contínuo que

é representativo por si só. O mesmo ocorre na rotulação do acompanhamento. Ainda que o objetivo final seja atribuir um nome ao acompanhamento, não é possível avaliá-los em termos discretos. Tal qual a hostilidade, o resultado da *defuzzificação* é um valor contínuo, que representa um grau de pertinência da avaliação. O grau de pertinência da classificação é apresentado para todos os tipos de acompanhamentos, deixando, a cargo do operador, a decisão final.

Assim como os testes de correlação, os testes de classificação foram feitos em duas baterias, a primeira com um erro de estimação de filtragem de 5 metros e a segunda bateria com erro de estimação de filtragem em 20 metros. Estas duas baterias foram executadas durante 3 minutos para cada caso de teste, e os valores, coletados a cada 10 segundos, dão base para computação de uma média aritmética dos cálculos de hostilidade e classificação (tabela 6.7).

S=5	**Hostilidade**		**Grau de pertinência na classificação de um cargueiro**		**Grau de pertinência na classificação de um navio comercial**		**Grau de pertinência na classificação de um míssil**	
	Média	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Caso de Teste 1	19,5	1,3	0,98	0,02	0	0	0	0
Caso de Teste 2	20,95	0	0	0	1	0	0	0
Caso de Teste 3	100	0	0	0	0	0	1	0
S=20	**Hostilidade**		**Grau de pertinência na classificação de um cargueiro**		**Grau de pertinência na classificação de um navio comercial**		**Grau de pertinência na classificação de um míssil**	
	Média	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
Caso de Teste 1	20,1	2,1	0,73	0,04	0	0	0	0
Caso de Teste 2	40	5	0	0	0,95	0,05	0	0
Caso de Teste 3	98	2	0	0	0	0	0,98	0,02

Tabela 6.7: resultado da classificação de acompanhamentos.

De maneira geral, o resultado foi satisfatório, à medida que as regras de classificação associadas a cada caso de teste retornaram resultados positivos.

Com um erro de estimação de filtragem em 5 e 20 metros, os graus de pertinência nas rotulações dos acompanhamentos referentes ao navio comercial e ao míssil obtiveram pontuações satisfatórias. Isto se deve ao fato de que o erro de filtragem

não influencia significativamente na investigação das regras de classificação, pois os dados cinemáticos suscetíveis à variação (velocidade, posição e rumo) não alteram a pertinência em relação as suas respectivas variáveis *fuzzy*.

Na regra de classificação de um míssil e na regra para classificação de um navio comercial, a pertinência para a resposta do IFF (“neutro/desconhecido”) e o tamanho do alvo (“tamanho”) sempre será 1, pois estes dados são simulados pelo operador, e não sofrem com o erro “s”. Mesmo com pertinência diferente de 1, a variável *fuzzy* “velocidade” está ampla o suficiente para abranger a variação da qualidade da filtragem, não comprometendo a classificação nem o cálculo de hostilidade.

Por outro lado, a investigação da regra para classificação de um navio cargueiro está suscetível à variação do erro, pois os dados cinemáticos são a essência da classificação, uma vez que o comportamento do acompanhamento é confrontado com uma rota comercial previamente definida. Contudo, mesmo com um erro de 5 e 20 metros, a capacidade de rotulação do alvo não foi afetada.

Para o cálculo de hostilidade, os erros de 5 e 20 metros também não comprometeram, uma vez que de forma semelhante às regras de classificação de um navio comercial e de um míssil, os erros de cálculo dos dados cinemáticos não são preponderantes na avaliação da hostilidade. O que ocorre é que a pertinência para a resposta do IFF (“neutro/desconhecido”) e o tamanho do alvo (“tamanho”) sempre será 1, pois estes dados são introduzidos artificialmente, além de terem variáveis *fuzzy* amplas o suficiente para não deixarem os erros influenciarem nas decisões.

Por fim, ressalta-se que o cálculo de hostilidade efetuado está em consonância com as rotulações obtidas, não apresentando nenhuma discrepância entre a hostilidade e os graus de pertinência obtidos na classificação.

6.5.4 Avaliação computacional do RT-MLR para classificação de acompanhamentos na investigação de contexto militar naval

Como mencionado anteriormente, os objetos *tracks* são instanciados ou destruídos à medida que as demandas no ambiente de simulação ocorrem. Com isso, as regras de classificação e as regras para cálculo de hostilidade são instanciadas para cada novo acompanhamento criado e destruídas a cada vez que um acompanhamento é destruído.

Seja, portanto, a relação de N o somatório dos acompanhamentos simulados pelo radar DECCA mais os acompanhamentos simulados do radar DRBV-15. Seja, ainda, 3

regras de classificação (objetos *rule*). Portanto, o número total de regras instanciadas é da ordem *Nx3* objetos de regras de classificação. A complexidade algorítmica computacional, quer seja em relação ao consumo de memória, quer seja em relação ao consumo do processador, cresce linearmente ao número de acompanhamentos existentes no pior caso. Assim, a associação de regras de classificação é da ordem de O(*N*).

O mesmo ocorre para as regras de cálculo de hostilidade. Seja o número de regras de obtenção do tamanho do alvo igual a 3. Seja, ainda, o somatório do número de acompanhamentos existentes. Então, o número total de regras instanciadas para cálculo de hostilidade é da ordem *N*x*3* objetos *rules*. A complexidade algorítmica computacional, quer seja em relação ao consumo de memória, quer seja em relação ao consumo do processador, cresce linearmente ao número de acompanhamentos existentes. Isto é, o pior caso para esta associação é da ordem O(*N*).

7. CONCLUSÃO E TRABALHOS FUTUROS

De maneira geral, a decisão de modelar e implementar o *framework* se mostrou acertada. Conforme a tabela 5.1, a amostra de produtos similares ao RT-MLR não apresentou nenhuma maneira de representar incerteza e condições associadas ao contexto temporal. Mais do que isso, a integração com a linguagem de programação e os mecanismos para monitoração do contexto não estão alinhados com os requisitos de sistemas sensíveis ao contexto e de tempo real.

Como havia sido pressuposto no início do trabalho, a transformação do conhecimento tácito para um conhecimento explícito do especialista pôde ser provido de uma forma mais adequada, através da compilação das regras pelo próprio compilador Java e na adoção de diferentes tipos de lógicas.

A capacidade de expressão do conhecimento ficou evidenciada por meio de todos os exemplos e casos de teste utilizados para definição de regras, cujas lógicas de primeira ordem, temporal e regras *fuzzy* puderam ser demonstradas e aplicadas a um contexto militar. Ademais, a não necessidade de aprendizado de outras linguagens para construção das regras em um projeto de *software* Orientado a Objetos constituiu um importante recurso para integração e justificação das tomadas de decisão no âmbito naval.

Em função do desenvolvimento do RT-MLR ter como motivação a aplicação a um contexto militar naval, certos aspectos do projeto de *software* necessitaram ser priorizados. As decisões na maioria das vezes eram relativas ao sacrifício ou simplificação de alguma característica presente em produtos similares, visando à adição de outro recurso que fosse mais importante para o sistema tático. Por exemplo, o motor de inferência foi desenvolvido incluindo apenas a abordagem de investigação do tipo “encadeamento para frente”. Segundo o especialista da Marinha do Brasil, apenas esta abordagem era necessária para o contexto naval.

Posto esta consideração, pode-se afirmar que a extensibilidade é a primeira vantagem encontrada na solução de projeto do RT-MLR. Como os operadores seguem uma relação hierárquica, adicionar um novo tipo de lógica ou mesmo novos operadores corresponde apenas à criação de classes que herdem das respectivas classes representativas do nível de mudança que se deseja. Se a lógica *fuzzy*, por exemplo, não

se mostrar suficiente para representar incerteza, a lógica de Dempster-Shafer poderá ser adicionada futuramente de uma maneira direta.

A segunda vantagem encontrada na solução de projeto de software está justamente na associação dos objetos OO às regras. A adoção da abordagem de monitoração por eventos permite que um número menor de regras sejam investigadas de acordo com eventos pré-estabelecidos do contexto. Em contrapartida, esta solução poderá incorrer em problema de manutenibilidade. Como pôde ser visto, o código fonte do sistema deverá ser re-compilado todas as vezes que uma regra necessitar ser adicionada ou retirada. Nesse sentido, como trabalho futuro, estão sendo estudadas algumas propostas de integração de uma base de dados que possa trazer algum tipo de flexibilidade para o gerenciamento de regras em tempo de execução, assim como o Drools [28]. Vale ressaltar apenas que isso não era uma necessidade do projeto TTI (motivação deste trabalho) e, portanto, poderia ser ignorada a *priori*.

Finalmente, em relação aos testes realizados, caso os dados fossem aplicados a mais tipos de classificações, o resultado poderia ser mostrar diferente do que foi apresentado. Como o problema da classificação foi apresentado e testado para casos isolados, a única afirmação possível é a proficiência do RT-MLR no apoio à implementação de regras e expressão do conhecimento por meio de lógicas heterogêneas. No entanto, segundo o especialista da Marinha do Brasil, a associação direta das regras aos objetos do domínio permitiu que o contexto fosse investigado de forma precisa e acurada, conforme os resultados obtidos com a correlação e classificação de alvos (tabela 6.6 e tabela 6.7). Ademais, a aplicabilidade do RT-MLR em sistemas de tempo real pode ser observada no estudo de caso do sistema naval, no qual a complexidade algorítmica mostrou-se, na maioria dos casos, linear no consumo de recursos computacionais (ver tópicos de avaliação computacional).

A avaliação dos critérios utilizados para correlação e classificação de acompanhamentos, bem como a avaliação da qualidade das regras definidas não faziam parte do escopo dos testes propostos pelo especialista do domínio. Mesmo não fazendo parte dos objetivos iniciais, torna-se razoável executar testes comparativos de fusão e classificação de dados militares com abordagens tradicionais, tais quais os tradicionais métodos probabilísticos e ontológicos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. Grudin, “Computer-supported cooperative work: history and focus”, Los Alamitos, CA, USA: IEEE Computer Society Press, vol. 27, no. 5, pp. 19-26, 1994.

[2] R. Want, A. Hopper, V. Falcão, e J. Gibbons, “The active badge location system”, *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, 1992.

[3] J. Hong, E. Suh, e S. Kim, “Context-aware systems: A literature review and classification,” *Expert Syst. Appl.*, vol. 36, 2009, pp. 8509-8522.

[4] F. Baader, A. Bauer, P. Baumgartner, A. Cregan, A. Gabaldon, K. Ji, K. Lee, D. Rajaratnam, e R. Schwitter, “A Novel Architecture for Situation Awareness Systems,” *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Oslo, Norway: Springer-Verlag, 2009, pp. 77-92.

[5] T. Demiris, “Context Revisited: A brief survey of research in context”, in Context awareness in ubiquitous environment, *Proc. of the 3rd international conference on Mobile multimedia communications*, ACM International Conference Proceeding Series, ICST, Brussels, Belgium: vol. 329, 2007.

[6] Merriam Webster Dictionary. Disponível em: http://www.merriam-webster.com/dictionary/context. Acesso em: 21 Junho de 2010, 13:36:30.

[7] H. Byun e K. Cheverst, “Utilizing Context History To Provide Dynamic Adaptations,” *Applied Artificial Intelligence*, vol. 18, 2004, pp. 533-548.

[8] B. Schilit, N. Adams, e R. Want, “Context-Aware Computing Applications,” *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, IEEE Computer Society, 1994, pp. 85-90.

[9] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, e P. Steggles, “Towards a Better Understanding of Context and Context-Awareness,” *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Karlsruhe, Germany: Springer-Verlag, 1999, pp. 304-307.

[10] G. Chen e D. Kotz, "A Survey of Context-Aware Mobile Computing Research," *Technical Repor*t, TR2000-381, Dept. of Computer Science, Dartmouth College, November, 2000.

[11] R. Hull, P. Neaves, e J. Bedford-Roberts, “Towards Situated Computing”, *Proceedings of the 1st IEEE International Symposium on Wearable Computers*, IEEE Computer Society, 1997, p. 146.

[12] M. Baldauf, S. Dustdar, e F. Rosenberg, “A survey on context-aware systems,” *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, 2007, pp. 263-277.

[13] T. Strang e C. Linnhoff-Popien, “A Context Modeling Survey”, in *1st Int. Workshop on Advanced Context Modeling*, Reasoning and Management, 2004.

[14] G. Gkekas, “A smart calendar application for mobile environments”, in Context awareness in ubiquitous environment, *Proc. of the 3rd international conference on Mobile multimedia communications*, ACM International Conference Proceeding Series, ICST, Brussels, Belgium: vol. 329, 2007.

[15] K. Fujinami , T. Yamabe, e T. Nakajima, “Take me with you! : A case study of context aware application integrating cyber and physical spaces”, in *Symposium on Applied Computing*, Proc. 2004 ACM symposium on Applied computing, pp. 1607-1614, 2004.

[16] A. Scherp e S. Boll, “Context driven Smart Authoring of Multimedia Content with xSMART”, in *International Multimedia Conference*, Proc. 13th annual ACM international conference on Multimedia, pp. 802-803, 2005.

[17] A. Brodt e D. Nicklas, “The TELAR Mobile Mashup Platform for Nokia Internet Tablets”, in ACM International Conference Proceeding Series, *Proc. 11th International conference on Extending database technology: Advances in database technology*, Nova York, USA: vol. 261, pp. 700-704, 2008.

[18] A. K. Dey e G. D. Abowd, “CybreMinder: A Context-Aware System for Supporting Reminders”, in *Lecture Notes In Computer Science*, Proc. 2nd international symposium on Handheld and Ubiquitous Computing, London, UK: Springer Verlag, vol. 1927, pp. 172-186, 2000.

[19] M. Pichler, U. Bodenhofer e W. Schwinger, “context-awareness and artificial intelligence”, in *Österreichische Gesellschaft für Artificial Intelligence (ÖGAI)*, 2004.

[20] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, e D. Riboni, “A survey of context modelling and reasoning techniques,” *Pervasive Mob. Comput.*, vol. 6, 2010, pp. 161-180.

[21] P. Coad e E. Yourdon, *Projeto baseado em objetos*, Editora Campus, 1993.

[22] B. Bouzy e T. Cazenave, “Using the Object Oriented Paradigm to Model Context in Computer Go,” *in Proceedings of Context’97*, 1997.

[23] Ó. Corcho e A. Gómez-Pérez, “A Roadmap to Ontology Specification Languages,” *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, Springer-Verlag, 2000, pp. 80-96.

[24] R. Rosati, “On combining description logic ontologies and nonrecursive datalog rules”. in *Proc. of the 2nd Int. Conf. on Web Reasoning and Rule Systems (RR 2008)*, 2008.

[25] K. Baclawski, M.K. Kokar, P.A. Kogut, L. Hart, J. Smith, J. Letkowski, e P. Emery, “Extending the Unified Modeling Language for ontology development,” *Software and Systems Modeling*, vol. 1, Dez. 2002, pp. 142-156.

[26] E. Friedman-Hill, *Jess in Action : Java Rule-Based Systems*, Manning, 2003.

[27] CLIPS: A Tool for Building Expert Systems. Disponível em: http://clipsrules.sourceforge.net/. Acesso em: 16 de Agosto de 2010, 13:30:05.

[28] Drools 5 - The Business Logic integration Platform. Disponível em: http://www.jboss.org/Drools/. Acesso em: 16 de Agosto de 2010, 13:37:09.

[29] Hammurapi Rules, Disponível em: http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/products/hammurapi-rules/. Acesso em: 16 de Agosto de 2010, 13:50:09.

[30] C. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp 17–37, 1982.

[31] M. Dörflinger, “Interpreting Courteous Logic Programs”, *Diploma Thesis*, 2005.

[32] L. A. M. Palazzo, *Introdução à Programação Prolog*, Editora da Universidade Católica de Pelotas, 1997.

[33] S.J. Russell e P. Norvig, *Artificial intelligence: a modern approach*, Prentice-Hall, Inc., 1995.

[34] L. Chittaro e A. Montanari, “Temporal representation and reasoning in artificial intelligence: Issues and approaches,” *Annals of Mathematics and Artificial Intelligence*, vol. 28, 2000, pp. 47-106.

[35] J.F. Allen, “Maintaining knowledge about temporal intervals,” *Communications of ACM*, vol. 26, 1983, pp. 832-843.

[36] B. Bennett e A.P. Galton, “A unifying semantics for time and events,” *Artif. Intell.* vol. 153, 2004, pp. 13-48.

[37] R. Dechter, I. Meiri e J.Pearl, "Temporal Constraints Networks", *Artificial Intelligence 49*, pp. 61-95, 1991.

[38] L.A. Zadeh, *Fuzzy sets. Information and Control*, 8, pp. 338-353,1965.

[39] P. E. M Almeida e A. G. Evsukoff, "Sistemas *fuzzy*", in *Sistemas inteligentes: fundamentos e aplicações*. Barueri: Manole, 2005. pp. 169–202.

[40] A.C.F. Guimarães e C.M.F. Lapa, "*Fuzzy* inference to risk assessment on nuclear engineering systems," *Appl. Soft Comput.*, vol. 7, 2007, pp. 17-28.

[41] J. McCarthy, "Notes on formalizing contexts", *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, San Mateo, California, United States: Morgan Kaufmann, 1993, pp. 555–560.

[42] J. McCarthy e S.Buvač, "Formalizing context (expanded notes)", *In Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language (Menlo Park, California*, American Association for Artificial Intelligence, 1993, pp. 99–135.

[43] P. Rangel, J.G. Carvalho, M.R. Ramirez, e J.M. Souza, "Context Reasoning through a multiple logic *framework*", *6th International conference on Intelligent Environments – IE'10*, Kuala Lumpur, Malaysia 2010.

[44] J.G. Carvalho, P. Rangel, e N. F. Ebecken, "A multi-logic *framework* for multi-level fusion in real time data fusion applications", *13th International Conference on Information Fusion*, Edinburgh, UK, 2010.

[45] O. Maler, D. Nickovic, e A. Pnueli, "Real Time Temporal Logic: Past, Present, Future," *Formal Modeling and Analysis of Timed Systems*, 2005, pp. 2-16.

[46] T. Murayama, K. Kushima, e S. Ishigaki, "An inference mechanism suited for real-time control," Proceedings of the 2nd international conference on Industrial and engineering applications of artificial intelligence and expert systems - Volume 1, Tullahoma, Tennessee, United States: ACM, 1989, pp. 245-253.

[47] A. Agostini, C. Bettini, e D. Riboni, "Hybrid reasoning in the CARE middleware for context awareness," *Int. J. Web Eng. Technol.*, vol. 5, 2009, pp. 3-23.

[48] D. Riboni e C. Bettini, "Context-Aware Activity Recognition through a Combination of Ontological and Statistical Reasoning," *Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing*, Brisbane, Australia: Springer-Verlag, 2009, pp. 39-53.

[49] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Bookman, 2000.

[50] A.M. Aziz, "*Fuzzy* track-to-track association and track fusion approach", in *distributed multisensor-multitarget multiple-attribute environment*, Signal Process., vol. 87, 2007, pp. 1474-1492.

[51] R. Singh e W. Bailey, "*Fuzzy* Logic applications to multisensor-multitarget correlation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, 1997, pp. 752-769.

[52] J.F. Baldwin, T.H. Cao, T.P. Martin, J.M. Rossiter, e A. Group, "Towards Soft Computing Object-Oriented Logic Programming," 2000.

[53] D. Sottara, P. Mello, e M. Proctor, "Adding Uncertainty to a Rete-OO Inference Engine," *Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*, Orlando, Florida: Springer-Verlag, 2008, pp. 104-118.

[54] J. Bento, B. Feijó, e D. L. Smith, "Engineering design knowledge representation based on logic and objects," Computers & Structures, vol. 63, n. 5, pp. 1015-1032, Jun. 1997.

[55] A. J. Prates, "Fundamentos e especificações de um ambiente de design baseado em logica e objetos", Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Civil, 1993.

[56] Carlos Santos da Figueira Filho, "JEOPS – Integração entre Objetos e Regras de Produção em Java", Dissertação de Mestrado, Universidade Federal de Pernambuco, Centro de Informática, 2000.

[57] D. Hall e S. McMullen, *Mathematical Techniques in Multisensor Data Fusion*, Artech House Publications, 2004.

[58] Y. Bar-Shalom e W. Blair, *Multitarget-Multisensor Tracking: Applications and Advances*, 2000

[59] O. Kessler, *Functional Description of the Data Fusion Process*, Warmininster, PA: Office of Naval Technology, Naval Air Development Center, 1992.

[60] M. Tummala e S. A. Midwood, "Multi Sensor Data Fusion Using Fuzzy Association Techniques", 1998.

[61] Pang-Ning Tan, Michael Steinbach e Vipin Kumar, *Introduction to Data Mining*, 2005.

APÊNDICE I: IMPLEMENTAÇÃO DAS REGRAS PARA OBTENÇÃO DO TAMANHO DE UM ACOMPANHAMENTO

```
import RTMLR.rules;
import RTMLR.types;
import RTMLR.operations;
import RTMLR.conclusions;
public class TrackSize{
        public static TrapezeFuzzyData closeDistance = new TrapezeFuzzyData (0,0,10,25);
        public static TrapezeFuzzyData mediumDistance = new TrapezeFuzzyData (10,25,50,100);
        public static TrapezeFuzzyData farDistance = new TrapezeFuzzyData (50,100,200,200);
        public static TrapezeFuzzyData lowCount = new TrapezeFuzzyData (0,0,4,10);
        public static TrapezeFuzzyData mediumCount = new TrapezeFuzzyData (4,10,16,20);
        public static TrapezeFuzzyData highCount = new TrapezeFuzzyData (16,20,25,25);
        public static TrapezeFuzzyData smallSize = new TrapezeFuzzyData (0,0,50,100);
        public static TrapezeFuzzyData mediumSize = new TrapezeFuzzyData (50,100,150,200);
        public static TrapezeFuzzyData bigSize = new TrapezeFuzzyData (150,200,250,250);

        public class TrackSizeConclusion extends FuzzyConclusion{
                public TrackSizeConclusion(FuzzyData fdata){
                        super(fdata);
                }
                public void conclusion (double result){
                        System.out.println("Belief on the rule: " + result);
                }
        }
        public class TrackSizeFuzzyRuleSet extends FuzzyRuleSet{
                public defuzzyfication (double result){
                        System.out.println("Size: " +  result) ;
                }
        }
        public TrackSize (Track track){
                FuzzyRuleSet fset = new TrackSizeFuzzyRuleSet();
                FuzzyRule rule = null;
                Operation premise = null;
                Conclusion conclusion = null;
                /*
                * REGRA 1:
```

```
* Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é
próxima
* Então o tamanho do alvo é pequeno.
*/
premise = new AndF (
                    new IsF(track.getCount(), lowCount),
                    new IsF(track.getDistance(),closeDistance)
          );
conclusion = new TrackSizeConclusion (smallSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
/*
* REGRA 2:
* Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é
média
* Então o tamanho do alvo é pequeno.
*/
premise = new AndF (
                    new IsF(track.getCount(), lowCount),
                    new IsF(track.getDistance(),mediumDistance)
          );
conclusion = new TrackSizeConclusion (smallSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
/*
* REGRA 3:
* Se a quantidade de vezes que o alvo foi iluminado é pequena E a distância do alvo é
longa
* Então o tamanho do alvo é médio.
*/
premise = new AndF (
                    new IsF(track.getCount(), lowCount),
                    new IsF(track.getDistance(),farDistance)
          );
conclusion = new TrackSizeConclusion (mediumSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
/*
* REGRA 4:
```

** Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é próxima*

** Então o tamanho do alvo é médio.*

*/

```
premise = new AndF (
                    new IsF(track.getCount(), mediumCount),
                    new IsF(track.getDistance(),closeDistance)
          );
conclusion = new TrackSizeConclusion (mediumSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
```

/*

* **REGRA 5:**

** Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é média*

** Então o tamanho do alvo é médio.*

*/

```
premise = new AndF (
                    new IsF(track.getCount(), mediumCount),
                    new IsF(track.getDistance(),mediumDistance)
          );
conclusion = new TrackSizeConclusion (mediumSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
```

/*

* **REGRA 6:**

** Se a quantidade de vezes que o alvo foi iluminado é média E a distância do alvo é longa*

** Então o tamanho o alvo é grande.*

*/

```
premise = new AndF (
                    new IsF(track.getCount(), mediumCount),
                    new IsF(track.getDistance(), farDistance)
          );
conclusion = new TrackSizeConclusion (bigSize);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
```

/*

* **REGRA 7:**

```
        * Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é
        próxima
        * Então o tamanho do alvo é médio.
        */
        premise = new AndF (
                        new IsF(track.getCount(), highCount),
                        new IsF(track.getDistance(), closeDistance)
                );
        conclusion = new TrackSizeConclusion (mediumSize);
        rule = new FuzzyRule (premise, conclusion);
        fset.addRule(rule);
        /*
        * REGRA 8:
        * Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é média
        * Então o tamanho do alvo é grande.
        */
        premise = new AndF (
                        new IsF(track.getCount(), highCount),
                        new IsF(track.getDistance(),mediumDistance)
                );
        conclusion = new TrackSizeConclusion (bigSize);
        rule = new FuzzyRule (premise, conclusion);
        fset.addRule(rule);
        /*
        * REGRA 9:
        * Se a quantidade de vezes que o alvo foi iluminado é alta E a distância do alvo é longa
        * Então o tamanho do alvo é grande.
        */
        premise = new AndF (
                        new IsF(track.getCount(), mediumCount),
                        new IsF(track.getDistance(), farDistance)
                );
        conclusion = new TrackSizeConclusion (bigSize);
        rule = new FuzzyRule (premise, conclusion);
        fset.addRule(rule);
    }
}
```

APÊNDICE II: IMPLEMENTAÇÃO DA REGRA PARA CORRELAÇÃO DE ACOMPANHAMENTOS

```
import RTMLR.types;
import RTMLR.rules;
import RTMLR.operations;
import RTMLR.conclusions;
public class Correlation{
        private static FirstOrderRuleSet fset = new FirstOrderRuleSet();
        public class CorrelationConclusion extends FirstOrderConclusion{
                public void conclusion (boolean result){
                        if(result) System.out.println("It's the same target!");
                }
        }
        public Classification (Track track1, Track track2){
                /* Se esta condição for verdadeira por 5 vezes: Se a diferença entre as posições de A1 e
                A2 for menor ou igual ao limiar E a diferença entre as velocidades de A1 e A2 for
                menor ou igual ao limiar E a diferença entre os rumos de A1 e A2 for menor ou igual
                ao limiar E a diferença entre os tamanhos de A1 e A2 for maior ou igual ao limiar,
                então os acompanhamentos A1 e A2 se tratam do mesmo alvo.
                */
                Operation premise =

                        New Count (
                                new And(
                                        new And (
                                        new le(track1.getDiferencePosition(track2), 2),
                                        new le(track1.getDiferenceSpeed(track2), 5),
                                        )
                                        new And (
                                        new le(track1.getDiferenceCourse (track2), 2),
                                        new He(track1.getDiferenceSize(track2), 0.5),
                                        )
                                ),
                                5
                        );
```

```
                Conclusion conclusion = new CorrelationConclusion ();
                FirstOrderRule rule = new FirstOrderRule (premise, conclusion);
                fset.addRule(rule);
        }
}
```

APÊNDICE III: IMPLEMENTAÇÃO DAS REGRAS PARA OBTENÇÃO DA HOSTILIDADE DE UM ACOMPANHAMENTO

```
import RTMLR.rules;
import RTMLR.types;
import RTMLR.operations;
import RTMLR.conclusions;
public class TrackHostility{
        Track track = null;
        public static TrapezeFuzzyData lowSpeed= new TrapezeFuzzyData (0,0,50,100);
        public static TrapezeFuzzyData mediumSpeed = new TrapezeFuzzyData (50,100,150,200);
        public static TrapezeFuzzyData highSpeed = new TrapezeFuzzyData (150,200,1000,1000);
        public static TrapezeFuzzyData smallSize = new TrapezeFuzzyData (0,0,50,100);
        public static TrapezeFuzzyData mediumSize = new TrapezeFuzzyData (50,100,150,200);
        public static TrapezeFuzzyData bigSize = new TrapezeFuzzyData (150,200,250,250);
        public static TrapezeFuzzyData lowHostility = new TrapezeFuzzyData (0,20,20,40);
        public static TrapezeFuzzyData mediumHostility = new TrapezeFuzzyData (20,40,40,60);
        public static TrapezeFuzzyData highHostility = new TrapezeFuzzyData (40,60,60,80);
        public static TrapezeFuzzyData criticalHostility = new TrapezeFuzzyData (60,80,80,100);

        public class TrackHostilityConclusion extends FuzzyConclusion{
                public TrackHostilityConclusion(FuzzyData fdata){
                        super(fdata);
                }
                public void conclusion (double result){
                        System.out.println("Belief on the rule: " + result);
                }
        }
        public TrackHostilityFuzzyRuleSet extends FuzzyRuleSet{
                public defuzzyfication (double result){
                        track.setHostility(result);
                }
        }
        public TrackHostility (Track track){
                this.track=track;
                FuzzyRuleSet fset = new TrackSizeFuzzyRuleSet();
                FuzzyRule rule = null;
                Operation premise = null;
```

```
Conclusion conclusion = null;
/*
* REGRA 1:
* Se o acompanhamento é “desconhecido” OU “amigo”  OU “neutro” E o tamanho do acompanhamento é grande E a velocidade do acompanhamento é baixa
* Então a hostilidade é baixa
*/
premise = new AndF (
                    new AndF(
                                new  Not(new Eq(track.getIff(),1)),
                                new IsF(track.getSize(),bigSize)
                    ),
                    new IsF(track.getSpeed(),lowSpeed)
          );
conclusion = new TrackHostilityConclusion (lowHostility);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
/*
* REGRA 2:
* Se o acompanhamento é “desconhecido” E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é média
* Então a hostilidade é média
*/
premise = new AndF (
                    new AndF(
                                new Eq(track.getIff(),3),
                                new IsF(track.getSize(),mediumSize)
                    ),
                    new IsF(track.getSpeed(),mediumSpeed)
          );
conclusion = new TrackHostilityConclusion (mediumHostility);
rule = new FuzzyRule (premise, conclusion);
fset.addRule(rule);
/*
* REGRA 3:
* Se o acompanhamento é “desconhecido” E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é alta
* Então a hostilidade é alta
*/
premise = new AndF (
```

```
                        new AndF(
                                new Eq(track.getIff(),3),
                                new IsF(track.getSize(),mediumSize)
                        ),
                        new IsF(track.getSpeed(),highSpeed)
                );
        conclusion = new TrackHostilityConclusion (highHostility);
        rule = new FuzzyRule (premise, conclusion);
        fset.addRule(rule);
        /*
        * REGRA 4:
        * Se o acompanhamento é “inimigo” ou o acompanhamento é “desconhecido” E a
        velocidade do acompanhamento é alta
        * Então a hostilidade é crítica
        */
        premise = new AndF (
                        new OrF(
                                new Eq(track.getIff(),3),
                                new Eq(track.getIff(),1),
                        ),
                        new IsF(track.getSpeed(),highSpeed)
                );
        conclusion = new TrackHostilityConclusion (highHostility);
        rule = new FuzzyRule (premise, conclusion);
        fset.addRule(rule);
    }
}
```

APÊNDICE IV: IMPLEMENTAÇÃO DAS REGRAS PARA OBTENÇÃO DA CLASSIFICAÇÃO DE UM ACOMPANHAMENTO

```
import RTMLR.rules;
import RTMLR.types;
import RTMLR.operations;
import RTMLR.conclusions;
public class TrackClassification{
        Track track = null;
        Route route=null;
        public static TrapezeFuzzyData lowSpeed= new TrapezeFuzzyData (0,0,50,100);
        public static TrapezeFuzzyData mediumSpeed = new TrapezeFuzzyData (50,100,150,200);
        public static TrapezeFuzzyData highSpeed = new TrapezeFuzzyData (150,200,1000,1000);
        public static TrapezeFuzzyData smallSize = new TrapezeFuzzyData (0,0,50,100);
        public static TrapezeFuzzyData mediumSize = new TrapezeFuzzyData (50,100,150,200);
        public static TrapezeFuzzyData bigSize = new TrapezeFuzzyData (150,200,250,250);
        public static TrapezeFuzzyData lowHostility = new TrapezeFuzzyData (0,20,20,40);
        public static TrapezeFuzzyData mediumHostility = new TrapezeFuzzyData (20,40,40,60);
        public static TrapezeFuzzyData highHostility = new TrapezeFuzzyData (40,60,60,80);
        public static TrapezeFuzzyData criticalHostility = new TrapezeFuzzyData (60,80,100,100);
        public static TrapezeFuzzyData closeDistanceRoute = new TrapezeFuzzyData (0,0,2,4);
        public static TrapezeFuzzyData closeCourseRoute = new TrapezeFuzzyData(0,0,5,7);
        public static TrapezeFuzzyData farDistanceRoute = new TrapezeFuzzyData(2,4,16,16);
        public static TrapezeFuzzyData farCourseRoute = new TrapezeFuzzyData(5,7,9,9);
        public static TrapezeFuzzyData comercialShipClassification = new
        TrapezeFuzzyData(0,0.25,0.75,1);
        public static TrapezeFuzzyData freighterClassification = new TrapezeFuzzyData(0,0.25,0.75,1);
        public static TrapezeFuzzyData missileClassification = new TrapezeFuzzyData(0,0.25,0.75,1);

        public class TrackClassificationConclusion extends FuzzyConclusion{
                public TrackClassificationConclusion(FuzzyData fdata){
                        super(fdata);
                }
                public void conclusion (double result){
                        System.out.println("Belief on the rule: " + result);
                }
        }
        public class FreighterClassificationFuzzyRuleSet extends FuzzyRuleSet{
```

```
        public void defuzzyfication (double result){
                if (track.getBeliefLabel() < result){
                        track.setLabel("Freighter");
                        track.setLabelBelief (result);
                }
        }
}
public void ComercialShipClassificationFuzzyRuleSet extends FuzzyRuleSet{
        public defuzzyfication (double result){
                if (track.getBeliefLabel() < result){
                        track.setLabel("Comercial Ship");
                        track.setLabelBelief (result);
                }
        }
}
public class MissileClassificationFuzzyRuleSet extends FuzzyRuleSet{
        public void defuzzyfication (double result){
                if (track.getBeliefLabel() < result){
                        track.setLabel("Missile");
                        track.setLabelBelief (result);
                }
        }
}
public TrackClassification (Track track, Route route){
        this.track=track;
        this,.route=route;
        FreighterClassificationFuzzyRuleSet FreigtherFset = new
FreighterClassificationFuzzyRuleSet();
        ComercialShipClassificationFuzzyRuleSet ComercialShipFset = new
ComecialShipClassificationFuzzyRuleSet();
        MissileClassificationFuzzyRuleSet missileFset = new
MissileClassificationFuzzyRuleSet();
        FuzzyRule rule = null;
        Operation premise = null;
        Conclusion conclusion = null;
        /*
        * REGRA 1:
        *Se o acompanhamento não é "inimigo" E a distância do acompanhamento para rota é
        pequena E a diferença entre o rumo do acompanhamento e a rota for pequena E o
        tamanho do acompanhamento é grande E a velocidade do acompanhamento é baixa
```

```
* Então o acompanhamento é um cargueiro
*/
premise = new AndF (
                    new AndF(
                                new  Not(new Eq(track.getIff(),1)),
                                new IsF(
                                            route.getDistance(track),
                                            closeDistanceRoute
                                )
                    ),
                    new AndF(
                                new IsF(
                                            route.getDifferenceCourse(track),
                                            closeCourseRoute
                                ),
                                new IsF(
                                            new IsF(track.getSize (),bigSize),
                                            new IsF(track.getSpeed(),lowSpeed)
                                )
                    ),
          );
conclusion = new TrackClassificationConclusion (freighterClassification);
rule = new FuzzyRule (premise, conclusion);
freighterFset.addRule(rule);
/*
```

* **REGRA 2:**

**Se o acompanhamento é neutro E a hostilidade do acompanhamento é baixa E o tamanho do acompanhamento é médio E a velocidade do acompanhamento é média Então o acompanhamento é um navio comercial*

```
*/
premise = new AndF (
                    new AndF(
                                new Eq(track.getIff(),2),
                                new IsF(track.getHostility (),lowHostility),
                    ),
                    new AndF(
                                new IsF(track.getSize (),mediumSize),
                                new IsF(track.getSpeed(),mediumSpeed)
                    )
          );
```

```
            conclusion = new TrackClassificationConclusion (comercialShipClassification);
            rule = new FuzzyRule (premise, conclusion);
            comercialShipFset.addRule(rule);
            /*
            * REGRA 3:
            *Se o acompanhamento é desconhecido E a hostilidade do acompanhamento é critica E
            o tamanho do acompanhamento é pequeno E a velocidade do acompanhamento é alta
            Então o acompanhamento é um míssil.
            */
            premise = new AndF (
                        new AndF(
                                new Eq(track.getIff(),3),
                                new IsF(track.getHostility (),criticalHostility),
                        ),
                        new AndF(
                                new IsF(track.getSize (),smallSize),
                                new IsF(track.getSpeed(),highSpeed)
                        )
                );
            conclusion = new TrackClassificationConclusion (missileClassification);
            rule = new FuzzyRule (premise, conclusion);
            missileFset.addRule(rule);
        }
}
```