

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC**  
**CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAI – CEAVI**  
**DEPARTAMENTO DE SISTEMAS DE INFORMAÇÃO**

**MARCELO DE SOUZA**

**FUZZYSTUDIO: FERRAMENTA WEB PARA MODELAGEM E SIMULAÇÃO DE**  
**SISTEMAS FUZZY**

**IBIRAMA – SC**

**2013**

**MARCELO DE SOUZA**

**FUZZYSTUDIO: FERRAMENTA WEB PARA MODELAGEM E SIMULAÇÃO DE  
SISTEMAS FUZZY**

Trabalho de Conclusão apresentado ao Curso de  
Sistemas de Informação, da Universidade do  
Estado de Santa Catarina, como requisito parcial  
para obtenção do grau de bacharel

Orientador: AdilsonVahldick

**IBIRAMA – SC**

**2013**

**MARCELO DE SOUZA**

**FUZZYSTUDIO: FERRAMENTA WEB PARA MODELAGEM E SIMULAÇÃO DE  
SISTEMAS FUZZY**

Trabalho de Conclusão apresentado ao Curso de Sistemas de Informação, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de bacharel.

**Banca Examinadora**

Orientador (a)

---

Prof. MSc. Adilson Vahldick  
Universidade do Estado de Santa Catarina

Membros:

---

Prof. MSc. Fernando dos Santos  
Universidade do Estado de Santa Catarina

---

Prof. MSc. Carlos Alberto Barth  
Universidade do Estado de Santa Catarina

**Ibirama – SC, 18 de junho de 2013**

A meus avós Lindo Dolzan (*in memoriam*) e  
Amália Dolzan (*in memoriam*).

## **AGRADECIMENTOS**

Agradeço a Deus por ter me concedido saúde e força para o alcance de mais este objetivo.

Aos meus pais, Paulo Renato e Marlene de Souza, por me ensinarem a ter fé e valores, tão importantes para a consecução deste trabalho.

Ao meu orientador Adilson Vahldick, por acreditar no trabalho desenvolvido e compartilhar do seu conhecimento e experiência para o sucesso do mesmo.

Aos professores e colaboradores da Universidade do Estado de Santa Catarina, que me acompanharam e participaram da construção do conhecimento nestes anos de estudo.

Aos professores da Universidade de León (Espanha), pela ajuda e apoio recebidos durante meus estudos no exterior.

A todos que, direta ou indiretamente, contribuíram para o desenvolvimento deste trabalho.

“Nunca, jamais desanimeis, embora venham ventos contrários”

- Santa Paulina

## RESUMO

SOUZA, Marcelo. **FuzzyStudio: Ferramenta Web para modelagem e simulação de sistemas fuzzy**. 2013. 168 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação – Área: Inteligência Artificial) – Universidade do Estado de Santa Catarina. Ibirama, 2013.

O estudo da lógica fuzzy em cursos superiores é prejudicado pelo escasso tempo dedicado ao tema, uma vez que o assunto é incorporado dentro de uma disciplina juntamente com outros temas. Apesar da possibilidade de aplicação em várias áreas do conhecimento, como educação, administração e engenharia, falta um ambiente para desenvolver e construir esses sistemas difusos. O presente trabalho trata do desenvolvimento de uma ferramenta para modelagem e simulação de sistemas fuzzy. Este sistema busca facilitar o processo de construção de um sistema difuso. Dessa forma, a ferramenta pode ser aplicada em ambiente acadêmico como forma de experimentação prática dos conceitos estudados, bem como no cenário comercial para o desenvolvimento de controladores e sistemas inteligentes. Como diferenciais, buscou-se o desenvolvimento de um ambiente de simulação do modelo, geração de gráficos de execução, geração de códigos em notação padrão FCL e em linguagem de programação Java. Além disso, a ferramenta possibilita o trabalho colaborativo, permitindo que mais de um usuário desenvolva simultaneamente o mesmo projeto. A aplicação é executada na Web, desenvolvida em Java Server Faces e componentes Primefaces, utiliza banco de dados MySQL, Java Persistence API, EJB e HTML5. A ferramenta suporta o modelo Mamdani de sistemas difusos e sua implementação é explorada da biblioteca *jFuzzyLogic*. Ao final do desenvolvimento, o sistema foi disponibilizado à experimentação, buscando validar seu funcionamento e eficácia frente ao problema proposto. A ferramenta foi aplicada com uma turma de acadêmicos da disciplina de Inteligência Artificial da Universidade do Estado de Santa Catarina. Na oportunidade, os acadêmicos realizaram exercícios na plataforma, bem como entregaram um trabalho proposto pelo professor. Ao final foi realizada uma pesquisa com esses alunos buscando avaliar a facilidade de utilização do software. O sistema se mostrou eficaz e simples na construção de sistemas fuzzy, bem como sua simulação e geração de artefatos, podendo ser aplicado em ambiente acadêmico para a prática das teorias fuzzy, bem como no contexto comercial, na construção de soluções inteligentes que utilizem lógica difusa.

**Palavras-chave:** Lógica difusa. Sistema fuzzy. Modelagem. Fuzzy Control Language. Simulação.

## ABSTRACT

SOUZA, Marcelo. **FuzzyStudio: Web tool to fuzzy systems modeling and simulating.** 2013. 168 f. Conclusion Course Work (Bachelor in Information Systems – Area: Artificial Intelligence) – Santa Catarina State University. Ibirama, 2013.

The fuzzy logic study in higher education is hampered by the limited time devoted to it, since it is embedded into a discipline along with other topics. Despite the fact that the potential for application in several areas of knowledge, such as education, administration and engineering, it lacks an environment to develop these fuzzy systems. This work deals with the development of a tool for modeling and simulating fuzzy systems. The system aims to facilitate the fuzzy systems building process. Thus, the tool can be applied in an academic environment as a form of practical experimentation of the concepts studied, and in the commercial environment to develop intelligent controllers and systems. As an important feature, a simulation environment has been developed. This feature improves graphing execution, FCL standard code generation and Java code generation. Furthermore, the tool enables collaborative work, allowing that more than one user simultaneously develop the same project. The application runs on the Web, developed in Java Server Faces with Primefaces components, it uses MySQL database, Java Persistence API, EJB and HTML5. The tool supports the Mamdani fuzzy systems model and its implementation is exploited from the *jFuzzyLogic* library. With the development phase accomplished, the system was released for tests, as an attempt to evaluate its correctness and efficiency, given the problem. The tool was applied to a class of Artificial Intelligence in the Santa Catarina State University. In the opportunity, the students performed exercises on the platform as well as they delivered a work proposed by the teacher. After that it was conducted a survey with those students seeking to evaluate the facility of using the software. The system showed to be efficient and simple in the construction of fuzzy systems as well as in the simulation and generation of artifacts. So, it can be applied in an academic environment for the fuzzy theories practice, as well as in the commercial context, in the construction of intelligent solutions that use fuzzy logic.

**Key-words:** Fuzzy logic. Fuzzy systems. Modeling. Fuzzy Control Language. Simulation.



## LISTA DE ILUSTRAÇÕES

Figura 1 - Representação gráfica da variável linguística altura .....	27
Figura 2 – Arquitetura de um sistema fuzzy .....	28
Figura 3 – Representações gráficas das funções de pertinência usuais .....	32
Figura 4 - Representação gráfica do método de defuzzificação pelo Centro-da-Área ...	37
Figura 5 - Representação gráfica do método de defuzzificação pelo Centro-do-Máximo.....	39
Figura 6 - Representação gráfica do método de defuzzificação pela Média-do-Máximo .....	40
Figura 7 - Comunicação entre distintos programas e um sistema fuzzy em modo texto .....	42
Figura 8 - Interface do sistema Matlab Fuzzy Toolbox .....	46
Figura 9 - Tela de edição de funções de pertinência do Matlab Fuzzy Toolbox.....	47
Figura 10 – Tela Visualizador de Regras do Matlab Fuzzy Toolbox .....	48
Figura 11 – Interface do sistema InFuzzy .....	49
Figura 12 – Interface do sistema FuzzyGen.....	51
Figura 13 – Etapas de utilização da ferramenta FuzzyStudio .....	55
Figura 14 – Diagrama de casos de uso da aplicação.....	60
Figura 15 – Diagrama de atividades geral.....	70
Figura 16 - Diagrama de atividades do processo de modelagem do sistema fuzzy .....	72
Figura 17 – Diagrama de pacotes da ferramenta .....	74
Figura 18 – Diagrama de classes do pacote <i>br.udesc.controle</i> .....	76
Figura 19 - Diagrama de classes do pacote <i>br.udesc.controle</i> .....	77
Figura 20 – Diagrama de classes do pacote <i>br.udesc.facade</i> .....	79
Figura 21 - Diagrama de classes do pacote <i>br.udesc.modelo</i> .....	80
Figura 22 - Diagrama de classes relacionadas ao processo de cadastro de variáveis de entrada .....	82
Figura 23 - Diagrama de classes relacionadas ao processo manutenção de motores de inferência .....	84
Figura 24 – Diagrama de sequência do processo de cadastro de variáveis de saída .....	85
Figura 25 - Diagrama entidade relacionamento da ferramenta .....	87

Figura 26 - Arquitetura de camadas de um sistema.....	91
Figura 27 - Formulário de edição de variáveis de entrada .....	100
Figura 28 – Interface de desenho utilizando <i>canvas</i> .....	102
Figura 29 - Funcionamento da biblioteca DWR.....	103
Figura 30 - Passos da execução do modelo na ferramenta FuzzyStudio .....	108
Figura 31 - Demonstração da execução do modelo na ferramenta FuzzyStudio .....	108
Figura 32 - Tela de autenticação do usuário.....	127
Figura 33 - Tela de cadastro de usuários.....	127
Figura 34 – Central de projetos do Usuário .....	128
Figura 35 - Interface de compartilhamento de projetos .....	129
Figura 36 - Tela de cadastro de projeto.....	129
Figura 37 - Tela de edição de projetos.....	130
Figura 38 - Tela de edição de variáveis de entrada.....	131
Figura 39 - Tela de edição de termos linguísticos .....	132
Figura 40 - Gráfico de fuzzificação para a variável habilidade.....	133
Figura 41 - Tela de edição da base de regras.....	134
Figura 42 - Tela de edição de motores de inferência .....	135
Figura 43 - Interface de desenho que representa o modelo construído.....	136
Figura 44 - Edição de variáveis de entrada pela interface de desenho .....	137
Figura 45 - Interface de execução do sistema .....	137
Figura 46 - Interface de geração de gráficos .....	138
Figura 47 – Interface de geração da notação FCL.....	139
Figura 48 - Interface de geração de código Java .....	140
Figura 49 - Opções da edição de projetos .....	141

## LISTA DE QUADROS

Quadro 1 - Exemplo de sistema fuzzy no padrão FCL .....	43
Quadro 2 – Comparativo de recursos entre os trabalhos correlatos .....	52
Quadro 3 – Regras de negócio da aplicação.....	56
Quadro 4 - Requisitos funcionais da aplicação .....	58
Quadro 5 - Requisitos não funcionais da aplicação .....	59
Quadro 6 - UC-01. Cadastrar-se .....	61
Quadro 7 - UC-02. Autenticar-se.....	61
Quadro 8 - UC-03. Alterar dados pessoais.....	62
Quadro 9 - UC-04. Manter projetos.....	62
Quadro 10 - UC-05. Compartilhar projeto .....	63
Quadro 11 - UC-06. Manter variáveis .....	63
Quadro 12 - UC-07. Manter termos linguísticos .....	64
Quadro 13 - UC-08. Manter regras.....	65
Quadro 14 - UC-09. Manter motores de inferência .....	66
Quadro 15 - UC-10. Executar projeto.....	67
Quadro 16 - UC-11. Gerar gráficos .....	67
Quadro 17 - UC-12. Gerar Fuzzy Control Language .....	68
Quadro 18 - UC-13. Gerar classe Java.....	68
Quadro 19 - UC-14. Editar componentes visuais .....	68
Quadro 20 - Relação entre atividades e casos de uso .....	73
Quadro 21 - Descrição das classes de controle de sistema .....	78
Quadro 22 - Descrição das classes envolvidas no processo de cadastro de variáveis de entrada .....	82
Quadro 23 - Descrição das classes envolvidas no cadastro e vinculação de regras e motores de inferência .....	84
Quadro 24 - Dicionário de dados para as entidades do sistema .....	88
Quadro 25 - Trecho do código da classe VariavelEntrada.....	92
Quadro 26 - Trecho do código da classe MotorInferencia.....	93
Quadro 27 - Conteúdo do arquivo <i>persistence.xml</i> que define a unidade de persistência .....	94

Quadro 28 - Conteúdo do arquivo <i>glassfish-resources</i> que define os recursos do servidor .....	95
Quadro 29 - Trecho de código da classe <i>RegraFacade</i> .....	96
Quadro 30 - Trecho de código da classe <i>TermoLinguisticoEntradaBean</i> .....	97
Quadro 31 - Trecho de código da tela que edita variáveis de entrada .....	99
Quadro 32 - Declaração do componente <i>canvas</i> na tela .....	102
Quadro 33 - Estrutura do arquivo <i>dwr.xml</i> .....	104
Quadro 34 - Trecho de código do arquivo <i>web.xml</i> .....	104
Quadro 35 - Importação dos arquivos JavaScript referentes ao DWR .....	105
Quadro 36 - Funções JavaScript de controle de usuários e redesenho da interface .....	105
Quadro 37 - Trechos de código do método de execução do sistema pelo motor de inferência .....	109
Quadro 38 - Manipulação dos gráficos das variáveis do sistema.....	113
Quadro 39 – Método de montagem da notação FCL.....	114
Quadro 40 - Classe Java gerada pela ferramenta.....	116
Quadro 41 - Métodos de desenho do <i>canvas</i> para as variáveis de entrada .....	118
Quadro 42 - Trecho de código do método de desenho de linhas no <i>canvas</i> .....	121
Quadro 43 - Eventos de escuta do sistema para redesenho do elemento <i>canvas</i> .....	123
Quadro 44 - Método executado a cada movimento do <i>mouse</i> com um elemento selecionado .....	124
Quadro 45 - Comparação entre os trabalhos correlatos e o FuzzyStudio .....	147

## LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
C-o-A	Center of Area
C-o-M	Center of Maximum
CSRF	Cross-Site Request Forgery
DWR	Direct Web Remoting
EE	Enterprise Edition
EJB	Enterprise Java Beans
FCL	Fuzzy Control Language
FIS	Fuzzy Inference System
HTML	HyperText Markup Language
IA	Inteligência Artificial
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
J2EE	Java to Enterprise Edition
JDK	Java Development Kit
JPA	Java Persistence API
JSF	Java Server Faces
M-o-M	Media of Maximum
MVC	Model View Controller
OS	Operating System
PNG	Portable Network Graphics
POJO	Plain Old Java Objects
RF	Requisito Funcional
RN	Regra de Negócio
RNF	Requisito Não Funcional
SGBD	Sistema de Gerenciamento de Banco de Dados
UC	Use Case

UDESC	Universidade do Estado de Santa Catarina
UDP	User Datagram Protocol
UML	Unified Modeling Language
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>16</b>
1.1 PROBLEMA.....	18
1.2 OBJETIVOS .....	19
<b>1.2.1 Objetivo geral .....</b>	<b>19</b>
<b>1.2.2 Objetivos específicos.....</b>	<b>19</b>
1.3 JUSTIFICATIVA.....	20
1.4 METODOLOGIA .....	21
1.5 ESTRUTURA.....	23
<b>2 FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>24</b>
2.1 LÓGICA FUZZY.....	24
<b>2.1.1 Bivalência x polivalência .....</b>	<b>24</b>
<b>2.1.2 Representação nebulosa através de conjuntos .....</b>	<b>25</b>
<b>2.1.2.1 Representação gráfica dos conjuntos nebulosos.....</b>	<b>27</b>
2.2 SISTEMAS FUZZY.....	28
<b>2.2.1 Arquitetura de um sistema fuzzy .....</b>	<b>29</b>
<b>2.2.2 Fuzzificação e as funções de pertinência .....</b>	<b>30</b>
<b>2.2.3 Máquina de inferência .....</b>	<b>34</b>
<b>2.2.3.1 Estrutura da base de regras .....</b>	<b>34</b>
<b>2.2.3.2 Inferência fuzzy .....</b>	<b>35</b>
<b>2.2.4 Defuzzificação.....</b>	<b>36</b>
<b>2.2.4.1 Defuzzificação pelo Centro-da-Área (C-o-A) .....</b>	<b>37</b>
<b>2.2.4.2 Defuzzificação pelo Centro-do-Máximo (C-o-M).....</b>	<b>38</b>
<b>2.2.4.3 Defuzzificação pela Média-do-Máximo (M-o-M) .....</b>	<b>39</b>
<b>2.2.4.4 Questões pertinentes à defuzzificação.....</b>	<b>40</b>
2.3 FUZZY CONTROL LANGUAGE.....	40
2.4 TRABALHOS CORRELATOS .....	44
<b>2.4.1 Matlab Fuzzy Toolbox .....</b>	<b>45</b>
<b>2.4.2 InFuzzy .....</b>	<b>48</b>
<b>2.4.3 FuzzyGen .....</b>	<b>49</b>
<b>2.4.4 Comparativo dos trabalhos correlatos .....</b>	<b>52</b>
<b>3 DESENVOLVIMENTO DA FERRAMENTA .....</b>	<b>53</b>

3.1 CONCEPÇÃO .....	53
3.2 REQUISITOS E REGRAS DE NEGÓCIO .....	56
3.3 CASOS DE USO .....	59
3.4 DIAGRAMA DE ATIVIDADES .....	69
3.5 DIAGRAMA DE CLASSES .....	73
3.6 MODELO ENTIDADE RELACIONAMENTO .....	86
3.7 IMPLEMENTAÇÃO .....	89
<b>3.7.1 Tecnologias utilizadas .....</b>	<b>90</b>
<b>3.7.1.1 Java Persistence API (JPA).....</b>	<b>90</b>
<b>3.7.1.2 EJB.....</b>	<b>95</b>
<b>3.7.1.3 PrimeFaces.....</b>	<b>97</b>
<b>3.7.1.4 HTML5 .....</b>	<b>100</b>
<b>3.7.1.5 Direct Web Remoting.....</b>	<b>102</b>
<b>3.7.2 Execução do motor .....</b>	<b>106</b>
<b>3.7.2.1 Biblioteca <i>jFuzzyLogic</i> .....</b>	<b>109</b>
<b>3.7.2.2 Geração de artefatos.....</b>	<b>112</b>
<b>3.7.3 Desenho do modelo.....</b>	<b>118</b>
3.8 OPERACIONALIZAÇÃO .....	126
<b>3.8.1 Utilização do sistema .....</b>	<b>126</b>
<b>3.8.2 Trabalho colaborativo .....</b>	<b>141</b>
3.9 EXPERIMENTAÇÃO E VALIDAÇÃO .....	141
3.10 RESULTADOS E DISCUSSÕES .....	143
<b>4 CONCLUSÕES .....</b>	<b>150</b>
4.1 TRABALHOS FUTUROS .....	152
<b>REFERÊNCIAS.....</b>	<b>153</b>
<b>APÊNDICES .....</b>	<b>159</b>



## 1 INTRODUÇÃO

A inteligência artificial (IA) surge como um ramo da ciência da computação capaz de prover aos sistemas computacionais inteligência similar à humana. Esta inteligência permite que os softwares sejam capazes de realizar tarefas de maneira tão eficiente quanto pessoas. Neste contexto, um sistema computacional torna-se capaz de melhorar a qualidade de vida dos seres humanos, uma vez podendo realizar atividades em seu lugar (LUGER, 2004).

De maneira geral, IA é o estudo dos sistemas que operam de modo que, para um observador pareceria inteligente. Para Coppin (2010), inteligência artificial é a utilização de métodos baseados no raciocínio e no comportamento humano aplicados na resolução de problemas complexos.

Buscando traduzir o raciocínio humano em máquinas são utilizadas algumas técnicas desenvolvidas ao longo do tempo. Os computadores utilizam em seu funcionamento a lógica binária, na qual se consideram dois valores, 0 (zero) e 1 (um), representando ligado/desligado, verdadeiro/falso. O raciocínio binário é conhecido por lógica aristotélica ou bivalente (KLIR; YUAN, 1995). Por outro lado, o homem em seu cotidiano utiliza uma maneira distinta nas suas decisões, pois não faz uso de apenas dois valores em suas ponderações. Um dia chuvoso na concepção aristotélica poderia ser analisado e concluído em dois valores, dia chuvoso ou dia não chuvoso (1 ou 0 respectivamente). Para o ser humano, esta conclusão na atribuição às condições climáticas possui uma vasta coleção de valores possíveis: chuva fraca, chuva forte, tempestade ou garoa por exemplo. O ser humano obtém conclusões de maneira mais complexa e incerta, expressando um grau de intensidade sobre determinado fato/estado. Uma vez concluído que a inteligência artificial busca simular o pensamento e comportamento humano (TANIMOTO, 1987), surge a necessidade de representar esse conjunto mais complexo de valores na busca de conclusões satisfatórias. Nasce assim a lógica polivalente e o raciocínio nebuloso, ou raciocínio fuzzy<sup>1</sup>.

A lógica fuzzy, segundo explica Jain e Martin (1998), foi desenvolvida na década de 60 por Lotfi Asker Zadeh com o objetivo de implementar a multivalência de valores lógicos, capaz de representar o conhecimento incerto e impreciso que o homem utiliza no seu cotidiano. Para a época, a teoria proposta por Zadeh desafiou algumas linhas de pensamento e teorias até então predominantes. Suas ideias expressavam uma nova forma de ver a lógica,

---

<sup>1</sup> Neste trabalho são usados os termos nebuloso, fuzzy e difuso como sinônimos

mais voltada à base do raciocínio humano. Em outras palavras, sua proposta se opunha à lógica clássica aristotélica, a base da computação. Conforme Ross (2004), o vago e o incerto, bem como o grau de intensidade sobre determinada perspectiva eram trabalhados através da teoria da probabilidade, sendo esta baseada na lógica clássica. Dessa forma, Zadeh (1965) propunha uma ótica diferenciada no tratamento do impreciso.

Desde então a lógica difusa vem crescendo e se desenvolvendo de forma acelerada. Encontra-se raciocínio fuzzy incorporado a uma gama extremamente variada de aplicações, caracterizando-se como uma técnica altamente interdisciplinar, utilizada nos mais diversos campos de estudo. A exemplo disso, Andrade et al. (2012) apresentam uma aplicação da lógica fuzzy em operadores morfológicos no processo de manipulação de imagens, com o objetivo de melhorar a contagem de fungos micorrízicos. Gabriel Filho et al. (2012) desenvolveram um sistema nebuloso que, com base na habilidade, conhecimento e atitude de um conjunto de trabalhadores, determina o mais apto para o cargo de operador de colheitadeiras no setor sucroalcooleiro.

Dada a vasta aplicação da lógica fuzzy em sistemas inteligentes, são necessárias ferramentas que ofereçam suporte a projetistas e desenvolvedores dessa categoria de software caracterizada pela sua complexidade. No mercado atual encontram-se vários softwares com a proposta de auxílio na modelagem de sistemas fuzzy. Apesar das ferramentas disponíveis, desenvolvedores de soluções baseadas em raciocínio nebuloso encaram alguns problemas durante o projeto: falta de conhecimento na lógica utilizada; carência de estrutura para modelagem, ausência de documentação e dificuldade em iniciar a codificação ou até mesmo mantê-la. A ferramenta proposta para o desenvolvimento no presente trabalho busca preencher estas lacunas no cenário fuzzy, ou seja, foi elaborada com base nas carências apresentadas pelas ferramentas já existentes, buscando ser um sistema para todo o projeto a ser desenvolvido, modelagem, simulação e codificação.

Além da complexidade de uso apresentada pelas ferramentas estudadas, poucas dispõem de geração de códigos de programação. Esta funcionalidade, bem como a visualização de gráficos e mecanismos de simulação, auxilia de maneira significativa em meio acadêmico, pois promove um melhor entendimento de sistemas fuzzy e sedimentação dos conceitos vistos em teoria. O tema fuzzy, quando ministrado nos cursos superiores, geralmente não abrange os conceitos em sua totalidade, devido à falta de tempo e complexidade. Para o entendimento e compreensão dos cálculos realizados em sistemas nebulosos se faz necessário um estudo intenso e, portanto, demanda-se tempo. Por conta

desses fatores comumente ocorrentes na academia, as teorias e cálculos estudados não podem ser comprovados na prática.

A proposta de desenvolvimento apresentada por este trabalho, mostra um projeto que se sustenta na aplicação em meio acadêmico com dois enfoques: o primeiro para os cursos da área da computação, em que a ferramenta pode ser utilizada no momento da modelagem de sistemas nebulosos, facilitando o aprendizado de inteligência artificial e do tema fuzzy. Além do apoio na modelagem da lógica nebulosa, a ferramenta gerará código fonte Java, o qual poderá ser utilizado para o desenvolvimento de aplicações que farão uso do sistema fuzzy modelado; o segundo enfoque será para universitários de outras áreas, uma vez que os conceitos fuzzy podem ser aplicados a problemas nos mais diversos campos do conhecimento, como na contabilidade, na economia, engenharia e medicina. Neste contexto, o usuário modelará o sistema nebuloso de acordo com sua necessidade e a ferramenta permitirá a realização de simulações, nas quais o usuário informará os valores de entrada e o sistema calculará as saídas de acordo com a modelagem realizada. Neste mesmo momento, a aplicação apresentará gráficos para cada variável envolvida no modelo desenvolvido. Dessa forma torna-se possível a realização de testes e a utilização do sistema difuso na aplicação em um problema específico.

Para o desenvolvimento de sistemas difusos, a *International Electrotechnical Commission* (IEC) mantém a norma IEC 1136-7 (1997), que trata da padronização de notação de sistemas fuzzy. Foi estabelecida uma linguagem para a representação textual desses sistemas, conhecida por *Fuzzy Control Language* (FCL). Essa padronização determina a sintaxe da notação textual que representa qualquer sistema difuso, garantindo a compatibilidade do sistema com qualquer software que forneça uma interface no padrão da norma, bem como sua escalabilidade para aplicações de variados portes (CINGOLANI; ALCALÁ-FDEZ, 2012). Neste sentido, é desejável que a ferramenta de modelagem até então comentada atenda às especificações do padrão estabelecido pela IEC (1997), gerando a notação textual do modelo construído.

## 1.1 PROBLEMA

O ensino de lógica fuzzy nos cursos superiores, em especial naqueles ligados à computação, passa por algumas dificuldades. Geralmente este assunto é incorporado dentro de uma disciplina específica, conseqüentemente o tempo para o estudo desses conteúdos é

limitado. Neste contexto, os alunos não aplicam de forma prática os conceitos estudados, o que torna o aprendizado limitado. Os alunos aprendem apenas a base da lógica fuzzy, a estrutura de sistemas nebulosos e como são realizados os cálculos para uma melhor representação dos dados imprecisos. A prática de sistemas difusos acaba sendo suprimida do programa.

Muitos sistemas são encontrados no mercado e na Internet com a proposta de modelagem de sistemas fuzzy. Alguns desses sistemas possuem algumas deficiências com relação às funcionalidades apresentadas, ao passo que são difíceis de operar, características que dificultam a prática das teorias estudadas ou o aprendizado no tema. Em suma, as ferramentas presentes hoje com tal proposta não são aplicáveis em meio acadêmico de forma satisfatória. Diante desta situação, a pergunta chave para o problema pode ser apresentada da seguinte forma:

Uma ferramenta de modelagem de sistemas fuzzy voltada para o meio acadêmico, intuitiva e fácil de usar, com possibilidade de geração de código de programação e ambiente de simulação facilitaria o aprendizado em lógica e sistemas nebulosos, uma vez que os conceitos podem ser colocados em prática?

## 1.2 OBJETIVOS

### 1.2.1 Objetivo geral

Produzir um ambiente para a modelagem e simulação de sistemas inteligentes que utilizem lógica difusa.

### 1.2.2 Objetivos específicos

- Desenvolver um ambiente Web que permita a modelagem de sistemas fuzzy;
- Desenvolver um simulador que execute os modelos dos sistemas fuzzy;
- Implementar, a partir do modelo de sistema fuzzy, a geração de código em FCL e Java.

### 1.3 JUSTIFICATIVA

Apesar da difusão de sistemas fuzzy, sua lógica muitas vezes não é utilizada pela sua complexidade em comparação aos demais modelos matemáticos clássicos. Além disso, a falta de ferramentas com capacidade de auxiliar, desde o momento de modelagem até a codificação de modelos nebulosos inibe desenvolvedores nas suas decisões.

A falta de conhecimento em raciocínio nebuloso é uma questão decorrente não apenas do fato de não ser um assunto trivial em computação, mas também pela maneira como é ensinada nas universidades. Os estudantes passam por este conteúdo de maneira rápida, não absorvendo os conceitos de forma concreta nem os aplicando na prática.

A ferramenta tratada como proposta de desenvolvimento no decorrer deste trabalho busca suprir essas necessidades. Em outras palavras, a mesma propõe sua aplicação em meio acadêmico, como auxílio na aprendizagem de sistemas fuzzy. Além disso, esta ferramenta poderá ser utilizada para a prática dos conceitos de raciocínio difuso, tanto para as disciplinas da computação como também em demais áreas. Para isso, algumas funcionalidades têm de ser contempladas, como a construção dos componentes de um sistema difuso, simulação do modelo desenvolvido e geração de gráficos da simulação.

Para auxiliar na aprendizagem de sistemas fuzzy, a ferramenta se baseará em gráficos e formas, representando as entradas e saídas e todos os componentes de um sistema nebuloso, bem como em tabelas a serem utilizadas pela máquina de inferência, buscando dar entendimento ao usuário acerca da estrutura de cada componente modelado.

Para a aplicação prática em cursos de computação, a ferramenta gerará códigos em linguagem de programação a partir de uma modelagem fuzzy, este código poderá ser aproveitado para a máquina difusa de um sistema a ser desenvolvido. Através dessa ferramenta, portanto, se pretende facilitar a implementação de um sistema dotado de lógica nebulosa, possibilitando sua prática durante o curso e melhorando a aprendizagem.

Para a aplicação prática em demais áreas do conhecimento, visto que lógica fuzzy pode ser aplicada em muitas áreas, a ferramenta apresentará um simulador, com opções gráficas e tabulares que permitirão a utilização do sistema fuzzy modelado. Depois de realizada a modelagem o aluno poderá testar a lógica desenvolvida de forma visual, através do sistema gerado.

Algumas ferramentas pesquisadas realizam a tarefa de modelagem de sistemas nebulosos. Porém, além de não serem voltadas para meio acadêmico, muitas não dispõem de

geração de código e seu uso é complexo. Outros fatores limitantes inviabilizam a utilização destas ferramentas em universidades, fatos como a dificuldade de acesso, por serem muitas vezes ferramentas pagas e seu uso submetido a licenças. Além disso, muitas soluções são limitadas a uma plataforma de utilização, softwares voltados ao sistema operacional Windows ou que necessitam de emuladores e máquinas virtuais para execução em outras plataformas. Também pode ser citada a falta de uma interface gráfica amigável e a operação dirigida apenas através de comandos textuais.

Essa gama de sistemas incompletos ou limitados resulta na necessidade de utilização de uma maior quantidade de softwares na produção de um projeto, o que desencoraja professores na utilização em sala de aula. A necessidade é, portanto, que haja uma ferramenta disponível a todos, alunos e professores. É necessário que a ferramenta seja fácil de utilizar e simples, que proponha ao usuário aprendizado, ao passo que este possa modelar a lógica de sua aplicação, gerando código de programação para o desenvolvimento de sistemas e um ambiente de simulação para testes e a validação da lógica modelada, proporcionando aprendizagem facilitada dos assuntos inerentes à fuzzy.

#### 1.4 METODOLOGIA

Nesta seção serão abordadas as questões referentes à metodologia utilizada para o desenvolvimento da pesquisa do projeto sob determinados aspectos. Segundo Wainer (2007), a pesquisa, quando incorporada nas áreas da ciência da computação, assume algumas características importantes. Ela geralmente envolve a construção de um software, algoritmo ou modelo de sistema e a novidade é algo fundamental no projeto.

Em muitos casos, um sistema novo desenvolvido pode ser considerado por si só uma pesquisa. Nos demais, para que o projeto possa ser considerado como pesquisa, o sistema deve passar por uma revisão ou avaliação, buscando encontrar algum conhecimento do produto final, obtido de forma metodológica. A avaliação consiste em duas fases: a verificação, onde o sistema é julgado na questão da aderência com sua especificação, e a validação, que consiste em medir a capacidade do sistema em resolver os problemas para os quais ele foi concebido (TROCHIM, 2006; WAINER, 2007).

Por conta destes fatores, o presente trabalho tratou do desenvolvimento da ferramenta proposta e, ao final do projeto, submeteu-se a uma avaliação de qualidade, a qual se preocupou em analisar a concordância entre especificação e software e a eficácia na resolução

dos problemas propostos. No tangente à classificação da pesquisa do ponto de vista da abordagem, foi utilizada uma pesquisa qualitativa como método de coleta de dados inerentes ao projeto.

De acordo com Wainer (2007), a pesquisa qualitativa preocupa-se em observar os ambientes nos quais o sistema será utilizado e os usuários envolvidos. A principal diferença em relação à pesquisa quantitativa configura-se no fato de que as variáveis a serem estudadas não podem ser medidas, mas sim observadas (MYERS, 1997). Em outras palavras, essa classificação de pesquisa realiza um estudo aprofundado do software no ambiente onde está sendo utilizado, com base nas pessoas que o utilizam (STRAUB et al., 2005; OLIVEIRA NETTO; MELO, 2008).

Quanto ao objetivo, esta mesma pesquisa classifica-se como exploratória, ou também conhecida por observacional positivista. De acordo com Wainer (2007), a pesquisa exploratória tenta seguir os fundamentos da pesquisa qualitativa, ou seja, considera que no seu domínio existam variáveis objetivas que, embora não possam ser mensuradas, podem ser observadas sob essa perspectiva (objetiva).

No início da elaboração do presente trabalho, foram pesquisados seus trabalhos correlatos. Através de análises e testes realizados nas ferramentas encontradas através de pesquisas na Internet, foram escolhidas três soluções para um estudo mais aprofundado. Através do uso das ferramentas Matlab Fuzzy Toolbox, InFuzzy e FuzzyGen foram definidas as funcionalidades básicas do sistema, bem como seus requisitos funcionais. Observando as deficiências e lacunas deixadas pelas soluções do mercado, foram apontados os diferenciais buscados no trabalho.

O seguinte passo foi a especificação do software, isto é, a construção dos diagramas e definição da estrutura do sistema. Para a elaboração da especificação foram considerados os padrões definidos pela UML para projetos de software. A implementação do software se deu acompanhada de uma pesquisa extensa nas tecnologias e padrões utilizados para o desenvolvimento. Após a implementação foram realizados testes em todas as funcionalidades do sistema, buscando por erros na implementação.

A experimentação do software produzido se deu em duas etapas. Na primeira a ferramenta foi utilizada por dois pesquisadores, de modo a comprovar a aderência da solução à sua especificação. Em um segundo momento o software foi submetido ao uso por uma turma de alunos, com os quais foi aplicado um questionário com perguntas objetivas. Este questionário avaliou a capacidade da ferramenta em resolver os problemas propostos e a

eficácia no cumprimento de seus objetivos.

## 1.5 ESTRUTURA

O presente trabalho divide-se em quatro seções. A seção 2 apresenta os conceitos e teorias inerentes a fuzzy que serviram de base para o desenvolvimento do trabalho (seção 2.1). Ainda nesta seção são levantados os trabalhos correlatos, ou seja, ferramentas similares disponíveis no mercado, detalhando as funcionalidades apresentadas por cada uma delas (seção 2.2).

A seção 3 apresenta o desenvolvimento da ferramenta. Parte-se de sua concepção (seção 3.1), especificação e diagramas (seções 3.2 a 3.6), detalhamento da implementação, técnicas e tecnologias utilizadas (seção 3.7) e operacionalização (seção 3.8). Conclui-se esta etapa com as seções de validação da ferramenta e discussão dos resultados (seções 3.9 e 3.10). A seção 4 destina-se às conclusões e trabalhos futuros (seção 4.1).



## 2 FUNDAMENTAÇÃO TEÓRICA

### 2.1 LÓGICA FUZZY

De acordo com Oliveira Junior et al. (2007), a lógica é a ciência cujo objetivo configura-se em estudar as leis do raciocínio. Nessa mesma linha, a lógica nebulosa preocupa-se em analisar os princípios formais do raciocínio aproximado. A ciência conhecida por lógica pode ser dividida em duas linhas de raciocínio. A primeira delas é conhecida por lógica clássica, também chamada aristotélica. Caracteriza-se pela sua exatidão e é composta por teorias matemáticas discretas. Analogamente, a segunda linha consiste na lógica fuzzy, que surgiu em 1965 através de Lotfi Asker Zadeh (ZADEH, 1965), como forma de raciocínio alternativa, pois propunha uma nova ótica na representação de um dado ou estado, preocupando-se em tratar situações vagas e imprecisas.

#### 2.1.1 Bivalência x polivalência

A bivalência consiste na utilização de dois valores possíveis para determinada proposição. Uma variável qualquer é verdadeira ou não-verdadeira. A ciência da computação utiliza-se fortemente da lógica bivalente, por meio da álgebra booleana, como ferramenta para embarcar as leis da verdade do mundo real na informática. Esta forma de pensar é comumente encontrada no cotidiano do ser humano, um bom exemplo consiste em uma lâmpada qualquer. Esta mesma encontra-se acesa, ou apagada. De maneira similar, o raciocínio aristotélico (clássico) baseia-se em dois valores para definir o estado de um elemento, podendo estes valores ser representados por ligado/desligado, um/zero, verdadeiro/falso, entre outros (SIMÕES; SHAW, 2007).

Coppin (2010) aborda um fato peculiar. Muitas vezes determinada situação não deixa claro ser verdadeira ou falsa. Nestes casos, algumas estratégias foram utilizadas para a representação dessas situações. A primeira delas consiste no uso das teorias da probabilidade, que buscam expressar qual a plausibilidade de determinada variável ser ou tornar-se verdadeira. Outra lógica neste sentido foi desenvolvida para operar sobre o Princípio da Incerteza da física. Nela, determinada proposição poderia assumir três possíveis valores: verdadeiro, falso ou indeterminado. Nasceram assim os primeiros estudos no tratamento da incerteza e imprecisão dos dados.

Retornando ao exemplo da lâmpada, na qual foi utilizada uma representação bivalente (ligada ou desligada), pode haver necessidade de uma representação diferenciada do seu estado. Supondo que a lâmpada possua regulagem da luminosidade por ela emitida, neste momento estão presentes graus de intensidade relacionados à luz, não apenas o fato de estar ou não acesa. É preciso elaborar uma representação que englobe todos estes estágios de iluminação, o fato passa a caracterizar-se como impreciso e esta vagueza precisa ser levada em consideração.

Uma extensão destas formas de raciocínio impreciso foi utilizar um conjunto de valores representativos acerca do estado de determinado dado. Utilizando 0 para falso, 1 para verdadeiro e valores no intervalo  $[0,1]$  representa-se o grau de verdade de determinado dado a um conjunto específico (ZADEH, 1965). Desenvolve-se, assim, a lógica e os conjuntos nebulosos. Neste contexto um elemento  $x$ , quando relacionado a um conjunto  $C$ , pode possuir um grau de pertinência no valor hipotético de 0,6. Isso representa que o elemento  $x$  pertence ao conjunto  $C$  em um grau de 60%. O que se difere da teoria da probabilidade, que aplicada neste caso, afirmaria que o elemento  $x$  tem uma probabilidade igual a 60% de pertencer ao conjunto  $C$ . Porém, neste último caso e análogo à teoria difusa, o elemento é caracterizado como apenas verdadeiro, ou apenas falso (COPPIN, 2010).

Na lógica polivalente baseada no raciocínio fuzzy, ou raciocínio nebuloso, o elemento  $x$  pertence ao conjunto  $C$  com um grau de pertinência  $y$ . O que afirma que não o pertence, na base de  $1-y$ . Ou ainda, o mesmo elemento  $x$  pode pertencer a outros conjuntos, em variados graus de pertinência, inclusive ao complementar do conjunto  $C$  (ou seja, ao  $\neg C$  ou  $C^c$ ).

### **2.1.2 Representação nebulosa através de conjuntos**

A lógica fuzzy utiliza-se da teoria dos conjuntos adaptada ao tratamento de dados imprecisos. Para isso, são utilizadas variáveis linguísticas, representando conjuntos, subconjuntos ou valores. Uma variável linguística baseada no mundo real pode ser a velocidade de um veículo, o peso de uma pessoa ou objeto, ou ainda a temperatura de determinado ambiente. Neste último exemplo, esta variável pode conter alguns conjuntos ligados a ela, como gelado, frio, morno e quente. Cada conjunto da variável linguística é tratado pelo sistema fuzzy como um termo linguístico associado à mesma. Essa linguística utilizada pelos sistemas nebulosos é responsável por transpor situações do mundo real em linguagem matemática, compreendida e tratada por computadores. Dessa forma, a

representação de determinada situação torna-se mais próxima do real, utilizando conceitos difusos (ZADEH, 1965).

Na teoria clássica dos conjuntos todo o conjunto possui algum subconjunto. De acordo com Oliveira Junior et al. (2007), quando se utiliza um subconjunto  $S$  de determinado conjunto  $C$  como um mapeamento dos elementos pertencentes a  $C$ , utilizam-se pares ordenados para representar esta pertinência. Dessa forma, um par ordenado é composto pelo elemento em questão, mais zero ou um. Ou seja, a representação  $\{x, 1\}$  implica no elemento  $x$  pertencente ao conjunto em questão, da mesma forma a representação  $\{x, 0\}$  implica na sua não pertinência.

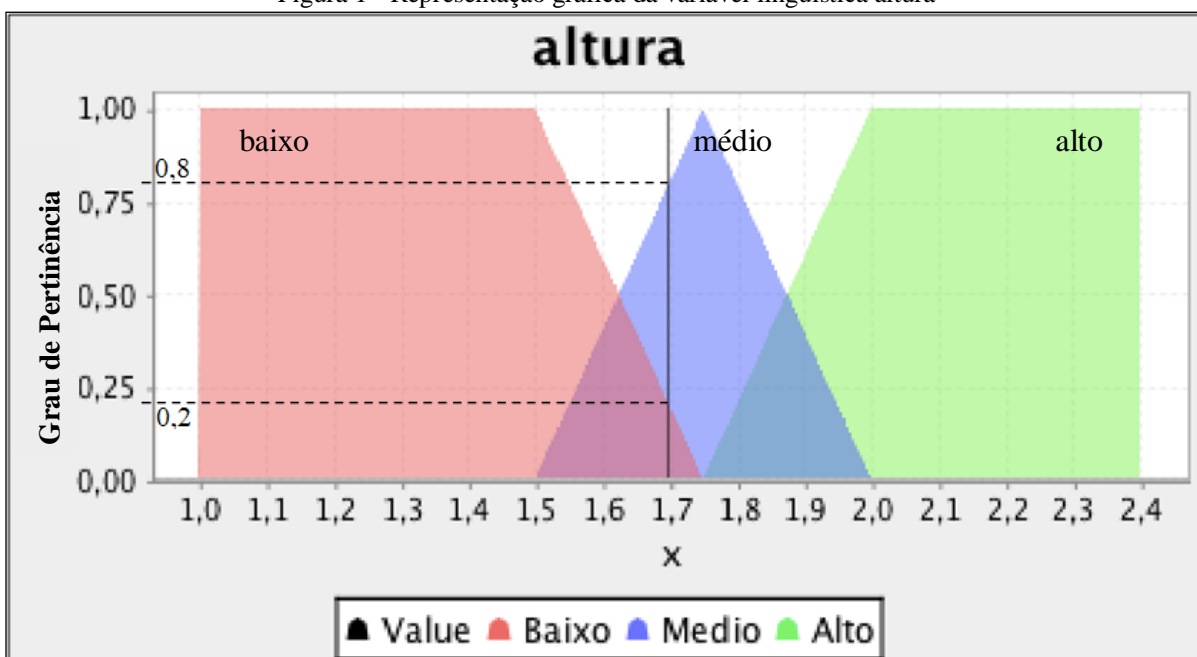
Aplicando estes conceitos matemáticos ao mundo real, pode-se utilizar o conceito de conjuntos utilizando situações do cotidiano. Conforme exemplifica Coppin (2010), o conjunto altura de pessoas representa claramente como a teoria exata dos conjuntos pode vir a caracterizar-se como imprecisa. Uma pessoa com 2,10m de altura certamente seria alocada ao conjunto de pessoas altas, enquanto uma pessoa com 1,20m seria alocada ao conjunto de pessoas baixas. Conquanto, uma pessoa de 1,78m de altura seria alocada ao conjunto de pessoas altas por muitos, como também ao conjunto de baixos por outros tantos analistas. Este fato caracteriza-se como vago e uma representação condizente se faz necessária. O ideal é que este dado específico utilize os conceitos da lógica polivalente: o raciocínio nebuloso. O indivíduo poderia caracterizar-se tanto como baixo, quanto como alto, em seus determinados graus de pertinência. Dessa forma, a variável linguística “altura da pessoa”, de acordo com seu valor (no caso exemplificado 1,78m) pertenceria aos três conjuntos (baixo, médio e alto) simultaneamente, representando com maior acurácia a forma como seres humanos categorizam valores e aspectos do mundo real. Essa representação é possível graças à utilização da teoria matemática dos conjuntos aplicada ao raciocínio nebuloso, conhecido por conjuntos fuzzy.

Conjuntos nebulosos são representados de maneira similar aos conjuntos matemáticos conhecidos (também chamados por conjuntos nítidos ou *crisp*), porém o segundo elemento da representação pode ser qualquer número real pertencente ao intervalo  $[0, 1]$ . Neste contexto, o segundo elemento representa o grau de pertinência daquele elemento ao conjunto em questão. Nesta linha, a representação  $\{y, 0.4\}$  implica que  $y$  pertence ao conjunto em questão a um grau de 0.4 (40% de pertinência). A utilização do número 1 como segundo elemento da representação implica na sua total pertinência, enquanto o 0 representa a total não pertinência (OLIVEIRA JUNIOR et al., 2007).

### 2.1.2.1 Representação gráfica dos conjuntos nebulosos

Uma forma de representar estes conjuntos nebulosos até então tratados é composta por pares ordenados, onde são informados o elemento e seu respectivo grau de pertinência ao conjunto. Uma maneira de representação mais usual e visível é composta por gráficos. Coppin (2010) baseia-se na ideia de um conjunto nebuloso baseado na altura de uma pessoa, indicando a quais grupos a mesma pertence: alto, médio e baixo. Uma pessoa com 1,70m de altura pode ser relacionada nos três grupos, cada qual com sua pertinência. A situação explicitada pode ser graficamente representada conforme a Figura 1.

Figura 1 - Representação gráfica da variável linguística altura



Fonte: Adaptado de Coppin (2010).

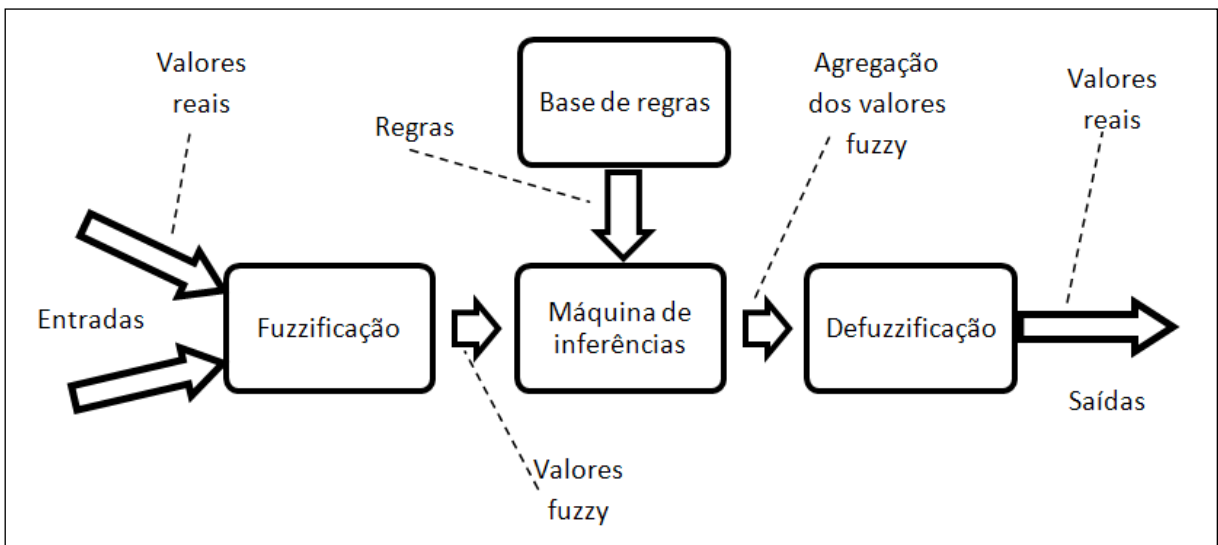
Exemplificando, a altura no valor de 1,70m pode ser analisada graficamente como pertencente aos três conjuntos em questão. Uma pessoa, portanto, com 1,70m de altura possui grau de pertinência 0 ao conjunto alto e pertinência aproximada de 0,2 e 0,8 aos conjuntos baixo e médio, respectivamente.

## 2.2 SISTEMAS FUZZY

Um sistema fuzzy, como a maioria dos sistemas inteligentes, tem como objetivo modelar ações de acordo com o conhecimento de um especialista. Ou seja, sistemas devem ser capazes de adotar comportamentos humanos na resolução dos problemas propostos (RUSSELL E NORVIG, 1995). Computadores possuem a característica precisa expressa pela lógica aristotélica. Assim sendo, para que um sistema computacional adote comportamento humano nas suas tarefas é necessário implementar os conceitos da lógica nebulosa nessas máquinas. Dessa forma, é possível prover ao computador a mesma base de raciocínio imprecisa de um ser humano, o que é possibilitado pelos sistemas fuzzy (ROSS, 2004).

Conforme apresentam Simões e Shaw (2007), um sistema fuzzy opera em três momentos distintos. No primeiro deles ocorre a transformação dos valores do mundo real para o domínio fuzzy. Em posse dos valores fuzzy o sistema realiza uma série de inferências, que faz uso da base de regras para a tomada de decisões. Por fim, o sistema transforma os valores fuzzy de saída gerados pela inferência em valores reais. Sintetizando as ideias até então apresentadas, a arquitetura de um sistema fuzzy consiste em alguns elementos básicos, os quais estão dispostos na Figura 2.

Figura 2 – Arquitetura de um sistema fuzzy



Fonte: Adaptado de Simões e Shaw (2007)

### 2.2.1 Arquitetura de um sistema fuzzy

Para o funcionamento de um sistema difuso é necessário que o sistema implemente quatro principais blocos funcionais. O primeiro trata-se de uma interface de fuzzificação, cujo objetivo reside em transformar valores reais de entrada, em valores fuzzy para alimentação das fases posteriores. O segundo bloco é composto pela base de conhecimento, onde estão especificadas todas as regras do sistema. O terceiro bloco é representado pelo motor de inferências. O motor de inferências coleta as entradas já fuzzificadas e utiliza-se das regras definidas para inferir as saídas fuzzy. Este bloco ainda pode ser definido como a lógica de tomada de decisões do sistema difuso. Por fim, a interface de defuzzificação realiza o processo inverso à fuzzificação, ou seja, transforma valores fuzzy em valores reais (SIMÕES; SHAW, 2007).

Ainda de acordo com Simões e Shaw (2007), sistemas e controladores fuzzy podem assumir três diferentes estruturas. A mais comum dentre elas são os sistemas fuzzy baseados em regras. Além dessa estrutura ainda existem os controladores fuzzy paramétricos e sistemas baseados em equações relacionais.

De acordo com Oliveira Junior et al. (2007), os sistemas nebulosos baseados em regras geralmente são baseados em estruturas condicionais do tipo “se... então”. Oliveira Junior et al. (2007) ainda dividem a fase de inferência em três momentos: primeiro ocorre o cálculo dos antecedentes, utilizando os conectivos lógicos do tipo “E” e “OU”. Após isso são calculadas as implicações de cada regra satisfeita na fase anterior, utilizando os operadores de “mínimo” ou “produto”. Por fim, os consequentes das regras satisfeitas são agregados pelos operadores de “máximo” ou “soma limitada”.

Dada uma base de regras (conjunto de regras fuzzy) representando determinado sistema e um vetor de entradas definido no conjunto dos números reais, pode-se definir inferência fuzzy como o processo pelo qual se obtêm as saídas do sistema, pela avaliação dos níveis de compatibilidade das entradas com os antecedentes das regras, ativando os consequentes. Tem-se então um conjunto fuzzy que é transformado em um número real (OLIVEIRA JUNIOR et al., 2007, p. 46).

De acordo com Simões e Shaw (2007), os sistemas paramétricos buscam combinar uma descrição global baseada em regras com aproximações lineares locais. Para isso é utilizado um modelo de regressão linear, o qual corresponde a um modelo linear de entrada e saída descrevendo o sistema. De maneira geral, essa estrutura pode ser considerada híbrida, pois faz uso de regras nas quais seus antecedentes são compostos pelas entradas fuzzificadas. Em contrapartida, seus consequentes são definidos como aproximações lineares de saída.

Os sistemas fuzzy baseados em equações relacionais adotam a utilização de um conjunto de equações relacionais fuzzy. O processo de obtenção das entradas do sistemas ocorre pelo recebimento de sinais, os quais são interpretados e construídos gradualmente, recalculados a cada iteração no sistema. Essa característica possibilita o aprendizado do sistema, uma vez que os sinais são recalculados e aproximados a cada entrada no sistema (SIMÕES; SHAW, 2007).

Com base na estrutura definida para sistemas nebulosos, dois modelos se desenvolveram, implementando as especificações determinadas e os conceitos da lógica fuzzy. O primeiro deles implementa o mecanismo de inferência fuzzy para sistemas nebulosos baseados em regras, ao passo que o segundo utiliza os conceitos de sistemas paramétricos de lógica fuzzy, são eles:

a) Modelo de Mamdani

A regra típica desse modelo consiste em: “se  $x$  é  $A$  e  $y$  é  $B$  (em que  $A$  e  $B$  são conjuntos fuzzy), então  $z$  é  $C$  (onde  $C$  também é um conjunto fuzzy)”. Neste modelo, o processo de defuzzificação obtém um resultado não-fuzzy na saída da inferência. A sua principal característica, como a função propõe, é o fato das relações difusas ocorrerem no antecedente e no consequente. Ou seja, a saída é um conjunto difuso (OLIVEIRA JUNIOR et al., 2007).

b) Modelo de Takagi-Sugeno

A regra típica para este modelo consiste em: “se  $x$  é  $A$  e  $y$  é  $B$  (em que  $A$  e  $B$  são conjuntos fuzzy), então  $z$  é  $f(x,y)$ ”. O que difere este modelo do anterior é no tratamento do consequente, que se utiliza de equações paramétricas ao invés de relações difusas. Ou seja, a saída é um valor ou função (OLIVEIRA JUNIOR et al., 2007).

## 2.2.2 Fuzzificação e as funções de pertinência

O primeiro bloco funcional básico para qualquer sistema difuso, seja ele baseado em regras, paramétrico ou baseado em equações relacionais é a fuzzificação (OLIVEIRA JUNIOR et al., 2007; SIMÕES; SHAW, 2007). De maneira geral, a fuzzificação consiste em

transformar valores numéricos reais em grandezas difusas a serem utilizadas pelo sistema (JANTZEN, 2007).

Gráficos são largamente utilizados para representação de conjuntos fuzzy para determinada variável a ser tratada. Um valor *crisp* pode então ser analisado nas formas de pares ordenados e de gráficos, obtendo-se o grau de pertinência a determinado conjunto. Todas estas formas de análise são resultado das funções de pertinência. Essas funções, quando aplicadas ao valor *crisp* determinado, resultam no seu valor fuzzy correspondente (COPPIN, 2010).

Funções de pertinência fuzzy representam os aspectos fundamentais de todas as ações teóricas e práticas de sistemas fuzzy. Uma função de pertinência é uma função numérica gráfica ou tabulada que atribui valores de pertinência fuzzy para valores discretos de uma variável, em seu universo de discurso (SIMÕES; SHAW, 2007, p. 46).

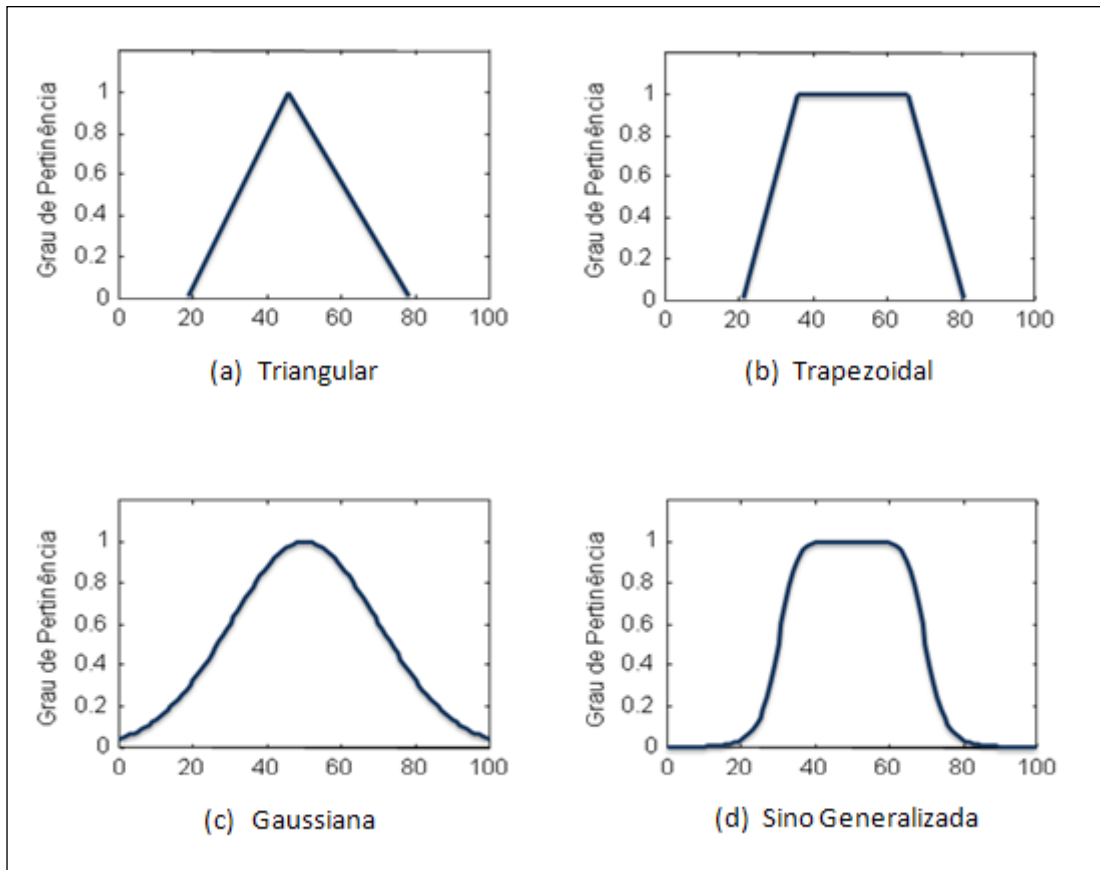
Funções de pertinência em conjuntos fuzzy são aplicadas, portanto, sobre uma variável qualquer, dentro do conjunto de todos os valores reais que esta pode assumir. Assim sendo, um valor *crisp* submetido a uma função de pertinência fuzzy passa a possuir um valor nebuloso de pertinência ao conjunto em questão. Esse processo de determinação de um valor difuso para uma variável valorada é conhecido por fuzzificação (SIMÕES; SHAW, 2007).

As funções de pertinência fuzzy estão diretamente relacionadas à forma gráfica que determinado conjunto assume. Simões e Shaw (2007) ressaltam que a quantidade de funções de um universo de discurso, bem como seu formato, são determinados com base na experiência no domínio do problema. Um número comumente utilizado de conjuntos fuzzy linguísticos, ou funções de pertinência gira em torno de 2 a 7. Quanto maior o número de conjuntos, mais precisa a representação, porém maior capacidade computacional é demandada.

Essa análise realizada durante a modelagem da etapa de fuzzificação define o formato gráfico a ser utilizado pelo conjunto fuzzy a ser representado. Esta representação geralmente segue os formatos triangulares, trapezoidais, gaussianos e sino generalizada. Uma representação mais suave pode ser atingida utilizando-se funções do tipo  $\cos^2(x)$ , *sigmóide* e *spline cúbico*, o que demanda maiores recursos computacionais, em consequência da maior complexidade na resolução dos cálculos (SIMÕES; SHAW, 2007). A Figura 3 apresenta alguns formatos usuais utilizados por sistemas nebulosos.



Figura 3 – Representações gráficas das funções de pertinência usuais



Fonte: Adaptado de Oliveira Junior et al. (2007)

Estas representações gráficas das funções de pertinência são comumente utilizadas para definir conjuntos fuzzy linguísticos. Com base em Oliveira Junior et al. (2007) e Simões e Shaw (2007), as funções de pertinência, geradoras dos gráficos são apresentadas nas equações 1 a 4. As equações 1, 2, 3 e 4 apresentam as funções de pertinência triangular, trapezoidal, gaussiana e sinoidal, respectivamente.

$$f_C(x) = \left\{ \begin{array}{l} 0 \dots\dots\dots (x \leq a) \\ \frac{x-a}{b-a} \dots\dots\dots (a < x \leq b) \\ \frac{c-x}{c-b} \dots\dots\dots (b < x \leq c) \\ 0 \dots\dots\dots (x > d) \end{array} \right\} \quad \text{eq. 1}$$

$$f_C(x) = \left\{ \begin{array}{l} 0 \dots\dots\dots (x \leq a) \\ \frac{x-a}{b-a} \dots\dots\dots (a < x \leq b) \\ 1 \dots\dots\dots (b < x \leq c) \\ \frac{d-x}{d-c} \dots\dots\dots (c < x \leq d) \\ 0 \dots\dots\dots (x > d) \end{array} \right\} \quad \text{eq. 2}$$

$$f_C(x) = \begin{cases} 0 & \text{se } x \text{ está fora do domínio} \\ \exp(-(x-\nu)^2) / 2\sigma^2 & \end{cases} \quad \text{eq. 3}$$

$$f_C(x) = \begin{cases} \frac{1}{1 + \left| \frac{x-c}{b} \right|^{2b}} & \end{cases} \quad \text{eq. 4}$$

É possível incorporar aos sistemas nebulosos a precisão da lógica clássica aristotélica, por meio de uma função de pertinência específica para essa representação. Esta função é denominada discreta, também conhecida por *singleton* (ESPINOSA et al., 2005). A função de pertinência discreta é composta por um único valor no domínio de valores possíveis para a variável de entrada. Caso a entrada real seja igual ao valor definido como *singleton*, a variável possui total pertinência ao conjunto em questão. Sendo o valor da variável de entrada diferente do valor *singleton*, a variável possui total não pertinência ao conjunto em questão (PASSINO; YURKOVICH, 1998).

Uma alternativa à fuzzificação por meio das equações conhecidas como funções de pertinência é a fuzzificação por tabela em memória, que visa diminuir o tempo de processamento computacional. Esta técnica consiste em calcular os valores fuzzy tomando por base uma tabela previamente armazenada em memória (SIMÕES; SHAW, 2007).

Simões e Shaw (2007) afirmam que o universo de discurso de uma variável a ser tratada é geralmente normalizado para um intervalo padrão quando trabalhado pelo sistema. Com isso, os valores fuzzy relacionados aos números *crisp* do conjunto podem ser pré-calculados e inseridos em uma tabela discretizada. Dessa forma, ao invés de realizar os

cálculos através das funções de pertinência toda vez que desejar-se o valor fuzzy correspondente, basta realizar uma consulta nessa tabela, fazendo interpolações quando necessário. Estudos apontam que a utilização das tabelas de fuzzificação apresenta-se eficaz no processo de determinação dos valores fuzzy para o domínio de um conjunto linguístico (SIMÕES; SHAW, 2007).

### **2.2.3 Máquina de inferência**

Depois de realizada a fuzzificação das variáveis do sistema, o resultado é utilizado pela máquina de inferência para realizar o processamento dessas informações. Nesta etapa é utilizada a base de conhecimento, onde se encontram as regras nas quais o sistema se baseia para a realização das ponderações nebulosas. Conforme Simões e Shaw (2007), a base de conhecimento representa o modelo do sistema a ser controlado, enquanto o motor de inferência (também tratado como inferência fuzzy) reflete a lógica implementada pelo sistema difuso.

Simões e Shaw (2007) abordam a estrutura dessa base de conhecimento, composta por uma base de dados, como sendo a junção das funções de pertinência associadas às variáveis linguísticas e da base de regras nebulosas. Essa base de regras é elaborada por um especialista na área do domínio, o qual se utilizará de testes e aferições, juntamente com sua experiência para determinação das regras de controle do sistema.

#### **2.2.3.1 Estrutura da base de regras**

De acordo com Kovacic e Bogdan (2006), a base de regras de um sistema difuso é seu componente central e representa a inteligência do controlador fuzzy. Neste momento se faz necessário que o conhecimento e experiência de um especialista sejam corretamente expressos e transformados em um conjunto apropriado de regras. A definição da base de regras é a fase mais importante da modelagem de um sistema ou controlador difuso. Este bloco funcional caracteriza-se como o mais importante de todo o sistema, ao passo que seus demais componentes representam apenas um serviço para a base de regras. Em outras palavras, o resultado do sistema fuzzy está influenciado sobretudo pela qualidade de suas regras e o quão representam o mundo real.

As regras em um sistema fuzzy seguem uma estrutura definida. Esta é dividida em duas partes, denominadas antecedente e consequente. Conforme Kovacic e Bogdan (2006), os antecedentes representam causas, enquanto os consequentes expressam consequências. Estas últimas, por sua vez, resultarão em ações do sistema difuso. Dessa forma, uma regra pode ser descrita na forma “SE <antecedente> ENTÃO <consequente>” (ROSS, 2004).

A formação da base de regras de um sistema nebuloso deve procurar manter algumas características básicas. A primeira delas busca garantir a consistência das regras, evitando sua contradição. Em suma, duas regras não podem possuir o mesmo antecedente resultando em distintos consequentes. A segunda premissa trata da completude da base de regras fuzzy, afirmando que a mesma é completa se, para cada relação entre variável de entrada e termo linguístico associado existe uma regra na qual seu antecedente contenha tal relação (KOVACIC; BOGDAN, 2006).

Tanto o antecedente quanto o consequente de uma regra podem ser compostos por mais de uma condição. Nestes casos, é necessário um conectivo lógico que relacione as condições envolvidas na determinada premissa. No caso de consequentes encadeados em mais de uma condição, o conectivo lógico é necessariamente o operador “E”. Ou seja, uma vez satisfeita a regra, todas as ações modeladas pela regra serão executadas. O mesmo não ocorre na parte antecedente da regra. Neste caso, além do operador “E”, que exige que todas as premissas sejam satisfeitas para que a determinada regra seja ativada, é utilizado o operador “OU”, no qual basta que uma das premissas a ele associada seja válida para que a regra seja ativada (ROSS, 2004; SIMÕES; SHAW, 2007).

#### 2.2.3.2 Inferência fuzzy

De maneira geral, os mecanismos de inferência de um sistema difuso incorporam outro aspecto importante para prover ao sistema inteligência: a lógica de tomada de decisões. Estas utilizam implicações fuzzy para simular um humano na decisão de quais ações executar. Em suma, o motor de inferência de um sistema fuzzy gera ações de controle (variáveis de saída), inferindo-as a partir de suas variáveis de entrada, com base nas regras estabelecidas (SIMÕES; SHAW, 2007).

Conforme Rezende (2005), no mecanismo de inferência são definidas e executadas as formas como serão processados os antecedentes das regras, quais serão os indicadores de ativação de regras satisfeitas e os operadores utilizados sobre os conjuntos difusos. A

inferência fuzzy também é responsável por implementar parte importante dos modelos de sistemas fuzzy, como Mamdani e Takagi-Sugeno.

Weber e Klein (2003), em sua abordagem sobre os modelos de inferência em sistemas fuzzy ressaltam o fato da máquina de inferência de um sistema difuso seguir a teoria de *modus ponens* no processamento dos dados e das regras. Neste sentido, o valor real para cada premissa da base de regras é computado e aplicado ao seu consequente, resultando em um conjunto fuzzy para cada variável de saída de cada regra. A inferência fuzzy, dado este contexto, se utiliza de operadores lógicos para a ativação das regras e para a inferência (composição) das suas saídas. No caso da ativação das regras, esses operadores são conhecidos como MÍNIMO e PRODUTO, enquanto para a composição são utilizados os operadores de MÁXIMO e SOMA.

[...] usualmente somente o MIN e PRODUTO são usados como regras de Inferência. Na Inferência de MIN, o grau de pertinência de saída é cortado a uma altura que corresponde ao grau de verdade computado pela regra de premissa (lógica fuzzy AND). Na Inferência de PRODUTO, o grau de pertinência de saída tem como escala o grau de verdade computado pela regra da premissa. (WEBER; KLEIN, 2003, p. 40).

Segundo Weber e Klein (2003), na composição os conjuntos fuzzy de saída são combinados de modo a definir um conjunto difuso único para cada variável de saída. São utilizados, para este efeito, os operadores MÁXIMO ou SOMA. No caso do operador MÁXIMO o conjunto fuzzy resultante é obtido com base no ponto máximo de todos os conjuntos combinados. Já na SOMA, que neste caso representa uma soma limitada, o resultado é calculado através do ponto de soma em cima de todos os conjuntos considerados.

#### **2.2.4 Defuzzificação**

A defuzzificação, conforme Oliveira Junior et al. (2007) e Simões e Shaw (2007), é o processo de transformar o valor da variável linguística inferida pelas regras fuzzy em um valor discreto. Esta transformação traduz a saída da inferência fuzzy para o domínio discreto. Existem vários métodos para a realização da defuzzificação. Para selecionar o método apropriado, segundo Simões e Shaw (2007), pode-se utilizar um enfoque voltado ao centróide ou então basear-se nos valores máximos resultantes da função de pertinência. Os métodos comumente utilizados são: Centro-da-Área (C-o-A), Centro-do-Máximo (C-o-M) e Média-do-Máximo (M-o-M).

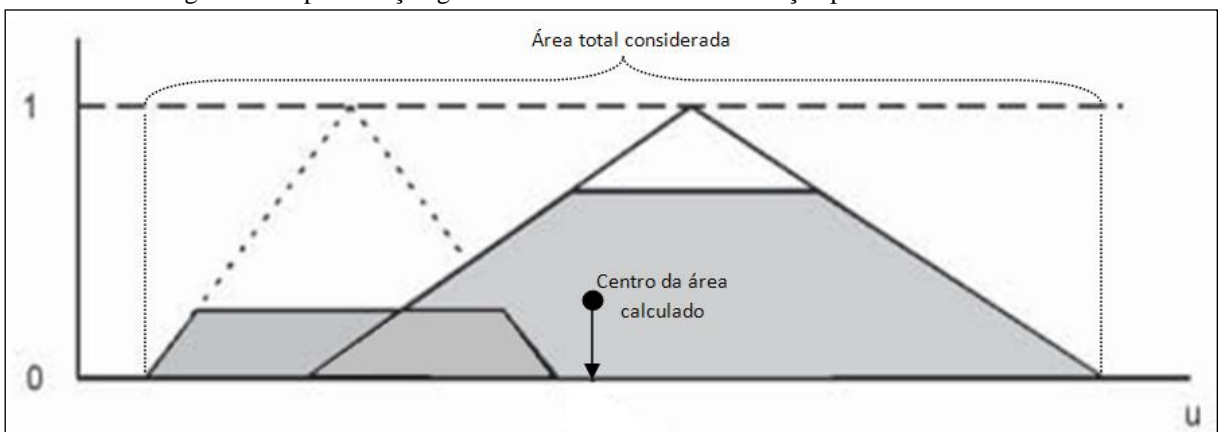
### 2.2.4.1 Defuzzificação pelo Centro-da-Área (C-o-A)

De acordo com Ross (2004), a defuzzificação pelo centro da área, ou centróide, ou ainda centro da gravidade como é conhecida, é a técnica mais utilizada para se traduzir uma grandeza fuzzy para um valor *crisp*. Ele calcula o centróide da área composta pelas saídas fuzzy. Essa área é determinada pela união de todas as contribuições de regras. Este valor de centróide calculado representa o centro que divide a área encontrada na inferência fuzzy em duas partes iguais. Este cálculo é ilustrado pela equação 5.

$$x^* = \frac{\sum_{j=1}^N x_j f_{OUT}(x_j)}{\sum_{j=1}^N f_{OUT}(x_j)} \quad \text{eq. 5}$$

$f_{OUT}(x_i)$  é a área de uma função de pertinência, ou seja, o resultado fuzzy decorrente do processo de fuzzificação e  $x_i$  é a posição do centróide da função de pertinência individual (SIMÕES; SHAW, 2007). O método de defuzzificação pelo centro da área pode ser graficamente representado conforme a Figura 4.

Figura 4 - Representação gráfica do método de defuzzificação pelo Centro-da-Área



Fonte: Reznik (1997)

Conforme explicam Simões e Shaw (2007), o método C-o-A apresenta alguns problemas no cálculo da defuzzificação. Um deles pode ser apontado nos casos em que as funções de pertinência não são sobrepostas. Outro problema decorre da situação em que mais de uma regra possui a mesma saída fuzzy, neste caso a área é contabilizada apenas uma vez, quando na verdade deveria ser considerada duas ou mais vezes, de acordo com o número de regras com a mesma saída.

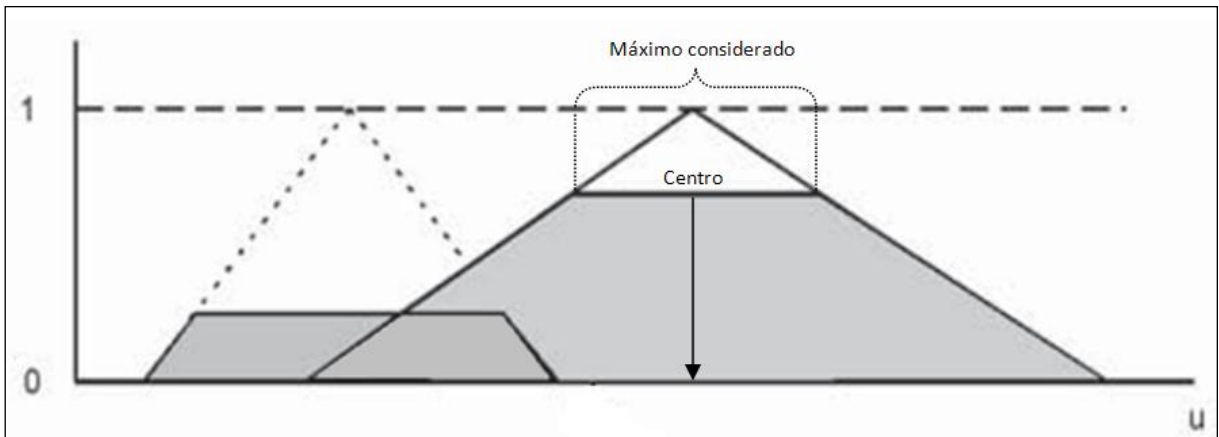
#### 2.2.4.2 Defuzzificação pelo Centro-do-Máximo (C-o-M)

Simões e Shaw (2007) explicam que nesse método os picos das funções de pertinência são considerados para fins de cálculo da defuzzificação, ignorando as áreas das funções de pertinência. Por conta disso também é chamado de método de defuzzificação pelas alturas. Os valores não nulos da fuzzificação são posicionados nos pontos de máximo (picos) e passam a representar pesos. Dessa forma, o valor discreto de saída é calculado encontrando-se o ponto de equilíbrio entre esses pesos. De maneira simplificada, a defuzzificação é calculada através da média ponderada dos máximos considerando os pesos, sendo as saídas da inferência fuzzy calculadas através da equação 6.

$$x^* = \frac{\sum_{i=1}^N x_i \sum_{k=1}^N f_{OUT}(x_i)}{\sum_{i=1}^N \sum_{j=1}^N f_{OUT}(x_i)} \quad \text{eq. 6}$$

$f_{OUT}(x_i)$  indica os pontos de máximo das funções de pertinência da inferência fuzzy (SIMÕES; SHAW, 2007). A Figura 5 apresenta a representação gráfica do método de defuzzificação pelo centro do máximo.

Figura 5 - Representação gráfica do método de defuzzificação pelo Centro-do-Máximo



Fonte: Reznik (1997)

#### 2.2.4.3 Defuzzificação pela Média-do-Máximo (M-o-M)

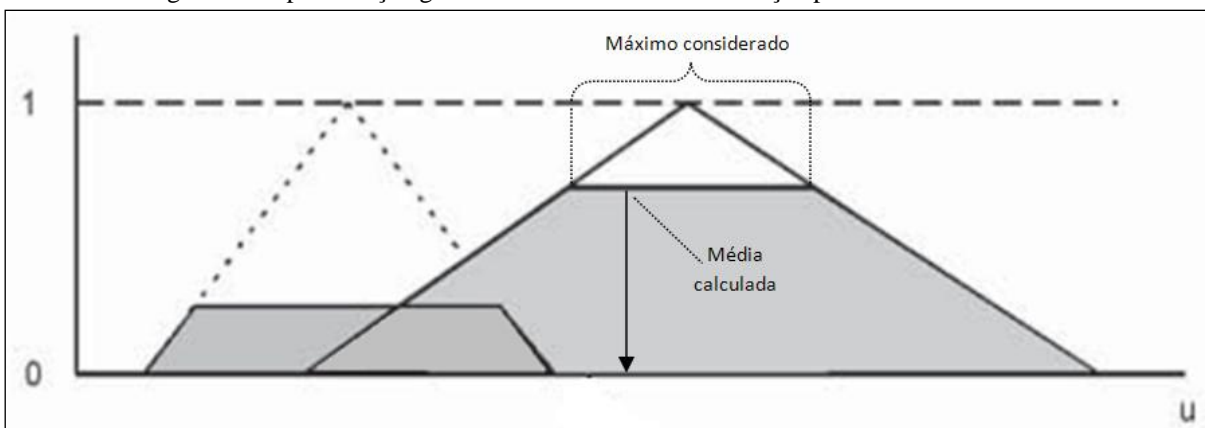
Outra abordagem para o cálculo da defuzzificação é utilizar a saída com maior valor de pertinência encontrado na inferência fuzzy. Este método logo foi desbancado, uma vez que não poderia ser utilizado em casos onde uma função de pertinência tenha mais de um máximo, o que também impossibilita o uso da abordagem C-o-M, pela necessidade de escolher qual máximo utilizar. Frente a essas situações, uma solução é calcular a média de todos esses máximos, de acordo com a equação 7.

$$x^* = \sum_{m=1}^M \frac{x_m}{M} \quad \text{eq. 7}$$

$x_m$  é o m-ésimo elemento do universo de discurso, em que a função  $f_{OUT}(x_i)$  tenha o máximo e  $M$  o total desses elementos máximos. Esta abordagem é conhecida por solução mais plausível, uma vez que desconsidera o formato das funções de pertinência de saída (SIMÕES; SHAW, 2007). A representação gráfica para o método de defuzzificação da média dos máximos é apresentada na Figura 6.



Figura 6 - Representação gráfica do método de defuzzificação pela Média-do-Máximo



Fonte: Reznik (1997)

#### 2.2.4.4 Questões pertinentes à defuzzificação

Nesta etapa de defuzzificação, Simões e Shaw (2007) abordam o fato de existirem métodos contínuos e não contínuos para o cálculo do valor *crisp* após a inferência fuzzy. Método contínuo é aquele que, dada uma variação suave nas variáveis de entrada, não resultará em nenhuma forte alteração nos valores de saída. Os métodos C-o-A e C-o-M são ditos contínuos, enquanto o método M-o-M é descontínuo. “Isso se deve ao fato de que o melhor compromisso jamais pode saltar para um valor diferente com uma pequena mudança nas entradas. Por outro lado, a solução mais plausível não é única e pode assim saltar para um valor diferente.” (SIMÕES; SHAW, 2007, p. 56).

Esta multiplicidade de fatores que diferenciam os resultados nos diferentes métodos de defuzzificação faz com que a escolha da estratégia ideal esteja atrelada ao projeto no qual será aplicada. De acordo com a aplicação do raciocínio nebuloso, determinado método de cálculo será mais apropriado e representará com maior acurácia a situação do mundo real. Cabe ao engenheiro do sistema a escolha desse método, através de cálculos, análises e testes, os quais poderão ser realizados através de uma ferramenta de auxílio na modelagem.

### 2.3 FUZZY CONTROL LANGUAGE

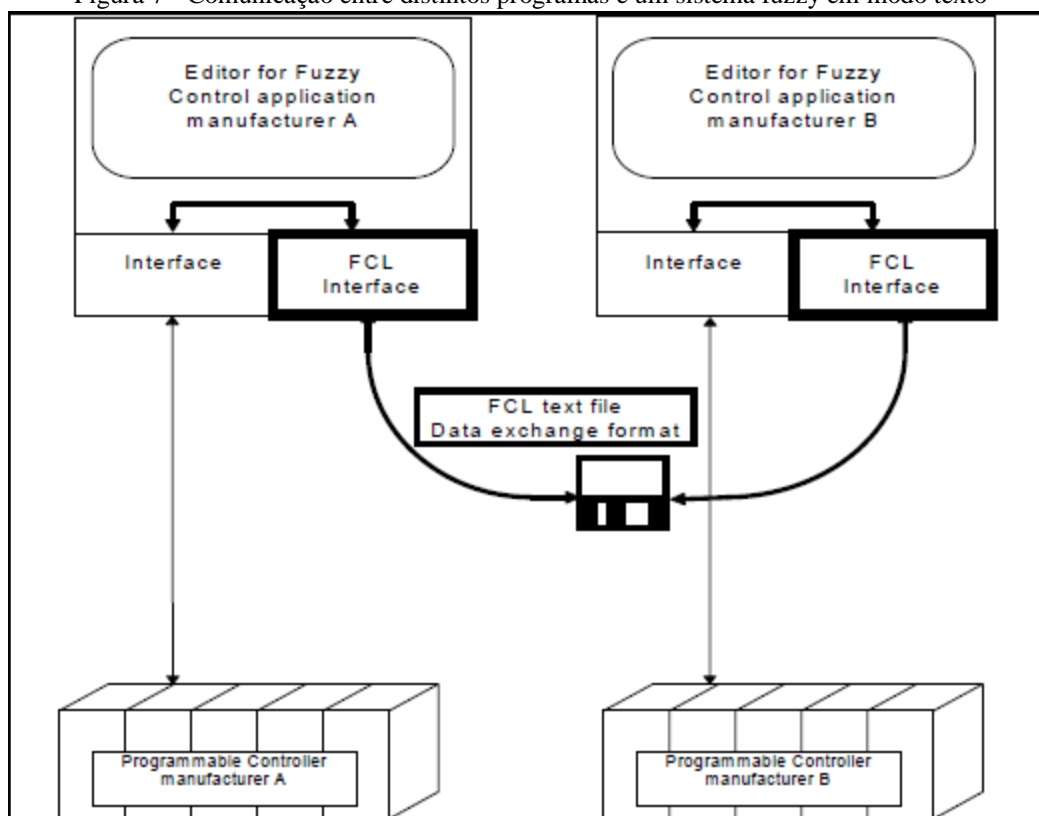
Dada a popularização dos sistemas inteligentes no desenvolvimento de aplicações e controladores industriais, pesquisadores trataram de criar um padrão para definir os conceitos e a implementação deste tipo de tecnologia. Nesta linha, a *International Electrothechnical Comission* (IEC) desenvolveu a norma IEC 1131, voltada a controladores programáveis. Para

este trabalho é importante a consideração da sua parte 7, que trata da programação de controladores fuzzy. O objetivo das definições da parte 7 da norma, de acordo com Cingolani e Alcalá-Fdez (2012), é fornecer um padrão comum na linguagem de escrita de sistemas fuzzy, de modo a permitir a portabilidade destes programas para diferentes plataformas. Seguindo as especificações da IEC 1131-7 (1997) é possível a utilização de um mesmo sistema difuso em distintas aplicações através do padrão textual sugerido (GALCERAN et al., 2001).

A especificação IEC 1131-7 (1997) se divide em quatro seções. A primeira delas traz as definições e terminologias para o padrão, isto é, a forma como serão denominados e tratados os componentes e termos relacionados a sistemas difusos. A segunda seção aborda a adaptação de controladores fuzzy dentro dos padrões de linguagem de controladores programáveis. A terceira seção, de maior importância para o presente trabalho, trata do padrão de linguagem para sistemas fuzzy e sua representação textual. Neste ponto são tratadas a sintaxe e semântica da padronização, resultando no FCL (*Fuzzy Control Language*), o qual utiliza os padrões definidos na primeira seção, como blocos de função e tipos de dados. Por fim, a última seção apresenta alguns itens de conformidade, os quais devem ser satisfeitos pelos sistemas fuzzy modelados. Esses itens, quando contemplados, garantem a escalabilidade de um sistema difuso para aplicações de distintas abrangências, desde menores até de grande porte (IEC, 1997).

Como resultado da padronização definida pela especificação IEC 1131-7 é proposta a notação textual através de um padrão denominado FCL. A definição para FCL, conforme IEC (1997) está atrelada às definições de linguagens para controladores, também publicada pelo mesmo comitê. De maneira geral, a interação entre programas que utilizem um mesmo sistema fuzzy deve ser transparente, isto é, a troca de dados ocorre de maneira independente à sua origem ou destino. Esta característica é obtida pelo padrão e estrutura definidos pela especificação do FCL. Outro objetivo com a criação do padrão é a portabilidade do sistema fuzzy entre distintas aplicações. Esses fatores são exemplificados pela Figura 7.

Figura 7 - Comunicação entre distintos programas e um sistema fuzzy em modo texto



Fonte: IEC 1131-7 (1997)

Como se pode observar, a comunicação entre os distintos programas e o sistema fuzzy ocorre de forma transparente. Basta que o sistema seja desenvolvido seguindo um padrão conhecido pelos programas que o utilizam e estes implementem suas interfaces seguindo a mesma especificação. Deste modo, a troca de dados é transparente e o sistema fuzzy torna-se portátil entre quaisquer aplicações. O FCL também se preocupa em manter uma estrutura escalável, atendendo aplicações de pequeno a grande porte.

A IEC 1131-7 (1997) também define os elementos constituintes de um sistema difuso. O Quadro 1 apresenta um exemplo de sistema fuzzy no padrão FCL.

Quadro 1 - Exemplo de sistema fuzzy no padrão FCL

```

1  FUNCTION_BLOCK Fuzzy_FB
2  VAR_INPUT
3      temp:    REAL;
4      pressure:REAL;
5  END_VAR
6  VAR_OUTPUT
7      valve:   REAL;
8  END_VAR
9  FUZZIFY temp
10     TERM cold := (3,1) (27,0);
11     TERM hot  := (3,0) (27,1);
12 END_FUZZIFY
13 FUZZIFY pressure
14     TERM low := (55,1) (95,0);
15     TERM high := (55,0) (95,1);
16 END_FUZZIFY
17 DEFUZZIFY valve
18     TERM drainage := -100;
19     TERM closed   := 0;
20     TERM inlet    := 100;
21     ACCU: MAX;
22     METHOD: COGS;
23     DEFAULT := 0;
24 END_DEFUZZIFY
25 RULEBLOCK No1
26     ACT: MIN;
27     AND: MIN;
28     RULE 1: IF temp IS cold AND pressure IS low THEN valve IS inlet;
29     RULE 2: IF temp IS cold AND pressure IS high THEN valve IS closed WITH 0.8;
30     RULE 3: IF temp IS hot AND pressure IS low THEN valve IS closed;
31     RULE 4: IF temp IS hot AND pressure IS high THEN valve IS drainage;
32 END_RULEBLOCK
33 END_FUNCTION_BLOCK

```

Fonte: IEC 1131-7 (1997)

O primeiro elemento constitui o todo, ou seja, representa todo o sistema fuzzy e é chamado bloco de função (FUNCTION\_BLOCK). Um bloco de função é composto por parâmetros de entrada e saída, constituindo-se como uma interface. Isto possibilita sua integração com distintos sistemas e uso em plataformas independentes. Deste modo, o bloco de função encapsula os demais elementos do controlador fuzzy: as variáveis de entrada (VAR\_INPUT), variáveis de saída (VAR\_OUTPUT) e base de regras (RULEBLOCK). Além

das variáveis do sistema, o bloco de funções ainda declara seus métodos de fuzzificação (FUZZIFY) e defuzzificação (DEFUZZIFY).

É declarado um bloco de fuzzificação para cada variável de entrada, conforme pode ser observado nas linhas 9 a 16 no Quadro 1. Neste bloco, é definido cada termo linguístico associado à variável (TERM) e seus respectivos parâmetros.

Para cada variável de saída do sistema é definido um bloco de defuzzificação. No seu interior são definidos os termos linguísticos associados à variável de saída (TERM) e o método de defuzzificação utilizado na inferência (METHOD), conforme pode ser observado nas linhas 17 a 24 no Quadro 1. Também é definido o método de acumulação das regras associadas à referida variável de saída (ACCU) e um valor padrão (DEFAULT), para o caso de nenhuma regra inferir uma grandeza para a variável (linhas 21 e 23).

Por fim, o FCL também declara em sua notação a base de regras. Além de definir cada regra (RULE), este elemento é responsável pela determinação do operador da regra (AND) e seu método de ativação (ACT), conforme linhas 25 a 32 do Quadro 1. É possível ainda a criação de múltiplos blocos de função, possibilitando a utilização de um arranjo diferente de métodos de ativação, acumulação e operador para distintos conjuntos de regras. Para isso, basta definir múltiplos blocos de regra para um mesmo bloco de função.

O sistema difuso apresentado pelo Quadro 1 recebe como variáveis de entrada um valor para temperatura e outro para pressão (linhas 2 a 5). A temperatura é fuzzificada com base nos termos frio e quente (linhas 9 a 12), enquanto a pressão se baseia nos termos baixa e alta (linhas 13 a 16). Como variável de saída, o sistema apresenta um valor para válvula (linhas 6 a 8), que é defuzzificada segundo os termos drenagem, fechada e entrada (linhas 18 a 20). A defuzzificação ocorre seguindo o método Centro da Gravidade (linha 22) e método Máximo para acumulação das regras (linha 21). Quatro regras ainda são definidas com seu conector AND respondendo ao método dos Mínimos (linhas 25 a 31).

## 2.4 TRABALHOS CORRELATOS

Esta seção destina-se a apresentar os trabalhos correlatos à ferramenta desenvolvida. Foram escolhidas três ferramentas que se apresentaram mais completas e eficazes frente ao objetivo buscado e são detalhadas nas próximas subseções.

### 2.4.1 Matlab Fuzzy Toolbox

O Matlab é um software, também considerado por muitos como uma linguagem, versátil em cálculos matemáticos, modelagens, simulações, gráficos, análises numéricas e desenvolvimento de algoritmos (MATHWORKS, 2013). Segundo Gilat (2005), o Matlab é largamente utilizado em universidades especialmente nos cursos de ciências exatas. Seu pacote padrão possui uma série de ferramentas para diversas áreas do conhecimento, as quais são completadas com a utilização de ferramentas adicionais, conhecidas por *toolboxes*. Uma dessas ferramentas adicionais trata de mecanismos para modelagem de sistemas nebulosos e é conhecida por Fuzzy Toolbox.

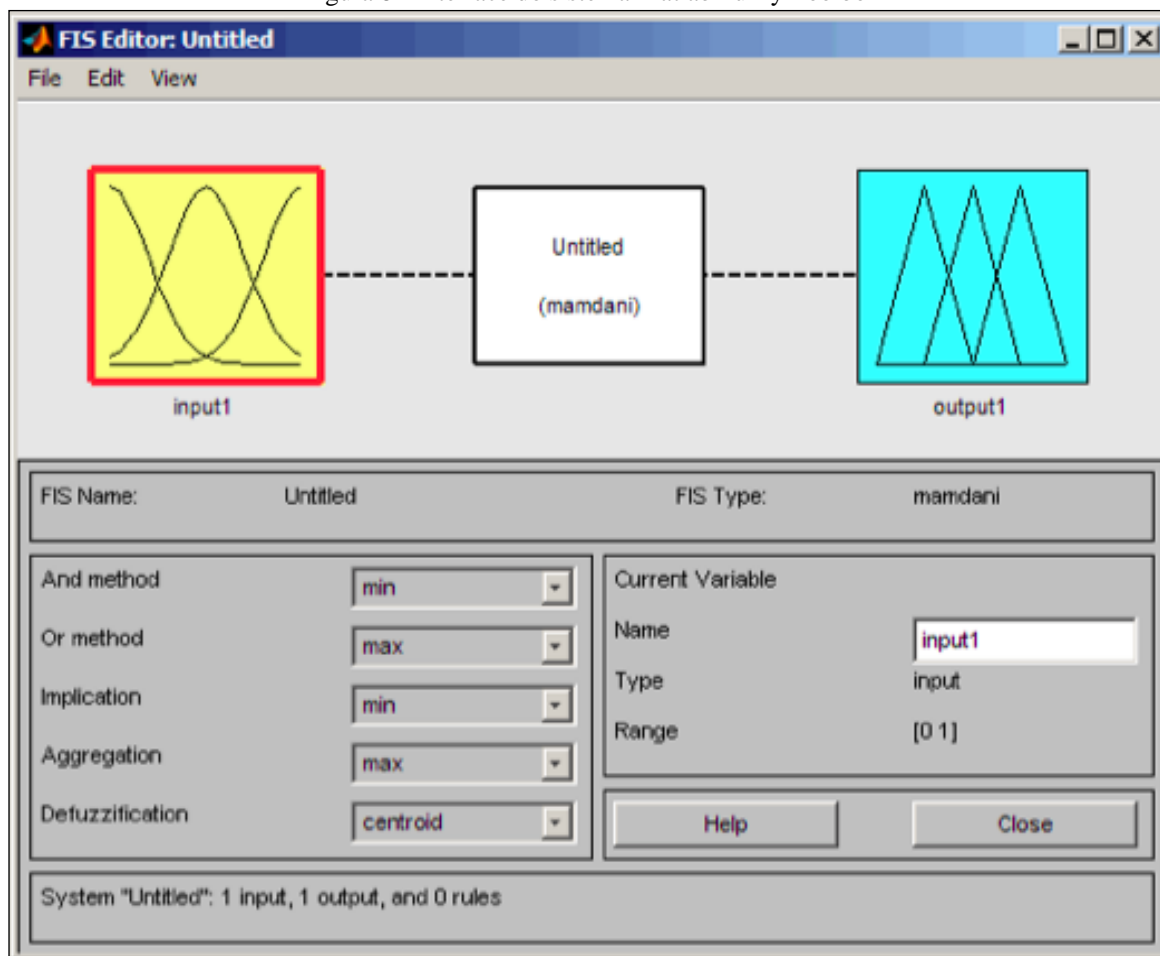
O Matlab Fuzzy Toolbox oferece um ambiente gráfico para a criação de sistemas de inferência fuzzy. A ferramenta está dividida em cinco grupos de funcionalidades. O primeiro grupo contempla as funcionalidades para a edição do sistema de inferência fuzzy, definindo as variáveis de entrada e saída do sistema. O segundo grupo define as funções de pertinência para cada uma das variáveis do sistema difuso. O terceiro grupo consiste no editor de regras, responsável por editar e listar as regras, determinando o comportamento do sistema. O quarto grupo representa a visualização das regras, através da qual é possível executar e visualizar o diagrama de inferência do sistema modelado, verificando os componentes influentes no resultado gerado. Por fim, o quinto grupo é denominado visualizador de superfície e apresenta o mapa de superfície de saída para o sistema construído (MATHWORKS, 2012).

O Matlab Fuzzy Toolbox apresenta uma interface gráfica com o diagrama contendo as variáveis de entrada e saída e o motor de inferência (Figura 8). É possível definir para o sistema os métodos dos conectores AND e OR, métodos de implicação e agregação de regras e de defuzzificação. A criação das funções de pertinência ocorre por variável, apresentando o gráfico correspondente para a variável selecionada. No momento de definição das regras do sistema, são apresentadas as variáveis e suas opções de valor, definidos no cadastro de funções de pertinência. É possível a utilização de um conectivo para todo o antecedente da regra e também uso do operador de negação para determinada premissa. O sistema ainda permite o cadastro de um peso para cada regra (MATHWORKS, 2012).

A simulação do sistema ocorre na visualização de regras, através da qual o usuário tem a possibilidade de informar os valores para cada variável de entrada. Para cada função de pertinência o sistema apresenta um gráfico, mostrando a relação entre o valor da variável e a função em questão. O sistema ainda apresenta as regras ativadas na execução

(MATHWORKS, 2012). A Figura 8 apresenta a tela do Matlab Fuzzy Toolbox no momento de criação do sistema, definição das variáveis e configurações.

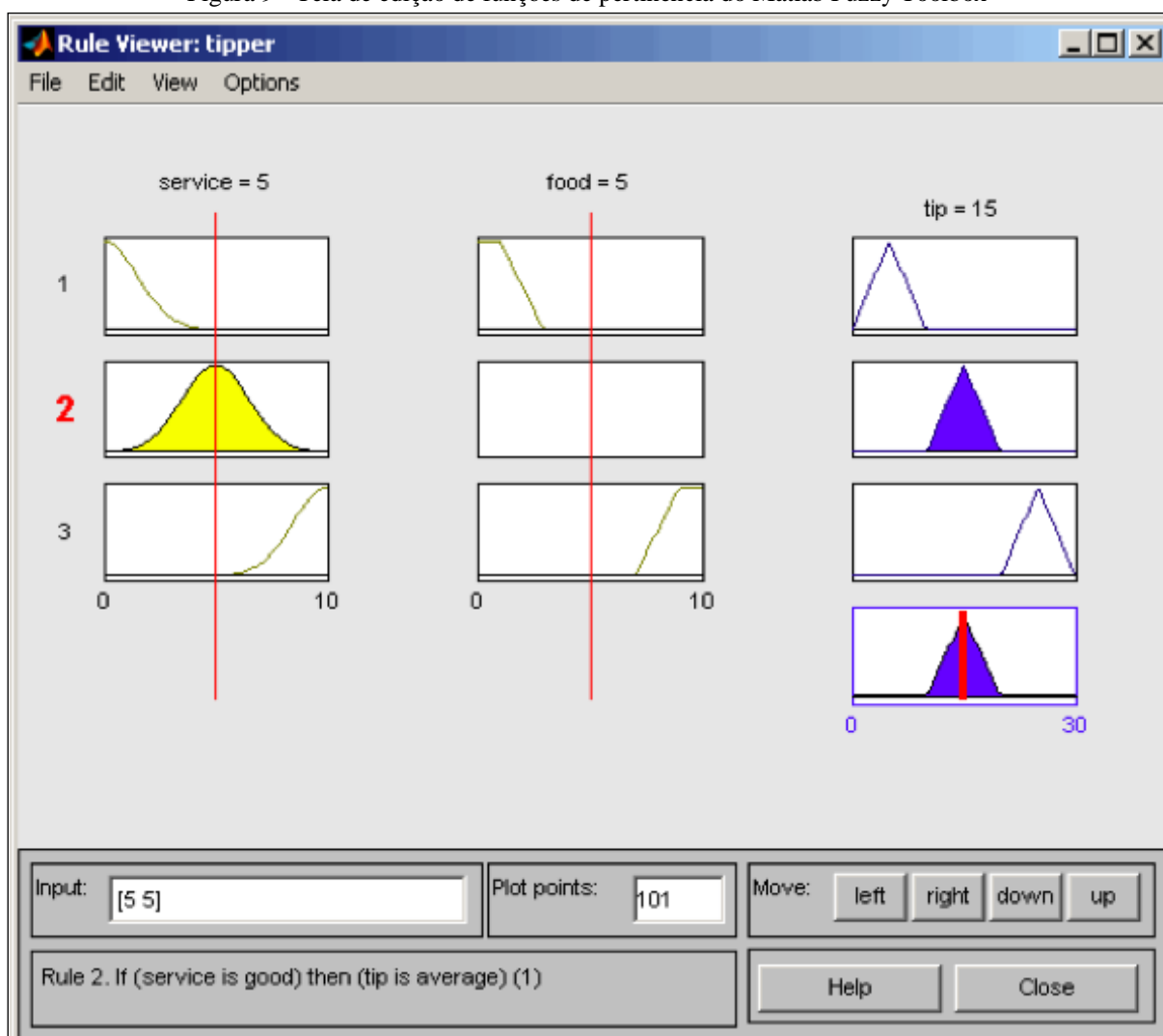
Figura 8 - Interface do sistema Matlab Fuzzy Toolbox



Fonte: Mathworks (2012)

O Matlab Fuzzy Toolbox, conforme abordado em Mathworks (2012), possui a possibilidade de salvar o sistema fuzzy construído, gerando um arquivo texto no formato *.fis*, que pode ser editado utilizando um editor de texto comum. O sistema ainda permite o desenvolvimento de sistemas difusos com funções de pertinência definidas pelo usuário. As Figuras 9 e 10 apresentam as interfaces de edição de termos linguísticos para uma variável de entrada e o visualizador de regras, respectivamente.

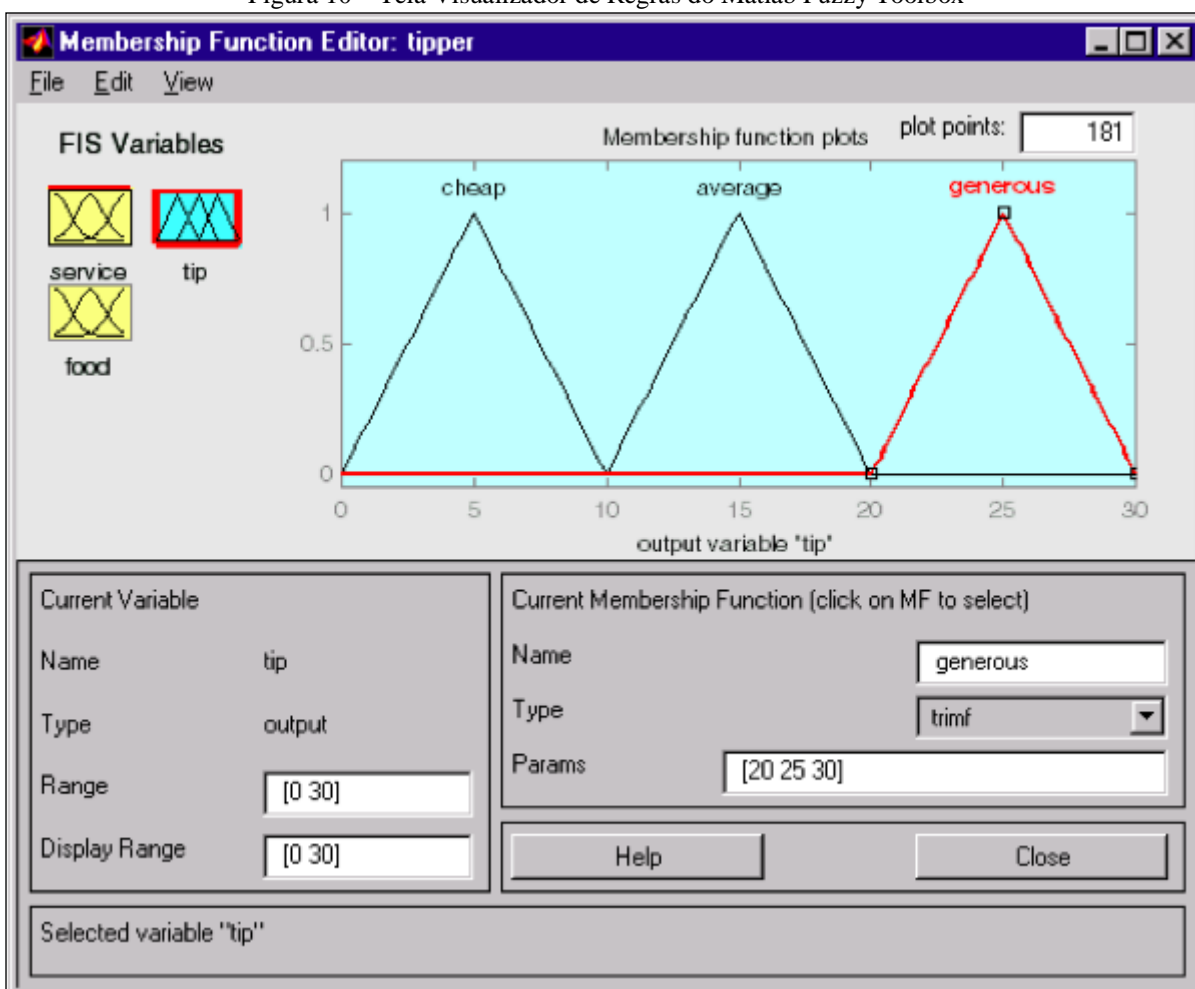
Figura 9 - Tela de edição de funções de pertinência do Matlab Fuzzy Toolbox



Fonte: Mathworks (2012)



Figura 10 – Tela Visualizador de Regras do Matlab Fuzzy Toolbox



Fonte: Mathworks (2012)

## 2.4.2 InFuzzy

O InFuzzy é um sistema desenvolvido como dissertação de mestrado (POSSELT, 2011). Consiste em um software para modelagem e simulação de sistemas fuzzy. Na criação de projetos, a ferramenta dispõe de uma área de desenho, através da qual se pode inserir, editar e remover os componentes do sistema. Para a construção do modelo é necessária a criação de variáveis de entrada, variáveis de saída e blocos de regras. Estes últimos definem a base de regras e o mecanismo de inferência do sistema. No cadastro de variáveis já são definidos seus termos linguísticos e funções de pertinência. É possível a definição de mais de um bloco de regras, incluindo uma premissa para cada variável de entrada e saída desejada, bem como o conector para o antecedente da regra (POSSELT et al., 2011).

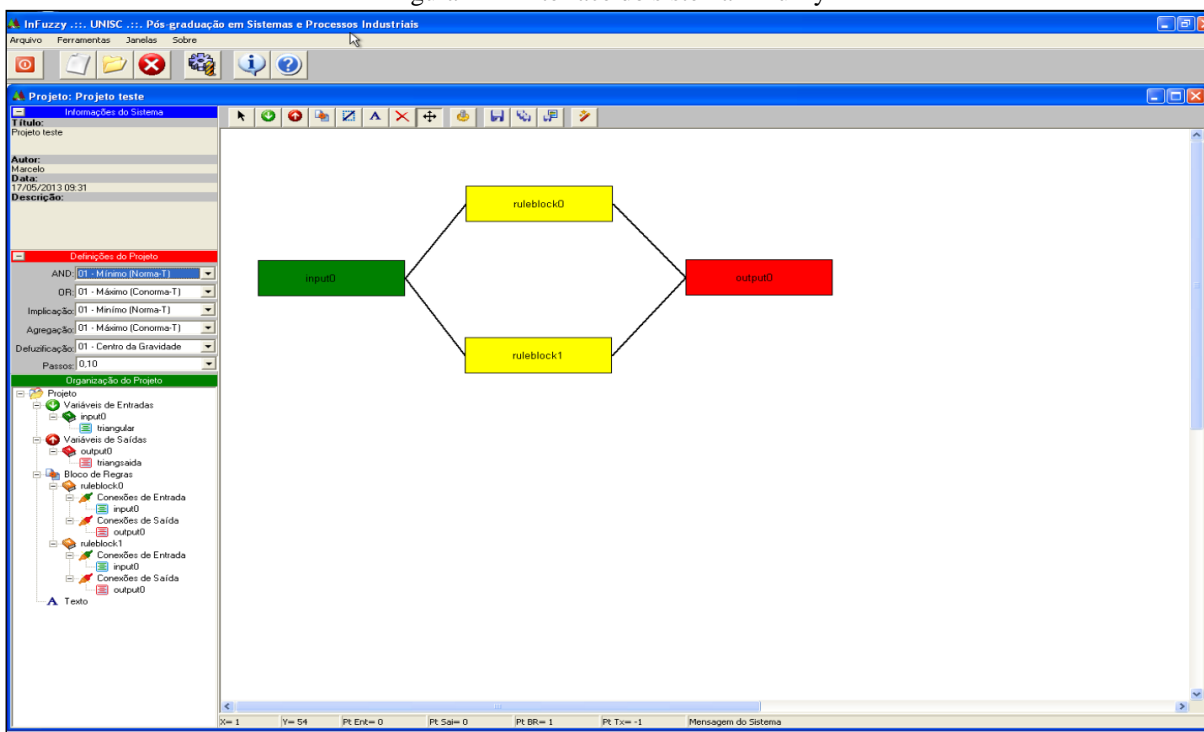
O software também disponibiliza algumas configurações válidas para todo o projeto, determinando o método de implicação e agregação das regras, método de defuzzificação e

definições dos operadores AND e OR. O sistema permite a criação de múltiplos blocos de regras, com configurações distintas de regra para cada um deles, porém todos estes elementos utilizam os mesmos métodos de inferência, isto é, não são independentes no momento da simulação (POSSELT, 2011).

Quanto à simulação do sistema fuzzy modelado, o InFuzzy permite a entrada de uma série de valores para cada variável, realizando várias simulações ao mesmo tempo. Nesta etapa, o sistema apresenta um descritivo textual do sistema, as etapas realizadas no processamento para inferir as saídas e o gráfico de defuzzificação. Uma funcionalidade específica deste sistema é a possibilidade de comunicar o sistema com um controlador ou aplicação externa através de rede, utilizando o protocolo UDP (POSSELT, 2011).

A Figura 11 apresenta a interface principal do software InFuzzy.

Figura 11 – Interface do sistema InFuzzy



Fonte: Posselt (2011)

### 2.4.3 FuzzyGen

O FuzzyGen é um sistema brasileiro, desenvolvido pela Embrapa como ferramenta para incorporação do conhecimento dos especialistas em sistemas para tomada de decisão (EMBRAPA, 2009). Frente ao alto custo das ferramentas disponíveis no mercado e dificuldade de integração do sistema com aplicações para o usuário final a organização optou

pelo desenvolvimento da ferramenta própria. O sistema foi desenvolvido em Java e, apesar da portabilidade possibilitada pela plataforma, sua execução em distintos sistemas operacionais apresentou problemas. Buscando suprir esta limitação, foi desenvolvido um sistema paralelo chamado WebFuzzy, cujo objetivo é fornecer em ambiente web a possibilidade de simulação e utilização dos controladores difusos construídos (LIMA; MASSRUHÁ, 2009).

O software FuzzyGen permite a construção de um modelo para o sistema fuzzy. Este modelo consiste em variáveis de entrada e saída com seus respectivos termos linguísticos. No caso das variáveis de saída é nesta etapa que são definidos os métodos de agregação e implicação. A base de regras é única para todo o sistema, bem como o método de ativação das mesmas e seu operador conectivo.

Uma característica do sistema é sua simplicidade de uso. A tela se divide em três abas: modelagem, inferência e gráfico. Na modelagem são definidas variáveis, termos linguísticos, regras e configurações gerais do sistema. É possível salvar o modelo desenvolvido em um arquivo, o qual segue o padrão FCL e é utilizado pela fase de inferência do sistema. Na inferência, são atribuídos os valores das variáveis de entrada do modelo, é apresentado o FCL correspondente e o resultado inferido. Por fim, a terceira aba apresenta os gráficos de defuzzificação das variáveis de saída (LIMA; MASSRUHÁ, 2009).

Uma vez desenvolvido o sistema fuzzy, é utilizado o software WebFuzzy para a simulação completa do sistema e depuração dos passos de processamento. Esta última ferramenta está disponível apenas para funcionários e usuários autorizados. A Figura 12 apresenta a tela principal do sistema FuzzyGen.

Figura 12 – Interface do sistema FuzzyGen

The screenshot shows the 'FuzzyGen 1.1.0' application window with the 'Modelagem' tab selected. The interface is organized into several sections:

- Nome do modelo:** A text box containing 'Gorjeta'.
- Variáveis de entrada:** A list box containing 'Comida' and 'Serviço'. To the right are 'Adicionar' and 'Remover' buttons.
- Variáveis de saída:** A list box containing 'Gorjeta'. To the right are 'Adicionar' and 'Remover' buttons.
- Conjunto de regras:** A list box containing three rules:
  - IF Serviço IS Pobre OR Comida IS Estragada OR null THEN Gorjeta IS Barata,...
  - IF Serviço IS Bom THEN Gorjeta IS Razoavel
  - IF Serviço IS Excelente AND Comida IS Deliciosa THEN Gorjeta IS GenerosaTo the right are 'Adicionar' and 'Remover' buttons.
- Método de ativação:** A dropdown menu set to 'MIN'.
- Operador:** Radio buttons for 'AND' (selected) and 'OR', with dropdown menus for 'MIN' and 'MAX'.
- Buttons:** 'Gravar em arquivo', 'Limpar', and 'Continuar >>' are located at the bottom.

Fonte: Lima e Massruhá (2009)

Como se percebe na interface apresentada pela Figura 12, nesta tela são criadas as variáveis de entrada (Comida e Serviço), bem como sua variável de saída (Gorjeta). Estes registros são utilizados para criação da base de regras, imediatamente abaixo. Além disso, a interface ainda permite a definição dos métodos de ativação e conectores AND e OR.

#### 2.4.4 Comparativo dos trabalhos correlatos

Com base na análise dos trabalhos correlatos, foi elaborada uma tabela comparativa, que apresenta os principais recursos disponíveis em cada ferramenta. Esta comparação pode ser observada no Quadro 2.

Quadro 2 – Comparativo de recursos entre os trabalhos correlatos

Recurso	Matlab Fuzzy Toolbox	InFuzzy	FuzzyGen
Exportação do sistema fuzzy no padrão FCL			X
Interface de desenho do modelo construído	X	X	
Exportação do desenho do modelo		X	
Possibilidade de simular / executar o sistema	X	X	X
Visualização de gráficos da execução	X	X	X
Comunicação da aplicação com outros sistemas	X	X	
Simulação de múltiplos valores simultaneamente		X	
Configuração dos métodos de agregação e acumulação de regras	X	X	X
Visualização das regras ativadas na simulação	X	X	
Depuração passo a passo	X		
Visualização gráfica de superfície	X		
Atribuição de peso para as regras	X		X
Distribuição gratuita		X	

Fonte: Acervo do Autor (2013)

### 3 DESENVOLVIMENTO DA FERRAMENTA

Nesta seção serão apresentados os aspectos importantes da fase de desenvolvimento da ferramenta proposta. Serão abordados os requisitos do software e os diagramas UML desenvolvidos durante a especificação da aplicação, como diagrama de casos de uso, classes e modelo de dados. Na seção implementação serão apontadas as técnicas utilizadas no desenvolvimento da ferramenta, como tecnologias e bibliotecas específicas. A seção operacionalização apresentará os passos para utilização do software. A seção de experimentação abordará a verificação e validação do software produzido, com base na sua especificação e objetivos. Por fim, a seção de resultados e discussões trará as conclusões quanto ao desenvolvimento do projeto, a comparação com seus trabalhos correlatos e uma explanação acerca dos resultados da pesquisa qualitativa realizada.

De acordo com Fowler (2005), UML (*Unified Modeling Language*) é uma família de notações gráficas cujo objetivo é descrever um sistema de software, sobretudo aqueles orientados a objetos. Sua principal característica é utilizar um modelo e métodos únicos, abstraindo o entendimento sobre o software. Os diagramas UML apresentados nas seções subsequentes visam o entendimento da arquitetura e funcionamento da ferramenta desenvolvida neste trabalho. Para a construção dos diagramas apresentados nas seguintes seções foi utilizado o software Enterprise Architect versão 10.

#### 3.1 CONCEPÇÃO

A ideia da ferramenta proposta neste trabalho foi concebida com base no cenário enfrentado atualmente por desenvolvedores de sistemas inteligentes que utilizam lógica fuzzy. As ferramentas de modelagem de sistemas difusos encontradas no mercado atual possuem algumas deficiências, as quais desencorajam esses profissionais no momento de optar pelo desenvolvimento de uma solução baseada em fuzzy. Lacunas como dificuldade de acesso, complexidade de uso e falta de funcionalidades estão presentes na maioria das soluções estudadas.

Pode-se mencionar que o assunto de lógica fuzzy é coberto nas disciplinas de inteligência computacional nos cursos de computação e nas disciplinas de sistemas inteligentes nos cursos de engenharia. O assunto carece de aplicações práticas, permanecendo

na parte teórica. Logo, necessita-se de ferramental para que os acadêmicos façam experimentações com lógica fuzzy.

Com base nestes dois contextos, uma ferramenta que supra as necessidades atuais deve ser fácil de usar, proporcionando agilidade na modelagem e validação do sistema fuzzy. Ainda, seria interessante que a solução proporcionasse um ambiente de simulação e execução do sistema difuso criado. Com base nessas características, as ferramentas Matlab Fuzzy Toolbox, InFuzzy e FuzzyGen foram analisadas como trabalhos correlatos ao desenvolvimento do presente trabalho. Todas estas soluções apresentam um ambiente para a modelagem de sistemas difusos, possibilitando a construção do modelo. Também apresentam funcionalidades de execução do modelo criado, inferindo respostas às entradas fornecidas pelo usuário. Além disso, apresentam-se gráficos referentes às execuções, expressando a forma como ocorreu a inferência por parte do sistema fuzzy.

Com base nessas funcionalidades básicas de uma ferramenta de modelagem para sistemas difusos, foi delimitado um escopo inicial para a ferramenta. Trata-se, portanto, de uma *shell* para sistemas fuzzy. Uma *shell* busca simplificar ao máximo o trabalho de implementação de um sistema inteligente, permitindo seu uso por pessoas sem conhecimentos de informática. Através de uma *shell*, o usuário preocupa-se apenas em transpor ao computador o conhecimento, através da construção da base de regras. A interpretação dessas regras, processamento e cálculos para a geração dos resultados são responsabilidades do software (HEINZLE, 1995; ALEXANDRE, 2000).

Neste sentido, a *shell* permitirá a criação dos componentes básicos de um sistema difuso para a sua modelagem, isto é, variáveis de entrada e saída, termos linguísticos, regras e motores de inferência. Também deverá possuir um ambiente para a execução do modelo, informando os resultados com base nos valores de entrada fornecidos pelo usuário e gerando gráficos da simulação. Buscando funcionalidades diferenciais, o sistema segue o padrão definido na norma IEC 1136-7 (1997), que especifica a linguagem de notação para controladores fuzzy conhecida por *Fuzzy Control Language*. Seguindo este modelo se garante a compatibilidade do modelo construído com qualquer software que siga a norma IEC 1136-7 (1997), bem como sua escalabilidade.

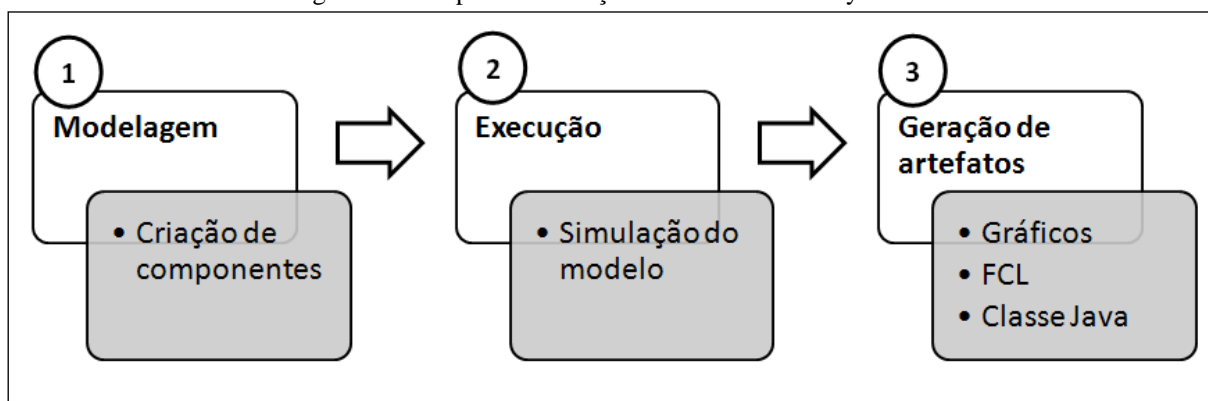
A ferramenta também deverá gerar artefatos com base no modelo construído. Além dos gráficos, a *shell* gerará código na notação FCL e código de programação em linguagem Java. Assim, torna-se possível a utilização do sistema fuzzy em qualquer aplicação que implemente as especificações da norma, bem como a construção de um software que faça uso

do modelo desenvolvido. Ainda, a ferramenta deverá permitir o trabalho colaborativo, ou seja, um usuário poderá compartilhar seu projeto com outras pessoas. Uma vez compartilhado, é possível o trabalho em conjunto e em tempo real no mesmo projeto. Este fato permite a realização de trabalhos em grupo quando aplicado em ambiente acadêmico.

Uma ferramenta com essas características, aplicada nos dois cenários citados poderia ajudar diretamente desenvolvedores e estudantes no trabalho com sistemas fuzzy. Desenvolvedores podem aproveitar as facilidades da ferramenta para modelar seus sistemas difusos, realizando testes e aferições. Através da geração de código de programação é possível a utilização do sistema nebuloso facilmente, acoplado a outro aplicativo sem necessidade de esforços adicionais de implementação. Para estudantes, é possível a experimentação das teorias estudadas em sala de aula de maneira facilitada.

Frente às funcionalidades levantadas até o momento, a Figura 13 apresenta o fluxo de passos para a construção de um sistema difuso. A ferramenta FuzzyStudio, desenvolvida no presente trabalho, se baseia nesta sequência de atividades para a implementação dos seus recursos.

Figura 13 – Etapas de utilização da ferramenta FuzzyStudio



Fonte: Acervo do Autor (2013)

De acordo com a imagem apresentada pela Figura 13, a primeira etapa consiste na modelagem do sistema fuzzy. Esta etapa aborda os recursos de criação dos componentes do modelo, definindo as variáveis de entrada e saída, a base de regras e a configuração do motor de inferência. A ferramenta está baseada no modelo Mamdani. Na execução, o usuário informa valores para cada variável de entrada. O sistema agrupa todos os componentes criados em forma textual e envia, juntamente com os valores de entrada, para a biblioteca *jFuzzyLogic*. Esta fará o processamento e execução do sistema, retornando os resultados para



apresentação ao usuário. Por fim, a geração de artefatos embarca as funcionalidades de visualização de gráficos de execução, exportação do sistema no padrão FCL e geração de uma classe Java que implementa o sistema e sua execução.

### 3.2 REQUISITOS E REGRAS DE NEGÓCIO

De maneira geral, Sommerville (2003) define os requisitos como descrições das funções e das restrições de um sistema de software. Neste sentido, levantar requisitos é estabelecer o que o sistema deve fazer e quais funcionalidades o mesmo disponibilizará. Os requisitos são divididos em três grandes grupos. O primeiro deles é composto pelos requisitos funcionais, que são declarações do que o sistema deve fazer, como deve reagir ou se comportar. O segundo grupo é composto pelos requisitos não funcionais, que são as restrições sobre as funções e serviços oferecidos pelo software como restrições de tempo e padrões. O terceiro grupo trata dos requisitos de domínio. Requisitos funcionais, não funcionais e de domínio estão dentro de um contexto que é regido pelas regras de negócio. Este grupo traduz as características do domínio, ou seja, requisitos que se originam na área da aplicação do sistema.

Uma vez delimitado o escopo geral do trabalho e a lacuna que o mesmo deseja preencher, são apresentadas as regras de negócio, requisitos funcionais e requisitos não funcionais do projeto. O Quadro 3 apresenta a lista das regras de negócio identificadas para a ferramenta. Cada regra de negócio vincula-se a pelo menos um requisito funcional, os quais são apresentados no Quadro 4.

Quadro 3 – Regras de negócio da aplicação

<b>Regra</b>	<b>Descrição</b>
<b>RN-01</b>	O usuário realizará seu próprio cadastro no sistema.
<b>RN-02</b>	O cadastro de usuários conterà seu nome, dados de acesso e dados de contato.
<b>RN-03</b>	Para se autenticar no sistema, o usuário informará seu nome de usuário e senha.
<b>RN-04</b>	Um projeto deverá ter vinculado um ou vários usuários.
<b>RN-05</b>	A interface de desenho conterà todos os componentes criados com suas respectivas conexões.
<b>RN-06</b>	O usuário poderá mover os componentes desenhados e ajustá-los conforme seu gosto / necessidade.
<b>RN-07</b>	O usuário poderá excluir um componente através do desenho visual, bem como editar seus atributos.

<b>RN-08</b>	As variáveis fuzzy conterão nome, limite de valores (mínimo e máximo) e unidade de medida.
<b>RN-09</b>	As variáveis conterão termos linguísticos a ela associados.
<b>RN-10</b>	Um termo linguístico será composto por uma função de pertinência, seu nome e valores para a função.
<b>RN-11</b>	As funções de pertinência aceitas pelo sistema serão: triangular, trapezoidal, rampa esquerda/direita, inclinação esquerda/direita e discreta.
<b>RN-12</b>	O gráfico de funções de pertinência será do tipo linear.
<b>RN-13</b>	Uma regra é composta por um antecedente e um consequente.
<b>RN-14</b>	O antecedente de uma regra é formado com as variáveis de entrada do sistema, ao passo que seu consequente é formado com as variáveis de saída.
<b>RN-15</b>	Quando o antecedente de regra possuir mais de uma condição, estas devem ser conectadas pelos operadores “E” e “OU”.
<b>RN-16</b>	A execução lê os valores de cada variável de entrada informados pelo usuário, calcula o valor das saídas e apresenta-os em tela.
<b>RN-17</b>	O código gerado pela ferramenta será uma classe Java que, quando executada em conjunto com a biblioteca <i>jFuzzyLogic</i> , solicita valores para as variáveis de entrada e apresenta o valor das saídas, juntamente com gráficos do sistema modelado.
<b>RN-18</b>	Cada variável terá seu gráfico correspondente. Nos gráficos das variáveis de entrada deverá ser apresentado o valor informado pelo usuário. Nos gráficos das variáveis de saída deverá ser apresentada a área referente à defuzzificação, bem como seu valor resultante.
<b>RN-19</b>	Quando o usuário fizer alguma alteração no ambiente de um projeto, todos os usuários conectados ao mesmo deverão receber a alteração de forma transparente.
<b>RN-20</b>	O motor de inferência terá regras associadas a ele, as quais utilizará para inferir os resultados.
<b>RN-21</b>	Os métodos de defuzzificação utilizados pelo motor serão: Centro da Gravidade; <i>Right Most Max</i> ; Centro da Área e <i>Left Most Max</i> .
<b>RN-22</b>	Os métodos de agregação de regras utilizados pelo motor serão: Soma Limitada; Máximo; Soma e Soma Normalizada.
<b>RN-23</b>	Os métodos do conector AND utilizados pelo motor serão: Mínimo; Produto e Soma Limitada.
<b>RN-24</b>	Os métodos de ativação de regras utilizados pelo motor serão: Mínimo e Produto.

Fonte: Acervo do Autor (2013)

O Quadro 4 é composto pelos requisitos funcionais do sistema. A terceira coluna do quadro representa as regras de negócio associadas a cada requisito em questão.

Quadro 4 - Requisitos funcionais da aplicação

<b>Requisito</b>	<b>Descrição</b>	<b>Regra(s) de negócio</b>
<b>RF-01</b>	O sistema deve permitir o cadastro de um novo usuário.	RN-01; RN-02;
<b>RF-02</b>	O sistema deve controlar a autenticação do usuário.	RN-03;
<b>RF-03</b>	O sistema deve manter projetos.	RN-04;
<b>RF-04</b>	O sistema deve permitir ao usuário alterar seus dados pessoais e de acesso.	RN-02;
<b>RF-05</b>	O sistema deve possuir interface gráfica de desenho para visualização e edição dos componentes do modelo.	RN-05; RN-06; RN-07;
<b>RF-06</b>	O sistema deve manter variáveis de entrada.	RN-08; RN-09;
<b>RF-07</b>	O sistema deve manter termos linguísticos das variáveis de entrada.	RN-09; RN-10; RN-11;
<b>RF-08</b>	O sistema deve manter variáveis de saída.	RN-08; RN-09;
<b>RF-09</b>	O sistema deve manter termos linguísticos das variáveis de saída.	RN-09; RN-10; RN-11;
<b>RF-10</b>	O sistema deve gerar gráfico de funções de pertinência para cada variável no momento da edição de termos linguísticos.	RN-12;
<b>RF-11</b>	O sistema deve manter regras.	RN-13; RN-14; RN-15;
<b>RF-12</b>	O sistema deve manter motores de inferência.	
<b>RF-13</b>	O sistema deve permitir a associação do motor de inferência com uma ou mais regras da base de regras modelada.	RN-20;
<b>RF-14</b>	O sistema deve permitir o cadastro do método de defuzzificação, conexão AND, método de agregação e ativação de regras associadas ao motor.	RN-21; RN-22; RN-23; RN-24;
<b>RF-15</b>	O sistema deve permitir a simulação / execução do sistema modelado.	RN-16;
<b>RF-16</b>	O sistema deve executar a simulação isolada por motor de inferência.	RN-16;
<b>RF-17</b>	O sistema deve informar ao usuário as saídas agrupadas por motor de inferência.	RN-16;
<b>RF-18</b>	O sistema deve gerar notação do sistema fuzzy modelado para cada motor de inferência.	
<b>RF-19</b>	O sistema deve gerar notação seguindo o padrão IEC 1131-7.	
<b>RF-20</b>	O sistema deve gerar código de programação em Java do sistema / motor modelado.	RN-17;
<b>RF-21</b>	O sistema deve gerar gráficos da execução de cada motor.	RN-18;
<b>RF-22</b>	O sistema deve permitir o compartilhamento do projeto com outros usuários.	RN-19;
<b>RF-23</b>	O sistema deve permitir a operação de um projeto em tempo real por mais de um usuário.	RN-19;
<b>RF-24</b>	O sistema deve permitir a visualização dos usuários conectados ao projeto.	RN-04;

Fonte: Acervo do Autor (2013)

No Quadro 5 são apresentados os requisitos não funcionais levantados para o sistema. De maneira geral, trata das tecnologias e plataforma utilizadas para o desenvolvimento do trabalho, com objetivo de prover ao sistema flexibilidade de uso e eficiência.

Quadro 5 - Requisitos não funcionais da aplicação

<b>Requisito</b>	<b>Descrição</b>
<b>RNF-01</b>	O sistema deve ser executado em plataforma Web.
<b>RNF-02</b>	O sistema deve ser compatível com os navegadores Google Chrome 26.0.x, Mozilla Firefox 21.0 e Safari 5.1.x.
<b>RNF-03</b>	O sistema deve utilizar banco de dados MySQL.

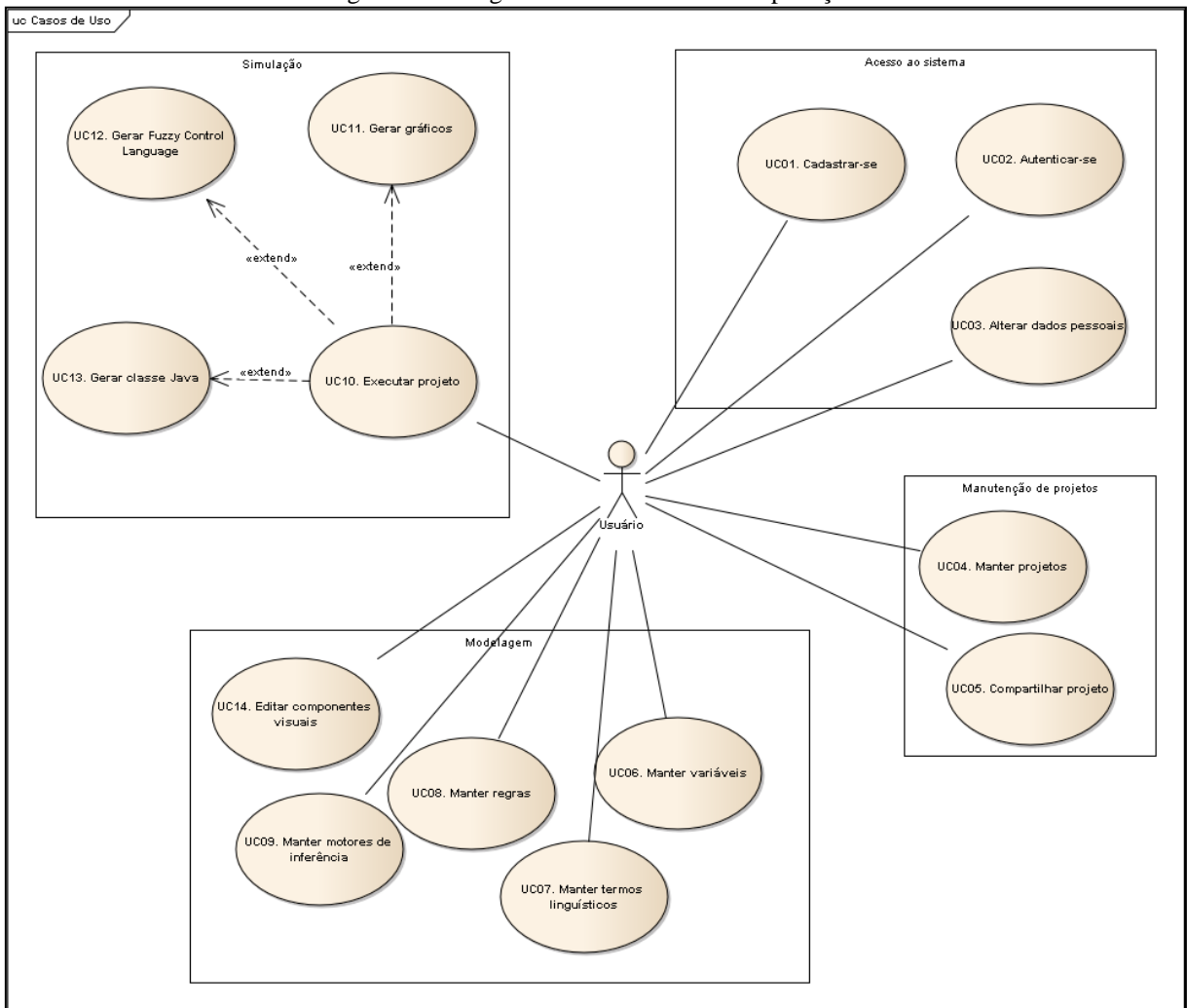
Fonte: Acervo do Autor (2013)

### 3.3 CASOS DE USO

Conforme Bezerra (2002), um caso de uso é uma especificação baseada na interação entre um agente externo e um sistema. Nessa especificação, é definido o uso de uma funcionalidade específica do sistema sem revelar sua estrutura interna. Pressman (2011) aborda o fato de um caso de uso descrever como um usuário interage com o sistema em circunstâncias específicas. Em suma, “um caso de uso descreve o software ou o sistema do ponto de vista do usuário” (PRESSMAN, 2011, p. 130).

Para a ferramenta FuzzyStudio, foi definido que a mesma possui um único ator: o usuário da aplicação. Neste sentido, será o único operador que irá interagir com a ferramenta. Com base nessas interações foi elaborado o diagrama de casos de uso, que apresenta uma visão geral da aplicação. O diagrama de casos de uso pode ser verificado na Figura 14.

Figura 14 – Diagrama de casos de uso da aplicação



Fonte: Acervo do Autor (2013)

Como se pode observar, o diagrama divide os casos de uso em quatro regiões. A área “Acesso ao sistema” consiste na autenticação no sistema, desde o cadastro de usuário, login e alteração dos dados pessoais. A área “Manutenção de projetos” inclui casos de uso para o gerenciamento dos projetos e seu compartilhamento com demais usuários. Na área “Modelagem” apresentam-se os casos de uso para a modelagem do sistema fuzzy e criação de seus componentes. Por fim, a área “Simulação” destina-se aos casos de uso acerca da validação do sistema modelado, incluindo a simulação do sistema e geração de artefatos.

Os casos de uso contidos na Figura 14 estão detalhados nos Quadros 6 a 19.

O usuário deve ser capaz de cadastrar-se no sistema, de forma que a partir de então tenha acesso às funcionalidades fornecidas pela aplicação. O cenário deste caso de uso é apresentado no Quadro 6.

Quadro 6 - UC-01. Cadastrar-se

<b>UC-01</b>	<b>Cadastrar-se</b>
<b>Descrição</b>	O usuário se cadastra para acesso às funcionalidades da aplicação.
<b>Rastreabilidade</b>	RF-01; RN-01; RN-02;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário acessar a aplicação
<b>Pós-Condições</b>	Usuário cadastrado no sistema.
<b>Fluxo Principal</b>	1. O usuário seleciona a opção de cadastro no sistema. (RN-01) 2. O usuário digita os dados de cadastro. (RN-02) 3. O sistema grava no banco de dados.
<b>Exceções</b>	2.a. O usuário informa um nome de usuário existente. 2.a.1. O sistema informa a impossibilidade de cadastro do referido nome de usuário. 2.a.2. Retorna ao passo 2.

Fonte: Acervo do Autor (2013)

O Quadro 7 apresenta o cenário para o caso de uso UC-02, que trata da autenticação do usuário no sistema. O sistema deverá controlar o acesso de cada usuário, bem como seus projetos vinculados.

Quadro 7 - UC-02. Autenticar-se

<b>UC-02</b>	<b>Autenticar-se</b>
<b>Descrição</b>	O usuário realiza login no sistema e acessa seus projetos.
<b>Rastreabilidade</b>	RF-02; RN-03;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário estar cadastrado no sistema.
<b>Pós-Condições</b>	Usuário autenticado no sistema.
<b>Fluxo Principal</b>	1. O usuário seleciona a opção de acesso ao sistema. 2. O usuário digita os dados de acesso. (RN-03) 3. O sistema retorna tela com os projetos do usuário.
<b>Exceções</b>	2.a. O usuário informa nome de usuário ou senha incorretos. 2.a.1. O sistema informa a inconsistência dos dados. 2.a.2. Retorna ao passo 2.

Fonte: Acervo do Autor (2013)

Uma vez autenticado no sistema, o usuário deve ser capaz de alterar seus dados de cadastro. Esta funcionalidade é prevista no caso de uso UC-03, o qual está decomposto no Quadro 8.

Quadro 8 - UC-03. Alterar dados pessoais

<b>UC-03</b>	<b>Alterar dados pessoais</b>
<b>Descrição</b>	O usuário altera seus dados pessoais.
<b>Rastreabilidade</b>	RF-04; RN-02;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário autenticado no sistema.
<b>Pós-Condições</b>	Dados do usuário alterados.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. O usuário seleciona a opção de alteração de dados pessoais.</li> <li>2. O usuário digita os novos dados. (RN-02)</li> <li>3. O sistema retorna tela com os projetos do usuário.</li> </ol>
<b>Exceções</b>	<ol style="list-style-type: none"> <li>2.a. O usuário informa um nome de usuário existente.             <ol style="list-style-type: none"> <li>2.a.1. O sistema informa a impossibilidade de cadastro do referido nome de usuário.</li> <li>2.a.2. Retorna ao passo 2.</li> </ol> </li> </ol>

Fonte: Acervo do Autor (2013)

Ao abrir a tela principal de projetos do usuário, este tem a possibilidade de criar um novo projeto, abrir ou excluir algum já existente. Essas opções estão definidas no caso de uso UC-04, cujo cenário é apresentado no Quadro 9.

Quadro 9 - UC-04. Manter projetos

<b>UC-04</b>	<b>Manter projetos</b>
<b>Descrição</b>	O usuário cria, abre ou exclui um projeto.
<b>Rastreabilidade</b>	RF-03; RN-04;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário autenticado no sistema.
<b>Pós-Condições</b>	Projeto criado, aberto ou excluído.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. O usuário seleciona um projeto na lista de projetos disponíveis.</li> <li>2. O usuário seleciona a opção de abrir o projeto.</li> <li>3. O sistema retorna à tela de edição de projetos.</li> </ol>
<b>Fluxo Alternativo</b>	<ol style="list-style-type: none"> <li>1.a. Usuário deseja criar novo projeto.             <ol style="list-style-type: none"> <li>1.a.1. Usuário seleciona opção de criação de novo projeto.</li> <li>1.a.2. Usuário informa nome do projeto.</li> <li>1.a.3. Sistema grava no banco de dados.</li> <li>1.a.4. Retorna ao passo 3.</li> </ol> </li> <li>1.b. Usuário deseja excluir o projeto.             <ol style="list-style-type: none"> <li>1.b.1. Usuário seleciona opção de excluir projeto.</li> <li>1.b.2. Sistema remove registro do banco de dados.</li> <li>1.b.3. Retorna ao passo 1.</li> </ol> </li> </ol>

Fonte: Acervo do Autor (2013)

O UC-05 aborda a possibilidade de compartilhamento de projetos. O Quadro 10 apresenta o cenário para o referido caso de uso.

Quadro 10 - UC-05. Compartilhar projeto

<b>UC-05</b>	<b>Compartilhar projeto</b>
<b>Descrição</b>	O usuário compartilha seu projeto com outros usuários.
<b>Rastreabilidade</b>	RF-22; RF-23; RN-19;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário autenticado no sistema.
<b>Pós-Condições</b>	Projeto compartilhado.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. O usuário seleciona um projeto na lista de projetos disponíveis.</li> <li>2. O usuário seleciona a opção de compartilhar projeto.</li> <li>3. O usuário seleciona os usuários com os quais deseja compartilhar seu projeto.</li> <li>4. O sistema salva no banco de dados.</li> <li>5. O sistema retorna para a tela de projetos.</li> </ol>
<b>Fluxo Alternativo</b>	<ol style="list-style-type: none"> <li>1.a. Usuário já possui o projeto aberto. <ol style="list-style-type: none"> <li>1.a.1. Usuário seleciona a aba de Opções.</li> <li>1.a.2. Usuário seleciona a opção de compartilhar projeto.</li> <li>1.a.3. O usuário seleciona os usuários com os quais deseja compartilhar seu projeto.</li> <li>1.a.4. O sistema salva no banco de dados.</li> <li>1.a.5. O sistema retorna para a tela de edição de projetos.</li> <li>1.a.6. Encerra Caso de Uso</li> </ol> </li> </ol>

Fonte: Acervo do Autor (2013)

No tangente às funcionalidades do recurso “Modelagem” do diagrama apresentado pela Figura 14, estão definidos os casos de uso de manutenção dos componentes de um sistema fuzzy. O UC-06 aborda a manutenção de variáveis do projeto. O Quadro 11 apresenta o cenário para o referido caso de uso.

Quadro 11 - UC-06. Manter variáveis

<b>UC-06</b>	<b>Manter variáveis</b>
<b>Descrição</b>	O usuário cria, edita e exclui variáveis do sistema fuzzy.
<b>Rastreabilidade</b>	RF-06; RF-08; RN-08; RN-09;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Pós-Condições</b>	Variável criada, atualizada ou excluída.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. O usuário seleciona a opção de criação de nova variável.</li> <li>2. O usuário informa os dados da variável. (RN-08)</li> <li>3. O sistema salva no banco de dados.</li> <li>4. O sistema atualiza a interface de desenho.</li> </ol>
<b>Fluxo Alternativo</b>	<ol style="list-style-type: none"> <li>1.a. Usuário deseja alterar uma variável existente <ol style="list-style-type: none"> <li>1.a.1. Usuário seleciona uma variável na lista de variáveis ou pela interface de desenho.</li> <li>1.a.2. Usuário informa novos dados para a variável. (RN-08)</li> <li>1.a.3. O sistema salva no banco de dados.</li> <li>1.a.4. O sistema atualiza a interface de desenho.</li> </ol> </li> </ol>



	<ul style="list-style-type: none"> <li>1.a.5. Encerra Caso de Uso.</li> <li>1.b. Usuário deseja excluir uma variável existente <ul style="list-style-type: none"> <li>1.b.1. Usuário seleciona uma variável na lista de variáveis ou pela interface de desenho.</li> <li>1.b.2. Usuário seleciona opção de exclusão.</li> <li>1.b.3. O sistema exclui o registro do banco de dados.</li> <li>1.b.4. O sistema atualiza a interface de desenho.</li> <li>1.b.5. Encerra Caso de Uso.</li> </ul> </li> </ul>
<b>Exceções</b>	<ul style="list-style-type: none"> <li>3.a. O usuário não informa algum campo obrigatório. <ul style="list-style-type: none"> <li>3.a.1. O sistema informa a necessidade de preenchimento dos referidos dados.</li> <li>3.a.2 Retorna ao passo 2.</li> </ul> </li> <li>1.a.2.a. O usuário não informa algum campo obrigatório. <ul style="list-style-type: none"> <li>1.a.2.a.1. O sistema informa a necessidade de preenchimento dos referidos dados.</li> <li>1.a.2.a.2. Retorna ao passo 1.a.2.</li> </ul> </li> </ul>

Fonte: Acervo do Autor (2013)

Para cada variável inserida no sistema, o usuário cadastra seus respectivos termos linguísticos, responsáveis pela fuzzificação e defuzzificação. Esta funcionalidade está prevista pelo caso de uso UC-07, apresentado no cenário do Quadro 12.

Quadro 12 - UC-07. Manter termos linguísticos

<b>UC-07</b>	<b>Manter termos linguísticos</b>
<b>Descrição</b>	O usuário cria termos linguísticos para as variáveis cadastradas.
<b>Rastreabilidade</b>	RF-07; RF-09; RN-09; RN-10; RN-11;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Variável selecionada na lista de variáveis cadastradas.
<b>Pós-Condições</b>	Termo linguístico criado, alterado ou excluído.
<b>Fluxo Principal</b>	<ul style="list-style-type: none"> <li>1. O usuário seleciona a opção de criação de novo termo linguístico.</li> <li>2. O usuário seleciona o tipo de função. (RN-11)</li> <li>3. O sistema atualiza os campos de inserção de parâmetros de acordo com a função desejada.</li> <li>4. O usuário informa demais dados do termo. (RN-10)</li> <li>5. O sistema grava no banco de dados.</li> <li>6. O sistema atualiza o gráfico de termos linguísticos.</li> </ul>
<b>Fluxo Alternativo</b>	<ul style="list-style-type: none"> <li>1.a. Usuário deseja alterar um termo existente <ul style="list-style-type: none"> <li>1.a.1. Usuário seleciona o termo na lista.</li> <li>1.a.2. Usuário informa novos dados para o registro. (RN-10 e RN-11)</li> <li>1.a.3. O sistema salva no banco de dados.</li> <li>1.a.4. O sistema atualiza a o gráfico de termos linguísticos.</li> <li>1.a.5. Encerra Caso de Uso.</li> </ul> </li> <li>1.b. Usuário deseja excluir um termo existente <ul style="list-style-type: none"> <li>1.b.1. Usuário seleciona o termo na lista.</li> <li>1.b.2. Usuário seleciona opção de exclusão.</li> <li>1.b.3. O sistema exclui o registro do banco de dados.</li> </ul> </li> </ul>

	1.b.4. O sistema atualiza o gráfico de termos linguísticos. 1.b.5. Encerra Caso de Uso.
<b>Exceções</b>	5.a. O usuário não informa algum campo obrigatório. 5.a.1. O sistema informa a necessidade de preenchimento dos referidos dados. 5.a.2. Retorna ao passo 4. 1.a.3.a. O usuário não informa algum campo obrigatório. 1.a.3.a.1. O sistema informa a necessidade de preenchimento dos referidos dados. 1.a.3.a.2. Retorna ao passo 1.a.2.

Fonte: Acervo do Autor (2013)

Uma vez cadastradas as variáveis e seus respectivos termos linguísticos é possível a construção da base de regras do sistema difuso. O caso de uso UC-08 cobre a funcionalidade de manutenção de regras, seu cenário é apresentado pelo Quadro 13.

Quadro 13 - UC-08. Manter regras

<b>UC-08</b>	<b>Manter regras</b>
<b>Descrição</b>	O usuário cria regras para o sistema fuzzy.
<b>Rastreabilidade</b>	RF-11; RN-13; RN-14; RN-15;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Pós-Condições</b>	Regra criada ou excluída.
<b>Fluxo Principal</b>	1. O usuário seleciona a opção de criação de nova regra. 2. O usuário monta os antecedentes e consequentes da regra, selecionando as variáveis correspondentes e seus termos linguísticos. 3. O sistema grava no banco de dados. 4. O sistema atualiza a lista de regras da base de regras.
<b>Fluxo Alternativo</b>	1.a. Usuário deseja excluir uma regra existente 1.a.1. Usuário seleciona a regra na lista. 1.a.2. Usuário seleciona opção de exclusão. 1.a.3. O sistema exclui o registro do banco de dados. 1.a.4. O sistema atualiza a lista de regras da base de regras.
<b>Exceções</b>	3.a. O usuário não informa algum campo obrigatório (ao menos uma variável e termo para antecedente e uma para consequente). 3.a.1. O sistema informa a falta de dados. 3.a.2. Retorna ao passo 2.

Fonte: Acervo do Autor (2013)

A base de processamento de um sistema fuzzy consiste no seu motor de inferência, que fará uso da base de regras para inferir as saídas desejadas. Para a ferramenta foi modelada uma estrutura que comporta a utilização de múltiplos motores de inferência. A base de regras é composta por várias regras definidas pelo usuário, porém há a possibilidade de que o mesmo

defina quais regras serão utilizadas para algum motor específico. Esta definição ocorre no momento de criação do motor, que é detalhado pelo cenário apresentado no Quadro 14.

Quadro 14 - UC-09. Manter motores de inferência

<b>UC-09</b>	<b>Manter motores de inferência</b>
<b>Descrição</b>	O usuário cria e manipula motores de inferência para o sistema fuzzy.
<b>Rastreabilidade</b>	RF-12; RF-13; RF-14; RN-20; RN-21; RN-22; RN-23; RN-24
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Pós-Condições</b>	Motor de inferência criado e configurado ou excluído.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. O usuário seleciona a opção de criação de novo motor de inferência.</li> <li>2. O usuário seleciona o método de defuzzificação a ser utilizado. (RN-21)</li> <li>3. O usuário seleciona o método de agregação de regras. (RN-22)</li> <li>4. O usuário seleciona o método da conexão AND. (RN-23)</li> <li>5. O usuário seleciona o método de ativação de regras. (RN-24)</li> <li>6. O usuário seleciona as regras do motor. (RN-20)</li> <li>7. O sistema grava no banco de dados.</li> <li>8. O sistema atualiza a lista de motores de inferência.</li> </ol>
<b>Fluxo Alternativo</b>	<ol style="list-style-type: none"> <li>1.a. Usuário deseja alterar um motor de inferência existente.               <ol style="list-style-type: none"> <li>1.a.1. Usuário seleciona o motor na lista.</li> <li>1.a.2. Usuário seleciona as opções do motor. (RN-20 a 24)</li> <li>1.a.3. O sistema grava no banco de dados.</li> <li>1.a.4. O sistema atualiza a lista de motores de inferência.</li> <li>1.a.5. Encerra Caso de Uso.</li> </ol> </li> <li>1.b. Usuário deseja excluir um motor de inferência existente.               <ol style="list-style-type: none"> <li>1.b.1. Usuário seleciona o motor na lista.</li> <li>1.b.2. Usuário seleciona opção de exclusão.</li> <li>1.b.3. O sistema exclui o registro do banco de dados.</li> <li>1.b.4. O sistema atualiza a lista de motores de inferência.</li> <li>1.b.5. Encerra Caso de Uso.</li> </ol> </li> </ol>
<b>Exceções</b>	<ol style="list-style-type: none"> <li>7.a. O usuário deixa de selecionar algum método do motor.           <ol style="list-style-type: none"> <li>7.a.1. O sistema informa a falta de dados.</li> <li>7.a.2. Retorna ao passo 2.</li> </ol> </li> <li>1.a.3.a. O usuário deixa de selecionar algum método do motor.           <ol style="list-style-type: none"> <li>1.a.3.a.1. O sistema informa a falta de dados.</li> <li>1.a.3.a.2. Retorna ao passo 1.a.2.</li> </ol> </li> </ol>

Fonte: Acervo do Autor (2013)

Uma vez modelado, é possível fazer a execução do sistema fuzzy. A execução é realizada de forma isolada para cada motor de inferência do sistema. O cenário para este caso de uso é apresentado no Quadro 15.

Quadro 15 - UC-10. Executar projeto

<b>UC-10</b>	<b>Executar projeto</b>
<b>Descrição</b>	O usuário executa o projeto.
<b>Rastreabilidade</b>	RF-15; RF-16; RF-17; RN-16;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto. Projeto conter variáveis de entrada e motores cadastrados.
<b>Fluxo Principal</b>	1. O usuário seleciona a opção de simulação do sistema. (RN-16) 2. O usuário informa o valor das variáveis de entrada. 3. O sistema executa cada motor de inferência, calculando seus valores de saída. 4. O sistema apresenta os resultados em tela. 5.a. <Extend> <b>UC-11 Gerar gráficos</b> 5.b. <Extend> <b>UC-12 Gerar Fuzzy Control Language</b> 5.c. <Extend> <b>UC-13 Gerar classe Java</b>
<b>Exceções</b>	3.a. O sistema detecta alguma inconsistência na modelagem durante a execução. 3.a.1. O sistema exibe a mensagem “Erro na execução” 3.a.2. Retorna ao passo 2.

Fonte: Acervo do Autor (2013)

A partir da modelagem e execução do sistema fuzzy, o usuário é capaz de visualizar os gráficos de cada variável para a execução realizada. O cenário para este caso de uso encontra-se no Quadro 16.

Quadro 16 - UC-11. Gerar gráficos

<b>UC-11</b>	<b>Gerar gráficos</b>
<b>Descrição</b>	O usuário visualiza os gráficos de cada variável para o motor executado.
<b>Rastreabilidade</b>	RF-21; RN-18;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Usuário ter realizado a simulação do sistema.
<b>Fluxo Principal</b>	1. O usuário seleciona o motor. 2. O usuário seleciona a opção de geração de gráficos. 3. O sistema apresenta tela com os gráficos. (RN-18)

Fonte: Acervo do Autor (2013)

O usuário também tem acesso à notação FCL, de acordo com o padrão IEC 1131-7. A geração desta notação é contemplada pelo caso de uso UC-12. Seu cenário é apresentado no Quadro 17.

Quadro 17 - UC-12. Gerar Fuzzy Control Language

<b>UC-12</b>	<b>Gerar Fuzzy Control Language</b>
<b>Descrição</b>	O usuário visualiza a notação FCL para seu sistema modelado.
<b>Rastreabilidade</b>	RF-19;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Fluxo Principal</b>	1. O usuário seleciona o motor. 2. O usuário seleciona a opção de geração de FCL. 3. O sistema apresenta tela com a notação do motor.

Fonte: Acervo do Autor (2013)

A ferramenta propõe, além da geração da notação no padrão FCL, a geração de código de programação em linguagem Java. Nesta funcionalidade, o sistema produz uma classe Java que, fazendo uso da biblioteca *jFuzzyLogic*, solicita ao usuário valores para as variáveis de entrada e calcula as saídas, apresentando o processo de execução graficamente em tela. Esta funcionalidade está prevista no caso de uso UC-13, cujo cenário é apresentado no Quadro 18.

Quadro 18 - UC-13. Gerar classe Java

<b>UC-13</b>	<b>Gerar classe Java</b>
<b>Descrição</b>	O usuário visualiza a classe Java referente ao sistema modelado.
<b>Rastreabilidade</b>	RF-20; RN-17;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Pós-Condições</b>	Geração de código de programação Java.
<b>Fluxo Principal</b>	1. O usuário seleciona o motor. 2. O usuário seleciona a opção de geração de classe Java. 3. O sistema apresenta tela com a referida classe. (RN-17)

Fonte: Acervo do Autor (2013)

A ferramenta possui um espaço destinado à visualização gráfica dos componentes criados para o sistema. Nesta interface, além de outras funcionalidades, é possível mover os componentes (variáveis e motores) e dispor os mesmos em tela conforme a necessidade. O cenário para tal funcionalidade é apresentado no Quadro 19.

Quadro 19 - UC-14. Editar componentes visuais

<b>UC-14</b>	<b>Editar componentes visuais</b>
<b>Descrição</b>	O usuário dispõe os componentes em tela conforme sua necessidade.
<b>Rastreabilidade</b>	RF-05; RN-05; RN-06; RN-07;
<b>Atores</b>	Usuário
<b>Pré-Condições</b>	Projeto aberto.
<b>Pós-Condições</b>	Componentes dispostos conforme necessidade.

<b>Fluxo Principal</b>	<ol style="list-style-type: none"><li>1. O usuário clica e mantém botão pressionado sobre determinado componente.</li><li>2. O usuário arrasta componente para posição desejada dentro do quadro destinado à visualização gráfica.</li><li>3. O usuário solta o botão do <i>mouse</i>, soltando o componente na posição.</li><li>4. O sistema grava nova posição no banco de dados.</li></ol>
------------------------	---

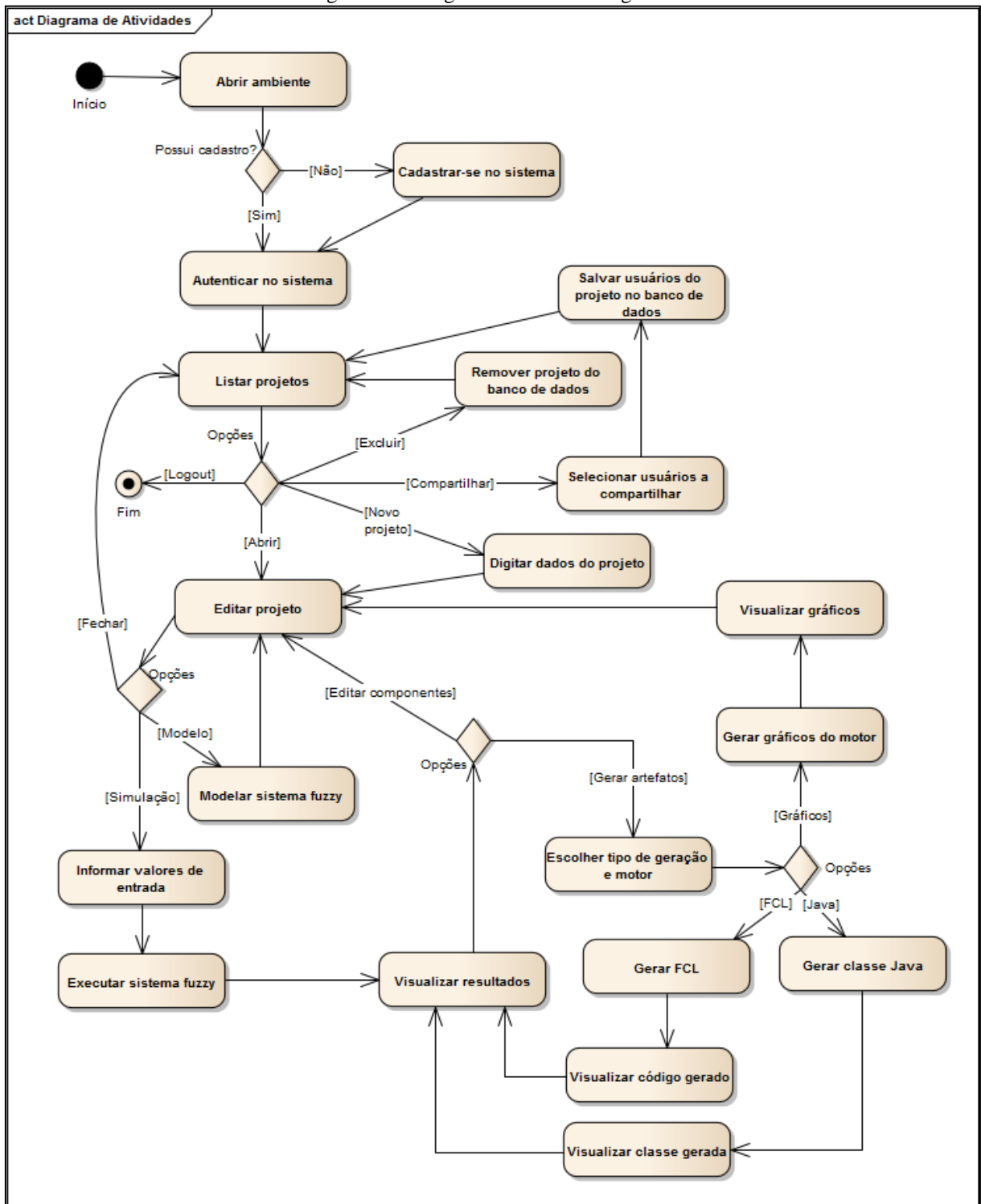
Fonte: Acervo do Autor (2013)

### 3.4 DIAGRAMA DE ATIVIDADES

Pressman (2011) discorre sobre a importância de prever e modelar o comportamento do sistema nos elementos de modelagem. “O comportamento de um sistema baseado em computador pode ter um profundo efeito sobre o projeto que é escolhido e a abordagem de implementação que é aplicada.” (PRESSMAN, 2011, p. 135). Conforme Bezerra (2002), o diagrama de atividades pode ser considerado um tipo de diagrama de estados. Contudo, o diagrama de atividades está orientado pelo fluxo de controle, ou seja, busca representar o fluxo do processo de execução ou utilização do sistema.

A Figura 15 apresenta o diagrama de atividades geral desenvolvido para a modelagem da ferramenta implementada neste trabalho. Nele é possível observar o fluxo de atividades, desde o momento em que o usuário acessa a aplicação, até o estágio de validação e geração de artefatos do sistema fuzzy modelado. O diagrama se inicia com o acesso do usuário ao ambiente, finalizando quando o mesmo fecha a aplicação.

Figura 15 – Diagrama de atividades geral



Fonte: Acervo do Autor (2013)

Uma vez acessada a aplicação, o usuário pode cadastrar-se no sistema ou, se já possuir cadastro, autenticar-se. Ao finalizar seu cadastro o usuário é automaticamente remetido à tela de autenticação. Quando o usuário se autentica no sistema, é apresentada a tela principal,

também chamada “Central do Usuário” onde estão listados todos os projetos do usuário. Neste momento o usuário tem a possibilidade de criar um novo projeto ou abrir algum já criado. O usuário é capaz de compartilhar um projeto com outros usuários ou ainda excluir um dos projetos existentes. Depois de criado é aberta a tela de edição de projetos. A mesma tela é retornada quando o usuário seleciona a opção de abrir um projeto já existente.

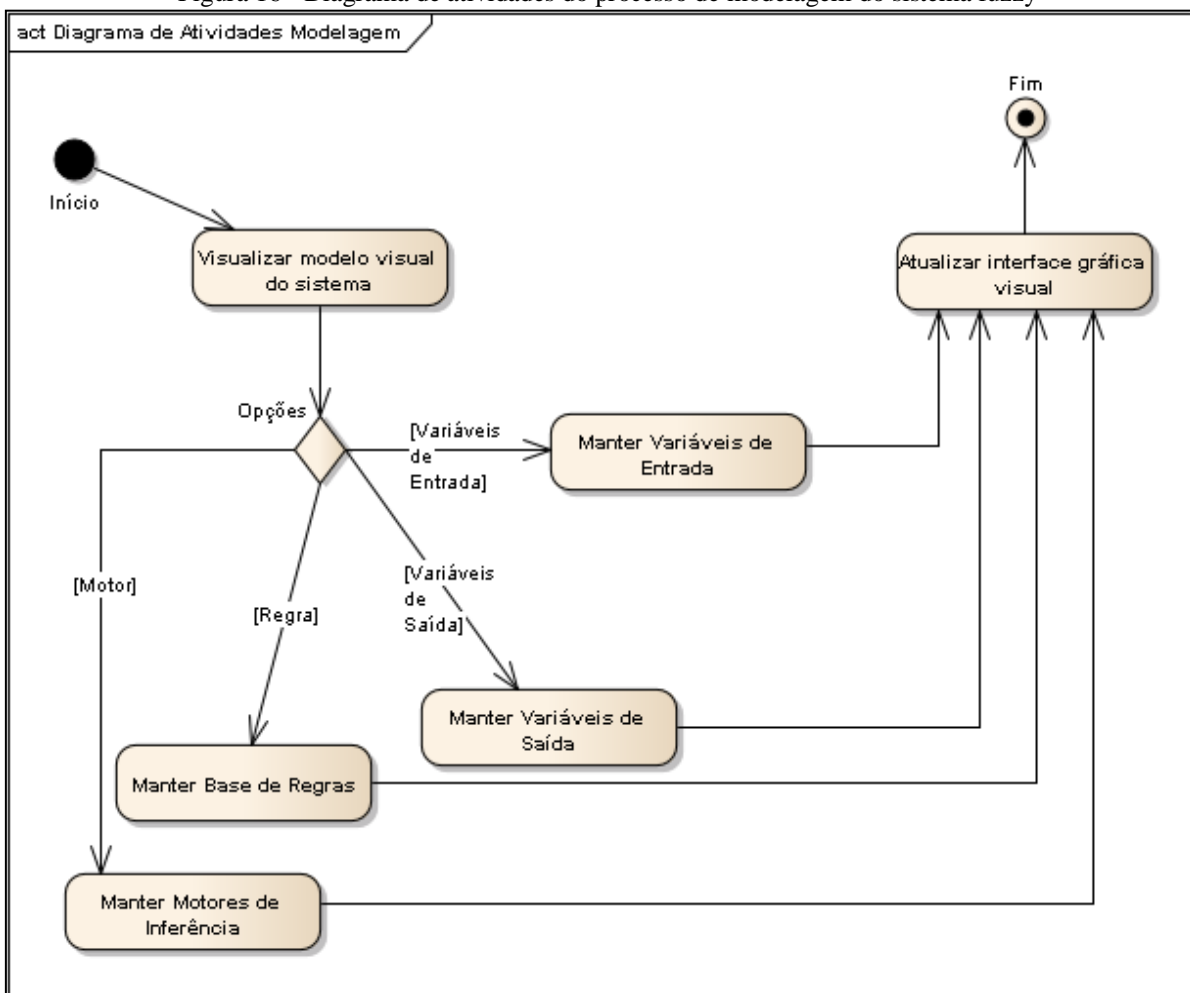
Na tela de edição de projetos é onde o usuário irá trabalhar no sistema fuzzy, modelando-o e simulando sua execução. Nesta interface se apresentam três opções: fechar o projeto; modelar o sistema fuzzy através da criação e manipulação de componentes ou simular o sistema através da execução de seus motores de inferência. Na modelagem é onde o usuário cria todos os componentes do sistema difuso (variáveis, regras e motores de inferência). Pela sua complexidade e importância e por compor o núcleo da ferramenta, seu fluxo de atividades é apresentado em um diagrama de atividades separado, apresentado na Figura 16. Na simulação o usuário informa o valor das variáveis de entrada e executa o sistema modelado, apresentando os resultados em tela através de valores e gráficos.

Na interface de simulação o usuário possui as funcionalidades de geração de artefatos relacionados ao sistema modelado. O usuário pode gerar os gráficos referentes à execução realizada, assim como código seguindo a notação FCL. O sistema ainda possibilita a geração de uma classe Java, a qual é preparada para executar o sistema modelado junto com a biblioteca *jFuzzyLogic*. Após a visualização dos resultados e/ou artefatos gerados, o usuário retorna à tela de edição e modelagem do projeto, através da qual é possível o fechamento do projeto e da aplicação.

A Figura 16 apresenta em detalhes a atividade “Modelar sistema fuzzy”, que se encontra no diagrama da Figura 15.



Figura 16 - Diagrama de atividades do processo de modelagem do sistema fuzzy



Fonte: Acervo do Autor (2013)

O processo de modelagem utiliza a interface gráfica visual, através da qual o usuário tem acesso à manutenção dos componentes que farão parte do sistema fuzzy. A tela principal de edição de projetos mostra o desenho do sistema fuzzy, onde serão realizadas as manutenções nos componentes do sistema: variáveis de entrada, variáveis de saída, regras e motores de inferência. Ao criar um novo componente, alterar um já existente ou realizar sua exclusão, o sistema atualiza os registros no banco de dados e redesenha o modelo visual, que é novamente apresentado ao usuário.

Com base nos diagramas de atividades observados nas Figuras 15 e 16, o Quadro 20 apresenta uma relação entre as atividades e casos de uso relacionados.

Quadro 20 - Relação entre atividades e casos de uso

<b>Atividade</b>	<b>Caso de uso</b>
Abrir ambiente	UC02
Cadastrar-se no sistema	UC01; UC03
Autenticar no sistema	UC02
Listar projetos	UC04
Editar projeto	UC04
Digitar dados do projeto	UC04
Selecionar usuários a compartilhar	UC05
Remover projeto do banco de dados	UC04
Visualizar modelo visual do sistema	UC14
Modelar sistema fuzzy	UC06; UC07; UC08; UC09
Manter variáveis de entrada	UC06; UC07
Manter variáveis de saída	UC06; UC07
Manter base de regras	UC08
Manter motores de inferência	UC09
Atualizar interface gráfica visual	UC14
Informar valores de entrada	UC10
Executar sistema fuzzy	UC10
Visualizar resultados	UC10
Escolher tipo de geração e motor	UC12; UC13
Gerar FCL / Classe Java	UC12; UC13
Visualizar código gerado	UC12; UC13
Gerar gráficos do motor	UC11
Visualizar gráficos	UC11

Fonte: Acervo do Autor (2013)

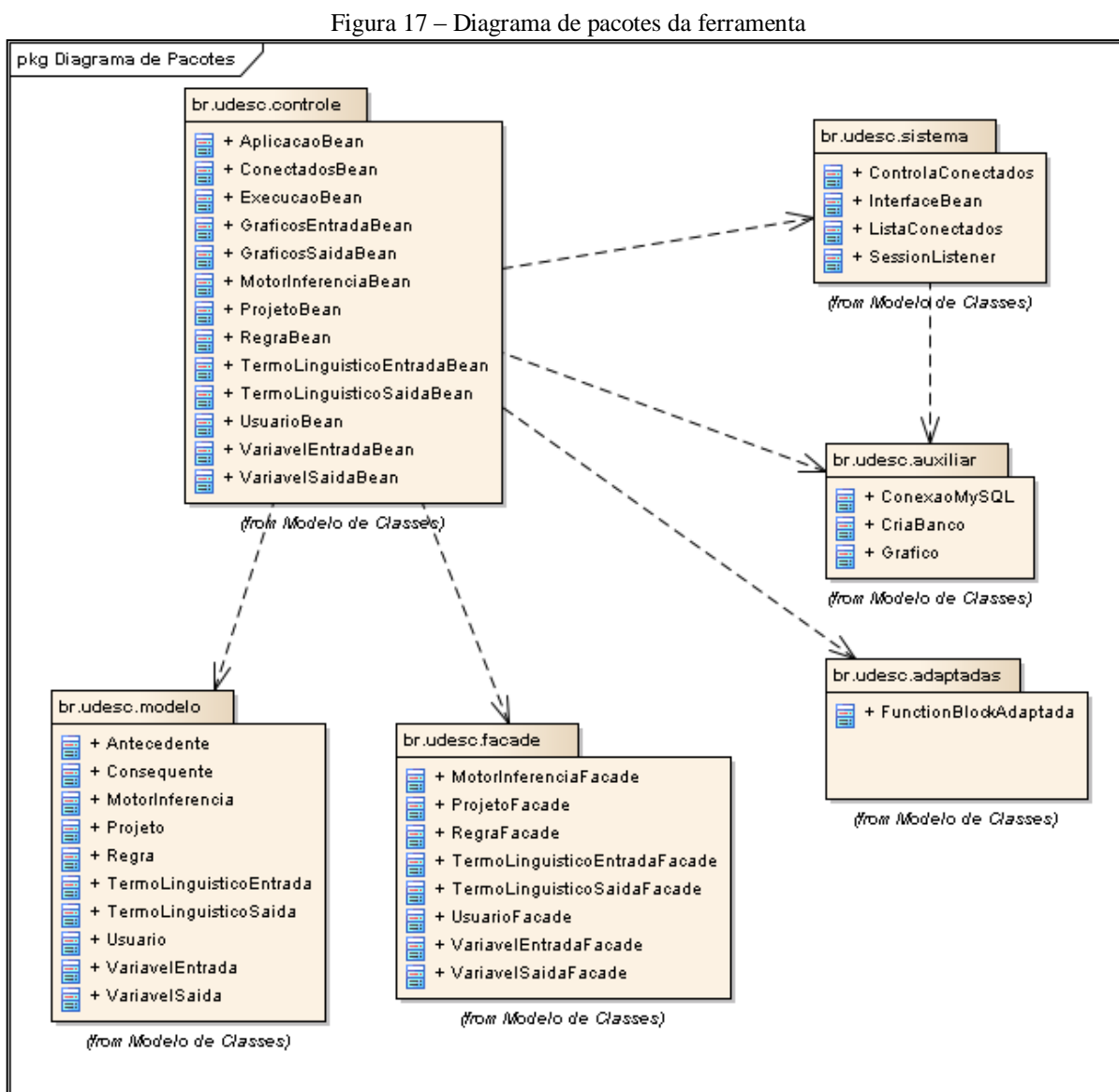
### 3.5 DIAGRAMA DE CLASSES

Um diagrama de classes descreve os tipos de objetos que compõem um sistema de software e os relacionamentos que existem entre si. Ao nível mais básico da especificação da UML, um diagrama de classes apresenta as classes que formam o sistema, seu nome, suas propriedades e operações (FOWLER, 2005).

Por outro lado, Fowler (2005) aborda a questão de que um diagrama de classes não é suficiente para estruturar e representar de forma satisfatória um sistema de grande porte. Um sistema pode ser composto por centenas de classes e a visualização de diagramas dessa natureza pode não representar com clareza a arquitetura do software. Para uma visualização mais clara da estrutura interna do sistema é utilizado o diagrama de pacotes. Um pacote pode ser entendido como o agrupamento de elementos da UML em unidades de nível mais alto.

Um uso comum desse tipo de diagrama é agrupar classes em pacotes, representando a arquitetura de classes de forma mais abstrata possível.

A Figura 17 apresenta o diagrama de pacotes da arquitetura da ferramenta. Através deste diagrama, é possível observar como as classes foram organizadas no sistema e qual a dependência entre os distintos pacotes criados.



Fonte: Acervo do Autor (2013)

O sistema segue a arquitetura MVC (Model View Controller). O MVC é um padrão onde a aplicação é composta por três tipos de objetos: modelo, visão e controlador. As responsabilidades são separadas entre estes elementos, isolando cada parte do sistema. Este padrão é adotado para aumentar a flexibilidade e reutilização de um sistema de software

(GAMMA et al., 1995). As classes do pacote *br.udesc.controle* realizam o papel de controladores do sistema, ou seja, fazem a comunicação e o controle das informações entre a interface gráfica e demais classes do sistema. As classes do pacote *br.udesc.modelo* definem as entidades do sistema, assim como mapeiam estas com suas respectivas tabelas do banco de dados.

Como camada de persistência se encontram as classes do pacote *br.udesc.facade*, que utilizam este padrão de projetos (*Façade*) para fornecer métodos de acesso ao banco de dados. Conforme Gamma et al. (1995), o padrão *Façade* busca “fornecer uma interface unificada para um conjunto de interfaces de um subsistema. O *Façade* define uma interface de mais alto nível, tornando assim mais fácil utilizar o subsistema”. No pacote *br.udesc.facade*, as classes são responsáveis por efetivar o acesso ao banco de dados em rotinas de gravação, alteração ou consulta de registros, por exemplo.

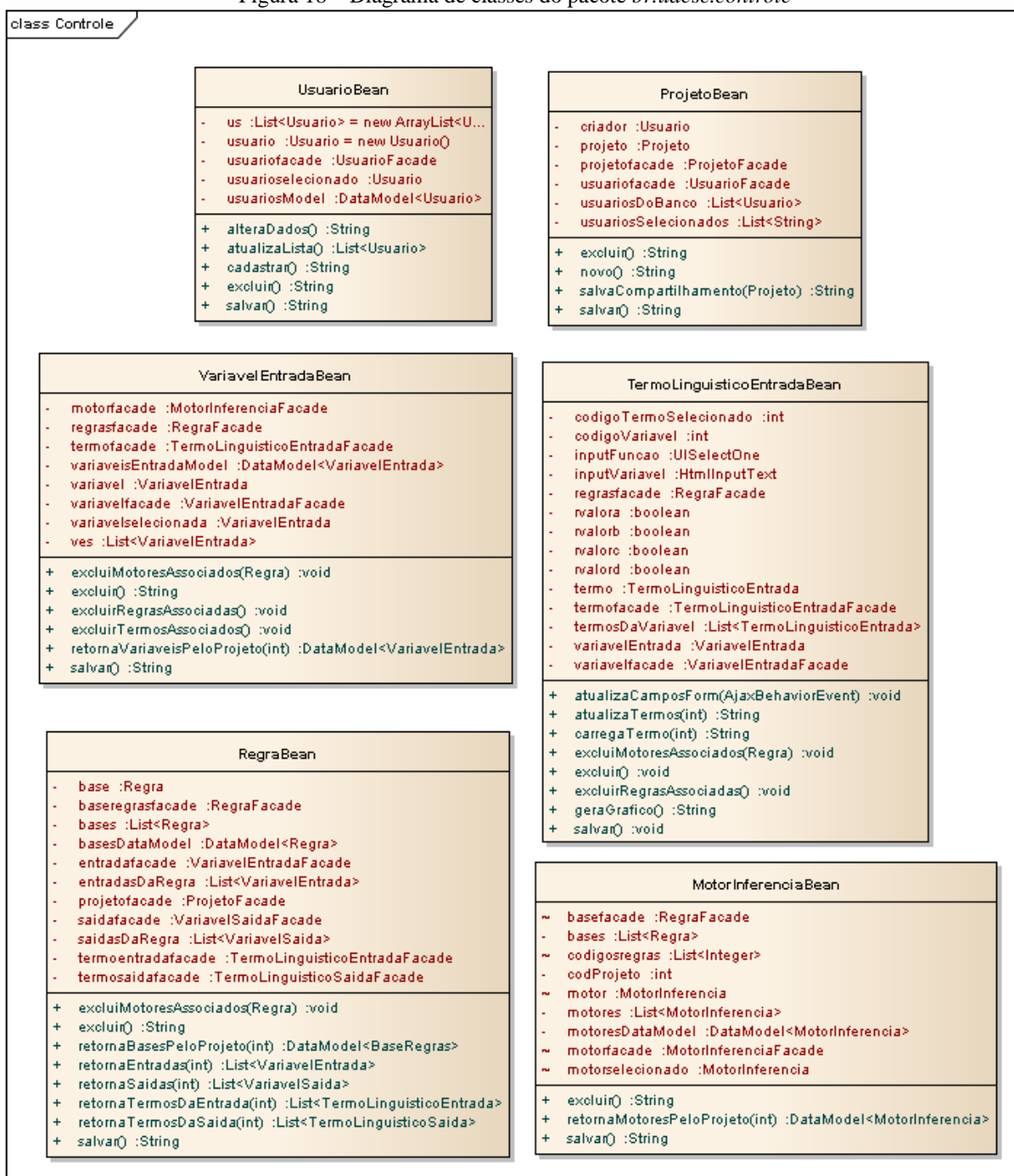
O pacote chamado *br.udesc.sistema* fornece algumas classes para controlar as sessões ativas, gerenciar os usuários conectados a determinado projeto, estruturar os gráficos e redesenhar componentes visuais em tela. O pacote *br.udesc.auxiliar* possui classes utilitárias para acesso direto ao banco de dados, necessárias nas requisições AJAX<sup>2</sup> através dos arquivos JavaScript e criação automática das tabelas. Por fim, o pacote *br.udesc.adaptadas* possui uma classe oriunda da biblioteca *jFuzzyLogic*<sup>3</sup>. Esta classe foi adaptada para possibilitar a execução de algumas funcionalidades em ambiente web no momento de executar o sistema fuzzy modelado e gerar seus artefatos.

As Figuras 18 a 23 apresentam os diagramas de classes dos pacotes da Figura 16, detalhando as responsabilidades de cada classe no sistema. Nesses diagramas, foram mantidos apenas os métodos mais importantes de cada classe, para fins de melhor visualização.

---

<sup>2</sup> AJAX é o acrônimo para Asynchronous JavaScript and XML. É uma técnica que organiza código fonte do lado servidor e JavaScript no lado cliente. É utilizado para carregamento e renderização de uma página de forma dinâmica, sem a necessidade de recarregá-la (HOLZNER, 2007).

<sup>3</sup> Mais informações sobre a biblioteca serão abordadas na seção 3.7.2.1.

Figura 18 – Diagrama de classes do pacote *br.udesc.controle*

Fonte: Acervo do Autor (2013)

A Figura 18 contém o diagrama de classes do pacote *br.udesc.controle*, contendo as classes controladoras para as entidades do sistema. Essas classes são responsáveis pela comunicação entre a camada de visão (interface com o usuário) com as classes de modelo e persistência de cada entidade do sistema. A classe *UsuarioBean*, por exemplo, é responsável por controlar a interface de cadastro de usuários e prover sua comunicação com a classe

correspondente da camada de modelo (*br.udesc.modelo.Usuario*). Já a classe *VariavelEntradaBean* trata de controlar a manutenção de variáveis de entrada do sistema, comunicando a interface correspondente com a classe modelo, neste caso *br.udesc.modelo.VariavelEntrada*. O mesmo comportamento é observado nas demais classes.

No diagrama apresentado pela Figura 18, as classes *VariavelSaidaBean* e *TermoLinguisticoSaidaBean* foram suprimidas, por serem idênticas às classes *VariavelEntradaBean* e *TermoLinguisticoEntradaBean*, respectivamente. De acordo com a arquitetura adotada no desenvolvimento do projeto, o relacionamento entre classes ocorre na camada de modelo, neste caso *br.udesc.modelo*. As camadas de controle e persistência não apresentam relacionamento direto entre seus componentes.

A Figura 19 apresenta classes do mesmo pacote (*br.udesc.controle*). Essas classes são responsáveis pelo controle de outros aspectos da aplicação, como a gestão dos usuários conectados à aplicação, controle das execuções dos motores e geração dos artefatos.

Figura 19 - Diagrama de classes do pacote *br.udesc.controle*



Fonte: Acervo do Autor (2013)

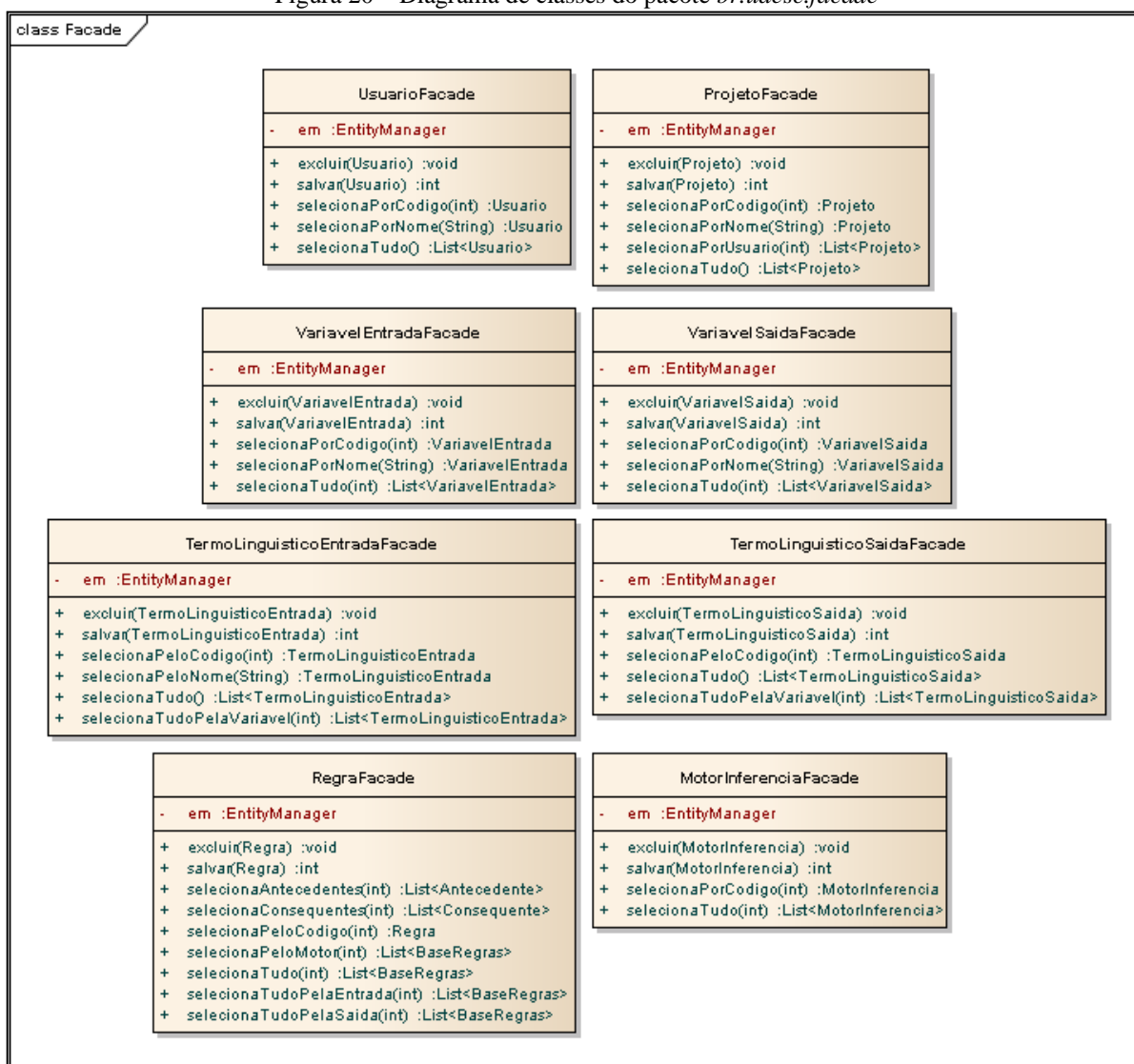
O Quadro 21 apresenta um resumo da responsabilidade de cada uma das classes apresentadas na Figura 19.

Quadro 21 - Descrição das classes de controle de sistema

Classe	Descrição (responsabilidades)
AplicacaoBean	Classe responsável por armazenar dados da sessão corrente. Armazena o usuário autenticado e projeto em edição. Também controla a lista de projetos disponíveis e controle de acesso.
ConectadosBean	Responsável por controlar os usuários que estão conectados na aplicação e o projeto que cada usuário possui em edição (aberto). Esta classe é utilizada para permitir o trabalho colaborativo em tempo real entre os usuários de um projeto compartilhado.
ExecucaoBean	Classe responsável por controlar a execução do sistema fuzzy modelado. Controla as entradas informadas pelo usuário, a execução independente de cada motor, geração do padrão FCL, da classe Java correspondente, geração dos gráficos para cada variável e apresentação dos resultados ao usuário.

Fonte: Acervo do Autor (2013)

O pacote *br.udesc.facade* é composto por classes que fornecem métodos e operações das entidades com o banco de dados. De maneira geral, há uma classe *Façade* para cada entidade do sistema, fornecendo métodos de inclusão, alteração, exclusão e consulta de objetos junto ao banco de dados. O diagrama de classes do pacote *br.udesc.facade* para o projeto é apresentado na Figura 20.

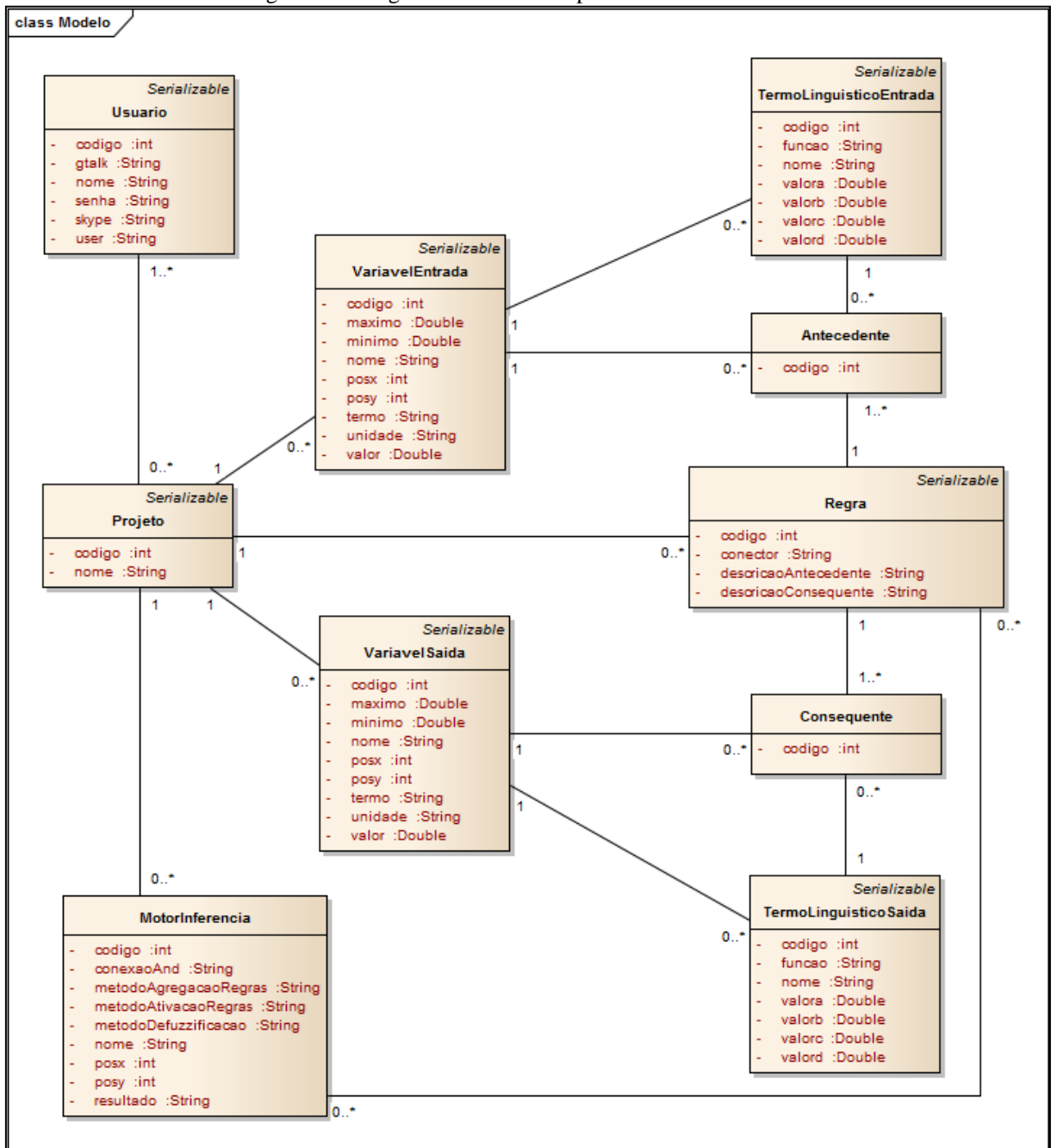
Figura 20 – Diagrama de classes do pacote *br.udesc.facade*

Fonte: Acervo do Autor (2013)

O pacote *br.udesc.modelo* possui as classes que modelam as entidades e as mapeiam com as tabelas do banco de dados através da especificação JPA<sup>4</sup>. A Figura 21 apresenta o diagrama de classes para o pacote *br.udesc.modelo*. Os métodos acessores foram suprimidos do diagrama.

<sup>4</sup> JPA é definido na seção 3.7.1.1.



Figura 21 - Diagrama de classes do pacote *br.udesc.modelo*

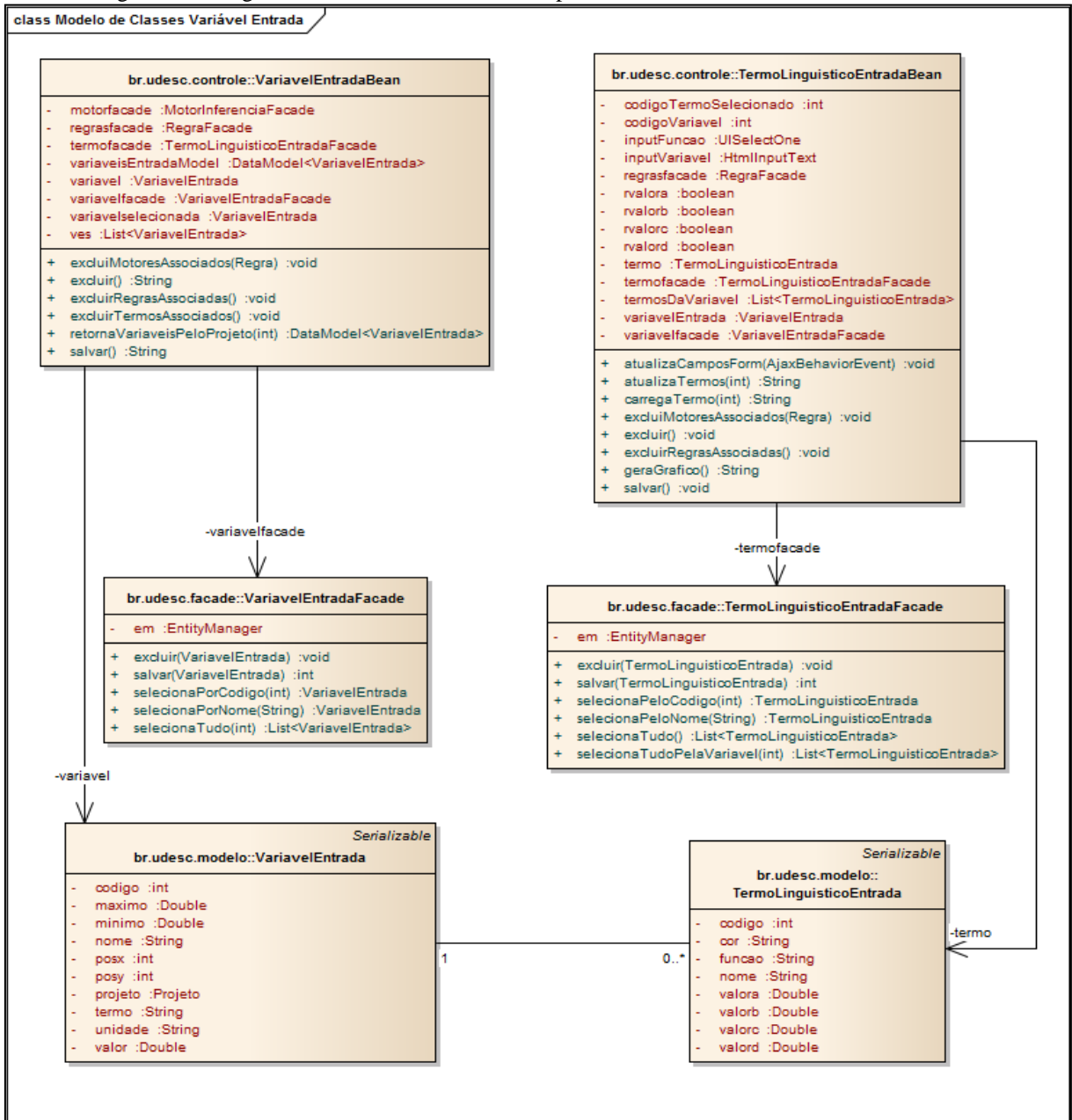
Fonte: Acervo do Autor (2013)

A entidade central do diagrama é a classe *Projeto*, a qual se relaciona com todos os outros componentes do sistema fuzzy. As variáveis de entrada contém termos linguísticos de entrada. O mesmo ocorre para as variáveis de saída e termos linguísticos de saída. A base de regras se relaciona com as variáveis por meio das classes *Antecedente* e *Consequente*, que compõem uma regra. A entidade *MotorInferencia* contém um conjunto de regras, as quais utiliza no momento de inferir os resultados.

A configuração do sistema fuzzy (método de defuzzificação, agregação de regras, etc.) está modelada no motor. Assim cada motor executa independentemente, facilitando múltiplos testes e aferições.

Os diagramas de classe das Figuras 22 e 23 exemplificam a relação entre as classes de distintos pacotes e suas dependências. A Figura 22 apresenta a relação entre as classes envolvidas no cadastro de variáveis de entrada e seus respectivos termos linguísticos.

Figura 22 - Diagrama de classes relacionadas ao processo de cadastro de variáveis de entrada



O Quadro 22 apresenta a descrição das classes apresentadas no diagrama da Figura 22 e suas responsabilidades.

Quadro 22 - Descrição das classes envolvidas no processo de cadastro de variáveis de entrada

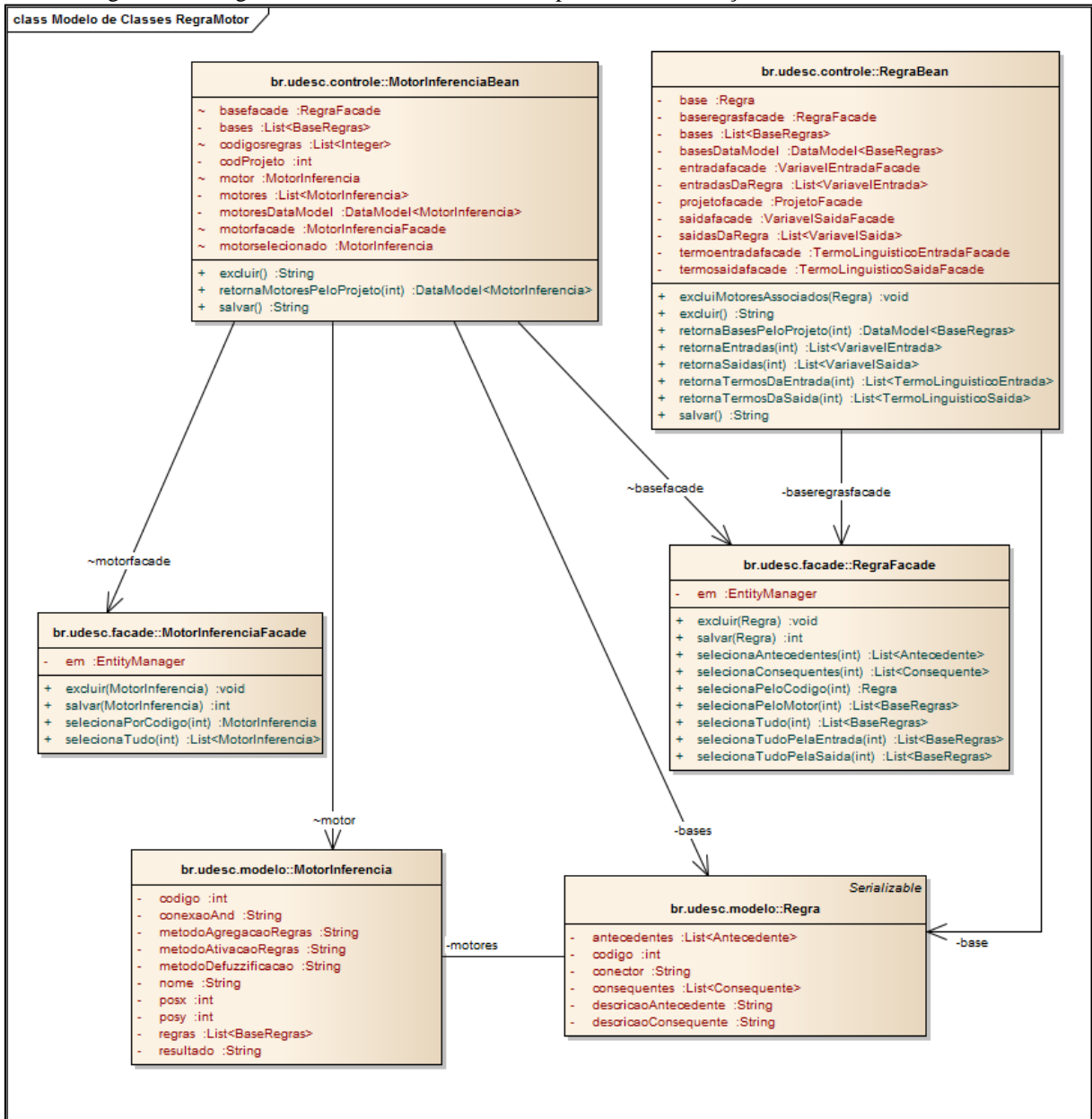
Classe	Descrição (responsabilidades)
VariavelEntradaBean	Responsável pela comunicação entre a interface com o usuário e as classes de modelo e persistência das variáveis de entrada.
TermoLinguisticoEntradaBean	Responsável pela comunicação entre a interface com o

	usuário e as classes de modelo e persistência dos termos linguísticos de entrada.
VariavelEntradaFacade	Classe responsável pelas operações das variáveis de entrada com o banco de dados (inclusão, alteração, exclusão e consulta).
TermoLinguisticoEntradaFacade	Classe responsável pelas operações dos termos linguísticos de entrada com o banco de dados (inclusão, alteração, exclusão e consulta).
VariavelEntrada	Classe que modela uma variável de entrada e mapeia seus atributos com colunas na tabela correspondente.
TermoLinguisticoEntrada	Classe que modela um termo linguístico de entrada e mapeia seus atributos com colunas na tabela correspondente.

Fonte: Acervo do Autor (2013)

A Figura 23 apresenta o diagrama de classes relacionado com o processo de cadastro e vinculação de regras e motores de inferência.

Figura 23 - Diagrama de classes relacionadas ao processo manutenção de motores de inferência



Fonte: Acervo do Autor (2013)

O Quadro 23 apresenta a função de cada uma das classes relacionadas no diagrama da Figura 23.

Quadro 23 - Descrição das classes envolvidas no cadastro e vinculação de regras e motores de inferência

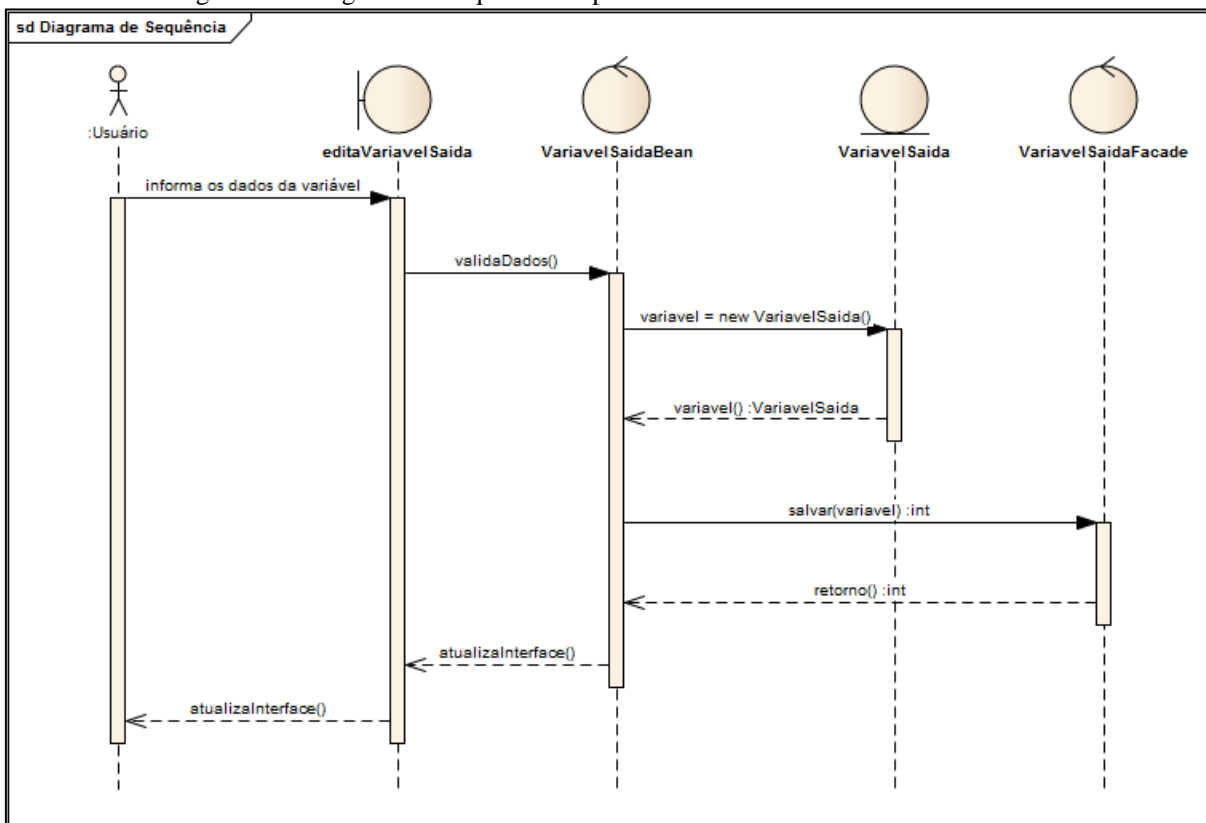
Classe	Descrição (responsabilidades)
RegraBean	Classe responsável por comunicar a interface com o usuário com as classes de modelo e persistência relacionadas à base de regras.
MotorInferenciaBean	Classe responsável por comunicar a interface com o usuário com as classes de modelo e persistência relacionadas ao motor de inferência.

RegraFacade	Classe responsável pelas operações da base de regras com o banco de dados (inclusão, alteração, exclusão e consulta).
MotorInferenciaFacade	Classe responsável pelas operações do motor de inferência com o banco de dados (inclusão, alteração, exclusão e consulta).
Regra	Classe que modela a entidade da base de regras e mapeia seus atributos à colunas da referente tabela do banco de dados.
MotorInferencia	Classe que modela a entidade do motor de inferência e mapeia seus atributos à colunas da referente tabela do banco de dados.

Fonte: Acervo do Autor (2013)

Buscando exemplificar o relacionamento entre as classes dos distintos pacotes, foi elaborado um diagrama de sequência. A Figura 24 apresenta o referido diagrama com base no processo de cadastro de variáveis de saída.

Figura 24 – Diagrama de sequência do processo de cadastro de variáveis de saída



Fonte: Acervo do Autor (2013)

Na Figura 24 é ilustrado o momento em que o usuário informa os dados da variável a ser cadastrada, comunicando-se com a interface do sistema. Ao solicitar o salvamento da variável, os dados são validados pela interface, que os envia para a classe controladora da variável de saída (*VariavelSaidaBean*). A classe controladora solicita um novo objeto da

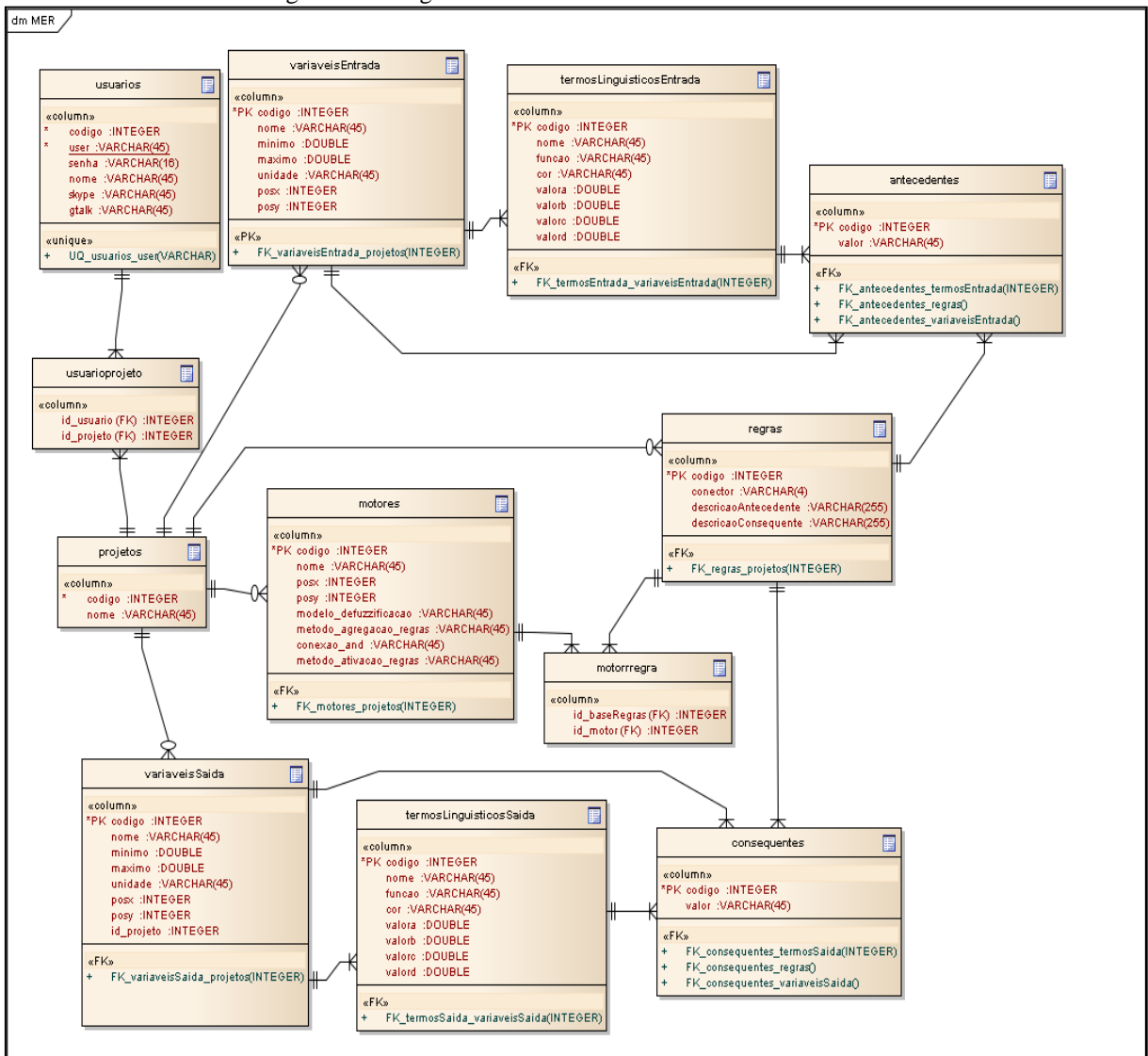
classe *VariavelSaida* e o popula com os dados recebidos da interface. Este objeto é persistido no banco de dados através da classe *VariavelSaidaFacade*, a qual fornece todos os métodos de acesso e manipulação com o banco de dados. O controlador envia o retorno para a interface, atualizando-a.

### 3.6 MODELO ENTIDADE RELACIONAMENTO

Mecenas e Oliveira (2005) explanam o modelo entidade relacionamento decompondo-o em suas duas partes fundamentais: entidades e relacionamentos. Uma entidade é entendida como um conjunto de elementos do domínio da aplicação, do qual a mesma faz uso para seu correto funcionamento. Relacionamentos são as associações entre os referidos elementos, ou seja, a interação e dependência entre os mesmos.

O modelo entidade relacionamento do presente trabalho é apresentado na Figura 25.

Figura 25 - Diagrama entidade relacionamento da ferramenta



Fonte: Acervo do Autor (2013)

O modelo entidade relacionamento reflete o diagrama de classes do pacote *br.udesc.modelo*. Para cada classe entidade do sistema é gerada uma tabela do banco de dados, mais suas devidas normalizações. A entidade *projetos* é uma das tabelas centrais do banco de dados, com a qual todos os componentes estão, direta ou indiretamente relacionados. A tabela *projetos* tem uma relação de muitos-para-muitos com a tabela *usuarios*, pois um usuário pode ter vários projetos e estes, por sua vez, podem se vincular a um ou mais usuários. As tabelas *variaveisEntrada* e *variaveisSaida* possuem relação com as tabelas *termosLinguisticosEntrada* e *termosLinguisticosSaida* respectivamente, uma vez que as variáveis são compostas por termos linguísticos que expressam as funções de fuzzificação e defuzzificação.



As variáveis ainda se relacionam com as tabelas *antecedentes* e *consequentes*, as quais são criadas a fim de formar uma regra (tabela *regras*). A tabela *regras* se relaciona com a tabela *motores* com relacionamento muitos-para-muitos, uma vez que um motor pode ter várias regras associadas e uma regra pode estar vinculada a mais de um motor de inferência. O Quadro 24 apresenta o dicionário de dados para o modelo apresentado na Figura 25.

Quadro 24 - Dicionário de dados para as entidades do sistema

<b>Tabela usuarios</b>	
codigo	Identificador do registro.
user	Nome de usuário para acesso ao sistema.
senha	Senha para acesso ao sistema.
nome	Nome do usuário.
skype	Informação de contato – Skype.
gtalk	Informação de contato GoogleTalk
<b>Tabela projetos</b>	
codigo	Identificador do registro.
nome	Nome do projeto.
<b>Tabela variaveisEntrada</b>	
codigo	Identificador do registro.
nome	Nome da variável de entrada.
minimo	Valor mínimo para a variável.
maximo	Valor máximo para a variável.
unidade	Unidade de medida da variável.
posx	Posição de desenho na visualização gráfica referente ao eixo x.
posy	Posição de desenho na visualização gráfica referente ao eixo y.
<b>Tabela termosLinguisticosEntrada</b>	
codigo	Identificador do registro.
nome	Nome do termo linguístico.
funcao	Função de pertinência.
cor	Cor para pintura do gráfico.
valora	Valor do parâmetro A da função de pertinência.
valorb	Valor do parâmetro B da função de pertinência.
valorc	Valor do parâmetro C da função de pertinência.
valord	Valor do parâmetro D da função de pertinência.
<b>Tabela variaveisSaida</b>	
codigo	Identificador do registro.
nome	Nome da variável de saída.
minimo	Valor mínimo para a variável.
maximo	Valor máximo para a variável.
unidade	Unidade de medida da variável.

posx	Posição de desenho na visualização gráfica referente ao eixo x.
posy	Posição de desenho na visualização gráfica referente ao eixo y.
<b>Tabela termosLinguisticosSaida</b>	
codigo	Identificador do registro.
nome	Nome do termo linguístico.
funcao	Função de pertinência.
cor	Cor para pintura do gráfico.
valora	Valor do parâmetro A da função de pertinência.
valorb	Valor do parâmetro B da função de pertinência.
valorc	Valor do parâmetro C da função de pertinência.
valord	Valor do parâmetro D da função de pertinência.
<b>Tabela regras</b>	
codigo	Identificador do registro.
conector	Conector lógico para condições do antecedente.
descricaoAntecedente	Descrição completa do antecedente.
descricaoConsequente	Descrição completa do consequente.
<b>Tabela motores</b>	
codigo	Identificador do registro.
nome	Nome do motor de inferência.
posx	Posição de desenho na visualização gráfica referente ao eixo x.
posy	Posição de desenho na visualização gráfica referente ao eixo y.
modelo_defuzzificacao	Modelo de defuzzificação para as saídas.
metodo_agregacao_regras	Método de agregação das regras na inferência.
conexao_and	Conector AND para as premissas.
metodo_ativacao_regras	Método de ativação das regras na inferência.

Fonte: Acervo do Autor (2013)

### 3.7 IMPLEMENTAÇÃO

Esta subseção apresenta conceitos importantes para a fase de implementação da ferramenta. Em um primeiro momento são abordadas as técnicas utilizadas no desenvolvimento do software tais como tecnologias utilizadas, padrões seguidos e métodos empregados. Em seguida são apresentados detalhes da implementação dos mecanismos de execução de sistemas fuzzy modelados pelo usuário, bem como do funcionamento da interface de desenho da aplicação.

### 3.7.1 Tecnologias utilizadas

A ferramenta foi desenvolvida na linguagem de programação Java, utilizando o ambiente de desenvolvimento integrado Netbeans IDE na sua versão 7.1 e JDK 1.6. O sistema operacional utilizado foi o Mac OS X Lion, na sua versão 10.7.4.

Para desenvolvimento do ambiente foi utilizada a especificação Java Server Faces (JSF 2.0). Para o desenvolvimento da interface gráfica foi adotada a biblioteca de componentes ricos Primefaces 3.3.1, sendo utilizado o tema Delta.

O banco de dados adotado foi o MySQL e o servidor de aplicações Glassfish na sua versão 3.1.

#### 3.7.1.1 Java Persistence API (JPA)

A Java Persistence API é um *framework* para persistência Java baseado em POJO<sup>5</sup> (*Plain Old Java Object*). Seu uso é comum e difundido em aplicações Java por fornecer um mapeamento objeto-relacional que facilita a integração dos mecanismos de persistência de dados em aplicações escaláveis (KEITH; SCHINCARIOL, 2009). Na ferramenta FuzzyStudio, utiliza-se o Hibernate como implementação da especificação JPA.

O Hibernate é uma “implementação dos padrões do Java Persistence API que substitui antigas soluções de persistência Java” (LINWOOD; MINTER, 2010, p. 1). Ele possui seus metadados armazenados em arquivos XML e linguagem de consulta própria. Sua grande característica é, apesar de implementar a especificação JPA, ser flexível e independente de *framework*. Dessa forma, pode ser executado utilizando qualquer servidor de aplicações Java EE/J2EE.

O JPA define o mapeamento das classes Java diretamente com as tabelas do banco de dados por meio de anotações. Keith e Schincariol (2009) abordam essa especificação de mapeamento como fator determinante na facilidade de uso dos padrões JPA/Hibernate. Essa nova maneira de definir metadados utiliza anotações *type-safe*<sup>6</sup> embutidas diretamente nas

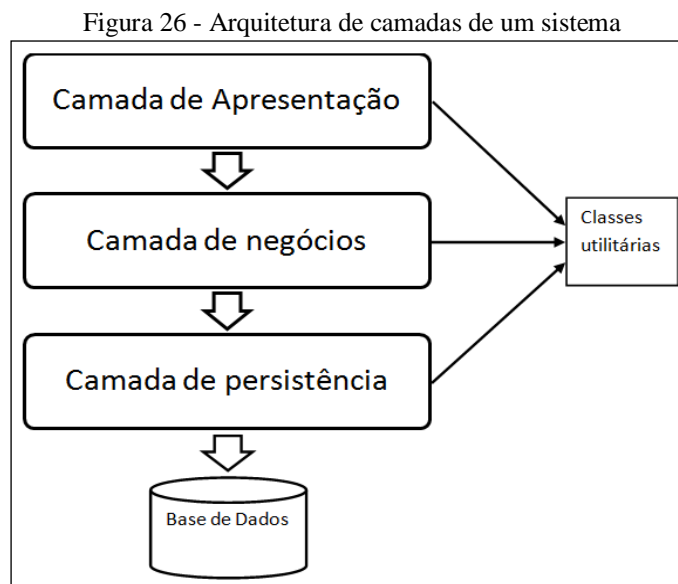
---

<sup>5</sup> POJO é a nomenclatura definida para a criação de classes simplificadas, dentro de um padrão definido. Um objeto POJO não é dependente de nenhum framework ou herança (SAM-BODDEN, 2006).

<sup>6</sup> As anotações *type-safe* garantem a correta implementação dos objetos de consulta Java, uma vez que o compilador analisa sua sintaxe.

classes Java. Essa possibilidade está disponível desde a versão 5.0 do JDK e substitui uma quantidade de arquivos XML utilizados até então para o mapeamento.

A camada de persistência utilizando JPA pode ser entendida conforme apresentado na Figura 26.



Fonte: Adaptado de Bauer e King (2007)

Percebe-se que a camada de persistência é a base da arquitetura de um sistema de software. Neste sentido, exerce um papel de extrema importância no sistema, merecendo uma especial atenção na sua implementação. Na ferramenta desenvolvida ao longo deste trabalho, foram utilizadas as especificações do JPA 2.0 e a implementação Hibernate 3.0 para a persistência dos dados.

As classes entidade do pacote *br.udesc.modelo* são mapeadas com suas tabelas do banco de dados através das anotações previstas no JPA. Essas anotações pertencem ao pacote *javax.persistence*. O Quadro 25 exemplifica o mapeamento para a classe *VariavelEntrada*.

Quadro 25 - Trecho do código da classe VariavelEntrada

```
[...]
@Entity
@Table(name="variaveisEntrada")
public class VariavelEntrada implements Serializable{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column
    private int codigo;

    @Column
    private String nome;

    @Column
    private Double minimo;

    @Column
    private Double maximo;

    @Column
    private String unidade;
[...]
```

Fonte: Acervo do Autor (2013)

O trecho de código apresentado no Quadro 25 exemplifica o mapeamento de uma entidade do modelo com sua respectiva tabela no banco de dados. A anotação *@Entity* define que a classe trata-se de uma entidade persistente e mapeada para uma tabela da base de dados. A anotação *@Table* define a tabela da entidade, onde o parâmetro apresentado (opcional) informa o nome da tabela. O *@Id* identifica quais dos atributos compõem a chave primária da tabela, seguido da anotação *@GeneratedValue*, definindo seu valor como auto incremento (gerado automaticamente pelo SGBD). As anotações *@Column* mapeiam cada atributo da classe para uma coluna do banco de dados. Opcionalmente se pode definir características para cada coluna como seu nome, tamanho, se aceita valores nulos, entre outros.

Nos mapeamentos também são definidos os relacionamentos entre as entidades. Para isso são utilizadas as anotações *@OneToMany* (mapeia relacionamentos um-para-muitos), *@ManyToOne* (relacionamentos muitos-para-um) e *@ManyToMany* (relacionamentos muitos-para-muitos). O Quadro 26 exemplifica a utilização de anotações para a definição de relacionamentos entre entidades. No exemplo é apresentado um trecho da classe *MotorInferencia*, que possui um relacionamento de muitos-para-um com a tabela de *projetos* através do atributo *projeto*. O relacionamento definido pela anotação *@ManyToOne* também faz uso da anotação *@JoinColumn*, informando o nome da coluna que representa a chave estrangeira e a propriedade de não aceitar valores nulos.

Esta mesma classe apresentada no Quadro 26 também possui um relacionamento de muitos-para-muitos com a tabela de regras (mapeada através do atributo *regras*). Esse relacionamento é descrito através da anotação *@ManyToMany* e *@JoinTable*. Esta última define a tabela que será criada para comportar a relação muitos-para-muitos, determinando o nome da tabela e as duas chaves estrangeiras.

Quadro 26 - Trecho do código da classe MotorInferencia

```
@ManyToOne
@JoinColumn(name="id_projeto", nullable=false)
private Projeto projeto;

@ManyToMany
@JoinTable(name="motorregra",
    joinColumns={@JoinColumn(name="id_motor")},
    inverseJoinColumns={@JoinColumn(name="id_regra")})
private List<Regras> regras = new ArrayList<Regras>();
```

Fonte: Acervo do Autor (2013)

Para a construção da camada de persistência da aplicação foi criada uma unidade de persistência<sup>7</sup>. O objetivo da utilização de uma unidade de persistência é definir qual implementação do JPA será utilizada, os parâmetros de conexão com o banco de dados e quais classes serão mapeadas. A unidade de persistência é definida através de um arquivo XML. O arquivo referente à ferramenta desenvolvida neste trabalho é apresentado no Quadro 27.

<sup>7</sup> A unidade de persistência configura as informações pertinentes ao provedor do JPA. As unidades de persistência são mapeadas pelo arquivo *persistence.xml* (KEITH; SCHINCARIOL, 2009).

Quadro 27 - Conteúdo do arquivo *persistence.xml* que define a unidade de persistência

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" [...]>
  <persistence-unit name="FSPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>FuzzyStudioDS</jta-data-source>
    <class>br.udesc.modelo.Projeto</class>
    <class>br.udesc.modelo.Usuario</class>
    <class>br.udesc.modelo.VariavelEntrada</class>
    <class>br.udesc.modelo.VariavelSaida</class>
    <class>br.udesc.modelo.TermoLinguisticoEntrada</class>
    <class>br.udesc.modelo.TermoLinguisticoSaida</class>
    <class>br.udesc.modelo.Regra</class>
    <class>br.udesc.modelo.MotorInferencia</class>
    <class>br.udesc.modelo.Antecedente</class>
    <class>br.udesc.modelo.Consequente</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="eclipselink.ddl-generation" value="create-tables"/>
    </properties>
  </persistence-unit>
</persistence>

```

Fonte: Acervo do Autor (2013)

Uma vez definida a unidade de persistência e mapeadas as classes com as tabelas do banco de dados, dois outros recursos do foram utilizados para compor a camada de persistência da aplicação, aproveitando funcionalidades do servidor Glassfish. O primeiro deles trata-se de um *pool* de conexões, através do qual o servidor se encarrega de fornecer uma conexão à aplicação quando necessário, melhorando o desempenho de todo o sistema. O segundo recurso do servidor utilizado trata-se da criação de um *DataSource*, ou fonte de dados. Este processo elimina as configurações de acesso ao banco da aplicação, tornando-a portátil. Ambas as definições estão contidas em um arquivo XML responsável por armazenar as informações acerca dos recursos utilizados do servidor (Quadro 28). Pode-se observar a definição do *DataSource* pelo elemento `<jdbc-resource>`. O nome ali definido, que neste caso é *FuzzyStudioDS*, é utilizado para definir na unidade de persistência através do elemento `<jta-data-source>`, ilustrado no Quadro 27.

Quadro 28 - Conteúdo do arquivo *glassfish-resources* que define os recursos do servidor

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server
3.1 Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-
resources_1_5.dtd">
<resources>
  <jdbc-connection-pool [...]>
    <property name="serverName" value="localhost"/>
    <property name="portNumber" value="3306"/>
    <property name="databaseName" value="fuzzy"/>
    <property name="User" value="root"/>
    <property name="Password" value="root"/>
    <property name="URL" value="jdbc:mysql://localhost:3306/fuzzy"/>
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
  </jdbc-connection-pool>
  <jdbc-resource enabled="true" jndi-name="FuzzyStudioDS" object-type="user"
pool-name="mysql_fuzzy_rootPool_1"/>
</resources>

```

Fonte: Acervo do Autor (2013)

### 3.7.1.2 EJB

A arquitetura EJB (Enterprise Java Beans) é formada por componentes que possibilitam a construção de aplicativos escalonáveis, transacionais e seguros. Um software desenvolvido sob a arquitetura EJB pode ser executado em qualquer servidor com suporte à tecnologia (BURKE; MONSON-HAEFEL, 2007). Os autores definem Enterprise Java Beans como “um modelo de componente padrão do lado do servidor para aplicativos de negócio distribuídos”.

De maneira geral, pode-se afirmar que um EJB é um componente gerenciado pelo servidor cuja principal característica é encapsular partes do sistema (a exemplo a lógica de negócio). Ao utilizar EJBs, o desenvolvimento é facilitado, uma vez que se delega para o servidor de aplicação o gerenciamento de determinados recursos (BAUER; KING, 2007).

A utilização de componentes EJB foi adotada para delegar ao servidor tarefas de conexão com o banco de dados, como a criação de unidades de persistência. Neste sentido, foi utilizado o padrão de projetos *Façade* para desenvolvimento das classes de operações com o banco de dados. Na declaração das classes *Façade* é definida a utilização da instância como um EJB através da anotação *@Stateless*. Dessa forma, a classe passa a comportar-se como uma provedora de operações, neste caso operações de acesso e manipulação de registros do banco de dados. Um exemplo de classe *Façade* é apresentado no Quadro 29, onde pode ser observado um trecho de código da classe *RegraFacade*.



Quadro 29 - Trecho de código da classe RegraFacade

```
@Stateless
public class RegraFacade {

    @PersistenceContext(type=PersistenceContextType.TRANSACTION)
    private EntityManager em;

    public int salvar(Regra br) {
        try{
            em.flush();
            em.merge(br);
            return 1;
        } catch (Exception e){
            [...]
        }
        return 3;
    }
    [...]
}
```

Fonte: Acervo do Autor (2013)

Nas classes *Facade* de operações com o banco de dados, além da utilização dos conceitos básicos de EJB, também são adotadas práticas de injeção de dependência. Com isso, não é necessária a instanciação dessas classes. O programador apenas declara um objeto da classe *Facade* com a anotação *@EJB* e, quando utilizar algum dos seus métodos, o servidor se encarrega de instanciar e injetar um objeto para utilização. A injeção de dependência também é adotada internamente na classe *Facade*, delegando ao servidor a tarefa de criação e injeção dos objetos *EntityManager*. No trecho de código do Quadro 29 se observa a declaração do objeto *EntityManager* e sua utilização posterior. A instanciação e disponibilização de um objeto criado ocorrem de forma transparente ao programador. O Quadro 30 apresenta um trecho de código da classe *RegraBean*, o qual mostra a declaração de um atributo *Facade* e sua utilização no momento de salvar um registro na base de dados.

Quadro 30 - Trecho de código da classe TermoLinguisticoEntradaBean

```

@ManagedBean
@SessionScoped
public class RegraBean {

    private Regra base = new Regra();

    @EJB
    private RegraFacade baseregrasfacade;

    public String salvar(){

        [...]
        base.setDescricaoAntecedente(descricaoAntecedente);
        base.setDescricaoConsequente(descricaoConsequente);
        base.setConsequentes(consequenteDaRegra);
        base.setAntecedentes(antecedenteDaRegra);
        int retorno = baseregrasfacade.salvar(base);
        [...]
    }

    [...]
}

```

Fonte: Acervo do Autor (2013)

### 3.7.1.3 PrimeFaces

No desenvolvimento do FuzzyStudio foi utilizada a tecnologia Java Server Faces (JSF) na sua versão 2.0. O JSF é um *framework* de aplicações Web que segue os padrões MVC (JACOBI; FALLOWS, 2006). Uma de suas principais características é a facilidade promovida na criação de componentes visuais. Além de conter componentes nativos, o JSF os vincula às classes Java correspondentes, gerenciando a comunicação entre as camadas de visão e controle do sistema. Para a construção das interfaces gráficas foi adotado o *framework* PrimeFaces de componentes ricos.

Conforme trata seu Guia do Usuário (PrimeFaces, 2012), o “PrimeFaces é uma suíte *open source* de componentes JSF com várias extensões”. Trata-se de uma biblioteca de componentes ricos, que fornece um conjunto de elementos desenvolvidos para uso na construção de aplicações visualmente ricas e interativas. O PrimeFaces disponibiliza componentes como caixas de diálogo, gráficos, interação AJAX com o servidor, tabelas e uma variedade de soluções (PRIMEFACES, 2012).

Devrates (2012) cita a popularidade da biblioteca PrimeFaces entre desenvolvedores de sistemas baseados na Web. Segundo o autor, o PrimeFaces é o *framework* mais optado por desenvolvedores para a criação de interfaces Java. PrimeFaces (2012) ressalta que sua

superioridade é resultado de algumas características presentes na sua implementação como simplicidade, alto desempenho, vasto e rico conjunto de componentes, boa documentação, facilidade de uso e suporte à interfaces para dispositivos móveis.

Para Luckon e Melo (2010) a biblioteca PrimeFaces é uma das mais completas bibliotecas de componentes do mercado e uma das primeiras a adotar os padrões do JSF 2.0. Carmisini e Vahldick (2012) realizaram alguns experimentos e testes sobre três soluções de mercado para a construção de interfaces Web. Os autores analisaram as características dos *frameworks* PrimeFaces, RichFaces e Tobago. Neste estudo, o PrimeFaces se mostrou superior em relação aos demais por apresentar maior coleção de componentes e funcionalidades e melhor documentação.

Neste trabalho, o PrimeFaces foi utilizado, em conjunto com as *tags* padrão do JSF para o desenvolvimento das interfaces com o usuário, ou seja, da camada de visualização do sistema. Vários componentes foram utilizados para garantir as funcionalidades da aplicação. O PrimeFaces foi fundamental para prover agilidade na criação de componentes ricos e qualidade nas funcionalidades desenvolvidas. O Quadro 31 apresenta um exemplo de uso das *tags* do PrimeFaces no desenvolvimento de uma interface gráfica.

Quadro 31 - Trecho de código da tela que edita variáveis de entrada

```
[...]
<p:panelGrid columns="1">
  <h:form>
    <p:panelGrid columns="2" styleClass="semBorda">

      <h:outputLabel for="nome" value="Nome:"/>
      <p:inputText value="#{variavelEntradaBean.variavel.nome}"
        id="nome" required="true" label="nome"
        requiredMessage="Nome obrigatório"/>

      <h:outputLabel for="minimo" value="De:"/>
      <p:inputText value="#{variavelEntradaBean.variavel.minimo}"
        id="minimo" required="true" label="minimo"
        requiredMessage="Valor mínimo obrigatório"/>

      <h:outputLabel for="maximo" value="Até:"/>
      <p:inputText value="#{variavelEntradaBean.variavel.maximo}"
        id="maximo" required="true" label="maximo"
        requiredMessage="Valor máximo obrigatório"/>

      <h:outputLabel for="unidade" value="Unidade medida:"/>
      <p:inputText value="#{variavelEntradaBean.variavel.unidade}"
        id="unidade" required="false" label="unidade"/>

      <f:facet name="footer">
      <center>

        <p:commandButton id="btSalvar" value="Salvar"
          update=":growl" ajax="false"
          action="#{variavelEntradaBean.salvar()}">

          <f:setPropertyActionListener value="#{aplicacaoBean.projeto}"
            target="#{variavelEntradaBean.variavel.projeto}"/>
        </p:commandButton>

        <p:commandButton id="btCancelar" value="Fechar"
          action="#{variavelEntradaBean.cancelar()}"
          immediate="true" ajax="false"/>

      </center>
      </f:facet>
    </p:panelGrid>
  </h:form>
</p:panelGrid>
[...]
```

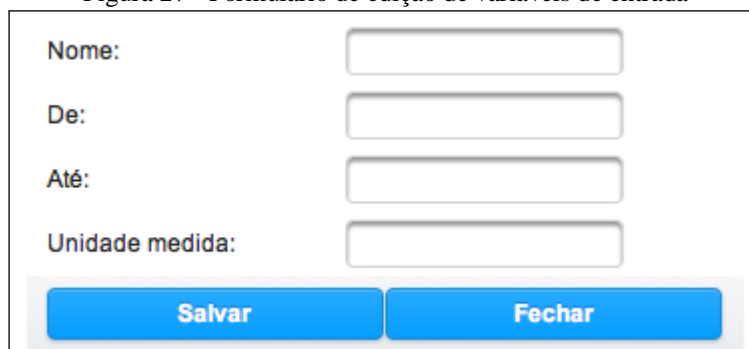
Fonte: Acervo do Autor (2013)

No trecho de código do Quadro 31 consta a criação de um formulário para edição dos dados básicos de uma variável de entrada. A tag `<p:panelGrid>` é responsável pela criação de uma estrutura de tabela, utilizada para o alinhamento dos componentes na interface. A tag `<h:outputText>` trata de um componente JSF de texto para apresentação de informações e rótulos em tela. As tags `<p:inputText>` representam entradas de dados na forma de texto, ou seja, campos para que o usuário informe dados ao sistema. O componente

`<p:commandButton>` cria um botão na tela, sua propriedade `action` define qual método de uma classe controladora será executado quando clicado. A tag `<f:setPropertyActionListener>` é responsável, neste caso, por carregar um objeto de uma classe controladora no momento em que o botão for pressionado.

Como se percebe, a comunicação entre as camadas de visualização e controle do sistema também são mapeadas pelas tags JSF e Primefaces. A propriedade `value="#{variavelEntradaBean.variavel.nome}"` vincula o campo de entrada de texto com o atributo `nome`, presente no objeto `variavel` da classe `VariavelEntradaBean`. Durante a fase de desenvolvimento da ferramenta diversas funcionalidades e vários componentes da biblioteca PrimeFaces foram utilizadas para a criação de interfaces ricas e interativas. Dentre os mais importantes estão componentes de tabelas, formulários, gráficos e funcionalidades AJAX. Na Figura 27 apresenta-se a tela de edição de variáveis de entrada, resultado do código observado no Quadro 31.

Figura 27 - Formulário de edição de variáveis de entrada



O formulário contém quatro campos de entrada de texto rotulados 'Nome:', 'De:', 'Até:' e 'Unidade medida:'. Abaixo dos campos há dois botões azuis: 'Salvar' e 'Fechar'.

Fonte: Acervo do Autor (2013)

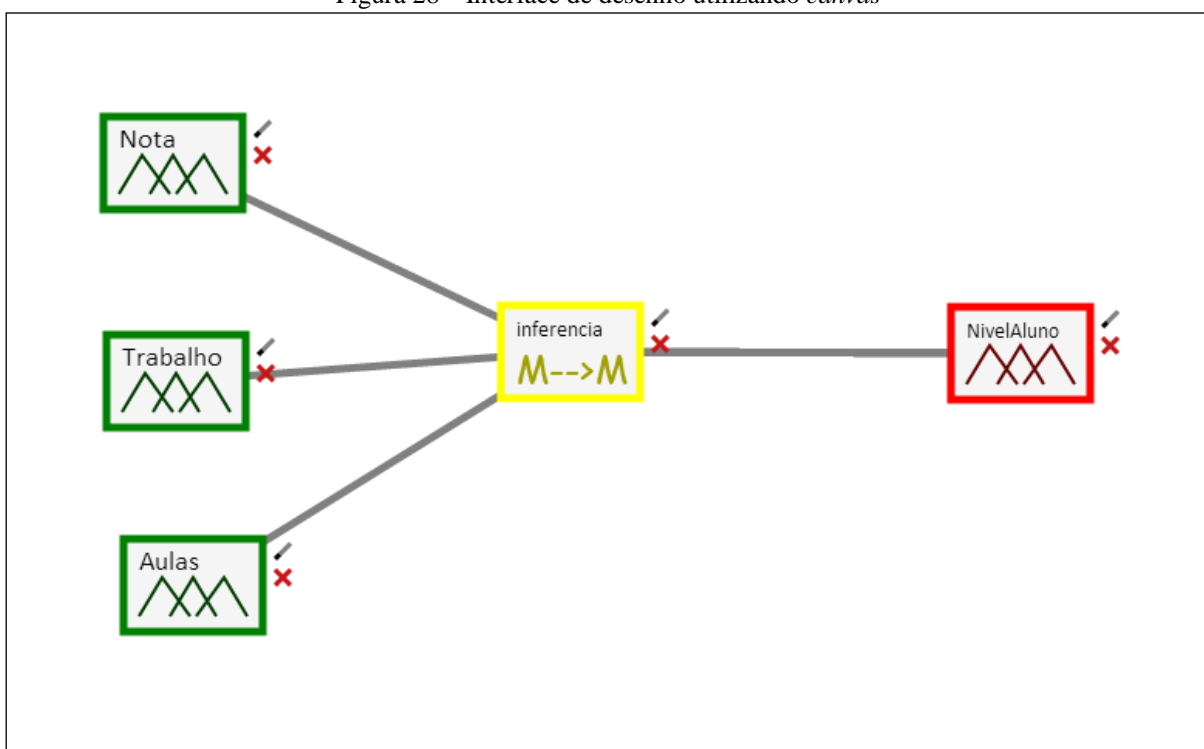
#### 3.7.1.4 HTML5

O HTML5 (*HyperText Markup Language*, ou Linguagem de Marcação de Hipertexto) é a nova versão da linguagem de marcação fundamental de uma página Web. Conforme Lawson e Sharp (2011), o HTML5 foi desenvolvido para a construção de aplicações interativas e a presença do JavaScript está cada vez mais marcante. Grannell et al. (2012) afirmam que a sintaxe do HTML5 foi desenhada para promover maior simplicidade, flexibilidade, ser mais amigável ao programador e compatível com a versão 4 do HTML e com o XHTML.

Várias novas funcionalidades são encontradas no HTML5. Dentre elas animações, gráficos avançados, facilidades na vinculação de áudio e vídeo e funcionalidades *off-line*. O HTML5 também corrige alguns problemas sofridos por desenvolvedores: a necessidade de adoção de tecnologias proprietárias para o desenvolvimento de componentes, como Flash por exemplo (GRANNELL et al., 2012). Em meio às novidades apresentadas por sua nova versão, este trabalho fez uso e se concentra na abordagem de uma delas, o *canvas*.

De acordo com Lawson e Sharp (2011), o *canvas* provê uma API para desenho 2D. Trata-se de um componente do HTML5, através do qual é possível o desenho de linhas, preenchimentos, imagens ou texto de forma fácil e poderosa com *canvas*. Para desenhar sobre o elemento é utilizada linguagem JavaScript, através de métodos simples de desenho 2D. Nesta tarefa é capturado o elemento *canvas* da tela e seu contexto, sendo utilizadas as coordenadas  $x$  e  $y$  para a construção das formas através do JavaScript. A API Canvas ainda permite o desenho de curvas, adição de efeitos, trabalho com imagens e gravação em arquivo (LUBBERS et al., 2010).

Neste trabalho o *canvas* foi utilizado para o desenho dos componentes em tela e apresentação do modelo ao usuário de forma visual, conforme Figura 28. O usuário é capaz de visualizar a estrutura do seu sistema fuzzy e as relações entre seus componentes. O Quadro 32 apresenta a declaração do componente *canvas* na interface através da utilização de sua *tag* específica.

Figura 28 – Interface de desenho utilizando *canvas*

Fonte: Acervo do Autor (2013)

Quadro 32 - Declaração do componente *canvas* na tela

```
<canvas id="canvasmesa" width="800" height="500"></canvas>
```

Fonte: Acervo do Autor (2013)

Ao declarar o componente *canvas* se define uma altura e uma largura. Esses parâmetros serão utilizados a cada atualização do desenho, pois o conteúdo anterior precisa ser apagado. Para realizar a exclusão se renova as coordenadas de tamanho do *canvas*, sobrescrevendo o conteúdo antigo. Na aplicação desenvolvida, o componente *canvas* é constantemente redesenhado para replicar as atualizações realizadas no modelo.

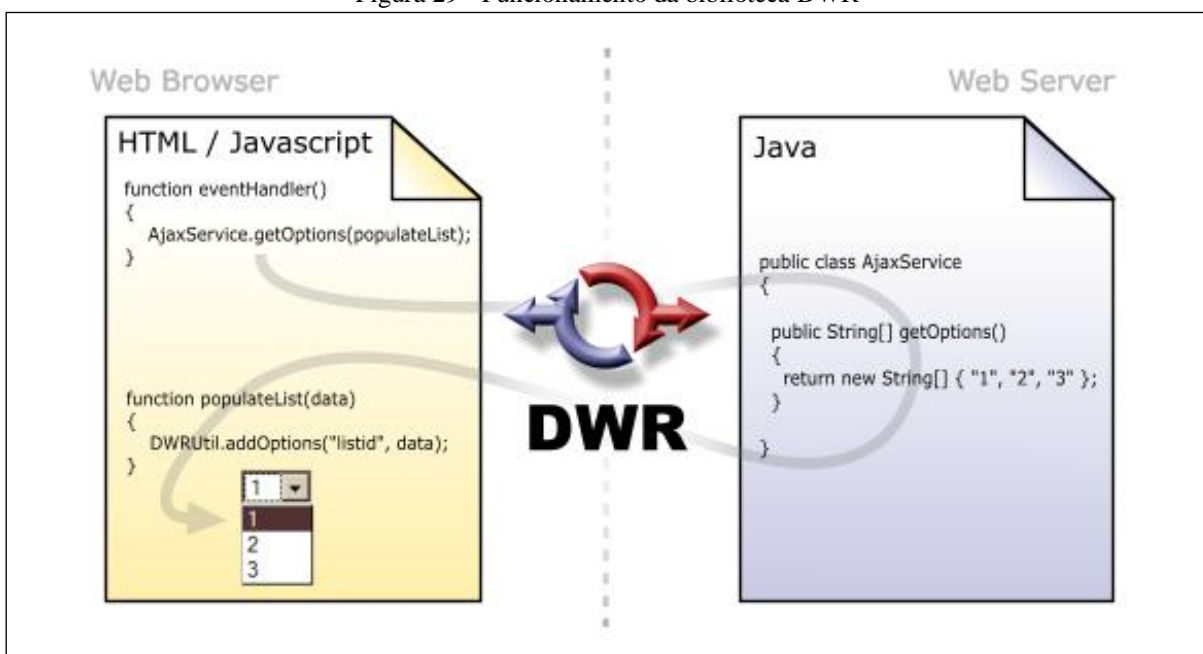
### 3.7.1.5 Direct Web Remoting

O Direct Web Remoting (DWR) é uma biblioteca *open source* desenvolvida em Java que facilita a utilização de rotinas AJAX em sistemas Web. Conforme abordado em Direct (2013), a biblioteca torna fácil a chamada de funções Java a partir do JavaScript e a chamada de funções JavaScript desde uma classe Java. O DWR encontra-se altamente difundido no

cenário atual do desenvolvimento de aplicações Web que sejam interativas e dinâmicas, contando com uma vasta base de usuários e projetos (DIRECT, 2013).

Através da biblioteca se pode fazer uso de uma série de funcionalidades como chamada em lotes, tratamento de exceções, proteção CSRF (*Cross Site Request Forgery*) e integração com tecnologias do lado do servidor como o Spring (DIRECT, 2013). Para garantir seu funcionamento, o DWR possui uma estrutura em duas partes, conforme explicado em Direct (2013). A primeira delas consiste em um Servlet Java no servidor, que processa as requisições e envia as respostas ao navegador (cliente). A segunda parte é composta por códigos JavaScript que enviam as requisições ao servidor e atualizam a página de forma dinâmica. As interações entre classes Java e arquivos JavaScript proporcionadas pela biblioteca DWR estão resumidas na Figura 29.

Figura 29 - Funcionamento da biblioteca DWR



Fonte: Direct (2013)

Neste sentido, o DWR gera código JavaScript a partir das classes Java mapeadas. Com isso, é possível realizar chamadas a métodos de classes Java através de rotinas JavaScript, como se as mesmas estivessem sendo chamadas e executadas no próprio navegador (DIRECT, 2013). Para utilização do DWR três passos são necessários: criar um arquivo XML para mapeamento das classes Java (*dwr.xml*); declarar o Servlet no arquivo *web.xml* e incluir os arquivos JavaScript do DWR e das classes mapeadas nas páginas desejadas. O Quadro 33 apresenta a estrutura do arquivo *dwr.xml*.



Quadro 33 - Estrutura do arquivo *dwr.xml*

```

<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 3.0//EN"
    "http://getahead.org/dwr/dwr30.dtd">

<dwr>
  <allow>
    <create creator="new" javascript="ClasseInterface">
      <param name="class" value="br.udesc.sistema.InterfaceBean"/>
    </create>

    <create creator="new" javascript="ControlaConectados">
      <param name="class" value="br.udesc.sistema.ControlaConectados"/>
    </create>
  </allow>
</dwr>

```

Fonte: Acervo do Autor (2013)

No arquivo apresentado no Quadro 33 pode-se observar o mapeamento de duas classes Java para manipulação a partir do JavaScript. A tag `<create>` define a criação de uma variável JavaScript para operações com a classe Java, a qual é mapeada pela tag `<param>`. Dessa forma, através das variáveis JavaScript *ClasseInterface* e *ControlaConectados*, o programador poderá realizar chamadas aos métodos das classes *InterfaceBean* e *ControlaConectados*, respectivamente. O Servlet necessita ser declarado no arquivo *web.xml*, apresentado no Quadro 34.

Quadro 34 - Trecho de código do arquivo *web.xml*

```

[...]
<servlet>
  <display-name>DWR Servlet</display-name>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>>true</param-value>
  </init-param>
</servlet>
[...]

```

Fonte: Acervo do Autor (2013)

Uma vez declarado o Servlet e mapeadas as classes Java que conterão acesso a partir do JavaScript, os arquivos são importados nas páginas. O Quadro 35 apresenta o código de importação destes arquivos.

Quadro 35 - Importação dos arquivos JavaScript referentes ao DWR

```
[...]
<script type="text/javascript" src="../../dwr/engine.js"></script>
<script type="text/javascript" src="../../dwr/util.js"></script>
<script type="text/javascript" src="../../dwr/interface/ClasseInterface.js">
</script>
<script type="text/javascript" src="../../dwr/interface/ControlaConectados.js">
</script>
[...]
```

Fonte: Acervo do Autor (2013)

Um fato importante a ser observado no código apresentado no Quadro 35 é que os arquivos JavaScript declarados nas *tags* de importação não estão disponíveis ao programador. Os mesmos são gerados pela biblioteca DWR em tempo de execução, com base nas configurações do seu arquivo XML.

A biblioteca DWR foi utilizada no desenvolvimento da ferramenta de modelagem fuzzy para suprir três necessidades fundamentais. A primeira delas é o desenho do modelo através da API Canvas. No momento de realizar o desenho, uma função JavaScript faz uma requisição ao servidor, que retorna os dados atualizados de componentes e suas respectivas coordenadas de posicionamento. A partir de então o desenho pode ser realizado na tela. O Quadro 36 apresenta o código JavaScript que realiza a chamada à classe Java por meio da variável *ClasseInterface*, utilizando o retorno desta chamada para a função de redesenho *atualiza\_interface2(data)*.

Quadro 36 - Funções JavaScript de controle de usuários e redesenho da interface

```
[...]
function recuperaDados(){
    [...]
    ControlaConectados.request(codUsuario, codProjeto, function(data){});

    var temcanvas = document.getElementById('canvasmesa');
    if(temcanvas != null){
        codProjeto = document.getElementById('tabView:codigoProjeto').value;
        ClasseInterface.retornaComponentes(codProjeto, function(data){
            atualiza_interface2(data);
        });
    }
    [...]
}
[...]
```

Fonte: Acervo do Autor (2013)

A segunda funcionalidade atendida pelo DWR é a atualização da interface de desenho em tempo real durante o trabalho colaborativo. Quando um usuário modifica algo na interface de desenho essa alteração é replicada aos demais usuários conectados ao projeto. Para isso, uma função JavaScript realiza requisições de tempos em tempos em busca de modificações nos componentes, redesenhando e atualizando a interface.

A biblioteca ainda foi utilizada para suprir uma terceira necessidade, de controle dos usuários conectados ao projeto. A cada requisição em busca de modificações nos componentes, o servidor armazena a requisição e atualiza a lista de usuários conectados ao projeto. No caso de algum usuário estar inativo por um período determinado, o mesmo é considerado desconectado do projeto. No código apresentado no Quadro 34, a chamada ao método *request()* através da variável mapeada *ControlaConectados* é responsável pelo controle de usuários autenticados e conectados ao projeto.

### 3.7.2 Execução do motor

O FuzzyStudio pode ser dividido em três etapas de funcionamento, conforme Figura 13. Estas etapas consistem na modelagem do sistema fuzzy, execução do modelo construído e geração de artefatos como código FCL, Java e gráficos.

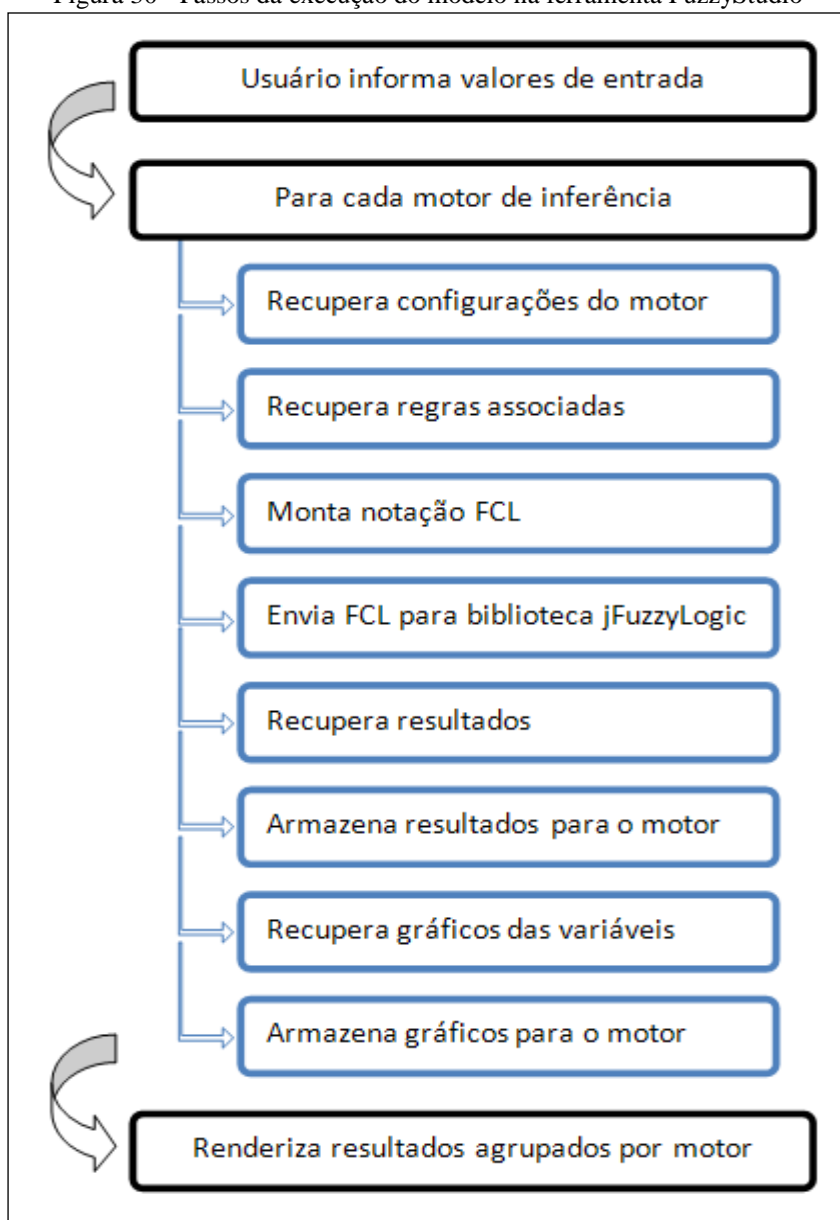
A execução do motor constitui o núcleo do sistema e suas funcionalidades centrais. A ferramenta permite o cadastro de regras e a criação de vários motores de inferência para um mesmo sistema. O detalhe importante deste modelo adotado é o fato do usuário poder vincular regras a motores. Ou seja, um motor não faz uso de todas as regras da base, senão apenas daquelas definidas pelo usuário. Com isso, é possível criar uma série de motores e vincular aos mesmos apenas as regras desejadas, facilitando a realização de testes e a comparação do desempenho e resultados apresentados por motores distintos.

Além da modelagem independente entre motores e regras, as definições e configurações do sistema fuzzy como o método de agregação de regras, método de defuzzificação, entre outros estão definidos no motor de inferência. Para cada motor se tem uma configuração distinta para o sistema difuso, utilizando regras também independentes. Com isso é possível, de maneira indireta, a criação de vários sistemas fuzzy em uma mesma estrutura, potencializando a tarefa de simular e executar distintos arranjos de sistema.

A execução do sistema difuso é realizada de forma independente entre os motores definidos na modelagem. A ferramenta processa as informações e devolve os valores de resultado separado por motor, para cada variável de saída vinculada ao motor correspondente.

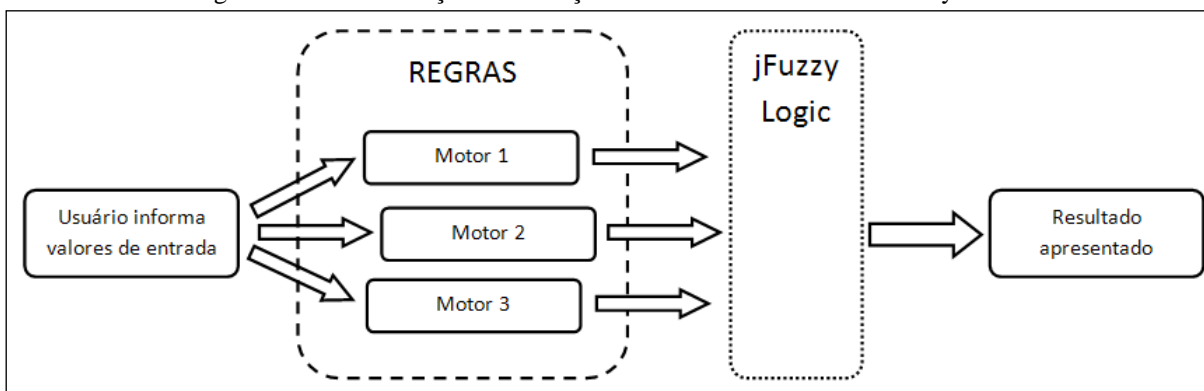
É executado um processo iterativo para cada motor. Primeiro se captura os valores de entrada definidos pelo usuário, as definições e configurações do motor. Também são recuperadas as regras vinculadas ao motor e, através delas, as variáveis que compõem o sistema. Com esses dados é construída a notação fuzzy no padrão FCL, a qual é passada às rotinas da biblioteca *jFuzzyLogic*, encarregada de processar tais dados e retornar as saídas do sistema. Uma vez recuperadas as saídas é feita a apresentação ao usuário. Como se percebe, a aplicação coleta os dados da modelagem e os valores informados pelo usuário, estrutura essas informações através do padrão FCL e utiliza a biblioteca *jFuzzyLogic* para as tarefas de cálculo das saídas. As Figuras 30 e 31 ilustram o processo de execução do sistema difuso na ferramenta FuzzyStudio.

Figura 30 - Passos da execução do modelo na ferramenta FuzzyStudio



Fonte: Acervo do Autor (2013)

Figura 31 - Demonstração da execução do modelo na ferramenta FuzzyStudio



Fonte: Acervo do Autor (2013)

### 3.7.2.1 Biblioteca *jFuzzyLogic*

O *jFuzzyLogic*, de acordo com Cingolani e Alcalá-Fdez (2012), é uma biblioteca *open source* escrita em Java que permite o desenho de controladores baseados em lógica fuzzy, suportando a especificação trazida pela IEC em sua norma 1131-7, que define a notação padrão de controladores fuzzy. O objetivo principal da biblioteca é trazer os benefícios de uma plataforma *open source* com a padronização dos sistemas fuzzy na comunidade de desenvolvedores. Além desses aspectos, a biblioteca preocupa-se com a extensibilidade da API, permitindo a criação de novas funcionalidades ou aplicações baseadas na sua arquitetura. Uma terceira questão preponderante na escolha da biblioteca utilizada é a independência de plataforma. Por ser desenvolvida em Java, pode ser utilizada em aplicações executando em qualquer estrutura de hardware e software com suporte à linguagem (CINGOLANI E ALCALÁ-FDEZ, 2012).

Conforme apresentado em *JFuzzyLogic* (2013), a biblioteca implementa, seguindo as especificações IEC 1131-7, os mecanismos de inferência fuzzy de acordo com o modelo Mamdani de sistemas difusos. Dentre as funcionalidades consideradas para a escolha da biblioteca no presente trabalho se destacam a variada coleção de funções de pertinência implementadas, os métodos de agregação e ativação das regras, variedade dos mecanismos de defuzzificação e geração de gráficos.

A biblioteca é disponibilizada em um arquivo único (extensão *jar*) que é importado na aplicação, permitindo o uso e a chamada de seus métodos. Na aplicação, o sistema traduz os componentes contidos no banco de dados ao padrão de notação utilizado pela biblioteca. Em posse desse registro é criado um objeto *FIS* (*Fuzzy Inference System*, ou Sistema de Inferência Fuzzy), que contém todos os dados do sistema difuso. A partir deste objeto se cria um bloco de funções, representado pela classe *FunctionBlock*. Utilizando o objeto desta classe é possível informar os valores de entrada ao sistema e coletar seus resultados. A utilização da biblioteca na ferramenta e os aspectos supracitados podem ser observados no código apresentado no Quadro 37.

Quadro 37 - Trechos de código do método de execução do sistema pelo motor de inferência

```

1 public String executaFCL(MotorInferencia motor) throws RecognitionException{
2
3     try{

```

```

4     FIS fis = FIS.createFromString(fcl, true);
5     FunctionBlock fb = fis.getFunctionBlock(motor.getNome());
6     FunctionBlockAdaptada functionBlock = new FunctionBlockAdaptada(fis);
7     functionBlock.setName(fb.getName());
8     functionBlock.setRuleBlocks(fb.getRuleBlocks());
9     functionBlock.setVariables(fb.getVariables());
10    List<Regra> basesDoMotor =
11        regrafacade.selecionaPeloMotor(motor.getCodigo());
12
13    for(Regra b : basesDoMotor){
14        for(Antecedente ant :
15            regrafacade.selecionaAntecedentes(b.getCodigo())){
16            for(VariavelEntrada entrada : valoresEntrada){
17                if(entrada.getCodigo() == ant.getVariavelEntrada().getCodigo())
18                    functionBlock.setVariable(ant.getVariavelEntrada().getNome(),
19                        entrada.getValor());
19            }
20        }
21    }
22
23    functionBlock.evaluate();
24
25    for(Regra b : basesDoMotor){
26        for(Consequente con :
27            regrafacade.selecionaConsequentes(b.getCodigo())){
28
29            for(int i = 0; i < variaveisSaida.size(); i++)
30                if(variaveisSaida.get(i).getCodigo() ==
31                    con.getVariavelSaida().getCodigo()){
32
33                    List<Double> value = new ArrayList<Double>();
34                    value.add(Double.parseDouble(
35                        String.valueOf(motor.getCodigo())));
36                    value.add(functionBlock.getVariable(
37                        variaveisSaida.get(i).getNome()).getValue());
38                    variaveisSaida.get(i).setValor(value);
39                    VariavelSaida vsaida = new VariavelSaida();
40                    vsaida = (VariavelSaida)variaveisSaida.get(i).clone();
41                    this.valoresSaida.add(vsaida);
42                }
43            }
44        }
45
46    Collection<Variable> var = functionBlock.variables();
47    Iterator<Variable> iterador = var.iterator();
48
49    while(iterador.hasNext()){
50        Variable v = iterador.next();

```

```

45
46     JFreeChart chart = v.chart(false);
47     Grafico grafico = new Grafico();
48     grafico.setGrafico(chart);
49     grafico.setMotor(motor);
50
51     if(v.isOutputVariable())
52         this.graficosSaida.add(grafico);
53     else
54         this.graficosEntrada.add(grafico);
55     }
56
57 } catch(Exception e){
58     FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_WARN, "Erro
59     na Execução", e.getMessage());
60     FacesContext.getCurrentInstance().addMessage(null, msg);
61 }
62 return "";
63 }

```

Fonte: Acervo do Autor (2013)

Como se pode observar na linha 1, o método recebe como parâmetro o motor a ser executado. Na linha 4 é criado o objeto *FIS* conforme a variável *fcl* já criada com os registros recuperados do banco de dados. A linha 6 apresenta a criação do bloco de função. Este objeto é do tipo *FunctionBlockAdaptada*, pois a classe original (*FunctionBlock*) foi alterada de modo a retornar os gráficos em objetos *JFreeChart*. Nas linhas 7 a 9 pode-se observar a atribuição das propriedades do bloco de função, bem como suas variáveis. É atribuído ao bloco de função os valores para cada variável de entrada (linhas 12 a 19). A partir do bloco de função formado e conhecendo os valores de entrada, o sistema fuzzy é executado a partir do comando *functionBlock.evaluate()*, contido na linha 21.

Ao realizar a execução do motor já se torna possível a coleta dos resultados. Esta tarefa é apresentada nas linhas 23 a 38, onde é percorrido o conjunto de regras e suas variáveis de saída, coletando os valores inferidos para as mesmas. Para o armazenamento dos gráficos da execução é criada uma coleção de variáveis (linha 40) e um iterador para percorrê-la (linha 41). Para cada variável do sistema é solicitado seu referido gráfico (linha 46), armazenando-o em uma estrutura definida pela classe *Grafico* (linhas 47 e 48).



### 3.7.2.2 Geração de artefatos

Outro diferencial buscado pela ferramenta FuzzyStudio é a geração de artefatos do sistema modelado. A ferramenta se baseia na geração de três tipos de resultados: gráficos representativos do sistema difuso, notação no padrão FCL para o sistema e código de programação Java para desenvolvimento de aplicativos utilizando o sistema fuzzy modelado.

Para a geração de gráficos foram utilizadas funcionalidades presentes na biblioteca *jFuzzyLogic*. São gerados gráficos para cada variável de entrada e saída que compõe o sistema. Para as variáveis de entrada são apresentados os termos linguísticos utilizados para a fuzzificação, bem como o valor informado pelo usuário. Para as variáveis de saída são apresentados os gráficos de defuzzificação, onde constam os termos linguísticos das variáveis e é demarcada a área inferida no processo, bem como seu resultado originalmente. A biblioteca *jFuzzyLogic* retorna os gráficos solicitados por meio de janelas *JFrame*. Para a utilização em ambiente Web essa solução não pode ser utilizada. Foi necessária a adaptação da classe geradora dos gráficos de modo a retorná-los através de objetos da classe *JFreeChart*. Com isso, esses objetos são convertidos em imagens e salvos em arquivos, os quais são utilizados pela aplicação para renderização em tela. O Quadro 38 apresenta o código responsável pela renderização dos gráficos.

Quadro 38 - Manipulação dos gráficos das variáveis do sistema

```

1  public String chamaGráficos(MotorInferencia m) throws IOException{
2      this.chartsEntrada = new ArrayList<StreamedContent>();
3      this.chartsSaida = new ArrayList<StreamedContent>();
4
5      int contador = 1;
6      for(Grafico graf : this.graficosEntrada){
7          if(graf.getMotor().getCodigo() == m.getCodigo()){
8              File chartFile = new File("grafico"+contador);
9              ChartUtilities.saveChartAsPNG(chartFile,
10                 graf.getGrafico(), 375, 300);
11                 StreamedContent chart = new DefaultStreamedContent(
12                     new FileInputStream(chartFile), "image/png");
13                 chartsEntrada.add(chart);
14                 contador++;
15             }
16         }
17
18         for(Grafico graf : this.graficosSaida){
19             if(graf.getMotor().getCodigo() == m.getCodigo()){
20                 File chartFile = new File("grafico"+contador);
21                 ChartUtilities.saveChartAsPNG(chartFile,
22                     graf.getGrafico(), 375, 300);
23                 StreamedContent chart = new DefaultStreamedContent(
24                     new FileInputStream(chartFile), "image/png");
25                 chartsSaida.add(chart);
26                 contador++;
27             }
28         }
29
30         return "graficos.xhtml";
31     }

```

Fonte: Acervo do Autor (2013)

Como se pode observar, os gráficos são armazenados em duas listas, uma para os gráficos de entrada e outra para os gráficos de saída (linhas 6 e 16). Estas listas são percorridas de modo que cada gráfico é convertido para um arquivo (linhas 8 e 18). Através do método *saveChartAsPNG* da classe *ChartUtilities* os arquivos são convertidos em imagens no formato PNG (linhas 9 e 19). Estas imagens são encapsuladas em objetos da classe *StreamedContent* e adicionadas às listas que serão utilizadas na renderização (linhas 9 a 10 e 20 a 21).

A geração de FCL já é realizada pelo sistema na fase de simulação do modelo. O sistema recupera as informações do banco de dados referentes a cada componente criado pelo

usuário e constrói a notação seguindo os padrões definidos pela norma IEC 1131-7. O momento da geração consiste em recuperar essas informações previamente construídas e apresentá-las ao usuário. Este processo é ilustrado pelo Quadro 39, que contém o código referente à funcionalidade.

Quadro 39 – Método de montagem da notação FCL

```

1 public String montaFCL(MotorInferencia motor){
2     String result = "";
3     result += "FUNCTION_BLOCK "+motor.getNome()+" \n";
4     result += "\n";
5
6     //===== VARIAVEL DE ENTRADA
7     result += "VAR_INPUT \n";
8     for(VariavelEntrada ve : retornaEntradasDoMotor(motor.getCodigo())){
9         result += "    "+ve.getNome()+" : REAL;\n";
10    }
11    result += "END_VAR\n";
12    result += "\n";
13
14    //===== VARIAVEL DE SAIDA
15    result += "VAR_OUTPUT \n";
16    for(VariavelSaida vs : retornaSaidasDoMotor(motor.getCodigo())){
17        result += "    "+vs.getNome()+" : REAL;\n";
18    }
19    result += "END_VAR\n";
20
21    //===== FUZZIFICACAO
22    for(VariavelEntrada ve2 : retornaEntradasDoMotor(motor.getCodigo())){
23        List<TermoLinguisticoEntrada> termosEntrada =
24            termoentradafacade.selecionaTudoPelaVariavel(ve2.getCodigo());
25        if(termosEntrada.size() > 0)
26            result += "FUZZIFY "+ve2.getNome()+" \n";
27
28        for(TermoLinguisticoEntrada te : termosEntrada){
29            [...]
30        }
31        result += "END_FUZZIFY\n";
32    }
33
34    result += "\n";
35
36    //===== DEFUZZIFICACAO
37    for(VariavelSaida ve3 : retornaSaidasDoMotor(motor.getCodigo())){

```

```

38     List<TermoLinguisticoSaida> termosSaida =
39         termosaidafacade.selecionaTudoPelaVariavel(ve3.getCodigo());
40     if(termoSaida.size() > 0)
41         result += "DEFUZZIFY "+ve3.getNome()+" \n";
42     for(TermoLinguisticoSaida te : termosSaida){
43         [...]
44     }
45     result += "  METHOD: "+motor.getMetodo_defuzzificacao()+"; \n";
46     result += "  DEFAULT := 0; \n";
47     result += "END_DEFUZZIFY\n";
48 }
49 result += "\n";
50
51 //===== BLOCO DE REGRAS
52 result += "RULEBLOCK blocoderegra\n";
53 result += "\n";
54
55 result += "  ACCU : "+motor.getMetodo_agregacao_regras()+"; \n";
56 result += "  AND : "+motor.getConexao_and()+"; \n";
57 result += "  ACT : "+motor.getMetodo_ativacao_regras()+"; \n";
58
59 int counter = 1;
60 //regrafacade.selecionaPeloMotor(motor.getCodigo())
61 List<Regra> brs =
62     regrafacade.selecionaPeloMotor(motor.getCodigo());
63 for(Regra br : brs){
64     result += "  RULE "+counter+" : "+montaRegra(br)+";\n";
65     counter++;
66 }
67
68 result += "END_RULEBLOCK\n";
69 result += "\n";
70 //=====
71
72 result += "END_FUNCTION_BLOCK\n";
73 return result;
74 }

```

Fonte: Acervo do Autor (2013)

Conforme apresentado no Quadro 39, a linha 3 apresenta o início do bloco de função, o qual recebe o nome idêntico ao nome do motor. Nas linhas 7 a 12 são mapeadas as variáveis de entrada, enquanto as variáveis de saída sofrem o mesmo procedimento nas linhas 15 a 19. O processo de fuzzificação percorre todas as variáveis de entrada e sua lista de termos

linguísticos. A montagem do bloco de fuzzificação pode ser observado nas linhas 22 a 32. De igual modo, o bloco de defuzzificação é montado para cada variável de saída com sua lista de termos linguísticos, o que ocorre nas linhas 37 a 48. No bloco de defuzzificação é ainda configurado o método desejado para inferir o resultado (linha 45). A montagem das regras está apresentada nas linhas 52 a 68, onde o sistema configura os métodos de acumulação, conexão AND e agregação (linhas 55, 56 e 57) e percorre a lista de regras do motor, construindo sua estrutura. Finalmente o bloco de regras é fechado na linha 72.

Por fim, a ferramenta gera código de programação na linguagem Java, voltado aos usuários que desejam desenvolver um sistema ou controlador utilizando o modelo criado. A classe gerada faz uso da biblioteca *jFuzzyLogic* e da notação FCL construída. O Quadro 40 apresenta um exemplo de classe gerada através da ferramenta. O código consiste em uma classe Java com a notação do sistema fuzzy modelado. Ao executar esta classe é solicitado ao usuário os valores para cada variável de entrada, conforme as linhas 59 e 60. Esses valores, juntamente com o modelo do sistema são passados à biblioteca que realiza o processamento, retornando seus resultados (linhas 62 a 66). Este resultado é informado ao usuário através dos gráficos resultantes.

Quadro 40 - Classe Java gerada pela ferramenta

```

1  import net.sourceforge.jFuzzyLogic.FIS;
2  import net.sourceforge.jFuzzyLogic.FunctionBlock;
3  import org.antlr.runtime.RecognitionException;
4
5  public class FuzzyStudioFCL {
6      private static String fcl;
7
8      public static void main(String[] args) throws RecognitionException {
9          fcl = "FUNCTION_BLOCK motor1 \n"+
10             "\n"+
11             "VAR_INPUT \n"+
12             "    beber : REAL;\n"+
13             "    fumar : REAL;\n"+
14             "END_VAR\n"+
15             "\n"+
16             "VAR_OUTPUT \n"+
17             "    impactoSaude : REAL;\n"+
18             "END_VAR\n"+
19             "FUZZIFY beber \n"+
20             "    TERM pouco := (1.0, 1) (2.0, 1) (4.0, 0); \n"+
21             "    TERM medio := (3.0, 0) (4.0, 1) (6.0, 1) (7.0, 0); \n"+

```

```

22         "    TERM muito := (6.0, 0) (8.0, 1) (10.0, 1); \n"+
23         "END_FUZZIFY\n"+
24         "FUZZIFY fumar \n"+
25         "    TERM pouco := (1.0, 1) (3.0, 0); \n"+
26         "    TERM medio := (2.0, 0) (4.0, 1) (6.0, 0); \n"+
27         "    TERM muito := (5.0, 0) (10.0, 1); \n"+
28         "END_FUZZIFY\n"+
29         "\n"+
30         "DEFUZZIFY impactoSaude \n"+
31         "    TERM pouco := (1.0, 1) (5.0, 0); \n"+
32         "    TERM muito := (4.0, 0) (6.0, 1) (10.0, 1); \n"+
33         "    METHOD : COG; \n"+
34         "    DEFAULT := 0; \n"+
35         "END_DEFUZZIFY\n"+
36         "\n"+
37         "RULEBLOCK blocoderegra\n"+
38         "\n"+
39         "    ACCU : MAX; \n"+
40         "    AND : BDIF; \n"+
41         "    ACT : PROD; \n"+
42         "    RULE 1 : IF beber IS pouco AND fumar IS pouco
43             THEN impactoSaude IS pouco;\n"+
44         "    RULE 2 : IF beber IS pouco AND fumar IS medio
45             THEN impactoSaude IS pouco;\n"+
46         "    RULE 3 : IF beber IS pouco AND fumar IS muito
47             THEN impactoSaude IS muito;\n"+
48         "    RULE 4 : IF beber IS medio AND fumar IS medio
49             THEN impactoSaude IS muito;\n"+
50         "    RULE 5 : IF beber IS muito AND fumar IS muito
51             THEN impactoSaude IS muito;\n"+
52         "END_RULEBLOCK\n"+
53         "\n"+
54         "END_FUNCTION_BLOCK\n";
55
56     try{
57         FIS fis = FIS.createFromString(fcl, true);
58         FunctionBlock functionBlock = fis.getFunctionBlock("motor1");
59         functionBlock.setVariable("beber",
60             Double.parseDouble(javax.swing.JOptionPane.
61                 showInputDialog("Informe valor para beber")));
62         functionBlock.setVariable("fumar",
63             Double.parseDouble(javax.swing.JOptionPane.
64                 showInputDialog("Informe valor para fumar")));
65
66         functionBlock.evaluate();
67         functionBlock.getVariable("beber").chart(true);

```

```

64         functionBlock.getVariable("fumar").chart(true);
65         functionBlock.getVariable("impactoSaude").chart(true);
66         functionBlock.getVariable("impactoSaude").
67             chartDefuzzifier(true);
68     } catch(Exception e){
69     }
70 }
71
72 public String getFcl(){
73     return this.fcl;
74 }
75 }

```

Fonte: Acervo do Autor (2013)

### 3.7.3 Desenho do modelo

O elemento *canvas* do HTML5 foi utilizado para o desenho da interface gráfica do modelo. Neste processo, são capturados os dados de cada componente da base de dados e separados em *arrays*. Dessa forma, o sistema organiza um conjunto de quatro vetores que armazenam as variáveis de entrada, variáveis de saída, motores de inferência e as linhas que conectam os componentes. O Quadro 41 apresenta o processo de desenho dos componentes de um dos arrays utilizando a linguagem JavaScript e os métodos da API Canvas.

Quadro 41 - Métodos de desenho do *canvas* para as variáveis de entrada

```

1  function atualiza_canvas_entradas(){
2      for(var i = 0; i < array_entradas.length; i++){
3
4          var posx = array_entradas[i].posx;
5          var posy = array_entradas[i].posy;
6          var nome = array_entradas[i].nome;
7          var minimo = array_entradas[i].minimo;
8          var maximo = array_entradas[i].maximo;
9          var unidade = array_entradas[i].unidade;
10
11         if(posx > 690) posx = 690;
12         if(posy > 440) posy = 440;
13
14         //DESENHA RETANGULO
15         context.beginPath();

```

```
16     var e = new Elemento(posx, posy, 90, 60);
17     context.rect(e.x, e.y, e.w, e.h);
18     context.fillStyle = '#F5F5F5';
19     context.fill();
20     context.lineWidth = 5;
21     context.strokeStyle = 'green';
22     context.stroke();
23
24     //ESCREVE TEXTO
25     context.fillStyle = '#0A0A0A';
26     if(nome.length > 12)
27         nome = nome.substring(0, 9) + "...";
28     if (nome.length > 9)
29         context.font = '14px Calibri';
30     else
31         context.font = '18px Calibri';
32     context.fillText(nome, (parseInt(posx) + 10), (parseInt(posy) + 20));
33
34     //DESENHA EDICAO
35     context.beginPath();
36     context.moveTo(parseInt(posx)+108, parseInt(posy)+3);
37     context.lineTo(parseInt(posx)+101, parseInt(posy)+10);
38     context.lineWidth = 3;
39     context.strokeStyle = 'grey';
40     context.stroke();
41     context.beginPath();
42     context.moveTo(parseInt(posx)+101, parseInt(posy)+10);
43     context.lineTo(parseInt(posx)+98, parseInt(posy)+13);
44     context.lineWidth = 3;
45     context.strokeStyle = 'black';
46     context.stroke();
47
48     //DESENHA EXCLUSAO
49     context.beginPath();
50     context.moveTo(parseInt(posx)+98, parseInt(posy)+20);
51     context.lineTo(parseInt(posx)+108, parseInt(posy)+30);
52     context.moveTo(parseInt(posx)+98, parseInt(posy)+30);
53     context.lineTo(parseInt(posx)+108, parseInt(posy)+20);
54     context.fill();
55     context.lineWidth = 3;
56     context.strokeStyle = '#C81414';
57     context.stroke();
58
59
60     //DESENHA EFEITO
```



```

61     context.beginPath();
62     //Triângulo 1
63     context.moveTo(parseInt(posx)+10, parseInt(posy)+50);
64     context.lineTo(parseInt(posx)+25, parseInt(posy)+25);
65     context.lineTo(parseInt(posx)+43, parseInt(posy)+50);
66     //Triângulo 2
67     context.moveTo(parseInt(posx)+30, parseInt(posy)+50);
68     context.lineTo(parseInt(posx)+45, parseInt(posy)+25);
69     context.lineTo(parseInt(posx)+60, parseInt(posy)+50);
70     //Triângulo 3
71     context.moveTo(parseInt(posx)+47, parseInt(posy)+50);
72     context.lineTo(parseInt(posx)+65, parseInt(posy)+25);
73     context.lineTo(parseInt(posx)+80, parseInt(posy)+50);
74
75     context.lineWidth = 2;
76     context.strokeStyle = '#003C00';
77     context.stroke();
78 }
79
80 }

```

Fonte: Acervo do Autor (2013)

Através da linha 2 pode-se observar que é percorrido o *array* de componentes para desenho em tela. Caso o componente esteja localizado fora dos limites do *canvas* o mesmo é desenhado imediatamente dentro do quadro. Essa verificação pode ser observadas nas linhas 11 e 12. As linhas 15 a 22 apresentam as funções de desenho dos retângulos. O comando *context.rect(e.x, e.y, e.w, e.h)* é responsável pelo desenho da forma, o qual é preenchido com o comando *context.fill()*. Os contornos do componente são desenhados com auxílio das definições *context.lineWidth*, que define a espessura da linha e *context.strokeStyle*, definindo a cor do contorno. O comando *context.stroke()* se encarrega de efetivar a renderização.

O desenho de texto em tela (nome do componente) é apresentado nas linhas 25 a 32. Na linha 25 se define a cor do texto. As linhas 26 a 31 verificam o tamanho do nome do componente, diminuindo ou aumentando o tamanho da fonte utilizada. Nos casos de nomes muito extensos, este é truncado, conforme se observa na linha 27. A escrita do texto em tela é possibilitada pelo comando *context.fillText(texto, posição\_x, posição\_y)*, observado na linha 32. Este comando utiliza a definição de fonte informada à propriedade *context.font*.

As linhas que conectam os componentes do modelo são desenhadas abaixo dos mesmos e partem do centro de cada elemento. Com isso, cria-se o efeito de que os

componentes estejam conectados pelas suas bordas, independente de seu posicionamento. O Quadro 42 apresenta o trecho de código referente ao desenho das linhas conectoras.

Quadro 42 - Trecho de código do método de desenho de linhas no *canvas*

```

1  function atualiza_canvas_linhas(){
2      for(var i = 0; i < array_linhas.length; i++){
3          var x1 = array_linhas[i].x1;
4          var y1 = array_linhas[i].y1;
5          var x2 = array_linhas[i].x2;
6          var y2 = array_linhas[i].y2;
7          if(x1 > 690) x1 = 690;
8          if(y1 > 440) y1 = 440;
9          if(x2 > 690) x2 = 690;
10         if(y2 > 440) y2 = 440;
11
12         context.beginPath();
13         context.moveTo(x1, y1);
14         context.lineTo(x2, y2);
15         context.lineWidth = 5;
16         context.strokeStyle = 'grey';
17         context.stroke();
18     }
19 }

```

Fonte: Acervo do Autor (2013)

A linha 2 apresenta o código que percorre todas as linhas calculadas a fim de desenhá-las. As linhas 3 a 10 apresentam os mecanismos de validação para componentes localizados nas extremidades do elemento *canvas*. As conexões do modelo são desenhadas através do comando *context.moveTo()*, para mover o cursor sem nenhum tipo de desenho e *context.lineTo()*, que movimentava o cursor até uma posição definida dessa vez desenhando uma linha desde a origem até o destino (linhas 13 e 14). Mais uma vez o desenho será efetivamente realizado com o comando *context.stroke()* (linha 17) utilizando os atributos de espessura e cor definidos nas linhas 15 e 16.

Outra funcionalidade da ferramenta desenvolvida é a possibilidade de arrastar e soltar os componentes desenhados em tela, ajustando-os e posicionando-os conforme a necessidade e desejo do usuário. Para tornar isso possível, foi necessária a implementação de métodos que monitorassem os cliques e o comportamento do usuário com o *mouse* sobre o elemento *canvas*. Cada vez que o usuário mantém o botão do *mouse* pressionado uma *flag* é ativada. A partir de então a cada movimento do *mouse* com o botão pressionado todo o *canvas* é

redesenhado, reposicionando o objeto selecionado. O redesenho contínuo só termina com o soltar de botão do usuário, quando o sistema armazena no banco de dados a nova posição do elemento arrastado, desativando a *flag* responsável por este controle. Neste processo, é recalculada a posição do componente selecionado, bem como das linhas que realizam a conexão deste elemento com os demais. Os eventos de escuta do *mouse* podem ser observados no Quadro 43.

Quadro 43 - Eventos de escuta do sistema para redesenho do elemento *canvas*

```
[...]  
1  canvas.onmousedown = function(evt){  
2      var rectNav = canvas.getBoundingClientRect();  
3      var xclicado = evt.clientX - rectNav.left;  
4      var yclicado = evt.clientY - rectNav.top;  
5      if(evt.button == 0)  
6          clicou_canvas(xclicado, yclicado);  
7  
8      clearInterval(intervalo);  
9  }  
10  
11 canvas.onmouseup = function(evt){  
12     var rectNav = canvas.getBoundingClientRect();  
13     var xclicado = evt.clientX - rectNav.left;  
14     var yclicado = evt.clientY - rectNav.top;  
15     soltou_canvas(xclicado, yclicado);  
16     intervalo = window.setInterval(recuperaDados, 2000);  
17 }  
18  
19 canvas.onmousemove = function(evt){  
20     var rectNav = canvas.getBoundingClientRect();  
21     var xatual = evt.clientX - rectNav.left;  
22     var yatual = evt.clientY - rectNav.top;  
23     mouse_moving(xatual, yatual);  
24 }  
25  
26 canvas.onmouseover = function(evt){  
27     document.body.style.cursor = 'pointer';  
28 }  
29  
30 canvas.onmouseout = function(evt){  
31     document.body.style.cursor = 'default';  
32 }  
33  
34 function clicou_canvas(a, b){  
35  
36     var xclicado = a;  
37     var yclicado = b;  
38     elementoSelecionado = null;  
39     clicado = true;  
40     [...]  
41 }  
[...]
```

Fonte: Acervo do Autor (2013)

A função `canvas.onmousedown()`, apresentada nas linhas 1 a 9 do Quadro 43 é responsável pela execução de algumas rotinas quando o usuário clica sobre o `canvas`. As linhas 2 a 4 capturam o posicionamento do clique, o evento `clizou_canvas` (linha 6) verifica se algum elemento foi selecionado no clique e ativa a `flag` para o redesenho contínuo do componente de desenho. A função `canvas.onmouseup()`, apresentada pelas linhas 11 a 17 é bastante similar à anterior, porém é executada quando o usuário solta o botão do `mouse`. É feita a mesma verificação de posicionamento e desativada a `flag` para o redesenho. A função `canvas.onmousemove()` (linhas 19 a 24) é executada a cada movimento do `mouse`. Esta função atualiza o posicionamento do ponteiro (linhas 20 a 22) e redesenha a interface através da função `mouse_moving()`, apresentada na linha 23. As funções das linhas 26 a 32 são responsáveis pela mudança do cursor, quando o `mouse` estiver sobre o elemento `canvas`. A resposta do sistema a cada movimentação do `mouse` com algum elemento selecionado pode ser observada no trecho de código constante no Quadro 44.

Quadro 44 - Método executado a cada movimento do `mouse` com um elemento selecionado

```
[...]
1  function mouse_moving(xatual, yatual){
2      if(elementoSelecionado != null){
3          xdestino = xatual;
4          ydestino = yatual;
5
6          var deslocamentox = parseInt(xdestino) - parseInt(xoriginal);
7          var deslocamentoy = parseInt(ydestino) - parseInt(yoriginal);
8          var codigo = elementoSelecionado.codigo;
9
10         if(elementoSelecionado.tipo == 1){
11
12             for(var i = 0; i < array_entradas.length; i++){
13
14                 if(array_entradas[i].codigo == codigo){
15                     posxantiga = array_entradas[i].posx;
16                     posyantiga = array_entradas[i].posy;
17                     array_entradas[i].posx = parseInt(array_entradas[i].posx)
18                         + parseInt(deslocamentox);
19                     array_entradas[i].posy = parseInt(array_entradas[i].posy)
20                         + parseInt(deslocamentoy);
21                     reposiciona_linhas(posxantiga, posyantiga,
```

```
22         array_entradas[i].posx, array_entradas[i].posy);
23         xoriginal = xatual;
24         yoriginal = yatual;
25         redesenha();
26     }
27 }
28 }
29
30 if(elementoSelecioneado.tipo == 2){
31     for(var j = 0; j < array_saidas.length; j++){
32         if(array_saidas[j].codigo == elementoSelecioneado.codigo){
33             posxantiga = array_saidas [i].posx;
34             posyantiga = array_saidas[i].posy;
35             array_saidas[i].posx = parseInt(array_saidas[i].posx)
36                 + parseInt(deslocamentox);
37             array_saidas[i].posy = parseInt(array_saidas[i].posy)
38                 + parseInt(deslocamentoy);
39             reposiciona_linhas(posxantiga, posyantiga,
40                 array_saidas[i].posx, array_saidas[i].posy);
41             xoriginal = xatual;
42             yoriginal = yatual;
43             redesenha();
44         }
45     }
46 }
47
48 if(elementoSelecioneado.tipo == 3){
49     for(var k = 0; k < array_motores.length; k++){
50         if(array_motores[k].codigo == elementoSelecioneado.codigo){
51             posxantiga = array_motores[i].posx;
52             posyantiga = array_motores[i].posy;
53             array_motores[i].posx = parseInt(array_motores[i].posx)
54                 + parseInt(deslocamentox);
55             array_motores[i].posy = parseInt(array_motores[i].posy)
56                 + parseInt(deslocamentoy);
57             reposiciona_linhas(posxantiga, posyantiga,
58                 array_motores[i].posx, array_motores[i].posy);
59             xoriginal = xatual;
60             yoriginal = yatual;
61             redesenha();
62         }
63     }
64 }
```

```
65     }  
66  }  
[...]
```

Fonte: Acervo do Autor (2013)

Nas linhas 10, 30 e 48 o sistema verifica qual a natureza do componente selecionado. De acordo com o tipo de elemento, o sistema percorre o *array* correspondente (linhas 12, 31 e 49), verificando qual o componente selecionado. As linhas 15 a 22; 33 a 40 e 51 a 58 atualizam as coordenadas  $x$  e  $y$  do elemento e a interface é redesenhada através do método *redesenha()* (linhas 25, 43 e 61). O conteúdo dos métodos de desenho pode ser observado no Quadro 41.

A cada redesenho, a aplicação recupera os dados atualizados de cada componente no banco de dados. Como o controle do desenho ocorre através de rotinas JavaScript, se fez necessária a utilização de uma tecnologia que permitisse a recuperação desses dados a partir do navegador (*client-side*) utilizando AJAX. Para a implementação das funções AJAX de suporte ao redesenho foi utilizada a biblioteca DWR, a qual permite a realização dessa comunicação de forma facilitada.

### 3.8 OPERACIONALIZAÇÃO

Nesta seção será abordada a operacionalidade do sistema, ou seja, o detalhamento da realização dos casos de uso definidos e da forma como ocorre a interação entre o usuário e a ferramenta. Além disso, é detalhado o funcionamento do sistema na utilização dos recursos de trabalho colaborativo.

#### 3.8.1 Utilização do sistema

O ambiente está disponível em <http://bsi.ceavi.udesc.br:8082/FuzzyStudio>. Ao acessá-lo com qualquer navegador compatível, é apresentada a tela de boas vindas do sistema. No canto superior direito são apresentadas as opções para autenticação e novo cadastro na ferramenta. Ao entrar na opção de autenticação é apresentada a tela conforme Figura 32.

Figura 32 - Tela de autenticação do usuário

FuzzyStudio fs  
MODELAGEM DIFUSA

Você não está logado!

### Tela de Login

Username:

Senha:

Não possui usuário e senha? [Cadastrar-se](#).

Desenvolvido por Marcelo de Souza  
Trabalho de conclusão de curso - Bacharelado em Sistemas de Informação  
Universidade do Estado de Santa Catarina - UDESC / CEAVI

Fonte: Acervo do Autor (2013)

No caso do usuário não possuir cadastro no sistema, o mesmo pode realizá-lo acessando a página de cadastro de usuários, conforme Figura 33.

Figura 33 - Tela de cadastro de usuários

FuzzyStudio fs  
MODELAGEM DIFUSA

Você não está logado!

### Tela de cadastro de novos usuários

Nome:

Skype:

GTalk:

Username:

Senha:

Desenvolvido por Marcelo de Souza  
Trabalho de conclusão de curso - Bacharelado em Sistemas de Informação  
Universidade do Estado de Santa Catarina - UDESC / CEAVI

Fonte: Acervo do Autor (2013)



Na interface apresentada pela Figura 33, o usuário informa seus dados como nome, informações de contato (*skype* e *gtalk*) e dados de acesso (usuário e senha). Clicando em “Cadastrar” o usuário é salvo e redirecionado à tela de autenticação.

Uma vez autenticado no sistema, o usuário é redirecionado para sua central de projetos (Figura 34). Neste ponto é possível realizar a manutenção dos projetos. O usuário poderá criar um novo projeto (A), excluir algum já existente (B) ou compartilhá-lo com demais usuários (C). Também é possível realizar a abertura do projeto para edição (D).

Figura 34 – Central de projetos do Usuário

Central de projetos do Usuário

Usuário logado: Marcelo de Souza

**A**

Projetos

- Novo

Usuário

- Alterar dados do usuário

Outras opções

- Logout

**D** seus projetos

Nome do projeto	Usuários conectados	Ações		
Saude	0 usuário(s) conectado(s)	Abrir	Compartilhar	Excluir
NivelAluno	0 usuário(s) conectado(s)	Abrir	Compartilhar	Excluir
LinhasTransporte	0 usuário(s) conectado(s)	Abrir	Compartilhar	Excluir

**C**

**B**

Desenvolvido por Marcelo de Souza  
Trabalho de conclusão de curso - Bacharelado em Sistemas de Informação  
Universidade do Estado de Santa Catarina - UDESC / CEAVI

Fonte: Acervo do Autor (2013)

Ao compartilhar um projeto é apresentada uma lista dos usuários da aplicação, através da qual poderão ser selecionados os usuários que se deseja ter acesso ao projeto (Figura 35).

Figura 35 - Interface de compartilhamento de projetos

Fonte: Acervo do Autor (2013)

Para exemplificar o funcionamento da aplicação será aplicado um estudo de caso desenvolvido por Gabriel Filho et al. (2012). Este estudo de caso é composto por um sistema fuzzy para a classificação da mão de obra, auxiliando na escolha de funcionários para operar colheitadeiras de cana no setor sucroalcooleiro. A Figura 36 apresenta a página de cadastro do projeto.

Figura 36 - Tela de cadastro de projeto

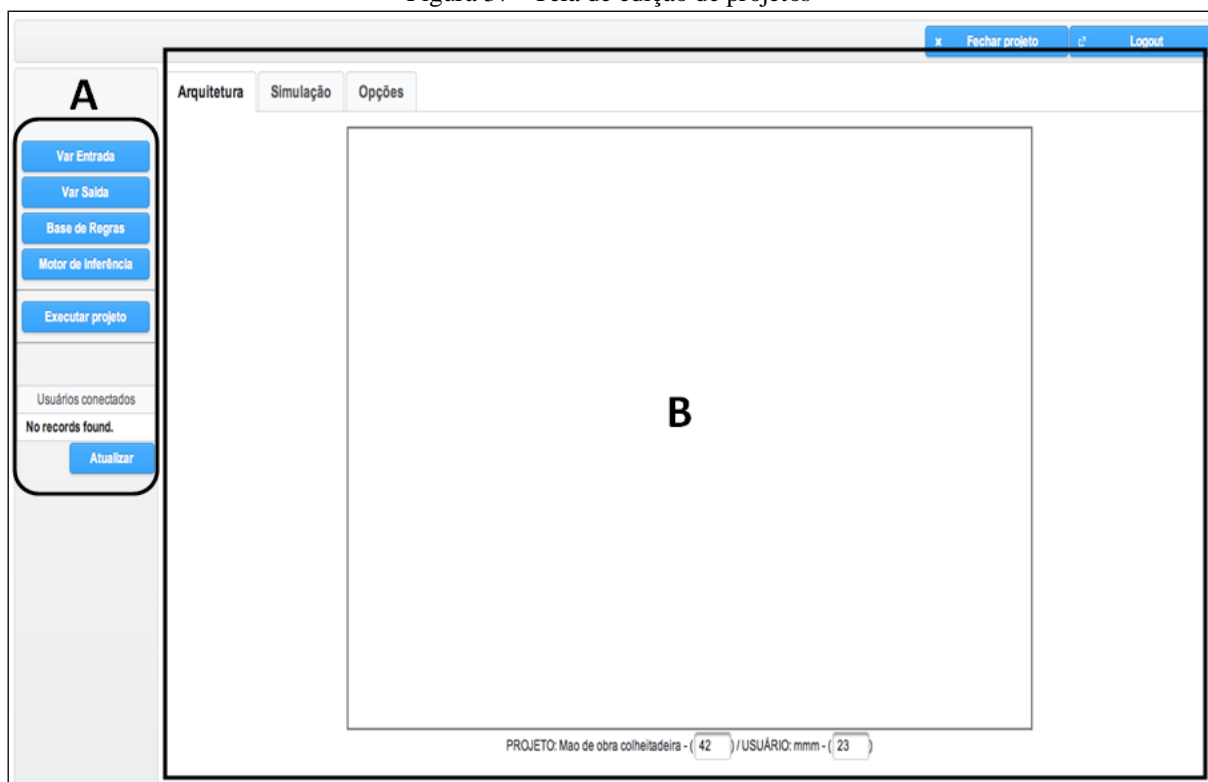
Fonte: Acervo do Autor (2013)

A tela de edição do projeto, ilustrada na Figura 37, está dividida em duas áreas fundamentais. A área A consiste no menu lateral esquerdo, através do qual é possível cadastrar e configurar os componentes do sistema fuzzy. Também é apresentada uma lista com os usuários conectados ao mesmo projeto naquele instante. Ao repousar o *mouse* sobre um usuário são apresentadas suas informações de contato (*skype* e *gtalk*).

A área B da tela de edição do projeto, ilustrada na Figura 37, configura a parte central da interface, onde o usuário efetivamente manipulará o modelo de sistema fuzzy. Esta área,

por sua vez, é dividida em guias para a separação das informações. A primeira guia apresenta a arquitetura desenvolvida, através de uma interface de desenho dos componentes. A segunda guia disponibiliza os mecanismos para a execução e simulação do sistema modelado (Figura 45). A terceira e última guia contém opções para o usuário exportar o modelo visual para uma imagem e o compartilhamento do projeto com demais usuários (Figura 48).

Figura 37 - Tela de edição de projetos



Fonte: Acervo do Autor (2013)

A criação das variáveis de entrada do sistema fuzzy é realizada na página ilustrada na Figura 38. Ao selecionar a opção “Var Entrada” no menu lateral, o usuário é redirecionado a essa página. No estudo de caso existem três variáveis: habilidade, conhecimento e atitude. Para cada variável foi definido o domínio de 0 a 10, que representa um valor de avaliação atribuído a cada pessoa para cada uma das variáveis analisadas.

Figura 38 - Tela de edição de variáveis de entrada

**Tela de edição de Variáveis de Entrada**

---

Nome:

De:

Até:

Unidade medida:

**A**

**Variáveis de entrada criadas**

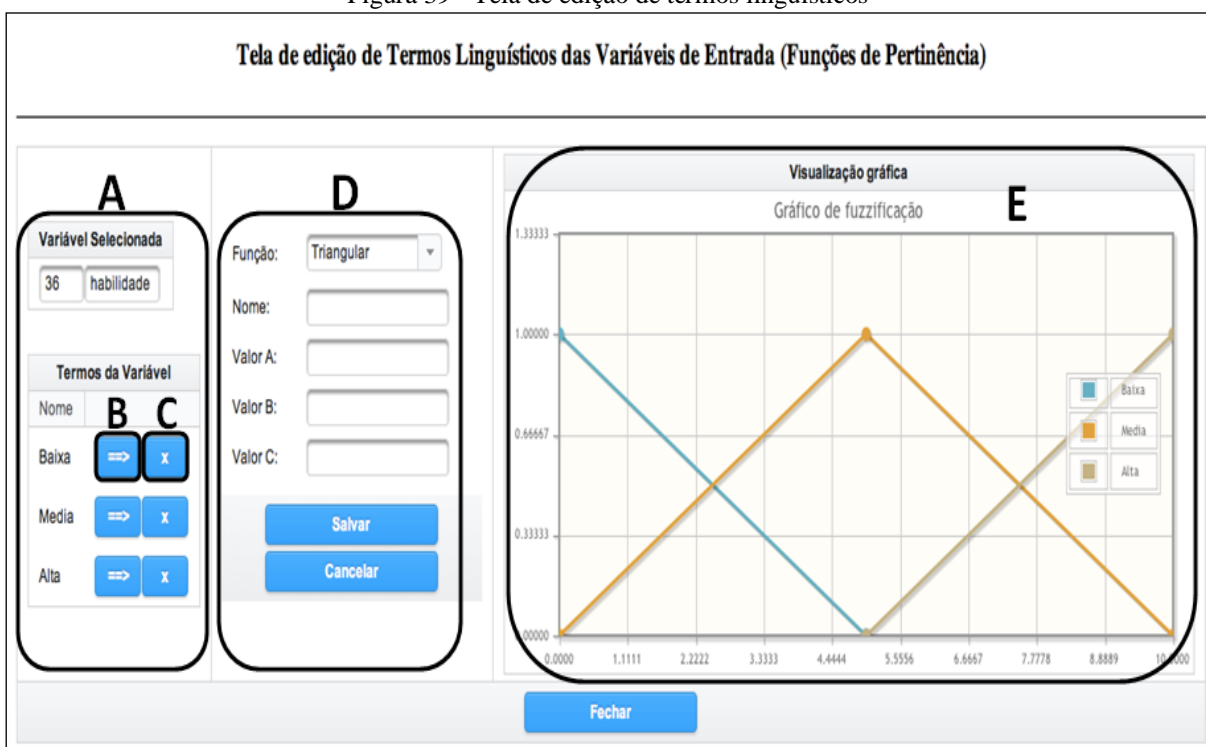
Nome	Mínimo	Máximo	Unidade	Ações
habilidade	0.0	10.0	nota	<div style="display: flex; gap: 5px;"> <span style="border: 1px solid black; padding: 2px;">C</span> <span style="border: 1px solid black; padding: 2px;">D</span> <span style="border: 1px solid black; padding: 2px;">E</span> </div>
conhecimento	0.0	10.0	nota	<div style="display: flex; gap: 5px;"> <span style="border: 1px solid black; padding: 2px;">C</span> <span style="border: 1px solid black; padding: 2px;">D</span> <span style="border: 1px solid black; padding: 2px;">E</span> </div>
atitude	0.0	10.0	nota	<div style="display: flex; gap: 5px;"> <span style="border: 1px solid black; padding: 2px;">C</span> <span style="border: 1px solid black; padding: 2px;">D</span> <span style="border: 1px solid black; padding: 2px;">E</span> </div>

**B**

Fonte: Acervo do Autor (2013)

Como se observa na Figura 38, é disponibilizado um formulário para a entrada dos dados de cada variável (A) e a listagem daquelas já cadastradas (B). Através dessa tabela é possível editar os dados de uma variável (C), excluí-la (D) ou cadastrar seus termos linguísticos (E). Ao selecionar a opção de cadastro de termos linguísticos (E) o usuário é redirecionado para a tela de edição ilustrada na Figura 39.

Figura 39 - Tela de edição de termos linguísticos

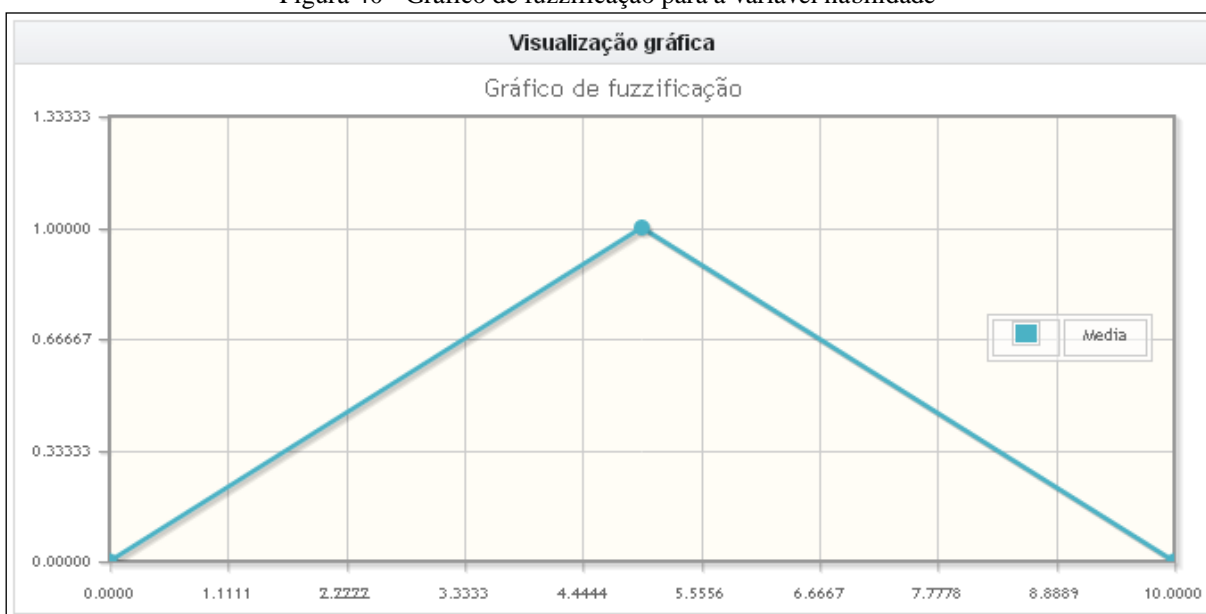


Fonte: Acervo do Autor (2013)

A Figura 39 apresenta os termos linguísticos para a variável de entrada habilidade: Baixa, Média e Alta, que respondem às funções de pertinência “Inclinação Esquerda”, “Triangular” e “Inclinação Direita”, respectivamente. As três variáveis de entrada do estudo de caso terão os mesmos termos linguísticos, inclusive suas configurações. A tela de edição de termos linguísticos, apresentada na Figura 38, está dividida em três áreas. A primeira (A) apresenta a variável em questão e a lista de termos linguísticos já cadastrados. Essa lista ainda permite, através dos botões, o carregamento do termo para edição (B) ou sua exclusão (C). A segunda parte (D) consiste na configuração do termo. Primeiramente é selecionada a função de pertinência para o termo linguístico (campo “Função”). O sistema implementa sete funções de pertinência: triangular, trapezoidal, discreta, inclinação direita e esquerda, rampa direita e esquerda. Ao selecionar uma função os campos referentes a seus parâmetros são apresentados para o preenchimento por parte do usuário, juntamente com o nome do termo.

Por fim, a área E destina-se ao gráfico de fuzziificação da variável de entrada selecionada, apresentando os termos linguísticos. A Figura 40 exemplifica o termo Média que usa a função triangular.

Figura 40 - Gráfico de fuzzificação para a variável habilidade



Fonte: Acervo do Autor (2013)

Uma vez cadastradas as variáveis de entrada do sistema fuzzy são cadastradas as variáveis de saída. O processo de inserção de variáveis de saída é idêntico ao de variáveis de entrada. Para o estudo de caso será considerada uma variável de saída denominada colheitadeira, representando o grau de competência do funcionário ao cargo de operador da máquina. Os limites dessa variável serão de 0 a 10 e seus termos linguísticos responderão ao mesmo comportamento das variáveis de entrada (Baixa, Média e Alta).

Depois de cadastradas as variáveis de entrada e saída é definida a base de regras. Para isso, o usuário deve selecionar a opção "Base de Regras" no menu lateral, sendo redirecionado para a tela de edição da base de regras, que está ilustrada na Figura 41.

Figura 41 - Tela de edição da base de regras

**Tela de edição de Base de Regras do Sistema Fuzzy**

Conector E (AND)

**A**

**Monte o ANTECEDENTE da regra**

habilidade	<input type="text"/>
conhecimento	<input type="text"/>
atitude	<input type="text"/>

**Monte o CONSEQUENTE da regra**

colheitadeira	<input type="text"/>
---------------	----------------------

Adicionar Regra

**B** **C**

Antecedente	Consequente	Ações
IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS Baixa	THEN colheitadeira IS Baixo	<span style="background-color: #007bff; color: white; padding: 5px 15px; border-radius: 5px;">Excluir</span>
IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS Media	THEN colheitadeira IS Medio	<span style="background-color: #007bff; color: white; padding: 5px 15px; border-radius: 5px;">Excluir</span>
IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS Alta	THEN colheitadeira IS Medio	<span style="background-color: #007bff; color: white; padding: 5px 15px; border-radius: 5px;">Excluir</span>

Fechar

Fonte: Acervo do Autor (2013)

A tela para edição de regras apresenta um quadro (A) com as opções para montagem da regra. O usuário define o conector para os antecedentes da regra e depois seleciona os valores para ao menos uma variável de entrada e uma variável de saída, definindo assim o antecedente e o consequente da regra. Ao adicionar a regra, a mesma é inserida em uma lista (B). Através dessa lista o usuário tem a opção de excluir (C) uma regra adicionada. Seguindo o estudo de caso de classificação de mão de obra, as 27 regras definidas pelos autores foram implementadas no sistema.

Para que o sistema fuzzy possa inferir as saídas com base nas regras desejadas é necessário a criação de um motor de inferência. Novamente essa página é acessada através da opção “Motor de Inferência” do menu lateral. Para cada motor de inferência devem-se configurar os parâmetros do sistema fuzzy conforme Figura 42, que apresenta a página de edição de motores de inferência.

Figura 42 - Tela de edição de motores de inferência

Configurações do motor de inferência

Identificador / Nome:

**Método de Defuzzificação:**

Centro da Gravidade  
  Right Most Max  
  Centro da Área  
  Left Most Max

**Método de Agregação de Regras:**

Soma Limitada (BoundedSum)  
  Máximo (Max)  
  Soma (Sum)  
  Soma Normalizada (NormedSum)

**Conexão AND:**

Mínimo  
  Produto  
  Soma Limitada

**Método de Ativação de Regras:**

Mínimo (Min)  
  Produto

Escolha as regras deste motor

IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS Baixa ==> THEN colheitadeira IS Baixo

IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS Media ==> THEN colheitadeira IS Medio

IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS Alta ==> THEN colheitadeira IS Medio

Motores criados	
Identificador	Ações
mamdani	<input style="background-color: #007bff; color: white; padding: 5px 10px;" type="button" value="Editar"/> <input style="background-color: #007bff; color: white; padding: 5px 10px; margin-left: 10px;" type="button" value="Excluir"/>

Fonte: Acervo do Autor (2013)

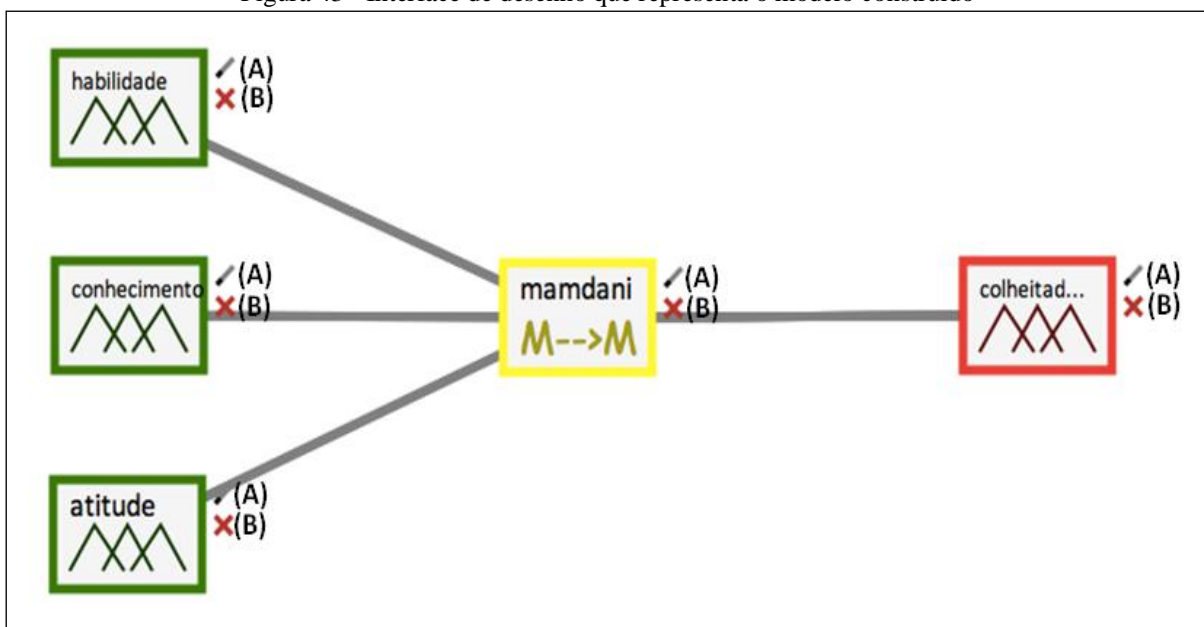
As configurações básicas do motor de inferência são o nome, o método de defuzzificação, método de agregação das regras, conector AND e método de ativação das regras. O sistema implementa quatro métodos de defuzzificação: Centro da Gravidade, *Right Most Max*, Centro da Área e *Left Most Max*. Como método de agregação das regras são disponibilizadas quatro opções: Soma Limitada, Máximo, Soma Simples e Soma Normalizada. Como conectores AND estão disponíveis três opções: Mínimo, Produto e Soma Limitada. Por fim, a ferramenta implementa dois métodos de ativação das regras: Mínimo e Produto. Além das configurações de processamento dos dados, também devem ser definidas as regras associadas ao motor, relacionando quais regras de toda a base que serão utilizadas pelo motor para a inferência das saídas. Na parte inferior da página é apresentada a lista de motores, através da qual se pode excluir ou editar um motor já existente.

Durante a construção do modelo, o sistema apresenta cada componente e suas respectivas relações na interface de desenho. Para as variáveis de entrada é desenhado um



retângulo verde contendo seu nome. As variáveis de saída são apresentadas através de um retângulo vermelho juntamente com seu nome. Os motores de inferência, por sua vez, são representados por retângulos amarelos. A Figura 43 apresenta a área de desenho gerada para o modelo construído a partir do estudo de caso analisado. O usuário ainda tem a possibilidade de mover os componentes em tela, de modo a dispor os mesmos conforme sua necessidade.

Figura 43 - Interface de desenho que representa o modelo construído



Fonte: Acervo do Autor (2013)

Ao lado de cada componente desenhado encontram-se dois ícones. Um deles (A) trata-se da edição do componente. Ao clicar sobre o mesmo é possível editar os atributos básicos do componente selecionado. No caso da seleção de uma variável de entrada é apresentado um formulário conforme apresentado pela Figura 44. O botão (B) trata-se da exclusão do componente selecionado.

Figura 44 - Edição de variáveis de entrada pela interface de desenho

The screenshot shows a dialog box titled "Edição de Variáveis de Entrada" with three tabs: "Arquitetura", "Simulação", and "Opções". The "Simulação" tab is active. The dialog contains the following fields:

Nome:	<input type="text" value="habilidade"/>
Mínimo:	<input type="text" value="0"/>
Máximo:	<input type="text" value="10"/>
Unidade de medida:	<input type="text" value="nota"/>

At the bottom of the dialog are two buttons: "Salvar" and "Cancelar".

Fonte: Acervo do Autor (2013)

A simulação consiste na execução do sistema modelado através da definição de valores para suas entradas, recuperando as saídas inferidas. Esta opção está disponível na segunda guia do sistema, denominada “Simulação”, ilustrada na Figura 45. Esta guia apresenta ao usuário uma tabela com as variáveis de entrada do sistema (A). Através desta tabela, o usuário informa o valor de cada uma das entradas que o sistema utilizará para o processamento. Clicando no botão “Executar sistema” é realizada a execução do modelo construído. Os resultados são apresentados na área B, a qual contém cada um dos motores do sistema e suas respectivas saídas.

Figura 45 - Interface de execução do sistema

The screenshot shows the "Simulação" tab in the software interface. It is divided into two main sections, A and B.

**Section A: Informe o valor das variáveis de entrada**

Nome	Valor
habilidade	<input type="text"/>
conhecimento	<input type="text"/>
atitude	<input type="text"/>

Below this table is a blue button labeled "Executar sistema".

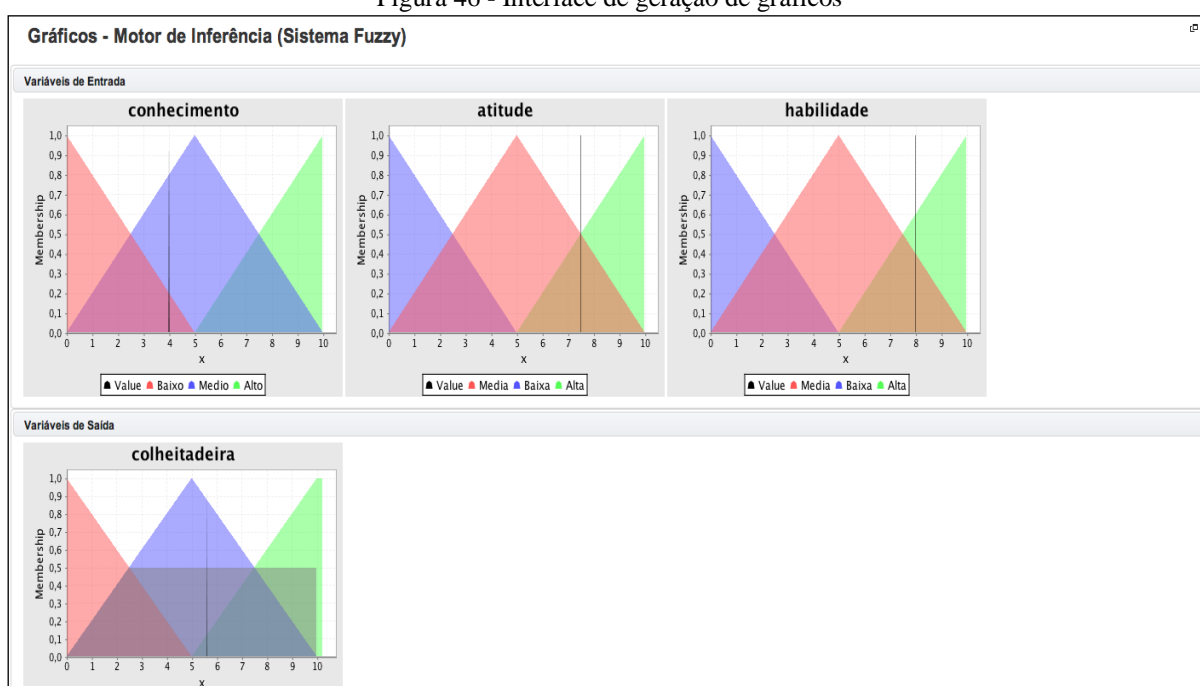
**Section B: Motores de Inferência disponíveis para execução**

Nome	Resultado	Ações
mamdani	<input type="text"/>	<input type="button" value="Gráficos"/> <input type="button" value="FCL"/> <input type="button" value="Java"/>

Fonte: Acervo do Autor (2013)

A partir da execução do sistema modelado é possível a geração de artefatos referentes à simulação realizada. Essas opções estão disponíveis para cada motor de inferência da tela de execução do sistema. A primeira delas apresenta um gráfico para cada variável envolvida no sistema fuzzy, incluindo as marcações de valores de entrada e defuzzificação. A Figura 46 apresenta um exemplo de geração de gráficos para o estudo de caso analisado. Observa-se que nas variáveis de entrada há uma linha vertical ilustrando o valor da entrada informada pelo usuário. O mesmo comportamento é observado nas variáveis de saída, com o valor inferido pelo sistema.

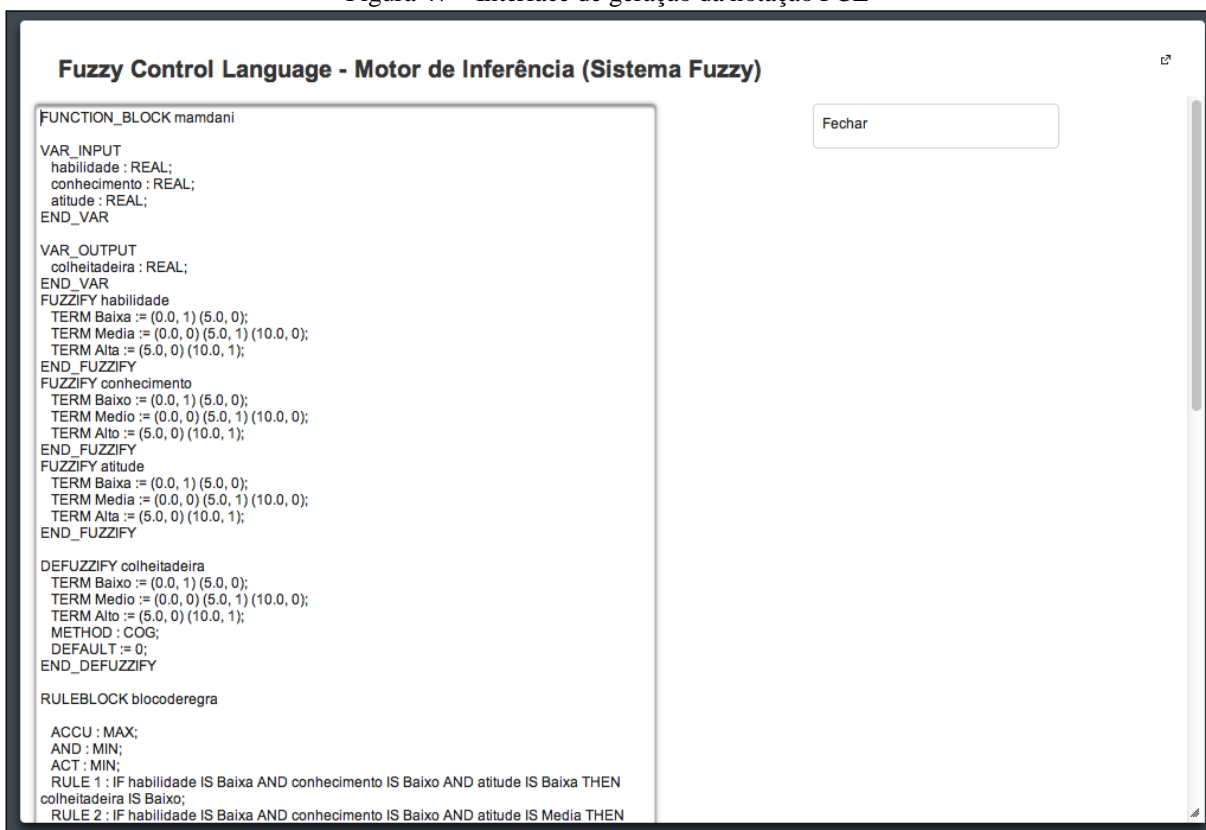
Figura 46 - Interface de geração de gráficos



Fonte: Acervo do Autor (2013)

A segunda opção de geração de artefatos se apresenta como FCL, contendo a geração de notação no padrão *Fuzzy Control Language* e a geração da classe Java. Para isso, o usuário seleciona a opção FCL e é redirecionado à página de apresentação da notação gerada, a qual é observada na Figura 47. O código FCL é apresentado em modo texto ao usuário.

Figura 47 – Interface de geração da notação FCL



```

FUNCTION_BLOCK mamdani
VAR_INPUT
  habilidade : REAL;
  conhecimento : REAL;
  atitude : REAL;
END_VAR
VAR_OUTPUT
  colheitadeira : REAL;
END_VAR
FUZZIFY habilidade
  TERM Baixa := (0.0, 1) (5.0, 0);
  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0);
  TERM Alta := (5.0, 0) (10.0, 1);
END_FUZZIFY
FUZZIFY conhecimento
  TERM Baixo := (0.0, 1) (5.0, 0);
  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0);
  TERM Alto := (5.0, 0) (10.0, 1);
END_FUZZIFY
FUZZIFY atitude
  TERM Baixa := (0.0, 1) (5.0, 0);
  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0);
  TERM Alta := (5.0, 0) (10.0, 1);
END_FUZZIFY
DEFUZZIFY colheitadeira
  TERM Baixo := (0.0, 1) (5.0, 0);
  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0);
  TERM Alto := (5.0, 0) (10.0, 1);
  METHOD : COG;
  DEFAULT := 0;
END_DEFUZZIFY
RULEBLOCK blocoderegra
  ACCU : MAX;
  AND : MIN;
  ACT : MIN;
  RULE 1 : IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS Baixa THEN
    colheitadeira IS Baixo;
  RULE 2 : IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS Media THEN

```

Fonte: Acervo do Autor (2013)

A terceira opção de geração de artefatos é a geração de código em linguagem de programação Java. Ao selecionar a opção, é apresentado ao usuário uma página contendo uma classe Java (A), conforme ilustrado pela Figura 48. Ao lado é apresentado um menu (B), o qual permite o *download* da biblioteca necessária para executar a classe gerada (*jFuzzyLogic*), assim como leva à página na Internet desta biblioteca.

Figura 48 - Interface de geração de código Java

The screenshot shows a web-based interface for generating Java code. The main area displays the following code:

```

Classe Java Gerada
import net.sourceforge.jFuzzyLogic.FIS;
import net.sourceforge.jFuzzyLogic.FunctionBlock;
import org.antlr.runtime.RecognitionException;

public class FuzzyStudioFCL {

    private static String fcl;

    public static void main(String[] args) throws RecognitionException {

        fcl = "FUNCTION_BLOCK mamdani \n"+
            "\n"+
            "VAR_INPUT \n"+
            "  habilidade : REAL;\n"+
            "  conhecimento : REAL;\n"+
            "  atitude : REAL;\n"+
            "END_VAR\n"+
            "\n"+
            "VAR_OUTPUT \n"+
            "  colheitadeira : REAL;\n"+
            "END_VAR\n"+
            "FUZZIFY habilidade \n"+
            "  TERM Baixa := (0.0, 1) (5.0, 0); \n"+
            "  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alta := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "FUZZIFY conhecimento \n"+
            "  TERM Baixo := (0.0, 1) (5.0, 0); \n"+
            "  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alto := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "FUZZIFY atitude \n"+
            "  TERM Baixa := (0.0, 1) (5.0, 0); \n"+
            "  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alto := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "\n"+
            "DEFUZZIFY colheitadeira \n"+
            "  TERM Baixo := (0.0, 1) (5.0, 0); \n"+
            "  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alto := (5.0, 0) (10.0, 1); \n"+
            "METHOD : COG; \n"+
            "DEFAULT := 0; \n"+
    
```

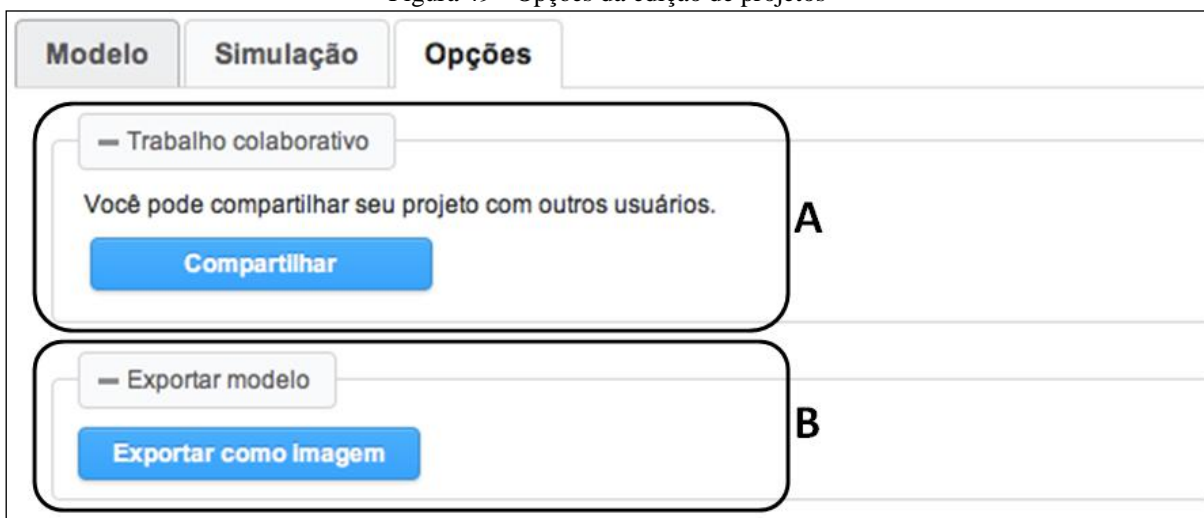
On the right side of the interface, there are three buttons: "Download jFuzzyLogic", "Leia mais..", and "Fechar".

Fonte: Acervo do Autor (2013)

A classe Java gerada para esse estudo de caso encontra-se no Apêndice A. O código FCL está no Apêndice B. A execução da classe Java apresenta as janelas contidas no Apêndice C.

A terceira guia da tela de edição de projetos chama-se “Opções”, conforme Figura 49. Esta guia apresenta as funcionalidades de compartilhamento do projeto (A) e exportação da imagem do modelo (B).

Figura 49 - Opções da edição de projetos



Fonte: Acervo do Autor (2013)

### 3.8.2 Trabalho colaborativo

O FuzzyStudio permite o trabalho colaborativo sobre um mesmo projeto. Para que mais de um usuário possa trabalhar sobre o mesmo projeto, primeiramente é necessário que o mesmo seja compartilhado (Figura 35 e 49).

Quando mais de um usuário estiver conectado ao mesmo projeto, qualquer alteração realizada no modelo é replicado aos demais usuários. Para garantir essa consistência, a aplicação realiza requisições ao servidor de tempos em tempos, buscando por alterações nos componentes constantes no banco de dados. No caso de haverem alterações, a aplicação atualiza a interface de desenho (Figura 43), apresentando as modificações realizadas.

Para tratar do acesso concorrente, a aplicação delega a responsabilidade para o banco de dados, isto é, o controle de leitura e alterações de elementos do sistema difuso é realizado pelo MySQL. Este controle busca evitar a inconsistência na construção do projeto. Um caso comum é ilustrado por dois usuários editando os dados de uma mesma variável simultaneamente. Neste impasse, o usuário que realiza a operação por último tem suas alterações salvas no banco de dados.

## 3.9 EXPERIMENTAÇÃO E VALIDAÇÃO

Para testar o sistema, buscando verificar seu correto funcionamento, foram realizados alguns experimentos. O sistema FuzzyStudio foi implantado no servidor GlassFish versão

3.1.1 localizado na UDESC Ibirama. A implantação ocorreu entre os dias quinze e dezesseis de maio de dois mil e treze. Durante o processo de implantação foram encontrados alguns problemas. O mapeamento dos arquivos DWR teve que ser modificado, atualizando as referências para direcionamento ao servidor. Os arquivos de configuração do banco de dados também sofreram ajustes pela mudança de porta. Após a correção dos problemas foram criados um *pool de conexões*<sup>8</sup> no servidor e um *DataSource*<sup>9</sup> para o sistema, que foi finalmente implantado. No mesmo dia dezesseis foi realizado o primeiro experimento.

Esse experimento consistiu no acesso simultâneo de dois usuários ao sistema, o autor Marcelo de Souza e Adilson Vahldick, professor orientador. Na ocasião, foram criados alguns projetos como teste, sendo um deles compartilhado entre os usuários. Ambos trabalharam simultaneamente no projeto compartilhado modelando o sistema fuzzy. Foram testadas a criação dos componentes do sistema, definição das regras, execução do motor, controle de sessão e atualização do ambiente em tempo real. Na oportunidade foram listados ajustes e pequenas correções para melhoria do sistema. Dentre elas estavam ajustes na usabilidade da ferramenta, erros na geração de gráficos de fuzzificação e defuzzificação, melhorias nas chamadas AJAX e ajustes no leiaute de algumas páginas.

Foram implementadas as melhorias no sistema, o qual foi novamente implantado em sua versão oficial no dia dezessete de maio de dois mil e treze. No dia vinte e dois de maio de dois mil e treze às dezenove horas o sistema passou pelo seu segundo experimento, buscando avaliar o funcionamento nas etapas de modelagem e execução do sistema difuso e a geração de seus artefatos. O FuzzyStudio foi aplicado na turma de Inteligência Artificial do curso de Sistemas de Informação da Universidade do Estado de Santa Catarina. Para realizar a aplicação da ferramenta na disciplina estava presente o professor Adilson Vahldick, que ministra a disciplina, juntamente com seis alunos.

Estes alunos realizaram seus cadastros no ambiente, autenticando-se para acesso completo às funcionalidades. Foi definido um exercício, composto por um sistema fuzzy completo que os alunos deveriam modelar e simular na ferramenta. Para desenvolvimento do exercício pelos alunos, nenhuma instrução de funcionamento ou utilização do FuzzyStudio foi fornecida. Durante a execução do exercício utilizando a ferramenta, os alunos não tiveram

---

<sup>8</sup> Recurso do servidor de aplicações que mantém uma série de conexões com o banco de dados, fornecendo uma delas quando solicitado. Evita a criação excessiva de conexões (SAMPAIO, 2011).

<sup>9</sup> É uma fonte de dados configurada no servidor de aplicações. Elimina da aplicação a configuração de acesso aos dados (SAMPAIO, 2011).

grandes dificuldades na sua utilização. Todos os acadêmicos conseguiram concluir o exercício com sucesso em uma hora de aula, modelando e simulando o sistema fuzzy fornecido. A ferramenta FuzzyStudio apresentou-se eficaz na construção de sistemas fuzzy, com bom tempo de resposta e sem apresentar nenhum erro de execução.

Após a realização do exercício, foi solicitada a entrega de um trabalho, o qual consistiu na modelagem, simulação e geração de artefatos para o controlador fuzzy proposto por Paula e Sousa (2005) em seu estudo sobre o sistema de controle de tráfego em rodovias dotadas de faixa exclusiva para ônibus. Na ocasião, os alunos tiveram quatro dias para a modelagem do sistema difuso no FuzzyStudio. Como parte do trabalho, os acadêmicos deveriam apontar melhorias para a ferramenta, bem como possíveis erros detectados durante sua utilização. Dos oito alunos, cinco realizaram o trabalho. Os demais não fizeram por motivos particulares não relacionados com a ferramenta.

Ao final da entrega dos trabalhos, foi aplicado um questionário com os acadêmicos, dos quais cinco estiveram presentes na sua aplicação, que buscou mensurar a facilidade de uso dos recursos oferecidos. O questionário divide-se em quinze atividades, desde o cadastro do usuário no sistema até a exportação do modelo para uma imagem. Para cada atividade o aluno deve responder seu grau de satisfação com a facilidade apresentada pela ferramenta. As possíveis respostas variam entre “Completamente Satisfeito”, “Satisfeito”, “Insatisfeito” e “Completamente Insatisfeito”. O questionário completo pode ser observado no Apêndice D.

### 3.10 RESULTADOS E DISCUSSÕES

Como resultado deste trabalho foi produzida uma ferramenta com a qual é possível realizar a modelagem de um sistema difuso. Essa modelagem ocorre através da inclusão e configuração dos componentes de um sistema fuzzy, definindo as variáveis de entrada e saída, seus termos linguísticos e funções de pertinência, a base de regras e motores de inferência.

Com o modelo desenvolvido é possível realizar sua simulação. Neste processo são informados os valores de entrada e a ferramenta calcula suas saídas. Como diferenciais buscados para a ferramenta procurou-se o desenvolvimento de uma aplicação em plataforma Web, facilitando a sua utilização por não necessitar de instalação ou software adicional no cliente. Além disso, a ferramenta disponibiliza a geração de gráficos para cada variável da execução, geração de código no padrão da notação FCL e de linguagem de programação em



Java. O FuzzyStudio permite o compartilhamento de um projeto com outros usuários, possibilitando o trabalho colaborativo em tempo real na modelagem do sistema fuzzy.

A decisão de isolar o motor de inferência das regras, definindo configurações exclusivas e um arranjo de regras específico para cada motor desenvolvido também permitiu a criação de uma funcionalidade diferente dos trabalhos correlatos. Seguindo esta arquitetura, é possível a execução isolada de cada motor e a possibilidade do usuário criar diversos sistemas fuzzy dentro de um mesmo modelo. Este fator facilita o processo de testes e aferições para ajustes no sistema difuso.

A análise dos trabalhos correlatos foi importante para o desenvolvimento da aplicação. A utilização do sistema InFuzzy permitiu um melhor entendimento de todo o processo de construção de um sistema fuzzy, bem como as características do modelo Mamdani. A implementação da interface gráfica para desenho do modelo também se baseou neste sistema. O Matlab Fuzzy Toolbox auxiliou no mapeamento das funções de pertinência e métodos de defuzzificação, bem como na geração do modelo em modo texto. A estrutura adotada para definição de antecedentes e consequentes das regras também se baseou nas soluções correlatas.

Quanto à execução do modelo pode-se afirmar que sua qualidade está intimamente relacionada com a biblioteca *jFuzzyLogic*, utilizada como implementação do modelo Mamdani. A biblioteca proporcionou facilidade em simular o modelo desenvolvido, uma vez que a mesma possui interfaces de integração bastante simples de utilizar, além do fato de ser *open source*.

A ferramenta FuzzyStudio foi submetida a duas etapas de experimentação. A primeira delas tratou da utilização do ambiente por dois usuários simultaneamente, testando todas as funcionalidades implementadas. Nestas verificações foi comprovado o funcionamento do sistema e apontadas algumas melhorias. Dentre as melhorias levantadas pelos usuários estiveram questões de usabilidade, erros na geração de gráficos, melhorias em funções AJAX e ajustes no layout das telas. Com essas melhorias seguiu-se na segunda experimentação.

A segunda experimentação consistiu no uso da ferramenta pela turma de Inteligência Artificial do curso de Bacharelado em Sistemas de Informação da Universidade do Estado de Santa Catarina de Ibirama. Em um primeiro momento a utilização do FuzzyStudio ocorreu de forma dirigida pelo professor, que acompanhou os alunos durante a implementação do exercício proposto em sala de aula. A ferramenta apresentou bom desempenho em tempo de resposta. Também não ocorreu nenhum erro de execução que impossibilitasse seu uso. Os

alunos não tiveram maiores dificuldades na construção do modelo e execução do sistema difuso.

Após a ambientação dos acadêmicos com o FuzzyStudio, foi solicitada a execução de um trabalho. Os alunos tiveram quatro dias para implementar o sistema proposto por Paula e Sousa (2005). Dos oito, cinco acadêmicos concluíram o trabalho no prazo estipulado. De maneira geral, as duas experimentações realizadas com a ferramenta apontaram o correto funcionamento do sistema, bem como sua aderência com a especificação. A solução resolve os problemas por ela propostos e é eficaz na modelagem e simulação de sistemas fuzzy.

Os acadêmicos ainda responderam a um questionário, informando do sucesso na utilização dos recursos disponibilizados pela ferramenta. Esta pesquisa buscou apontar a facilidade de operação do software desenvolvido através do mapeamento das funcionalidades da ferramenta e do grau de satisfação dos usuários em utilizá-las. Neste sentido, o questionário se baseou em quinze atividades, para as quais o acadêmico respondeu sua satisfação com base na facilidade encontrada. Os resultados desta pesquisa são apresentados no Apêndice E.

Conforme os resultados coletados, os acadêmicos não apresentaram maiores dificuldades em operar a ferramenta e utilizar seus recursos. De acordo com as avaliações, os recursos de autenticação e criação de projetos foram avaliados como os mais fáceis e com maior nível de satisfação. A rotina de criação de regras foi a atividade avaliada com menor grau de satisfação por parte dos usuários, possivelmente por não apresentar o encadeamento de antecedentes com mais de um tipo de conector lógico. A edição de componentes visuais, geração dos gráficos e exportação do modelo para uma imagem também teve uma ocorrência de insatisfação cada, apesar dos demais usuários avaliarem os aspectos como fáceis. De maneira geral, a ferramenta pode ser considerada fácil de utilizar, pelo sucesso de todos os acadêmicos nas experimentações e resultados do questionário aplicado como pesquisa qualitativa.

Os acadêmicos ainda sugeriram ajustes e melhorias na ferramenta, com base na experiência de utilização para a resolução dos exercícios propostos. Os aspectos colocados pelos alunos foram:

- a) Adicionar um atributo de descrição na variável, detalhando seu significado;
- b) Múltiplos conectores lógicos nos antecedentes das regras;
- c) Adicionar campo de selecionar todas as regras na vinculação com o motor;
- d) Opção de salvar os gráficos gerados na execução;

- e) Ajustes no controle de foco nos campos e botões dos formulários;
- f) Reposicionar o botão fechar da tela de gráficos para que o mesmo não se oculte;
- g) Ajustar a vinculação de regras para exibir aquelas que já estão vinculadas ao motor.

A maior parte das sugestões levantadas pelos acadêmicos referem-se à usabilidade do sistema. Com exceção do aspecto b, as demais melhorias facilitam o uso e tornam a ferramenta mais intuitiva e amigável. Já a possibilidade de múltiplos conectores nos antecedentes das regras configura-se como uma nova funcionalidade que impacta na estrutura do sistema. Encadeando conectores no antecedente de uma regra faz com que seja possível representar várias regras na mesma. Esta funcionalidade é apresentada como uma extensão futura da ferramenta.

Com relação aos trabalhos correlatos, produziu-se uma tabela comparativa, enumerando algumas funcionalidades e sua contemplação ou não por cada uma das soluções, inclusive a ferramenta FuzzyStudio. Esta comparação é apresentada pelo Quadro 45.

Quadro 45 - Comparação entre os trabalhos correlatos e o FuzzyStudio

Recurso	Matlab Fuzzy Toolbox	InFuzzy	FuzzyGen	FuzzyStudio
Sistema executado em plataforma Web				X
Criação de múltiplos motores de inferência independentes				X
Exportação do sistema fuzzy no padrão FCL			X	X
Geração de códigos de programação				X
Compartilhamento do projeto com outros usuários				X
Trabalho colaborativo				X
Interface de desenho do modelo construído	X	X		X
Exportação do desenho do modelo		X		X
Possibilidade de simular / executar o sistema	X	X	X	X
Visualização de gráficos da execução	X	X	X	X
Comunicação da aplicação com outros sistemas	X	X		
Simulação de múltiplos valores simultaneamente		X		
Configuração dos métodos de agregação e acumulação de regras	X	X	X	X
Visualização das regras ativadas na simulação	X	X		
Depuração passo a passo	X			
Visualização gráfica de superfície	X			
Atribuição de peso para as regras	X		X	
Distribuição gratuita		X		X

Fonte: Acervo do Autor (2013)

A tabela apresentada no Quadro 45 está composta por dezoito itens identificados na ferramenta FuzzyStudio e seus trabalhos correlatos. O recurso de ser executado em plataforma Web é um aspecto importante, pois torna a aplicação independente de plataforma e facilita seu acesso e utilização. Este fator é contemplado com exclusividade pelo FuzzyStudio. A criação de múltiplos motores de inferência e sua execução isolada permite a criação de distintos sistemas fuzzy em um mesmo modelo, através da vinculação de um conjunto de regras para cada motor e distintas configurações. Esta é uma funcionalidade presente na ferramenta FuzzyStudio como diferencial em relação às demais. A possibilidade de exportação do modelo no padrão FCL permite a utilização do sistema fuzzy em qualquer aplicação que

implemente o padrão da norma IEC 1136-7, garantindo sua portabilidade e escalabilidade. Este recurso está presente no FuzzyStudio e também na ferramenta FuzzyGen. As ferramentas Matlab e InFuzzy exportam o sistema construído em notação textual seguindo padrões próprios.

Outra funcionalidade diferencial do FuzzyStudio em relação às demais soluções é a geração de código em linguagem de programação. Com isso, o desenvolvimento de aplicações dotadas de inteligência através da lógica difusa é facilitado. As possibilidades de compartilhamento do projeto com outros usuários e o trabalho colaborativo em um mesmo modelo permitem a construção de sistemas fuzzy por mais de uma pessoa em tempo real. Além disso, em ambiente acadêmico é possível realização de atividades em grupo. Estas funcionalidades estão presentes com exclusividade na ferramenta FuzzyStudio. Uma interface gráfica que desenhe o modelo desenvolvido através de componentes visuais facilita o entendimento tanto de sistemas fuzzy, quanto da estrutura do modelo. Este recurso está presente nas ferramentas Matlab, InFuzzy e FuzzyStudio. Além disso, as ferramentas InFuzzy e FuzzyStudio permitem a exportação deste desenho para uma imagem.

O recurso de executar o sistema modelado, simulando seu comportamento é necessário para a verificação da corretude do modelo. Através da execução é possível realizar testes e observar como o sistema reage a diferentes entradas. Este recurso está presente em todas as ferramentas analisadas. A geração de gráficos relacionados à execução também é um recurso implementado por todas as soluções. As ferramentas Matlab e InFuzzy oferecem o recurso de integração com outras aplicações, o que ocorre por meio de uma linguagem específica, no primeiro caso e por protocolos de comunicação em rede, no segundo sistema. Este recurso é implementado de forma diferencial por estes softwares. A simulação do sistema com múltiplos valores de entrada facilita as tarefas de testes e aferições, uma vez que se pode provar uma série de valores ao mesmo tempo. Este recurso é oferecido com exclusividade pelo InFuzzy. Todas as ferramentas consideradas possibilitam a definição de métodos de agregação e acumulação de regras pelo usuário, permitindo que o mesmo configure a forma de processamento na inferência dos resultados.

Quanto à simulação, algumas ferramentas implementam funcionalidades para auxílio na execução do modelo. As ferramentas Matlab e InFuzzy apresentam ao usuário quais regras foram ativadas na execução. O Matlab ainda oferece a possibilidade de depuração passo a passo da execução e a visualização gráfica da superfície gerada pelos componentes do sistema modelado. Estes recursos permitem um melhor entendimento dos passos realizados pelo

sistema para a consecução dos resultados, bem como facilitam na identificação de erros e inconsistências. As ferramentas Matlab e FuzzyGen permitem o cadastro de peso para as regras, no momento de ativação e acumulação das regras é considerada sua representatividade através deste valor. Desta forma é possível a modelagem da base com regras prioritárias e de maior importância. Quanto à distribuição das ferramentas, apenas os softwares InFuzzy e FuzzyStudio são disponibilizados gratuitamente.

É possível observar que a ferramenta FuzzyStudio traz funcionalidades não encontradas em outras soluções do mercado. Não há nenhuma ferramenta completa para modelagem de sistemas fuzzy sendo executada na Web, o que dificulta o acesso de muitos usuários às demais soluções apresentadas, por não serem independentes de plataforma. A criação de motores de inferência com configurações independentes potencializa as possibilidades de construção e testes do modelo, funcionalidade que só está presente na ferramenta FuzzyStudio. Outra exclusividade desta ferramenta em relação às outras três é a geração de códigos de programação para a construção de aplicações que utilizem o sistema modelado. Por fim, o FuzzyStudio é a única solução que possibilita o compartilhamento de projetos com outros usuários e o trabalho colaborativo em tempo real.

Com base na aplicação da ferramenta em sala de aula e a coleta de informações com a pesquisa qualitativa, conclui-se que o FuzzyStudio apresenta um conjunto de funcionalidades suficiente para a aplicação da ferramenta no desenvolvimento de sistemas fuzzy, bem como em ambiente acadêmico. É um software fácil de usar e simples na modelagem de sistemas difusos.

A solução apresentada possui diferenciais importantes em relação às demais ferramentas existentes e permite ao usuário uma série de funcionalidades inovadoras como o trabalho colaborativo e a geração de artefatos. O fato da ferramenta seguir o padrão FCL de notação de sistemas difusos faz com que os sistemas gerados por ela sejam facilmente comercializados com outros dispositivos e softwares que atendam à norma IEC 1131-7.

## 4 CONCLUSÕES

O presente trabalho teve como objetivo o desenvolvimento de uma ferramenta para a modelagem e simulação de sistemas fuzzy. Através das experimentações e comparações com os trabalhos correlatos, conclui-se que o objetivo foi alcançado.

Haviam três objetivos específicos que também foram atingidos. O FuzzyStudio é um software executado em ambiente Web, o que facilita o acesso à ferramenta, uma vez que não é dependente de plataforma. O FuzzyStudio pode ser utilizado a partir de qualquer sistema operacional. Também não é necessária a instalação de nenhum software adicional no cliente.

Juntamente com as funcionalidades de desenvolvimento e construção dos componentes de um sistema difuso, o FuzzyStudio permite a execução e simulação do modelo criado. Através do simulador, o usuário informa os valores de entrada e o sistema infere as saídas. Também são apresentados ao usuário os gráficos referente à simulação realizada. Estes aspectos permitem ao desenvolvedor testar seu sistema fuzzy, realizando possíveis ajustes para que o mesmo responda à realidade.

O FuzzyStudio possui a funcionalidade de geração de códigos referentes ao modelo criado. Uma vez desenvolvido o sistema difuso, o usuário é capaz de exportá-lo para a linguagem FCL, padrão de notação IEEE de sistemas fuzzy. Com isso, o usuário poderá embarcar seu sistema fuzzy em um software ou controlador inteligente. Ao seguir o padrão de notação garante-se a portabilidade do modelo para distintas aplicações que seguem a norma, bem como a escalabilidade do sistema difuso para aplicações de variados portes. O FuzzyStudio também gera código de programação em Java do sistema criado. É gerada uma classe Java que executa o modelo desenvolvido, facilitando a tarefa de desenvolvimento de aplicações que utilizem lógica fuzzy.

A ferramenta possui uma interface de desenho, na qual é apresentado o sistema difuso criado de forma visual. Através do desenho, é possível visualizar a arquitetura do sistema, ou seja, quais seus componentes e como estão relacionados entre si. Este aspecto facilita o entendimento do sistema desenvolvido, bem como agiliza a construção do modelo. Outro diferencial do FuzzyStudio é a possibilidade de trabalho colaborativo, também avaliada nas fases de experimentação. Um usuário pode compartilhar seu projeto com outros usuários, os quais trabalharão juntos no sistema em tempo real. Este aspecto permite o trabalho em grupos, seja de acadêmicos ou profissionais.

Durante o desenvolvimento da ferramenta foram encontradas dificuldades que merecem destaque. Desenvolver a interface de desenho utilizando HTML5 Canvas foi uma tarefa complexa no seu início, devido à falta de conhecimento na tecnologia. O desenvolvimento das rotinas AJAX também foram uma tarefa custosa, sendo resolvida com a utilização da biblioteca DWR. O controle de usuários conectados ao projeto, bem como a atualização automática da interface em tempo real no trabalho em equipe também se caracterizaram como uma dificuldade. Estes problemas foram resolvidos controlando cada usuário conectado através de um controlador, utilizando o padrão de projetos *Singleton* para garantir sua exclusividade. A atualização da interface foi construída utilizando requisições AJAX, que buscam as atualizações no servidor para renderização. As dificuldades apresentadas foram superadas em sua totalidade, resultando em um ganho de conhecimento extra no desenvolvimento do trabalho.

Em suma, a ferramenta FuzzyStudio apresenta funcionalidades diferenciais com relação às soluções encontradas no mercado, preenchendo algumas lacunas importantes. Dentre elas, se destacam a facilidade de acesso à ferramenta, facilidade em utilização da mesma, geração de código FCL do modelo construído, geração de código Java e possibilidade de trabalho colaborativo. Essas características configuram o FuzzyStudio como uma ferramenta única e inovadora na modelagem de sistemas fuzzy. Com isso, o software pode ser utilizado em ambiente acadêmico, para práticas com lógica difusa, permitindo a realização de trabalhos e experimentações com o tema. Também é possível de aplicação em ambientes comerciais, no desenvolvimento de sistemas fuzzy para controladores industriais ou softwares inteligentes.

Através das experimentações realizadas em sala de aula, pode-se afirmar que a utilização do FuzzyStudio em meio acadêmico possibilita a prática de sistemas difusos, bem como a realização de trabalhos e exercícios. Estes fatores apontam que o FuzzyStudio facilita o ensino de lógica difusa em ambiente acadêmico, através da disponibilização dos recursos supracitados. Como na época da conclusão deste trabalho não foram realizadas avaliações de aprendizado com os alunos, não é possível concluir se houveram ganhos quanto à aquisição dos seus conhecimentos.



#### 4.1 TRABALHOS FUTUROS

O sistema aqui desenvolvido pode ser ampliado, de modo a fornecer algumas funcionalidades adicionais e tornar a ferramenta mais completa. Dentre essas possibilidades, se destacam as seguintes extensões:

- a) Implementar outras funções de pertinência, como gaussiana por exemplo;
- b) Permitir a construção de sistemas fuzzy seguindo outros modelos como o de Takagi-Sugeno;
- c) Permitir a fuzzificação por tabela em memória;
- d) Permitir a execução de múltiplos casos de simulação ao mesmo tempo;
- e) Desenvolver rotinas de depuração da simulação, como pausar, acompanhar os valores de inferência e executar passo a passo;
- f) Permitir ao usuário a criação de funções próprias na fuzzificação e defuzzificação;
- g) Possibilitar a atribuição de pesos para as regras;
- h) Implementar o desenho da superfície gráfica para as variáveis do sistema;
- i) Importar um sistema fuzzy a partir do FCL;
- j) Melhorar a aparência e usabilidade na manutenção dos componentes;
- k) Permitir o encadeamento de antecedentes de uma regra com múltiplos conectores.

## REFERÊNCIAS

ALEXANDRE, A. B. **Protótipo de um Sistema Especialista Utilizando a Ferramenta Expert Sinta Shell para Auxílio no Setor de Suporte de uma Software House**. Blumenau: FURB, 2000. Trabalho de Conclusão de Curso, Bacharelado em Ciências da Computação.

ANDRADE et al. **Uso da morfologia matemática fuzzy na contagem esporos de fungos micorrízicos**. II Congresso Brasileiro de Sistemas “Fuzzy”. Natal, 2012.

BAUER, C.; KING, G. **Java Persistence com Hibernate**. India: Dreamtech Press, 2007.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

BURKE, B.; MONSON-HAEFEL, R. **Enterprise JavaBeans 3.0**. New Jersey: Prentice Hall, 2009.

CARMISINI, A.; VAHLDICK, A. Comparativo entre Frameworks de JavaServer Faces: Apache Tobago, PrimeFaces e RichFaces. **Revista Eletrônica do Alto Vale do Itajaí**. Ibirama, p. 10 – 18, 2012.

CINGOLANI, P.; ALCALÁ-FDEZ, J. **jFuzzyLogic: A Robust and Flexible Fuzzy-Logic Inference System Language Implementation**. **WCCI 2012 IEEE World Congress on Computational Intelligence**. Brisbane, p. 1090 – 1097, 2012.

COPPIN, B. **Inteligência Artificial**. Rio de Janeiro: LTC, 2010.

DEVSTATES. **Open source reviews by real users**. 2013. Disponível em: <<http://devrates.com/stats/index>>. Acesso em: 22 mai. 2013.

DIRECT Web Remoting. Versão 3.1.rc1, 2012. Disponível em: <[www.directwebremoting.org/dwr/index.html](http://www.directwebremoting.org/dwr/index.html)>. Acesso em: 15 mai. 2013.

EMBRAPA. **Sistema FuzzyGen**. Campinas: Embrapa Informática Agropecuária, 2009. Disponível em: <<http://www.infoteca.cnptia.embrapa.br/handle/doc/663745>>. Acesso em: 05 abr. 2013.

ESPINOSA, J.; VANDEWALLE, J.; WERTZ, V. **Fuzzy Logic, Identification and Predictive Control**. London: Springer-Verlag, 2005.

FOWLER, M. **UML Essencial**: Um breve guia para a linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2005.

GABRIEL FILHO et al. **Aplicação da Lógica Fuzzy para Avaliação da Eficiência e Racionalidade de Usinas Sucroalcooleiras**. II Congresso Brasileiro de Sistemas “Fuzzy”. Natal. 2012.

GALCERAN et al. **Use of IEC1131 programming in virtual laboratory**. Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on. Antibes-Juan les Pins, p. 645 – 649, 2001.

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

GILAT, A. **MATLAB com Aplicações em Engenharia**. Porto Alegre: Artmed, 2005.

GRANNELL, C.; SUMNER, V.; SYNODINOS, D. **The Essencial Guide to HTML5 and CSS3 Web Design**. New York: Apress, 2012.

HEINZLE, R. **Protótipo de uma ferramenta para criação de sistemas especialistas baseados em regras de produção**. Florianópolis : UFSC, 1995. Dissertação de Mestrado, Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

HOLZNER, S. **Ajax Bible**. Indianapolis, Indiana: Wiley Publishing, 2007.

IEC. **IEC 1131 – Programmable Controllers: Part 7 – Fuzzy Control Programming**. Committee Draft CD 1.0, 1997.

JACOBI, J.; FALLOWS, J. R. **Pro JSF and Ajax**: Building Rich Internet Components. New York: Apress, 2006.

JAIN, L. C.; MARTIN, N. M. **Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications**. Adelaide: CRC Press LLC, 1998.

JANTZEN, J. **Foundations of Fuzzy Control**. Chichester: John Wiley & Sons, 2007.

JFUZZYLOGIC. Versão 2.1a, 2013. Disponível em:  
<<http://jfuzzylogic.sourceforge.net/html/index.html>>. Acesso em: 10 jun. 2013.

KEITH, M.; SCHINCARIOL, M. **Pro JPA 2: Mastering the Java Persistence API**. New York: Apress, 2009.

KLIR, G. J.; YUAN, B. **Fuzzy Sets and Fuzzy Logic: Theory and Applications**. New Jersey: Prentice Hall, 1995.

KOVACIC, Z.; BOGDAN, S. **Fuzzy Controller Design: Theory and Applications**. Boca Raton: Taylor & Francis Group, 2006.

LAWSON, B.; SHARP, R. **Introducing HTML 5**. Berkeley: New Riders, 2011.

LIMA, H. P.; MASSHURÁ, S. M. F. S. **Sistema FuzzyGen: manual do usuário. ISSN 1677-9274**. Campinas : Embrapa Informática Agropecuária, 2009.

LINWOOD, J.; MINTER, D. **Beginning Hibernate: An Introduction to persistence using Hibernate 3.5**. New York: Apress, 2010.

LUBBERS, P.; ALBERS, B.; SALIM, F. **Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development**. New York: Apress, 2010.

LUCKOW, D. H; MELO, A. A. **Programação Java para a Web**. São Paulo: Novatec, 2010.

LUGER, G. F. **Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos**. 4 ed. Porto Alegre: Bookman, 2004.

MATHWORKS. **Fuzzy Logic Toolbox: User's Guide**. Natick: The MathWorks, 2012.

\_\_\_\_\_. **Matlab Fuzzy Toolbox**. 2013. Disponível em:  
<<http://www.mathworks.com/products/fuzzy-logic/index.html>>. Acesso em: 19 mar. 2013.

MECENAS, I.; OLIVEIRA, V. **Banco de Dados: do modelo conceitual à implementação física**. Rio de Janeiro: Alta Books, 2005.

MYERS, M. D. **Qualitative Research in Information Systems**. 1997. Disponível em:  
<<http://www.qual.auckland.ac.nz/>>. Acesso em: 22 de nov. 2011.

OLIVEIRA JUNIOR, H. A. et al. **Inteligência Computacional Aplicada à Administração, Economia e Engenharia em Matlab**. São Paulo: Thomson Learning, 2007.

OLIVEIRA NETTO, A. A.; MELO, C. **Metodologia da Pesquisa Científica: Guia Prático para a Apresentação de Trabalhos Acadêmicos**. 3 ed. Florianópolis: Visual Books, 2008.

PASSINO, K. M.; YURKOVICH, S. **Fuzzy Control**. Menlo Park: Addison-Wesley Longman, 1998.

PAULA e SOUSA, J. N. **Aplicação de Lógica Fuzzy em Sistemas de Controle de Tráfego Metropolitano em Rodovias Dotadas de Faixas Exclusivas para Ônibus**. Rio de Janeiro: UFRJ, 2005. Tese de Doutorado, Universidade Federal do Rio de Janeiro, Programa de Pós-Graduação de Engenharia.

POSSELT, E. L. **InFuzzy: Ferramenta para Desenvolvimento de Aplicações de Sistemas Difusos**. 2011. Disponível em: <<http://btd.unisc.br/Dissertacoes/EdersonPosselt.pdf>>. Acesso em: 25 out. 2011.

POSSELT, E. L.; FROZZA, R.; MOLZ, R. F. **Software InFuzzy 2011**. Programa de Mestrado em Sistemas e Processos Industriais PPGSPI, UNISC, 2011. Disponível em: <<http://www.unisc.br/ppgsapi>>. Acesso em: 25 mar. 2013.

PRESSMAN, R. S. **Engenharia de Software: Uma abordagem profissional**. 7 ed. New York: Bookman, 2011.

PRIMEFACES. **User Guide**. Versão 3.5. Optimus Prime: Prime Faces, 2013. Disponível em: <[http://primefaces.googlecode.com/files/indexed\\_primefaces\\_users\\_guide\\_3\\_5.pdf](http://primefaces.googlecode.com/files/indexed_primefaces_users_guide_3_5.pdf)>. Acesso em: 15 mai. 2013.

REZENDE, S. O. **Sistemas inteligentes: fundamentos e aplicações**. Barueri: Manole, 2005.

REZNIK, L. **Fuzzy Controllers**. Melbourne: Newnes, 1997.

ROSS, T. J. **Fuzzy Logic With Engineering Applications**. 2 ed. Chichester: John Wiley & Sons, 2004.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. New Jersey: Prentice Hall, 1995.

SAM-BODDEN, B. **Beginning POJOs: From Novice to Profesional**. New York: Apress, 2006.

SAMPAIO, C. **Java Enterprise Edition 6: Desenvolvendo Aplicações Corporativas**. Rio de Janeiro: Brasport, 2011.

SIMÕES, M. G.; SHAW I. S. **Controle e Modelagem Fuzzy**. 2 ed. São Paulo: Blucher, 2007.

SOMMERVILLE, I. **Engenharia de Software**. 8 ed. São Paulo: Addison Wesley, 2003.

STRAUB, D.; GEFEN, D; BOUDREAU, M. C. **Quantitative, Positivist Research Methods Website**. 2005. Disponível em: <<http://dstraub.cis.gsu.edu:88/quant/default.asp>>. Acesso em: 18 nov. 2011.

TANIMOTO, S. L. **The Elements of Artificial Intelligence: An Introduction Using LISP**. Rockville: Computer Science Press, 1987.

TROCHIM, W. M. K. **Research Methods Knowledge Base**. Disponível em: <<http://www.socialresearchmethods.net/kb/index.php>>. Acesso em: 18 nov. 2011.

WAINER, J., **Métodos de pesquisa quantitativa e qualitativa para a ciência da computação**. Rio de Janeiro: Editora PUC-Rio, 2007.

WEBER, L.; KLEIN, P. A. R. **Aplicação da Lógica Fuzzy em Software e Hardware**. Canoas: Ulbra, 2003.

ZADEH, L. A., **Fuzzy Sets**. 1965. Disponível em:  
<<http://www-bisc.cs.berkeley.edu/Zadeh-1965.pdf>>. Acesso em: 15 ago. 2011.

**APÊNDICE A** – Classe Java gerada para o estudo de caso de Gabriel Filho et al. (2012)

```

import net.sourceforge.jFuzzyLogic.FIS;
import net.sourceforge.jFuzzyLogic.FunctionBlock;
import org.antlr.runtime.RecognitionException;

public class FuzzyStudioFCL {
    private static String fcl;
    public static void main(String[] args) throws RecognitionException {
        fcl = "FUNCTION_BLOCK mamdani \n"+
            "\n"+
            "VAR_INPUT \n"+
            "  habilidade : REAL;\n"+
            "  conhecimento : REAL;\n"+
            "  atitude : REAL;\n"+
            "END_VAR\n"+
            "\n"+
            "VAR_OUTPUT \n"+
            "  colheitadeira : REAL;\n"+
            "END_VAR\n"+
            "FUZZIFY habilidade \n"+
            "  TERM Baixa := (0.0, 1) (5.0, 0); \n"+
            "  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alta := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "FUZZIFY conhecimento \n"+
            "  TERM Baixo := (0.0, 1) (5.0, 0); \n"+
            "  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alto := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "FUZZIFY atitude \n"+
            "  TERM Baixa := (0.0, 1) (5.0, 0); \n"+
            "  TERM Media := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alta := (5.0, 0) (10.0, 1); \n"+
            "END_FUZZIFY\n"+
            "\n"+
            "DEFUZZIFY colheitadeira \n"+
            "  TERM Baixo := (0.0, 1) (5.0, 0); \n"+
            "  TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0); \n"+
            "  TERM Alto := (5.0, 0) (10.0, 1); \n"+
            "  METHOD : COG; \n"+
            "  DEFAULT := 0; \n"+
            "END_DEFUZZIFY\n"+
            "\n"+
            "RULEBLOCK blocoderegra\n"+
            "\n"+
            "  ACCU : MAX; \n"+
            "  AND : MIN; \n"+
            "  ACT : MIN; \n"+
            "  RULE 1 : IF habilidade IS Baixa AND conhecimento IS
Baixo AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
            "  RULE 2 : IF habilidade IS Baixa AND conhecimento IS
Baixo AND atitude IS Media THEN colheitadeira IS Medio;\n"+

```



```

"    RULE 3 : IF habilidade IS Baixa AND conhecimento IS
Baixo AND atitude IS Alta THEN colheitadeira IS Medio;\n"+
"    RULE 4 : IF habilidade IS Baixa AND conhecimento IS
Medio AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 5 : IF habilidade IS Baixa AND conhecimento IS
Medio AND atitude IS Media THEN colheitadeira IS Baixo;\n"+
"    RULE 6 : IF habilidade IS Baixa AND conhecimento IS
Medio AND atitude IS Alta THEN colheitadeira IS Medio;\n"+
"    RULE 7 : IF habilidade IS Baixa AND conhecimento IS
Alto AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 8 : IF habilidade IS Baixa AND conhecimento IS
Alto AND atitude IS Media THEN colheitadeira IS Baixo;\n"+
"    RULE 9 : IF habilidade IS Baixa AND conhecimento IS
Alto AND atitude IS Alta THEN colheitadeira IS Alto;\n"+
"    RULE 10 : IF habilidade IS Media AND conhecimento IS
Baixo AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 11 : IF habilidade IS Media AND conhecimento IS
Baixo AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 12 : IF habilidade IS Media AND conhecimento IS
Baixo AND atitude IS Alta THEN colheitadeira IS Medio;\n"+
"    RULE 13 : IF habilidade IS Media AND conhecimento IS
Medio AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 14 : IF habilidade IS Media AND conhecimento IS
Medio AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 15 : IF habilidade IS Media AND conhecimento IS
Medio AND atitude IS Alta THEN colheitadeira IS Alto;\n"+
"    RULE 16 : IF habilidade IS Media AND conhecimento IS
Alto AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 17 : IF habilidade IS Media AND conhecimento IS
Alto AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 18 : IF habilidade IS Media AND conhecimento IS
Alto AND atitude IS Alta THEN colheitadeira IS Alto;\n"+
"    RULE 19 : IF habilidade IS Alta AND conhecimento IS
Baixo AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 20 : IF habilidade IS Alta AND conhecimento IS
Baixo AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 21 : IF habilidade IS Alta AND conhecimento IS
Baixo AND atitude IS Alta THEN colheitadeira IS Medio;\n"+
"    RULE 22 : IF habilidade IS Alta AND conhecimento IS
Medio AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 23 : IF habilidade IS Alta AND conhecimento IS
Medio AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 24 : IF habilidade IS Alta AND conhecimento IS
Medio AND atitude IS Alta THEN colheitadeira IS Alto;\n"+
"    RULE 25 : IF habilidade IS Alta AND conhecimento IS
Alto AND atitude IS Baixa THEN colheitadeira IS Baixo;\n"+
"    RULE 26 : IF habilidade IS Alta AND conhecimento IS
Alto AND atitude IS Media THEN colheitadeira IS Medio;\n"+
"    RULE 27 : IF habilidade IS Alta AND conhecimento IS
Alto AND atitude IS Alta THEN colheitadeira IS Alto;\n"+
"END_RULEBLOCK\n"+
"\n"+
"END_FUNCTION_BLOCK\n";
try{
    FIS fis = FIS.createFromString(fcl, true);
    FunctionBlock functionBlock = fis.getFunctionBlock("mamdani");
    functionBlock.setVariable("habilidade", Double.parseDouble(
        javax.swing.JOptionPane.showInputDialog(

```

```
        "Informe valor para habilidade"));
        functionBlock.setVariable("conhecimento", Double.parseDouble(
            javax.swing.JOptionPane.showInputDialog(
                "Informe valor para conhecimento")));
        functionBlock.setVariable("atitude", Double.parseDouble(
            javax.swing.JOptionPane.showInputDialog(
                "Informe valor para atitude")));

        functionBlock.evaluate();

        functionBlock.getVariable("habilidade").chart(true);
        functionBlock.getVariable("conhecimento").chart(true);
        functionBlock.getVariable("atitude").chart(true);

        functionBlock.getVariable("colheitadeira").chart(true);

        functionBlock.getVariable("colheitadeira").chartDefuzzifier(true);
    } catch(Exception e){
    }
}

public String getFcl(){
    return this.fcl;
}
}
```

**APÊNDICE B** – Código FCL gerado para o estudo de caso de Gabriel Filho et al. (2012)

```

FUNCTION_BLOCK mamdani

VAR_INPUT
    habilidade : REAL;
    conhecimento : REAL;
    atitude : REAL;
END_VAR

VAR_OUTPUT
    colheitadeira : REAL;
END_VAR

FUZZIFY habilidade
    TERM Baixa := (0.0, 1) (5.0, 0);
    TERM Media := (0.0, 0) (5.0, 1) (10.0, 0);
    TERM Alta := (5.0, 0) (10.0, 1);
END_FUZZIFY

FUZZIFY conhecimento
    TERM Baixo := (0.0, 1) (5.0, 0);
    TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0);
    TERM Alto := (5.0, 0) (10.0, 1);
END_FUZZIFY

FUZZIFY atitude
    TERM Baixa := (0.0, 1) (5.0, 0);
    TERM Media := (0.0, 0) (5.0, 1) (10.0, 0);
    TERM Alta := (5.0, 0) (10.0, 1);
END_FUZZIFY

DEFUZZIFY colheitadeira
    TERM Baixo := (0.0, 1) (5.0, 0);
    TERM Medio := (0.0, 0) (5.0, 1) (10.0, 0);
    TERM Alto := (5.0, 0) (10.0, 1);
    METHOD : COG;
    DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK blocoderegra

    ACCU : MAX;
    AND : MIN;
    ACT : MIN;

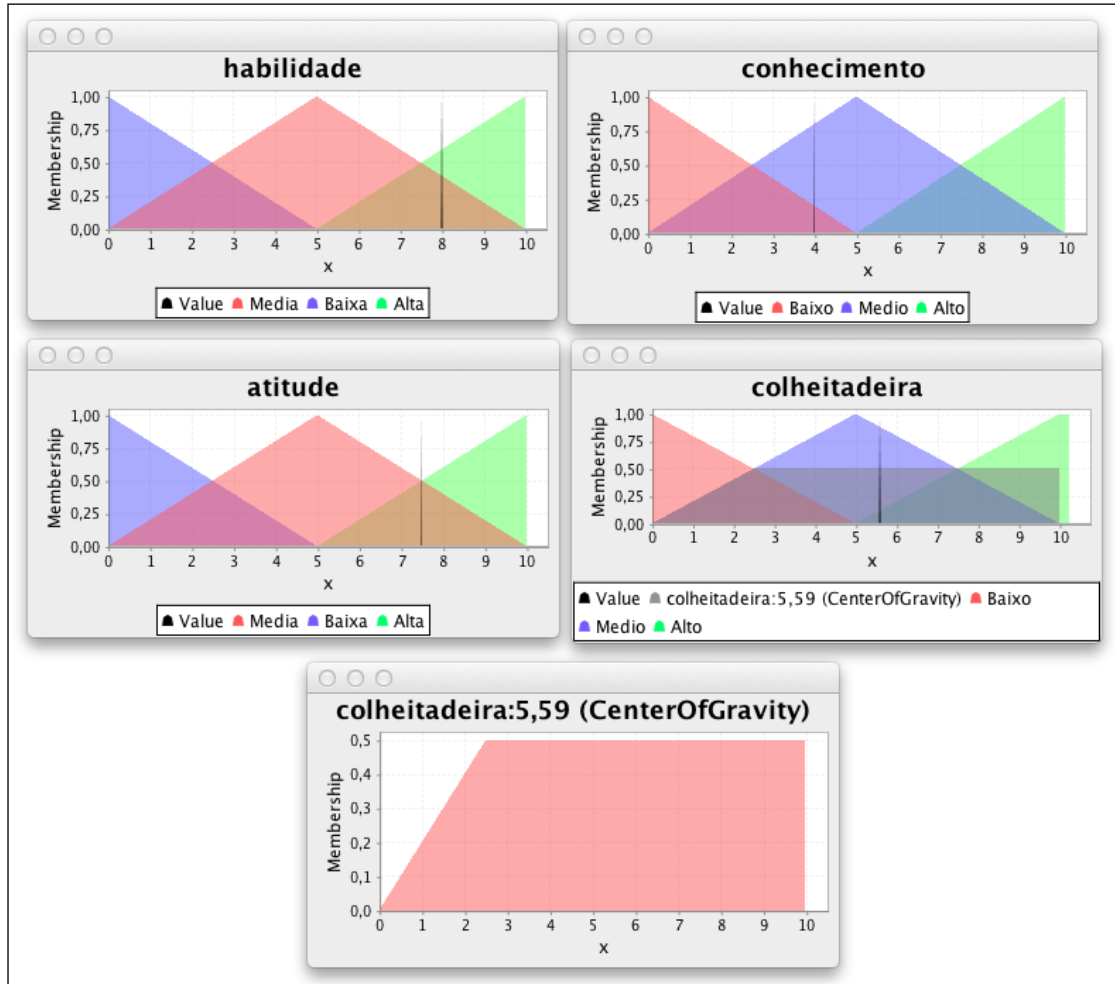
    RULE 1 : IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
    RULE 2 : IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS
        Media THEN colheitadeira IS Medio;
    RULE 3 : IF habilidade IS Baixa AND conhecimento IS Baixo AND atitude IS Alta
        THEN colheitadeira IS Medio;
    RULE 4 : IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
    RULE 5 : IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS
        Media THEN colheitadeira IS Baixo;

```

```
RULE 6 : IF habilidade IS Baixa AND conhecimento IS Medio AND atitude IS Alta
        THEN colheitadeira IS Medio;
RULE 7 : IF habilidade IS Baixa AND conhecimento IS Alto AND atitude IS Baixa
        THEN colheitadeira IS Baixo;
RULE 8 : IF habilidade IS Baixa AND conhecimento IS Alto AND atitude IS Media
        THEN colheitadeira IS Baixo;
RULE 9 : IF habilidade IS Baixa AND conhecimento IS Alto AND atitude IS Alta
        THEN colheitadeira IS Alto;
RULE 10 :IF habilidade IS Media AND conhecimento IS Baixo AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
RULE 11 :IF habilidade IS Media AND conhecimento IS Baixo AND atitude IS
        Media THEN colheitadeira IS Medio;
RULE 12 : IF habilidade IS Media AND conhecimento IS Baixo AND atitude IS
        Alta THEN colheitadeira IS Medio;
RULE 13 :IF habilidade IS Media AND conhecimento IS Medio AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
RULE 14 :IF habilidade IS Media AND conhecimento IS Medio AND atitude IS
        Media THEN colheitadeira IS Medio;
RULE 15 : IF habilidade IS Media AND conhecimento IS Medio AND atitude IS
        Alta THEN colheitadeira IS Alto;
RULE 16 : IF habilidade IS Media AND conhecimento IS Alto AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
RULE 17 : IF habilidade IS Media AND conhecimento IS Alto AND atitude IS
        Media THEN colheitadeira IS Medio;
RULE 18 : IF habilidade IS Media AND conhecimento IS Alto AND atitude IS Alta
        THEN colheitadeira IS Alto;
RULE 19 : IF habilidade IS Alta AND conhecimento IS Baixo AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
RULE 20 : IF habilidade IS Alta AND conhecimento IS Baixo AND atitude IS
        Media THEN colheitadeira IS Medio;
RULE 21 : IF habilidade IS Alta AND conhecimento IS Baixo AND atitude IS Alta
        THEN colheitadeira IS Medio;
RULE 22 : IF habilidade IS Alta AND conhecimento IS Medio AND atitude IS
        Baixa THEN colheitadeira IS Baixo;
RULE 23 : IF habilidade IS Alta AND conhecimento IS Medio AND atitude IS
        Media THEN colheitadeira IS Medio;
RULE 24 : IF habilidade IS Alta AND conhecimento IS Medio AND atitude IS Alta
        THEN colheitadeira IS Alto;
RULE 25 : IF habilidade IS Alta AND conhecimento IS Alto AND atitude IS Baixa
        THEN colheitadeira IS Baixo;
RULE 26 : IF habilidade IS Alta AND conhecimento IS Alto AND atitude IS Media
        THEN colheitadeira IS Medio;
RULE 27 : IF habilidade IS Alta AND conhecimento IS Alto AND atitude IS Alta
        THEN colheitadeira IS Alto;
```

END\_FUNCTION\_BLOCK

APÊNDICE C – Janelas geradas pela execução da classe Java do Apêndice A



**APÊNDICE D – Questionário aplicado aos acadêmicos após a utilização da ferramenta  
FuzzyStudio**

**Questionário sobre a experiência de utilização da ferramenta FuzzyStudio na  
modelagem e simulação de sistemas difusos**

Este questionário faz parte da experimentação realizada como método de validação do sistema FuzzyStudio, desenvolvido como Trabalho de Conclusão do Curso de Bacharelado em Sistemas de Informação da Universidade do Estado de Santa Catarina. O objetivo deste instrumento é avaliar a efetividade na utilização dos recursos disponibilizados pela ferramenta.

Para responder ao questionário, marque uma das opções para cada ponto analisado, informando sua satisfação acerca da facilidade em utilizar tal recurso.

ATIVIDADE	AVALIAÇÃO			
1. Cadastrar-se no sistema	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
2. Autenticar-se no sistema	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
3. Criar um novo projeto	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
4. Compartilhar um projeto	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
5. Criar variáveis	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
6. Vincular termos linguísticos às variáveis	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
7. Criar regras	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
8. Criar motores de inferência	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
9. Vincular regras aos motores de inferência	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
10. Editar os componentes visuais	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
11. Executar o modelo criado	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
12. Visualizar os gráficos da execução	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
13. Visualizar o código FCL do modelo construído	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
14. Visualizar a classe Java para o modelo construído	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
15. Exportar o modelo para uma imagem	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito

**Obrigado pela sua colaboração.**

**APÊNDICE E – Resultado do questionário aplicado aos acadêmicos após a utilização da ferramenta FuzzyStudio**

Atividade	Quantidade de pessoas			
	Completamente satisfeito	Satisfeito	Insatisfeito	Completamente insatisfeito
1 Cadastrar-se no sistema	2	3	0	0
2 Autenticar-se no sistema	4	1	0	0
3 Criar um novo projeto	4	1	0	0
4 Compartilhar um projeto	3	2	0	0
5 Criar variáveis	1	4	0	0
6 Vincular termos linguísticos às variáveis	1	4	0	0
7 Criar regras	2	1	2	0
8 Criar motores de inferência	1	4	0	0
9 Vincular regras aos motores de inferência	0	5	0	0
10 Editar os componentes visuais	2	2	1	0
11 Executar o modelo criado	1	4	0	0
12 Visualizar os gráficos da execução	1	3	1	0
13 Visualizar o código FCL do modelo construído	3	2	0	0
14 Visualizar a classe Java para o modelo construído	3	2	0	0
15 Exportar o modelo para uma imagem	2	2	1	0

