

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

**1º TEN GABRIEL BOZZA
1º TEN MILENA MAYARA RUY**

**DESENVOLVIMENTO DE UM AMBIENTE PARA TESTES DE RACIOCÍNIO
EM UM RÁDIO COGNITIVO**

**RIO DE JANEIRO
2021**

**1º TEN GABRIEL BOZZA
1º TEN MILENA MAYARA RUY**

**DESENVOLVIMENTO DE UM AMBIENTE PARA TESTES DE RACIOCÍNIO
EM UM RÁDIO COGNITIVO**

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia da Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Orientadores: Profª. Maria Cláudia Reis Cavalcanti,
D.Sc.
Cel David Fernandes Cruz Moura, D.Sc.

Rio de Janeiro
2021

©2021

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade dos autores e dos orientadores.

Bozza, 1º Ten Gabriel; Mayara Ruy, 1º Ten Milena.

Desenvolvimento de um Ambiente para Testes de Raciocínio em um Rádio Cognitivo / 1º Ten Gabriel Bozza e 1º Ten Milena Mayara Ruy. – Rio de Janeiro, 2021.

90 f.

Orientadores: Profª. Maria Cláudia Reis Cavalcanti e Cel David Fernandes Cruz Moura.

Projeto de Final de Curso (graduação) – Instituto Militar de Engenharia, Engenharia da Computação, 2021.

1. RDS. 2. rádio cognitivo. 3. MDE. 4. modelos. 5. adaptabilidade. 6. tempo de execução. i. Reis Cavalcanti, Profª. Maria Cláudia (orient.) ii. Fernandes Cruz Moura, Cel David (orient.) iii. Título

**1º TEN GABRIEL BOZZA
1º TEN MILENA MAYARA RUY**

Desenvolvimento de um Ambiente para Testes de Raciocínio em um Rádio Cognitivo

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia da Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Orientadores: Prof^a. Maria Cláudia Reis Cavalcanti e Cel David Fernandes Cruz Moura.

Aprovado em Rio de Janeiro, 14 de outubro de 2021, pela seguinte banca examinadora:

Prof^a. Maria Cláudia Reis Cavalcanti - D.Sc. do IME - Presidente

Cel David Fernandes Cruz Moura - D.Sc. do CTEX

Prof. Luís Ferreira Pires - Ph.D. da Universidade de Twente

TC Marcelo José Camilo - D.Sc. do CTEX

Prof. Ricardo Choren Noya - D.Sc. do IME

Maj Marcus Albert Alves da Silva - M.Sc. do IME

Rio de Janeiro
2021

AGRADECIMENTOS

Aos nossos orientadores Maria Cláudia Cavalcante (Yoko), Maj Marcus Albert, Cel David Fernandes Cruz Moura e ao professor da Universidade de Twente Luís Ferreira Pires por nos guiarem neste trabalho e a todos que participaram, direta ou indiretamente do desenvolvimento deste trabalho de pesquisa, enriquecendo nosso aprendizado.

RESUMO

O presente trabalho está associado ao comportamento de dispositivos de comunicação no incessantemente mutável teatro de operações. Nesse ambiente tático, os militares operadores de equipamentos rádio se beneficiariam da agilidade proporcionada pelo apoio de sistemas de software que sejam capazes detectar o ambiente externo. Dessa maneira, é possível indicar em tempo de execução as mudanças na maneira com a qual seus rádios se comunicam, ou seja, o modo de operação destes, com a finalidade de evitar a ação de interferidores de forças inimigas e otimizar a comunicação.

Nesse contexto, os rádios cognitivos, os quais se baseiam nos conceitos do Rádio Definido por Software (RDS) e de inteligência da rede de aparelhos de telecomunicação, são uma solução que aumentaria a capacidade e efetividade operacional das tropas ao proporcionar adaptabilidade à rede de dispositivos. Neste trabalho foram estudados os conceitos de RDS e de rádio cognitivo, além de abordagens que viabilizem e facilitem sua implementação. A abordagem que mais se destaca por possibilitar flexibilidade de projeto e modificação tanto em tempo de planejamento quanto em tempo de operação é a engenharia dirigida a modelos (MDE), a qual é amplamente utilizada na indústria e é suportada por diversos softwares de código aberto.

O foco deste trabalho é apresentar uma estrutura que permita a adaptabilidade do modo de operação do rádio, de modo que este reaja a estímulos externos em tempo de execução / operação e determine como os aparelhos de comunicação da rede devem se comportar durante suas comunicações. Para alcançar esse objetivo e simplificar modificações futuras do sistema, inicialmente foi estudada a abordagem MDE para que então, partindo de modelos com níveis de abstração mais altos, o código final da aplicação fosse gerado, seguido de testes do sistema a partir de dados de entrada previamente determinados.

Palavras-chave: RDS. rádio cognitivo. MDE. modelos. adaptabilidade. tempo de execução.

ABSTRACT

The present work is associated with the behavior of communication devices in the incessantly changing theater of operations. In this tactical environment, military radio equipment operators would benefit from the agility provided by the support of software systems capable of detecting the external environment. Thus, it is possible to indicate, at runtime, changes in the way radios communicate, that is, their operating mode, in order to avoid the action of interfering enemy forces and to optimize communication.

In this context, cognitive radios, which are based on the concepts of Software Defined Radio (SDR) and intelligence of the telecommunication device network, are a solution that would increase the troops' operational capacity and effectiveness by providing adaptability to the device network. In this work, the concepts of SDR and cognitive radio were studied, as well as approaches that enable and facilitate their implementation. The approach that stands out the most for allowing design flexibility and modification both in planning time and in operation time is Model-Driven Engineering (MDE), which is widely used in the industry and is supported by several open source software.

The focus of this work is to present a structure that allows the adaptability of the radio's mode of operation, so that it reacts to external stimuli at run / operation time and determines how the network communication devices should behave during their communications. To achieve this objective and simplify future system modifications, the MDE approach was initially studied so that, starting from models of higher levels of abstraction, the final application code could be generated, followed by system tests from previously determined input data.

Keywords: SDR. cognitive radio. MDE. models. adaptability. runtime.

LISTA DE ILUSTRAÇÕES

Figura 1 – Esquema de um RDS ideal (1)	21
Figura 2 – Estrutura básica real de um RDS (2)	21
Figura 3 – Comparativo entre as etapas de aprendizado natural e artificial (3) . .	24
Figura 4 – Comparaçao entre RDS e rádio cognitivo (4)	25
Figura 5 – Ciclo cognitivo simplificado	26
Figura 6 – Classificação das operações militares	27
Figura 7 – Níveis da arquitetura MOF (5)	30
Figura 8 – Visao geral da abordagem proposta no artigo (6)	34
Figura 9 – Diagrama de sequênciadas comunicações em operações (aspectos técnicos)	38
Figura 10 – Diagrama de sequênciadas comunicações em operações (aspectos táticos)	40
Figura 11 – Conjunto de regras hipotético para testes da arquitetura	41
Figura 12 – Descrição das regras da Figura 11	41
Figura 13 – Visão geral do desenvolvimento da solução proposta	44
Figura 14 – Etapa de testes da solução proposta	45
Figura 15 – Cenário pretendido da solução proposta	47
Figura 16 – Metamodelo simplificado de um rádio cognitivo em um ambiente de operação	49
Figura 17 – Metamodelo que descreve a formação de regras de um rádio cognitivo .	51
Figura 18 – Validação da instanciação das regras pela IDE do Eclipse	53
Figura 19 – Geração de código através do software Acceleo (7)	54
Figura 20 – Trecho de código do <i>template</i> desenvolvido	54
Figura 21 – Utilização da DSL desenvolvida	56
Figura 22 – Visão geral da arquitetura e das etapas do processo	59
Figura 23 – Funcionamento da janela deslizante	60
Figura 24 – Sequência de logs e modo de operação resultante retirados de (a) Anexo I - Saída do sistema original (b) Anexo J - Saída do sistema com logs alterados	62
Figura 25 – Conjunto de regras alterado	63
Figura 26 – Sequência de logs e modo de operação resultante retirados de (a) Anexo I - Saída do sistema original (b) Anexo K - Saída do sistema com novo conjunto de regras	64
Figura 27 – Sequência de logs e modo de operação resultante retirados do Anexo L - Saída do sistema sem regras determinadas	65
Figura 28 – (a) Sequência de logs com erros; (b) Sequência de logs e modo de operação resultantes retirados do Anexo M - Saída do sistema com Log inicial com erros	66

Figura 29 – Interface gráfica para a instanciação do metamodelo 74

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

RDS	Rádio Definido por <i>Software</i>
DAC	<i>Digital-to-Analog Converter</i>
ADC	<i>Analog-to-Digital Converter</i>
DDC	<i>Digital Down Converter</i>
DUC	<i>Digital Up Converter</i>
HF	<i>High Frequency</i>
VHF	<i>Very High Frequency</i>
UHF	<i>Ultra High Frequency</i>
CTEX	Centro Tecnológico do Exército
SCA	<i>Software Communications Architecture</i>
MDE	<i>Model Driven Engineering</i>
MDA	<i>Model Driven Architecture</i>
MOF	<i>Meta-Object Facility</i>
OMG	<i>Object Management Group</i>
M2M	<i>Model to Model</i>
M2T	<i>Model to Text</i>
MOFM2T	<i>Meta-Object Facility Model to Text</i>
CIM	<i>Computation Independent Model</i>
PIM	<i>Platform Independent Model</i>
PSM	<i>Platform Specific Model</i>
DSL	<i>Domain Specific Language</i>
GPL	<i>General Purpose Language</i>
OCL	<i>Object Constraint Language</i>
UML	<i>Unified Modeling Language</i>

EMP	<i>Eclipse Modeling Project</i>
EMF	<i>Eclipse Modeling Framework</i>
IDE	<i>Integrated Development Environment</i>
MTL	<i>Model to Text Language</i>
SML	<i>Situation Modeling Language</i>
SA	<i>Situation Awareness</i>
VC	<i>Verificação corrente</i>
VF	<i>Verificação final</i>

SUMÁRIO

1	INTRODUÇÃO	15
1.1	MOTIVAÇÃO	15
1.2	OBJETIVO	16
1.3	JUSTIFICATIVA	16
1.4	METODOLOGIA	17
1.5	ESTRUTURA	17
2	CONTEXTUALIZAÇÃO	19
2.1	RÁDIO DEFINIDO POR SOFTWARE	19
2.1.1	INTRODUÇÃO E DEFINIÇÃO	19
2.1.2	CARACTERÍSTICA E ARQUITETURA	20
2.1.3	APLICAÇÕES DO RDS	22
2.1.4	RDS NO ÂMBITO MILITAR BRASILEIRO	22
2.2	RÁDIO COGNITIVO	23
2.2.1	DEFINIÇÃO	23
2.2.2	EVOLUÇÃO DO RDS PARA O RÁDIO COGNITIVO	24
2.2.3	CARACTERÍSTICAS E ARQUITETURA	25
2.3	CENÁRIOS OPERACIONAIS	27
2.4	MDE	29
2.5	DSL	31
3	TRABALHOS MOTIVADORES	33
3.1	<i>A MODEL-DRIVEN APPROACH TO SITUATIONS</i>	33
3.2	<i>TOWARDS ONTOLOGY-DRIVEN SITUATION-AWARE DISASTER MANAGEMENT</i>	34
3.3	COMPARATIVO COM O TRABALHO PROPOSTO	36
4	ABORDAGEM PROPOSTA	37
4.1	CENÁRIOS DE EMPREGO DE RÁDIOS EM OPERAÇÕES MILITARES	37
4.2	TECNOLOGIAS DISPONÍVEIS	42
4.2.1	EMF	42
4.2.1.1	ACCELEO	42
4.2.1.2	XTEXT	43
4.3	DESENVOLVIMENTO E TESTES	43
4.4	FERRAMENTAS UTILIZADAS	45
4.5	CENÁRIO PRETENDIDO DA SOLUÇÃO PROPOSTA	46

5	RESULTADOS OBTIDOS	48
5.1	METAMODELAGENS	48
5.1.1	RÁDIO COGNITIVO	48
5.1.2	FORMAÇÃO DAS REGRAS	50
5.2	GERAÇÃO DE CÓDIGO ATRAVÉS DO ACCELEO	53
5.2.1	TEMPLATE DE CÓDIGO	54
5.2.2	CÓDIGO JAVA GERADO	55
5.3	DESENVOLVIMENTO DE UMA DSL	56
5.3.1	UTILIZAÇÃO DO XTEXT	56
6	SIMULAÇÃO E TESTES	58
6.1	DESCRÍÇÃO DO AMBIENTE DE SIMULAÇÃO	58
6.2	ADAPTAÇÃO DO PROGRAMA SIMULADOR	59
6.3	TESTES	60
6.3.1	TESTE DE FUNCIONAMENTO DO SISTEMA	61
6.3.2	TESTE DE ALTERAÇÃO DAS INFORMAÇÕES RECEBIDAS PELO SISTEMA .	61
6.3.3	MUDANÇAS NO CONJUNTO DE REGRAS	62
6.3.4	TESTES LIMITE	64
7	COMPARAÇÃO COM OUTRAS FERRAMENTAS	67
8	CONCLUSÃO E PROPOSTAS PARA PROJETOS FUTUROS	68
REFERÊNCIAS		69
ANEXO A – EXEMPLO DE REGRA GERADA ATRAVÉS DA TRANS-		
FORMAÇÃO M2T		71
ANEXO B – ARQUIVO XMI GERADO PELA INSTANCIACÃO		
DO METAMODELO DE FORMAÇÃO DE REGRAS		72
ANEXO C – INTERFACE GRÁFICA DA IDE ECLIPSE PARA A		
INSTANCIACÃO DO METAMODELO DE FORMA-		
ÇÃO DE REGRAS		74
ANEXO D – MODELO DE CÓDIGO DESENVOLVIDO		75
ANEXO E – ARQUIVO XTEXT MODIFICADO PARA A GERA-		
ÇÃO DA DSL		79
ANEXO F – INSTANCIACÃO DE REGRAS ATRAVÉS DA DSL		
GERADA		83

ANEXO G – COMPARAÇÃO DOS MÉTODOS DE INSTANCIACÃO DE REGRAS	84
ANEXO H – RESULTADO DO TESTE - <i>HARD CODED</i>	85
ANEXO I – RESULTADO DO TESTE - MDE	86
ANEXO J – RESULTADO DO TESTE - LOGS ALTERADOS . . .	87
ANEXO K – RESULTADO DO TESTE - ADIÇÃO DE NOVA REGRA	88
ANEXO L – RESULTADO DO TESTE - NENHUMA REGRA PRESENTE	89
ANEXO M – RESULTADO DO TESTE - ERROS NOS LOGS . . .	90

1 INTRODUÇÃO

1.1 Motivação

No desenvolvimento de sistemas de *software*, a utilização de aparelhos de comunicação adaptáveis e inteligentes com base em dispositivos cognitivos se faz uma questão decisiva para os combatentes, os quais não podem expor suas localizações e tão pouco os objetivos e as ordens enviadas às tropas. Isso se dá especialmente num contexto em constante mudança como o teatro de operações, onde poucos segundos podem ser determinantes para a decisão de uma batalha, e onde o ambiente de combate está sempre em evolução, com cada vez mais disputas em terrenos urbanos e extremamente complexos, com muitas fontes de interferência eletromagnética.

O projeto de uma rede de dispositivos de rádios flexíveis é estratégica para o país em função de diversos fatores. Além do descrito no parágrafo anterior, sua importância se dá em função dos elevados custos de aquisição, treinamento da tropa para a utilização de novos aparelhos de comunicação e os recursos necessários durante todo o ciclo de vida dos rádios. Adicionalmente, destaca-se que um *software* nacional que proporciona mudanças de modos e frequência de operação em tempo de execução possibilita não só a manutenção da segurança e soberania nacional, mas também a evolução das táticas de operações da força terrestre.

Ao desenvolver o *software* de um dispositivo que cumpra com os requisitos que podem ser extraídos das necessidades da tropa explicitadas anteriormente, uma característica primordial deste é a capacidade de adaptação para que seu modo de operação reflita automaticamente as mudanças no teatro de operações. Para atingir esse objetivo devem ser analisados e determinados quais parâmetros são relevantes na tomada de decisão de mudanças no estado de algum rádio, e então determinar regras baseadas nesses dados para que o software dos aparelhos possa se adaptar às mudanças.

Nesse contexto militar, portanto, por causa da flexibilidade em operar numa ampla faixa de frequências, entre diversos padrões e monitorar o ambiente externo, identificando interferências de outros equipamentos, o rádio cognitivo se torna interessante como apetrecho militar, reconhecendo comunicação inimiga e sendo capaz de traçar alternativas para as mais diversas situações.

Com as finalidades propostas acima, o projeto do sistema que será utilizado pelos dispositivos será de extrema relevância e deve ser facilmente modificado, uma vez que os cenários de operação que determinam a composição das regras de funcionamento podem ser modificados substancialmente com o avanço de tecnologias e da expertise do inimigo

em burlar e tirar partido do sistema. Com base nisso, uma abordagem que se destaca para o desenvolvimento de um projeto de tal complexidade e mutabilidade é a engenharia dirigida por modelos (MDE), que se baseia na geração de modelos e suas transformações para que mudanças no projeto possam ser mais facilmente visualizadas e propagadas desde os níveis mais altos de abstração até o código executável que será gravado nos aparelhos da ponta final. Outra vantagem que uma abordagem pautada nos princípios da engenharia dirigida por modelos proporciona é uma menor dependência dos programadores para a realização de pequenas modificações em regras de operação, com um menor custo em homens hora para esses ajustes do sistema, e tornando-as mais rápidas de serem implementadas, características que devem ser almejadas em projetos dessa natureza.

1.2 Objetivo

O objetivo deste trabalho é desenvolver uma arquitetura que possibilita a adaptabilidade do modo de operação de um rádio através da adoção de técnicas de MDE e a realização de testes para a validação do seu funcionamento. Será fornecida uma entrada com uma sequência temporal de eventos associados a parâmetros relevantes do sistema (ambiente e plano de comunicações de uma operação militar) e terá como resultado o estado (modo de operação) em que um dado equipamento de comunicação deve estar, possibilitando dessa maneira uma reação do rádio em tempo de execução / operação. A abordagem a ser determinada também deve proporcionar flexibilidade em tempo de planejamento / configuração, possibilitando que mudanças de projeto possam ser implementadas com maior facilidade e menor dependência de pessoal especializado.

O presente trabalho não aborda as interfaces com um rádio real, sendo a contribuição principal deste a adoção de técnicas de MDE para a modelagem de regras que determinam como um rádio cognitivo deve se comportar dados os parâmetros percebidos. Dessa forma, a estrutura proposta será desenvolvida para ser compatível com o software desenvolvido pelo Maj Marcus Albert Alves da Silva, o qual simula os parâmetros percebidos pelo rádio.

Como objetivos deste projeto também podem-se citar o aprendizado a respeito de técnicas de MDE e como sua utilização pode ser benéfica em situações de leitura do ambiente externo e consequente resposta automática à mudanças no mesmo.

1.3 Justificativa

O problema de prover e simular cognição em uma rede de rádios definidos por *software* é um empecilho para a evolução de sistemas dessa natureza, pois, apesar de existirem soluções, estas são proprietárias ou de cunho estratégico para os poucos países que a detém.

Logo, a proposta do desenvolvimento de um ambiente que possibilita a adaptação do comportamento dos dispositivos de uma rede deste tipo que seja pautado nos princípios do MDE e que seja compatível com as vantagens técnicas apontadas na seção 1.1 se faz extremamente relevante, em especial no contexto atual, onde em poucos anos os avanços tecnológicos podem mudar completamente o processo decisório do sistema, exigindo uma alta adaptabilidade do projeto.

Dado o exposto, pode-se concluir que as nações detentoras de ferramentas com as características citadas anteriormente acabam obtendo uma grande vantagem competitiva em relação aos seus pares com ferramentas engessadas e mais dependentes de pessoal especializado.

1.4 Metodologia

O ponto de partida foi a leitura dos artigos “*A Model-Driven Approach to Situations*” (6) e “*Towards ontology-driven situation-aware disaster management*” (8), os quais abordam o desenvolvimento de aplicações de consciência situacional utilizando conceitos de MDE.

Em seguida, foi realizado um estudo acerca de conceitos sobre o RDS, a relação deste com princípios dos rádios cognitivos, além do estudo paralelo sobre a aplicação da abordagem MDE no desenvolvimento de soluções de cunho similar ao tema abordado. Então, foram analisadas e definidas as ferramentas que podem ser utilizadas em cada etapa do trabalho, seguido da modelagem inicial de como um dispositivo de comunicação percebe o ambiente para compreender melhor um cenário de operação e então modelar a formação de regras baseadas nos parâmetros percebidos.

Após as modelagens iniciais, foi realizado um estudo sobre a formação e determinação das regras, com a posterior transformação do modelo criado em código executável através de uma ferramenta de transformação modelo para texto (M2T) (conceitos explorados na seção 2.4). Por fim, foram realizados testes do sistema a partir de arquivos de dados brutos previamente gerados seguido da análise destes resultados.

Os passos adotados para a realização do projeto estão explicitados na seção 5 (abordagem proposta).

1.5 Estrutura

O objetivo deste trabalho é documentar todas as etapas da concepção à finalização do projeto, seguindo a estrutura descrita a seguir.

O capítulo 2 apresenta os conceitos básicos para o desenvolvimento do trabalho. A seção 2.1 contempla a definição, contextualização e caracterização do RDS. A seção 2.2

explana o conceito de rádio cognitivo e suas características. A seção 2.3 explica os cenários operacionais militares, a seção 2.4 apresenta as definições da abordagem MDE e a seção 2.5 apresenta a definição de linguagens específicas de domínio (DSLs).

O capítulo 3 apresenta trabalhos motivadores relacionados aos tópicos abordados no projeto, bem como as abordagens propostas para a resolução de questões de cunho similar às dificuldades inerentes ao RDS.

O capítulo 4 contempla uma proposta da abordagem que será adotada no desenvolvimento do projeto e a visão geral das etapas do trabalho, além de expor as tecnologias e ferramentas passíveis de serem utilizadas nas diferentes etapas do desenvolvimento do projeto.

O capítulo 5 apresenta os resultados obtidos neste projeto.

O capítulo 6 apresenta os testes realizados e as respectivas análises dos resultados.

O capítulo 7 contempla comparações com ferramentas e abordagens alternativas para o desenvolvimento do projeto.

Por fim, o capítulo 8 sintetiza o que foi obtido no presente trabalho e expõe propostas para projetos futuros.

2 CONTEXTUALIZAÇÃO

A seguir definimos e desenvolvemos os princípios de rádio definido por software, rádio cognitivo, engenharia baseada em modelos e arquitetura baseada em modelos, conceitos necessários para elaboração do projeto, uma vez que a ferramenta elaborada deve ser aplicada a um RDS com o objetivo de permitir que ele se torne um rádio cognitivo e isso foi feito por meio de técnicas de engenharia dirigida a modelos.

2.1 Rádio Definido por Software

2.1.1 Introdução e Definição

O Rádio Definido por *Software* (RDS) teve origem nas pesquisas militares, inicialmente nos Estados Unidos, na tentativa de fornecer rádios flexíveis de grande escala de forma barata e eficiente, garantindo interoperabilidade dos rádios militares de outras agências governamentais, ou seja, equipamentos capazes de se adaptarem a diferentes padrões operacionais (3).

Com essa finalidade, o RDS foi desenvolvido como uma evolução dos rádios tradicionais na qual alguma ou todas as funções (como modulação, demodulação e filtragens) são implementadas através de *software* ao invés de serem responsabilidades do *hardware* permitindo assim que seja reconfigurado a qualquer momento (dentro dos limites do sistema de *hardware* e do próprio *software* desenvolvido).

Essa capacidade de mudança e adição ou remoção de funcionalidades implica em sistemas de comunicação capazes de operar em mais de um modo com uma única configuração de *hardware* (9). Estes modos de operação podem ser caracterizados pela ausência de transmissão de informações, bem como por formas de onda que descrevem, segundo Galdino et al.(10),

"...mecanismos de segurança na transmissão de dados, codificação de fonte (voz, imagem e compressão de vídeo), codificação de canal, com mecanismos de retransmissão, bem como de detecção e correção de erros, técnicas de modulação e demodulação, equalização adaptativa, controle automático de ganho, sincronização, filtragem e controle de acesso ao meio, dentre outras funcionalidades."(10)

É importante lembrar que o rádio definido por *software* não deve ser confundido com os rádios baseados em *software* (já que praticamente todos os rádios utilizam *software*),

os quais necessitam também de mudanças no *hardware* para qualquer ajuste de *software*. No RDS todo o processamento do sinal deve ser feito através do *software* (2).

O RDS, portanto, se apresenta como um sistema de comunicação no qual não há necessidade de um *hardware* para a interpretação do sinal, já que o mesmo é processado, transmitido ou capturado por meio de um *software* (2).

Essa propriedade confere ao RDS alta flexibilidade, uma vez que problemas que são tratados através de mudanças de *hardware* em um rádio tradicional, passam a ser tratados em *software*. Por exemplo, para adicionar alguma funcionalidade em um rádio tradicional, é necessário adicionar um microcontrolador em uma entrada do hardware e, se essa entrada mudar, a configuração inteira do rádio muda, o que, além de trabalhoso, representa custos expressivos. Já no RDS, o trabalho fica abstruído, pois basta executar uma função de *software* embutida neste.

Resumindo, o objetivo principal desta tecnologia é transformar problemas relacionados ao *hardware* em problemas relacionados ao *software*, aumentando flexibilidade e diminuindo custo de produção.

2.1.2 Característica e Arquitetura

As três principais características apresentadas pelo RDS são a reconfigurabilidade, a flexibilidade e a modularidade. A primeira confere ao rádio a sua capacidade de alterar o funcionamento conforme a necessidade sem intervenção física e a flexibilidade é a qualidade de aceitar essas alterações sem mudanças na arquitetura do rádio. A modularidade, por sua vez, determina que as partes integrantes do sistema sejam executadas em módulos distintos, contribuindo também para a flexibilidade. (1).

Além disso, como as especificações para cada caso de utilização podem ser determinadas pelo usuário, é possível fazer o reuso de *hardware* para diferentes funções, barateando o custo para o fornecedor (3).

Idealmente, um rádio definido por software é representado por 3 blocos (Figura 1), um composto pelas antenas, o segundo pelos conversores digital-analógico (DAC) e analógico-digital (ADC) e o terceiro responsável pelo processamento do sinal (1).

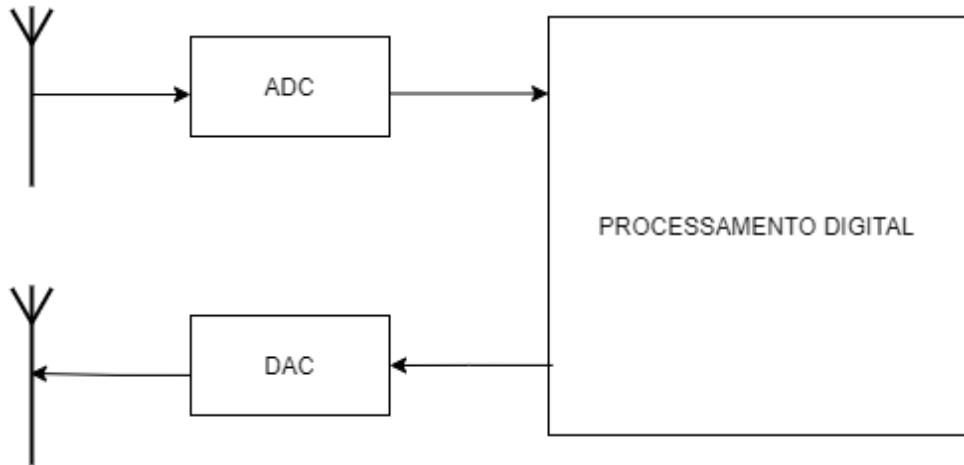


Figura 1 – Esquema de um RDS ideal (1)

O alto desempenho necessário para a execução de todo o processamento do *software* e os requisitos de taxa de amostragem, largura de banda e faixa dinâmica dos ADC's e DAC's não são totalmente realizáveis e, por isso, representam empecilhos ao modelo ideal do RDS. Esses problemas são comumente contornados por algumas alterações do sinal via *hardware* (2). Abaixo, a Figura 2 representa um esquema real de um RDS.

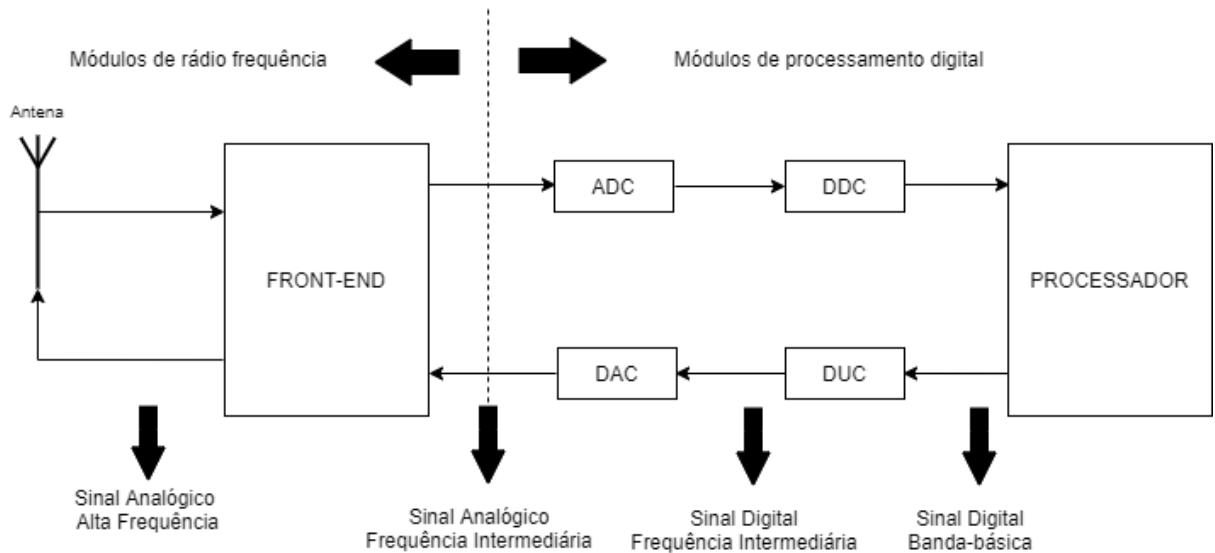


Figura 2 – Estrutura básica real de um RDS (2)

Acima, o *front-end* é responsável por cuidar do recebimento e transmissão das frequências de rádio, ele prepara o sinal para a conversão analógico-digital e/ou prepara o sinal para a transmissão após a conversão digital-analógico. Essa preparação é feita através de amplificação do sinal, controle de ganho, deslocamento para uma frequência intermediária ou uma frequência original e filtragem.

A Figura 2 também apresenta dois novos componentes quando comparado à Figura 1, o *Digital Down Converter* (DDC), que faz a re-amostragem do sinal digital de frequência intermediária para um sinal digital de banda base, e o *Digital Up Converter* (DUC) - que tem função inversa ao DDC. Ambos são utilizados para que o processamento não seja efetuado em velocidades muito altas.

Vale ressaltar que o rádio definido por software também apresenta desvantagens, como a tendência de ter um desempenho inferior quando comparado a sistemas implementados totalmente em circuitos integrados por ser baseado em *hardwares* genéricos (e não em hardwares otimizados para certas funções, o que resulta em um maior tempo para realizá-las), além de poder gastar mais energia do que a solução tradicional (3).

O desafio principal do desenvolvimento do RDS é justamente a implantação das funcionalidades no *software* e garantir a performance do mesmo, já que pode apresentar falhas no sistema e ser vulnerável a ataques de segurança caso não seja desenvolvido corretamente.

2.1.3 Aplicações do RDS

Apesar da origem militar, o rádio definido por *software* é cada vez mais utilizado no campo comercial, em mercados que necessitam de interoperabilidade (como na segurança pública), para reduzir custos para os operadores de telecomunicações e em rádios amadores.

Também encontramos aplicações no âmbito educacional, sendo utilizado na pesquisa experimental em Redes de Computadores e na prototipagem e implementação rápida de novas tecnologias de comunicação de forma mais barata, permitindo pesquisas dentro de laboratórios de universidades, não só em grandes empresas (3).

Além disso, o RDS pode, é claro, ser utilizado em seu campo de origem, o militar, que será abordado no próximo item.

2.1.4 RDS no Âmbito Militar Brasileiro

No Brasil, o projeto do RDS-Defesa se iniciou em 2012 e desde então segue sendo aprimorado, com o objetivo de promover a interoperabilidade entre as Forças Armadas do Brasil, desenvolvendo uma família de equipamentos rádio multibanda (operando nas faixas de HF, VHF e UHF) capazes de executar diversas formas de onda (descritas na seção 2.1.1) - que podem ser utilizados em veículos terrestres e navais, rádios portáteis e ultra portáteis - contribuindo para o fortalecimento da Base Industrial de Defesa do País. O CTEx é responsável pela gerência do projeto que conta com participações de Instituições de Ciência e Tecnologia das três Forças Armadas, além de empresas civis (11).

O CTEx utiliza uma infraestrutura de software aberta chamada SCA, que foi desenvolvida nos Estados Unidos, para suportar plataformas de diferentes capacidades,

melhorando a portabilidade das aplicações de rádio e fornecendo abstração do *hardware* para a forma de onda. Com o SCA, a modelagem é feita através de um diagrama de componentes e o software gera o código a ser utilizado na configuração do ambiente operacional e das aplicações de comunicações - as denominadas formas de onda (9).

O padrão SCA é uma arquitetura escalável que especifica mecanismos para criar, implantar, gerenciar e interconectar aplicações rádio baseadas em componentes (plataformas distribuídas). Dessa forma, permite aumentar consideravelmente a interoperabilidade de comunicação de sistemas rádio, além de reduzir o tempo de implantação e custos de desenvolvimento dos mesmos (10).

2.2 Rádio Cognitivo

2.2.1 Definição

Cognição é derivada da palavra latina *cognitione*, que significa “aquisição de conhecimento através da percepção”. Em sua tese de doutorado, Mitola(12) propôs o conceito de rádios cognitivos como rádios que adaptam sua operação com base em condições do ambiente, aprendendo com o comportamento passado e melhorando o seu funcionamento com o tempo, análogo ao que um humano faria (comparativo na Figura 3) (3). Esse equipamento buscaria o equilíbrio entre a confiabilidade de comunicação e o aumento do número de usuários no espectro, fundamentando-se na flexibilidade e adaptabilidade autônomas (13).

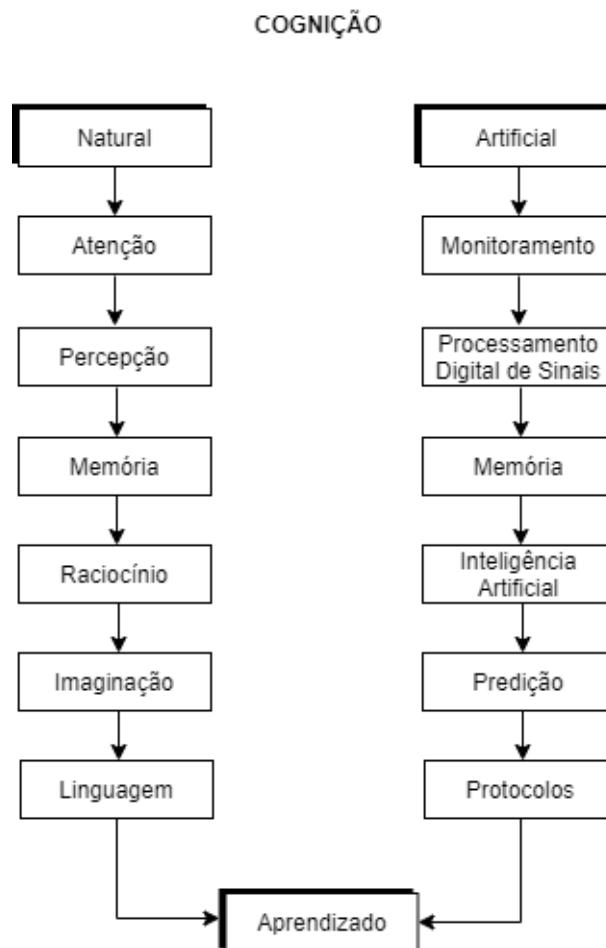


Figura 3 – Comparativo entre as etapas de aprendizado natural e artificial (3)

Inicialmente, o rádio cognitivo não recebeu muita atenção, por causa da sua alta necessidade de processamento para funcionar conforme o proposto. Ademais, esse tipo de equipamento também precisa ser capaz de identificar e “se comunicar” com diversas tecnologias, tais como diferentes tipos de onda, para que possa operar de forma eficaz em frequências com diferentes características (3).

Depois, com o crescente interesse, o desenvolvimento do rádio cognitivo determinou que o mesmo é uma integração de um raciocínio baseado em modelos com softwares de rádio não só programáveis, mas também treináveis (4), resultando em ganhos de eficiência e flexibilidade.

2.2.2 Evolução do RDS para o Rádio Cognitivo

A principal evolução do rádio cognitivo frente ao RDS é que o primeiro envolve mecanismos de inteligência artificial e aprendizado para identificar o meio e propor qual a configuração mais adequada para o mesmo, resultando em aplicações mais amplas (vide Figura 4).

Os rádios definidos por *software*, apesar de serem bastante flexíveis, possuem pouca inteligência computacional, além de pouco conhecimento sobre o próprio rádio e habilidade de se comunicar com outros equipamentos utilizando esse conhecimento. Portanto o objetivo da evolução para o rádio cognitivo é torná-lo “autoconsciente” (4). Já o rádio cognitivo é capaz de maior profundidade de conhecimento e rapidez na reconfiguração.

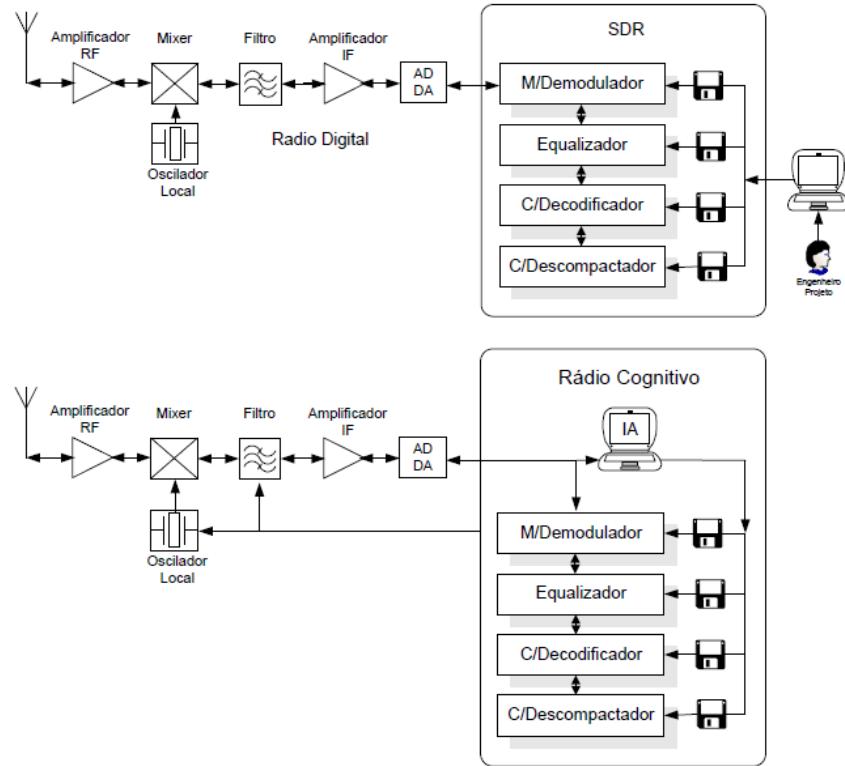


Figura 4 – Comparaçāo entre RDS e rádio cognitivo (4)

2.2.3 Características e arquitetura

O rádio cognitivo é pré-programado baseado em objetivos (para escolher a frequência, a largura de banda, a interface de comunicação com aplicações de comando e controle, bem como os protocolos de comunicação a serem adotados na forma de onda instanciada), possui conhecimento não só dele próprio, mas também do ambiente, sendo capaz de planejar, negociar, aprender e adaptar planos e protocolos para funcionar de maneira responsiva.

Os desenvolvedores de aplicativos continuarão a se especializar, dadas as diferenças no comportamento dos canais rádio e das aplicações. A mudança será para o operador rádio, que poderá usufruir de um melhor desempenho de um sistema dinâmico e adaptativo que utiliza as técnicas de aprendizado de máquina indicadas. O rádio cognitivo fará as vezes de um agente já projetado especificamente para trabalhar com essa diversidade de informações, utilizando técnicas de aprendizado de máquina que aumentam a robustez para ambientes externos que mudam rapidamente (4).

O que torna o rádio cognitivo capaz de interagir com o ambiente externo é o ciclo de cognição (Figura 5). O equipamento observa continuamente o ambiente externo até receber dele um estímulo, que é processado e tem uma prioridade associada a ele que permite que o rádio se oriente e possa criar opções de planos a serem seguidos. Depois da avaliação das alternativas geradas, há uma decisão de qual será a escolhida e os recursos do rádio são alocados para iniciar o processo na fase de ação. Essa última é uma resposta ao ambiente externo que iniciou o ciclo (12).

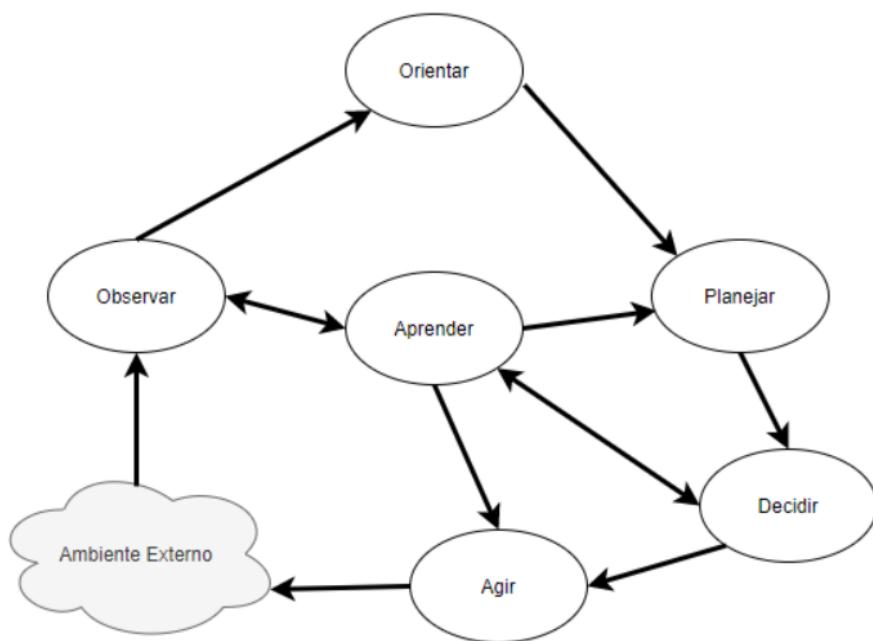


Figura 5 – Ciclo cognitivo simplificado

Como a Figura 5 representa, aprendizado de máquina pode ser adicionado ao ciclo cognitivo caracterizando épocas ativas - quando um especialista pode atuar no equipamento e também um estímulo pode iniciar um novo ciclo - e dormentes - nas quais o rádio não está em uso e pode rodar os algoritmos de aprendizado de máquina. Essa etapa é função dos planos e decisões geradas, bem como das observações do ambiente, já que os estados antigos e atuais podem ser comparados com os resultados esperados para estudar a efetividade da comunicação proporcionada (12).

Resumidamente, o rádio continuamente observa o ambiente, até que seus sensores captam um estímulo - tal como o aumento da densidade espectral de potência do ruído, a identificação de transmissores no mesmo canal ocupado pelo rádio, a degradação da qualidade de serviço provida e o aumento da taxa de perda de pacotes, dentre outros - acionando o ciclo cognitivo. Em seguida o equipamento se orienta, cria planos, decide e, por fim, age (12). No projeto em questão, atuaremos nas etapas de análise e decisão do rádio.

As principais funções de um rádio cognitivo são reconhecer o contexto da comunicação e mediar serviços de informação baseando-se nesse contexto de forma que o usuário não precise interferir, adicionando dados e tomando decisões, a não ser em último caso, como meio de prestar assistência às capacidades cognitivas.

Com todas essas características, os rádios cognitivos podem ser usados na configuração autônoma de sistemas de rádio, assim como na gerência autônoma do espectro. Contudo, essa tecnologia também apresenta desafios no seu desenvolvimento, principalmente no que diz respeito à alta complexidade necessária dos receptores - o que impacta diretamente no custo de produção do equipamento -, à detecção de sinais de baixa potência, à alocação dinâmica do espectro e em entender os mecanismos utilizados pelo cérebro no processo cognitivo e reproduzi-los em mecanismos para o rádio.

2.3 Cenários Operacionais

As operações militares são caracterizadas por possuírem atividades ininterruptas e, muitas vezes, em cenários desconhecidos. Portanto, para que ocorram com sucesso, é fundamental que as comunicações entre os escalões superiores e subordinados sejam mantidas durante toda a duração da missão (14).

No Exército Brasileiro temos operações de guerra - nas quais todas as capacidades operativas do exército são empregadas, aplicando princípios e procedimentos de combate - e operações de não guerra, nas quais as forças armadas agem em ações humanitárias ou de apoio a questões de segurança regionais no país. As operações militares, segundo o Manual as Comunicações nas Operações (15), podem ser classificadas quanto às forças empregadas e quanto à sua finalidade (Figura 6).

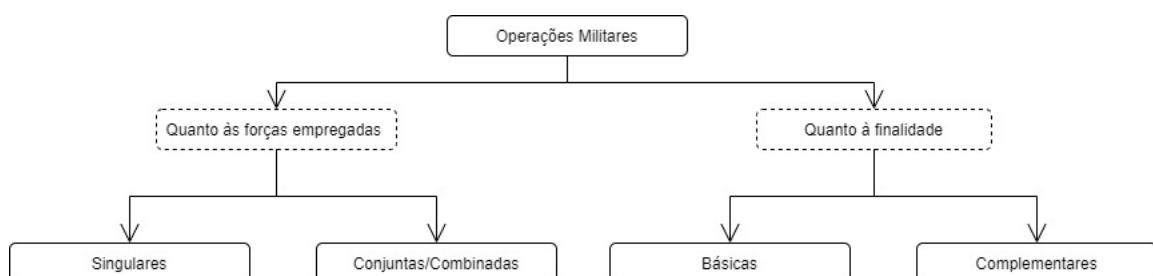


Figura 6 – Classificação das operações militares

Quanto às forças empregadas, podem ser singulares ou conjuntas/combinadas. As singulares são desenvolvidas por apenas uma das forças armadas, e o recebimento de pequenas frações e/ou meios de outra força não modifica este conceito. De maneira diferente, as operações conjuntas (Op Cj) são caracterizadas pelo emprego de meios ponderáveis de mais de uma força singular, com propósitos interdependentes ou complementares, sob

um comando único, com representantes das forças singulares no estado-maior (grupo de assessores de alto nível diretamente ligado ao comando da operação).

Por outro lado, quando observadas sob seu aspecto finalístico podem ser classificadas como básicas ou complementares. As operações básicas são operações que, por si mesmas, podem atingir os objetivos determinados por uma autoridade militar ou civil, em situação de guerra ou em situação de não guerra. Em situações de guerra, as operações básicas podem ser ofensivas ou defensivas, podendo contar também com operações de cooperação e coordenação com agências - estas últimas, mais comuns em situações de não guerra.

As operações complementares se destinam a ampliar, aperfeiçoar e/ou complementar as operações básicas, a fim de maximizar a aplicação dos elementos do poder de combate terrestre. Abrangem também, operações que, por sua natureza, características e condições em que são conduzidas, exigem especificidades quanto ao seu planejamento, preparação e condução, particularmente, relacionadas às táticas, técnicas e procedimentos ou aos meios (pessoal e material) empregados. Considerando as particularidades de cada operação, os meios de comunicação também precisam atender aos requisitos de cada operação militar. Desta maneira, em um ambiente de comunicações militares, fatores como mobilidade, capacidade, alcance, segurança, além dos procedimentos operacionais - também conhecidos como doutrina militar - influenciam na escolha do meio de comunicação a ser utilizado.

Conforme descreve o Manual de Comunicações nas Operações, as comunicações militares no âmbito do Exército Brasileiro podem utilizar como meios de comunicação: cabos/fios metálicos ou ópticos, o ambiente ou espectro eletromagnético, e também mensageiros ou recursos áudio visuais. Tratando em particular o meio eletromagnético, considerando o rádio como o equipamento de acesso ao meio, diferentes faixas de frequências podem ser utilizadas, como por exemplo as faixas de HF (3 MHz – 30 MHz), VHF (30 MHz - 300 MHz) e UHF (300 MHz – 3000 MHz) dentre outras. Cada uma dessas faixas é dividida em canais, ou seja, pequenas fatias do espectro eletromagnético, com larguras de banda definidas e com aplicações regulamentadas pela ANATEL (Agência Nacional de Telecomunicações).

As características técnicas do rádio como tipo de modulação, nível de potência de sinal transmitido ou recebido, susceptibilidade a ruídos e interferências, tipo de antenas utilizado e o ambiente geográfico em que o equipamento está inserido, podem influenciar o modo em que o rádio deve operar. Todo este conjunto de características definem a qualidade e até mesmo a possibilidade ou não de comunicação.

Além disso, em comunicações militares, também devem ser considerados fatores táticos como a operação em que o equipamento está inserido, o papel que o usuário do equipamento desempenha, a fase em que a operação se encontra, dentre outros. Em alguns casos, os fatores táticos podem ter precedência sobre as questões técnicas, ou seja, em uma determinada operação, mesmo tendo condições técnicas de se utilizar um rádio para

comunicar, o procedimento adequado a ser seguido pode ser o de manter o equipamento em silêncio, ou melhor, sem emitir nenhum sinal. Desta forma, é possível observar que podemos ter variações no modo de operação dos rádios durante as operações militares. Diante deste tipo de desafio, a flexibilidade para mudanças presente nos rádios definidos por software pode ser considerada um fator relevante, ainda mais se possuírem habilidades cognitivas, uma vez que com o uso de um mesmo equipamento será possível atender a mais de um modo de operação de forma autônoma.

2.4 MDE

No contexto da engenharia de *software*, um modelo é uma forma de representação gráfica ou textual de um sistema, fornecendo uma visão limitada deste, uma vez que sistemas do mundo real são muito complexos e precisam ser fragmentados e simplificados para que uma modelagem seja capaz de representá-los sob algum aspecto desejado. Para melhor defini-lo, as abstrações e os relacionamentos empregados neste são descritos por um metamodelo (5).

Modelos são extremamente úteis no processo de planejamento de um projeto, em especial de um projeto de *software*, porque possibilitam que todos os envolvidos, sejam eles da área técnica ou não, tenham uma visão mais concreta e clara do sistema a ser projetado, tornando a fase de planejamento mais eficiente (16).

Esta facilitação do entendimento do problema a ser atacado através de uma visão comum do sistema analisado, também foi o foco de metodologias mais antigas como a programação estruturada e a programação orientada a objetos. Porém, uma metodologia mais recente, a engenharia dirigida por modelos, também conhecida como engenharia orientada a modelos (MDE) vem sendo adotada por proporcionar outros benefícios ao público estratégico. Ao modificar a abordagem de solução do problema baseando-se na criação, refinamento e transformação de modelos do sistema e posterior conversão em código executável ao invés das antigas soluções focadas diretamente na codificação, o MDE possibilita uma flexibilização do projeto ao simplificar a propagação de mudanças.

Segundo Rodrigues da Silva(16), metodologias que concebem modelos como sendo artefatos elementares no processo de desenvolvimento de *software* e não apenas como documentação do projeto podem ser genericamente classificadas como engenharia dirigida a modelos (MDE). Esta abordagem tem como vantagens em relação às óticas mais antigas (programação estruturada e programação orientada a objetos), a capacidade de geração automática de código executável diretamente dos modelos do sistema mediante a utilização conjunta de técnicas como meta-modelagem, transformação de modelo para modelo, transformação de modelo para texto e geração de código. Já Bezivin, Jouault e Touzet(17) citam o aumento da qualidade dos sistemas de *software*, o grau de reutilização, além da

eficiência de seu desenvolvimento como as vantagens principais provenientes da adoção de princípios do MDE (5).

Com o intuito de padronizar a modelagem e meta-modelagem e consequentemente tornar estas representações inteligíveis por qualquer pessoa familiar com estas técnicas, uma arquitetura conhecida como *Meta-Object Facility* (MOF) foi proposta pelo *Object Management Group* (OMG), uma reconhecida organização internacional de desenvolvimento de padrões de projeto de código aberto com enfoque na orientação a objetos. Esta padronização divide a metamodelagem em quatro níveis (ou camadas), onde os elementos de um nível são instâncias de membros do nível imediatamente acima, sendo o último definido por elementos do próprio nível, conforme a Figura 7.

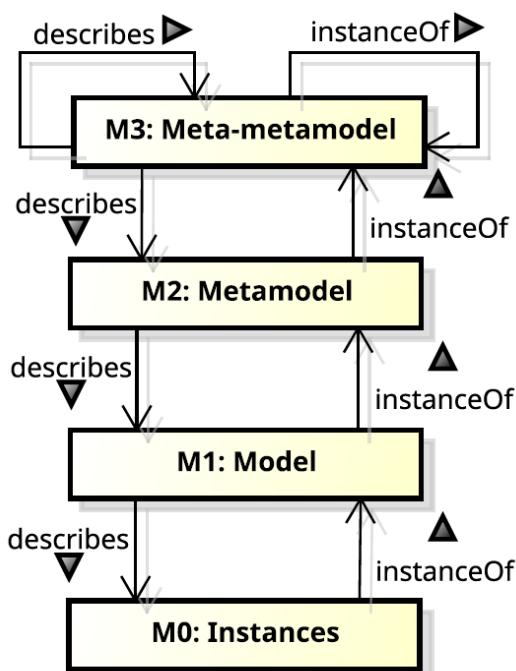


Figura 7 – Níveis da arquitetura MOF (5)

As camadas da arquitetura MOF, ilustradas na Figura 7, podem ser detalhadas como:

Nível M3: Engloba os meta-metamodelos da arquitetura MOF, sendo utilizado na definição de metamodelos. O também denominado Modelo MOF define seus próprios elementos e desta forma não necessita de um nível superior, estabelecendo assim um limite essencial do nível de abstração que se pode alcançar na modelagem de sistemas. Membros deste nível incluem o MOF¹ e o Ecore², que é baseado no primeiro.

Nível M2: Engloba os metamodelos da arquitetura, que possibilitam a modelagem

¹ <<https://www.omg.org/mof/>>

² <<https://wiki.eclipse.org/Ecore>>

de sistemas de domínio específico. Um membro deste nível é a *Unified Modeling Language* (UML).

Nível M1: Engloba modelos de sistemas cujas definições se encontram nos respectivos metamodelos de M2.

Nível M0: Composto pelas instâncias ou entidades do sistema em execução, as quais são geradas baseadas nas definições dos respectivos modelos em M1 (5).

Para que modificações possam ser propagadas pelos diferentes modelos até o código fonte da aplicação em desenvolvimento, existem dois tipos básicos de transformações que são utilizadas ao adotar uma abordagem condizente com as práticas do MDE: transformações Modelo-para-Modelo (*Model-to-Model Transformations* - M2M) e as transformações Modelo-para-Texto (*Model-to-Text Transformations* - M2T).

As transformações Modelo-para-Modelo permitem a transformação de um modelo em outro, normalmente para um nível de abstração menor do que o original ou simplesmente que o novo modelo derivado seja mais conveniente aos envolvidos no projeto. Já as transformações Modelo-para-Texto geram, a partir dos modelos, artefatos de *software* através de uma técnica chamada de geração de código.

Para que as transformações M2T sejam viabilizadas, ferramentas disponíveis como o Acceleo³ (explorada na seção 3.1.2) necessitam receber como entradas um *template* de código com informações de como o programa final deve ser gerado (seção 6.2.1) e uma instância do modelo em questão. Para a geração desta, pode-se utilizar ferramentas com interfaces gráficas ou que possibilitem a geração e edição de uma DSL, que então pode ser utilizada para este fim.

Ambos os tipos de transformações citados podem ser determinados através de linguagens de programação (como C, Java, Python, etc), porém existem linguagens projetadas especificamente para estas aplicações, cada qual com sua especificidade e vantagens.

2.5 DSL

De acordo com os princípios do MDE, pode-se considerar que a geração de uma DSL é uma forma de facilitar a instanciação de metamodelos para transformações M2T, especialmente com a ajuda de ferramentas como o Xtex (explorado na seção 3.1.2). Uma linguagem específica de domínio (DSL) é uma linguagem criada especificamente para um determinado domínio de aplicação, como as linguagens de consulta (SQL, XPath), as linguagens de documento (Latex, HTML, CSS), entre outras. Ao contrário desta, uma linguagem de propósito geral (GPL) é desenvolvida para ser aplicável independente do

³ <<https://www.eclipse.org/acceleo/>>

domínio em questão, como C, C++, Java, Python, JavaScript, etc.

A definição de uma linguagem apropriada ao domínio do problema é uma etapa essencial da sua solução. Pode-se optar por utilizar uma DSL ou GPL já existentes ou desenvolver uma DSL. O desenvolvimento de uma nova DSL pode ser mais útil do que usar uma linguagem que já existe caso essa linguagem criada possibilite que se possa expressar de uma maneira mais compreensível e simples os elementos do domínio e se o trabalho para desenvolvê-la compensar a utilização de DSLs ou GPLs disponíveis

3 TRABALHOS MOTIVADORES

3.1 *A Model-Driven Approach to Situations*

Um dos trabalhos que inspiraram utilizar MDE como solução para tornar o RDS um rádio cognitivo foi o “*A Model-Driven Approach to Situations*” (6). Esse artigo apresenta uma abordagem sobre a capacidade de um sistema ter conhecimento da sua situação, ou seja, de perceber e reagir ao ambiente no qual se encontra de acordo com situações de interesse. Esse conhecimento vem do sistema conseguir reconhecer situações e associar à elas diversas propriedades, também comunicando essas descrições para terceiros.

O artigo também pontua uma característica importante dessa capacidade, que é a de afetar diferentes etapas do projeto, tanto em tempo de planejamento - no qual comportamentos e políticas de funcionamento são definidas baseadas em quais situações são aplicáveis - quanto em tempo de execução, no qual a detecção da situação permite que o sistema reaja prontamente à mudanças no ambiente.

Para permitir a abordagem de interesse, o texto utiliza técnicas de engenharia baseada em modelos, especificamente SML (*Situation Modeling Language*), que é uma linguagem gráfica para modelagem, e uma plataforma que executa regras geradas a partir da transformação do modelo SML com a finalidade da detecção das situações. Nesse caso especificamente, a plataforma escolhida foi o Drools.

Além disso, destaca-se que, para a modelagem, também é necessário caracterizar os elementos básicos do domínio do projeto em questão. Com o contexto, define-se um vocabulário que será usado nas definições de cada situação.

O trabalho também define situações, que são formadas por padrões, entidades, relações e propriedades. Elas podem ser complexas (conjunto de situações simples) e podem levar em conta o aspecto temporal (se a situação é presente ou ocorreu no passado). Para detectar situações, é necessário, portanto, detectar as instâncias das entidades nelas envolvidas.

Para exemplificar as explicações durante o artigo, foi utilizado um cenário de um banco móvel (*mobile*), principalmente na área de detectar fraudes. Em seguida, com base nesse panorama, descreve-se como o SML funcionaria para esse caso - principalmente no que diz respeito à padronizações de representação das situações, entidades, propriedades e relações, e à possibilidade de se definir restrições para as propriedades das situações - e como são feitas as construções de regras e restrições a partir do modelo SML no Drools.

Outro importante conceito apresentado é a de uma visão global de abordagens orientadas à modelos e baseadas em regras para situações (Figura 8 abaixo). Vale lembrar

também que o ciclo de vida da situação se inicia na sua criação e ativação e acaba na sua desativação e destruição uma vez que a regra que a define não é mais válida de acordo com o ambiente externo.

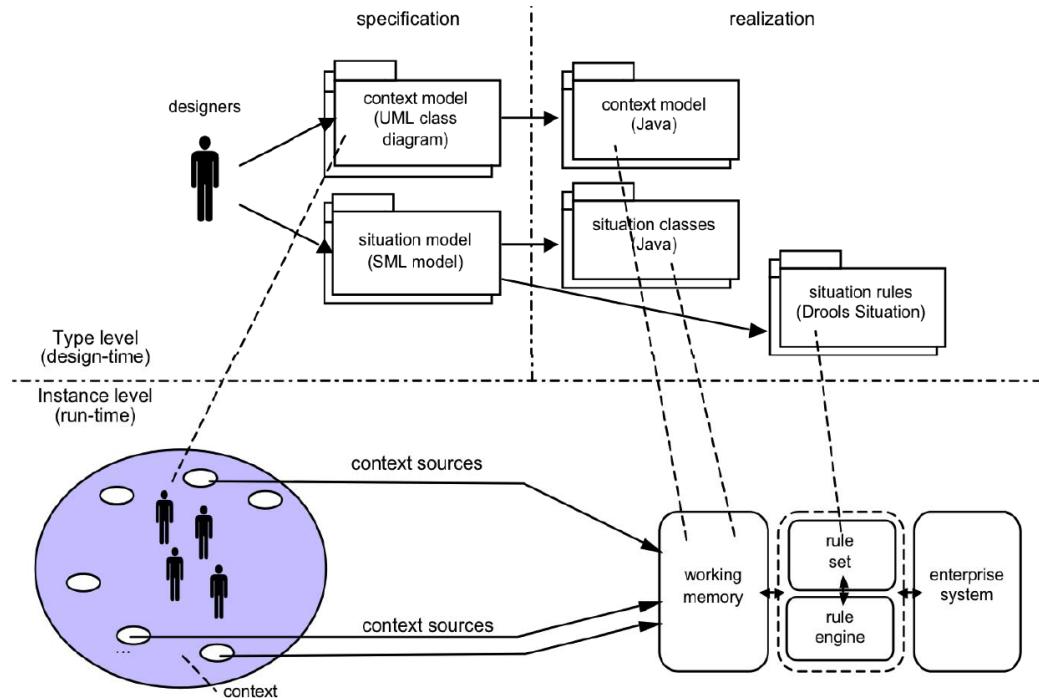


Figura 8 – Visão geral da abordagem proposta no artigo (6)

Com base nos conceitos e exemplificações apresentadas no artigo, foi possível confirmar a relevância de utilizar engenharia baseada em modelos e como ela pode ser um importante artifício para facilitar a geração de soluções e leitura de situações do ambiente externo de forma a gerar um sistema responsável.

3.2 *Towards ontology-driven situation-aware disaster management*

O artigo "*Towards ontology-driven situation-aware disaster management*" (8) discute a importância de uma ontologia bem fundamentada para gestão de desastre com o intuito de suportar as especificações de aplicações baseadas em consciência da situação e do ambiente no qual a situação ocorre.

Inicialmente, define-se um desastre como sendo uma situação na qual ocorre uma série de interrupções no bom funcionamento de uma comunidade e que excedem a habilidade dessa comunidade em lidar com o acontecimento com recursos próprios. Além disso, trata de como métodos de suporte à decisão e sistemas de gestão de emergências que detectam uma situação e reagem à mesma se desenvolveram ao longo dos anos com a finalidade de apoiar as quatro fases da gestão de desastres: mitigação - na qual os riscos de um desastre são analisados e definem-se técnicas de prevenção -, preparação - ocorrência de um desastre

iminente é antecipada e executam-se ações para diminuir os impactos -, resposta - quando recursos são fornecidos para reduzir os estragos - e recuperação.

A especificação, obtida na fase de planejamento desses sistemas, chamados de aplicações de consciência da situação (SA, do inglês *Situation Awareness*), é um dos pontos mais importantes para o desenvolvimento, não só para aspectos estruturais, mas também temporais e as relações entre os mesmos. Para tanto, bons resultados têm sido obtidos com abordagens de modelagem conceitual baseadas em ontologia, que formalizam um sistema de categorias dos conceitos mais fundamentais e suas relações. Uma ontologia bem fundamentada fornece uma definição precisa de uma área, sendo independente do domínio de aplicação.

Para gestão de desastres, outra importante característica é a interoperabilidade, isso porque lida com a heterogeneidade da informação, evitando mal entendidos ao comunicar sobre questões delicadas. O artigo também apresenta diversas questões que representam as dificuldades ontológicas relacionadas a essa propriedade, bem como aos outros empecilhos na criação de uma ontologia bem fundamentada.

O *framework* utilizado foca tanto no tempo de planejamento quanto de execução no suporte de aplicações de SA. Para os casos tratados no texto, as situações são modeladas em linguagem de modelagem de situação (SML), que pode aproveitar a ontologia assumindo suas categorias como estereótipos e determinar restrições baseadas na mesma.

O artigo também explica o uso de uma linguagem de modelagem chamada OntoUML, que foi aplicada em uma variedade de domínios e inclui uma abordagem baseada em modelos, ou seja uma metodologia de engenharia baseada em modelos na qual modelos são ontologicamente bem fundamentados.

Para a modelagem de situações, podem ser utilizadas duas técnicas diferentes, uma baseada em especificação, na qual as características que definem uma situação são determinadas por um especialista de conhecimento, durante o tempo de planejamento, e outra baseada em aprendizado, utilizando métodos de aprendizado de máquina e inteligência artificial. Também é possível aplicar ambas as técnicas em uma abordagem híbrida.

O texto também explica mais sobre SML, como ela suporta a abordagem baseada em especificação, permitindo que as definições dos tipos de situações sejam transformadas em restrições que englobam suas propriedades e relações, além de mostrar sua representação gráfica.

No tempo de planejamento do *framework* citado, o contexto do modelo é projetado em uma linguagem ontológica e mapeia a especificação da aplicação SA a diferentes tecnologias, como sistemas baseados em regras. Já em tempo de execução, um evento é responsável por compartilhar as situações detectadas entre as aplicações SA.

Talvez o conceito mais relevante desse artigo para o nosso trabalho é o de que

a modelagem de situação permite descrever situações por meio de padrões, que serão detectados por aplicações SA, e descrever conjuntos de regras para definir as ativações dessas situações. Além disso, durante o tempo de planejamento também são realizáveis processos de validação e verificação, nos quais pode-se visualizar possíveis instâncias do modelo e fazer alterações, caso necessário, melhorando a semântica.

Uma vez que a qualidade das especificações atingir um nível adequado e satisfatório, pode-se aplicar um processo de engenharia baseado em modelos que gera uma implementação da especificação por meio de transformações do modelo, resultando em um código para a aplicação SA, que por sua vez é menos sujeito a erros já que é gerado automaticamente por alguma tecnologia.

Esse artigo, portanto, é de total importância por demonstrar como o desenvolvimento de aplicações SA é crucial e como as abordagens baseadas em ontologia ajudam a resolver os problemas apresentados, dependendo da especificação de situações. Além disso, mostra também como existem muitos desafios para a construção e adaptação de uma ontologia bem fundamentada para esse fim.

3.3 Comparativo com o trabalho proposto

Os trabalhos motivadores se relacionam com o trabalho proposto no ponto em que ambos explicam como utilizar engenharia dirigida a modelos para gerar uma estrutura com a finalidade de suportar um sistema com consciência de situação, sendo capaz de reagir a estímulos externos.

O que diferencia a nossa proposta é que os trabalhos acima citados utilizam SML e Drools, que são mais robustos e pesados, os quais são adequados para os propósitos dos artigos. Porém, como queremos que seja possível empregar a estrutura obtida em um rádio definido por software para torná-lo cognitivo e este possui capacidade de processamento limitada, utilizaremos um ferramental mais leve, uma vez que as decisões são baseadas apenas na linguagem de programação Java.

A ferramenta Drools é uma solução de gestão de regras e necessita de um servidor como o tomcat *server* (ou seja, tem um intermediário). Logo, comparando com um código puramente em uma linguagem de programação de alto nível, como Java, sendo executado localmente, o primeiro será mais custoso computacionalmente. Dessa maneira a ferramenta citada se mostra menos adequada para a utilização em ambientes de rápida decisão e com recursos limitados, onde não se pode centralizar o processamento dos dados em uma máquina externa.

4 ABORDAGEM PROPOSTA

Para atender o objetivo de levar o “poder de decisão” para os rádios definidos por software (que atuam nas pontas de um cenário de operação militar), de acordo com um planejamento prévio, adotamos uma abordagem baseada em MDE. A ideia é dotar o rádio de poder cognitivo através de um programa decisor, que mude o comportamento do rádio (modo de operação) conforme a situação varie. As situações previstas e o comportamento desejado para o rádio foram modeladas em alto nível e transformadas em código a ser entregue ao rádio, que por sua vez passa a operar segundo tal modelagem. Para entender melhor tal abordagem, inicialmente são apresentados cenários de operação dos rádios e levantadas algumas regras que explicitam como estes deveriam se comportar. Em seguida, uma visão geral de como desenvolvemos nossa abordagem MDE é apresentada, além das ferramentas utilizadas. Por fim, é explicitado o cenário idealizado onde a solução proposta se insere.

4.1 Cenários de emprego de rádios em operações militares

Para contextualizar nosso entendimento, serão descritos cenários hipotéticos reduzidos, demonstrando o emprego de rádios em operações militares com possíveis modos de operação a serem empregados em cada tipo de contexto, além das transições entre os modos de operação.

No cenário descrito inicialmente na Figura 9, temos três rádios envolvidos em duas operações (A e B). Na operação A temos os rádios do comandante da operação e de um operador, e na operação B temos apenas o rádio do comandante da operação.

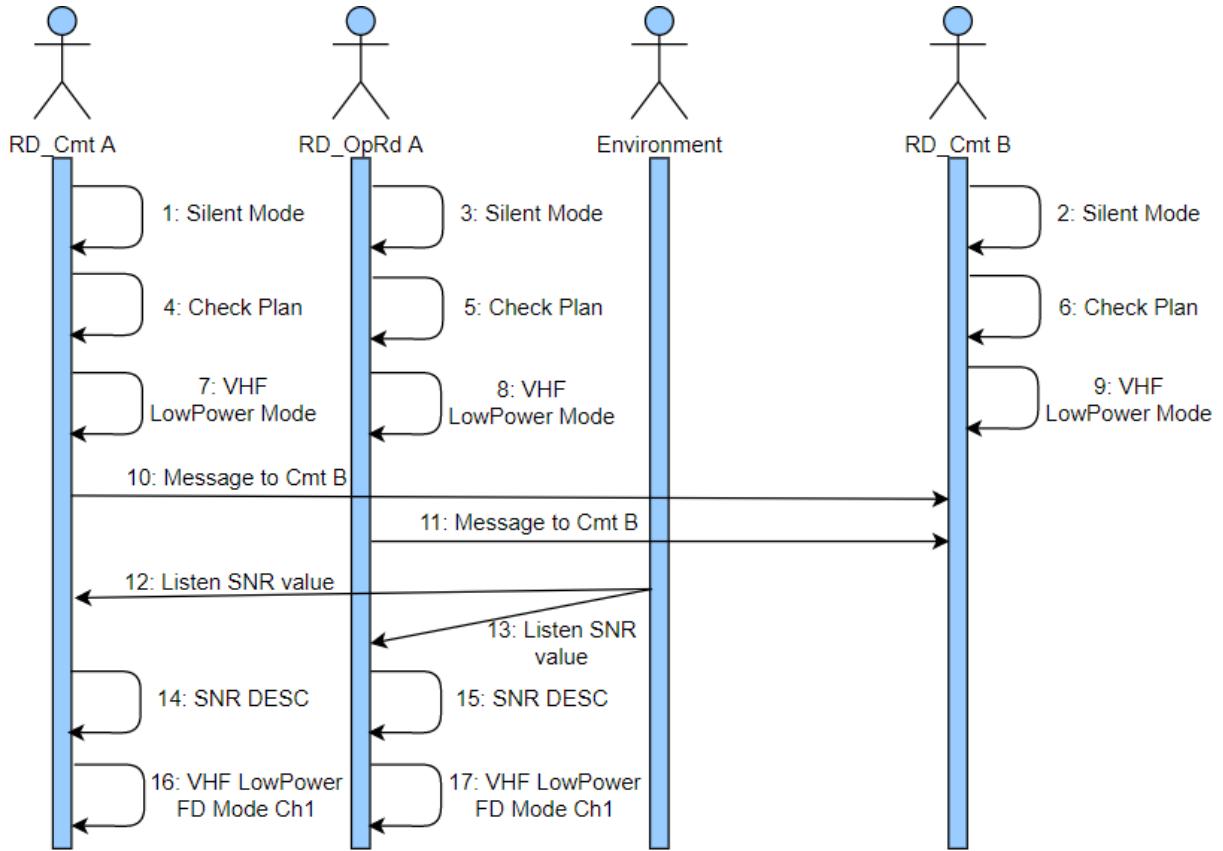


Figura 9 – Diagrama de sequência das comunicações em operações (aspectos técnicos)

Neste ambiente temos quatro modos de operação para os equipamentos: o modo silêncio (SilentMode) - onde o equipamento está ligado, porém não emite nenhum sinal -, o modo alerta, o modo VHF de baixa potência (modo normal) e o modo VHF de baixa potência FD. Neste último modo a sigla FD significa que o equipamento pode operar no modo In-band Full Duplex, no qual o rádio transmite e recebe na mesma frequência de forma simultânea, produzindo uma interferência sobre seu próprio sinal e aumentando a segurança na camada física (18). No modo de alerta, o qual corresponderia a uma indicação luminosa no painel do rádio para o operador.

Outro elemento é o plano de comunicações, que contém detalhes sobre a operação na qual o equipamento está envolvido - se ela é de guerra ou não-guerra, por exemplo -, a frequência de transmissão, o papel do operador de cada equipamento na operação etc. Em nosso exemplo, consideraremos que esses parâmetros já estão disponíveis para o software do rádio.

Outro indicador de interesse é a relação sinal – ruído (SNR, do inglês Signal-to-Noise Ratio), ou seja, a relação entre o nível de sinal recebido e o nível de ruído percebido no ambiente eletromagnético. Uma queda em SNR pode ser ocasionada por uma baixa no nível de sinal recebido ou pelo aumento do nível de ruído percebido. Representaremos essa

relação pelo parâmetro *Signal To Noise Ratio*.

Descrevendo o diagrama da Figura 9, verificamos inicialmente, que o ambiente eletromagnético (Environment) também é representado a fim de ilustrar a ação de sensoriamento ou percepção do ambiente executada pelos rádios. Inicialmente, todos os rádios funcionam em modo silêncio. Em seguida, uma aplicação de checagem de plano de operação é executada e informações como nível de segurança (*Security Level*, que depende de quão crítica é a operação), papel do equipamento na operação (*Operational Role*), a frequência do canal utilizado (*Carrier Frequency*) dentre outras podem ser coletadas. A partir destas informações, em conjunto com uma regra pré-definida no plano de operações, o rádio já pode definir seu modo de operação. Em nosso exemplo, o modo definido foi o VHF baixa potência. Na sequência, mensagens são trocadas e os equipamentos da operação A verificam o ambiente e percebem que a relação SNR apresenta um decréscimo (SNR DESC), ou seja, houve uma perda na qualidade do sinal recebido. Diante disso a ação passa a operar no modo VHF baixa potência FD.

Observando o diagrama da Figura 10, podemos verificar que as situações iniciais são idênticas às descritas anteriormente, porém ao ser dada a ordem para que a operação B seja iniciada, mesmo após ser percebida uma queda na relação sinal ruído, o modo de operação passou a ser o modo silêncio, considerando, neste caso, um requisito tático ligado à operação.

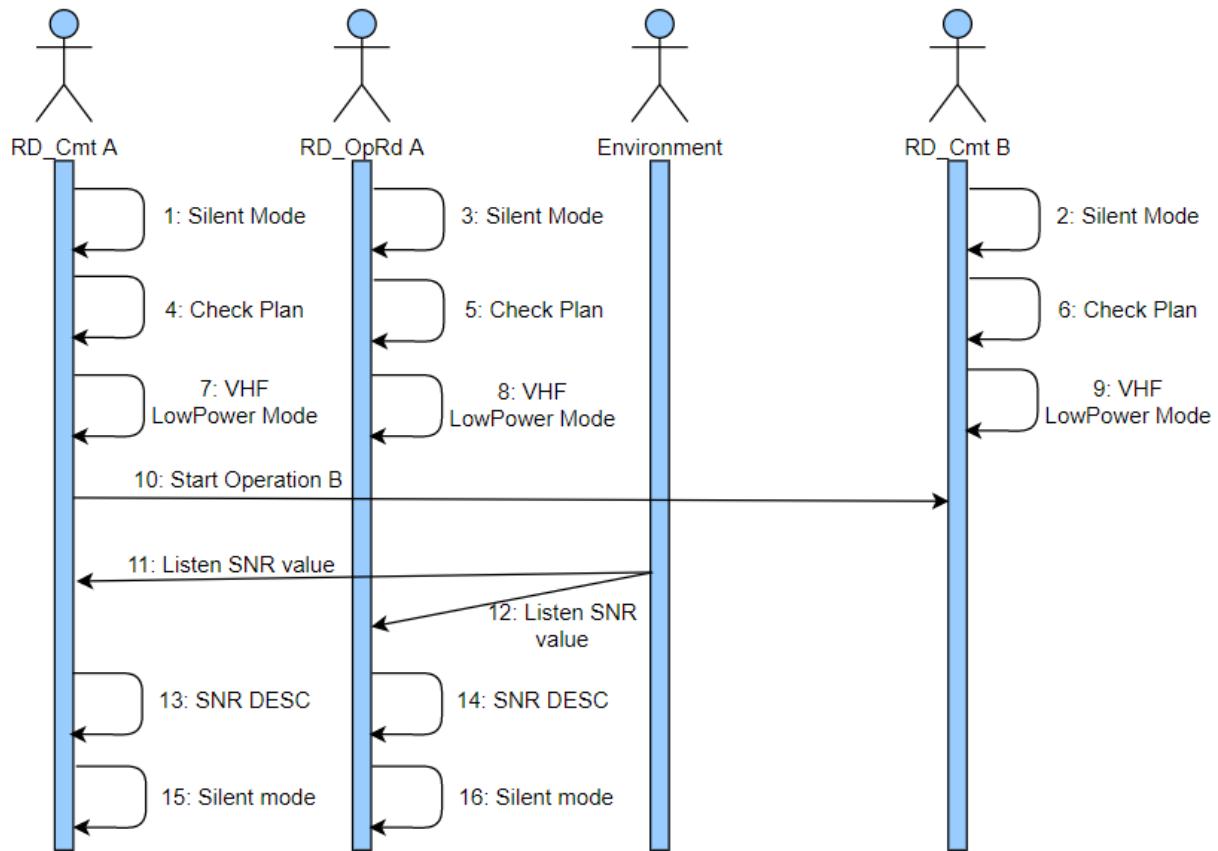


Figura 10 – Diagrama de sequência das comunicações em operações (aspectos táticos)

A fim de realizar algumas validações, as figuras 11 e 12 apresentam um conjunto de possíveis variações de regras que poderiam ser aplicáveis em ambientes dessa natureza e que serão tomadas como base para o nosso projeto.

Tipo	Objeto	Variáveis	Valores possíveis	ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7
Entrada	Environment	(w3) Signal To Noise Ratio Value	ASC	true		true		true		false
			DESC		true		true		true	false
	Communication's Plan	(*) Security level	>=0,5	true			true			
			< 0,5		true	true				
	(*) Carrier Frequency	50 MHz	true				true			
			100 MHz		true	true				
		(*) Operational Role	Cmt	true		true				
			Soldier		true		true			
	Saída	Transceiver	Tx mode	Normal Mode						
				FD Mode						
				Silent Mode						
				Alert Mode						
				Mantém modo						

(w3) Tamanho da janela indicando que os 03 (três) valores mais modernos percebidos serão classificados como ASC (tendência ascendente) ou DESC (tendência descendente);

(*) Último valor capturado pelo rádio; ST → Scenery Type;

Figura 11 – Conjunto de regras hipotético para testes da arquitetura

$$ST_1 = ((ASC=true) \wedge ((SL>=0,5) \wedge (frequency=50) \wedge (role=cmt)))$$

$$ST_2 = ((DESC=true) \wedge ((SL<0,5) \wedge (frequency=100) \wedge (role=soldier)))$$

$$ST_3 = ((ASC=true) \wedge ((SL<0,5) \wedge (frequency=100) \wedge (role=cmt)))$$

$$ST_4 = ((DESC=true) \wedge ((SL>=0,5) \wedge (frequency=50) \wedge (role=soldier)))$$

$$ST_5 = ((ASC=true) \wedge (!((SL>=0,5) \wedge (frequency=50) \wedge (role=cmt)) \vee ((SL<0,5) \wedge (frequency=100) \wedge (role=soldier)) \vee ((SL>=0,5) \wedge (frequency=50) \wedge (role=soldier)) \vee ((SL>=0,5) \wedge (frequency=50) \wedge (role=cmt))))$$

$$ST_6 = ((DESC=true) \wedge (!((SL>=0,5) \wedge (frequency=50) \wedge (role=cmt)) \vee ((SL<0,5) \wedge (frequency=100) \wedge (role=soldier)) \vee ((SL>=0,5) \wedge (frequency=50) \wedge (role=soldier)) \vee ((SL>=0,5) \wedge (frequency=50) \wedge (role=cmt))))$$

$$ST_7 = ((ASC=false) \wedge (DESC=false))$$

OBS: $\forall asc = true \rightarrow desc = false$

Figura 12 – Descrição das regras da Figura 11

Tomando como exemplo a regra determinada pela situação ST_1 da Figura 11 (coluna ST_1), pode-se perceber que; caso a tendência dos últimos três valores do SNR recebido seja ascendente (ou seja, o valor SNR dos últimos três instantes percebidos está aumentando), o nível de segurança do canal seja maior ou igual a um limiar (0,5 nesse caso), a frequência de transmissão seja de 50 MHz e se o "papel operacional" do rádio em questão for o de comandante ("Cmt"); então o modo de operação a ser utilizado no equipamento é o modo silêncio.

Os valores dos limiares de nível de segurança utilizados nas operações militares reais não necessariamente são os expostos na Figura 11, porém, o mais relevante para o presente trabalho é que uma mudança deste parâmetro pode afetar o modo de transmissão que um rádio deve adotar. A mesma análise pode ser feita para os valores das frequências de transmissão utilizadas, as quais estão na faixa do VHF, o que é coerente com a operação de rádios militares, mas cujos valores exatos utilizados em campo podem não corresponder à tabela.

Também vale ressaltar que, conforme a Figura 12, a tabela apresentada não traduz completamente as condições das regras apresentadas. Isso acontece pois existe uma ordem de prioridade implícita na Figura 11, sendo a regra da situação ST_1 a de maior prioridade e a ST_7 a de menor precedência. Dessa forma, dados os parâmetros percebidos, o conjunto de regras sempre resultará em apenas um resultado para o modo do rádio.

4.2 Tecnologias Disponíveis

A utilização de um *framework* amplamente utilizado em projetos de software e que possui uma vasta rede de apoiadores é de suma importância para que os conceitos atrelados à Engenharia Dirigida por Modelos sejam implementados em projetos reais. A escolha dessa ferramenta tem implicações ao longo de todo o ciclo de vida de um projeto, especialmente na fase após sua implementação através da manutenção e suporte do sistema, o que reforça a adoção desses princípios no desenvolvimento de um ambiente de simulação da cognição dos equipamentos no contexto de um rádio cognitivo.

4.2.1 EMF

Nesse contexto, um framework que se destaca é o *Eclipse Modeling Framework* (EMF), o qual foi desenvolvido sobre o conhecido ambiente de desenvolvimento integrado (IDE) Eclipse e possui uma ampla comunidade de suporte. O EMF facilita a geração, modificação e validação de modelos e metamodelos através da disponibilização de ferramentas específicas para cada uma das etapas citadas, além de possibilitar que, a partir dos modelos, código seja gerado mapeando as classes do metamodelo em classes na linguagem Java. No EMF os metamodelos construídos são instâncias do meta metamodelo Ecore, que se baseia no meta metamodelo MOF, o qual, por sua vez, é um padrão da OMG. Muitas ferramentas utilizadas atualmente foram construídas sobre o EMF, como Sirius, GMF, Papyrus, Xtext, Acceleo, ATL entre outras.

4.2.1.1 Acceleo

Ao adotar uma abordagem pautada nos princípios do MDE, na última etapa de uma cadeia de transformação de um modelo (imediatamente antes da compilação) são aplicadas

as transformações M2T, as quais podem ser realizadas com a ajuda de diversas ferramentas disponíveis, como XPand, JET e Acceleo. Dentre elas, a que mais se destaca é o Acceleo, o qual é uma implementação do padrão MOF *Model to Text Language* (MTL) definido pela OMG e proporciona uma ótima usabilidade, além de uma curva de aprendizado alta, sendo muito simples a sua utilização em comparação com os concorrentes.

Como entradas, a ferramenta em questão recebe uma instância do metamodelo em formato XMI e um *template* de código. O resultado da transformação M2T é um documento em linguagem especificada no *template* e formatado de acordo com este.

4.2.1.2 Xtext

Xtext é uma ferramenta de código aberto amplamente utilizada que faz parte do EMF e é dedicada à criação de linguagens de programação e linguagens de domínio específico (DSLs). A partir de uma DSL do sistema a ser estudado, possibilita a instanciação de elementos através do ambiente de desenvolvimento e a validação de regras sobre o modelo, as quais podem ser definidas através da linguagem OCL.

4.3 Desenvolvimento e testes

Como já mencionado, o objetivo deste trabalho era desenvolver uma arquitetura que possibilitasse a adaptabilidade do modo de operação de um rádio através da adoção de técnicas de MDE. Assim, nesta seção é apresentada uma visão geral dos passos que foram percorridos, ilustrada na Figura 13. Partiu-se do desenvolvimento do metamodelo de um rádio em um ambiente operacional (passo 1), bem como da estruturação de como ocorre a formação de regras que determinam o modo de operação deste através de um segundo metamodelo (passo 2). Em seguida, foi desenvolvido um *template* de código em linguagem de transformação MTL baseado no metamodelo de formação das regras (passo 3). Após este modelo de código estar pronto, cada uma das regras necessárias para os testes do presente trabalho foram então instanciadas a partir do metamodelo base do *template* com a ajuda da interface gráfica do Eclipse (passo 4).

A partir do modelo de código e das instâncias de regras criadas, uma ferramenta de transformação M2T, que neste projeto foi o Acceleo, possibilitou a geração automática de uma classe em linguagem Java para cada regra (passo 5), cuja formatação está definida no próprio *template* desenvolvido. Na última etapa, as classes de todas as regras geradas na etapa anterior são fornecidas a um programa decisor em linguagem Java - o qual é um componente modificado de um projeto desenvolvido em (19), o qual simula os dados percebidos por um rádio -, que é então compilado, gerando como artefato final do trabalho o programa decisor executável (passo 6).

Uma vez feito esse caminho, foi possível repensar a instanciação das regras, e para

isso foi necessário desenvolver uma linguagem específica de domínio - DSL - para facilitar a formulação das regras (passos 7 e 8).

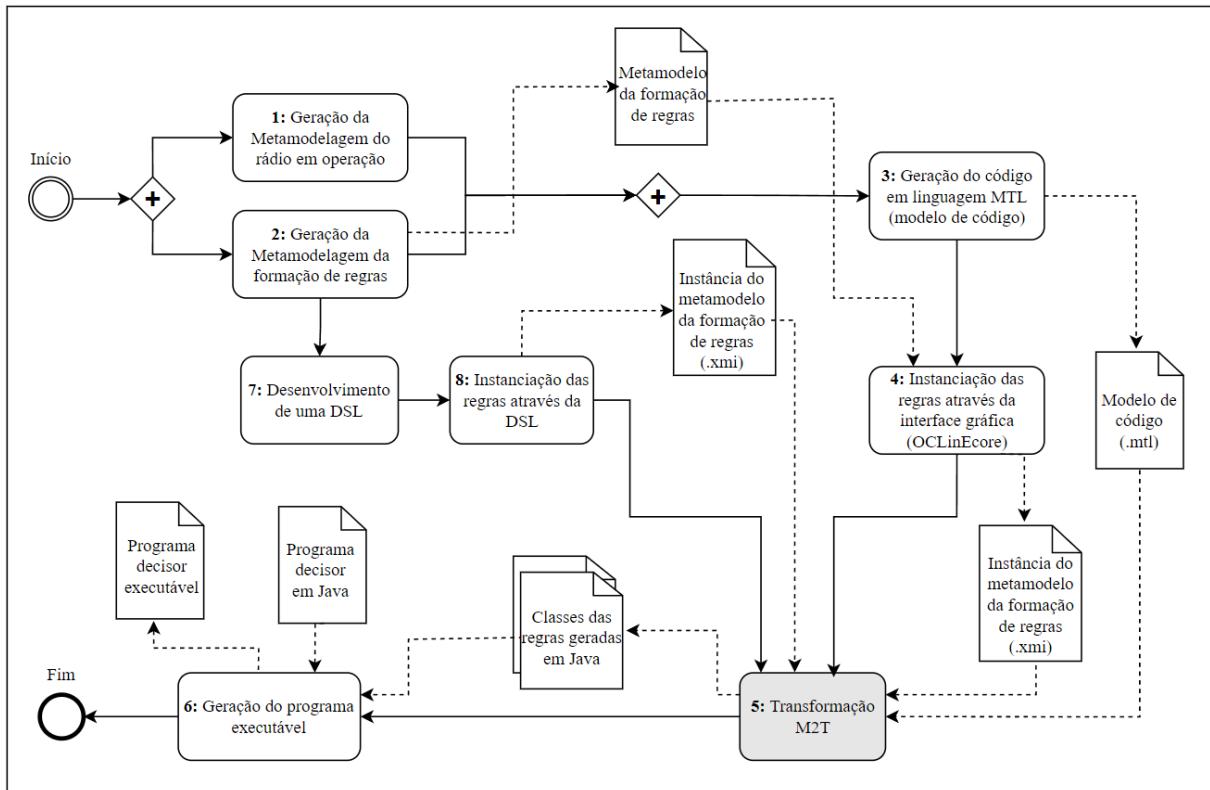


Figura 13 – Visão geral do desenvolvimento da solução proposta

Na Figura 14, pode-se observar como foi organizada a fase de testes do sistema desenvolvido. Inicialmente, a partir de um programa gerador de arquivos de teste presente no projeto desenvolvido em (19), é gerado um arquivo com os dados brutos, em formato que simula o sensoriamento do ambiente (passo 1). Com os dados simulados disponíveis, o programa executável gerado na última etapa do trabalho (Figura 13) realiza a leitura e o tratamento (validação) dos dados deste arquivo (passo 2), seguido da atualização de uma classe de Log - a qual armazena os valores mais recentes percebidos pelo dispositivo - (passo 3) e então determina para cada instante de tempo presente nos dados de teste (cada leitura), o novo estado/modo de operação do rádio (passo 4), levando em consideração apenas os valores presentes na classe Log no instante da leitura.

Esse programa gerado pela etapa de transformação M2T determina o estado do rádio baseado nos dados que percebe do ambiente. Uma vez definido, o modo poderá ser ativado por meio de aplicações locais presentes no rádio (passo 5), como por exemplo uma aplicação que o configura em *Silent Mode*. Estas aplicações locais estão fora do escopo dessa pesquisa. Ou seja, o código final apenas indicará qual o novo modo de operação, sendo que o *deployment* e interface deste código com *software* real de um RDS não serão

abordados.

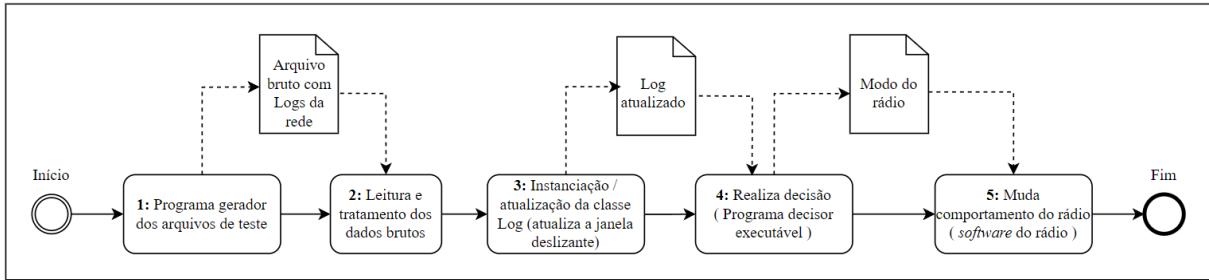


Figura 14 – Etapa de testes da solução proposta

Ao final da geração automática dos códigos em java referentes às regras (passo 5 da Figura 13), foram comparados os resultados com as saídas obtidas a partir da execução de um modulo (arquivo CheckMode.java) validador de regras (também de acordo com os cenários propostos na Figura 11), previamente escrito (também em linguagem java), de forma não automática. Este projeto utilizado e modificado para os testes do presente trabalho está disponível em: <<https://github.com/GabrielBozza/FileMakerV2PFC>>.

Para verificação da aplicação das regras, em ambos os casos, foram conferidas as saídas do programa, com base nas entradas do arquivo bruto de Logs, considerando inclusive, variações controladas realizadas nas entradas. Todas as etapas dos testes do sistema estão descritas na seção 7.3.

4.4 Ferramentas utilizadas

Para o desenvolvimento das etapas propostas no item anterior e expostas na Figura 13, com base na avaliação das tecnologias disponíveis de código aberto, como foi explanado no capítulo 3 do presente trabalho, foi utilizada da IDE Eclipse, a qual suporta ferramentas do chamado *Eclipse Modeling Framework* (EMF), que engloba múltiplas *frameworks* de código aberto que simplificam e agilizam o desenvolvimento de projetos baseados em modelos.

Para o desenvolvimento das metamodelagens iniciais (do ambiente e da formação das regras), foi utilizada a extensão Sirius do Eclipse. Esta proporciona uma interface gráfica amigável e intuitiva, o que torna o trabalho mais ágil.

Na etapa de transformação de modelo para um programa em linguagem Java, foi usada a ferramenta Acceleo para viabilizar e simplificar o passo de M2T da solução proposta, uma vez que esta é uma das ferramentas mais simples disponível.

O *plugin OCL in Eclipse* permite que se possam gerar instâncias dinâmicas do metamodelo que descreve a formação das regras a partir de sua classe raiz. Dessa maneira,

cada regra pode ser instanciada e validada através de uma interface gráfica. Como ponto negativo, pode-se citar que esta interface pode ser complicada de ser utilizada por algum usuário mais leigo, uma vez que a regra não aparece no monitor com a formatação com a qual ela é concebida previamente e que o usuário estaria familiarizado.

Para simplificar a instanciação do metamodelo das regras, como etapa final do trabalho, a ferramenta Xtext foi utilizada para a geração e edição de uma DSL com base neste metamodelo. A criação de uma DSL através do Xtext possibilita com que o usuário responsável por instanciar as regras dos rádios não precise do conhecimento de nenhuma linguagem de programação específica e tão pouco dependa da interface gráfica do *plugin OCL in Eclipse* (que pode ser confusa caso haja um alto nível de aninhamento das classes). Esta DSL foi desenvolvida de forma que o usuário pudesse escrever a regra da maneira mais próxima possível do formato que lhe é conhecido, vide Figura 11.

4.5 Cenário pretendido da solução proposta

Como foi visto nas seções anteriores, no presente trabalho, o protótipo desenvolvido gerou os seguintes produtos: - um metamodelo do ambiente de operação do rádio - um metamodelo de regras - uma linguagem específica de domínio para instanciação de regras de operação de rádio - um *template* de código para a incorporação de tais regras como um código. - um ambiente de simulação da operação do rádio.

Usando o ambiente da ferramenta Acceleo é possível gerar as regras e depois transformá-las na forma de código Java (ou qualquer outra linguagem que seja pertinente). No presente trabalho, esse código é testado em ambiente simulado. No entanto, a ideia é, a partir desse protótipo, gerar uma solução em um contexto mais amplo de interoperabilidade de sistemas de comando e controle. Mais especificamente, no contexto do Projeto Sistemas de Sistemas de Comando e Controle (S2C2), uma vez que nesse cenário o Rádio Cognitivo é um projeto integrante relevante para a constituição de um sistema de comunicações orgânico.

A Figura 15 apresenta o cenário pretendido onde a solução desenvolvida se insere. A ideia é facilitar a interoperabilidade entre o que é planejado e os combatentes que estão na ponta da operação militar, de modo ágil e simples. A primeira atividade representa o sistema de planejamento onde o grupo de comando da operação concebe as regras de operação dos rádios, como as regras exemplificadas na seção 5.1. Essas regras, uma vez geradas em determinado formato, são entregues para a segunda atividade onde são transformadas em um código executável (como ocorre no passo 6 da Figura 13). Esse código (SW decisor) é distribuído (entregue) para os rádios através de uma rede segura (pode ser entendido como uma atualização dos parâmetros do rádio mais do que como uma atualização de *software* em si). Por fim, uma vez com a nova versão do SW decisor,

que realiza a operação do rádio, o equipamento passa a se comportar segundo as regras planejadas, mudando seu modo de operação conforme a situação de combate se apresenta. Por exemplo, entrando em modo de operação FullDuplex.

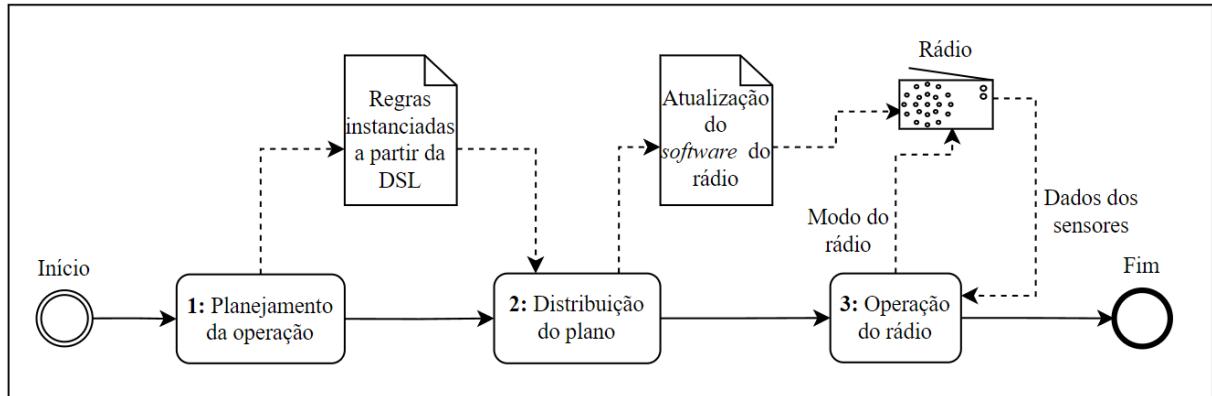


Figura 15 – Cenário pretendido da solução proposta

Essa solução permite agilizar a interoperabilidade entre o comando e o portador do rádio. Quando uma operação termina e outra começa, o grupo do comando pode mudar as regras e redistribuir uma nova versão do SW de operação do rádio, conforme a situação demande. Com esta solução os sistemas de planejamento e o rádio passam a interoperar de modo ágil. Além disso, a complexidade de operação do rádio está embutida no SW, ficando transparente para o usuário do equipamento.

5 RESULTADOS OBTIDOS

Tendo em vista os conceitos de MDE e as características de um rádio cognitivo estudadas no presente trabalho, inicialmente foi desenvolvida uma metamodelagem simplificada de um rádio cognitivo em um ambiente de operação. Esta primeira metamodelagem (Figura 16) serviu para que fosse possível descrever de uma forma simples um equipamento de rádio que possui capacidade de perceber dados do seu ambiente e do plano de comunicações de uma operação militar e desta forma tomar alguma decisão sobre o próprio modo de operação.

A partir desta metamodelagem inicial, pôde-se compreender melhor o ambiente de operação e, então, modelar a maneira com a qual os dados são percebidos e como formam regras que possam ser posteriormente verificadas pelo software do rádio (Figura 17). Depois de prontos, estes modelos podem ser revisitados de forma independente e refinados de acordo com cada contexto de interesse.

Em posse da metamodelagem da Figura 17, através da utilização do software Acceleo foi possível, após a geração de um *template* de código, a geração de uma classe Java que possui como um de seus métodos a checagem de uma dada regra instanciada pelo usuário. Vale ressaltar que o resultado da verificação das regras apenas indica uma mudança no modo de operação do aparelho, não sendo do escopo deste trabalho a interface com um rádio cognitivo real nem o *deployment* destas para o software dos rádios.

5.1 Metamodelagens

5.1.1 Rádio cognitivo

Para que fosse possível desenvolver uma arquitetura que possibilitasse modificações posteriores nos modelos, baseando-se nos conceitos centrais de um rádio cognitivo estudados e nas características deste que são mais relevantes, primeiramente foi desenvolvido o metamodelo da Figura 16. Para simplificar a análise feita no presente trabalho, mas ainda possibilitando a evolução do projeto, decidiu-se desenvolver este modelo de forma simplificada, o que é perfeitamente possível, uma vez que a arquitetura proposta se mostra relevante justamente por possibilitar que mudanças nas modelagens sejam facilmente aplicadas e propagadas até a geração de código final.

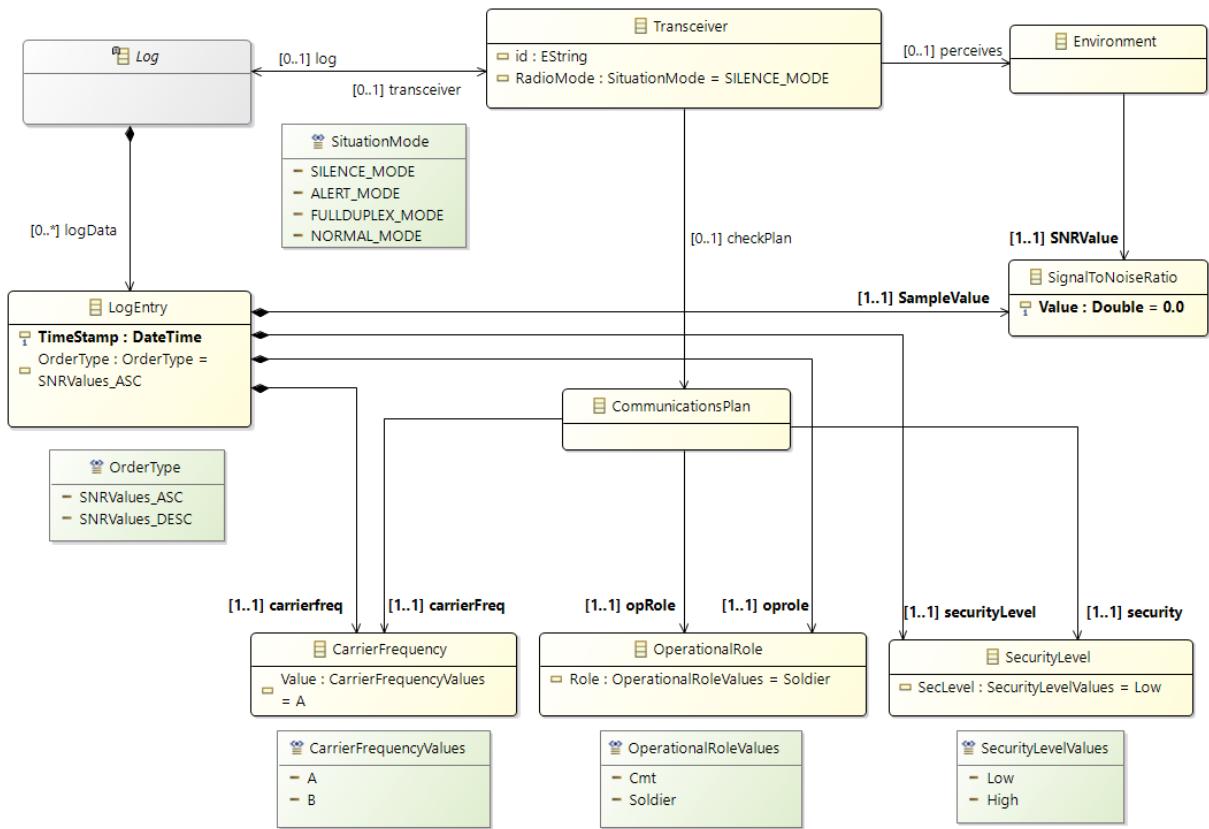


Figura 16 – Metamodelo simplificado de um rádio cognitivo em um ambiente de operação

A metamodelagem desenvolvida considera que um aparelho de rádio cognitivo é caracterizado pela classe *Transceiver* (representando o *hardware* do aparelho), a qual se relaciona com as classes *Environment*, *Communication's Plan* e *Log*. Esta última classe contém elementos da classe *Log Entry*, os quais possuem elementos das classes *Signal To Noise Ratio*, *Carrier Frequency*, *Operational Role* e *Security*, representando os dados que um rádio percebe de seu ambiente (*Signal To Noise Ratio*) e da base de dados da operação (*Carrier Frequency*, *Operational Role* e *Security*) a cada verificação dos parâmetros realizada pelo rádio.

Como foi elucidado no tópico 5.1, *Signal To Noise Ratio* pode ser entendido como o valor da relação sinal-ruído do sinal recebido. *Carrier Frequency* é a frequência do espectro utilizada para a comunicação. *Operational Role* representa o "papel operacional" do rádio, o qual pode assumir apenas os valores "comandante"(cmt) ou "soldado"(soldier). Já a classe *Security* é o nível de segurança de uma determinada operação. Com exceção do valor do SNR, estes três outros parâmetros são determinados pelo plano de comunicações da operação militar na qual o equipamento está inserido (classe *Communication's Plan*), cujos parâmetros podem ser consultados através da base de dados da operação. Logo, consideramos que estes valores são recebidos e tratados pelo software do rádio e que ficam

disponíveis para as análises.

Para realizar a leitura dos dados percebidos pelo aparelho, em posse dos dados de Log, selecionam-se apenas os três últimos valores de *Log Entry*, como uma janela deslizante. A partir desta seleção, pode-se determinar a característica do sinal (valores de SNR ascendentes, descendentes ou nenhuma das duas tendências) e juntamente com os valores dos parâmetros da última entrada do Log (os dados mais recentes percebidos), então verificar se atende a alguma das regras que mudariam o estado (modo de operação) do rádio.

5.1.2 Formação das regras

Tomando como base a metamodelagem simplificada da Figura 16, foi desenvolvido um metamodelo que representa a maneira como as regras utilizadas pelo rádio cognitivo são montadas. Cada regra é composta por um único antecedente (representado pela classe LeftSide) e um único consequente (representado pela classe RightSide).

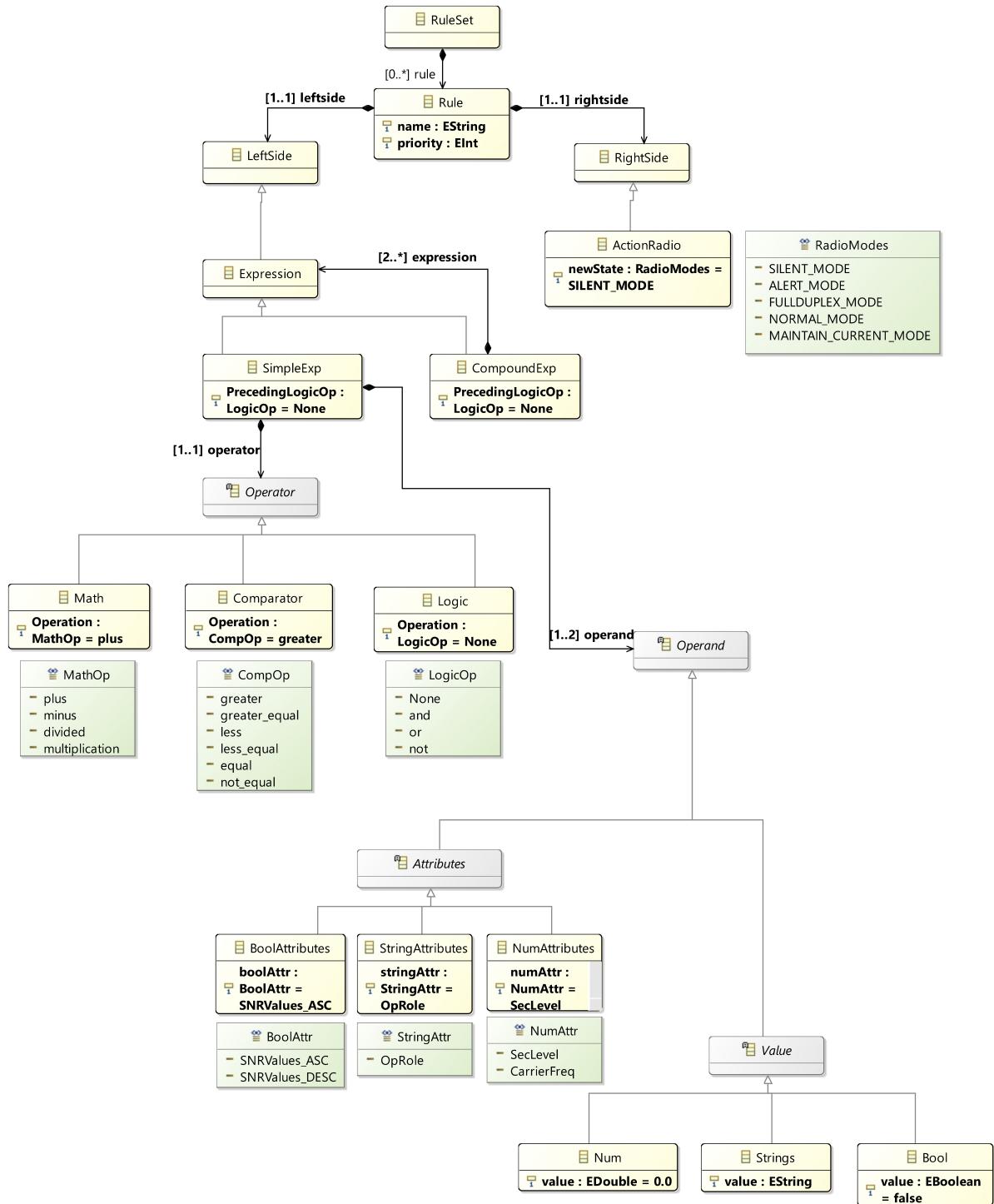


Figura 17 – Metamodelo que descreve a formação de regras de um rádio cognitivo

O lado esquerdo da regra é composto por expressões simples (classe **SimpleExp**) e expressões compostas (classe **CompoundExp**). Uma expressão simples deve ter um único operador (\$) e um ou dois operandos (P_i) ($\{P_1 \$ P_2\}$ ou $\{\$ P_2\}$), já uma expressão composta deve conter duas ou mais expressões (simples ou compostas). Tanto as expressões simples quanto as compostas devem necessariamente possuir como atributo um operador

lógico que precede a expressão completa (atributo PrecedingLogicOp), sendo que seu valor padrão é *none* (sem operador).

O lado direito da regra possui apenas o novo modo de operação do rádio, representado pelo atributo NewState da classe ActionRadio, o qual é obrigatório e único.

Os operadores podem ser dos tipos: lógico (and, or, not), comparador ($>$, $>=$, $<$, $<=$, $==$, $!=$) ou matemático ($+$, $-$, $*$, $/$). Os operandos são os nomes das variáveis que armazenam os parâmetros percebidos do ambiente e lidos da base de dados da operação pelo rádio ou valores fixos de *strings* ou *doubles*.

Com isso, podem ser formadas regras como o exemplo exposto abaixo e conforme a Figura 11:

$$\#^1 (\#^2 (P_1^2 \$_2 P_2^2) \#^3 (P_1^3 \$_3 P_2^3) \#^4 (\#^5 (P_1^5 \$_5 P_2^5) \#^6 (P_1^6 \$_6 P_2^6))) \rightarrow R_1$$

$\#^i$: Operador lógico que precede a expressão (simples ou composta) i.

$\$_i$: Operador da expressão simples i.

P_i^j : Operando da expressão simples j. O índice i indica se operando vem antes ($i=1$) ou após ($i=2$) o operador.

R_i : Consequente da regra i.

Para exemplificar a estrutura proposta acima, tomando a regra ST_2 presente na Figura 11, pode-se escrevê-la da seguinte forma, onde o quadrado em branco representa a ausência de um operador lógico antes da expressão associada:

$$\square (\square (SNRValues_DESC == true) and (\square (SecLevel < 0.5) and (CarrierFreq == 100.0) and (OpRole.equalsIgnoreCase("soldier")))) \rightarrow NormalMode$$

A arquitetura da solução proposta no presente trabalho possibilita que regras possam ser instanciadas e geradas em forma de classes Java a qualquer momento. Logo, para que seja possível extrair uma hierarquia das regras criadas dentro de um conjunto de regras (classe raiz RuleSet), estas possuem um atributo do tipo inteiro que as identifica unicamente de acordo com a prioridade com a qual devem ser testadas pelo *software* (1 é a prioridade mais alta).

Estas restrições citadas, bem como a obrigatoriedade da determinação dos valores dos atributos das classes, são checadas (validadas) durante o processo de instanciação do metamodelo na IDE do Eclipse, o que elimina parte do trabalho de verificação do usuário ao instanciar uma regra nova a partir deste metamodelo. Porém, uma instanciação correta das expressões também depende do usuário e este deve conferir se o que será gerado faz sentido neste contexto.

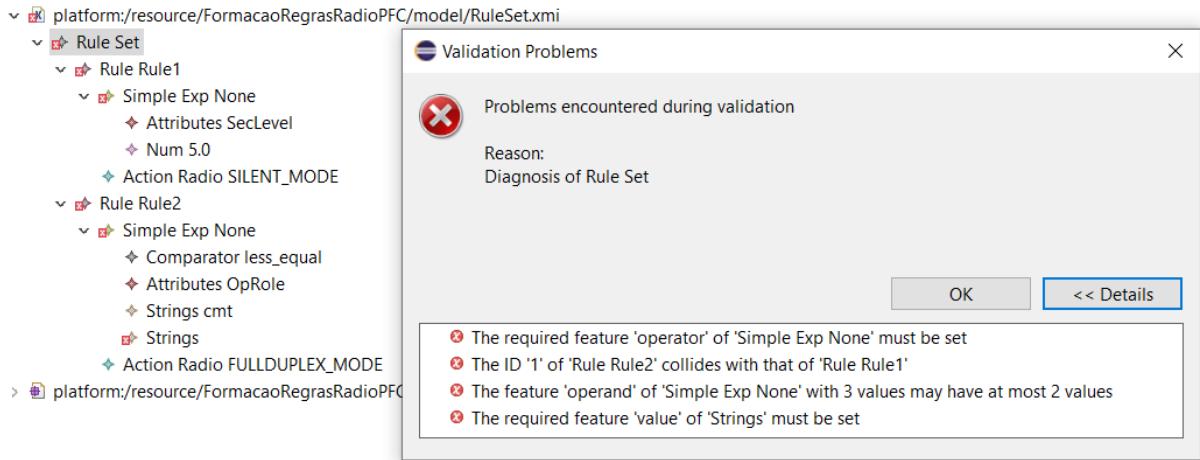


Figura 18 – Validação da instanciação das regras pela IDE do Eclipse

No exemplo da Figura 18, a ferramenta identificou automaticamente que as duas regras do conjunto de regras (*Rule Set*) gerado possuíam os mesmos valores de prioridade (atributo identificado como 'ID' da regra). Além disso, a ausência de uma instância da classe *Operator* em uma expressão simples, o limite máximo de dois operandos por expressão e a tentativa de deixar um valor obrigatório em branco também foram alertados ao usuário pela IDE.

5.2 Geração de código através do Acceleo

Primeiramente, há a instanciação de uma regra ou conjunto de regras através da interface gráfica da IDE (Anexo C) a partir do metamodelo da Figura 17, que gera um arquivo de formato XMI (Anexo B), o qual é um padrão da OMG que tem como base o XML. Depois, para que a ferramenta Acceleo possa interpretá-lo, deve-se criar um *template* de código em linguagem MTL. A partir do arquivo XMI e do *template* gerado, a ferramenta cria um arquivo da maneira especificada no modelo de código desenvolvido.

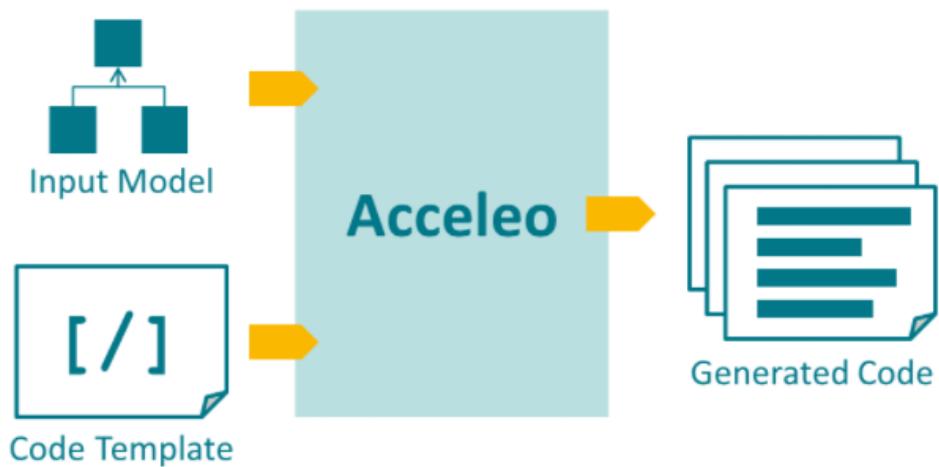


Figura 19 – Geração de código através do *software* Acceleo (7)

5.2.1 Template de código

Este *template* (ou modelo de código) permite com que possam ser inseridos no meio do arquivo gerado pela ferramenta elementos obtidos da instanciação do metamodelo. Neste arquivo, existem partes estáticas, as quais serão apenas inseridas no mesmo exatamente como estão definidas nesse *template* e partes dinâmicas, identificadas por estarem dentro de colchetes e que possibilitam a extração de informações da instância do metamodelo fornecida.

Com isso, por exemplo, pode-se criar uma classe Java para cada regra fornecida, utilizando este único arquivo de *template* ao iterar sobre os objetos da classe Rule contidos na instância da classe RuleSet escolhida.

Para exemplificar este conceito de modelo de código, o fragmento de código da Figura 20, por exemplo, mostra a iteração sobre elementos do tipo Rule da instância fornecida (Anexo B) e a criação de um arquivo com extensão ".java", cujo nome e o nome de sua classe é o atributo "name" deste elemento formatado de acordo com as boas práticas de nomeação da linguagem Java.

```
[template public generateElement(aRule : Rule) post (replaceAll('\n', '').trim()) {packageName : String = 'br.eb.ime.rules';}]
[comment @main /]
[file (aRule.name.trim().replaceAll(' ', '').toUpperCase()+' .java', false, 'UTF-8')]
package [packageName];]

/**
 *
 * Class [aRule.name.toUpperCase()]
 *
 */
public class [aRule.name.replaceAll(' ', '').trim().toUpperCase()/] {
```

Figura 20 – Trecho de código do *template* desenvolvido

Um exemplo de código gerado pela ferramenta ao fornecer o trecho de código da Figura 20 e o arquivo XMI presente no Anexo B pode ser observado no Anexo A.

A grande vantagem de utilizar a ferramenta Acceleo é justamente por possibilitar que diferentes modelos de código possam ser desenvolvidos. Ou seja, caso seja necessário criar arquivos de regras em outra linguagem de programação, ou que o código gerado faça chamada a funções específicas do *software* de um equipamento, basta desenvolver um outro *template* que atenda aos requisitos da outra tecnologia.

O código do projeto desenvolvido pode está disponibilizado em <<https://github.com/GabrielBozza/ProjetoAcceleoPFC>>. O arquivo que representa esse *template* é RadioPFC.acceleo/src/RadioPFC/Acceleo/Common/generate.mtl e também está presente no Anexo D.

5.2.2 Código Java gerado

A linguagem Java foi escolhida porque existem rádios reais cujo *software* a utiliza e por ser uma linguagem de alto nível muito disseminada, facilitando a demonstração dos conceitos abordados. Outro fator que foi levado em conta é a facilidade de trabalhar com arquivos distintos representando classes, o que é muito conveniente ao possibilitar a compartmentalização de cada regra. Também é conhecido que a linguagem C++ é muito utilizada para esses fins, mas, como bastaria apenas modificar o *template* de código, e como a linguagem Java possui as vantagens listadas acima, o presente trabalho foca na geração de código em Java.

Inicialmente, após estarmos familiarizados com a ferramenta, foi decidido que, com o intuito de verificar o correto funcionamento da estrutura desenvolvida, seriam instanciadas regras extremamente simples, cuja formatação não foi organizada de forma que fosse compatível com o ambiente de testes. Depois de aprendermos o funcionamento da ferramenta, trabalhamos com ela para gerar as regras conforme os cenários abordados na seção 5.1.

Cada classe que representa uma regra gerada possui dois métodos, um que retorna o modo de operação determinado pela aplicação da regra e outro que retorna a prioridade desta (Anexo A). A ideia é que esses arquivos sejam criados diretamente em um mesmo diretório que estará acessível pelo software de simulação.

Como próximos passos, com base no software de simulação disponibilizado pelo aluno de doutorado Maj. Marcus, tanto o *template* desenvolvido quanto o ambiente de simulação serão modificados para que sejam compatíveis, possibilitando a etapa final do presente trabalho, a qual consiste dos testes das regras geradas nesse software fornecido.

5.3 Desenvolvimento de uma DSL

Através do desenvolvimento de uma linguagem específica para o domínio do problema abordado, a instanciação das regras se resume à estruturação de um documento como mostra a Figura 21 e o Anexo F. Partindo de uma situação na qual o operador do sistema terá como base apenas uma tabela, como a da Figura 11, fica evidente a maior facilidade que este terá utilizando a abordagem da linguagem desenvolvida de introduzir regras no sistema, uma vez que a sintaxe da DSL foi desenvolvida para que fosse intuitiva e o mais próximo possível da forma com que as regras serão recebidas para serem implementadas.

Como pode ser visto na Figura 21, a qual mostra um trecho de código exemplificando a utilização da DSL desenvolvida no presente trabalho, pode-se concluir que através perspectiva de um operador do sistema com pouco conhecimento de linguagens e lógica de programação, a geração de um documento com essa estrutura é uma tarefa mais simples e sucinta do que o método utilizado anteriormente. Embora a interface gráfica proporcionada pelo *plugin OCL in Eclipse* (Anexo C) possa, a princípio, parecer mais fácil de ser utilizada por alguma pessoa leiga, a estrutura hierárquica das regras torna a instanciação por esse método extremamente massante e não intuitiva. Além disso, não há flexibilidade quanto à aparência e funcionalidades da interface, impossibilitando sua adequação customização ao contexto específico discutido.

Na DSL desenvolvida, os caracteres ‘_’ indicam que uma expressão (que pode ser simples ou composta, como mostrado na Figura 17) não possui um operador lógico que a precede. Esse símbolo é análogo ao quadrado branco explicado na seção 5.1.2 e a linguagem desenvolvida no presente trabalho foi feita com base nas expressões dessa mesma seção.

```

RuleSet{
    Rule 'Rule_ST1' {
        priority = 1
        rule = _[ _{(SNRValues_ASC=true) and [_{(SecurityLevel>=0.5)and(CarrierFrequency=50.0)and(OperationalRole="Cmt")]}]
        --> SILENT_MODE
    },
    Rule 'Rule_ST2' {
        priority = 2
        rule = _[ _{(SNRValues_DESC=true) and [_{(SecurityLevel<0.5)and(CarrierFrequency=100.0)and(OperationalRole="Soldier")]}]
        --> NORMAL_MODE
    }
}

```

Figura 21 – Utilização da DSL desenvolvida

5.3.1 Utilização do Xtext

Xtext é um dos *frameworks* de código aberto mais conhecido e utilizado para o desenvolvimento de DSLs. A ferramenta gera um analisador sintático e um modelo de classe para a árvore de sintaxe abstrata, fornecendo ainda uma interface de desenvolvimento baseada no Eclipse.

A partir de um metamodelo, o Xtext possibilita a geração de uma DSL padrão, a qual, no entanto, não é prática de ser utilizada, uma vez que tornaria o processo de instanciação das regras mais complicado. Pode-se observar no Anexo G esse aumento da complexidade ao utilizar a linguagem gerada automaticamente pela ferramenta a partir do metamodelo da Figura 17. Nessa comparação exemplifica-se como seria a instanciação de apenas uma regra tanto na linguagem gerada pela ferramenta quanto a que foi modificada. A notável diferença entre a utilização das duas linguagens advém do fato de a ferramenta ter de inferir como deve ser a sintaxe da DSL apenas tendo o metamodelo fornecido como base, resultando em um código carregado de informações que ainda podem ser omitidas ou encurtadas ao modificar a linguagem gerada (linguagem "verbosa").

O arquivo principal do projeto Xtext da DSL desenvolvida no presente trabalho está disponível no Anexo D.

6 SIMULAÇÃO E TESTES

Nesta seção é descrito um ambiente simulado, que busca caracterizar o ambiente de operação do rádio cognitivo em uma operação militar. Por meio dessa atividade é possível demonstrar a aplicação dos modelos desenvolvidos e das ferramentas descritas neste trabalho. Além disso, mostrar como essa abordagem pode ser útil ao modelo cognitivo do rádio, principalmente no que tange a agilidade que essa arquitetura oferece no processo de atualização das regras.

6.1 Descrição do ambiente de simulação

Para a fase de testes do trabalho desenvolvido, como citado anteriormente, foi utilizado um programa simulador de logs (parâmetros percebidos pelo dispositivo) desenvolvido em (19) e modificado no presente trabalho. Este simulador pode ser compreendido ao dividi-lo em três etapas principais, conforme a Figura 22, iniciando pela criação de um documento de texto com valores de logs para os testes, sendo estes gerados de maneira aleatória dentro dos valores possíveis dos parâmetros analisados (atribuição de FileMaker.java). Este arquivo de texto possui cem linhas, cada qual representando um evento de leitura dos parâmetros pelo dispositivo (percepção do ambiente). Uma linha do log possui quatro valores: valor da relação sinal ruído (SNR), frequência de transmissão (*Carrier Frequency*), nível de segurança e papel operacional do rádio.

Na etapa seguinte do processo (executada por FileLoader.java), o arquivo gerado anteriormente é lido, gerando inicialmente uma instância da classe Log, a qual é atualizada a cada leitura de uma linha dos logs. Esta classe armazena um determinado número de logs (linhas do arquivo) mais recentes, sendo esse valor determinado pelo tamanho da janela deslizante utilizada (essa janela define o número de logs mais recentes que devem ser levados em consideração para determinar o novo modo do dispositivo).

Ao ler uma linha do arquivo, esta é validada e armazenada na classe Log como um LogEntry e o log (LogEntry) mais antigo é removido. Como etapa final da leitura de uma linha, é chamada uma função que recebe esta instância da classe Log e o tamanho da janela deslizante (CheckMode.java). Como resultado da chamada desta função, são exibidos na IDE o modo que o rádio simulado deve estar a cada leitura de parâmetros do ambiente, junto ao valor da data e hora da leitura dos dados.

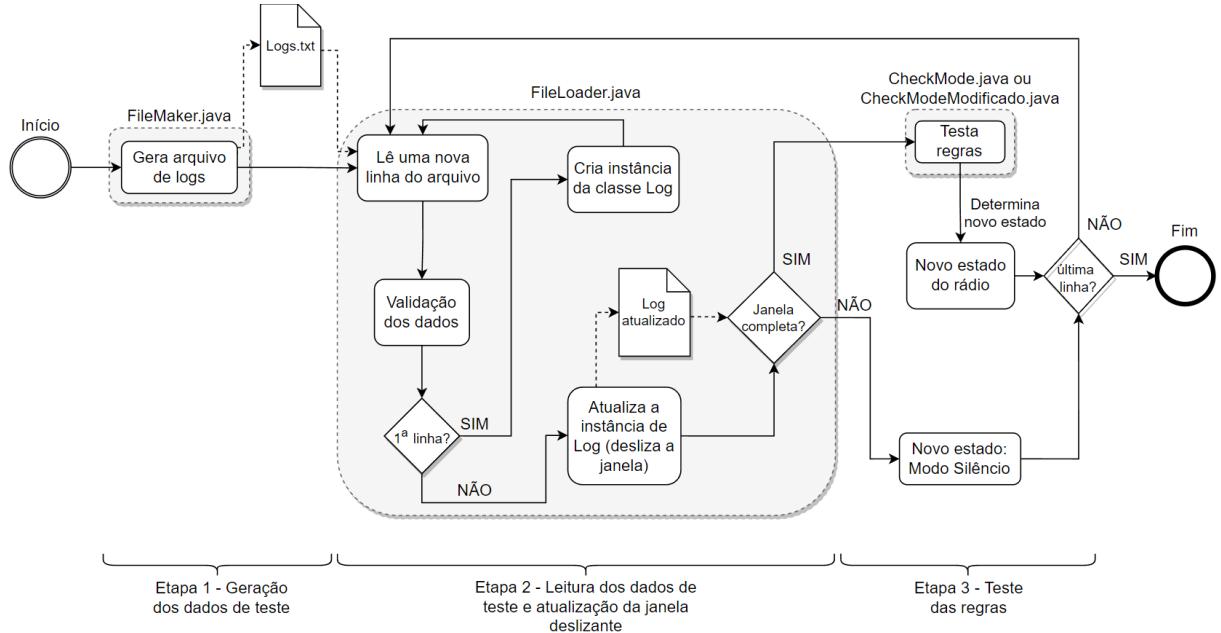


Figura 22 – Visão geral da arquitetura e das etapas do processo

6.2 Adaptação do programa simulador

A função que determina o novo modo de operação do rádio foi desenvolvida de duas maneiras. Na elaboração de regras de forma tradicional, usando cláusulas If /Then/ Else (representada por CheckMode.java), ao testar uma condição antes de outra, automaticamente é priorizada a validação de uma regra mais ou menos restritiva, em função da ordem lógica das condições. Por outro lado, na abordagem proposta neste trabalho (representada por CheckModeModificado.java), as regras são descritas individualmente e de forma não necessariamente encadeadas, como no caso anterior. Desta forma, haverá a possibilidade de duas regras serem atendidas ao mesmo tempo e a necessidade de decidir qual deverá ser cumprida. O uso da prioridade possibilita este tipo de decisão. Outros critérios de decisão poderiam ser inseridos, porém para este ambiente de teste o uso da prioridade é suficiente.

Nesta última implementação, foi criado um pacote (br.ime.rules) no projeto fornecido, o qual foi usado como local de geração das classes de regras pelo Acceleo (vide seção 6.2).

Logo, na implementação desenvolvida, para que se possa verificar as regras, primeiramente é criado um TreeMap cuja chave é a prioridade e cujo valor é o nome de uma regra. Como esta estrutura armazena suas instâncias em ordem crescente de acordo com a chave do par, basta iterar sobre o TreeMap e testar as regras (que já estão ordenadas por prioridade). As regras são testadas até que alguma delas indique uma mudança no

estado do rádio, sendo este o novo modo de operação do dispositivo. Desta maneira, a abordagem proposta garante que o estado do rádio será determinado pela regra de maior prioridade que indique uma mudança no modo de operação do rádio.

Como citado anteriormente, a utilização de uma janela deslizante possibilita que seja determinada a tendência dos valores de SNR mais recentes percebidos pelo dispositivo. No presente trabalho, para efeitos de simplificação da análise dos testes, foi utilizada uma janela de tamanho três (Figura 23). Ou seja, para determinar se os valores da relação sinal ruído percebidos estão em tendência ascendente ou descendente são levados em conta os três últimos valores lidos. Para a análise dos demais parâmetros, apenas os valores do log mais recente são considerados. Também vale ressaltar que, caso a janela ainda não esteja completa (dados insuficientes), o simulador determina, por segurança, que o estado do rádio deve ser o modo mais restrito (modo silêncio).

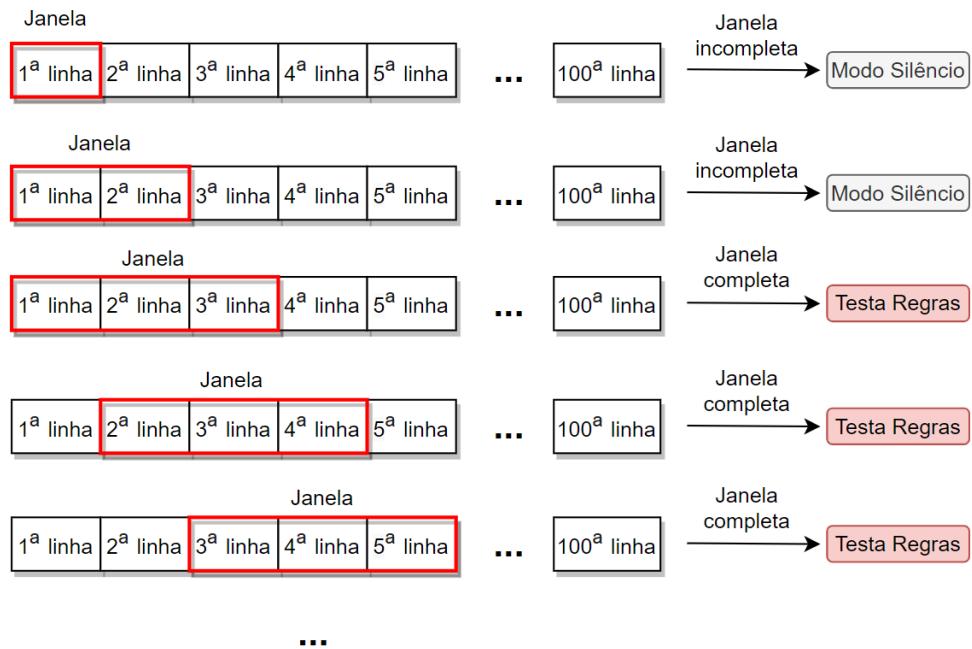


Figura 23 – Funcionamento da janela deslizante

O projeto utilizado e modificado para etapa de testes está disponível em: <<https://github.com/GabrielBozza/FileMakerV2PFC>>.

6.3 Testes

Nesta etapa, foram realizados testes do nosso sistema para verificar seu correto funcionamento, os quais englobaram comparações com os valores esperados, alterações do cenário lido pelo sistema e também alterações das regras nele embutidas. A seguir serão discutidos os resultados de cada ensaio.

6.3.1 Teste de funcionamento do sistema

Esse teste tem por objetivo confirmar que o sistema está funcionando conforme o esperado, de maneira exata. Para isso, foi executado o programa desenvolvido pelo Maj Marcus Albert, no qual o arquivo CheckMode.java foi construído por codificação rígida, utilizando a uma lógica pouco flexível (utilizando apenas aninhamentos de if/else) e os resultados obtidos foram guardados no arquivo de texto Resultado_HardCoding.txt (Anexo H).

Depois, sem gerar uma nova sequência aleatória de logs, ou seja, utilizamos os mesmos valores recebidos pelo rádio, foi executado o arquivo CheckModeModificado.java, o qual foi gerado a partir da metodologia adotada por esse trabalho (seções 6.1 e 6.2), resultando em uma sequência de log/modo de rádio definido a ser guardada no arquivo de texto Resultado_MDE.txt (Anexo I).

Por fim, com apoio da ferramenta de Excel, os dois arquivos de texto foram comparados resultando na confirmação de que, para os mesmos valores de SNR, frequência de transmissão, nível de segurança e papel operacional do rádio, em uma mesma sequência, o modo de operação do rádio a cada período de análise é o mesmo. Para verificar essa informação, basta consultar os anexos H e I, nos quais a única diferença entre os arquivos será a marca temporal (determinada pelo horário da execução), uma vez que os dois programas não foram executados simultaneamente.

6.3.2 Teste de alteração das informações recebidas pelo sistema

O segundo teste realizado visa mostrar como, baseando-se no mesmo conjunto de regras, o modo de operação do sistema é modificado na medida que as informações de log percebidas por ele também mudam. Para tanto, algumas entradas foram alteradas manualmente no arquivo de log, para verificar pontualmente as mudanças ocorridas.

A Figura 24 destaca trechos contidos nos Anexos I e J.

<terminated> Manager [Java Application] C:\Users\g-boz\OneDrive\	<terminated> Manager [Java Application] C:\Users\g-boz\OneDriv
2021-09-11T21:15:38.843 7.0 100 0.6 cmt	2021-09-11T21:10:47.855 7.0 100 0.6 cmt
SilentMode	SilentMode
2021-09-11T21:15:38.872 9.0 50 1.0 soldier	2021-09-11T21:10:47.880 9.0 50 1.0 soldier
SilentMode	SilentMode
2021-09-11T21:15:38.903 5.0 100 0.1 cmt	2021-09-11T21:10:47.912 5.0 100 0.1 cmt
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:38.935 9.0 100 0.9 soldier	2021-09-11T21:10:48.003 9.0 100 0.9 soldier
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:38.967 2.0 100 0.4 soldier	2021-09-11T21:10:48.036 2.0 100 0.4 soldier
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:38.999 3.0 100 0.6 cmt	2021-09-11T21:10:48.066 3.0 100 0.6 cmt
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:39.030 2.0 50 0.4 cmt	2021-09-11T21:10:48.097 2.0 50 0.4 cmt
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:39.063 3.0 50 0.4 soldier	2021-09-11T21:10:48.128 3.0 50 0.4 soldier
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:39.094 2.0 100 0.4 cmt	1 2021-09-11T21:10:48.160 5.0 50 0.8 soldier
MAINTAIN_CURRENT_MODE	ALERT_MODE
2021-09-11T21:15:39.126 8.0 100 0.6 soldier	2 2021-09-11T21:10:48.191 7.0 50 0.9 soldier
MAINTAIN_CURRENT_MODE	ALERT_MODE
2021-09-11T21:15:39.157 7.0 50 0.8 soldier	3 2021-09-11T21:10:48.221 9.0 50 0.7 soldier
MAINTAIN_CURRENT_MODE	ALERT_MODE
2021-09-11T21:15:39.189 9.0 50 0.9 soldier	4 2021-09-11T21:10:48.253 6.0 50 0.6 soldier
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:39.220 5.0 50 0.7 soldier	5 2021-09-11T21:10:48.284 5.0 50 0.6 soldier
MAINTAIN_CURRENT_MODE	NORMAL_MODE
2021-09-11T21:15:39.251 3.0 100 0.1 soldier	6 2021-09-11T21:10:48.315 4.0 50 0.4 soldier
NORMAL_MODE	ALERT_MODE
2021-09-11T21:15:39.281 2.0 100 0.5 cmt	7 2021-09-11T21:10:48.347 2.0 50 0.3 cmt
ALERT_MODE	ALERT_MODE
2021-09-11T21:15:39.312 2.0 100 0.1 cmt	8 2021-09-11T21:10:48.377 5.0 100 0.1 cmt
MAINTAIN_CURRENT_MODE	MAINTAIN_CURRENT_MODE
2021-09-11T21:15:39.343 3.0 100 0.6 cmt	9 2021-09-11T21:10:48.408 6.0 100 0.1 cmt
MAINTAIN_CURRENT_MODE	FULL DUPLEX MODE
2021-09-11T21:15:39.374 8.0 50 0.0 soldier	2021-09-11T21:10:48.439 8.0 50 0.0 soldier
ALERT_MODE	ALERT_MODE

(a)

(b)

Figura 24 – Sequência de logs e modo de operação resultante retirados de (a) Anexo I - Saída do sistema original (b) Anexo J - Saída do sistema com logs alterados

A seleção em vermelho na Figura 24 identifica as mudanças realizadas nos logs e as consequentes alterações nos modos de operação resultantes. Foram feitas alterações em todos os quatro campos relevantes para análise do nosso sistema. Por exemplo, quando a janela deslizante de análise chega nas linhas 1, 2 e 3 indicadas, encontramos uma sequência ascendente de SNR e, como nenhuma das regras 1, 2, 3 e 4 (Figura 11) é atendida, a regra ST_5 é aplicada, ou seja, o modo de operação muda para ALERT_MODE (modo alerta).

Analogamente, cada caso pode ser compreendido por uma das regras pré-definidas e, como os logs recebidos em (a) são diferentes dos recebidos em (b) (Figura 24), é possível notar que os modos de operação decorrentes de cada etapa também mudam. Esse resultado está em concordância com o esperado, uma vez que confirma que, dado uma alteração na percepção do sistema - considerando tanto o plano de comunicações quanto o ambiente externo -, o comportamento do sistema também muda, seguindo o padrão de regras a ele imposto.

6.3.3 Mudanças no conjunto de regras

Uma das facilidades da abordagem adotada é a adaptação do sistema de acordo com os requisitos necessários. Portanto, modificando o conjunto de regras utilizado,

também devem ser observadas mudanças nos modos de operação resultantes de cada janela deslizante. Por esse motivo, um novo teste foi executado utilizando os mesmos logs usados para gerar Resultado_MDE.txt (Anexo I), porém com as regras presentes na Figura 25.

Tipo	Object	Variáveis	Partições	ST_1	ST_2	ST_3	ST_4	ST_5	ST_6	ST_7	ST_8
Entrada	SampleSignal	(w3) SampleValue	ASC	true		true		true		false	false
			DESC		true		true		true	false	false
		(*) Security level	>=0,5	true			true				
			< 0,5		true	true					
	transceiver	(*) Carrier Frequency	50MHz	true			true				
			100MHz		true	true				true	
		(*) Operational Role	Cmt	true		true				true	
			Soldier		true		true				
		Tx mode	normal								
			padrão FD								
			Silent Mode								
			Alert_mode								
			Matém Sit_Ant								
Saída											

Figura 25 – Conjunto de regras alterado

A diferença entre os conjuntos de regras exposto na Figura 11 e o da Figura 25, é essencialmente a adição de uma nova regra. A regra que antes era ST_7 passa a ser nomeada ST_8, tendo também sua prioridade alterada, e uma nova regra ST_7 é inclusa.

Nesse novo conjunto, ST_7 possui maior prioridade do que ST_8, o que também exemplifica a importância desse parâmetro, uma vez que, se ele não existisse ou a regra ST_8 tivesse sempre precedência, a regra ST_7 poderia jamais ser atendida.

O arquivo de texto gerado como resultado pode ser encontrado em Resultado_NewRule.txt (Anexo K). Na Figura 26 é apresentado um trecho dos logs e suas respectivas mudanças nos resultados com esse novo teste (linhas destacadas para facilitar a visualização).

2021-09-10T09:41:01.248 7.0 100 0.6 cmt SilentMode 2021-09-10T09:41:01.280 9.0 50 1.0 soldier SilentMode	2021-09-15T10:07:23.021 7.0 100 0.6 cmt SilentMode 2021-09-15T10:07:23.050 9.0 50 1.0 soldier SilentMode
2021-09-10T09:41:01.311 5.0 100 0.1 cmt MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.081 5.0 100 0.1 cmt FULLDUPLEX_MODE
2021-09-10T09:41:01.342 9.0 100 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.371 2.0 100 0.4 soldier MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.113 9.0 100 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.143 2.0 100 0.4 soldier MAINTAIN_CURRENT_MODE
2021-09-10T09:41:01.402 3.0 100 0.6 cmt MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.173 3.0 100 0.6 cmt FULLDUPLEX_MODE
2021-09-10T09:41:01.434 2.0 50 0.4 cmt MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.465 3.0 50 0.4 soldier MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.204 2.0 50 0.4 cmt MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.235 3.0 50 0.4 soldier MAINTAIN_CURRENT_MODE
2021-09-10T09:41:01.493 2.0 100 0.4 cmt MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.266 2.0 100 0.4 cmt FULLDUPLEX_MODE
2021-09-10T09:41:01.523 8.0 100 0.6 soldier MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.556 7.0 50 0.8 soldier MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.588 9.0 50 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.617 5.0 50 0.7 soldier MAINTAIN_CURRENT_MODE 2021-09-10T09:41:01.649 3.0 100 0.1 soldier NORMAL_MODE 2021-09-10T09:41:01.680 2.0 100 0.5 cmt ALERT_MODE	2021-09-15T10:07:23.297 8.0 100 0.6 soldier MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.328 7.0 50 0.8 soldier MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.358 9.0 50 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.389 5.0 50 0.7 soldier MAINTAIN_CURRENT_MODE 2021-09-15T10:07:23.420 3.0 100 0.1 soldier NORMAL_MODE 2021-09-15T10:07:23.451 2.0 100 0.5 cmt ALERT_MODE
2021-09-10T09:41:01.711 2.0 100 0.1 cmt MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.482 2.0 100 0.1 cmt FULLDUPLEX_MODE
2021-09-10T09:41:01.743 3.0 100 0.6 cmt MAINTAIN_CURRENT_MODE	2021-09-15T10:07:23.514 3.0 100 0.6 cmt FULLDUPLEX_MODE

(a)

(b)

Figura 26 – Sequência de logs e modo de operação resultante retirados de (a) Anexo I - Saída do sistema original (b) Anexo K - Saída do sistema com novo conjunto de regras

6.3.4 Testes limite

Com a finalidade de perceber possíveis falhas no sistema - que podem aparecer durante o trabalho do usuário - e como o mesmo se comporta na presença delas, também foram realizados testes limite.

Entre eles, foi executado o programa com o mesmo conjunto de logs do original, mas sem nenhuma regra, ou seja, sem que o comportamento do sistema seja determinado previamente segundo diretrizes. O resultado obtido neste experimento está no Resultado_SemRegras.txt (Anexo L) e um trecho desse documento foi retirado e está representado na Figura 27.

```
2021-09-23T11:29:06.105 7.0 100 0.6 cmt
SilentMode
2021-09-23T11:29:06.129 9.0 50 1.0 soldier
SilentMode
2021-09-23T11:29:06.160 5.0 100 0.1 cmt
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.192 9.0 100 0.9 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.223 2.0 100 0.4 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.255 3.0 100 0.6 cmt
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.287 2.0 50 0.4 cmt
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.317 3.0 50 0.4 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.349 2.0 100 0.4 cmt
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.380 8.0 100 0.6 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.411 7.0 50 0.8 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.443 9.0 50 0.9 soldier
MAINTAIN_CURRENT_MODE
2021-09-23T11:29:06.474 5.0 50 0.7 soldier
MAINTAIN_CURRENT_MODE
```

Figura 27 – Sequência de logs e modo de operação resultante retirados do Anexo L - Saída do sistema sem regras determinadas

Nota-se que, na Figura 27, os dois primeiros modos de operação resultantes são “Silent_Mode”, uma vez que esse é o modo padrão de início do sistema e a janela deslizante só começa a ser observada a partir da leitura de três logs, momento no qual, como não há regras a serem analisadas, o modo de operação se mantém inalterado.

O outro teste limite foi avaliado mediante erros nos logs. A Figura 28 (a) indica quais os casos considerados e a Figura 28 (b) mostra um trecho de quais os resultados obtidos, também presentes no Resultados_Erros.txt (Anexo M).

<pre> 1 7 100 0.6 cmt 2 9 50 1.0 soldier 3 5 100 0.1 cmt 4 9 100 0.9 soldier 5 2 100 0.4 soldier 6 3 100 0.6 cmt 7 8 2 50 0.4 cmt 9 @#7 50 0.4 soldier 10 2 10*%0 0.4 cmt 11 8 100 0.6 W@DFF 12 7 50 0.-L~8 soldier 13 9 50 0.9soldier 14 5 100 soldier 15 0.1 soldier 16 2 100 0.5 cmt 2 100 0.1 cmt 17 3 100 0.6 cmt 18 8 50 0.0 soldier 19 5 50 0.7 cmt 20 4 100 1.0 soldier 21 8 100 0.9 soldier 22 3 50 0.9 soldier 23 4 100 0.8 cmt </pre>	<pre> 2021-09-23T12:53:30.626 7.0 100 0.6 cmt SilentMode 2021-09-23T12:53:30.656 9.0 50 1.0 soldier SilentMode 2021-09-23T12:53:30.688 5.0 100 0.1 cmt MAINTAIN_CURRENT_MODE 2021-09-23T12:53:30.719 9.0 100 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-23T12:53:30.750 2.0 100 0.4 soldier MAINTAIN_CURRENT_MODE 2021-09-23T12:53:30.782 3.0 100 0.6 cmt 6 MAINTAIN_CURRENT_MODE 2021-09-23T12:53:30.845 2.0 50 0.4 cmt 8 MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.095 2.0 100 0.5 cmt 16 MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.126 3.0 100 0.6 cmt MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.156 8.0 50 0.0 soldier ALERT_MODE 2021-09-23T12:53:31.187 5.0 50 0.7 cmt MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.218 4.0 100 1.0 soldier ALERT_MODE 2021-09-23T12:53:31.249 8.0 100 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.279 3.0 50 0.9 soldier MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.311 4.0 100 0.8 cmt MAINTAIN_CURRENT_MODE 2021-09-23T12:53:31.342 8.0 50 0.1 soldier ALERT_MODE 2021-09-23T12:53:31.373 4.0 100 0.7 cmt MAINTAIN_CURRENT_MODE </pre>
--	--

(a)

(b)

Figura 28 – (a) Sequência de logs com erros; (b) Sequência de logs e modo de operação resultantes retirados do Anexo M - Saída do sistema com Log inicial com erros

As situações limite estão retratadas na Figura 28 (a) estão nas linhas 7 - sem receber entrada de log -, 9, 10, 11 e 12 - logs com lixo em cada um dos parâmetros analisados -, 13 - falta de separação entre dois parâmetros-, 14 e 15 - faltando um ou dois parâmetros - e, por fim, linha 16, na qual dois logs diferentes foram concatenados.

Com base no resultado presente na Figura 28 (b), pode-se afirmar que o sistema foi desenvolvido de forma a ignorar linhas de logs vazias, com lixo, faltando parâmetros ou espaçamento entre os mesmos, o que pode ser observado uma vez que no lado esquerdo da Figura 28, se considerarmos as linhas de 1 até 15, apenas os log correspondentes às linhas 6 e 8 são levados em conta para a análise, gerando algum resultado. Sobre o erro presente na linha 16, o sistema entende e verifica os 4 primeiros parâmetros e ignora o restante.

7 COMPARAÇÃO COM OUTRAS FERRAMENTAS

No presente projeto, a abordagem de MDE é utilizada para facilitar a formação de regras a serem inseridas em um rádio cognitivo, para que o mesmo possa atuar de diferentes formas, dependendo do ambiente e sem a necessidade de um operador. Porém, existem outras ferramentas que poderiam ser utilizadas com essa finalidade, nesta seção serão avaliadas duas delas, *Easy Rules*¹ e Rulebook², ambas baseadas na linguagem java.

Após analisar exemplos dessas duas ferramentas, pôde-se observar que a *Easy Rules* não utiliza e nem permite o desenvolvimento de uma nova DSL, o que dificulta a escrita das regras, ficando pouco intuitivo para pessoas que não saibam programar. Essa abordagem se encaixa melhor na resolução de problemas mais genéricos e nos quais a simplificação da linguagem de programação utilizada não é crucial para sua utilização.

Quanto ao Rulebook, o recurso utiliza a classe “Facts”, formada por dados fornecidos às regras - equivalente ao “Log” apresentado. Além disso, emprega uma DSL, mas que é construída especificamente para essa ferramenta, então, para utilizá-la, é necessário se familiarizar com como as regras são escritas nesse caso, limitando o uso para leigos na linguagem Java.

Portanto, é possível perceber que, mesmo com outras alternativas mais consolidadas no mercado para a geração de regras, a técnica de MDE é vantajosa por permitir um encadeamento dos processos, então, caso algo mude na modelagem do problema, não será necessário reformulá-lo por inteiro, uma vez que essa abordagem possibilita que os diferentes modelos desenvolvidos sejam apenas ajustados, sendo apoiada por inúmeras ferramentas de código aberto que automatizam a propagação de modificações entre esses artefatos. Além disso, a estratégia MDE possibilita a geração de código em várias linguagens diferentes (podendo gerar simultaneamente códigos nessas linguagens), não se limitando a uma única linguagem específica.

Também, a utilização da DSL planejada para essa problemática em particular ajuda muito para que uma pessoa inexperiente em programação possa montar o conjunto de regras mais facilmente, já que ela é bem intuitiva e semelhante a escrever uma regra normalmente.

¹ <<https://www.jeasy.org>>

² <<http://deliveredtechnologies.com/>>

8 CONCLUSÃO E PROPOSTAS PARA PROJETOS FUTUROS

Como conclusão do presente projeto foi apresentada uma ferramenta que permite a instanciação simplificada de regras a serem utilizadas por um sistema que simula um rádio cognitivo, de modo que o mesmo possa alterar seu modo de operação em tempo de execução, de acordo com o plano de operações e o meio ambiente no qual se encontra.

Foi verificado ainda que, ao utilizar a técnica de engenharia dirigida a modelos, é possível analisar o problema desde um nível de abstração mais alto até sua implementação em código fonte através dos artefatos (modelos) distintos criados, o que facilita a evolução do projeto e possibilita a utilização de ferramentas que automatizam parte das transformações entre esses modelos, ressaltando também a importância de conhecer a problemática como um todo. No contexto abordado, foi imprescindível entender como os parâmetros considerados afetam o modo de operação do dispositivo, para só então buscar a solução de como o sistema seria desenvolvido.

Ademais, considera-se relevante apontar possibilidades de trabalhos a serem elaborados com base no projeto apresentado. Seria interessante um estudo mais aprofundado a respeito de critérios de prioridade e desempate das regras, de modo a solucionar algum conflito que pode ser gerado entre elas de maneira mais otimizada do que a utilização do atributo de prioridade proposto.

Além disso, vale lembrar que a abordagem utilizada neste projeto é baseada em especificação, na qual um especialista de conhecimento previamente define as regras a serem utilizadas pelo rádio na etapa de planejamento. Portanto, outro avanço a ser deliberado é a mudança dessa técnica para o emprego de aprendizado de máquina e inteligência artificial, com a qual o rádio seria capaz de aprender para qual modo de operação deve alterar para otimizar as comunicações.

Por fim, destaca-se que seria interessante amadurecer o projeto até viabilizar sua implementação e *deployment* reais em rádios definidos por software, principalmente o desenvolvido pelo exército - que é a inspiração para este trabalho. Neste contexto deve-se analisar também o nível de segurança do sistema desenvolvido, tanto nas etapas de desenvolvimento do projeto, quanto na instanciação das regras e principalmente na distribuição do código aos dispositivos através de uma rede local segura.

REFERÊNCIAS

- 1 BARROS, L. G. *O Radio Definido por Software*. 53 p. Monografia (Graduação) — Faculdade de Tecnologia, Universidade de Brasilia, Brasilia - DF, 2017.
- 2 JANSON, J. *Radio Definido por Software: Estudo e Realização de Teste com uma Plataforma Livre*. 59 p. Monografia (Graduação) — Centro Federal de Educação Tecnologica de Santa Catarina, São Jose - SC, 2012.
- 3 SILVA, W. S.; CORDEIRO, J. R. S.; MACEDO, D. F.; VIEIRA, M.; VIEIRA, L.; NOGUEIRA, J. M. S. Introdução a rádios definidos por software com aplicações em gnu radio. *Minicursos do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos—SBRC*, v. 1, p. 216–265, 2015.
- 4 III, J. An integrated agent architecture for software defined radio. *Ph.D. Dissertation, KTH*, 07 2000.
- 5 VIEIRA, M. A.; CARVALHO, S. Model-driven engineering in the development of ubiquitous applications: Technologies, tools and languages. In: . [S.l.: s.n.], 2017. p. 29–32.
- 6 COSTA, P. D.; MIELKE, I. T.; PEREIRA, I.; ALMEIDA, J. P. A. A model-driven approach to situations: Situation modeling and rule-based situation detection. In: *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*. Vitória: [s.n.], 2012. p. 154–163.
- 7 ACCELEO Overview. <<https://www.eclipse.org/acceleo/overview.html>>. Accessed: 2021-06-20.
- 8 MOREIRA, J.; PIRES, L. F.; COSTA, P. Towards ontology-driven situation-aware disaster management. *Applied ontology*, Preprint, p. 1–15, 12 2015.
- 9 (JTNC), J. T. N. C. Software communications architecture specification <sca 4.1 draft issues adjudication>. 2015.
- 10 GALDINO, J.; MOURA, D.; MORAES, R.; SILVA, F. da; MARQUES, E.; JUNIOR, N. Introdução ao desenvolvimento de rádios definidos por software para aplicações de defesa. In: . Rio de Janeiro: [s.n.], 2012.
- 11 CTEX. *Radio Definido por Software de Defesa (RDS-DEFESA)*. 2020. Disponível em: <<http://www.ctex.eb.mil.br/projetos-em-andamento/84-radio-definido-por-software-rds>>.
- 12 MITOLA, J. Chapter 14 - cognitive radio architecture. In: FETTE, B. A. (Ed.). *Cognitive Radio Technology (Second Edition)*. Second edition. Oxford: Academic Press, 2009. p. 429–482. ISBN 978-0-12-374535-4. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B978012374535400014X>>.
- 13 MENEZES, A. S. *Avaliação de Desempenho de Rádios Cognitivos e Proposta de Estrutura de Equalização Temporal em Sistemas OFDM*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2007.

- 14 CAMILO, M. J.; MOURA, D. F. C.; SALLES, R. M. Redes de comunicações militares: desafios tecnológicos e propostas para atendimento dos requisitos operacionais do exército brasileiro. *Revista Militar De Ciência E Tecnologia*, v. 37, n. 3, 2020. Disponível em: <<http://ebrevistas.eb.mil.br/CT/article/view/6910>>.
- 15 BRASIL; Estado Maior do Exército; EB70-MC-10.246 - Manual de Campanha - AS COMUNICAÇÕES NAS OPERAÇÕES, Estado Maior do Exército, Brasília, 2020.
- 16 Rodrigues da Silva, A. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems Structures*, v. 43, p. 139–155, 2015. ISSN 1477-8424. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1477842415000408>>.
- 17 BEZIVIN, J.; JOUAULT, F.; TOUZET, D. Principles, standards and tools for model engineering. In: . [S.l.: s.n.], 2005. p. 28 – 29. ISBN 0-7695-2284-X.
- 18 NETO, D. J. S. *Estudo e avaliação de redes sem fio Full-Duplex*. 49 p. Monografia (Graduação) — Curso de Eletronica e de Computação, Escola Politecnica da Universidade Federal do Rio de Janeiro, Rio de Janeiro - RJ, 2016.
- 19 SILVA, M. A. A. da. *Interoperabilidade e Modelagem Cognitiva de Rádios em Sistemas de Comando e Controle, baseadas em Ontologias e Engenharia Guiada a Modelos*. Tese (Doutorado) — Instituto Militar de Engenharia, em andamento.

ANEXO A – EXEMPLO DE REGRA GERADA ATRAVÉS DA TRANSFORMAÇÃO M2T

```
1 package br.eb.ime.rules;
2
3 /**
4 *
5 * Class RuleST1
6 *
7 */
8
9 public class RuleST1 {
10
11     static Integer RulePriority = 1;
12
13     public static String checkMode(String CurrentMode, boolean
14         SNRValues_ASC, boolean SNRValues_DESC, double SecLevel, double
15         CarrierFreq, String OpRole){
16
17         final String NewMode;
18
19         if(((SNRValues_ASC==true)&&((SecLevel>=0.5)&&(CarrierFreq==50.0)&&
20             OpRole.equalsIgnoreCase("cmt")))){
21             NewMode = "SILENT_MODE";
22             return NewMode;
23         }
24     }
25
26     public static Integer getRulePriority(){
27         return RulePriority;
28     }
29 }
```

ANEXO B – ARQUIVO XMI GERADO PELA INSTÂNCIAÇÃO DO METAMODELO DE FORMAÇÃO DE REGRAS

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <action:RuleSet
3   xmi:version="2.0"
4   xmlns:xmi="http://www.omg.org/XMI"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xmlns:action="http://action/1.0"
7   xsi:schemaLocation="http://action/1.0 formacaoRegrasRadioPFC.ecore">
8 <rule name="RuleST2"
9   priority="2">
10  <leftside
11    xsi:type="action:CompoundExp"
12      PrecedingLogicOp="None">
13    <expression
14      xsi:type="action:SimpleExp"
15      PrecedingLogicOp="None">
16      <operator
17        xsi:type="action:Comparator"
18        Operation="equal"/>
19      <operand
20        xsi:type="action:Attributes"
21        name="SNRValues_DESC"/>
22      <operand
23        xsi:type="action:Strings"
24        value="true"/>
25    </expression>
26    <expression
27      xsi:type="action:CompoundExp"
28      PrecedingLogicOp="and">
29    <expression
30      xsi:type="action:SimpleExp"
31      PrecedingLogicOp="None">
32      <operator
33        xsi:type="action:Comparator"
34        Operation="less"/>
35      <operand
36        xsi:type="action:Attributes"
37        name="SecLevel"/>
38      <operand
39        xsi:type="action:Num"
40        value="0.5"/>
41    </expression>
```

```
42     <expression
43         xsi:type="action:SimpleExp"
44         PrecedingLogicOp="and">
45         <operator
46             xsi:type="action:Comparator"
47             Operation="equal"/>
48         <operand
49             xsi:type="action:Attributes"
50             name="CarrierFreq"/>
51         <operand
52             xsi:type="action:Num"
53             value="100.0"/>
54     </expression>
55     <expression
56         xsi:type="action:SimpleExp"
57         PrecedingLogicOp="and">
58         <operator
59             xsi:type="action:Comparator"
60             Operation="equal"/>
61         <operand
62             xsi:type="action:Attributes"
63             name="OpRole"/>
64         <operand
65             xsi:type="action:Strings"
66             value="soldier"/>
67         </expression>
68     </expression>
69   </leftside>
70   <rightside
71       xsi:type="action:ActionRadio"
72       newState="NORMAL_MODE"/>
73   </rule>
74 </action:RuleSet>
```

ANEXO C – INTERFACE GRÁFICA DA IDE ECLIPSE PARA A INSTANCIACÃO DO METAMODELO DE FORMAÇÃO DE REGRAS

```
<> platform:/resource/FormacaoRegrasRadioPFC/model/RuleSet1.xmi
    <> Rule Set
        <> Rule RuleST2
            <> Compound Exp None
                <> Simple Exp None
                    <> Comparator equal
                    <> Attributes SNRValues_DESC
                    <> Strings true
            <> Compound Exp and
                <> Simple Exp None
                    <> Comparator less
                    <> Attributes SecLevel
                    <> Num 0.5
                <> Simple Exp and
                    <> Comparator equal
                    <> Attributes CarrierFreq
                    <> Num 100.0
                <> Simple Exp and
                    <> Comparator equal
                    <> Attributes OpRole
                    <> Strings soldier
            <> Action Radio NORMAL_MODE
> platform:/resource/FormacaoRegrasRadioPFC/model/formacaoRegrasRadioPFC.ecore
```

Figura 29 – Interface gráfica para a instanciação do metamodelo

ANEXO D – MODELO DE CÓDIGO DESENVOLVIDO

```

1 [comment encoding = UTF-8 /]
2 [module generate('http://action/1.0')]
3
4 [query public className(aRule: Rule) : String = 'RadioPFC.'.concat(aRule
5     .name.replaceAll(' ', '_').toUpperCase()) /]
6
7 [template public generateElement(aRule : Rule) post (replaceAll('\n', '_')
8     .trim()) {packageName : String = 'br.eb.ime.rules';}]
9 [comment @main /]
10 [file (aRule.name.trim().replaceAll(' ', '_').toUpperCase() + '.java',
11     false, 'UTF-8')]
12 package [packageName];
13
14 public class [aRule.name.replaceAll(' ', '_').trim().toUpperCase()]{}
15
16     static Integer RulePriority = [aRule.priority];
17
18     public static String checkMode(String CurrentMode, boolean
19         SNRValues_ASC, boolean SNRValues_DESC, double SecLevel, double
20         CarrierFreq, String OpRole){
21
22         final String NewMode;
23
24         if([loopExp(aRule.leftside->filter(Expression))]){
25             NewMode = [NewState(aRule.rightside->filter(ActionRadio))]/]
26             return NewMode;
27         }
28         else{
29             return "MAINTAIN_CURRENT_MODE";
30         }
31     }
32
33     public static Integer getRulePriority(){
34         return RulePriority;
35     }
36
37     [/file]
38     [/template]
39
40     [template public condition(o : Operator) post (replaceAll('\n', '_')
41         .replaceAll(' ', '_').trim())]
42     [for (c: Comparator | o->filter(Comparator)) separator('_')]

```

```

38 [if (c.Operation.toString()=='equal')] ==
39 [elseif (c.Operation.toString()=='not_equal')] !=
40 [elseif (c.Operation.toString()=='greater_equal')] >=
41 [elseif (c.Operation.toString()=='greater')] >
42 [elseif (c.Operation.toString()=='less')] <
43 [elseif (c.Operation.toString()=='less_equal')] <= [/if]
44 [/for]
45 [for (m: Math | o->filter(Math)) separator(',')]
46 [if (m.Operation.toString()=='plus')] +
47 [elseif (m.Operation.toString()=='minus')] -
48 [elseif (m.Operation.toString()=='divided')] /
49 [elseif (m.Operation.toString()=='multiplication')] *
50 [/if]
51 [/for]
52 [for (l: Logic | o->filter(Logic)) separator(',')]
53 [if (l.Operation.toString()=='and')] &&
54 [elseif (l.Operation.toString()=='or')] ||
55 [elseif (l.Operation.toString()=='not')] !
56 [elseif (l.Operation.toString()=='None')] [/if]
57 [/for]
58 [/template]
59
60 [template public logicop(l : LogicOp) post (replaceAll('\n', ',').
       replaceAll(' ', ',').trim())]
61 [if (l.toString()=='and')] &&
62 [elseif (l.toString()=='or')] ||
63 [elseif (l.toString()=='not')] !
64 [elseif (l.toString()=='None')] [/if]
65 [/template]
66
67 [template public NewState(n :Set(ActionRadio)) post (replaceAll('\n', ',').
       trim())]
68 "[n->asSequence()->at(1).newState/]";
69 [/template]
70
71 [template public SimpleExp(l: LogicOp, ba :Set(BoolAttributes), sa :Set(
      StringAttributes), na :Set(NumAttributes), o: Operator, s:Set(Strings)
      , n:Set(Num), b:Set(Bool)) post (replaceAll('\n', ',').replaceAll(' ', ',').
      trim())]
72 [if (s->asSequence()->at(1).value.toString().equalsIgnoreCase('invalid')
      and b->asSequence()->at(1).value.toString().equalsIgnoreCase('
      invalid') and not(n->asSequence()->at(1).value.toString().
      equalsIgnoreCase('invalid')))]
73 [logicop(l)/][na.numAttr/][condition(o)/][n->asSequence()->at(1).value
      /]
74 [elseif (not(b->asSequence()->at(1).value.toString().equalsIgnoreCase('
      invalid')))]

```

```
75 [logicop(1)/][ba.boolAttr/][condition(o)/][b->asSequence()->at(1).value  
    /])  
76 [elseif (not(s->asSequence()->at(1).value.toString().equalsIgnoreCase('  
        invalid')) and o->filter(Comparator)->asSequence()->at(1).Operation.  
        toString()=='equal')]  
77 [logicop(1)/][sa.stringAttr/].equalsIgnoreCase("[s->asSequence()->at(1)  
    .value.toString()/]")]  
78 [elseif (not(s->asSequence()->at(1).value.toString().equalsIgnoreCase('  
        invalid')) and o->filter(Comparator)->asSequence()->at(1).Operation.  
        toString()=='not_equal')]  
79 [logicop(1)/](![sa.stringAttr/].equalsIgnoreCase("[s->asSequence()->at  
    (1).value.toString()/]"))  
80 [elseif (s->asSequence()->at(1).value.toString().equalsIgnoreCase('  
        invalid'))]  
81 [logicop(1)/][condition(o)/][ba.boolAttr/])  
82 [/if]  
83 [/template]  
84  
85 [template public CompoundExp(c : CompoundExp) post (replaceAll('\n', '')  
    .replaceAll(' ', '').trim())]  
86 [logicop(c.PrecedingLogicOp)/]  
87 [for (Exp: Expression | c.expression) separator('')]  
88 [for (simpleExp: SimpleExp | Exp->filter(SimpleExp)) separator('')]  
89 [SimpleExp(simpleExp.PrecedingLogicOp,simpleExp.operand->filter(  
    BoolAttributes),simpleExp.operand->filter(StringAttributes),simpleExp  
    .operand->filter(NumAttributes),simpleExp.operator,simpleExp.operand  
    ->filter(Strings),simpleExp.operand->filter(Num),simpleExp.operand->  
    filter(Bool))/]  
90 [/for]  
91 [for (compoundExp: CompoundExp | Exp->filter(CompoundExp)) separator('')  
    ]  
92 [CompoundExp(compoundExp)/]  
93 [/for]  
94 [/for])  
95 [/template]  
96  
97 [template public loopExp(e : Set(Expression)) post (replaceAll('\n', '')  
    .replaceAll(' ', '').trim())]  
98 [for (Exp: Expression | e) separator('')]  
99 [for (simpleExp: SimpleExp | Exp->filter(SimpleExp)) separator('')]  
100 [SimpleExp(simpleExp.PrecedingLogicOp,simpleExp.operand->filter(  
    BoolAttributes),simpleExp.operand->filter(StringAttributes),simpleExp  
    .operand->filter(NumAttributes),simpleExp.operator,simpleExp.operand  
    ->filter(Strings),simpleExp.operand->filter(Num),simpleExp.operand->  
    filter(Bool))/]  
101 [/for]  
102 [for (compoundExp: CompoundExp | Exp->filter(CompoundExp)) separator('')]
```

```
        ]  
103 [CompoundExp(compoundExp)]  
104 [/for]  
105 [/for]  
106 [/template]
```

ANEXO E – ARQUIVO XTEXT MODIFICADO PARA A GERAÇÃO DA DSL

```

1 grammar ime.xtext.radioDsl.Dsl with org.eclipse.xtext.common.Terminals
2
3 import "http://action/1.0"
4 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
5
6
7 RuleSet returns RuleSet:
8   {RuleSet}
9   'RuleSet'
10  '{'
11    (rule+=Rule ( "," rule+=Rule)* )?
12  '}';
13
14
15 LeftSide returns LeftSide:
16   SimpleExp | CompoundExp;
17
18 RightSide returns RightSide:
19   ActionRadio;
20
21 Operator returns Operator:
22   Logic | Comparator | Math;
23
24 OperatorNum returns Operator:
25   Comparator | Math;
26
27 OperatorString returns Operator:
28   ComparatorString;
29
30 OperatorBool returns Operator:
31   LogicMiddle | ComparatorBool;
32
33 OperatorBoolBefore returns Operator:
34   LogicBefore;
35
36 Operand returns Operand:
37   Num | Strings| Bool | BoolAttributes | StringAttributes |
38     NumAttributes ;
39
40 OperandNum returns Operand:
41   Num | NumAttributes ;

```

```
41
42 OperandBool returns Operand:
43   Bool | BoolAttributes;
44
45 OperandString returns Operand:
46   Strings | StringAttributes;
47
48 Expression returns Expression:
49   SimpleExp | CompoundExp;
50
51 Rule returns Rule:
52   'Rule'
53   name=EString
54   '{'
55   'priority' + '=' + priority=EInt
56   'rule' + '=' + leftside=LeftSide
57   '-->' rightside=RightSide
58   '}';
59
60 SimpleExp returns SimpleExp:
61   PrecedingLogicOp=LogicOp
62   '('
63   (operand+=OperandNum operator=OperatorNum operand+=OperandNum) ')'
64   | PrecedingLogicOp=LogicOp
65   '('
66   (operand+=OperandString operator=OperatorString operand+=
67   OperandString) ')'
68   | PrecedingLogicOp=LogicOp
69   '('
70   (operand+=OperandBool operator=OperatorString operand+=OperandBool)
71   ')'
72   | PrecedingLogicOp=LogicOp
73   '('
74   (operator=OperatorBoolBefore operand+=OperandBool) ')';
75
76 CompoundExp returns CompoundExp:
77   PrecedingLogicOp=LogicOp
78   '['
79   expression+=Expression ( expression+=Expression)*
80   ']';
81
82 enum LogicOp returns LogicOp:
83   None = '_'
84   and = 'and'
85   or = 'or'
86   not = 'not';
87
88 enum LogicOpMiddle returns LogicOp:
89   and = 'and'
90   or = 'or';
```

```
86 enum LogicOpBefore returns LogicOp:
87     None = '_' | not = 'not';
88
89 Logic returns Logic:
90     Operation=LogicOp;
91
92 LogicMiddle returns Logic:
93     Operation=LogicOpMiddle;
94
95 LogicBefore returns Logic:
96     Operation=LogicOpBefore;
97
98 Comparator returns Comparator:
99     Operation=CompOp;
100
101 ComparatorString returns Comparator:
102     Operation=CompOpString;
103
104 ComparatorBool returns Comparator:
105     Operation=CompOpBool;
106
107 Math returns Math:
108     Operation=MathOp;
109
110 enum CompOp returns CompOp:
111     greater = '>' | greater_equal = '>=' | less = '<' | less_equal = '<='
112         | equal = '=' | not_equal = '!=';
113
114 enum CompOpString returns CompOp:
115     equal = '=' | not_equal = '!=';
116
117 enum CompOpBool returns CompOp:
118     equal = '=' | not_equal = '!=';
119
120 enum MathOp returns MathOp:
121     plus = '+' | minus = '-' | divided = '/' | multiplication = '*';
122
123 EString returns ecore::EString:
124     STRING | ID;
125
126 EInt returns ecore::EInt:
127     INT;
128
129 Num returns Num:
130     value=EDouble;
131
132 Strings returns Strings:
```

```
132     value=EString;
133
134 Bool returns Bool:
135     value=EBoolean;
136
137 BoolAttributes returns BoolAttributes:
138     boolAttr=BoolAttr;
139
140 StringAttributes returns StringAttributes:
141     stringAttr=StringAttr;
142
143 NumAttributes returns NumAttributes:
144     numAttr=NumAttr;
145
146 EDouble returns ecore::EDouble:
147     '-'? INT? '.' INT ((('E' | 'e') '-'? INT)?;
148
149 EBoolean returns ecore::EBoolean:
150     'true' | 'false';
151
152 enum BoolAttr returns BoolAttr:
153     SNRValues_ASC = 'SNRValues_ASC' | SNRValues_DESC = 'SNRValues_DESC';
154
155 enum StringAttr returns StringAttr:
156     OpRole = 'OperationalRole';
157
158 enum NumAttr returns NumAttr:
159     SecLevel = 'SecurityLevel' | CarrierFreq = 'CarrierFrequency';
160
161 ActionRadio returns ActionRadio:
162     newState=RadioModes;
163
164 enum RadioModes returns RadioModes:
165     SILENT_MODE = 'SILENT_MODE' | ALERT_MODE = 'ALERT_MODE' |
        FULLDUPLEX_MODE = 'FULLDUPLEX_MODE' | NORMAL_MODE = 'NORMAL_MODE' |
        MAINTAIN_CURRENT_MODE = 'MAINTAIN_CURRENT_MODE';
```

ANEXO F – INSTANCIACÃO DE REGRAS ATRAVÉS DA DSL GERADA

```

1 RuleSet{
2     Rule 'Rule_ST1' {
3         priority = 1
4         rule = _[_(SNRValues_ASC) and [_(_SecurityLevel >= 0.5) and (
5             CarrierFrequency=50.0) and (OperationalRole="Cmt")]]
6         --> SILENT_MODE
7     },
8     Rule 'Rule_ST2' {
9         priority = 2
10        rule = _[_(SNRValues_DESC=true) and [_(_SecurityLevel < 0.5) and (
11            CarrierFrequency=100.0) and (OperationalRole="Soldier")]]
12        --> NORMAL_MODE
13    },
14    Rule 'Rule_ST3' {
15        priority = 3
16        rule = _[_(SNRValues_ASC=true) and [_(_SecurityLevel < 0.5) and (
17            CarrierFrequency=100.0) and (OperationalRole="Cmt")]]
18        --> FULLDUPLEX_MODE
19    },
20    Rule 'Rule_ST4' {
21        priority = 4
22        rule = _[_(SNRValues_DESC=true) and [_(_SecurityLevel >= 0.5) and (
23            CarrierFrequency=50.0) and (OperationalRole="Soldier")]]
24        --> NORMAL_MODE
25    },
26    Rule 'Rule_ST5' {
27        priority = 5
28        rule = _(_SNRValues_ASC=true) --> ALERT_MODE
29    },
30    Rule 'Rule_ST6' {
31        priority = 6
32        rule = _(_SNRValues_DESC=true) --> ALERT_MODE
33    },
34    Rule 'Rule_ST7' {
35        priority = 7
36        rule = _[_(SNRValues_ASC=false) and (SNRValues_DESC=false)] -->
37            MAINTAIN_CURRENT_MODE
38    }
39 }

```

ANEXO G – COMPARAÇÃO DOS MÉTODOS DE INSTÂNCIAÇÃO DE REGRAS

Instanciação de uma regra através da DSL desenvolvida:

```

1 RuleSet{
2   Rule 'Rule_ST7' {
3     priority = 7
4     rule = _[ _(SNRValues_ASC=false)and(SNRValues_DESC=false)] -->
5       MAINTAIN_CURRENT_MODE
6   }

```

Instanciação da mesma regra através da DSL gerada automaticamente pela ferramenta Xtext:

```

1 RuleSet{
2   rule {
3     Rule 7 {
4       name "Rule_ST7"
5       leftside
6         CompoundExp {
7           PrecedingLogicOp None
8           expression {
9             SimpleExp {
10               PrecedingLogicOp None
11               operator Comparator { Operation equal }
12               operand { BoolAttributes { boolAttr SNRValues_ASC} , Bool {
13                 value false } }
14               }
15               SimpleExp {
16                 PrecedingLogicOp and
17                 operator Comparator { Operation equal }
18                 operand { BoolAttributes { boolAttr SNRValues_DESC} , Bool {
19                   value false } }
20                 }
21               rightside ActionRadio { newState MAINTAIN_CURRENT_MODE }
22             }
23           }
24         }
25       }

```

ANEXO H – RESULTADO DO TESTE - *HARD CODED*

```
1 2021-09-10T09:57:55.775 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-10T09:57:55.810 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-10T09:57:55.842 5.0 100 0.1 cmt
6 MAINTAIN_CURRENT_MODE
7 2021-09-10T09:57:55.874 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-10T09:57:55.905 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-10T09:57:55.936 3.0 100 0.6 cmt
12 MAINTAIN_CURRENT_MODE
13 2021-09-10T09:57:55.968 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-10T09:57:55.999 3.0 50 0.4 soldier
16 MAINTAIN_CURRENT_MODE
17 2021-09-10T09:57:56.032 2.0 100 0.4 cmt
18 MAINTAIN_CURRENT_MODE
19 2021-09-10T09:57:56.064 8.0 100 0.6 soldier
20 MAINTAIN_CURRENT_MODE
21 2021-09-10T09:57:56.095 7.0 50 0.8 soldier
22 MAINTAIN_CURRENT_MODE
23 2021-09-10T09:57:56.126 9.0 50 0.9 soldier
24 MAINTAIN_CURRENT_MODE
25 2021-09-10T09:57:56.158 5.0 50 0.7 soldier
26 MAINTAIN_CURRENT_MODE
27 2021-09-10T09:57:56.191 3.0 100 0.1 soldier
28 NORMAL_MODE
29 2021-09-10T09:57:56.222 2.0 100 0.5 cmt
30 ALERT_MODE
31 2021-09-10T09:57:56.253 2.0 100 0.1 cmt
32 MAINTAIN_CURRENT_MODE
33 2021-09-10T09:57:56.284 3.0 100 0.6 cmt
34 MAINTAIN_CURRENT_MODE
35 2021-09-10T09:57:56.316 8.0 50 0.0 soldier
36 ALERT_MODE
37 2021-09-10T09:57:56.347 5.0 50 0.7 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-10T09:57:56.378 4.0 100 1.0 soldier
40 ALERT_MODE
```

ANEXO I – RESULTADO DO TESTE - MDE

```
1 2021-09-10T09:41:01.248 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-10T09:41:01.280 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-10T09:41:01.311 5.0 100 0.1 cmt
6 MAINTAIN_CURRENT_MODE
7 2021-09-10T09:41:01.342 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-10T09:41:01.371 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-10T09:41:01.402 3.0 100 0.6 cmt
12 MAINTAIN_CURRENT_MODE
13 2021-09-10T09:41:01.434 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-10T09:41:01.465 3.0 50 0.4 soldier
16 MAINTAIN_CURRENT_MODE
17 2021-09-10T09:41:01.493 2.0 100 0.4 cmt
18 MAINTAIN_CURRENT_MODE
19 2021-09-10T09:41:01.523 8.0 100 0.6 soldier
20 MAINTAIN_CURRENT_MODE
21 2021-09-10T09:41:01.556 7.0 50 0.8 soldier
22 MAINTAIN_CURRENT_MODE
23 2021-09-10T09:41:01.588 9.0 50 0.9 soldier
24 MAINTAIN_CURRENT_MODE
25 2021-09-10T09:41:01.617 5.0 50 0.7 soldier
26 MAINTAIN_CURRENT_MODE
27 2021-09-10T09:41:01.649 3.0 100 0.1 soldier
28 NORMAL_MODE
29 2021-09-10T09:41:01.680 2.0 100 0.5 cmt
30 ALERT_MODE
31 2021-09-10T09:41:01.711 2.0 100 0.1 cmt
32 MAINTAIN_CURRENT_MODE
33 2021-09-10T09:41:01.743 3.0 100 0.6 cmt
34 MAINTAIN_CURRENT_MODE
35 2021-09-10T09:41:01.775 8.0 50 0.0 soldier
36 ALERT_MODE
37 2021-09-10T09:41:01.807 5.0 50 0.7 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-10T09:41:01.838 4.0 100 1.0 soldier
40 ALERT_MODE
```

ANEXO J – RESULTADO DO TESTE - LOGS ALTERADOS

```
1 2021-09-11T21:10:47.855 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-11T21:10:47.880 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-11T21:10:47.912 5.0 100 0.1 cmt
6 MAINTAIN_CURRENT_MODE
7 2021-09-11T21:10:48.003 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-11T21:10:48.036 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-11T21:10:48.066 3.0 100 0.6 cmt
12 MAINTAIN_CURRENT_MODE
13 2021-09-11T21:10:48.097 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-11T21:10:48.128 3.0 50 0.4 soldier
16 MAINTAIN_CURRENT_MODE
17 2021-09-11T21:10:48.160 5.0 50 0.8 soldier
18 ALERT_MODE
19 2021-09-11T21:10:48.191 7.0 50 0.9 soldier
20 ALERT_MODE
21 2021-09-11T21:10:48.221 9.0 50 0.7 soldier
22 ALERT_MODE
23 2021-09-11T21:10:48.253 6.0 50 0.6 soldier
24 MAINTAIN_CURRENT_MODE
25 2021-09-11T21:10:48.284 5.0 50 0.6 soldier
26 NORMAL_MODE
27 2021-09-11T21:10:48.315 4.0 50 0.4 soldier
28 ALERT_MODE
29 2021-09-11T21:10:48.347 2.0 50 0.3 cmt
30 ALERT_MODE
31 2021-09-11T21:10:48.377 5.0 100 0.1 cmt
32 MAINTAIN_CURRENT_MODE
33 2021-09-11T21:10:48.408 6.0 100 0.1 cmt
34 FULLDUPLEX_MODE
35 2021-09-11T21:10:48.439 8.0 50 0.0 soldier
36 ALERT_MODE
37 2021-09-11T21:10:48.470 5.0 50 0.7 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-11T21:10:48.502 4.0 100 1.0 soldier
40 ALERT_MODE
```

ANEXO K – RESULTADO DO TESTE - ADIÇÃO DE NOVA REGRA

```
1 2021-09-15T10:07:23.021 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-15T10:07:23.050 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-15T10:07:23.081 5.0 100 0.1 cmt
6 FULLDUPLEX_MODE
7 2021-09-15T10:07:23.113 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-15T10:07:23.143 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-15T10:07:23.173 3.0 100 0.6 cmt
12 FULLDUPLEX_MODE
13 2021-09-15T10:07:23.204 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-15T10:07:23.235 3.0 50 0.4 soldier
16 MAINTAIN_CURRENT_MODE
17 2021-09-15T10:07:23.266 2.0 100 0.4 cmt
18 FULLDUPLEX_MODE
19 2021-09-15T10:07:23.297 8.0 100 0.6 soldier
20 MAINTAIN_CURRENT_MODE
21 2021-09-15T10:07:23.328 7.0 50 0.8 soldier
22 MAINTAIN_CURRENT_MODE
23 2021-09-15T10:07:23.358 9.0 50 0.9 soldier
24 MAINTAIN_CURRENT_MODE
25 2021-09-15T10:07:23.389 5.0 50 0.7 soldier
26 MAINTAIN_CURRENT_MODE
27 2021-09-15T10:07:23.420 3.0 100 0.1 soldier
28 NORMAL_MODE
29 2021-09-15T10:07:23.451 2.0 100 0.5 cmt
30 ALERT_MODE
31 2021-09-15T10:07:23.482 2.0 100 0.1 cmt
32 FULLDUPLEX_MODE
33 2021-09-15T10:07:23.514 3.0 100 0.6 cmt
34 FULLDUPLEX_MODE
35 2021-09-15T10:07:23.544 8.0 50 0.0 soldier
36 ALERT_MODE
37 2021-09-15T10:07:23.575 5.0 50 0.7 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-15T10:07:23.606 4.0 100 1.0 soldier
40 ALERT_MODE
```

ANEXO L – RESULTADO DO TESTE - NENHUMA REGRA PRESENTE

```
1 2021-09-23T11:29:06.105 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-23T11:29:06.129 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-23T11:29:06.160 5.0 100 0.1 cmt
6 MAINTAIN_CURRENT_MODE
7 2021-09-23T11:29:06.192 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-23T11:29:06.223 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-23T11:29:06.255 3.0 100 0.6 cmt
12 MAINTAIN_CURRENT_MODE
13 2021-09-23T11:29:06.287 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-23T11:29:06.317 3.0 50 0.4 soldier
16 MAINTAIN_CURRENT_MODE
17 2021-09-23T11:29:06.349 2.0 100 0.4 cmt
18 MAINTAIN_CURRENT_MODE
19 2021-09-23T11:29:06.380 8.0 100 0.6 soldier
20 MAINTAIN_CURRENT_MODE
21 2021-09-23T11:29:06.411 7.0 50 0.8 soldier
22 MAINTAIN_CURRENT_MODE
23 2021-09-23T11:29:06.443 9.0 50 0.9 soldier
24 MAINTAIN_CURRENT_MODE
25 2021-09-23T11:29:06.474 5.0 50 0.7 soldier
26 MAINTAIN_CURRENT_MODE
27 2021-09-23T11:29:06.505 3.0 100 0.1 soldier
28 MAINTAIN_CURRENT_MODE
29 2021-09-23T11:29:06.536 2.0 100 0.5 cmt
30 MAINTAIN_CURRENT_MODE
31 2021-09-23T11:29:06.567 2.0 100 0.1 cmt
32 MAINTAIN_CURRENT_MODE
33 2021-09-23T11:29:06.598 3.0 100 0.6 cmt
34 MAINTAIN_CURRENT_MODE
35 2021-09-23T11:29:06.628 8.0 50 0.0 soldier
36 MAINTAIN_CURRENT_MODE
37 2021-09-23T11:29:06.659 5.0 50 0.7 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-23T11:29:06.690 4.0 100 1.0 soldier
40 MAINTAIN_CURRENT_MODE
```

ANEXO M – RESULTADO DO TESTE - ERROS NOS LOGS

```
1 2021-09-23T12:53:30.626 7.0 100 0.6 cmt
2 SilentMode
3 2021-09-23T12:53:30.656 9.0 50 1.0 soldier
4 SilentMode
5 2021-09-23T12:53:30.688 5.0 100 0.1 cmt
6 MAINTAIN_CURRENT_MODE
7 2021-09-23T12:53:30.719 9.0 100 0.9 soldier
8 MAINTAIN_CURRENT_MODE
9 2021-09-23T12:53:30.750 2.0 100 0.4 soldier
10 MAINTAIN_CURRENT_MODE
11 2021-09-23T12:53:30.782 3.0 100 0.6 cmt
12 MAINTAIN_CURRENT_MODE
13 2021-09-23T12:53:30.845 2.0 50 0.4 cmt
14 MAINTAIN_CURRENT_MODE
15 2021-09-23T12:53:31.095 2.0 100 0.5 cmt
16 MAINTAIN_CURRENT_MODE
17 2021-09-23T12:53:31.126 3.0 100 0.6 cmt
18 MAINTAIN_CURRENT_MODE
19 2021-09-23T12:53:31.156 8.0 50 0.0 soldier
20 ALERT_MODE
21 2021-09-23T12:53:31.187 5.0 50 0.7 cmt
22 MAINTAIN_CURRENT_MODE
23 2021-09-23T12:53:31.218 4.0 100 1.0 soldier
24 ALERT_MODE
25 2021-09-23T12:53:31.249 8.0 100 0.9 soldier
26 MAINTAIN_CURRENT_MODE
27 2021-09-23T12:53:31.279 3.0 50 0.9 soldier
28 MAINTAIN_CURRENT_MODE
29 2021-09-23T12:53:31.311 4.0 100 0.8 cmt
30 MAINTAIN_CURRENT_MODE
31 2021-09-23T12:53:31.342 8.0 50 0.1 soldier
32 ALERT_MODE
33 2021-09-23T12:53:31.373 4.0 100 0.7 cmt
34 MAINTAIN_CURRENT_MODE
35 2021-09-23T12:53:31.404 6.0 50 0.6 soldier
36 MAINTAIN_CURRENT_MODE
37 2021-09-23T12:53:31.434 4.0 100 1.0 cmt
38 MAINTAIN_CURRENT_MODE
39 2021-09-23T12:53:31.466 6.0 50 0.4 soldier
40 MAINTAIN_CURRENT_MODE
41 2021-09-23T12:53:31.497 8.0 50 0.9 soldier
42 ALERT_MODE
```