UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Giane Alessio Binotto

LE-FOM: DESENVOLVIMENTO DE UM SOFTWARE PARA LEITURA E EDIÇÃO DE ARQUIVOS FOM

Santa Maria, RS
2016

Giane Alessio Binotto

LE-FOM: DESENVOLVIMENTO DE UM SOFTWARE PARA LEITURA E EDIÇÃO DE ARQUIVOS FOM

Trabalho Final de Graduação apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Raul Ceretta Nunes

421
Santa Maria, RS
2016

Giane Alessio Binotto

LE-FOM: DESENVOLVIMENTO DE UM SOFTWARE PARA LEITURA E EDIÇÃO DE ARQUIVOS FOM

Trabalho Final de Graduação apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Aprovado em 15 de dezembro de 2016:

Raul Ceretta Nunes, Prof Dr. (UFSM)
(Presidente/Orientador)

Giovani Rubert Librelotto, Prof Dr. (UFSM)

Edison Pignaton de Freitas, PhD. (UFRGS)

Santa Maria, RS
2016

DEDICATÓRIA

Dedico este trabalho à base da minha vida: minha família.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Nelsi e Arlete, por todo o carinho, educação e incentivo que me deram, fazendo com que eu conseguisse chegar onde estou agora. Agradeço a eles também, por não medirem esforços para me proporcionar tudo o que eu sempre precisei, nunca me deixando faltar nada. Palavras não definem a gratidão que sinto por tudo isso. Essa frase soa meio clichê, mas por mim nunca foi dita com tanta verdade. Agradeço com muito amor também às minhas irmãs, que sempre se preocuparam comigo e sempre procuraram me consolar com palavras de conforto e abraços nos meus momentos mais difíceis. Com toda sua experiência, souberam me orientar quando não sabia o que fazer.

Agradeço ao meu namorado Robson, por todo amor e carinho dados durante todos esses anos que passamos juntos. Agradeço também pela compreensão e paciência nos meus momentos de ansiedade e nervosismo, confortando-me com muito amor sempre que precisei. Agradeço por toda ajuda, apoio, conselhos e momentos de alegria em meio a esse semestre o qual foi uma etapa difícil para mim. Agradeço por sempre tentar fazer o possível para me fazer sentir melhor e sorrir.

Agradeço ao professor Raul, por toda orientação dada durante este trabalho. Agradeço pela atenção, pelos ensinamentos e pelo auxílio prestado sempre que precisei. Agradeço a ele e a todos os professores da Ciência da Computação por toda sabedoria transmitida e ensinamentos que ajudaram no meu crescimento profissional. Agradeço aos professores Edison de Freitas e Giovani Librelotto pela disponibilidade de fazerem parte da banca de avaliação. Agradeço também à Coordenação do curso de Ciência da Computação, juntamente com a secretaria por todos os serviços prestados e orientações. Agradeço ao Programa de Educação Tutorial de Ciência da Computação pela oportunidade de ser uma integrante e por toda experiência que me agregou tanto profissional quanto pessoal.

Agradeço às minhas amigas de infância, Danise, Letícia, Pauline, Eduarda, Thayse por todo apoio, palavras de conforto, conselhos e torcidas nesse semestre para que tudo sempre acabasse certo.

Agradeço aos meus queridos colegas e amigos, Jessica, Liza, Andressa, Ana, Felipe, Vanderlan, Ricardo e Tiago pelo companheirismo de sempre. Não sei o que seria de mim sem vocês durante esses anos de universidade. Agradeço e admiro vocês por não encararem os seus colegas como concorrentes de profissão e por sempre terem ajudado uns aos outros. Agradeço por todas risadas, apoio, festas, tragos e histórias que temos para compartilhar. Agradeço por nossa amizade não ter se acabado com o fim da nossa trajetória na universidade. Agradeço também a outros amigos que ganhei durante a graduação, que sempre me apoiaram e torceram pelo meu sucesso.

RESUMO

LE-FOM: DESENVOLVIMENTO DE UM SOFTWARE PARA LEITURA E EDIÇÃO DE ARQUIVOS FOM

AUTORA: Giane Alessio Binotto
ORIENTADOR: Raul Ceretta Nunes

Os desenvolvedores, os quais trabalham com simulação utilizando o padrão High Level Architecture, tendem a ter dificuldades em se tratando de visualização e manipulação do modelo de objeto FOM. Este software tem como propósito auxiliar nessas duas tarefas por meio de uma interface gráfica, onde os componentes estarão separados e disponíveis para serem editados. O trabalho baseia-se na versão do FOM de acordo com a norma IEEE (2010a), sendo genérico a todo tipo de modelo de objeto que se enquadra nesta norma, com exceção a FOMs modulares. O desenvolvimento da aplicação consiste em leitura, edição e validação. A leitura é realizada com o auxílio de uma biblioteca Java e os dados são armazenados em estruturas customizadas. A edição é feita através da interface, onde o usuário pode interagir com os componentes os alterando e adicionando novos itens. A validação obteve-se através da comparação com o XML Schema e o arquivo gerado com as edições, a fim de que consistisse à norma HLA. Através de testes de leitura, entradas, edição e validação, foi possível verificar que o software funciona sem erros e realiza o objetivo principal. Portanto, o software desenvolvido atende a todos os requisitos, sendo um software livre à toda comunidade de desenvolvedores na área de simulação.

Palavras-chave: Federation Object Model. High Level Architecture. Object Model Template.

ABSTRACT

LE-FOM: DEVELOPMENT OF SOFTWARE FOR FOM FILE READING AND EDITING

AUTHOR: Giane Alessio Binotto
ADVISOR: Raul Ceretta Nunes

Developers, who work with simulation using the High Level Architecture standard, tend to have difficulties regarding visualization and manipulation of the object model FOM. This software has as purpose to assist in these two tasks through a graphic interface, where components will be separated and available to be edited. The application is based on the FOM version, according to the standard IEEE (2010a), being generic to all types of object models which fit in this standard, with the exception of modular FOMs. The application development consists of reading, editing and validating. Reading is accomplished with the support of a Java library and the data are stored in customized structures. Editing is performed through an interface, where a user can interact with the components, changing them and adding new items. Validating was obtained by a comparison of the XML Schema and the generated file with changes, so that it consisted to the standard. Through reading, input, editing and validating tests, a check that the software works without errors and accomplishes the main purpose was possible. Therefore, the software developed meet the requirements, being a free software to the whole community of developers in the simulation area.

Keywords: Federation Object Model. High Level Architecture. Object Model Template.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

$DMSO$	Defense Modeling and Simulation Office
DoD	Department of Defense
DOM	Document Object Model
FOM	Federation Object Model
HLA	High Level Architecture
$HTML$	HyperText Markup Language
$HTTP$	Hypertext Transfer Protocol
$IEEE$	Institute of Electrical and Electronics Engineers
MVC	Model-view-controller
OMT	Object Model Template
RTI	Run-Time Interface Specification
SAX	Simple API for XML
$SimGE$	Simulation Generator
SOM	Simulation Object Model
URI	Uniform Resource Identifier
$W3C$	World Wide Web Consortium
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A simulação computacional era última opção no passado, sendo utilizada apenas quando todas as técnicas possíveis falhassem (CHWIF; MEDINA, 2010). Segundo Gavira (2003), a simulação computacional surgiu em meio a necessidade de uma abordagem mais generalista e sistêmica para problemas mais complexos. Cardoso e Joaquim (2016) afirmam que atualmente "a simulação é uma das técnicas mais utilizadas na pesquisa operacional, amparando fortemente o desenvolvimento de técnicas e projetos de otimização de sistemas". Além disso, o campo de simulação vem crescendo e ganhando espaço em diversas áreas. Como exemplo, temos simuladores na área automobilística, na área de defesa e educacional.

Para tornar possível o crescimento da simulação, comportando problemas complexos, surgiu a simulação distribuída. A simulação distribuída possui vantagens como redução no tempo de execução, maior tolerância a falhas e integração de diferentes simuladores. Entretanto, assim como todo sistema distribuído, a simulação distribuída precisou de uma padronização para que todas as partes pudessem comunicar-se. Em 1998, foi publicada uma versão de um padrão para simulação computacional chamado HLA V1.3, sendo desenvolvido a partir do DIS pelo Defense Modeling and Simulation Office (DMSO) do Departamento de Defesa dos Estados Unidos (DoD). Mais adiante em 2000, o DMSO submeteu o HLA V1.3 ao Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) que por sua vez foi aprovado como um padrão aberto (KRÁLÍCEK, 2011). O seu propósito é suportar e administrar uma ampla variedade de aplicações de simulação militares. Ademais, essa normatização pode ser facilmente reutilizada em outras aplicações, quer individualmente ou em combinação. A HLA é composta de regras, Objetc Model Templates (OMT) e uma camada chamada Run-Time Interface Specification (RTI). Além disso, dentro da norma existe a obrigatoriedade da existência de dois Modelos de Objeto: o Simulation Object Model (SOM) e o Federation Object Model (FOM). Esse último, contém elementos muito importantes como classes de objeto e interações entre simuladores. Porém, pode-se dizer que é um desafio conseguir localizar objetos específicos dentro deste arquivo, uma vez que é um arquivo grande, com centenas de elementos que possuem uma hierarquia.

Em vista desse problema, foi desenvolvido um software especializado para a leitura e edição do arquivo FOM que auxiliará desenvolvedores da área de simulação que usufruem da norma HLA. Já existem trabalhos que executam essa atividade, porém são pagos ou não específicos a esse problema. Esse trabalho será um software livre e genérico, ou seja, FOMs diferentes que estão de acordo com a norma IEEE (2010a) poderão ser lidos e modificados. A visualização é feita através de uma interface gráfica, onde o usuário tem acesso a elementos e atributos pertencentes a um determinado FOM. Além disso, o desenvolvedor não precisa procurar o objeto entre os elementos do arquivo extenso para

alterá-lo, pois pode obter a modificação através da interface.

1.1 OBJETIVOS

O objetivo geral deste trabalho consiste em desenvolver um software na linguagem Java para manipulação de arquivos FOM. Pretende-se com esse software prover um auxílio no uso do FOM em federados através de uma melhor representação e gerenciamento de elementos novos e existentes. Com o software, deve ser possível a visualização e a edição do Modelo de Objeto da Federação com alterações do conteúdo das tags. Além disso, diferentes FOMs de diferentes federados poderão ser modificados, ou seja, o software será genérico.

Este trabalho tem os seguintes objetivos específicos:

- Aprofundar os conhecimentos relacionados à linguagem de marcação XML;
- Implementar um parser XML para a leitura do modelo FOM, onde cada elemento será estruturado de acordo com sua definição no padrão HLA (IEEE, 2010a);
- Criar uma representação gráfica para cada elemento;
- Implementar a edição de um FOM por meio da interface gerada com os elementos, modificando e/ou adicionando tags no arquivo;
- Testar o arquivo editado para que esteja consistente com o padrão HLA (IEEE, 2010a);
- Desenvolver um software que seja de livre acesso a todos da área de simulação.

1.2 JUSTIFICATIVA

Este tema foi escolhido tendo em vista a dificuldade de interação com o modelo de objeto da federação (FOM). Desenvolvedores que trabalham com a simulação e que utilizam a norma HLA necessitam de uma forma simples e eficiente para mudanças neste modelo, uma vez que ele possui uma estrutura hierárquica com muitos elementos. Além disso, desenvolvedores que customizam um FOM já criado anteriormente, chamado FOM de Referência também passam por dificuldades. Isso porque o usuário precisa entender o FOM já criado e então customizá-lo, atendendo à norma HLA e também aos seus objetivos na simulação. Com intuito de solucionar essa adversidade, um software para visualização

e edição de arquivos FOM foi desenvolvido neste trabalho. Foi tomada como base a especificação do Object Model Template elaborado pelo instituto IEEE (IEEE, 2010a), a qual define os elementos presentes no modelo.

1.3 ESTRUTURA DO TEXTO

Este trabalho está estruturado em seis capítulos. O primeiro (este capítulo) descreve uma introdução geral sobre o que se trata o trabalho e o que será feito. Além disso, são descritos os objetivos gerais e específicos que se espera alcançar com esse trabalho e a justificativa. No segundo capítulo, é fundamentada a parte teórica, gerando um embasamento sobre assuntos que farão parte do trabalho. Ainda, trabalhos relacionados são descritos, com semelhanças e distinções do que é proposto. O terceiro capítulo trata dos requisitos necessários para o software bem como a forma que os mesmos foram modelados e estruturados para serem implementados. No quarto capítulo aspectos do desenvolvimento da aplicação são descritos, juntamente com recursos utilizados e diagramas para uma melhor visualização do sistema. A quinta seção apresenta testes que foram realizados e seus resultados, os quais validam o software de acordo com a norma (IEEE, 2010a). Por fim, na última seção é apresentada a conclusão, retomando pontos importantes e destacando possíveis trabalhos que darão continuidade ao software de leitura e edição de arquivos FOM.

2 FUNDAMENTAÇÃO E REVISÃO BIBLIOGRÁFICA

Em se tratando de contexto do assunto principal do trabalho, temos a simulação computacional como campo de aplicação do arquivo FOM. A norma High Level Architecture rege e padroniza as simulações, requerendo interfaces e serviços comuns aos simuladores, incluindo o Modelo de Objeto abordado. Além disso, tem-se XML como linguagem a qual o FOM é escrito e que deverá ser estudada a fim do desenvolvimento de um parser, parte do objetivo do software desenvolvido neste trabalho.

2.1 SIMULAÇÃO COMPUTACIONAL

Simulação computacional é a imitação de funcionalidades de operações ou processos da realidade com menor custo, através de técnicas matemáticas empregadas em computadores. Ainda segundo Pegden, Sadowski e Shannon (1995) “a simulação é um processo de projetar um modelo computacional de um sistema real e conduzir experimentos com este modelo com o propósito de entender seu comportamento e/ou avaliar estratégias para sua operação”. Para Shannon (1975) a simulação computacional se trata de um método de modelagem que é capaz de prever possíveis erros, os quais poderiam gerar alto custo e um maior tempo para resolução, se testados na realidade. Segundo Strack (1984) deve-se aplicar este método quando certas condições existirem. A inexistência de uma formulação matemática completa para o problema ou não é possível ou é muito difícil a experimentação no sistema real são exemplos de circunstâncias em que o uso seria considerado.

A simulação auxilia na preparação e treinamento de diversas tarefas de diversos ramos. Por exemplo, antes de obter a carteira de habilitação automobilística deve-se utilizar um simulador de direção para que se tenha uma prévia de como é dirigir na realidade, além de aumentar habilidades e adquirir experiência. Ademais, conforme Gavira (2003), a simulação é uma ferramenta que permite vários profissionais executar as atividades as quais se propõem. Através do uso da simulação computacional consegue-se aquisição, organização e construção do conhecimento e da visão sistêmica.

2.2 SIMULAÇÃO DISTRIBUÍDA

A simulação distribuída consiste em equipamentos geograficamente dispersos que são interconectados através de uma rede de comunicação, obtendo-se um super compu-

tador virtual, com o objetivo de executar programas computacionais necessários para uma simulação (FUJIMOTO, 2000). Desta forma um modelo de simulação é disperso entre múltiplos processadores. Junqueira (2006) afirma que o alvo para utilização da simulação distribuída não é somente sistemas complexos, mas também sistemas simples que possuem um alto grau de interação. Conforme Fujimoto (2000), esse tipo de simulação visa auxiliar na simulação de grandes modelos, compostos por muitos elementos, trazendo vários benefícios como:

- Redução do tempo de simulação, através da execução concorrente utilizando vários processadores. O tempo pode ser diminuído em até um fator igual ao número de processadores que são usados. A importância dessa vantagem se aplica a simulações que levam muito tempo para serem executadas, tais como uma simulação de comunicação rede contendo vários nós;

- Aumento na tolerância a falhas, permitindo que um processador substitua outro, caso haja falha;

- Distribuição geográfica, criando-se mundos virtuais com vários participantes localizados fisicamente em lugares diferentes. Esse benefício faz com que os custos com viagens para exercícios envolvendo participantes de diversos locais sejam aliviados. Além disso, de acordo com Venkateswaran, Jafferali e Son (2001), a distribuição geográfica pode fazer com que empresas virtuais (organização onde várias companhias se unem para produzir um produto) sejam criadas;

- Integração de simuladores, combinando simulações localizadas em máquinas de diferentes fabricantes. A economia de cada simulador ser executado na máquina de seu fabricante é muito maior que se tivessem que ser ligados a uma única máquina.

2.3 HIGH LEVEL ARCHITECTURE

HLA é uma norma de modelagem e simulação distribuída desenvolvida por Defense Modeling and Simulation Office (DMSO) para o Departamento de Defesa dos Estados Unidos (DoD). Conforme Králícek (2011), o padrão teve sua primeira versão publicada em 1998 e era chamada de HLA V1.3, a qual teve muitas atualizações e passou a ser chamada de HLA V1.3 New Generation 6. Em 2000, o DMSO submeteu o padrão para a aprovação do IEEE, que por sua vez tornou-se um padrão aberto. A partir desse momento passou a ser chamado de IEEE 1516 - IEEE Standard for Modeling and Simulation (MS): High Level Architecture (HLA) e sua versão conhecida como HLA IEEE 1516.1. Ao passar dos anos, a norma foi sendo revisada pela organização Simulation Interoperability Standards

Organization (SISO) HLA-Evolved Product Development Group o que resultou em uma nova versão, aprovada novamente pelo IEEE em agosto de 2010. Essa última é chamada de IEEE 1516-2010 e conhecida como HLA Evolved (KRÁLÍCEK, 2011).

Segundo IEEE (2010b), o HLA define um framework comum usado para a interconexão e interoperabilidade de simulações que interagem entre si. Além disso, são definidos interfaces e serviços os quais são usados por federados (aplicação que está dentro do padrão HLA e é capaz de participar de uma simulação) para trocas de informações eficientes dentro da federação (conjunto de federados com descrições do como e o que será simulado). O padrão HLA é composto por três componentes: regras, Objetc Model Templates (OMT) e uma Especificação de Interface.

As regras são dez em sua totalidade e definem o comportamento que federados devem seguir a fim de alcançar uma interação de sucesso durante a execução da federação. Além disso, elas garantem a interação de simulações adequada dentro da federação e descrevem responsabilidades da simulação e de cada federado. Algumas das determinações deste regulamento são a obrigatoriedade de um Simulation Object Model (SOM) documentado usando OMT e a capacidade de transferir de forma dinâmica e aceitar a propriedade de atributos durante a execução da federação, conforme especificado em seu SOM. Ademais, também existem regras para a federação e dentre elas estão a obrigatoriedade de um Federation Object Model (FOM) documentado usando o OMT e a interação de federados com o RTI de acordo com a especificação de interface estabelecido pela norma HLA.

Os OMTs são responsáveis por prover um mecanismo o qual especificará trocas de dados e coordenação dentro da federação. De acordo com Möller et al. (2013), os objetos de modelo descrevem classes de objeto com atributos, interações com parâmetros, tipos de dados e muitas outras coisas. Os OMTs são gerados a fim de facilitar a implementação de ferramentas comuns na criação de objetos compatíveis com a norma HLA. Existem dois tipos de Object Model Template os quais são o Federation Object Model (FOM) e o Simulation Object Model (SOM). Segundo Möller et al. (2013), o FOM contém um modelo de troca de informações que é usado por uma federação durante o tempo de execução. Por outro lado, o SOM é usado para descrever as capacidades de um federado e as informações, como objetos e atributos, expostas e consumidas por um federado.

A Especificação de Interface define um padrão para uma Run-Time Infrastructure (RTI). Conforme IEEE (2010b), RTI é uma camada a qual disponibiliza serviços e interfaces comuns a todos federados. Ademais, segundo Králícek (2011), a interface define a maneira a qual os federados irão interagir uns com os outros. Essa distribuição de serviços é similar a maneira que um sistema operacional distribuído disponibiliza serviços a suas aplicações. As interfaces são organizadas em sete grupos, as quais possuem a obrigatoriedade do uso por federados interagirem com os demais. Os grupos descrevem as interfaces entre os federados e a camada RTI, bem como entre os federados e os serviços de software

oferecidos pela camada. São eles:

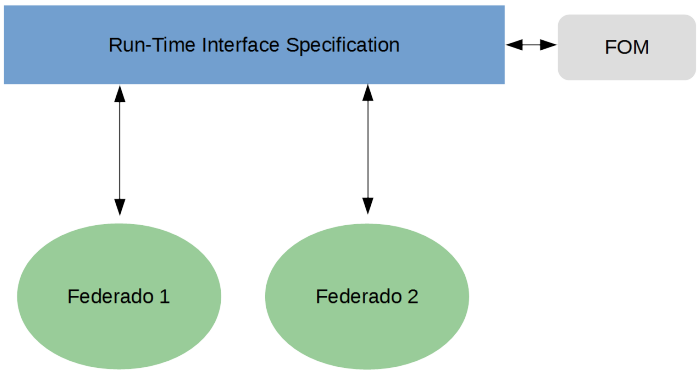
1. Gerenciamento de federação
2. Gerenciamento de declaração
3. Gerenciamento de objeto
4. Gerenciamento de propriedade
5. Gerenciamento de tempo
6. Gerenciamento de distribuição de dados
7. Serviços de suporte

Em se tratando de implementações, a RTI possui várias de código aberto e pagos. Cada RTI é considerado um middleware e oferece uma biblioteca de programação bem como uma interface de programação de aplicativos (API). A camada pode ser executada em uma única máquina ou através da rede. O componente principal é chamado RTI Executive Process, o qual é o processo acessível por toda federação e responsável pela inicialização de componentes RTI. Outro componente importante dentro do RTI chama-se Federation Executive Process e é incumbido a gerenciar a federação, ou seja, coordenar a entrada e retirada de federados e também garante a capacidade da troca de dados entre eles (KRÁLÍCEK, 2011). Por fim, a biblioteca chamada LibRTI é vinculada a cada federado, fornecendo o serviço HLA especificado na Especificação de Interface, através de um método atingível à federação. Ela é escrita em C++, e comumente oferece interfaces em C++ e outras linguagens (KRÁLÍCEK, 2011).

2.4 FEDERATION OBJECT MODEL

O Federation Object Model é um modelo o qual define o tipo de informações transmitidas entre os federados, ou seja, descreve um conjunto de classes de objetos e interações escolhidas para determinada federação. Segundo IEEE (2010b), as regras e terminologias as quais descrevem um FOM são documentadas em (IEEE, 2010a) e seguem um formato e sintaxe padrão chamado HLA OMT. A criação deste modelo surgiu com a necessidade de uma compreensão total por todos os membros de uma federação sobre as comunicações entre os mesmos. Sendo assim, o FOM tem como principal propósito padronizar a troca de informação em um formato o qual seja de comum entendimento a todos os federados. Sua especificação inclui a interface entre os federados e a camada RTI, juntamente com os serviços HLA, formando um "contrato de modelo de informação"(IEEE,

Figura 2.1 – Arquitetura HLA



Fonte: Próprio autor

2010a). O modelo possui três versões respectivas a cada norma HLA estabelecida. A primeira versão HLA 1.3 se manteve entre os anos 1996 a 1998, a segunda HLA 1516-2000 do ano 2000, usava DTD como formato para especificar o documento e a última e mais recente HLA 1516-2010 do ano de 2010 utiliza XML Schema e é o padrão foco deste trabalho.

A partir do FOM, são geradas instâncias de classes de objetos e podem ter um grupo de atributos relacionados a eles. Como exemplo, no Listing 2.1 está uma classe de objeto chamada BreachableLinearObject que possui um atributo chamado SegmentRecords. Um atributo é uma parte diferenciada e identificada do estado do objeto, possuindo um designador que é gerado dinamicamente através da RTI e um valor. Além disso, os objetos podem ter atributos comuns à toda federação, bem como ter atributos particulares associados a federados específicos. Por outro lado, as classes de interações possuem parâmetros. Os tipos de interações possíveis para um Modelo de Objeto da Federação e os parâmetros são descritos dentro do FOM.

Listing 2.1: Exemplo de uma classe de objeto com um atributo

```
<objectClass>
    <name>BreachableLinearObject</name>
    <sharing>PublishSubscribe</sharing>
    <semantics>A linear object that can be breached.</semantics>
    <attribute notes="RPRnoteSE1">
        <name>SegmentRecords</name>
        <dataType>BreachableSegmentStructLengthlessArray</dataType>
        <updateType>Conditional</updateType>
        <updateCondition>On change</updateCondition>
        <ownership>NoTransfer</ownership>
        <sharing>PublishSubscribe</sharing>
        <transportation>HLAbestEffort</transportation>
        <order>Receive</order>
        <semantics>Specifies a breachable linear object</semantics>
    </attribute>
</objectClass>
```

De acordo com Möller et al. (2014), é comum usar FOMs padronizados chamados FOMs de Referência e customizá-los de acordo com o interesse e requerimentos de um projeto. Dentre as vantagens de se usar um FOM de Referência estão a economia de dinheiro e tempo, uma vez que os esforços de um projeto similar passado podem ser reutilizados e moldados para uma nova proposta. Ademais, Möller et al. (2014) afirma que reusar FOMs de Referência reduz riscos, devido a uma realização prévia de testes e ajustes para o bom funcionamento no projeto anterior. O Real-Time Platform Reference FOM (RPR FOM) é um exemplo de FOM de Referência e é regularmente usado para simulações de defesa, suportando recursos como plataformas (aviões, veículos terrestres, de superfície (como navios), submergíveis, anfíbios e naves espaciais), humanos, comunicações de rádio, entre outros. O RPR é focado na simulação em tempo real, diferenciando daquelas que correm mais rápido ou mais lento do que em tempo real. Além disso, incluiu-se na norma IEEE (2010a) a possibilidade do FOM ser composto em módulos e mesclado formando um único modelo. Tem-se como vantagem desse novo aspecto, que o reuso, desenvolvimento e manutenção podem ser específicos e realizados modularmente (MÖLLER et al., 2013). Entretanto, o objetivo do trabalho não englobará essa nova característica, sendo possível ler o FOM somente de um módulo.

2.4.1 Componentes do FOM

Conforme IEEE (2010a), FOMs são um tipo de HLA OMT, portanto possuem os componentes presentes nesse template. Os HLA OMTs são compostos de um grupo de componentes inter-relacionados que especificam informações sobre classes de objetos com seus atributos e interações com seus parâmetros. O FOM é formado pelos seguintes componentes:

- Object model identification: responsável por documentar informações de identificação importantes dentro da descrição do modelo de objeto. O objetivo principal é facilitar o reuso, pois fornecem informações que permitem inferências sobre o potencial de reutilização de federados individuais para novas aplicações.

- Object class: responsável por conter classes de objetos de todos federados ou federação bem como descrever suas relações entre superclasses e subclasses, uma vez que possuem hierarquia. As classes de objetos fornecem meios para que os federados assinem informações sobre todas as instâncias individuais de objetos HLA com características comuns. Além disso, elas especificam características de objetos de simulação, chamados atributos.

- Interaction class: responsável por conter as classes de interação de todos os federados ou federação bem como descrever suas relações entre superclasses e subclasses, uma vez que possuem hierarquia. Classes de interação são definidas como ações tomadas por um federado que podem afetar outros federados pertencentes a mesma federação. Ademais, elas são os principais determinantes de interoperabilidade entre simulações.

- Attribute: responsável por especificar características de atributos de objetos em um federado ou federação, previamente a inicialização da execução. Os atributos são especificados a fim de auxiliar o acesso de seus valores por outros federados interessados. Os valores dos atributos podem ser modificados e são atualizados aos outros federados através da camada RTI.

- Parameter: responsável por especificar características de parâmetros de interações em um federado ou federação, associando informações úteis e relevantes às suas classes.

- Dimension: responsável por especificar um conjunto de dimensões para filtrar atributos e interações. Cada subconjunto definido em um componente define um sistema de coordenadas multidimensionais através do qual os federados expressam interesse em receber dados ou declaram sua intenção de enviar dados.

- Time representation: responsável por especificar a representação de valores de tempo durante a execução para atender a duas funções. As federações podem associar-se com pontos de eixo de tempo HLA e podem associar algumas de suas atividades, como por exemplo atualizar o valor de um atributo de instância, com pontos no eixo de tempo HLA. À medida que a execução da federação progride, os federados podem avançar ao longo do eixo do tempo HLA.

- User-supplied tag: responsável por especificar a representação de tags usadas em serviços HLA. Além disso, fornece meios de documentar os acordos de federação, em se tratando do tipo de dados a ser usado com essas tags.

- Synchronization: responsável por especificar a representação e tipos de dados utilizados em serviços de sincronização HLA. A sincronização fornece meios para um federado descrever os pontos de sincronização (mecanismo da RTI para federados sincronizarem atividades) que é capaz de contemplar. Além disso, uma federação pode documentar acordos sobre pontos de sincronização a serem usados.

- Transportation type: responsável por descrever os mecanismos de transporte de dados entre federados que podem ser suportados por uma federação.

- Update rate: responsável por especificar a taxa de atualização de informação. A taxa de atualização pode ser reduzida pela RTI para atualizações de atributos usando um tipo de transporte chamado Best Effort. Além disso, para atualizações de atributos usando outro tipo de transporte chamado Reliable e também o Best Effort, a RTI fornece sinalização que permite que federados proprietários ajustem suas taxas de atualização.

- Switches: responsável por especificar as configurações iniciais para ações executadas pela RTI. Essas ações podem ser habilitadas ou desabilitadas, conforme as capacidades dos federados ou desejos da federação. Além disso, algumas delas podem ser modificadas durante a execução, afetando a maneira a qual a camada RTI interage com todos os federados.

- Datatype: responsável por especificar detalhes de representação de dados no modelo de objeto. Muitos dos componentes do FOM possuem colunas referentes à especificação do tipo de dado. Há cinco tipos de dados os quais são: simple, enumerated, fixed record, array e variant record. Membros desses tipos de dados podem estar contidos em outros, tornando-os complexos.

- Notes: responsável por complementar explicações de qualquer item nos componentes do FOM. Informações descritivas para facilitar o uso dos dados podem ser fornecidas por usuários, associando-as a itens específicos desejados.

2.5 EXTENSIBLE MARKUP LANGUAGE

XML é uma linguagem de marcação auto descritiva a qual possui dados entre tags, assim como HTML. Desenvolvida pelo consórcio W3C (World Wide Web Consortium), essa linguagem, segundo Braganholo e Moro (2009), é usada para "publicação, combinação e intercâmbio de documentos multimídia". Em XML representa-se dados através de marcas flexíveis, ou seja, com qualquer sintaxe para o elemento que define a tag, diferindo de HTML que possui marcas predefinidas como <head> e <body>. Além disso, nos elementos podem conter atributos com seus valores, os quais ajudam na definição de características e são posicionados dentro das marcas. A estrutura de um arquivo XML é descrita como uma árvore hierárquica, onde nodos representam elementos, atributos ou valores e arestas representam a relação entre eles. De acordo com Braganholo e Moro (2009) um documento XML bem-formado deve possuir somente um nó raiz, marcas fechadas, elementos bem aninhados e atributos não repetidos.

A linguagem, devido à sua versatilidade, pode ser usada como meio de comunicação entre sistemas diferentes, tornando possível a interoperabilidade. O que faz com que essa troca de informações ocorra, é o conhecimento da semântica de cada tag através de um vocabulário escrito usando uma linguagem de esquema para XML (DTD ou XML Schema). Em se tratando de aplicações onde a linguagem é utilizada, segundo Braganholo e Moro (2009), pode-se listar Web Services, processamento de características avançadas, como privacidade e estabelecimento de padrões de arquivos, como é o caso do arquivo FOM. Conforme Wilde e Glushko (2008), XML é até agora a tecnologia Web de maior sucesso e ubíqua, juntamente com HTTP, HTML e URI.

2.6 TRABALHOS RELACIONADOS

Esta seção apresenta dois trabalhos existentes que são relacionados ao objetivo principal do trabalho. É feita uma comparação, onde são apresentadas semelhanças e diferenças entre os dois softwares e o trabalho desenvolvido. Características como custo e flexão de versões do FOM também são descritas.

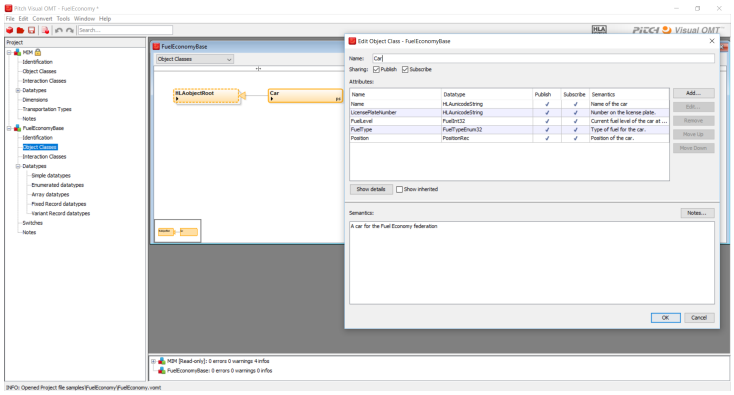
2.6.1 Pitch Visual OMT

O Pitch Visual OMT é um software desenvolvido pela companhia Pitch, usado por milhares de desenvolvedores espalhados pelo mundo, nas áreas de defesa, espacial, gestão do tráfego aéreo, energia, fabricação, tráfego rodoviário e fabricação. Através dele, é possível desenvolver FOMs de acordo com um propósito e também estender e editar os

FOMs de Referência como o RPR. Uma visualização do FOM é criada com as classes de objeto e interações em uma interface gráfica. Além disso, há uma flexibilidade quanto ao versionamento da norma HLA, onde versões mais velhas são suportadas, bem como as mais recentes. Há a possibilidade de comparação entre modelos mais recentes e mais antigos (Pitch Technologies, 2016).

Apesar de todos esses recursos, o Pitch Visual OMT não é gratuito para uso e um pacote acadêmico com três softwares chega a custar R$23.000,00. Além da não gratuidade, deve-se utilizar um dongle para a execução do mesmo, não podendo ser compartilhado ao mesmo tempo em diferentes computadores. Em vista disso, a proposta desse trabalho é o desenvolvimento de um software com as mesmas funcionalidades principais do Pitch Visual OMT, porém livre e gratuito para toda comunidade. Segundo Camara e Fonseca (2007), para países em desenvolvimento como o Brasil, softwares livres são uma maneira de gerar conhecimento tecnológico e criar soluções adaptadas às necessidades da população.

Figura 2.2 – Pitch Visual OMT screenshot



Fonte: (Pitch Technologies, 2016)

2.6.2 Simulation Generator (SimGE)

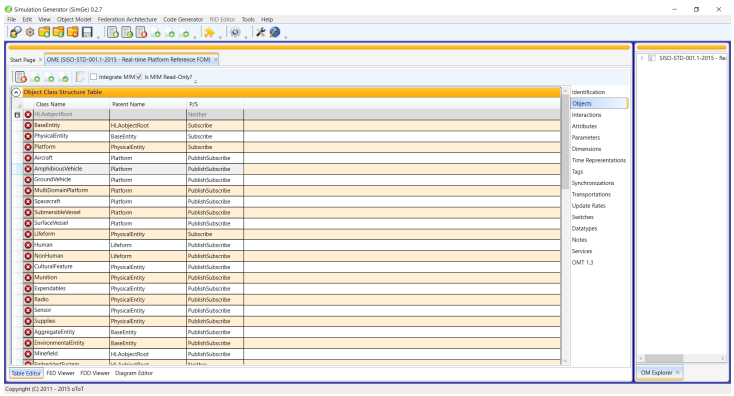
Simulation Generator (SimGE) possui várias funcionalidades como um editor de Modelo de Objeto, design de simulação e desenvolvimento e gerador de código para simulações distribuídas, baseadas na norma HLA. O software permite visualizar a arquitetura da federação e editar algumas propriedades. Há uma aba especialmente para a edição e criação de Modelo de Objeto e templates, onde é possível gerenciar e modificar o FOM. O gerador de código, gera código para uma plataforma de destino, a qual é uma camada de abstração do RTI. Novas versões do software podem não carregar versões mais velhas de Modelos de Objeto (Okan Topçu, 2016). Enfim, o SimGE não é específico para a leitura e edição do FOM, ou seja, há muitas outras características dentro do software que o podem

fazer mais complexo do que o objetivo deste trabalho. Além disso, o software não condiz totalmente com a norma IEEE (2010a) devido ao fato de não reconhecer tipos de dados e gerar erros durante sua execução.

2.6.3 Síntese

A Tabela 2.1 sintetiza e faz uma comparação entre os dois softwares citados e o LE-FOM. O primeiro quesito de comparação é o custo, onde o SimGE e o LE-FOM são gratuitos e o Pitch Visual OMT possui um custo elevado, no pacote acadêmico em que é incluído. O segundo é a funcionalidade, onde o LE-FOM e o Pitch Visual OMT apresentam como utilidade específica do software a edição e leitura de arquivos FOM e o SimGE possui mais de uma funcionalidade. Por último, o quesito flexibilidade de versão, onde o Pitch Visual OMT é o único que suporta versões de FOMs anteriores, como a de 2000. Portanto, é possível concluir que o LE-FOM é um software ideal em comparação aos demais, para desenvolvedores que utilizam a versão mais recente do FOM até presente momento.

Figura 2.3 – SimGE screenshot



Fonte: (Okan Topçu, 2016)

Tabela 2.1 – Tabela comparativa entre os trabalhos relacionados

Software	Custo	Funcionalidade	Flexibilidade de versão
LE-FOM	sem custo	específica	não
Pitch Visual OMT	R$ 23.000,00 (pacote)	específica	sim
SimGE	sem custo	não específica	não

Fonte: Pitch Technologies (2016), Okan Topçu (2016) e próprio autor.

3 MODELAGEM E REQUISITOS DA APLICAÇÃO

Neste capítulo são detalhados os requisitos fundamentais ao software desenvolvido definidos a partir do problema encontrado. Ademais, a metodologia da implementação a qual orienta o desenvolvimento é descrita gradativamente. A fim de um melhor entendimento do trabalho bem como auxiliar a modelagem do protótipo e definição de requisitos, são utilizados diagramas UML (Linguagem de Modelagem Unificada), tais como o Diagrama de Atividades e o Diagrama de Casos de Uso.

3.1 REQUISITOS DA APLICAÇÃO

Os requisitos da aplicação foram definidos a partir do problema em questão. Desenvolvedores que trabalham com simulação e que utilizam a norma HLA sofrem com dificuldades de manipulação e visualização dos objetos e interações necessárias para a customização e entendimento do ambiente simulador. Essa adversidade é devido ao fato de que o Modelo de Objeto da Federação é um documento extenso, onde contém centenas de elementos dispostos de forma hierárquica. Por exemplo, um desenvolvedor precisa achar o tamanho em bits do tipo de dado que um certo atributo usa. Além de ter que localizar onde está a classe de objeto a qual pertence este atributo, ele precisa ir até o fim do documento onde estão os tipos de dados, procurar dentre os tipos complexos o tipo de dado básico para então encontrar a característica que no caso é o tamanho do tipo de dado.

Tendo em vista isso, o primeiro requisito definido foi a visualização dos elementos do arquivo FOM por meio de uma interface gráfica. Através dela, o desenvolvedor pode ver por meio de tabelas específicas para cada elemento, as tags e seus respectivos valores. Ademais, seguindo o modelo hierárquico do arquivo, foi fundamental a exposição de cada elemento do nível acima, ou seja o nó pai, para cada componente que se aplica à essa circunstância. Desta maneira, não é necessário percorrer várias linhas para encontrar a unidade desejada.

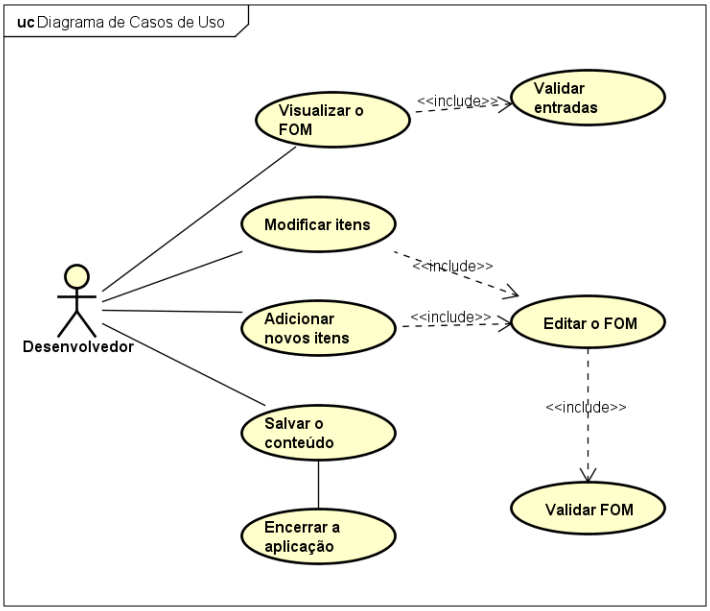
Segundo requisito estabelecido foi a edição do modelo FOM. Uma vez que os componentes são encontrados, eles poderão ser editados através da interface gráfica. A modificação é restrita, seguindo a norma IEEE (2010a), a qual delimita certas condições aos componentes. Além disso, é possível que se adicionem novos elementos aos já existentes, oferecendo o mesmo suporte dado à edição dos já existentes.

Por fim, a validade dos dois requisitos anteriores é considerada nula caso não exista uma validação do modelo que se deseja ler e do que é gerado com modificações. Com base no XML Schema de cada FOM, determinou-se como requisito a comprovação do ar-

quivo gerado diante da norma IEEE (2010a). É analisado cada componente, com propósito de verificar se algum obrigatório está faltando e então alertar o usuário do software.

O Diagrama de Casos de Uso, na Figura 3.1, ilustra os requisitos através de casos de uso. O desenvolvedor inicialmente pode visualizar os elementos. Logo após ele é capaz de modificar os itens já existentes e lidos do FOM e adicionar novos itens para o documento final. Para gerar um novo documento com as modificações o desenvolvedor deve salvar as informações, as quais são validadas e então a aplicação é encerrada.

Figura 3.1 – Diagrama de Casos de Uso



Fonte: Próprio autor

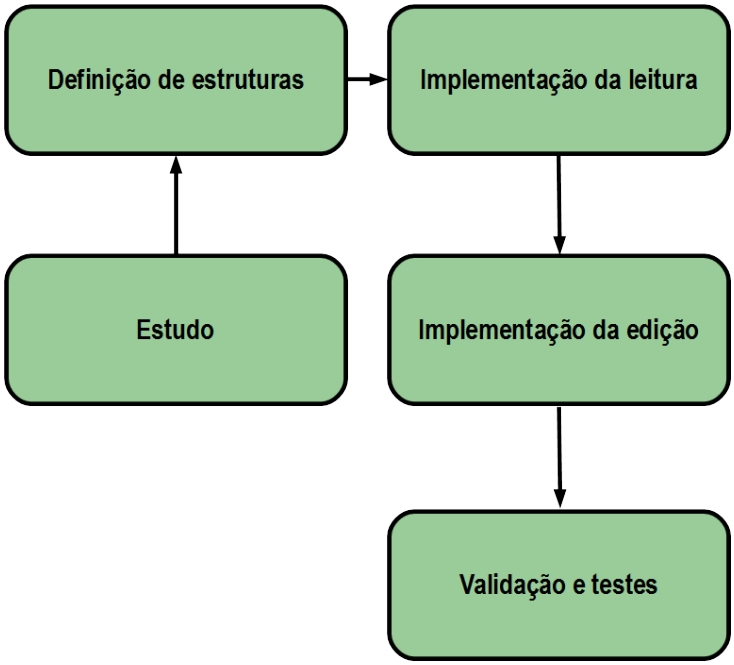
3.2 METODOLOGIA E MODELAGEM

A metodologia de implementação segue uma sequência de passos definidos a fim de atender aos requisitos citados na seção 3.1. O modelo de desenvolvimento de software utilizado foi o Modelo em Cascata, onde estão presentes as fases de análise de requisitos (previamente à metodologia), projeto, implementação e testes (validação), excluindo somente a etapa de manutenção. Como primeira etapa da metodologia, é necessário obter-se uma contextualização do assunto em que o modelo de objeto a ser trabalhado se engloba. Em vista disso, é realizado um estudo sobre o padrão High Level Architecture, para o entendimento do funcionamento e termos utilizados. Ademais, um estudo detalhado sobre cada componente do FOM, com suas restrições, predefinições e seus itens padrões presentes em qualquer modelo de objeto que segue a norma atual também é efe-

tuado. Com essa análise e conhecimento, uma parte da validação pode ser executada e opções podem ser pré-dispostas para o momento da edição. Levando em consideração a linguagem que o FOM é descrito, procura-se entender a linguagem de marcação XML para que haja clareza no momento em que a leitura, edição e principalmente validação são desenvolvidas. Após o conhecimento adquirido, é possível iniciar uma pesquisa sobre uma estrutura de dados que melhor comporta os elementos do modelo de objeto. A forma em que os elementos do FOM são organizados precisa ser planejada previamente ao desenvolvimento do software. São analisadas as hipóteses da estrutura em listas, filas, pilhas e árvores, visando atingir aos requisitos de uma maneira eficiente e não desconsiderando a hierarquia das informações. Juntamente com essa análise, inicia-se uma pesquisa de como o parser para a leitura do arquivo é desenvolvido, pois a manipulação da estrutura está diretamente relacionada com a forma com que é percorrido. Posteriormente, o software de fato começa a ser desenvolvido.

Como padrão de arquitetura do sistema escolheu-se o Model-View-Controller (MVC), sendo a view as classes estendidas do Swing, controller as classes que fazem o parser, montam as views e reescrevem as alterações no arquivo e model comporta o modelo dos dados. Inicialmente, a leitura do arquivo XML FOM é realizada. Essa etapa é a principal para o trabalho, pois concentra a criação da representação gráfica dos elementos e a disposição dos mesmos para a edição. Como forma de organização é feita uma checklist de tarefas que devem ser produzidas tais como, a leitura propriamente dita, a criação de uma interface amigável com Swing e uma revisão de software focando item a item pertencentes à checklist. Tendo os elementos dispostos na interface gráfica, propicia-se então a edição. Nesse estágio consta o desenvolvimento das funcionalidades que possibilitam a modificação do arquivo FOM. A interface gráfica é adaptada para a realização das edições dos elementos. Assim como a etapa da leitura, há uma checklist com tarefas como o ajuste da interface para edição com opções que se adequem à norma IEEE (2010a), a escrita do documento gerado e a revisão de software. As revisões de software são realizadas com o intuito de procurar erros para que após a entrega do software o usuário não presencie defeitos e consequentemente aumente a qualidade do trabalho. Portanto, considera-se a compreensão de todos os elementos presentes no modelo de objeto, sua disponibilização na interface, sua integridade hierárquica e sua pertinência à norma utilizada. Os erros encontrados são corrigidos usando a Depuração Automática e Eliminação da Causa. Por fim, a etapa de validação e testes. A validação é realizada de acordo com o XML Schema de cada modelo, verificando se elementos obrigatórios estão de fato presentes no arquivo gerado após a edição. Caso contrário, o FOM não será compatível à norma e, consequentemente, inutilizado por um federado. Além disso são feitos testes com diferentes FOMs que estão de acordo com a norma IEEE (2010a) a fim de garantir uma maior qualidade do trabalho. Todas as etapas descritas para a implementação do software são ilustradas sequencialmente na Figura 3.2.

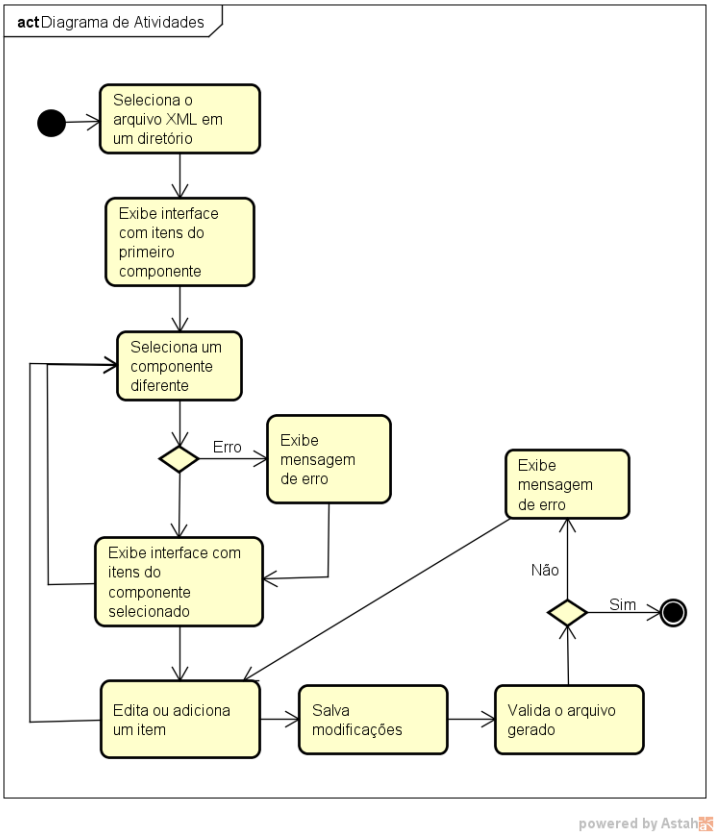
Figura 3.2 – Etapas da metodologia de implementação



Fonte: Próprio autor

As atividades são apresentadas conforme a Figura 3.3 no Diagrama de Atividades, onde pode ser possível visualizar também o fluxo do software. Inicialmente é escolhido um arquivo XML pelo seletor de arquivos e o mesmo é lido através de um parser. Uma interface gráfica então é exibida com o primeiro componente do FOM. Pode-se selecionar um componente diferente, e caso haja erro, o usuário é avisado, mas seus itens são exibidos de qualquer maneira. Após a exibição do elemento desejado, é possível a edição e adição de novos itens ao arquivo. As modificações são salvas quando requisitadas e então ocorre a validação. Caso o arquivo não possua os componentes obrigatórios e siga o padrão de acordo com a norma IEEE (2010a), o programa permanece com as alterações com o intuito de o usuário corrigir os erros. O programa só é encerrado com as alterações salvas, caso o arquivo seja validado.

Figura 3.3 – Diagrama de Atividades



Fonte: Próprio autor

4 DESENVOLVIMENTO DO LE-FOM

Este capítulo descreve detalhadamente os passos da implementação do software LE-FOM. Além disso, é descrita a estrutura utilizada para a manutenção dos dados, bem como a biblioteca escolhida para auxiliar a leitura do modelo de objeto. Uma breve explicação é dada sobre cada componente do FOM, a fim do melhor entendimento do que se trata o modelo de objeto.

A implementação está dividida em quatro etapas sequenciais, devido a dependência uma da outra. Na primeira etapa (Seção 4.1.1.) foi definida uma biblioteca para auxiliar na leitura do arquivo FOM. Na segunda etapa (Seção 4.1.2.) definiu-se a estrutura de dados que melhor comportaria os elementos do FOM, de maneira que mantivesse a hierarquia do arquivo. A partir disso, na terceira etapa (Seção 4.1.3.), a leitura foi implementada, juntamente com a visualização de cada componente em uma interface gráfica. Na quarta etapa (Seção 4.2.1 e 4.2.2.), buscou-se então levar em conta as restrições da norma, para que o desenvolvedor pudesse realizar uma edição orientada e válida. Ainda, nessa etapa ocorre a reescrita do modelo de objeto com as mudanças realizadas. A seguir cada uma das etapas é melhor detalhada.

4.1 IMPLEMENTAÇÃO DA LEITURA

Nesta seção é apresentada como a leitura do arquivo FOM foi realizada. Dividida em três subseções, sendo a primeira a definição da biblioteca utilizada como parser. É feito uma comparação entre duas APIs encontradas, ressaltando os motivos da escolhida. Na segunda subseção, a estrutura utilizada para comportar os dados lidos é descrita. Várias opções de estruturas são listadas, entretanto, bem como a biblioteca, é explicada a razão da estrutura adotada. Na última subseção, são mostrados screenshots de telas do programa, mostrando como são exibidas as informações aos desenvolvedores.

4.1.1 Definição da biblioteca de leitura

Partindo do fato que a linguagem escolhida para a implementação do LE-FOM foi Java, procurou-se bibliotecas existentes nessa linguagem para auxiliar na leitura do arquivo XML. Durante a pesquisa, foram encontradas duas APIs chamadas Document Object Model (DOM) e Simple API for XML (SAX). Uma comparação entre as duas foi realizada a fim de decidir a melhor opção para este software. Uma descrição mais sucinta dessa comparação pode ser visualizada na Tabela 4.1.

Segundo Kale et al. (2016), a representação do DOM é através de uma árvore armazenada totalmente em memória. Em consequência disso, a API DOM é menos eficiente em se tratando de tempo e armazenamento, pois para cada acesso e leitura a velocidade de processamento é reduzida. Portanto, para arquivos grandes não é recomendado o uso do DOM. Entretanto, há vantagens como a facilidade do uso onde uma árvore de objetos é proporcionada e para aplicações que interagem com o XML, permite usuários a manipular e acessar os elementos da árvore armazenada em memória (KALE et al., 2016).

Por outro lado, o SAX é serial e dirigida por eventos, ou seja, a cada elemento ou atributo encontrado no arquivo, há uma callback invocada. Existem eventos para início de elemento com seus atributos, para o texto do elemento e para o fim do mesmo. Diferindo do DOM, o SAX lê somente uma vez o arquivo sequencialmente e não armazena dados em memória. Sendo assim, a API SAX é eficiente em se tratando de arquivos extensos.

Para esse trabalho, a melhor opção foi a biblioteca SAX. O DOM é simples para a manipulação de elementos, no entanto ele possui uma estrutura engessada, não podendo ser moldada às necessidades do programa. Além disso, certos componentes possuem valores de tags dependentes a outros componentes. Levou-se em consideração o tempo que levaria para encontrar essa dependência para a disponibilização na interface para edição para cada item. Portanto, o tempo foi um fator secundário na decisão, a fim de que o usuário não tivesse que esperar a cada carregamento da interface.

Tabela 4.1 – Tabela comparativa entre as APIs DOM e SAX

	Velocidade	Armazenamento	Capacidade de manipulação	Arquivos
DOM	Lenta	Alto	Sim	Menores
SAX	Rápida	Nenhum	Não	Grandes

Fonte: (SAX, 2016) e (KALE et al., 2016)

4.1.2 Definição da estrutura de dados

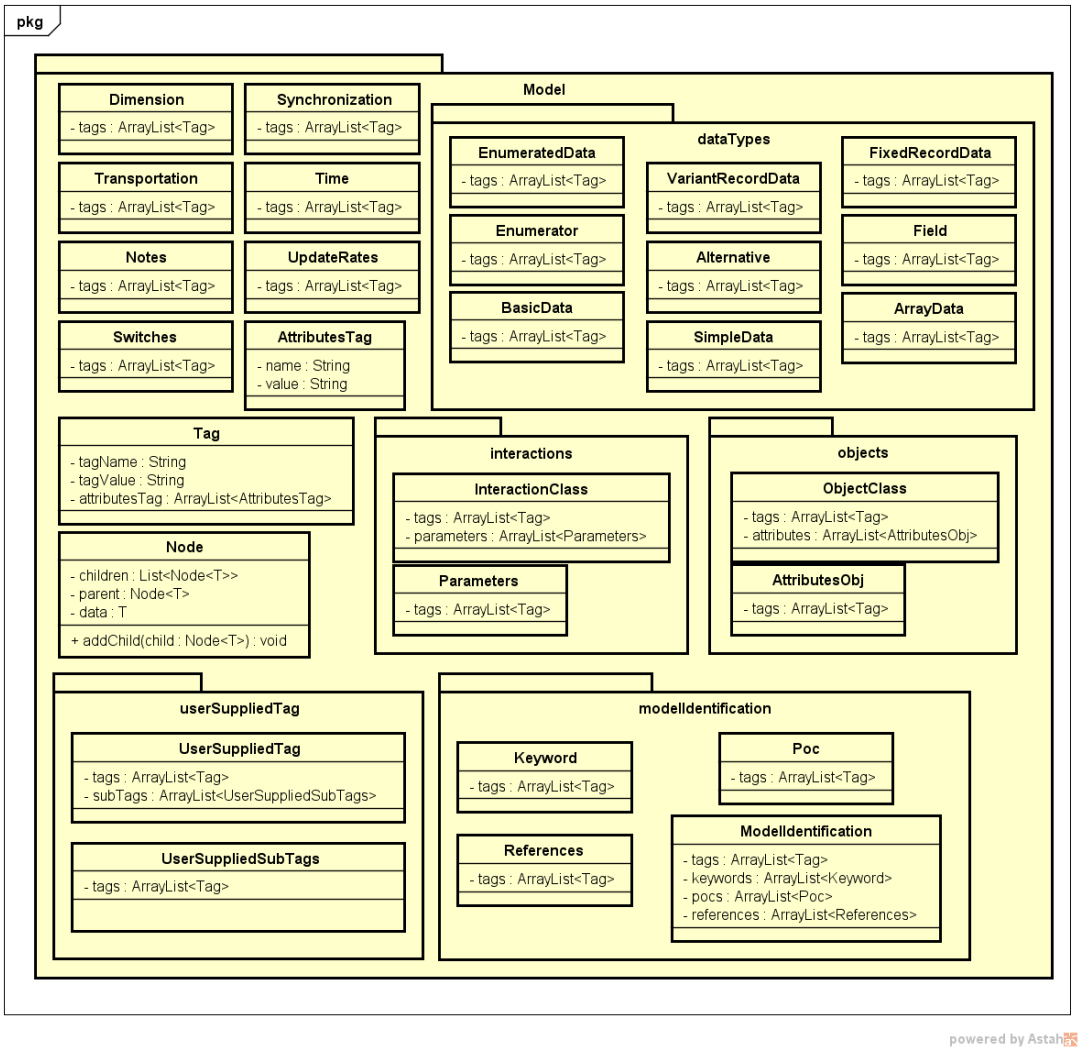
Em razão da escolha da biblioteca, foi necessário a definição de uma estrutura para armazenar os dados temporariamente, pois o SAX não possui suporte a isso. Foram elencadas estruturas como fila, pilha, lista e árvore. Analisando a estrutura do arquivo FOM e levando em conta sua hierarquia, optou-se pelo armazenamento em árvore, onde cada um dos nodos contém o seu pai, filhos e dados genéricos, como no Listing 4.1. Cada componente do FOM possui sua árvore, a qual é atributo de uma classe com padrão Singleton, ou seja, uma classe a qual possui somente uma instância global e permite criar objetos únicos também acessíveis à toda aplicação. A estrutura do nodo é genérica, sendo utilizada para qualquer um dos componentes.

Listing 4.1: Nodo genérico da árvore

```
1 public class Node<T> {
2     private List<Node<T>> children = new
          ArrayList<Node<T>>();
3     private Node<T> parent = null;
4     private T data = null;
5
6     public Node(T data) {
7         this.data = data;
8     }
9
10    public Node(T data, Node<T> parent) {
11        this.data = data;
12        this.parent = parent;
13    }
14
15    /* GETTERS AND SETTERS */
16 }
```

O que é considerado um dado no nodo é um modelo de informações criados para cada componente do FOM. Para a melhor representação desses modelos, o diagrama da Figura 4.1 foi feito, representando cada pacote para cada componente. Esses modelos compõe a estrutura que armazena os dados lidos do arquivo FOM. Essas classes são necessárias para a definição de cada componente, diferenciando-os no momento da leitura e consulta de informações, como os atributos de tags. Geralmente, os modelos possuem uma lista de uma classe chamada Tag. Essa classe possui um nome, um valor e uma lista de atributos (podendo ser vazia), assim como toda tag em XML. Além disso, componentes que possuem hierarquia, também contém uma lista de objetos pertencentes à classe do seu filho. Como exemplo, no pacote Interactions, onde a classe InteractionClass possui suas informações na lista de tags e informação sobre seus filhos que no caso são chamados parameters e pertencem a classe Parameter que está no mesmo pacote. Cada pacote com mais de uma classe, com exceção do Model, possui hierarquia com dois ou mais níveis.

Figura 4.1 – Diagrama de classes dos modelos



Fonte: Próprio autor

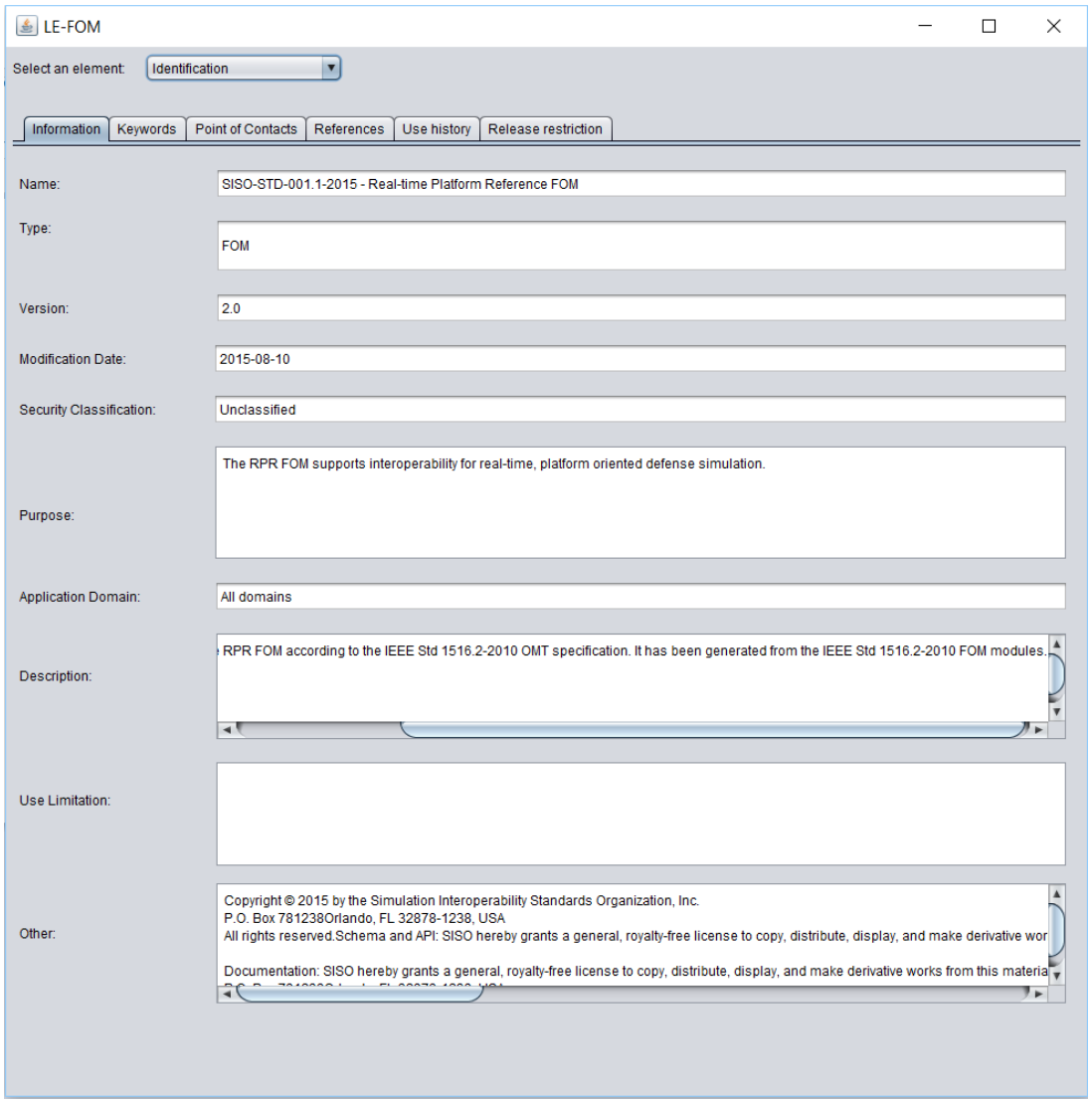
4.1.3 Descrição e apresentação da interface

Após a leitura e organização da estrutura, os dados são dispostos na interface gráfica. Foi criado um painel com abas para cada tipo de componente do FOM. Os componentes podem ter uma ou mais abas, dependendo da quantidade hierárquica que possuem. Por exemplo, se um dado componente possui cinco elementos filhos com suas estruturas diferentes, o painel terá seis abas em sua totalidade, como é o caso na Figura 4.2. Os dados são carregados da estrutura previamente à sua visualização, a fim de tornar o processamento de interação com o software mais rápido.

A primeira tela do LE-FOM é representada na Figura 4.3. De início o usuário deve inserir o caminho do diretório onde está o arquivo XML FOM ou selecioná-lo no seletor de arquivos. O seletor só permite arquivos XML e caso não tenha componentes de um FOM ou não esteja no formato UTF-8, um erro é disparado e a aplicação encerra. Quando o

botão Ler é clicado o parser é iniciado, gerando a Figura 4.2.

Figura 4.2 – Parte da identificação do modelo

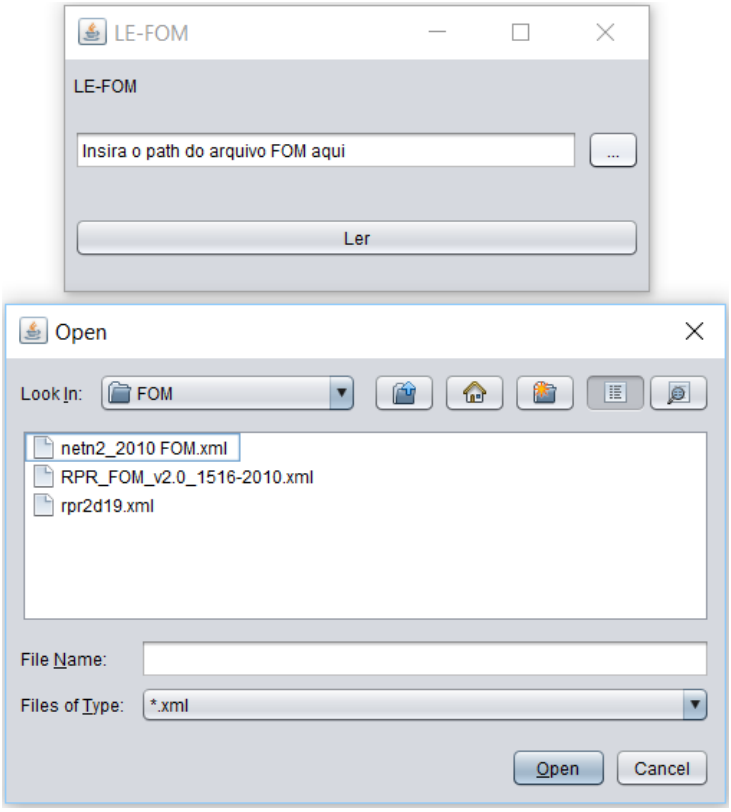


Fonte: Próprio autor

A Figura 4.2 mostra a tela inicial, após a leitura do arquivo com o elemento de identificação selecionado. Na aba selecionada quando feito o screenshot, aparece informações sobre a identificação do modelo FOM, tais como data de modificação, nome e versão. Essa é a única tela em que os dados não são organizados em tabelas, devido a singularidade da ocorrência da estrutura dessas informações.

Na figura 4.4 são mostrados todos os componentes do FOM presentes na norma IEEE (2010a). Quando um elemento é selecionado, a view muda mostrando as abas e dados correspondentes. Os dados da estrutura são carregados na view ao primeiro clique, sendo que alguns componentes maiores demoram cerca de quatro segundos para carregar. Entretanto, na próxima seleção esse tempo de espera será zero.

Figura 4.3 – Seletor de arquivos XML



Fonte: Próprio autor

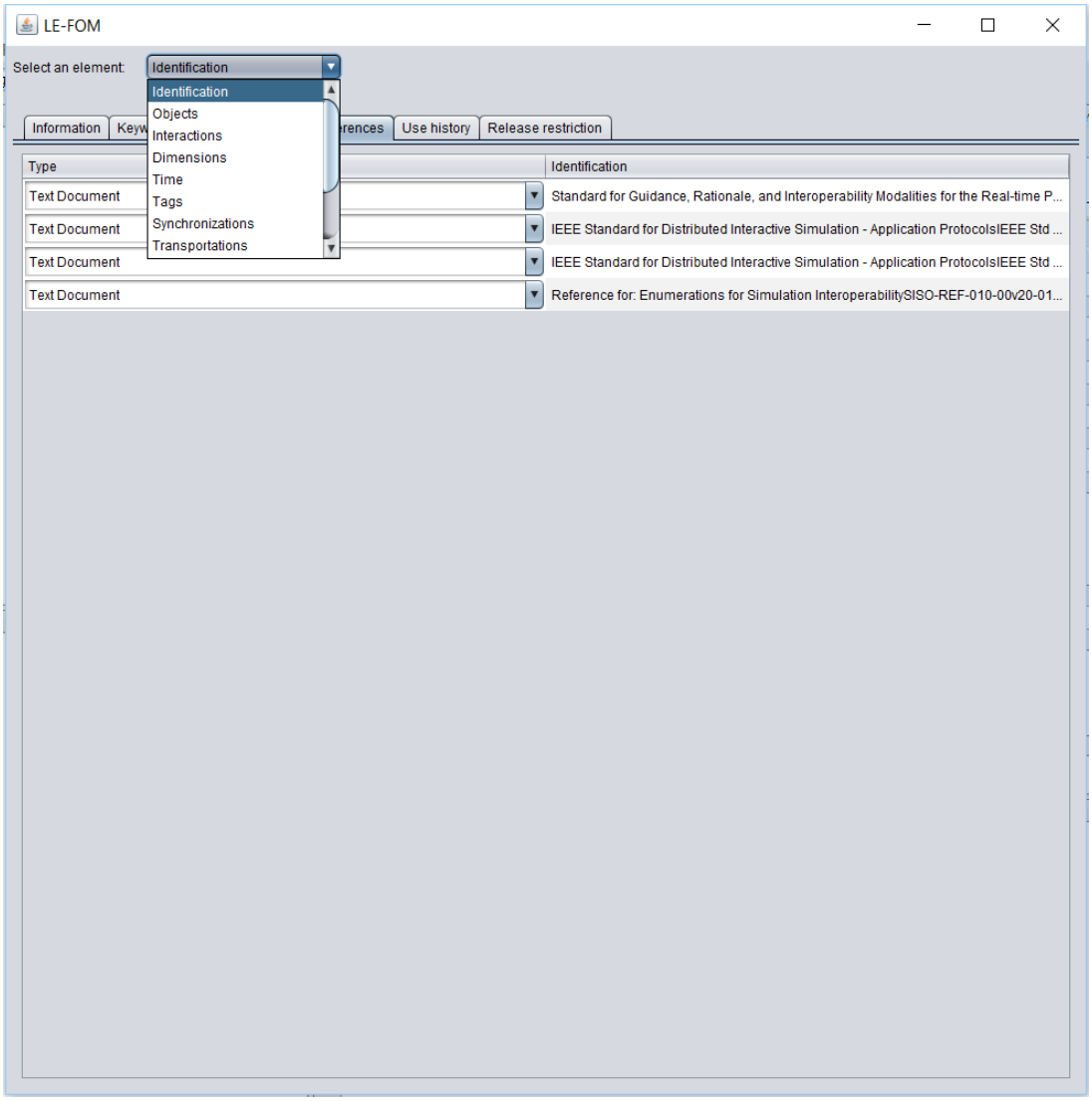
4.2 IMPLEMENTAÇÃO DA EDIÇÃO

Nesta seção é descrita como a edição do arquivo FOM foi implementada, após a leitura do mesmo. Ela foi dividida em duas subseções. A primeira apresenta a adaptação realizada na interface para que o usuário pudesse alterar campos existentes bem como adicionar novos. Na segunda subseção é descrita a geração do arquivo com as modificações realizadas pelo usuário.

4.2.1 Adaptação da Interface

O primeiro passo para a habilitação da edição do arquivo FOM foi a adaptação da interface. A tabela com a disposição dos elementos já foi configurada para ser editável no momento da leitura, visando às mudanças que ocorreriam na próxima etapa. Entretanto, há exceções de tabelas as quais os itens não podem ser alterados, uma vez que possuem algum padrão de nomes ou valores de acordo com a norma IEEE (2010a). Além disso, para valores numéricos, é disponibilizado um componente gráfico específico, com o limite mínimo de acordo com a norma. Na Figura 4.5 há um exemplo de tipo de dado que contém

Figura 4.4 – ComboBox com os componentes do FOM

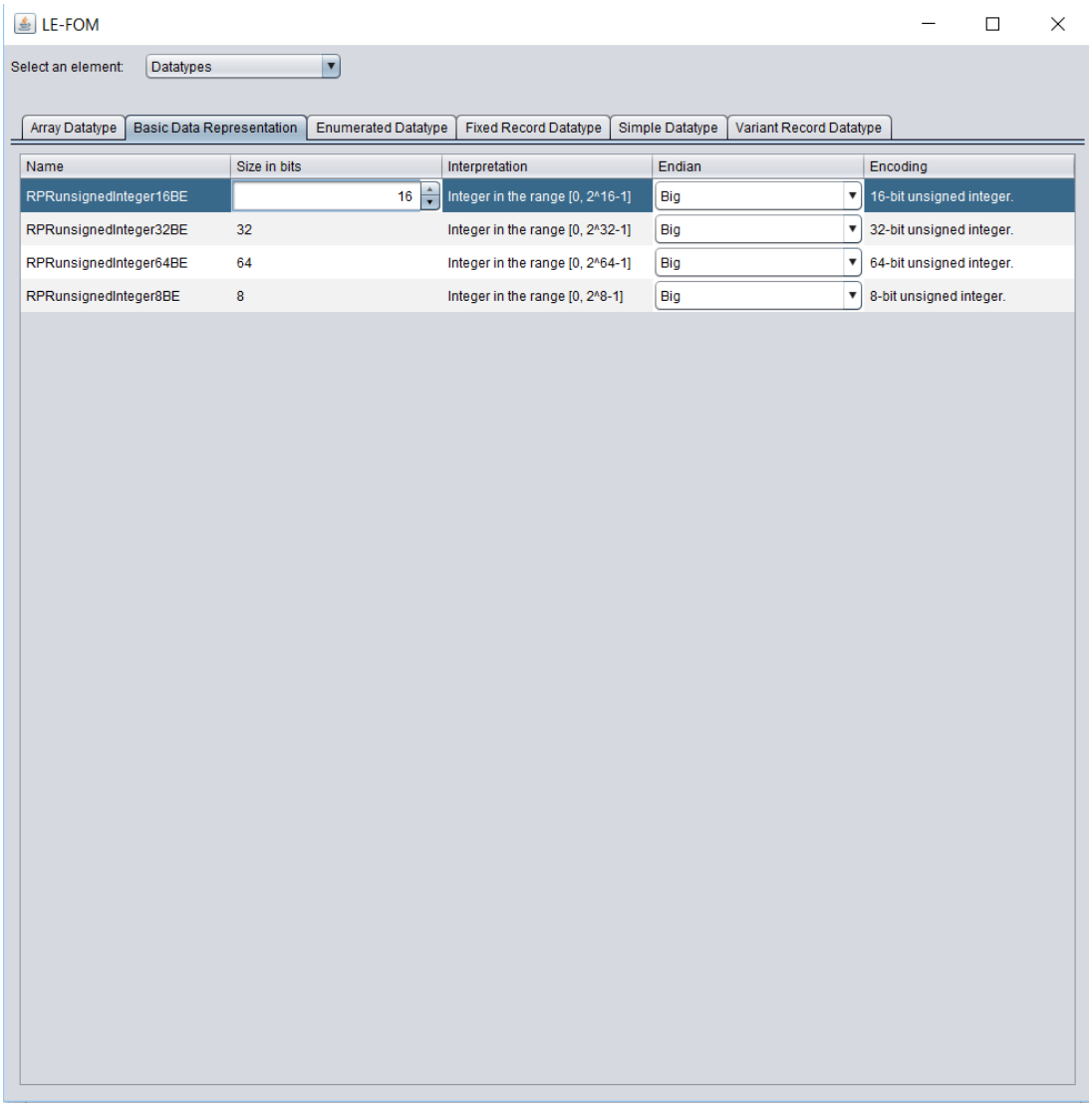


Fonte: Próprio autor

um elemento numérico. Esse campo terá o valor mínimo de acordo com o que rege a norma. A célula da tabela foi customizada especialmente para caracteres numéricos.

Além disso, certos componentes possuem uma característica dentro de sua estrutura a qual depende de outro componente que é definido dentro do arquivo. A edição para essa característica será um ComboBox com todos os itens que tiverem definidos naquele arquivo. Na Figura 4.6 a aba selecionada apresentada é a dos atributos de uma classe de objeto. Cada coluna representa uma parte da estrutura que forma como um todo um atributo, exceto a segunda coluna que representa a classe de objeto a qual ele pertence. Essa figura possui um detalhe importante que é o ComboBox com itens definidos. Isso é porque um atributo tem um tipo de dado, definido dentro do arquivo FOM, o qual é setado inicialmente. Todavia, o usuário é capaz de mudar esse tipo de dado, uma vez que ele esteja descrito no FOM. Por isso, todas as opções são dadas a ele, de forma que seja

Figura 4.5 – Componente numérico representado

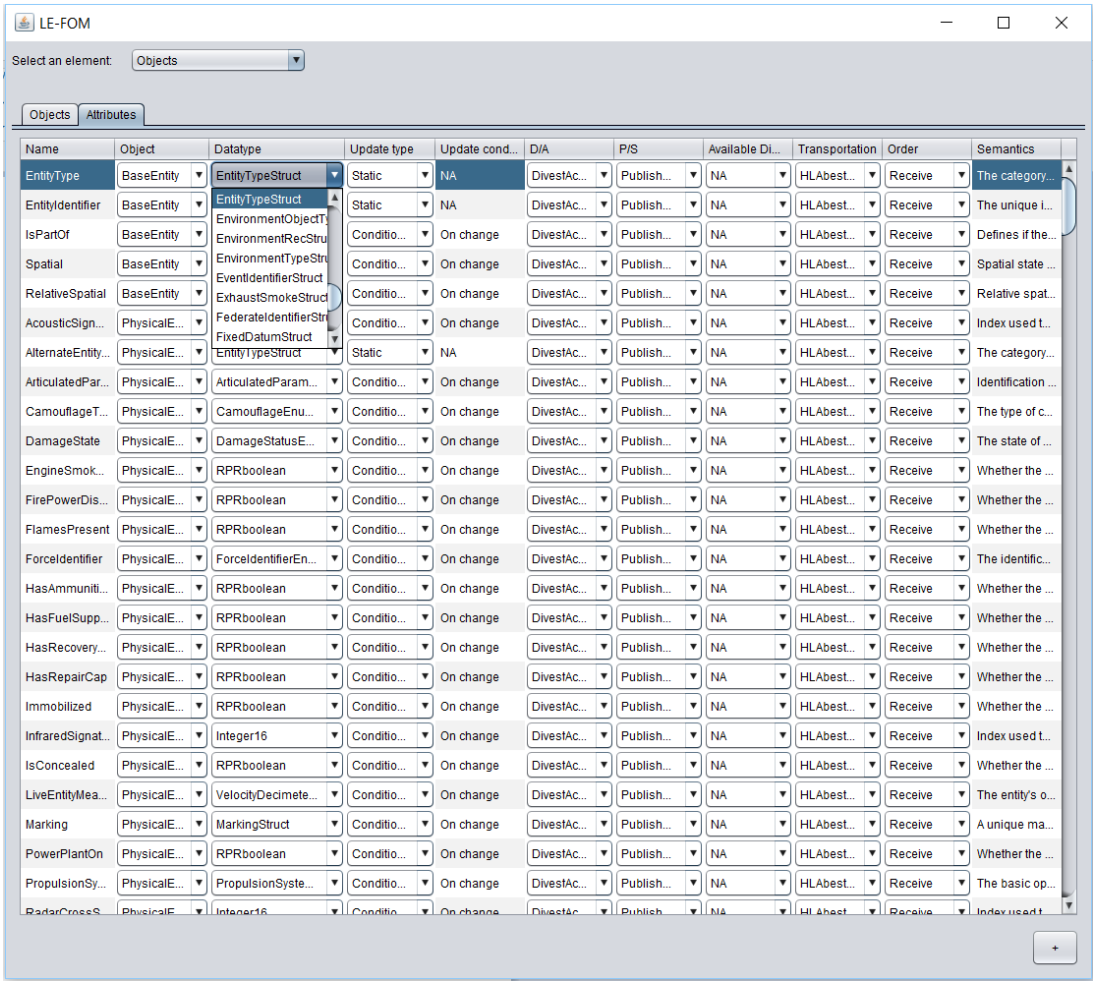


Fonte: Próprio autor

consistente e garanta que o tipo de dado exista naquele modelo. Todos os outros ComboBoxes presentes nas tabelas seguem essa condição. Além disso, há alguns itens padrões, definidos pela norma IEEE (2010a), os quais são adicionados para que apareçam também como um opção ao usuário.

Para adição de novos itens, um botão foi incluído na interface, mostrado na Figura 4.7. A ação do botão dispara a inserção de uma nova linha vazia na respectiva tabela, para que o usuário possa personalizar de acordo com seus interesses. Os ComboBoxes com itens pré-preenchidos também são disponibilizados na nova inserção, para garantir que a norma seja atendida. Ademais, um outro botão com o intuito de acionar a exportação do novo arquivo FOM bem como sua validação foi incorporado à interface e também é ilustrado na Figura 4.7.

Figura 4.6 – Aba dos Atributos das Classes de Objeto



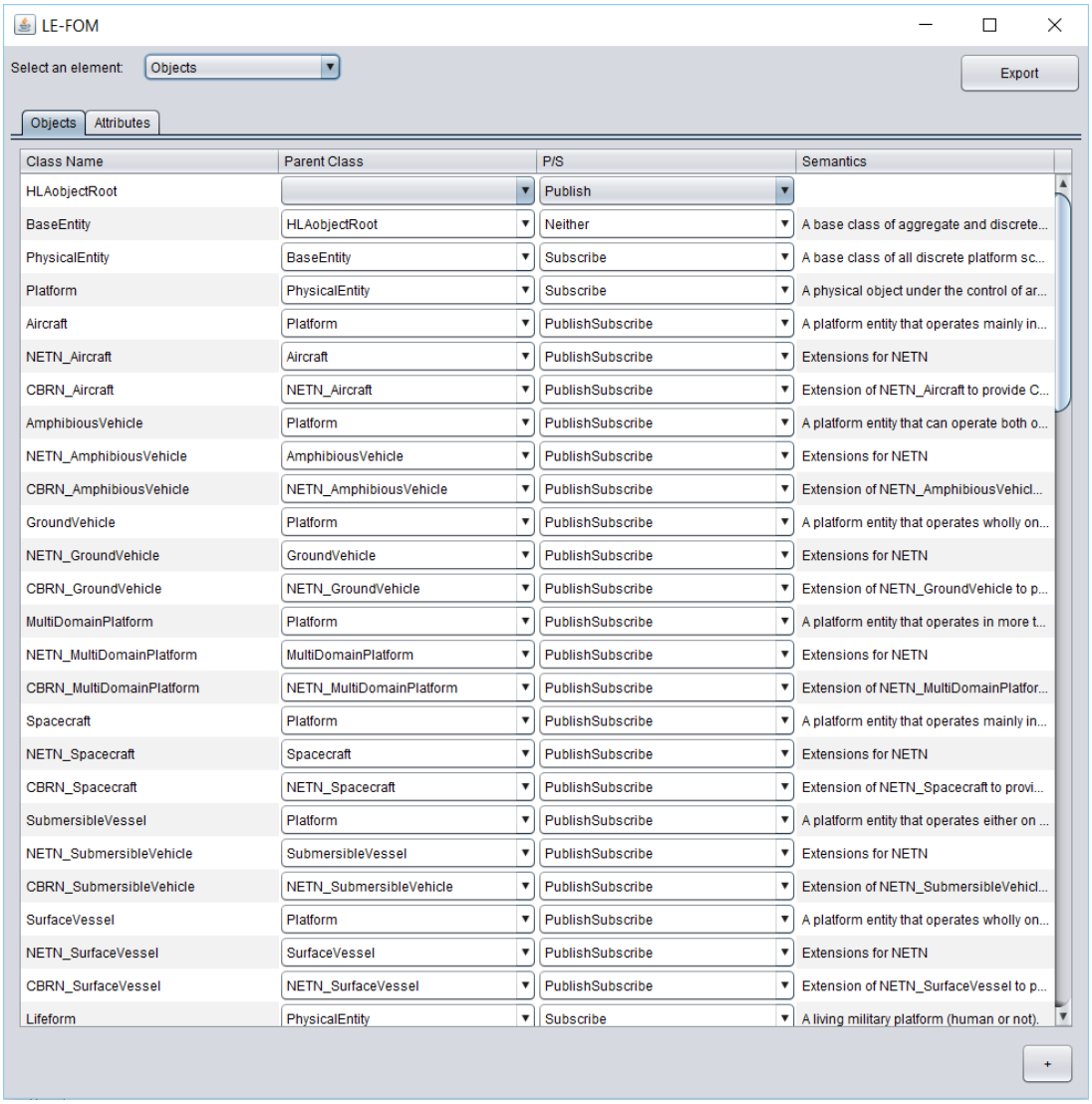
Fonte: Próprio autor

4.2.2 Geração do arquivo editado

A geração do novo arquivo é iniciada no momento em que o usuário pressiona o botão "Export". As tabelas de cada componente são lidas, linha a linha, gerando sua representação no arquivo XML. Como o carregamento de todas as estruturas para as tabelas foi realizada no início da execução do programa, não há problemas se o usuário não visualizar a tabela de algum componente. Além disso, para a implementação de uma exclusão de item, foi considerado o preenchimento do mesmo como vazio. Colunas que são fundamentais aos componentes como por exemplo, nome e tipos de dados, quando estiverem vazias são interpretadas como removidas e então, o item não é escrito no documento. Com o intuito de não modificar o FOM original, é criado um novo arquivo com as alterações. O usuário pode escolher o nome do arquivo a ser gerado, antes da geração e validação, e o mesmo é salvo no mesmo diretório o qual o arquivo de entrada se encontra.

Os atributos das tags dos componentes não são salvos juntamente às suas tabelas, sendo somente mantidos na estrutura. Sendo assim, deve-se procurar os atributos na ár-

Figura 4.7 – Interface com novas funcionalidades



Fonte: Próprio autor

vore de cada componente, mantida da leitura no momento da geração do arquivo. Para que eles fossem escritos nas tags em que pertenciam anteriormente, usou-se uma estratégia de contagem de nodos. O número da linha na tabela representa a posição do item na sua respectiva árvore, ou seja, é feito o mesmo processo de percurso ao preencher as tabelas, porém contando os nodos a fim de achar a posição correta. Esse método foi considerado ideal, uma vez que novos itens não são adicionados no meio da tabela e a procura através dos valores de tag não seria confiável, uma vez que ele é editável. Ademais, para atributos de componentes que possuem filhos, basicamente, o mesmo processo é realizado. Todavia, para encontrar os números das linhas dos filhos de um certo elemento, uma busca com o nome do pai em sua respectiva coluna é efetuada. O número de cada linha que representa a posição de um filho na árvore então é retornado. Como filhos podem mudar seus pais de acordo com o que a interface disponibiliza, eles não são buscados de acordo

com o pai, mas sim conforme sua posição dentre todos os nodos do mesmo tipo dentro de sua árvore.

4.3 VISÃO GERAL DA IMPLEMENTAÇÃO

A implementação do software LE-FOM baseou-se na arquitetura MVC, portanto suas classes estão divididas em três pacotes. Na Figura 4.8 há a representação de algumas das classes, com alguns métodos e atributos contidas no software. Primeiramente o pacote Model, que é conectado diretamente com o Controller e que possui classes que representam as estruturas de dados. Após, o pacote Controller, o qual gerencia os dados na interface e estruturas bem como executa as funções necessárias ao funcionamento do software. Por fim, o pacote View que também se conecta com o Controller. Sua responsabilidade é a apresentação dos dados para leitura e edição através dos componentes da interface. Algumas das classes necessárias para a implementação do LE-FOM, bem como sua estrutura, são descritas abaixo:

- Tag: classe responsável por manter o nome das tags, seus valores e seus atributos representados pela classe AttributesTag;
- AttributesTag: mantém o nome e valor de um atributo;
- Node: classe que representa a estrutura da árvore genérica que mantém cada componente;
- Time: classe de exemplo às classes que não possuem mais de dois níveis de hierarquia, e consequentemente, só suas tags com seus atributos;
- InteractionClass: classe de exemplo às classes que possuem filhos de tipos diferentes aos seus, no caso a classe Parameter;
- Parameter: classe de exemplo às classes que são filhos de pais de diferentes tipos, no caso a classe InteractionClass;
- FileGetter: classe responsável por possibilitar o usuário escolher o caminho até o diretório onde está o arquivo FOM e chamar a função que inicia a leitura;
- InteractionsView: classe que representa views de elementos que possuem uma mais de dois níveis de hierarquia. No caso, essa classe apresenta as interações e seus parâmetros, contendo um painel com duas abas. Cada aba possui uma tabela e um botão que possui a funcionalidade de adicionar linhas à tabela;

- TimeView: classe que representa views de elementos que não possuem mais de dois níveis de hierarquia. Possui somente uma aba com uma tabela e, no caso, não possui botão para adicionar, pois a norma não permite adicionar novos itens. Algumas classes permitem a inserção de novos itens e portanto, possuem o botão;

- TabsPanel: classe que possui os controllers e aciona funções nos mesmos quando um componente é selecionado no ComboBox. Além disso, dispõe o botão Export, o qual dispara a função para a geração do novo arquivo;

- Parser: classe responsável por realizar a leitura do arquivo e armazená-lo nas estruturas. Possui os métodos da biblioteca SAX, que direcionam aos métodos de criação, inserção de tags, valores e filhos de cada componente específico. Além disso, possui constantes que definem o nome cada componente conforme a norma;

- InteractionsController: classe que representa controllers de elementos que possuem mais de dois níveis de hierarquia. Sua principal função é ler da sua respectiva árvore e escrever os itens em sua tabela na view. Possui constantes com valores padrões definidos na norma, para o preenchimento automático na interface. Ademais, há um método para a validação dos valores de cada tag, a fim de manter a consistência à norma. Há métodos para a adição de um novo item às tabelas, com colunas que possuem ComboBoxes pré preenchidos. Além disso, para a escrita há métodos que procuram pais e filhos pelo número da linha na tabela, ou seja, pela contagem dos nodos na estrutura, a qual corresponde com o número da linha;

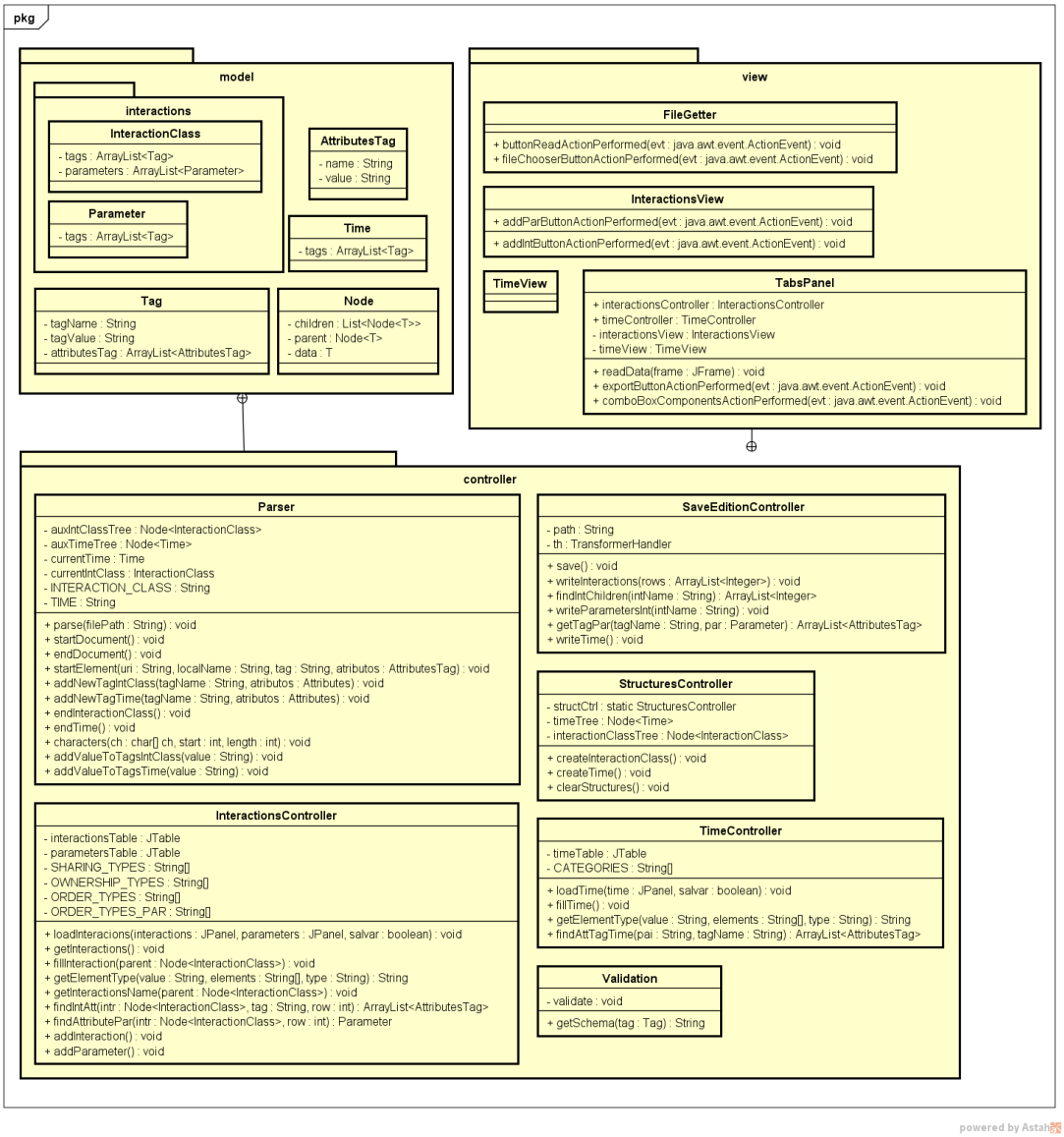
- TimeController: classe que representa controllers de elementos que não possuem mais de dois níveis de hierarquia. Possui basicamente as mesmas características da InteractionClass, com exceção de métodos que se tratam dos filhos. Para a escrita, há um método em que procura na estrutura atributos de tags, de acordo com seu pai que não pode ter o nome alterado de acordo com o padrão;

- SaveEditionController: classe que contém os métodos de geração do novo arquivo. Há métodos de escrita, onde cada tabela da view é lida para a geração de cada componente. Ainda, para componentes que se aplicam, há métodos que procuram o número da linha em que os filhos de um certo pai estão retornando uma lista. Posteriormente, quando achados os objetos filhos, há métodos que procuram por atributos nas tags do mesmo;

- StructuresController: classe Singleton que armazena todas as estruturas lidas em diferentes árvores, até a finalização do LE-FOM;

- Validation: classe a qual possui o método que valida o arquivo gerado conforme o XML Schema do FOM de entrada.

Figura 4.8 – Diagrama de Classe



Fonte: Próprio autor

5 TESTES E VALIDAÇÃO

Após a geração do novo arquivo com edições realizadas, passou-se para a fase de validação. A validação neste software consiste em analisar o XML Schema, a fim de conferir se a estrutura gerada está de acordo ou não. Para o desenvolvimento da validação, primeiramente foi realizada uma cópia local, no mesmo diretório de destino do FOM gerado do XML Schema, o qual é obtido através da estrutura lida. Posteriormente, iniciou-se a validação. Uma biblioteca em Java chamada Validator foi usada para auxiliar a comparação. Na Figura 5.1 é apresentado o trecho de código usado para a manutenção da integridade do FOM gerado. Caso o arquivo não seja validado, a validação é interrompida, um aviso com a especificação do erro e sua localização no arquivo (número da linha e coluna) é dado, mas o novo arquivo é salvo da mesma maneira. O programa continua com as modificações e o usuário pode editar os componentes novamente para que possa ser efetuado a produção de um novo arquivo. Os erros mais comuns são valores de tags vazios, caracteres especiais como espaços em branco, dois pontos, ponto e vírgula e sinais matemáticos, os quais não são permitidos de acordo com o XML Schema.

Listing 5.1: Código de validação

```
1  try {
2      File xml = new File(path + "\\generatedFOM.xml");
3
4      /* geracao da copia do XML Schema */
5
6      File xsd = new File(path + "\\schema.xsd");
7
8      Source schemaFile = new StreamSource(xsd);
9      Source xmlFile = new StreamSource(xml);
10     SchemaFactory schemaFactory =
           SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI
11     Schema schema = schemaFactory.newSchema(schemaFile);
12     Validator validator = schema.newValidator();
13     validator.validate(xmlFile);
14 } catch (SAXException | IOException ex) {
15     input = JOptionPane.showOptionDialog(null, "The file
           is not valid. Please, review it.", "Error",
           JOptionPane.PLAIN_MESSAGE,
           JOptionPane.ERROR_MESSAGE, null, null, null);
16 }
```

Uma validação inicial também é realizada, quando um componente é lido de sua estrutura e mostrado na interface. Nessa etapa, os nomes das tags não são analisados, somente seus valores. O motivo da exclusão dos nomes das tags é a possibilidade de o desenvolvedor gerar um novo FOM, de acordo com a norma e ao mesmo tempo corrigir o que havia de errado no anterior, com o nome de tags corretas.

Além disso, quatro testes foram realizados a fim de garantir uma melhor qualidade ao trabalho. O primeiro teste foi o teste de leitura, realizado durante a implementação da mesma. Esse teste consiste em verificar se todos os componentes presentes no arquivo estavam sendo lidos e mostrados na interface. A verificação foi realizada através da contagem de nodos de cada componente. O segundo teste foi o de entradas, realizado também durante a fase de leitura. Cada valor de tag que possuía alguma restrição, como por exemplo, um tipo de dado estar presente no arquivo ou existir conforme a norma, foi averiguado. Ademais, foi incorporado esse teste ao software, avisando o usuário a inconsistência que o arquivo possuía, mas deixando ele alterá-la e continuar a usar normalmente o LE-FOM. O terceiro teste foi o de edição, aplicado na fase de adaptação da interface. Verificou-se se todas tabelas que poderiam ser editáveis estavam de fato habilitadas e as que possuíam elementos padrões e com quantidades definidas estavam editáveis, porém dentro da norma. Por fim o teste de validação de acordo com o XML Schema, aplicado quando o software estava finalizado. Alterou-se o arquivo de forma que gerasse inconsistências a fim de verificar se o algoritmo de validação era efetivo e correto.

Cada teste foi executado com FOMs diferentes e modificados para simular possíveis erros e verificar se o LE-FOM estava respondendo a eles de maneira correta. Os testes geraram bugs, os quais foram corrigidos através da Depuração Automática e Eliminação da Causa. Posterior às correções, foram feitos novos testes. Procurou-se não deixar passarem defeitos para as etapas seguintes. Após a finalização do software, todos os testes foram feitos outra vez, com o propósito de nenhuma etapa ter afetado, ou seja, gerado erros nas outras.

6 CONCLUSÃO

Neste trabalho foi descrito um software para leitura e edição de arquivos FOM. O software surgiu através da dificuldade de interação com o arquivo FOM, uma vez que ele possui uma estrutura hierárquica com muitos elementos. O objetivo principal foi auxiliar desenvolvedores na área de simulação os quais usam o padrão HLA, a personalizar um FOM de acordo com seus propósitos em seus projetos. Durante a fase de estudo compreendeu-se como o padrão HLA funcionava, juntamente com as especificações dos componentes do Federation Object Model. Além disso, ganhou-se conhecimento sobre a linguagem de marcação XML, através do estudo realizado para a manipulação do arquivo FOM escrito nessa linguagem. Com isso, foi possível implementar um parser em XML, com o auxílio da biblioteca SAX, onde estruturas em forma de árvore são preenchidas conforme cada componente presente no arquivo. O conteúdo das estruturas foi apresentado graficamente, em tabelas, as quais foram adaptadas para a edição dos itens presentes. Além disso, botões com o intuito de adicionar novos itens foram inclusos na interface. Colunas com itens predefinidos pela norma ou por outros componentes presentes no arquivo foram formatadas em ComboBoxes, garantindo que o usuário não inserisse um conteúdo incoerente à norma. O arquivo então, pode ser salvo e validado de acordo com seu XML Schema, a fim de mantê-lo dentro da norma HLA. Em caso de erros de validação ou de leitura, quando o arquivo de entrada não está de acordo com o padrão, o programa avisa o usuário e se mantém em um estado onde é possível que ele corrija as inconsistências.

Assim, com esse trabalho podemos concluir que o software desenvolvido atende aos requisitos definidos no capítulo 3. Ademais, buscou-se alcançar todos os objetivos específicos, e consequentemente o geral, construindo gradativamente o trabalho. Nenhuma dificuldade agravante foi encontrada durante o desenvolvimento. Espera-se que por ser um software livre, desenvolvedores da área de simulação usufruam do LE-FOM e que ele facilite e torne eficiente a customização do FOM.

Como trabalho futuro, o LE-FOM poderia permitir que FOMs divididos em módulos, contidos na versão HLA abordada, fossem lidos e editados. Uma maior interatividade com o usuário, mostrando os erros específicos ao invés de uma mensagem de erro geral, tornando o software mais industrial também pode ser considerada como uma melhoria futura. Além disso, sugere-se como um próximo trabalho a geração de código, ou seja, a geração de classes a partir do FOM, o que facilita a comunicação com o RTI o qual se comunica através de vetores de bytes. Uma outra continuação do trabalho possível seria a adaptação do código para que atendesse a diferentes versões do Federation Object Model, sendo flexível a todas versões HLA.

REFERÊNCIAS BIBLIOGRÁFICAS

BRAGANHOLO, V.; MORO, M. Desmistificando xml: da pesquisa à prática industrial. In: SBC (Ed.). **Atualizações em Informática 2009**. [S.l.]: A. de Carvalho e T. Kowaltowski, 2009. cap. 5, p. 231–279.

CAMARA, G.; FONSECA, F. Information policies and open source software in developing countries. **Journal of the American Society for Information Science and Technology**, Wiley Online Library, v. 58, n. 1, p. 121–132, 2007.

CARDOSO, L.; JOAQUIM, C. Simulação computacional: Análise comparativa dos software arena e promodel. **Tekhne e Logos**, v. 7, n. 1, p. 14–30, 2016.

CHWIF, L.; MEDINA, A. **Modelagem e simulação de eventos discretos: teoria e aplicações**. São Paulo: [s.n.], 2010. 320 p.

FUJIMOTO, R. M. **Parallel and distributed simulation systems**. [S.l.]: Wiley New York, 2000. v. 300.

GAVIRA, M. de O. **Simulação computacional como uma ferramenta de aquisição de conhecimento**. 2003. Tese (Doutorado) — Universidade de São Paulo, 2003.

IEEE. IEEE, **IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)**: Object model template (omt) specification. 2010.

____. IEEE, **IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)**: Federate interface specification. 2010.

JUNQUEIRA, F. **Modelagem e simulação distribuída de sistemas produtivos**. 2006. Tese (Doutorado) — Universidade de São Paulo, 2006.

KALE, R. A. et al. Xml dom parsing using tree-branch symbiosis algorithm. **International Journal of Scientific Engineering Research**, v. 7, 2016.

KRÁLÍCEK, B. T. **Building an air traffic simulation platform using open-source components**. 2011. Tese (Doutorado) — Masaryk University, 2011.

MÖLLER, B. et al. Extended fom module merging capabilities. 2013.

____. Rpr fom 2.0: A federation object model for defense simulations. 2014.

Okan Topçu. **SimGe**. 2016. Acesso em 10 set. 2016. Disponível em: <https://sites.google.com/site/okantopcu/simge>.

PEGDEN, C. D.; SADOWSKI, R. P.; SHANNON, R. E. **Introduction to simulation using SIMAN**. [S.l.]: McGraw-Hill, Inc., 1995.

Pitch Technologies. **Pitch Technologies**. 2016. Acesso em 22 set. 2016. Disponível em: <http://www.pitchtechnologies.com/>.

SAX. 2016. Acesso em 19 out. 2016. Disponível em: <http://www.saxproject.org/event.html>.

SHANNON, R. E. R. E. **Systems simulation; the art and science**. [S.l.], 1975.

STRACK, J. **GPSS: modelagem e simulação de sistemas**. [S.l.]: LTC, 1984.

VENKATESWARAN, J.; JAFFERALI, M. Y. K.; SON, Y.-J. Distributed simulation: an enabling technology for the evaluation of virtual enterprises. In: IEEE COMPUTER SOCIETY. **Proceedings of the 33nd conference on Winter simulation**. [S.l.], 2001. p. 856–862.

WILDE, E.; GLUSHKO, R. J. Xml fever. **Queue**, ACM, v. 6, n. 6, p. 46–53, 2008.