Universidade Federal do Pampa
Gabriel Bronzatti Moro

Uma Ferramenta de Apoio à Especificação de Requisitos para Sistemas Autoadaptativos

Alegrete
2015

Gabriel Bronzatti Moro

Uma Ferramenta de Apoio à Especificação de Requisitos para Sistemas Autoadaptativos

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: João Pablo Silva da Silva

Alegrete

2015

Gabriel Bronzatti Moro

Uma Ferramenta de Apoio à Especificação de Requisitos para Sistemas Autoadaptativos

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 8 de julho de 2015

Banca examinadora:

João Pablo Silva da Silva
Orientador

Jean Felipe Patikowski Cheiran
UNIPAMPA

Sam da Silva Devincenzi
UNIPAMPA

Este trabalho é dedicado primeiramente a Deus, que me fortaleceu em todos os momentos de maiores dificuldades, devo a Ele todo o conhecimento adquirido para sua realização. Também o dedico a minha família, que sempre torceu pelo meu sucesso, auxiliando-me em todas as minhas necessidades, minha mãe, pai, tias, tios, madrinhas e padrinhos. Além disso, quero trazer presente a memória das minhas avós, pois, sem elas, não seria possível chegar a essa conquista. Da mesma forma, agradeço a minha namorada pelo apoio, paciência e compreensão. Por final, agradeço ao meu orientador, que não cessou em auxiliar-me durante toda a pesquisa, para ele o meu respeito e admiração.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12:2)

Resumo

O desenvolvimento de Sistemas Autoadaptativos (SAs) requer soluções diferentes da Engenharia de Requisitos (ER) tradicional, pois esses sistemas tratam de aspectos de incerteza nos requisitos e a contínua troca de contexto de usuários. Whittle et al. (2010) apresentam a Linguagem Relax para apoiar essa problemática, essa linguagem permite representar os aspectos ambientais, temporais, modais, ordinais e de incerteza nos requisitos. A falta de suporte para a especificação de requisitos em Linguagem Relax é a motivação do presente trabalho. O objetivo principal dessa pesquisa é o desenvolvimento de uma ferramenta *plugin* para apoiar a especificação de requisitos utilizando a Linguagem Relax definida por Whittle et al. (2010). A metodologia utilizada nesse trabalho, consiste em um mapeamento sistemático da literatura para explorar os trabalhos relacionados, estudo aprofundado sobre Engenharia de Requisitos, Sistemas Autoadaptativos e Linguagem Relax. O Eclipse para o desenvolvimento de *plugins* e Processador de Linguagem foram as tecnologias exploradas como base tecnológica nesse trabalho. Ao decorrer do trabalho foi possível compreender a relevância da linguagem Relax para a especificação de requisitos de Sistemas Autoadaptativos, visto que vários aspectos de um requisito podem ser mapeados pela utilização da linguagem. O resultado final obtido foi a ferramenta *plugin* RelaxEditor, através dela é possível especificar requisitos utilizando a Linguagem Relax. A partir de um experimento realizado com usuários foi possível comprovar que a ferramenta é adequada ao seu propósito. A ferramenta desenvolvida atende ao objetivo principal do trabalho e oferece suporte ao Engenheiro de Software na especificação de requisitos.

Palavras-chave: Sistemas Autoadaptativos. Engenharia de Requisitos. Ferramenta de Suporte. Linguagem Relax.

Abstract

The development of Self-Adaptive Systems (SAs) require different solutions of traditional Requirements Engineering (ER), because those systems treat the uncertainty aspects in the requirements and the prolonged exchange of the context from the users. Whittle et al. (2010) show the Relax Language as support for this problematic, this language allows to represent the environmental, temporal, modal, ordinal and uncertainty aspects in the requirements. The lack of a support for Requirements Specification in Relax Language, is the motivation of these present work. The main research goal is the development of a plugin tool that offers support for specify requirements using the Relax Language defined by Whittle et al. (2010). The Methodology used in this work consists in a systematic mapping of literature to explore the related works, depth study on Requirements Engineering, Self-Adaptive Systems and Relax Language. Also, it was investigated the technologies: Eclipse for development of the plugins and Language Processor, which were used to development the tool. The course of the work was possible understand the relevance of Relax Language for the specification of the requirements for Self-Adaptive Systems, seen that several aspects of the a requirement may be mapped using the language. The result obtained was a plugin tool RelaxEditor, through it you can specify requirements using the Relax Language, from the experiment done with users was possible to prove that the tool is suitable for its purpose. The tool developed meets the main objective of the work and supports the Software Engineer to specify requirements.

Key-words: Self Adaptive Systems. Requirements Engineering. Support-tool. Relax Language.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*

CRUD *Create-Read-Update-Delete*

DAO *Data Acess Object*

ER Engenharia de Requisitos

FBTL *Fuzzy Branching Temporal Logic*

IDE *Integrated Development Environment*

JAD *Joint Application Design*

JAR Java *Archive*

JVM Java *Virtual Machine*

MVC *Model-View-Controller*

SA Sistema Autoadaptativo

SBS Sistema Baseado em Serviços

UML *Unified Modeling Language*

XML *Extensible Markup Language*

XP *Extreme Programming*

Sumário

1 Introdução

Sommerville (2011) define a Engenharia de Requisitos (ER) como o processo de investigação, análise, especificação e validação de serviços ou restrições que um sistema deve oferecer. Pressman (2011) complementa essa definição destacando que a utilização da ER permite o entendimento do desenvolvimento de software a ser realizado, ou seja, o quanto deve ser projetado, codificado e validado, quais as regras de negócio que o projeto deve atender, a definição das prioridades de trabalho e quais são os requisitos mais relevantes e necessários. O objetivo desse processo é entender o problema do cliente e quais são suas necessidades, ou seja, seu foco está no problema e não na solução (PFLEEGER; ATLEE, 2010).

A investigação e análise de requisitos é a fase dedicada para o conhecimento das necessidades do cliente e o seu ambiente. Nessa fase, utilizam-se técnicas de apoio que são aplicadas de acordo com o público alvo, como etnografia, questionário, entrevista, *workshop*, entre outras. Na fase de especificação, as informações adquiridas nas etapas anteriores, são documentadas para auxiliar a comunicação entre a equipe de desenvolvimento. Como exemplo pode ser classificado as regras de negócio, requisitos funcionais, requisitos não-funcionais, requisitos de usuário, entre outros. Geralmente o processo de ER é concluído com a fase de validação de requisitos que compreende a verificação de todo o artefato resultante com o cliente para avaliar se a especificação de requisitos define o comportamento esperado do software (SOMMERVILLE, 2011).

Sistemas Autoadaptativos (SAs) são softwares que possuem a capacidade de alterar autonomamente o seu comportamento em tempo de execução como resposta às mudanças do seu ambiente (WHITTLE et al., 2010). Esse tipo de sistema é ciente de contexto e atua detectando os eventos necessários e reagindo com conformidade a eles. Além disso, o comportamento autoadaptativo permite que o sistema realize a partir do contexto as ações presentes em seu conjunto de metas (CHENG et al., 2009).

Qureshi e Perini (2010b) apresentam um exemplo de Sistemas Baseado em Serviços (SBSs) para reserva de passagens aéreas. Esse sistema utiliza serviços da internet oferecidos por empresas de transporte aéreo para atender às necessidades do usuário. O sistema identifica o serviço mais adequado para o usuário, tendo por base suas preferências, como condição financeira, data e horário disponível, duração da viagem e assim por diante. Os SBSs tratam a incerteza na utilização desses serviços, pois os mesmos podem sofrer modificações, manutenção ou até mesmo estarem indisponíveis com o tempo.

Bencomo et al. (2012) demonstram um exemplo de evolução de software que inclui a capacidade de autoexplicação em um sistema chamado GridStix. Esse sistema utiliza

uma rede de sensores sem fio (RSSF) para identificar e prever alagamentos. Existem versões implantadas no Rio Ribble (Inglaterra) e Rio Dee (País de Gales). No GridStix a autoexplicação está em sua capacidade de relatar ao usuário os motivos pelos quais foi tomada determinada decisão. Vale ressaltar que, a autoexplicação é uma das características de um comportamento autoadaptativo (BENCOMO et al., 2012).

O desenvolvimento de SAs requer soluções diferentes da ER tradicional, pois esses sistemas tratam de aspectos de incerteza nos requisitos e a contínua troca de contexto. Esses fatores, muitas vezes, não podem ser previstos na fase de projeto (QURESHI; PERINI, 2010a). Whittle et al. (2010) propõe uma solução para auxiliar o processo de ER para SAs. Uma linguagem estruturada para a especificação de requisitos que envolvem aspectos de incerteza. Diferentemente da especificação de requisitos clássica, a qual determina os possíveis requisitos, essa linguagem permite a representação das fontes geradoras de incerteza nos requisitos.

Whittle et al. (2010) destacam a falta de suporte para a criação e especificação real de requisitos em SA. Percebe-se que, os tópicos mais pesquisados estão relacionados com as técnicas de apoio a análise de requisitos (SAWYER et al., 2010; QURESHI; PERINI, 2010b; QURESHI; PERINI, 2009), o monitoramento de contexto (QURESHI; PERINI, 2010a; BENCOMO et al., 2010) e as abordagens arquiteturais e de projeto (BENCOMO et al., 2012; WELSH; SAWYER, 2010). Nesse sentido, identifica-se a carência de ferramentas para apoiarem o processo de ER para SAs.

1.1 Objetivos

O objetivo geral é desenvolver uma ferramenta para a especificação de requisitos em linguagem Relax que dê suporte para a ER em SAs. Tal objetivo pode ser divido nas seguintes metas específicas:

- explorar a linguagem Relax em aspectos semânticos e sintáticos;
- tornar o formalismo da linguagem transparente;
- utilizar o Eclipse como plataforma de desenvolvimento.

1.2 Metodologia

A metodologia utilizada para realizar o presente trabalho pode ser listada nos seguintes tópicos:

- realizar um mapeamento sistemático da literatura para explorar os trabalhos relacionados como ferramentas de apoio a ER e de apoio a ER para SAs;

- pesquisar sobre sobre ER, SAs e linguagem Relax;
- pesquisar sobre as tecnologias: compiladores e plataforma eclipse para o desenvolvimento de *plugins*;
- percorrer atividades de desenvolvimento de software, organizadas pelas etapas: análise, projeto, implementação e testes;
- realizar o experimento para validar a ferramenta.

1.3 Organização do Documento

O presente trabalho está estrutrurado da seguinte maneira:

- Capítulo 2: Fundamentação Teórica, onde é abordado sobre a ER, SAs e linguagem Relax;
- Capítulo 3: Base Tecnológica, nele é apresentado conceitos básicos sobre as tecnologias de processador de linguagem (compilador) e plataforma Eclipse para o desenvolvimento de *plugins*;
- Capítulo 4: Trabalhos Relacionados, correspondentes a questão base de pesquisa utilizada no trabalho;
- Capítulo 5: Desenvolvimento da Ferramenta, o qual apresenta os primeiros passos no desenvolvimento da ferramenta proposta. Nessa é apresentado a fase de concepção, primeira iteração da fase de elaboração e os artefatos desenvolvidos;
- Capítulo 6: Experimento em Ambiente Controlado, nesse capítulo é descrito o experimento realizado com usuários para avaliar a aceitação dos mesmos na utilização da ferramenta desenvolvida;
- Capítulo 7: Conclusões, nela é descrito os desafios enfrentados durante a pesquisa, o aprendizado e o resultado final obtido no trabalho. Também, esse capítulo apresenta os próximos passos da pesquisa (trabalhos futuros).

2 Fundamentação Teórica

Neste capítulo são apresentados os fundamentos teóricos necessários para a compreensão do trabalho. Na seção 2.1 é apresentada a definição de ER, suas características, abordagens de diferentes autores e suas principais etapas. Na seção 2.2 é abordado o conceito de SAs, suas propriedades, tipos de sistemas que possuem essas propriedades e as necessidades para o desenvolvimento desses sistemas. O capítulo é finalizado com a seção 2.3, a qual apresenta a linguagem Relax, descrevendo o seu vocabulário e o processo de aplicação.

2.1 Engenharia de Requisitos

A ER é um conjunto de abordagens focadas para o entendimento dos requisitos pela equipe de desenvolvimento podendo ser entendida como uma prática de Engenharia de Software, que pode iniciar antes das demais fases do processo de desenvolvimento, se estendendo até a fase de projeto. A ER também pode ser aplicada a diferentes processos de desenvolvimento de software que muitas vezes utilizam um ciclo de vida definido, onde uma das fases é destinada para a investigação dos requisitos (PRESSMAN, 2011). Além disso, através da ER é possível construir e manter um artefato de documentação de requisitos, chamado de SRS (*Software Requirements Specification*) que deve apresentar as necessidades do cliente em requisitos e que deve ser seguido pela equipe de desenvolvimento, pois tudo o que foi especificado deve ser implementado no sistema a ser entregue (SOMMERVILLE, 2011).

É possível aplicar a ER de várias maneiras, a abordagem a ser escolhida pela equipe de análise deve contemplar questões como o problema do cliente, quantidade de clientes para dado problema, processo de desenvolvimento utilizado pela equipe, quantidade de membros da equipe, experiência da equipe de análise na abordagem de ER escolhida e assim por diante.

2.1.1 Processo de Engenharia de Requisitos

Sommerville (2011) destaca o processo de ER em diferentes fases como estudo de viabilidade, elicitação, especificação e validação de requisitos. O estudo de viabilidade é utilizado para verificar se o sistema a ser desenvolvido pode auxiliar a problemática do cliente, se o prazo do cliente é adequado para o desenvolvimento do sistema, entre outros fatores que devem ser esclarecidos antes do início do processo de ER e o artefato gerado dessa fase é um relatório de viabilidade. Após isso, realiza-se a elicitação de requisitos, fase

na qual a equipe de análise identifica quais os serviços que o sistema deve oferecer e quais restrições deverão ser respeitadas. A partir disso, gera-se como artefato um modelo de sistema. A especificação de requisitos é a próxima fase, nela são construídos os requisitos de usuário e os de sistema, de acordo com as informações levantadas nas fases anteriores. Também, o processo possui uma fase de validação de requisitos destinada a revisão dos requisitos com as fontes investigadas podendo ser utilizadas várias técnicas de validação como prototipagem, revisões e assim por diante.

Diferentemente da abordagem apresentada por Sommerville (2011), Pfleeger e Atlee (2004) acrescentam um grupo de fases chamado identificação e análise de requisitos que contém as fases de análise, descrição do problema e prototipação com testes. A análise do problema é destinada para identificar as carências do cliente e entender o seu problema. Após isso, na fase de descrição do problema, o problema é construído de acordo com a fase anterior. Nessa fase devem ser avaliadas também as técnicas utilizadas para a investigação de requisitos identificando se elas foram aplicadas de maneira correta e verificando se as informações investigadas possuem coesão. As fases de prototipação e testes são destinadas à construção de protótipos para serem verificados pelo cliente.

Além disso, Pressman (2011) destaca que as fases de concepção, elaboração, negociação e gestão de requisitos são necessárias em um processo de ER tradicional. A fase de concepção deve ser utilizada para ao conhecimento do problema do cliente podendo ser obtido pela equipe de análise através de uma conversa informal para identificar de maneira geral as necessidades do cliente, quais meios são utilizados para resolver determinadas necessidades e assim por diante. Na fase de elaboração deve ser aplicado o refinamento dos requisitos adquiridos nas fases anteriores podendo também ser criado cenários para representar o usuário e seu contexto.

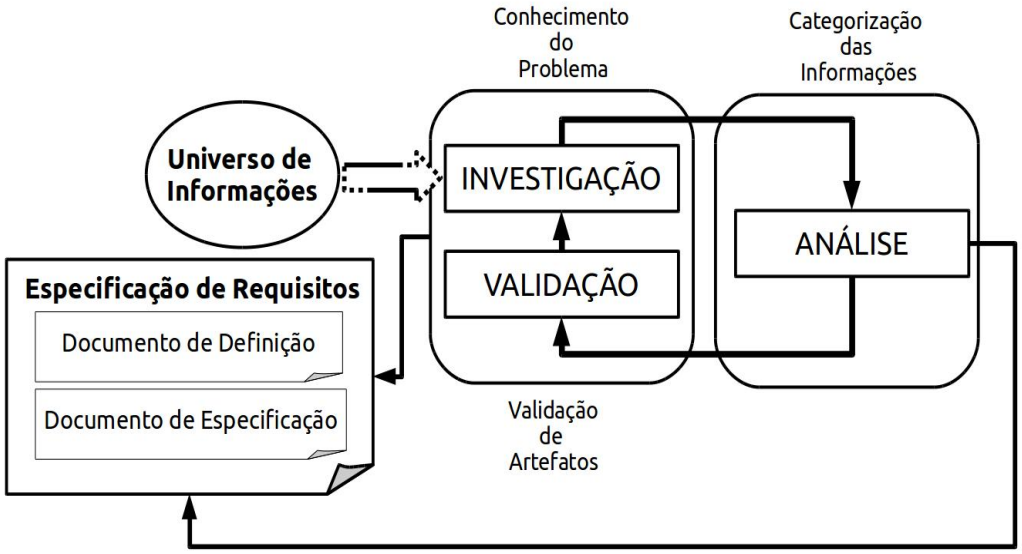
Na fase de negociação deve ser promovida a resolução de conflitos entre clientes e a equipe de análise. Nessa fase são abordadas diferentes temáticas referentes a custos, riscos e prioridade de requisitos. A fase de gestão é destinada ao monitoramento das mudanças de requisitos, geralmente os requisitos são mutáveis e podem sofrer alterações de acordo com tempo. Essas alterações muitas vezes podem ser derivadas das leis organizacionais, características de contexto, necessidades dos usuários, entre outros fatores.

De acordo com as abordagens de ER apresentadas, pode-se entender que cada uma possui vantagens e desvantagens. Uma boa indicação seria a utilização de uma abordagem mista que integra os pontos fortes de ambas abordagens. Como exemplo na abordagem do Pressman (2011), quando comparada com as demais é a mais completa, pois possui mais fases e divide melhor as preocupações de gerenciamento das mudanças de requisitos, mas pode ser difícil de ser utilizada em metodologias ágeis, pois sua aplicação pode exigir da equipe de análise mais tempo de planejamento e reuniões para o gerenciamento do processo de requisitos.

Além disso, na abordagem do Pfleeger e Atlee (2004) é utilizado uma proposta mais simples que pode ser aplicada tanto para modelos de desenvolvimento sequenciais como ágeis. O que o diferencia das demais é que possui uma fase destinada para a prototipação e testes, nisso é possível visualizar que a abordagem motiva a utilização de técnicas práticas de investigação e validação de requisitos. Já a proposta de Sommerville (2011) possui uma abordagem orientada a artefatos, ou seja, a cada fase incentiva-se o desenvolvimento de artefatos para servirem como insumos para o artefato principal do processo que é a especificação de requisitos.

Neste trabalho foi definido uma abordagem de ER (apresentada na Figura 1) baseada nas diferentes perspectivas apresentadas por Sommerville (2011), Pfleeger e Atlee (2004) e Pressman (2011).

Figura 1 – Visão de processo de Engenharia de Requisitos utilizada nesse trabalho.



2.1.1.1 Investigação

A fase de investigação tem por objetivo a descoberta de quais funcionalidades (requisitos funcionais) o sistema deve oferecer, a partir do conhecimento das necessidades do cliente, ou seja, o entendimento do seu problema. Também, nessa fase as restrições (requisitos não funcionais) sob as funcionalidades do software deverão ser encontradas podendo tratar aspectos como desempenho de processamento, usabilidade de interface, segurança de dados e assim por diante (WAZLAWICK, 2004).

Alguns problemas podem ser encontrados na investigação de requisitos, a maioria deles parte do cliente que muitas vezes tem dificuldade em apresentar as suas necessidades omitindo informações importantes, descrevendo requisitos com ambiguidade e

conflito (PRESSMAN, 2011). Pressman (2011) destaca três categorias de problemas que podem ser encontrados referentes ao escopo, entendimento e volatilidade. Os problemas de escopo são encontrados quando o cliente não apresenta as informações de maneira geral especificando detalhes desnecessários que podem gerar dúvida na equipe de analistas. Quanto aos problemas de entendimento, eles surgem quando o cliente não compreende a extensão do seu problema. Os problemas de volatilidade descrevem os requisitos como entidades dinâmicas que podem sofrer alterações com o decorrer do tempo.

A equipe de desenvolvimento nesse contexto é responsável por obter as informações do cliente. Os requisitos são descobertos e não criados pelos analistas. As informações do cliente devem ser a principal fonte de requisitos. Nisso, a equipe de desenvolvimento deve utilizar técnicas para obter as informações do cliente.

As técnicas de investigação de requisitos devem ser utilizadas pela equipe de desenvolvimento, pois organizam o processo de investigação e são aplicadas de acordo com o público-alvo.

Para auxiliar a fase de investigação pode ser utilizado técnicas como levantamento orientado a ponto de vista, entrevista, etnografia, questionário, *BrainStorm*, *Workshop*, prototipagem e *Joint Application Design* (JAD).

Levantamento Orientado a Ponto de Vista

O levantamento orientado a ponto de vista é muito utilizado nos momentos iniciais da fase de investigação de requisitos, pois classifica os *stakeholders* em tipos de pontos de vista com o intuito de organizar os requisitos que vão surgindo de acordo com esses tipos. Pode-se classificar tipos genéricos de pontos de vista, os que descrevem os usuários e os sistemas que atuam diretamente com o sistema a ser desenvolvido, usuários ou sistemas que influenciam nas funcionalidades e os aspectos do contexto da aplicação (SOMMERVILLE, 2011).

Entrevista

A entrevista é uma técnica para auxiliar a equipe de analistas sobre a atividade e as necessidades do cliente e pode ser utilizada com um roteiro de perguntas definidas ou como uma conversa informal que aborda diversos temas relacionados com as atividades do cliente (SOMMERVILLE, 2011).

Etnografia

A entnografia é uma técnica que visa o endendimento do contexto do cliente, pois alguns requisitos surgem do âmbito organizacional em que o usuário do sistema está inserido. Essa técnica consiste na observação diária das atividades do usuário e no registro de possíveis requisitos (SOMMERVILLE, 2011).

Questionário

A técnica de questionário consiste em um conjunto definido de perguntas com a finalidade de obter requisitos através das respostas dos usuários. Também, o questionário pode ser utilizado para a validação de protótipos, ou seja, sua aplicação não é restrita apenas para a investigação de requisitos.

BrainStorm

O *BrainStorm* pode ser utilizado em reuniões para a geração de ideias e pode ser divido em três passos: identificação dos membros participantes, explicação das regras da técnica e produção de ideia. No primeiro passo são selecionadas pessoas que farão parte da técnica, teram preferência os usuários que possuem um bom conhecimento sobre as necessidades que o produto de software deve atender. Após isso, o líder apresenta quais serão as regras da técnica, pois a mesma é desenvolvida com todos os participantes. O *BrainStorm* é finalizado com a produção de ideias, onde o líder apresenta tópicos e para cada tópico os membros por vez apresentam suas ideias de acordo com o número de rodadas (MORAES, 2009). Na finalização da técnica, a equipe de desenvolvimento verifica o conjunto de ideias gerado e o revisa para classificar as ideias potenciais que podem auxiliar na definição de requisitos.

Workshop

O *Workshop* é uma técnica de levantamento de requisitos realizada em grupo com os *stakeholders* selecionados de acordo com seu conhecimento sobre o problema, a equipe de análise e um coordenador para a reunião que deve possuir um perfil neutro e deve atuar como mediador. A partir do *Workshop* é construído pela equipe de desenvolvimento um documento que contém os aspectos principais que foram levantados pelos *stakeholders* durante a reunião (MORAES, 2009).

Prototipagem

A prototipagem tem por objetivo auxiliar a equipe de desenvolvimento na investigação de requisitos principalmente dos não funcionais. Através do protótipo, o usuário pode visualizar como as suas necessidades estão sendo tratadas pela equipe de desenvolvimento podendo ser encontrado novos requisitos ou ser alterado requisitos já levantados (MORAES, 2009).

Joint Application Design

Através do JAD é possível obter um documento de requisitos que é construído no decorrer das sessões, essa técnica pode ser dividida em duas etapas, uma etapa destinada ao planejamento dos requisitos e outra para o projeto de software. Além disso, a técnica utiliza vários papéis em suas sessões tendo um líder para coordenar a reunião, o engenheiro de requisitos que documenta os resultados durante a aplicação da técnica, o executor que é responsável pelo controle das ações referentes ao desenvolvimento do software, os usuários selecionados que vão utilizar o software, profissionais que auxiliam os usuários quanto as

dúvidas sobre as operações que o software deve realizar e um especialista que deve possuir conhecimento sobre os assuntos abordados (tecnologias, processos de desenvolvimento, entre outros) (MORAES, 2009).

2.1.1.2 Análise

A fase de Análise consiste na categorização dos requisitos que foram levantados na fase de investigação identificando grupos de requisitos e categorizando-os de acordo com suas propriedades.

Classificação de Requisitos

Os requisitos de um sistema são os serviços que o software deve oferecer com o objetivo de apoiar as necessidades do cliente (SOMMERVILLE, 2011). Também, o termo requisito pode ser aplicado para definir as restrições de usuário, organizacionais, desempenho, hardware, entre outras. Essas restrições devem ser respeitadas pelos serviços oferecidos do sistema.

Os requisitos funcionais descrevem o comportamento do sistema, ou seja, como ele deve reagir a determinados eventos disparados muitas vezes pelo usuário. Quanto aos não funcionais, eles representam as limitações que podem influenciar nos requisitos funcionais de maneira indireta (SOMMERVILLE, 2011). Por exemplo o requisito funcional do sistema é a ordenação de um vetor de elementos, soma-se isso o requisito não funcionais de desempenho definindo que todas as ações de ordenação devem ser efetuadas em um limite de tempo aceitável, por exemplo dez segundos.

Os requisitos permanentes ou transitórios agrupam os requisitos não funcionais descrevendo requisitos que podem mudar (transitórios) e os requisitos que não sofreram mudanças (permanentes). Essa classificação é definida pela equipe de análise que deve realizar sua seleção de permanentes ou transitórios de acordo com o seu ponto de vista (WAZLAWICK, 2004). Os requisitos permanentes podem surgir muitas vezes dos conceitos organizacionais como padrões de relatórios, hierarquia de usuários, recursos de interface gráfica e assim por diante. Quanto aos requisitos transitórios, eles podem ser derivados das necessidades dos usuários que muitas vezes podem sofrer alterações com o decorrer do tempo.

Os requisitos funcionais podem ser classificados como evidentes ou ocultos. São evidentes quando necessitam de informações do usuário para realizar sua tarefa como cadastro de informações, consultas ao banco de dados e assim por diante. Quanto aos requisitos ocultos, eles descrevem as funcionalidades internas do sistema como processamento de dados, comunicação com o banco de dados, cálculos, ações que a interface do sistema não apresenta ao usuário (WAZLAWICK, 2004). Nisso o usuário não visualiza tais funcionalidades.

Além disso, podemos classificar os requisitos funcionais e não funcionais em obrigatórios e desejáveis. O conjunto de requisitos obrigatório descreve os principais requisitos do sistema que devem ser implementados pelos desenvolvedores. Também, temos o conjunto de requisitos desejáveis que descrevem os requisitos de segundo plano, os quais serão implementados após os obrigatórios, ou seja, seu desenvolvimento é de prioridade inferior (WAZLAWICK, 2004).

Modelagem Conceitual

O modelo conceitual é um artefato que pode ser construído para representar as informações que o sistema deve administrar, esse artefato deve ser desenvolvido a partir do domínio do problema (WAZLAWICK, 2004). Também, a modelagem conceitual não descreve os aspectos internos do sistema como por exemplo tipo de dados ou a arquitetura utilizada pelo software e sim as informações referentes ao contexto do usuário demonstrando como elas estão organizadas sob o ponto de vista da equipe de análise.

Na fase de análise a modelagem conceitual é construída a partir das informações levantadas na fase de investigação, muitas vezes essas informações podem estar replicadas ou desordenadas. A técnica de modelagem conceitual auxiliará também no processo de organização dessas informações, pois os modelos podem ser validados juntamente com o cliente para refinar o conjunto de requisitos levantados.

Casos de Uso de Alto Nível

Os casos de uso de alto nível são criados a partir dos artefatos da fase de investigação. Eles são diagramas de caso de uso da *Unified Modeling Language* (UML) e servem para descrever as principais funcionalidades do sistema. Através dessa técnica é possível visualizar quais usuários (atores) são responsáveis ou estão relacionados a determinado caso de uso (WAZLAWICK, 2004). Também é possível visualizar que os casos de uso são independentes entre si, ou seja, as funcionalidades descritas são isoladas com os seus objetivos tratando informações diferentes uma das outras.

2.1.1.3 Especificação

A fase de especificação tem por objetivo registrar os requisitos para que possam ser acessados pela equipe de desenvolvimento e pelo próprio cliente a qualquer etapa do processo de desenvolvimento (PFLEEGER; ATLEE, 2004). Esse registro de requisitos também pode conter os artefatos construídos nas fases anteriores do processo de ER incluindo artefatos de investigação como protótipos, registros de entrevistas, respostas de questionários, entre outros e os artefatos de análise como modelos conceituais e casos de uso de alto nível. Vale salientar que, a ferramenta desenvolvida nesse trabalho tem por objetivo apoiar essa etapa do processo de ER.

O documento de requisitos deve descrever os serviços que o sistema deve ofere-

cer e também quais as restrições impostas por esses serviços. Os requisitos podem ser apresentados em tópicos e divididos em seções de acordo com a classificação da fase de análise. Também, o documento de requisitos pode ser escrito em texto e utilizar diagramas gráficos.

O processo de ER pode ser realizado várias vezes, a cada interação do processo o documento de requisitos não vai ser criado novamente e sim mantido, ou seja, as mudanças que ocorrerão nos requisitos durante o desenvolvimento deverão ser registradas ou atualizadas no documento de requisitos para manter a integridade dos artefatos do sistema.

Pfleeger e Atlee (2004) descrevem dois documentos de requisitos, um destinado a definição de requisitos e outro para a especificação de requisitos.

Documento de Definição de Requisitos

O documento de definição de requisitos deve ser escrito para o cliente e deve conter o conjunto completo de requisitos que serão desenvolvidos utilizando uma linguagem simples para facilitar o entendimento do cliente (PFLEEGER; ATLEE, 2004). Esse documento deve descrever qual o objetivo principal do desenvolvimento do sistema, as contribuições que serão alcançadas e quais problemas serão resolvidos. A estrutura do documento de definição pode ser organizada iniciando com uma explicação geral do problema do usuário como ele realiza suas tarefas para suprir tais necessidades. Após isso, pode ser descrito o conjunto de funções e restrições do sistema. O documento pode ser finalizado tratando sob o ambiente em que o sistema deve ser implantado (PFLEEGER; ATLEE, 2004).

Documento de Especificação de Requisitos

Diferentemente do documento de definição de requisitos que descreve os requisitos em uma linguagem acessível pelo usuário, o documento de especificação de requisitos contém as informações detalhadas que serão utilizadas pela equipe de projetistas para o desenvolvimento de uma solução de acordo com as necessidades do usuário (PFLEEGER; ATLEE, 2004).

2.1.1.4 Validação

A fase de validação é destinada para a identificação de inconscistências na especificação de requisitos para prever problemas futuros no projeto do software que muitas vezes surgem de requisitos levantados ou especificados de maneira incorreta. Nisso, o objetivo principal dessa fase é autenticar se os requisitos especificados atendem realmente as necessidades do cliente (SOMMERVILLE, 2011). Nessa etapa do processo de ER deve ser analizado:

- se os requisitos definem realmente os objetivos do cliente;
- o nível de abstração utilizado para definir tal requisito é apropriado;
- requisitos desnecessários que muitas vezes podem estar replicados ou descreverem incorretamente as necessidades do cliente podendo ser reescritos ou retirados da especificação de requisitos;
- a coesão das informações documentadas com as fontes dos requisitos (usuários, documentações, sistemas existentes e assim por diante);
- se o requisito pode ser implementado com os recursos existentes (equipe de desenvolvimento, contexto do cliente, entre outros);
- se os requisitos podem ser testados;
- a escrita dos requisitos para identificar se os mesmos apresentam as informações necessárias para a fase de projeto;
- a estrutura da especificação de requisitos para verificar se a divisão dos requisitos está apropriada;
- quais padrões de requisitos foram utilizados e se foram aplicados corretamente (PRESSMAN, 2011).

As técnicas de validação de requisitos podem ser aplicadas em grupo ou individualmente. Como principais abordagens utilizadas temos: revisões de requisitos, prototipação de validação e geração de casos de teste.

Revisões de Requisitos

É uma técnica tradicional de validação de requisitos que consiste na verificação manual do documentos de especificação. Essa verificação pode ser realizada em grupo dividindo a especificação de requisitos em partes para cada revisor. O objetivo principal dessa técnica é verificar visivelmente as inconscistências do modelo de requisitos (SOMMERVILLE, 2011).

Prototipação de Validação

Além da fase de análise, a prototipação pode ser aplicada para a validação de requisitos utilizando versões do sistema para identificar opiniões sobre os requisitos implementados. Pode ser utilizado protótipos que utilizam interfaces gráficas desenhadas em papel ou desenvolvidas como protótipos funcionais, onde a partir da utilização de sua interface o cliente pode formar suas opiniões ou críticas sobre o que lhe é apresentado.

Geração de Casos de Teste

A geração de casos de teste é uma prática muito utilizada pelo processo *Extreme Programming* e consiste na criação de testes para avaliar determinado caso de uso identificando problemas nos requisitos de usuário. Pode ser aplicado utilizando modelos de documentação existentes. Vale salientar que, se a criação ou a implementação do caso de teste for complexa a implementação do caso de uso pode ser difícil de ser realizada (SOMMERVILLE, 2011).

2.2 Sistemas Autoadaptativos

Diferentemente dos sistemas tradicionais que são baseados no princípio da menor surpresa [1], os sistemas autoadaptativos são capazes de surpreender seus usuários com ações inesperadas afim de suprir suas necessidades. Esses sistemas podem ser definidos como sistemas que correspondem as mudanças de ambiente podendo alterar o seu estado ou comportamento de acordo com elas (WHITTLE et al., 2010). Vale salientar que um sistema autoadaptativo é ciente de contexto, ou seja, utiliza seus próprios recursos (hardware ou software) para o conhecimento da intenção do usuário, o ambiente no qual ele está inserido e quais requisitos surgem para poder tratá-los e implementá-los (WHITTLE et al., 2010).

Sawyer et al. (2010) destacam que os sistemas autoadaptativos geralmente são encomendados para contextos complexos para tratar situações que podem surgir em tempo de execução dificultando o levantamento de requisitos, projeto e a codificação desses sistemas.

2.2.1 Propriedades de Autoadaptação

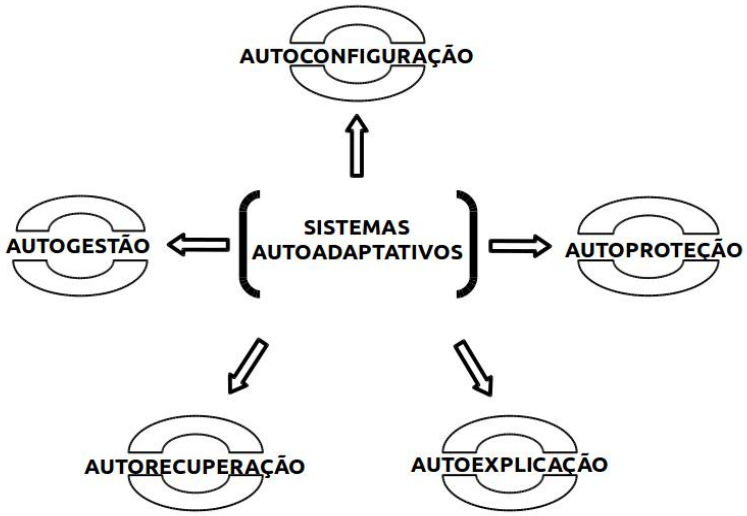
O termo "auto"deve ser utilizado para descrever sistemas que possuem potencial de realizar suas escolhas autonomamente utilizando ou não alguma intervenção humana (BRUN et al., 2009). A autoadaptação é um comportamento que pode incluir propriedades de autoconfiguração, autorecuperação, auto-otimização, autoproteção e autogestão (JUNIOR, 2013). Na Figura 2 podemos visualizar essas propriedades.

Autogestão

A autogestão é uma propriedade essencial em um sistema autoadaptativo, pois através dela o software é ciente das suas capacidades, funcionalidades, possíveis combinações para adaptar um novo recurso ou componente que pode pertencer a outra tecnologia. Também, a autogestão permite o gerenciamento de requisitos, ou seja, a capacidade do software identificar os requisitos de sua especificação que foram alterados por alguma

[1] O princípio da menor surpresa é derivado da engenharia humana, o qual diz que um software deve condizer com as experiências, expectativas e modelos mentais dos usuários (JEROME; KAASHOEK, 2009).

Figura 2 – Principais propriedades de um Sistema Autoadaptativo.



circunstância podendo ser ativado posteriormente uma propriedade de autoconfiguração para a atualização do modelo de requisitos (ANDRÉ et al., 2010).

Como exemplo de autogestão em sistemas autoadaptativos, tem-se o gerenciamento de consumo de energia, aplicação muito utilizada quando os sistemas são executados sobre uma RSSF (Rede de Sensores Sem Fios) conforme visto na abordagem de Bencomo et al. (2012), onde o consumo de energia elétrica é vital em suas funcionalidades. Nesse cenário, a autogestão pode ser visualizada quando os sensores decidem optar pela comunicação de rede *bluetooth* ao invés de *wifi*, pois a localização dos sensores alvos a serem comunicados estão em um curto alcançe, por outro lado a utilização *wifi* consumiria mais energia da bateria do sensor.

Autoconfiguração

Através da autoconfiguração é possível adicionar ou remover funcionalidades do software. Também, essa propriedade aplica-se a componentes sendo possível adicioná-los ou remove-los em tempo de execução (SURI; CABRI, 2014).

Em sistemas clássicos a adição de componentes ocorre normalmente de maneira manual, ou seja, o sistema é desligado e seu código é adaptado para receber tal componente. Já em um SA, a autoconfiguração pode ser utilizada de acordo com as características contextuais, na abordagem de Qureshi e Perini (2010b) utiliza-se o conceito de SBSs que utilizam componentes (sites de serviços) para apresentar ao usuário preços de passagens aéreas, hospedagens, pontos turísticos, entre outras informações que são convenientes de acordo com o contexto do usuário. Nisso, pode-se perceber que os componentes

nessa abordagem são sites de serviços que são utilizados pelo software exigindo do mesmo uma capacidade de autoconfiguração para interpretar e utilizar tais serviços da web.

Autorecuperação

A propriedade de autorecuperação busca identificar falhas de software ou hardware. Essa propriedade deve estar relacionada com a autoconfiguração, pois quando é detectada alguma falha em um dos componentes utilizados pelo software a autoconfiguração deve ser ativada para realizar uma troca de componente (SURI; CABRI, 2014).

Autoproteção

A autoproteção é utilizada para tratar a segurança em um SA evitando ataques que ameaçam as funcionalidades ou os dados do software (SURI; CABRI, 2014). No exemplo de Qureshi e Perini (2010b) os SBSs utilizam serviços da *web* como componentes, nesse cenário a capacidade de autoproteção deve garantir a preservação do sistema na utilização desses serviços que podem disparar ameaças tanto para o sistema como também aos dados do usuário.

Autoexplicação

Através da autoexplicação um SA pode apresentar quais os motivos foram considerados ao realizar determinada ação. O exemplo de Bencomo et al. (2012) apresentado na propriedade de autogestão pode ser utilizado para o entendimento dessa propriedade. No cenário apresentado anteriormente, o SA instalado no sensor escolhe a rede de *bluetooth* ao invés da wifi para comunicar determinadas informações a outro sensor essa escolha foi selecionada visando a economia de energia. Nisso, a capacidade de autoexplicação desse sistema deve informar ao usuário que a rede *bluetooth* foi escolhida, pois, consome menos energia da bateria.

2.2.2 Abordagens de Sistemas Autoadaptativos

Nessa seção será apresentado as principais abordagens de SAs que implementam algumas das propriedades de autoadaptação apresentadas anteriormente.

Sistemas Autonômicos

A *International Business Machines* (IBM) apresentou em 2001 o conceito de computação autonômica para sistemas complexos comparando um sistema nervoso humano a uma abordagem computacional de sistemas afim de demonstrar que como o sistema nervoso humano controla as demais partes do corpo (braços, pernas e outros membros), um sistema autonômico pode se auto-gerenciar controlando seus componentes podendo realizar manutenções em si próprio (HUEBSCHER; MCCANN, 2008).

Além disso, o objetivo principal de um sistema autonômico é manter sua capacidade funcional estável, através de atividades de auto-otimização, autorecuperação e auto-

reconfiguração (PIMENTEL, 2008). Esse tipo de sistema possui um comportamento de autoadaptação quando identifica quais são as suas necessidades reagindo em conformidade a elas.

Sistemas Inteligentes

Os Sistemas Inteligentes empregam métodos de raciocínio para resolver problemas, muitas vezes utilizam recursos (hardware ou software) limitados para atender as expectativas do usuário (HERCZEG, 2010). Também, esses sistemas possuem a capacidade de explicar suas ações (propriedade de autoexplicação), justificar os resultados obtidos podendo utilizar uma linguagem simples para o entendimento do usuário.

Essa abordagem de sistemas pode ser aplicada por exemplo no gerenciamento de falha no hardware, caso o sistema computacional estiver ligado e ocorrer alguma falha de hardware o sistema inteligente diagnostica a falha e realiza o processo necessário, primeiramente tratando o hardware e em segundo plano não interrompendo a tarefa do usuário. Caso a falha tenha sido identificada em um dos núcleos do processador, os processos podem ser transferidos para outro núcleo, assim o fluxo de tarefas do usuário não será afetado. Após isso, o sistema pode informar a falha, o processo realizado e justificar porque a decisão escolhida foi a melhor maneira de resolver tal problemática.

Sistemas Baseados em Serviços

SBSs utilizam serviços de terceiros (empresas tecnológicas, mecanismos de busca, listas telefônicas online, entre outros) disponíveis na internet como componentes para suas funcionalidades. O comportamento autoadaptativo desses sistemas se encontra em sua capacidade de adaptação e utilização desses serviços que podem sofrer modificações ou estarem indisponíveis com o tempo (QURESHI; PERINI, 2010b).

Como exemplo de aplicação, um SBS pode ser utilizado para auxiliar o usuário em um supermercado informando uma lista de produtos necessários para o seu consumo, essa lista pode ser construída de acordo com o histórico de compras desse usuário podendo ser obtido por algum serviço disponível pelo supermercado que contém os registros das últimas compras realizadas pelo titular do CPF (usuário). Também, ao formular uma lista de produtos o sistema leva em conta as condições finaceiras do usuário podendo auxiliá-lo na escolha de produtos.

2.2.3 Abordagens de apoio ao Desenvolvimento de Sistemas Autoadaptativos

A autoadaptação é um assunto muito pesquisado em diferentes áreas da computação como: sistemas distribuídos, computação autonômica, interfaces adaptáveis, inteligência artificial, aprendizagem de máquina, sistemas financeiros de apoio a bolsa de valores, planejamento estratégico-militar, Redes de Sensores Sem Fio, computação ubíqua ou pervasiva, engenharia de software e tantas outras (BRUN et al., 2009).

Brun et al. (2009) descreve que a autoadaptação na engenharia de software é um tópico muito investigado principalmente no desenvolvimento de *middleware* adaptativo, ER, adaptabilidade por composição, arquitetura e projeto de software, entre outros.

O *middleware* adaptativo possibilita a adequação de componentes de software a um sistema aumentando sua potencialidade funcional. Um exemplo dessa abordagem poderia ser os SBSs que podem utilizar um *middleware* adaptativo para auxiliar na utilização dos serviços diponíveis na internet (SAWYER et al., 2010).

A ER para SAs também é um tópico muito investigado, pois como um SA deve ser ciente de contexto nem sempre a ER tradicional pode auxiliar na investigação, análise, especificação e validação de requisitos. Nisso, surgem novas técnicas para auxiliar esse processo como linguagens para descrever requisitos derivados do contexto, modelagem de metaobjetivos, *framework*s de apoio e assim por diante.

Além disso, a adaptabilidade por composição é uma técnica econômica para manutenção e otimização de sistemas que são compostos por componentes. Como exemplo, pode-se entender que em um sistema de gerenciamento de redes que utiliza essa técnica pode realizar mudanças inpendementes na topologia da rede lógica, balanceamento de carga ou nas características da rede. Também, essa abordagem pode ser encontrada em roteadores inteligentes que possuem a capacidade de otimizar o seu desempenho de acordo com as características da rede (SAWYER et al., 2010).

Acrescenta-se a esses tópicos a arquitetura e projeto de SA, onde surgem técnicas de projeto orientadas à arquitetura e requisitos. Um exemplo de abordagem orientada à arquitetura é o *framework* Rainbow que utiliza um conjunto de elementos arquiteturais para realizar adaptações funcionais no software. Quanto a modelos orientados a requisitos, como exemplo de abordagem tem-se o método Zanshin que é destinado ao desenvolvimento de sistemas que utilizam aspectos de *Control Theory* (Teoria de Controle), essa técnica é orientada à ER e utiliza em seu escopo um modelo de especificação *Goal-Oriented Requirements Specification* que trata os requisitos capturados do ambiente (problema-alvo) (JUNIOR, 2013).

2.3 Linguagem Relax

Geralmente, um requisito de sistema é descrito em formato textual para representar uma ação que o sistema deve executar. Além dessa ação, a descrição de um requisito pode fornecer informações de ambiente, quantidade, frequência, tempo, ordem, eventos e assim por diante. Em um SA esses aspectos são fundamentais para as atividades de projeto, a compreensão do contexto (ambiente ou intenção) do usuário é a principal base de conhecimento para se projetar um sistema ciente das necessidades do mesmo.

A linguagem Relax tem por objetivo apoiar o desenvolvedor de software na especificação de requisitos para SAs, oferecendo recursos para representar os aspectos de incerteza contido nos requisitos (WHITTLE et al., 2010). Esses aspectos exigem de um SA uma flexibilidade para comportar situações não convecionais que surgem do ambiente que o sistema está inserido.

A seguir é apresentado um exemplo (traduzido e retirado de (WHITTLE et al., 2010)) de aplicação dos operadores da linguagem Relax, como também a geração da expressão gramática desse requisito e a expressão formal associada a sua expressão.

RF1 em Linguagem Textual Descritiva

O frigorífico deve detectar e comunicar os alimentos com embalagens.

RF1 em Linguagem Textual com Operadores

O frigorífico *SHALL* detectar e comunicar *AS MANY AS POSSIBLE* os alimentos com embalagens.

RF1 em Expressão Gramática

SHALL (AS MANY AS POSSIBLE p)

RF1 em Expressão Formal Relax

AGF$(\Delta(p) \in S)$

A partir da utilização dos operadores é possível mapear por exemplo que a ação "detectar e comunicar"ocorre no presente, já os alimentos com embalagens podem ser identificados de forma flexível. Essa análise é possível pelo fato de que cada operador possui um significado, a expressão gramática gerada do requisito fornece a síntese do que é relevante no requisito mapeado.

Na expressão formal Relax é definido um quantificador (representado pela sigla **AGF**), a preposição representada pelo símbolo "p" é utilizada como parâmetro do símbolo Δ, afim de mostrar que essa preposição é variável e que essa variação possui uma relação de pertinência com um conjunto *Fuzzy* (definido pela letra "S"). Esse formalismo é adotado pelos autores da linguagem para definir a semântica das expressões gramáticas geradas no processo de conversão.

2.3.1 Vocabulário da Linguagem

O vocabulário da linguagem Relax pode ser divido em operadores e fatores de incerteza. Os operadores possibilitam a representação dos aspectos temporais, modais e ordinais contidos nos requisitos, a Tabela 1 apresenta os operadores fornecidos pela linguagem e sua descrição.

Os operadores da linguagem são fundamentais, pois através deles que a descrição

Tabela 1 – Operadores Relax (WHITTLE et al., 2010).

Operador Relax	**Descrição**
Operadores Modais	
SHALL	O requisito deve conter.
MAY..OR	O requisito especifica uma ou mais alternativas.
Operadores Temporais	
EVENTUALLY	O requisito deve conter eventualmente.
UNTIL	O requisito deve garantir até uma posição futura.
BEFORE, AFTER	O requisito deve realizar antes ou depois de um determinado evento.
IN	O requisito deve realizar durante um determinado tempo.
AS EARLY, LATE AS POSSIBLE	Um requisito especifica algo que deve realizar, logo ou deve se atrasar o maior tempo possível.
AS CLOSE AS POSSIBLE TO [frequência]	Um requisito especifica algo que acontece repetidamente, mas a frequência pode ser flexibilizada.
Operadores Ordinais	
AS CLOSE AS POSSIBLE TO [quantidade]	Um requisito especifica algo que acontece repetidamente, mas a quantidade pode ser flexibilizada.
AS MANY, FEW AS POSSIBLE	O requisito especifica uma quantidade contável, mas o número exato pode ser flexibilizado.
Fatores de Incerteza	
ENV	Define um conjunto de propriedades que compreendem o ambiente do sistema.
MON	Define um conjunto de propriedades que podem ser monitoradas através do sistema.
REL	Define a relação entre as propriedades **ENV** e **MON**.
DEP	Identifica as dependências entre os requisitos (relaxados e invariantes).

textual do requisito é mapeada. A seguir é apresentado outro exemplo de uma descrição informal de requisito, após isso o requisito é convertido para uma expressão relax (utilizando a gramática definida).

Exemplo de Descrição de um Requisito

```
The system SHALL tap AS EARLY AS POSSIBLE ensure

that the user's hands to find the water is released

and according to the remoteness of the hands the

water flow decreases IN 3 seconds.
```

Exemplo de Expressão Relax do Requisito Anterior

```
SHALL (AS EARLY AS POSSIBLE ( IN t p))
```

A partir da utilizaçao dos operadores Relax é possível mapear as expressão do requisito, pois os operadores sinalizam aspectos importantes na descrição do requisito. No exemplo de descrição de um requisito, o operador *SHALL* precede uma proposição (ação), já o operador *AS EARLY AS POSSIBLE* anuncia uma ação que ocorre antecipadamente, indicando prioridade perante as demais. Também, no exemplo é apresentado o operador *IN* que indica que a preposição ou ação anterior deve ocorrer em 3 segundos (unidade de tempo).

A expressão Relax no segundo exemplo é o resultado do processo de interpretação do requisito *relax-ed*, essa expressão representa os aspectos fundamentais contidos no requisito, os quais são importantes para a construção de um SA. Com base nessa expressão, pode-se eliminar redundâncias e destacar informações relevantes em uma especificação de requisitos. Vale salientar que, o símbolo "t"na expressão representa a unidade de tempo e o símbolo "p"define a preposição de propósito geral do requisito.

2.3.1.1 Sintáxe

As regras de produção da gramática Relax estabelecem as possibilidades de construção a partir do vocabulário da linguagem. A seguir é possível visualizar essas regras.

ϕ : = true | false | p | SHALL ϕ

| MAY ϕ_1 OR MAY ϕ_2

| EVENTUALLY ϕ

| ϕ_1 UNTIL ϕ_2

| BEFORE e ϕ

| AFTER e ϕ

| IN t ϕ

| AS CLOSE AS POSSIBLE TO f ϕ

| AS CLOSE AS POSSIBLE TO q ϕ

| AS EARLY, LATE, MANY, FEW AS POSSIBLE ϕ

Além dos operadores podemos visualizar nas regras de produção que alguns sufixos são utilizados para representar eventos, frequências e quantidades. O sufixo de eventos "e"é utilizado pelos operadores *BEFORE* e *AFTER* para sinalizar eventos, já o sufixo "f"e "q"são utilizados pelo operador *AS CLOSE AS POSSIBLE TO*, representando frequência e quantidade. Também, os aspectos temporais podem sinalizados pelo sufixo "t", tal sufixo é utilizado pelo operador *IN*.

2.3.1.2 Semântica

A semântica da linguagem Relax utiliza recursos da *Fuzzy Branching Temporal Logic* (FBTL) para possibilitar a representação das possíveis alternativas disparadas pelas proposições, esse formalismo é baseado em lógica *fuzzy* (WHITTLE et al., 2010). A relação dos formalismos da linguagem Relax é representada pela Tabela 2.

Tabela 2 – Semântica para Expressão Relax (WHITTLE et al., 2010).

Expressão Relax	**Informal**	**Formalização FBTL**
SHALL ϕ	ϕ é verdade em qualquer estado	**AG** ϕ
MAY ϕ_1 *OR MAY* ϕ_2	em qualquer estado, ou ϕ_1 ou ϕ_2 são verdades	**AG** $(\phi_1 or \phi_1)$
EVENTUALLY ϕ	será verdade em algum estado futuro	**AF** ϕ
ϕ_1 *U* ϕ_2	ϕ_1 será verdadeiro até ϕ_2 tornar-se verdadeiro	**AF** ϕ
BEFORE e ϕ	ϕ é verdade em qualquer estado ocorrendo antes do evento **e**	**A** $\chi_<$ e_d é a duração até a próxima ocorrência de **e**
AFTER e ϕ	ϕ é verdade em qualquer estado que ocorre após o evento **e**	**A** $\chi_>$ e_d ϕ
IN t ϕ	ϕ é verdade em todo o estado no intervalo de tempo **t**	(*AFTER* $t_s start$ ϕ and *BEFORE* $t_e nd$ ϕ) onde $t_s start$, $t_e nd$ são eventos que denotam o início e o fim do intervalo **t**, respectivamente
AS EARLY AS POSSIBLE ϕ	ϕ torna-se verdadeiro em algum estado tão perto do tempo atual quanto possível	**A** $\chi_>$ $=_d$ ϕ onde d é a duração difusa definida de tal forma que seus membros de função tem o seu máximo em 0 (ou seja, M(0)= 1) e diminui continuamente para os valores > 0
AS LATE AS POSSIBLE ϕ	ϕ torna-se verdadeiro em algum estado tão próximo do tempo t $= \infty$ possível	**A** $\chi_>$ $=_d$ ϕ onde d é a duração difusa definida de tal forma que seus membros de função tem o seu valor mínimo em 0 (ou seja, M(0)= 0) e aumenta continuamente para os valores >0

A lógica *Fuzzy* permite representar valores aproximados, diferente da lógica *Crisp* que assume valores precisos como verdadeiro ou falso. Também, a lógica *Fuzzy* permite descobrir qual é o grau de pertinência de um elemento ao seu conjunto, um exemplo dessa abordagem é analisar qual é o grau de pertinência de um pixel a cor azul e vermelha, divergente a isso, em lógica *Crisp* o pixel só poderia ser azul ou vermelho (BILOBROVEC; KOVALESKI, 2004).

Expressão Relax	Informal	Formalização FBTL
AS CLOSE AS POSSIBLE TO f ϕ	ϕ é verdade em intervalos periódicos, onde o período é tão perto de f possível	**A** ($\chi =_d \phi$ and $\chi = 2_d \phi$ and $\chi = 3_d \phi$ and...) onde **d** é a duração difusa definida de tal modo que a sua função de adesão tem o seu valor máximo, no período definido por f (ou seja, M(d)= M(2d)=...=1) e diminui continuamente para valores inferiores e superiores a d (e 2d, ...)
AS CLOSE AS POSSIBLE TO q ϕ	existe alguma função Δ tal que $\Delta(\phi)$ são quantificáveis e $\Delta(\phi)$ é tão próximo de 0 possível	**AF**(($\Delta(\phi)$-q) pertence a S, onde S é um conjunto fuzzy cuja função de composição tem um valor no zero (M(0)= 1) e diminui continuamente em torno de zero. $\Delta(\phi)$ "conta"o quantificável, que será comparado com q
AS MANY AS POSSIBLE ϕ	existe alguma função Δ tal que $\Delta(\phi)$ é tão perto de ∞ possível	**AF** ($\Delta(\phi)$ pertence a S), onde S é um conjunto fuzzy cuja função de pertinência tem 0 valor no zero (M(0)= 0) e aumenta continuamente em torno de zero
AS FEW AS POSSIBLE ϕ	existe alguma função Δ tal que $\Delta(\phi)$ é quantificáveis e é o mais próximo possível a 0	**AF** ($\Delta(\phi)$ pertence a S), onde S é um conjunto fuzzy cuja função de pertinência tem um valor no zero (M(0)= 1) e diminui continuamente em torno de zero

2.4 Considerações do Capítulo

Ao longo desse capítulo foi possível compreender a importância do processo de ER no desenvolvimento de software, pois muitas vezes a maioria dos problemas que surgem durante a fase de projeto ou de codificação são derivados de requisitos mal compreendidos. A ER proporciona o conhecimento das necessidades do cliente, sendo que ao propor uma solução de sistema a ele, primeiramente é necessário entender o seu problema. Também, foi possível conhecer as propriedades de um Sistema Autoadaptativo, as principais necessidades para o desenvolvimento e compreender sua diferença quando comparado a sistemas tradicionais.

Da mesma forma, o estudo sobre a linguagem Relax possibilitou o entendimento do suporte a especificação de requisitos oferecido para SAs, através de seu vocabulário é possível mapear aspectos de incerteza, ambiente, frequência, quantidade e eventos contidos no requisito. Tais aspectos são necessários para que o SA conheça o contexto no qual o usuário está inserido, a partir da utilização dessa linguagem é possível eliminar a redundância e as inconsistências nas especificações de requisitos, facilitando a construção do projeto desses sistemas.

3 Base Tecnológica

O presente capítulo tem por objetivo apresentar as tecnologias utilizadas no trabalho. A seção 3.1 contextualiza o funcionamento básico de um processador de linguagem (compilador), sua estrutura e tecnologia utilizada para a geração de analisadores léxicos e sintáticos. Após isso, a seção 3.2 apresenta a plataforma Eclipse para o desenvolvimento de *plugins*, a arquitetura disponibilizada pela plataforma para a adição de *plugins* e a estrutura básica de um *plugin*.

3.1 Processador de Linguagem

Um processador de linguagem (compilador) é programa capaz de interpretar um artefato de linguagem e apartir desse produzir um artefato traduzido em linguagem objeto (AHO; SETHI; LAM, 2008). A partir do processo de interpretação e tradução, o processador de linguagem pode sinalizar erros léxicos e de construção (sintáticos).

Um exemplo de utilização de compiladores são as linguagens de programação de alto-nível, por exemplo, um programa Java seria o artefato de entrada (linguagem fonte) para o compilador, já a saída seria um programa de linguagem intermediária (linguagem objeto), o qual é interpretado pela Java *Virtual Machine* (JVM) (DEITEL, 2010).

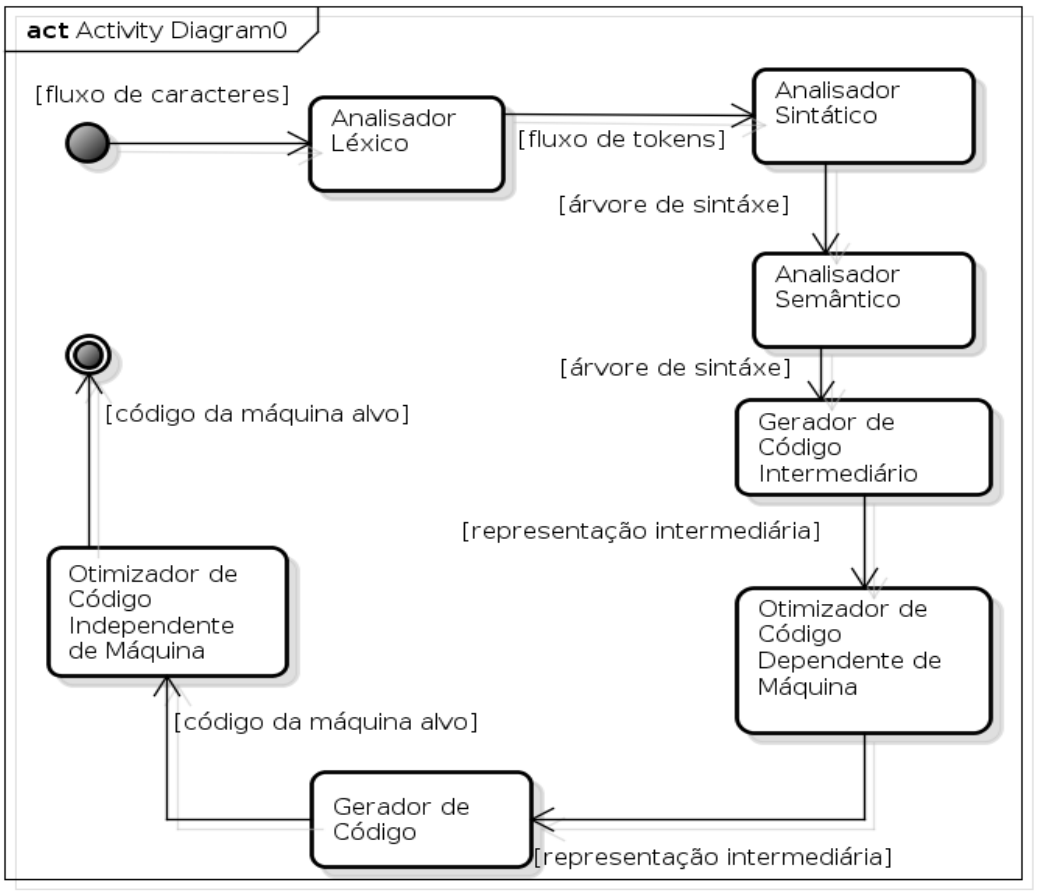
O processador de linguagem possui uma estrutura que pode ser diferenciada de acordo com a sua abordagem, a estrutura fundamental é apresentada na Figura 3.

A primeira fase é a análise léxica, nela o compilador mapeia um fluxo de caracteres identificando possíveis *lexemas*, os quais definem *tokens* (símbolos do alfabeto da linguagem) e o seu valor associado (AHO; SETHI; LAM, 2008). A partir disso, para cada *token* mapeado, o mesmo é registrado junto com o seu valor em uma tabela de símbolos. Posteriormente a isso, na segunda fase, o compilador realiza a análise sintática para representar a sequência de *tokens* mapeados anteriormente na análise léxica. Essa fase gera a primeira ávore de sintáxe, através dela é possível avaliar a construção da linguagem baseando-se na prioridade dos *tokens* (AHO; SETHI; LAM, 2008).

Na terceira fase, o compilador utiliza a árvore sintática e a tabela de símbolos para implementar a análise semântica, esse tipo de verificação é fundamental, pois com base nisso, o compilador analisa a consistência da expressão com o sentido da linguagem (AHO; SETHI; LAM, 2008). Um exemplo de violação do sentido de linguagem seria a atribuição de um valor texto a uma variável de tipo inteiro.

A quarta fase é dedicada a geração do código intermediário, geralmente esse código é produzido pelo compilador para facilitar a tradução para a máquina-alvo. A otimização

Figura 3 – Estrutura fundamental de um compilador (AHO; SETHI; LAM, 2008).



de código é realizada no código intermediário gerado, essa etapa é dependente da arquitetura onde o compilador está sendo executado (AHO; SETHI; LAM, 2008). Finalizando as etapas, o compilador gera o respectivo código (linguagem objeto) utilizando como base o código intermediário gerado anteriormente, em seguida é realizado uma otimização de código independente da máquina-alvo.

3.1.1 Geradores para Criar Analisadores Léxico e de Sintáxe

O JFlex é um programa que gera analisadores léxicos para Java utilizando especificações léxicas escritas pelo programador em Java (KLEIN; DéCAMPS, 2015).

Diferente do JFlex, o CUP é um programa que gera analisadores sintáticos para Java, as duas tecnologias podem ser integradas, geralmente as implementações em CUP utilizam o *scanner* sintático gerado pelo JFlex (KLEIN; DéCAMPS, 2015). Ambos geradores possuem como entrada especificações escritas em formato JFlex ou CUP, a partir disso a tecnologia gera as classes analisadoras. Essas classes disponibilizam métodos gerais para contagem de caracteres, verificação de sintáxe (construção) e demais funções que podem ser utilizadas no processo de análise léxica.

3.1.1.1 Estrutura de uma Especificação Léxica JFlex

A estrutura de uma especificação JFlex é dividida em três blocos, o primeiro bloco é chamado de *user code*, nele é descrito o nome do pacote que a classe está alocada e suas importações (caso utilize métodos de outra classe) (KLEIN; DéCAMPS, 2015). O segundo bloco de especificação define declarações e macros, as declarações descrevem o nome da classe e variáveis, já as macros definem diretivas interpretadas pelo programa JFlex que são utilizadas para sinalizar determinados formatos. Já no terceiro bloco são definidas as regras léxicas, nessa seção da especificação JFlex podem ser definidos estados léxicos e expressões regulares.

A seguir é apresentado um exemplo de especificação léxica, no primeiro bloco de especificação é definido o nome da classe, a macro *unicode* configura os caracteres mapeados pelo analisador, outra macro definida é *cup* para sinalizar que o leitor léxico pode ser utilizado pela biblioteca CUP. Também, são definidas as macros *line* e *column*, as quais indicam que a classe gerada fornece métodos para a contagem de linhas e colunas (muito utilizado para indicar a localização do erro sintático).

```
1  %%
2
3  %class Lexer
4  %unicode
5  %cup
6  %line
7  %column
8
9  %{
10     StringBuffer string = new StringBuffer();
11     private Symbol symbol(int type){
12         return new Symbol(type,yyline,yycolumn);
13     }
14     private Symbol symbol(int type, Object value){
15         return new Symbol(type,yyline,yycolumn,value);
16     }
17 %}
18
19 LineTerminator = \bar | \n | \r\n
20 InputCharacter = [\r\n]
21 WhiteSpace = {LineTerminator} | [ \t\f]
22
23 /*comments*/
24 Comment = {TraditionalComment} | {EndOfLineComment} | {
     DOcumentationComment}
```

```
25
26 TraditionalComment = "/+" ["*/"] ""*/" | "/*" "*" + "/"
27
28 /*Comment can be the last line of the file, without line
      terminator*/
29 EndOfComment = "//" {InputCharacter}* {LineTerminator}?
30 DocumentationComment = "/**" {CommentContent} "*"+ "//"
31 CommentContent = ([\^*] | //*+ [\^//*])*
```

3.1.1.2 Estrutura de uma Especificação Sintática CUP

Uma especificação sintática possibilita a definição dos símbolos terminais (símbolos não férteis), não terminais, precendentes pela esquerda ou direita e assim por diante (HUDSON, 1999). Essa definição é utilizada pelo bloco de especificação de gramática ao definir a construção da linguagem. A seguir é apresentado um exemplo de especificação CUP (HUDSON, 1999).

```
1  //Cup specification for a simple expression evaluator (no actions
      )
2  import java_cup.runtime.*;
3
4  /*Preliminaries to set up and use the scanner.*/
5  init with {: scanner.init(); :}
6  scan with {: return scanner.next_token(); :}
7
8  /*Terminals (tokens returned by the scanner)*/
9
10 terminal SEMI, PLUS, MINUS, TIMES, DIVIDES, MOD;
11 terminal UMINUS, LPAREN, RPAREN;
12 terminal Integer NUMBER;
13
14 /*Non terminals*/
15 non terminal expr_list, expr_part;
16 non terminal Integer expr, term, factor;
17
18 /*Precedences*/
19 precedence left PLUS, MINUS;
20 precedence left TIMES, DIVIDE, MOD;
21 precedence left UMINUS;
22
23 /*The grammar*/
24 expr_list ::= expr_list expr_part |
```

```
25 expr_part;
26 expr_part ::= expr SEMI;
27 expr ::= expr PLUS expr
28 | expr MINUS expr
29 | expr TIMES expr
30 | expr DIVIDE expr
31 | expr MOD expr
32 | MINUS expr prec UMINUS
33 | LPAREN expr RPAREN
34 | NUMBER
35 ;
```

Na especificação CUP anterior foi definido uma gramática de expressões aritméticas, a primeira parte da especificação descreve os símbolos que serão utilizados, após isso é especificado o bloco de gramática que indica as regras de produção que serão utilizadas pelo analisador sintático gerado.

3.2 Plataforma Eclipse para o Desenvolvimento de *Plugins*

A plataforma Eclipse é baseada em *plugins* que são utilizados para ampliar as funcionalidades da *Integrated Development Environment* (IDE) (ECLIPSE FOUNDATION, 2014). Esses *plugins* são codificados na linguagem de programação Java e podem oferecer diversas modalidades de serviço como biblioteca de códigos, guias de documentação ou uma extensão da própria plataforma (DESRIVIERES; WIEGAND, 2004).

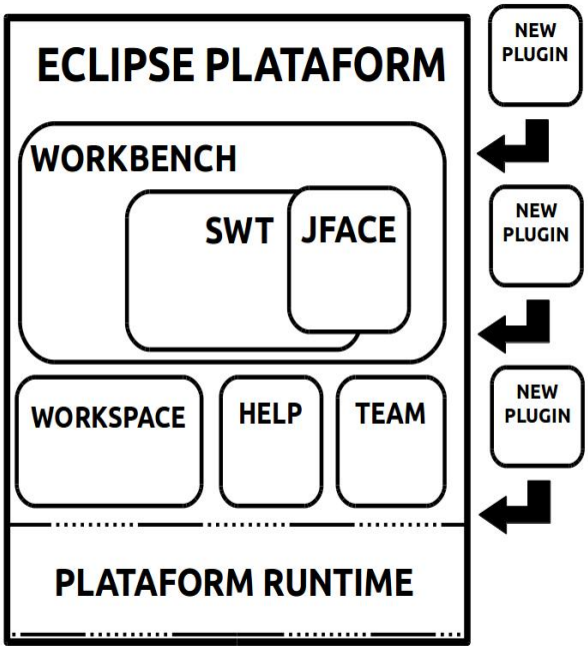
As bibliotecas de código auxiliam os programadores com funcionalidades já implementadas e podem ser oferecidas pelo *plugin* em formato de *Application Programming Interface* (API). A documentação auxilia os desenvolvedores nos aspectos técnicos da linguagem de programação, bibliotecas de código, demonstrando os recursos disponíveis e sua aplicações. Na extensão da plataforma, os desenvolvedores utilizam os componentes e recursos da plataforma Eclipse como interface gráfica, mecanismos de interpretação textual e de compilação para desenvolverem soluções tecnológicas para outras finalidades como por exemplo uma ferramenta de apoio a uma nova linguagem de programação.

3.2.1 Arquitetura da Plataforma Eclipse

A arquitetura da plataforma Eclipse é composta por *Workspace*, *Workbench*, *JFace*, *Standard Widget Toolkit* (SWT), *Help*, *Team* e os demais *plugins* acoplados. Na Figura 4 é demonstrado esses elementos e sua organização na plataforma.

Workspace

Figura 4 – Arquitetura da Plataforma Eclipse (DESRIVIERES; WIEGAND, 2004).



É o espaço de trabalho do desenvolvedor, uma pasta destinada ao armazenamento dos projetos de alto-nível como por exemplo projetos escritos em Java, PHP, C++, entre outros (DESRIVIERES; WIEGAND, 2004).

Workbench

É o ambiente principal da ferramenta, construído com a utilização dos componentes *JFace* e SWT, proporciona recursos de edição de código, execução, depuração, visualização das estruturas do projetos (localizados no *workspace*), adição de *plugins*, criação de novos projetos e assim por diante.

JFace

Define um conjunto de ferramentas de interface de usuário, como área de preferências, módulo de configuração do ambiente de desenvolvimento e de projeto (DESRIVIERES; WIEGAND, 2004).

Standard Widget Toolkit

Biblioteca de componentes gráficos compatível com diferentes Sistemas Operacionais. Através dela é possível construir interfaces gráficas, como por exemplo a *JFace*.

Help

Mecanismo de auxílio para utilizadores do ambiente, através dele é possível acessar guias e documentações referentes ao ambiente ou ao *plugin* utilizado.

Team

Mecanismo que permite o versionamento de projeto através da inicialização, clone, configuração e gerenciamento de repositórios remotos.

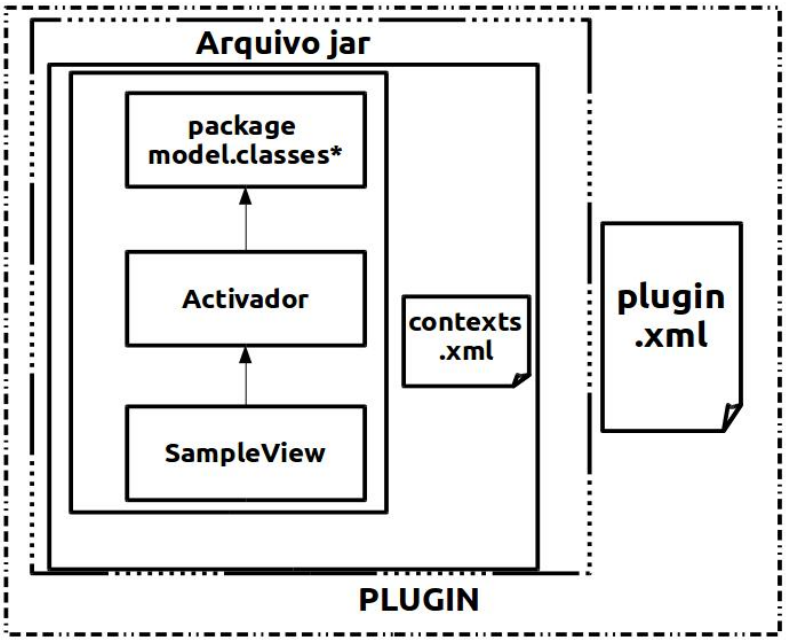
Plataforma de tempo de execução

É acionado na inicialização da IDE, sua tarefa é descobrir quais *plugins* estão disponíveis e interpretar o seus arquivos de configuração.

3.2.2 Arquitetura de um *Plugin* Eclipse

Um *plugin* é composto por um Java *Archive* (JAR) e um arquivo denominado *plugin*, escrito em XML. Na Figura 5 é possível visualizar a organização desses elementos e seus itens.

Figura 5 – Arquitetura de um *plugin* eclipse.



Java *Archive* (JAR)

Um JAR contém os códigos java e os arquivos utilizados pela aplicação como imagens, dados e configurações. Também, os arquivos JAR são interpretados pela máquina virtual Java podendo ser utilizados como bibliotecas, sistemas executáveis ou componentes de software.

No desenvolvimento de *plugins* o arquivo JAR é utilizado como componente de software que será acomplado ao ambiente de desenvolvimento Eclipse. O JAR *plugin* pode ser dividido em classes java e arquivo *contexts*.

As classes são organizadas de acordo com suas finalidades, como por exemplo podem ser definidos níveis arquiteturais para o controle dos recursos da aplicação, manipulação de componentes de interface gráfica e tratamento das operações que envolvem processamento de dados. Na classe *Activator* é realizado o controle do ciclo de execução do *plugin* e o acesso das preferências do ambiente Eclipse para realizar possíveis configurações (BALSIGER, 2010). A classe *SampleView* é utilizada para registrar e controlar os elementos da interface gráfica do ambiente. As classes de processamento de dados são utilizadas para gerenciar as informações da aplicação (BALSIGER, 2010).

O JAR também é composto pelo arquivo *contexts* que é escrito em XML. Através dele é possível registrar as principais informações sobre o *plugin* para auxiliar o usuário no processo de instalação, utilização e configuração do mesmo no ambiente Eclipse (BALSIGER, 2010).

Arquivo de Configuração

Conforme apresentado na figura a arquitetura do *plugin* contém um arquivo de configuração definido como "plugin.xml", nele se registra as dependências, os pacotes utilizados pelo *plugin*, as extensões e os pontos de extensão (BALSIGER, 2010).

3.2.3 *Plugin Development Environment*

O PDE (*Plugin Development Environment*) permite o desenvolvimento de *plugins* para a plataforma eclipse incluindo atividades de teste, depuração e implantação de *plugins* (ECLIPSE FOUNDATION, 2014). Esse *plugin* é composto por componentes de interface gráfica de usuário, ferramentas da API e *Build*.

Os componentes de interface gráfica de usuário fornecem editores de codificação, assistentes de editor e manifesto, recursos de exportação, módulos de configuração de *plugin* e assim por diante. Já as ferramentas da API são destinadas ao versionamento, manutenção de incompatibilidades, otimizações das APIs fornecidas pelo *plugin*. Além disso, o *Build* automatiza o desenvolvimento de *plugins* gerando arquivos de configuração como "build.properties"e "plugin.xml". Esse componente é muito utilizado no empacotamento e distribuição do *plugin* desenvolvido que pode ser disponibilizado em um repositório remoto (ECLIPSE FOUNDATION, 2014).

3.3 Considerações do Capítulo

O estudo sobre processadores de linguagem possibilita a compreensão de quais elementos são analisados pela óptica léxica e como é realizado uma verificação sintática. A utilização de tecnologias para apoiar essas verificações auxiliam o desenvolvimento de interpretadores de linguagem, garantindo maior confiabilidade na geração de resultados

(em linguagem objeto) e produtividade na codificação.

Além disso, a construção de um *plugin* exige o discernimento dos componentes disponibilizados pela plataforma Eclipse, a partir desse estudo foi possível compreender os recursos disponibilizados por essa plataforma, como também suas vantagens sob os demais ambientes de desenvolvimento.

4 Trabalhos Relacionados

Esse capítulo apresenta um conjunto de trabalhos relacionados com as questões de pesquisa investigadas no presente estudo, essa seleção de trabalhos descrevem ferramentas de apoio a ER e são propostas por diferentes autores. Na seção 4.1 será apresentado a metodologia utilizada para coleta dos trabalhos, já na seção 4.2 será abordado os trabalhos relacionados e a seção 5.1 descreverá a análise sobre esses trabalhos.

4.1 Metodologia

Para coletar os trabalhos relacionados realizou-se, antecipadamente, um planejamento de pesquisa, no qual foi estabelecido a questão de pesquisa, base de busca, palavras chave e critérios de seleção dos artigos.

Foram definidas duas questões de pesquisa, a primeira direcionada a obter soluções de ferramentas de suporte à ER e outra destinada a descoberta de ferramentas de suporte à especificação de requisitos para SAs.

A base de busca utilizada foi o Google *Scholar* pela indexação de inúmeras bases de pesquisa, como IEEE (*Institute of Electrical and Electronics Engineers*) *Xplore Digital Library*, ACM (*Association Computing Machinery*) *Digital Library*, *Science Direct*, *Springer*, entre outras.

Além disso, foram definidas algumas palavras-chave para orientar a pesquisa como *requirements engineering*, *support tool* e *self-adaptive systems*. Considerando publicações em português e inglês.

Foram utilizados os seguintes critérios para a seleção dos artigos:

- deve ter no mínimo 6 páginas;
- publicados entre os anos 2009 a 2014 (últimos 5 anos);
- disponível para download;
- a ferramenta proposta pelo trabalho deve apresentar o experimento da ferramenta e os resultados adquiridos com a mesma.

4.2 Trabalhos

Os trabalhos coletados foram agrupados em dois conjuntos, o primeiro que apresenta as ferramentas de apoio ao processo de ER e outro destinado as ferramentas que

apoiam a ER para SAs.

4.2.1 Ferramentas de Apoio ao Processo de Engenharia de Requisitos

Butler e TSSG (2011) apresentam a ferramenta STS-*tool* que suporta a linguagem gráfica STS para modelagem de requisitos. Essa linguagem apresenta uma abordagem orientada ao mapeamento de requisitos de segurança, nisso a ferramenta desenvolvida pelos autores apoia o processo de diagramação desses requisitos. Também, Azevedo et al. (2011) abordam o desenvolvimento de um editor de diagramas *brainstorming* colaborativo e síncrono afim de apoiar o processo de investigação de requisitos.

Quanto ao suporte a verificação e validação de modelos de requisitos, Ali, Dalpiaz e Giorgini (2010) propõe a ferramenta RE-*Context* para o suporte a análise automatizada de modelos meta-contextuais afim de verificar inconscistências e conflitos em requisitos. Em um trabalho mais recente dos autores (ALI; DALPIAZ; GIORGINI, 2014) foi proposto uma nova funcionalidade para a ferramenta *RE-Context*, um mecanismo automatizado para detectar inconsistências em modelo de requisitos.

Acrescenta-se a categoria de verificação e validação, o mecanismo automatizado de validação de requisitos proposto por Ito et al. (2011) que oferece dois módulos, um destinado para a validação de requisitos e outro para a geração de relatórios. A validação de requisitos é o módulo que analisa os requisitos de acordo com os critérios de ambiguidade, conflito, gramática, coesão, completeza, consistência, padronização, relevância, viabilidade, organização, identificação única, clareza e domínio. A partir da validação automatizada, o mecanismo também gera relatórios de acordo com os resultados adquiridos.

Umoh, Sampaio e Theodoulidis (2011) mostram uma ferramenta para orientar o processo de ER em projetos RAD (*Rapid Application Development*) para apoiar o desenvolvimento de aplicativos financeiros compondo o *framework* Refinto. Do mesmo modo, como ferramenta de suporte a metodologias, MK e S (2009) apresentam a ferramenta FGD-*Relicit* que semi-automatiza o processo de especificação de requisitos. Também, Beckers et al. (2013) desenvolveram um método sistemático para a investigação de requisitos de segurança em cenários de *Clound Computing* apoiado pela ferramenta gráfica desenvolvida com o objetivo de apresentar os cenários investigados.

Além disso, Daramola Tor Stalhane e Omoronyia (2011), Daramola e Stalhane (2012) definem em seu trabalho a abordagem KROSA (*Knowledge Reuse-Oriented Safety Analysis*) que utiliza padrões pré-definidos para analisar ameaças à segurança em especificações de requisitos, os autores apoiam seu *framework* com sua ferramenta para automatizar a gerência de segurança.

Varela João Araújo (2011) descreve uma abordagem que gera automaticamente modelos de *feature* (características) a partir de modelos AORA (*Aspect-Oriented Re-*

quirements Analysis), a ferramenta PLAORA (*Product Lines for Aspect-Oriented*) foi desenvolvida para apoiar essa abordagem.

4.2.2 Ferramentas de Apoio ao Processo de Engenharia de Requisitos para Sistemas Autoadaptativos

Além das ferramentas de apoio à ER clássica, têm sido investigado abordagens de suporte à ER para SAs (ALI; DALPIAZ; GIORGINI, 2013; HUSSEIN et al., 2013; FREDERICKS; DEVRIES; CHENG, 2014).

Ali, Dalpiaz e Giorgini (2013) propõe dois mecanismos automatizados: mapeamento da especificação de requisitos em um modelo de metas para identificar inconsistências e requisitos conflitantes de acordo com a mudança de contexto. Esses mecanismos são acrescentados como funcionalidades para a ferramenta CASE RE-*Context*, já desenvolvida pelos autores em um trabalho anterior.

Do mesmo modo, Hussein et al. (2013) apresenta uma ferramenta de validação de requisitos automatizada baseada em cenários através do processamento da especificação das variantes dos requisitos levantados (características, estado, e assim por diante). A partir disso, a ferramenta gera automaticamente scripts de acordo com o cenário de entrada e os verifica. Após isso, as inconscistências encontradas são apresentadas. Também, a ferramenta pode gerar resultados em formato aceitável pela ferramenta Romeo para realizar as validações necessárias.

Fredericks, DeVries e Cheng (2014) relatam uma abordagem que automatiza a análise de modelos objetivos e utiliza como entrada modelos de metas construído em KAOS (*Keep All Objectives Satisfied*). A partir disso a ferramenta desenvolvida pelos autores gera possíveis soluções em modelos *Relax*, onde contém operadores da própria linguagem como também funções que caracterizam o objetivo definido no modelo de entrada.

4.3 Análise

Os trabalhos coletados relatam ferramentas que auxiliam o processo de ER para sistemas tradicionais e SAs. Essas ferramentas podem ser classificadas de acordo com a sua abordagem, quanto ao suporte a diagramação (BUTLER; TSSG, 2011; AZEVEDO et al., 2011), verificação e validação de modelo de requisitos (ALI; DALPIAZ; GIORGINI, 2010; ALI; DALPIAZ; GIORGINI, 2013; ALI; DALPIAZ; GIORGINI, 2014; HUSSEIN et al., 2013), metodologias desenvolvidas (UMOH; SAMPAIO; THEODOULIDIS, 2011; MK; S, 2009; BECKERS et al., 2013; DARAMOLA TOR STALHANE; OMORONYIA, 2011; DARAMOLA; STALHANE, 2012) e geração de modelos (VARELA JOãO ARAúJO, 2011;

FREDERICKS; DEVRIES; CHENG, 2014).

Dentre os trabalhos apresentados pode-se destacar as principais abordagens que contribuem diretamente com a presente pesquisa como a ferramenta *Auto-Relax* proposta por Fredericks, DeVries e Cheng (2014), pois, utiliza a linguagem *Relax* para a construção de um modelo de requisitos e a *RE-Context* proposta por Ali, Dalpiaz e Giorgini (2010), Ali, Dalpiaz e Giorgini (2013), Ali, Dalpiaz e Giorgini (2014) que também possui uma abordagem voltada a especificação de requisitos utilizando mecanismos para verificar erros nos requisitos.

4.4 Considerações do Capítulo

Através desse capítulo foi possível conhecer várias ferramentas que auxiliam o processo de ER, tanto para sistemas tradicionais quanto para SAs. Além de conhecer as funcionalidades dessas ferramentas, também foi entendido a metodologia da qual foram implementadas, ou seja, os conceitos e técnicas que a ferramenta utiliza para cumprir os seus objetivos.

A revisão sistemática aplicada resultou em diversos trabalhos que foram analizados mas não mencionados nesse capítulo, pois, não contemplavam diretamente as questões definidas no protocolo de pesquisa. Esses trabalhos não selecionados também contibuíram para o estudo, através deles foi possível conhecer modelos de metas utilizados em ER, técnicas de análise e projeto para SAs, temáticas pouco investigadas fornecidas pelas agendas de pesquisa e assim por diante.

5 Desenvolvimento da Ferramenta

Esse capítulo apresenta os artefatos obtidos nas atividades de desenvolvimento da ferramenta, a seção 5.1 trata sobre a análise realizada, nessa seção é apresentado o problema a ser resolvido, técnicas para o entendimento desse problema e os respectivos artefatos gerados. Após isso, a seção 5.2 apresenta o projeto da ferramenta descrevendo a solução proposta, os artefatos de projeto e a arquitetura do software. Já a seção 5.3 apresenta a etapa de implementação, nela será apresentado aspectos de codificação. A etapa de testes é descrita na seção 5.4, nela será abordado os testes realizados na ferramenta para comprovar a sua eficiência a nível de componente unitário até a integração desses componentes.

5.1 Análise

Conforme apresentado por Whittle et al. (2010) existe uma carência de ferramentas de suporte a criação e especificação de requisitos para SA, muitas vezes a preocupação está direcionada somente a arquitetura e projeto desses sistemas. O objetivo principal do nosso estudo é desenvolver um *plugin* para a IDE Eclipse que possibilite a especificação de requisitos em SA utilizando a linguagem de requisitos proposta por Whittle et al. (2010).

As atividades de análise são destinadas para o conhecimento do problema do cliente, nessa fase foram gerados artefatos como: *storyboard*, histórias de usuário, documento de casos de uso e diagrama de caso de uso.

5.1.1 Histórias de Usuário

Através do conhecimento da problemática abordada por Whittle et al. (2010) foi construído histórias de usuário para compreender o que o usuário (desenvolvedor) pode realizar (ações) na ferramenta, a partir dessa técnica também é possível visualizar o motivo dessas ações.

As histórias são utilizadas para descrever requisitos em processos de desenvolvimento ágil como por exemplo: *Extreme Programming* (XP), *Scrum*, entre outros. Nessa fase foi utilizado essa prática para simplificar o modelo de caso de uso (Apêndice D) e demonstrar os objetivos e motivos de cada funcionalidade. Na Tabela 3 é possível visualizar as histórias construídas.

Tabela 3 – Narrativas de requisitos.

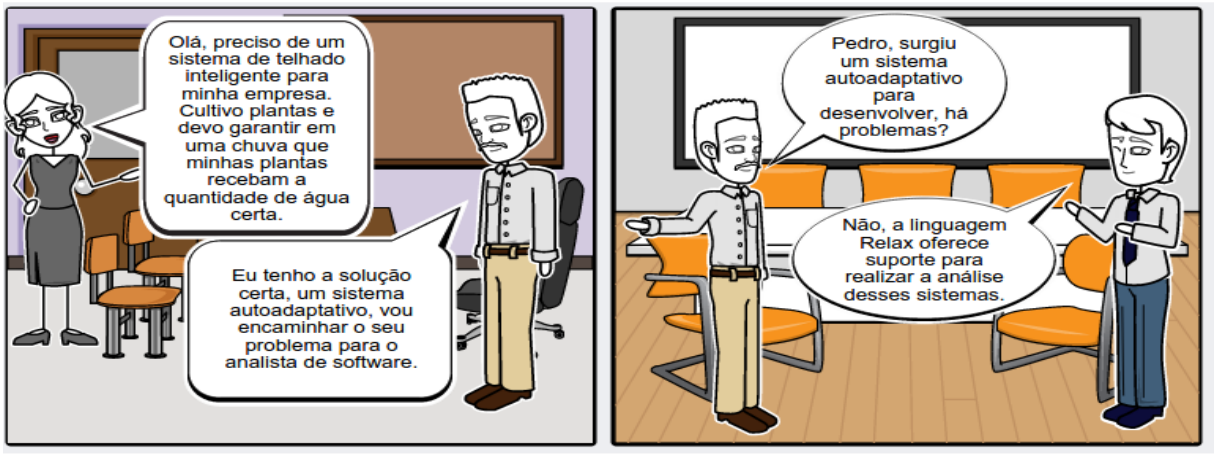
ID	Descrição
1	Como desenvolvedor, quero escrever requisitos utilizando a linguagem Relax, para que eu possa expressar os aspectos ambientais e de incerteza nos requisitos de Sistemas Autoadaptativos.
2	Como desenvolvedor, quero salvar minha especificação de requisitos, para que eu possa editá-la em outro momento.
3	Como desenvolvedor, quero converter minha especificação Relax em um modelo semântico, para que nas fases posteriores do projeto seja possível a compreensão dos requisitos por diferentes papéis da equipe de desenvolvimento como analistas, projetistas, codificadores, testadores e assim por diante.
4	Como desenvolvedor, quero verificar os erros sintáticos da minha especificação, para que eu possa construí-la e manté-la com mais qualidade.

5.1.2 *StoryBoard*

A técnica *StoryBoard*[1] foi utilizada para apresentar de maneira geral as funcionalidades da ferramenta, as narrativas do personagem descreverão os principais objetivos da ferramenta e suas possibilidades.

Na Figura 6, Figura 7, Figura 8 e Figura 9 é possível visualizar o contexto em que a ferramenta pode ser utilizada e suas principais funcionalidades como a escrita de requisitos e a coversão para modelo semântico.

Figura 6 – *Storyboard* - parte I



O *StoryBoard* inicia com a chegada de uma cliente solicitando um sistema autoadaptativo para sua problemática, após isso o atendente encaminha o pedido para o setor de analistas, o analista utiliza o RelaxEditor para especificar os requisitos em linguagem Relax.

[1] *StoryBoard* é uma técnica que utiliza a narração de cenários organizados em roteiro para demonstrar os objetivos (funcionalidades) principais do software a ser desenvolvido.

Figura 7 – *Storyboard* - parte II

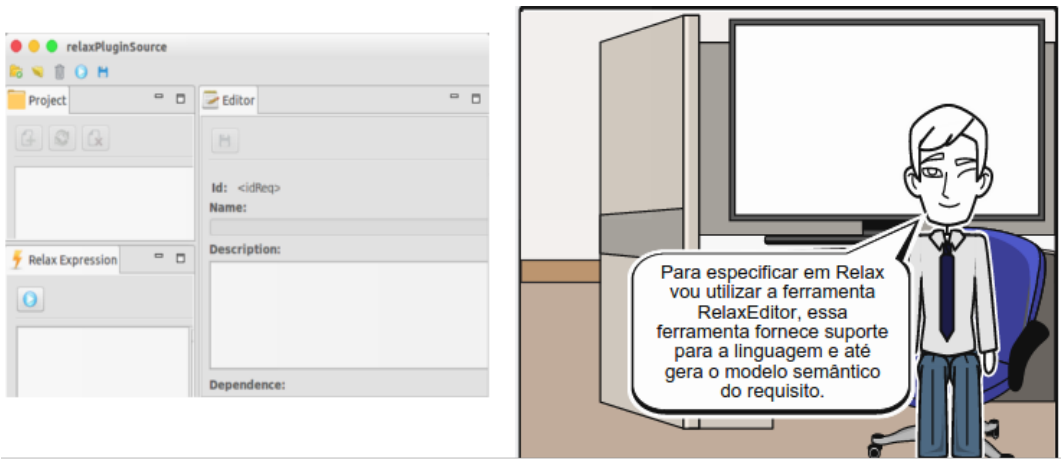


Figura 8 – *Storyboard* - parte III

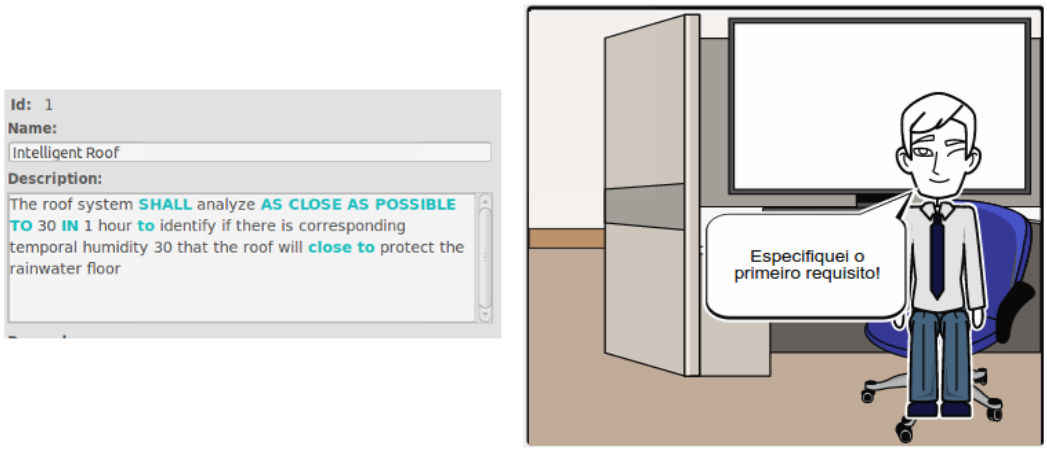
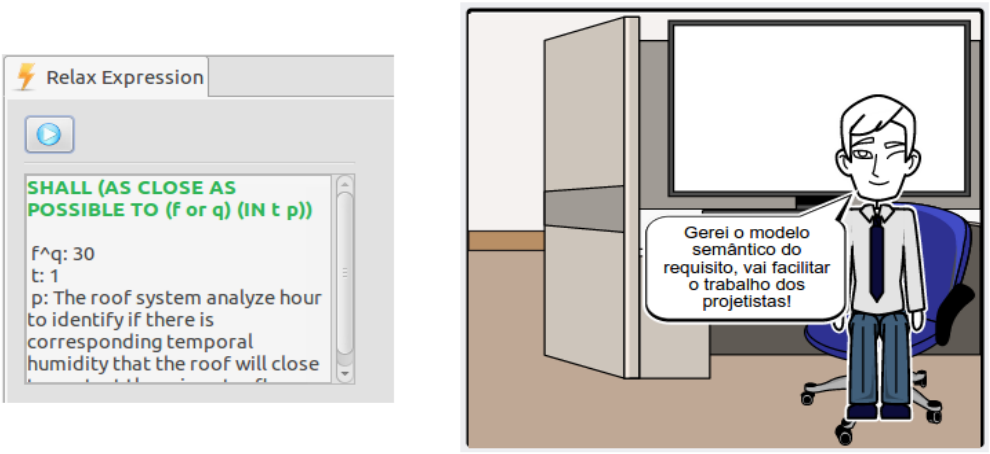


Figura 9 – *Storyboard* - parte IV

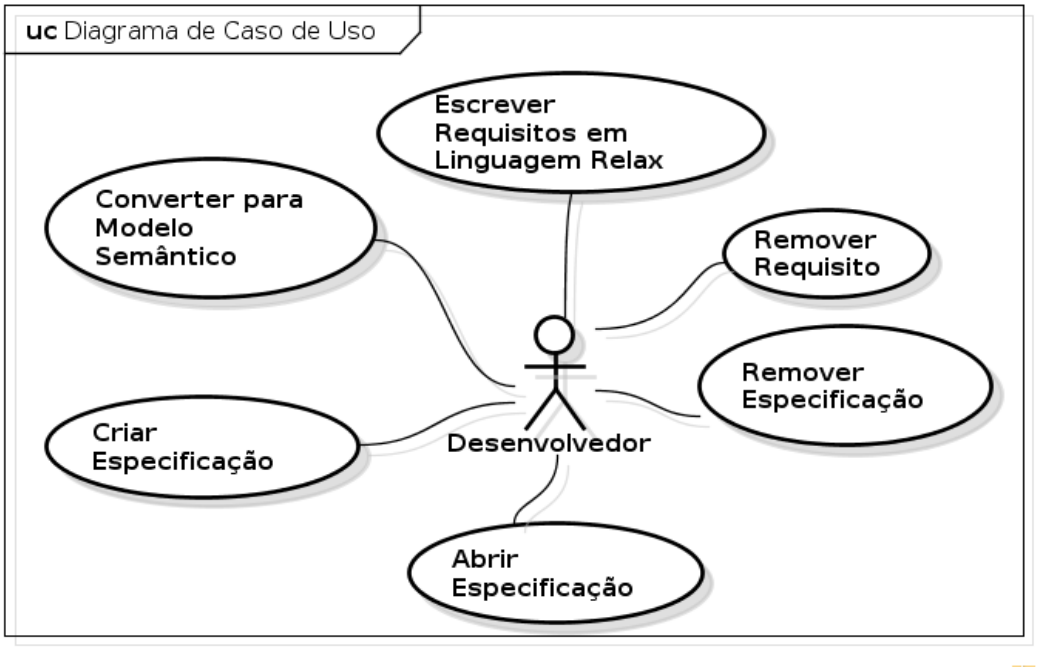


5.1.3 Modelo de Caso de Uso

A histórias de usuário foram fundamentais para a construção dos casos de uso do sistema (disponível no Apêndice D), os quais estão representados em forma de diagrama na Figura 10. A partir desse é possível visualizar as funcionalidades que a ferramenta

disponibiliza para a resolução da problemática abordada na etapa de análise.

Figura 10 – Diagrama de caso de uso.



Dentre os casos de uso apresentados na Figura 10, o caso de uso de maior prioridade é o escrever requisitos em linguagem Relax, essa funcionalidade do sistema foi a primeira a ser implementada, a partir dela o desenvolvedor pode escrever os requisitos em um editor, no qual os operadores reservados da linguagem são reconhecidos e destacados. Outra funcionalidade de prioridade alta foi a conversão do requisito especificado para um modelo semântico.

Além dessas funcionalidades, as operações básicas de usuário *Create-Read-Update-Delete* (CRUD) são fundamentais para a ferramenta, pois a mesma persiste dados de especificação de requisitos.

5.2 Projeto

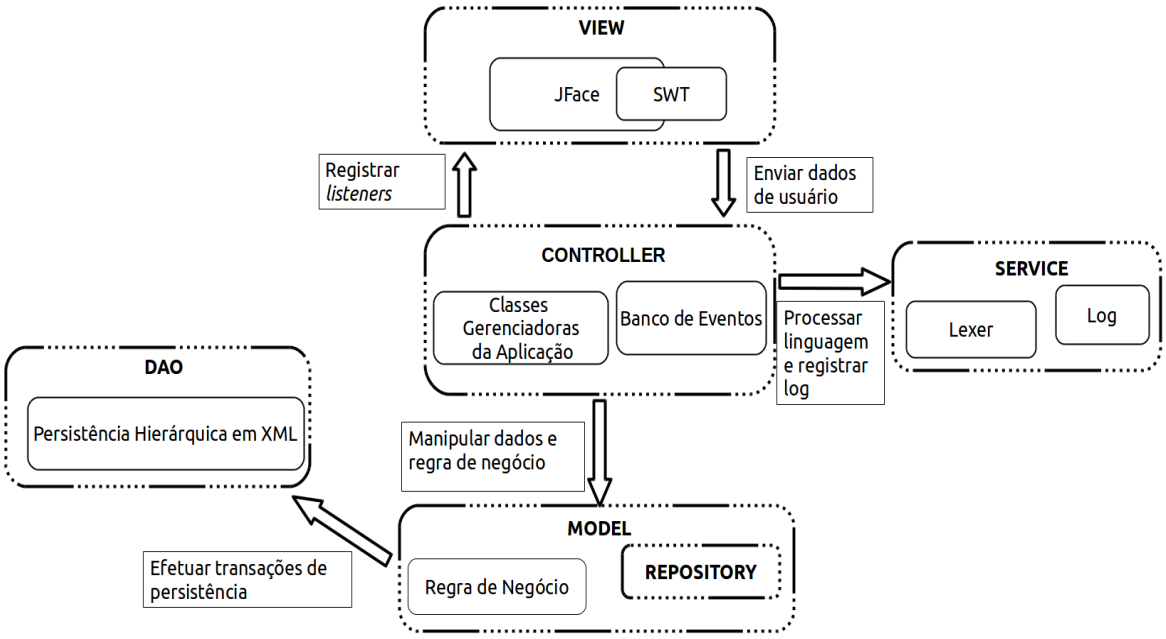
A atividade de projeto tem por objetivo propor a solução para o problema investigado na etapa de análise. Dentre os artefatos construídos, os principais são os diagramas de arquitetura, classes e de sequência.

Os artefatos de projeto devem respeitar as restrições impostas pela plataforma Eclipse, como também deve utilizar os recursos disponibilizados pela mesma.

5.2.1 Diagrama de Arquitetura

A arquitetura proposta utiliza duas camadas de suporte, a *Service* e a *Data Acess Object* (DAO) para complementar o modelo tradicional *Model-View-Controller* (MVC). A arquitetura é representada pela Figura 11.

Figura 11 – Arquitetura utilizada.



O módulo arquitetural *View* contém os componentes gráficos utilizados pela aplicação. Na plataforma Eclipse a interface gráfica utiliza em seus *plugins* as bibliotecas *JFace* e *SWT*. Soma-se a isso o módulo *Controller* que centraliza as estratégias da aplicação, esse módulo é responsável pelo gerenciamento das classes de negócio (*Model*), os serviços disponibilizados pela camada *Service* e os *listeners* disparados pela *View*. Quanto a camada DAO, ela é responsável pela infraestrutura de persistência da aplicação.

Além disso, o módulo *Model* é responsável por tratar as classes de domínio da aplicação. Esse módulo possui uma camada arquitetural interna chamada *Repository*, na qual são gerenciadas as coleções de objetos.

5.2.2 Diagrama de Classes

O diagrama de classes reflete a estrutura estática do software descrevendo as classes (atributos e métodos) e relacionamentos. O diagrama abordado apresenta a estrutura do módulo arquitetural *Model*. A Figura 12 e Figura 13 apresentam o diagrama de classes do software.

Figura 12 – Modelo de classes - parte 1.

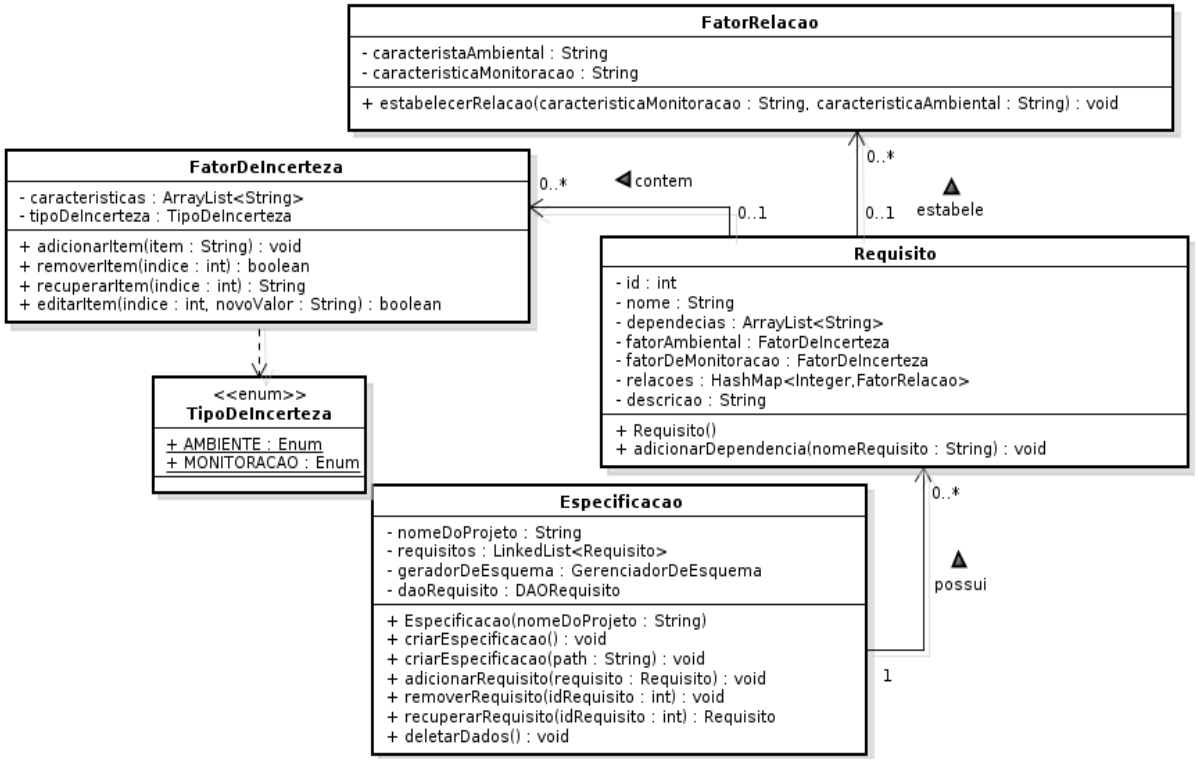
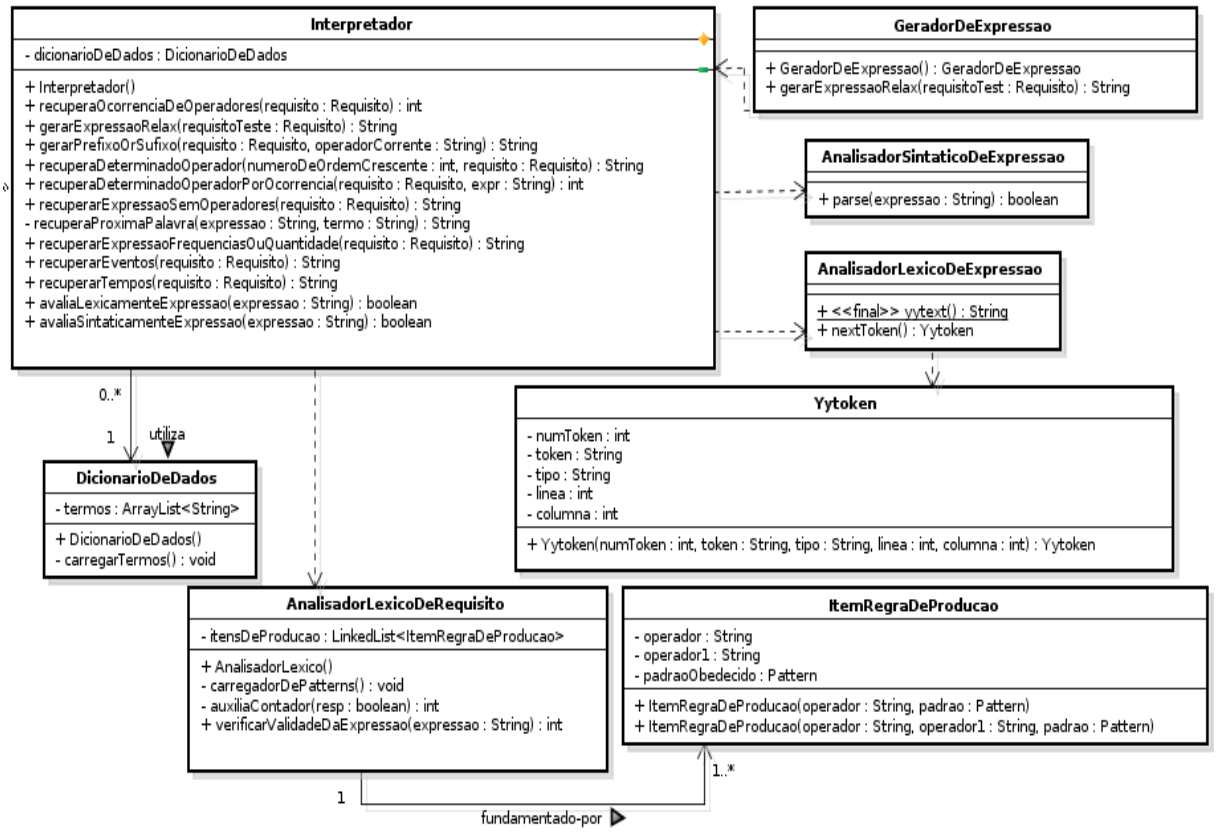


Figura 13 – Modelo de classes - parte 2.



A Figura 12 apresenta a classe especificação que possui uma lista de requisitos, cada requisito possui fatores de incerteza, os quais podem surgir do ambiente ou de fatores de monitoração. Também, um requisito possui relacionamentos entre esses fatores (ambiente e monitoração) e dependências.

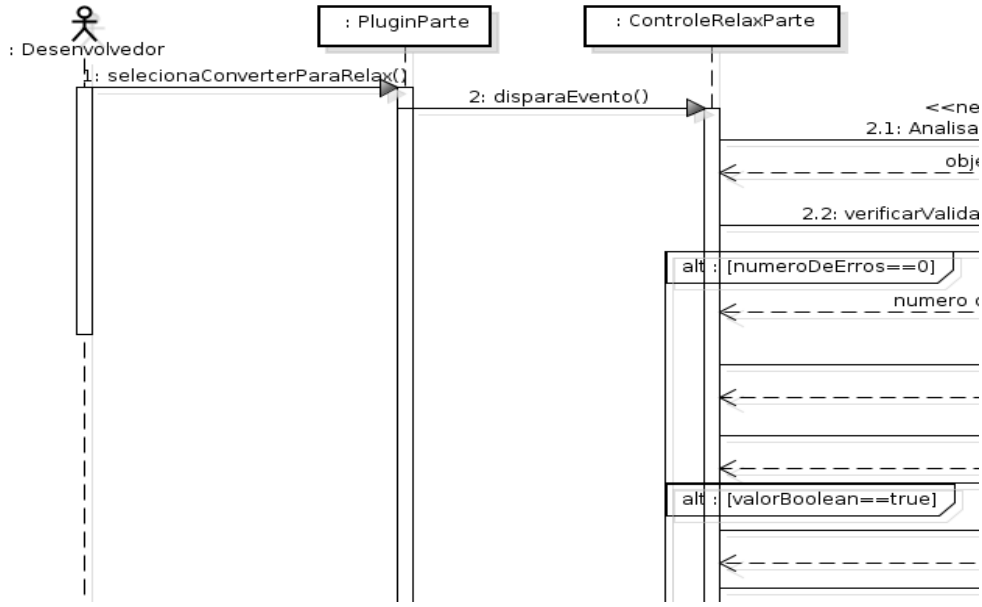
A Figura 13 possui as classe interpretador, ela é reponsável pela maior parte das operações relacionadas a interpretação da descrição de um requisito, o "GeradorDeExpressao"utiliza todos os recursos do "Intepretador"para poder gerar uma expressão semântica adequada para a descrição textual de entrada. Em parceria com o 'AnalisadorSintaticoDeExpressao', "AnalisadorLexicoDeExpressao", o "Interpretador"realiza operações como verificações léxicas e sintáticas.

5.2.3 Diagrama de Sequência

Os diagramas de sequência são utilizados para representar a ordem temporal das trocas de mensagem entre os objetos, esse modelo reflete aspectos dinâmicos do sistema, diferente do diagrama de classes. A Figura 14 e Figura 15 apresentam a sequência de mensagens que ocorrem para a conversão de uma descrição de requisito para um modelo semântico.

A Figura 14 apresenta o início da troca de mensagens, a primeira etapa é o evento disparado pelo usuário, o qual é capturado pela instância do "ControleRelaxParte", após isso o objeto controlador inicializa o "AnalisadorLexicoDeRequisito"para verificar a quantidade de erros contidos na expressão de requisito.

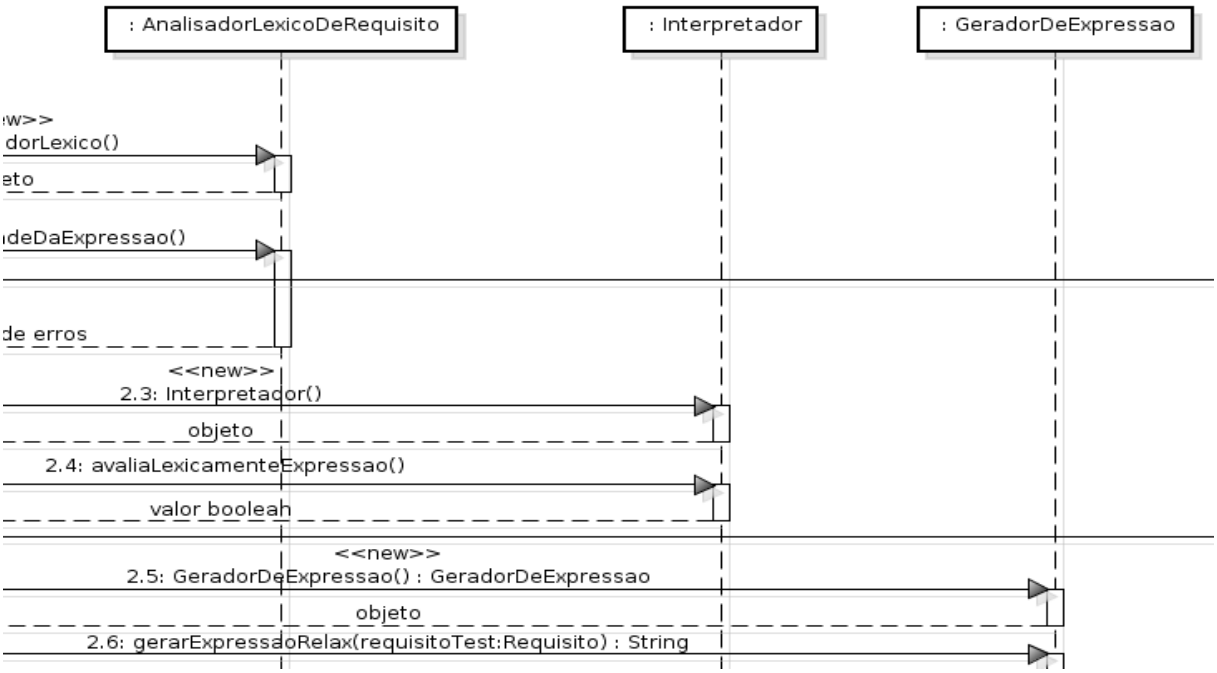
Figura 14 – Modelo de sequência - parte 1.



A Figura 15 apresenta a inicialização do "Interpretador", a partir dele é possível avaliar lexicamente a expressão, essa verificação é fundamental, caso o valor retornado

seja verdadeiro, o "GeradorDeExpressao"é inicializado para gerar a expressão semântica do requisito.

Figura 15 – Modelo de sequência - parte 2.



5.3 Implementação

A implementação foi realizada utilizando a linguagem de programação Java e os recursos disponibilizados pela plataforma Eclipse.

5.3.1 Interface Gráfica

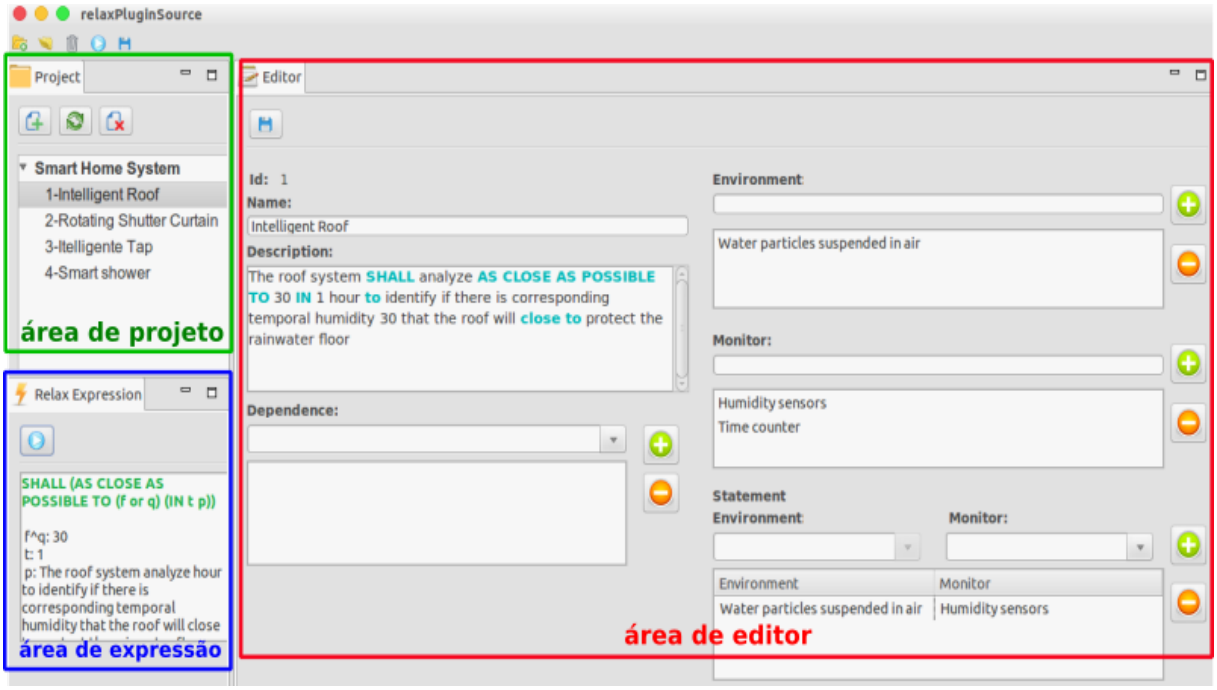
A interface gráfica foi desenvolvida a partir dos componentes disponibilizados pela *JFace* e os recursos *SWT*[2]. A Figura 16 apresenta a interface gráfica da tela principal.

A partir da Figura 16 é possível visualizar o recurso de abas utilizado, o qual possibilita que o usuário minimize ou maximize sua área. Além disso, a interface gráfica possui na aba de projetos um menu árvore, nele é organizado os requisito como itens.

Na área de editor da Figura 16 é possível visualizar a aba editor, ela possui um recurso *highlighter text* para destacar as palavras reservadas da linguagem. O editor conta também com outros campos para que o usuário possa especificar os aspectos de ambiente, monitoração, relação e dependência.

[2] *SWT* é uma biblioteca que disponibiliza componentes gráficos de interface.

Figura 16 – Interface gráfica - tela principal



Além da aba *Project* (apresentada na área de projeto) e *Editor*, o *plugin* conta com a aba *Relax Expression* (apresentada na área de expressão) nela é apresentado o resultado do processo de geração da expressão semântica do requisito.

5.3.2 Processamento de Expressões

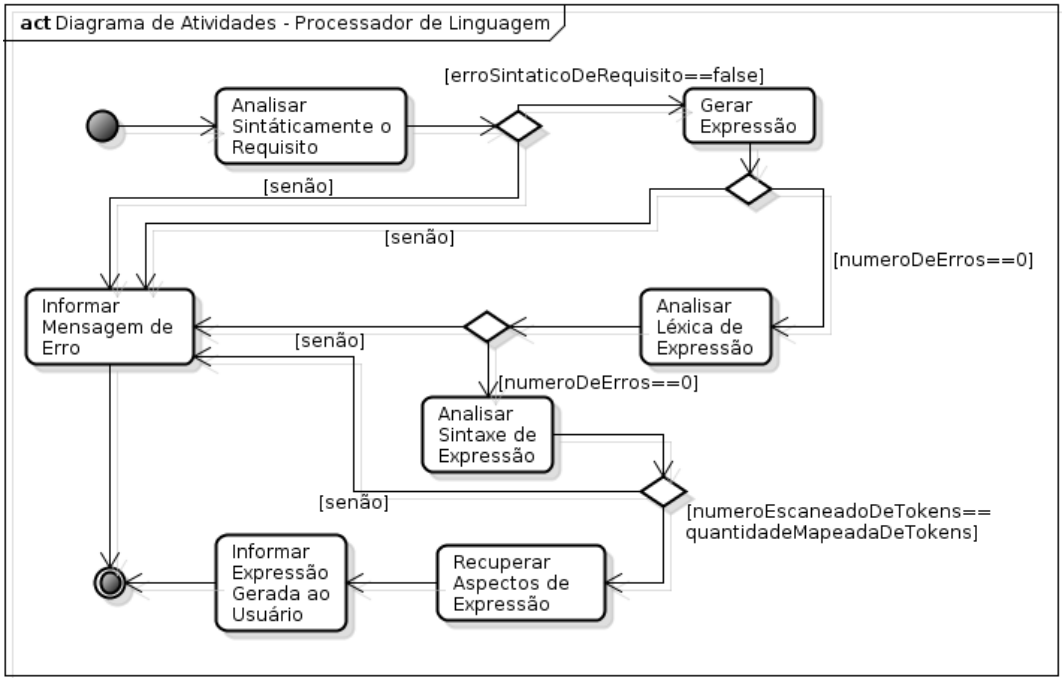
O processamento de expressão pode ser compreendido através do diagrama de atividades apresentado na Figura 17.

A primeira atividade é efetuar a análise sintática da expressão do requisito, caso a verificação resultar em verdadeiro a expressão Relax é gerada. A partir dessa expressão é avaliado sua léxica, posteriormente a isso sua sintaxe. A última atividade é recuperar os aspectos de expressão, esses aspectos representam tempo, frequência, quantidade ou eventos.

As verificações léxicas foram apoiadas pela utilização do gerador de analisadores léxicos JFlex, a especificação criada pode ser visualizada no Apêndice A. A partir dessa especificação o JFlex gera uma classe para auxiliar o processo de análise léxica de uma expressão, nessa especificação é descrito os *tokens* a serem interpretados pela classe *Scanner* gerada.

Quanto as verificações de sintaxe, as mesmas foram realizadas por meio de classes analisadoras geradas pelo *CUP*, o Apêndice B apresenta a especificação CUP, nela deve ser descrito as regras de produção possibilitadas pela gramática da linguagem e deve ser

Figura 17 – Diagrama de atividades - Processador de Linguagem



definido quais são os símbolos terminais, não terminais e de precedência. Vale salientar que, a ferramenta não realiza a análise semântica das expressões geradas.

5.3.3 Persistência

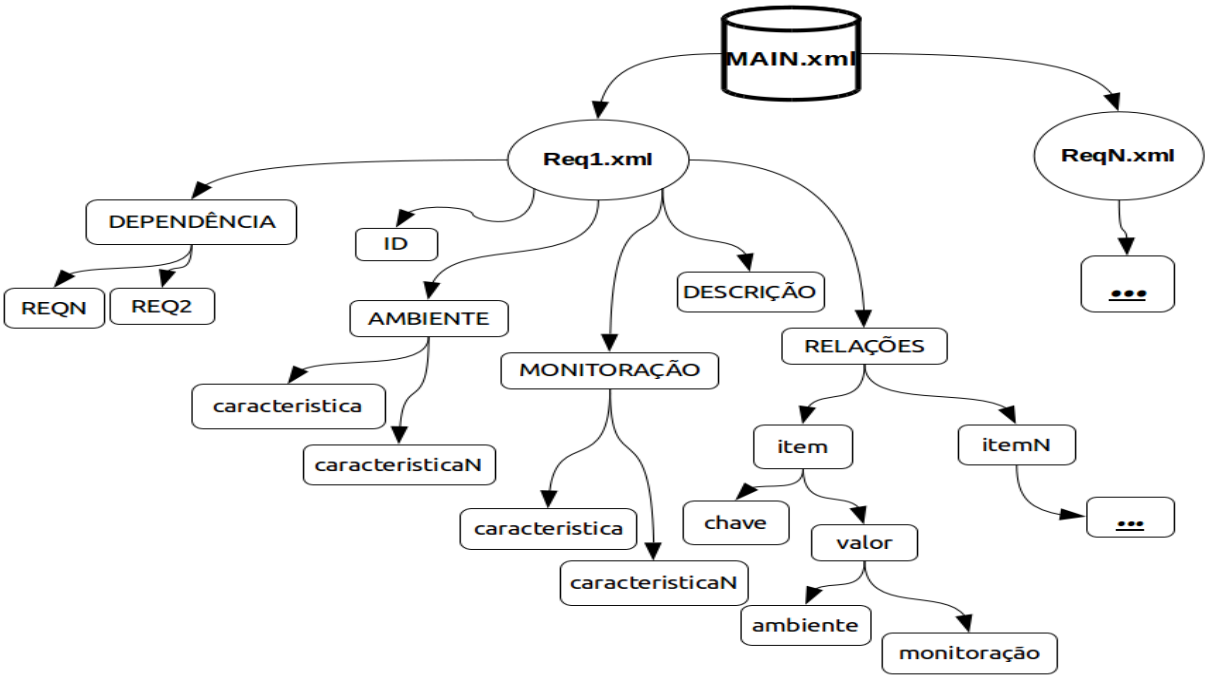
A persistência da ferramenta foi implementada utilizando uma hierarquia de arquivos XML, esses arquivos são conectados por um arquivo principal, cada arquivo (exceto o principal) representa um requisito, todos os arquivos juntos representam uma especificação de requisitos. Na Figura 18 é apresentado uma ilustração da hierarquia de persistência adotada.

A persistência adotada facilita a leitura das informações, já que os nós são indexados e referenciados no arquivo principal. As *tags* do arquivo requisito são id, ambiente, monitoração, descrição, relações, e dependência. As *tags* ambiente e monitoração representam uma lista de elementos, já na *tag* relações é representado uma estrutura de dados de tipo mapa, cada elemento contém uma chave e um valor, o valor representa dois itens, um derivado do ambiente e outro da monitoração. A dependência é também é uma lista de elementos, seu diferencial é que cada item dessa lista aponta para um arquivo requisito.

5.4 Testes

Na etapa de testes foram realizadas atividades de particionamento por equivalência, testes unitários e estruturais. O particionamento por equivalência divide em grupos os dados de entrada dos testes unitários de acordo com pontos de vista, a partir disso

Figura 18 – Persistência do *plugin* - hierarquia de arquivos XML.



é possível visualizar o escopo de teste. Essa técnica auxilia a implementação dos testes unitários (caixa preta), os quais tem por objetivo analisar as saídas geradas, levando em consideração apenas as suas entradas.

Já a análise de cobertura é uma técnica de teste estrutural (caixa-branca) que tem por objetivo analisar a estrutura do código, através dela é possível visualizar quais estruturas ou comandos foram executados, a partir disso é possível determinar a porcentagem de eficiência dos testes unitários implementados.

5.4.1 Particionamento por Equivalência

A Tabela 4 apresenta as classes de equivalência e o número de casos de teste escritos para cada classe, ao total foram escritos 106 casos de teste, todos os testes executados foram aprovados, retornando o resultado esperado.

5.4.2 Análise de Cobertura

O *plugin* EclEmma 2.3.2 (disponível para a plataforma Eclipse) foi utilizado para analisar a cobertura dos testes unitários construídos, esse *plugin* é executado juntamente com os testes unitários e verifica qual a porcentagem de cobertura das classes analisadas, sua perspectiva apresenta as estruturas exercitadas, não exercitadas e exercitadas apenas uma única vez.

A cobertura dos testes pode ser visualizada na Tabela 5, os testes realizados para

Tabela 4 – Particionamento por equivalência.

Classe de Equivalência	**Testes Escritos**
Operadores com frequência ou quantidade escrita numérica	9
Operadores com frequência ou quantidade de escrita somente texto	3
Operadores com tempo utilizando limitador ":"	2
Operadores com tempo utilizando nenhum limitador	4
Operadores com tempo utilizando o valor por extenso	2
Operador *BEFORE* com evento em verbo	3
Operador *AFTER* com evento em verbo	3
Operador *BEFORE* ou *AFTER* com mais de um evento associado	3
Preposições sem operadores associados	9
Expressão com operadores em *UpperCase*	3
Expressão com operadores em *MixedCase*	3
Expressão com operadores em *LowerCase*	29
Expressão sem operadores	5
Expressão com um ou mais operadores	28

a classe "AnalisadorLexicoDeRequisito"corresponderam em 90,2% de cobertura, já para a classe "GeradorDeExpressao", os testes cobriram cerca de 94,2% de código, quanto a classe "Interpretador"cobriu cerca de 88,7% de cobertura.

Tabela 5 – Cobertura dos testes.

Classe	**Estruturas Cobertas**	**Estruturas Não Cobertas**
AnalisadorLexicoDeRequisito	90.2%	9,8%
GeradorDeExpressao	94,2%	5,8%
Interpretador	88,7%	11,3%

5.5 Considerações do Capítulo

Esse capítulo apresentou os artefatos desenvolvidos nas atividades de análise, projeto, implementação e testes. Cada atividade realizada contribuiu diretamente com o produto final, as atividades de análise possibilitaram o conhecimento da linguagem Relax e dos principais problemas que ocorrem nas atividades de análise de SAs. Já na fase de projeto foi construído modelos estáticos e dinâmicos para propor soluções para a problemática.

A fase de implementação possibilitou a construção da ferramenta, através dela os artefatos de projeto foram refinados de acordo com novas necessidades de desenvolvimento ou restrições impostas pela plataforma Eclipse. Soma-se a isso as atividades de teste, a cada bateria de testes a implementação era revisitada com o objetivo de garantir a robustez da ferramenta, os casos de teste foram criados racionalmente utilizando a técnica

de particionamento por equivalência, após a construção dos testes unitários foi possível analisar a cobertura dos mesmos, pois em algumas baterias não exercitavam regiões importantes do código.

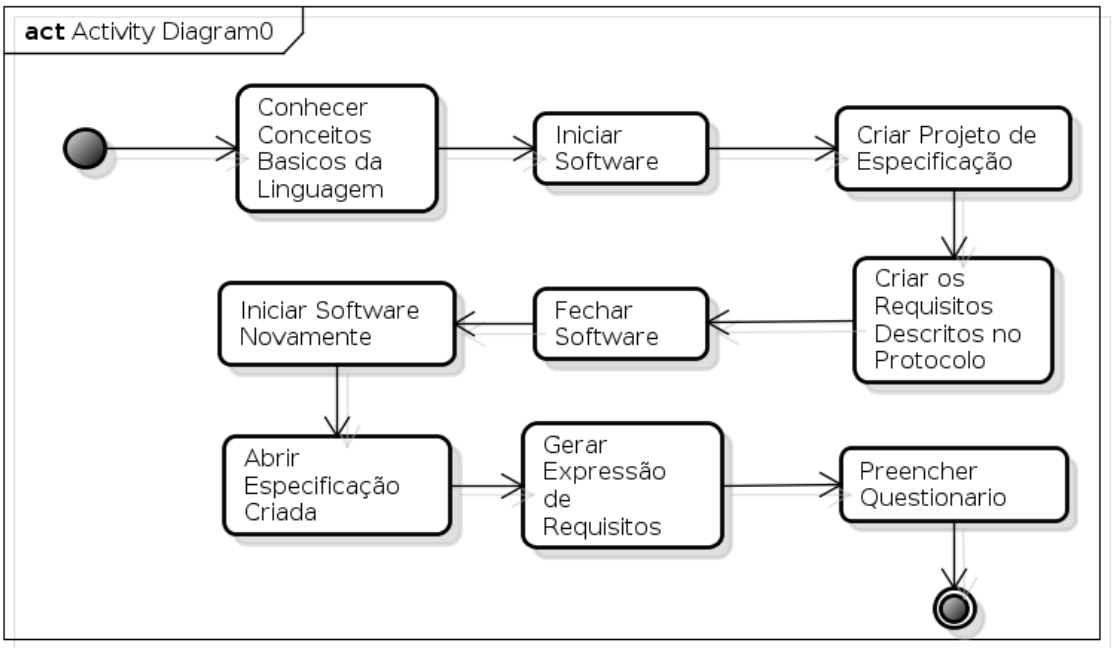
6 Experimento em Ambiente Controlado

Nesse capítulo será apresentado o experimento realizado com usuários, a seção 6.1 apresenta o protocolo definido para orientar a execução do experimento, esse protocolo é organizado em atividades. Após isso na seção 6.2 é apresentado os resultados obtidos no experimento, como também a interpretação desses.

6.1 Protocolo

A execução do experimento é organizada pelo protocolo de teste, o qual é dividido em etapas para facilitar a compreensão das tarefas a serem executadas pelos participantes. A Figura 19 apresenta o fluxo de atividades do protocolo.

Figura 19 – Protocolo executado no experimento



A primeira etapa do protocolo tem por objetivo apresentar os conceitos básicos da linguagem Relax, após isso o participante pode iniciar o software para realizar um exemplo de especificação de requisito, tal exercício já instrui o participante para realizar a etapa "Criar Projeto de Especificação". A próxima etapa consiste na criação de requisitos, os quais estão definidos no protocolo (tabela dos requisitos utilizados no Apêndice C). Na sequência das atividades o usuário sai do *plugin* e o inicia novamente para realizar a etapa de "Abrir Especificação Criada"e posteriormente a isso gera a expressão Relax de cada requisito da especificação.

O protocolo encerra com o preenchimento de um questionário (disponível no Apêndice E), tal técnica foi utilizada para coletar as percepções dos usuários sobre os atributos de qualidade: funcionalidade, confiabilidade, usabilidade e eficiência. A construção do questionário foi embasada na ISO/IEC 9126-1: 2003.

6.2 Resultados Obtidos

O experimento foi realizado com objetivo de avaliar a aceitação dos usuários (desenvolvedores de software) na utilização da ferramenta RelaxEditor para especificar requisitos de um SA. A aplicação do experimento teve 15 participantes, os quais são acadêmicos dos cursos de Ciência da Computação e Engenharia de Software.

Além disso, o experimento foi planejado e controlado, através do planejamento foi possível estipular o tempo necessário para a execução das atividades, criar exemplos para auxiliar o testador na realização das tarefas e organizar as atividades em um protocolo. O protocolo define os conceitos básicos necessários para a compreensão do papel da ferramenta, exemplos de operações realizadas na ferramenta e a descrição passo-a-passo das atividades a serem executadas no experimento.

A execução do experimento foi controlada, os testadores realizaram todas as tarefas em aproximadamente 60 minutos, as atividades eram observadas afim de evitar qualquer tipo de interferência externa do ambiente ou interna dos próprios testadores. O material disponibilizado aos usuários possuia um manual do protocolo e o executável do software.

Os resultados coletados do questionário possibilitaram uma visão geral da aceitação dos usuários nos aspectos de tempo de resposta, utilização de recursos da máquina, inteligibilidade, confiabilidade, adequação, acurácia, apreensibilidade e operacionalidade. Tais aspectos são derivados dos atributos de qualidade descritos na norma ISO/IEC 9126-1: 2003 e foram escolhidos levando em consideração fatores como: o problema abordado pela ferramenta e as principais características que a ferramenta deve atender para que seja utilizável pelos usuários.

Na Tabela 6 é possível visualizar os resultados obtidos para cada atributo de qualidade analisado, essa tabela foi baseada nos resultados derivados da escala *likert*[1] de cada resposta do questionário.

A tabela apresenta que a ferramenta possui na maioria dos atributos avaliados a nota máxima 5, outro aspecto a ser considerado é que a nota mínima atribuída para o tempo de resposta, operacionalidade, confiabilidade e inteligibilidade se concentrou abaixo de 3.

[1] A escala *likert* é muito utilizada em questionários, essa escala utiliza notas de 1 a 5 para medir a satisfação do usuário sobre alguma afirmativa.

Tabela 6 – Tabela de resultados obtidos no experimento (Notas de 1 a 5).

Atributo de Qualidade	**Mínima**	**Máxima**	**Média**	**Moda**
Utilização de Recursos da Máquina	3	5	4,07	5
Tempo de Resposta	1	5	4,07	5
Operacionalidade	2	5	3,64	4
Apreensibilidade	3	5	4,28	4
Acurácia	3	5	4,42	5
Adequação	3	5	5	5
Confiabilidade	1	5	3,42	3
Inteligibilidade	2	5	3,92	4

Além disso, a média mais alta de nota foi para o atributo de adequação, nisso é possível visualizar que a ferramenta é adequada para o problema alvo. A média mais baixa foi atribuída ao atributo de confiabilidade, o motivo dessa baixa está associado aos problemas de executável da ferramenta, os quais são originados da configuração da plataforma de sistema operacional que será executado o *plugin*. Visto que, o executável gerado para o experimento necessitava de informações como: arquitetura da máquina, versão de sistema operacional e permissão de executável no arquivo do *plugin* (somente para ambientes linux).

Outro medida utilizada na análise foi a moda, a nota 5 foi a que mais se repetiu para os atributos de desempenho, tempo de resposta, acurácia e adequação. Quanto aos demais atributos, foi repetido pela maioria dos usuários as notas 3 e 4.

Os resultados coletados demonstram que a ferramenta possui um conjunto de características de software consolidadas, nesse sentido a mesma possui indicativos de qualidade. Vale salientar que, os resultados definem também pontos a serem aperfeiçoados, como por exemplo o aspecto de confiabilidade, o qual obteve menor nota pela maioria dos usuários quando comparado com os demais.

6.3 Considerações do Capítulo

Ao decorrer desse capítulo foi descrito o experimento realizado e os resultados obtidos pelo mesmo. Esse experimento reforça a avaliação da ferramenta, além dos testes unitários, de integração e estruturais, o teste de aceitação com o usuário é indispensável, pois, esse possibilita verificar se a ferramenta está de acordo com o problema abordado, possui tolerância à falhas, desempenho aceitável, entre outros aspectos que somente o usuário pode determinar.

Os atributos de qualidade analisados possibilitam a visualização do estado das características de produto da ferramenta, os resultados adquiridos nesse experimento são

fundamentais para o aperfeiçoamento da ferramenta, consequentemente a isso, a satisfação do usuário final.

7 Conclusões

O principal objetivo do trabalho é o desenvolvimento de uma ferramenta de apoio a especificação de requisitos, esse objetivo foi atentido pela ferramenta produzida. A ferramenta é disponibilizada em formato de *plugin* para a IDEs Eclipse e oferece suporte a linguagem Relax proposta por Whittle et al. (2010). Também, a ferramenta fornece mecanismos funcionais simples para que o Engenheiro de Software possa utilizar os recursos que a linguagem Relax oferece.

Ao decorrer do desenvolvimento da ferramenta foram encontrados desafios gerados pela manipulação dos componentes oferecidos pela IDEs Eclipse, outro fator que demandou tempo foi a falta de documentação sobre alguns erros disparados pelo próprio ambiente. Soma-se a isso, a utilização de geradores de analisadores léxicos e sintáticos foi outro desafio, visto que para sua utilização é necessário conhecimentos sobre o funcionamento interno de um compilador.

No presente trabalho foi possível compreender a importância do processo de ER para SAs, a relevância da linguagem Relax criada por Whittle et al. (2010) para a especificação de requisitos de SA viabilizaram a construção desse trabalho. Além disso, o contato com o desenvolvimento de *plugins* para a plataforma Eclipse e a utilização do programa JFlex para a criação de geradores de analisadores léxicos e sintáticos possibilitou o aprendizado técnico dos recursos oferecidos por essas tecnologias.

O resultado gerado pelo trabalho foi a ferramenta (*plugin*) RelaxEditor, através dessa ferramenta o desenvolvedor de software pode especificar requisitos utilizando a linguagem Relax. A ferramenta disponibiliza um editor adequado para a escrita de requisitos, nela os operadores da linguagem são destacados, os fatores associados aos requisitos podem ser definidos e a expressão Relax do requisito pode ser gerada. Essa ferramenta pode auxiliar o Engenheiro de Software na etapa de análise de qualquer sistema (autoadaptativo ou não), por oferecer suporte a linguagem Relax, a ferramenta pode proporcionar ao Engenheiro de Software a eliminação da redundância nos requisitos e das inconcistências contidas nas especificações.

7.1 Trabalhos Futuros

Até então a ferramenta gera a expressão Relax do requisito, o próximo passo é a geração de um modelo Relax do requisito, nele será representado graficamente: os operadores utilizados pelos requisitos, suas dependências, fatores do ambiente, aspectos de monitoração e relacionamentos (ambiente e monitoração). Tal modelo será o artefato

conceitual utilizado para a visualização do problema na etapa de análise, posteriormente se tornará o artefato base da etapa de projeto.

Referências

AHO, A.; SETHI, R.; LAM, S. *Compiladores: princípios, técnicas e ferramentas*. Pearson Addison Wesley, 2008. ISBN 9788588639249. Disponível em: <https://books.google.com.br/books?id=hahXPgAACAAJ>. Citado 3 vezes nas páginas 11, 45 e 46.

ALI, R.; DALPIAZ, F.; GIORGINI, P. Reasoning about contextual requirements for mobile information systems: a goal-based approach. University of Trento, 2010. Citado 3 vezes nas páginas 56, 57 e 58.

ALI, R.; DALPIAZ, F.; GIORGINI, P. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology*, Elsevier, v. 55, n. 1, p. 35–57, 2013. Citado 2 vezes nas páginas 57 e 58.

ALI, R.; DALPIAZ, F.; GIORGINI, P. Requirements-driven deployment. *Software & Systems Modeling*, Springer, v. 13, n. 1, p. 433–456, 2014. Citado 3 vezes nas páginas 56, 57 e 58.

ANDRÉ, F. et al. Architectures & infrastructure. In: *Service research challenges and solutions for the future internet*. [S.l.]: Springer, 2010. p. 85–116. Citado na página 35.

AZEVEDO, D. et al. An integrative approach to diagram-based collaborative brainstorming: a case study. 2011. Citado 2 vezes nas páginas 56 e 57.

BALSIGER, M. A quick-start tutorial to eclipse plug-in development. Universität Bern, 2010. Citado na página 52.

BECKERS, K. et al. Structured pattern-based security requirements elicitation for clouds. In: IEEE. *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. [S.l.], 2013. p. 465–474. Citado 2 vezes nas páginas 56 e 57.

BENCOMO, N. et al. Self-explanation in adaptive systems. In: IEEE. *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*. [S.l.], 2012. p. 157–166. Citado 4 vezes nas páginas 21, 22, 35 e 36.

BENCOMO, N. et al. Requirements reflection: requirements as runtime entities. In: ACM. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. [S.l.], 2010. p. 199–202. Citado na página 22.

BILOBROVEC, R. F. M. M. M.; KOVALESKI, J. L. Implementação de um sistema de controle inteligente utilizando a lógica fuzzy. *XI SIMPEP, Bauru/Brasil*, 2004. Citado na página 42.

BRUN, Y. et al. Engineering self-adaptive systems through feedback loops. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 48–70. Citado 3 vezes nas páginas 34, 37 e 38.

BUTLER, B.; TSSG, A. B. Secure and trustworthy composite services. 2011. Citado 2 vezes nas páginas 56 e 57.

CHENG, B. H. et al. Software engineering for self-adaptive systems: A research roadmap. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 1–26. Citado na página 21.

DARAMOLA, G. S. O.; STALHANE, T. Pattern-based security requirements specification using ontologies and boilerplates. 2012. Citado 2 vezes nas páginas 56 e 57.

DARAMOLA TOR STALHANE, G. S. O.; OMORONYIA, I. Enabling hazard identification from requirements and reuse-oriented hazop analysis. 2011. Citado 2 vezes nas páginas 56 e 57.

DEITEL, H. *Java: Como programar*. PRENTICE HALL BRASIL, 2010. ISBN 9788576050193. Disponível em: <https://books.google.com.br/books?id=U5AyAgAACAAJ>. Citado na página 45.

DESRIVIERES, J.; WIEGAND, J. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, IBM, v. 43, n. 2, p. 371–383, 2004. Citado 3 vezes nas páginas 11, 49 e 50.

ECLIPSE FOUNDATION. Eclipse documentation - current release. 2014. Citado 2 vezes nas páginas 49 e 52.

FREDERICKS, E. M.; DEVRIES, B.; CHENG, B. H. Autorelax: automatically relaxing a goal model to address uncertainty. *Empirical Software Engineering*, Springer, p. 1–36, 2014. Citado 2 vezes nas páginas 57 e 58.

HERCZEG, M. The smart, the intelligent and the wise: roles and values of interactive technologies. In: ACM. *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia*. [S.l.], 2010. p. 17–26. Citado na página 37.

HUDSON, S. E. *CUP User's Manual*. [s.n.], 1999. Disponível em: <http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>. Citado na página 48.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, ACM, v. 40, n. 3, p. 7, 2008. Citado na página 36.

HUSSEIN, M. et al. Scenario-based validation of requirements for context-aware adaptive services. In: IEEE. *Web Services (ICWS), 2013 IEEE 20th International Conference on*. [S.l.], 2013. p. 348–355. Citado na página 57.

ITO, M. et al. Support tool to the validation process of functional requirements. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, IEEE, v. 9, n. 5, p. 889–894, 2011. Citado na página 56.

JEROME, H. S.; KAASHOEK, M. F. *Principles of computer system design: an introduction*. [S.l.]: Morgan Kaufmann, 2009. Citado na página 34.

JUNIOR, O. A. de A. Engenharia de requisitos para sistemas auto-adaptativos. 2013. Citado 2 vezes nas páginas 34 e 38.

KLEIN, S. R. G.; DéCAMPS, R. *JFlex User's Manual*. [s.n.], 2015. Disponível em: <http://jflex.de/>. Citado 2 vezes nas páginas 46 e 47.

MK, Z. M. Z.; S, S. S. S. S. S. Supporting collaborative requirements elicitation using focus group discussion technique. *International Journal of Software Engineering and Its Applications*, v. 3, n. 3, p. 59–70, 2009. Citado 2 vezes nas páginas 56 e 57.

MORAES, J. B. D. Análise de pontos de função. *Engenharia de Software Magazine*, p. 54, 2009. Citado 2 vezes nas páginas 29 e 30.

PFLEEGER, S.; ATLEE, J. *Software Engineering: Theory and Practice*. [S.l.]: Prentice Hall, 2004. Citado 4 vezes nas páginas 26, 27, 31 e 32.

PFLEEGER, S.; ATLEE, J. *Software Engineering: Theory and Practice*. Prentice Hall, 2010. ISBN 9780136061694. Disponível em: <http://books.google.ca/books?id=7zbSZ54JG1wC>. Citado na página 21.

PIMENTEL, J. H. C. Abordagens de engenharia de requisitos para computação autonômica. 2008. Citado na página 37.

PRESSMAN, R. *Engenharia de Software*. McGraw Hill Brasil, 2011. ISBN 9788580550443. Disponível em: <http://books.google.com.br/books?id=y0rH9wuXe68C>. Citado 6 vezes nas páginas 21, 25, 26, 27, 28 e 33.

QURESHI, N. A.; PERINI, A. Engineering adaptive requirements. In: IEEE. *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. [S.l.], 2009. p. 126–131. Citado na página 22.

QURESHI, N. A.; PERINI, A. Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications. In: IEEE. *Requirements@ Run. Time (RE@ RunTime), 2010 First International Workshop on*. [S.l.], 2010. p. 17–24. Citado na página 22.

QURESHI, N. A.; PERINI, A. Requirements engineering for adaptive service based applications. In: IEEE. *Requirements Engineering Conference (RE), 2010 18th IEEE International*. [S.l.], 2010. p. 108–111. Citado 5 vezes nas páginas 21, 22, 35, 36 e 37.

SAWYER, P. et al. Requirements-aware systems: A research agenda for re for self-adaptive systems. In: IEEE. *Requirements Engineering Conference (RE), 2010 18th IEEE International*. [S.l.], 2010. p. 95–103. Citado 3 vezes nas páginas 22, 34 e 38.

SOMMERVILLE, I. *Engenharia de Software*. Pearson Brasil, 2011. ISBN 9788579361081. Disponível em: <http://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado 9 vezes nas páginas 21, 25, 26, 27, 28, 30, 32, 33 e 34.

SURI, N.; CABRI, G. *Adaptive, Dynamic and Resilient Systems*. [S.l.]: CRC Press, 2014. Citado 2 vezes nas páginas 35 e 36.

UMOH, E.; SAMPAIO, P. R. F.; THEODOULIDIS, B. Refinto: An ontology-based requirements engineering framework for business-it alignment in financial services organizations. In: IEEE. *Services Computing (SCC), 2011 IEEE International Conference on*. [S.l.], 2011. p. 600–607. Citado 2 vezes nas páginas 56 e 57.

VARELA JOãO ARAúJO, I. B. e. A. M. P. Aspect-oriented analysis for software product lines requirements engineering. 2011. Citado 3 vezes nas páginas 56, 57 e 58.

WAZLAWICK, R. S. *Análise e Projeto de Sistemas da Informação*. [S.l.]: Elsevier Brasil, 2004. Citado 3 vezes nas páginas 27, 30 e 31.

WELSH, K.; SAWYER, P. Understanding the scope of uncertainty in dynamically adaptive systems. In: *Requirements Engineering: Foundation for Software Quality*. [S.l.]: Springer, 2010. p. 2–16. Citado na página 22.

WHITTLE, J. et al. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, Springer, v. 15, n. 2, p. 177–196, 2010. Citado 10 vezes nas páginas 7, 9, 21, 22, 34, 39, 40, 42, 59 e 77.

Apêndices

APÊNDICE A – Especificação JFlex

```
1  package br.unipampa.lesa.servico.lexer;
2
3  import java_cup.runtime.*;
4  import java.io.Reader;
5  %implements java_cup.runtime.Scanner;
6
7  %%
8  %class ScannerDeAnalisadorSintatico
9  %line
10 %column
11 %cup
12 %{
13   private Symbol symbol(int type)
14   {
15     return new Symbol(type, yyline, yycolumn);
16   }
17   private Symbol symbol(int type, Object value){
18     return new Symbol(type,yyline, yycolumn, value);
19   }
20   }%
21
22   ESPACO= " "
23   %%
24   <YYINITIAL> {
25     "SHALL" {return symbol(sym.SHALL);}
26     "MAY" {return symbol(sym.MAY);}
27     "OR" {return symbol(sym.OR);}
28     "EVENTUALLY" {return symbol(sym.EVENTUALLY);}
29     "UNTIL" {return symbol(sym.UNTIL);}
30     "BEFORE" {return symbol(sym.BEFORE);}
31     "e" {return symbol(sym.E);}
32     "AFTER" {return symbol(sym.AFTER);}
33     "IN" {return symbol(sym.IN);}
34     "t" {return symbol(sym.T);}
35     "CLOSE" {return symbol(sym.CLOSE);}
36     "POSSIBLE" {return symbol(sym.POSSIBLE);}
37     "EARLY" {return symbol(sym.EARLY);}
38     "LATE" {return symbol(sym.LATE);}
```

```
39      "FEW" {return symbol(sym.FEW);}
40      "p" {return symbol(sym.P);}
41      "true" {return symbol(sym.TRUE);}
42      "false" {return symbol(sym.FALSE);}
43      "AS" {return symbol(sym.AS);}
44      "TO" {return symbol(sym.TO);}
45      "MANY" {return symbol(sym.MANY);}
46      "forq" {return symbol(sym.FORQ);}
47
48    {ESPACO} {}
49  }
50  [^] {throw new Error("Caracter iLegal <"+yytex()+">");}
```

APÊNDICE B – Especificação CUP

```
1  package br.unipampa.lesa.servico.lexer;
2
3  import java_cup.runtime.*;
4  import java.io.FileReader;
5
6  parser code {:
7
8    public void report_error(String message, Object info)
9    {
10     StringBuilder m = new StringBuilder("Error");
11     if(info instanceof java_cup.runtime.Symbol)
12     {
13       java_cup.runtime.Symbol s = ((java_cup.runtime.Symbol) info
           );
14
15       if(s.left >=0)
16       {
17         m.append(" in line "+ (s.left+1));
18         if(s.right >=0)
19           m.append(", column "+ (s.right+1));
20       }
21     }
22
23     m.append(" : " + message);
24
25     System.err.println(m);
26   }
27   public void report_fatal_error(String message, Object info){
28     report_error(message,info);
29     System.exit(1);
30   }
31   public static void main(String[] args)
32   {
33     try{
34       AnalisadorSintatico asin = new AnalisadorSintatico(new
           AnalisadorLexico(new FileReader(args[0])));
35       Object result = asin.parse().value;
36       System.out.println("\n*** Resultados Finais ***");
```

```
37      }catch(Exception erro){
38        ex.printStackTrace();
39      }
40    }
41    :};
42
43    terminal SHALL, TRUE, FALSE, P, E, T, ForQ, MAY, OR, EVENTUALLY
          , UNTIL, BEFORE, AFTER, IN, EARLY, LATE, AS, POSSIBLE, TO,
          CLOSE, FEW, MANY;
44
45    non terminal EXP;
46
47    EXP ::= TRUE {: RESULT = new Boolean(true);:}
48    | FALSE {:RESULT = new Boolean(true);:}
49    | P {:RESULT = new Boolean(true);:}
50    | SHALL EXP {:RESULT = new Boolean(true);:}
51    | MAY EXP {:RESULT = new Boolean(true);:}
52    | OR MAY EXP {:RESULT = new Boolean(true);:}
53    | EVENTUALLY EXP {:RESULT = new Boolean(true);:}
54    | UNTIL EXP {:RESULT = new Boolean(true);:}
55    | BEFORE E EXP {:RESULT = new Boolean(true);:}
56    | AFTER E EXP {:RESULT = new Boolean(true);:}
57    | IN T EXP {:RESULT = new Boolean(true);:}
58    | AS CLOSE AS POSSIBLE TO ForQ EXP {:RESULT = new Boolean(true)
          ;:}
59    | AS EARLY AS POSSIBLE EXP {:RESULT = new Boolean(true);:}
60    | AS LATE AS POSSIBLE EXP {:RESULT = new Boolean(true);:}
61    | AS MANY AS POSSIBLE EXP {:RESULT = new Boolean(true);:}
62    | AS FEW AS POSSIBLE EXP {:RESULT = new Boolean(true);:}
63    ;
```

APÊNDICE C – Tabela de Requisitos Utilizada no Protocolo - Experimento da Ferramenta

Tabela 7 – Requisitos da Especificação Utilizada no Protocolo.

ID	Description	Env	Mon	Statement	Dep
1	The roof system SHALL analyze AS CLOSE AS POSSIBLE TO 30 IN 1 hour to identify if there is corresponding temporal humidity 30% that the roof will close to protect the rainwater floor	1. Water particles suspended in air	1. Humidity sensors, 2. Time Counter	1-1	-
2	The curtain system SHALL ensure that according to the light curtain fits IN 4 seconds, contemplating MAY open OR ajar closed OR closed	1. Sunlight, 2. Sleep user	1. Light sensor, 2. Weight sensor and extension in bed, 3. Research on history of user actions to close the curtain, 4. Time counter	1-1; 2-2,3	-
3	The system SHALL tap AS EARLY AS POSSIBLE ensure that the user's hands to find the water is released and according to the remoteness of the hands the water flow decreases IN 3 seconds	1. User hand washing	1. Approach sensor, 2. Time counter	1-1	-
4	The shower system SHALL AS EARLY AS POSSIBLE ensure that the find the user, the water is released and according to the remoteness of the same water flow decreases IN 12 seconds	1. Bathing user	1. Approach sensor, 2. Time counter, 3. Climate Research	1-2,3	-
5	The water drainage system SHALL ensure that after bathing or cleaning the bathroom area, the accumulated water is drained UNTIL the drying AS CLOSE AS POSSIBLE TO 30	1. Accumulated water	1. Approach sensor	1-1	4

APÊNDICE D – Documentação de Caso de Uso

Ator Principal: Desenvolvedor.

Tabela 8 – Requisitos Não Funcionais.

REQUISITOS NÃO FUNCIONAIS
ID 1- Sistema deverá oferecer interface gráfica intuitiva utilizando ícones com *tooltip* apresentando a funcionalidade proposta por seu evento.
ID 2- Sistema deverá conhecer a hierarquia de diretórios independente do sistema operacional utilizado pelo usuário.
ID 3- Sistema deverá registrar todas as suas ações, data e horário de sua ocorrência.
ID 4- Sistema deverá gerar especificações nos formatos: xml, pdf ou txt.
ID 5- Sistema deverá oferecer um manual adequado na aba "Help".
ID 6- Para a primeira utilização do ambiente o sistema deverá apresentar as possibilidades do mesmo.

Tabela 9 – CDU1: Criar Especificação.

Criar Projeto de Especificação
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
Fluxo Principal
1. Desenvolvedor cria novo projeto de especificação relax.
2. Sistema apresenta uma janela para a configuração do novo projeto.
3. Desenvolvedor informa o nome do projeto, diretório de destino e o seu nome (campo desejável).
4. Desenvolvedor dispara o evento para criar novo projeto.
5. Sistema cria um novo projeto de especificação relax.
6. Sistema gera os arquivos de configuração.
7. Sistema apresenta a pasta de projeto com a hierarquia de diretórios.
8. Sistema informa que o projeto foi criado com sucesso.
Requisitos Não Funcionais
RNF 1, RFN 2, RNF 6 e RNF 7

Tabela 10 – CDU2: Abrir Especificação.

Abrir Projeto de Especificação
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
Fluxo Principal
1. Desenvolvedor seleciona a opção específica para abrir projeto de especificação.
2. Sistema apresenta uma janela para busca de projeto.
3. Desenvolvedor busca diretório de projeto pelo buscador.
4. Desenvolvedor seleciona o diretório desejado.
5. Sistema direciona Desenvolvedor para a tela principal.
6. Sistema carrega a especificação buscada no menu de projetos.
Requisitos Não Funcionais
RNF 1, RNF 6 e RNF 7

Tabela 11 – CDU3: Remover Especificação.

Remover Projeto de Especificação
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
2. Especificação deve estar carregada no *Plugin*.
Fluxo Principal
1. Desenvolvedor seleciona a opção específica para remover o projeto de especificação.
2. Sistema apresenta uma janela interrogando se realmente é isso que deve ser realizado.
3. Desenvolvedor confirma ação.
4. Sistema remove o arquivo indexador do projeto.
5. Sistema informa que o projeto foi removido com sucesso.
Requisitos Não Funcionais
RNF 1, RNF 6 e RNF 7

Tabela 12 – CDU4: Escrever Requisitos em Linguagem Relax.

Escrever Requisitos em Linguagem Relax
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
2. Especificação deve estar carregada no *Plugin*.
3. Algum requisito deve estar selecionado.
Fluxo Principal
1. Desenvolvedor seleciona a opção responsável por criar novo requisito.
2. Sistema direciona usuário para a área do editor.
3. Desenvolvedor especifica requisito.
4. Sistema avalia constantemente em tempo de execução a utilização dos operadores para destacá-los como palavras-chave.
5. Desenvolvedor seleciona a opção salvar.
6. Sistema verifica os campos definidos pelo Desenvolvedor.
7. Sistema persiste os dados do requisito.
8. Sistema apresenta uma mensagem informando que a operação foi realizada com sucesso.
Fluxos Alternativos
1 a) Desenvolvedor seleciona a opção responsável por abrir determinado requisito.
Requisitos Não Funcionais
RNF 1, RNF 6 e RNF 7

Tabela 13 – CDU5: Converter para Modelo Semântico.

Converter para Modelo Semântico
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
2. Especificação deve estar carregada no *Plugin*.
3. Algum requisito deve estar selecionado.
Fluxo Principal
1. Desenvolvedor seleciona a opção responsável para a conversão da especificação em modelo semântico.
2. Sistema verifica a especificação relax do desenvolvedor.
3. Sistema verifica a especificação relax.
4. Sistema cria perfil sintático textual.
5. Sistema avalia sintáticamente a descrição.
6. Sistema avalia lexicamente a descrição.
5. Sistema apresenta na área de conversões a especificação resultante, descrita em modelo sintático.
Fluxos de Exceção
3 a) Sistema não interpreta a especificação relax do desenvolvedor.
b) Sistema trata os erros de processamento.
c) Sistema solicita que o Desenvolvedor corriga a descrição textual.
d) Sistema informa os erros encontrados.
Regras de Negócio
RN 2.
Requisitos Não Funcionais
RNF 1, RNF 3 e RNF 5

Tabela 14 – CDU6: Remover Requisito.

Remover Requisito
Pré-condição
1. O Plugin deverá estar instalado no IDE (Eclipse).
2. Especificação deve estar carregada no *Plugin*.
3. Algum requisito deve estar selecionado.
Fluxo Principal
1. Desenvolvedor seleciona a opção responsável por remover o requisito.
2. Sistema apresenta uma janela interrogando se realmente é isso que deve ser realizado.
3. Desenvolvedor confirma ação.
4. Sistema remove o requisito.
5. Sistema informa que o requisito foi removido com sucesso.
Requisitos Não Funcionais
RNF 1, RNF 6 e RNF 7

E Questionário Utilizado no Experimento

Questionário de Avaliação da Ferramenta RelaxEditor

Sua opinião é de extrema importância, através dessas informações será possível melhorar a ferramenta.

*Obrigatório

Testador:

Não é obrigatório, seu anonimato será preservado!

É muito fácil instalar a ferramenta. *

Você deve levar em consideração se ocorreu algum problema ao realizar a instalação da ferramenta.

	1	2	3	4	5	
Fácil demais	○	○	○	○	○	Difícil

O tempo de resposta ao efetuar as operações é muito rápido. *

Não foi possível medir o tempo de resta, mas empiricamente atribua uma nota.

	1	2	3	4	5	
Lento demais	○	○	○	○	○	Rápido

A ferramenta utiliza poucos recursos de máquina. *

Além de demorar, você notou alguma lentidão no processamento da máquina, o qual está associado com a execução da ferramenta.

	1	2	3	4	5	
Utiliza muitos recursos	○	○	○	○	○	Utiliza poucos recursos

É possível entender o modelo de interface proposto pela ferramenta. *

O modelo de interface gráfica é compreensível, a disposição dos elementos, ícones e elementos possibilitam sua compreensão.

	1	2	3	4	5	
Impossível	○	○	○	○	○	Em sua totalidade

É fácil aprender a utilizar a ferramenta. *

As dificuldades encontradas ao realizar as operações do protocolo devem ser consideradas.

	1	2	3	4	5	
Difícil	○	○	○	○	○	Muito fácil

O funcionamento da ferramenta é conforme o esperado. *

Quando você realizou sua ação, a ferramenta resultou de acordo com o esperado.

	1	2	3	4	5	
Nunca	○	○	○	○	○	Sempre

A ferramenta está em conformidade com o problema abordado. *

O problema é amenizado ou resolvido pela ferramenta?

	1	2	3	4	5	
Nunca	○	○	○	○	○	Sempre

A ferramenta possui tolerância à falhas. *

Você deve analizar se todo erro ocorrido foi tratado, ou seja, a ferramenta apresentou um erro que o impossibilitou de prosseguir?

	1	2	3	4	5	
Insuficientemente	○	○	○	○	○	Robusta

Os mecanismos de interface gráfica proporcionados ao usuário são adequados. *

O que você acha sobre os recursos disponibilizados para viabilizar sua interação com a ferramenta, exemplo desses mecanismos podem ser: menus, botões, barras de rolagem, eventos por mouse e teclado, entre outros.

	1	2	3	4	5	
Péssimos	○	○	○	○	○	Excelentes

Todas funcionalidades resultam em um resultado esperado pelo usuário. *

Dentre as operações realizadas na ferramenta, os resultados obtidos estavam em conformidade com o que você esperava?

	1	2	3	4	5	
Nunca	○	○	○	○	○	Sempre

Sugestões:

Enviar

100% concluído.

Nunca envie senhas em Formulários Google.