UNIVERSIDADE FEDERAL DO PAMPA

Eduardo Florindo Amaral

Uma Abordagem para Especificação de Requisitos Sensíveis ao contexto Baseado em Casos de Uso para Sistemas Autoadaptativos

Alegrete
2017

Eduardo Florindo Amaral

Uma Abordagem para Especificação de Requisitos Sensíveis ao contexto Baseado em Casos de Uso 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: Prof. João Pablo Silva da Silva

Alegrete
2017

Eduardo Florindo Amaral

Uma Abordagem para Especificação de Requisitos Sensíveis ao contexto Baseado em Casos de Uso 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 27 de Junho de 2017
Banca examinadora:

Prof. João Pablo Silva da Silva
Orientador
UNIPAMPA

Prof. Elder De Macedo Rodrigues
UNIPAMPA

Prof. Gilleanes Thorwald Araujo Guedes
UNIPAMPA

Humildemente dedicado às pessoas que me fizeram chegar até aqui:

Aos meus pais, Luís e Patrícia,
A minha irmã e melhor amiga, Brenda,
Com todo meu amor, por todos os sacrifícios feitos por mim
Espero ter sido merecedor do esforço dedicado por vocês.

AGRADECIMENTOS

Agradeço aos meus familiares pelo apoio e sacrifícios feitos, os quais me trouxeram até esse momento. Eu não seria capaz de chegar até aqui sem vocês.

Agradeço aos meus professores por incentivarem sempre a busca por conhecimento. Principalmente ao meu orientador João Pablo pelos importantes direcionamentos passados durante todo o período de orientação.

Aos meus melhores amigos: Brenda Mendes, Matheus Fillapi, Natália Mello, Vinícius Deneque, Talita Menezes, Kaio Rezende e Neto Iung; não poderia deixar de agradecer especialmente à vocês pelo apoio e motivação durante todo este tempo. Grato por terem ouvido minhas reclamações, minhas músicas de inspiração nas madrugas e por estarem sempre ao meu lado dizendo que tudo ia dar certo no final. Vocês tornaram os últimos anos uma experiência incrível, e espero muitos outros pela frente.

O final chegou e enfim, deu tudo certo!

"No matter what you do in life,
it's not legendary unless your friends are there to see it.
Friendship is an involuntary reflex,
it just happens,
you can't avoid it."
(Barney Stinson and Ted Mosby, How I Met Your Mother 2005-1014)

RESUMO

Sistemas autoadaptativos possuem certas peculiaridades com relação a sistemas tradicionais de software, fazendo com que a engenharia de requisitos tradicional não seja capaz de expressar corretamente suas funcionalidades. Por este motivo, alguns autores vêm tentando adaptar as técnicas existentes, de modo a tirar um maior proveito das mesmas. Este trabalho tem como objetivo estudar essas técnicas e com isso, criar uma abordagem para especificação de requisitos sensíveis ao contexto baseando-se em casos de uso, para dar auxílio aos engenheiros de requisitos que necessitam especificar requisitos com propriedades autoadaptativas. A metodologia executada neste trabalho consistiu na realização de um mapeamento sistemático da literatura para explorar linguagens para especificação de requisitos e compreender como as peculiaridades de sistemas autoadaptativos vem sendo tratadas nas mesmas. Com os conhecimentos adquiridos desenvolvemos um perfil UML para dar suporte a abordagem criada e possibilitar a especificação dos requisitos de forma expressiva. Este perfil UML foi exposto a uma estratégia de validação através da utilização do mesmo por usuários externos ao projeto e coleta de suas opiniões via questionário. Através da análise das respostas coletadas obtivemos respostas positivas sobre a expressividade, facilidade de modelagem e utilidade dos artefatos gerados, considerando assim alcançados os objetivos da abordagem.

Palavras-chave: Sistemas Autoadaptativos. Engenharia de Requisitos. Especificação de Requisitos.

ABSTRACT

Self-adaptive systems have certain peculiarities with respect to traditional software systems, so that traditional requirements engineering is not able to correctly express its functionalities. For this reason, some authors have been trying to adapt existing techniques in order to take advantage of them. This work aims to study these techniques and, therefore, to create an approach to specification of context-sensitive requirements based on use cases, to assist requirements engineers who need to specify requirements with self-adaptive properties. The methodology performed in this work consisted of a systematic mapping of the literature to explore languages for specification of requirements and to understand how the peculiarities of self-adaptive systems are being treated in them. With the acquired knowledge we developed a UML profile to support the created approach and enable the specification of the requirements in an expressive way. This profile was exposed to a validation strategy through the use of the same by users external to the project and collection of their opinions via questionnaire. Through the analysis of the collected responses we obtained positive responses about the expressiveness, ease of modeling and usefulness of the generated artifacts, thus considering the objectives of the approach.

Key-words: Self-adapt Systems. Requirements Engineering. Requirements Specification.

LISTA DE SIGLAS

ACM *Association for Computing Machinery*

ACS *American Chemical Society*

CR4AS *Context Requirement for Adaptive Systems*

EJB *Enterprise Java Beans*

ER Engenharia de Requisitos

IBM *International Business Machines*

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos

MOF *Meta-Object Facility*

OCL *Object Constraint Language*

OMG *Object Management Group*

SA Sistema Autoadaptativo

UML *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Sistemas Autoadaptativos (SAs) são definidos por Kneer e Kamsties (2015), como sistemas que possuem a capacidade de dinâmica e autonomamente, reconfigurar seus comportamentos para adequar-se às mudanças que ocorrem no contexto em que está inserido. Welsh e Sawyer (2010) alegam que a autoadaptação surgiu como uma estratégia para mitigar os custos de manutenção em sistemas que, por fatores como: complexidade elevada, fator de criticidade ou afastamento do sistema, fazem com que o desligamento do sistema para manutenção seja impraticável.

Segundo Welsh e Sawyer (2010) a autoadaptação atua oferecendo ao sistema um meio de responder às mudanças do ambiente em que se encontra, através de um sistema de detecção de mudança contextual ou ambiental em tempo de execução, possibilitando a adaptação dos comportamentos do sistema baseado nestas mudanças. Souza e Mylopoulos (2015) em seu trabalho, falam sobre o crescente interesse neste tipo de sistema, apontando como indicadores os recentes *workshops* e conferências com tópicos como adaptação e sistemas de software autônomos.

A consciência de contexto é uma das propriedades destes sistemas autoadaptativos, onde o contexto é percebido pelo mesmo e interfere no seu modo de execução. É definida por Abowd et al. (1999) como sendo a capacidade de um software em se adaptar de acordo com seu local de uso, a coleção de pessoas e objetos próximos, bem como as mudanças desses objetos ao longo do tempo.

Segundo Coutaz et al. (2005), a noção de contexto vem sendo modelada e explorada em diversas áreas desde a década de 1960. Apesar de ser debatida desde então pela comunidade científica, não se chegou a um consenso claro sobre uma definição de contextos. Uma das definições aceitas para contexto é dada por Dey (2001), que nos diz que:

> Contexto é qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que possui uma interação relevante entre um usuário e uma aplicação, incluindo os próprios usuário e aplicação.

Sommerville (2011) define a Engenharia de Requisitos (ER) como sendo um processo de investigação, análise, especificação e validação de serviços ou restrições que um sistema deve oferecer. Pressman (2009) complementa essa definição destacando que a ER tem como função efetuar a ligação entre a alocação de software em nível de sistema e o projeto de software. Ainda segundo Pressman (2009), faz-se necessário uma compreensão completa dos requisitos de um software para que o desenvolvimento do mesmo seja bem sucedido, tendo em vista que indiferente do quão bem um software seja projetado ou codificado, acabará desapontando o usuário se não suprir suas necessidades. O que pode ser evitado através de uma boa ER, ressaltando assim, a importância da mesma no desenvolvimento de softwares.

Parte do processo de engenharia de software consiste na especificação dos requisitos de software, produzida como uma consequência da tarefa de análise. A especificação de requisitos, segundo Pressman (2009), pode ser vista como um processo de representação, independentemente do modo pelo qual é realizada, resultando nas exigências dos clientes quanto ao software representadas de uma forma que leva à implementação bem-sucedida do software. Sommerville (2011) define os requisitos especificados resultantes desta etapa como versões expandidas dos requisitos de usuário que são capturados durante a etapa de análise. Nesta etapa, são acrescidos detalhes aos requisitos que explicam como os requisitos de usuário devem ser fornecidos pelo sistema. Sommerville (2011) define que além de uma boa especificação de requisitos ser crucial para o desenvolvimento correto do sistema esperado pelo usuário, eles podem também ser usados como parte do contrato para a implementação do sistema, sendo necessário para isso, ser uma especificação completa e consistente de todo o sistema.

Como dito por Pressman (2009), a ER é o primeiro passo técnico da engenharia de software. É o ponto onde as declarações gerais do escopo são obtidas do cliente e aprimoradas numa especificação completa, sendo a base para todas as atividades de engenharia de software que se seguirão. Sawyer et al. (2010) aponta que nesta ER tradicional, é implicitamente assumido que o contexto em que o software está incluído é estático e pode ser suficientemente entendido à ponto de permitir um modelo de requisitos para uma solução viável seja formulada com segurança. Na prática, sabemos que o contexto envolto ao software raramente é estático por um longo período, porém, a ER possui algumas técnicas capazes de mitigar ou evitar estes problemas, permitindo aos desenvolvedores avaliar as implicações e tomar as medidas necessárias. Mas como tratar o desenvolvimento de sistemas autoadaptativos que possuem uma ênfase muito maior na incerteza e mudanças constantes de contexto?

Alguns autores tem buscado adaptar as técnicas tradicionais. Luckey et al. (2011) definem uma adaptação de diagramas de casos de uso, buscando uma melhor definição do contexto durante o desenvolvimento, separando-o da lógica das funcionalidades do sistema. Já Whittle et al. (2010) propõem em seu trabalho a RELAX, uma linguagem natural estruturada para a escrita de requisitos para sistemas autoadaptativos que lida com os fatores de incerteza inerentes a este tipo de software. Estes trabalhos cumprem seus objetivos nas áreas específicas em que foram propostos, faltando uma abordagem mais geral para que os requisitos sejam especificados de forma expressiva para que se possam cumprir os objetivos da ER. Acredita-se que isso possa ser alcançado criando uma abordagem que combine o melhor das técnicas e outras abordagens propostas até então, sendo isso que esse trabalho se propõe.

1.1 Objetivos

Tendo por base as limitações na especificação de requisitos no contexto de sistemas autoadaptativos, este trabalho tem como objetivo a criação de uma abordagem que auxilie a especificação de casos de uso para este tipo de sistema, levando em conta suas incertezas e diferentes contextos. Para isto, segue-se os objetivos específicas a serem realizadas:

- Realizar um estudo exploratório para obter uma visão geral do estado da arte das linguagens de especificação de requisitos que deem suporte ao desenvolvimento de sistemas autoadaptativos.
- Investigar maneiras para a inserção da ideia de contextos nos requisitos de software.
- Propor um modelo de estrutura para casos de uso para inserção de contexto.
- Unir os conhecimentos e saídas adquiridos nos objetivos anteriores para desenvolver a abordagem para especificação de requisitos voltada a sistemas auto-adaptativos.
- Criar um perfil *Unified Modeling Language* (UML) que dê suporte a abordagem proposta.
- Realizar uma validação para o perfil UML proposto e coletar opinião de usuários.

1.2 Metodologia

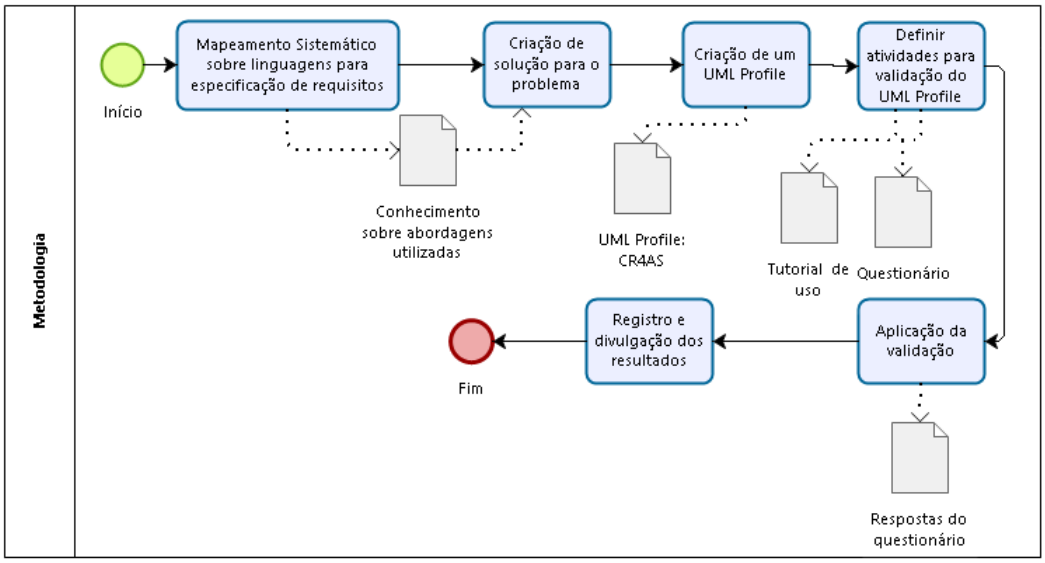
Para a realização deste trabalho, primeiramente foi realizado um mapeamento sistemático buscando explorar trabalhos relacionados à linguagens para especificação de requisitos e contextos em sistemas autoadaptativos. Após isso, buscou-se criar esboços de especificações de requisitos para prova de conceito e compreensão prática do problema. O primeiro passo da investigação foi a criação de um Perfil UML, onde foram analisados os elementos da UML e escolhidos elementos que poderiam ser estendidos adicionando uma nova semântica para que se encaixasse em nossa solução. Com o perfil UML criado, foram convidados usuários com familiaridade em ER para que testassem nosso perfil UML e comentassem sobre possíveis melhorias.

1.3 Organização do Documento

O presente trabalho está estruturado da seguinte maneira:

- Capítulo 2 - Fundamentação Teórica: onde são abordados assuntos que auxiliam no entendimento deste trabalho, como sistemas autoadaptativos, ER, linguagem RELAX e perfis UML.
- Capítulo 3 - Trabalhos Relacionados: descreve a metodologia de busca e os trabalhos relacionados encontrados.

Figura 1 – Metodologia aplicada no desenvolvimento do trabalho.



Fonte:o próprio autor.

- Capítulo 4 - Desenvolvimento: onde é descrito o problema, a proposta de solução e o que foi desenvolvido para alcançarmos nosso propósito.
- Capítulo 5 - Verificação e Validação do perfil UML: onde é apresentada e descrita em detalhes a estratégia para validação do perfil UML.
- Capítulo 6 - Considerações Finais: onde são descritos os desafios encontrados durante a pesquisa, o que foi aprendido e o resultado do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA

Neste capítulo são apresentados os fundamentos teóricos e tecnológicos necessários para a compreensão deste trabalho. Na seção 2.1 é apresentado o conceito de SAs, suas características, e as necessidades envoltas no desenvolvimento destes sistemas. Na seção 2.2 é apresentada a definição de especificação de requisitos e a sua importância na ER. A seção 2.3 apresenta uma visão geral sobre contexto e sobre como sistemas lidam com o mesmo. Na subseção 2.4.1 são apresentadas as definições de perfil UML. Na seção 2.5 são apresentadas as lições aprendidas neste capítulo.

2.1 Sistemas Autoadaptativos

Segundo Chen, Hiltunen e Schlichting (2001), SAs são softwares que tem a capacidade de alterar o seu comportamento em tempo de execução e com isso trazer uma série de potenciais benefícios, que vão desde a habilidade de responder rapidamente a falhas de segurança até a oportunidade de otimizar sua performance, baseando-se nas características do ambiente em que está inserido . Outra definição é dada pela DARPA *Broad Agency Announcement*:

> Softwares autoadaptativos avaliam seu próprio comportamento e o mudam quando a avaliação indica que não está sendo cumprido satisfatoriamente o que o software se destina a fazer, ou quando uma funcionalidade ou performance melhor é possível.

Uma definição similar é dada por Oreizy et al. (1999):

> Softwares autoadaptativos tem a capacidade de modificar seu próprio comportamento em resposta à mudanças em seu ambiente operacional. Por ambiente operacional, nós queremos dizer qualquer coisa observável pelo sistema do software, como entrada de usuários finais, dispositivos de hardware e sensores externos, ou instrumentação do programa.(SALEHIE; TAHVILDARI, 2009).

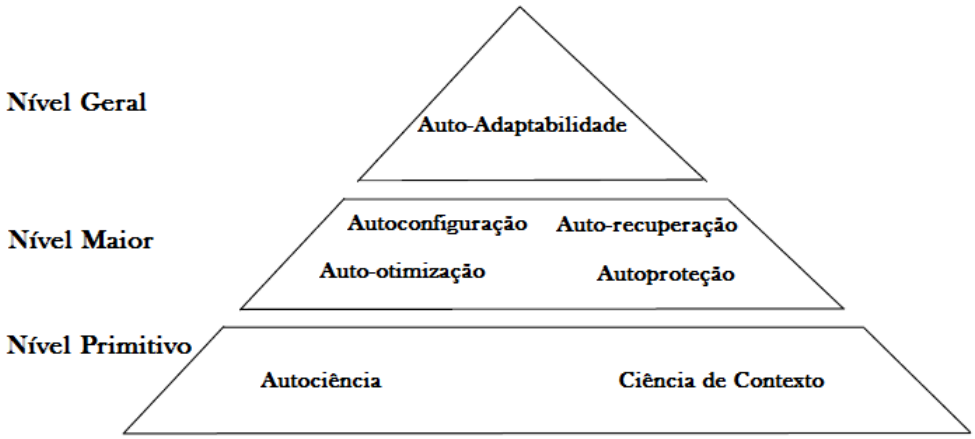
Segundo Salehie e Tahvildari (2009), propriedades autoadaptativas são muitas vezes chamadas de *self-* properties*. Ele relata que um dos primeiros conjuntos conhecidos de *self-* properties* foi introduzido pela *International Business Machines* (IBM) e inclui oito propriedades, que serão analisadas nesta subseção juntamente com outras propriedades relacionadas formando o conjunto hierárquico das propriedades autoadaptativas.

A Figura 2 apresenta a hierarquia das *self-* properties* em três níveis, sendo a auto-adaptabilidade e a auto-organização classificadas como propriedades gerais que são decompostas em propriedades menores divididas em dois diferentes níveis.

No nível geral são encontradas as propriedades globais dos softwares autoadaptativos. Sob a propriedade de auto-adaptabilidade, existe um sub-conjunto definido por Oreizy et al. (1999), composto por autogestão, autoconfiguração e automanutenção e definidos por Kephart e Chess (2003), autocontrole e auto-avaliação.

- **Autogestão** é a essência dos sistemas de computação autonômicos. Tem como objetivo que o sistemas se administre sozinho, não sendo necessário uma pessoa para

Figura 2 – Hierarquia das propriedades autoadaptativas.



FONTE:(SALEHIE; TAHVILDARI, 2009)

desempenhar esta função, buscando assim uma máquina executando sua máxima performance 24hrs/dia. O sistema autônomo continuamente monitora seu próprio uso, podendo buscar por atualizações de seus componentes, por exemplo.

- **Autoconfiguração** é a capacidade do sistema autonômico automaticamente se configurar seguindo políticas de alto nível (como objetivos de nível empresarial, por exemplo) que especificam o que é desejado, mas não o modo como realizar. Como exemplo, se um novo componente é introduzido no sistema como visto na Tabela 1, o sistema automaticamente aprende sobre ele levando em consideração a composição e configuração do sistema. Após isso o componente é registrado e o sistema adaptado, buscando modificar seus comportamentos apropriadamente.

No nível maior estão o conjunto de quatro propriedades definidas pela *IBM autonomic computing initiative* (IBM-AC, 2001). Esta classificação serve como padrão deste domínio e foram definidos de acordo com mecanismos auto-adaptativos biológicos, segundo Kephart e Chess (2003). Como exemplo das propriedades do corpo humano similares a estes princípios, temos a mudança de temperatura do corpo de acordo com os fatores ambientais, assim como a tentativa de recuperação do corpo à doenças ou falhas em órgãos internos.

- **Autoconfiguração** é a capacidade de se reconfigurar automaticamente e responder dinamicamente à mudanças por instalações, atualizações, integrações e composições/decomposições de entidades de software.

- **Auto-recuperação** é a capacidade de descobrir, diagnosticar e reagir a disrupções. Pode também antecipar potenciais problemas e tomar as ações adequadas para evitá-los.

Tabela 1 – Comparação de quatro aspectos da auto-adaptabilidade em sistemas tradicionais e sistemas autonômicos.

Aspecto	**Computação Tradicional**	**Computação Autonômica**
Autoconfiguração	*Data centers* corporativos possuem vários fornecedores e plataformas. As instalações, configurações e integrações de sistemas são demoradas e propensas a erros	As configurações automáticas de componentes e sistemas seguem uma política de alto-nível. O restante do sistema se ajusta automaticamente.
Auto-otimização	Os sistemas tem centenas de configurações manuais, parâmetros de ajuste não-lineares e seu número cresce a cada nova atualização.	Componentes e sistemas buscam continuamente oportunidades de aumentar sua própria performance e eficiência.
Auto-recuperação	A determinação dos problemas em grandes e complexos sistemas pode levar semanas, mesmo para um time de desenvolvedores	O sistema detecta, diagnostica e repara os problemas do software e hardware.
Autoproteção	Recuperação e detecção de ataques ou falhas em cascata são feitas manualmente	O sistema se defende automaticamente contra ataques maliciosos ou falhas em cascata. Ele utiliza de um alerta precoce para antecipar e prevenir falhas no sistema.

FONTE:(KEPHART; CHESS, 2003)

- **Auto-otimização** é a capacidade de gerenciar a performance e a alocação dos recursos para satisfazer os requisitos de diferentes usuários.
- **Autoproteção** é a capacidade de detectar problemas e brechas de segurança e se recuperar dos seus efeitos. Possui dois aspectos, defender o sistema de ataques maliciosos e antecipar problemas e tomar ações para mitigá-los.

No nível primitivo se encontram as propriedades primitivas adjacentes: autoconhecimento e ciência de contexto. Como dito por Salehie e Tahvildari (2009), existem outras propriedades que são mencionadas neste nível como a abertura em IBM-AC (2001) e a antecipação por Parashar e Hariri (2005), que são opcionais.

- **Autociência** Hinchey e Sterritt (2006) descrevem esta propriedade como o sistema tendo a capacidade de conhecer os seus próprios estados e comportamentos. Esta propriedade é baseada na auto-monitoração e reflete o que é monitorado.
- **Ciência de Contexto** Parashar e Hariri (2005) descrevem esta propriedade como o sistema tendo ciência do contexto em que ele está operando.

2.2 Especificação de Requisitos

Essa etapa, parte integrante da ER, refere-se à produção de um documento que pode ser sistematicamente revisto, avaliado e validado. Abran et al. (2001) aponta que para softwares mais complexos, podem ser criados três tipos de documentos nesta etapa,

sendo eles: definição do sistema, requisitos de sistema e requisitos de software. Já sistemas simples necessitam apenas do terceiro tipo. Pressman (2009) nos apresenta uma ideia parecida, indicando que o grau de formalidade das especificações deve ser proporcional ao tamanho e complexidade do software.

A especificação de requisitos de software estabelece a base para um acordo entre os clientes e os desenvolvedores, indicando tudo que o software deve ou não fazer. Além disso, por ser uma documentação precisa sobre todas as funcionalidades necessárias, ele permite uma avaliação rigorosa sobre as necessidades antes do desenvolvimento ter início, reduzindo o *re-design* mais tarde. Podendo também, devido à sua precisão, fornecer uma base realista para estimar os custos do produto, riscos e tempo necessário para o desenvolvimento (ABRAN et al., 2001).

Esta é uma importante etapa do desenvolvimento de software, onde os requisitos dever estar descritos em sua forma final, documentados em detalhes, indicando como e o que o software deve fazer. O documento criado nesta etapa guia as fases de projeto e desenvolvimento posteriores, além de servir como um contrato indicando todas as necessidades do cliente que devem ser satisfeitas pelo software, por isso, deve ser validado juntamente com todos os *stakeholders*.

Pfleeger e Atlee (2010) definem que este documento é uma re-escrita do documento de definição de requisitos gerado na elicitação de requisitos, porém, acrescido de detalhes e aspectos técnicos, necessários para o projeto e desenvolvimento do sistema.

Existem algumas técnicas para auxiliar a especificação de requisitos, a escolhida para nosso trabalho foi escrita de casos de uso, devido à sua expressividade e por ser definida como parte da linguagem UML, onde é possível criar uma extensão, modificando as regras para se adequar aos nossos objetivos.

2.2.1 Linguagem de Especificação RELAX

Como forma de especificar requisitos de SAs Whittle et al. (2010) apresentam em seu trabalho a RELAX, uma linguagem que trata os aspectos de incerteza presentes neste tipo de sistema. Os autores apontam que requisitos textuais normalmente são descritos utilizando operador SHALL (ou WILL) para definir as ações ou funcionalidades que o software deve cumprir. Já SAs nem sempre tem a possibilidade de satisfazer essas declarações, devido aos fatores de incerteza. A RELAX foi desenvolvida para auxiliar nesse aspecto, dando a possibilidade ao engenheiro de requisito especificar explicitamente requisitos que nunca devem mudar (invariantes), e também os requisitos que podem ser RELAX-ados[1] temporariamente sob certas condições.

Para que estes aspectos de incerteza possam ser explicitados durante a especificação de requisitos, são definidos uma série de operadores modais, temporais, ordinais e

[1] Assim como no trabalho de Whittle et al. (2010), utilizaremos a palavra RELAX como um verbo para indicar a inserção de operadores RELAX.

Tabela 2 – Operadores RELAX.

Operador RELAX	**Descrição**
Operadores Modais	
SHALL	um requisito deve possuir
MAY ... OR	um requisito apresenta uma ou mais alternativas
Operadores Temporais	
EVENTUALLY	um requisito deve acontecer eventualmente
UNTIL	um requisito deve manter até certa posição futura
BEFORE, AFTER	um requisito deve acontecer antes ou depois de um evento em particular
IN	um requisito deve acontecer durante um determinado intervalo de tempo
AS EARLY, LATE AS POSSIBLE	especifica que um requisito deve acontecer o mais cedo ou o mais tarde possível
AS CLOSE AS POSSIBLE TO [frequency]	especifica que um requisito deve acontecer repetidamente, mas a frequencia pode ser relaxada
Operadores Ordinais	
AS CLOSE AS POSSIBLE TO [quantity]	o requisito especifica uma quantidade contável, mas a quantidade pode ser relaxada
AS MANY, FEW AS POSSIBLE	o requisito especifica uma quantidade contável, mas a quantidade pode ser relaxada
Fatores de Incerteza	
ENV	define um conjunto de propriedades que definem o ambiente do sistema
MON	define um conjunto de propriedades que podem ser monitoradas pelo sistema
REL	define a relação entre ENV e MON
DEP	identifica a dependência entre os (relaxados e invariáveis) requisitos

Fonte: (WHITTLE et al., 2010)

fatores de incerteza. Os operadores suportados, assim como sua descrição, são exibidos na Tabela 2.

A sintaxe das expressões RELAX é definida pela gramática apresentada a seguir. Onde: p é uma proposição atômica, e é um evento, t é um intervalo de tempo, f uma frequência e q uma quantidade. Uma expressão RELAX é um conjunto de declarações S_1;...;S_m, onde cada S_i é gerado pela gramática a seguir.

2.3 Sistemas Sensíveis a Contexto

Segundo Gay e Hembrooke (2004), dispositivos computacionais estão cada vez mais presentes nas vidas das pessoas, sendo utilizados cada vez mais em cenários distintos, o que faz com que pesquisadores tenham que fazer cada vez mais perguntas para compreender suas funcionalidades e os contextos em qual serão utilizados. Ainda segundo os autores, é necessário que os projetistas aprendam lições de antropologia cultural, sociologia, ciências

Figura 3 – Gramática da RELAX.

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

Fonte: (WHITTLE et al., 2010)

da informação e psicologia para que possam compreender os usuários e as maneiras como essas novas ferramentas afetaram suas rotinas.

Gasparini (2013) comenta em seu trabalho que sistemas sensíveis (ou cientes) ao contexto pertencem à categoria de computação ubíqua, e incluem funções simples como sensibilidade à localização (à partir de posicionamento infravermelho ou global para determinar a localização do usuário para disseminar informação que possa ser relevante para ele) ou apresentar sensores inteligentes (como sensibilidade à iluminação, geladeiras falantes, latas de lixo inteligentes, casas inteligentes, etc). Segundo a autora, compreender como esses níveis de contexto influenciam e impactam na computação é fundamental para o desenvolvimento de tecnologias cientes de contexto úteis e eficazes.

Gay e Hembrooke (2004) em seu livro, separam a forma como o contexto influencia no comportamento geral e computacional em três abordagens teóricas principais.

A primeira abordagem diz que o contexto é o resultado das atividades realizadas. Com o ambiente e as variáveis do ambiente incentivando atividades diferentes, as atividades mudam, o que altera o contexto. Nessa abordagem, as variáveis do ambiente referem-se a objetivos e intenções do usuário, assim como variáveis do ambiente físico.

A segunda abordagem vem da teoria da ação situada, onde a cognição e as atividades planejadas são conectadas e ambas produções das interações sociais e físicas que os indivíduos têm com o ambiente. Fazendo assim com que o contexto seja dinâmico, devido às sequencias de ações realizadas em um contexto serem fluidas e sensíveis às alterações sociais e condições físicas.

Já a terceira abordagem, a Fenomenologia, sustenta que os indivíduos fazem sentido no mundo através de sua participação. O significado é extraído diretamente à partir da informação disponível pelo ambiente, sendo ação e interação apenas instrumentos para que essa extração se torne possível.

Essas três abordagens tem como ponto em comum focar no indivíduo e não defi-

nem o contexto como um conjunto definido de propriedades, mas sim como um atributo emergente e fluido, que desencadeia-se através das atividades impostas a ele. Conclui-se que contexto é um termo operacional, e não uma construção física. Ou seja, algo está em contexto devido a forma que é utilizada, não devido às suas propriedades inerentes (GASPARINI, 2013).

Uma definição de *Context-aware computing*[2] é apresentada por Dey (2001) como "um sistema é ciente de contexto se ele usa contexto para fornecer informações relevantes, e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário". Ele define três categorias de recursos que uma aplicação ciente de contexto pode suportar, sendo elas:

- Apresentar informações e serviços a um usuário.
- A execução automática de um serviço para um usuário.
- A marcação de contexto à informação para apoiar a recuperação posterior.

Hong, Suh e Kim (2009) ressaltam em seu trabalho que a área que trata sistemas de contexto ainda está em desenvolvimento e esses sistemas ainda não estão implementados da melhor forma na vida real.

2.4 DSML

Segundo Pati, Kolli e Hill (2017), a criação de linguagens gráficas para modelagem de domínio específico é uma técnica de engenharia orientada a modelo, onde são desenvolvidas linguagens gráficas para representação de elementos, restrições e semânticas de um determinado domínio específico. São utilizadas para modelar conceitos ou soluções para o domínio alvo, sendo assim possível uma modelagem mais conveniente para o projetista.

Exemplos de ferramentas DSML incluem:

- o ambiente de modelagem genérico (GME)
- o sistema genômico de modelagem do eclipse (GEMS)
- a estrutura de modelagem do eclipse (EMF)
- ferramentas de linguagem específica de domínio (DSLs)

Segundo Šilingas et al. (2009) o desenvolvimento de DSML's envolve a definição de sintaxe, semântica, validação e transformações em modelos ou códigos. As abordagens utilizadas no campo de desenvolvimento de DSML's podem ser distinguidas entre duas abordagens: criar uma DSML como uma instância de um certo metamodelo ou criar uma

[2] Ciência de contexto, refere-se a ideia de que computadores podem sentir e reagir baseados no cenário em que se encontram.

DSML estendendo um outra linguagem de modelagem. A primeira abordagem trata da construção de uma nova linguagem partindo do zero, já a segunda está relacionada à extensão de uma linguagem já existente, estendendo suas características para adicionar um novo significado que se adeque ao domínio específico que pretende ser modelado.

Em nosso trabalho optamos pela utilização da segunda abordagem, estendendo o metamodelo da UML através de perfis UML, que são explicados na subseção 2.4.1 à seguir.

2.4.1 Perfil UML

Segundo Booch, Rumbaugh e Jacobson (2006), a UML (Linguagem Unificada de Modelagem) é uma linguagem gráfica com a qual é possível a visualização, especificação, construção e documentação de artefatos de sistemas complexos de software, proporcionando uma forma padrão para a preparação de planos de arquitetura de projetos de sistema, incluindo aspectos conceituais como processos de negócios até aspectos concretos, como classes escritas em determinada linguagem.

A UML é uma linguagem ampla projetada buscando a capacidade de ser utilizada na modelagem dos mais variados tipos de software, porém, não é incomum vê-la não sendo capaz de representar um sistema ou aplicação de forma conveniente do ponto de vista dos projetistas, como podemos ver nas motivações de Hsu (2012), Benselim e Seridi-Bouchelaghem (2017) e Boulil, Bimonte e Pinet (2014). Para situações como esta, um perfil UML pode oferecer os recursos necessários para isto.

Segundo Fuentes-Fernández e Vallecillo-Moreno (2004), quando necessitamos definir uma nova linguagem que restrinja o número de elementos UML ou adicione novas restrições ou um novo valor sintático a eles, não é necessário a criação de uma nova linguagem a partir do zero utilizando o *Meta-Object Facility* (MOF). A própria UML já possui uma série de mecanismos de extensão para evitar este re-trabalho. A UML 2.0 trás o pacote *Profiles* que define um conjunto de artefatos UML que permitem a especificação de um modelo MOF para lidar com conceitos específicos de certos domínios de aplicação.

A UML 2.0 apresenta várias razões pelas quais um desenvolvedor de sistemas deve querer personalizar seu domínio, sendo elas:

- Para se ter uma terminologia adaptada a uma plataforma ou domínio específico, como por exemplo, ser capaz de capturar a terminologia *Enterprise Java Beans* (EJB).
- Para se ter uma sintaxe para construções que não possuem uma notação.
- Para se ter uma notação diferente para elementos já existentes que seja mais apropriada para um domínio específico.
- Para adicionar semânticas não especificadas no metamodelo.

- Para adicionar semânticas que não existem no metamodelo (como definir um *timer*).
- Para adicionar restrições que restringem a forma como o metamodelo é utilizado, baseando-se no domínio específico modelado.
- Para adicionar informações que podem ser usadas para transformação de modelo em outro modelo ou em código (como definir regras de mapeamento entre modelo e código java).

Fuentes-Fernández e Vallecillo-Moreno (2004) nos apresentam as seguintes orientações para a definição e uso de perfis UML:

1. Definir o conjunto de elementos que irão constituir a modelagem de domínio específico, o que pode ser expresso por forma de metamodelo, que pode ser facilmente definido utilizando os mecanismos padrões oferecidos pela UML. No metamodelo é preciso incluir a definição das entidades de domínio, as relações entre eles e as restrições que governam a estrutura e o comportamento dessas entidades.
2. O próximo passo é incluir um estereótipo para cada elemento relevante do metamodelo que será incluído no perfil UML.
3. Após isso, cada estereótipo deve representar um elemento do metamodelo UML através de extensão de sua metaclasse.
4. Devem ser adicionados os atributos como *tagged values*, adicionando seu tipo e valor inicial.
5. Devem ser definidas as restrições do domínio específico com a linguagem *Object Constraint Language* (OCL), por exemplo.

2.4.1.1 Papyrus

Para o desenvolvimento do perfil UML de nosso trabalho utilizamos a ferramenta Papyrus. Como descrito em Atomique Atos (2016), Papyrus é um *plugin* Eclipse de código aberto que busca fornecer um ambiente integrado para a edição de modelos UML e SysML. Ele fornece um suporte avançado à criação de perfis UML.

Foi desenvolvido pelo *Laboratory of Model Driven Engineering for Embedded Systems* (LISE), que é parte da *French Alternative Energies and Atomic Energy Commission* (CEA-List).

Entre suas características está a compatibilidade com UML 2.5, servindo como um editor gráfico para a UML 2 definida pela *Object Management Group* (OMG), implementando 100% de sua especificação.

2.4.1.2 OCL

Segundo Warmer e Kleppe (2003), a OCL é uma linguagem de modelagem com a qual você pode construir modelos de software que é definido como um complemento padrão para a UML. Toda expressão escrita em OCL depende dos tipos que são definidos nos diagramas UML, sendo assim, sendo assim, inclui o uso de pelo menos alguns aspectos em UML. Egea e Dania (2017) explicam em seu trabalho que a OCL surgiu originalmente como uma forma de modelar propriedades que não podem ser facilmente ou naturalmente capturadas utilizando notações gráficas (por exemplo, classes invariantes de um diagrama de classes UML).

Ainda segundo Egea e Dania (2017), a OCL é uma linguagem de especificação pura, também considerada como uma linguagem de modelagem textual, sendo as expressões OCL sempre escritas no contexto de um modelo e são avaliadas em cenários desse modelo. Esta avaliação retorna um valor, mas não altera nada do modelo, sendo uma linguagem livre de efeitos colaterais. A OCL pode ser usada como linguagem de restrição e como linguagem de consulta, ou seja, OCL pode ser usado para analisar modelos e validá-los em cenários selecionados ou estados de sistema concretos, bem como para iniciar consultas arbitrárias sobre modelos.

2.5 Lições do Capítulo

Neste capítulo foi possível um estudo mais aprofundado sobre os fundamentos utilizados neste trabalho. Podemos compreender o que são SAs, as propriedades que eles podem possuir e as suas peculiaridades frente a sistemas tradicionais, que fazem com que técnicas da ER devam ser adaptadas para serem utilizadas neste tipo de sistema. Percebemos também a importância da ER, seja no desenvolvimento de SAs ou no desenvolvimento de sistemas tradicionais, sendo ela a base para todo o restante do desenvolvimento.

Foi possível constatar que o contexto, que tem influência direta nos SAs não é uma área muito exata, ainda sendo desenvolvida e com várias definições possíveis. Além disso, foi possível o estudo sobre o que são perfis UML e as situações em que são utilizados.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados um conjunto de trabalhos relacionados à linguagens para especificação de requisitos que possuam algum tipo de suporte para propriedades autoadaptativas, que buscam responder às questões de pesquisa investigadas neste estudo. Além disso, é apresentada a metodologia aplicada para que se chegasse a estes resultados. Na seção 3.1 é apresentada a metodologia utilizada para coleta dos trabalhos. Na seção 3.2 são apresentados os resultados obtidos após aplicação destas metodologias e como eles se relacionam com as questões de pesquisa que guiam este trabalho. Na seção 3.3 é apresentado um resumo sobre o entendimento do autor e as lições aprendidas após a leitura dos trabalhos relacionados.

3.1 Mapeamento Sistemático

Os trabalhos relacionados foram selecionados baseados no protocolo para mapeamento sistemático proposto por Petersen et al. (2008) em seu trabalho.

Seguindo a ideia proposta por Petersen, primeiramente foram definidas as questões de pesquisa que guiariam as buscas, sendo elas:

RQ1: Quais as linguagens permitem especificar requisitos de adaptação?

RQ2: Quais as linguagens permitem capturar a incerteza de requisitos?

RQ3: Quais as linguagens permitem especificar requisitos sensíveis ao contexto?

RQ4: Quais linguagens permitem especificar os comportamentos fuzzy?

Estas questões de busca foram definidas buscando cobrir todas as características necessárias para que uma linguagem dê suporte às peculiaridades de requisitos para SAs.

A fonte escolhida para pesquisa foi a Scopus, autodenominada como "a maior base de dados de resumos e citações de literatura peer-reviewed" (ELSEVIER, 2016), indexando revistas científicas, livros e anais de conferências de bases como Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE), *American Chemical Society* (ACS), *Association for Computing Machinery* (ACM), entre outras.

Para a pesquisa e coleta dos resultados primários, foi definida uma string de busca, levando-se em conta as variações possíveis para a busca da população e intervenção a serem buscadas, que podem ser vistas na Tabela 3, baseada nos operadores disponíveis no sistema de busca da Scopus, sendo o resultado a seguinte string:

"TITLE-ABS-KEY(requirement* AND ((specification OR modeling OR requirements) PRE/0 language) AND (*adapt* OR uncertain* OR context* OR fuzzy))"

Tabela 3 – População e Intervenção utilizados na pesquisa.

População	**Intervenção**
Requirement	adapt
Requirement**(s)**	**(Self-)**adapt
Specification language	adapt**(ive)**
Modeling language	adapt**(ation)**
Requirements language	uncertain
	uncertain**(ty)**
	context
	context**(ual)**
	context**(-aware)**
	fuzzy

Essa *string* retornou um total de 818 resultados, onde foram aplicados filtros disponíveis na própria Scopus de modo a diminuir a população de artigos, para que possam ser analisados somente artigos que estejam dentre o interesse da pesquisa. Sendo eles:

- Artigos publicados até 18/11/2016, dia em que a pesquisa foi realizada;
- Artigos pertencentes a área de Ciência da Computação, a área de interesse da pesquisa;
- Trabalhos publicados em conferências e *journals*;
- Trabalhos publicados em língua inglesa, portuguesa e espanhola.

Após essa filtragem inicial, foram obtidos um total de 464 resultados, expostos a critérios de inclusão e exclusão, definidos anteriormente, buscando encontrar os artigos de interesse para este estudo, que podem ser vistos na Tabela 4.

Tabela 4 – Critérios de Inclusão e Exclusão.

Critérios de Exclusão	**Critérios de Inclusão**
Trabalhos não disponíveis através do *proxy* da UNIPAMPA	Trabalhos que abordem em seus objetivos explorar linguagens de especificação de requisitos
Revisões bibliográficas	Trabalhos que relatem algum tipo de aplicação da linguagem abordada
Trabalhos que apresentem a mesma contribuição	Trabalhos com menos de 6 páginas

A análise teve início a partir do primeiro critério de exclusão, buscando encontrar quais dos trabalhos encontram-se disponíveis à partir do proxy disponibilizado pela UNIPAMPA. Ao final da aplicação deste critério obteve-se um total de 366 trabalhos aprovados para serem analisados quanto ao critério seguinte, como visto no gráfico 4.

O critério seguinte analisou os trabalhos quanto à quantidade de páginas, buscando por trabalhos com mais de 6 páginas, pois entende-se que trabalhos com uma quantidade de páginas menor que essa não trazem uma contribuição suficiente para o tema. Ao final da aplicação deste critério, contatou-se que dos 366 trabalhos analisados, 312 possuíam mais de 6 páginas, como visto no gráfico 4. Passando para a aplicação do próximo critério de exclusão.

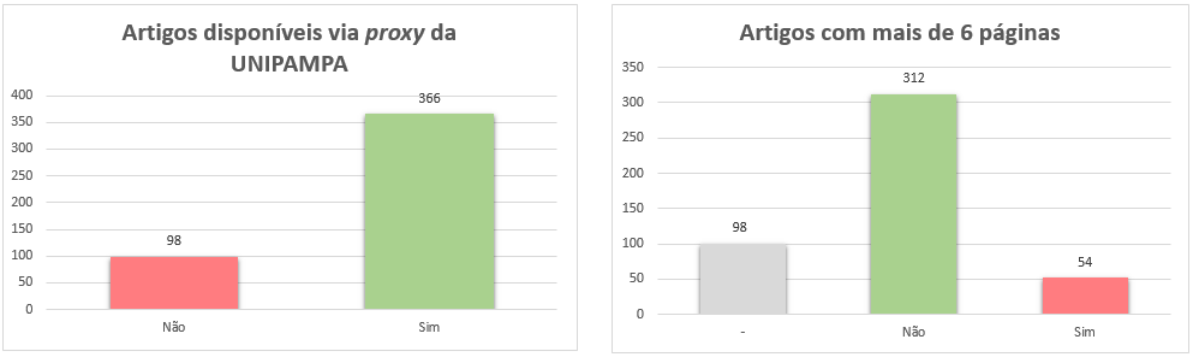
Destes 312 trabalhos, foram eliminados os trabalhos onde eram realizadas apenas revisões bibliográficas e não traziam um estudo sobre uma linguagem específica, não se aprofundando o satisfatoriamente no assunto, resultando em um total de 306 trabalhos, como visto no gráfico 5.

O critério seguinte foi o de trabalhos que explorassem linguagens de especificação de requisitos, que seriam o foco principal de nosso estudo. Dos 306 trabalhos analisados, foi constatado que 31 satisfaziam o critério, como visto no gráfico 5.

Para efeito de qualidade, optou-se por eliminar destes, trabalhos que não apresentassem alguma forma de teste da linguagem, como estudos de caso ou experimentos empíricos. Sendo eliminados destes, 17 artigos, como visto no gráfico 6.

Dos 14 trabalhos restantes, constatou-se que 3 apresentavam a mesma linguagem, como versões resumidas de artigos já presentes na pesquisa, sendo assim eliminados, como visto no gráfico 6, Resultando um total de 11 artigos selecionados e expostos a leitura

Figura 4 – Gráficos representando critérios de inclusão e exclusão (1)

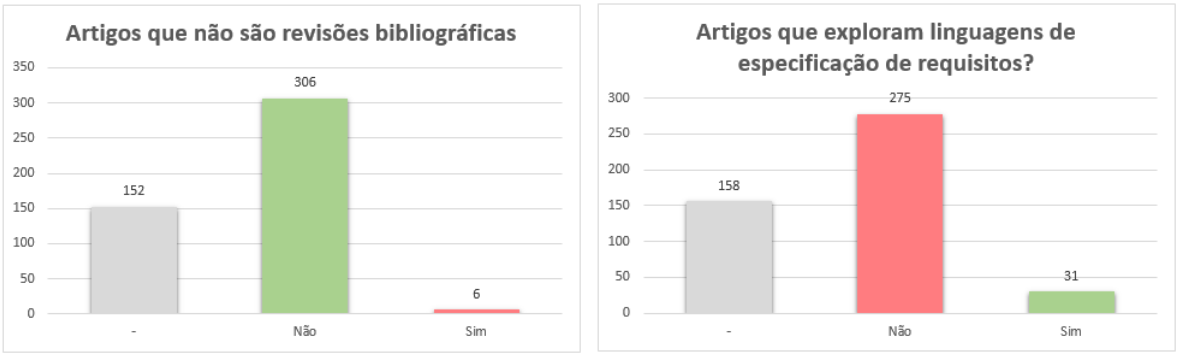


Fonte: o próprio autor.

3.2 Resultados da Pesquisa

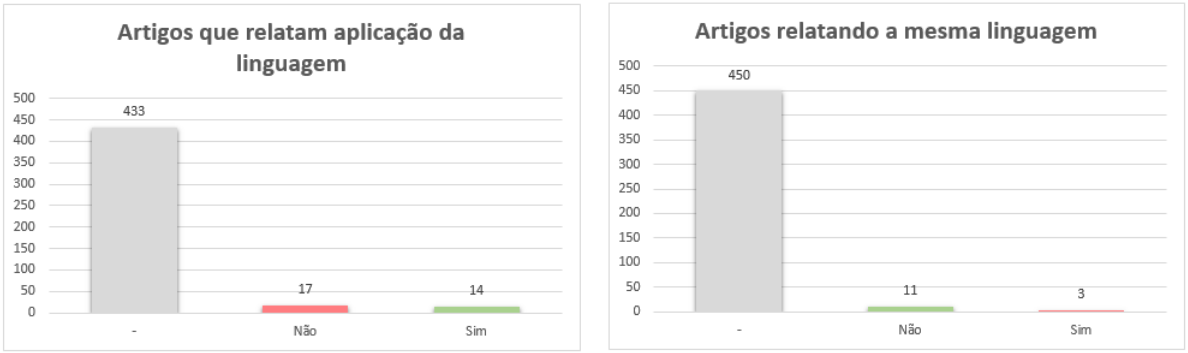
Após a realização do mapeamento e análise dos 11 trabalhos finais buscou-se responder às questões de pesquisa definidas anteriormente. Uma análise individual de cada uma destas perguntas é exposta nas subseções a seguir.

Figura 5 – Gráficos representando critérios de inclusão e exclusão (2)



Fonte: o próprio autor.

Figura 6 – Gráficos representando critérios de inclusão e exclusão (3)



Fonte: o próprio autor.

3.2.1 Quais as linguagens permitem especificar requisitos de adaptação?

A cada dia é maior a demanda por softwares complexos e distribuídos que possam suprir as necessidades criadas pelos softwares estarem cada vez mais presentes em nosso cotidiano, criando uma necessidade de desenvolvimento de sistemas que sejam capazes de se adaptar autonomamente a situações diversas.

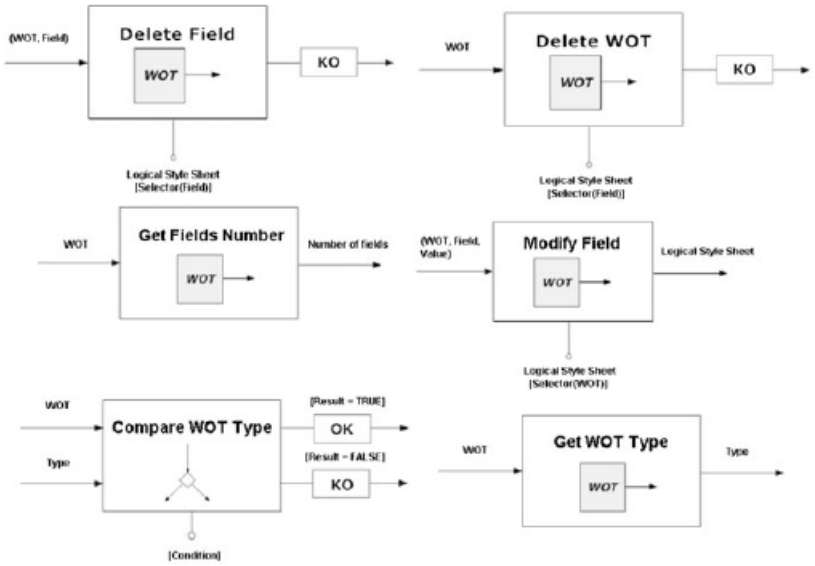
Segundo Morandini et al. (2017), a propriedade de adaptação em si, se preocupa sobre **como** o software reage de acordo com mudanças esperadas e inesperadas no ambiente que o cerca. Isto não é uma capacidade prevista na ER tradicional, sendo necessário a busca por novas metodologias ou adaptação de antigas para uma ER satisfatória.

Uma destas é a **Tropos4AS**, proposta por Morandini et al. (2017), que busca fornecer aos analistas características de modelagem capazes de representar capacidades adaptativas, para reforçar a análise de requisitos sobre os conhecimentos específicos e critérios de decisão necessários para um sistema adaptativo adaptar-se autonomamente a mudanças dinâmicas. Segundo os autores, a Tropos4AS baseia-se em 3 pilares principais, sendo o primeiro ligado diretamente a capacidade de adaptação: "a criação de um modelo de objetivos que inclui informações sobre os tipos de objetivos e as condições de satisfação

associadas", sendo assim possível através de um monitor de requisitos serem efetuadas mudanças efetivas para que seja buscado a satisfação dos mesmos.

Diferente da anterior, a **AML**, proposta por Virgilio (2012), é uma linguagem focada para sistemas web, que segundo os autores, tem uma grande necessidade de adaptação devido as diferenças de dispositivos e preferências de usuário a quais o software será exposto. As características de adaptação são tratadas na AML através de um conjunto extensível pra modelagem primitiva, representando manipulações elementares para definir de forma simples o *workflow* do processo adaptativo, como mostrado na Figura 7.

Figura 7 – Representação em AML das capacidades adaptativas visuais.



Fonte: (VIRGILIO, 2012)

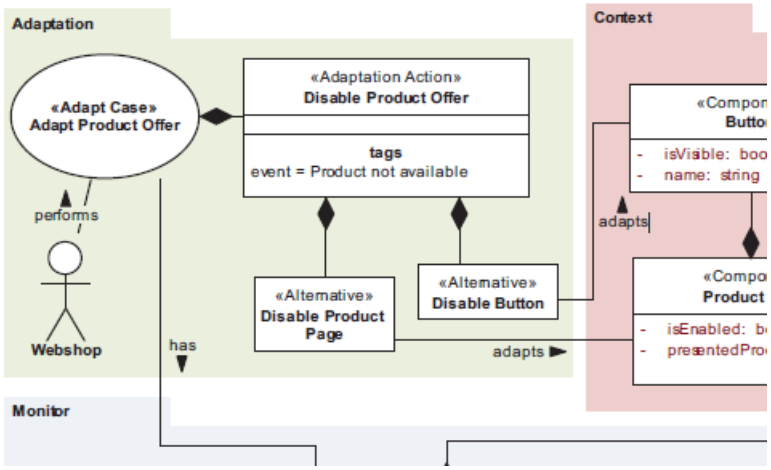
Outra linguagem que busca facilitar a modelagem de SAs é a **Adapt Case**, onde os autores Luckey et al. (2011) possuem motivações parecidas com as já apresentadas em outros trabalhos, porém com o diferencial de acreditar que em grande parte das metodologias criadas com o propósito de representar características adaptativas, não há uma distinção clara sobre o que é funcionalidade e o que são as capacidades adaptativas do software, o que julgam prejudicial ao desenvolvimento.

Sendo assim, os autores propõem nesta abordagem uma separação mais clara dos conceitos de adaptação das funcionalidades no sistema, como pode ser visto na Figura 8 retirada do trabalho de Luckey et al. (2011).

A **Extensão de diagrama de atividade** proposta por Al-alshuhai e Siewe (2015) trata a adaptação pelo acréscimo de notações para que possam ser representadas as capacidades adaptativas do sistema.

A **RELAX**, proposta por Whittle et al. (2010), por outro lado, é uma gramática que age diretamente na escrita dos requisitos usando um conjunto sintático e semântico de operadores para que requisitos que necessitam de adaptação sejam "RELAX-ados",

Figura 8 – Representação da divisão dos conceitos adaptativos.



Fonte: (LUCKEY et al., 2011)

como definido pelo autor. Segundo ele, outras abordagens necessitam que o engenheiro de requisitos enumere todos os possíveis pontos de adaptação requeridos. Ao invés disso, a **RELAX** possibilita ao mesmo definir pontos de flexibilidade ou incerteza aos requisitos, desta forma potencializando a antecipação das adaptações que podem ser necessárias. Na Figura 9 podemos ver um requisito RELAX-ado, sendo os pontos de flexibilidade (palavras em maiúsculo como SHALL, AS CLOSE AS POSSIBLE, etc), indicando onde pode ocorrer adaptação para satisfação das funcionalidades.

Figura 9 – Requisito RELAX-ado.

S1': The synchronization process *SHALL* be initiated *AS EARLY AS POSSIBLE AFTER* Alice enters the room and *AS CLOSE AS POSSIBLE TO* 30 min intervals thereafter.
ENV: location of Alice; synchronization interval.
MON: motion sensors; network sensors
REL: motion sensors provide location of Alice; network sensors provide synchronization interval

Fonte: (WHITTLE et al., 2010)

3.2.2 Quais as linguagens permitem capturar a incerteza de requisitos?

Um dos maiores desafios na ER para SAs, além dos já citados anteriormente, é como lidar com falta ou incerteza de informações sobre onde o sistema irá operar.

Os operadores utilizados pela **Relax**, proposta por Whittle et al. (2010) para relax-ar seus requisitos, na verdade baseiam-se nestas questões de incerteza, utilizando de lógica *fuzzy* para definir um certo intervalo de valores para que certa decisão esteja certa ou errada. Enquanto que na **Tropos4AS**, proposta por Morandini et al. (2017) a

gestão da incerteza é realizada através da modelagem de comportamentos alternativos, associados às condições e qualidades do ambiente.

3.2.3 Quais as linguagens permitem especificar requisitos sensíveis ao contexto?

Outro fator importante, que influencia nas questões levantadas acima, é o contexto em que o SA está envolto, pois se relaciona diretamente com as necessidades autoadaptativas do sistema.

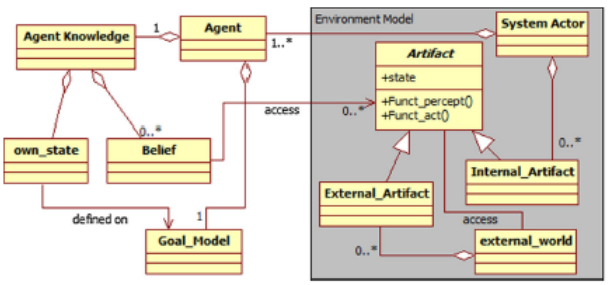
A **Extensão de diagrama de atividade** proposta por Al-alshuhai e Siewe (2015) trata o contexto de forma igual a adaptação, com o acréscimo de notações para que possam ser representadas as características do contexto em que o sistema será envolto. As notações podem ser vistas na Figura 10. Já a **Tropos4AS** lida com este fator através da construção de um modelo de ambiente, que pode ser visto na Figura 11 para representar os elementos-chave no contexto do sistema, que podem vir a afetar a satisfação dos objetivos definidos.

Figura 10 – Conjunto de notações adicionados.

Context related notations	UML notations
Context Object	Object
Context Constraint	Decision
Adaptation Action	State
Meta-swimlane separation	Swimlane separation

Fonte: (AL-ALSHUHAI; SIEWE, 2015)

Figura 11 – Meta-modelo da Tropos4AS, focando na relação dos agentes com o modelo de ambiente.

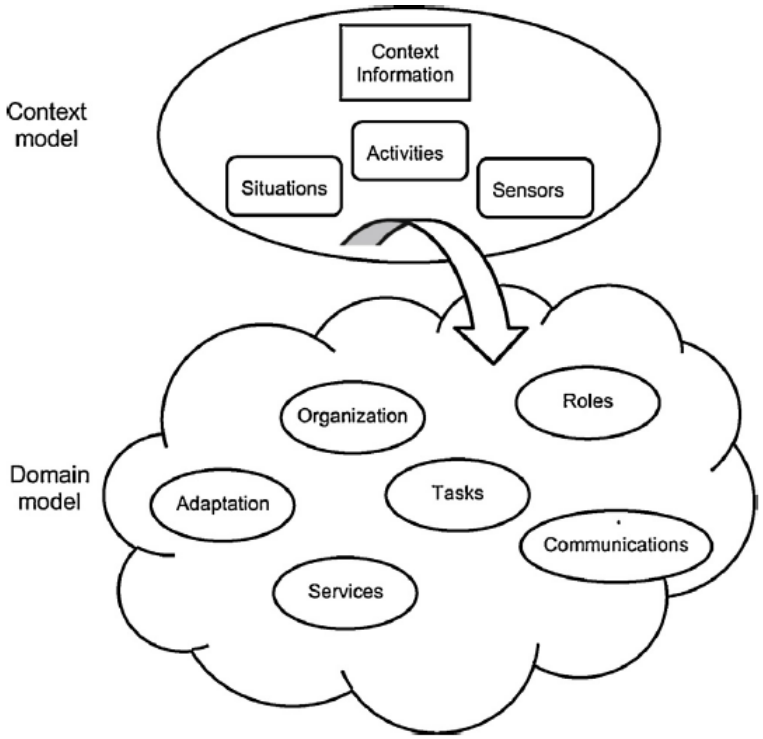


Fonte: (MORANDINI et al., 2017)

Na **Relax** o contexto é passado através dos operadores **ENV, MON** e **REL**, como visto na figura 9, sendo definido em ENV os contextos envolvidos no requisito, MON são as fontes pelas quais as informações do contexto serão recebidas pelo sistema (como sensores, por exemplo) e REL é a relação dos sensores com os contextos.

A **MLContext** proposta em Hoyos, García-Molina e Botía (2013) define como uma peça chave do desenvolvimento de sistemas sensíveis ao contexto a criação de modelos que auxiliem a compreender e raciocinar sobre os contextos. Por este motivo, os autores propõem em sua abordagem a construção de uma DSL onde seja possível de forma clara, modelar os diferentes tipos de contexto em que o software está envolto. O modelo de contexto utilizado pode ser visto na figura 12. De forma parecida, a **AML** também utiliza em sua metodologia a construção de um meta-modelo conceitual, descrevendo as informações de contexto em que o software estará envolto.

Figura 12 – Modelo do contexto utilizado na MLContext.



Fonte: (**??**)

A abordagem **ContextUML** proposta por (SHENG; BENATALLAH, 2005), assim como as citadas anteriormente, baseia-se na modelagem das informações de contexto, diferindo-se ao dividir o contexto em dois tipos: atômico e composto. Contextos atômicos são contextos de baixo nível que não dependem de outros contextos e pode ser fornecido diretamente por sensores ou serviços que forneçam estas informações. Em contraste, os contextos compostos são contextos de alto nível que podem não ter contrapartidas diretas na provisão de contexto. Um contexto composto agrega múltiplos contextos, seja

atômico ou composto. O conceito de contexto composto pode ser usado para fornecer um vocabulário rico de modelagem, segundo os autores Sheng e Benatallah (2005).

3.2.4 Quais linguagens permitem especificar os comportamentos fuzzy?

Diferente dos conceitos de lógica proposicional que nos apresenta 2 valores, certo ou errado, a lógica fuzzy pode nos apresentar um conjunto de valores possíveis, podendo indicar o quanto certa informação está certa ou errada. Esta característica faz com que este tipo de lógica se encaixe muito bem em SAs que possuem um certo grau de incerteza envolto.

A **FTL-CFree** é uma linguagem de especificação funcional *fuzzy* temporal livre de contexto, criada por Pérez et al. (2014) utilizando os conceitos de lógica *fuzzy* para possibilitar a industria modelar e verificar cenários complexos em tempo de execução.

Diferente dos trabalhos até então que tratam de requisitos funcionais do sistema, a **NFRQ** vem de uma forma diferente, abordando os requisitos não-funcionais e as incertezas inerentes a este tipo de requisito, principalmente pelo pouco conhecimento do cliente sobre o mesmo. Para suprir a falta de informações fornecidas pelo cliente, os autores (LI et al., 2014) propõem em sua abordagem o uso de lógica *fuzzy*.

Já a **Relax**, como já dita anteriormente, utiliza de lógica fuzzy em seus parâmetros, para relax-ar os requisitos, criando pontos de flexibilidade que podem ser utilizados para adaptação.

3.2.5 Linguagens que não atenderam os critérios

Apesar das questões e *string* de busca buscarem por trabalhos que atendessem as quatro características vistas, dois dos trabalhos relacionados retornaram trabalhos que não atendem diretamente nenhuma das características, sendo eles:

CASL

Em Astesiano et al. (2002), os autores nos apresentam a Common Algebraic Specification Language (CASL), definida pelos autores como "uma linguagem expressiva para a especificação formal de requisitos funcionais e *design* modular de software". Suas principais *features* são:

- uma seleção criteriosa dos **construtores** escolhidos
- ser expressiva, simples e pragmática
- ser capaz de especificar requisitos e projetos de softwares convencionais
- apresentar restrições à sub-linguagens

Tabela 5 – Tabela de análise das linguagens encontradas.

Linguagem	Adaptação	Incerteza	Contexto	Fuzzy	Cit	H-index	Ano
UML Activity Diagram Ext.			X		2	3.5	2015
Tropos4AS	X	X	X		3	9.75	2015
FTL-CFree				X	8	4.4	2014
NFRQ				X	14	16.6	2014
RSL-IL					9	5	2013
MLContext			X		19	7.5	2013
AML	X		X		1	9	2012
Adapt Case	X				31	8.5	2010
Relax	X	X	X	X	125	11.2	2010
ContextUML			X		270	25	2005
CASL					232	13.6	2005

Fonte: o próprio autor.

RSL-IL

Em Ferreira e Silva (2013), os autores apresentam a RSL-IL, uma linguagem para especificação de requisitos que tenta com um conjunto mínimo de **parâmetros** fornecer algum tipo de formalização para os mesmos. A motivação para isso, segundo os autores, se dá pelo fato de mesmo a linguagem natural sendo a mais utilizada para a escrita de requisitos, ela possui uma ambiguidade intrínseca que acaba diminuindo a qualidade e clareza dos mesmos. Sendo assim, o acréscimo de certo formalismo pode aumentar a qualidade destes requisitos.

3.3 Lições do Capítulo

Ao decorrer de nosso mapeamento tivemos nossa motivação sendo cada vez mais reforçada, com vários dos artigos encontrados apresentando pontos que corroboram com a mesma, reforçando cada vez mais a importância da preocupação com as fases de ER no desenvolvimento de SAs. O mapeamento também nos proporcionou o estudo sobre diversas linguagens de especificação de diferentes formas, de modo que fosse possível uma análise de abordagens diversas para se lidar com requisitos de SAs.

Pudemos perceber abordagens mais focadas em questões adaptativas, como Tropos4AS e AML, por exemplo. Assim como vimos abordagens mais focadas em questões de contexto, como ContextUML e MLContext. Mas também foi possível notar um grande destaque da Relax, que como pode ser visto na Tabela 5, propõe uma solução para todas as características de sistemas autoadaptativas investigadas neste trabalho.

Além disso, como também podemos ver nesta tabela, o artigo possui uma das mais altas quantidades de citações entendendo-se assim, ser um trabalho que representa um

avanço considerável na área, tendo suas ideias replicadas em vários trabalhos durante seus sete anos.

4 PROJETO E DESENVOLVIMENTO DO PERFIL UML

Este capítulo apresenta a solução proposta, assim como as atividades desenvolvidas. A seção 4.1 apresenta os problemas encontrados no cenário atual de ER para SAs que motivam nosso trabalho. Na 4.2 é apresentada a maneira como a solução foi planejada. Na seção 4.3 é apresentada a maneira como a solução foi desenvolvida. Já a seção 4.4 apresenta as lições aprendidas do capítulo.

4.1 Visão Geral

Como apresentado por Whittle et al. (2010) e confirmado pela dificuldade de encontrar-se trabalhos relacionados ao tema, existe uma carência de ferramentas para dar suporte à criação e especificação de requisitos em SAs. Como vimos, a ER para o desenvolvimento de sistemas adaptativos necessita de adaptações quando comparada à aplicada a sistemas tradicionais, sendo necessário tratar aspectos como incerteza, contextos variados e propriedades adaptativas.

Alguns autores têm tentado mudar este cenário, como em Luckey et al. (2011) onde é proposto o *Adapt Case*, uma linguagem de modelagem de domínio específico. Para isso, é usado e refinado o conceito de casos de uso UML, para se adaptar ao propósito do trabalho, que é a especificação explícita e formal da adaptividade e diferentes contextos em que são envoltos os SAs.

Segundo os autores, a principal motivação para a realização deste trabalho é o fato de que mesmo existindo tentativas de se adaptar a ER para o desenvolvimento de SAs, por muitas vezes as peculiaridades de tais sistemas (como propriedades autoadaptativas e troca de contexto) acabam ficando perdidas entre as funcionalidades do sistema, não havendo uma separação muito clara entre elas. Segundo os autores, em seu trabalho foi apresentado como o conceito de *Adapt Case* pode ser utilizado para modelar a adaptvidade em diferentes níveis de abstração e como é fácil a distinção das causas e das reações das adaptações.

O trabalho de Luckey et al. (2011) realmente cumpre seu papel, apresentando uma maneira de separar aspectos próprios de SAs das funcionalidades do sistema, porém, o trabalho chama a atenção por apresentar várias formas de representação, como diagramas de caso de uso, diagramas de sequência e até mesmo trechos de código, mas em nenhum momento é representada uma forma textual para especificação dos requisitos.

Já Whittle et al. (2010) tem seu trabalho focado neste quesito, criando uma linguagem natural estruturada para a especificação de requisitos, que inclui operadores designados especificamente para capturar aspectos de incerteza. Para a demonstração de validação da RELAX em seu trabalho, os autores realizam um estudo de caso baseado em um documento de concepção disponibilizado pela *Fraunhofer IESE*[1], onde é possível

[1] Fraunhofer IESE é um instituto de engenharia experimental de software de uma reputação em nível mundial por seus métodos e processos baseados em evidências empíricas.

ver que a linguagem cumpre seu papel, lidando com os aspectos de incerteza pertencentes aos requisitos, porém, diferentemente do trabalho anterior, pode-se notar que os aspectos de diferentes contextos não é um dos focos principais, acabando por ficar em meio às funcionalidades do sistema. Em suas conclusões os autores observam que foram alcançados vários benefícios na utilização da RELAX, além de citar as diversas direções possíveis para trabalhos futuros, porém, nenhum tratando dos aspectos de troca de contexto.

4.2 Análise da Solução

Após a aplicação do mapeamento sistemático e leitura dos trabalhos, foi possível compreender qual o estado atual da especificação de requisitos para SAs, além da possibilidade de entrar em contato com diversas abordagens criadas por vários autores para lidar com este problema.O próximo passo foi abstrair das abordagens encontradas, quais seriam úteis para o objetivo de nosso trabalho.

Como visto ao final do mapeamento, quando o assunto é especificação de requisitos, a RELAX tem grande influência, tendo sua ideia replicada em vários trabalhos ao longo de seus 7 anos. A Relax é uma linguagem completa para especificação de requisitos de SAs, possuindo uma série de operadores definidos para lidar com os aspectos únicos deste tipo de sistema, sendo assim, foi escolhida para ser parte integrante de nossa solução, auxiliando na especificação dos requisitos.

A segunda grande colaboração para nosso trabalho vem da pesquisa de Benselim e Seridi-Bouchelaghem (2017), que propõe a criação de um profile UML para o domínio específico de sensibilidade de contexto, motivado pelo fato de a UML não possuir ferramentas para modelagem de contextos de uso. O autor baseia sua abordagem na extensão dos elementos UML já existentes, adicionando a eles uma semântica adicional, para que seja possível a representação de modelagem de contexto.

4.2.1 Esboço dos Modelos

Após a identificação de técnicas e abordagens que tinham a agregar a nosso trabalho, era necessário criar um esboço desta solução. Para isto, foram escolhidos requisitos para SAs, retirados do trabalho de Whittle et al. (2010) com o objetivo de criar modelos, buscando especificá-los da melhor forma possível, levando em conta aspectos de contexto e com a preocupação de que uma pessoa leiga no assunto (geralmente o cliente em um cenário de desenvolvimento de software) fosse capaz de interpretar os modelos e validá-los, o que é de grande importância na ER.

Os requisitos escolhidos para estes modelos iniciais estão listados na Tabela 6.

Para a criação destes esboços foi utilizada a ferramenta Astah, que nos possibilita a criação de diagramas dos mais variados tipos, incluindo os diagramas baseados em UML, além de possibilitar uma maneira de simular uma semântica extra, podendo ser

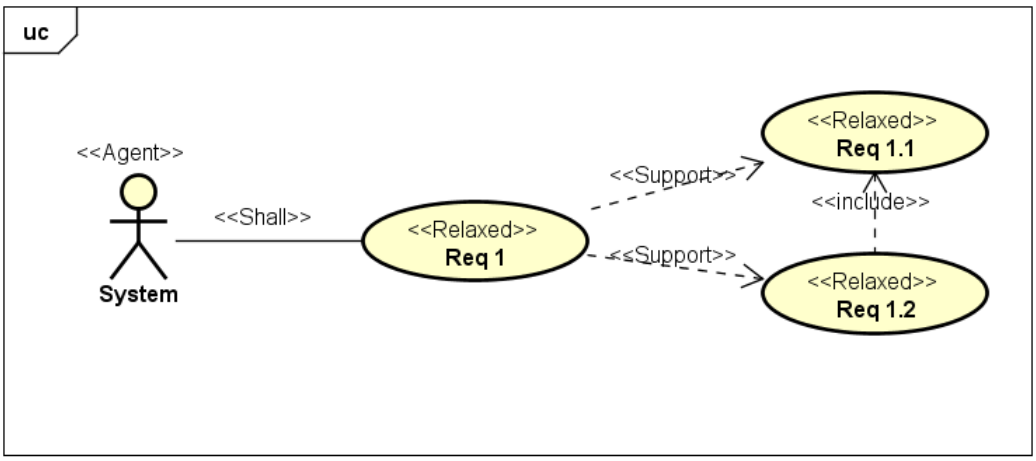
Tabela 6 – Requisitos selecionados para criação dos esboços iniciais da solução.

R1.1'	The fridge SHALL detect and communicate information with AS MANY food packages AS POSSIBLE.
	ENV: Food locations, food item information (type, calories) & food state (spoiled,unspoiled).
	MON: RFID readers; Cameras; Weight sensors.
	REL: RFID tags provide food locations/food information/ food state; Cameras provide food locations; Weight sensors provide food information (whether eaten or not).
R1.2'	The fridge SHALL suggest a dietplan with total calories AS CLOSE AS POSSIBLE TO the daily ideal calories. The fridge SHALL adjust the dietplan in linewith Mary's actual calorie consumption.
	ENV: Mary's daily calorie consumption.
	MON: RFID readers and weight sensors in fridge and trash can.
	REL: RFID readers and weight sensors provide consumed items; items vanish fromfridge and the items (if uneaten) or the packaging (if eaten) appears in trash can.

Fonte: (WHITTLE et al., 2010)

adicionados estereótipos nos elementos padrões da UML.

Figura 13 – Visão geral da solução.



Fonte: o próprio autor.

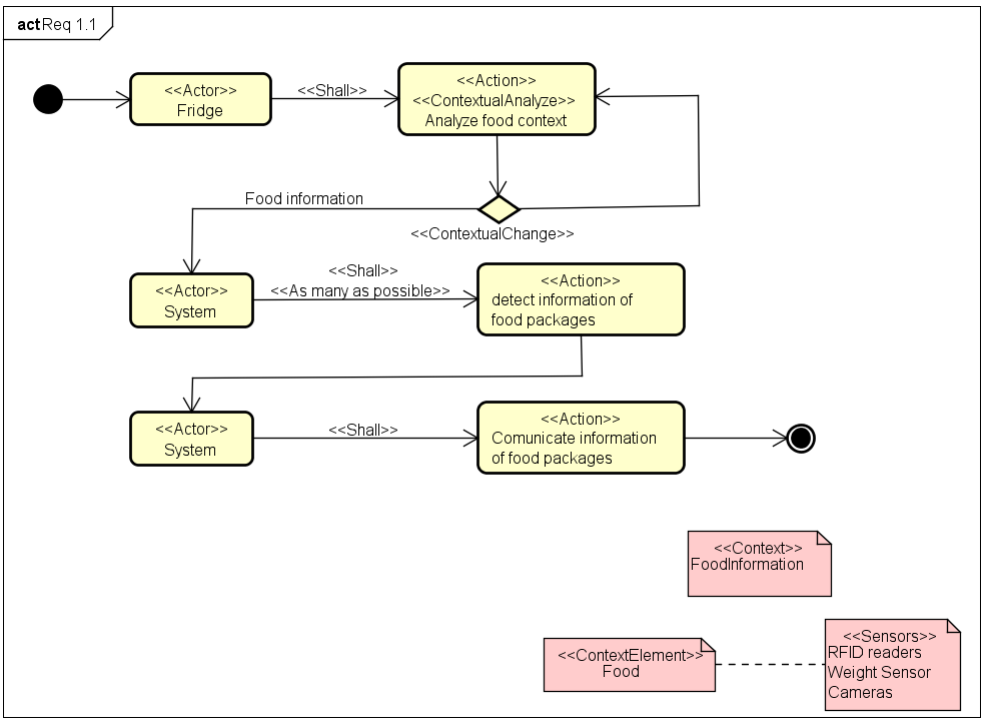
A solução esboçada é dividida em duas partes, a primeira sendo baseada em diagramas de caso de uso e a segunda em diagramas de atividades, de modo a descrever os cenários de cada requisito.

Na Figura 13, podemos ver o diagrama de casos de uso contendo os requisitos se-

lecionados. Ele foi pensado basicamente na ideia da UML padrão, onde um ator participa de um caso de uso, que pode incluir sub-requisitos. A contribuição aqui foi pensada em forma de estereótipos, que acrescentam valor semântico ao diagrama.

A segunda parte da solução esboçada trata da parte funcional do requisito e pode ser vista nas figuras 14 e 15, baseando-se nos operadores da Relax, assim como visto na linguagem RelaxML, e em conceitos retirados de Benselim e Seridi-Bouchelaghem (2017) e da RelaxML.

Figura 14 – Visão geral da solução.

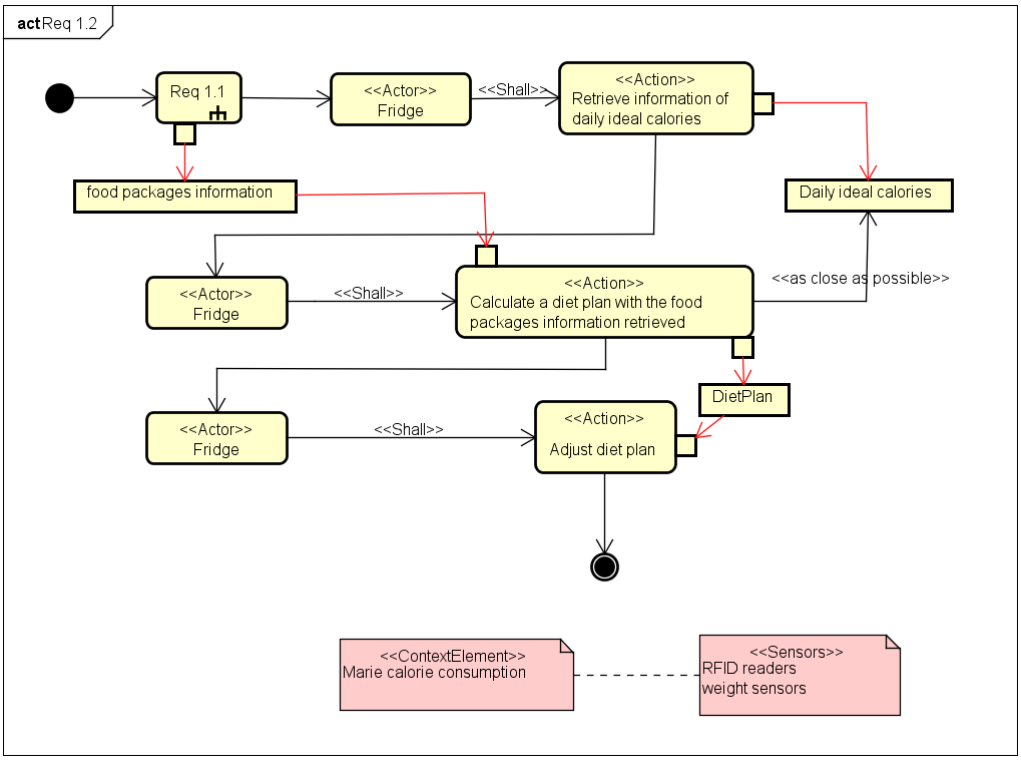


Fonte: o próprio autor.

4.3 Desenvolvimento da Solução

Context Requirement for Adaptive Systems (CR4AS) foi o nome escolhido temporariamente para nossa DSML para especificação de requisitos sensíveis a contexto. Após a solução ter sido pensada e esboçada, o próximo passo foi a construção da CR4AS, utilizando a ferramenta Papyrus para a criação de um Profile UML. O objetivo desta DSML é a especificação de requisitos para SAs levando em conta aspectos de contexto. Para isso, foram definidos um conjunto de estereótipos e regras OCL, baseados em abordagens estudadas. Este tipo de técnica escolhida para a criação da DSML nos permite utilizar o metamodelo já definido pela UML, permitindo o reuso de suas metaclasses e a utilização de ferramentas já existentes para desenvolver e aplicar a CR4AS.

Figura 15 – Visão geral da solução.



Fonte: o próprio autor.

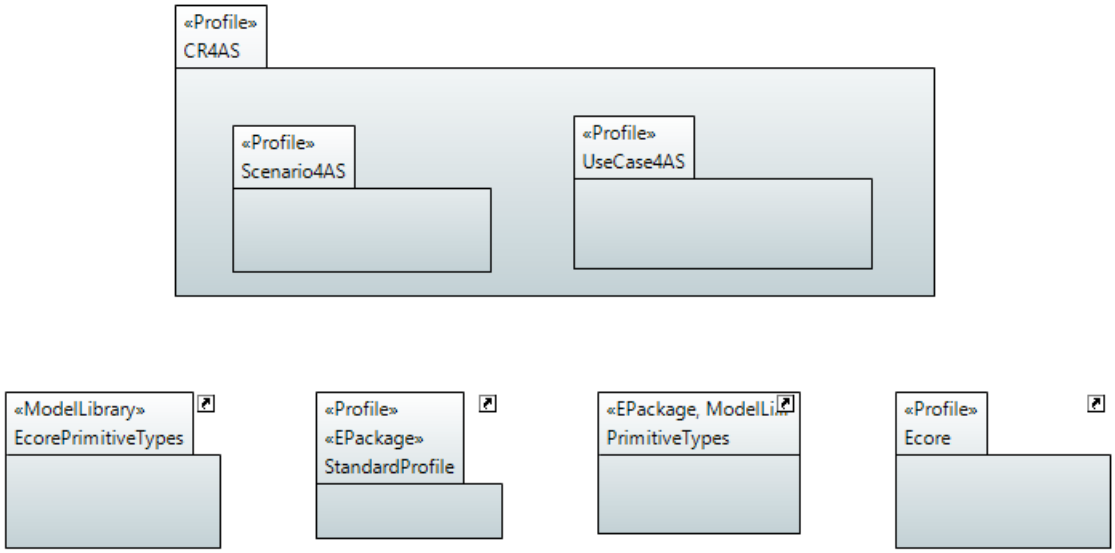
4.3.1 Estrutura

Seguindo a ideia aplicada por Benselim e Seridi-Bouchelaghem (2017) no desenvolvimento de seu profile, a CR4AS também possui seu profile dividido em sub-profiles, organizando seus estereótipos por tipo de diagrama, como pode ser visto na Figura 16. O sub-profile UseCase4AS contém os estereótipos extendidos de metaclasses utilizadas na modelagem de diagramas de caso de uso, como atores e casos de uso. Já o sub-profile Scenario4AS contém os estereótipos extendidos de metaclasses utilizadas na modelagem de diagramas de atividade, utilizados em nosso profile para criar a ideia de cenários de casos de uso.

4.3.2 Estereótipos

Como explicado anteriormente, em um UML perfil os estereótipos são a maneira utilizada para adicionar novos siginificados as metaclasses pertencentes ao meta-modelo da UML, nesta subseção apresentaremos os estereótipos criados, assim como a razão por trás do mesmo.

Figura 16 – Estrutura da CR4AS.



Fonte: o próprio autor.

Agents

Em SAs, devido à suas características, além do usuário do sistema poder ser um ator, o sistema tem a capacidade de ser ator de si próprio.

A UML dispõe de uma metaclasse chamada *Actor*, especificada como uma função desempenhada por um usuário ou algum outro sistema que interaja com o sistema modelado. Assim como nos trabalhos de Luckey et al. (2011) e Benselim e Seridi-Bouchelaghem (2017) optamos pela extensão desta metaclasse para a representação do nosso estereótipo *Agent* que é generalizado em:

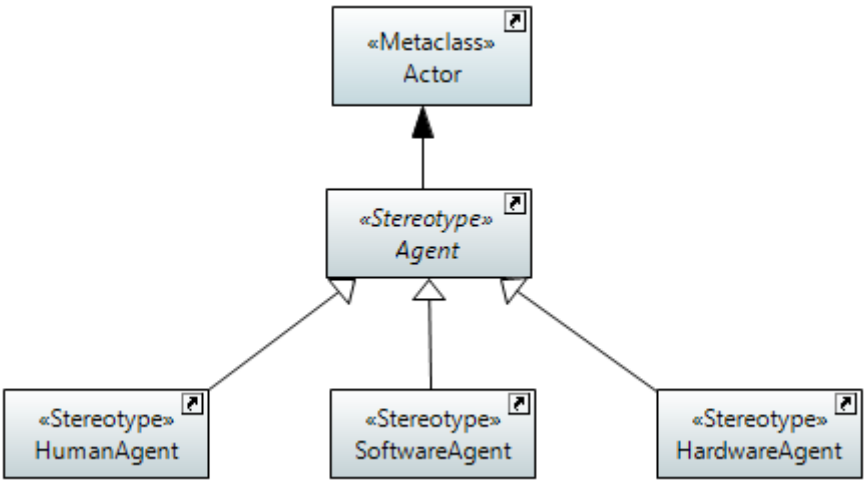
Atores são especificados na UML como uma função desempenhada por um usuário ou algum outro sistema que interaja com o assunto.Pensando nisso, optamos pela extensão da metaclasse *Actor* para adição deste significado ao diagrama, criando o estereótipo abstrato *Agent* que é generalizado em:

- ***HumamAgent***: Quando o ator trata-se de um usuário.
- ***SoftwareAgent***: Quando o ator trata-se do sistema.
- ***HardwareAgent***: Quando o ator trata-se de uma peça física.

Apesar desta metaclasse ser utilizada em outros trabalhos para essa finalidade, a UML a especifica como sendo uma função desempenhada por um usuário ou algum outro sistema que interaja com o sistema modelado. Essa definição pode ser interpretada como uma restrição significando que essa metaclasse não pode ser utilizada para esse fim. Tendo

esse problema em vista, planeja-se em trabalhos futuros a criação de uma nova classe, com as características da metaclasse *Actor* porém livre desta restrição.

Figura 17 – Hierarquia dos estereótipos extendidos da metaclasse *Actor*.



Fonte: o próprio autor.

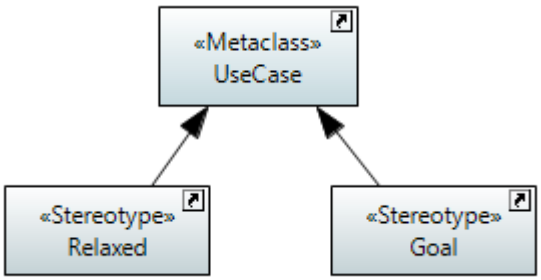
Goals* e *Relaxeds

Casos de uso são especificados na UML como sendo um conjunto de ações realizadas que produzem um resultado observável. Na CR4AS, optamos pela criação de dois estereótipos extendendo a metaclasse *UseCase* baseado em ideias retiradas do trabalho de Whittle et al. (2010), sendo eles:

- ***Relaxed***: Este termo surgiu do trabalho do Whittle, criado pelo mesmo, que refere-se a um requisito descrito na linguagem Relax. Na CR4AS este estereótipo indica que o caso de uso em questão está especificado em nossa adaptação visual para a linguagem Relax.

- ***Goal***: Este estereótipo surgiu da leitura de trabalhos e de experiência própria em desenvolvimento, onde vemos que algumas pessoas, incluindo o próprio Whittle em seu artigo, optam pela definição de um caso de uso que indica um objetivo mais geral do sistema, composto por casos de uso menores que dão suporte a este principal. Na CR4AS optou-se pela nomenclatura de *Goal*, que nada mais é do que um dos objetivos macro do sistema, que são realizados por um conjunto de casos de uso *relaxeds*.

Figura 18 – Hierarquia dos estereótipos extendidos da metaclasse *UseCase*.



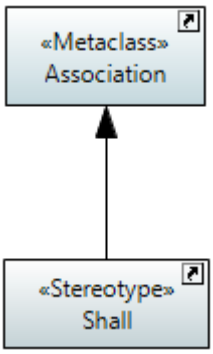
Fonte: o próprio autor.

Shall

Relações e associações também são passíveis de extensão em perfis UML. Na CR4AS para a criação do conceito de relação *Agent-Goal/Relaxed* optamos pela extensão da metaclasse *Association*. A extensão pode ser visualizada na Figura 19

- ***Association***: Na CR4AS indica o relacionamento *Agents-Goals/Relaxeds*, indicando que o ator é responsável por realizar o *Goal/Relaxed* correspondente.

Figura 19 – Estereótipo extendido da metaclasse *Association*.



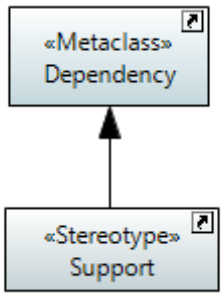
Fonte: o próprio autor.

Support

Na CR4AS para a criação do conceito de relação *Goal-Relaxeds* optamos pela extensão da metaclasse *Dependency*, especificada na UML como um relacionamento que significa que um ou mais elementos dependem de outros elementos para sua especificação ou implementação. A extensão pode ser visualizada na Figura 20

- ***Support***: Na CR4AS indica o relacionamento *Goal-Relaxeds*, indicando que um certo conjunto de casos de uso dão suporte a certo objetivo do sistema.

Figura 20 – Estereótipo extendido da metaclasse *Dependency*.



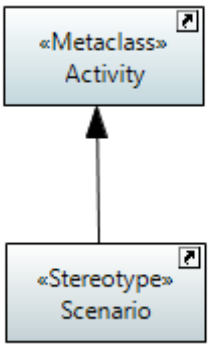
Fonte: o próprio autor.

Scenario

Activitys são especificadas na UML como sendo a especificação do comportamento parametrizado com o sequenciamento coordenado de unidades subordinadas. Na CR4AS optou-se pela extensão desta metaclasse para a representação dos cenários de um caso de uso, pois possui elementos capazes de representar mudanças de fluxo, sendo assim, é possível descrever os diversos fluxos alternativos de um caso de uso utilizando somente um diagrama. A extensão pode ser visualizada na Figura 21

- ***Scenario***: Na CR4AS é utilizada para representar os cenários de um caso de uso, é onde encontra-se a maior contribuição do trabalho. Possuindo um conjunto de estereótipos inspirados nos operadores RELAX, permitindo que um caso de uso possa ser especificado de maneira híbrida entre visual e textual, facilitando a interpretação dos membros envolvidos.

Figura 21 – Estereótipo extendido da metaclasse *Dependency*.



Fonte: o próprio autor.

Step

As *Activitys* apresentadas anteriormente possuem uma série de elementos para a descrição de atividades. Na CR4AS optamos pelas Actions para comportar as partes textuais de nossa especificação como pode ser visto na Figura 22.

- ***Step***: A CR4AS representa um conjunto de diferentes elementos utilizados para a especificação da parte textual do requisito, é uma classe abstrata que não pode ser instanciada.
- ***Agent***: Possui o mesmo significado do estereótipo *Agent* apresentado anteriormente, tem o objetivo de representar o ator responsável pelas ações.
- ***Action***: Na CR4AS este estereótipo é responsável por apresentar as ações executadas durante o requisito, descritas em linguagem natural.
- ***ContextualAnalyze***: É uma generalização do estereótipo anterior, representa uma ação em que é executada uma análise do contexto atual em que o software se encontra, atividade comum em sistemas auto-adaptativos que respondem a mudanças de contexto.
- ***Event***: Define um evento que pode ocorrer durante a execução do sistema e pode vir a interferir na execução de um ou mais requisitos.
- ***State***: Define um estado do contexto ou do sistema durante sua execução e pode vir a interferir na execução de um ou mais requisitos.
- ***Timer***: Como definido por Whittle et al. (2010), define um intervalo de tempo *t*, definido entre dois evento.É utilizado para indicar um intervalo de tempo onde certa ação deve ocorrer.

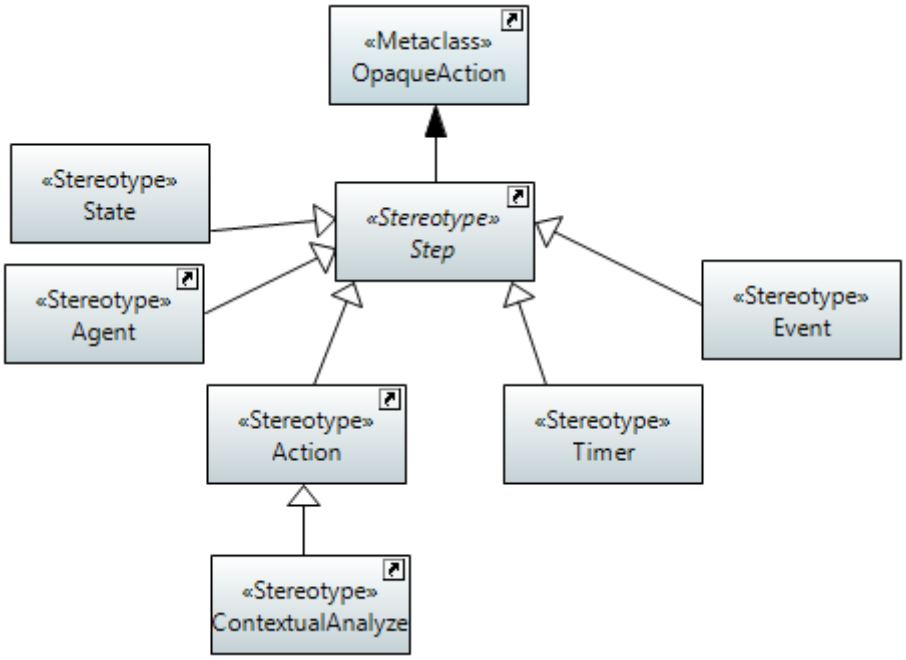
ContextualChange

Sistemas sensíveis a contexto tem a habilidade de reagir de diferentes formas, realizando ações diferentes baseado no contexto em que se encontra. Sendo assim, na CR4AS, optamos por representar esta habilidade estendendo a metaclasse *DecisionNode*, que é especificado na UML como um nó de controle, onde a partir de uma decisão pode-se partir para diferentes fluxos de atividade. A extensão pode ser visualizada na Figura 23

ContextualInformation

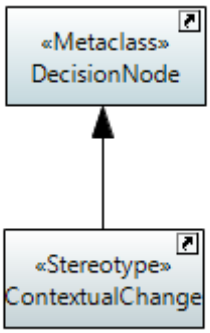
Como dito anteriormente, sistemas sensíveis a contexto reagem de forma diferente à diferentes condições do contexto em que se encontram, logo, ao decorrer de uma tarefa é comum que estas informações sofram alterações. Sendo assim, na CR4AS, optamos por

Figura 22 – Hierarquia de estereótipos extendidos da metaclasse *OpaqueAction*.



Fonte: o próprio autor.

Figura 23 – Estereótipo extendido da metaclasse *DecisionNode*.



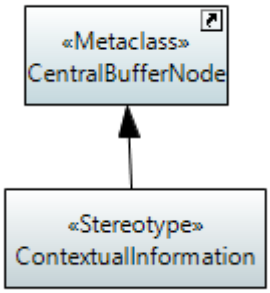
Fonte: o próprio autor.

representar esta habilidade estendendo a metaclasse *CentralBufferNode*, pois acreditamos que é importante ter um estereótipo identificando que trata-se de uma informação contextual para aumentar a expressividade do modelo. A extensão pode ser visualizada na Figura 24.

Operadores

A principal contribuição da Relax encontra-se em sua série de operadores que suportam aspectos de incerteza baseados em lógica *fuzzy* oferecendo mais flexibilidade sobre como e quando as funcionalidades serão satisfeitas. Na CR4AS optamos por separá-los

Figura 24 – Estereótipo extendido da metaclasse *CentralBufferNode*.



Fonte: o próprio autor.

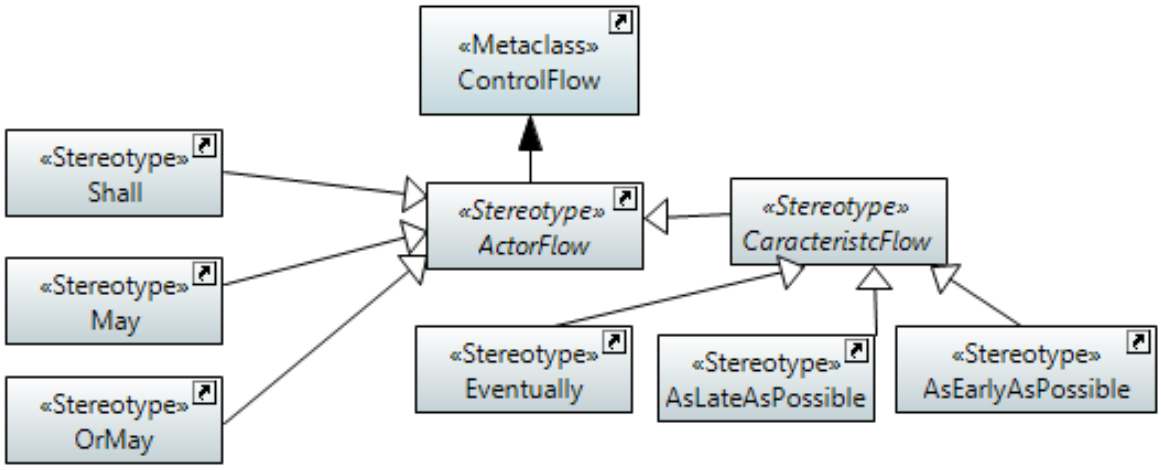
em uma hierarquia definida por suas características, extendendo da metaclasse *ControlFlow* os conjuntos de operadores divididos em *ActorFlow*, *TemporalFlow* e *OrdinalFlow* e o operador isolado *Until* que não se encontra em um conjunto por não possuir as características necessárias para nenhum deles. A hierarquia destes estereótipos pode ser vista nas figuras 25, 26, 27 e 28.

- ***ActorFlows***: É o conjunto de operadores que definem a condição pela qual uma ação será executada por um agente. Pode ser visto na Figura 25.
  - ***Shall***: Este operador define que uma ação deve ser executada por um ator. Whittle et al. (2010) define sua semântica como *ϕ é verdadeiro em qualquer estado* e sua formalização FBTL como: **AG**ϕ.
  - ***May orMay***: Estes dois operadores trabalham em conjunto definindo que uma de duas ações deve ser executada. Whittle define sua semântica como *em qualquer estado, ou ϕ_1 ou ϕ_2 são verdades* e sua formalização FBTL como: **AG** $(\phi_1 or \phi_2)$.
- ***CaracteristicFlow***: É o conjunto de operadores que definem características extras para as condições de um *ActorFlow*, eles complementam as características dos *ActorFlows* e devem ser utilizados em conjunto. Pode ser visto na Figura 25.
  - ***AsEarlyAsPossible***: Este operador define que uma ação deve ser realizada o mais cedo possível. Whittle define sua semântica como *ϕ torna-se verdadeiro em algum estado tão perto do tempo atual quanto possível* e sua formalização FBTL como: **A** $\chi_{>} =_d \phi$ 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.
  - ***AsLateAsPossible***: Ao contrário do anterior, este operador define que uma ação deve ser realizada o mais tarde possivel. Whittle define sua semântica

como ϕ *torna-se verdadeiro em algum estado tão próximo do tempo* $t = \infty$ *possível* e sua formalização FBTL como: **A** $\chi_{>} =_d \phi$ 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

– ***Eventually***: Este operador indica que a ação ocorre em algum momento no futuro, não tendo necessariamente um tempo certo para ocorrer. Whittle define sua semântica como ϕ *será verdade em algum estado futuro* e sua formalização FBTL como: **AF** ϕ

Figura 25 – Hierarquia de estereótipos extendidos da metaclasse *ControlFlow*(1).



Fonte: o próprio autor.

- ***TemporalFlow*** É o conjunto de operadores da RELAX que realizam funções dependentes da condição temporal. Pode ser visto na 26.

  – ***TimerDependent***: É um subconjunto de *TemporalFlow*, contém os operadores que são dependentes de um *Timer*.

  – ***EventDependent***: É um subconjunto de *TemporalFlow*, contém os operadores que são dependentes de *Events*.

- ***TimerDependent***

  – ***AsCloseAsPossibleTo<f>***: Este operador indica que a ação irá ocorrer em intervalos periódicos de tempo definidos por um *Timer*. Whittle define sua semântica como ϕ *é verdade em intervalos periódicos, onde o período é tão perto de f possível* e sua formalização FBTL como: **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

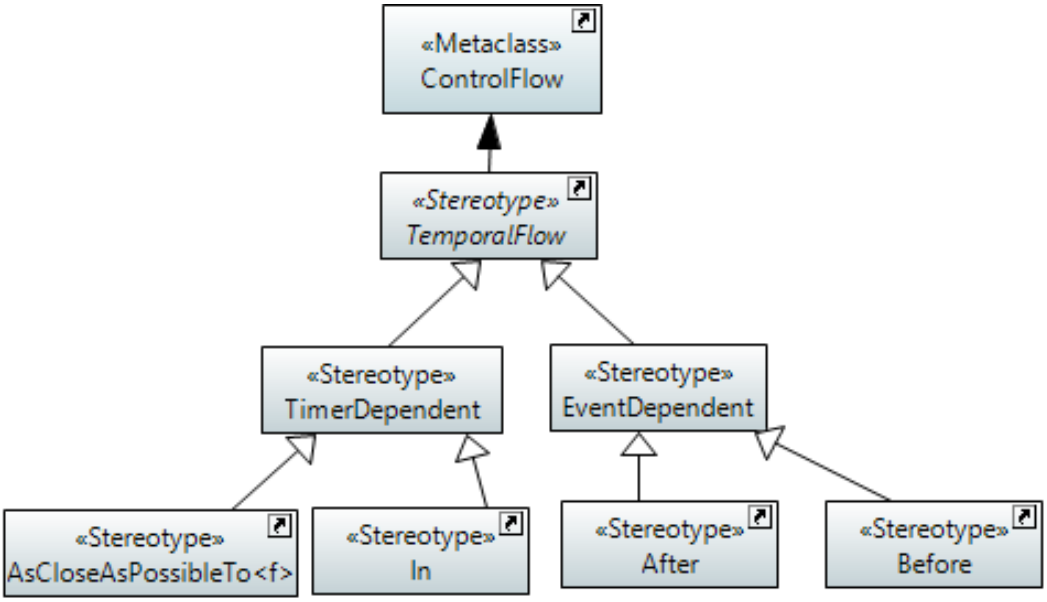
sua função de adesão tem o seu valor máximo, no período definido por f (ou seja, M(d)= M(2d)=...=1) e diminui continuamente para valores inferiores e superiores a d (e 2d, ...)

  - ***In***: Este operador indica que a ação ocorrerá dentro de um certo intervalo de tempo definido por um *Timer*. Whittle define sua semântica como ϕ *é verdade em todo o estado no intervalo de tempo* $\boldsymbol{t}$ e sua formalização FBTL como: ($AFTER\ t_start\ \phi$ and $BEFORE\ t_end\ \phi$) onde t_start, t_end são eventos que denotam o início e o fim do intervalo **t**, respectivamente

- ***EventDependent***

  - ***After***: Este operador indica que a ação ocorrerá após determinado evento ocorrer. Whittle define sua semântica como ϕ *é verdade em qualquer estado ocorrendo antes do evento* e sua formalização FBTL como: $\mathbf{A}\ \chi_{>}\ e_d\ \phi$
  - ***Before***: Este operador indica que a ação ocorrerá antes de determinado evento ocorrer. Whittle define sua semântica como ϕ *é verdade em qualquer estado ocorrendo antes do evento* $\boldsymbol{e}$ e sua formalização FBTL como: $\mathbf{A}\ \chi_{<}\ e_d$ é a duração até a próxima ocorrência de **e**

Figura 26 – Hierarquia de estereótipos extendidos da metaclasse *ControlFlow* (2).



Fonte: o próprio autor.

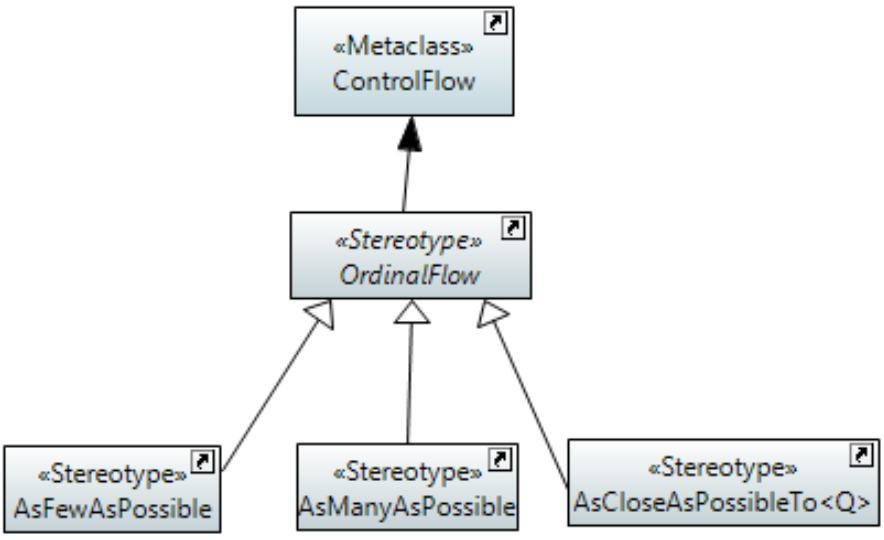
- ***OrdinalFlow***: É o conjunto de operadores da RELAX que realizam funções dependentes de quantificadores. Pode ser visto na Figura 27.

  - ***AsFewAsPossible***: Este operador indica que a ação possui uma função que é quantificável e que deve ser feita o menor possível. Whittle define sua semântica

como *existe alguma função* Δ *tal que* $\Delta(\phi)$ *é quantificáveis e é o mais próximo possível a 0* e sua formalização FBTL como: **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

- ***AsManyAsPossible***: Este operador indica que a ação possui uma função que é quantificável e que deve ser feita tanto quanto possível. Whittle define sua semântica como *existe alguma função* Δ *tal que* $\Delta(\phi)$ *é tão perto de* ∞ *possível* e sua formalização FBTL como: **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.
- ***AsCloseAsPossibleTo<q>***: Este operador indica que a ação possui uma função que é quantificável e que deve ser feita o mais próximo possível de um determinado valor. Whittle define sua semântica como *existe alguma função* Δ *tal que* $\Delta(\phi)$ *são quantificáveis e* $\Delta(\phi)$ *é tão próximo de 0 possível* e sua formalização FBTL como: **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.

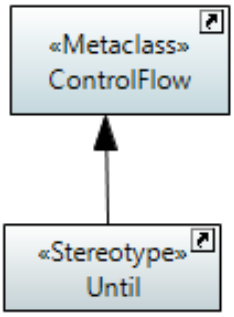
Figura 27 – Hierarquia de estereótipos extendidos da metaclasse *ControlFlow* (3).



Fonte: o próprio autor.

- ***Until***: Este operador indica que uma ação ocorre até alcançar um determinado estado. Whittle define sua semântica como: ϕ_1 *será verdadeiro até* ϕ_2 *tornar-se verdadeiro* e sua formalização FBTL como: $\mathbf{A}(\phi_1 \mu \phi_2)$.

Figura 28 – Hierarquia de estereótipos extendidos da metaclasse *ControlFlow* (4).



Fonte: o próprio autor.

Contexts

Entendemos que no tipo de requisito que visamos atingir, é importante aos envolvidos entenderem quais os aspectos de contexto estão envolvidos e podem interferir na execução do mesmo. Além disso, é importante que esta informação esteja bem visível ao usuário final. Sendo assim, optamos pela extensão da metaclasse *Comment* para servir como uma representação visual do contexto envolto no requisito, baseando-se nas abordagens de Whittle, Benselim e RelaxML, que é um perfil UML desenvolvido em um projeto em andamento do Laboratório de Engenharia de Software Aplicada (LESA) . A extensão pode ser vista na Figura 29.

- ***Enviroment***: Na CR4AS é onde ficam descritas quais as características de contexto estão relacionadas e podem interferir na execução do requisito que está sendo especificado.
- ***Monitor***: Na CR4AS é onde ficam descritos quais os sensores que são responsáveis por monitorar as características do contexto definidas no estereótipo anterior.
- ***Relationship***: Na CR4AS executa o papel de relacionar os estereótipos anteriores, definindo de que maneira cada sensor captura a característica do contexto.

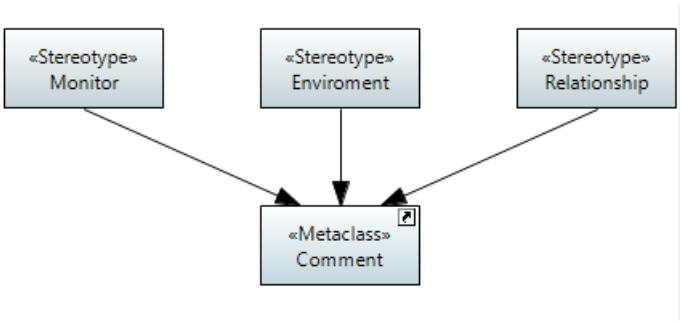
4.3.3 Restrições

Além dos estereótipos definidos anteriormente, perfis UML permitem a criação de restrições escritas em OCL que definem regras a serem seguidas, contribuindo em aspectos de qualidade, pois evita modelos sem sentido.

Shall

Como apresentado na subseção anterior,a relação *Shall* é utilizado na CR4AS para definir a relação *Agent-Goal/Relaxed*, definindo que usuário participada ação. Para que

Figura 29 – Estereótipos extendidos da metaclasse *Comment*.



Fonte: o próprio autor.

isto seja respeitado em nosso modelo foram criadas duas regras OCL, aplicadas a relação *Shall* para restringir a utilização do mesmo aos estereótipos corretos. As restrições OCL podem ser vistas a seguir.

Restrição *source*

```
1 self.base_Association.endType.getAppliedStereotypes()->exists(
    stereotype | stereotype.name = 'HumanAgent' or
2   stereotype.name = 'SoftwareAgent' or stereotype.name = '
      HardwareAgent'
3 )
```

Restrição 4.1 – Código OCL responsável por definir as características válidas para a entidade origem da relação *Shall*.

Restrição *target*

```
1 self.base_Association.endType.getAppliedStereotypes()->exists(
    stereotype | stereotype.name = 'Goal' or stereotype.name = '
    Relaxed')
```

Restrição 4.2 – Código OCL responsável por definir as características válidas para a entidade alvo da relação *Shall*.

Support

Como apresentado na subseção anterior, a relação *Support* é utilizado na CR4AS para definir a relação *Goal-Relaxed*, definindo que um certo caso de uso contribui para um certo objetivo. Para que isto seja respeitado em nosso foram criadas duas regras OCL, aplicadas a relação *Support* para restringir a utilização do mesmo aos estereótipos corretos. As restrições OCL podem ser vistas a seguir.

Restrição *source*

```
self.base_Dependency.client.getAppliedStereotypes() -> exists(
   stereotype | stereotype.name='Relaxed')
```

Restrição 4.3 – Código OCL responsável por definir as características válidas para a entidade origem da relação *Support*.

Restrição *target*

```
self.base_Dependency.supplier.getAppliedStereotypes() -> exists(
   stereotype | stereotype.name='Goal')
```

Restrição 4.4 – Código OCL responsável por definir as características válidas para a entidade destino da relação *Support*.

ActorFlow

O conjunto de operadores modais definem as diferentes ligações de *Agent-Action*, sendo assim, foram adicionadas duas regras OCL para que esta definição seja respeitada. As regras OCL podem ser vistas a seguir.

Restrição *source*

```
self.base_ControlFlow.source.getAppliedStereotypes() ->
 exists(stereotype | stereotype.name='Agent')
```

Restrição 4.5 – Código OCL responsável por definir as características válidas para a entidade origem das relações *ActorFlow*.

Restrição *target*

```
self.base_ControlFlow.target.getAppliedStereotypes() ->
 exists(stereotype | stereotype.name='Action' or stereotype.name=
    'ContextualAnalyze' )
```

Restrição 4.6 – Código OCL responsável por definir as características válidas para a entidade destino das relações *ActorFlow*.

TimerDependents

O conjunto de operadores *TimerDependents* definem as diferentes ligações de *Action-Timer*, sendo assim, foram adicionadas duas regras OCL para que esta definição seja respeitada. As regras OCL podem ser vistas a seguir.

Restrição *source*

```
self.base_ControlFlow.source.getAppliedStereotypes() ->
```

```
2  exists(stereotype | stereotype.name='Action' or stereotype.name=
       'ContextualAnalyze')
```

Restrição 4.7 – Código OCL responsável por definir as características válidas para a entidade origem das relações *TimerDependents*.

Restrição *target*

```
1 self.base_ControlFlow.target.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='Timer')
```

Restrição 4.8 – Código OCL responsável por definir as características válidas para a entidade destino das relações *TimerDependents*.

EventDependents

O conjunto de operadores *EventDependents* definem as diferentes ligações de *Action-Event*, sendo assim, foram adicionadas duas regras OCL para que esta definição seja respeitada. As regras OCL podem ser vistas a seguir.

Restrição *source*

```
1 self.base_ControlFlow.source.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='Action' or stereotype.name=
       'ContextualAnalyze')
```

Restrição 4.9 – Código OCL responsável por definir as características válidas para a entidade origem das relações EventDependents.

Restrição *target*

```
1 self.base_ControlFlow.target.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='Event')
```

Restrição 4.10 – Código OCL responsável por definir as características válidas para a entidade destino das relações *EventDependents*.

OrdinalFlows

O conjunto de operadores *OrdinalFlows* definem as diferentes ligações de *Action-Quantifier*, sendo assim, foram adicionadas duas regras OCL para que esta definição seja respeitada. As regras OCL podem ser vistas a seguir.

Restrição *source*

```
1 self.base_ControlFlow.source.getAppliedStereotypes() ->
```

```
2  exists(stereotype | stereotype.name='Action' or stereotype.name=
       'ContextualAnalyze')
```

Restrição 4.11 – Código OCL responsável por definir as características válidas para a entidade origem das relações *OrdinalFLow*.

Restrição *target*

```
1 self.base_ControlFlow.target.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='ContextualInformation')
```

Restrição 4.12 – Código OCL responsável por definir as características válidas para a entidade destino das relações *OrdinalFLow*.

Until

A relação *Until* define a ligação entre *Action-State*, sendo assim, foram adicionadas duas regras OCL para que esta definição seja respeitada. As regras OCL podem ser vistas a seguir.

Restrição *source*

```
1 self.base_ControlFlow.source.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='Action' or stereotype.name=
       'ContextualAnalyze')
```

Restrição 4.13 – Código OCL responsável por definir as características válidas para a entidade origem da relação *Until*.

Restrição *target*

```
1 self.base_ControlFlow.target.getAppliedStereotypes() ->
2  exists(stereotype | stereotype.name='State')
```

Restrição 4.14 – Código OCL responsável por definir as características válidas para a entidade destino da relação *Until*.

4.4 Lições do Capítulo

Durante o decorrer desse capítulo retomamos nossos objetivos, desenvolvemos nossa solução baseada nos conhecimentos adquiridos nas etapas anteriores e seguindo as orientações de Fuentes-Fernández e Vallecillo-Moreno (2004), desenvolvemos um perfil UML que dê suporte a abordagem criada. Um dos principais desafios foram as limitações de conhecimento sobre as tecnologias utilizadas, sendo necessário muita leitura e vários erros ao decorrer do desenvolvimento até que se chegasse a abordagem definitiva.

Ao final, foi desenvolvido um perfil UML com quarenta e um estereótipos, estendendo um total de 14 metaclasses do metamodelo UML, para que fossem representados

conceitos de diferentes abordagens, propostas por autores como Whittle et al. (2010), Benselim e Seridi-Bouchelaghem (2017), Luckey et al. (2011) e a RelaxML, desenvolvida pelo LESA.

5 VERIFICAÇÃO E VALIDAÇÃO DO PERFIL

Este capítulo tem como objetivo realizar a verificação e validação do perfil, buscando verificar se os objetivos deste trabalho foram alcançados. Para isto, optou-se pela especificação da lista de requisitos para SAs apresentados por Whittle et al. (2010) em seu trabalho. Na seção subseção 5.1.1 é apresentada a metodologia que Whittle et al. (2010) utilizou em seu trabalho para validação da RELAX e que será utilizada para validação de nosso perfil. Na subseção 5.1.2 são apresentados os passos para especificação dos requisitos apresentados anteriormente, utilizando nosso perfil.

5.1 Verificação

Nesta seção serão apresentadas as atividades desenvolvidas pelo próprio autor para verificação e teste do perfil, aplicando-o a especificação dos requisitos para SAs propostos por Whittle et al. (2010).

5.1.1 Exemplo de Aplicação da Linguagem RELAX

Whittle et al. (2010) apresenta em seu trabalho o seguinte contexto para o exemplo de aplicação:

"Maria é uma viúva. Ela tem 65 anos, sobrepeso, pressão arterial alta e níveis elevados de colesterol. Maria possui um moderno refrigerador inteligente que possui 4 sensores de temperatura e 2 sensores de umidade e é capaz de ler, armazenar e transmitir informações sobre RFID 8 (identificação por radiofrequência, do inglês, Radio-Frequency IDentification) de embalagens de alimentos. O refrigerador comunica-se e está integrado ao sistema Ambientede Vida Assistida (AAL - Ambient Assisted Living) existente na moradia. Em particular, o refrigerador detecta a presença de alimentos estragados e descobre e recebe um plano de dieta a ser monitorado com base em itens alimentares que Maria está consumindo.

Uma parte importante da dieta de Maria é garantir a ingestão mínima de líquidos. O refrigerador inteligente parcialmente contribui com esta dieta. Para melhorar a precisão, copos especiais ativados por sensores são utilizados: alguns têm sensores que emitem um sinal sonoro quando a ingestão de líquidos é necessária e têm um controle de nível para monitorar o líquido consumido; outros, adicionalmente, possuem um giroscópio para detectar derrames (perda de líquidos). Estes copos se comunicam e se coordenam com o intuito de estimar a quantidade de líquido ingerida: o último tipo de sensor informa o primeiro sobre os derrames e, assim, pode atualizar o nível de ingestão de líquidos. Contudo, Maria às vezes usa os copos para aguar as suas flores. Sensores nas torneiras e no vaso sanitário também fornecem um meio de monitorar esta medida."

Whittle descreve em seu trabalho a aplicação da RELAX como sendo a execução dos seguintes passos:

1. Extração dos requisitos do domínio do problema;

2. Identificação dos requisitos invariantes (*invariants*) e de requisitos que podem operar sobre fatores de incerteza (*relaxeds*).

3. Aplicação dos operadores RELAX nos requisitos *relaxeds*

Para o domínio apresentado anteriormente, Whittle define o seguinte requisito de alto nível:

- **R1** - The system SHALL monitor Marys health and SHALL notify emergency services in case of emergency.

Este requisito de alto nível é composto por uma série de sub-requisitos, que são apresentados na tabela 7.

Tabela 7 – Requisitos propostos por Whittle et al. (2010)

R1.1'	The fridge SHALL detect and communicate with food packages.
R1.2'	The fridge SHALL monitor and adjust the dietplan.
R1.3'	The system SHALL ensure a minimum of liquid intake.
R1.4'	The system SHALL minimize energy consumption during normal operation.
R1.5'	The system SHALL raise an alarm if no activity by Mary is detected for t.b.d.[1]to be defined} hours during normal waking hours.
R1.6'	The system SHALL minimize latency when an alarm has been raised.

Após a identificação dos requisitos, passa-se para o próximo passo que é "Relax-ar"[2] os requisitos. Como resultado, obteve-se os seguintes requisitos apresentados na Tabela 8.

[2] palavra utilizada pro whittle para indicar o ato de adicionar os operadores da RELAX a um requisito.

Tabela 8 – Requisitos Relax-ados apresentados por Whittle et al. (2010).

R1.1'	The fridge SHALL detect and communicate information with AS MANY food packages AS POSSIBLE.
	ENV: Food locations, food item information (type, calories) & food state (spoiled,unspoiled).
	MON: RFID readers; Cameras; Weight sensors.
	REL: RFID tags provide food locations/food information/ food state; Cameras provide food locations; Weight sensors provide food information (whether eaten or not).
R1.2'	The fridge SHALL suggest a dietplan with total calories AS CLOSE AS POSSIBLE TO the daily ideal calories. The fridge SHALL adjust the dietplan in linewith Mary's actual calorie consumption.
	ENV: Mary's daily calorie consumption.
	MON: RFID readers and weight sensors in fridge and trash can.
	REL: RFID readers and weight sensors provide consumed items; items vanish fromfridge and the items (if uneaten) or the packaging (if eaten) appears in trash can.
R1.3'	The fridge **SHALL** detect and communicate information with **AS MANY** food packages **AS POSSIBLE**.
	ENV: Food locations, food item information (type, calories) and food state (spoiled, unspoiled)
	MON: RFID readers; Cameras; Weight sensors.
	REL: RFID tags provide food locations/food information/food state; Cameras provide food locations; Weight sensors provide food information (whether eaten or not).
R1.4'	The fridge **SHALL** suggest a dietplan with total calories **AS CLOSE AS POSSIBLE TO** the daily ideal calories. The fridge SHALL adjust the dietplan in line with Marys actual calorie consumption.
	ENV: Marys daily calorie consumption.
	MON: RFID readers and weight sensors in fridge and trash can.
	REL: RFID readers and weight sensors provideconsumed items; items vanish from fridge and the items (ifuneaten) or the packaging (if eaten) appears in trash can.

Como percebido, os requisitos **R1.5'** e **R1.6'** não foram relax-ados, pois foram identificados como invariantes. Aqui encerramos a apresentação dos requisitos e do processo de "relax-amento"dos mesmos, agora apresentaremos na sub-seção a seguir um passo-a-passo de utilização de nosso perfil para especificação dos requisitos apresentados até então.

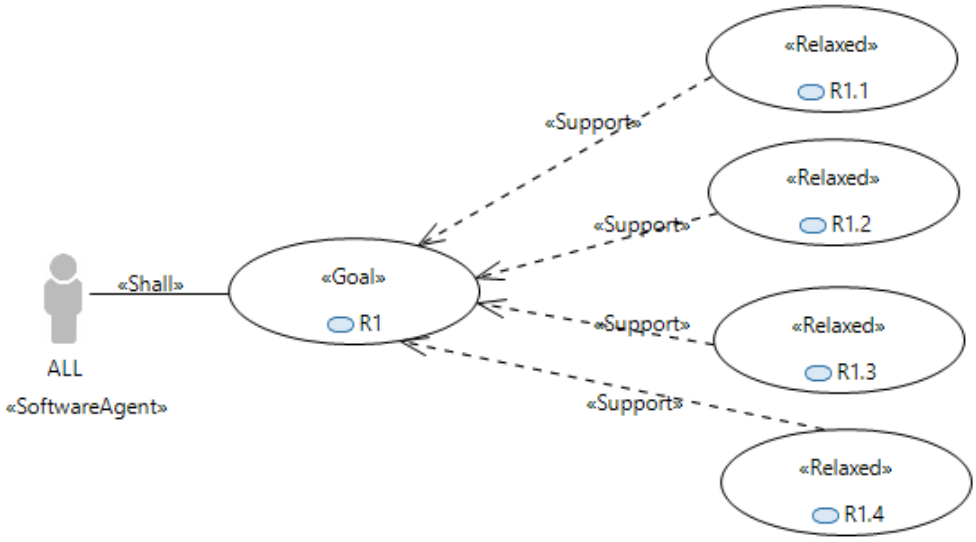
5.1.2 Especificação dos requisitos com a CR4AS

Para fins de teste da CR4AS, escolhemos a utilização do *plugin* Papyrus[3] do IDE Eclipse, que nos permite utilizar um perfil UML próprio para realização da modelagem UML.

O processo para especificação dos requisito com a CR4AS utilizando a ferramenta Papyrus é descrito passo-a-passo no Apêndice A. Através deste passo-a-passo obtivemos as modelagens que podem ser vistas nas figuras 30 e 31.

A Figura 30 refere-se a modelagem dos casos de uso **R1** que foi modelado como um objetivo do sistema e seus requisitos **R1.1'**, **R1.2'**, **R1.3'** e **R1.4'** definidos como requisitos *Relaxed's* que dão suporte ao objetivo **R1**.

Figura 30 – Diagrama de Casos de Uso CR4AS.



Na Figura 31 podemos ver a primeira versão do requisito especificada para testes utilizando a CR4AS, onde espera-se que devido a expressividade da CR4AS, possa ser visualizada e interpretada de forma fácil pelas partes interessadas ao desenvolvimento do sistema.

Na Figura 32 podemos visualizar o mesmo requisito especificado anteriormente, porém, com uma versão mais atual da CR4AS, onde um erro descoberto na validação com usuários foi corrigido.

Assim como as figuras anteriores, as figuras 33, 34, 35 representam os requisitos propostos restantes, especificados da maneira definida em A.

O restante dos requisitos foram modelados seguindo a mesma premissa e podem ser vistos nas figuras de 32 a 35.

[3] https://eclipse.org/papyrus/

Figura 31 – R1.1 especificado utilizando CR4AS.

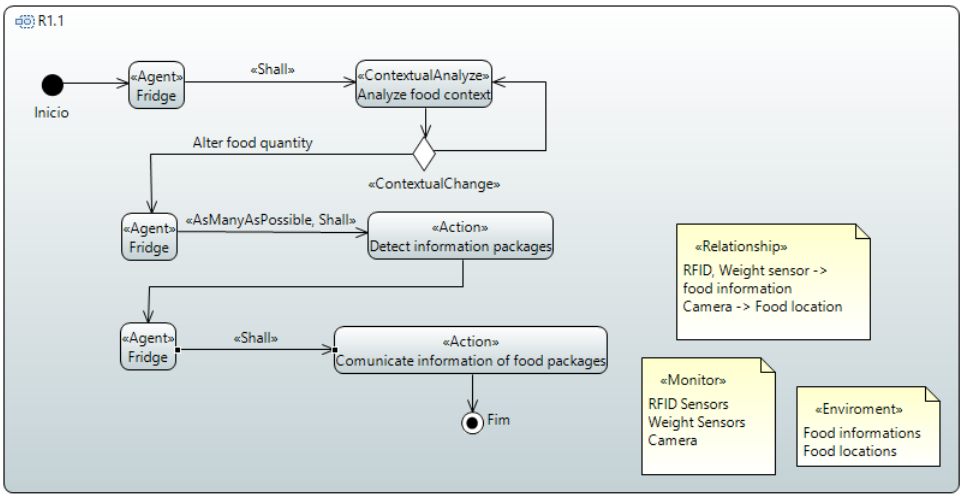
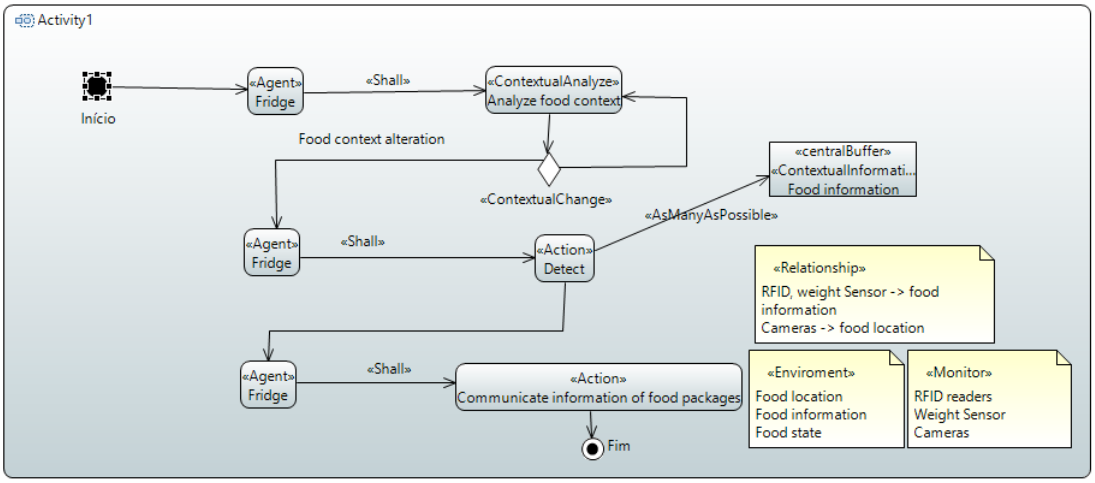


Figura 32 – R1.1 re-especificado após correção da CR4AS descoberta no teste com usuários.



5.2 Validação com Usuários

Nesta seção apresentamos a metodologia para validação da CR4AS.

5.2.1 Estratégia de Validação

Após a aplicação da CR4AS apresentada na 5.1.2, sentiu-se a necessidade de expô-la à usuários reais para que fosse possível coletar dados sobre a utilização da mesma, como sua expressividade, possíveis melhorias, problemas, entre outras informações.

Para isso definiu-se uma estratégia para validação que pode ser vista na Figura 36.

1. A primeira etapa foi a especificação dos requisitos propostos por Whittle et al. (2010), apresentada em detalhes na subseção 5.1.1.

2. A partir desta aplicação, criou-se um tutorial para utilização da CR4AS por meio da ferramenta Papyrus, que pode ser visto no Apêndice A.

Figura 33 – R1.2 especificado utilizando CR4AS.

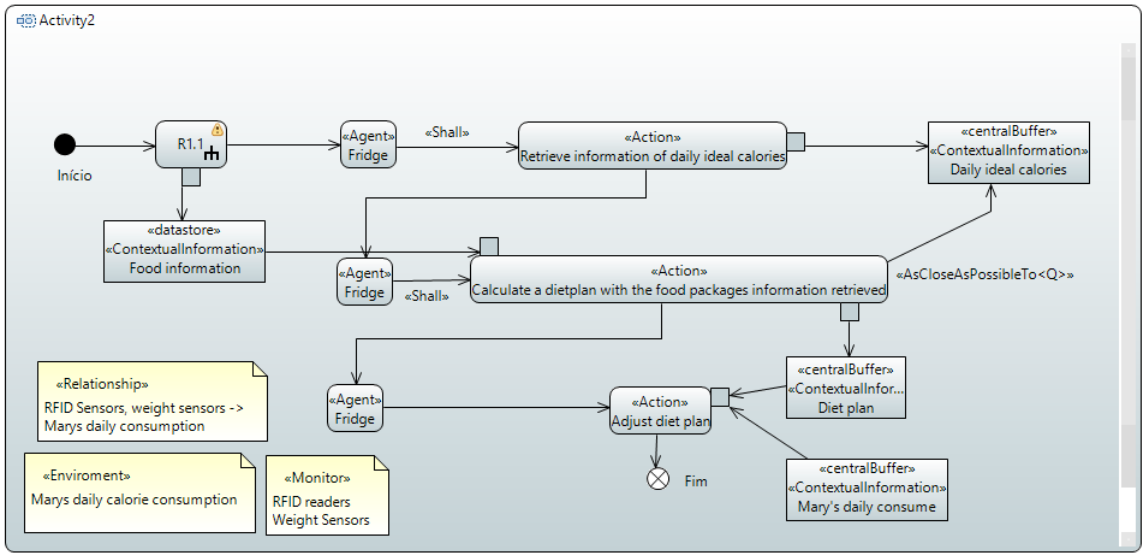
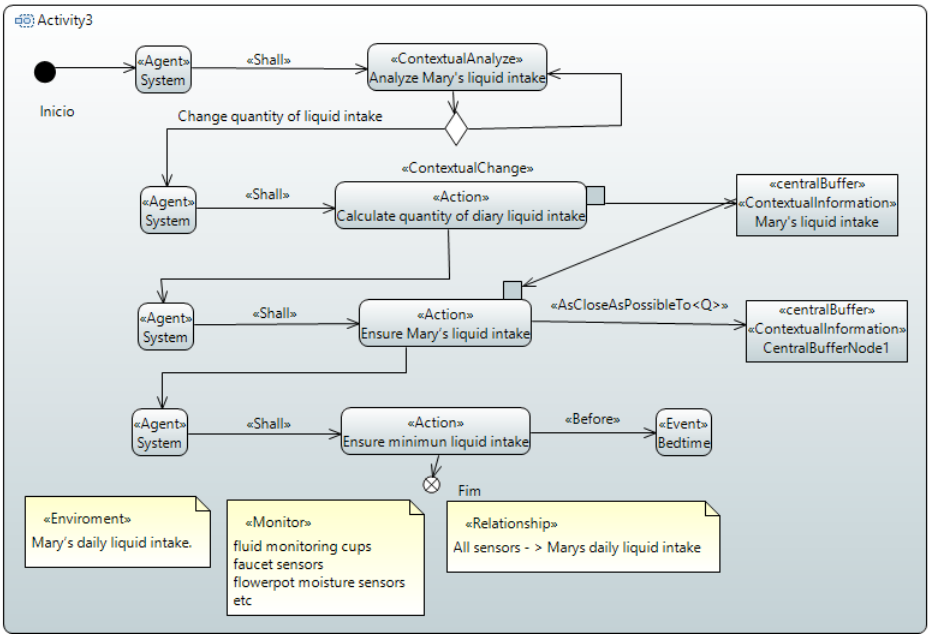


Figura 34 – R1.3 especificado utilizando CR4AS.



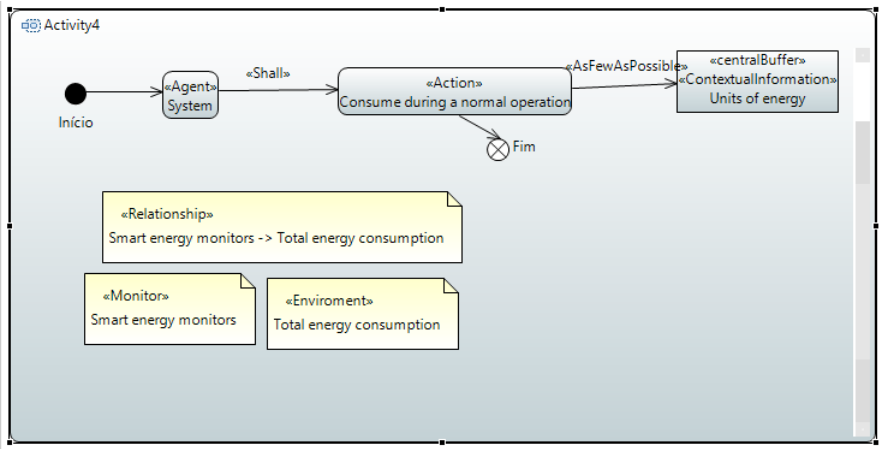
3. Pensando na coleta de informações e opiniões sobre a aplicação da CR4AS, foi desenvolvido um questionário online, criado por meio do Google Docs[4].

4. Após, foram selecionados e convidados a participar da validação usuários discentes do curso de Engenharia de Software, por possuírem familiaridade com técnicas para especificação de requisitos.

5. A próxima etapa foi pré-configurar o ambiente para que os usuários fossem capazes de realizar o teste com o máximo de autonomia, sem interferência do aplicador. Foi criada uma pasta contendo: o requisito **R1.1'** proposto por Whittle et al. (2010);

[4] https://www.google.com/docs/about/

Figura 35 – R1.4 especificado utilizando CR4AS.

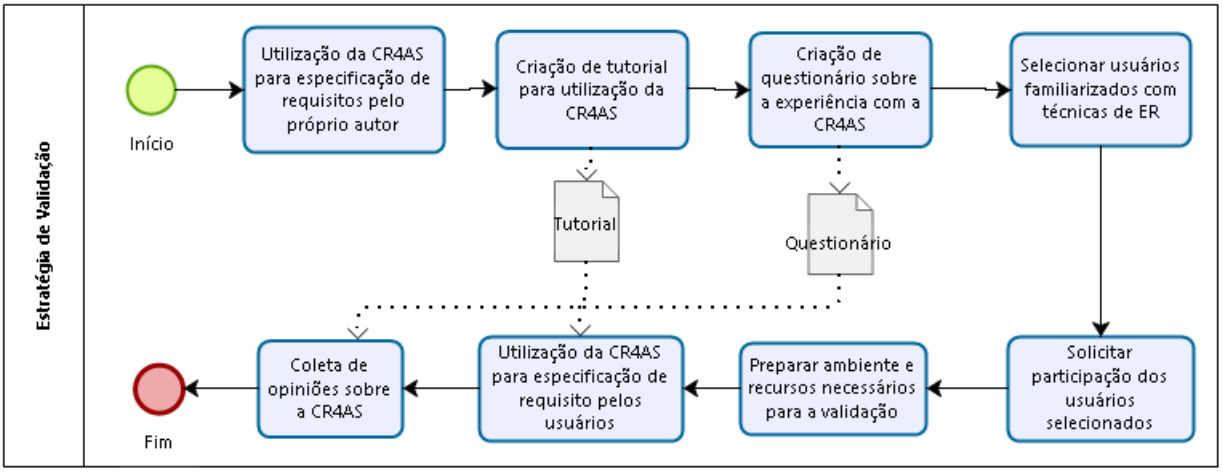


uma versão executável da ferramenta Papyrus; o perfil CR4AS e o tutorial criado no passo anterior.

6. Execução da validação descrita em detalhes na subseção 5.2.2.

7. Coleta e análise das opiniões capturadas no questionário.

Figura 36 – Estratégia utilizada para validação da CR4AS.



Fonte: o próprio autor.

5.2.2 Execução da Validação

A execução da validação da CR4AS por meio dos usuários selecionados ocorreu no dia 14 de junho de 2017, das 15:30h até 18:30h. A execução foi realizada em um dos laboratórios de informática na Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete, cedida pela mesma para realização da validação.

A metodologia aplicada foi a preparação do ambiente em computadores *Desktop*, parte do acervo pertencente ao laboratório de informática, para os usuários sem computador próprio. Já para os usuários que trouxeram seus próprios computadores foi entregue um *pen drive* contendo os artefatos necessários para a execução da validação.

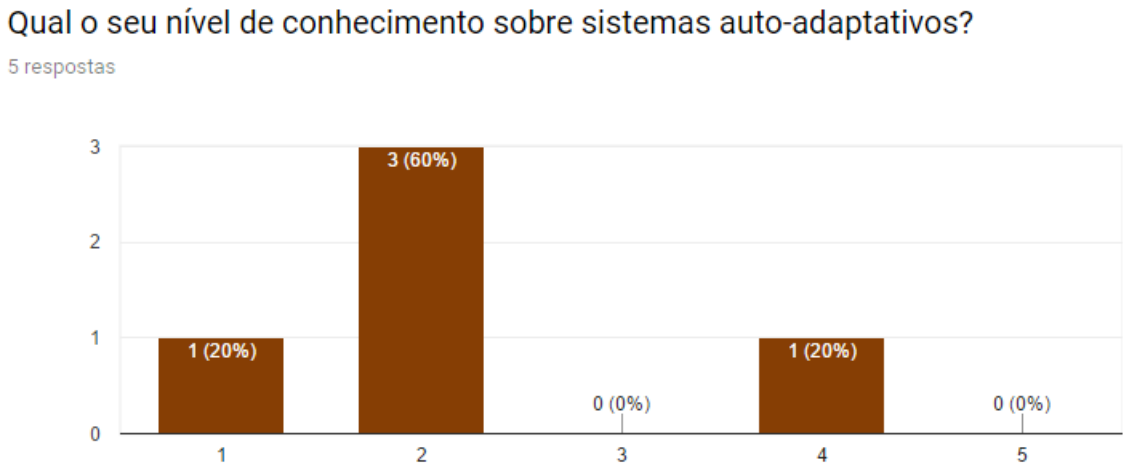
Além do material disponibilizado, os usuários tiveram em toda a totalidade do tempo o auxílio do aplicador (o próprio autor deste trabalho) disponível para a resposta de quaisquer dúvidas sobre a aplicação.

Ao concluir a execução da validação, foi disponibilizado para cada um dos cinco participantes um link contendo o formulário para coleta de opiniões sobre a CR4AS. As opiniões coletadas são analisadas em detalhes na subseção 5.2.3.

5.2.3 Resultado da Validação

A primeira questão do questionário, assim como as respostas coletadas podem ser vistas na Figura 37. Esta questão tem o intuito de analisar qual o grau de familiaridade dos participantes com SAs, baseado em uma escala de 1 a 5. Podemos perceber por meio das respostas que os participantes possuíam pouco conhecimento sobre este tipo de sistema, o que está longe de ser um cenário perfeito, mas se dá ao fato de haverem poucas pessoas com este tipo de conhecimento ao nosso alcance.

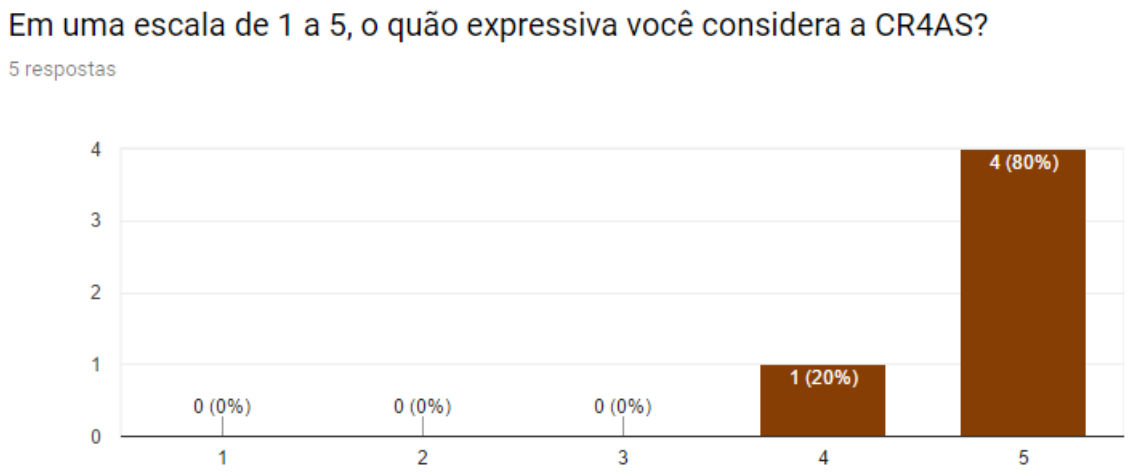
Figura 37 – Questão 1 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A segunda questão busca capturar o grau de expressividade percebido pelos participantes na CR4AS. Obtivemos respostas muito positivas nessa questão com os usuários percebendo um grau elevado de expressividade em nossa linguagem. Entendemos que este fator é um grande influenciador na aceitação da abordagem, sendo possível compreender a maneira como o requisito é executado apenas visualizando sua especificação no diagrama.

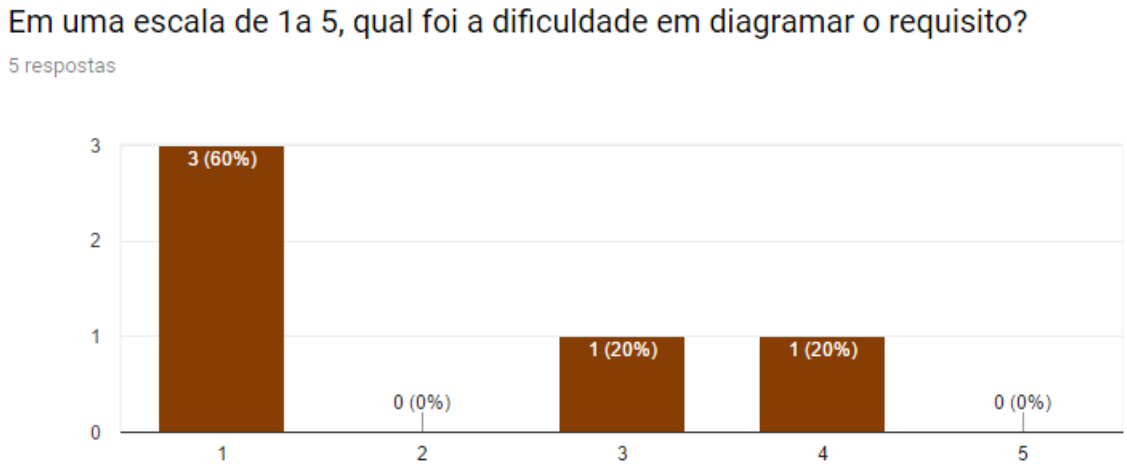
Figura 38 – Questão 2 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A terceira questão busca capturar o grau de dificuldade dos participantes em diagramar o requisito. Aqui também obtivemos respostas positivas, com a maior parte dos participantes não encontrando dificuldades para realizar a diagramação. No entanto podemos perceber que um dos participantes classificou sua dificuldade como grau 3 e outro como grau 4, o que indica que nem todos os participantes encontram facilidade ao se adaptar rapidamente a abordagem.

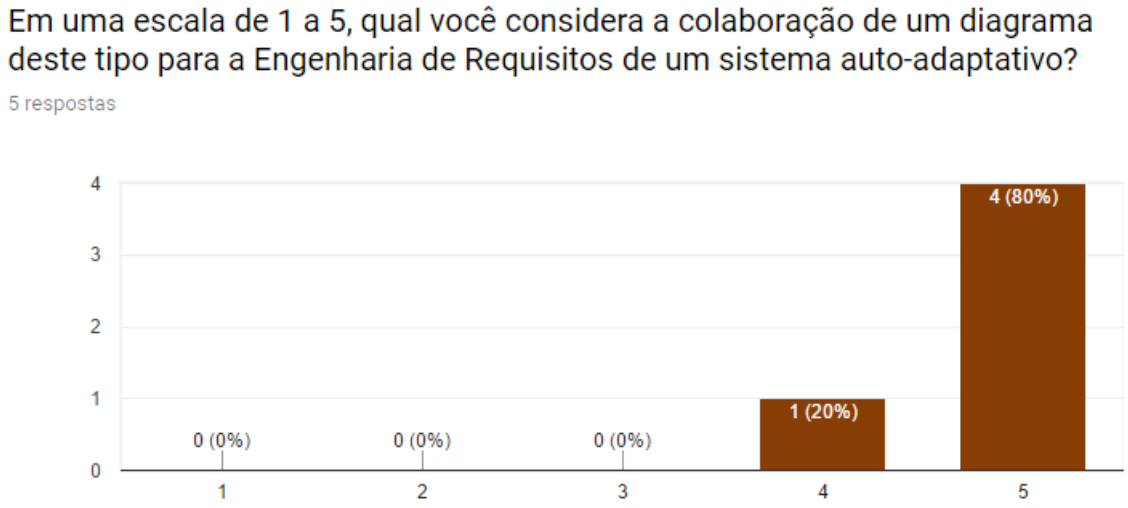
Figura 39 – Questão 3 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A quarta questão busca capturar o grau de colaboração deste tipo de abordagem para especificação de requisitos para SAs. Nesta questão também obtivemos um retorno positivo, porém vale analisar que esta questão é diretamente impactada pela questão 1, onde poucos usuários demonstraram conhecimento sobre o assunto.

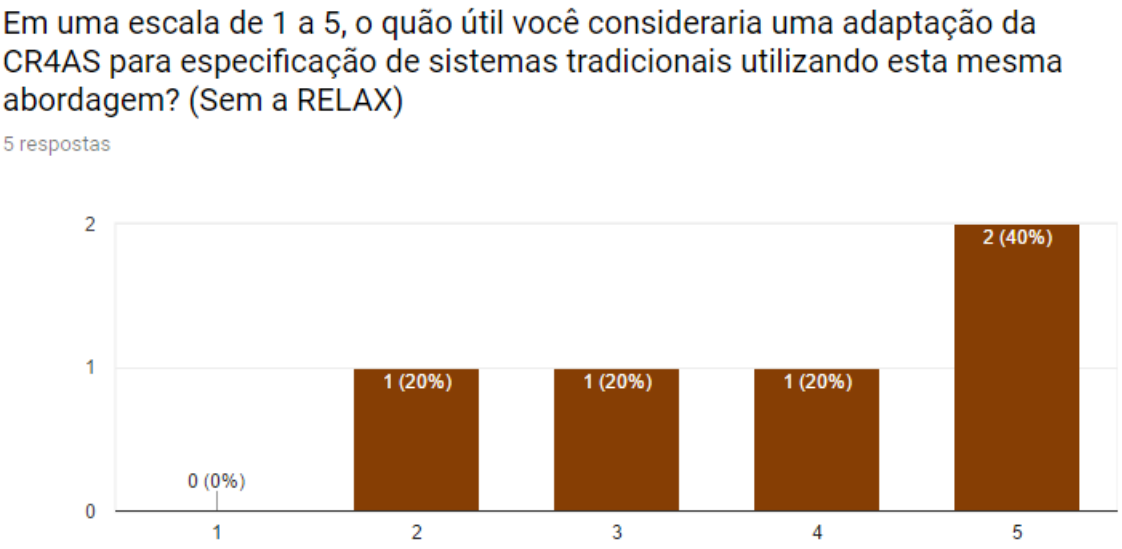
Figura 40 – Questão 4 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A quinta questão busca capturar o grau de colaboração que os participantes acreditam que este tipo de abordagem (sem as características de SAs) traria. Esta pergunta está relacionada a curiosidade do autor para ser aplicada em possíveis trabalhos futuros.

Figura 41 – Questão 5 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

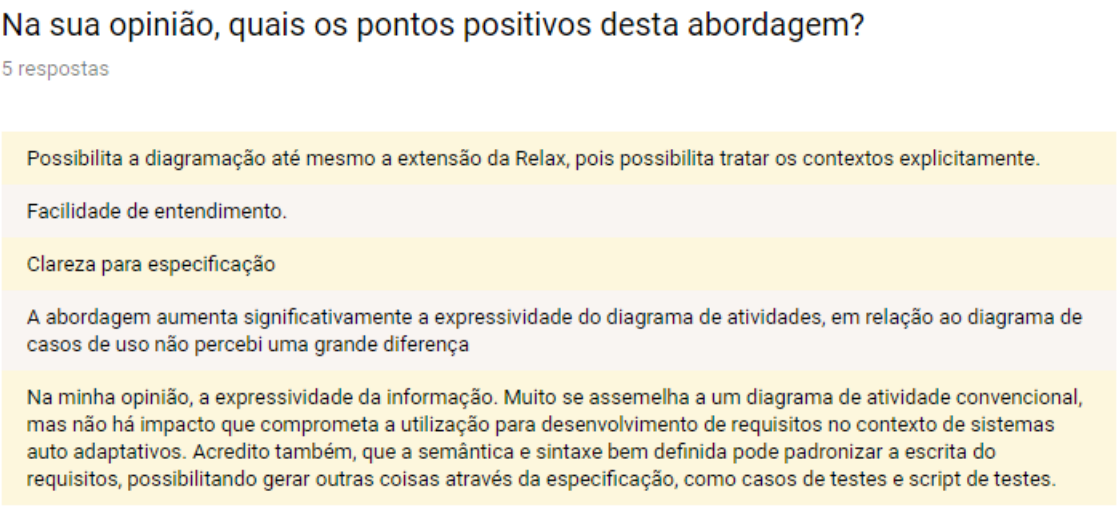
A sexta questão busca coletar o grau de aceitação da abordagem pelos participantes. As respostas mostraram-se positivas, com todos os participantes considerando vantajosa a utilização desta abordagem, principalmente em requisitos críticos do sistema.

Figura 42 – Questão 6 do questionário para coleta de opiniões sobre a CR4AS.

Fonte: o próprio autor.

A sétima questão busca coletar os pontos positivos observados pelos participantes na utilização da abordagem. Como podemos perceber pelas respostas, a maior colaboração apontada pelos participantes é a expressividade do modelo criado, sendo de fácil visualização e compreensão pelas partes interessadas. Podendo ser utilizado no desenvolvimento de sistemas como validação dos requisitos do sistema pelo cliente, o que é uma das funções dos artefatos gerados durante a ER, como definido por Sommerville (2011).

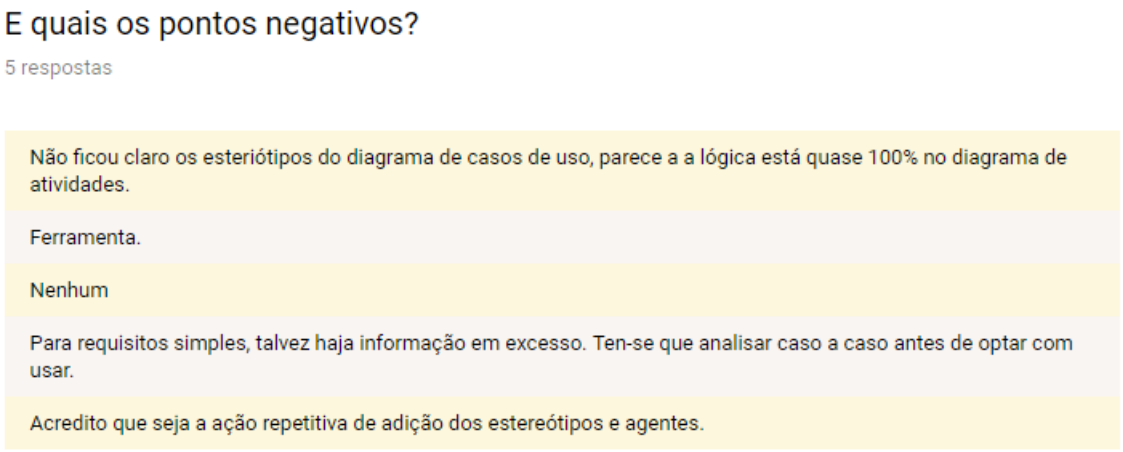
Figura 43 – Questão 7 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A oitava questão busca compreender os pontos negativos da CR4AS para planejamento de melhorias. Como podemos perceber nas respostas coletadas, 60% delas refere-se à questões de usabilidade ligadas a ferramenta Papyrus utilizada para os testes. A ferramenta foi escolhida somente para teste do perfil UML, pois como já era planejado e foi reforçado pela opinião dos participantes, é necessário a construção de uma ferramenta própria para que as questões de usabilidade dos usuários seja atendida.

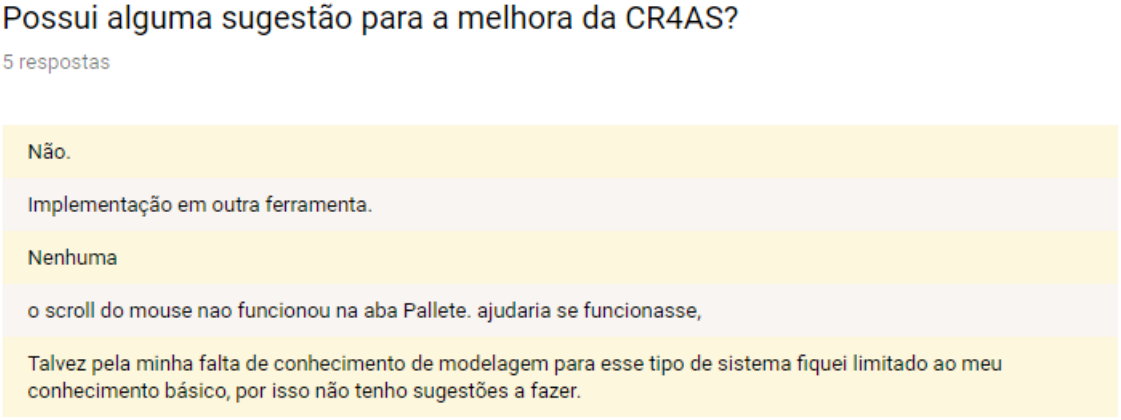
Figura 44 – Questão 8 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

A nona questão busca receber sugestões para a melhoria da abordagem. Como citado na questão anterior, fica reforçado novamente a necessidade de construção de ferramenta própria para utilização da abordagem.

Figura 45 – Questão 9 do questionário para coleta de opiniões sobre a CR4AS.



Fonte: o próprio autor.

5.3 Lições do Capítulo

Através das opiniões coletadas com o auxílio do questionário podemos perceber que o foco principal de nossa abordagem que é a expressividade do requisito especificado foi alcançado. Foi uma ação positiva, pois possibilitou ver como usuários familiarizados com técnicas de ER acharam a abordagem positiva. Ficou clara também a necessidade da construção de uma ferramenta própria para a abordagem aumentando consideravelmente aspectos de usabilidade, buscando sempre alcançar uma aceitação maior para a mesma.

6 CONSIDERAÇÕES FINAIS

O objetivo geral deste trabalho é a criação de uma abordagem para especificação de requisitos sensíveis ao contexto baseada em casos de uso para SAs, auxiliada por abordagens encontradas em trabalhos relacionados, buscando tornar auxiliar os engenheiros de requisitos a especificar este tipo de requisito de forma mais expressiva.

Consideramos que os objetivos do trabalho foram alcançados. Com o mapeamento sistemático foi possível compreender melhor o problema e encontrar autores tentando solucionar diferentes partes dele, seja propondo novas técnicas e abordagens ou adaptando as já existentes. Com o estudo foi possível desenvolver um perfil UML que acreditamos ser expressivo até para pessoas não familiarizadas com o assunto, sendo de fácil entendimento o que o requisito deve fazer. Isso fica mais evidente após teste com usuários que se mostrou positivo baseado nas respostas obtidas.

Vários desafios foram encontrados ao decorrer do trabalho. Um deles foi a baixa quantidade de requisitos para SAs encontrados e como nossa abordagem baseia-se em RELAX, este número é ainda mais limitado, fazendo com que seja difícil cobrir um grande número de possibilidades de modelagem.

Outra limitação do trabalho, que fica evidente durante o teste com usuário são as limitações do ambiente atual. Atualmente a CR4AS é executada diretamente no Papyrus, onde pode ser definida e aplicada a um projeto de modelagem, o que permite testar o perfil, adicionando estereótipos e testando as restrições definidas, porém, não é um ambiente produtivo, possui vários problemas que seriam resolvidos com a construção de uma ferramenta própria. Houveram tentativas de desenvolvimento para essa ferramenta, porém não é uma tarefa trivial, demonstrando ser necessário um tempo maior para que essa ideia seja aplicada.

Em nosso trabalho é proposta uma abordagem para modelagem de requisitos para SAs, que possuem características únicas que trazem desafios para a ER. Acreditamos que esta abordagem apoiada pelo perfil criado facilite a tarefa de especificar este tipo de requisito, tornando-o mais expressivo e de fácil entendimento, podendo ser utilizado para validação com clientes que não são familiarizados com esta área.

Para trabalhos futuros, pretende-se encontrar ou construir uma base maior de requisitos "Relax-ados"de modo a criar cada vez mais regras e tornar o CR4AS cada vez mais completo. Além da construção de uma ferramenta personalizada, visando aumentar a produtividade de nossa especificação.

REFERÊNCIAS

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: **Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing**. [S.l.]: Springer-Verlag, 1999. (HUC '99), p. 304–307. ISBN 3-540-66550-1. Citado na página 17.

ABRAN, A. et al. (Ed.). **Guide to the Software Engineering Body of Knowledge - SWEBOK**. Piscataway, NJ, USA: IEEE Press, 2001. ISBN 0769510000. Citado 2 vezes nas páginas 23 e 24.

AL-ALSHUHAI, A.; SIEWE, F. An extension of uml activity diagram to model the behaviour of context-aware systems. In: **2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing**. [S.l.: s.n.], 2015. p. 431–437. Citado 2 vezes nas páginas 35 e 37.

ASTESIANO, E. et al. Casl: the common algebraic specification language. **Theoretical Computer Science**, Elsevier, v. 286, n. 2, p. 153–196, 2002. Citado na página 39.

ATOMIQUE ATOS, C. D. Commissariat à l'Énergie. **Papyrus**. 2016. <http://www.eclipse.org/papyrus>. (Accessed on 13/06/2017). Citado na página 29.

BENSELIM, M.-S.; SERIDI-BOUCHELAGHEM, H. Towards a uml profile for context-awareness domain. **International Arab Journal of Information Technology**, v. 14, n. 2, 2017. Citado 6 vezes nas páginas 28, 44, 46, 47, 48 e 63.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. CAMPUS - RJ, 2006. ISBN 9788535217841. Disponível em: <https://books.google.com.br/books?id=ddWqxcDKGF8C>. Citado na página 28.

BOULIL, K.; BIMONTE, S.; PINET, F. Spatial olap integrity constraints: From uml-based specification to automatic implementation: Application to energetic data in agriculture. **Journal of Decision Systems**, v. 23, n. 4, p. 460–480, 2014. Citado na página 28.

CHEN, W.-K.; HILTUNEN, M.; SCHLICHTING, R. Constructing adaptive software in distributed systems. In: . [S.l.: s.n.], 2001. p. 635–643. Citado na página 21.

COUTAZ, J. et al. Context is key. **Communications of the ACM**, v. 48, n. 3, p. 49–53, 2005. Citado na página 17.

DEY, A. K. Understanding and using context. **Personal Ubiquitous Comput.**, Springer-Verlag, London, UK, UK, v. 5, n. 1, p. 4–7, jan. 2001. ISSN 1617-4909. Citado 2 vezes nas páginas 17 e 27.

EGEA, M.; DANIA, C. Sql-pl4ocl: an automatic code generator from ocl to sql procedural language. **Software & Systems Modeling**, May 2017. ISSN 1619-1374. Disponível em: <http://dx.doi.org/10.1007/s10270-017-0597-6>. Citado na página 30.

ELSEVIER, B. **Scopus | The largest database of peer-reviewed literature | Elsevier**. 2016. <https://www.elsevier.com/solutions/scopus>. (Accessed on 05/20/2016). Citado na página 31.

FERREIRA, D. de A.; SILVA, A. R. da. Rsl-il: An interlingua for formally documenting requirements. In: **2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE)**. [S.l.: s.n.], 2013. p. 40–49. Citado na página 40.

FUENTES-FERNÁNDEZ, L.; VALLECILLO-MORENO, A. An introduction to uml profiles. **UML and Model Engineering**, Citeseer, v. 2, 2004. Citado 3 vezes nas páginas 28, 29 e 62.

GASPARINI, I. **Aspectos culturais no modelo do usuário em sistemas adaptativos educacionais : fundamentos, proposta e experimentação**. [S.l.]: Universidade Federal do Rio Grande do Sul, 2013. Citado 2 vezes nas páginas 26 e 27.

GAY, G.; HEMBROOKE, H. **Activity-Centered Design: An Ecological Approach to Designing Smart Tools and Usable Systems**. [S.l.]: MIT Press, 2004. (Acting with Technology). ISBN 9780262262866. Citado 2 vezes nas páginas 25 e 26.

HINCHEY, M.; STERRITT, R. Self-managing software. **Computer**, v. 39, n. 2, p. 107–109, 2006. Citado na página 23.

HONG, J. yi; SUH, E. ho; KIM, S.-J. Context-aware systems: A literature review and classification. **Expert Systems with Applications**, v. 36, n. 4, p. 8509 – 8522, 2009. Citado na página 27.

HOYOS, J. R.; GARCíA-MOLINA, J.; BOTíA, J. A. A domain-specific language for context modeling in context-aware systems. **Journal of Systems and Software**, v. 86, n. 11, p. 2890 – 2905, 2013. Citado na página 38.

HSU, I.-C. Extending uml to model web 2.0-based context-aware applications. **Software - Practice and Experience**, v. 42, n. 10, p. 1211–1227, 2012. Citado na página 28.

IBM-AC. **Autonomic computing 8 elements**. 2001. <http://www.research.ibm.com/autonomic/overview/elements.html>. Citado 2 vezes nas páginas 22 e 23.

KEPHART, J.; CHESS, D. The vision of autonomic computing. **Computer**, v. 36, n. 1, p. 41–50+4, 2003. Citado 3 vezes nas páginas 21, 22 e 23.

KNEER, F.; KAMSTIES, E. Model-based generation of a requirements monitor. In: . [S.l.: s.n.], 2015. v. 1342, p. 156–170. Citado na página 17.

LI, F. L. et al. Non-functional requirements as qualities, with a spice of ontology. In: **2014 IEEE 22nd International Requirements Engineering Conference (RE)**. [S.l.: s.n.], 2014. p. 293–302. ISSN 1090-705X. Citado na página 39.

LUCKEY, M. et al. Adapt cases: Extending use cases for adaptive systems. In: . [S.l.: s.n.], 2011. p. 30–39. Citado 6 vezes nas páginas 18, 35, 36, 43, 48 e 63.

MORANDINI, M. et al. Engineering requirements for adaptive systems. **Requirements Engineering**, v. 22, 2017. Citado 3 vezes nas páginas 34, 36 e 37.

OREIZY, P. et al. An architecture-based approach to self-adaptive software. **IEEE Intelligent Systems and Their Applications**, v. 14, n. 3, 1999. Citado na página 21.

PARASHAR, M.; HARIRI, S. Unconventional programming paradigms: International workshop upp 2004, le mont saint michel, france, september 15-17, 2004, revised selected and invited papers. In: _____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. cap. Autonomic Computing: An Overview. Citado na página 23.

PATI, T.; KOLLI, S.; HILL, J. H. Proactive modeling: a new model intelligence technique. **Software & Systems Modeling**, v. 16, n. 2, p. 499–521, May 2017. Disponível em: <http://dx.doi.org/10.1007/s10270-015-0465-1>. Citado na página 27.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: **Proceedings of the Evaluation and Assessment in Software Engineering (EASE'08)**. Bari, Italy: [s.n.], 2008. p. 1–10. Citado na página 31.

PFLEEGER, S.; ATLEE, J. **Software Engineering: Theory and Practice**. [S.l.]: Prentice Hall, 2010. Citado na página 24.

PRESSMAN, R. **Engenharia de Software - 7.ed.:**. McGraw Hill Brasil, 2009. ISBN 9788580550443. Disponível em: <https://books.google.com.br/books?id=y0rH9wuXe68C>. Citado 3 vezes nas páginas 17, 18 e 24.

PéREZ, J. et al. Ftl-cfree: A fuzzy real-time language for runtime verification. **IEEE Transactions on Industrial Informatics**, v. 10, n. 3, Aug 2014. Citado na página 39.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. **ACM Trans. Auton. Adapt. Syst.**, ACM, New York, NY, USA, v. 4, n. 2, maio 2009. Citado 3 vezes nas páginas 21, 22 e 23.

SAWYER, P. et al. Requirements-aware systems: A research agenda for re for self-adaptive systems. In: . [S.l.: s.n.], 2010. p. 95–103. Citado na página 18.

SHENG, Q. Z.; BENATALLAH, B. Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In: **International Conference on Mobile Business (ICMB'05)**. [S.l.: s.n.], 2005. ISSN 1935-4908. Citado 2 vezes nas páginas 38 e 39.

ŠILINGAS, D. et al. Domain-specific modeling environment based on uml profiles. In: TECHNOLOGIJA. **Information Technologies' 2009: proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Kaunas, Lithuania, April 23-24, 2009**. [S.l.], 2009. p. 167–177. Citado na página 27.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Pearson, 2011. (International Computer Science Series). ISBN 9780137053469. Citado 3 vezes nas páginas 17, 18 e 75.

SOUZA, V. S.; MYLOPOULOS, J. Designing an adaptive computer-aided ambulance dispatch system with zanshin: An experience report. **Software - Practice and Experience**, v. 45, n. 5, p. 689–725, 2015. Citado na página 17.

VIRGILIO, R. D. Aml: a modeling language for designing adaptive web applications. **Personal and Ubiquitous Computing**, v. 16, 2012. Citado na página 35.

WARMER, J. B.; KLEPPE, A. G. **The object constraint language: getting your models ready for MDA**. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 30.

WELSH, K.; SAWYER, P. Understanding the scope of uncertainty in dynamically adaptive systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 6182 LNCS, 2010. Citado na página 17.

WHITTLE, J. et al. Relax: A language to address uncertainty in self-adaptive systems requirement. **Requirements Engineering**, v. 15, n. 2, p. 177–196, 2010. Citado 18 vezes nas páginas 18, 24, 25, 26, 35, 36, 43, 44, 45, 49, 52, 54, 63, 65, 66, 67, 69 e 70.

Apêndices

APÊNDICE A – TUTORIAL PARA ESPECIFICAÇÃO DE REQUISITOS UTILIZANDO A CR4AS

Iniciando um novo projeto

1. Com o editor Papyrus aberto, o primeiro passo é iniciar um novo projeto, isso pode ser feito acessando o menu:

   File > New > Papyrus Project

2. Um *wizard* será aberto para que a linguagem dos novos diagramas seja escolhida, como trabalhamos com uma extensão da UML, selecionaremos **UML** na raia **UML Core**.

3. Na janela seguinte escolheremos um nome para nosso projeto, optou-se pelo nome **CR4AS Model**.

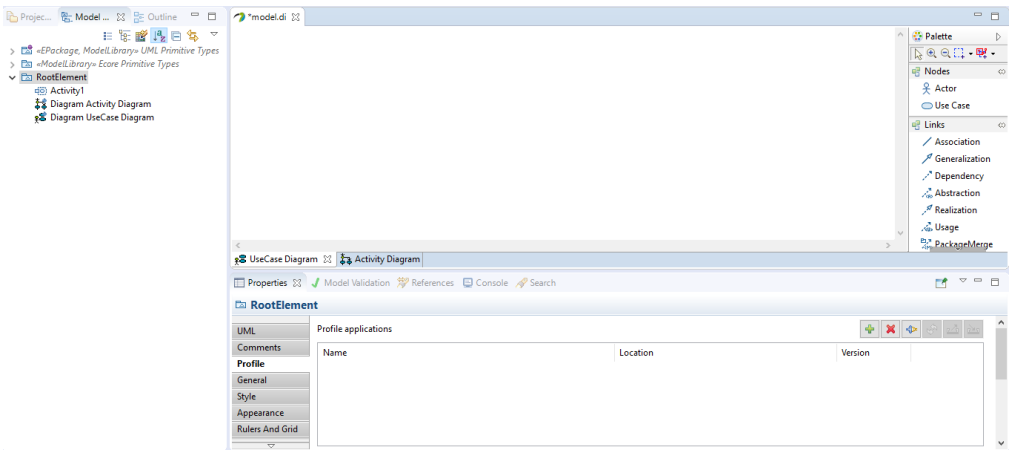
4. Na janela seguinte escolheremos os tipos de diagrama a serem criados, escolheremos os tipos estendidos pela CR4AS: ***Activity Diagram*** e ***UseCase Diagram*** e selecionamos a opção *Finish*.

5. Somos apresentados então ao ambiente de modelagem visto na Figura 46. O próximo passo é a aplicação do perfil CR4AS a este projeto para a modelagem do domínio específico.

6. Selecionamos o ***Root Element*** como exibido na Figura 46 e na aba ***Properties*** selecionamos ***Profile > Apply Profile*** e selecionamos a CR4AS, após isso selecionamos todos os seus subprofiles.

7. O projeto agora inclui a CR4AS e está pronto para o início da especificação de nossos requisitos para sistemas autoadaptativos.

Figura 46 – Tela inicial do projeto.



Iniciando o processo de modelagem

Iniciaremos o processo de especificação modelando os requisitos apresentados por Whittle em um diagrama de casos de uso.

1. Na paleta selecionaremos um objeto ***Actor*** que receberá o nome de **ALL**. Este ator é o sistema apresentado por Whittle e responsável pelos requisitos.

2. Após, criaremos os requisitos apresentados por Whittle. Selecionamos o objeto ***Use Case*** na paleta e criamos cinco destes no diagrama com os nomes: **R1**, **R1.1**, **R1.2**, **R1.3**, **R1.4**

3. Agora, selecionando a relação ***Association*** na paleta, realizaremos a ligação entre **ALL** e **R1**.

4. Após, utilizando a relação ***Dependency*** faremos as seguintes ligações: **R1.1 -> R1**, **R1.2 -> R1**, **R1.3 -> R1** e **R1.4 -> R1**.

5. O próximo passo é a aplicação dos esteriótipos da CR4AS aos elementos. Para fazermos isso na Papyrus, selecionamos o elemento desejado e na aba ***Properties > Profile*** selecionamos ***Apply stereotypes*** e selecionamos o estereótipo desejado dentre uma lista dos disponíveis.

6. Aplicaremos os seguintes estereótipos em nosso modelo: **ALL -> SoftwareAgent**; **R1 -> Goal**; **R1.1, R1.2, R1.3, R1.4 -> Relaxed**; **Association -> Shall** e **Dependency -> Support**.

7. Nesta etapa o diagrama deve ficar como o visto na Figura 30.

8. Para verificarmos se nenhuma regra definida na CR4AS foi quebrada em nosso modelo, clicamos como botão direito no modelo > ***Validation > Validate Model***

9. Utilizaremos o diagrama de atividade já criado para especificarmos **R1.1**. Para isso, vamos primeiramente renomeá-lo para **R1.1**.

10. Utilizaremos a propriedade de *Hiperlink* para fazermos a ligação ***Relaxed -> Scenario***. Para isso, posicione o mouse acima do requisito escolhido com a tecla ***Alt*** pressionada. No menu flutuante selecione a opção ***Modify Hiperlinks***.

11. Na janela aberta selecionamos ***New Hyperlink > R1.1 > OK***. Na aba ***Default HyperLinks*** adicionamos R1.1 em *Default Hyperlinks* para podermos abrir sua forma especificada com um duplo clique sobre o requisito.

12. Agora com um duplo clique sobre o requisito, somos direcionados ao diagrama de atividades correspondente.

Especifiando o Requisito R.1.1

R1.1: *The fridge* ***SHALL*** *detect and communicate information with* ***AS MANY*** *food packages* ***AS POSSIBLE****.*

1. A primeira etapa é adicionarmos o esteriótipo ***Scenario*** ao objeto *Activity* seguindo os passos já descritos anteriormente para adição de esteriótipos.

2. o requisito será modelado baseado na ideia de *Autonomic COntrol Loop* como visto na Figura 47, que nos diz sobre como um sistema autonômico funciona, Coletando dados através de sensores, analisando estes dados, decidindo como agir e tomando uma ação baseada no contexto coletado.

3. Como em um diagrama de atividades, iniciamos o fluxo criando um ***Initial node*** **Início**.

4. Criamos um ***Opaque Action*** *Fridge* estereotipado como ***Agent***.

5. É necessário ao sistema analisar o contexto para saber quando as informações sobre a comida na geladeira são alteradas, para isso, criamos uma ***Opaque Action*** com nome *Analyze contextual food*, esteriotipada como ***Contextual Analyze***.

6. Indicamos que esta é uma ação de ***Fridge*** adicionando um ***Control Flow*** estereotipado como ***Shall***.

7. Uma mudança de contexto é identificada em nosso perfil através de um ***Decision Node*** esteriotipado como ***Contextual Change***.

8. O sistema deve seguir analisando o contexto caso nenhuma mudança ocorra, indicamos isso com um ***Control Flow*** voltando a nossa ***Contextual Analyze***. Caso uma mudança ocorra, o sistema deve decidir como agir. Modelaremos aqui um contexto onde a quantidade de comida foi alterada.

9. Criamos um ***Control Flow*** de ***Contextual Change*** até um novo ***Actor Fridge***.

10. Este ator será responsável por uma nova ***Action Detect food information***

11. Whittle aplica aqui o operador ***AS MANY AS POSSIBLE*** para identificar a incerteza desta ação, por fatores como a incapacidade dos sensores em capturarem todas as informações dos alimentos. Adicionaremos este operador também, adicionando além de ***Shaal***, o operador ***AsManyAsPossible*** como esteriótipo desta relação.

12. A próxima etapa é o sistema informar os dados coletados. Para isso, criamos outro ***Agent Fridge*** indicando através de um ***ControlFlow Shall*** que ele deve realizar a ***Action Comunicate information of food packages***.

13. Após isso, um ***FinalNode*** indica o fim do requisito.

14. Para informarmos quais os elementos de contexto que interferem neste requisito, usamos três ***Comments*** estereotipados como ***Enviroment, Monitor e Relationship***.

15. Como descrito no requisito, ***Enviroment*** possui as informações: *food location*, *food information* e *food state*.

16. ***Monitor*** possui as informações: *RFID Readers*, *Cameras* e *Weight Sensor*.

17. ***Relationship*** possui as informações: *RFID, weight sensor -> food informations; Cameras -> Food location.*

18. Assim como o modelo anterior, este também pode ser validado seguindo os mesmos passos para garantir que as regras definidas foram seguidas.

19. O requisito final especificado é apresentado na Figura 31.

Figura 47 – Autonomic Control Loop.