Pós-Graduação em Ciência da Computação

CEManTIKA CASE: uma Ferramenta de Apoio ao Desenvolvimento de Sistemas Sensíveis ao Contexto

Por

Raphael Freire de Araújo Patrício

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, SETEMBRO/2010

Universidade Federal de Pernambuco
Centro de Informática

Raphael Freire de Araújo Patrício

CEManTIKA CASE: uma Ferramenta de Apoio ao Desenvolvimento de Sistemas Sensíveis ao Contexto

Trabalho apresentado ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Patricia Cabral de Azevedo Restelli Tedesco*
Co-orientadora: *Vaninha Vieira dos Santos*

Recife
SETEMBRO, 2010

Catalogação na fonte
Bibliotecária Joana D'Arc L. Salvador, CRB 4-572

Patrício, Raphael Freire de Araújo.
CEManTIKA CASE: uma Ferramenta de Apoio ao Desenvolvimento de Sistemas Sensíveis ao Contexto / Raphael Freire de Araújo Patrício. - Recife: O Autor, 2010.
xvi, 99 p.: fig. tab.

Orientadora: Patricia Cabral de Azevedo R. Tedesco.
Dissertação (Mestrado) - Universidade Federal de Pernambuco, CIN, Ciência da Computação, 2010.

Inclui bibliografia e apêndice.

1.Engenharia de software. 2.Engenharia de software auxiliado por computador. I. Tedesco, Patricia Cabral de Azevedo R, Tedesco (orientadora). II.Título.

005.1 (22.ed.) MEI 2011-105

Dissertação de Mestrado apresentada por **Raphael Freire de Araújo Patrício** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"CEManTIKA CASE: uma Ferramenta para o Desenvolvimento de Sistemas Sensíveis ao Contexto"**, orientada pela **Profa. Patrícia Cabral de Azevedo Restelli Tedesco** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

Prof. José Maria Nazar David
Universidade Salvador

Profa. Patrícia Cabral de Azevedo Restelli Tedesco
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 10 de setembro de 2010.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

*Eu dedico este trabalho ao meu avô e padrinho Pedro Araújo (*in memorian*) e à Neuza Câmara (*in memorian*), eterna bisavó.*

Agradecimentos

A Deus, por ter me dado a oportunidade de viver e de concretizar este sonho.

À minha família, em especial minha mãe Rosanjala, por ter suportado meu alto nível de estresse durante essa jornada.

Às minhas orientadoras, Patricia e Vaninha, pela dedicação e paciência infinitas.

À Suzanna, com quem compartilhei experiências, por ter estado ao meu lado em todas as etapas importantes da minha vida.

Aos meus amigos de curso, com os quais pude contar durante esses dois longos anos, que me ajudaram e aconselharam na busca pelo melhor caminho, em especial Anderson e Raony.

Ao Centro de Informática da UFPE pela forma acolhedora que me recebeu.

Agradeço também aos professores Ana Carolina Salgado e José Maria David por avaliarem o trabalho realizado e por participarem da banca examinadora dele.

Aos meus familiares, que de alguma forma me ajudaram a chegar até aqui.

Aos colegas do NTI da UFPB, empresa na qual trabalho, pela compreensão nos momentos mais difíceis.

A todas as pessoas que torceram por mim e que me apoiaram, direta ou indiretamente, desde o primeiro passo.

Aquilo que se faz por amor está sempre além do bem e do mal.

—FRIEDRICH NIETZSCHE

Resumo

Os sistemas computacionais são usados para apoiar a execução de várias tarefas cotidianas e fornecem aos usuários uma grande quantidade de informações. Para torná-los mais adaptativos e fáceis de usar surge Contexto. Contexto é o conjunto de informações que caracterizam as entidades relevantes envolvidas na interação entre um usuário e uma aplicação. Sistemas sensíveis ao contexto (SSC) usam contexto para proverem informações e serviços mais relevantes aos usuários por meio de mecanismos de adaptação, assistência e percepção. Todavia, o desenvolvimento dessas aplicações não é trivial e apresenta desafios na aquisição, processamento, uso e disseminação de contexto. Para auxiliar o desenvolvimento de aplicações sensíveis ao contexto foram propostas várias abordagens de *middlewares*, *toolkits*, *APIs* e metamodelos. O CEManTIKA apoia a modelagem e projeto de SSC de forma independente de domínio por meio de uma arquitetura genérica, um metamodelo e um processo de *software*. No entanto, não foi encontrada uma ferramenta que permita a construção de SSCs guiada por um processo partindo da sua especificação e que use artefatos já modelados na aplicação. Esse trabalho apresenta a ferramenta CEManTIKA CASE que permite identificar o contexto relevante para uma aplicação e os mecanismos de adaptação em função dele. Um estudo experimental preliminar, realizado com nove participantes, verificou a aplicabilidade da ferramenta para o projeto de SSC e permitiu a modelagem dos elementos propostos no CEManTIKA.

Palavras-chave: Sistemas sensíveis ao contexto, Ferramentas CASE, Metamodelos UML

Abstract

Computer systems are used to support daily tasks and provide users with an increasing amount of information. Context makes them more adaptable and easier to use. Context is the set of information that characterizes the relevant entities in the interaction between a user and an application. Context Sensitive Systems (CSS) use context to provide information and relevant services to the users through adaptation, assistance and perception mechanisms. Developing a CSS is not a trivial task and presents challenges associated with tasks such as acquisition, processing, usage and dissemination of contextual information. Several approaches of middlewares, toolkits, APIs and metamodels were proposed to help software engineers in developing a CSS. CEManTIKA supports context modeling and CSS design, in a generic, domain-independent way. The CEManTIKA approach involves the provisioning of a generic architecture, a metamodel and a CSS design process. However, there is no tool that enables building a context sensitive system guided by a software process and that uses common artifacts already defined in the application. This work presents the CEManTIKA CASE tool that enables identifying structure and behavior aspects of context. A preliminary study, conducted with nine participants, confirmed the applicability of the tool for design CSS and enabled the modeling of proposed elements in CEManTIKA framework.

Keywords: Context Sensitive Systems, CASE tools, UML metamodeling

Sumário

1 Introdução **1**

2 Contexto em Computação **5**

2.1 Contexto 6

2.2 Sistemas Sensíveis ao Contexto 9

2.3 Técnicas para representação de contexto 12

2.3.1 Grafos Contextuais 14

2.4 Processos de Engenharia de Software para SSC 16

2.5 Considerações Finais 17

3 Trabalhos Relacionados e Ferramentas **18**

3.1 Meta-Ferramentas 19

3.2 Ferramentas de apoio ao desenvolvimento de SSC 23

3.2.1 Apoio a Contexto 23

CAPpella – Context-aware Prototyping 24

iCAP – Interactive Context-Aware Prototyper 25

Pópulo 26

Context-ADDICT Designer 28

CML – Context Modelling Language 29

Ferramentas que usam Metamodelos UML 30

3.2.2 Discussão sobre os trabalhos relacionados 33

3.2.3 Ferramentas CASE implementadas com Metamodelos UML 35

3.3 Considerações Finais 37

4 CEManTIKA CASE 38

4.1 CEManTIKA 39

4.1.1 Arquitetura de Manipulação de Contexto 39

4.1.2 Metamodelo de Contexto 42

4.1.3 Processo de Projeto de SSC 46

4.1.4 Alterações no Processo de Projeto de SSC 48

4.2 CEManTIKA Process 49

4.3 CEManTIKA Modeling 50

4.3.1 Requisitos Funcionais 52

4.3.2 Requisitos não-funcionais 56

4.3.3 Visão Arquitetural 57

4.4 Considerações Finais 60

5 Implementação e Estudos Experimentais 61

5.1 CEManTIKA Process 61

5.2 CEManTIKA Modeling 64

5.3 Experimentação 72

5.4 Considerações Finais 76

6 Conclusões e Trabalhos Futuros 77

6.1 Contribuições 78

6.2 Dificuldades Encontradas 79

6.3 Trabalhos Futuros 80

Referências 82

A Restrições OCL 89

B Sistema de Auxílio a Missões 92

C Questionário de Avaliação 95

Lista de Figuras

2.1 Classificação de contexto em relação ao foco 9
2.2 Exemplo de grafo contextual 15

3.1 CAPpella sendo treinada para reconhecer o cenário de uma reunião 26
3.2 Interface do usuário do iCAP 27
3.3 Modelo UML executável orientado a aspectos 28
3.4 Arquitetura da ferramenta CADD 29
3.5 Arquitetura da ferramenta CML 31
3.6 Extensões do diagrama de classes da UML para modelar contexto 32

4.1 Integração entre os elementos do CEManTIKA 40
4.2 Arquitetura de Contexto 41
4.3 Metamodelo estrutural de Contexto 43
4.4 Perfil de Contexto com propriedades e estereótipos 45
4.5 Processo de Desenvolvimento de Sistemas Sensíveis ao Contexto (SSC) 47
4.6 Fluxo de execução da atividade Especificação de Contexto 51
4.7 Tarefas e artefatos de entrada e saída da atividade Especificação de Contexto 52
4.8 Diagrama de Casos de Uso do CEManTIKA Modeling 53
4.9 Simulação do Grafo Contextual 59
4.10 Visão arquitetural do CEManTIKA Modeling. 60

5.1 Biblioteca do CEManTIKA Process 62

5.2 Navegação entre os elementos do processo 64
5.3 Apresentação inicial do CEManTIKA Modeling 66
5.4 Percepção de erro após verificação de regra OCL 67
5.5 Assistente para importação do Diagrama de Casos de Uso 68
5.6 Configuração de um estereótipo na ferramenta CEManTIKA CASE 69
5.7 Componentes do JBoss Drools Flow 71
5.8 Criação de um grafo contextual na ferramenta no CEManTIKA Modeling 72
5.9 Experiência dos participantes no desenvolvimento de SSCs 73
5.10 Comparação da tarefa Identificar Entidades Contextuais e Elementos Contextuais com e sem ferramenta 74
5.11 Apoio do Eclipse no uso do CEManTIKA CASE 75

B.1 Diagrama de Casos de Uso 93
B.2 Biblioteca do CEManTIKA Process 94

Lista de Códigos Fonte e Exemplos

3.1 Exemplo de regra construída no iCAP 27

4.1 Exemplo de regra de produção criada 46

4.2 Exemplo de Variações de Contexto 55

4.3 Regra OCL para um Elemento Contextual 58

5.1 Exemplo de template do JET 70

5.2 Exemplo de código Java gerado para uma Entidade Contextual 70

A.1 Regra OCL para um Agent 89

A.2 Regra OCL para uma Task 89

A.3 Regra OCL para uma Associação do tipo *executes* 90

A.4 Regra OCL para uma Entidade Contextual 90

A.5 Regra OCL para um Elemento Contextual que verifica se a propriedade ContextType possui um valor válido 90

A.6 Regra OCL que verifica se um Elemento Contextual (Associação) está associado a uma Entidade Contextual 90

A.7 Regra OCL que verifica se um Elemento Contextual (Propriedade) está associado a uma Entidade Contextual 90

Lista de Tabelas

2.1 Comparativo entre técnicas de representação de contexto 13

2.2 Abordagens para desenvolvimento de SSCs 17

3.1 Comparativo entre meta-ferramentas CASE 22

3.2 Ferramentas para desenvolvimento de SSCs 34

Lista de Abreviaturas e Siglas

AOEM	Aspect-Oriented Executable Modeling
API	Application Programming Interface
CAS	Context-Aware Service
CASE	*Computer-Aided Software Engineering*
CEManTIKA	*Contextual Elements Modeling and Management through Incremental Knowledge Acquisiton*
CEKB	Contextual Elements Knowledge Base
CML	Context Modelling Language
CSS	Context Sensitive System
CxG	Grafo Contextual
DDS	Desenvolvimento Distribuído de Software
EC	Elemento Contextual
EMF	Eclipse Modeling Framework
EPF	Eclipse Process Framework
ER	Entidade Relacionamento
GMP	Graphical Modeling Project
GPS	Geo-Posicionamento por Satélite
IDE	Integrated Development Environment
JET	Java Emitter Templates

JSON	JavaScript Object Notation
M2M	Model to Model
M2T	Model to Text
MDA	Model Driven Architecture
MDD	*Model Driven Development*
MDT	Model Development Tools
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
ORM	Object Role Modeling
PbD	Programming by Demonstration
POA	Programação Orientada a Aspectos
RUP	Rational Unified Process
UML	Unified Modeling Language
SDK	Software Development Kit
SGBD	Sistema Gerenciador de Bancos de Dados
SPEM	Software Process Engineering Metamodel
SSC	Sistemas Sensíveis ao Contexto
UWE	UML-based Web Engineering
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Capítulo 1

Introdução

A grande quantidade de informação disponível e a complexidade das atividades do dia-a-dia inundam os usuários de sistemas computacionais com serviços com pouca semântica e que dependem de escolhas e interações explícitas para solucionar problemas. Segundo Vieira, Tedesco e Salgado (2009) os sistemas computacionais devem prover serviços mais ricos e autônomos que permitam, de forma transparente, aproximar o homem da máquina. Para alcançar esse ideal, novos desafios relacionados a Sistemas Distribuídos, Interação Homem-Máquina e Contexto Computacional devem ser resolvidos.

Vieira, Tedesco e Salgado (2009) afirmam que contexto permite restringir o que é ou não importante em uma situação e, assim, aprimora a compreensão do ambiente, ações e eventos que cercam o indivíduo. Usamos informações contextuais a todo momento para tomar decisões cotidianas. Se vamos ao cinema, podemos assistir a um filme de ação ou a uma comédia dependendo da companhia (e.g. amigo, namorada). Escrevemos um e-mail de maneiras diferentes de acordo com o destinatário e o assunto. Se o e-mail contém as fotos do último passeio, somos informais, se o e-mail é de trabalho procuramos uma linguagem mais adequada e formal.

As pessoas mudam constantemente o objetivo da interação com os sistemas computacionais: trabalham, estudam, divertem-se, comunicam-se. Diante disso, vários pesquisadores (CHAVES, 2009; CHEVERST et al., 2000; SALBER; DEY; ABOWD, 1999) observaram a relevância das informações contextuais (e.g. localização, tempo e atividade do usuário) para identificar o que é relevante nessa interação e adaptar o uso das aplicações. A definição de Salber, Dey e Abowd (1999) diz que "contexto é qualquer informação que pode ser usada para caracterizar a situação de entidades (pessoas, obje-

tos e coisas) que são consideradas relevantes para a interação entre um usuário e uma aplicação [...]". Por meio de Contexto Computacional é possível perceber as intenções do usuário ao usar uma aplicação, adaptá-la e torná-la mais autônoma. Assim, os engenheiros de software podem construir sistemas mais ricos e atrativos que se adaptam ao seu ambiente de uso, denominados Sistemas Sensíveis ao Contexto (SSC).

No entanto, segundo Vieira (2008), o desenvolvimento de um sistema sensível ao contexto envolve novos desafios, como: que tipos de informação considerar como contexto, como representar essas informações, como adquiri-las e processá-las, como integrá-las ao *software* e como apresentá-las ao usuário. Para Henricksen e Indulska (2006) as aplicações sensíveis ao contexto ainda não fizeram a transição dos laboratórios de pesquisa para o dia a dia das pessoas devido a três problemas principais: *a*) dificuldades de implementação; *b*) desafios sociais relacionados com privacidade e usabilidade; e *c*) entendimento imperfeito de como usar contexto nos sistemas.

Para facilitar a implementação de SSC algumas ferramentas foram desenvolvidas. Diversas abordagens desenvolveram *middlewares* (KANG et al., 2007), *APIs* (BARDRAM, 2010) e *toolkits* (SALBER; DEY; ABOWD, 1999), mas a maioria não definiu um modelo sistemático baseado em um processo para a criação de SSC. A maior parte das abordagens pode ser usada apenas em domínios específicos, como aplicações Web (KAPITSAKI et al., 2009) e *healthcare services* (KANG et al., 2007). A construção de um SSC guiada por um processo de *software* auxilia a identificação das informações de contexto relevantes e apoia a criação dos seus mecanismos de uso. Segundo Vieira (2008), um processo de *software* apoia os desenvolvedores de SSC a especificar os requisitos de contexto de uma aplicação. Outros trabalhos (SOHN; DEY, 2003b; MCFADDEN; HENRICKSEN; INDULSKA, 2004; VIEIRA, 2008; KANG et al., 2007; DEY et al., 2004) mostram a importância e necessidade de ferramentas para prototipação ou que auxiliem a criação de aplicações sensíveis ao contexto.

Nesse ambiente, Vieira (2008) abordou o desenvolvimento de sistemas sensíveis ao contexto por meio de um processo, arquitetura e metamodelo UML, preocupando-se com modularidade e reusabilidade e, definiu o *Contextual Elements Modeling and Management through Incremental Knowledge Acquisiton* (CEManTIKA), um *framework* conceitual que apoia o projeto de SSCs de forma genérica e independente de domínio.

O CEManTIKA auxilia o entendimento de contexto e o projeto de SSC. Contexto é definido em termos de elementos contextuais e do foco da interação entre usuário e aplicação. CEManTIKA é dividido em três elementos principais: uma *Arquitetura de*

Referência para SSC, um *Metamodelo de Contexto* e um conjunto de Perfis UML que definem conceitos ligados à modelagem estrutural e comportamental de contexto; e um *Processo de Projeto de SSC*, que define um conjunto de atividades, papéis e artefatos que permitem o projeto sistemático de SSC. Uma das limitações do CEManTIKA é a ausência de uma ferramenta que facilite o uso e integração dos seus componentes, bem como que apóie a transição da fase de projeto para a fase de implementação de um SSC. Além disso, um estudo experimental conduzido por Vieira (2008) mostrou que os participantes, usuários do CEManTIKA, tiveram dificuldades em compreender e utilizar alguns aspectos do metamodelo e do processo na construção de um SSC, o que evidenciou a necessidade de *templates* e *guidelines* que ajudem os engenheiros de *software* na abordagem sugerida.

Nós investigamos a viabilidade de construir ferramentas que dão apoio à construção de SSC por meio de processos e metamodelos independentes de domínio. Engenheiros de *software* e projetistas de sistemas sensíveis ao contexto são a principal audiência desta dissertação.

A principal questão deste trabalho é: Como sistematizar e apoiar a construção de uma aplicação sensível ao contexto, por meio de um processo de *software*, usando conhecimento e artefatos já modelados na aplicação?

O objetivo principal desta dissertação é implementar uma ferramenta CASE, independente de domínio, de apoio ao projeto e desenvolvimento de SSC. As ferramentas *Computer-Aided Software Engineering* (CASE) auxiliam os engenheiros de *software* e gerentes de projeto em cada tarefa associada ao processo de *software* (PRESSMAN, 2009). Elas provêem a automatização de atividades manuais e ajudam a garantir a qualidade do *software* durante sua construção.

As principais contribuições desta dissertação são: i) a avaliação de ferramentas de modelagem e prototipação de sistemas sensíveis ao contexto; ii) a especificação de uma ferramenta CASE de apoio à contrução de SSC por meio do processo e metamodelo de contexto definidos no *framework* conceitual CEManTIKA, chamada CEManTIKA CASE; iii) a implementação de requisitos de modelagem de contexto da ferramenta CEManTIKA CASE e sua posterior avaliação; iv) uma análise crítica do Processo de Projeto de SSC definido por Vieira (2008).

O restante desta dissertação está organizado da seguinte forma:

Capítulo 2 faz uma revisão de contexto e sistemas sensíveis ao contexto.

Capítulo 3 faz uma revisão das ferramentas de apoio ao desenvolvimento de SSC e apresenta a abordagem adotada para o desenvolvimento da ferramenta CEManTIKA CASE.

Capítulo 4 aborda o processo de engenharia de software e metamodelo definidos no CEManTIKA e apresenta as contribuições adicionadas a ele. Depois, especifica a ferramenta CEManTIKA CASE, a qual foi dividida em dois módulos: CEManTIKA Modeling e CEManTIKA Process.

Capítulo 5 apresenta a ferramenta construída e discute o experimento realizado para sua avaliação.

Capítulo 6 resume o trabalho feito, apresenta as conclusões e considerações finais e indica direções para trabalhos futuros.

Capítulo 2

Contexto em Computação

As pessoas usam aplicações computacionais para realizar tarefas diárias, como escolher um filme para assistir ou descobrir o melhor caminho para ir à casa de um amigo. Vivemos na era da informação, na qual as pessoas estão cada vez mais dependentes dos computadores para aprender e tomar decisões. Para permitir que os usuários executem essas tarefas com maior eficiência surge um novo desafio: como prover serviços mais relevantes, adaptativos e proativos que antecipem as necessidades do usuário?

Nesse ambiente, o contexto aparece como peça fundamental. Sistemas que usam contexto são cientes do ambiente que os envolve e capazes de executar ações específicas de acordo com as informações relevantes para cada tarefa.

Assim, o conceito de Contexto vem sendo estudado por diversas áreas da Computação, como: Inteligência Artificial, Computação Ubíqua e Sistemas Colaborativos. Este capítulo apresenta uma revisão dos conceitos de contexto e sistemas sensíveis ao contexto.

O capítulo está organizado da seguinte forma: as Seções 2.1 e 2.2 apresentam os conceitos de contexto e sistemas sensíveis ao contexto definidos por vários pesquisadores e a definição adotada neste trabalho; a Seção 2.3 faz uma revisão das diversas técnicas sugeridas para representar contexto e comportamentos contextuais; a Seção 2.4 abrange os processos de Engenharia de Software para o desenvolvimento sistemático de SSC. Por fim, a Seção 2.5 termina o capítulo.

2.1 Contexto

Contexto é o conjunto de circunstâncias ou fatos interrelacionados que envolvem um evento particular, uma situação, etc. (AULETE, 2010). Por exemplo, é o contexto que permite gritarmos em um estádio assitindo a um jogo de futebol ou ficar em silêncio em um teatro. Informações relacionadas à localidade, tempo, atividade, entre outras podem ser consideradas como contexto.

Em Computação, os pesquisadores procuram aplicar a definição de Contexto para adaptar o comportamento das aplicações. Segundo Dey e Abowd (2000), compreender melhor o conceito de Contexto permitirá aos engenheiros de software escolher quais informações contextuais usar nas aplicações e ajudará a especificar os comportamentos relacionados a estas informações.

Os primeiros trabalhos em Computação que tentaram definir contexto não o descreveram de forma genérica, mas enumeraram informações que podem alterar o comportamento das aplicações, como lugar, tempo e temperatura. No entanto, definir contexto por meio de exemplos torna difícil a generalização do seu conceito. Tentando resolver esse problema, Dey et al. (2001) afirmam que "contexto é qualquer informação que pode ser usada para caracterizar a situação de entidades (i.e. pessoas, locais, objetos) que são consideradas relevantes para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação".

Para Brézillon (1999), o contexto delimita os elementos e as entidades que formam a solução de um problema. Assim, contexto pode ser visto como o conjunto de restrições instanciadas na execução de uma tarefa.

Como apontado por Chaves (2009) as definições de contexto são apresentadas de acordo com os interesses das áreas que o estudam. No entanto, segundo Vieira (2008), os pesquisadores concordam em alguns aspectos das definições: i) contexto existe somente quando relacionado a uma entidade (tarefa, interação ou agente, que pode ser humano ou de software); ii) contexto é um conjunto de itens associados a uma entidade; e iii) um item é considerado parte do contexto somente se for útil para apoiar a resolução de um problema.

Contexto é definido ora como dados (e.g. temperatura, umidade) ora como ações que ao serem executadas levam em consideração mudanças no ambiente. Neste trabalho nós usamos as definições adotadas por Vieira (2008), que fazem uma distinção entre os

termos **elemento contextual** e **contexto**, apresentados abaixo:

> Um **elemento contextual** (EC) é qualquer item de dados ou informação que permite caracterizar uma entidade em um domínio.

> O **contexto** de uma interação entre um agente e uma aplicação, com o objetivo de executar alguma tarefa, é o conjunto de elementos contextuais instanciados que são necessários para apoiar a execução da tarefa.

Essa definição considera o contexto aplicado à interação entre um agente e uma aplicação, na qual o agente pode ser humano ou de software. Além disso, os elementos que compõem o contexto devem possuir um relacionamento de relevância com a tarefa que o agente está executando. Os elementos contextuais definem informações (e.g. idade, temperatura, horário, atividade em execução) estáveis que podem ser conhecidas antecipadamente em tempo de projeto. Por outro lado, o contexto é definido em tempo de execução e representa a instanciação desses elementos na realização de uma tarefa pelo usuário. Por exemplo, em um sistema de guia turístico a idade é um elemento contextual definido no projeto da aplicação. Durante sua execução, vários contextos são instanciados de acordo com a idade de cada usuário. Para um usuário mais jovem a aplicação apresentará atrações ao ar livre e para os mais velhos locais fechados.

Um dos desafios em usar contexto é especificar as informações relevantes e as ações dependentes do contexto para cada situação. Dey e Abowd (2000) afirmam que categorizar contexto ajuda aos engenheiros de software identificarem que informações são úteis para suas aplicações. Para ajudar a identificação dessas informações foram sugeridas várias classificações que separam as informações contextuais de acordo com alguns critérios. Dois deles serão examinados devido a sua importância para a identificação dos elementos contextuais e dos comportamentos dinâmicos de contexto. O primeiro identifica seis dimensões dos elementos de contexto e outro classifica contexto em três tipos de conhecimento levando em consideração o foco do usuário.

Uma das classificações mais conhecidas (MORSE; ARMSTRONG; DEY, 2000; TRUONG; ABOWD; BROTHERTON, 2001), divide as informações contextuais em seis dimensões, também chamadas de 5W+H, apresentadas abaixo:

- *Who* (quem): referente à identidade de um agente na execução de uma tarefa. Exemplos: e-mail e número da conta bancária do usuário.

- *Where* (onde): é um dos elementos mais usados e indica localização. Exemplos: cidade, número da sala de aula e coordenadas geográficas.
- *What* (o quê): indica atividade sendo executada pelo agente. Muitas vezes essa informação é derivada de outras. Exemplos: conversando no MSN, assistindo a um filme, estudando e escrevendo.
- *When* (quando): indica o momento ou duração da execução de uma tarefa. Exemplos: horário e estação do ano.
- *Why* (por que): indica a razão pela qual o agente está realizando determinadas ações. Exemplos: um sistema de comércio eletrônico pode inferir as preferências de um usuário a partir de seu histórico de navegação.
- *How* (como): indica o meio como os elementos contextuais são capturados. Exemplos: a localização do celular de um usuário pode acontecer por meio de Geo-Posicionamento por Satélite (GPS) ou triangulação de estações rádio base.

Segundo Brézillon e Araujo (2005) contexto é sempre relativo a uma coisa, um objeto, uma ideia, ou resumidamente, um *foco de atenção*. Como citado por Vieira et al. (2006), o foco pode ser uma tarefa, a solução de um problema ou uma tomada de decisão. O foco restringe o que pode ser considerado como contexto em um determinado momento. Por exemplo, para lavar roupas, informações como cor, tipo de tecido, etc. são elementos contextuais, ou seja, são úteis para a execução do foco *lavar roupa*. No entanto, o local onde a roupa foi comprada e seu preço não influenciam na execução do foco.

A segunda classificação destaca a relevância da informação contextual na execução de uma tarefa. Brézillon e Pomerol (1999) dividiram contexto em relação ao foco de atenção em dois tipos, conforme ilustra a Figura 2.1: a) conhecimento contextual (CK - contextual knowledge) – todo conhecimento que é relevante e pode ser utilizado para entender um problema; e b) conhecimento externo (EK - external knowledge) – conhecimento não relevante para uma dada situação. Durante a execução de uma tarefa, o Contexto Procedimental (PC - Proceduralized Context), também chamado de *contexto ativo* por Vieira, Tedesco e Salgado (2009), é instanciado a partir do CK.

CK indica os elementos contextuais relevantes para um foco, enquanto EK representa os elementos não relacionados ao foco. No foco *lavar roupa*, o EK inclui o *preço*

e o *local de compra*, e o CK inclui a *cor* e o *tipo de tecido*, ou seja, os elementos que podem auxiliar na execução do foco.

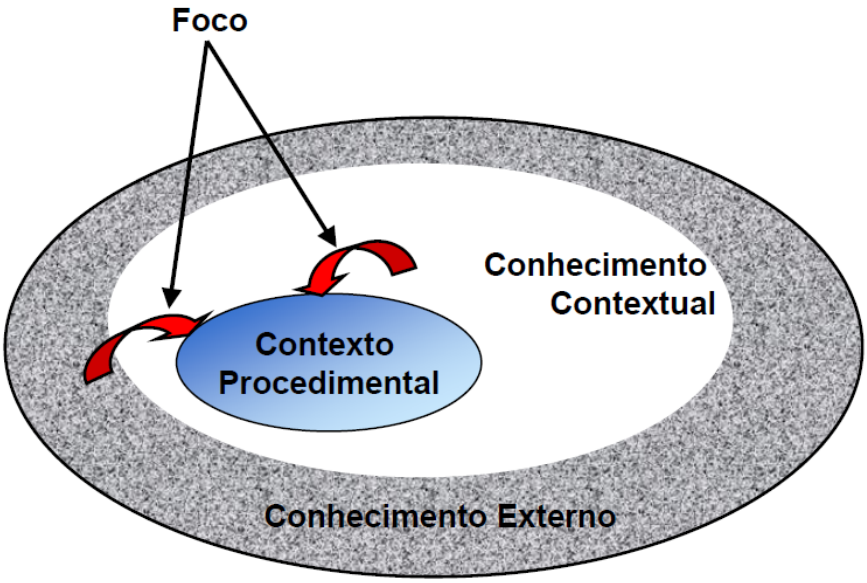

Figura 2.1 Classificação de contexto em relação ao foco (BRÉZILLON; ARAUJO, 2005).

Para representar o aspecto dinâmico de contexto, a construção do contexto ativo é feita de acordo com o conhecimento contextual disponível e o foco em execução. Assim, quando o foco muda um novo contexto é instanciado.

2.2 Sistemas Sensíveis ao Contexto

As aplicações que usam informações contextuais para adaptar ou modificar seu comportamento são chamadas de sistemas sensíveis ao contexto, sistemas adaptativos ou sistemas cientes do contexto. De acordo com Schilit, Adams e Want (1994) os sistemas sensíveis ao contexto se adaptam ao ambiente considerando a localização de uso, as pessoas e objetos próximos, os dispositivos acessíveis. Para Dey e Abowd (2000), "um sistema é sensível ao contexto se ele usa contexto para prover informações relevantes e/ou serviços para os usuários, onde relevância depende da tarefa do usuário". Seguindo essa última abordagem e os conceitos de elementos contextuais e contexto definidos anteriormente, Vieira (2008) define sistemas sensíveis ao contexto como:

Sistemas sensíveis ao contexto (SSC) são aqueles que gerenciam elemen-

> tos contextuais relacionados a um domínio de aplicação e que usam esses elementos para apoiar um agente na execução de alguma tarefa. Essa ajuda pode ser alcançada melhorando a percepção do agente sobre a tarefa ou provendo adaptações que facilitem a execução da tarefa.

Um guia turístico eletrônico é um dos exemplos mais usados para explicar SSCs. O GUIDE (CHEVERST et al., 2000) é um sistema sensível ao contexto que permite que turistas explorem e conheçam uma cidade considerando informações contextuais do ambiente e das pessoas. Os roteiros são montados de acordo com a faixa etária do turista, seus interesses de visitação, clima, atrações abertas para visitas, entre outros. Por exemplo, turistas jovens podem optar por trilhas e locais abertos, enquanto que os mais velhos por atrações históricas em locais fechados. No entanto, uma chuva ou o fechamento de uma atração turística, respectivamente, pode alterar os planos dos dois grupos. Assim, o *tourist guide* pode reagir de acordo com a atualização dessas informações contextuais e modificar os roteiros indicando outros pontos turísticos. O segundo grupo poderia visitar outros locais próximos à localização corrente, como, museus e igrejas.

A diferença entre as aplicações tradicionais e os sistemas sensíveis ao contexto é que as primeiras agem levando em consideração apenas as solicitações e informações fornecidas explicitamente pelos usuários, enquanto que as últimas também levam em consideração informações inferidas de bases de conhecimento e percebidas do ambiente (e.g. por meio de sensores) (VIEIRA; TEDESCO; SALGADO, 2009).

Para entender como as aplicações podem usar efetivamente contexto, Salber, Dey e Abowd (1999) e Vieira, Tedesco e Salgado (2009) classificam os tipos de serviços apoiados por um SSC. Os primeiros dividem os serviços em três categorias:

- apresentação de informações e serviços: aplicações que apresentam informações contextuais para os usuários ou que usam contexto para propor ações configuradas previamente. Exemplo: O GMail [1], serviço de e-mail da Google, exibe sites relacionados ao e-mail aberto como sugestão de leitura.

- execução automática de serviços: descreve aplicações que executam um comando ou reconfiguram o sistema em resposta às mudanças no contexto. Exemplo: O *tourist guide* (CHEVERST et al., 2000) reconfigura o roteiro do turista em resposta às mudanças climáticas.

[1] http://www.gmail.com

- gravação de informações contextuais para recuperação posterior: aplicações que associam informações contextuais a dados crus. Um exemplo é a filtragem de informações de um Sistema Gerenciador de Bancos de Dados (SGBD) pela associação de contexto a grupos de dados (BOLCHINI et al., 2007).

Já Vieira, Tedesco e Salgado (2009) destacam os serviços de apoio à:

- percepção: é responsável por notificar o usuário ou grupo sobre o contexto associado à ocorrência de um evento. Ambientes colaborativos como o Desenvolvimento Distribuído de Software (DDS) (CHAVES, 2009) usam serviços de percepção para notificar o grupo sobre o contexto dos seus participantes, e assim, garantir que a interação seja a mais produtiva possível;
- assistência: o serviço de assistência permite que os usuários realizem suas tarefas da maneira mais eficiente possível. Por exemplo, o contexto pode ser utilizado para permitir que um ambiente de aprendizagem virtual se adapte ao nível de conhecimento do usuário. Nessa situação, o contexto pode ser usado para, por exemplo, fornecer dicas ou calibrar o nível de dificuldade de uma avaliação; e
- adaptação: as aplicações adaptativas modificam seu comportamento em resposta às mudanças no contexto. Como exemplo, sistemas de comércio eletrônico como Amazon [2] modificam sua apresentação levando em consideração o perfil e histórico de navegação do usuário.

Apesar dos muitos benefícios para o usuário final, a construção de sistemas sensíveis ao contexto não é trivial e inclui tarefas desafiadoras, como: especificação de elementos contextuais, criação de componentes para aquisição de ECs, módulo de processamento e inferência de ECs e adaptação do sistema de acordo com ECs instanciados para o foco em uso (VIEIRA et al., 2007).

Um dos desafios no desenvolvimento de sistemas sensíveis ao contexto é representar as informações e comportamentos contextuais em um modelo formal e compreensível por diversas aplicações. A seção 2.3 descreve algumas técnicas para representação de contexto e destaca a representação em *grafo contextual*, utilizada para modelar a parte comportamental dos SSC.

[2] http://www.amazon.com

2.3 Técnicas para representação de contexto

Tipicamente, um SSC usa informações contextuais de várias fontes heterogêneas. Portanto, é necessária uma representação formal que permita o envio e recebimento dessas informações por agentes de software e humanos e que possibilite a interoperabilidade semântica entre eles. Vários pesquisadores investigaram técnicas e modelos para representar contexto considerando a expressividade e a formalidade desses modelos (BETTINI et al., 2010). Algumas técnicas de representação de informações contextuais e conhecimento são sumarizadas na tabela 2.1.

O modelo baseado em chave-valor é o mais simples. Os dados são representados por pares que relacionam um valor a uma chave, também conhecido como dicionário ou mapa. Por exemplo, um SSC pode armazenar os elementos contextuais de uma entidade e os comportamentos relacionados a eles como:

```
entidade: pessoa
elementos_contextuais: peso, idade, sexo
comportamentos: quando peso < 25 Kg e idade >= 10 anos
                então
                pessoa está desnutrida.
```

Embora seja fácil de implementar, a técnica torna difícil a construção de estruturas de dados complexas e é de difícil evolução.

Técnicas baseadas em linguagens de marcação como Extensible Markup Language (XML) [3] e JavaScript Object Notation (JSON) [4] permitem a modelagem de dados hierárquicos e são usadas para trocas de mensagens entre sistemas heterogêneos. No entanto, segundo Vieira, Tedesco e Salgado (2009) elas não provêem suporte para visualização do modelo e são difíceis de manter e evoluir.

Mapas de tópicos é um padrão para representação de conhecimento composto por tópicos, associações e ocorrências. São similares aos mapas mentais e permitem organizar os objetos em uma rede semântica. No entanto, a ausência de tecnologias e padrões inibe seu uso.

Contexto também pode ser representado por meio de ontologias, como em Kang

[3] http://www.w3.org/XML/
[4] http://www.json.org

Tabela 2.1 Comparativo entre técnicas de representação de contexto (VIEIRA; TEDESCO; SALGADO, 2009).

Técnica	**Vantagens**	**Desvantagens**	**Processamento e Recuperação**
Par Chave-Valor	Estrutura simples, de fácil implementação e uso.	Não considera hierarquia. Inadequado para aplicações com estruturas complexas.	Busca linear com casamento exato de nomes.
Linguagem de Marcação	Prevê hierarquia. Esquema de marcação implementa o próprio modelo.	Modelo não resolve incompletude e ambiguidade. Inadequado para representar estruturas complexas.	Linguagem de consulta baseada em marcação.
Mapas de Tópicos	Facilita a navegação entre os elementos contextuais e a leitura por humanos.	Técnica imatura com baixo apoio de ferramentas.	Navegação por redes semânticas.
Ontologias	Agrega regras, conceitos e fatos em um só modelo. Padrões facilitam reuso e compartilhamento. Viabiliza compreensão semântica entre humanos e máquinas.	Não permite modelar o comportamento do SSC. Tecnologia ainda imatura e com poucas ferramentas. Impacto no desempenho.	Motor de Inferência, Linguagens de consulta baseadas em OWL ou *frames*.
Modelos Gráficos	Facilita a especificação dos conceitos e definição do comportamento do SSC.	Não permite processar os conceitos: mapeamento para estruturas de dados.	Pode ser traduzido para XML e usa de processamento em XML.

et al. (2007). Uma ontologia é uma representação formal do conhecimento no qual objetos, conceitos, entidades e relacionamentos do mundo real são usados para descrever um domínio (USCHOLD; GRUNINGER, 1996). Uma ontologia possui alto nível de formalismo permitindo a aplicação de técnicas de raciocínio.

O comportamento e as informações contextuais também podem ser representados

por meio de modelos gráficos, como os baseados na Unified Modeling Language (UML) (AYED; BERBERS, 2006; SHENG; BENATALLAH, 2005), Object Role Modeling (ORM) (MCFADDEN; HENRICKSEN; INDULSKA, 2004) e grafos contextuais.

Os modelos gráficos apoiam a modelagem conceitual de contexto e ajudam a visualizar como os elementos contextuais se relacionam e como afetam o comportamento de um SSC. Modelos baseados em UML permitem o reuso de modelos definidos no projeto da aplicação e ferramentas de modelagem. Os grafos contextuais descrevem a natureza dinâmica de contexto.

Entre as vantagens de usar UML destacam-se sua capacidade de customização para domínios específicos e sua ampla aceitação no meio acadêmico e no mercado. A próxima seção apresenta o conceito de grafos contextuais devido à sua relevância para a representação dos comportamentos contextuais.

2.3.1 Grafos Contextuais

Um Grafo Contextual (CxG) [5] é uma representação baseada em contexto da solução de um problema. É representado por um grafo acíclico direcionado com uma entrada e uma saída e apresenta diversas ações a serem tomadas de acordo com o contexto (BRéZILLON, 2005; BRéZILLON; PASQUIER; POMEROL, 2002). Um caminho de um CxG representa uma forma, também chamada de prática, de resolver um problema. Os grafos contextuais representam o contexto ativo, definido na Seção 2.1, pronto para ser usado em uma ação.

Um grafo contextual é composto por:

- *Action* (Ação): é o elemento principal de um CxG e representa uma ação a ser realizada, como uma tarefa ou um passo na solução de um problema. É representado pelos retângulos na Figura 2.2.
- *Contextual node* (Nó Contextual): corresponde a uma decisão dependente de contexto. É representado pelos círculos abertos na figura. Um nó contextual tem uma entrada e N saídas (*branches*) que representam as N instâncias do elemento contextual pertencente ao nó.
- *Recombination node* (Nó de Recombinação): representado pelos círculos fecha-

[5] http://www.cxg.fr/

dos. Os caminhos de um nó contextual convergem para um nó de recombinação. Assim, o nó de recombinação possui N entradas e uma saída. O nó de recombinação finaliza uma prática (caminho do grafo) e transfere o Elemento Contextual (EC) do contexto ativo para o conhecimento contextual, ou seja, o valor do EC não é mais importante para a solução do problema.

- *Subgraph* (Subgrafo): são grafos contextuais usados para agrupar ou expandir outro grafo. São úteis para estruturar grafos complexos e facilitar sua construção.
- *Activity* (Atividade): é uma ação complexa representada por um subgrafo. Uma mudança em uma atividade aparece em todos os grafos contextuais onde a atividade foi identificada.
- *Parallel action grouping*: são usados para representar subgrafos ou ações cuja ordem de execução não é definida.

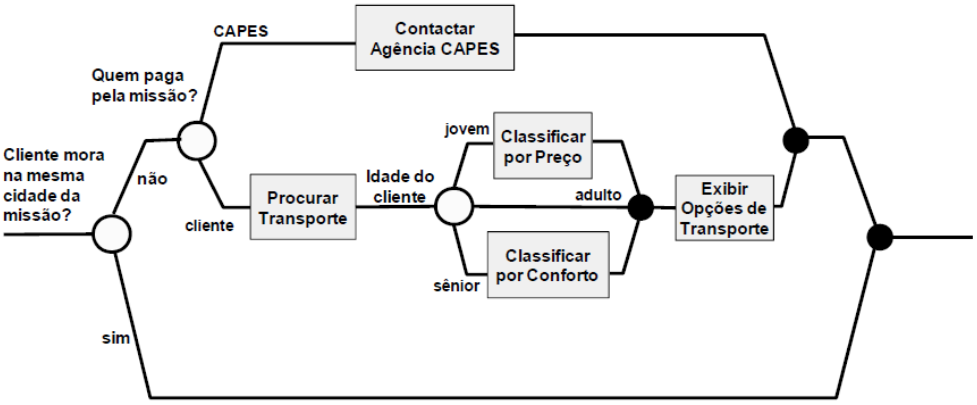

Figura 2.2 Exemplo de grafo contextual (VIEIRA; TEDESCO; SALGADO, 2009).

O exemplo da Figura 2.2 mostra um grafo contextual para a reserva de transporte de uma pessoa que pretende ir a um evento acadêmico (missão). O primeiro nó contextual avalia se o participante mora na cidade da missão. Em caso positivo, o problema está resolvido e nenhuma ação é tomada. Do contrário, o segundo nó contextual avalia quem é responsável por pagar a missão. Se o resultado for CAPES, então a ação `Contactar Agência CAPES` é acionada. Os outros caminhos (*branches*) avaliam as opções de transporte de acordo com a idade do participante.

Segundo Vieira (2008), os grafos contextuais são interessantes para modelar comportamento contextual, pois descrevem como as ações devem ser executadas por uma aplicação e especificam como o contexto as afeta.

Segundo Brézillon (BRéZILLON; PASQUIER; POMEROL, 2002), os grafos contextuais permitem uma representação clara de múltiplas ações que dependem do contexto e são facilmente entendidos pelos usuários e engenheiros de *software*.

Outro desafio no desenvolvimento de SSC é a ausência de práticas de engenharia de software que dêem apoio às atividades de representação, gerenciamento e uso de contexto. A próxima seção apresenta algumas abordagens de processos de apoio ao desenvolvimento de SSC.

2.4 Processos de Engenharia de Software para SSC

Segundo Pressman (2009) processos permitem o desenvolvimento racional de *software*. Assim, na medida em que a pesquisa em SSC foi evoluindo, começaram a aparecer trabalhos dedicados a apoiar o desenvolvimento desse tipo de sistema através de processos, modelos e ferramentas.

Vieira, Tedesco e Salgado (2009) comparam algumas abordagens para o desenvolvimento de sistemas sensíveis ao contexto (Tabela 2.2) considerando os seguintes critérios: i) *Independência de Domínio*: apoio à construção de SSC em diversos domínios de aplicações; ii) *Arquitetura Genérica*: arquitetura genérica para gerenciamento de contexto; iii) *Processo*: apoio ao desenvolvimento de SSC com atividades e tarefas bem definidas; iv) *Modelagem Estrutural e Comportamental*: apoio à modelagem estrutural e comportamental de contexto; v) *Reuso de Modelos já Definidos na Aplicação*: uso de artefatos já produzidos e conhecidos pelos engenheiros de software, reduzindo a complexidade da modelagem de contexto; vi) *Apoio à implementação*: apoio à implementação de SSC.

Entre os trabalhos apresentados (HENRICKSEN; INDULSKA, 2004; NETO; KUDO; PIMENTEL, 2006; AYED; DELANOTE; BERBERS, 2007) o CEManTIKA destaca-se por ser a abordagem mais completa, apresentando um processo, metamodelo e arquitetura. Todavia, não possui apoio à implementação ou ferramenta de apoio à modelagem. Vários pesquisadores (SOHN; DEY, 2003b; MCFADDEN; HENRICKSEN; INDULSKA, 2004; VIEIRA, 2008; KANG et al., 2007; DEY et al., 2004) mostram a importância e necessidade de uma ferramenta para prototipação ou que auxilie a criação de aplicações sensíveis ao contexto.

Tabela 2.2 Abordagens para desenvolvimento de SSCs (VIEIRA; TEDESCO; SALGADO, 2009, Adaptado).

Abordagem	**Independência de Domínio**	**Arquitetura Genérica**	**Processo**	**Modelagem Estrutural e Comportamental**	**Reuso de modelos já definidos**	**Apoio à implementação**
Henricksen (2004)	√	√				√
Bulcão (2006)	√		√			
Ayed et al. (2007)	√			√		√
CEManTIKA (2008)	√	√	√	√	√	

2.5 Considerações Finais

O objetivo deste capítulo foi apresentar uma revisão das definições de contexto, sistemas sensíveis ao contexto e técnicas para modelagem de contexto. Algumas técnicas de representação de contexto foram analisadas, e a técnica de grafo contextual, que é usada para modelar comportamentos, foi detalhada.

Os comportamentos contextuais adicionam ações adaptativas aos sistemas, mas não são triviais de serem especificados e implementados. Nesse ambiente, processos de *software* definem atividades e tarefas que apóiam a identificação do contexto relevante na construção de SSC. No entanto, as abordagens pesquisadas carecem de ferramentas que auxiliem a construção dessas aplicações.

O próximo capítulo discute os trabalhos relacionados à proposta desta dissertação, aborda a necessidade de construção de uma ferramenta CASE para a construção de SSC e compara algumas ferramentas disponíveis para a construção de aplicações CASE.

Capítulo 3

Trabalhos Relacionados e Ferramentas

Várias abordagens de *frameworks*, *middlewares*, APIs e componentes de *software* foram sugeridas para facilitar a criação de sistemas sensíveis ao contexto. Por outro lado, vários pesquisadores relatam a necessidade de ferramentas para a construção desses sistemas. Segundo Sohn e Dey (2003a), embora algumas aplicações tenham sido desenvolvidas, a ausência de apoio à construção de SSCs inibe sua proliferação. Para Mcfadden, Henricksen e Indulska (2004), ferramentas com apoio à verificação da consistência de modelos de contexto e automatização de tarefas relacionadas ao gerenciamento e disseminação de contexto beneficiam a construção de SSCs.

Durante a nossa pesquisa, não foi possível encontrar ferramentas que permitam a construção de SSCs de maneira independente de domínio da aplicação e guiada por um processo de *software*. O objetivo deste capítulo é apresentar os trabalhos que se relacionam à proposta desta dissertação e algumas meta-ferramentas disponíveis para a construção da ferramenta CASE proposta neste trabalho.

O capítulo está organizado da seguinte maneira: a Seção 3.1 compara algumas meta-ferramentas que apoiam o desenvolvimento de ferramentas CASE; a Seção 3.2 lista algumas abordagens para a construção de SSCs e também discute um trabalho que possibilita a integração entre uma ferramenta CASE e um metamodelo UML para a modelagem de contexto; e a Seção 3.3 apresenta algumas considerações finais.

3.1 Meta-Ferramentas de apoio ao Desenvolvimento de Ferramentas CASE

Ferramentas CASE auxiliam os engenheiros de *software* e gerentes de projeto em cada tarefa associada ao processo de *software* (PRESSMAN, 2009). Elas provêem a automatização de atividades manuais e ajudam a garantir a qualidade do produto durante sua construção.

Uma ferramenta CASE pode ser entendida como um ambiente que inclui vários componentes: banco de dados, editores, visualizadores, entre outros (PRESSMAN, 2009). A sua construção é custosa e envolve a implementação de módulos de persistência, geração de gráficos, relatórios, instalação, entre outros. Para facilitar o seu desenvolvimento, algumas ferramentas, chamadas de meta-ferramentas, implementam requisitos comuns a um domínio específico de aplicações e usam mecanismos de extensão que permitem a adição de novas funcionalidades por terceiros.

Para definir a plataforma de desenvolvimento adotada neste trabalho, quatro ferramentas foram avaliadas de acordo com os seguintes parâmetros:

- **API:** verifica a existência de bibliotecas de programação para criação de componentes de interface gráfica, persistência de objetos e manipulação de modelos e relatórios;
- **Assistentes de Ajuda:** identifica o suporte à criação de sistemas de ajuda por meio de exemplos, tutoriais e *cheat sheets* [1];
- **Documentação:** identifica a disponibilidade de documentação em fóruns, tutoriais e livros. A quantidade de documentação está relacionada à maturidade e ao nível de adoção da ferramenta pela comunidade;
- **Geração de Código:** verifica o suporte a ferramentas de transformação de modelos em outros modelos (Model to Model (M2M)) e de modelos em texto (Model to Text (M2T)). Essa característica permite, por exemplo, construir código automaticamente a partir de modelos gráficos;
- **Geração de Wizards:** verifica o suporte à implementação de assistentes, também chamados de *wizards*;

[1] Cheat Sheet é um guia interativo no qual o usuário executa um conjunto de tarefas para atingir um objetivo.

- **Integração com IDE:** verifica a integração com Ambientes de Desenvolvimento Integrado (*Integrated Development Environment (IDE)*). Essa característica permite a construção de um sistema comum para atividades de análise, projeto e codificação;

- **Licença:** verifica o custo da ferramenta. Embora existam vários tipos de licença, esse trabalho as classifica apenas em Gratuita e Paga;

- **Linguagem de Extensão:** o uso de uma linguagem de programação amplamente adotada diminui a curva de aprendizado dos mecanismos de extensão da ferramenta;

- **Mecanismo de Extensão:** existem várias formas de customizar uma ferramenta. Esse parâmetro avalia a extensão por meio de *plugins* ou pela alteração direta do código fonte. A primeira forma é mais simples e permite a adição de novas funcionalidades sem a necessidade de conhecer o código fonte da ferramenta;

- **Plataformas:** especifica em quais plataformas de Sistema Operacional a ferramenta pode ser instalada;

- **Relatórios:** verifica a geração de relatórios através de *templates*;

- **Simulação de Modelos:** identifica o suporte à simulação ou depuração dos modelos gerados;

- **Suporte à Perfil UML:** verifica o suporte à adição de um perfil UML;

- **Suporte à XMI:** XML Metadata Interchange (XMI) é um padrão baseado em XML definido pela Object Management Group (OMG) que permite a interoperabilidade dos dados entre várias aplicações. Sua adoção permite que os modelos gerados em uma ferramenta possam ser editados em outra; e

- **Validação com OCL:** permite a verificação de modelos UML por meio de restrições definidas em Object Constraint Language (OCL). OCL é uma linguagem formal usada para descrever expressões em modelos UML que especificam restrições a serem respeitadas ou consultas sobre os objetos descritos no modelo (OMG, 2010).

A Tabela 3.1 sintetiza a comparação entre as ferramentas Eclipse Modeling Project (Eclipse Foundation, 2010a), ArgoUML (COLLABNET, 2010), Papyrus (Papyrus

UML, 2010) e MagicDraw [2] (MAGICDRAW, 2010). As três primeiras ferramentas foram selecionadas considerando os critérios de licença, difusão na comunidade e possibilidade de extensão, e a última, pela sua completude.

Todas as ferramentas possuem interfaces de integração com perfis UML, mecanismos de geração de código, bibliotecas de programação e são independentes de plataforma.

O projeto Eclipse Modeling (Eclipse Foundation, 2010a) provê mecanismos de extensão por meio de *plugins*, os quais tornam possível adicionar funcionalidades diversas à ferramenta Eclipse, tais como: assistentes de ajuda, componentes de interface gráfica e relatórios. O Eclipse Modeling possui uma diversidade de ferramentas para desenvolvimento baseado em modelos, como linguagens de domínio específico (e.g. Eclipse Modeling Framework (EMF)), desenvolvimento de notações gráficas (e.g. Graphical Modeling Project (GMP)) e transformação de modelos em texto por meio de *templates* (e.g. Java Emitter Templates (JET)). Além disso, a ferramenta possui vasta documentação, haja vista a ampla adoção da IDE Eclipse pela comunidade.

O ArgoUML é uma ferramenta de código aberto para modelagem UML com funcionalidades de apoio cognitivo que permitem refletir o projeto de modelos orientados a objetos por meio de *checklists*, lista de coisas a fazer (*todo lists*) e agentes que sugerem melhorias de projeto nos modelos criados. A documentação disponível não discute detalhadamente aspectos de extensão da ferramenta, que deve ser feita por meio de evoluções em seu código fonte. Não existem mecanismos de geração de ajuda, *wizards* e relatórios. Por fim, a integração com o Eclipse não apresenta conformidade entre as interfaces gráficas das duas ferramentas.

O Papyrus é construído sobre o Eclipse e incorpora seu mecanismo de extensão por meio de *plugins*. A ferramenta permite a geração de *wizards* e sistemas de ajuda. No entanto, existe pouca documentação e os relatórios devem ser produzidos com ferramentas de terceiros.

O MagicDraw não permite a adição de sistemas de ajuda e a geração de *wizards*. Destacam-se a documentação ampla, a integração com as IDEs Eclipse (Eclipse Foundation, 2010b) e Netbeans (Netbeans IDE, 2010), a geração de relatórios e o mecanismo de extensão por meio de plugins codificados na linguagem de programação Java (ORACLE,

[2] O MagicDraw está disponível nas versões: *Community Edition*, *Personal Edition*, *Standard Edition*, *Professional Edition*, *Architect Edition* e *Enterprise edition*. Na comparação é usada a versão *Enterprise Edition*.

Tabela 3.1 Comparativo entre meta-ferramentas CASE

	Eclipse Modeling Project	**ArgoUML**	**Papyrus**	**MagicDraw**
API	√	√	√	√
Assistentes de Ajuda	√	–	√	–
Documentação	Fóruns, Site, Livros etc	Site, Manuais	Site	Fóruns, Manuais, Vídeos, Site
Geração de Wizards	√	–	√	–
Integração com IDE	Eclipse	Eclipse	Eclipse	Eclipse, NetBeans, JBuilder
Licença	Gratuita	Gratuita	Gratuita	Comercial
Linguagem de Extensão	Java	Java	Java	Java e Python
Mecanismo de Extensão	Plugin	Código aberto	Plugin	Plugin
Plataformas	Independente de plataforma	Independente de plataforma	Independente de plataforma	Mac, Unix e Windows
Relatórios	√	–	–	√
Geração de Código	√	√	√	√
Simulação de Modelos	–	–	–	–
Suporte à Perfil UML	√	√	√	√
Suporte à XMI	√	√	√	√
Validação com OCL	√	√	√	√

2010). No entanto, sua licença comercial tornou sua adoção proibitiva.

3.2 Ferramentas de apoio ao desenvolvimento de SSC

Pressman (2009) afirma que as ferramentas CASE devem ser usadas para complementar as práticas de engenharia de *software* e não substituí-las. Portanto, um processo de *software* deve ser estabelecido anteriormente ao uso de uma ferramenta CASE.

As ferramentas CASE são usadas em várias atividades, do gerenciamento de projetos, com ferramentas de estimativa de custos e agendamento de tarefas, à programação por meio de compiladores, editores e *debuggers*. Um dos seus objetivos é aumentar o nível de abstração e permitir a implementação de artefatos a partir de representações gráficas que reduzem o esforço de codificação e depuração (SCHMIDT, 2006). Outro exemplo conhecido e adotado pelos engenheiros de *software* são as ferramentas CASE de gerenciamento de banco de dados que simplificam a criação de objetos e abstraem detalhes dos SGBDs.

Este trabalho propõe a criação de uma ferramenta de apoio ao desenvolvimento de sistemas sensíveis ao contexto por meio de atividades que enriqueçam artefatos de modelagem da aplicação previamente construídos. A partir dessas características, foram investigadas duas classes de ferramentas, destacadas nas próximas seções: i) ferramentas de apoio ao projeto e desenvolvimento de SSC; e ii) ferramentas que usam metamodelos UML de domínio específico.

A Seção 3.2.1 destaca várias abordagens para a construção de SSC; a Seção 3.2.2 discute os trabalhos encontrados; e a Seção 3.2.3 identifica uma ferramenta, não relacionada a contexto, implementada por meio da adição de um metamodelo, especificado em um perfil UML, a uma ferramenta CASE. A última abordagem possibilita o uso de meta-ferramentas de modelagem UML.

3.2.1 Apoio a Contexto

Esta seção apresenta as seguintes ferramentas de apoio ao desenvolvimento de sistemas sensíveis ao contexto: CAPpella (DEY et al., 2004), uma abordagem que usa programação por demonstração (LIEBERMAN, 2001) para permitir o desenvolvimento de aplicações sensíveis ao contexto pelos usuários finais; iCAP (SOHN; DEY, 2003b), uma ferramenta composta por uma interface gráfica e um processador de regras que permite a criação de SSC baseados em regras pelos usuários; Pópulo (FUENTES; GáMEZ; SáN-

CHEZ, 2009), uma abordagem usando modelos executáveis orientados a aspectos; a ferramenta Context-ADDICT Designer (BOLCHINI et al., 2007) que permite filtrar informações associando contexto a dados provenientes de um SGBD; a linguagem de modelagem de contexto CML (MCFADDEN; HENRICKSEN; INDULSKA, 2004) e suas ferramentas; e duas abordagens (AYED; DELANOTE; BERBERS, 2007; SHENG; BENATALLAH, 2005) que usam metamodelos UML. Essas ferramentas foram escolhidas, pois mecanizam algumas tarefas de análise, projeto e construção de SSC. Foram excluídas da análise, APIs, *middlewares* e outras abordagens específicas de domínio ou que não fornecem apoio visual para modelagem de contexto pelos projetistas.

CAPpella – Context-aware Prototyping

A ferramenta CAPpella (Context-aware Prototyping) (DEY et al., 2004) permite que os usuários finais das aplicações desenvolvam seus SSCs. Os autores desta ferramenta argumentam que usuários finais têm maior conhecimento sobre suas atividades e, portanto, são capazes de construir comportamentos mais adequados às suas necessidades.

CAPpella usa programação por demonstração (Programming by Demonstration), que permite ao computador aprender tarefas e ações descritas sem o uso de uma linguagem de programação, para auxiliar os usuários finais a criar suas próprias aplicações.

A CAPpella combina aprendizado de máquina e entradas do usuário para construir SSCs sem que os usuários finais necessitem escrever código. A ferramenta usa o seguinte processo de aprendizagem: um usuário treina, através de várias iterações, um comportamento sensível ao contexto composto por uma situação e uma ação associada, indicando que partes dos dados coletados são relevantes para o comportamento estudado. Por exemplo, um sistema que verifica se um paciente tomou seu remédio pode ser treinado para registrar a medicação no prontuário do paciente ou soar um alarme caso a medicação não tenha sido administrada corretamente. Nesse ambiente são dispostos sensores no paciente e nos medicamentos e o usuário da CAPpella identifica, por meio de algumas iterações de treinamento, em que situações o paciente tomou o remédio. Após o treinamento a aplicação construída é capaz de verificar se o paciente tomou sua medicação.

A ferramenta é formada por quatro componentes principais:

1. **Sistema de gravação:** responsável por capturar as situações e as ações provenien-

tes de sensores e armazená-las em logs com o *timestamp* [3] de sua aquisição.

2. **Detecção de eventos:** é o processo de derivar eventos de alto nível a partir dos dados produzidos pela câmera de vídeo, microfone e qualquer outro sensor que não produza eventos de alto nível. Por exemplo, para cada *frame* capturado pela câmera de vídeo, o número de pessoas e suas localizações (coordenadas) são detectados e adicionados ao sistema de gravação.

3. **Interface do usuário:** permite que os usuários escolham os fluxos de dados e as ações relevantes para o comportamento em questão. A Figura 3.1 mostra a CAPpella configurada para um cenário de reuniões.

4. **Sistema de aprendizado de máquina:** usa uma rede bayesiana para treinar e testar os dados configurados pelos usuários, sem que os últimos precisem entender detalhes de implementação ou como as redes bayesianas dinâmicas (MURPHY, 2002) são usadas.

iCAP – Interactive Context-Aware Prototyper

O iCAP (*Interactive Context-Aware Prototyper*) é um sistema baseado em regras que associa uma ação (Então) a uma situação (Se). Ele é dividido em duas partes: uma interface gráfica para construção de regras e um processador de regras responsável pela gravação e armazenamento delas.

Assim como o CAPpella, o iCAP (SOHN; DEY, 2003b) é uma ferramenta que auxilia os usuários finais na prototipação de aplicações sensíveis ao contexto. A ferramenta permite que usuários definam dispositivos para aquisição de informações contextuais, regras de produção e comportamentos contextuais sem escrever código.

A interface gráfica do iCAP (Figura 3.2) possui duas áreas: o lado esquerdo apresenta os componentes usados na criação das regras, como dispositivos de entrada e saída; e a outra parte permite a formação de regras no formato (Se–Então) associando situações a ações. A área com rótulo *Situation* representa a parte "Se" de uma regra e a área com o rótulo *Actions* compõe o "Então". A Figura 3.2 apresenta a regra identificada no Exemplo 3.1.

[3] *timestamp* é uma representação de tempo usada em Computação que representa a quantidade de milissegundos desde 01/01/1970

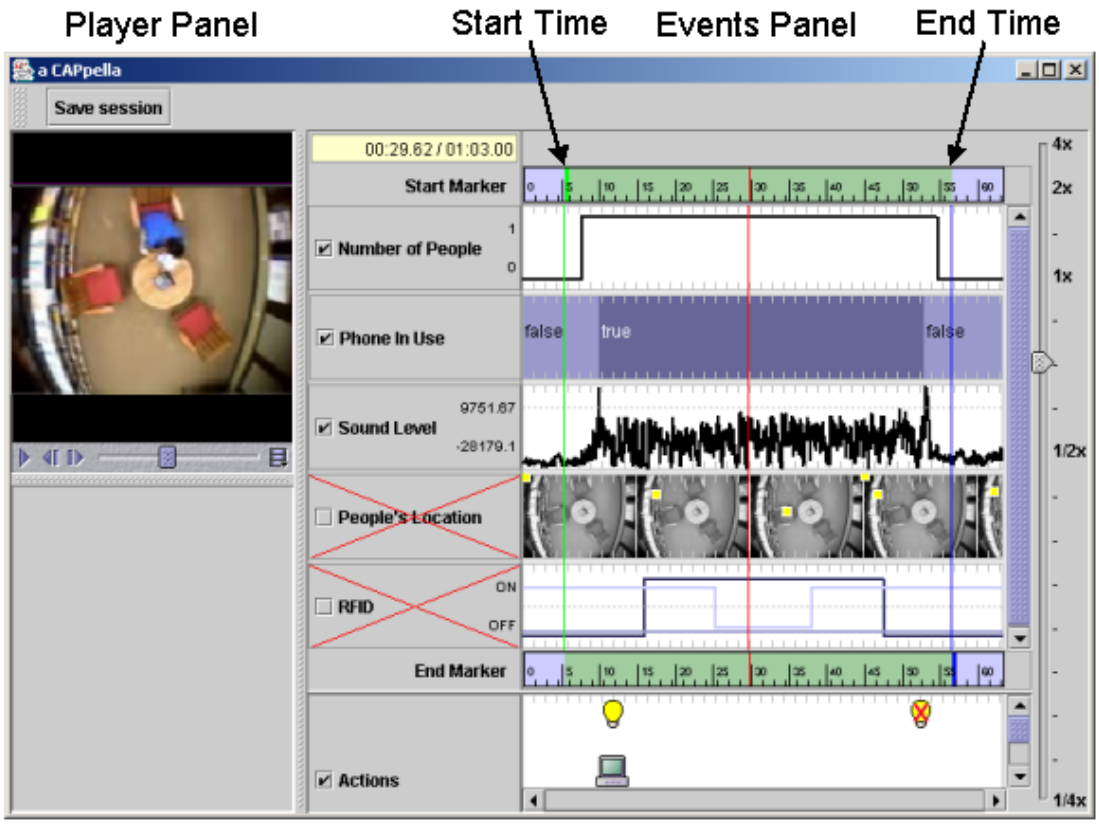

Figura 3.1 CAPpella sendo treinada para reconhecer o cenário de uma reunião (DEY et al., 2004)

Cada elemento é representado através de um ícone. A ferramenta também permite a simulação dos valores das informações contextuais e a visualização da execução das ações correspondentes.

Pópulo – Uma ferramenta para verificar modelos Orientado a Aspectos

Fuentes, Gámez e Sánchez (2009) afirmam que verificar mentalmente a corretude de um modelo sem auxílio de uma ferramenta não é uma tarefa simples. Para amenizar esse problema foi desenvolvida a ferramenta Pópulo (FUENTES; GáMEZ; SáNCHEZ, 2009), um plugin do Eclipse, que permite aos projetistas visualizarem a execução dos comportamentos dos modelos UML. Por fim, os modelos criados e testados no Pópulo podem ser transformados em uma implementação. No entanto, a abordagem não explica como essa transformação é realizada.

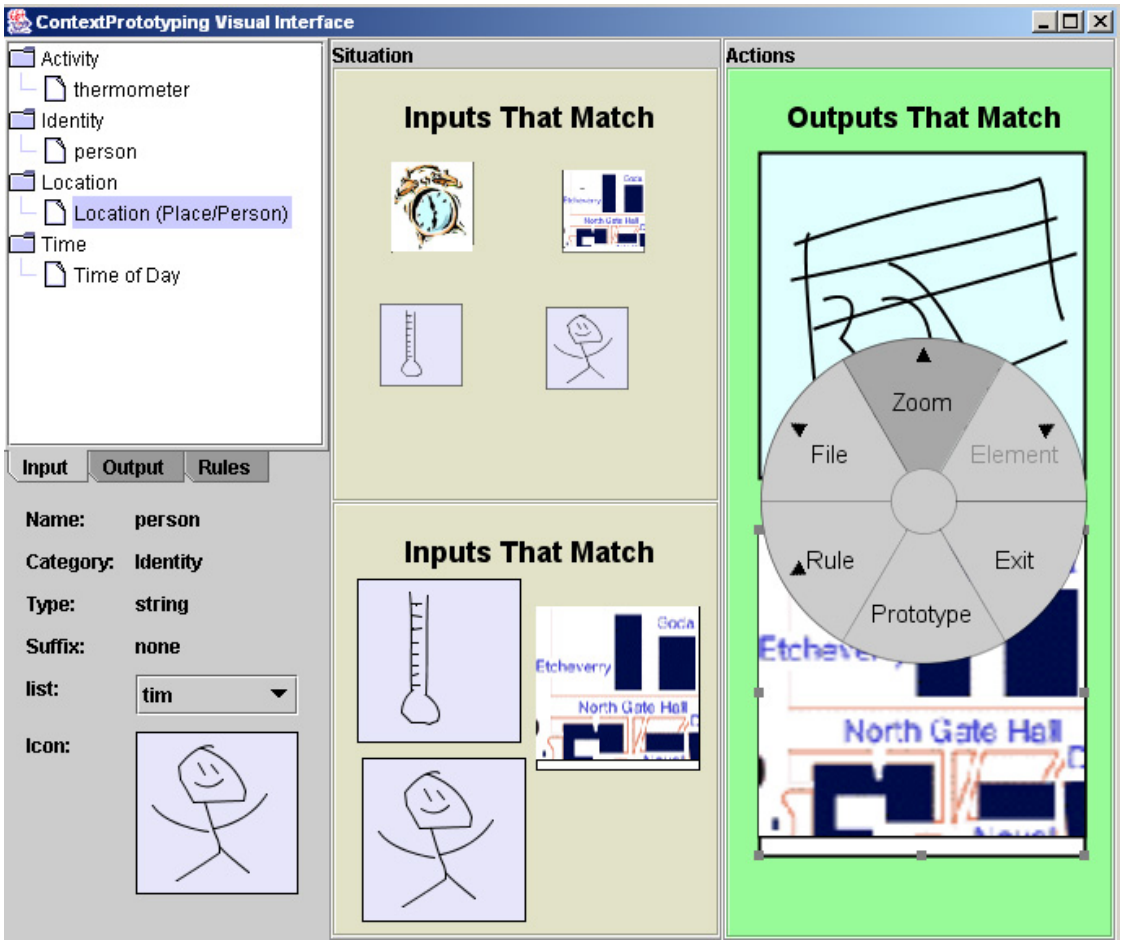

Figura 3.2 Interface do usuário do iCAP (DEY et al., 2004)

Lista de Códigos Fonte e Exemplos 3.1 Exemplo de regra construída no iCAP

```
Se
  John está no escritório depois de 5pm
  e a temperatura é menor que 50 °F
Ou Se
  Jane está no quarto e
  a temperatura está entre 30 °F e 60 °F
Então
  Ligue o aquecedor da casa.
```

Para os autores, as aplicações sensíveis ao contexto estão em constante adaptação ocasionando dois problemas principais: a) grande número de interesses transversais (*crosscutting concerns*); e b) complexa verificação da corretude dos modelos de projeto.

Para resolver estes problemas é proposta uma abordagem usando um modelo exe-

cutável orientado a aspectos (Aspect-Oriented Executable Modeling (AOEM)) definido em um *profile* UML. A abordagem mostra como modelos de execução ajudam a validar, prototipar, testar e depurar SSCs. O trabalho afirma que a execução de modelos em vários contextos e situações auxilia a avaliação do projeto da aplicação e antecipa a detecção de erros, não os propagando para a implementação.

Os principais elementos utilizados na abordagem sugerida são apresentados na Figura 3.3. Inicialmente um modelo executável de projeto é construído na área destacada com o **rótulo 1**. Nele são especificados os componentes para aquisição de informações contextuais (**a**), os comportamentos sensíveis ao contexto definidos como *crosscuting concerns* (**b**) e as regras de composição, também chamadas de *pointcuts* (**c**). O **rótulo 2** destaca o *weaver*, um tipo de compilador que combina os três elementos anteriores e cria o modelo de projeto. No **rótulo 3** o modelo é testado pela ferramenta Pópulo (FUENTES; GáMEZ; SáNCHEZ, 2009). Por fim, o **rótulo 4** transforma o modelo numa implementação.

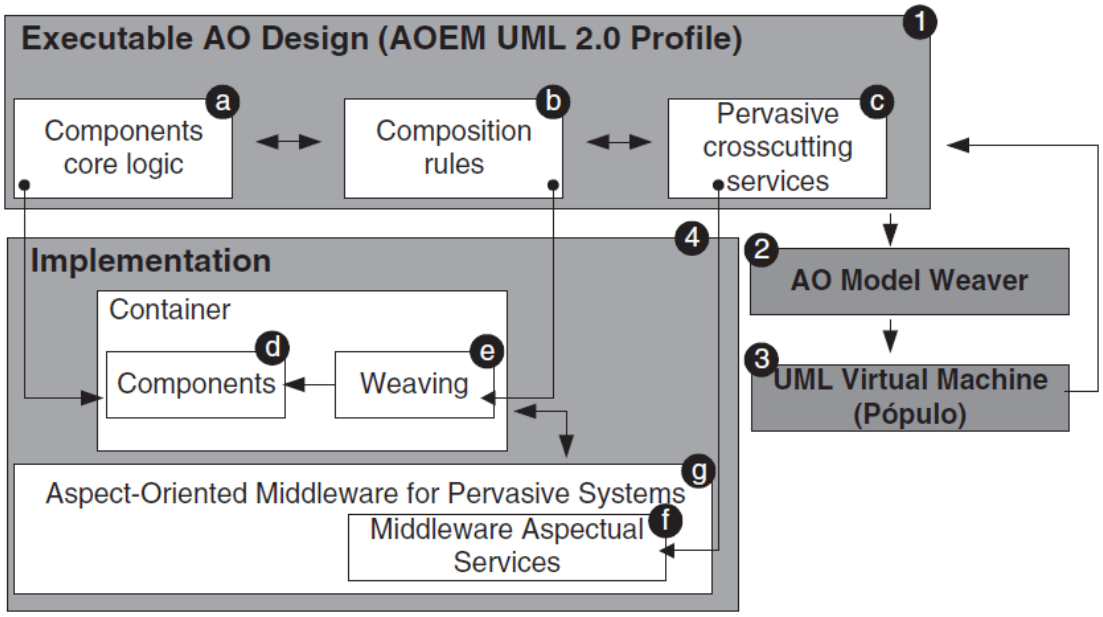

Figura 3.3 Modelo UML executável orientado a aspectos (FUENTES; GáMEZ; SáNCHEZ, 2009).

Context-ADDICT Designer

O Context–ADDICT (Context-Aware Data Design, Integration, Customization and Tailoring) (BOLCHINI et al., 2007) tem o objetivo de filtrar quais informações de um banco

de dados são relevantes para um dado contexto. A adaptação dos dados é importante por causa de duas razões principais: a) tornar a quantidade de dados gerenciável para não confundir o usuário com informações desnecessárias (ruídos) e b) gravar apenas as partes dos dados relevantes nos dispositivos móveis, uma vez que alguns possuem capacidade de armazenamento limitada.

A ferramenta CADD (Context-ADDICT Designer) é usada para associar partes de dados a contextos específicos. A Figura 3.4 apresenta uma visão arquitetural da ferramenta em termos das três fases apoiadas:

- *Context Space Designer Assistant:* definição do modelo de contexto por meio de uma árvore contextual chamada de *Context Dimension Tree*. A árvore é formada pelas dimensões relevantes para a aplicação, como papel do usuário, seus interesses, entre outros;
- *Valid Contexts Generation:* após definir a árvore de contexto é realizada a geração de todos os contextos válidos, como combinações das dimensões da árvore; e
- *Data Tailoring Assistant* associação entre cada contexto com um subconjunto de dados.

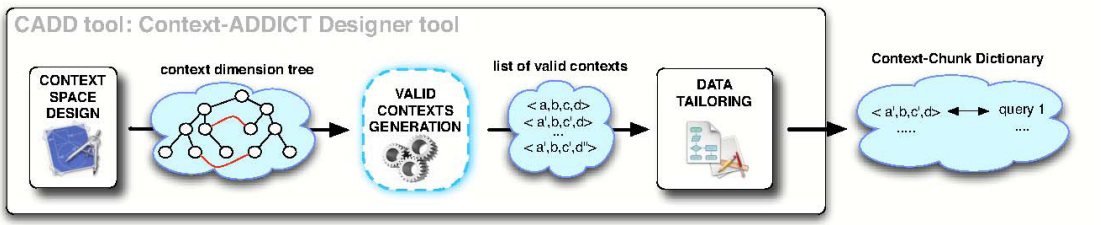

Figura 3.4 Arquitetura da ferramenta CADD (BOLCHINI et al., 2007).

A ferramenta CADD possui mecanismos de verificação de modelos e geração de artefatos. No entanto, o modelo usado restringe o uso de contexto à apenas um escopo: recuperação de dados em ambientes móveis.

CML – Context Modelling Language

Mcfadden, Henricksen e Indulska (2004) apresentam um conjunto de ferramentas para o desenvolvimento de SSCs modelados por meio da Context Modelling Language (CML).

A CML (HENRICKSEN; INDULSKA; MCFADDEN, 2005) permite especificar formalmente, por meio de Object Role Modeling (ORM) (HALPIN, 1996), as entidades e suas propriedades e as fontes de aquisição relevantes para uma aplicação sensível ao contexto.

Os modelos de contexto definidos em ORM são mapeados para a notação relacional de banco de dados e usados por um sistema de gerenciamento de contexto responsável por tarefas como aquisição, processamento e disseminação. Experiências iniciais demonstraram que os esforços de codificação e os erros poderiam ser reduzidos pela introdução de ferramentas para automatização de aspectos de instalação, administração e programação. A ferramenta proposta apoia as seguintes tarefas: i) validação do modelo de contexto e automatização do seu mapeamento para o modelo suportado pelo sistema de gerenciamento de contexto; ii) geração automática de bibliotecas de programação para uso na aplicação sensível ao contexto a partir do modelo de contexto; e iii) geração automática de um mecanismo de troca de mensagens (envio e notificação) usado entre a aplicação e o sistema de gerenciamento de contexto.

A Figura 3.5 apresenta a arquitetura do conjunto de ferramentas proposto com suas entradas e saídas. Inicialmente um modelo CML é validado e compilado para uma representação intermediária e depois é executado por três ferramentas. A primeira, chamada de *SQL Database Mapping Tool* automatiza o mapeamento entre o modelo de contexto e o banco de dados do sistema de gerenciamento de contexto. A segunda, conhecida como *Java Binding Library Tool* é responsável por gerar classes Java [4] auxiliares para a representação e a manipulação de contexto. Por fim, a *Elvin Notification Library Tool* processa um conjunto de classes e interfaces para serem usadas com o sistema de notificação *Elvin* (SEGALL et al., 2000), que oferece serviços de envio e recebimento de dados.

Ferramentas que usam Metamodelos UML

Vários pesquisadores (AYED; DELANOTE; BERBERS, 2007; de Farias et al., 2007; SHENG; BENATALLAH, 2005; KAPITSAKI et al., 2009; SIMONS; WIRTZ, 2007) destacam a importância do uso de metamodelos UML para modelagem de contexto. A definição de modelos estruturais e comportamentais de contexto por meio de um formalismo já consolidado auxilia o desenvolvimento de ferramentas para sua modelagem e

[4] http://www.java.com

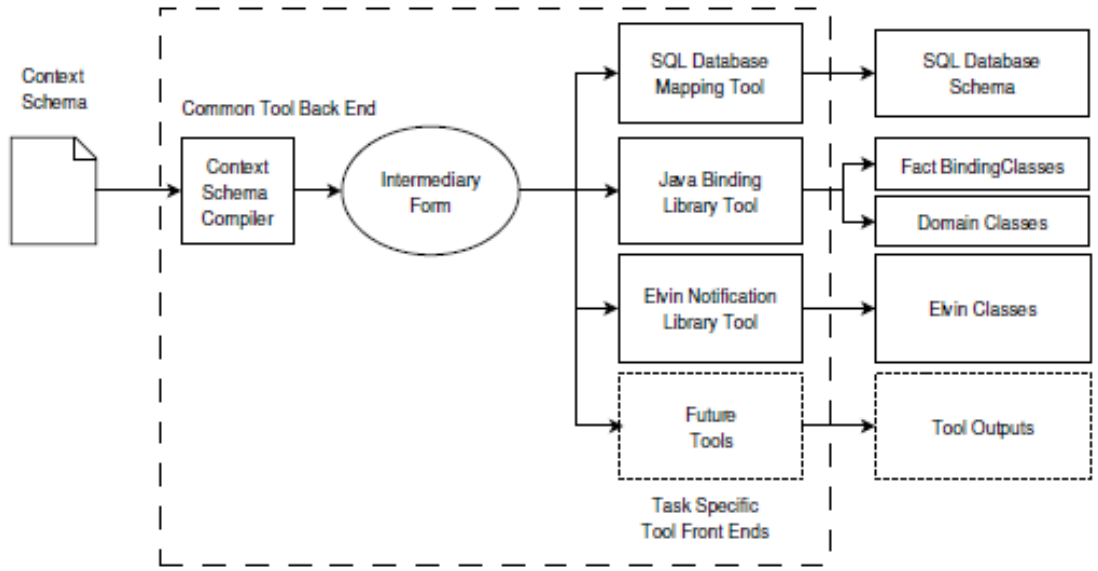

Figura 3.5 Arquitetura da ferramenta CML (MCFADDEN; HENRICKSEN; INDULSKA, 2004).

implementação. Além disto, o uso da UML permite o desenvolvimento de aplicações independentes de domínio e facilita a compreensão dos artefatos de contexto pelos engenheiros de *software* e outros interessados envolvidos no desenvolvimento de um SSC. Abaixo, discutimos dois trabalhos que usam metamodelos UML para a construção de SSC.

Ayed, Delanote e Berbers (2007) propõem um perfil (*profile*) UML que permite especificar o contexto que afeta uma aplicação e suas variações de comportamento. O objetivo é possibilitar o projeto de SSC independente de plataforma usando *Model Driven Development* (MDD) (MELLOR; CLARK; FUTAGAMI, 2003), haja vista que detalhes técnicos de implementação inibem o reuso de ferramentas já desenvolvidas.

Um conjunto de fases, que parte da especificação de contexto até a geração de código para plataformas específicas, para o desenvolvimento de sistemas sensíveis ao contexto é proposto por Ayed, Delanote e Berbers (2007). O perfil UML associado à abordagem de Ayed, chamado de *Context-Aware Profile* provê extensões ao diagrama de classes e ao diagrama de sequência e, está organizado em seis pacotes: o *StaticContextIdentification* permite identificar os tipos de informações contextuais usadas na aplicação; *StaticAdaptationAspect* e *DynamicAdaptationAspect* modelam os aspectos comportamentais de contexto; o *StaticContextMechanisms* identifica os mecanismos e as tecnologias para

a coleta de informações contextuais; e *StaticAdaptationMechanisms* e *DynamicAdaptationMechanisms* abstraem os mecanismos de adaptação da aplicação.

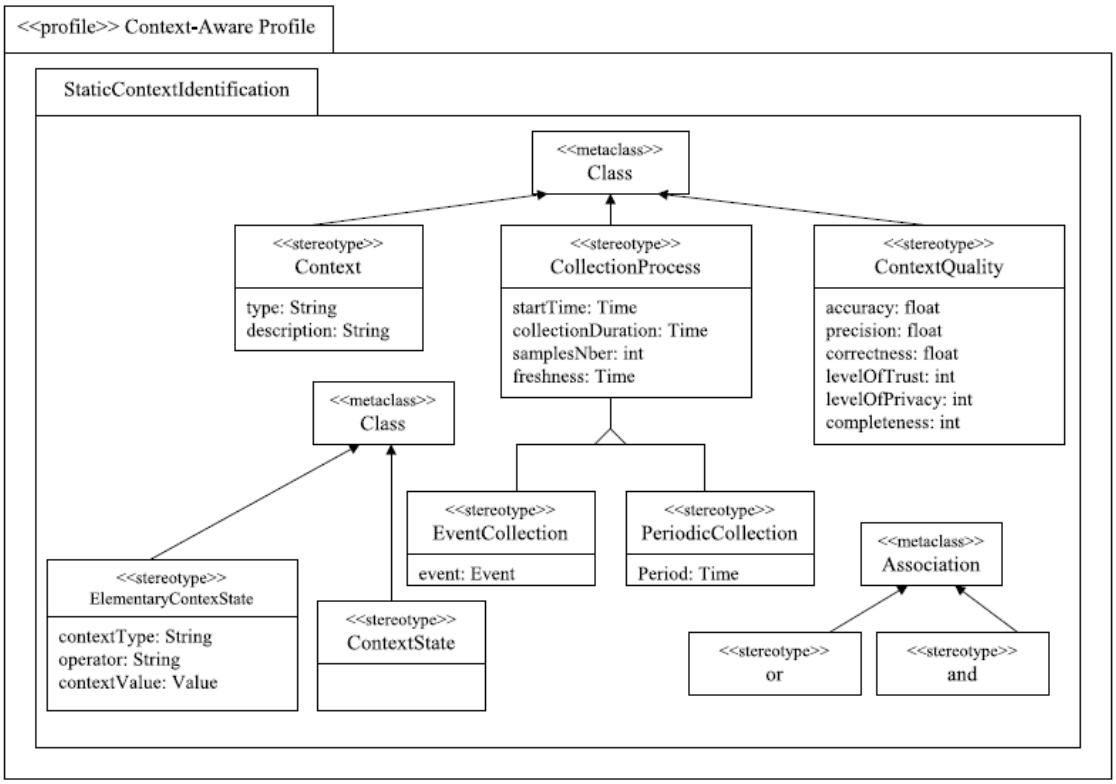

Figura 3.6 Extensões do diagrama de classes da UML para modelar contexto (AYED; DELANOTE; BERBERS, 2007).

O *StaticContextIdentification* é apresentado na Figura 3.6 e possui os seguintes elementos:

- O estereótipo **«Context»** descreve o tipo de contexto, como preferências do usuários e largura de banda da rede;
- O estereótipo **«CollectionProcess»** especifica os requisitos de aquisição de contexto, como o número de amostras (*samplesNber*). Esse estereótipo é especializado em dois tipos: baseado em evento (**«EventCollection»**) e baseado em período (**«PeriodicCollection»**);
- O estereótipo **ContextQuality** representa atributos de qualidade de contexto que devem ser satisfeitos, como precisão (*precision*) e nível de confiança (*levelOfTrust*); e

- O estereótipo **ContextState** especifica os estados de contexto relevantes para a aplicação.

Usar uma abordagem MDD permite a geração de código a partir de modelos e especificações da aplicação, aumentando a produtividade no seu desenvolvimento. Entretanto, o conjunto de fases para construção de SSC apresentado por Ayed, Delanote e Berbers (2007) não especifica artefatos de entrada e saída e guias.

Sheng e Benatallah (2005) propõem uma extensão à UML chamada de ContextUML destinada ao projeto de serviços *context-aware* (Context-Aware Service (CAS)). Através do metamodelo da ContextUML é possível especificar dois aspectos de um sistema sensível ao contexto: i) mecanismos de aquisição de contexto; e ii) mecanismos de uso e percepção de contexto em SSC.

Os seguintes elementos foram definidos para a aquisição de contexto: *Context* e *ContextSource*. A percepção de contexto ocorre de dois modos: pela invocação de um serviço baseada na modificação das informações contextuais ou pela amarração (*binding*) entre um contexto e um valor referenciado por um objeto na execução de algum comportamento contextual. Essas abordagens são especificadas por meio dos elementos *ContextTriggering* e *ContextBinding*, respectivamente.

3.2.2 Discussão sobre os trabalhos relacionados

Os trabalhos apresentados auxiliam a construção de sistemas sensíveis ao contexto usando diferentes abordagens. A Tabela 3.2 apresenta uma análise comparativa das ferramentas investigadas de acordo com os seguintes parâmetros:

- **Usuários:** identifica quem usa a ferramenta;
- **Simulação:** verifica o apoio à simulação ou depuração dos modelos construídos. Simular a execução de comportamentos contextuais em resposta às mudanças das informações de contexto antes de disponibilizar a aplicação aos usuários minimiza a quantidade de erros e antecipa a resolução deles;
- **Verificação:** analisa o apoio à verificação sintática e semântica dos modelos;
- **Domínio:** identifica os domínios de aplicações que podem ser construídas por meio da ferramenta. A proposta pode ser empregada em um domínio particular ou

ser independente dele;

- **Implementação:** verifica se a proposta gera uma aplicação executável;
- **Processo:** identifica se a construção dos artefatos é guiada por algum processo de *software*.
- **Meta-Ferramenta:**: identifica qual meta-ferramenta foi usada para construir a abordagem indicada. O uso de meta-ferramentas auxilia a adição de funcionalidades e reduz a curva de desenvolvimento de uma ferramenta CASE.

Tabela 3.2 Ferramentas para desenvolvimento de SSCs

	Usuários	**Simulação**	**Verificação**	**Domínio**	**Implementação**	**Processo**	**Meta-ferramenta**
CAPpella	Usuários finais	√		Independente de domínio	√		
iCAP	Usuários finais	√		Independente de domínio	√		
Pópulo	Engenheiros de *software*	√	√	Aplicações Pervasivas			Eclipse
CADD	Engenheiros de *software*		√	Usado para filtrar dados em aplicações móveis		√	
CML	Engenheiros de *software*		√	Independente de domínio		√	
Abordagem UML (Ayed 2007)	Engenheiros de *software*		√	Independente de domínio		√	
ContextUML	Engenheiros de *software*			*Web-Services* sensíveis ao Contexto			

As duas primeiras abordagens (CAPpella e iCAP) permitem que usuários finais criem aplicações sensíveis ao contexto independentes de domínio. Ambas permitem a simulação dos modelos criados e apóiam a execução dos sistemas produzidos. A CAPpella usa reconhecimento de padrões, necessitando uma nova fase de treinamento para a adição de uma nova situação de contexto. O iCAP possui um modo de simulação que permite definir manualmente dados de sensores e um modo de execução que configura

dispositivos reais para captura e execução de ações. O propósito de apoiar usuários finais, sem conhecimentos prévios em programação e contexto, na construção dos SSC inibe a definição de processos de software para o desenvolvimento desses sistemas.

Os outros trabalhos são destinados ao projeto de sistemas sensíveis ao contexto pelos analistas e projetistas de sistemas, reunidos no papel de engenheiros de *software*. A abordagem usando Programação Orientada a Aspectos (POA) (FUENTES; GáMEZ; SáNCHEZ, 2009) é usada no domínio das aplicações pervasivas. A ferramenta Pópulo permite a verificação e depuração dos modelos executáveis criados. A ferramenta CADD possui mecanismos de verificação da corretude da árvore dimensional de contexto e gera um dicionário de contexto usado por um servidor de dados.

As ferramentas definidas para auxiliar o projeto de SSCs por meio da CML permitem a verificação do modelo de contexto por meio de um compilador e a geração de classes, interfaces e banco de dados usados pelo módulo de gerenciamento de contexto.

As outras duas abordagens usam perfis UML. Ayed, Delanote e Berbers (2007) seguem uma metodologia baseada em MDD e permite a verificação dos modelos gerados. Um conjunto de fases guia o desenvolvimento de uma aplicação ciente de contexto, mas não detalha as atividades e tarefas com seus artefatos de entrada e saída. A ContextUML descreve um metamodelo para especificar *web-services* cientes de contexto. Todavia, não existe processo, nem apoio à implementação e verificação dos modelos.

Apenas a ferramenta Pópulo usa uma meta-ferramenta para sua construção. As demais foram construídas sem levar em consideração funcionalidades comuns já desenvolvidas por terceiros.

3.2.3 Ferramentas CASE implementadas com Metamodelos UML

A construção de ferramentas CASE usando metamodelos UML pode ser identificada em domínios como o desenvolvimento de aplicações para Internet (KOCH; KRAUS, 2003). Esta seção apresenta a ferramenta ArgoUWE para exemplificar a extensão de uma ferramenta CASE por meio de metamodelos.

A ferramenta ArgoUWE auxilia o projeto de sistemas para Internet por meio da metodologia UML-based Web Engineering (UWE) que provê uma abordagem sistematizada para o desenvolvimento de aplicações Web e complementa a modelagem dos seus aspectos conceituais, de navegação e de apresentação.

Segundo os autores, o projeto de aplicações Web deve: ser baseado em um padrão reconhecido, definir um processo de desenvolvimento e prover ferramentas que permitam o projeto dirigido por modelos e a posterior geração automática da aplicação. Para tanto, foi proposto um processo iterativo e incremental denominado de UWE que cobre todo o ciclo de vida das aplicações Web focando no projeto e geração automática delas. UWE usa um perfil UML, composto por estereótipos, propriedades e restrições, para modelar seus artefatos. O processo provê guias e um conjunto de passos para a construção dos três modelos abaixo:

- **Modelo Conceitual:** captura os requisitos da aplicação por meio dos diagramas de atividades e de casos de uso;
- **Modelo de Navegação:** derivado do modelo conceitual, representa a especificação dos objetos que podem ser navegados através da aplicação Web e como esses objetos são estruturados. Os seguintes estereótipos podem ser usados: «navigation class», «direct navigability», «index», «guided tour», «query» e «menu»; e
- **Modelo de Apresentação:** descreve como os objetos de navegação definidos acima serão apresentados ao usuário. Os seguintes estereótipos são usados para abstrair elementos da interface gráfica: «text», «form», «button», «image», «audio», «anchor», «collection» e «anchored collection».

A ferramenta ArgoUWE é um plugin do ArgoUML (ver seção 3.1) que permite a construção dos três modelos e verifica a consistência deles por meio de restrições definidas em OCL. Inicialmente o modelo conceitual é construído e em seguida os outros dois são gerados de maneira semi-automática com base nos modelos definidos anteriormente. Os três diagramas foram implementados por meio de especializações de classes do ArgoUML.

Neste exemplo podemos verificar que modelos que adotam uma notação definida como uma extensão conservativa de um metamodelo UML, na qual os elementos de modelagem da UML não são modificados, mas especializados por meio de um perfil UML, podem ter suas construções auxiliadas por ferramentas CASE de metamodelagem UML.

A abordagem usada na construção do ArgoUWE permite a adição de um perfil UML a uma ferramenta CASE existente, diminuindo seu custo e tempo de desenvolvimento. Portanto, é possível construir ferramentas CASE por meio de meta-ferramentas.

3.3 Considerações Finais

Este capítulo apresentou o conceito de ferramentas CASE e sua importância nas tarefas de engenharia de *software*. Vimos também, a necessidade de ferramentas de apoio ao projeto e ao desenvolvimento de sistemas sensíveis ao contexto. Foram apresentados alguns trabalhos relacionados à proposta desta dissertação, no entanto, nenhum deles permite a implementação de SSCs independentes de domínio guiada por um processo de *software* por meio da adição de informações contextuais aos modelos de análise e projeto da aplicação.

O desenvolvimento de uma ferramenta CASE é custoso e envolve várias funcionalidades periféricas, como a criação de interfaces gráficas e a persistência dos modelos gerados. Uma solução, exemplificada na ferramenta ArgoUML, que diminui a curva de implementação é a extensão de meta-ferramentas por meio de metamodelos UML.

Capítulo 4

CEManTIKA CASE

Entre os trabalhos analisados no Capítulo 2 e Capítulo 3, o CEManTIKA destaca-se por apresentar um metamodelo, uma arquitetura genérica e um processo de *software* (Seção 2.4). Todavia, o CEManTIKA, por ser apenas conceitual, não apoia questões de implementação ligadas ao desenvolvimento de SSC. Também, experimentos apresentados por Vieira (2008) evidenciaram dois problemas no uso do *framework* conceitual CEManTIKA: a compreensão do Processo de Projeto de SSC e a criação dos artefatos definidos nas suas tarefas. Buscando resolver esses problemas, este trabalho propõe uma ferramenta CASE de apoio à construção de SSC usando os conceitos da arquitetura, metamodelo e processo definidos no *framework* conceitual CEManTIKA. Esta ferramenta recebe, pois, o nome de CEManTIKA CASE.

A ferramenta CEManTIKA CASE está dividida em dois módulos: **CEManTIKA Process** e **CEManTIKA Modeling**. O primeiro auxilia o entendimento dos conceitos do *framework* descrevendo suas atividades, tarefas e artefatos. O segundo módulo apoia a modelagem de SSC e dá suporte a construção dos artefatos definidos em cada tarefa do Processo de Projeto de SSC.

Esta separação tem o objetivo de oferecer ferramentas distintas considerando o nível de conhecimento dos engenheiros de *software* em relação aos conceitos definidos no CEManTIKA. Os iniciantes em contexto podem usar o *CEManTIKA Process* para compreender o Processo de Projeto de SSC antes de projetar uma aplicação e, só depois, usar o *CEManTIKA Modeling* para modelar os elementos de contexto do SSC.

Este capítulo apresenta a especificação da ferramenta CEManTIKA CASE. A Se-

ção 4.1 descreve o *framework* conceitual CEManTIKA; a Seção 4.2 apresenta o CEManTIKA Process; a Seção 4.3 apresenta o CEManTIKA Modeling; e, por fim, a Seção 4.4 termina o capítulo com algumas considerações finais.

4.1 CEManTIKA

Sistemas sensíveis ao contexto percebem o seu ambiente de execução e se adaptam a ele. No entanto, sua construção apresenta desafios na aquisição, processamento e disseminação de informações contextuais. O CEManTIKA *(*Contextual Elements Modeling and Management through Incremental Knowledge Acquisiton*)* (VIEIRA et al., 2007; VIEIRA, 2008) permite modularizar o desenvolvimento de SSC separando os elementos relacionados ao negócio da aplicação daqueles relacionados ao contexto.

O CEManTIKA possui três objetivos principais: i) auxiliar o projeto dos elementos arquiteturais relacionados à manipulação de contexto; ii) auxiliar a especificação e representação de contexto independentemente de domínio da aplicação; e iii) auxiliar os engenheiros de *software* na modelagem e projeto de sistemas sensíveis ao contexto. Para atingir esses objetivos o CEManTIKA está dividido em *Arquitetura de Manipulação de Contexto* (Seção 4.1.1), *Metamodelo de Contexto* (Seção 4.1.2) e *Processo de Desenvolvimento de SSC* (Seção 4.1.3). Os elementos do CEManTIKA (Figura 4.1) apoiam o projeto de sistemas sensíveis ao contexto por meio de um processo dividido em tarefas relacionadas a uma arquitetura genérica de *software*. Essa mesma arquitetura identifica os elementos de um metamodelo, que é usado para criar os artefatos de contexto durante as tarefas do processo, descrito em um perfil UML.

4.1.1 Arquitetura de Manipulação de Contexto

Farias (de Farias et al., 2007) afirma que o desenvolvimento de aplicações sensíveis ao contexto lida com diversos desafios tecnológicos e necessita de uma infraestrutura para facilitar sua construção, instalação e execução. Ainda, é necessário tratar diferentes fontes de aquisição e tipos de informações contextuais providos por ambientes heterogêneos e em constante mudança.

Vieira (VIEIRA; TEDESCO; SALGADO, 2009) classifica as tarefas relacionadas

Figura 4.1 Integração entre os elementos do CEManTIKA (VIEIRA, 2008).

ao desenvolvimento de SSC em três categorias principais, considerando o desenvolvimento de SSC a partir de uma perspectiva independente de domínio e outra dependente de domínio:

- **Especificação de Contexto:** trata da definição dos elementos contextuais e das variações comportamentais relacionadas a eles.
- **Gerenciamento de Contexto:** especifica como contexto será manipulado. É composto pelas tarefas de aquisição, armazenamento, processamento e disseminação. É responsável por recuperar informações contextuais, processá-las e identificar comportamentos sensíveis ao contexto.
- **Uso de Contexto:** define como as variações do contexto influenciam a aplicação. O contexto pode apresentar informações e serviços ao usuário, dar assistência na execução de uma tarefa, adaptar recursos, executar um serviço, entre outros usos relacionados ao domínio da aplicação.

O *Gerenciamento* de Contexto atua como intermediador entre a *Especificação* e o *Uso* de Contexto. As três categorias formam a base da Arquitetura de Contexto (Figura 4.2), composta pelos seguintes módulos:

- **Fontes de Contexto:** são os componentes responsáveis pela aquisição dos ECs. Um SSC pode demandar a criação de novas fontes de contexto ou usar uma existente. Para permitir compatibilidade, cada fonte de contexto deve implementar

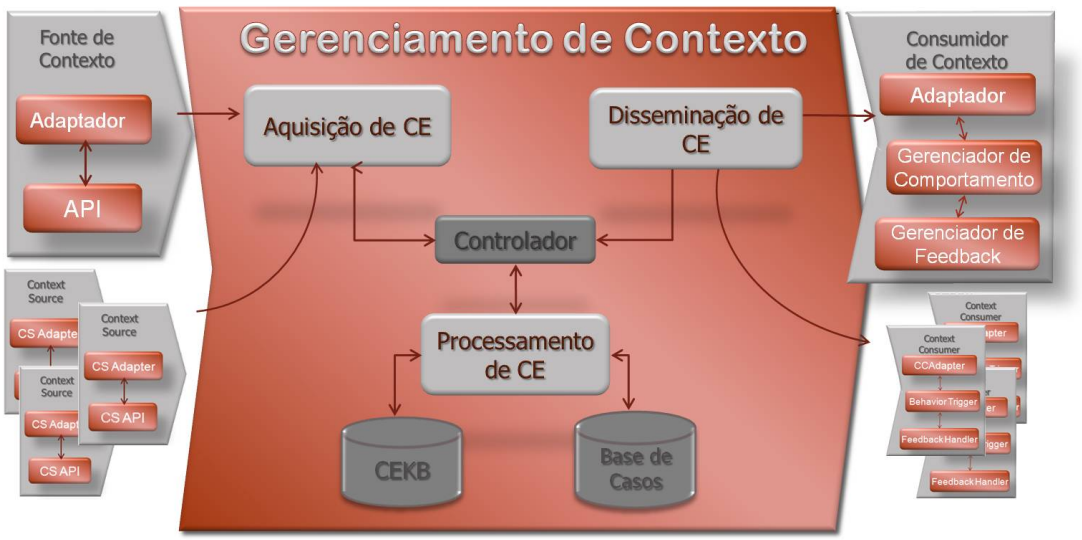

Figura 4.2 Arquitetura de Contexto (VIEIRA, 2008).

dois módulos: API e Adaptador. O primeiro implementa as funcionalidades internas para aquisição de informações contextuais, permitindo acessá-las por várias aplicações. O último implementa o Padrão de Projeto Adaptator (GAMMA et al., 1995) para converter a informação contextual no formato esperado pelo Gerenciador de Contexto.

- **Gerenciamento de Contexto:** é dividido em quatro módulos principais. O *Módulo Controlador* [1] gerencia as atividades de aquisição, processamento, aquisição, disseminação e armazenamento de contexto. O *Módulo Aquisição de ECs* é responsável por registrar, ativar, desativar e gerenciar a comunicação entre as fontes de contexto. O *Módulo de Processamento* usa a base de conhecimento (CEKB), que armazena sentenças do mundo real, para processar e derivar informações contextuais relacionados a um foco. Por fim, o *Módulo de Disseminação* permite a comunicação entre o Gerenciador de Contexto e os Consumidores de Contexto.

- **Consumidores de Contexto:** são partes do *software* que mudam seu comportamento de acordo com o contexto. Um SSC pode ter vários consumidores responsáveis por diferentes focos. Compreende os elementos *Adaptador*, *Trigger de Comportamento* e *Gerenciador de Feedback*. O primeiro, especifica um protocolo de comunicação entre os consumidores e o Gerenciador de Contexto, abstraindo funcionalidades internas do Gerenciador. O segundo é responsável por disparar os

[1] Veja o padrão de projeto MVC. http://java.sun.com/blueprints/patterns/MVC-detailed.html

comportamentos do SSC de acordo com o foco e as variações dos ECs. O último é responsável por explicar ao usuário o motivo da ativação do comportamento contextual e perguntar se as ações disparadas se adequadam às necessidades dos usuários.

4.1.2 Metamodelo de Contexto

Os sistemas sensíveis ao contexto geralmente são construídos com os recursos tecnológicos disponíveis no momento, não considerando aspectos de reuso e generalização. Segundo Ayed (AYED; DELANOTE; BERBERS, 2007), essa restrição inclui detalhes que são específicos a uma tecnologia e demanda esforços de reimplementação para outras plataformas. O CEManTIKA propõe um metamodelo independente de domínio para descrever os elementos estruturais e comportamentais de um SSC. O *Context Metamodel* define conceitos que abstraem e permitem a manipulação de modelos contextuais, e possui os seguintes objetivos:

- oferecer um *framework* conceitual que identifique os principais conceitos relacionados ao uso e manipulação de contexto, independente de domínio;
- apoiar o reuso e extensão de modelos de contexto existentes ou modelos de aplicação.

Para atingir esses objetivos, o metamodelo é implementado como extensão do UML 2.0 *Metamodel* [2], seguindo a semântica e a notação padrões da UML. O metamodelo é organizado em dois perfis (*profile*) UML:

- `Context Profile` é o metamodelo estrutural e descreve os conceitos relacionados aos elementos estruturais e conceituais do *software* pela extensão do Diagrama de Classes e de Casos de Uso;
- `CxG Profile` é o metamodelo comportamental e permite ao projetista construir os modelos comportamentais de um SSC por meio do Diagrama de Atividades.

A Figura 4.3 apresenta o metamodelo estrutural definido no *Context Profile* por meio da notação e semântica do Diagrama de Classes. Os principais conceitos são:

[2] http://www.omg.org/spec/UML/

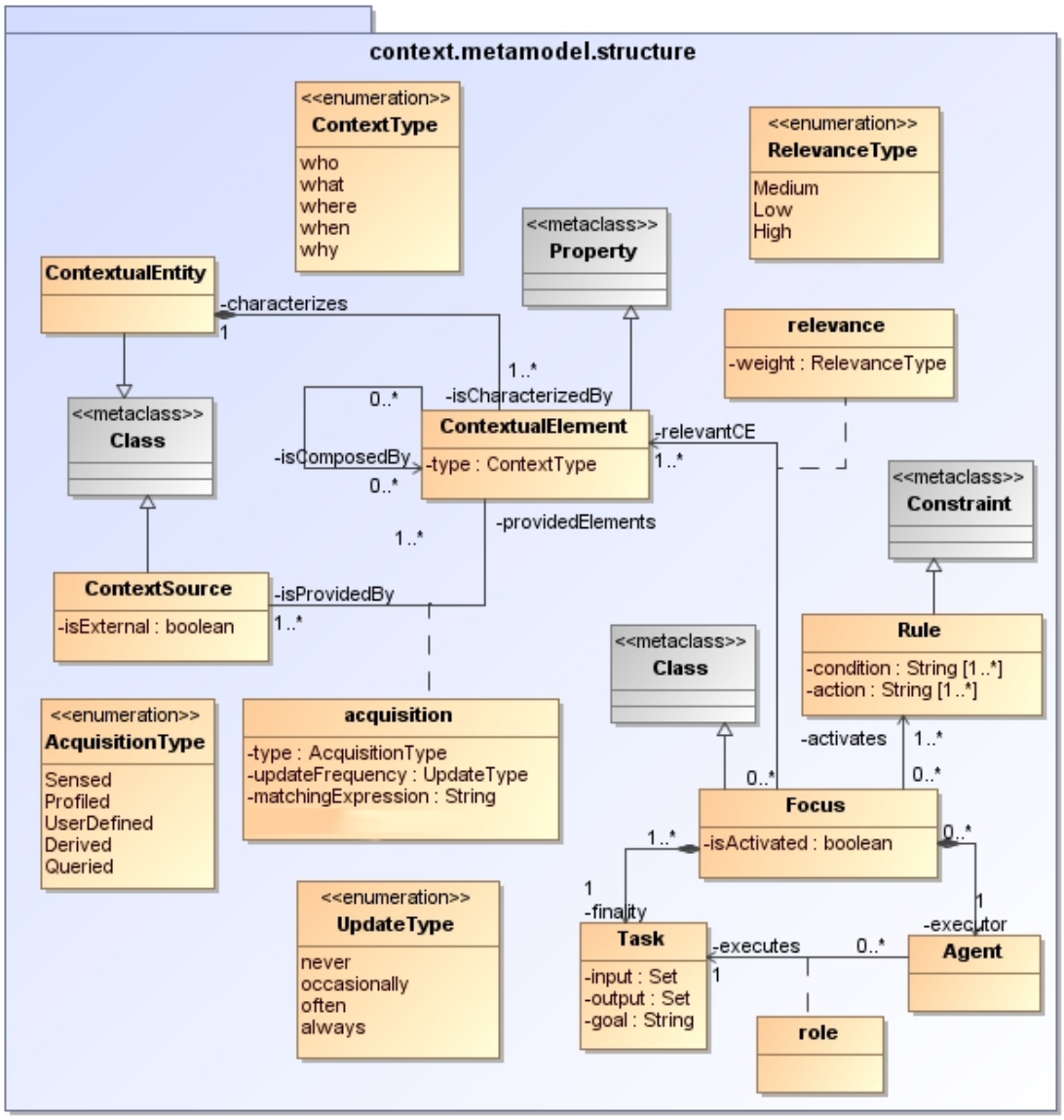

Figura 4.3 Metamodelo estrutural de Contexto (VIEIRA, 2008).

- **ContextualEntity**: (entidade contextual) representa as entidades que devem ser consideradas para manipulação de contexto. Deve possuir ao menos um *ContextualElement*. Uma entidade representa um objeto do mundo real relevante para descrever um domínio;

- **ContextualElement**: (elemento de contexto) é a unidade elementar do metamodelo e representa uma propriedade usada para caracterizar uma Entidade Contextual. Possui um dos seguintes tipos (Seção 2.1): *who*, *what*, *when*, *where* ou *why*;

- **ContextSource**: (fonte contextual) descreve as fontes de contexto responsáveis pela aquisição dos elementos contextuais. Uma associação de aquisição (*acquisition association*) descreve os atributos tipo e frequência de atualização (*type* e *updateFrequency*, respectivamente) entre um Elemento Contextual e uma Fonte de Contexto. O tipo pode ser percebido (*Sensed*), extraído de perfis (*Profiled*), definido pelo usuário (*User Defined*), derivado (*Derived*) ou consultado (*Queried*) e a frequência de atualização assume um dos seguintes valores: nunca (*Never*), ocasionalmente (*Occasionally*), frequentemente (*Often*) ou sempre (*Always*).

- **Focus**: (foco) é o que permite determinar quais elementos contextuais devem ser usados para compor o contexto. Representa uma Tarefa (*Task*) executada por um Agent (*Agent*) de *software* ou humano. Uma relação de relevância (*relevance association*) entre o Foco e cada Elemento Contextual descreve sua importância através do atributo peso (*weight*). O peso pode assumir os valores Baixo (*Low*), Médio (*Medium*) ou Alto (*High*).

- **Rule** (regra): é representado com um conjunto de condições (*conditions*) e um conjunto de ações (*actions*). Cada condição especifica uma expressão que retorna um valor verdadeiro, falsou ou desconhecido quandos os dados estão disponíveis. Uma ação indica um procedimento que deve ser executado quando as condições forem satisfeitas. Em um SSC uma ação inclui um gatilho para um comportamento e a atribuição de um novo peso em uma associação de relevância entre um foco e um elemento contextual.

O metamodelo estrutural de contexto é implementado por meio do perfil UML apresentado na Figura 4.4, denominado *context profile*. Um perfil permite a definição de extensões por meio de estereótipos, propriedades (*tags*) e restrições (*constraints*). Dessa forma, é possível moldar a UML para um domínio particular, como modelagem de processos de software [3].

O *context profile* (Figura 4.4) possui extensões que permitem enriquecer os modelos de Casos de Uso e de Classes com informações contextuais. Para o modelo de Casos de Uso são definidos os estereótipos *«Agent»*, *«Task»* e *«executes»* que são adicionados a atores, casos de uso e associações, respectivamente, para descrever um foco. Para o modelo de Classes os seguites estereótipos são especificados:

- *«Contextual Entity»*: aplicado a classes para especificar as Entidades Contextuais;

[3] http://www.omg.org/technology/documents/formal/spem.htm

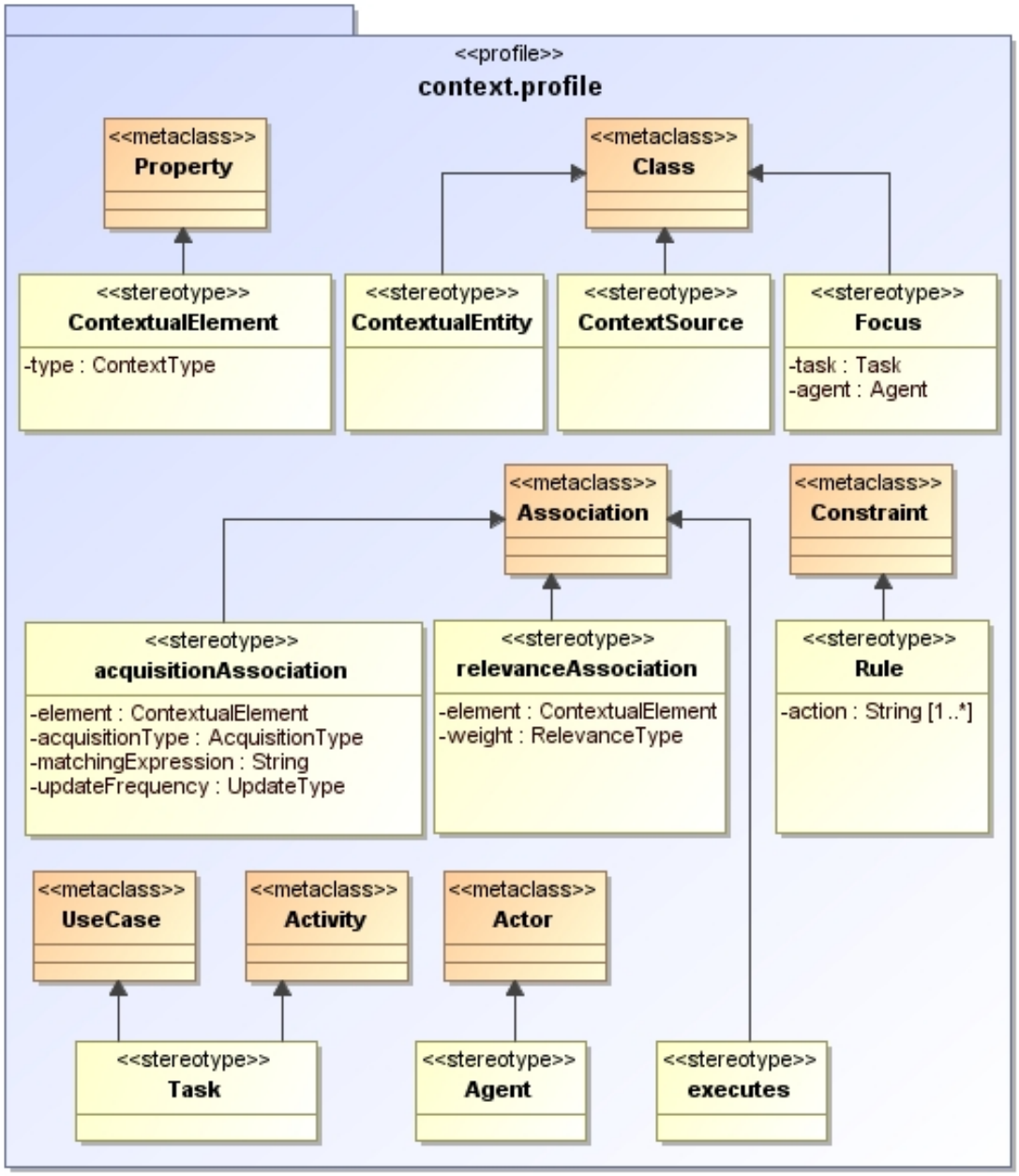

Figura 4.4 Perfil de Contexto com propriedades e estereótipos (VIEIRA, 2008).

- *«Contextual Element»*: aplicado a atributos ou associações de uma Entidade Contextual que representam Elementos Contextuais;
- *«Context Source»*: aplicado a classes que implementam Fontes de Contexto;
- *«acquisition Association»*: modela a associação entre um Elemento Contextual e uma Fonte de Contexto;

- *«Focus»*: aplicado a classes no modelo de contexto que descrevem um Foco;
- *«relevance Association»*: define uma relação de relevância (peso) entre um Foco e um Elemento Contextual;
- *«Rule»*: indica regras de comportamento para o modelo de contexto. A propriedade *action* representa o que será executado quando todas as condições definidas em uma *Contraint* forem satisfeitas.

O metamodelo comportamental está relacionado com os aspectos dinâmicos de manipulação de contexto, ou seja, ele é responsável por identificar a relevância entre um EC e um Foco (*CERelevance*) e as regras que indicam como o sistema deve ser executado de acordo com as variações de contexto (*Rule*).

Os modelos comportamentais são construídos por meio de grafos contextuais (Seção 2.3.1). Cada caminho do grafo contém a definição de uma regra de produção que indica como o contexto atua no comportamento da aplicação. Para o exemplo da Figura 2.2, uma das regras é identificada no Exemplo 4.1:

Lista de Códigos Fonte e Exemplos 4.1 Exemplo de regra de produção criada

```
Rule 1:
  Conditions
    not(Cliente.cidade = Missão.cidade)
    Missão.quemPaga = CAPES
  Actions
    CallBehavior("Contactar Agência CAPES")
```

O metamodelo UML (CxG Profile) permite o projeto dos aspectos comportamentais de um SSC por meio do diagrama de Atividades com a semântica definida nos grafos contextuais.

4.1.3 Processo de Projeto de SSC

O *CSS Design Process*, descrito na Figura 4.5, compreende as principais atividades para o desenvolvimento de SSC: *Context Specification* (Especificação de Contexto), *Context Management Design* (Projeto de Gerenciamento de Contexto), *Context Usage Design* (Projeto de Uso de Contexto), *Code Generation* (Geração de Código), *Testing* (Teste) e *CSS Evaluation* (Avaliação). As atividades e tarefas do processo usam o metamodelo e

a arquitetura de contexto explicadas anteriormente. Apenas as três primeiras atividades foram definidas. O processo adiciona à equipe de desenvolvimento de software o papel do projetista de contexto que deve possuir habilidades multidisciplinares relacionadas à usabilidade, privacidade, Inteligência Artificial, aquisição de dados de sensores, etc.

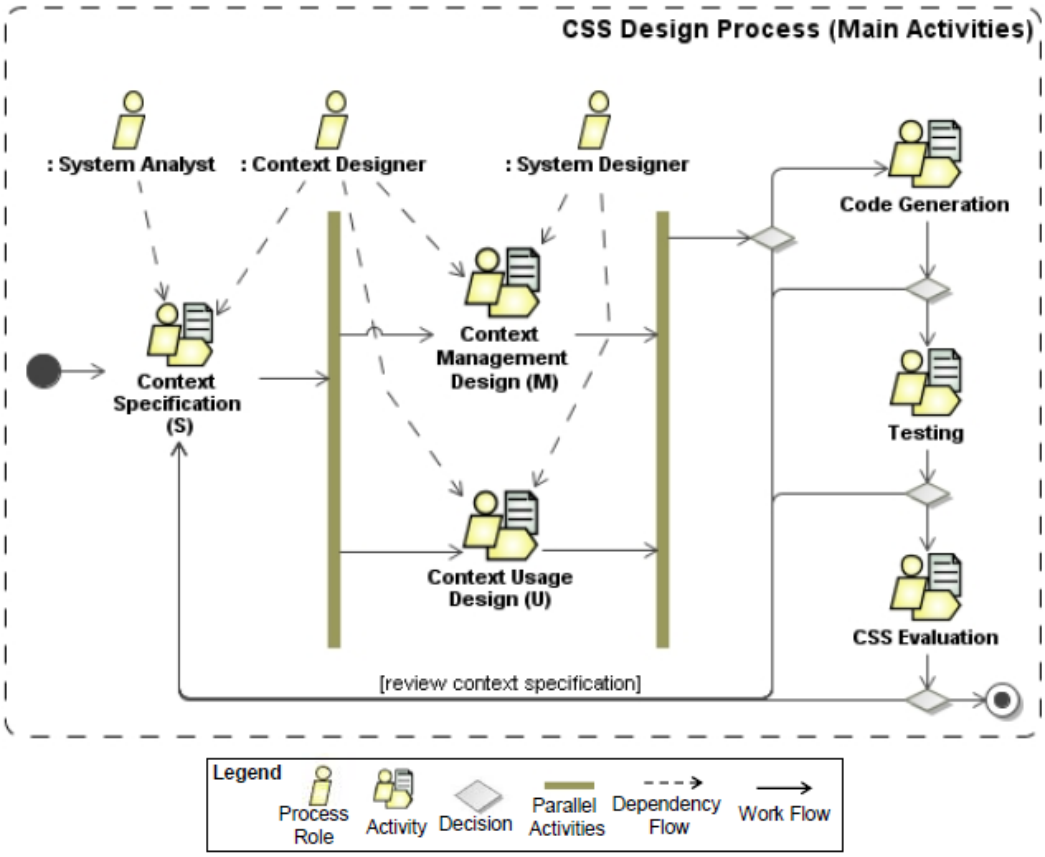

Figura 4.5 Processo de Desenvolvimento de SSC (VIEIRA, 2008).

A Especificação de Contexto identifica os requisitos de contexto e cria os modelos conceitual e comportamental de contexto com base nos requisitos de negócio da aplicação. É formada pelas seguintes tarefas: Identificar Focos (S1), Identificar as Variações de Comportamento (S2), Identificar Entidades Contextuais e Elementos Contextuais (S3), Projetar o Modelo Comportamental de Contexto (S4) e Verificar a Relevância dos Elementos Contextuais (S5).

Após identificar os requisitos de contexto e os modelos conceitual e comportamental de contexto, o projetista deve investigar como os ECs devem ser adquiridos, processados e compartilhados na atividade Projeto de Gerenciamento de Contexto, que é formada pelas seguintes tarefas: Especificar Aquisição de Contexto (M1), Projetar o Módulo de

Aquisição (M2), Projetar o Módulo de Processamento (M3) e Projetar o Módulo de Disseminação (M4).

Em seguida, o Projeto de Uso de Contexto tem por objetivo projetar como o contexto será efetivamente usado no SSC. O processo divide o uso de contexto em duas categorias: adaptação do comportamento da aplicação e percepção dos usuários com informações contextuais gerenciadas pelo sistema. A atividade é formada pelas seguintes tarefas: Projetar Adaptação de Contexto (U1) e Projetar Apresentação de Contexto (U2).

4.1.4 Alterações no Processo de Projeto de SSC

A identificação dos elementos contextuais que devem ser usados em um SSC está relacionada às mudanças desejadas no comportamento da aplicação em função das alterações nos valores desses elementos. No CEManTIKA, a atividade de Especificação de Contexto tem o objetivo de identificar os requisitos de contexto a partir dos requisitos de negócio da aplicação e criar o modelo conceitual de contexto enriquecendo o modelo conceitual da aplicação com os elementos definidos no Context Profile (Seção 4.1.2). Por exemplo, numa aplicação de indicação de filmes, a idade do usuário é um elemento contextual que permite variações no comportamento da mesma.

Na versão inicial do CEManTIKA (VIEIRA, 2008), a tarefa Projetar Modelo Comportamental de Contexto, inserida na atividade de Projeto de Uso de Contexto, permite identificar, para cada foco, como as mudanças do contexto afetam um SSC por meio de adaptações no comportamento da aplicação e mecanismos de percepção. O modelo comportamental de contexto é representado como um grafo contextual, que possui ações que devem ser acionadas durante a execução do SSC e condições que restringem a execução dessas ações, para cada foco. No entanto, durante a análise do modelo comportamental de contexto é necessário evoluir o modelo conceitual de contexto. Dessa forma, é interessante que os dois modelos sejam projetados num ciclo iterativo, no qual os elementos e comportamentos contextuais são refinados. Ainda, ao especificarmos o modelo de comportamento de contexto na atividade de Especificação de Contexto conseguimos projetar o Módulo de Processamento de Contexto com mais detalhes, pois já sabemos como o contexto será usado na aplicação. Dessa forma, a tarefa Projetar Modelo Comportamental de Contexto foi transferida para a atividade de Especificação de Contexto e é executada após a identificação dos elementos e entidades contextuais.

O Processo de Projeto de SSC definido no CEManTIKA explica seis atividades para

a construção de um SSC (seção 4.1.3). A ferramenta CEManTIKA CASE, especificada neste capítulo, foca na atividade *Especificação de Contexto* que permite a identificação dos elementos contextuais e produz os modelos estruturais e comportamentais de contexto.

4.2 CEManTIKA Process

O processo de desenvolvimento de *software* é dinâmico e deve adaptar-se a vários parâmetros, como: *expertise* da equipe, natureza do problema e necessidade do cliente. Várias abordagens tentaram definir formas para construir aplicações de alta qualidade considerando três variáveis: tempo, custo e escopo. Entre elas, destacam-se os processos mais rígidos como o Rational Unified Process (RUP) e as metodologias ágeis que evidenciam características de auto-organização e cooperação do time.

Como apontado em experimento realizado por Vieira (2008), o Processo de Projeto de SSC deve adaptar-se ao nível de *expertise* dos projetistas de SSCs, ou seja, o processo deve prover exemplos e guias para os iniciantes e usar métodos mais ágeis para os mais experientes.

Para apoiar os engenheiros de processo na integração entre as tarefas relacionadas à contexto e outras com ênfase nas regras de negócio da aplicação e também permitir aos engenheiros de *software* navegar entre os elementos do CEManTIKA de acordo com seu nível de conhecimento, é proposta uma biblioteca de documentação composta pelos elementos necessários ao desenvolvimento de sistemas sensíveis ao contexto.

Sendo assim, o CEManTIKA *Process* tem os seguintes objetivos:

- auxiliar a compreensão das atividades, atores, artefatos e tarefas do Processo de Projeto de SSC;
- permitir a navegação no Processo de Projeto de SSC de acordo com o nível de experiência de cada engenheiro de *software*;
- publicar o Processo de Projeto de SSC em um site para simples navegação e consulta; e
- prover uma base de conhecimento extensível capaz de adaptar-se a outros processos.

Uma abordagem para customizar um processo de *software* é configurá-lo por meio do Software Process Engineering Metamodel (SPEM). SPEM é um metamodelo para engenharia de processos e um *framework* conceitual que provê os conceitos necessários para modelagem, documentação, apresentação, gerenciamento, intercâmbio e execução de métodos de desenvolvimento e processos (OMG, 2008).

O SPEM separa o conhecimento sobre métodos, ferramentas e técnicas em um processo. Papéis, tarefas, guias e artefatos são chamados de *method content*s, que explicam passo a passo como os objetivos de desenvolvimento são alcançados independente do momento nos quais são executados dentro de um ciclo de vida. Os processos relacionam os *method content*s em conjuntos de tarefas ordenados de acordo com as especificidades dos projetos e indicam quando uma tarefa deve ser executada. Por exemplo, a tarefa *Projetar Casos de Uso* pode ser customizada, considerando os papéis envolvidos e artefatos de entrada e saída, de formas diferentes para um sistema crítico e para um sistema de informação. Os *method content*s definem *como, por que, quanto e como* realizar uma tarefa e o processo identifica *quando* fazê-la.

4.3 CEManTIKA Modeling

Diferente das ferramentas de apoio ao uso de contexto apresentadas no Capítulo 3, a proposta do CEManTIKA é apoiar a construção sistematizada de SSCs por meio de um processo com papéis, tarefas e artefatos bem definidos. Portanto, o CEManTIKA Modeling possui os seguintes objetivos:

- apoiar a construção de um SSC a partir dos modelos de Casos de Uso e Conceitual definidos na aplicação; e
- configurar um único ambiente integrado de desenvolvimento (IDE) para as atividades de análise, projeto, codificação e testes de SSC.

O Digrama de Atividades na Figura 4.6 apresenta o fluxo de execução das tarefas da atividade de *Especificação de Contexto*. A primeira tarefa, Identificar Foco, é executada pelo Projetista de Contexto e possui o modelo de casos de uso e o diagrama de casos de uso como entrada e o modelo de casos de uso enriquecido como saída. O modelo de casos de uso é enriquecido com os estereótipos *«Agent»*, *«Task»* e *«executes»* para

identificar quais são os focos da aplicação. As outras tarefas são executadas de forma iterativa para cada foco encontrado. A tarefa Identificar as Variações de Comportamento descreve quais variações de comportamento na aplicação são esperadas e que fatores afetam esses comportamentos, como mecanismos de adaptação de interface gráfica para pessoas com problemas visuais. A tarefa Identificar Entidades Contextuais e Elementos Contextuais produz o modelo conceitual de contexto enriquecendo o modelo conceitual da aplicação com os estereótipos *«ContextualEntity» e «ContextualElement»*. Em seguida, o modelo comportamental de contexto é projetado por meio de um grafo contextual com as ações que devem ser executadas para determinadas condições. Por fim, a tarefa Verificar a Relevância dos Elementos Contextuais é opcional, e permite avaliar se os usuários e projetistas possuem o mesmo entendimento sobre a relevância (alta, média e baixa, por exemplo) dos elementos contextuais para um determinado foco.

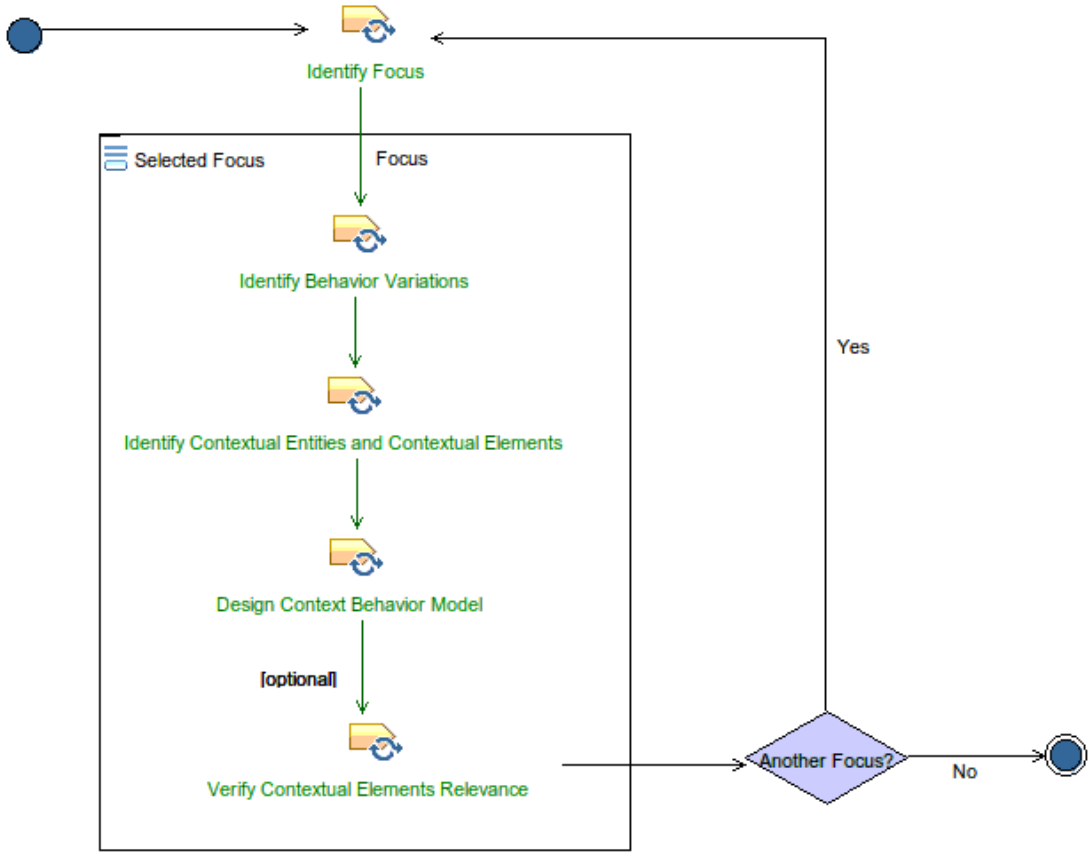

Figura 4.6 Fluxo de execução da atividade Especificação de Contexto

Embora o CEManTIKA compreenda atividades de análise, desenvolvimento e testes, a versão inicial do CEManTIKA *Modeling* especifica apenas a atividade de Especifica-

ção de Contexto e suas tarefas, seguindo os artefatos de entrada e saída descritos no diagrama de atividades detalhado da Figura 4.7. A próxima seção especifica os requisitos funcionais do CEManTIKA *Modeling* considerando esse escopo.

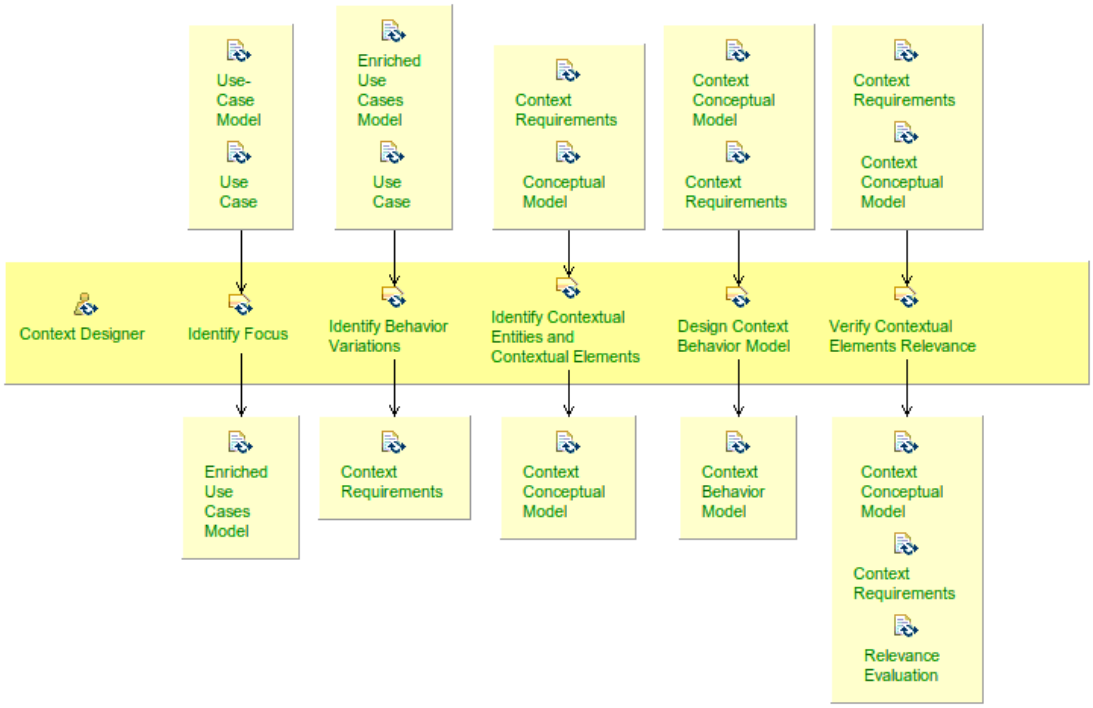

Figura 4.7 Tarefas e artefatos de entrada e saída da atividade Especificação de Contexto

4.3.1 Requisitos Funcionais

Os requisitos do CEManTIKA *Modeling* são apresentados no Diagrama de Casos de Uso da Figura 4.8. Eles correspondem à execução das seguintes tarefas do Processo de Projeto de SSC: *Identificar Foco, Identificar Variações de Comportamento, Identificar Entidades Contextuais e Elementos Contextuais, Projetar Modelo de Comportamento de Contexto e Verificar Relevância dos Elementos Contextuais*. A ferramenta CEManTIKA *Modeling* interage com dois atores externos:

- **Projetista de Contexto:** representa o usuário da ferramenta, o qual é responsável por projetar um SSC; e
- **Sistema:** executa procedimentos automatizados de verificação e geração de artefatos.

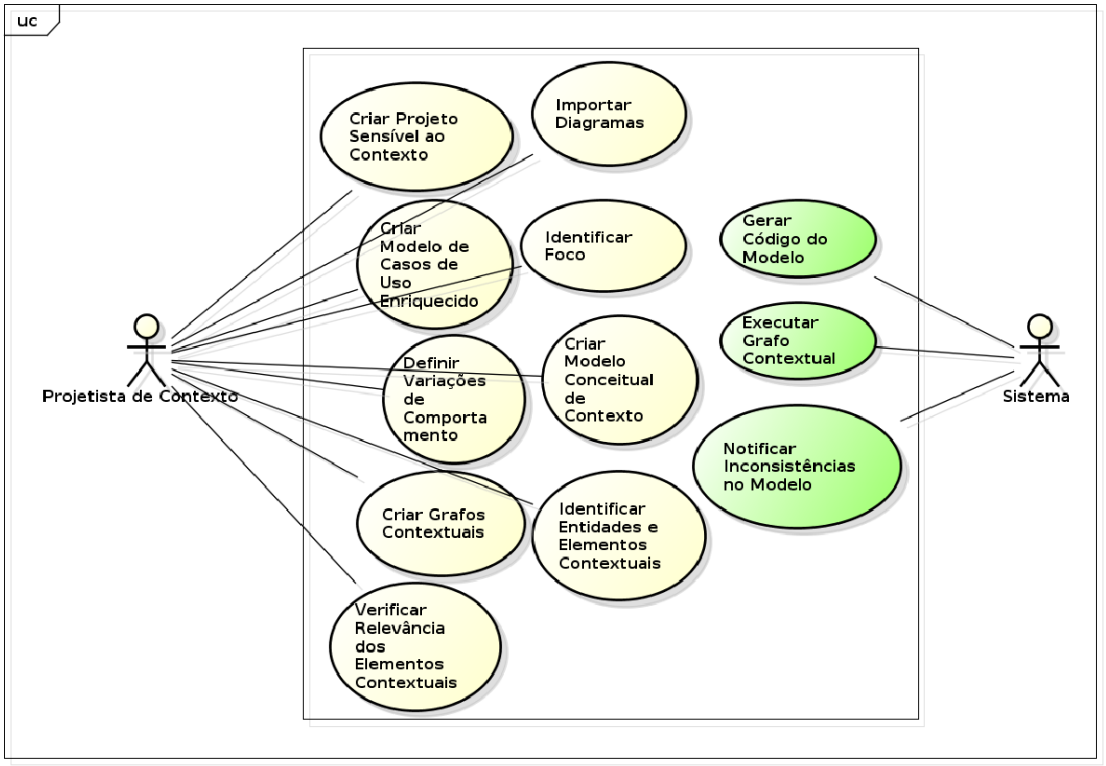

Figura 4.8 Diagrama de Casos de Uso do CEManTIKA Modeling

Para o ator *Projetista de Contexto* foram identificados os seguintes requisitos funcionais:

- **Criar Projeto Sensível ao Contexto:** permite a criação de um projeto de um sistema sensível ao contexto com as dependências necessárias para a execução dos outros requisitos. Entre as dependências, destacam-se: bibliotecas, arquivos para edição e processadores de modelos.
- **Importar Diagramas:** o usuário pode importar o Diagrama de Casos de Uso e o Modelo Conceitual criados em outra ferramenta, inibindo o retrabalho.
- **Criar Modelo de Casos de Uso Enriquecido** e **Criar Modelo Conceitual de Contexto:** o projetista pode criar os modelos de contexto, atribuindo-lhes os estereótipos e propriedades definidos no perfil UML correspondentes ao metamodelo de contexto.
- **Identificar Foco:** o usuário identifica no Diagrama de Casos de Uso Enriquecido quais requisitos devem ser considerados para compor os focos do SSC. Para esco-

lher um foco, o Projetista de Contexto deve aplicar estereótipos em atores e casos de uso criando *Agent*s e *Task*s, respectivamente e, em seguida criar uma associação do tipo *executes* entre um *Agent* e uma *Task*. Por exemplo, numa versão simplificada de um sistema de *e-commerce* com os casos de uso: Criar usuário, Listar produtos e Efetuar pagamento, apenas o segundo caso de uso será considerado na formação do foco, pois os produtos são listados de acordo com o perfil do usuário. Os demais casos de uso são executadas da mesma forma para todos os usuários. Portanto, o Usuário (ator) é um Agente (estereótipo *Agent*) que executa (estereótipo *executes*) a tarefa (estereótipo *Task*) Listar produtos.

As demais funcionalidades, definidas abaixo, são realizadas separadamente para cada foco. Dessa forma, o usuário pode restringir apenas as entidades e elementos contextuais do foco trabalhado. Essa abordagem tenta solucionar a explosão de elementos UML durante a modelagem de um SSC (classes, relacionamentos e propriedades) apresentada, por exemplo, em Chaves (2009).

- **Definir Variações de Comportamento:** o usuário do CEManTIKA *Modeling* adiciona fluxos de contexto a um caso de uso. Esses fluxos representam diferentes ações, relacionadas a um foco, que o sistema sensível ao contexto pode executar. O objetivo da funcionalidade é gerar o artefato *Requisitos de Contexto* composto pelas variações de comportamento contextual que são derivadas dos casos de uso da aplicação e adicionam fluxos de contexto ao cenário de execução de um caso de uso. As variações de comportamento estão relacionadas às mudanças de comportamento do sistema devido ao seu contexto e incluem:

  - mecanismos de percepção que fornecem informações contextuais para um agente durante a execução de uma tarefa, como informar o usuário que um novo relatório está disponível no repositório de trabalho;
  - mecanismos de adaptação e personalização que permitem adaptar interfaces gráficas.

  No caso de uso *Emprestar Livro* de um sistema de biblioteca, especificado no Exemplo 4.2, os fluxos de contexto estão destacados entre as linhas 19 e 21: logo após a escolha do livro pelo aluno (passo 4.2), o *software* atua como um sistema de recomendação e exibe uma relação de outros livros que possam interessar o aluno.

- **Identificar Entidades e Elementos Contextuais:** o usuário seleciona, a partir do

Lista de Códigos Fonte e Exemplos 4.2 Exemplo de Variações de Contexto

```
1 Descrição
  Esse caso de uso descreve o empréstimo de um livro por um Aluno.
2 Atores
  2.1 Aluno
  2.2 Biblioteca
3 Pré-Condições
  O usuário possui cadastro na Biblioteca.
4 Fluxo Básico de Eventos
  1. O aluno se autentica
  2. O aluno informa o código do livro a ser emprestado
  3. O sistema exibe um comprovante com o código do livro e a data de
     devolução
5 Fluxos Alternativos
  5.1 Usuário Inválido
    No passo 1 se o aluno não realizar autenticação, então
      1. O caso de uso Cadastrar Aluno é acionado
  5.2 Aluno possui multa de mora
    No passo 2 se o aluno tem multa de mora, então
      1. O caso de uso Calcular Multa é acionado
6 Fluxos de Contexto
  4.2 Indicação de outros Livros
    O sistema sugere outras obras que contribuam com o estudo do aluno
      baseando-se no seu perfil e em meta-informações do Livro
7 Pós-Condições
  Sucesso
    O aluno recebe o livro emprestado
```

modelo conceitual de contexto, as Entidades e os Elementos Contextuais e define o tipo de cada Elemento Contextual (Quem, O quê, Onde, Quando ou Por que).

- **Criar Grafos Contextuais:** o projetista de contexto cria um grafo contextual por meio de representações gráficas dos nós contextuais, nós de recombinação e ações. Em cada nó contextual é possível escolher elementos contextuais definidos no modelo conceitual de contexto e criar ramificações envolvendo expressões *booleanas*. Para cada ação é possível associar código executável que será avaliado na execução do grafo.

- **Verificar Relevância dos Elementos Contextuais:** essa funcionalidade permite que o usuário identifique a relevância de uma associação entre um *Foco* e um *Elemento Contextual* com os valores *Baixo, Médio ou Alto*. Uma associação de relevância indica a importância de um EC para um foco. Por exemplo, em um cenário de aluguel de carros, um estudante e um executivo ponderam o preço de locação de formas diferentes. Após identificar as associações de relevância, o documento

Avaliação de Relevância é gerado. Ele apresenta uma lista dos Elementos Contextuais e sua relevância para cada Foco. O exemplo apresentado abaixo indica a relevância dos ECs *idade* e *clima* para o foco composto pelo agente *Pessoa* e pela tarefa *Pesquisar Rota*:

1. Foco: («Pessoa», «Pesquisar Rota»)

   Pessoa.idade Alto

   Ambiente.clima Médio

E, para o ator *Sistema* foram identificadas os seguintes requisitos funcionais:

- **Gerar Código do Modelo:** é responsável por gerar código em linguagem de programação a partir dos modelos definidos pelo projetista de contexto.
- **Executar Grafo Contextual:** execução de um grafo contextual permitindo analisar os valores dos elementos contextuais em cada nó contextual na medida que as ações estão sendo executadas.
- **Notificar Inconsistências no Modelo:** reporta ao usuário os erros identificados na construção do Diagrama de Casos de Uso Enriquecido, modelo Conceitual de Contexto e nos Grafos Contextuais por meio da verificação das restrições OCL definidas no metamodelo de contexto.

4.3.2 Requisitos não-funcionais

Os requisitos não-funcionais são restrições ou requisitos de qualidade que restringem a produção de um *software*, como confiabilidade, segurança e manutenibilidade (IEEE Computer Society, 2004). A ferramenta CEManTIKA CASE apresenta os seguintes requisitos não-funcionais:

- Gratuita: deve ser produzida por meio de tecnologias de *software* livre e deve ser disponibilizada de forma gratuita;
- Extensibilidade: deve ser extensível, permitindo que outros desenvolvedores evoluam e adaptem a ferramenta para outros ambientes;
- Manutenibilidade: deve ser codificada em uma linguagem de programação com ampla difusão na comunidade;

- Portabilidade: deve permitir a sua execução nos principais Sistemas Operacionais para computadores pessoais.

A próxima seção identifica os componentes principais do CEManTIKA *Modeling* tendo em vista os requisitos funcionais especificados nessa seção.

4.3.3 Visão Arquitetural

Considerando os conceitos, requisitos funcionais e trabalhos vistos nos capítulos anteriores, foram identificados os seguintes componentes necessários para a implementação:

1. **Representação de Modelos:** é responsável por definir como os modelos UML, os grafos contextuais e os documentos em texto são representados preocupando-se com questões de compatibilidade e reusabilidade. Essa característica permite a importação e exportação dos artefatos por diferentes ferramentas.

2. **Editor Gráfico de Modelos:** o componente de edição permite a criação dos artefatos por meio de notações gráficas. A definição inicial do *Modelo de Comportamento de Contexto*, usando uma extensão do diagrama de atividades da UML foi mal compreendida pelos engenheiros de *software* que tendiam a usar o artefato com a semântica original de diagrama de atividades (VIEIRA, 2008). O conhecimento prévio do diagrama de atividades e da sua semântica gerou conflitos com a semântica do grafo contextual. Para solucionar esse problema, um editor de grafos contextuais com ícones e elementos gráficos apropriados é proposto. Um editor UML com capacidade de adicionar estereótipos e definir o valor das propriedades especificadas nos metamodelos também é proposto.

3. **Verificador de Modelos:** para assegurar que os modelos gerados sigam a semântica definida no metamodelo de contexto (Seção 4.1.2), são usadas restrições em OCL. O Exemplo 4.3 mostra uma regra simples aplicada a um Elemento Contextual que identifica se ele está associado a uma Entidade Contextual, na qual `base_Property` é uma referência para uma propriedade com estereótipo `cemantika_class::ContextualElement` e `cemantika_class` é o nome do pacote que define o perfil UML. Se o EC for uma propriedade (Linha 1), então deve estar associado a uma Entidade Contextual. A implicação lógica

(`implies`) só é falsa quando os dois operadores forem também falsos. As demais restrições OCL estão no Apêndice A. O módulo também é responsável por notificar os erros dos modelos ao usuário.

Lista de Códigos Fonte e Exemplos 4.3 Regra OCL para um Elemento Contextual

```
self.base_Property <> null
implies
self.base_Property.owner.hasAppliedStereotype('cemantika_class::
    ContextualEntity')
```

4. **Processador de Grafos Contextuais:** os sistemas sensíveis ao contexto são naturalmente distribuídos e possuem fontes de contexto heterogêneas, diversos meios de processamento, inferência e disseminação de informações contextuais. Essa natureza torna sua simulação e execução complexa e custosa, dificultando a detecção de erros e *bugs*. Para minimizar esses problemas é proposta a simulação das variações de comportamento representadas pelos grafos contextuais.

   O ambiente de simulação permite a execução de um grafo contextual passo a passo. O protótipo de baixa fidelidade apresentado na Figura 4.9 demonstra um cenário de uso, no qual o nó contextual em vermelho e a ação em azul representam os elementos avaliados: em cada nó contextual é apresentado um campo de entrada de dados que simula o valor de um elemento contextual, em seguida as ações são executadas até o nó contextual seguinte. Uma pilha com os passos executados é construída para permitir a identificação do conjunto de regras e ações avaliadas.

5. **Transformação de Modelos em Código:** a semântica definida num metamodelo UML auxilia o entendimento dos conceitos definidos nele. Outra vantagem da UML é a possibilidade de geração de artefatos por meio dos modelos já construídos. O CEManTIKA *Modeling* propõe usar transformações de modelos em texto, também chamadas de transformações Model to Text (M2T), para transformar o modelo Conceitual de Contexto em código e gerar relatórios.

6. **Sistema de Ajuda:** dois mecanismos são usados para prover sistemas de suporte e consulta. O primeiro permite a pesquisa de tópicos relevantes associados a contexto e ao CEManTIKA. O segundo, cria assistentes interativos para facilitar a execução de funcionalidades da ferramenta.

7. **Outros Serviços:** representam as funcionalidades comuns de persistência e organização dos modelos em projetos. Esse componente oferece algumas funcionalida-

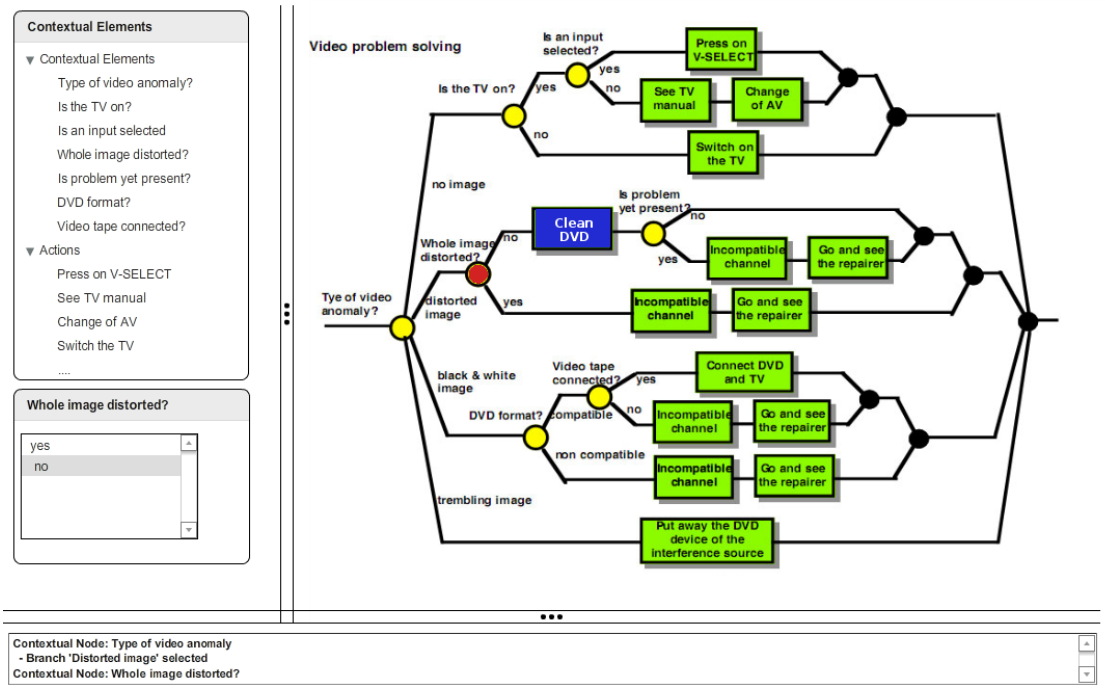

Figura 4.9 Simulação de Grafo Contextual - Grafo Contextual adaptado de http://www.cxg.fr/

des básicas, como mecanismos para importação, exportação e busca de arquivos, criação de projetos sensíveis ao contexto e navegação entre os elementos de um projeto.

Os componentes apresentados dão uma visão de alto nível das funcionalidades da ferramenta e compõem os elementos comuns para a implementação do CEManTIKA Modeling. A interação entre os componentes é visualizada na Figura 4.10: as transformações M2T e as funções comuns de gravação, leitura, importação e exportação são usadas como serviços pelos componentes que estão em camadas superiores. O Editor Gráfico permite a modificação dos modelos de contexto com o auxílio das notificações de inconsistências apontadas pelo Verificador. O sistema de ajuda é usado em qualquer camada, para tirar dúvidas básicas do uso da ferramenta e outras mais avançadas sobre a construção do modelo de contexto.

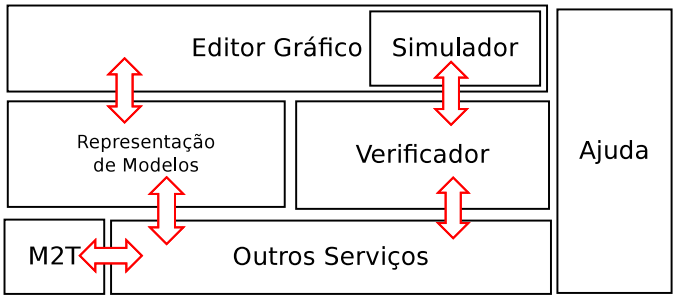

Figura 4.10 Visão arquitetural do CEManTIKA Modeling.

4.4 Considerações Finais

Construir um sistema sensível ao contexto envolve, dentre outras coisas, o entendimento de novos conceitos relativos ao gerenciamento do contexto e a criação de novas representações do mundo real. A ferramenta CEManTIKA CASE proposta considera esses problemas e é dividida em dois módulos: CEManTIKA *Process* que apoia o entendimento do Processo de Projeto de SSCs e o CEManTIKA *Modeling* que apoia o projeto de uma aplicação sensível ao contexto.

O CEManTIKA *Process* é uma biblioteca do SPEM composta de *method content*s, isto é, papéis, atividades, tarefas, guias e artefatos e um processo. Essa abordagem habilita a extensão do processo para outras metodologias e processos de *software*.

No próximo capítulo serão destacados aspectos técnicos de implementação do CEManTIKA *Modeling* e CEManTIKA *Process* e será discutido um estudo experimental realizado para verificar a aplicabilidade da ferramenta no projeto de SSC.

Capítulo 5

Implementação e Estudos Experimentais

Como apresentado no capítulo 4, a ferramenta CEManTIKA CASE [1] é formada por dois módulos: um apoia o entendimento do Processo de Projeto de SSC e o outro permite a criação dos modelos estruturais e comportamentais por meio de um *plugin* do Eclipse.

Esse capítulo destaca os aspectos técnicos de implementação e mostra os artefatos produzidos pela ferramenta. A Seção 5.1 apresenta a implementação do CEManTIKA *Process*. Em seguida, a Seção 5.2 mostra detalhes e exemplos de uso da ferramenta de modelagem. A Seção 5.3 descreve o experimento preliminar realizado para verificar a aplicabilidade da ferramenta. Por fim, a Seção 5.4 finaliza o capítulo com algumas considerações finais.

5.1 CEManTIKA Process

O CEManTIKA *Process* é uma biblioteca de elementos do SPEM construída na ferramenta Eclipse Process Framework (EPF). O EPF é uma plataforma de gerenciamento de processos e um *framework* conceitual que apoia engenheiros de processo na autoria, configuração e instalação de processos de desenvolvimento (HAUMER, 2007). O EPF organiza elementos comuns em bibliotecas que podem ser usadas como base para construir outros processos.

[1] Instruções de instalação e uso da ferramenta estão disponíveis no site http://www.nti.ufpb.br/∼raphael/cemantika/

A biblioteca do processo de projeto de SSCs do CEManTIKA pode ser combinada com outras metodologias e processos de *software* para adaptá-la às necessidades dos *stakeholders*. A Figura 5.1 apresenta os componentes desenvolvidos separados em dois tipos: *method contents* e *processos*.

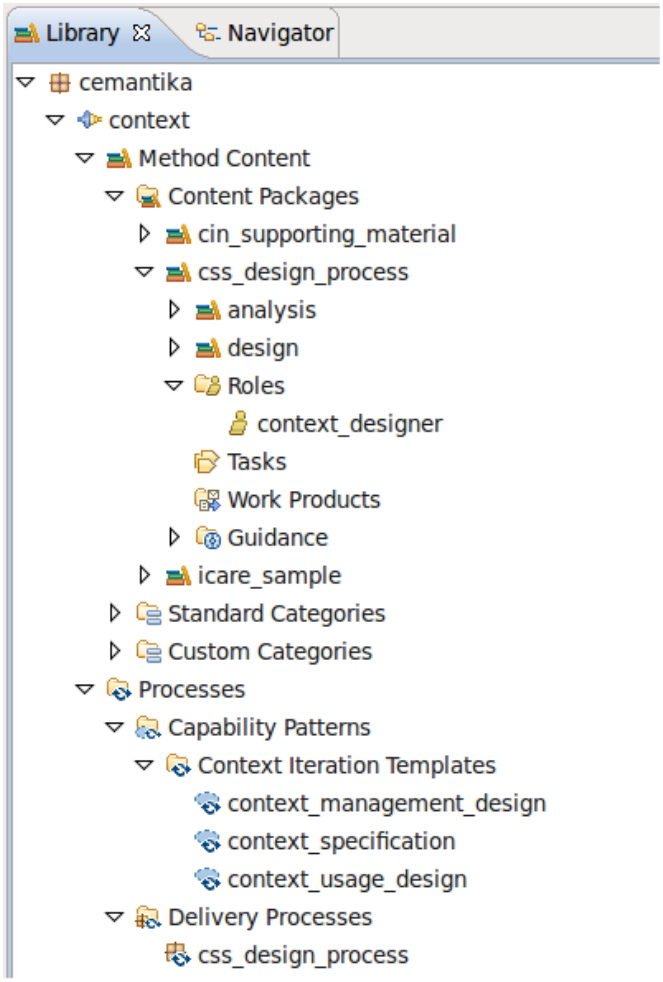

Figura 5.1 Biblioteca do CEManTIKA Process

Os *method contents* identificam as técnicas e meios de atingir um objetivo no desenvolvimento de um *software*. Eles podem ser:

- Papéis (*Roles*): definem um conjunto de habilidades, competências e responsabilidades. Os papéis executam tarefas e são responsáveis pelos produtos de trabalho;
- Produtos de Trabalho (*Work Products*): representam as coisas tangíveis usadas, modificadas ou produzidas por uma tarefa.

- Tarefas (*Tasks*): são executas por um papel e definem uma unidade de trabalho. Provêem uma descrição detalhada dos passos que devem ser executados para atingir um objetivo. As tarefas não definem em que momento são executadas em um processo;

- Guias (*Guidance*): formados por exemplos, práticas, *templates*, estimativas, conceitos, *checklists*, entre outros, que podem ser associados a tarefas, papéis e produtos de trabalho.

Para organizar melhor os elementos no EPF, foram criados dois pacotes principais: *css_design_process* e *icare_sample*. O primeiro descreve as atividades relacionadas às fases de especificação e projeto de SSCs definidas no CEManTIKA e o segundo apresenta um exemplo de projeto de uma aplicação sensível ao contexto. O pacote *css_design_process* é subdividido em outros dois: *analysis* e *design*. O primeiro é composto pelas cinco tarefas, artefatos e guias relacionados à atividade Especificação de Contexto. O pacote *design* representa as atividades para o projeto de SSC: Projeto de Gerenciamento de Contexto e Projeto de Uso de Contexto que não foram consideradas no escopo desse trabalho.

Usar bibliotecas criadas no SPEM na autoria de processos permite o reuso de artefatos já desenvolvidos. Por exemplo, o modelo de Casos de Uso foi configurado como um dos artefatos de entrada para a tarefa Identificar Foco, não tornando necessário a especificação do modelo. Ainda, o reuso de componentes facilita a construção de processos de acordo com as necessidades de cada ambiente.

Outro mecanismo definido no SPEM que contribui para a customização de processos é a Variabilidade de Elementos (*Method Variability*). Ela permite configurar um elemento por meio de uma técnica similar à herança em programação orientada a objetos. Por exemplo, é possível usar mecanismos de extensão e adicionar responsabilidades a um papel já existente ou adicionar novos guias a uma tarefa definida em outra biblioteca. O uso dessa técnica é identificado, por exemplo, na especificação do Modelo Conceitual de Contexto como uma extensão do Modelo Conceitual da aplicação definido no RUP.

Um processo organiza os *method contents* em atividades, fases, iterações e *milestones* e define o ciclo de vida de desenvolvimento de um projeto. O CEManTIKA *Process* organiza o processo de projeto de SSCs por meio de uma estrutura analítica de projeto. Essa abordagem permite a adaptação do processo usando os elementos necessários para cada ambiente.

Depois de definida, a biblioteca do CEManTIKA é processada pelo EPF e transformada em um site Web que permite a navegação entre os elementos do processo de SSC [2]. O exemplo de navegação, apresentado na Figura 5.2, mostra a estrutura de navegação da ferramenta. O usuário pode selecionar *links* (e.g. papéis, tarefas, guias) para entender o *framework* conceitual CEManTIKA de acordo com seu nível de *expertise*.

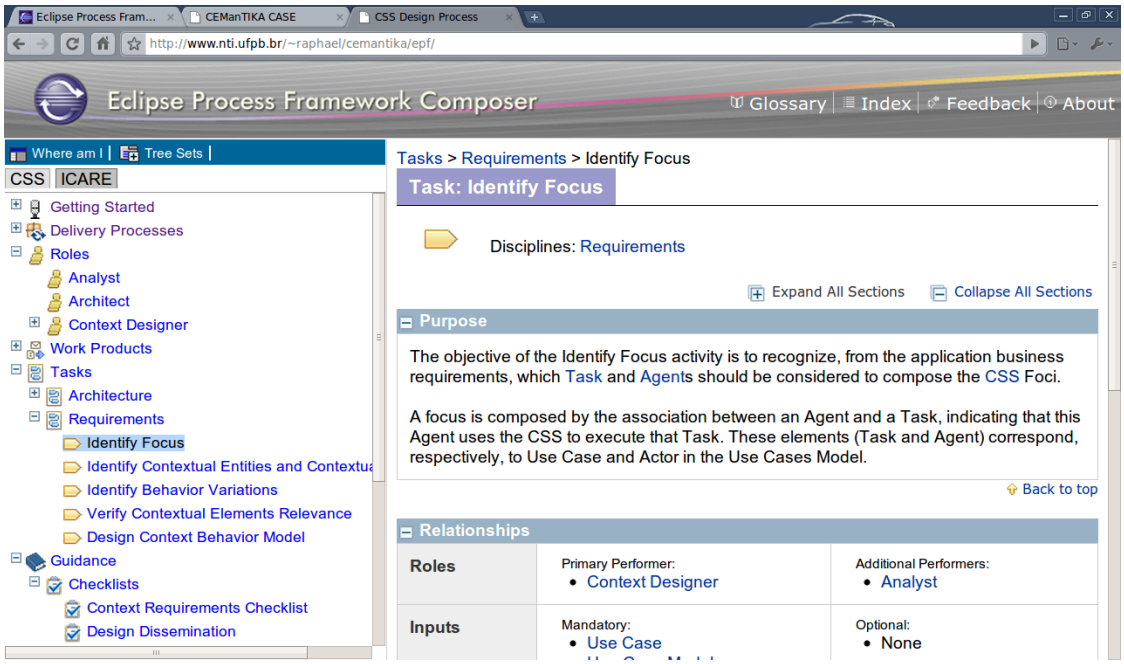

Figura 5.2 Navegação entre os elementos do processo

O CEManTIKA *Process* apoia o entendimento do CEManTIKA permitindo a navegação entre suas tarefas e conceitos de acordo com o nível de conhecimento de cada engenheiro de *software*.

5.2 CEManTIKA Modeling

O CEManTIKA *Modeling* é um plugin que pode ser instalado na IDE Eclipse. A ferramenta usa os componentes UML2, UML2Tools e OCL do Eclipse Modeling para modelagem UML e incorpora as funcionalidades de processamento e *workflow* definidas no módulo *Flow* do JBoss Drools (JBoss Community, 2010) para a implementação de grafos contextuais. O JBoss Drools é um sistema de gerenciamento de regras de ne-

[2]Uma versão está disponível em http://www.nti.ufpb.br/~raphael/cemantika/epf/

gócio capaz de processar regras de produção por meio de uma máquina de inferência. Sua adoção entrega uma solução para o processamento dos comportamentos contextuais definidos nos grafos contextuais.

Para apresentar a ferramenta serão explicadas as implementações dos requisitos funcionais detalhados no capítulo anterior. O primeiro deles define a criação de um projeto sensível ao contexto cujo objetivo é configurar o ambiente inicial para o desenvolvimento de um SSC.

Para entender os componentes da ferramenta é importante destacar alguns conceitos do ambiente de desenvolvimento (*Workbench*) do Eclipse (SILVA, 2009):

- Perspectivas (*Perspectives*): definem o *layout* inicial das visões. Cada perspectiva provê um conjunto de funcionalidades para solucionar um tipo específico de tarefa, como Java, plugin e assim por diante.
- Visões (*Views*): as visões suportam a exibição de Editores, permitem diferentes representações para um mesmo recurso e a navegação entre eles;
- Editores (*Editors*): são usados para modificar um recurso, como um arquivo Java, uma imagem, um site da Web, entre outros; e
- Projetos: são containeres usados para agrupar pastas e arquivos relacionados.

O ambiente inicial do CEManTIKA Modeling pode ser dividido em dois aspectos: interface gráfica e configuração de dependências. A interface gráfica inicial é apresentada na Figura 5.3. As Visões Explorador de Projetos (rótulo 1), Propriedades e Problemas (ambas no rótulo 4), e o Editor do CEManTIKA (rótulo 2) representam a perspectiva do CEManTIKA que inclui os elementos gráficos necessários ao uso da ferramenta e inibe as Visões não usadas pela ferramenta. O rótulo 3 destaca um botão que permite visualizar a perspectiva CEManTIKA.

O outro aspecto destaca a criação de um projeto sensível ao contexto e a adição de dependências para compilar o projeto criado. A configuração de um projeto SSC é realizada adicionando uma *nature* e a biblioteca do CEManTIKA [3] a um projeto do Eclipse. Uma *nature* associa um projeto a *builders*, que são responsáveis por executar automaticamente algum procedimento sempre que um recurso (e.g. arquivo) de um projeto é

[3]Disponível no arquivo cemantika.jar

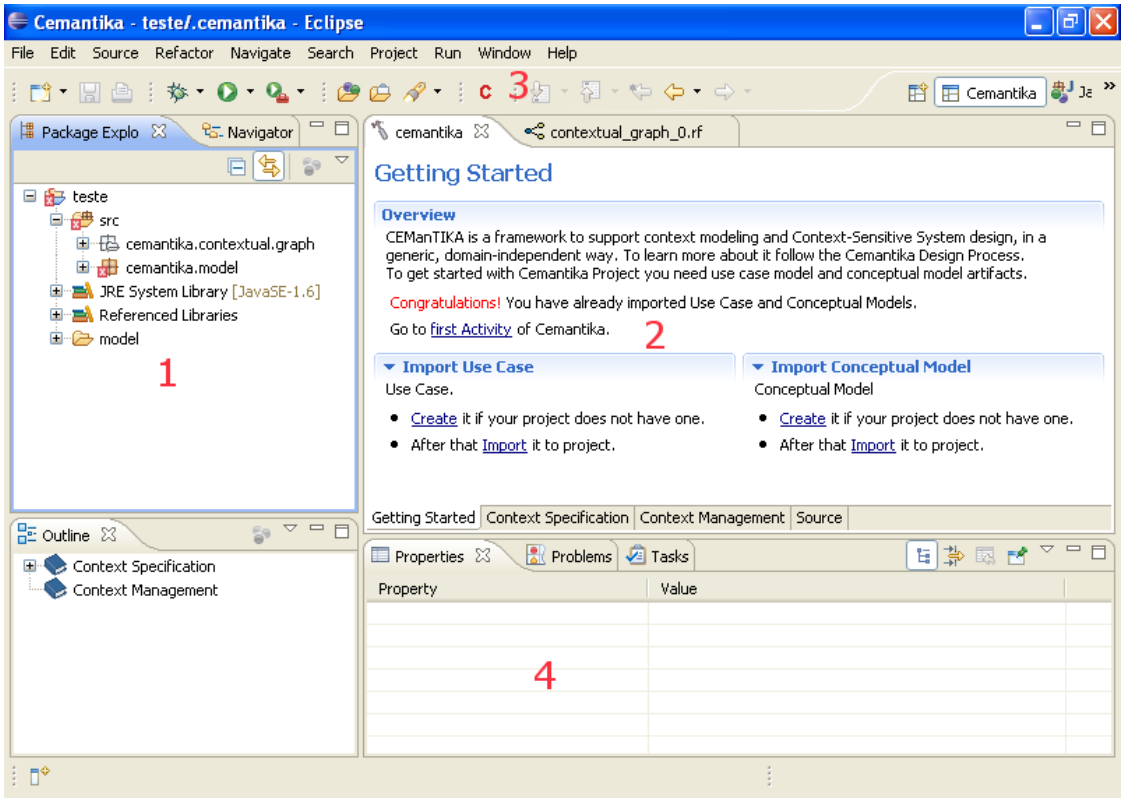

Figura 5.3 Apresentação inicial do CEManTIKA Modeling

alterado. Por exemplo, toda vez que um arquivo Java é salvo um *builder* [4] é acionado para compilar o arquivo (CLAYBERG, 2008).

A ferramenta do CEManTIKA especifica um *builder* para verificar a consistência dos modelos conceituais por meio das restrições OCL do metamodelo definidas no Apêndice A. Se um modelo possuir alguma inconsistência, a Visão Problemas indicará qual o problema apresentado. No exemplo da Figura 5.4 o modelo de Casos de Uso Enriquecido é sinalizado como incorreto, pois a *Task Finalizar um Missão* não está associada a um *Agent*, pois o ator Estudante não possui o estereótipo *Agent*. Essa implementação também satisfaz o requisito *Notificar Inconsistências no Modelo* do ator *Sistema*.

O segundo requisito funcional (*Importar Diagramas*) permite a importação dos modelos de casos de uso e conceitual para a ferramenta por meio de assistentes, como o exemplo apresentado na Figura 5.5 que permite a importação do modelo de casos de

[4] O builder responsável pela compilação de arquivo Java chama-se Eclipse incremental Java compiler

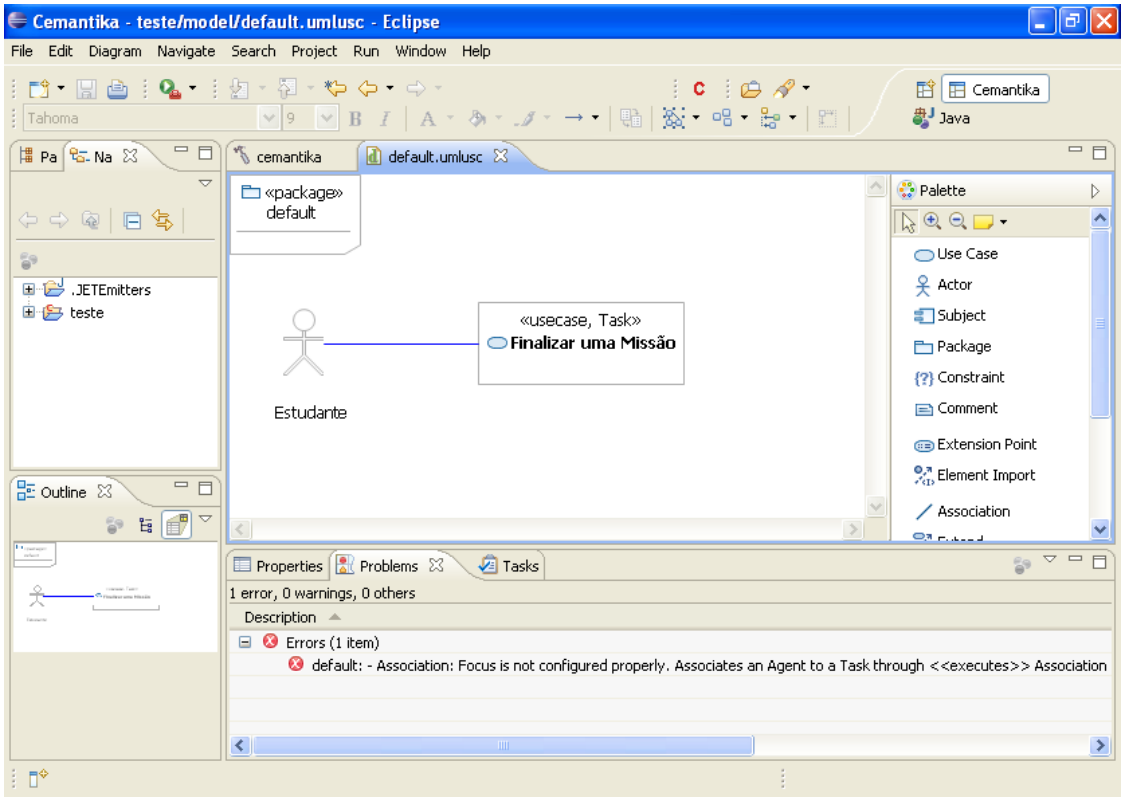

Figura 5.4 Percepção de erro após verificação de regra OCL

uso [5]. O Eclipse *Modeling* adota o padrão XMI [6] para representar os diagramas UML. Sendo assim, diagramas construídos em outras ferramentas podem ser importados.

Os dois próximos requisitos definem uma funcionalidade comum: a criação de modelos UML. Os requisitos Criar Modelo de Casos de Uso Enriquecido e Criar Modelo Conceitual de Contexto definem a criação de artefatos de mesmo nome e são executados pela adição do perfil UML do CEManTIKA, por meio de uma API, aos modelos de casos de uso e conceitual importados anteriormente. Para aplicar os perfis UML considerando apenas as extensões [7] UML de cada modelo, os elementos relacionados ao primeiro modelo estão no pacote *cemantika* e os outros no pacote *cemantika_class*.

Os requisitos Identificar Foco e Identificar Entidades e Elementos Contextuais são suportados pela ferramenta por meio da adição de estereótipos a elementos UML. A

[5] Dois arquivos devem ser importados, pois o Eclipse Modeling separa o modelo visual do modelo de domínio.

[6] XMI é um padrão definido pela OMG suportado por ferramentas pagas e gratuitas. Uma relação de ferramentas compatíveis com o Eclipse Modeling está disponível em: http://wiki.eclipse.org/MDT-UML2-Tool-Compatibility

[7] Estereótipos e propriedades

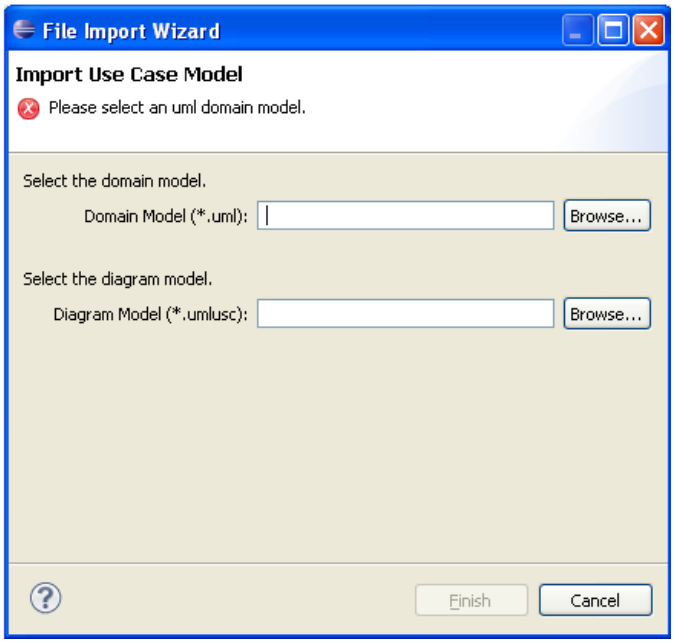

Figura 5.5 Assistente para importação do Diagrama de Casos de Uso

ferramenta exibe uma lista dos possíveis estereótipos de acordo com o tipo do elemento selecionado. Por exemplo, para um elemento do tipo Ator apenas o estereótipo *Agent* é apresentado, como mostrado na Figura 5.6. O requisito Identificar Foco produz o Diagrama de Casos de Uso Enriquecido com a identificação dos Focos possíveis para o sistema. Um foco é uma associação entre um *Agent* e uma *Task*. A seleção de um foco é realizada pela adição dos estereótipos *executes*, *Agent* e *Task* a uma associação, um Ator e um Caso de Uso, respectivamente. O requisito Identificar Entidades e Elementos Contextuais cria o modelo conceitual de contexto, artefato produzido pela identificação das entidades e elementos contextuais, no qual o usuário pode aplicar os estereótipos *ContextualEntity* e *ContextualElement* a elementos dos tipos classe e propriedade, respectivamente, do metamodelo da UML, no qual uma propriedade pode ser um atributo de uma classe ou uma associação entre duas classes. Inconsistências nos modelos são notificadas ao usuário, como a apresentada na Figura 5.4.

O próximo requisito permite a identificação das variações de comportamento para um foco. O objetivo dessa tarefa é gerar o documento de Requisitos de Contexto. Um editor simples de texto plano permite que o projetista de contexto insira as variações de comportamento para cada requisito do sistema sensível ao contexto identificado como foco.

Os outros dois requisitos implementados, Criar Grafos Contextuais (executado pelo

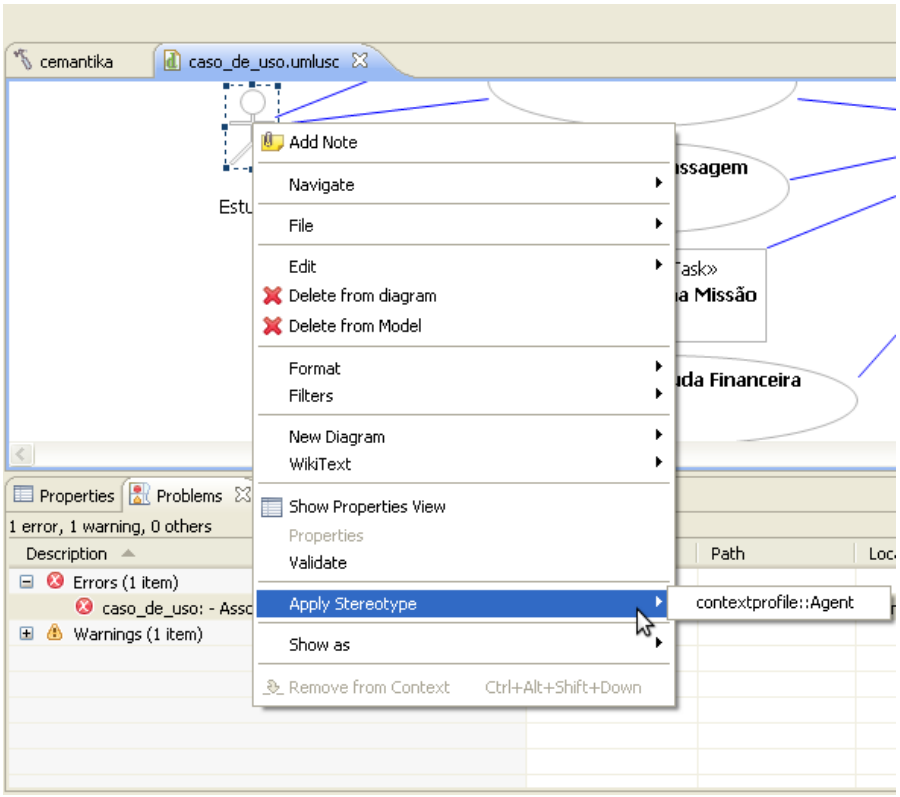

Figura 5.6 Configuração de um estereótipo na ferramenta CEManTIKA CASE

projetista de contexto) e Gerar Código do Modelo (executado pelo ator Sistema), são usados para projetar os modelos comportamentais de contexto por meio dos grafos contextuais. Para inserir expressões *booleanas* envolvendo os elementos contextuais nos nós contextuais e indicar quais ações serão realizadas nos grafos contextuais é necessário gerar uma representação em código de linguagem de programação do modelo conceitual de contexto e integrá-lo ao modelo comportamental. Essa tarefa é necessária para integrar o modelo conceitual de contexto com o JBoss Drools.

A opção escolhida foi gerar código em Java, pois a linguagem de programação integra-se à ferramenta JBoss Drools usada para construir os grafos contextuais e possui construções próprias que adicionam metadados a um elemento.

O Eclipse Modeling oferece duas abordagens para gerar artefatos a partir de um modelo: Model to Text (M2T) e Model to Model (M2M). A primeira permite a geração de texto a partir de modelos usando *templates*. A segunda gera modelos intermediários. O propósito da geração de código da ferramenta é permitir a integração dos modelos conceituais de contexto com o modelo comportamental, por isso a solução mais simples

e menos extensível foi escolhida: M2T.

Para a transformação M2T, a ferramenta JET foi escolhida, pois permite gerar texto a partir de instâncias de um modelo usando *templates* (GRONBACK, 2009). O Código Fonte 5.1 apresenta um trecho do *template* de geração de uma classe Java a partir de uma classe modelada em UML, representada pela variável `clazz` [8]. Os trechos entre `<%=` e `%>` possuem expressões que serão avaliadas para uma *String* durante a transformação do *template* em código fonte Java. Assim, na linha 5 o método `getAnnotation` retorna o tipo de elemento representado naquela classe UML e as linhas 8 à 10 retornam todos os atributos da classe.

Lista de Códigos Fonte e Exemplos 5.1 Exemplo de template do JET

```
1  /**
2   * This class implements a ContextualEntity with annotations
3   * A Contextual Entity is a class used to build context sensitive
        behaviors.
4   */
5  <%= clazz.getAnnotation() %>
6  public class <%=clazz.getClassName()%><% if (clazz.getSuperClass() !=
       null) { %> extends <%= clazz.getSuperClass() %> <% } %> {
7
8  <% for (Attribute attribute : clazz.getAttributes()) { %>
9          <%= attribute.toString() %>;
10 <% } %>
11 ...
```

As classes geradas possuem anotações [9] que adicionam metadados a um objeto (e.g. classe, método, variável) e descrevem seu papel no CEManTIKA. Por exemplo, uma Entidade Contextual `Pessoa` com um Elemento Contextual `horaDeAcordar` é transformada no código em Java apresentado no Código Fonte 5.2.

Lista de Códigos Fonte e Exemplos 5.2 Exemplo de código Java gerado para uma Entidade Contextual

```
1 @ContextualEntity
2 public class Pessoa {
3   @ContextualElement(type=ContextType.WHEN)
4   private int horaDeAcordar;
5   ...
6 }
```

[8] A variável *class* referencia uma metaclasse que representa um elemento UML criado usando o Eclipse

[9] Anotação é uma construção da linguagem Java

Com um modelo executável gerado (código Java) o projetista de contexto pode criar os grafos contextuais relacionados aos focos. Para construir o modelo comportamental foi adotado o projeto JBoss Drools Flow. O Drools Flow é um componente do JBoss Drools que permite a criação de *workflows* ou processos de negócio descrevendo a ordem na qual uma série de passos devem ser executados usando um diagrama de fluxos (JBoss Community, 2010). Essa abordagem entrega dois benefícios: um editor gráfico para processos e um ambiente de execução e depuração de grafos contextuais, sendo que o último satisfaz a implementação do requisito Executar Grafo Contextual especificado no capítulo anterior.

Para adicionar os conceitos dos grafos contextuais ao editor foram identificados que os componentes *Split*, *Join* e *Action*, apresentados na Figura 5.7, possuem a mesma semântica dos elementos Nó Contextual, Nó de Recombinação e Ação dos grafos contextuais. Portanto, um *plugin* com as representações gráficas do grafo contextual foi incorporado à ferramenta como mostra a Figura 5.8.

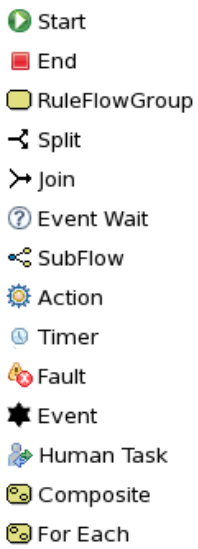

Figura 5.7 Componentes do JBoss Drools Flow

Um exemplo do ambiente de edição e de um grafo contextual construído por meio do *plugin* é apresentado na Figura 5.8, no qual os rótulos 1, 2 e 3 (destacados no menu à esquerda do editor) representam os ícones para o nó contextual, nó de recombinação e ação, respectivamente.

O requisito *Verificar Relevância dos Elementos Contextuais* será implementado em trabalhos futuros, conforme apresentado no Capítulo 6. A próxima seção verifica a

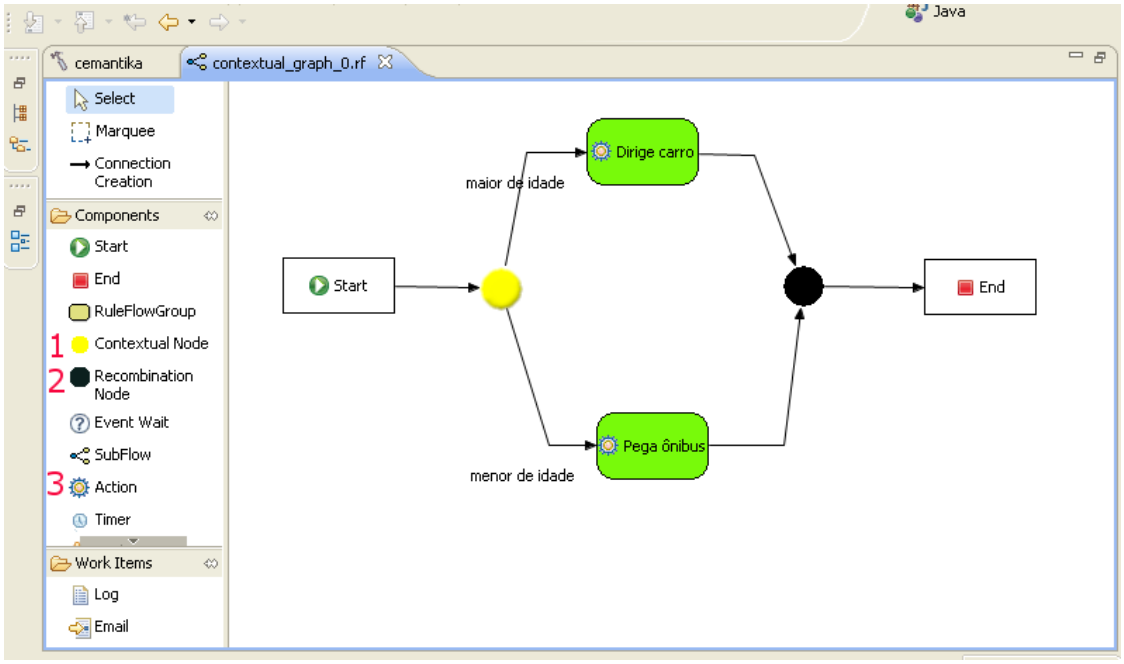

Figura 5.8 Criação de um grafo contextual na ferramenta no CEManTIKA Modeling

aplicabilidade da ferramenta no projeto de SSC por meio de um experimento preliminar.

5.3 Experimentação

Para verificar o uso da ferramenta CEManTIKA CASE no projeto de sistemas sensíveis ao contexto foi realizado um experimento preliminar com os seguintes objetivos:

- Verificar a aplicabilidade da ferramenta CEManTIKA CASE no projeto de SSCs por meio do *framework* CEManTIKA. O termo aplicabilidade corresponde ao uso da ferramenta para a modelagem de sistemas sensíveis ao contexto por meio do conjunto de atividades e tarefas definidos no *framework* conceitual CEManTIKA e a criação dos artefatos de saída especificados nessas tarefas;
- Obter sugestões de melhoria da ferramenta.

O estudo foi conduzido com nove (9) participantes, funcionários do Núcleo de Tecnologia de Informação da UFPB, de diferentes níveis de formação: um (1) mestre, três (3) mestrandos, um (1) especialista e quatro (4) graduados. Eles deveriam projetar o SSC

de auxílio a missões, apresentado no Apêndice B, que apoia seus usuários na participação de um evento acadêmico. A maioria dos participantes não possuíam conhecimentos aprofundados em sistemas sensíveis ao contexto como apresentado na Figura 5.9. Apenas um havia construído algum SSC e outro possuía conhecimentos de modelagem de contexto.

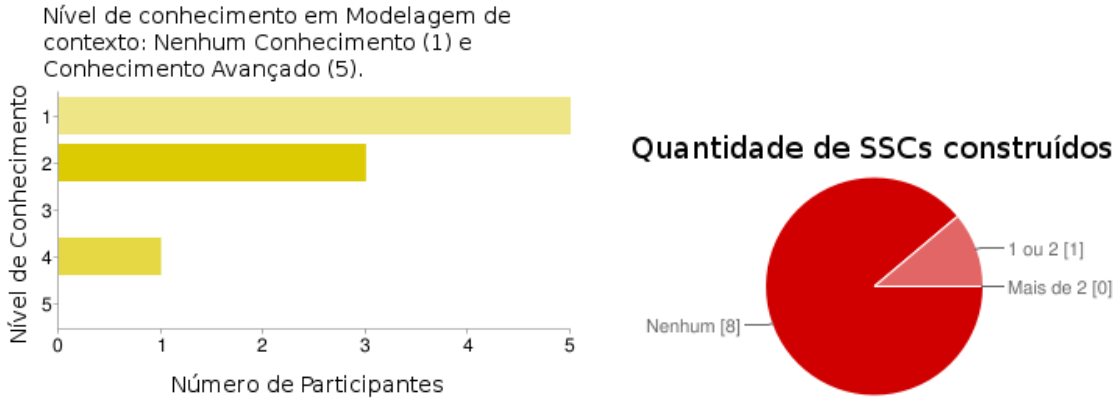

Figura 5.9 Experiência dos participantes no desenvolvimento de SSCs

Na fase de preparação do experimento, os indivíduos foram separados em dois grupos: um formado por três (3) membros projetou o SSC usando outras ferramentas, como papel e processadores de texto; o outro grupo, formado por seis (6) integrantes, teve o apoio dos dois módulos do CEManTIKA CASE, pois possuíam menor *expertise* relacionadas a contexto e modelagem UML. Com essa abordagem conseguimos avaliar como a ferramenta apoia o projeto de um SSC pelo grupo com menor conhecimento em contexto. Os grupos foram separados de acordo com o perfil identificado nas respostas às questões dois (2) e três (3) do questionário de avaliação.

O estudo foi executado de acordo com os seguintes passos:

1. Explicação da proposta do *framework* CEManTIKA aos participantes;
2. Apresentação da ferramenta CEManTIKA CASE;
3. Discussão do problema a ser executado;
4. Execução do problema;
5. Resposta ao questionário apresentado no Apêndice C.

Para averiguar o resultado do experimento são usadas duas abordagens: a primeira compara as respostas do questionário fornecidas pelos dois grupos; a outra verifica qualitativamente os artefatos produzidos pelos participantes. Um integrante do grupo que não usava a ferramenta não conseguiu concluir o experimento.

A primeira análise mostrou que as tarefas Identificar Foco, Identificar Entidades Contextuais e ECs e Projetar o Modelo de Comportamento de Contexto foram mais bem compreendidas pelos que usaram a ferramenta. Para a segunda tarefa, a Figura 5.10 mostra que os participantes que usaram a ferramenta ficaram entre os níveis de entendimento três (regular) e cinco (melhor).

No entanto, o tempo de execução do experimento foi equivalente entre os dois grupos. No grupo que usou a ferramenta, cinco (5) pessoas demoraram mais de uma hora para concluir o experimento, no outro, uma (1) pessoa.

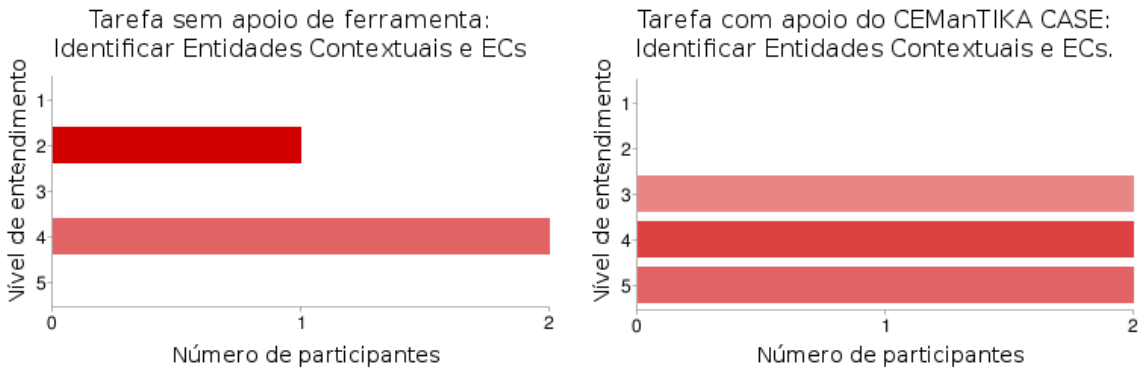

Figura 5.10 Comparação da tarefa Identificar Entidades Contextuais e Elementos Contextuais com e sem ferramenta

Todos os integrantes do grupo que usou o CEManTIKA CASE afirmaram que a adoção da IDE Eclipse como plataforma auxiliou o uso da ferramenta, como apresentado na Figura 5.11.

A outra abordagem compara os artefatos criados entre os grupos. Os dois conseguiram criar o artefato Casos de Uso Enriquecido e o documento de Requisitos de Contexto, saída da tarefa Identificar Variações de Comportamento, corretamente. A aplicabilidade da ferramenta é encontrada à medida que as tarefas se tornam mais complexas. Os modelos conceitual e comportamental de contexto foram mais bem desenvolvidos com o apoio da ferramenta. O grupo que não usou a ferramenta identificou elementos contextuais não usados no modelo comportamental. Os grafos contextuais apresentaram outros dois problemas: quantidades diferentes de nós contextuais e nós de recombinação e au-

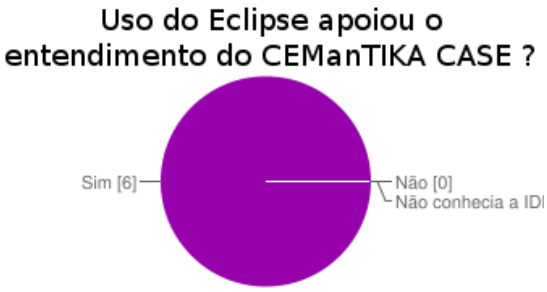

Figura 5.11 Apoio do Eclipse no uso do CEManTIKA CASE

sência de ações. As verificações dos modelos oferecidas pela ferramenta contribuíram para a criação de modelos bem formados, que correspondem a modelos semanticamente válidos em relação ao metamodelo de contexto.

Os participantes que usaram a ferramenta identificaram algumas possibilidades de melhoria que serão analisadas em trabalhos futuros:

- facilidade de manuseio dos diagramas: essa deficiência é decorrente de limitações técnicas de manipulação dos diagramas usando o Eclipse;
- ativação de uma tarefa após a conclusão da anterior: a visualização de todas as tarefas para a construção de um SSC inibem a execução passo a passo do processo.
- engenharia reversa: geração dos diagramas a partir de código.

O estudo preliminar mostrou que é possível usar uma ferramenta CASE para apoiar a construção sistematizada de um SSC partindo de artefatos de *software* já modelados na aplicação, como o documento de casos de uso e o modelo conceitual da aplicação. Todavia, a adição de outras atividades do CEManTIKA e a implementação das sugestões de melhoria devem ser verificadas para permitir o desenvolvimento de um sistema sensível ao contexto. A realização de outros experimentos com pessoas mais familiarizadas no domínio também deve ser realizada.

O experimento também mostrou que desenvolver um sistema sensível ao contexto é uma tarefa que despende tempo para assimilação dos conceitos relacionados a contexto e às possibilidades de mudanças no comportamento desses sistemas. Talvez por isso, contexto não faça parte dos principais requisitos da maioria das aplicações. Dessa forma, para que uma ferramenta seja adotada pelos engenheiros de *software* é necessário torná-la simples e intuitiva com os comportamentos contextuais mais comuns, como adaptação

de interfaces gráficas, já pré-configurados. Ainda, o uso de um processo de *software* apoia bastante o entendimento de como usar contexto em cada atividade do ciclo de desenvolvimento de um SSC.

5.4 Considerações Finais

Esse capítulo descreveu os detalhes técnicos de implementação da ferramenta CEManTIKA. A primeira seção destaca o CEManTIKA *Process* que foi construído como uma biblioteca do EPF. Essa abordagem permitiu o reuso de artefatos definidos em outros processos (e.g. casos de uso do RUP) e habilita a adequação do processo de projeto de SSC para outros ambientes. Um *site* para navegação entre os elementos da biblioteca foi construído.

O CEManTIKA *Modeling*, apresentado na Seção 5.2, permite a execução passo a passo das tarefas do CEManTIKA por meio de um *plugin* do Eclipse. A ferramenta apoia as quatro primeiras tarefas da atividade Especificação de Contexto e cria os modelos conceitual e comportamental de contexto. Mecanismos de verificação contribuem para a criação de artefatos bem formados.

Por fim, o capítulo apresenta um experimento preliminar realizado no Núcleo de Tecnologia da Informação da UFPB. O uso da ferramenta contribuiu para a compreensão do CEManTIKA e apoiou a criação de artefatos bem formados. No entanto, a ferramenta CEManTIKA CASE deve ser aprimorada para ser aplicada em ambientes reais com cenários de uso mais complexos. Para isso é necessário especificar e implementar outras tarefas relacionadas ao gerenciamento e uso de contexto, bem como tarefas de validação e verificação do SSC.

Capítulo 6

Conclusões e Trabalhos Futuros

Contexto vem sendo aplicado em sistemas computacionais para permitir a criação de aplicações mais adaptativas, proativas, fáceis de usar e que se ajustam às necessidades dos usuários. No entanto, o uso de contexto em uma aplicação implica em vários desafios a serem investigados pelos pesquisadores, tais como: Que tipos de informação considerar como contexto? Como representar essas informações? Como adquiri-las e processá-las? Como integrá-las ao sistema? Como apresentá-las ao usuário? Além disso, a ausência de modelos, métodos e ferramentas torna a criação de um sistema sensível ao contexto mais desafiadora.

Para abstrair alguns desses problemas, diversas abordagens foram sugeridas, como middlewares (KANG et al., 2007), *APIs* (BARDRAM, 2010) e *toolkits* (SALBER; DEY; ABOWD, 1999). No entanto, essas abordagens apoiam aspectos de implementação de SSC para um domínio específico, sem levar em consideração tarefas de análise e projeto.

Nesse contexto, Vieira (2008) abordou o desenvolvimento de sistemas sensíveis ao contexto por meio de um processo, arquitetura e metamodelo UML, preocupando-se com modularidade e reusabilidade e definiu o CEManTIKA, um framework conceitual que apoia o projeto de SSC de forma genérica e independente de domínio. Entretanto, uma das limitações do CEManTIKA era a ausência de uma ferramenta que facilitasse o uso e integração dos seus componentes, bem como que apoiasse a transição da fase de projeto para a fase de implementação de um SSC.

Este trabalho especifica e implementa uma ferramenta CASE para apoiar o uso do CEManTIKA e foi motivado pela ausência de ferramentas de projeto e implementação

de SSC independentes de domínio por meio de um processo com atividades, papéis e tarefas.

Os trabalhos relacionados encontrados não satisfazem a necessidade de uma ferramenta para construção e prototipação de SSC e, então, foi construída a ferramenta CEManTIKA CASE, que é composta por dois módulos: CEManTIKA *Process* e CEManTIKA *Modeling*. O primeiro é uma biblioteca de documentação construída no Eclipse Process Framework (EPF) que permite o entendimento do Processo de Projeto de SSC. O segundo apoia a construção de SSC.

Este capítulo é organizado da seguinte forma: a Seção 6.1 discute as contribuições alcançadas com esta dissertação; e Seção 6.2 comenta algumas dificuldades e restrições encontradas durante a especificação e construção do CEManTIKA CASE; e a Seção 6.3 apresenta algumas direções para a implementação de outras funcionalidades na ferramenta CEManTIKA CASE.

6.1 Contribuições

Este trabalho investigou ferramentas de apoio ao desenvolvimento de SSC que auxiliam a construção e mecanização de SSC por meio de processos e modelos independentes de domínio. Entre as contribuições desta dissertação, destacam-se:

- Análise crítica e alteração das atividades Especificação de Contexto e Projeto e Uso de Contexto pertencentes ao Processo de Projeto de Sistemas Sensíveis ao Contexto. Na definição inicial do CEManTIKA proposta por Vieira (2008), a tarefa Projetar o Modelo Comportamental de Contexto (S4) (Seção 4.1.3) pertencia à atividade de Projeto de Uso de Contexto, sendo realizada, portanto, após a definição das atividades Especificação de Contexto e Projeto e Uso de Contexto. No entanto, observou-se que a tarefa S4 pode ser executada durante a atividade de Especificação de Contexto, logo após a definição dos elementos contextuais do SSC, conforme apresentado na Seção 4.1.4.
- A especificação de uma ferramenta CASE para a implementação de SSC por meio de um Processo de Projeto de SSC definido no *framework* conceitual CEManTIKA;

- Implementação do CEManTIKA *Process* com a definição de papéis, guias, atividades e tarefas por meio de uma biblioteca de documentação desenvolvida no Eclipse Process Framework (EPF). O CEManTIKA *Process* auxilia o entendimento do Processo de Projeto de SSC por engenheiros de *software* com menor experiência;

- Implementação das tarefas referentes à atividade de Especificação de Contexto na ferramenta CEManTIKA *Modeling*, que apoia o projeto dos modelos conceituais e comportamentais de contexto seguindo uma sequência definida de tarefas. Experimentos preliminares mostraram que os artefatos de contexto desenvolvidos com o uso da ferramenta apresentaram uma quantidade menor de erros de modelagem.

6.2 Dificuldades Encontradas

Desenvolver um sistema sensível ao contexto não é trivial. Construir uma ferramenta para a especificação e construção desses sistemas é uma tarefa mais desafiadora ainda. Entre as dificuldades encontradas durante esta pesquisa, destacam-se:

- Identificação de ferramentas relacionadas: a maioria das ferramentas de modelagem de contexto foram desenvolvidas para domínios específicos de aplicações. Portanto, foi necessário realizar uma pesquisa extensa para encontrar ferramentas que estão relacionadas a este trabalho;

- Estender uma meta-ferramenta CASE: embora o uso de uma meta-ferramenta para construção de uma ferramenta CASE auxilie em tarefas comuns, o esforço de aprendizagem é grande. Entender os mecanismos de extensão e customização da IDE Eclipse despendeu um longo período;

- Especificar requisitos da ferramenta CASE: uma ferramenta CASE possui o objetivo de auxiliar e tornar mais fácil a execução de tarefas complexas pelo usuário. Entender como simplificar a modelagem de contexto e, ao mesmo tempo, permitir que usuários mais experientes definam modelos mais ricos foi uma das dificuldades encontradas;

- Realização de Experimento de Avaliação: para compreender contexto e aplicá-lo a um *software* é necessário entender vários conceitos adjacentes, como mecanismos

de percepção e adaptação e regras de produção. Por isso, durante a preparação do experimento de avaliação da ferramenta CEManTIKA CASE (Seção 5.3) foi necessário explicar várias definições relacionadas a contexto, além do *framework* conceitual CEManTIKA. Ainda, ocorreu a dificuldade em definir quais variáveis deveriam ser avaliadas durante o experimento e como avaliá-las.

6.3 Trabalhos Futuros

O objetivo inicial deste trabalho foi implementar uma ferramenta de apoio ao CEManTIKA que auxiliasse o projeto de SSC de forma sistematizada. O trabalho deixou algumas lacunas que devem ser preenchidas em trabalhos futuros:

- Implementar as demais atividades e tarefas do Processo de Projeto de SSC: a ferramenta CEManTIKA CASE implementou a atividade de Especificação de Contexto do Processo de Projeto de SSC. Para oferecer um ambiente completo para o desenvolvimento de SSC e apoiar a criação de um modelo executável (código em linguagem de programação) da aplicação, as atividades de Projeto de Gerenciamento de Contexto, Projeto de Uso de Contexto, Geração de Código e Teste devem ser implementadas em trabalhos futuros.

- Usar uma abordagem Model Driven Architecture (MDA) para a construção de código fonte: foram usadas técnicas de transformação de modelos em texto (M2T) que permitiram a geração de código Java a partir de modelos UML. Essa abordagem permitiu a integração dos modelos conceituais de contexto com os modelos comportamentais definidos no JBoss Drools. No entanto, a geração de modelos para diferentes plataformas por meio de MDA deve ser investigada.

- Melhorar interface com o usuário: Durante o experimento, os usuários evidenciaram problemas de usabilidade na ferramenta, como a configuração da multiplicidade das associações UML e o manuseio dos diagramas. Também, foi sugerido que a execução de uma tarefa do Processo de Projeto de SSC seja permitida apenas após a conclusão da tarefa antecedente.

- Executar outros experimentos com projetistas de contexto mais experientes: O experimento foi realizado com usuários sem experiência no desenvolvimento de

SSC. Para obter melhor *feedback* do CEManTIKA CASE é interessante avaliá-lo em ambientes mais complexos com engenheiros de *software* experientes em contexto e usando outros exemplos de sistemas sensíveis ao contexto.

Referências Bibliográficas

AULETE. *Dicionário Aulete*. 2010. Disponível em: <http://aulete.uol.com.br>. Acesso em: 12 ago. 2010.

AYED, D.; BERBERS, Y. Uml profile for the design of a platform-independent context-aware applications. In: *Proceedings of the 1st workshop on MOdel Driven Development for Middleware (MODDM '06)*. New York, NY, USA: ACM, 2006. (MODDM '06), p. 1–5. ISBN 1-59593-423-5.

AYED, D.; DELANOTE, D.; BERBERS, Y. MDD approach for the development of context-aware applications. In: KOKINOV, B. et al. (Ed.). *Modeling and Using Context*. [S.l.]: Springer Berlin / Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4635). p. 15–28.

BARDRAM, J. E. *The Java Context-Awareness Framework*. 2010. Disponível em: <http://www.daimi.au.dk/ bardram/jcaf/>. Acesso em: 01 ago. 2010.

BETTINI, C. et al. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, v. 6, n. 2, p. 161 – 180, 2010. ISSN 1574-1192.

BOLCHINI, C. et al. CADD: A tool for context modeling and data tailoring. In: *Mobile Data Management, 2007 International Conference on*. [S.l.: s.n.], 2007. p. 221 –223.

BRÉZILLON, P.; ARAUJO, R. M. de. Reinforcing shared context to improve collaboration. *Revue d'Intelligence Artificielle*, v. 19, n. 3, p. 537–556, 2005.

BRéZILLON, P. Context in Artificial Intelligence: I. A survey of the literature. *Computers and artificial intelligence*, v. 18, p. 321–340, 1999.

BRéZILLON, P. Task-realization models in contextual graphs. In: DEY, A. et al. (Ed.). *Modeling and Using Context*. [S.l.]: Springer Berlin / Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3554). p. 55–68.

BRéZILLON, P.; PASQUIER, L.; POMEROL, J.-C. Reasoning with contextual graphs. *European Journal of Operational Research*, v. 136, n. 2, p. 290 – 298, 2002. ISSN 0377-2217.

BRéZILLON, P.; POMEROL, J.-C. Contextual knowledge sharing and cooperation in intelligent assistant systems. *Le Travail Humain*, v. 62, p. 223–246, 1999.

CHAVES, A. P. *Um Modelo Baseado em Context-Awareness para Disseminação de Informações em um Ambiente de Desenvolvimento Distribuído de Software*. Dissertação (Mestrado) — UEM - Universidade Estadual de Maringá, 2009.

CHEVERST, K. et al. Developing a context-aware electronic tourist guide: some issues and experiences. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2000. (CHI '00), p. 17–24. ISBN 1-58113-216-6.

CLAYBERG, D. R. E. *Eclipse Plug-ins*. 3. ed. [S.l.]: Addison-Wesley Professional, 2008.

COLLABNET. *ArgoUML*. 2010. Disponível em: <http://argouml.tigris.org/>. Acesso em: 01 ago. 2010.

de Farias, C. R. G. et al. A mof metamodel for the development of context-aware mobile applications. In: *Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2007. (SAC '07), p. 947–952. ISBN 1-59593-480-4.

DEY, A. K. et al. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, v. 16, n. 2/4, p. 97–166, 2001.

DEY, A. K.; ABOWD, G. D. Towards a better understanding of context and context-awareness. In: *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*. [S.l.: s.n.], 2000.

DEY, A. K. et al. a CAPpella: programming by demonstration of context-aware applications. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2004. (CHI '04), p. 33–40. ISBN 1-58113-702-8.

Eclipse Foundation, T. *Eclipse Modeling Project*. 2010. Disponível em: <http://www.eclipse.org/modeling/>. Acesso em: 01 ago. 2010.

Eclipse Foundation, T. *The Eclipse Project*. 2010. Disponível em: <http://www.eclipse.org/>. Acesso em: 01 ago. 2010.

FUENTES, L.; GáMEZ, N.; SáNCHEZ, P. Aspect-oriented design and implementation of context-aware pervasive applications. *Innovations in Systems and Software Engineering*, Springer London, v. 5, p. 79–93, 2009. ISSN 1614-5046.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Professional, 1995.

GRONBACK, R. C. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. 1. ed. [S.l.]: Addison-Wesley Professional, 2009. Paperback. ISBN 0321534077.

HALPIN, T. *Conceptual schema and relational database design*. 2. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN 0-13-355702-2.

HAUMER, P. *Eclipse Process Framework Composer Part 1 and 2*. 2007. User Documentation. Disponível em: <http://www.eclipse.org/epf/general/getting_started.php>.

HENRICKSEN, K.; INDULSKA, J. A software engineering framework for context-aware pervasive computing. *Pervasive Computing and Communications, IEEE International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 77, 2004.

HENRICKSEN, K.; INDULSKA, J. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, v. 2, n. 1, p. 37–64, fev. 2006. ISSN 15741192.

HENRICKSEN, K.; INDULSKA, J.; MCFADDEN, T. Modelling context information with orm. In: MEERSMAN, R.; TARI, Z.; HERRERO, P. (Ed.). *On the Move to Meaningful Internet Systems 2005: OTM Workshops*. [S.l.]: Springer Berlin / Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3762). p. 626–635.

IEEE Computer Society. *Software Engineering Body of Knowledge (SWEBOK)*. EUA: Angela Burgess, 2004. Disponível em: <http://www.swebok.org/>.

JBoss Community. *JBoss Drools*. 2010. Disponível em: <http://jboss.org/drools>. Acesso em: 01 ago. 2010.

KANG, D.-o. et al. A systematic design tool of context aware system for ubiquitous healthcare service in a smart home. *Future Generation Communication and Networking*, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, p. 49–54, 2007.

KAPITSAKI, G. M. et al. Model-driven development of composite context-aware web applications. *Information and Software Technology*, v. 51, n. 8, p. 1244 – 1260, 2009. ISSN 0950-5849.

KOCH, N.; KRAUS, A. Towards a common metamodel for the development of web applications. In: LOVELLE, J. et al. (Ed.). *Web Engineering*. [S.l.]: Springer Berlin / Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2722). p. 419–422.

LIEBERMAN, H. *Your wish is my command: Programming by example*. [S.l.]: Morgan Kaufmann Publishers, 2001.

MAGICDRAW. *MagicDraw*. 2010. Disponível em: <http://www.magicdraw.com/>. Acesso em: 01 ago. 2010.

MCFADDEN, T.; HENRICKSEN, K.; INDULSKA, J. Automating context-aware application development. In: *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*. [S.l.]: The University of Southampton, The United Kingdom, 2004. p. 90 – 95.

MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. Guest editors' introduction: Model-driven development. *IEEE Software*, IEEE Computer Society, Los Alamitos, CA, USA, v. 20, p. 14–18, 2003. ISSN 0740-7459.

MORSE, D. R.; ARMSTRONG, S.; DEY, A. K. The what, who, where, when, why and how of context-awareness. In: *CHI '00 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2000. (CHI '00), p. 371–371. ISBN 1-58113-248-4.

MURPHY, K. P. *Dynamic bayesian networks: representation, inference and learning*. Tese (Doutorado) — University of California, Berkeley, 2002.

Netbeans IDE. *The Eclipse Project*. 2010. Disponível em: <http://www.netbeans.org/>. Acesso em: 01 mai. 2010.

NETO, R. F. B.; KUDO, T. N.; PIMENTEL, M. d. G. C. Pocap: A software process for context-aware computing. In: *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*. Washington, DC, USA: IEEE Computer Society, 2006. (IAT '06), p. 705–708. ISBN 0-7695-2748-5.

OMG. *Software & Systems Process Engineering Metamodel specification (SPEM) 2.0*. [S.l.], 2008. Disponível em: <http://www.omg.org/spec/SPEM/2.0/PDF>.

OMG. *Object Constraint Language 2.2*. [S.l.], 2010.

ORACLE. *Java*. 2010. Disponível em: <http://www.java.com/>. Acesso em: 01 ago. 2010.

Papyrus UML. *Papyrus UML*. 2010. Disponível em: <http://www.papyrusuml.org/>. Acesso em: 01 ago. 2010.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 5th. ed. [S.l.]: McGraw-Hill Higher Education, 2009. ISBN 0072496681.

SALBER, D.; DEY, A. K.; ABOWD, G. D. The context toolkit: aiding the development of context-enabled applications. In: *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*. New York, NY, USA: ACM, 1999. (CHI '99), p. 434–441. ISBN 0-201-48559-1.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. [S.l.: s.n.], 1994. p. 85 –90.

SCHMIDT, D. C. Model-driven engineering. *IEEE Computer*, v. 39, n. 2, p. 25–31, February 2006.

SEGALL, B. et al. Content based routing with elvin4. In: *in Proceedings of AUUG2K*. [S.l.: s.n.], 2000.

SHENG, Q.; BENATALLAH, B. Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In: *Mobile Business, 2005. ICMB 2005. International Conference on*. [S.l.: s.n.], 2005. p. 206 – 212.

SILVA, V. *Practical Eclipse Rich Client Platform*. [S.l.]: Apress, 2009.

SIMONS, C.; WIRTZ, G. Modeling context in mobile distributed systems with the uml. *Journal of Visual Languages & Computing*, v. 18, n. 4, p. 420 – 439, 2007. ISSN 1045-926X. Visual Interactions in Software Artifacts.

SOHN, T.; DEY, A. K. iCAP: an informal tool for interactive prototyping of context-aware applications. In: *CHI '03 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2003. (CHI '03), p. 974–975. ISBN 1-58113-637-4.

SOHN, T.; DEY, A. K. iCAP: Rapid prototyping of context-aware applications. In: *Proceedings of the CHI 2004 ACM Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2003.

TRUONG, K.; ABOWD, G.; BROTHERTON, J. Who, what, when, where, how: Design issues of capture & access applications. In: ABOWD, G.; BRUMITT, B.; SHAFER, S. (Ed.). *Ubicomp 2001: Ubiquitous Computing*. [S.l.]: Springer Berlin / Heidelberg, 2001, (Lecture Notes in Computer Science, v. 2201). p. 209–224.

USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, v. 11, n. 02, p. 93–136, 1996.

VIEIRA, V. *CEMantika: A Domain-Independent Framework for Designing Context-Sensitive System*. Tese (Doutorado) — UFPE, October 2008.

VIEIRA, V. et al. Tópicos em sistemas interativos e colaborativos. In: ____. [S.l.]: Cesar A. C. Teixeira; Clever Ricardo G. de Farias; Jair C. Leite; Raquel O. Prates. (Org.). São Carlos: UFSCAR, 2006. v. 1, cap. Uso e representação de contexto em sistemas computacionais, p. 127 – 166.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Jornadas de atualização em informática 2009 (jai'09). In: ____. [S.l.]: André Ponce de Leon F. de Carvalho, Tomasz Kowaltowski. (Org.). Porto Alegre: SBC, 2009. v. 1, cap. Modelos e Processos para o Desenvolvimento de Sistemas Sensíveis ao Contexto, p. 381 – 431.

VIEIRA, V. et al. Investigating the specifics of contextual elements management: The cemantika approach. In: KOKINOV, B. et al. (Ed.). *Modeling and Using Context*. [S.l.]: Springer Berlin / Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4635). p. 493–506.

Apêndice

Restrições OCL

As regras OCL apresentadas abaixo foram aplicadas ao metamodelo do CEManTIKA. Por motivos de organização e limitações técnicas da ferramenta, os metamodelos foram organizados em dois pacotes:

- `contextprofile`: organiza os elementos do metamodelo associados ao diagrama de Casos de Uso Enriquecido; e
- `cemantika_class`: é composto pelos elementos do modelo Conceitual de Contexto.

O pacote `contextprofile` define as três primeiras restrições apresentadas abaixo. O Código Fonte A.1 verifica se um *Agent* possui uma associação do tipo *executes*.

Lista de Códigos Fonte e Exemplos A.1 Regra OCL para um Agent

```
let associations : Set(Association) = Association.allInstances()->
    select(p | p.endType->exists(ends | ends = self.base_Actor))
in
associations->exists(association | association.getAppliedStereotype('
    contextprofile::executes') = contextprofile::executes )
```

O Código Fonte A.2 verifica se uma *Task* possui uma associação do tipo *executes*.

Lista de Códigos Fonte e Exemplos A.2 Regra OCL para uma Task

```
let associations : Set(Association) = Association.allInstances()->
    select(p | p.endType->exists(ends | ends = self.base_UseCase))
in
associations->exists(association | association.getAppliedStereotype('
    contextprofile::executes') = contextprofile::executes )
```

O Código Fonte A.3 verifica se uma associação do tipo *executes* é composta por uma *Task* e um *Agent*.

Lista de Códigos Fonte e Exemplos A.3 Regra OCL para uma Associação do tipo *executes*

```
self.base_Association.endType->exists(p | p.getAppliedStereotype('
    contextprofile::Agent') = contextprofile::Agent)
and
self.base_Association.endType->exists(p | p.getAppliedStereotype('
    contextprofile::Task') = contextprofile::Task)
```

As restrições abaixo verificam a corretude do modelo conceitual de Contexto. O Código Fonte A.4 verifica se uma Entidade Contextual possui ao menos um Elemento Contextual.

Lista de Códigos Fonte e Exemplos A.4 Regra OCL para uma Entidade Contextual

```
self.base_Class.member->exists(p | p.getAppliedStereotype('
    cemantika_class::ContextualElement') <> null)
or
let associations : Set(Association) = Association.allInstances()->
    select(assoc | assoc.endType->exists(ends | ends = self.base_Class
    )) in associations->exists(association | association.
    getAppliedStereotype('cemantika_class::ContextualElement') <> null
    )
```

O Código Fonte A.5 identifica se a propriedade *type* de um Elemento Contextual possui um dos seguintes valores: *ContextType::who, ContextType::what, ContextType::where, ContextType::when e ContextType::why*.

Lista de Códigos Fonte e Exemplos A.5 Regra OCL para um Elemento Contextual que verifica se a propriedade ContextType possui um valor válido

```
not self.type.oclIsUndefined() implies self.type <> cemantika_class::
    ContextType::none
```

Os Códigos Fonte A.6 e A.7 verificam ser um Elemento Contextual está associado a uma Entidade Contextual.

Lista de Códigos Fonte e Exemplos A.6 Regra OCL que verifica se um Elemento Contextual (Associação) está associado a uma Entidade Contextual

```
self.base_Association <> null implies self.base_Association.endType->
    exists(p | p.getAppliedStereotype('cemantika_class::
    ContextualEntity') <> null)
```

Lista de Códigos Fonte e Exemplos A.7 Regra OCL que verifica se um Elemento Contextual (Propriedade) está associado a uma Entidade Contextual

```
1 self.base_Property <> null implies self.base_Property.owner.
    getAppliedStereotype('cemantika_class::ContextualEntity') <> null
```

Sistema de Auxílio a Missões

Considere um sistema que permite pesquisadores planejarem suas missões. Uma missão é qualquer evento científico ou acadêmico destinado a pesquisadores, professores ou estudantes (e.g. conferência, reunião, aula, palestra).

A pessoa que participa de uma missão é chamada de *missionário*. As missões possuem algumas características (e.g. duração, local, tarefas a serem cumpridas). Diferentes missionários podem executar diferentes passos e ter requisitos específicos quando planeja uma missão. Por exemplo, em uma universidade, os passos a serem executados e os recursos disponíveis (e.g. suporte financeiro) para um professor são diferentes daqueles disponíveis para um estudante.

Para participar de um missão existem tarefas comuns a serem executadas, como: registrar um missionário no evento, requisitar suporte financeiro, procurar um hotel para hospedagem (nos casos onde a missão ocorre em uma cidade diferente da residência do missionário), comprar passagens (quando a cidade da missão for diferente do local onde o missionário mora), e alguns passos finais para finalizar a missão.

Esses requisitos estão resumidos no Diagrama de Cados de Uso (Figura B.1). Os requisitos podem ser realizados por um professor ou um estudante.

Os requisitos que podem ser influenciados pelo contexto incluem: identificar os passos necessários a serem executados de acordo com o tipo da missão e perfil do missionário; auxiliar o missionário a comprar uma passagem e a reservar uma acomodação; e identificar outros interesses relacionados à missão que podem ser úteis ao missionário (e.g. atividades acadêmicas, troca de câmbio ou dicas para passeios turísticos). O histórico do missionário em missões parecidas pode ser útil para identificar preferências e decisões prévias.

A Figura B.2 representa o diagrama Conceitual de Classes do sistema. Uma Pessoa

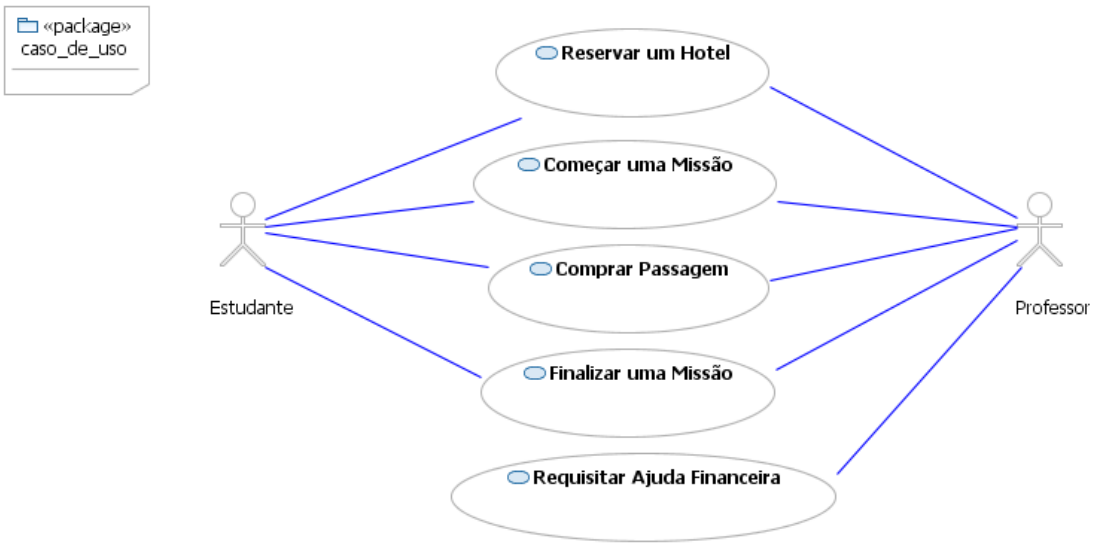

Figura B.1 Diagrama de Casos de Uso

tem os atributos idade, nome, corDosOlhos, disponibilidade e sexo. Um estudante tem o atributo nivelAcemico. Estudante tem um ou dois orientadores, e um professor pode orientar vários Estudantes. Uma Pessoa: participa de uma ou mais Missões; ehCliente de zero ou mais Hotéis, mora em um Local e estah em um Local. Uma Missão possui os atributos quemPaga, duracao, inicio, termino e tipo, e um relacionamento indicando seu local. Um Hotel tem os atributos: categoria, ehBarato e distanciaParaOCentro.

Escolha um foco e projete seus elementos estruturais e comportamentais de contexto de acordo com os conceitos definidos no CEManTIKA.

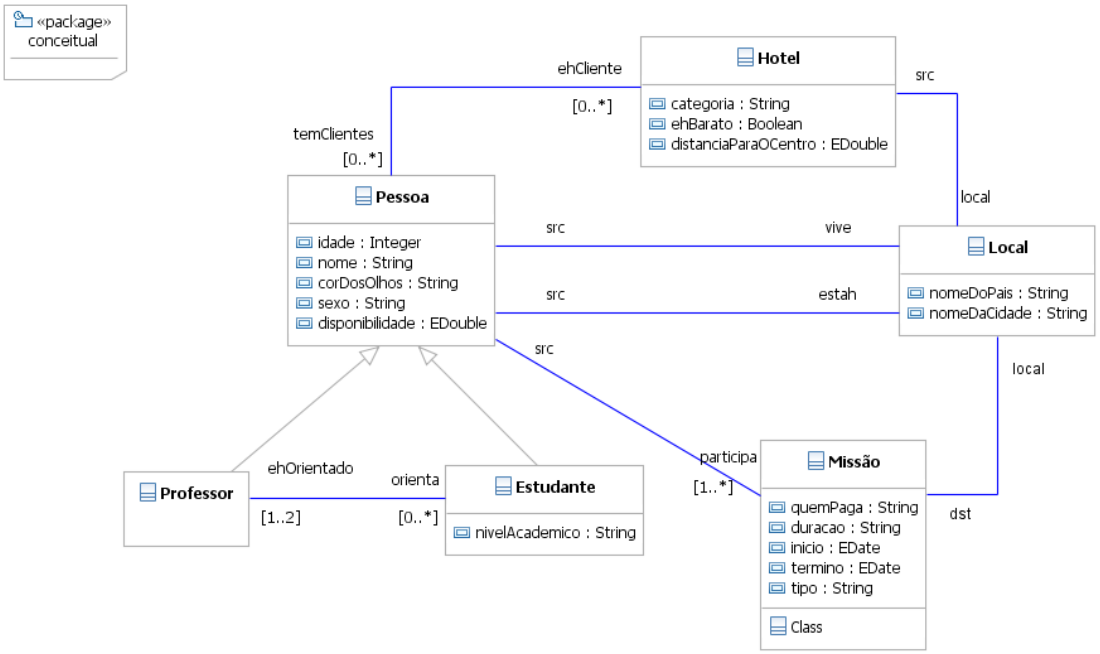

Figura B.2 Biblioteca do CEManTIKA Process

Questionário de Avaliação

Questionário de avaliação do uso do CEManTIKA CASE

Esse questionário tem o objetivo de obter as impressões sobre a aplicabilidade da ferramenta CEManTIKA CASE no projeto de sistemas sensíveis ao contexto e obter sugestões de possíveis pontos de melhoria.

Identificação do Participante

1. **Titulação**:
   - □ Mestre
   - □ Mestrando(a)
   - □ Especialização
   - □ Graduação
   - □ Graduando

2. **Usando uma escala entre Nenhum Conhecimento e Conhecimento Avançado, qual seu nível de conhecimento em:**

a. Sistemas sensíveis ao Contexto nenhum □—□—□—□—□ avançado

b. Modelagem usando UML nenhum □—□—□—□—□ avançado

c. Modelagem de Contexto nenhum □—□—□—□—□ avançado

d. IDE Eclipse nenhum □—□—□—□—□ avançado

e. Análise e Projeto de Sistemas nenhum □—□—□—□—□ avançado

f. Codificação de Sistemas nenhum □—□—□—□—□ avançado

g. Inglês (Leitura) nenhum □—□—□—□—□ avançado

3. Você já participou da construção de quantos sistemas sensíveis ao contexto?

□ Nenhum

□ 1 ou 2

□ Mais de 2

Avaliação do Experimento

4. O experimento foi realizado com o auxílio ou sem o auxílio da ferramenta CEManTIKA CASE?

□ Com o auxílio

□ Sem o auxílio

5. Quanto tempo você gastou para começar a produzir os artefatos?

□ Menos de 15 minutos

□ Entre 15 e 30 minutos

□ Entre 30 minutos e 1 hora

□ Não consegui produzir

6. Quanto tempo você utilizou para concluir o experimento?

□ Menos de 30 minutos

□ Mais de 30 minutos e menos de 1 hora

□ Mais de 1 hora

□ Não consegui concluir

7. Usando uma escala entre Não Entendi e Entendi Completamente, qual seu entendimento sobre as tarefas e seus artefatos de entrada e saída abaixo?

a. Identificar Focos

Não Entendi □—□—□—□—□ Entendi Completamente

b. Identificar Variações de Comportamento

Não Entendi □—□—□—□—□ Entendi Completamente

c. Identificar Entidades Contextuais e Elementos Contextuais
Não Entendi □—□—□—□—□ Entendi Completamente

d. Projetar o Modelo de Comportamento de Contexto
Não Entendi □—□—□—□—□ Entendi Completamente

8. Se você não usou o CEManTIKA CASE, que outras ferramentas foram utilizadas para executar o experimento?

□ Processador de Texto

□ Ferramenta Gráfica

□ Papel

□ Outras: _____

As perguntas abaixo devem ser respondidas apenas pelas pessoas que usaram a ferramenta CEManTIKA CASE. Ela é dividida em dois módulos:

- **CEManTIKA CASE Modeling** é um *plugin* do Eclipse [1] que permite a modelagem de sistemas sensíveis ao contexto através do *framework* CEManTIKA; e
- **CEManTIKA CASE Process** que explica as atividades, tarefas, artefatos e papéis do CEManTIKA de forma estruturada por meio de *hyperlinks*.

Avaliação da Ferramenta CASE Modeling

9. O uso da IDE Eclipse e seus conceitos já conhecidos pelos engenheiros de *software* ajudou no entendimento da ferramenta?

□ Sim

□ Não

□ Não conhecia a IDE Eclipse

Avaliação da Ferramenta CASE Process

[1] http://www.eclipse.org/

10. **A descrição do processo de projeto de sistemas sensíveis ao contexto através de atividades, guias, tarefas e seus artefatos de entrada e saída auxiliou na execução do experimento?**
    - □ Auxiliou
    - □ Não auxiliou
    - □ Não foi necessário usar a ferramenta

A próxima questão deve ser respondida se você escolheu a opção **Auxiliou** ou **Não Auxiliou** na questão anterior.

11. **Usando uma escala entre Não Relevante e Relevante, qual sua avaliação sobre os seguintes itens da ferramenta CEManTIKA CASE Process:**

    a. **Navegabilidade entre os elementos**
    Não Relevante □—□—□—□—□ Relevante

    b. **Descrição das tarefas**
    Não Relevante □—□—□—□—□ Relevante

    c. **Descrição dos artefatos**
    Não Relevante □—□—□—□—□ Relevante

    d. **Descrição dos papéis**
    Não Relevante □—□—□—□—□ Relevante

    e. **Descrição do Processo**
    Não Relevante □—□—□—□—□ Relevante

    f. **Guias: *checklist*, conceitos, exemplos, etc**
    Não Relevante □—□—□—□—□ Relevante

As próximas questões referem-se aos dois módulos da ferramenta.

12. **Que funcionalidades poderiam ser implementadas para facilitar o uso da ferramenta?**

    _____

    _____

_____

_____

13. Quais foram as principais dificuldades para a construção do sistema sensível ao contexto do experimento?

_____

_____

_____

_____