Universidade Federal da Bahia

Universidade Estadual de Feira de Santana

DISSERTAÇÃO DE MESTRADO

CEManTIKA Test Creator: Um Método de Geração de Casos de Teste para Aplicativos Móveis Sensíveis ao Contexto Baseado em Modelos e Padrões de Defeitos

André Luís Monteiro Pacheco dos Santos

Mestrado Multiinstitucional em Ciência da Computação

MMCC

Salvador – BA

2016

ANDRÉ LUÍS MONTEIRO PACHECO DOS SANTOS

CEMANTIKA TEST CREATOR: UM MÉTODO DE GERAÇÃO DE CASOS DE TESTE PARA APLICATIVOS MÓVEIS SENSÍVEIS AO CONTEXTO BASEADO EM MODELOS E PADRÕES DE DEFEITOS

Dissertação apresentada ao Mestrado em Ciência da Computação da Universidade Federal da Bahia e Universidade Estadual de Feira de Santana, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Vaninha Vieira dos Santos
Co-orientador: Adolfo Almeida Duran

Salvador
Novembro de 2016

Ficha catalográfica.

Santos, André Luís Monteiro Pacheco dos

CEManTIKA Test Creator: Um Método de Geração de Casos de Teste para Aplicativos Móveis Sensíveis ao Contexto Baseado em Modelos e Padrões de Defeitos/ André Luís Monteiro Pacheco dos Santos– Salvador, Novembro de 2016.

141p.: il.

Orientador: Vaninha Vieira dos Santos.
Co-orientador: Adolfo Almeida Duran.
Dissertação (mestrado)– Universidade Federal da Bahia, Instituto de Matemática, Novembro de 2016.

1. Sistema sensível ao contexto. 2. Teste baseado em modelo. 3. Padrão de defeito. 4. Geração de caso de teste. 5. Teste de sensor.
I. Vieira, V. II. DURAN, Adolfo Almeida.
III. Universidade Federal da Bahia. Instituto de Matemática. IV. Título.

ANDRÉ LUÍS MONTEIRO PACHECO DOS SANTOS

CEMANTIKA TEST CREATOR: UM MÉTODO DE GERAÇÃO DE CASOS DE TESTE PARA APLICATIVOS MÓVEIS SENSÍVEIS AO CONTEXTO BASEADO EM MODELOS E PADRÕES DE DEFEITOS

Esta Dissertação foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa Multi-institucional de Pós-Graduação em Ciência da Computação da UFBA-UEFS.

Salvador, 30 de novembro de 2016.

Profª. Drª. Vaninha Vieira dos Santos
Examinadora Interna

Prof. Dr. Ivan do Carmo Machado
Examinador Interno

Prof. Dr. Artur Henrique Kronbauer
Examinador Externo

AGRADECIMENTOS

O apoio de muitas pessoas me ajudou na realização deste trabalho, e é com muito prazer que exponho para cada uma a minha gratidão.

À minha orientadora Vaninha, obrigado por me ajudar nas questões cruciais da minha pesquisa, e por ter acreditado na minha capacidade de conduzir esta pesquisa. Aprendi muito com você neste período de orientação.

Ao meu coorientador Adolfo, que, assim como Vaninha, auxiliou no amadurecimento da proposta e também desde o início esteve ali junto.

Aos participantes do estudo empírico, cuja preciosa ajuda permitiu que eu conseguisse obter um retorno válido para as ideias propostas.

Agradeço também aos colegas do Grupo CEManTIKA, desde os que estavam na antiga formação quanto aos que estão no grupo renascido, os quais me auxiliaram em diversas etapas, desde o início das pesquisas até a conclusão deste trabalho, passando pelas duras prévias de defesa.

Agradeço aos meus colegas de trabalho pela compreensão neste longo tempo de mestrado.

E aos meus pais, irmãos e esposa, pelo incentivo, compreensão e cumplicidade.

RESUMO

O número de aplicações móveis e sensíveis ao contexto aumenta a cada dia. Estas aplicações precisam ser testadas para assegurar um nível adequado de qualidade. Quando a aplicação móvel depende de informações de contexto, sejam elas obtidas via sensores físicos e/ou virtuais, a complexidade e o custo relativos à etapa de testes aumenta consideravelmente, pois os sensores fornecem à aplicação um volume maior e mais diverso de dados em comparação com a inserção direta pelo usuário, como ocorre nas aplicações tradicionais.

Existem abordagens que almejam a geração de casos de teste com informações de contexto para estes aplicativos. Algumas delas utilizam modelos de contexto da aplicação para gerar casos de teste, ou então usam padrões de eventos de contexto. Estas abordagens possuem uma desvantagem no tipo de informação que é gerada, cujo formato geralmente é uma série de dados de sensores em baixo nível, como conjuntos de coordenadas GPS. Dados de contexto de alto nível, como a chegada, presença e saída de uma reunião, por exemplo, não são considerados por estas abordagens.

Esta pesquisa tem como objetivo investigar como a geração sistemática de casos de teste com dados de contexto em alto nível de abstração pode ajudar o testador de aplicativos móveis. Nesta dissertação propomos um método (denominado CEmanTIKA Test Creator) para geração de casos de teste onde o testador confecciona de forma sistemática os dados de contexto para os casos de teste. Para apoiar o testador na geração dos casos de teste, combinamos os dados de contexto obtidos nos modelos de contexto da aplicação com padrões de defeitos em fontes de contexto.

Para avaliar a proposta implementamos um protótipo dentro da ferramenta de modelagem de contexto CEManTIKA CASE e integramos os casos de teste gerados ao simulador de contexto Context Simulator. Realizamos um estudo empírico com uma abordagem qualitativa na qual participantes voluntários executaram o método proposto em uma aplicação móvel de teste denominada *NoCamInMeeting*. Ao analisar os dados obtidos, concluímos que a abordagem gera casos de teste com boa cobertura, embora com alguma redundância, e também que a divisão sistemática das tarefas para a geração dos casos de teste ajuda os testadores a poupar tempo e esforço.

Palavras-chave: sistema sensível ao contexto, teste baseado em modelo, padrão de defeito, geração de caso de teste, teste de sensor

ABSTRACT

The number of mobile and context-sensitive applications increases every day. These applications need to be tested to ensure an adequate level of quality. When the mobile application depends on context information, whether it is obtained via physical and/or virtual sensors, the complexity and cost of test phase increases considerably as the sensors provide the application with a larger and more diverse volume of data compared with direct insertion by the user, as in traditional applications.

There are approaches that aims the generation of test cases with context information for these applications. Some of them use application context models to generate test cases, or use context event patterns. These approaches are based on the type of information that is generated, whose format is usually a series of low-level sensor data, such as GPS coordinate sets. High-level context data, such as the arrival, presence and leaving of a meeting, for example, are not considered by these approaches.

This research aims to investigate how the systematic generation of test cases with context data in high level of abstraction can help the mobile application tester. In this dissertation we propose a method (called CEmanTIKA Test Creator) for generating test cases where the tester systematically prepares the context data for the test cases. To support the tester in the generation of test cases, we combine the context data obtained in the context models of the application with defect patterns in context sources.

To evaluate the proposal, we implemented a prototype within the CEManTIKA CASE context modeling tool and integrated the generated test cases into the Context Simulator tool for context simulation. We conducted an empirical study with a qualitative approach in which voluntary participants performed the proposed method in a mobile test application called *NoCamInMeeting*. After analyzing the gathered data, we conclude that the approach generates test cases with good coverage, but with some redundancy, and also that the systematic division of tasks into the generation of test cases helps testers to save time and effort.

Keywords: context-sensitive system, model based test, defect pattern, test case generation, sensor test

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BRS	Bigraphical Reaction System
CASE	Computer-Aided Software Engineering
CDP	Context Design Process
CEManTIKA	Contextual Elements Modeling and Management through Incremental Knowledge Acquisiton
CKTB	Context Knowledge Test Base
CTC	CEManTIKA Test Creator
DEVS	Discrete Event System Specification
EFSM	Extended Finite State Machine
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MBT	Model Based Testing
OCL	Object Constraint Language
PSCP	Ponto Sensível ao Contexto no Programa
SSC	Sistema Sensível ao Contexto
SSID	Service Set Identifier
UML	Unified Modeling Language
XML	Extensible Markup Language

Capítulo 1

INTRODUÇÃO

Neste capítulo apresentamos a motivação e justificativa para a pesquisa realizada, as questões de pesquisa e os objetivos, além da metodologia utilizada para abordar o problema de pesquisa. Ao final deste capítulo mostramos a estrutura do restante desta dissertação.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

O desenvolvimento de aplicativos móveis visando melhorar a eficiência e eficácia para a resolução de problemas de ordem comercial e tecnológica é uma tendência (Jones, 2015). A Figura 1.1 mostra que em 3 anos o número de aplicativos móveis que foram disponibilizados nas lojas das plataformas móveis Android e iOS mais do que dobrou (Statista, 2016a, 2016b). Como consequência deste crescimento, temos também um aumento no número de aplicativos que utilizam sensores para ajudar os usuários na execução de suas tarefas. Estas aplicações que são capazes de perceber o ambiente são denominadas sensíveis ao contexto (Schilit, Adams e Want, 1994).

Para obter sucesso frente à concorrência, estas aplicações, além de possuírem elementos que agradem o usuário, como uma boa usabilidade, devem possuir também qualidade, ou seja, devem se comportar conforme esperado pelo cada vez mais exigente usuário. Uma das maneiras de assegurar a qualidade de uma aplicação é a realização de testes, que são a verificação dinâmica para a descoberta de defeitos (Bourque e Fairley, 2014). Para a execução dos testes é preciso submeter dados à aplicação sob teste. Nas aplicações tradicionais, estes dados são disponibilizados geralmente através de interações diretas do usuário via interface homem-máquina. No caso dos aplicativos móveis sensíveis ao contexto, além dos dados inseridos pelo usuário, existem os dados fornecidos pelos sensores, os quais caracterizam a circunstância em que o usuário se encontra no ambiente (Vieira, 2008). Para testar aplicativos móveis com esta característica, o testador deve se preocupar em gerar dados de entrada que caracterizem informações de contexto, como localização, conectividade, disponibilidade, dentre outras. A combinação destes dados de

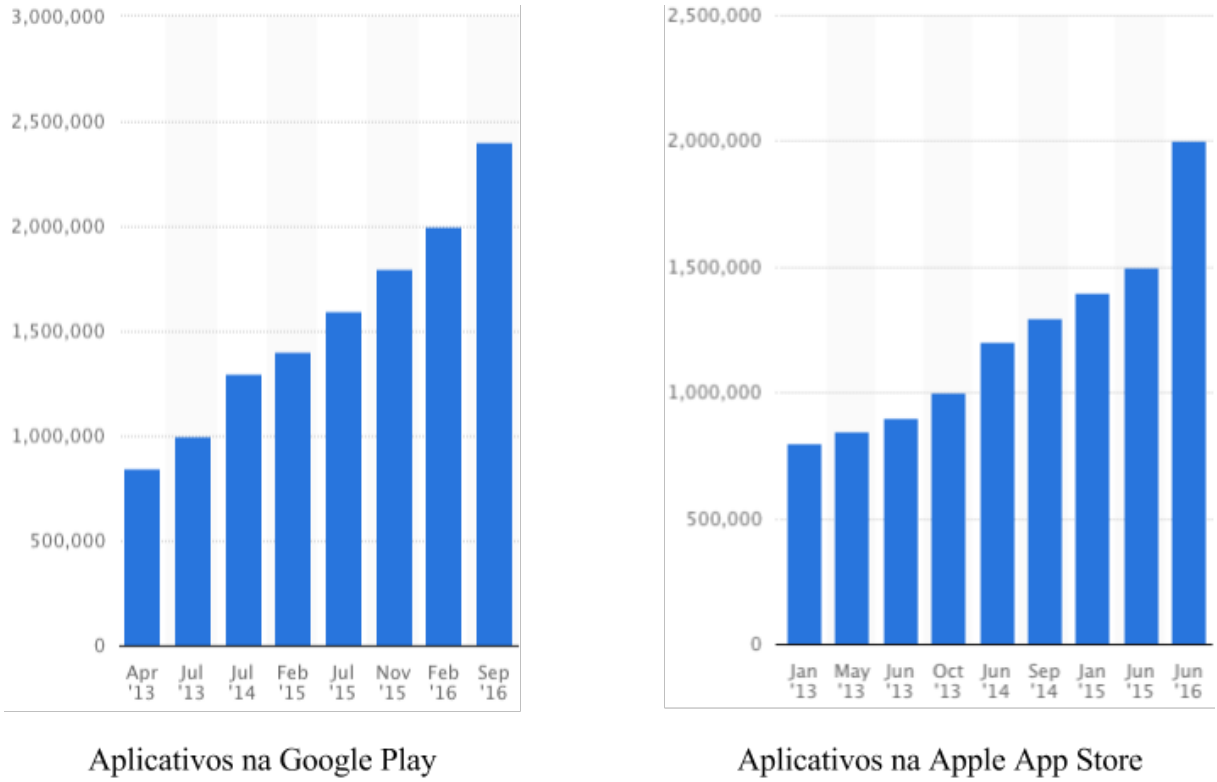

Figura 1.1 Crescimento no número de aplicativos disponíveis nos últimos 3 anos (adaptado de (Statista, 2016a) e (Statista, 2016b)).

contexto traz mais complexidade à tarefa de gerar os casos de teste para descobrir defeitos neste tipo de aplicação, pois estas fontes de contexto fornecem para a aplicação um volume grande e diverso de dados (McKinley et al., 2004; Muccini, Francesco e Esposito, 2012).

Para ajudar o testador a gerar casos de teste para este tipo de aplicação, alguns autores propuseram abordagens para contornar a complexidade adicional trazida pela adaptação aos dados contextuais, utilizando técnicas de teste baseadas em modelo (Garzon e Hritsevskyy, 2012; Griebe e Gruhn, 2014; Yu et al., 2014), ou técnicas que usam padrões de eventos de contexto para gerar os casos de teste (Amalfitano et al., 2013; Yu e Takada, 2015). A partir da análise das abordagens existentes identificamos que as mesmas geram casos de teste com dados de contexto no nível de sensor, ou seja, com baixo nível de abstração. Um exemplo de dado de contexto com baixo nível de abstração são coordenadas brutas de GPS, temperatura, dentre outras. Ao adicionar significado a estes dados brutos de sensores, os quais podem ser combinados entre si para caracterizar uma situação, aumentamos o nível de abstração de contexto. A presença em uma sala de reuniões é um exemplo de dado de contexto de alto nível.

Como consequências diretas desta ausência de representação de alto nível, temos: dificuldade no reuso de dados de contexto, pois a informação de contexto gerada não é agrupada de acordo com um significado que possa ser resgatado em outro momento (ex:

a presença numa reunião); e a ausência de uma sequência sistemática de passos a serem dados pelo testador para definir uma informação de contexto que seja significativa para os casos de teste, pois o preenchimento de dados de contexto é feito de forma *ad-hoc*. Com isto o trabalho do testador torna mais dependente de sua experiência, além de ser uma atividade repetitiva e propensa a erros, podendo causar impactos no resultado de seu trabalho.

1.2 QUESTÕES DE PESQUISA E OBJETIVOS

A questão de pesquisa que este trabalho visa responder é: "Sistematizar o preenchimento de dados de contexto de alto nível de abstração para geração de casos de teste pode ajudar o testador a descobrir defeitos em aplicativos móveis sensíveis ao contexto?"

A partir desta questão, definimos o objetivo geral deste trabalho, que é:

> "Definir um método de preenchimento sistemático de dados de contexto para geração de casos de teste para aplicativos móveis sensíveis ao contexto."

Para alcançar o objetivo geral, elencamos os seguintes objetivos específicos:

1. Definir e implementar os artefatos utilizados pelo método para ajudar o testador na geração dos casos de teste.
2. Sistematizar o uso destes artefatos na etapa de preenchimento dos dados de contexto para a geração dos casos de teste.
3. Verificar como o método proposto pode ajudar o testador.

1.3 METODOLOGIA

Para alcançar o objetivo geral e os específicos desta pesquisa, percorremos os passos descritos abaixo.

Para definir os artefatos utilizados pelo método para ajudar o testador na geração dos casos de teste, investigamos os trabalhos presentes na literatura. Vimos que as abordagens poderiam ser divididas em duas categorias: as que utilizam geração com uso de modelos (Garzon e Hritsevskyy, 2012; Griebe e Gruhn, 2014; Yu et al., 2014) e as que usam padrões de evento de contexto (Amalfitano et al., 2013; Yu e Takada, 2015). Após realizar uma análise destes trabalhos relacionados, elaboramos uma proposta que utiliza estes dois tipos de artefatos combinados: modelos da aplicação, que descrevem sua estrutura e comportamento em relação ao contexto; e padrões de defeitos de contexto, os quais são aplicados sobre os casos de teste confeccionados pelo testador.

Para sistematizar o uso destes artefatos na etapa de preenchimento dos dados de contexto para a geração dos casos de teste, criamos um conjunto de tarefas a serem executadas pelo testador. Para garantir que as informações de contexto não sejam perdidas entre o início e o fim desta etapa, elaboramos uma arquitetura com componentes e suas respectivas responsabilidades. Ao construir a arquitetura utilizada descobrimos que o método em questão causa uma explosão combinatória ao permutar os casos de teste confeccionados pelo testador com os padrões de defeitos em fontes de contexto. Para amenizar este efeito indesejado, utilizamos uma técnica para redução no conjunto

de casos de teste gerados. Definimos desta forma o método de geração de casos de teste denominado CEManTIKA Test Creator (CTC).

Para verificar como o método proposto pode ajudar o testador, implementamos um protótipo do método integrado a ferramentas de modelagem e de teste de aplicações. Depois, definimos e executamos estudos empíricos com testadores e desenvolvedores de aplicativos móveis utilizando o protótipo implementado. Este estudo consistiu em uma abordagem qualitativa com uso de entrevistas e observações. A partir dos resultados da avaliação coletamos informações relativas à ajuda prestada aos testadores pelo método proposto.

1.4 ESTRUTURA DO TRABALHO

Descrevemos abaixo a estrutura do restante desta dissertação.

O Capítulo 2 contém a revisão bibliográfica e apresenta: os conceitos referentes a contexto e teste de software; algumas ferramentas de apoio ao desenvolvimento e teste de aplicativos móveis sensíveis ao contexto; e a revisão e a análise comparativa com trabalhos correlatos.

O Capítulo 3 apresenta a proposta de solução, composta por uma arquitetura e um conjunto sistemático de tarefas a serem executadas pelo testador, além da implementação de um protótipo e sua execução em um aplicativo móvel sensível ao contexto.

O Capítulo 4 apresenta a avaliação da proposta de solução, a qual é feita a partir da definição, projeto, execução e análise de um estudo empírico. Neste capítulo apresentamos os prós e contras da proposta definida no Capítulo 3.

O Capítulo 5 apresenta um resumo do que foi proposto, discute as contribuições alcançadas e indica os trabalhos futuros descobertos a partir da realização da pesquisa apresentada nesta dissertação.

Capítulo 2

TESTE DE APLICAÇÕES MÓVEIS SENSÍVEIS AO CONTEXTO

Neste capítulo apresentamos os conceitos básicos relacionados ao teste em aplicativos móveis sensíveis ao contexto, ferramentas que podem apoiar o desenvolvimento e teste deste tipo de aplicação computacional, e os conceitos relacionados à geração de casos de teste. Ao final temos uma discussão das abordagens que tratam da geração de casos de teste para aplicativos móveis sensíveis ao contexto.

2.1 CONCEITOS BÁSICOS DE TESTE DE SOFTWARE

A realização de testes serve para evitar prejuízos decorrentes de falhas de *software*, nas diversas fases do desenvolvimento (Boehm e Basili, 2001). O teste é uma das formas de verificação existentes para determinar se a aplicação desenvolvida atende aos requisitos especificados, e também se a aplicação não desempenha um comportamento indesejado (Utting, Pretschner e Legeard, 2012). A atividade de teste é parte essencial de um processo de *software*.

Para conceituar o teste de *software*, usamos a definição presente em (Bourque e Fairley, 2014):

> Teste de *software* consiste da verificação **dinâmica** de que um programa provê o comportamento **esperado** num conjunto **finito** de casos de teste, sendo que estes foram adequadamente **selecionados** em um domínio de execução geralmente infinito.

As palavras em negrito presentes na citação acima ajudam a entender o escopo deste trabalho: o fato da verificação ser **dinâmica** implica na verificação sobre o programa em execução; **esperado** implica em definir se o comportamento observado corresponde ao resultado desejado na execução de um caso de teste; **finito** traz à tona o problema da inviabilidade da execução exaustiva de todas as possibilidades de entrada em um programa, e vai de encontro aos limites de custo e cronograma de projetos de sistemas

computacionais; **selecionados** trata da maneira como o conjunto de entradas que serão executadas é selecionado dentre todos os conjuntos possíveis de entradas para um dado programa.

2.1.1 Falhas e Faltas em Aplicativos Móveis

Após conceituar o teste de software, vamos aqui tratar de alguns termos comuns à esta etapa do desenvolvimento de software. A primeira distinção que fazemos é entre a causa de um mau funcionamento numa aplicação e seu resultado inesperado (em relação à especificação) observado. Para a causa do mau funcionamento, usamos o termo **falta**. Para descrever o resultado de uma falta, usamos o termo **falha**. Por exemplo, uma falta é um sensor de GPS não conseguir ler uma coordenada dos satélites enquanto uma pessoa se move, enquanto que a falha é a exibição de uma posição congelada num mapa de uma aplicação. **Defeito** é um termo que pode ser utilizado caso esta diferenciação não seja relevante (Lyu, 1996).

2.1.2 Tipo do Teste

Outro conceito que é de recorrente uso no texto é o tipo do teste, o qual pode ser de caixa-preta, caixa-branca, ou caixa-cinza, sendo que este último utiliza características de ambos. A Figura 2.1 mostra os pontos principais de diferença entre eles.

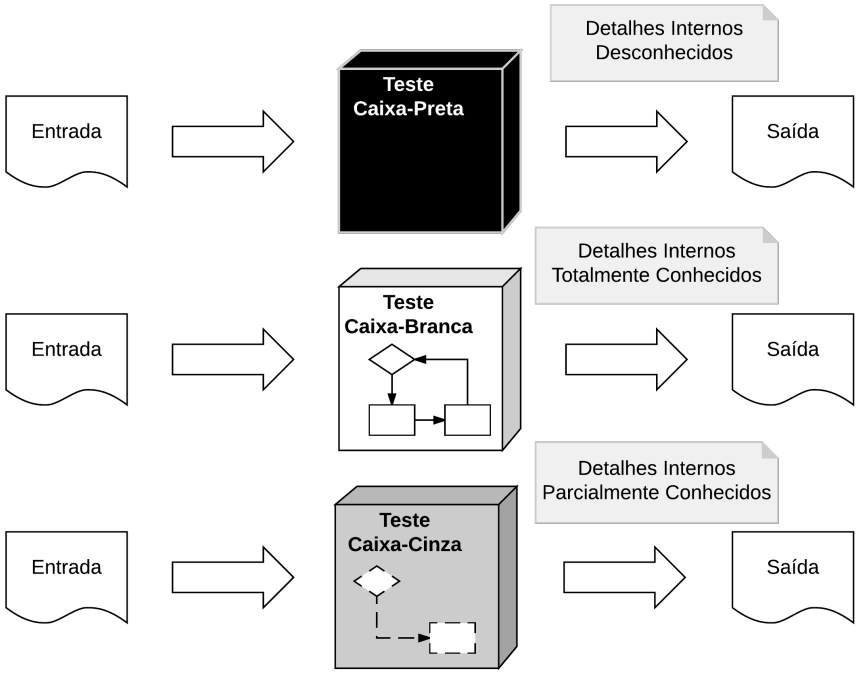

Figura 2.1 Diferença no nível de conhecimento exigido pelo testador em relação à aplicação entre o teste de caixa-preta, caixa-branca e caixa-cinza.

O teste de caixa-preta, dirigido a dados ou dirigido a entrada/saída é aquele que

considera a aplicação sob teste uma caixa preta. Para executar um teste deste tipo, o testador não precisa ter conhecimento da estrutura interna da aplicação. Neste tipo de teste a intenção é descobrir se a aplicação diverge da especificação (Myers, Sandler e Badgett, 2011). No entanto, o testador não tem ideia de que houve a cobertura de toda a extensão da aplicação testada. Para descobrir todos os defeitos o testador deve fornecer todas as combinações de entradas possíveis, o que em muitos casos é impossível (Copeland, 2003).

Já o teste de caixa branca, ou dirigido à lógica, é aquele que avalia a estrutura da aplicação sob teste. Um dos pontos de atenção em relação a esta técnica é que a mesma exige que o testador tenha conhecimento da estrutura interna da aplicação sob teste. Neste tipo de teste o objetivo é alcançar a execução de todas os fluxos de código (Myers, Sandler e Badgett, 2011). Uma desvantagem é que o número de fluxos a serem percorridos no código-fonte é muito grande para a descoberta de todos os defeitos da aplicação. Além disso, o teste de caixa-branca tem como limitação a existência de fluxos implementados. Caso um fluxo especificado não esteja presente no código-fonte a ausência deste fluxo não é detectada como falha (Copeland, 2003).

No teste de caixa-cinza, o testador não deve ter conhecimento de todos os detalhes internos da aplicação, mas pode saber como os componentes da aplicação sob teste interagem. Em vez de acessar o código-fonte da aplicação, o testador pode utilizar documentos do projeto da aplicação para realizar os testes. Uma vantagem é a combinação dos benefícios das abordagens de caixa-preta e caixa-branca, e, uma desvantagem é o fato do teste feito por esta técnica possuir cobertura limitada, pois o testador não tem acesso ao código-fonte da aplicação (Khan e Khan, 2012).

2.1.3 Caso de Teste

De acordo com Mcgregor (2007), temos a seguinte definição para caso de teste:

> "Um caso de teste é a descrição de um exercício a ser aplicado ao artefato sob teste."

Este mesmo autor define a representação de um caso de teste na forma de uma tupla de três elementos (pré-condições, entrada, saída esperada). As pré-condições são as condições necessárias para a execução do teste; a entrada compreende os dados que são inseridos na aplicação sob teste; e a saída esperada especifica o que a aplicação deve produzir ao executar a entrada na aplicação sob teste. A Tabela 2.1 mostra um exemplo de um caso de teste neste formato, onde o usuário deseja abastecer o carro e usa a aplicação para obter os postos de gasolina de acordo com sua localização.

Existe uma maneira mais simples de representar um caso de teste, que é a partir da tupla $(d, S(d))$, onde $d \in D$, sendo que D é o conjunto de entradas possíveis do programa, e $S(d)$ representa a saída esperada para a entrada d de acordo com uma especificação S (Machado, Vincenzi e Maldonado, 2010). Representação semelhante é apresentada em (Naik e Tripathy, 2011). Para este trabalho consideramos suficiente para caracterizar casos de teste a definição feita em (McGregor, 2007) e as representações apresentadas por (Machado, Vincenzi e Maldonado, 2010; Naik e Tripathy, 2011). Representações

Tabela 2.1 Especificação caso de teste onde o usuário de uma aplicação deseja abastecer o carro com gasolina e procura os postos mais próximos de sua localização.

Caso de teste
Pré-condição: Usuário está autenticado no sistema.
Ação: O usuário pesquisa por postos de gasolina próximos a ele.
Pós-condição: Com base na localização enviada pelo usuário, o sistema exibe uma lista ordenada de postos de gasolina num raio de 2km de distância.

mais complexas, como a que inclui na sua especificação a ordem de execução do caso de teste, como apresentado em (Copeland, 2003), excede aquilo que desejamos utilizar na representação de um caso de teste.

Utilizamos o termo "caso de teste abstrato" para descrever os casos de teste em sua forma não executável, ou seja, seu projeto ou plano de execução. O caso de teste apresentado na Tabela 2.1 é um exemplo de caso de teste abstrato. Para os casos de teste que podem ser executáveis, utilizamos o termo "caso de teste concreto". Um exemplo é o mostrado na Listagem 2.1, onde codificamos o caso de teste da Tabela 2.1 para o formato utilizado na ferramenta de teste automatizado calabash-android[1].

Listagem 2.1 Caso de teste concreto criado a partir do caso de teste abstrato da Tabela 2.1.

```
1 Scenario: Obtendo postos de gasolina por perto
2     Given I log in
3     When I touch "Buscar postos"
4     Then I wait
5     Then I should see a list of "Postos perto daqui"
```

2.1.4 Processo de Teste de Software

Para atingir sua finalidade, que é a descoberta de defeitos na aplicação, o testador necessita seguir um conjunto de passos entre o projeto do teste e a coleta dos resultados dos casos de testes executados. Para estruturar estes passos, diversos autores propõem processos de teste. Machado; Vincenzi; Maldonado (2010) relatam que de forma geral o processo de teste envolve os seguintes subprocessos, ilustrados na Figura 2.2: 1) Planejamento de Teste; 2) Projeto de Teste; 3) Execução de Teste; 4) Registro de Teste.

No subprocesso "Planejamento de Teste" todas as atividades relacionadas ao teste são planejadas e documentadas num plano de teste. Este plano de teste define quais partes do produto serão testadas, em que nível a aplicação deve ser testada, quais os recursos necessários, dentre outras informações que permitam a execução dos subprocessos seguintes.

No subprocesso de "Projeto de Teste" o objetivo é a elaboração dos casos de teste. Estes casos de teste são criados com a intenção de se alcançar o nível desejado de qualidade

[1] http://calaba.sh/

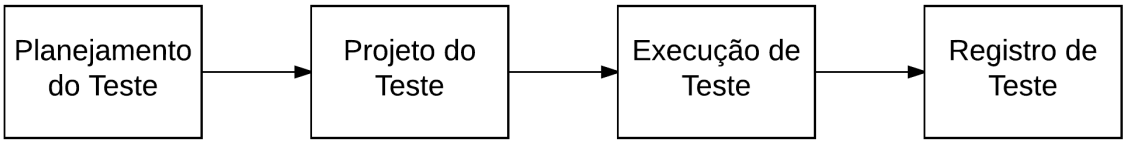

Figura 2.2 Processo de teste de *software* adaptado de (IEEE, 2008).

da aplicação. Na abordagem apresentada em (Graham et al., 2006), esta atividade está contemplada na implementação do caso de teste, e, além da definição dos casos de teste, ocorre também sua priorização, ou seja, a definição de quais casos de teste devem ser executados primeiro. Além disso, são definidas as suítes de teste, as quais são conjuntos de casos de teste com objetivos em comum. Outra atividade prevista nesta etapa é a implementação e verificação do ambiente de teste. Assim que estes casos de teste estão prontos, os mesmos podem ser executados, o que é feito no subprocesso "Execução de Teste".

Ao executar os testes, os resultados produzidos são documentados no subprocesso "Registro de Teste". Caso ocorra alguma discrepância no que diz respeito à especificação da aplicação, a mesma deve ser registrada e sua correção deve ser realizada.

Como o foco deste trabalho é ajudar o testador na geração de casos de teste, que é uma atividade presente no subprocesso de Projeto de Teste, detalhamos na subseção a seguir, as técnicas que podem ajudar o mesmo nesta atividade. Para realizar a avaliação deste trabalho, precisamos executar os quatro subprocessos descritos acima. O planejamento, projeto, execução e registro dos testes são abordados no Capítulo 4 e nos procedimentos documentados no Apêndice C.

2.1.5 Geração de Casos de Teste

A geração de casos de teste é uma tarefa de natureza intensiva, e isto ocorre porque casos de teste utilizam recursos de forma intensa quando feitos de forma manual (Naik e Tripathy, 2011). Uma maneira de tornar esta atividade menos repetitiva e propensa a erros é o uso de geradores de casos de teste (Bourque e Fairley, 2014). A seguir listamos e discutimos alguns aspectos destas abordagens em relação ao que tratamos nesta dissertação.

De acordo com o *survey* apresentado em (Anand et al., 2013), casos de teste podem ser gerados de maneira geral a partir dos seguintes artefatos: estrutura e/ou código-fonte do programa; especificação e/ou modelo da aplicação; informações do domínio dos valores de entrada e saída da aplicação; e informações capturadas dinamicamente a partir da execução da aplicação. A seguir descrevemos algumas destas técnicas:

- **Execução simbólica:** Técnica de teste de caixa-branca que realiza análise do código-fonte da aplicação para gerar casos de teste. Em vez de fornecer dados de entrada diretamente – números, por exemplo, se utiliza símbolos que representam valores arbitrários para executar o programa sob teste (King, 1976).

- **Baseado em Modelos:** Técnica de caixa-preta ou caixa-cinza que utiliza modelos da aplicação para gerar casos de teste (El-Far e Whittaker, 2001). Os modelos utilizados podem ser aproveitados de outras fases do desenvolvimento da aplicação.
- **Combinatório:** Utiliza conceitos de *design* combinatório para gerar casos de teste. Embora de caixa-preta, o foco é a cobertura de código, com o menor número de combinações de entrada possível em comparação com o teste exaustivo (Cohen et al., 1996, 1997).
- **Aleatório:** Abordagem de caixa-preta que depende da identificação do domínio dos valores de entrada. A partir de então valores aleatórios são escolhidos para a execução dos casos de teste (Hamlet, 1994). A desvantagem desta abordagem é que se não for aplicada uma técnica de redução ou priorização corre-se o risco de gerar muitos casos de teste redundantes.
- **Aleatório adaptativo:** Descreve algumas variantes do método aleatório: Numa delas, procura distribuir as amostras de forma mais uniforme dentro de um intervalo, como apresentado em (Chen, Leung e Mak, 2004); Em outra, torna-se uma abordagem que usa o *feedback* de execuções anteriores para gerar casos de teste melhores (Pacheco et al., 2007).
- **Mutante:** É um método de caixa-branca baseado em falhas. Para testar um programa, diversas versões levemente modificadas são executadas de modo a representar erros comuns que programadores cometem na codificação. Dependendo do tipo de aplicação sua execução pode ser muito custosa.
- ***Fuzzing* e mutante de dados:** Diferentemente do teste mutante, nesta abordagem os dados são modificados, em vez da aplicação (Shan e Zhu, 2006).

Pelos pontos negativos descritos anteriormente para as abordagens de caixa-branca, como a necessidade do testador ter conhecimento sobre a estrutura interna da aplicação e a impossibilidade de descobrir defeitos causados por fluxos que não foram implementados, as mesmas não foram consideradas para a solução proposta para este trabalho. Estratégias de caixa-preta que geram dados aleatórios ou a partir de análise combinatória também não foram considerados neste trabalho, pois aplicações que utilizam dados de contexto recebem dados de outras fontes além da entrada do usuário, além disso os dados de contexto podem ser atualizados frequentemente (detalhamos os problemas relativos ao uso de sensores no teste de aplicativos móveis na Seção 2.2.2). Um exemplo que poderia trazer complicações na geração e execução do teste é a combinação entre dados de data (com 365 possibilidades dentro de um ano), horário (86400 segundos por dia) e localização (em qualquer ponto do globo terrestre) para determinar se é dia ou noite para o usuário de uma aplicação móvel. O uso de teste mutante pode ser interessante caso se consiga automatizar a execução e registro de teste feitos com os programas mutantes, os quais poderiam inclusive ser executados na nuvem, como ocorre no FireBase Test Lab[2].

Uma das abordagens encontradas que permite que os casos de teste possam ser criados e analisados de forma independente da codificação da aplicação é baseada em modelos (El-Far e Whittaker, 2001). Consideramos isto um ponto positivo. Também consideramos promissores os seguintes pontos desta abordagem (Machado, Vincenzi e Maldonado,

[2]https://firebase.google.com/docs/test-lab/

2010):

- Aumenta a eficácia no teste, pois ajuda os envolvidos no teste a focar e a cobrir o que é necessário a ser testado;
- Reduz o custo, ao possibilitar o reuso de artefatos gerados em outras fases do desenvolvimento, além de permitir a automatização da geração de artefatos de teste;
- Aumento na confiabilidade no teste devido ao fato de que a maioria dos artefatos de teste foram criados de forma automática em vez de manual.

Outra abordagem que pode ser considerada interessante é a que gera casos de teste com dados mutantes, como percebemos em algumas abordagens correlatas. Nesta dissertação nos referimos a este tipo de abordagem, quando direcionada a contexto, como baseada em padrões de eventos de contexto ou baseada em padrões de defeitos de fontes de contexto. Neste tipo de teste o testador pode criar um caso de teste base e derivar outros a partir de uma estratégia de mutação aplicada sobre o caso base.

2.2 CONTEXTO EM APLICATIVOS MÓVEIS

No âmbito da ciência da computação, sistemas capazes de interpretar e se adaptar ao contexto em que estão inseridos são chamados sensíveis ao contexto (Schilit, Adams e Want, 1994). Para Abowd et al. (1999), sistemas sensíveis ao contexto são definidos como sistemas capazes de utilizar informações de contexto para prover informações e serviços relevantes ao usuário, de acordo com a tarefa que o mesmo executa. Seguindo estas definições temos que aplicativos móveis que percebem o ambiente através de sensores (ou por outros meios) também são aplicativos sensíveis ao contexto.

Abowd et al. (1999) definem contexto como a informação que pode ser utilizada para caracterizar uma situação ou entidade, sendo que a entidade pode ser uma pessoa, um local, um objeto que é considerado relevante para a interação entre o usuário e a aplicação, sendo que estes mesmos também podem ser considerados entidades. Nesta abordagem o contexto é a representação dos dados obtidos de uma dada situação.

Para separar a interação e os dados provenientes das interações no corrente trabalho, adotamos a concepção de contexto e elementos contextuais, apresentada por (Vieira, 2008), onde o elemento contextual é uma porção de dados ou informação que permite caracterizar um domínio ou entidade, enquanto que o contexto de uma interação entre um agente e uma aplicação é um conjunto de elementos contextuais instanciados que são necessários para apoiar a execução de uma tarefa.

Outra definição importante para o entendimento da proposta que apresentamos é a do termo foco. Em (Vieira, 2008), foco é a representação de uma tarefa que é executada por um agente de software ou humano. Determina quais elementos contextuais devem ser instanciados e usados para compor o contexto.

Utilizando as definições contidas nos trabalhos apresentados previamente sobre contexto e sistemas sensíveis ao contexto, apresentamos a seguinte definição sobre sistemas sensíveis ao contexto, dada em (Vieira, 2008):

> "Sistemas sensíveis ao contexto são aqueles que gerenciam elementos contextuais relacionados a um domínio de aplicação e usam estes elementos para

> auxiliar um agente na execução de uma tarefa. Este suporte pode ser alcançado melhorando a percepção do agente sobre a tarefa ou então provendo adaptações para melhorar a performance da execução da tarefa."

Segundo (Vieira, Tedesco e Salgado, 2009), a principal diferença entre aplicações sensíveis ao contexto e as tradicionais é que nas aplicações tradicionais a entrada de dados no sistema é dada exclusivamente através da interação com usuário, enquanto que em sistemas sensíveis ao contexto as informações são capturadas de outras fontes, tais como bases de dados contextuais, mecanismos de raciocínio e as obtidas via monitoramento do ambiente com uso de sensores. Um exemplo de aplicativo móvel sensível ao contexto é o Waze[3], que provê dados de trânsito em tempo real, que inclusive sugere rotas de acordo com a intensidade do tráfego de automóveis nas vias, e também avisa se uma pessoa na lista de contatos está no mesmo trajeto que o usuário. No restante do texto iremos nos referir a este tipo de aplicação com o termo "aplicativo(s) móvel(is)".

2.2.1 Abstração de Contexto

Existem variados níveis de abstração referente ao conceito de contexto. Bettini et al. (2010) elaboram uma pirâmide (Figura 2.3), cujas camadas separam as informações de contexto de acordo com o nível de abstração. Na base da pirâmide encontra-se o nível mais baixo de abstração, onde os dados de contexto são obtidos sem nenhuma interpretação adicional a partir de sensores físicos. Vieira; Holl; Hassel (2015) propôs uma extensão no nível inferior da pirâmide de forma a acomodar sensores virtuais como fontes de contexto. Desta forma, além dos dados de sensores físicos, como a localização obtida através de GPS ou uma rede sem-fio, seriam contemplados também dados brutos de sensores virtuais como datas e horários de compromissos armazenados numa agenda digital. Estes dados brutos isolados são desprovidos de um significado que permita a inferência de contextos mais complexos. Para resolver este problema, foi definida uma camada intermediária na qual os dados de diversos sensores são aglutinados em uma composição de alto nível, chamada de situações por alguns autores. Definidos no nível superior, estão os relacionamentos entre estas situações. Um exemplo seria a temporalidade entre estas situações.

Baseado na definição de sensores presente em (Baldauf, Dustdar e Rosenberg, 2007), Yürür et al. (2016) adicionam mais um elemento à camada relativa aos dados de baixo nível de abstração: o sensor lógico, conforme exibido na Figura 2.4. Este tipo de abstração compreende uma combinação entre sensores físicos e virtuais, adicionando a eles informações de outras fontes de dados, como bancos de dados por exemplo. Além disso, eles relacionam cada nível da pirâmide com características da informação de contexto. No nível intermediário, caracterizado como contexto inferido temos: o contexto do dispositivo, com dados de conectividade, recursos e custos de comunicação; o contexto do usuário, incluindo perfil do usuário, localização geográfica, situação social, dentre outras; o contexto físico, como luz, nível de ruído, condições de trânsito; e o contexto temporal, que contempla dia, semana, mês, estações do ano, dentre outros. O nível superior da pirâmide representa a situação do usuário, em vez de uma relação de situações, como na

[3]http://www.waze.com

Figura 2.3 Pirâmide representando três níveis de abstração de contexto (Hassel, 2014) adaptado de (Bettini et al., 2010)

pirâmide da Figura 2.3. Em (Bettini et al., 2010) e (Vieira, Holl e Hassel, 2015) a aglutinação entre dados de diversos sensores é feita apenas na camada superior da abstração (apenas com sensores físicos em (Bettini et al., 2010), e com sensores físicos e virtuais no Contexto Lógico definido em (Vieira, Holl e Hassel, 2015)).

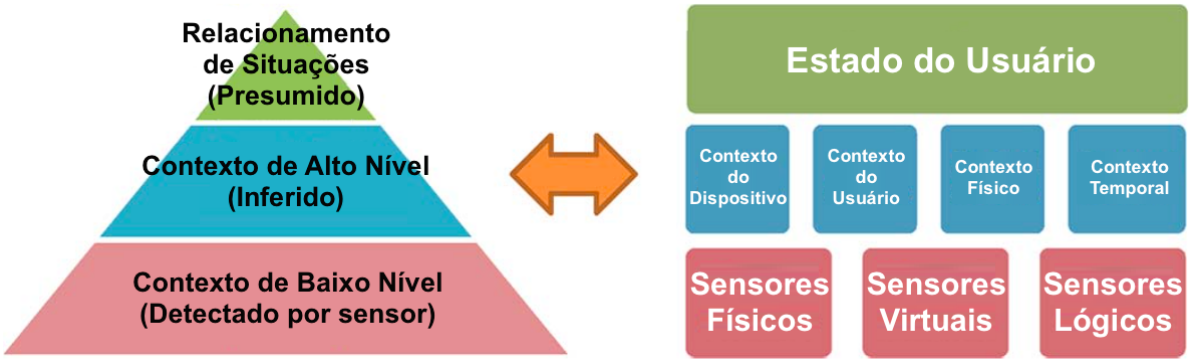

Figura 2.4 Pirâmide representando níveis de abstração de contexto segundo (Yürür et al., 2016)

Embora dados de sensores de qualquer tipo sejam considerados contexto de baixo nível de abstração em (Baldauf, Dustdar e Rosenberg, 2007) e (Yürür et al., 2016), os dados de um sensor lógico são inferidos a partir da combinação de dados de sensores físicos e virtuais, o que os caracterizariam como contexto de alto nível, como defendido por (Bettini et al., 2010) e (Vieira, Holl e Hassel, 2015). Para representar o contexto dentro dos casos de teste neste trabalho, a representação adotada é a apresentada em (Hassel, 2014) e ilustrada na Figura 2.5:

- *Contexto Físico*: Fonte de dados de baixo nível com uso exclusivo determinado sensor físico. Ex: localização, pressão atmosférica, dentre outros;
- *Contexto Virtual*: Fonte de dados de baixo nível obtidos a partir de sensores puramente virtuais. Ex: compromisso marcado numa agenda digital;

- *Contexto Lógico*: Fonte de informação de alto nível que representa o resultado da inferência realizada entre sensores físicos e virtuais. Ex: 1) A definição de uma sala de reuniões, que pode ser feita através da combinação entre a localização GPS e o SSID da rede sem-fio da referida sala; 2) O agendamento de uma reunião num dia e horário específicos;
- *Situação*: Contexto de alto nível que denota uma circunstância em que o usuário se encontra. Ex: Presença do usuário na sala de reuniões na hora da reunião. Neste caso, combina-se o dado de contexto da localização da sala com o horário marcado da reunião na agenda digital.
- *Cenário*: Surge quando se adiciona um encadeamento temporal entre diversas situações. Ex: Uma pessoa está numa sala de reuniões durante a reunião, sai da sala e volta instantes depois.

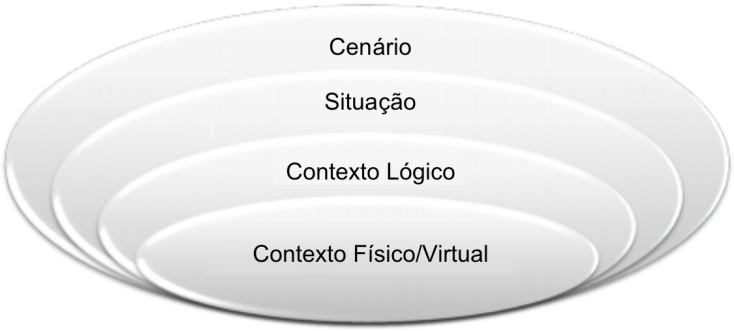

Figura 2.5 Níveis de abstração de contexto aninhados para representação de contexto em casos de teste para aplicativos móveis (Hassel, 2014)

2.2.2 Testes em Aplicativos Móveis Sensíveis ao Contexto

O teste de aplicações consome recursos, o que pode ocorrer de forma mais acentuada se as tarefas do processo de teste são manuais. Em aplicativos móveis este custo pode aumentar, pois sobre as características do teste de aplicações tradicionais adiciona-se as particularidades do teste de aspectos relativos a contexto, que são, de acordo com (Muccini, Francesco e Esposito, 2012; Wang, Elbaum e Rosenblum, 2007):

- Quantidade de sensores envolvidos;
- Valores capturados por sensores variam de acordo com o ambiente e/ou ações do usuário;
- Explosão combinatória causada pelo número de sensores e seus possíveis valores lidos;
- Interpretação correta do contexto, a qual impacta no comportamento da aplicação;
- Manipuladores de eventos de contexto podem operar de forma assíncrona para processar as alterações de contexto. Isso pode exigir suporte a multi-thread e pode levar a condições de corrida dentro de um manipulador de contexto ou entre mani-

puladores de contexto diferentes.

Analisando os itens mencionados, temos que quanto maior o número de sensores, maiores são as possibilidades de entrada, aumentando assim o esforço e custo para realização de testes. Além disso, a depender da aplicação, os dados de sensores precisam de atualização em pequenos intervalos de tempo. Por exemplo, uma aplicação de navegação precisa obter dados de GPS atualizados para ajudar o condutor a manter o percurso planejado até o destino; se a aplicação não obtém dados de forma atualizada, o usuário pode enfrentar situações adversas. E isto impacta nos custos da etapa de testes. Outra questão relativa a este tipo de aplicação é a junção de vários sensores para determinar um contexto significativo para a aplicação, sendo que para cada combinação de dados a aplicação pode ter um comportamento diferente.

Para reduzir o peso dos aspectos mencionados acima no teste do aplicativo móvel, além da automatização de algumas atividades do processo de teste, pode-se adotar as diversas formas de abstrair contexto como as que apresentamos na Seção 2.2.1. Afinal, o dado que a aplicação aceita como entrada de contexto (dados brutos de sensores, por exemplo) pode não ser significativo ou de fácil entendimento para um ser humano manipular, pois este compreende com mais facilidade uma circunstância do usuário. Por exemplo, reconhecer que é dia ou noite é mais fácil para uma pessoa do que ler uma coordenada GPS bruta e uma data e horário e inferir se é dia ou noite, por exemplo.

2.2.3 Padrões de Defeitos Associados a Sensores Físicos e Virtuais em Aplicativos Móveis

Para elaborar casos de teste que se preocupam com problemas em sensores físicos e virtuais, Vieira; Holl; Hassel (2015) considera importante classificar estes defeitos em padrões. Levando isto em conta, encontramos algumas classificações para defeitos em sensores, como as feitas em (Elnahrawy e Nath, 2003), que são: ruídos causados por fontes externas; ruído de *hardware*; imprecisões; e diversas variações de ambiente que podem causar impactos numa aplicação que espera receber dados de sensores com certa confiabilidade. Já Ye et al. (2008) identificaram incertezas na medição de dados de sensores como fruto de: limitações técnicas do sensor (como atrasos, variações nas taxas de atualização, dentre outras); ruído ambiental (que pode afetar a exatidão, como atenuação de sinal Wi-fi causada por objetos no ambiente); e o comportamento do usuário (que pode desligar ou modificar configurações dos sensores do dispositivo).

Sama et al. (2008) fazem um mapeamento das principais falhas que podem ser causadas por faltas em sensores. Existem sete categorias de faltas, as quais causam uma ou mais falhas, conforme mostra a Figura 2.6.

- **Incompleto:** Ocorre quando a informação de contexto é fornecida de forma incompleta.

  - Indisponível: O dado da fonte de contexto em algum momento não se encontra mais disponível para a aplicação. Ex: Sensor GPS não recebe mais dados de coordenadas;

  - Não Interpretável: A informação de contexto é transmitida parcialmente para a aplicação, de forma que a aplicação não consegue atribuir um significado

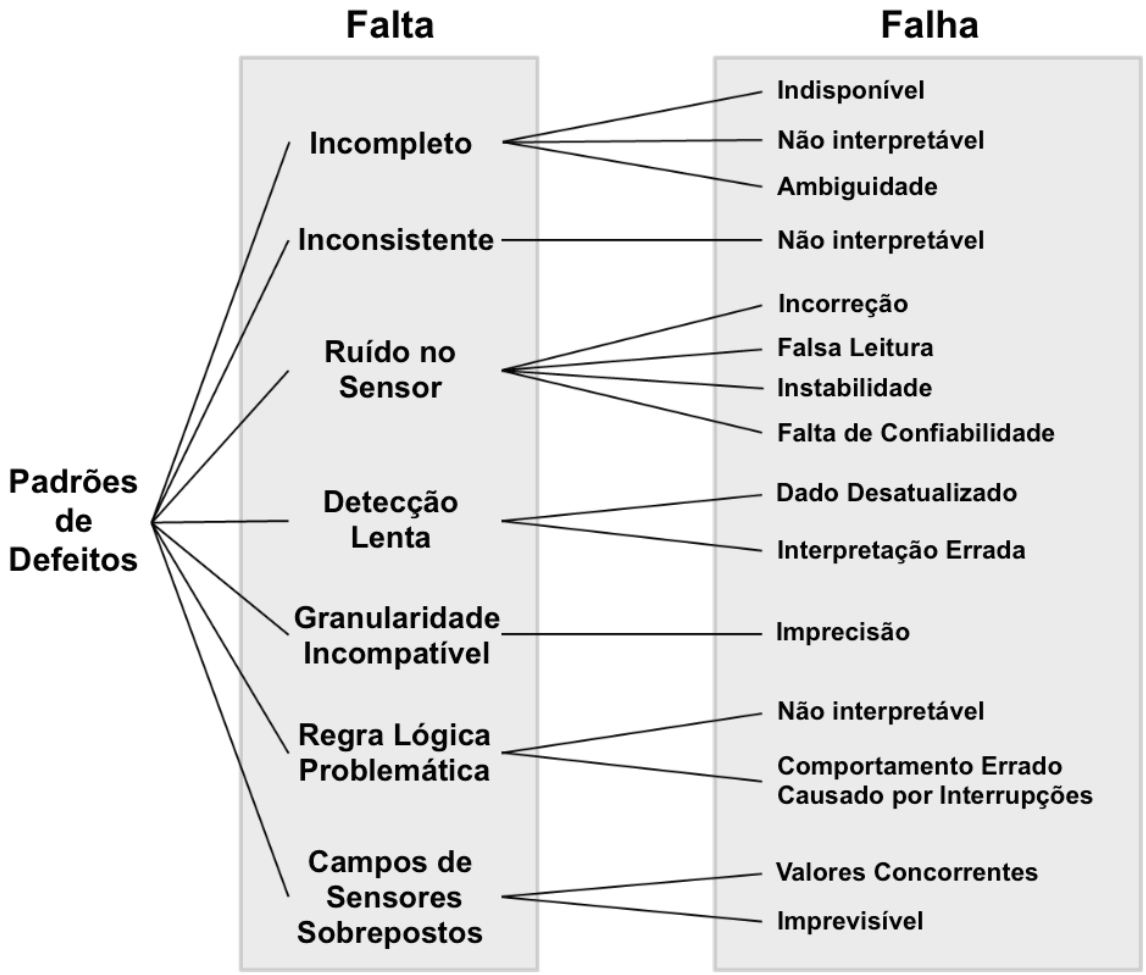

Figura 2.6 Padrões de defeitos que podem afetar sensores físicos em um aplicativo móvel (Hassel, 2014), adaptado de (Sama et al., 2008).

válido para a informação recebida. Ex: O fluxo de dados do sensor GPS é interrompido e apenas a coordenada de latitude é obtida;

- Ambiguidade: Ocorre quando a informação é enviada de forma incompleta e a aplicação pode interpretar de forma ambígua o dado recebido da fonte de contexto. Ex: Informar um horário 09:30 sem indicar se é AM/PM.

- **Inconsistente:** Ocorre quando fontes de contexto relacionadas não fornecem dados consistentes entre si de acordo com o ambiente.

  - Não Interpretável: Ocorre quando duas ou mais fontes de contexto dependentes entre si fornecem dados conflitantes.
- **Ruído no Sensor:** Ocorre quando o dado adquirido pela fonte de contexto varia entre o certo e o errado, ou quando a potência do sinal de uma fonte de contexto é interpretada incorretamente.

  - Incorreção: Ocorre quando uma fonte de contexto transmite continuamente

dados válidos intercalados com dados inválidos. Ex: Dados de presença em ambientes *indoor* que são influenciados por objetos no ambiente;

- Falsa Leitura: Ocorre quando uma fonte de contexto transmite dados corretos e de repente a mesma emite um dado incorreto;

- Instabilidade: Ocorre quando uma ou mais fontes de contexto atualizam seus dados de maneira tão rápida que a aplicação não consegue se adaptar a um estado de contexto;

- Falta de Confiabilidade: Ocorre quando uma fonte de contexto altera seu comportamento de forma imprevisível. Ex: O sinal de wifi indica a presença do usuário em uma sala enquanto ele está no corredor.

- **Detecção Lenta:** A fonte de contexto demora a atualizar os dados de contexto.

  - Dado Desatualizado: Ocorre quando a aplicação recebe da fonte de contexto dados muito antigos em relação ao tempo da aplicação;

  - Interpretação Errada: Ocorre quando a aplicação passa a receber dados das fontes de contexto em uma taxa inferior ao esperado. Ex: Num carro a 60km/h o espaçamento entre dois sinais de GPS aumenta para 10 segundos e o carro percorre mais de 150 metros, de forma que um alerta do dispositivo deixa de fazer sentido para o condutor.
- **Granularidade Incompatível:** A fonte de contexto fornece dados diferentes para um mesmo contexto de usuário.

  - Imprecisão: Ocorre quando a aplicação recebe seguidas vezes dados de contexto pouco diferentes entre si no decorrer do tempo. Ex: duas coordenadas GPS dificilmente serão exatamente iguais, mesmo que tomadas do mesmo lugar.
- **Regra Lógica Problemática:** Problemas em fontes de contexto que ocorrem devido a problemas de interpretação de contexto.

  - Não Interpretável: Ocorre quando a aplicação espera um conjunto de dados de contexto relacionados mas o que é fornecido pelas fontes de contexto difere do que a aplicação espera. Ex: Aplicação deseja obter uma localização indoor combinando em sinal Wifi e bluetooth mas recebe apenas coordenadas GPS;

  - Comportamento Errado Causado por Interrupções: Ocorre quando o dispositivo aciona uma interrupção, a qual pode afetar a aquisição de contexto. Ex: A bateria do aparelho atinge um nível crítico de carga e o dispositivo entra em modo de economia de energia, alterando configurações de leitura de sensores.
- **Campos de Sensores Sobrepostos:** Dados de fontes de contexto diferentes podem levar a estados incompatíveis ou imprevisíveis.

  - Valores Concorrentes: Ocorre quando fontes de contexto informam dados conflitantes entre si. Ex: Pessoa leva o computador do trabalho para casa e se conecta à rede doméstica, de forma que o gerenciador de contexto percebe que ele está em casa pela rede, mas isto conflita pelo uso do computador do trabalho;

  - Imprevisível: Ocorre quando mesmos dados de contexto fornecidos pelas fontes de contexto podem desencadear comportamentos diferentes.

No seu trabalho, Hassel (2014) elaborou um caso de teste abstrato para cada falta/falha da relação da Figura 2.6, conforme exemplos mostrados nas Tabelas 2.2, 3.1, 3.2 e 3.3.

Tabela 2.2 Especificação de um caso de teste abstrato contendo a Falta Incompleto e Falha Indisponível para uma fonte de contexto. Adaptado de (Hassel, 2014).

Falta: Incompleto / **Falha:** Indisponível
Caso de teste abstrato
Pré-condição: Fonte de contexto está disponível. **Ação:** Fonte de contexto é desabilitada. **Pós-condição:** Fonte de contexto encontra-se indisponível

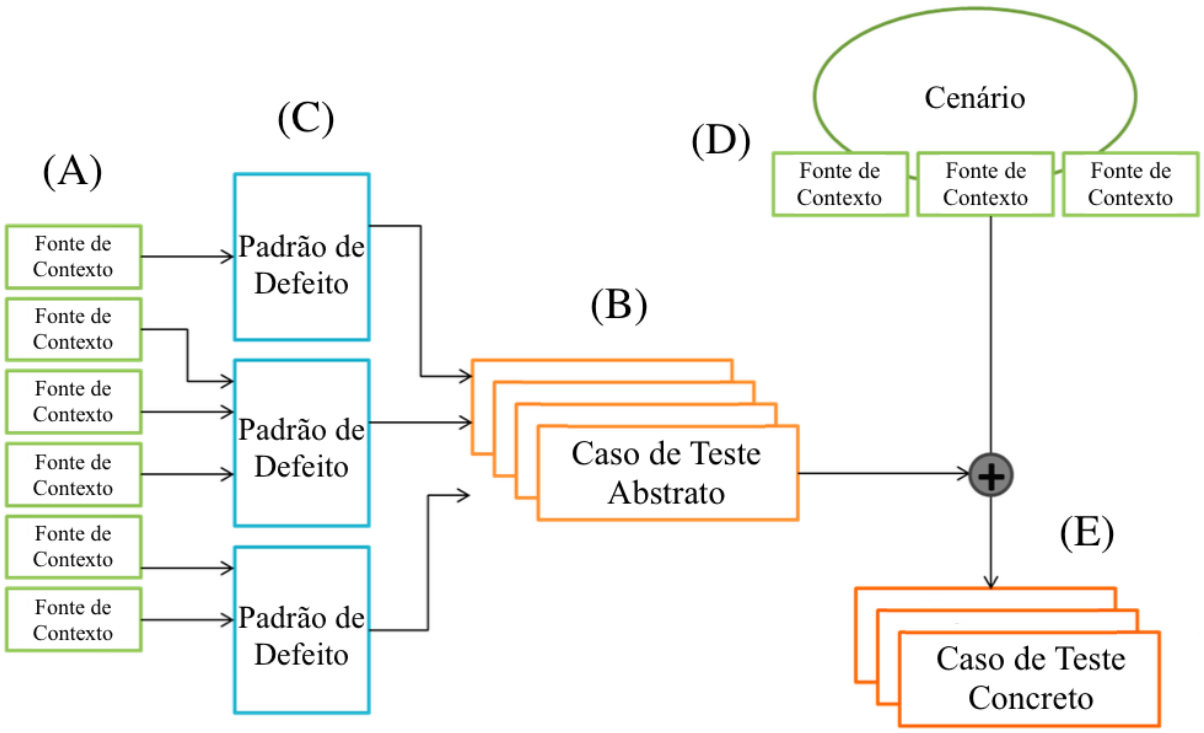

Figura 2.7 Relação entre os padrões de defeitos em fontes de contexto e os artefatos de teste, como casos de teste abstratos e concretos (Hassel, 2014)

Para a geração de casos de teste concretos, Hassel (2014) definiu uma relação de dependência entre fontes de contexto, padrões de defeitos, casos de teste abstratos, cenários com uso de fontes de contexto e casos de teste concretos, conforme mostra a Figura 2.7. De acordo com as fontes de contexto da aplicação (A), pode-se gerar casos de teste abstratos (B) combinando-os com padrões de defeitos em fontes de contexto (C). Ao combinar os casos de teste abstratos (B) com os cenários de contexto instanciados (D) temos os casos de teste concretos (E), os quais devem ser executados para a descoberta de defeitos na aplicação.

2.3 PROJETO E TESTE DE APLICATIVOS MÓVEIS SENSÍVEIS AO CONTEXTO

Nesta seção abordamos algumas ferramentas utilizadas no projeto e teste de aplicações móveis sensíveis ao contexto. Inicialmente tratamos dos fatores que nos levaram à escolha da ferramenta *Computer-Aided Software Engineering* (CASE) de especificação e projeto de sistemas sensíveis ao contexto CEManTIKA CASE (Patricio, 2010), e os conceitos que foram implementados nesta ferramenta. Depois tratamos das ferramentas que podem apoiar o teste deste tipo de aplicação.

2.3.1 Modelagem de Contexto com a Ferramenta CEManTIKA CASE

Existem diversas ferramentas CASE que apoiam o projetista no processo de desenvolvimento de aplicativos móveis. Nesta subseção discutimos os aspectos que nos levaram à escolha da ferramenta utilizada na proposta defendida por esta dissertação, que é a CEManTIKA CASE.

Os fatores que nos levaram a escolher esta ferramenta CASE para apoiar a geração de casos de teste de contexto na proposta deste trabalho são: o metamodelo de contexto adotado, pois é a partir do modelo da aplicação que os dados de contexto são obtidos; a aderência a um processo de desenvolvimento de sistema sensível ao contexto, pois pretendemos encaixar as atividades executadas na geração dos casos de teste dentro de um processo de desenvolvimento de aplicativos móveis sensíveis ao contexto; e a possibilidade de ajustes em sua implementação caso não exista suporte para as atividades de teste, pois não pretendemos implementar uma ferramenta de modelagem completa para obter os modelos necessários para a geração dos casos de teste.

Para atender às necessidades de modelagem requeridas pela proposta apresentada neste trabalho, precisamos de um metamodelo que suportasse a aquisição de contexto e também a modelagem de comportamento contextual. Em outras palavras, este metamodelo deve fornecer a informação dos sensores que são utilizados para a aquisição de um determinado elemento contextual num cenário de uso. O metamodelo definido no CEManTIKA comporta este requisito, e o fato dele ser proposto sobre a UML torna isto ainda mais favorável, pois além de ser extensível, a UML como forma de representação gráfica pode ser mais familiar aos projetistas, se comparada a outras formas de representação de contexto, como redes de Petri (Püschel, Seiger e Schlegel, 2012), ou ontologias (Gu, Pung e Zhang, 2004), dentre outras (Perera et al., 2014).

Como estamos propondo uma abordagem sistemática para a geração dos casos de teste, é interessante que esta fase esteja contida em um processo já definido de desenvolvimento de aplicativos móveis. A ferramenta CEManTIKA CASE foi desenvolvida para implementar, além do metamodelo de contexto, o processo de projeto de contexto (*Context Design Process* - CDP) definido em (Vieira, 2008). Embora a etapa de teste seja mencionada no processo, a mesma não foi detalhada no CEManTIKA, nem implementada no CEManTIKA CASE. Como o código-fonte da aplicação se encontra num repositório público[4], isto facilita a realização dos ajustes necessários na ferramenta. Além disto, o de-

[4]https://github.com/raphaeljpb/cemantika-modeling

senvolvedor estava disponível para tirar dúvidas em relação a aspectos da implementação. A seguir detalhamos os pontos principais desta ferramenta de apoio ao desenvolvimento de sistemas sensíveis ao contexto.

Para gerar os artefatos previstos pelo framework conceitual CEManTIKA (Vieira, 2008) e apoiar projetistas de sistemas sensíveis ao contexto (SSC), Patricio (2010) propôs a implementação de uma ferramenta. Esta ferramenta é denominada CEManTIKA CASE, cuja pretensão é sistematizar o desenvolvimento de SSC através do processo e metamodelo de contexto definidos em (Vieira, 2008). O CEManTIKA CASE foi dividido em dois módulos, o CEManTIKA Modeling e o CEManTIKA Process. Antes de mostrar a ferramenta, apresentamos alguns conceitos definidos no framework conceitual CEManTIKA (Vieira, 2008).

2.3.1.1 Framework CEManTIKA

Com o objetivo de separar as atividades relacionadas ao negócio da aplicação e dos elementos relacionados ao contexto, Vieira (2008) propôs o framework conceitual CEManTIKA (Contextual Elements Modeling and Management through Incremental Knowledge Acquisiton).

O framework proposto apoia o processo de desenvolvimento de SSC ao abordar três pontos: (i) no projeto de elementos arquiteturais relativos à manipulação de contexto; (ii) na especificação e representação do contexto, de forma independente do domínio da aplicação; (iii) na modelagem e especificação do sistema sensível ao contexto. Para apoiar os desenvolvedores, o framework compreende a elaboração dos seguintes artefatos: *Arquitetura de Manipulação de Contexto*; *Metamodelo de Contexto*, associado a perfis UML e um *Processo de Projeto de Sistemas Sensíveis ao Contexto*. Detalhamos abaixo o Metamodelo de Contexto, com os metamodelos estrutural e comportamental, e o Processo de Projeto de Sistemas Sensíveis ao Contexto.

2.3.1.2 Metamodelo Estrutural do CEManTIKA

O metamodelo estrutural trata do modo como o contexto deve ser modelado em uma aplicação sensível ao contexto. Neste metamodelo são abordados os seguintes conceitos: a entidade contextual, a fonte de contexto, elemento de contexto e aquisição, tarefa, agente, papel, relevância, foco e regra . Ou seja, esta parte do framework CEManTIKA permite a modelagem de artefatos que compõem um sistema sensível ao contexto. A Figura 2.8 ilustra a constituição do metamodelo de contexto.

Os seguintes elementos do metamodelo são relevantes para este trabalho:

- *Focus* (Foco): Representação de uma Tarefa (*Task*) que é executada por um Agente (*Agent*) de software ou humano. Determina quais elementos contextuais devem ser instanciados e usados para compor o contexto.
- *ContextualEntity* (Entidade contextual): Representa as entidades que devem ser consideradas para a manipulação de contexto. Exige a definição de ao menos um elemento contextual.

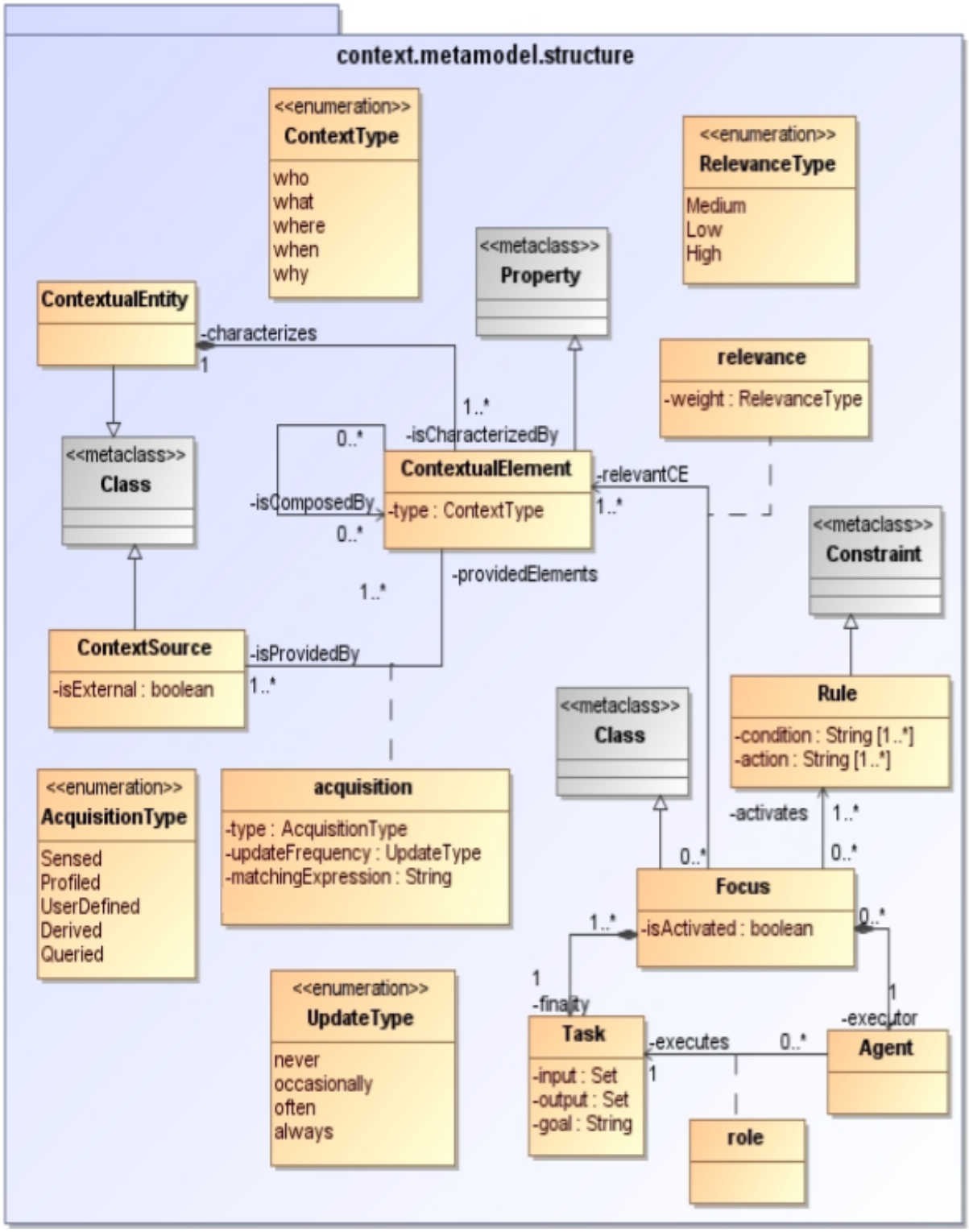

Figura 2.8 Metamodelo de contexto (Vieira, 2008)

- *ContextSource* (Fonte de contexto): Descreve a fonte pela qual os valores possíveis para o elemento contextual são adquiridos.
- *ContextualElement* (Elemento de contexto): É a unidade mínima do metamodelo estrutural e representa a definição de uma propriedade utilizada para caracterizar uma Entidade Contextual.
- *Acquisition* (Aquisição): Representa a associação entre um elemento contextual e uma fonte de contexto. Qualifica o modo como o elemento contextual é preenchido pela fonte de contexto.

Os demais componentes do metamodelo estrutural não são abordados na solução proposta neste trabalho, portanto seu detalhamento não se faz necessário.

2.3.1.3 Metamodelo Comportamental do CEManTIKA

O metamodelo da categoria comportamental define como o comportamento do SSC é afetado pelo contexto. Para tanto, foi usado o formalismo definido para o grafo contextual (Brézillon, Pasquier e Pomerol, 2002): nó contextual, ação, nó de recombinação, subgrafo, atividade e agrupamento de ação paralela. As restrições do grafo contextual original também foram mantidas, como: existência obrigatória de um nó inicial e um final, além da exigência de correspondência entre um nó contextual e um nó de recombinação. Seguem as definições dos elementos formais do grafo contextual:

- **Nó contextual:** Representa uma decisão que depende de contexto. Um nó contextual possui apenas uma aresta de entrada e n de saídas, as quais representam as n instâncias do elemento contextual avaliado na decisão do nó. Presentes na Figura 2.9 como os círculos claros.
- **Ação:** Representa uma ação a ser feita. Pode ser um passo na solução de um ou uma tarefa. Representado pelos retângulos na Figura 2.9.
- **Nó de recombinação**: Vértice para onde os caminhos iniciados por um nó contextual convergem. Representa a finalização de um caminho do grafo. Presentes na Figura 2.9 como os círculos escuros.
- **Subgrafo:** São grafos contextuais também. São utilizados para estruturar grafos mais complexos, de forma a permitir a abstração de componentes do grafo para uma melhor visualização, por exemplo.
- **Atividade:** Tipo especial de subgrafo, que define uma sequência padrão de ações a serem executadas.
- **Agrupamento de ação paralela:** Permite a execução em ordem aleatória ou simultânea mais de um subgrafo.

2.3.1.4 Processo de Projeto de Contexto

O CDP contempla as seguintes atividades para a elaboração de um SSC: Especificação de Contexto, Projeto de Gerenciamento do Contexto, Projeto de Uso do Contexto, Geração de Código, Teste e Avaliação do SSC. No trabalho original, Vieira (2008) define apenas as três primeiras atividades, inclusive atribuindo responsabilidades nas ati-

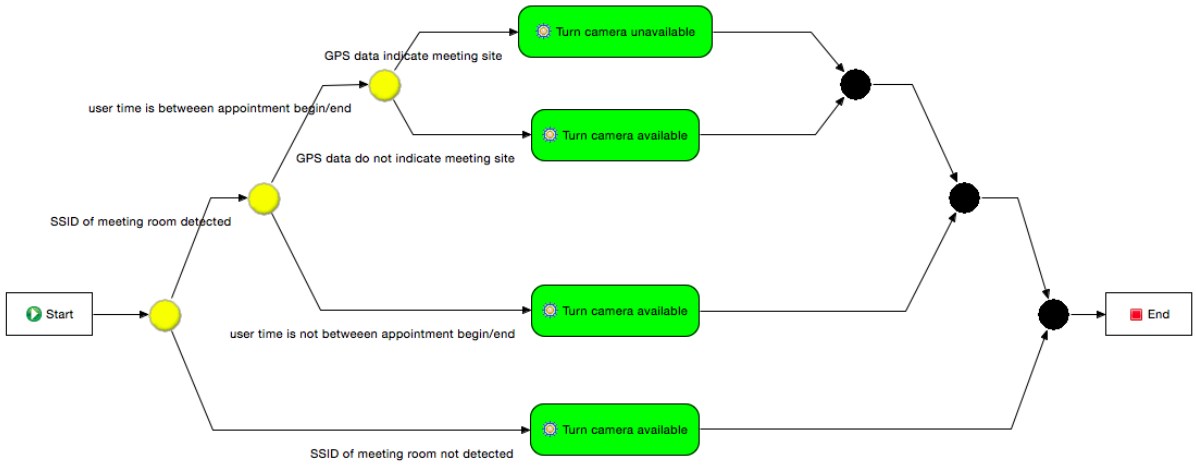

Figura 2.9 Grafo contextual de uma aplicação que gerencia a disponibilidade de uma câmera de acordo com o contexto do usuário.

vidades para os papéis. Detalhamos a seguir estas três atividades, onde algumas estão implementas no CEManTIKA CASE. Não detalhamos as outras atividades do CDP pois elas não são relevantes a esta dissertação.

A Especificação de Contexto tem como função identificar os requisitos de contexto e criar os modelos conceitual e comportamental de contexto, tendo como base os requisitos de negócio do SSC. Possui as seguintes tarefas: Identificar Focos (S1), Identificar as Variações de Comportamento (S2), Identificar Entidades Contextuais e Elementos Contextuais (S3) e Verificar a Relevância dos Elementos Contextuais (S4). A última tarefa é opcional.

Depois de identificar os requisitos de contexto (S1 e S2) e gerar o modelo conceitual de contexto (S3), o projetista deve investigar como os elementos contextuais devem ser adquiridos, processados e compartilhados na atividade Projeto de Gerenciamento de Contexto. Esta atividade possui as 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). Ao final desta etapa o modelo conceitual de contexto é enriquecido com as definições de aquisição de contexto.

Após a execução das atividades descritas anteriormente, o Projeto de Uso de Contexto visa apoiar o projeto de especificação de como o contexto será efetivamente usado no SSC. O processo divide esta atividade em duas tarefas: Projetar Adaptação de Contexto (U1) e Projetar Apresentação de Contexto (U2).

2.3.1.5 Implementação do CEManTIKA CASE

Com os conceitos apresentados, podemos agora apresentar a ferramenta, naquilo que é relevante para este trabalho. Para ilustrar, usamos as etapas executadas no CEManTIKA CASE na aplicação exemplo apresentada na Seção 3.4.

O CEManTIKA Modeling foi implementado como um plugin do Ambiente Integrado

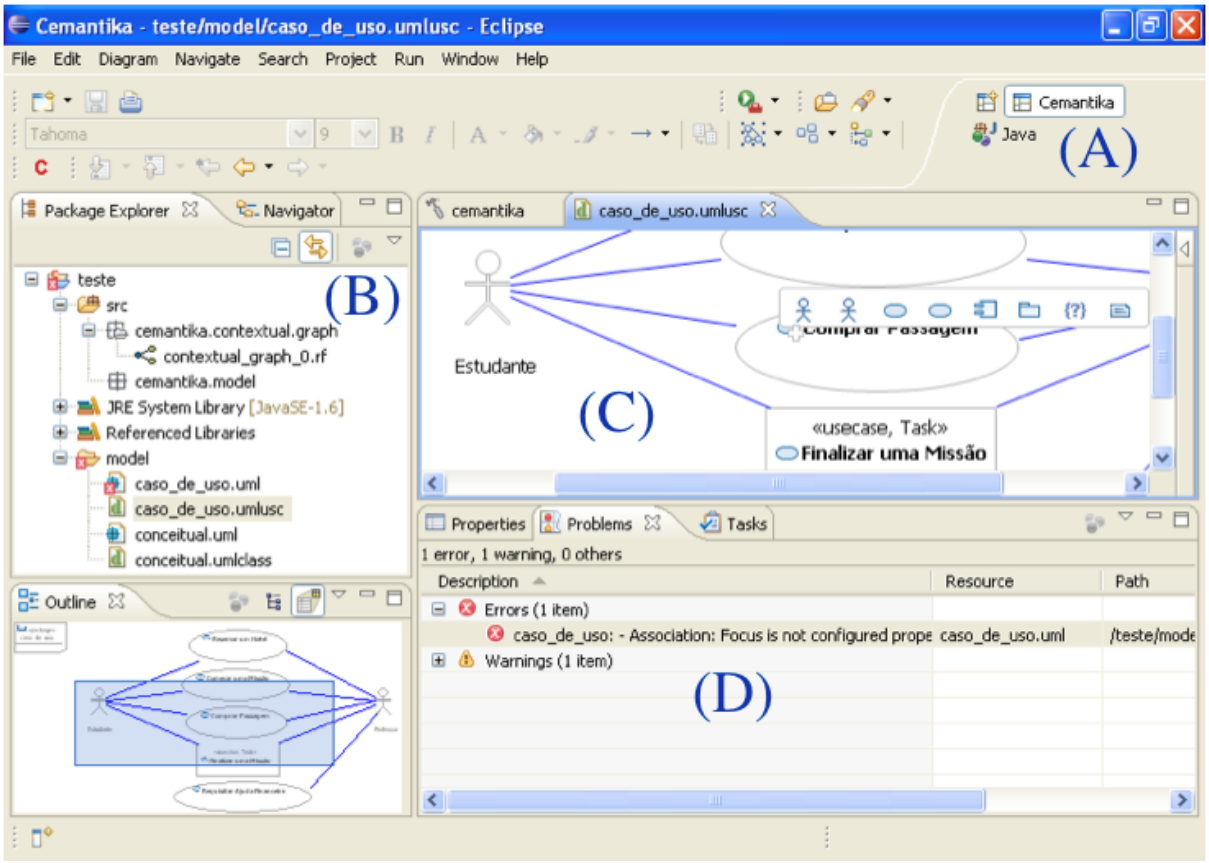

Figura 2.10 Modelagem de contexto com uso do metamodelo definido no *framework* CEManTIKA na ferramenta CEManTIKA Modeling (Patricio, 2010).

de Desenvolvimento (IDE na sigla em inglês) Eclipse, e é responsável pelo projeto do sistema sensível ao contexto. A Figura 2.10 exibe uma tela do CEManTIKA Modeling em execução, onde temos a IDE Eclipse com a perspectiva do CEManTIKA acionada (A) para a elaboração do projeto de contexto (B). Além de usar os diagramas UML pra projeto de SSC (C), a ferramenta provê validações nos modelos elaborados pelo projetista, através de verificação de restrições definidas em Object Constraint Language (OCL), de modo que o responsável pelo projeto consiga identificar falhas na elaboração dos diagramas UML (D).

O CEManTIKA Process é uma biblioteca que descreve o CDP. Esta biblioteca provê informações ao desenvolvedor através de um *site*, de forma a apoiar o desenvolvedor na aquisição do conhecimento contido no framework CEManTIKA. Na ferramenta CEManTIKA CASE, as tarefas (S1), (S2), (S3) e (U1) do CDP foram implementadas na aba *Context Specification*, conforme exibido na Figura 2.11.

Com os requisitos da aplicação em mãos, o projetista executa a primeira tarefa da atividade especificação de contexto, que é Identificar Focos (S1). Nesta atividade o projetista define o foco da aplicação e gera um diagrama de caso de uso enriquecido. O foco da aplicação em questão é o gerenciamento do recurso de envio de dados. O diagrama

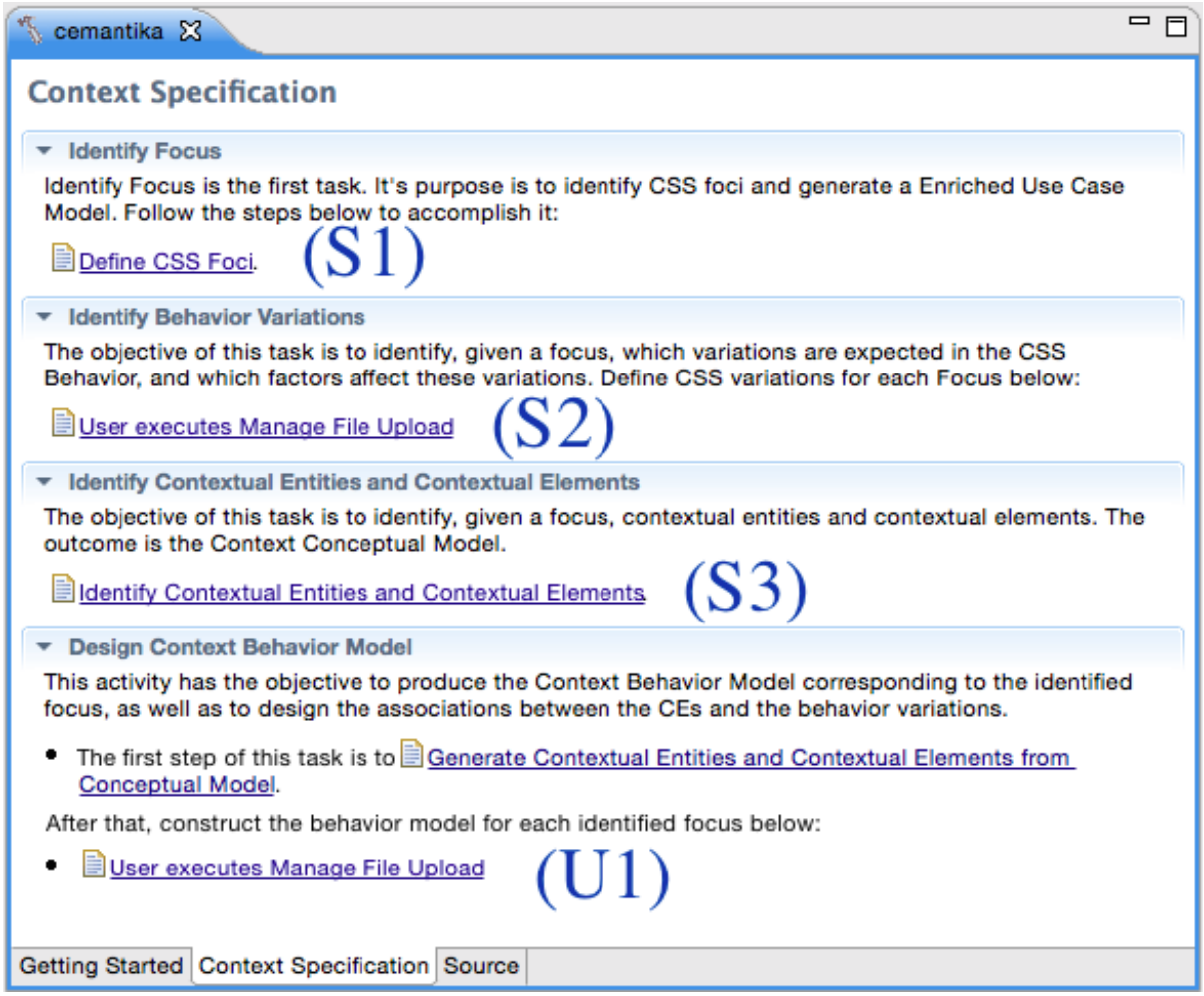

Figura 2.11 CEManTIKA Modeling (Patricio, 2010)

de caso de uso enriquecido com o agente que executa uma tarefa é o mostrado na Figura 2.12. Por limitações da ferramenta implementada, os estereótipos *Agent*, *Executes* e *Task* não são exibidos na figura.

Depois, o projetista executa a tarefa Identificar as Variações de Comportamento (S2). Estas variações de comportamento da aplicação estão associadas com os focos identificados na tarefa anterior, e são descritas em texto natural, como mostramos na Figura 2.13. Aqui o projetista define como a aplicação se comporta em dada situação contextual.

Depois de executar S2, o projetista executa então a modelagem conceitual da aplicação, na qual ele identifica as entidades contextuais e seus elementos contextuais. O diagrama conceitual de contexto elaborado neste passo está na Figura 2.14. Neste diagrama temos uma entidade contextual e dois elementos contextuais associados, que é o resultado da tarefa S3. A entidade contextual é o usuário, e este possui como elementos contextuais o SSID da rede Wi-fi e sua localização.

Após identificar as entidades contextuais e seus elementos contextuais, o projetista deve definir o comportamento da aplicação, de acordo com estes elementos contextuais.

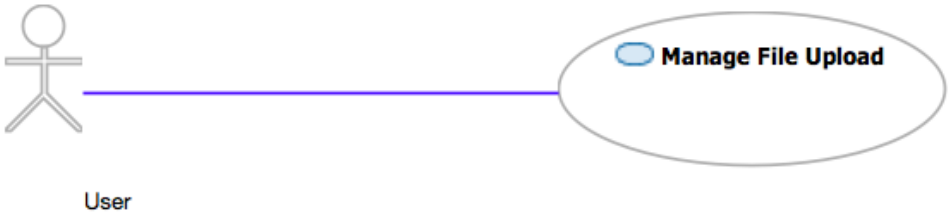

Figura 2.12 Diagrama de caso de uso enriquecido da aplicação *PersonalUploadManager*, onde definimos o foco.

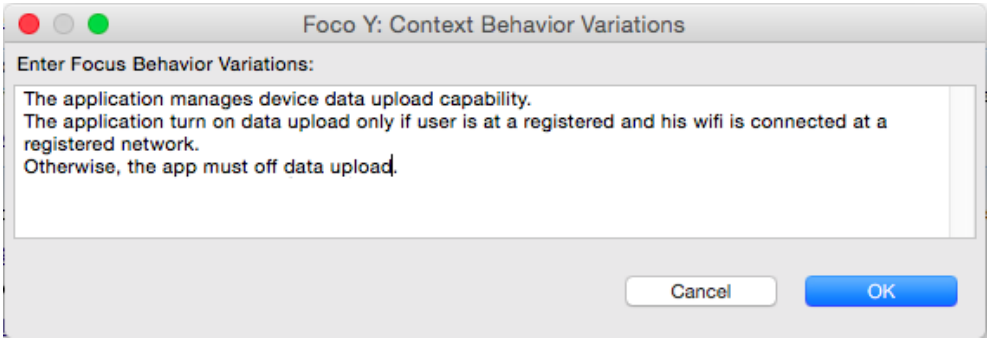

Figura 2.13 Tela do CEManTIKA CASE onde o projetista identifica as variações de comportamento de um foco.

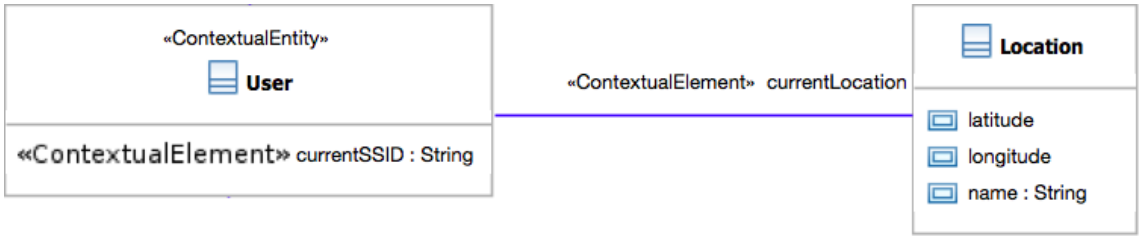

Figura 2.14 Diagrama com entidade contextual e elementos contextuais da aplicação *PersonalUploadManager*

O mesmo executa a tarefa Projetar Adaptação de Contexto (U1). Para tanto, o projetista elabora o grafo contextual para o foco em questão, o qual mostramos na Figura 3.18, onde apresentamos o exemplo executável. Embora esta tarefa esteja definida na atividade Projeto de Uso de Contexto do CDP, ela foi considerada pelo autor do CEManTIKA CASE como parte essencial atividade Especificação de Contexto. A definição desta tarefa fecha o escopo da implementação do CEManTIKA CASE.

O foco do CEManTIKA CASE é a implementação das seguintes atividades do CDP: Especificação de Contexto, e uma porção do Projeto de Uso de Contexto. Como resultado, esta implementação consegue modelar tanto a estrutura, através de um modelo conceitual de contexto, quanto o comportamento do SSC, com uso de grafos contextuais

(Brézillon, Pasquier e Pomerol, 2002), gerando inclusive código-fonte para a aplicação em desenvolvimento. A implementação de outras atividades do CDP (Projeto de Gerenciamento de Contexto e Teste, por exemplo) foi definida como trabalho futuro.

2.3.2 Execução de Testes em Aplicativos Móveis Sensíveis ao Contexto

Existem diversas ferramentas de apoio à execução de testes de contexto em aplicativos móveis. Nesta subseção discutimos algumas abordagens e os pontos que nos levaram à escolha da ferramenta utilizada para executar os casos de teste gerados a partir da proposta defendida por esta dissertação, que é o simulador de contexto Context Simulator (Vieira, Holl e Hassel, 2015).

2.3.2.1 Ferramentas de Entrada Manual de Dados de Contexto

Para a plataforma Android, uma das ferramentas que já vêm no kit de desenvolvimento fornecido pelo Google é o painel de controles estendidos, o qual vem acoplado no emulador. Na Figura 2.15 mostramos uma tela de entrada de coordenadas de GPS. Esta ferramenta suporta diversos sensores, como GPS, redes celulares, bateria, eventos do telefone e sensor de impressão digital (A). O dado do sensor pode ser inserido manualmente pelo testador no momento da execução do teste (B). Para o sensor de GPS, a ferramenta permite que o testador importe os dados para transmissão em série (C). Caso o testador deseje iniciar uma transmissão de dados de GPS em série, o mesmo aciona o botão de reprodução (D).

Um recurso que já vem embutido em dispositivos Android, sejam eles emulados ou físicos, é o terminal de comandos disponibilizado via protocolo *telnet*, conforme mostrado na Figura 2.16. Neste caso, o testador deve se conectar ao dispositivo e a partir de então disparar os comandos com os dados de contexto. Estes comandos podem ser disparados com uma aplicação que se conecte ao dispositivo Android ou via script de teste. A elaboração deste script pode ser custosa, pois os dados de contexto são todos informados pelo testador em formato de texto, além de usar dados em nível de sensor. Na Figura 2.16, após a conexão (A) e autenticação com o dispositivo (B), informamos algumas coordenadas geográficas através do comando *geo fix*, o qual recebe como argumentos a latitude, logitude e altitude (C).

2.3.2.2 Simuladores de Contexto

Para sistematizar a entrada de dados para a execução de testes de contexto em aplicativos móveis, diversos autores propõem simuladores de contexto. Estes simuladores podem ser classificados como interativos e não interativos (Bylund e Espinoza, 2002). Simuladores interativos são aqueles que permitem o uso de ambientes virtuais para a geração de dados de contexto, enquanto que nos simuladores não interativos o teste se resume à entrada de valores para as fontes de contexto. Encontramos exemplos de simuladores interativos em: UbikMobile (Campillo-Sanchez, Serrano e Botía, 2013); CraftContext (Raymundo e Costa, 2012); uso do jogo Quake III Arena em (Bylund e Espinoza, 2002),

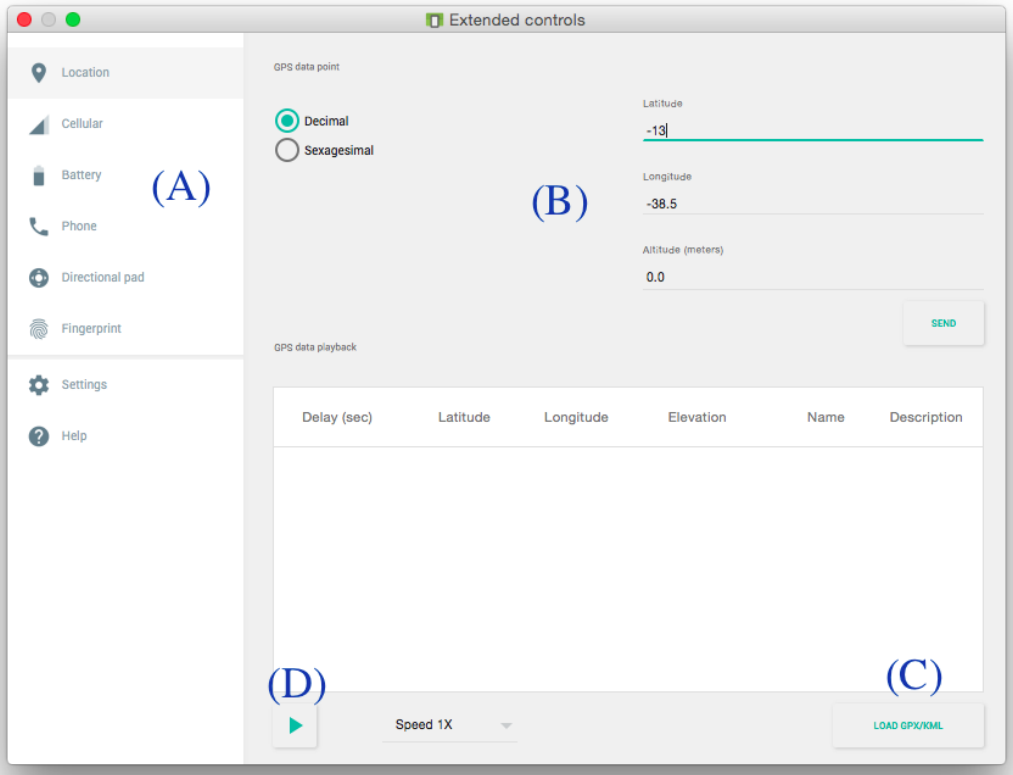

Figura 2.15 Envio de coordenadas de GPS para um emulador Android através do painel de controles estendidos do emulador.

dentre outros. Como exemplos de simuladores não interativos temos: Open Intents Simulator (OpenIntents, 2014); e Samsung Sensor Simulator (Samsung, 2011).

Embora sejam propostas com diferentes características entre si, as mesmas utilizam a representação de dados de contexto em baixo nível de abstração. Visando cobrir esta lacuna, Vieira; Holl; Hassel (2015) propõe um simulador de contexto com suporte a dados de contexto com alto nível de abstração. Este simulador de contexto é denominado Context Simulator. Por usar dados de contexto de alto nível de abstração esta abordagem se beneficia das seguintes vantagens sobre as propostas anteriormente vistas: uso de informações de contexto de melhor entendimento por pessoas e reuso de dados de contexto anteriormente definidos (Vieira, Holl e Hassel, 2015).

2.3.2.3 Context Simulator

O Context Simulator é a ferramenta que utilizamos na avaliação da proposta para a execução dos casos de teste gerados pelo método proposto. Detalhamos a seguir alguns aspectos relativos à implementação, à modelagem de contexto e também da execução dos

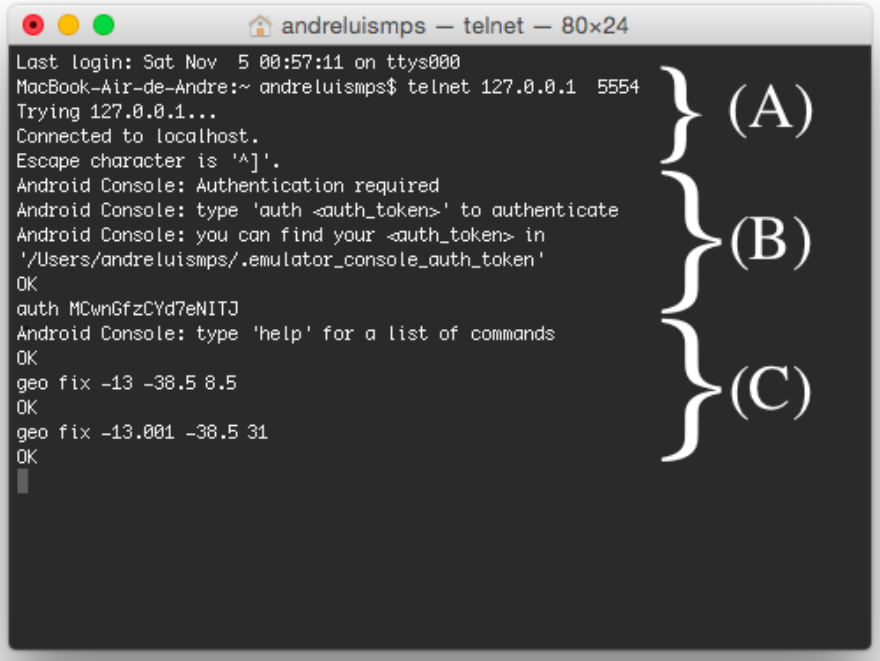

Figura 2.16 Envio de coordenadas de GPS com uso do comando *geo fix* para um dispositivo Android através de uma conexão *telnet*.

cenários de teste. Para ilustrar, utilizamos a aplicação exemplo apresentada na Seção 3.1.

2.3.2.3.1 Aspectos da implementação

O Context Simulator foi implementado na linguagem Java. Esta ferramenta tem duas funcionalidades principais: Modelagem de Contexto, com agregação de contexto em diferentes níveis de abstração e com definição de valores de sensores; e a Execução de Cenários.

2.3.2.3.2 Modelagem de Contexto

Para permitir a modelagem de contexto em alto nível, o desenvolvedor implementou o pacote com.logicalcontextsimulator.gui.panel.contextList, o qual mostramos na Figura 2.17.

Este pacote implementa os níveis de abstração presentes na Figura 2.5. As classes que não constam como representação de contexto na referida figura são *AbstractContext* e *TimeSlot*.

A classe abstrata *AbstractContext* serve para determinar atributos e comportamentos

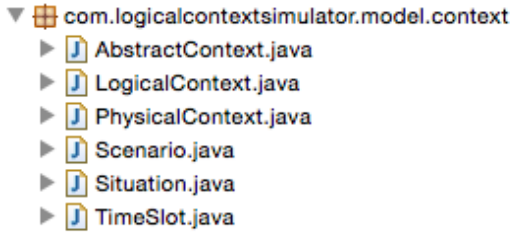

Figura 2.17 Pacote contendo as classes Java com que representam dados de contexto em alto nível de abstração.

comuns às outras formas de representação, como o nome do contexto representado, e a forma de adicionar contextos agregados. Em geral, o contexto de alto nível de abstração agrega uma lista de contexto de mais baixo nível. Ex: O contexto lógico "Em casa" pode agregar um ou mais contextos físicos (sensores) que determinem localização, por exemplo.

A classe *TimeSlot* representa a associação temporal de uma informação contextual com o cenário. É a partir dela que informamos a posição de um dado contexto dentro do cenário.

Para modelar as fontes de contexto, o desenvolvedor criou as classes presentes no pacote com.logicalcontextsimulator.model.context.contextSource. Todas as classes deste pacote, que estão representadas na Figura 2.18, são herdadas da classe *PhysicalContext*. Estas classes são responsáveis por armazenar os valores de contexto de nível físico (sensor) que são informadas pelo testador. Neste pacote temos sensores físicos (como o acelerômetro, GPS, dentre outros) e virtuais (Calendar, HTTPResponse, dentre outros).

O testador pode modelar o contexto para o teste nos seguintes níveis, do mais baixo para o mais alto: Lógico, Situação e Cenário. Para cada tipo de representação o testador tem uma opção diferente. Para modelar contextos lógicos, o testador precisa agregar contextos físicos, os quais são representados pelos sensores. Para modelar os as situações, o testador pode agregar tanto contextos lógicos quanto físicos. Para modelar os cenários, que são os dados de contexto em alto nível de abstração que encadeiam contexto no decorrer do tempo, o testador precisa escolher o intervalo de tempo (*TimeSlot*) antes de agregar os dados de contexto. Temos na Figura 2.19 um exemplo de cenário completo com os dados de contexto que foram modelados na aplicação.

2.3.2.3.3 Execução de Cenário

Para executar os casos de teste modelados com o nível de abstração que explicamos acima, o testador deve adicionar o caso de teste na linha de tempo (*Timeline*). Para realizar o envio dos dados presentes na linha do tempo do simulador para o emulador Android, o Context Simulator primeiro decompõe os dados de alto nível de abstração em dados no nível de sensores. A partir de então o simulador faz o envio destes dados decompostos através de conexão *telnet* ou TCP/IP com o emulador Android. A Figura 2.20 exibe um exemplo de uso do simulador com a aplicação exemplo sob teste.

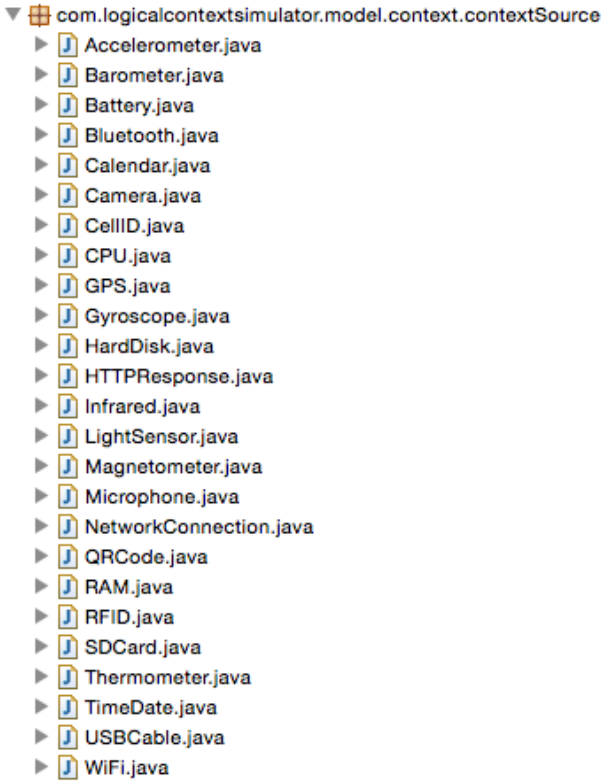

Figura 2.18 Pacote contendo as classes Java com que representam as fontes de contexto utilizadas pelo Context Simulator.

2.4 GERAÇÃO DE CASOS DE TESTE PARA APLICATIVOS MÓVEIS

Diversos trabalhos abordam a área de geração de casos de teste para aplicativos móveis. Alguns tratam exclusivamente de eventos de interface gráfica do usuário (*Graphical User Interface* - GUI), como (Amalfitano et al., 2012) e (Jensen, Prasad e Møller, 2013). Algumas utilizam a simulação de um ambiente virtual com agentes espalhados no espaço, como o Siafu (Martin e Nurmi, 2006) e CraftContext (Raymundo e Costa, 2012). Estas abordagens não se encaixam no escopo do trabalho, pois nas primeiras, embora testem aplicativos móveis, elas não tratam de contexto e suas implicações nos casos de teste gerados; e as outras não geram casos de teste a partir de representação comportamental de contexto e/ou de padrões de eventos de contexto.

Durante as pesquisas vimos que a representação de dados de contexto pode derivar em casos de teste a partir de uso de técnicas de Teste Baseado em Modelos (MBT na sigla em inglês), como em (Garzon e Hritsevskyy, 2012), (Griebe e Gruhn, 2014) e (Yu et al., 2014); ou com uso de padrões de eventos de contexto, como em (Amalfitano et al., 2013) e (Yu e Takada, 2015). A seguir detalhamos os trabalhos que usam estas abordagens, e ao final realizamos uma análise comparativa entre os trabalhos.

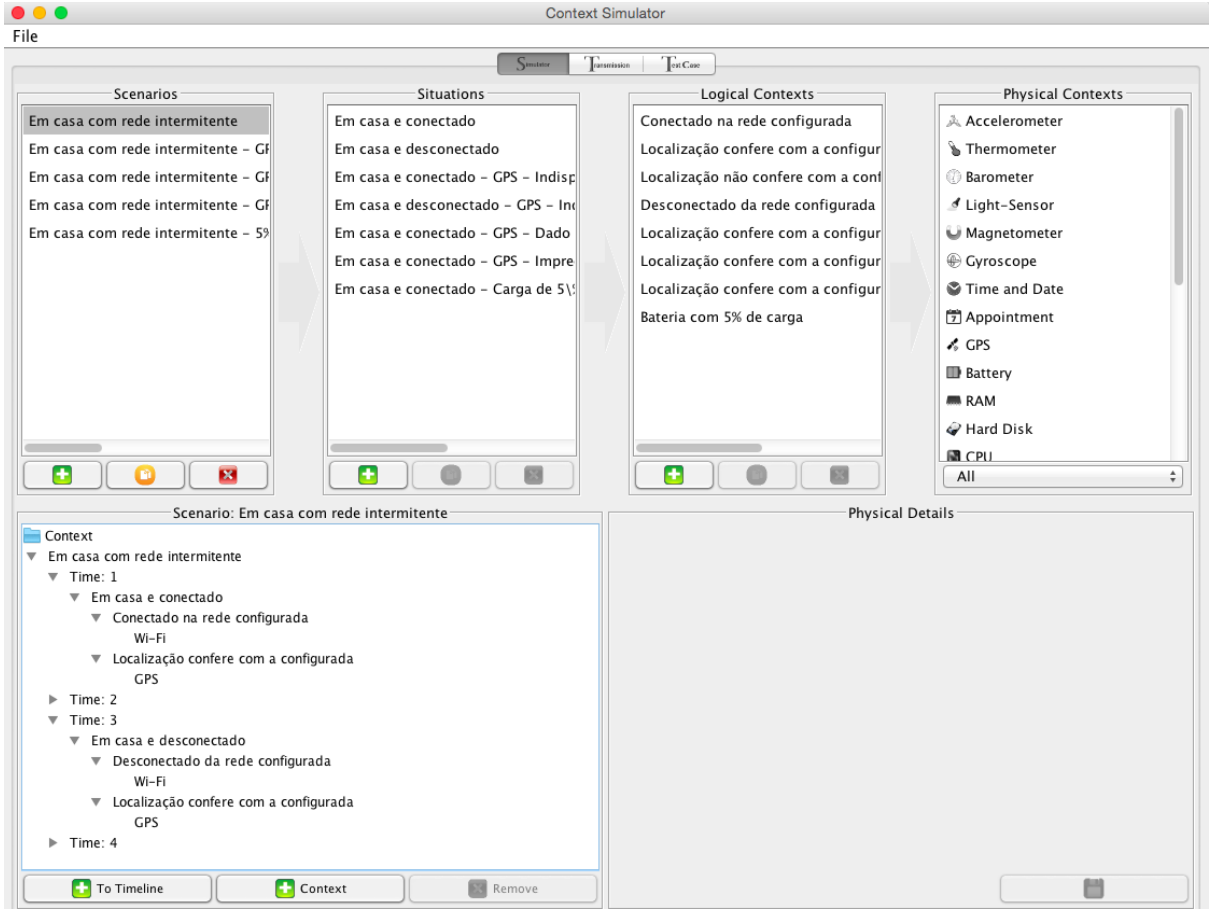

Figura 2.19 Exemplo de modelagem de um cenário. No cenário em destaque (Em casa com rede intermitente), cada intervalo de tempo (4 no total) contém uma situação, que pode ser "Em casa conectado" ou "Em casa e desconectado". Cada situação pode ter contextos lógicos, e estes últimos são agregados com contextos físicos (sensores de GPS e Wi-fi).

2.4.1 Uso de teste baseado em modelos

Nas abordagens detalhadas nesta seção os autores geram casos de teste para aplicativos móveis com o apoio de modelos de comportamento contextual. Estes modelos podem ser dos seguintes tipos: do usuário do aplicativo móvel, ou seja, o comportamento recorrente do usuário é alvo de modelagem; da aplicação sob teste, a qual possui seu comportamento representado em um modelo; do ambiente no qual a aplicação se insere, de forma que as variáveis de contexto presentes no ambiente são mapeadas; e/ou do *middleware* dos sensores que estão inseridos no ambiente, onde os sensores presentes no ambiente possuem suas estruturas de dados representadas nos modelos.

2.4.1.1 Simulação de comportamentos de usuário recorrentes

Em (Garzon e Hritsevskyy, 2012) os autores propõem uma ferramenta que gera dados

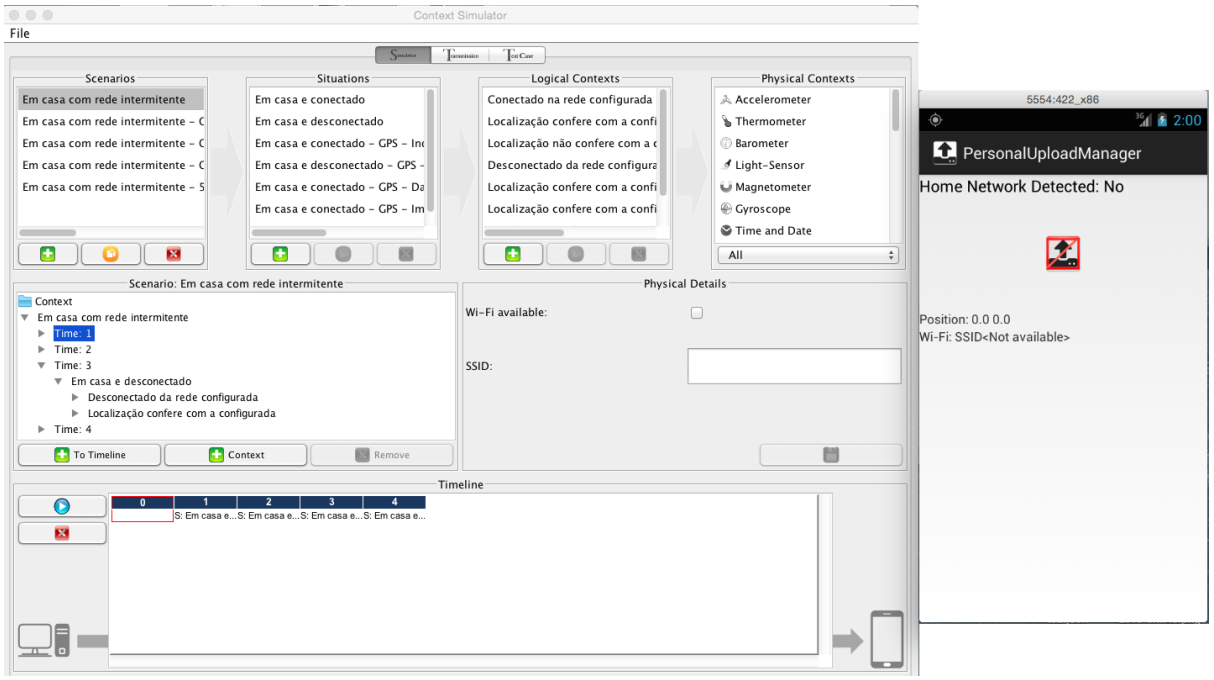

Figura 2.20 Execução de um cenário na aplicação exemplo. À esquerda temos o Context Simulator, e à direita temos a aplicação *PersonalUploadManager* sob teste. Os dados de contexto que estão destacados na linha do tempo são os enviados para o simulador de contexto no momento.

sequenciais de contexto para dispositivos móveis. Sua estratégia visa gerar dados de contexto simulando o comportamento do usuário em um dado cenário. Para tanto, a abordagem em questão utiliza o formalismo de especificação discreta de eventos de sistema (DEVS na sigla em inglês) combinado com simulações quantitativas para gerar sequências de eventos semelhantes utilizando uma especificação de cenário baseado em modelo.

Para gerar os casos de teste, o método proposto exige a execução dos seguintes passos:

- Elaboração do modelo de comportamento do usuário com o formalismo DEVS;
- Elaboração de um produtor de ruídos (passo opcional);
- Definição de um intervalo de tempo para que os eventos sejam gerados (Ex: manhãs de Sábados e Domingos em dois meses consecutivos);
- Combinação entre os modelos de comportamento DEVS, o de ruído (informado opcionalmente) e o intervalo de tempo definido.

Ao combinar os artefatos elaborados com o intervalo de tempo, o método gera as sequências de eventos, e então o testador pode executá-las na aplicação desenvolvida.

Um exemplo de cenário típico de uso do método é o mostrado na Figura 2.21. Este cenário representa as variações de contexto percebidas por uma aplicação automotiva que alerta o usuário para abastecer o carro em um posto dentro de uma cidade, caso o tanque de combustível esteja em um nível crítico. Neste exemplo, dois elementos contextuais são manipulados: o nível de combustível e a posição do carro. O grafo exibido na Figura 2.21 representa o percurso de um motorista no qual ele liga o motor do carro (*StartingEngine*), inicia o deslocamento, se encontra antes de uma cidade (*Position before city*), passa por

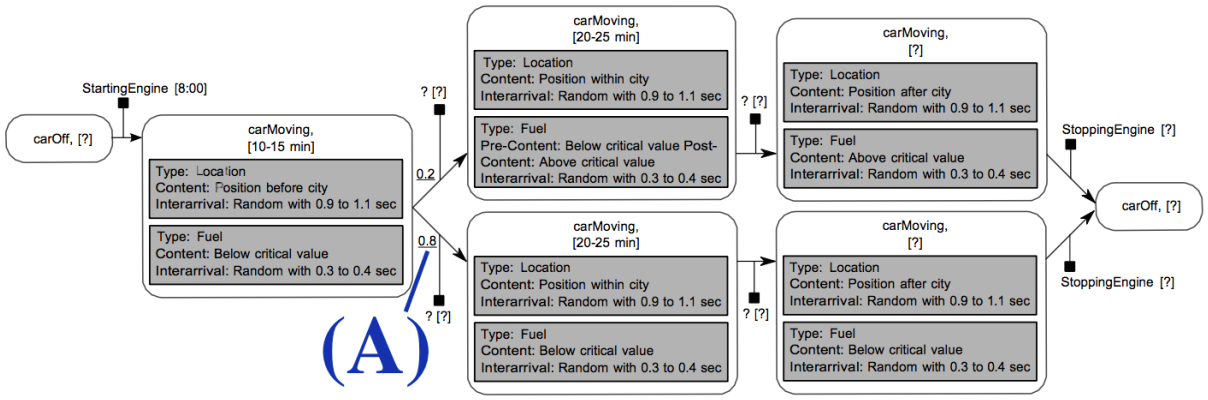

Figura 2.21 Cenário de uso típico de uma aplicação cujo foco é a emissão de alertas para motoristas em um percurso (Garzon e Hritsevskyy, 2012)

dentro da cidade (*Position within city*), depois passa da cidade (*Position After City*) e ao fim do percurso desliga o motor (*StoppingEngine*) . Durante o percurso, existe 80% de chances do nível de combustível estar abaixo do crítico (*Below Critical Value*). Esta probabilidade é representada na bifurcação (A).

A Figura 2.22 mostra a execução do protótipo implementado. Antes de ser utilizado pela ferramenta, o modelo DEVS é convertido em um documento XML que é utilizado como entrada para o gerador de eventos (A). Opcionalmente o testador pode incluir eventos de ruído de sensores, que estão armazenados em outro documento XML (B). Outra entrada informada pelo testador são os intervalos de tempo nos quais os eventos são definidos (C). O gerador de eventos lê o arquivo XML com as informações do modelo e gera eventos de acordo com as entradas fornecidas pelo testador. Depois de gerar as sequências de eventos de contexto no intervalo de tempo informado, a ferramenta permite a simulação destes eventos na aplicação sob teste, mostrando inclusive o progresso desta atividade (D).

2.4.1.2 Customização para o calabash-android

Em (Griebe e Gruhn, 2014) os autores propõem uma abordagem para gerar casos de teste a partir de uma customização da ferramenta calabash-android[5], a qual originalmente não suporta a geração de valores de sensores. Para gerar os casos de teste com suporte a sensores, a abordagem utiliza o modelo de contexto da aplicação.

A geração de casos de teste ocorre da seguinte maneira nesta abordagem:

- O projetista elabora um diagrama de atividades UML enriquecido com estereótipos próprios para a descrição de elementos de contexto dentro da aplicação;
- Realiza-se uma transformação de modelos para traduzir do modelo UML elaborado para uma rede de Petri, de forma que seus elementos estruturais originais (laços e paralelismos) sejam analisados;
- Gera-se então um modelo independente de plataforma contendo dados de contexto

[5]http://calaba.sh/

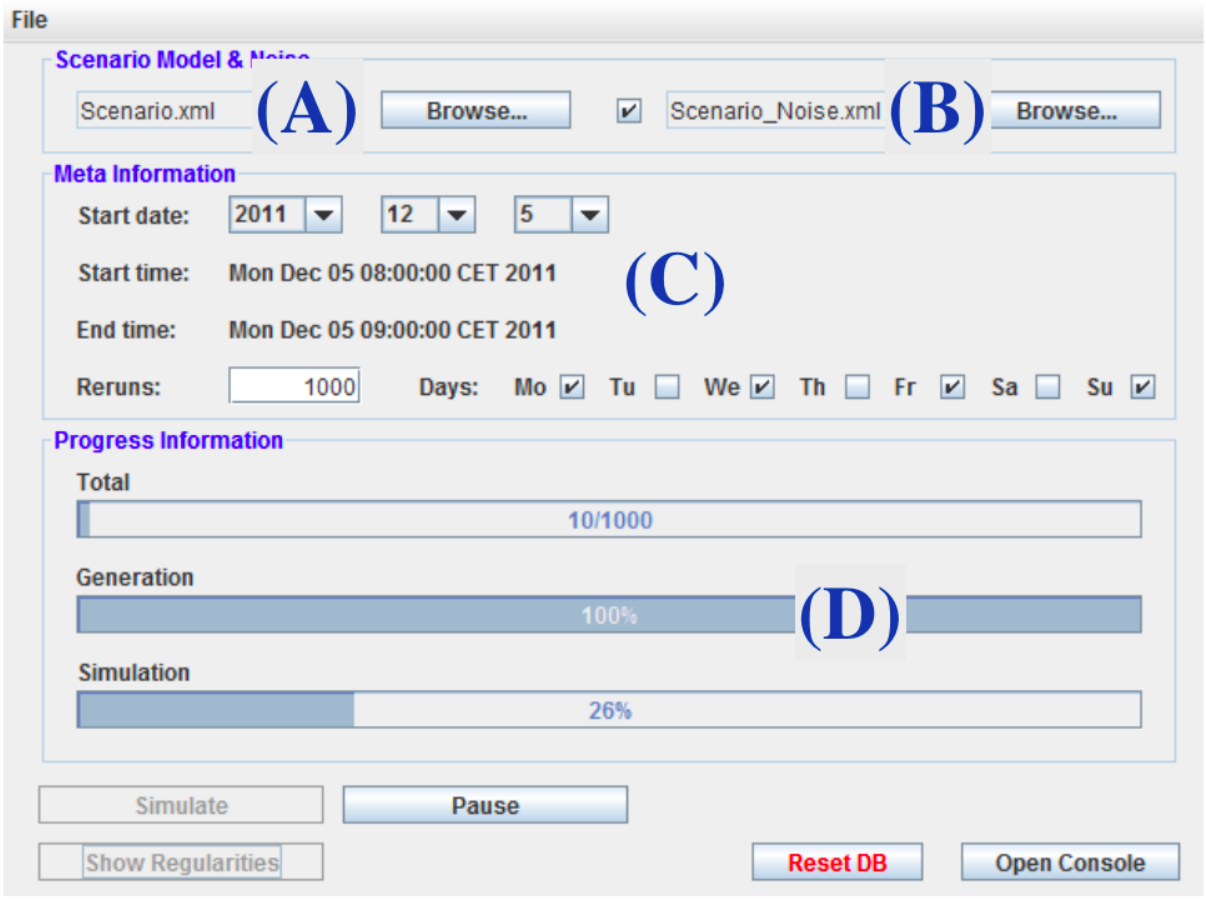

Figura 2.22 Protótipo do gerador de eventos de contexto que utiliza o formalismo DEVS. O protótipo implementado contempla também um simulador de contexto (Garzon e Hritsevskyy, 2012).

relevantes para execução dos testes;

- Ao fim gera-se os casos de teste específicos para plataforma (calabash-android/ios, JUnit ou Robotium) para execução na respectiva plataforma.

O código mostrado na Listagem 2.2 exibe um caso de teste gerado para a plataforma de testes calabash-android. Neste cenário a ferramenta estabelece a pré-condição de que a aplicação está em execução (linha 1). Na linha 2, a ferramenta de conferência da interface, própria do calabash, aguarda a exibição de um diálogo na interface gráfica. Depois, na linha 3, a ferramenta aguarda a exibição do botão cujo identificador é *button_gps*. Na linha 4, a ferramenta pressiona o botão com o referido identificador. Na linha 6 a aplicação recebe como entrada uma coordenada de GPS defeituosa. Nas linhas de 7 a 9, a ferramenta exibe o resultado, que é a exibição de um diálogo com uma mensagem que transmite ao usuário a informação de que localização não pôde ser determinada.

O método de geração de casos de teste nesta abordagem gera, para os casos de teste derivados do modelo, alguns defeitos de contexto. No entanto, a proposta apresentada não estabelece de forma clara como são aplicados os padrões de defeitos para injetar nos

casos de teste.

Listagem 2.2 Caso de teste gerado para a ferramenta calabash-android (Griebe e Gruhn, 2014)

```
1 Scenario: testcase03
2 Given my app is running
3 Then I wait for the view with id dialog_locationmode to appear
4 Then I wait for the view with id button_gps to appear
5 Then I press view with id button_gps
6 Then the location is 51.461 , 7.016 accuracy 400.00 age 350000
7 Then I wait for the view with id dialog_warning to appear
8 Then I wait for the view with id button_commit_warning to appear
9 Then I see the text "Sorry your location could not be determined ."
```

2.4.1.3 Uso de Modelo de Sistema de Reação com Bígrafo

Em (Yu et al., 2014), os autores propõem uma abordagem de geração de casos de teste que utiliza dois modelos como entrada: um Sistema de Reação com Bígrafo (Bigraphical Reaction System - BRS em inglês) (Milner, 2006), para modelar o ambiente em que a aplicação está inserida; e uma máquina de estados finitos estendida (EFSM na sigla em inglês), para modelar cada sensor inserido no ambiente da aplicação sob teste. A composição destas EFSMs forma um modelo para o *middleware*.

Para gerar os casos de teste, a abordagem exige a execução dos seguintes passos:

- Modelar, com uso de BRS, o ambiente no qual a aplicação estará inserida;
- Modelar EFSMs para os sensores que interagem com o *middleware*;
- Sincronizar as EFSMs com o *middleware*;
- Modelar a interação entre o *middleware* e o ambiente. O resultado é um produto cartesiano que tende a possuir muitos estados.

A partir do momento em que se obtém o modelo de interação entre a representação do ambiente e a do *middleware* é possível gerar os casos de teste. Isto é feito ao atravessar o modelo de interação de um ponto ao outro. Ao fim, é feita uma redução no número de casos de teste gerados mantendo a capacidade de detecção de faltas. Para atingir este objetivo, utiliza-se uma estratégia com uso de padrão de fluxos em bígrafos. Esta estratégia é similar à estratégia de redução baseada em fluxo de dados, e aplica regras de reações na forma de subgrafos em vez de variáveis, como ocorre nas abordagens baseadas em fluxos de dados.

2.4.2 Uso de Padrões de Eventos de Contexto

Nesta seção detalhamos os trabalhos onde os casos de teste para aplicativos móveis são gerados com técnicas que usam padrões de eventos de contexto. Um exemplo de padrão de evento de contexto é a perda e posterior recuperação do sinal de GPS durante uma caminhada (Amalfitano et al., 2013). Nestas abordagens, o comportamento contextual da aplicação é transformado (manualmente ou de forma automatizada) em uma representação de contexto que é combinada com padrões de eventos de contexto. Casos de

teste base também podem ser combinados com os padrões de eventos, a depender da abordagem. A partir de então os casos de teste são gerados.

2.4.2.1 Geração com Manipulador de Contexto

Em (Wang, Elbaum e Rosenblum, 2007) os autores apresentam uma proposta que gera casos de teste ao identificar pontos de mudança de contexto e depois, de forma sistemática, alimenta a aplicação com dados de contexto alterados com o objetivo de expor falhas na aplicação.

De acordo com os autores, a abordagem apresentada consiste na aplicação integrada de técnicas de análise, a qual vai identificar e controlar os cenários contextuais a serem explorados nos casos de teste. Para gerar os casos de teste, a proposta define os seguintes componentes:

- Identificador de pontos sensíveis ao contexto no programa (PSCP);
- Gerador de Controlador de Contexto;
- Instrumentador de Programa;
- Manipulador de Contexto.

A geração de casos de teste ocorre da seguinte maneira:

1. O identificador de PSCP (que são os pontos onde ocorre a adaptação ao contexto) recebe como entrada a aplicação e as assinaturas dos métodos de manipulação de contexto do *middleware* de contexto e retorna um grafo de controle de fluxo para cada tipo de manipulador de contexto;
2. O Gerador de Controlador de Contexto recebe como entrada o grafo gerado no passo anterior e fornece, após atravessar o mesmo, um conjunto de controladores que serão usados para o controle do teste em outra fase;
3. O Instrumentador de Programa injeta linhas de código antes e depois dos PSCPs identificados. Estas linhas injetadas são responsáveis por atrasar a execução PSCPs dentro do programa;
4. O Manipulador de Contexto então gerencia a ordem de execução dos PSCPs delimitados pelo Instrumentador. Neste caso, o manipulador de contexto determina um intervalo de tempo após o qual o PSCP selecionado deve ser executado.

Como vemos, esta abordagem manipula o momento em que o dado de contexto afeta a aplicação sob teste.

2.4.2.2 Geração com Três Técnicas de Injeção de Padrões de Eventos de Contexto

Amalfitano et al. (2013) elaboram casos de teste levando em conta tanto eventos de interface gráfica de usuário quanto eventos de contexto. Neste trabalho define-se um conjunto de padrões de evento de contexto para serem injetados nos casos de teste. Seguem exemplos de padrões de eventos que podem ser injetados: Instabilidade de rede, recebimento de chamada telefônica, perda e recuperação de sinal de GPS, dentre outros.

Para gerar os casos de teste com os padrões de eventos a abordagem em questão

apresenta três técnicas, que são: uso de padrões de eventos para geração manual de casos de teste; uso de teste mutante e a exploração sistemática do comportamento do aplicativo implementado.

Na *técnica manual*, o testador utiliza padrões de eventos de contexto para gerar manualmente os casos de teste. Um caso de teste pode conter uma ou mais instâncias de padrão de evento. Na técnica com uso de *teste mutante*, os padrões de eventos são utilizados para alterar casos de teste já existentes. Para isto, aplica-se técnicas de mutação nos casos de teste que já existem. Na técnica de *exploração sistemática*, os padrões de eventos são usados a partir da análise dinâmica do aplicativo móvel. Nesta técnica, a aplicação é iniciada em um dado contexto, então a técnica detecta, de forma iterativa, o conjunto atual de eventos de sensores do aplicativo, então planeja um modo como disparar a sequência de eventos de contexto e depois executa este conjunto. Em todas as técnicas, a definição dos padrões de eventos é feita de forma manual.

2.4.2.3 Geração com eventos externos

Em (Yu e Takada, 2015) os autores propõem uma abordagem para gerar casos de teste para aplicativos móveis na plataforma Android. Neste caso, a abordagem considera como eventos externos tanto aqueles que são manipulados (denominados explícitos) quanto os que não são manipulados (chamados implícitos) pela aplicação. Exemplos de eventos externos são: ligações telefônicas, coordenadas de GPS e entradas de valores de outros sensores físicos.

Para alcançar o objetivo, a proposta apresentada em (Yu e Takada, 2015) define cinco componentes, conforme Figura 2.23: 1) Analisador estático de código-fonte; 2) Repositório de Eventos, 3) Gerador de Padrões de Evento, 4) Repositório de Padrões de Eventos, e 5) Gerador de Casos de Teste.

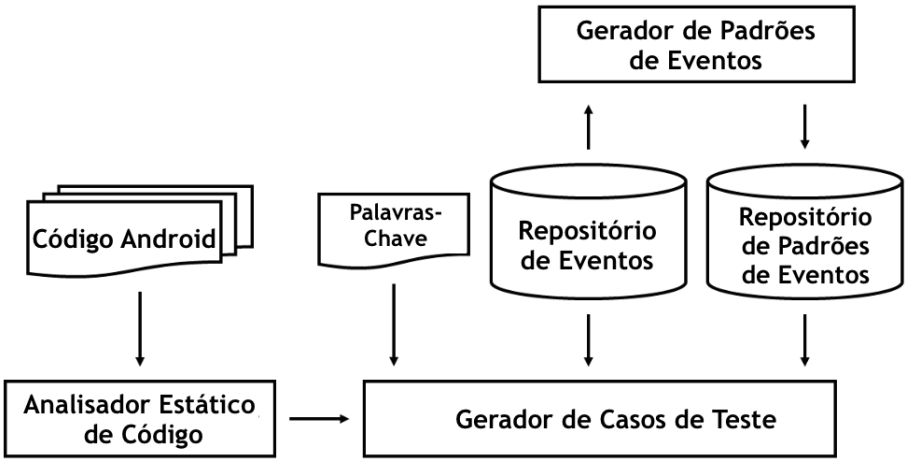

Figura 2.23 Arquitetura de uma solução que gera casos de teste a partir de código-fonte de aplicações Android, com uso de padrões de eventos de contexto (Yu e Takada, 2015)

Cada componente presente nesta arquitetura possui um papel na geração dos casos

de teste. O *Analisador Estático de Código* serve para detectar os eventos de contexto que podem ser percebidos e em que estado a aplicação reage aos eventos. O *Repositório de Eventos* armazena os diferentes eventos e suas respectivas categorias. Um exemplo do que pode ser armazenado nesta base: Categoria "GPS", Evento "GPS ligado" e "GPS desconectado". O *Gerador de Padrões de Evento* combina os padrões de eventos entre si e os armazenam no *Repositório de Padrões de Eventos*. Um exemplo de padrão de evento dado a seguir: 1) GPS ligado, 2) GPS desconectado e 3) GPS recuperado.

Para gerar os casos de teste o usuário da abordagem pode escolher, após a verificação do código, as categorias de eventos que serão testados. Com a categoria selecionada, e uma categoria pode conter diversos padrões de eventos associados (explícitos e implícitos), todos os padrões de eventos da categoria selecionada são incluídos nos casos de teste, mesmo que o código da aplicação não manipule tais eventos (neste caso, implícitos).

2.5 DISCUSSÃO DOS TRABALHOS CORRELATOS

Cada um dos trabalhos relacionados apresentados na Seção 2.4 visa a resolução do mesmo problema do presente trabalho: a geração de casos de teste para aplicativos móveis. No entanto, as abordagens citadas divergem em alguns pontos.

Estas divergências são melhor visualizadas na Tabela 2.3, a qual apresenta os critérios utilizados para avaliar os trabalhos correlatos. Seis abordagens foram consideradas similares à proposta neste trabalho (presente na coluna à direita da referida tabela). Os critérios de comparação entre estes trabalhos são: (i) *Uso de modelo de contexto*, que define se a abordagem em questão utiliza modelos de contexto para ajudar o testador a gerar os casos de teste; (ii) *Definição de padrões de eventos/defeitos de contexto*, que indica se a abordagem especifica padrões de eventos ou de defeitos de contexto para adicionar aos casos de teste gerados; (iii) *Reuso de dados de contexto entre casos de teste*, que implica no armazenamento e disponibilização de dados de contexto para diversos testes numa aplicação ou entre aplicações; (iv) *Casos de teste incluem dados de GUI*, que significa que a abordagem gera dados de interação do usuário com a interface gráfica do aplicativo móvel sob teste; (v) *Uso de redutor de casos de teste*, que informa se a abordagem analisada utiliza alguma estratégia para redução de casos de teste; e (vi) *Uso de contexto em alto nível*, que indica se a abordagem utiliza dados de contexto em alto nível de abstração para representação dos casos de teste.

Nas abordagens que utilizam modelo de contexto para especificar comportamento, a *representação gráfica* é a preferência dos autores selecionados. Diversos tipos de modelo representam o comportamento nestes trabalhos, dentre eles o uso de grafos (Garzon e Hritsevskyy, 2012), de UML (Griebe e Gruhn, 2014) ou BRS e EFSM (Yu et al., 2014). A vantagem de se utilizar modelos está na possibilidade de antecipação da elaboração dos casos de teste os quais podem ser gerados antes da codificação do aplicativo móvel. Ao fornecer um modelo para o testador gerar os casos de teste, diminui-se a dependência da experiência e visão pessoal do testador sobre a aplicação sob teste. A proposta detalhada no Capítulo 3 utiliza modelos gráficos para a representação de contexto. Além disso, o método proposto também lida com dados de contexto em alto nível de abstração, pois informações neste formato são mais fáceis de ler e interpretar por pessoas do que dados

Tabela 2.3 Análise comparativa dos trabalhos correlatos.

	(Wang, Elbaum e Rosenblum, 2007)	(Garzon e Hritsevskyy, 2012)	(Amalfitano et al., 2013)	(Griebe e Gruhn, 2014)	(Yu et al., 2014)	(Yu e Takada, 2015)	CEManTIKA Test Creator
Uso de modelo de contexto	✗	✓	✗	✓	✓	✗	✓
Uso de contexto em alto nível	✗	✗	✗	✗	✗	✗	✓
Definição de padrões de eventos/defeitos de contexto	✓	✓	✓	✗	✗	✓	✓
Reuso de dados contexto entre casos de teste	✗	✓	✗	✗	✗	✓	✓
Casos de teste incluem dados de GUI	✗	✗	✓	✓	✗	✗	✗
Uso de redutor de casos de teste	✗	✗	✗	✗	✓	✗	✓

de sensores brutos. As outras abordagens tratam os dados de contexto apenas em nível de sensor, enquanto que o CEManTIKA Test Creator almeja a manipulação sistemática de dados de contexto em alto nível de abstração. Com esta sistematização, a tarefa de criar dados de teste de contexto pode se tornar menos cansativa e propensa a erros.

As propostas abordadas em (Wang, Elbaum e Rosenblum, 2007), (Garzon e Hritsevskyy, 2012), (Amalfitano et al., 2013) e (Yu e Takada, 2015) utilizam algum tipo de *definição de padrão de eventos/defeitos de contexto*. Em (Wang, Elbaum e Rosenblum, 2007), ocorre a injeção de código-fonte para a geração do casos de teste, enquanto que nas outras abordagens isto não ocorre, pois em vez de injetar padrões de defeitos no código-fonte, as outras abordagens fazem isto nos casos de teste gerados. Ainda em relação à forma de representação de padrões de eventos, temos que em (Garzon e Hritsevskyy, 2012) um padrão de eventos de contexto é definido para cada cenário da aplicação sob teste. Já em (Amalfitano et al., 2013) e (Yu e Takada, 2015) os padrões de eventos são pré-definidos e aplicados nos casos de teste. Uma diferença entre (Amalfitano et al., 2013) e (Yu e Takada, 2015) é que na segunda abordagem casos de teste são gerados considerando eventos de contexto não manipulados pela aplicação. Isto ocorre porque nesta abordagem os padrões de eventos de contexto são armazenados num repositório, e toda vez que o testador solicita a geração de casos de teste e seleciona uma categoria de padrão de evento, todos os padrões daquela categoria derivam casos de teste, mesmo que a aplicação não manipule diretamente alguns destes padrões. A existência deste repositório caracteriza o reuso de dados de contexto entre casos de teste. Já em (Garzon e Hritsevskyy, 2012), este reuso se resume à definição de ruídos de sensores a serem aplicados aos cenários, não ficando isto definido como parte principal da abordagem. Como a estratégia apresentada em (Wang, Elbaum e Rosenblum, 2007) é aplicada direto no código-fonte da aplicação, o

reuso se resume à replicação de padrões em formato de código-fonte a serem injetados na aplicação. Em termos gerais, a pré-definição dos padrões de eventos também diminui a necessidade do testador de ter ciência dos principais eventos de contexto que podem afetar a aplicação sob teste. A existência do repositório de padrões de eventos ajuda ainda mais o testador neste sentido. Na proposta apresentada no Capítulo 3 apresentamos um repositório de padrões de defeitos de contexto a serem aplicados nos dados de contexto dos cenários.

As abordagens de geração de casos de teste que consideram a interação em GUI são (Griebe e Gruhn, 2014) e (Amalfitano et al., 2013). Na abordagem apresentada em (Griebe e Gruhn, 2014), utiliza-se uma ferramenta já existente para realizar este tipo de teste, a qual é estendida para contemplar dados de contexto. Já em (Amalfitano et al., 2013) os casos de teste são gerados em código Java para JUnit, o qual consegue alcançar qualquer ponto do código da aplicação, incluindo aí a manipulação de eventos de interface e injeção de dados de sensores na aplicação. O teste com uso de eventos de interface gráfica é interessante para a geração de casos de teste que podem ser executados automaticamente, de forma que a conferência do testador serve apenas para visualizar os resultados finais da execução do teste. A preocupação na redução no número de casos de teste gerados ocorre apenas em (Yu et al., 2014), pois os casos de teste são gerados ao obter um produto cartesiano entre os modelos de ambiente e do *middleware* de contexto. Esta é uma preocupação que decorre da inviabilidade da execução de numerosos casos de teste, sendo que estes casos de teste podem ser redundantes, ou seja, um conjunto menor de casos de teste pode descobrir o mesmo número de defeitos que o conjunto de todos os casos de teste gerados por uma dada abordagem. Temos, na proposta apresentada no Capítulo 3, uma estratégia para reduzir o número de casos de teste decorrentes da combinação dos cenários com os padrões de defeitos de fontes de contexto.

2.6 RESUMO DO CAPÍTULO

Neste capítulo apresentamos a alguns conceitos relacionados a esta pesquisa, tendo como ponto de partida conceitos elementares sobre testes, contexto e aplicativos móveis sensíveis ao contexto.

Depois apresentamos os critérios de escolha para a ferramenta de apoio ao desenvolvimento escolhida para a modelagem e representação de contexto utilizada na proposta. Também mostramos algumas ferramentas para execução de testes em aplicativos móveis.

Ao final deste capítulo discutimos seis trabalhos correlatos, os quais foram divididos em duas categorias: os que usam teste baseado em modelos (três); e os que usam padrões de evento de contexto (três). Estes trabalhos foram avaliados e os critérios que consideramos como principais destacamos na Tabela 2.3.

No próximo capítulo apresentamos o desenvolvimento da solução proposta neste trabalho.

Capítulo 3

CEMANTIKA TEST CREATOR

Neste capítulo apresentamos o CEManTIKA Test Creator (CTC), um método que visa dar suporte aos testadores de aplicativos móveis sensíveis ao contexto na geração de casos de teste para avaliar o funcionamento da adaptação ao contexto nestes aplicativos. Para tanto, o método proposto utiliza como entrada os modelos da aplicação sensível ao contexto, dos quais extraímos dados de contexto de alto nível de abstração, e depois combinamos estes dados de contexto com padrões de defeitos de fontes de contexto.

O método é composto por cinco componentes, os quais mostramos na Figura 3.2: 1) Analisador de Modelo; 2) Base de Conhecimento de Contexto para Testes (*Context Knowledge Test Base* - CKTB); 3) Repositório de Padrões de Defeitos de Contexto; 4) Combinador de Casos de Teste; e 5) Redutor de Casos de Teste. Para estruturar a atividade de teste dividimos esta em duas etapas: geração dos casos de teste e execução dos casos de teste. O foco desta proposta é a geração dos casos de teste, portanto a outra etapa não é alvo de detalhamento. Para proceder com a geração dos casos de teste, dividimos esta etapa em seis tarefas: 1) Acessar CKTB; 2) Identificar Contextos Lógicos; 3) Identificar Situações; 4) Elaborar Cenários Base; 5) Combinar Cenários Base com Padrões de Defeitos; 6) Exportar Casos de Teste para Execução.

Para ilustrar a proposta, primeiro apresentamos uma aplicação exemplo. Depois, detalhamos o método proposto. A seguir apresentamos a implementação de um protótipo incorporado à ferramenta CEManTIKA CASE (Patricio, 2010) cujo objetivo é possibilitar a avaliação do método. Para executar os casos de teste gerados, utilizamos o Context Simulator (Vieira, Holl e Hassel, 2015), o qual precisou ser adaptado para permitir: a importação dos casos de teste gerados; e a exibição do comportamento esperado pela aplicação em uma dada situação de contexto.

3.1 APLICAÇÃO EXEMPLO

Para ajudar no entendimento dos conceitos abordados neste capítulo apresentamos o exemplo de um aplicativo móvel que utiliza sensores de um dispositivo móvel da plataforma Android.

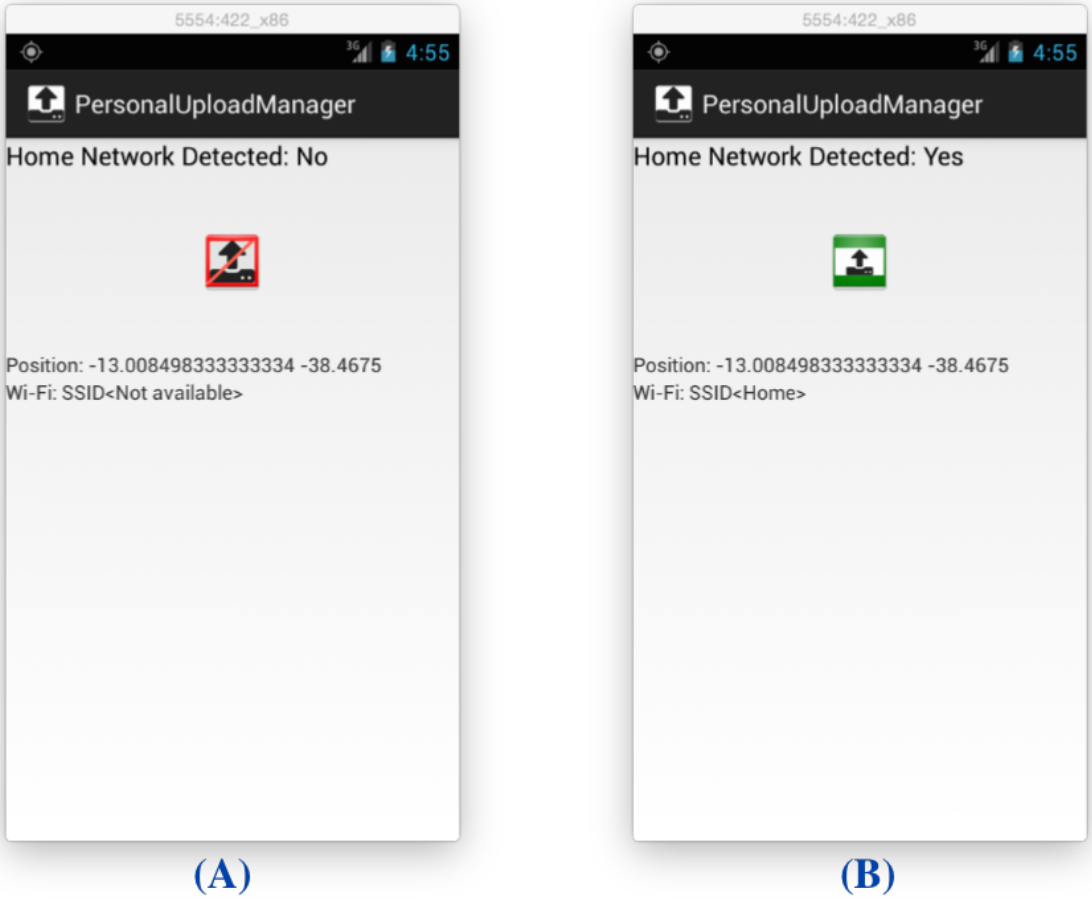

Figura 3.1 Aplicação *PersonalUploadManager* gerenciando o envio de dados do dispositivo. Na captura de tela à esquerda (A), a aplicação não reconhece o contexto de uma conexão configurada pelo usuário e bloqueia o envio de dados. Na tela à direita (B), o aplicativo reconhece o contexto de uma conexão de dados configurada pelo usuário e libera o envio de dados pelo dispositivo.

A aplicação é a *PersonalUploadManager*. Esta aplicação utiliza dados de conectividade e localização do usuário para liberar ou bloquear o envio (*upload*) de dados pelo dispositivo. Para desenvolver a aplicação, o autor desta dissertação gastou 8 horas. Estes dois são os elementos contextuais manipulados para a tarefa de gerenciamento da função de envio de dados da aplicação. O diagrama UML presente na Figura 2.14 mostra a relação entre os elementos contextuais manipulados e a entidade contextual usuário nesta aplicação. Para identificar a conectividade do dispositivo, a aplicação verifica o identificador (*SSID*) da rede Wi-fi conectada. Para obter a localização do usuário, a aplicação utiliza coordenadas obtidas pelo sensor de GPS do dispositivo. Se o usuário registrar na configuração da aplicação que a mesma deve permitir envio de dados apenas de sua casa, o usuário deve informar a coordenada de GPS de sua casa (por exemplo, latitude igual a -13.0093 e longitude igual a -38.4683) e o nome da rede Wi-fi utilizada

(por exemplo, “Home”). Pela imprecisão na coordenada de GPS obtida pelo dispositivo, a aplicação deve considerar um valor de tolerância para definir um raio de abrangência para a coordenada do usuário.

A Figura 3.1 mostra os dois comportamentos possíveis da aplicação de acordo com o contexto do usuário. Na primeira situação a aplicação bloqueia o acesso à função de envio de dados, pois embora o usuário esteja numa localidade que foi identificada como sendo a casa dele, por algum motivo a rede Wi-fi não estava disponível. Na segunda situação a aplicação, na mesma coordenada consegue obter os dados da rede Wi-fi da casa do usuário e libera o envio de dados pelo dispositivo.

A seguir apresentamos a arquitetura proposta no método de geração de casos de teste.

3.2 ARQUITETURA DA SOLUÇÃO

Nesta seção detalhamos os componentes da solução proposta, os quais encontram-se ilustrados na Figura 3.2. Cada componente possui uma responsabilidade específica na etapa de geração dos casos de teste.

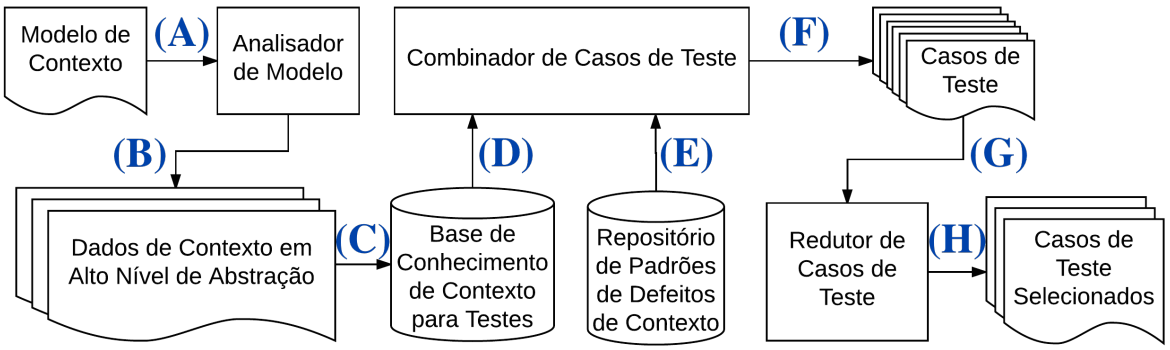

Figura 3.2 Componentes da arquitetura proposta

Nesta arquitetura, o modelo de contexto serve como entrada (A) para o Analisador de Modelo (detalhado na Seção 3.2.1), o qual obtém, através da execução de um algoritmo de busca, os dados de contexto para teste em alto nível de abstração (B). Estes dados de contexto alimentam a Base de Conhecimento de Contexto para Testes (definido na Seção 3.2.2) (C). Os dados de contexto armazenados nesta base são combinados com as estratégias presentes no Repositório de Padrões de Defeitos de Contexto (apresentado na Seção 3.2.3), e servem como entrada para o Combinador de Casos de Teste (D) e (E). O Combinador de Casos de Teste (explicado na Seção 3.2.4) gera um conjunto de casos de teste através da realização de todas as combinações possíveis entre os dados de contexto escolhidos na CKTB e os padrões de defeitos (F). Esta combinação gera um número grande de casos de teste. Para reduzir este conjunto de casos de teste, o Redutor de Casos de Teste (detalhado na Seção 3.2.5) aplica uma função de similaridade para retirar os casos de teste mais parecidos entre si dentro do conjunto, retornando assim um conjunto menor de casos de teste (G). Detalhamos a seguir o funcionamento de cada um

destes componentes dentro do método proposto.

3.2.1 Analisador de Modelo

O Analisador de Modelo é responsável por extrair os dados de contexto dos modelos da aplicação através da execução de algoritmo de busca em profundidade. Para isto este componente recebe como entrada o modelo de contexto da aplicação e gera como saída dados de contexto para teste em alto nível de abstração, como mostrado na Figura 3.3. O modelo de contexto é fornecido pelo responsável pelo projeto da aplicação.

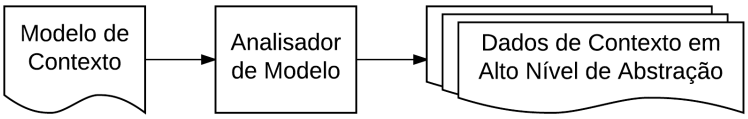

Figura 3.3 Representação do componente Analisador de Modelo com sua entrada e saída.

No caso da solução proposta, o modelo precisa conter informações sobre o comportamento de adaptação ao contexto e a estrutura de aquisição de contexto para as entidades mapeadas na aplicação. O aspecto de comportamento serve para identificar como a aplicação se adapta ao contexto. No nosso exemplo de uso, ele define quais elementos contextuais vão causar o bloqueio ou liberação da função de envio de dados da aplicação, e em quais casos o comportamento é acionado. O aspecto estrutural fornece informações acerca dos elementos contextuais que são utilizados na execução de uma tarefa pelo usuário da aplicação. O aspecto estrutural deve mencionar também quais são as fontes de contexto responsáveis por fornecer os dados para cada elemento contextual. Utilizando a aplicação exemplo, temos como elementos contextuais a localização e a conectividade do usuário. Estas informações são adquiridas respectivamente através de sensores de GPS e Wi-fi do dispositivo.

Para gerar os dados de contexto em alto nível de abstração, adotamos uma abordagem que fixa um ponto inicial e final dentro do modelo comportamental, representando assim a resolução de um problema pela aplicação. Definidos os pontos de início e fim, fazemos uma busca em profundidade para percorrer todas as possibilidades dentro deste modelo. Para a aplicação exemplo, temos duas avaliações de elementos contextuais: uma verifica se a localização do usuário coincide com a configurada na aplicação, e a outra faz uma verificação com o identificador da rede Wi-fi.

Durante este processo de busca, identificamos cada ponto em que ocorre uma decisão com um elemento contextual. Cada decisão pode ter diversas saídas. Cada saída é identificada como um contexto lógico. Na aplicação exemplo, temos dois elementos contextuais sendo avaliados e duas condições para cada elemento contextual. No total, temos as quatro saídas (contextos lógicos). Para o elemento contexual de localização do usuário, temos que a coordenada de GPS coincide com o que está configurado ou não. E para o elemento contextual de conectividade, temos que o identificador da rede Wi-fi do usuário coincide com o configurado ou não.

Embora tenham mais significados que dados de contexto físico, os contextos lógicos

extraídos do modelo não são capazes de definir uma circunstância de usuário, que é a que define o comportamento esperado pela aplicação. Isto é definido na abstração de contexto de situação. Na aplicação exemplo, embora se consiga inferir que a função de envio de dados possa ser bloqueada apenas ao obter um identificador de rede Wi-fi não cadastrado, o contrário não consegue ser dito de forma isolada. Ou seja, para este exemplo não se pode inferir que o envio de dados deve ser liberado apenas ao verificar que a rede Wi-fi configurada na aplicação foi detectada pelo dispositivo. Para liberar o envio de dados, a coordenada de GPS também deve ser obtida e coincidir com o que foi configurado no aplicativo, o que caracteriza a situação do usuário.

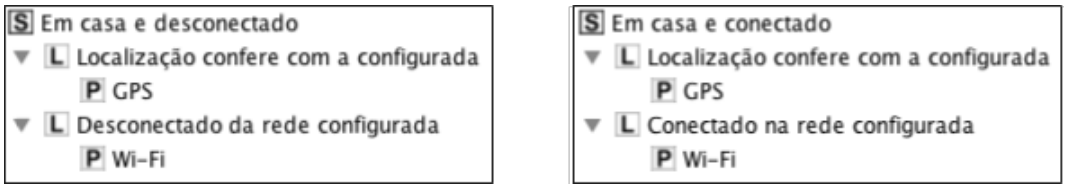

Figura 3.4 Algumas situações descobertas pelo Analisador de Modelo na aplicação exemplo. Cada situação [S] possui dois contextos lógicos [L] e cada contexto lógico está associado a um contexto físico [P], o qual representa o sensor.

Para obter a situação do usuário de forma que seja mais próxima do entendimento humano, o Analisador de Modelo percorre o modelo comportamental da aplicação e verifica todas as possibilidades de combinação entre cada decisão de contexto. Cada situação deve possuir um comportamento esperado. Na aplicação exemplo, temos algumas situações como as mostradas na Figura 3.4: "Em casa e conectado" e "Em casa e desconectado". A situação "Em casa e conectado' é formada pelos seguintes contextos lógicos: "Localização confere com a configurada", onde a coordenada obtida pelo dispositivo confere com a configuração de localização feita pelo usuário para o local de sua casa; e "Conectado na rede Configurada", onde o usuário se encontra conectado numa rede Wi-fi cujo nome seja igual ao da rede de sua casa. Já a situação "Em casa e desconectado" é formada pelos seguintes contextos lógicos: "Localização confere com a configurada';' e "Desconectado da rede Configurada", onde o identificador da rede Wi-fi obtido pelo dispositivo difere do configurado pelo usuário como sendo da casa dele. Na primeira situação a aplicação tem como comportamento esperado a liberação do envio de dados, enquanto que na segunda situação esta funcionalidade deve ser bloqueada.

Após extrair os dados do modelo de contexto da aplicação, os mesmos são armazenados na CKTB. Detalhamos o funcionamento desta base na próxima subseção.

3.2.2 Base de Conhecimento de Contexto para Testes

A CKTB é responsável por armazenar os dados de contexto de diferentes níveis de abstração que foram extraídos do modelo e/ou fornecidos pelo testador, tais como: contexto de baixo nível (físico ou virtual); contexto lógico, situação e cenário. A Figura 3.5 mostra as entidades e relacionamentos entre estes diferentes tipos de dados de contexto. Detalhamos a seguir os componentes presentes na CKTB.

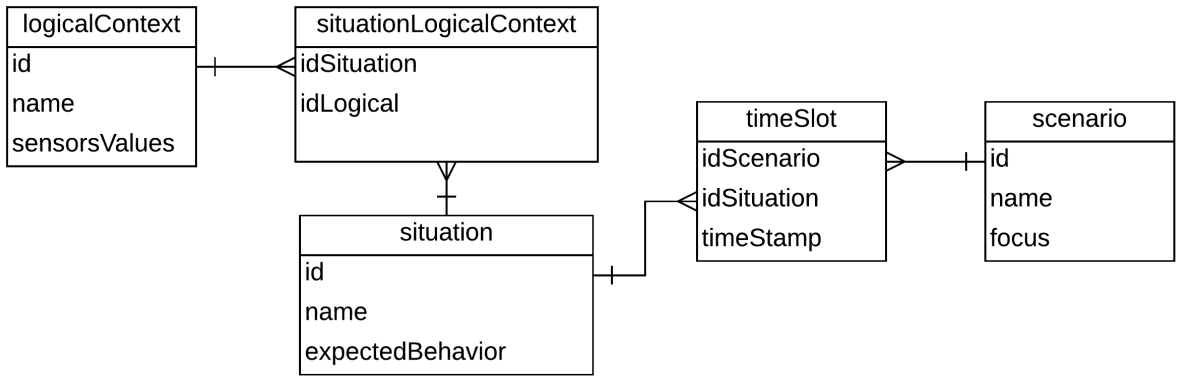

Figura 3.5 Diagrama de Entidade e Relacionamento da Base de Conhecimento de Contexto para Testes.

- **Contexto Lógico** (*logicalContext*): Entidade que representa o contexto lógico.
  - Identificador (*id*): Número que identifica o contexto lógico de forma única;
  - Nome (*name*): Nome dado para o contexto lógico;
  - Valores de sensores (*sensorsValues*): Atributo texto que armazena os valores de contextos físicos associados ao contexto lógico. Neste caso, utilizamos o formato JSON para armazenar os contextos físicos. Exemplo para uma coordenada GPS: [{"type":"GPS","latitude":-13.0003,"longitude":-38.5068,"altitude":0.0}]. Utilizamos este formato para que não haja necessidade de criação de uma entidade para cada tipo de contexto físico (sensor) diferente (GPS, Wi-fi, acelerômetro, dentre outros), pois cada sensor requer para sua representação um conjunto de dados diferente.
- **Situação** (*situation*): Entidade que representa uma situação.
  - Identificador (*id*): Número que identifica a situação de forma única.
  - Nome (*name*): Nome dado para a situação.
  - Comportamento esperado (*expectedBehavior*): Texto que indica o comportamento esperado pela aplicação em uma dada situação.
- **Cenário** (*scenario*): Entidade que representa um cenário.
  - Identificador (*id*): Número que identifica o cenário de forma única.
  - Nome (*name*): Nome dado ao cenário.
  - Foco (*focus*): Identificador do foco (do modelo comportamental) escolhido pelo testador ao criar o cenário.
- **Contextos Lógicos na Situação** (*situationLogicalContext*): Mapeia a relação entre as entidades Contexto Lógico e Situação, definindo quais contextos lógicos fazem parte de uma determinada situação.
- **Intervalo de Tempo** (*timeSlot*): Entidade que representa a associação entre a situação e o cenário.
  - Marca de tempo (*timeStamp*): Número que representa o momento que a situação ocorre dentro do cenário.

O cenário, diferentemente dos outros níveis de abstração de contexto (como contextos físicos, contextos lógicos e situações), não é obtido diretamente pelo Analisador de

Modelo. Isto ocorre pois o Analisador de Modelo não foi concebido para inferir relações temporais entre as situações possíveis dentro do modelo comportamental de contexto. O cenário é obtido através da ação do testador, o qual escolhe um modelo comportamental (relacionado a um foco) e realiza a combinação entre as situações descobertas do modelo de forma sequencial em intervalos de tempo.

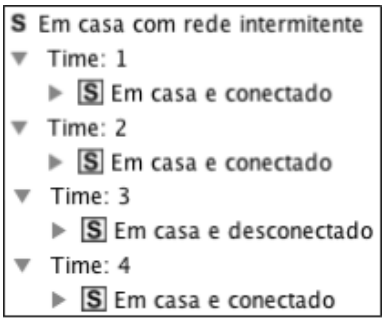

Figura 3.6 Cenário elaborado pelo testador. O cenário estrutura uma sequência de situações que indicam que o usuário está localizado em casa mas sua conectividade está intermitente. Este cenário é armazenado na CKTB e fica disponível para execução em diversos testes da aplicação.

Denominamos este cenário criado pelo testador como cenário base. A partir de sua criação, os cenários base são armazenados na CKTB e estão aptos a serem combinados com padrões de defeitos presentes no Repositório de Padrões de Defeitos de Contexto. Na aplicação exemplo, um cenário base pode ser definido como "Em casa com rede intermitente", onde as situações "Conectado em casa" e "Desconectado da rede de casa" são intercaladas entre si num período de tempo, como mostrado na Figura 3.6.

3.2.3 Repositório de Padrões de Defeitos de Contexto

O Repositório de Padrões de Defeitos de Contexto é responsável por armazenar as estratégias que indicam o modo como os cenários base são modificados para incluir padrões de defeitos nas fontes de contexto. Estes padrões foram abordados na Seção 2.2.3 e são registrados de acordo com o modelo mostrado na Figura 3.7. Detalhamos a seguir os componentes presentes neste repositório.

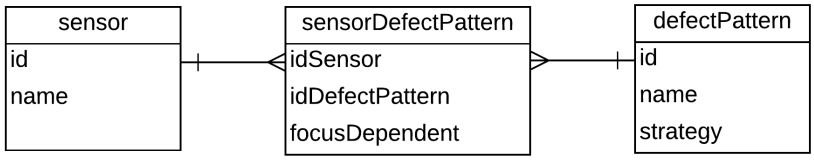

Figura 3.7 Diagrama de Entidade e Relacionamento do Repositório de Padrões de Defeitos de Contexto.

- **Sensor** (*sensor*): Entidade que representa o sensor (físico ou virtual).
  - **Identificador** (*id*): Número que identifica o sensor de forma única.
  - **Nome** (*name*): Nome do sensor.

- **Padrão de Defeito** (*defectPattern*): Entidade que descreve um padrão de defeito para sensores.
  - Identificador (*id*): Número que identifica o padrão de defeito de forma única.
  - Nome (*name*): Nome do padrão de defeito.
  - Estratégia (*strategy*): Atributo texto que descreve os passos da estratégia utilizada pelo padrão de defeito.
- **Padrão de Defeito do Sensor** (*sensorDefectPattern*): Entidade que define qual sensor pode ser afetado por determinado padrão de defeito.
  - Dependência do foco (*focusDependent*): Valor booleano que indica se o padrão de defeito associado ao sensor depende ou não do foco do modelo comportamental.

De acordo com o atributo *focusDependent*, separamos estes padrões em duas categorias: os que são dependentes dos sensores especificados no modelo da aplicação, e os que são independentes de sensores identificados no modelo. Os padrões de defeito dependentes de um sensor específico devem ser aplicados sobre os sensores utilizados em um cenário, enquanto os independentes utilizam fontes de contexto que não foram mapeadas para o cenário.

Um exemplo de um padrão de defeito que depende de um sensor identificado no modelo da aplicação é o de Granularidade Incompatível/Imprecisão, que deve ser aplicado a um sensor presente no cenário base armazenado na CKTB. No cenário exemplificado na Figura 3.6 podemos aplicar a falha no sensor de GPS ou de Wi-fi. Um exemplo de padrão de defeito que independe dos sensores é o Lógica Problemática/Comportamento Errado Causado por Interrupções. No cenário exemplificado, não temos o sensor de nível de bateria mapeado no modelo de contexto da aplicação. No entanto, o testador tem a possibilidade de incluir verificações deste tipo para avaliar como a aplicação se comporta com interrupções disparadas pelo dispositivo, mesmo que o projetista não tenha se preocupado com esta questão na concepção das regras de contexto da aplicação.

Portanto, destas duas formas os cenários base oriundos da CKTB e os padrões de defeitos existentes no Repositório de Padrões de Defeitos de Contexto são utilizados como entrada para o Combinador de Casos de Teste.

3.2.4 Combinador de Casos de Teste

O Combinador de Casos de Teste (destacado na Figura 3.8) é responsável por aplicar as estratégias associadas aos padrões de defeitos de fontes de contexto sobre os cenários base disponibilizados na CKTB. O resultado desta combinação é um conjunto contendo cenários modificados com o padrão de defeito.

O tamanho do conjunto de cenários modificados (ou simplesmente casos de teste gerados) depende da categoria do padrão de defeito: caso o padrão de defeito dependa de um sensor presente no cenário, este número é igual à quantidade de situações que utilizem o sensor dentro do cenário; caso o padrão de defeito independa do sensor mapeado no cenário, o número de casos de teste gerado é igual ao número de situações encadeadas no decorrer do cenário.

Na aplicação apresentada como exemplo, o testador pode criar um cenário base com

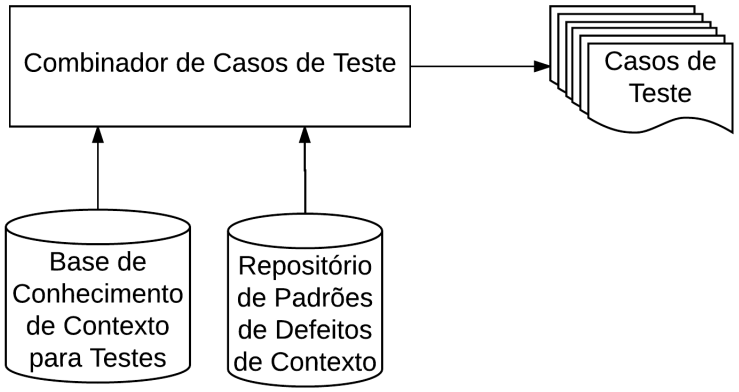

Figura 3.8 Representação do componente Combinador de Casos de Teste com sua entradas e saídas.

10 situações encadeadas no tempo. Em três destas situações a aplicação afere a posição do usuário com uso de GPS, e nas outras sete, não. Caso se deseje gerar um defeito dependente do sensor GPS (imprecisão, por exemplo) para este cenário, três cenários são derivados, cada um com o padrão de defeito aplicado em um dos tempos. Caso se deseje aplicar um padrão que independe do modelo da aplicação (bateria a 5%, por exemplo), são gerados 10 cenários, onde cada um possui o padrão de defeito aplicado em um tempo diferente.

A partir dos exemplos listados acima podemos perceber que a solução proposta pode gerar um conjunto numeroso de casos de teste para um determinado padrão de defeito. Para não prover um conjunto de casos de teste de inviável execução, propomos um componente para diminuir o número destes casos de teste, o qual chamamos Redutor de Casos de Teste.

3.2.5 Redutor de Casos de Teste

O Redutor de Casos de Teste, mostrado na Figura 3.9, tem como principal objetivo reduzir o conjunto gerado de casos de teste. Uma preocupação que existe na redução de casos de teste é a possibilidade de perda da capacidade de detectar defeitos no conjunto de casos de teste resultante da redução.

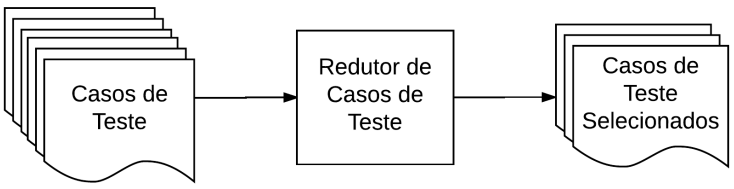

Figura 3.9 Representação do componente Redutor de Casos de Teste com sua entrada e saída.

O método escolhido para a redução dos casos de teste gerado foi o baseado em similaridade com uso de função de distância. Este método foi escolhido observando os resultados

apontados no estudo conduzido em (Coutinho, Cartaxo e Machado, 2014), o qual conclui na pesquisa bibliográfica que métodos baseados em similaridade que usam funções de distância geram conjunto de casos de teste com menor risco de perda de capacidade de detecção de falhas.

Este método de redução analisa casos de teste par a par com uma função de distância e elimina os casos de teste mais semelhantes entre si. Para determinar um grau de divergência entre dois casos de teste distintos utilizamos uma função de distância. Adotamos uma função que calcula o índice de Jaccard, proposto em (Jaccard, 1901). Este índice calcula a razão entre os dois casos de teste ao fornecer a razão entre o número de elementos comuns aos dois e o número de elementos distintos dos dois.

$$Jac(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Para calcular o grau de similaridade entre dois casos de teste (que no CTC representamos na forma de cenários), avaliamos as situações presentes em cada um. Para ilustrar, consideremos os cenários $c_1 = \{A, B, BImpY, C, D\}$ e $c_2 = \{A, B, C, CImpY, D\}$. A, B, C, D são situações existentes na CKTB, e $BImpY, CImpY$ são as situações B e C modificadas para apresentar o padrão de defeito de imprecisão num sensor Y. Para obter o índice de Jaccard dos cenários c_1 e c_2, realizamos o seguinte cálculo:

$$Jac(c_1, c_2) = \frac{|c_1 \cap c_2|}{|c_1 \cup c_2|} = \frac{\big|\{A, B, C, D\}\big|}{\big|\{A, B, BImpY, C, CImpY, D\}\big|} = \frac{4}{6} = 0.66666$$

Assim determinamos que o grau de similaridade entre c_1 e c_2 é de 66.666% ao usar o índice de Jaccard.

Para cada caso de teste devemos verificar o seu grau de similaridade com todo o restante do conjunto de casos de teste. Para armazenar os resultados destas verificações utilizamos o que se chama de matriz de similaridade, como mostrado no exemplo da Figura 3.10. Esta matriz é definida da seguinte maneira (Cartaxo, Machado e Neto, 2011):

- É uma matriz quadrada (n × n) , onde n é o número de casos de teste e cada linha e coluna representa um caso de teste;
- Cada elemento da matriz a_{ij} representa o grau de similaridade entre dois casos de teste i e j. Este grau de similaridade é obtido ao aplicar a função de similaridade escolhida entre estes casos de teste;
- A matriz é simétrica pois o valor calculado (grau de similaridade) para a_{ij} é igual ao de a_{ji}.

Após obter as distâncias entre todos os casos de teste e preencher a matriz de similaridade, o Algoritmo 1 pode ser executado. Este algoritmo executa até que toda a matriz seja analisada (através da condição avaliada na linha 2). Dentro do laço *while*, o primeiro passo (linhas 3-6) é encontrar o valor máximo dentro da matriz. Este valor é associado ao dois casos de teste mais similares entre si. No próximo passo (linhas 7-17), escolhe-se caso de teste com menos passos ($primeiraEscolha$), ou quando os dois casos de teste possuem o mesmo tamanho, escolhe-se um aleatoriamente. Depois (linhas 18-19), o caso de teste $segundaEscolha$ é removido da matriz de similaridade, e o caso de

$$
\begin{array}{c}
 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6
\end{array}
\begin{array}{c}
\begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\
\left(\begin{array}{cccccc}
 & 0.333 & 0.145 & 0.892 & 0.882 & 0.75 \\
 & & 0.667 & 0.888 & 0.433 & 0.333 \\
 & & & 0.500 & 0.666 & 0.889 \\
 & & & & 0.25 & 0.567 \\
 & & & & & 0.342 \\
 & & & & &
\end{array}\right)
\end{array}
$$

Figura 3.10 Exemplo de matriz de similaridade para seis casos de teste.

teste *primeiraEscolha* é adicionado ao conjunto reduzido de casos de teste. Ao final o algoritmo retorna o conjunto de casos de teste reduzido.

Algoritmo 1 Algoritmo de redução de casos de teste baseado em similaridade adaptado de (Coutinho, Cartaxo e Machado, 2014)

```
1:  function ReduzirCasosDeTeste(CasosDeTeste, MatrizSimilaridade)
2:      while (!todosValoresDaMatrizAnalisados()) do
3:          valoresMaximos ← obterTodosValoresMaximos(MatrizSimilaridade)
4:          par ← valoresMaximos.Obter(1)
5:          cenario1 ← par[0]
6:          cenario2 ← par[1]
7:          if cenario1.tamanho() < cenario2.tamanho() then
8:              primeiraEscolha ← cenario1
9:              segundaEscolha ← cenario2
10:         else if cenario1.tamanho() > cenario2.tamanho() then
11:             primeiraEscolha ← cenario2
12:             segundaEscolha ← cenario1
13:         else
14:             parEscolhido ← escolhaAleatoria(cenario1, cenario2)
15:             primeiraEscolha ← parEscolhido.ObterPrimeiroCasoDeTeste()
16:             segundaEscolha ← parEscolhido.ObterSegundoCasoDeTeste()
17:         end if
18:         MatrizSimilaridade.remover(segundaEscolha)
19:         conjuntoReduzido.adicionar(primeiraEscolha)
20:     end while
21:     return conjuntoReduzido
22: end function
```

Diferentemente do algoritmo original apresentado em (Coutinho, Cartaxo e Machado, 2014), o algoritmo aqui utilizado não avalia se os requisitos de teste estão satisfeitos. Consideramos que os casos de teste da entrada sempre satisfazem os requisitos de teste, pois todos os casos de teste da matriz de similaridade são fornecidos pela combinação de um cenário base e um padrão de defeito. A partir da execução do Algoritmo 1 os casos

de teste são disponibilizados para execução pelo testador.

3.3 IMPLEMENTAÇÃO DO PROTÓTIPO

Nesta seção apresentamos a implementação de um protótipo que realiza a prova de conceito da proposta definida neste trabalho, cuja visão geral é ilustrada na Figura 3.11.

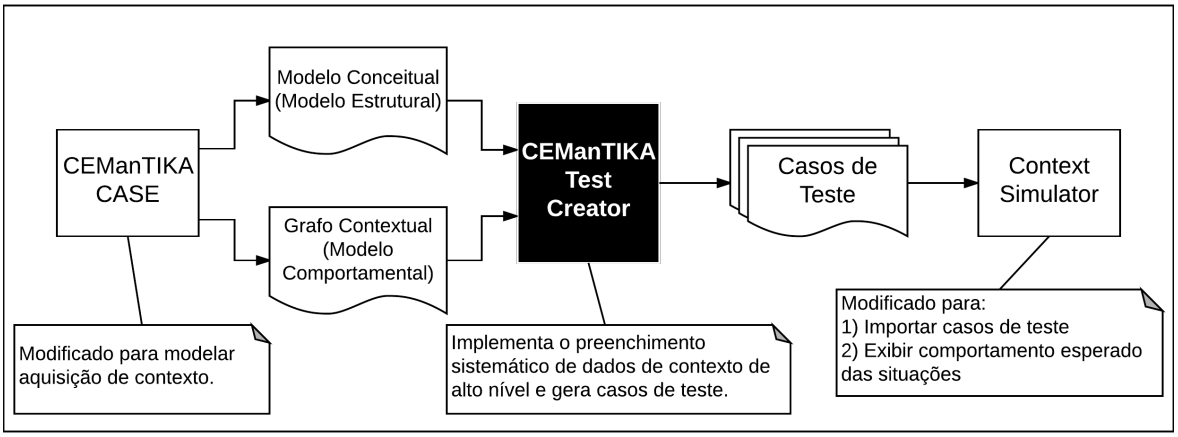

Figura 3.11 Visão geral da implementação do protótipo. Exibimos os ajustes feitos no CEManTIKA CASE, os modelos que são gerados e aproveitados como entrada para o CEManTIKA Test Creator, e os casos de teste que são gerados pelo CTC, os quais são executados por uma versão adaptada do Context Simulator.

Para prover os artefatos requeridos pela arquitetura proposta, implementamos uma adaptação no metamodelo de contexto do framework CEManTIKA (Vieira, Tedesco e Salgado, 2011), em conjunto com uma série de extensões na ferramenta CEManTIKA CASE (Patricio, 2010). Desta forma, esta ferramenta pode ser utilizada para a especificação da aquisição do contexto pela aplicação, gerando os artefatos (o modelo conceitual e o grafo contextual) que são utilizados como entrada para o CTC.

Como o CEManTIKA CASE foi implementado visando o apoio nas atividades de especificação e projeto de SSC, precisamos modificar esta ferramenta para sistematizar, através de uma série de tarefas, a criação de dados de contexto para a geração de casos de teste pelo CTC. A correta execução destas tarefas depende do suporte provido pela arquitetura que definimos na Seção 3.2.

Para viabilizar a execução dos casos de teste gerados pelo CTC, implementamos algumas modificações no simulador de contexto Context Simulator (Vieira, Holl e Hassel, 2015) para ajudar o testador na importação dos casos de teste gerados e na avaliação do comportamento da aplicação durante os testes.

3.3.1 Modelos de Aquisição de Contexto no CEManTIKA CASE

A ferramenta CEManTIKA CASE implementa parcialmente o processo e o metamodelo de contexto propostos no framework CEManTIKA (Vieira, Tedesco e Salgado, 2011).

Explicamos nesta subseção a implementação que tem como objetivo permitir a definição dos sensores utilizados como fontes de contexto dos elementos contextuais. A seguir detalhamos como este objetivo foi alcançado.

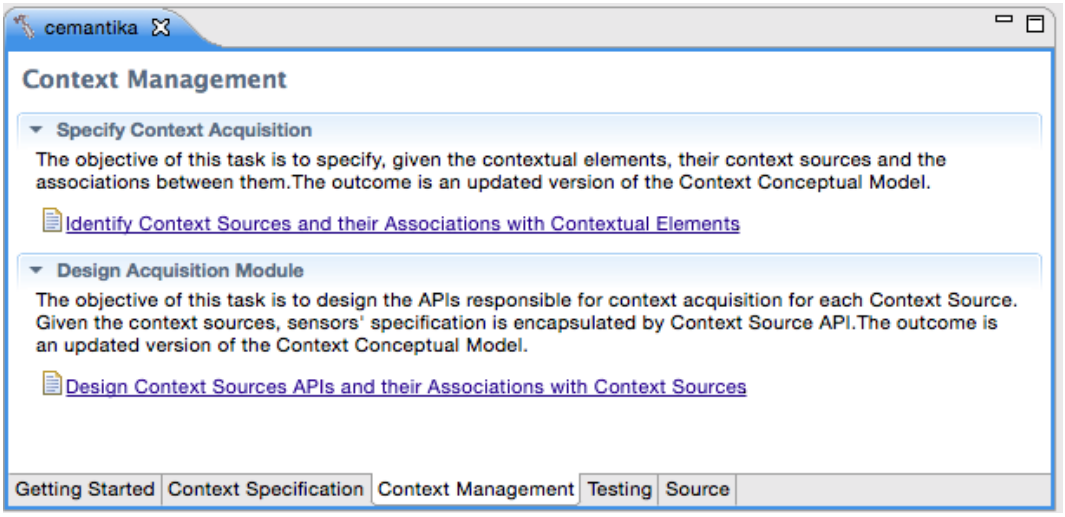

Figura 3.12 Implementação na ferramenta CEManTIKA CASE das tarefas Especificar Aquisição de Contexto (M1) e Projetar o Módulo de Aquisição (M2), da atividade de Projeto de Gerenciamento de Contexto.

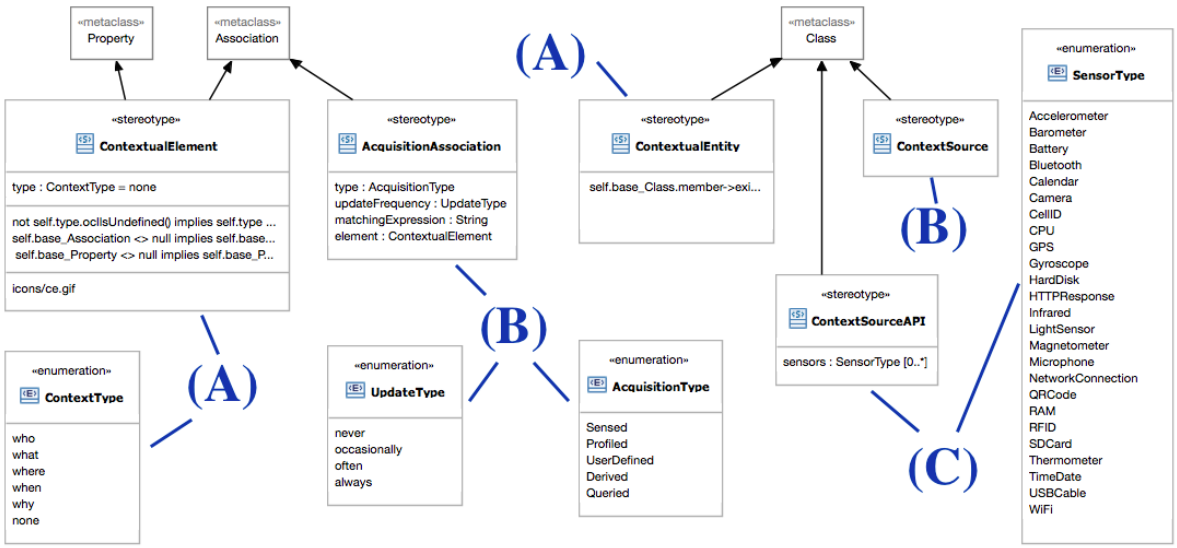

Figura 3.13 Metamodelo de contexto do CEManTIKA CASE. Os estereótipos associados a (A) são os originais da ferramenta CEManTIKA CASE, os que estão em (B) e (C) são frutos de ajustes na ferramenta. Os que estão associados a (B) já estavam previstos no framework CEManTIKA mas não estavam implementados na ferramenta, enquanto que os que se relacionam com (C) foram concebidos nesta dissertação.

As tarefas do CDP necessárias para descrever os sensores e que estão ausentes na

ferramenta CEManTIKA CASE são: Especificar Aquisição de Contexto (M1) e Projetar o Módulo de Aquisição (M2). A Figura 3.12 mostra como estas duas tarefas foram estruturadas dentro da ferramenta, e a Figura 3.13 ilustra os ajustes realizados no UML *Profile* do metamodelo CEManTIKA.

Para apoiar a tarefa Especificar Aquisição de Contexto (M1), implementamos no metamodelo presente na ferramenta CEManTIKA CASE os estereótipos *ContextSource* e *AcquisitionAssociation*. Ambos estereótipos estão presentes no framework CEManTIKA, assim como as enumerações *UpdateType* e *AcquisitionType*.

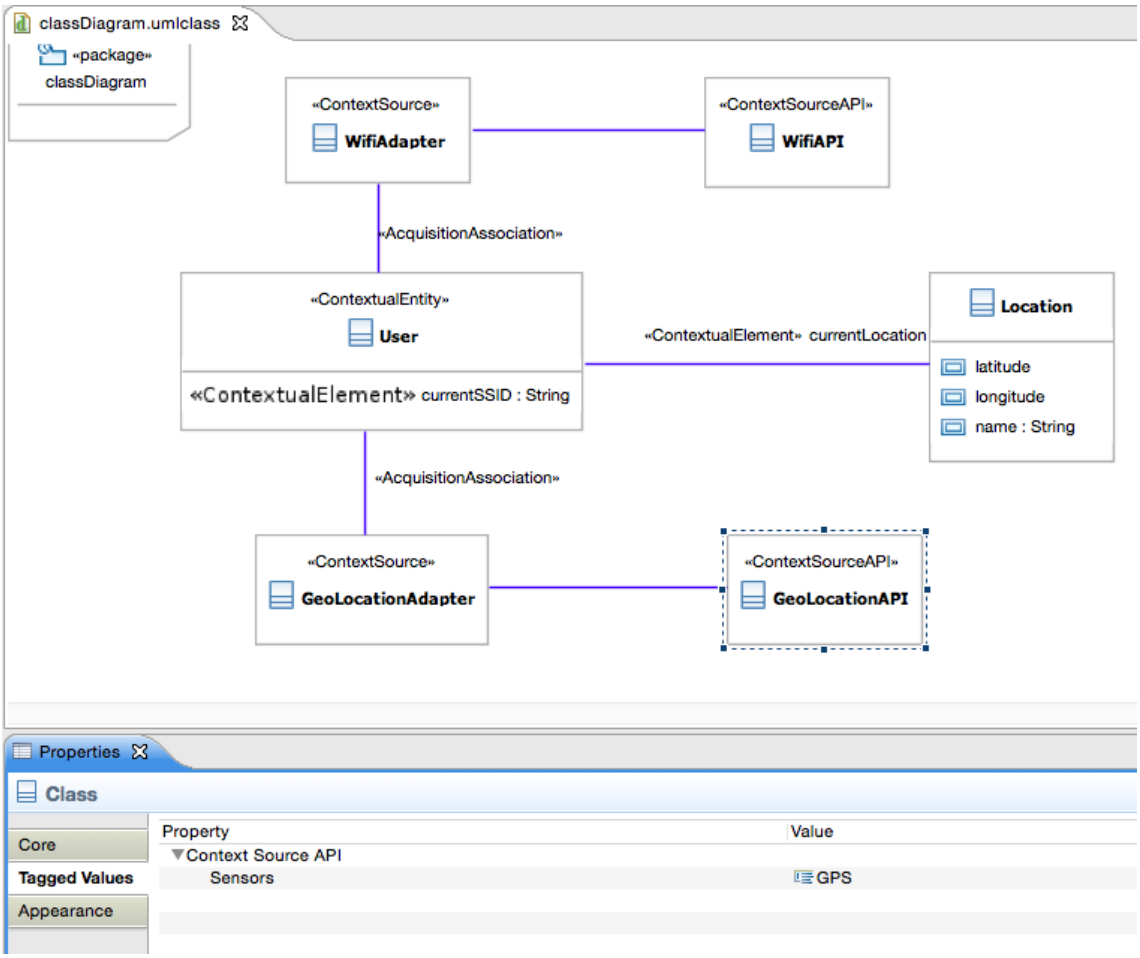

Figura 3.14 Diagrama com especificação de aquisição de contexto e das API's de fontes de contexto da aplicação *PersonalUploadManager*. Destaque para a especificação do sensor de GPS para a API de fonte de contexto *GeoLocationAPI*, a qual fornece dados de contexto para o elemento contextual de localização do usuário.

Para a tarefa Projetar o Módulo de Aquisição (M2), propomos um estereótipo UML que não foi idealizado no framework CEManTIKA, que é o *ContextSourceAPI*. Este estereótipo é aplicado sobre as classes que representam as *application programming interfaces* (APIs) que servem para abstrair detalhes internos da aquisição de dados de fontes de contexto, sejam elas físicas ou virtuais. Dentro do estereótipo criado temos também

uma lista dos sensores utilizados pela API da fonte de contexto, os quais estão listados dentro da enumeração *SensorType*, também não idealizada no framework CEManTIKA.

Na aplicação de exemplo, como ilustrado na Figura 3.14, para a tarefa de Especificação de Aquisição de Contexto criamos dois adaptadores de contexto (*WifiAdapter* e *GeolocationAdapter*) e os relacionamos respectivamente com os elementos contextuais *currentSSID* e *currentLocation* através de associações com estereótipo *AcquisitionAssociation*. Na tarefa de projeto especificamos duas API's das fontes de contexto (*WifAPI* e *GeoLocationAPI*).

Com a implementação destas modificações o projetista do aplicativo móvel pode definir um modelo conceitual de contexto que indique quais sensores fornecem dados para um elemento contextual.

3.3.2 Aspectos Gerais da Implementação do CEManTIKA Test Creator

Neste subseção mostramos algumas especificações técnicas do protótipo desenvolvido nesta dissertação. Disponibilizamos publicamente o código fonte tanto do CEManTIKA CASE estendido[1] quanto do Context Simulator modificado[2].

O CEManTIKA Test Creator foi implementado como uma extensão do CEManTIKA CASE, que é um plugin para o Eclipse. Como o CEManTIKA CASE é originalmente uma ferramenta para projeto, precisamos implementar as funcionalidades referentes ao teste, cujos pacotes inserimos no projeto Eclipse org.cemantika.modeling. A Figura 3.15 mostra os pacotes adicionados, que são:

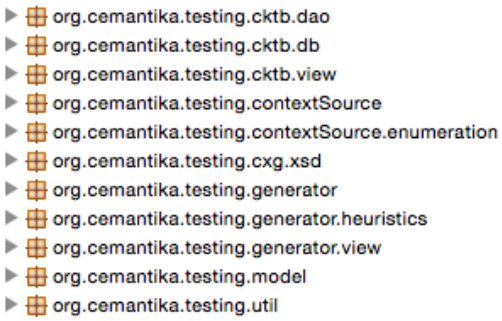

Figura 3.15 Pacotes java adicionados à ferramenta CEManTIKA CASE para a implementação do CEManTIKA Test Creator.

- *org.cemantika.testing.cktb.dao*: Classes de acesso aos dados de contexto de alto nível de abstação que estão armazenados na CKTB;
- *org.cemantika.testing.cktb.db*: Classes de configuração da CKTB;
- *org.cemantika.testing.cktb.view*: Classes de visualização das telas desenvolvidas que realizam o preenchimento de dados de contexto, nos mais variados níveis de abstração;
- *org.cemantika.testing.contextSource*: Classes que representam fontes de contexto. Foi implementada tendo como base classes similares presentes no Context Simulator;

[1]https://github.com/andreluismps/cemantika-modeling
[2]https://github.com/andreluismps/ContextSimulator

- *org.cemantika.testing.contextSource.enumeration*: Classes de enumeração para as fontes de contexto;
- *org.cemantika.testing.cxg.xsd*: Classes que representam o schema XML utilizado pelo Analisador de Modelos para ler o conteúdo do modelo comportamental
- *org.cemantika.testing.generator*: Classes que implementam o redutor de casos de teste;
- *org.cemantika.testing.generator.heuristics:*: Classes que implementam o repositório de padrões de defeitos e o combinador de testes;
- *org.cemantika.testing.generator.view*: Classes que implementam a visualização do Combinador de Casos de Teste;
- *org.cemantika.testing.model*: Classes que representam contexto em vários níveis de abstração; Classes que implementam o Analisador de Modelo;
- *org.cemantika.testing.util*: Classes utilitárias, incluindo classes que tratam da exportação de dados em formato JSON.

A seguir detalhamos as tarefas que são realizadas para a criação de casos de teste com a implementação apresentada nesta subseção.

3.3.3 Tarefas para a Geração de Casos de Teste

Nesta subseção detalhamos as tarefas necessárias para a geração dos casos de teste utilizando os artefatos de projeto de contexto fornecidos pela ferramenta CEManTIKA CASE modificada. A Figura 3.16 ilustra a relação entre cada tarefa para obter os casos de teste, e a Figura 3.17 mostra como estas tarefas foram implementadas na aba Testing do CEManTIKA CASE. A seguir mencionamos cada uma das tarefas presentes no método proposto.

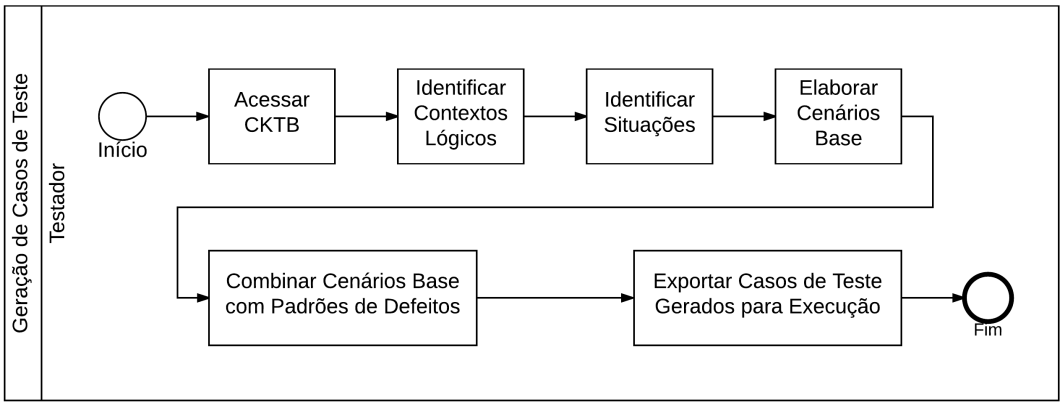

Figura 3.16 Passos realizados pelo testador para a criação dos casos de teste de contexto.

Na aba Testing assume-se que o testador já executou todas as tarefas previstas no framework CEManTIKA para um determinado foco (na aplicação exemplo é *User executes Manage File Upload*), e está apto a gerar o conjunto de casos de teste para execução.

A primeira tarefa a ser executada pelo testador é *Acessar CKTB*, e ela é acionada na seção *Import Context Knowledge Test Base*.

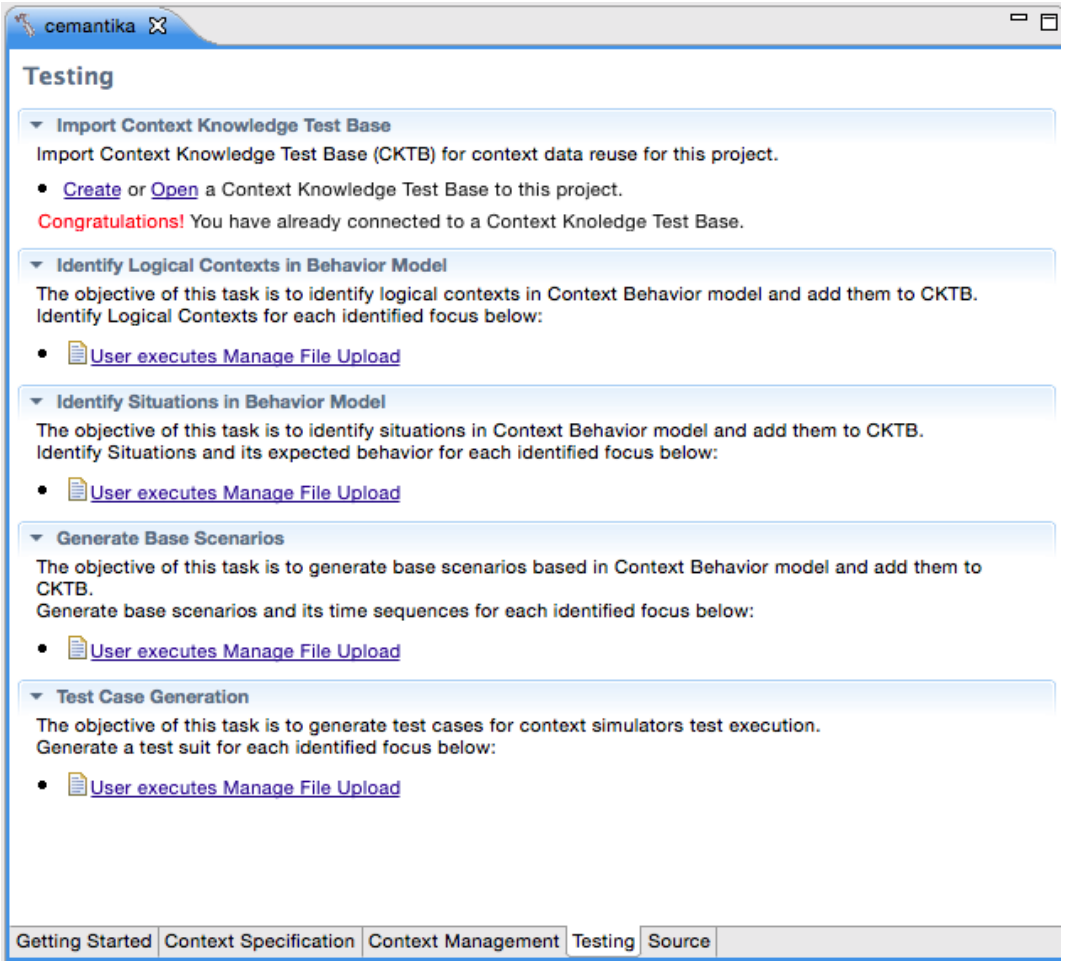

Figura 3.17 Implementação na ferramenta CEManTIKA CASE das tarefas de geração de casos de teste definidas no método CTC.

Depois de se conectar a uma CKTB, o testador pode executar a tarefa *Identificar Contextos Lógicos*, implementada na seção *Identify Logical Contexts in Behavior Model*. Para tanto, o testador deve selecionar um foco da aplicação e identificar os contextos lógicos presentes nele. Na aplicação exemplo, definimos inicialmente apenas um foco, que é o "Gerenciar envio de dados". Aqui o Analisador de Modelo obtém os contextos lógicos do modelo de contexto e solicita o preenchimento dos dados dos sensores. Ao final desta tarefa os dados dos sensores presentes nos contextos lógicos pertencentes à tarefa escolhida estão informados e armazenados na CKTB.

Após identificar os contextos lógicos o testador pode executar a tarefa *Identificar Situações*, presente na seção *Identify Situations in Behavior Model*. De forma semelhante ao feito no passo anterior, os dados obtidos do Analisador de Modelo e da CKTB são combinados e o testador deve preencher as informações pertinentes às situações presentes no foco selecionado. Cada situação pode conter um comportamento esperado pela aplicação. É aqui que se define como a aplicação se comporta em determinada circunstância do usuário.

Ao término da identificação das situações e seus respectivos comportamentos esperados, é o momento do testador gerar os cenários base através da tarefa *Elaborar os Cenários Base*, implementada na seção *Generate Base Scenarios*. Denominamos cenário base para diferenciar estes cenários, que são criados pelo testador, dos que são criados a partir da execução da tarefa *Combinar Cenários Base com Padrões de Defeitos*. Na tarefa *Elaborar os Cenários Base* o Analisador de Modelo apenas obtém a lista de situações pertinentes ao foco selecionado pelo testador. Nesta tarefa o testador deve criar cenários que descrevam o uso típico da aplicação ao encadear diversas situações no tempo. O testador define quantas situações são encadeadas e qual a ordem das mesmas no decorrer do tempo. Como resultado da execução desta tarefa temos o conjunto de cenários base, os quais são armazenados na CKTB. A partir de então o testador pode gerar os casos de teste combinados com padrões de defeitos de contexto.

Após elaborar os cenários base, o testador pode executar a tarefa *Combinar Cenários Base com Padrões de Defeitos*, presente na seção *Test Case Generation*. Nesta tarefa o testador deve realizar a combinação entre os cenários base armazenados na CKTB e os padrões de defeitos de contexto. O testador pode escolher um dos dois tipos de padrão de defeito e aplicar aos cenários selecionados. Caso seja um tipo que dependa do sensor identificado no modelo comportamental, o testador deve primeiro selecionar o sensor e depois o tipo de defeito desejado. Caso seja um tipo de padrão de defeito que independe do modelo comportamental, o testador seleciona apenas o padrão de defeito desejado. Na aplicação exemplo, o testador pode escolher o cenário base "Em casa com rede intermitente" e combinar com o padrão de defeito "Granularidade Incompatível/Imprecisão" no sensor de GPS, ou então combinar o mesmo cenário selecionado com o padrão de defeito "Regra Lógica Problemática/Comportamento Errado Causado por Interrupções - bateria com nível de 5%". A partir de então o testador pode executar a tarefa *Exportar Casos de Teste Gerados para Execução*, cujo resultado é um conjunto de cenários fruto da combinação entre os cenários base escolhidos e os padrões de defeitos selecionados. No momento em que o testador aciona a exportação dos casos de teste, os componentes Combinador de Casos de Teste e Redutor de Casos de Teste são acionados.

Após a execução da última tarefa, um arquivo JSON é gerado e exportado para uso pelo Context Simulator. Mostramos os detalhes internos deste arquivo na Seção 3.3.8. O detalhamento da execução desta implementação com a aplicação exemplo está presente na Seção 3.4.

3.3.4 Analisador de Modelo

O Analisador de Modelo foi implementado na ferramenta CEManTIKA CASE de forma a tratar três níveis de abstração de contexto: contexto físico, contexto lógico e situação. Para obter os dados de contexto nestes níveis de abstração, o Analisador de Modelo recebe como entrada o modelo conceitual (mostrado na Figura 3.14) e o grafo contextual da aplicação, mostrado na Figura 3.18 e detalhado a seguir.

Para a aplicação exemplo, temos um grafo contextual para o foco "Gerenciar envio de dados". O primeiro nó contextual (círculo claro à esquerda) avalia o elemento contextual SSID atual do usuário, enquanto que o outro nó contextual avalia a localização do usuário.

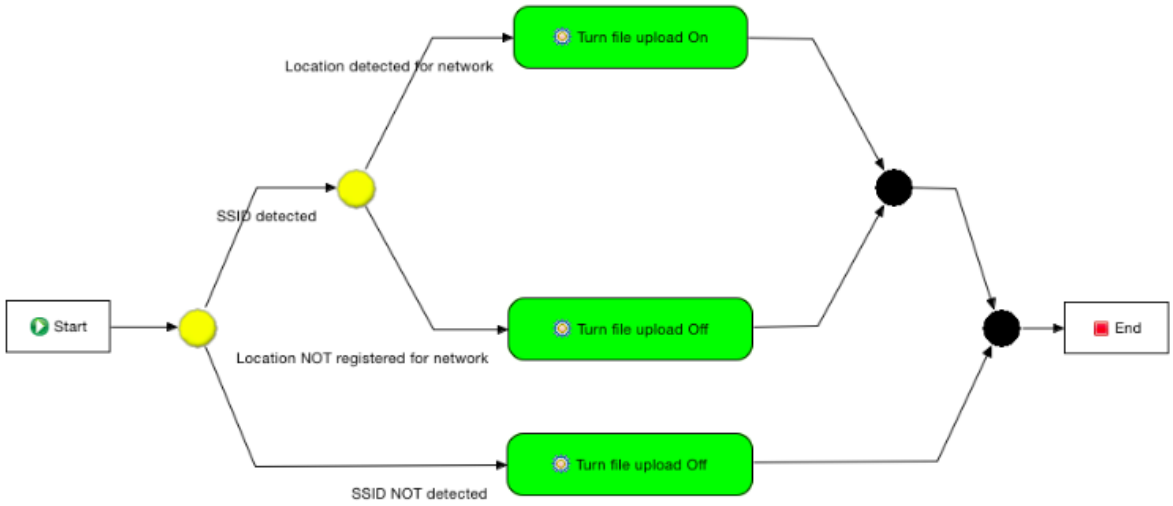

Figura 3.18 Modelo comportamental de contexto da aplicação *PersonalUploadManager* seguindo grafo contextual do metamodelo CEManTIKA.

Os retângulos representam as ações tomadas pelo sistema de acordo com as informações de contexto capturadas pelo aplicativo.

Para obter os contextos físicos (sensores) e lógicos, o algoritmo de busca em profundidade identifica todas as saídas dos nós contextuais do grafo. Em cada saída do nó contextual está um elemento contextual avaliado na decisão, o qual caracteriza um contexto lógico. A identificação dos sensores que alimentam o elemento contextual é obtida através de uma consulta no modelo conceitual de contexto. Tendo como ponto de partida o elemento contextual e o modelo conceitual, o método de busca identifica, através da associação de aquisição, qual é a fonte de contexto do elemento contextual. Encontrada a fonte de contexto, a busca identifica a API da fonte de contexto, a qual contém a especificação do sensor que fornece dados para o elemento contextual.

Para obter os dados da situação, o algoritmo de busca em profundidade identifica todos os caminhos dentro do grafo contextual (com seus contextos lógicos já descobertos) e realiza a agregação destes contextos lógicos em uma situação, criando a relação de uma situação para cada caminho do grafo contextual. Quando o algoritmo de busca encontra uma ação no grafo contextual, a descrição da mesma é adicionada ao texto que descreve o comportamento esperado da aplicação na situação.

Desta maneira, o Analisador de Modelo prepara os dados de contextos para a elaboração de casos de teste e os persiste na CKTB.

3.3.5 Base de Conhecimento de Contexto para Testes

A CKTB foi implementada como um banco de dados sqlite[3] dentro da ferramenta CEManTIKA CASE. Os comandos de definição de dados seguem na Listagem 3.1 e obedecem ao esquema definido na Figura 3.5.

[3]https://sqlite.org/

Listagem 3.1 Criação das tabelas da CKTB no banco sqlite.

```
1
2 public static void createDb(Connection conn){
3   List<String> tablesDDL = new ArrayList<String>();
4   tablesDDL.add("CREATE TABLE scenario (id INTEGER PRIMARY KEY
      AUTOINCREMENT, name text, focus text)");
5   tablesDDL.add("CREATE TABLE situation (id INTEGER PRIMARY KEY
      AUTOINCREMENT, name text, expectedBehavior text)");
6   tablesDDL.add("CREATE TABLE logicalContext (id INTEGER PRIMARY
      KEY AUTOINCREMENT, name text, sensorsValues text)");
7   tablesDDL.add("CREATE TABLE situationLogicalContext (idSituation
      numeric, idLogical numeric)");
8   tablesDDL.add("CREATE TABLE timeSlot (idScenario numeric,
      idSituation numeric, timeStamp numeric)");
9
10  DataBase.executeUpdate(tablesDDL, conn);
11 }
```

3.3.6 Repositório de Padrões de Defeito de Contexto

Para implementar o Repositório de Padrões de Defeito de Contexto no CEManTIKA CASE, escolhemos codificá-lo diretamente no seu código-fonte. Em vez de criar tabelas em um banco de dados fazemos na CKTB, optamos pela praticidade de utilizar classes de enumeração do Java para descrever de forma fixa as propriedades dos padrões de defeitos, pois no escopo do protótipo não vislumbramos necessidade de atualização nestes dados. Apresentamos os exemplos de cenários gerados na Seção 3.4.

Por restrições de escopo escolhemos implementar quatro dos 15 padrões de defeitos apresentados na Seção 2.2.3. Escolhemos estes quatro padrões de defeitos levando em conta os padrões de eventos mencionados em (Wang, Elbaum e Rosenblum, 2007; Amalfitano et al., 2013; Yu e Takada, 2015), como os que tratam de atrasos no recebimento de dados de sensores, da perda e recuperação de sinal, mudanças repentinas de valor de sensor, e de interrupções causadas pelo dispositivo do usuário. A seguir enumeramos estes quatro defeitos, no formato falta/falha: Incompleto/Indisponível (Tabela 2.2), que caracteriza perda e recuperação de sinal; Detecção Lenta/Dado Desatualizado (Tabela 3.1), que caracteriza atraso no recebimento de dados de sensores; Granularidade Incompatível/Imprecisão (Tabela 3.2), que caracteriza mudanças repentinas de valor de sensor causadas pela imprecisão; e Regra Lógica Problemática/Comportamento Errado Causado por Interrupções (Tabela 3.3), que caracteriza interrupções causadas pelo dispositivo.

Usamos a seguinte abordagem para explicar as estratégias: inicialmente especificamos a estratégia em termos genéricos, e depois ilustramos a mesma com dados de contexto de alto nível da aplicação exemplo. Definimos tanto os termos genéricos e os dados de contexto de alto nível utilizados na próxima subseção. A partir de então abordamos as estratégias de combinação para os padrões de defeitos implementados.

Tabela 3.1 Especificação de um caso de teste abstrato contendo a Falta Detecção Lenta e Falha Dado Desatualizado para uma fonte de contexto. Adaptado de (Hassel, 2014).

Falta: Detecção Lenta / **Falha:** Dado Desatualizado
Caso de teste abstrato
Pré-condição: Aplicação requisita dado da fonte de contexto. **Ação:** Selecione outro valor (não o esperado) para a fonte de contexto, para simular um valor defasado. **Pós-condição:** Simulador transmite dados com informações defasadas.

Tabela 3.2 Especificação caso de teste abstrato contendo a Falta Granularidade Incompatível - Falha Imprecisão para uma fonte de contexto. Adaptado de (Hassel, 2014).

Falta: Granularidade Incompatível / **Falha:** Imprecisão
Caso de teste abstrato
Pré-condição: Uma fonte de contexto transmite dados. **Ação:** Modifique levemente o dado simulado e o envie novamente. **Pós-condição:** Mesmo que a fonte de contexto envie o dado atualizado, não implica que o contexto de fato tenha sido modificado.

Tabela 3.3 Especificação caso de teste abstrato contendo a Falta Lógica Problemática - Falha Comportamento Errado Causado por Interrupções para uma fonte de contexto. Adaptado de (Hassel, 2014).

Falta: Lógica Problemática / **Falha:** Comportamento Errado Causado por Interrupções
Caso de teste abstrato
Pré-condição: Aplicação móvel está em execução. **Ação:** Simule um evento de interrupção (ex: bateria com carga muito baixa). **Pós-condição:** A interrupção influencia a aplicação móvel.

3.3.6.1 Definições de Dados de Contexto Manipulados

Antes de detalhar a estratégia de combinação destes padrões de defeitos com os cenários, devemos definir os seguintes termos, os quais são utilizados no detalhamento de cada um dos padrões de defeitos:

- Cenário base S, formado pelas situações A, B, C e D no decorrer do tempo, como mostrado na Tabela 3.4;
- Contextos lógicos $L1$ e $L2$ dentro das situações presentes no cenário S;

- O sensor Y, presente no modelo comportamental;
- O sensor Z, ausente no modelo comportamental da aplicação.
- Para descrever sucintamente um padrão de defeito, usamos as seguintes abreviações:
  - UNV para Incompleto/Indisponível;
  - DTN para Detecção Lenta/Dado Desatualizado;
  - GRN para Granularidade Incompatível/Imprecisão;
  - INT para Regra Lógica Problemática/Comportamento Errado Causado por Interrupções.

Quando uma situação do cenário S é afetada por um padrão de defeito, alteramos a representação dela para "Situação_AbreviaçãoDoPadrão_Sensor". Exemplo: A situação B com o padrão de defeito Incompleto/Indisponível no sensor Y é representada como B_UNV_Y. Raciocínio similar é aplicado nos contextos lógicos. Um exemplo é a ocorrência, no contexto lógico $L2$, do padrão de defeito Detecção Lenta/Dado Desatualizado no sensor Y: L_DTN_Y.

Tabela 3.4 Definição do cenário base S com as situações A, B, C e D no decorrer do tempo.

Cenário	**T0**	**T1**	**T2**	**T3**
S	A	B	C	D

Para ilustrar as estratégias, instanciamos as definições de cenário base, situações, contextos lógicos e sensores com os dados existentes na aplicação exemplo:

- O cenário base é "Em casa com rede intermitente", conforme mostrado na Figura 3.6;
- As situações possíveis dentro deste cenário base são "Em casa e conectado" e "Em casa e desconectado";
- O sensor escolhido para aplicação do padrão de defeito que depende da presença do sensor no modelo comportamental é o de GPS. Sobre ele aplicamos os padrões de defeito Incompleto/Indisponível, Detecção Lenta/Dado Desatualizado e Granularidade Incompatível/Imprecisão;
- O contexto lógico que acessa dados do sensor de GPS e sofre as modificações por estes três padrões é "Localização confere com a configurada";
- O sensor escolhido que está ausente no modelo comportamental é o de bateria. Sobre ele aplicamos o padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções.
- Para representar o nome do contexto lógico e da situação que são afetados pelo padrão de defeito, usamos o seguinte formato: "nome original" - nome_sensor - "nome reduzido do padrão de defeito". Os nomes reduzidos para os padrões de defeito são:
  - *Indisponível* para Incompleto/Indisponível;
  - *Dado desatualizado* para Detecção Lenta/Dado Desatualizado;
  - *Imprecisão* para Granularidade Incompatível/Imprecisão;
  - *Comportamento Errado* para Regra Lógica Problemática/Comportamento Errado Causado por Interrupções

Com as definições e instanciações realizados nesta subseção, podemos detalhar as estratégias para os padrões de defeito implementados. Primeiro especificamos as estratégias nos termos definidos e depois ilustramos com a aplicação exemplo instanciada.

3.3.6.2 Estratégia para Padrões de Defeito em Sensores Modelados na Aplicação

Para os padrões de defeitos em sensores que estão presentes no modelo da aplicação, que são Incompleto/Indisponível, Detecção Lenta/Dado Desatualizado e Granularidade Incompatível/Imprecisão, definimos uma estratégia com seis passos, como mostramos a seguir:

- **Passo 1:** Identificar no cenário base S as situações que utilizam o sensor Y;
- **Passo 2:** Obter, nas situações identificadas, os contextos lógicos que utilizam o sensor Y;
- **Passo 3:** Gerar cópias destas situações;
- **Passo 4:** Obter contextos lógicos modificados com o padrão de defeito;
- **Passo 5:** Modificar as cópias das situações substituindo o contexto lógico original pelo afetado com o padrão de defeito;
- **Passo 6:** Gerar os cenários derivados com estas situações de acordo com o padrão de defeito.

Os três primeiros passos são idênticos para os três padrões de defeito contemplados. É a partir do Passo 4 que as estratégias revelam suas diferenças. Executamos os três primeiros passos a seguir:

No **Passo 1**, percorremos o cenário base S e descobrimos que as situações B, C e D utilizam o sensor Y. No **Passo 2**, obtemos o contexto lógico que utiliza o sensor Y: $L2$. No **Passo 3** geramos as cópias das situações, que são B', C' e D'.

Na aplicação exemplo, a execução destes três passos acontece da seguinte maneira: No **Passo 1** verificamos que as duas situações diferentes que estão presentes no cenário utilizam o sensor de GPS. No **Passo 2** obtemos o contexto lógico "Localização confere com a configurada", o qual utiliza o sensor de GPS. No **Passo 3** criamos, a partir das situações obtidas no Passo 1, as situações "Em casa e conectado - Cópia" e "Em casa e desconectado - Cópia". A seguir temos os passos seguintes da estratégia (3-6), os quais mudam de acordo com o padrão de defeito.

3.3.6.2.1 Padrão de Defeito Incompleto / Indisponível

O caso de teste abstrato para este padrão de defeito encontra-se na Tabela 2.2. O caso de teste concreto derivado deste padrão de defeito tem como característica a substituição de diversas situações por cópias modificadas onde o sensor escolhido é suprimido.

Neste padrão de defeito, no **Passo 4** criamos uma versão modificada do contexto lógico $L2$. Este novo contexto lógico é o $L2\_UNV\_Y$, e nele o dado do sensor Y é suprimido. No **Passo 5**, obtemos as situações cópia (B', C' e D') e as modificamos substituindo o contexto lógico $L2$ por $L2\_UNV\_Y$. Desta forma, obtemos as situações B_UNV_Y, C_UNV_Y e D_UNV_Y. No **Passo 6** geramos os casos de teste conforme apresentado

na Tabela 3.5. Para gerar os casos de teste a partir do cenário base S, substituímos as situações originais pelas modificadas cobrindo todas as combinações possíveis do defeito ocorrer a partir de um tempo t dentro do cenário S. Em outras palavras, o cenário gerado deve possuir o sensor desabilitado desde o tempo t até o final. Antes deste tempo t, o dado do sensor é fornecido conforme especificado no cenário base S.

Tabela 3.5 Cenários derivados do cenário base S com o dado do sensor Y indisponível a partir de um tempo t compreendido entre T0 e T3

Cenário	**T0**	**T1**	**T2**	**T3**
S	A	B	C	D
Defeito a partir de T1	A	B_UNV_Y	C_UNV_Y	D_UNV_Y
Defeito a partir de T2	A	B	C_UNV_Y	D_UNV_Y
Defeito a partir de T3	A	B	C	D_UNV_Y

Na aplicação exemplo, no **Passo 4** o contexto lógico "Localização confere com a configurada - GPS - Indisponível" é gerado a partir do contexto lógico identificado no Passo 2. Este novo contexto lógico teve a informação relativa ao sensor de GPS suprimida. No **Passo 5** modificamos as situações cópia substituindo o contexto lógico identificado no Passo 2 (original) pelo criado no Passo 4 (com o padrão de defeito). Desta forma, definimos as situações "Em casa e conectado - GPS - Indisponível" e "Em casa e desconectado - GPS - Indisponível". Mostramos esta transformação na Figura 3.19, onde à esquerda temos a situação original do cenário base escolhido, e à direita temos a aplicação do padrão de defeito Incompleto/Indisponível no sensor de GPS na situação "Em casa e conectado".

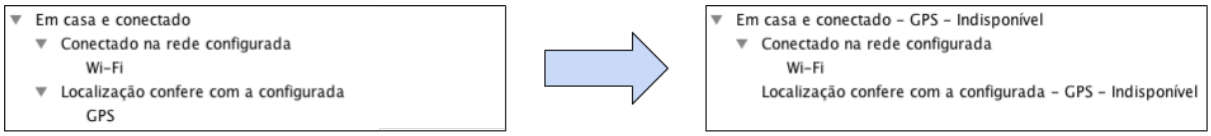

Figura 3.19 Exemplo de transformação de uma situação com o padrão de defeito Incompleto/Indisponível.

No **Passo 6** geramos os casos de teste de acordo com o mostrado na Figura 3.20. Nesta figura mostramos a criação de um dos cenários com o padrão de defeito usando o cenário base da aplicação exemplo. À esquerda temos o cenário base, e à direita o cenário derivado após aplicação do padrão de defeito Incompleto/Indisponível a partir do tempo 3. Como em todos os tempos do cenário existem situações que podem ser afetadas pelo padrão de defeito, geramos casos de teste com o defeito a partir dos tempos 1, 2 e 4.

3.3.6.2.2 Padrão de Defeito Detecção Lenta / Dado Desatualizado

O caso de teste abstrato para este padrão de defeito encontra-se na Tabela 3.1. O caso de teste concreto derivado deste padrão de defeito tem como característica a substi-

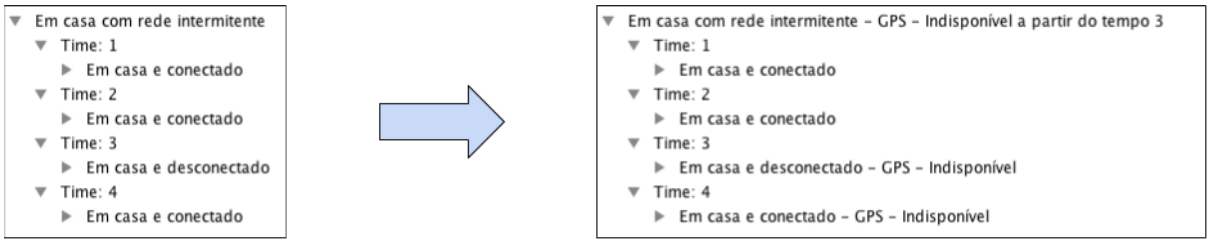

Figura 3.20 Exemplo de transformação do cenário da aplicação exemplo pelo padrão de defeito Incompleto/Indisponível, a partir do tempo 3.

tuição de diversas situações por cópias modificadas onde o dado do sensor escolhido está defasado.

Neste padrão de defeito, no **Passo 4** obtemos da CKTB uma versão modificada do contexto lógico $L2$. Este contexto lógico é o $L2\_DTN\_Y$, e nele o dado do sensor Y encontra-se defasado. No **Passo 5**, obtemos as situações cópia (B', C' e D') e as modificamos substituindo o contexto lógico $L2$ por $L2\_DTN\_Y$. Desta forma obtemos as situações B_DTN_Y, C_DTN_Y e D_DTN_Y. No **Passo 6** geramos os casos de teste conforme apresentado na Tabela 3.6. Para gerar os casos de teste a partir do cenário base S, substituímos as situações originais pelas modificadas cobrindo todas as combinações possíveis do defeito ocorrer até um tempo final t. Em outras palavras, o cenário gerado deve possuir o sensor com dados defasados até o tempo t. Depois deste tempo, o dado do sensor é atualizado como no cenário base S.

Tabela 3.6 Cenários derivados do cenário base S com o dado do sensor Y defasado a até um tempo t compreendido entre T0 e T3

Cenário	**T0**	**T1**	**T2**	**T3**
S	A	B	C	D
Defeito até T1	A	B_DTN_Y	C	D
Defeito até T2	A	B_DTN_Y	C_UNV_Y	D
Defeito até T3	A	B_DTN_Y	C_UNV_Y	D_UNV_Y

Na aplicação exemplo, no **Passo 4** o contexto lógico "Localização confere com a configurada - GPS - Dado Desatualizado" é obtido a partir da CKTB. Os valores dos sensores deste contexto lógico defeituoso foram preenchidos pelo testador na tarefa Identificar Contextos Lógicos. No **Passo 5** modificamos as situações cópia substituindo o contexto lógico identificado no Passo 2 (original) pelo criado no Passo 4 (com o padrão de defeito). Desta forma definimos as situações "Em casa e conectado - GPS - Dado Desatualizado" e "Em casa e desconectado - GPS - Dado Desatualizado". Mostramos esta transformação na Figura 3.21, onde à esquerda temos a situação original do cenário base escolhido, e à direita temos a aplicação do padrão de defeito Detecção Lenta/Dado Desatualizado no sensor de GPS na situação "Em casa e conectado".

No **Passo 6** geramos os casos de teste de acordo com o mostrado na Figura 3.22. Nesta figura mostramos a criação de um dos cenários com o padrão de defeito usando o

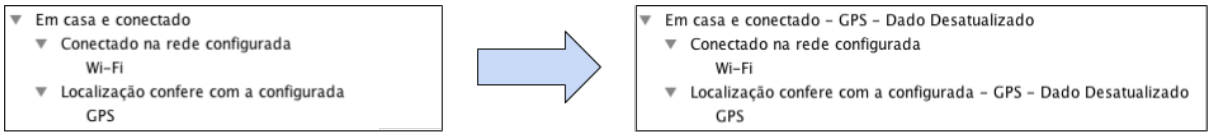

Figura 3.21 Exemplo de transformação de uma situação com o padrão de defeito Detecção Lenta/Dado Desatualizado.

cenário base da aplicação exemplo. À esquerda temos o cenário base, e à direita o cenário derivado após aplicação do padrão de defeito Detecção Lenta/Dado Desatualizado até o tempo 2. Como em todos os tempos do cenário existem situações que podem ser afetadas pelo padrão de defeito, geramos casos de teste com o defeito até os tempos 1, 3 e 4.

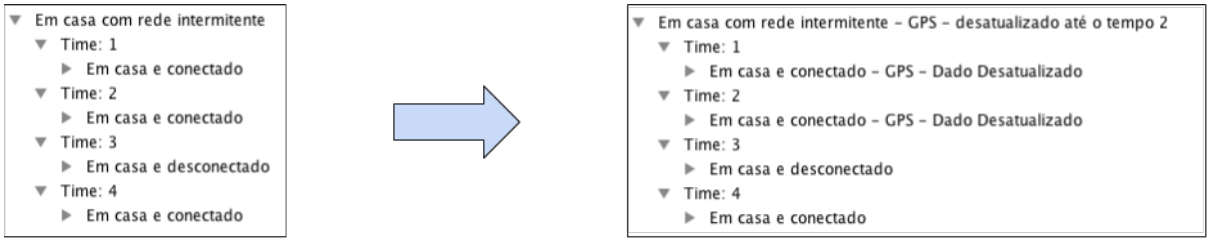

Figura 3.22 Exemplo de transformação do cenário da aplicação exemplo pelo padrão de defeito Detecção Lenta/Dado Desatualizado, até o tempo 2.

3.3.6.2.3 Padrão de Defeito Granularidade Incompatível / Imprecisão

O caso de teste abstrato para este padrão de defeito encontra-se na Tabela 3.2. O caso de teste concreto derivado deste padrão de defeito tem como característica a inclusão de uma situação que reflete uma imprecisão no dado de um sensor que foi lido na situação imediatamente anterior.

Neste padrão de defeito, no **Passo 4** obtemos da CKTB uma versão modificada do contexto lógico $L2$. Este contexto lógico é o $L2\_GRN\_Y$, e nele o dado do sensor Y encontra-se sutilmente diferente em relação ao presente em $L2$. No **Passo 5**, obtemos as situações cópia (B', C' e D') e as modificamos substituindo o contexto lógico $L2$ por $L2\_GRN\_Y$. Desta forma obtemos as situações B_GRN_Y, C_GRN_Y e D_GRN_Y. No **Passo 6** geramos os casos de teste conforme apresentado na Tabela 3.7. Para gerar os casos de teste a partir do cenário base S, adicionamos a situação modificada sempre após a sua ocorrência original, de forma a simular uma leitura imprecisa depois de uma leitura regular do sensor. Para cada cenário com um tempo t que possui uma situação que pode ser afetada, adicionamos uma situação em $t + 1$ com a leitura imprecisa do sensor escolhido. Antes e depois do tempo $t + 1$ o dado do sensor é atualizado como no cenário base S.

Na aplicação exemplo, no **Passo 4** o contexto lógico "Localização confere com a configurada - GPS - Imprecisão" é obtido a partir da CKTB. Os valores dos sensores deste contexto lógico defeituoso foram preenchidos pelo testador na tarefa Identificar

Tabela 3.7 Cenários derivados do cenário base S com o dado do sensor Y impreciso nos tempos compreendidos entre T0 e T4

Cenário	**T0**	**T1**	**T2**	**T3**	**T4**
S	A	B	C	D	
Defeito depois de T1	A	B	B_GRN_Y	C	D
Defeito depois de T2	A	B	C	C_GRN_Y	D
Defeito depois de T3	A	B	C	D	D_GRN_Y

Contextos Lógicos. No **Passo 5** modificamos as situações cópia substituindo o contexto lógico identificado no Passo 2 (original) pelo criado no Passo 4 (com o padrão de defeito). Desta forma definimos as situações "Em casa e conectado - GPS - Imprecisão" e "Em casa e desconectado - GPS - Imprecisão". Mostramos esta transformação na Figura 3.23, onde à esquerda temos a situação original do cenário base escolhido, e à direita temos a aplicação do padrão de defeito Granularidade Incompatível/Imprecisão no sensor de GPS na situação "Em casa e conectado".

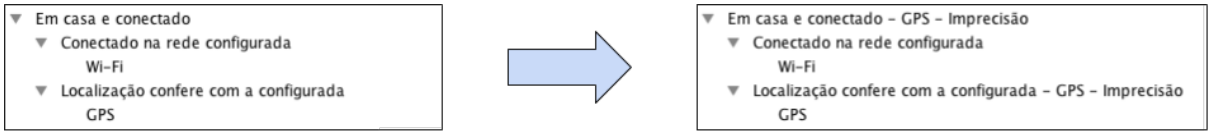

Figura 3.23 Exemplo de transformação de uma situação com o padrão de defeito Granularidade Incompatível/Imprecisão.

No **Passo 6** geramos os casos de teste de acordo com o mostrado na Figura 3.24. Nesta figura mostramos a criação de um dos cenários com o padrão de defeito usando o cenário base da aplicação exemplo. À esquerda temos o cenário base, e à direita o cenário derivado após aplicação do padrão de defeito Granularidade Incompatível/Imprecisão aplicado depois do tempo 1. Como em todos os tempos do cenário existem situações que podem ser afetadas pelo padrão de defeito, geramos casos de teste com o defeito depois dos tempos 2, 3 e 4. Dos três tipos de padrão de defeitos implementados desta categoria, este é o único que aumenta o tamanho, em termos de intervalos de tempo, do cenário afetado pela falha. Os outros dois padrões alteram os cenários apenas substituindo as situações existentes pelas modificadas.

3.3.6.3 Estratégia para Padrões de Defeito em Sensores Não Modelados na Aplicação

Para o padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções, o qual não depende dos sensores presentes no modelo comportamental da aplicação, definimos a seguinte estratégia para gerar os casos de teste:

- **Passo 1:** Listar todas as situações presentes no cenário base S;
- **Passo 2:** Criar um contexto lógico com o evento do sensor Z;
- **Passo 3:** Gerar cópias das situações;

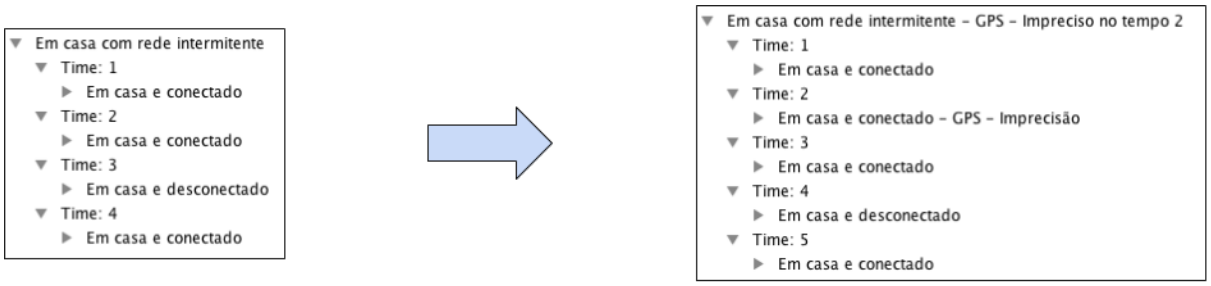

Figura 3.24 Exemplo de transformação de uma situação com o padrão de defeito Granularidade Incompatível/Imprecisão no tempo 2, o qual reflete uma imprecisão no sensor GPS após a leitura ocorrida no tempo 1.

- **Passo 4:** Modificar as cópias das situações adicionando o contexto lógico criado no passo 2;
- **Passo 5:** Gerar os cenários derivados com estas situações de acordo com o padrão de defeito.

3.3.6.3.1 Padrão de Defeito Regra Lógica Problemática / Comportamento Errado Causado por Interrupções

O caso de teste abstrato para este padrão de defeito encontra-se na Tabela 3.3. O caso de teste concreto derivado deste padrão de defeito tem como característica a substituição de uma situação por uma cópia modificada a qual dispara uma interrupção feita pelo dispositivo.

No **Passo 1**, percorremos o cenário base S e listamos as situações A, B, C e D. No **Passo 2**, criamos o contexto lógico que dispara a interrupção no sensor Z: L_INT_Z. No **Passo 3** geramos as cópias das situações, que são A', B', C' e D'. No **Passo 4**, obtemos as situações cópia (A', B', C' e D') e as modificamos adicionando o contexto lógico L_INT_Z. Desta forma, obtemos as situações A_INT_Z, B_INT_Z, C_INT_Z e D_INT_Z. No **Passo 5** geramos os casos de teste conforme apresentado na Tabela 3.8. Para gerar os casos de teste a partir do cenário base S, substituímos as situações originais pelas modificadas cobrindo todas as combinações possíveis do defeito ocorrer em um tempo t dentro do cenário S. No restante dos tempos diferentes de t, os dados de contexto são iguais aos fornecidos pelo cenário base S. Cada cenário derivado deve ter apenas uma situação defeituosa no decorrer do tempo.

Na aplicação exemplo, a execução destes cinco passos acontece da seguinte maneira: No **Passo 1** listamos as duas situações diferentes que estão presentes no cenário. No **Passo 2** criamos o contexto lógico "Bateria com 5% de carga" com o sensor bateria indicando este valor de carga. No **Passo 3** criamos, a partir das situações obtidas no Passo 1, as situações "Em casa e conectado - Cópia" e "Em casa e desconectado - Cópia". No **Passo 4** modificamos as situações cópia adicionando em cada uma o contexto lógico criado no Passo 2 (original). Desta forma definimos as situações "Em casa e conectado - Carga de 5% na Bateria - Comportamento Errado" e "Em casa e desconectado - Carga de

Tabela 3.8 Cenários derivados do cenário base S com o dado de interrupção causada pelo sensor Z nos tempos compreendidos entre T0 e T3.

Cenário	**T0**	**T1**	**T2**	**T3**
S	A	B	C	D
Defeito em T0	A_INT_Z	B	C	D
Defeito em T1	A	B_INT_Z	C	D
Defeito em T2	A	B	C_INT_Z	D
Defeito em T3	A	B	C	D_INT_Z

5% na Bateria - Comportamento Errado". Mostramos esta transformação na Figura 3.25, onde à esquerda temos a situação original obtida do cenário escolhido, e à direita, temos a aplicação do padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções na situação "Em casa e conectado" para indicar o nível de 5% da bateria.

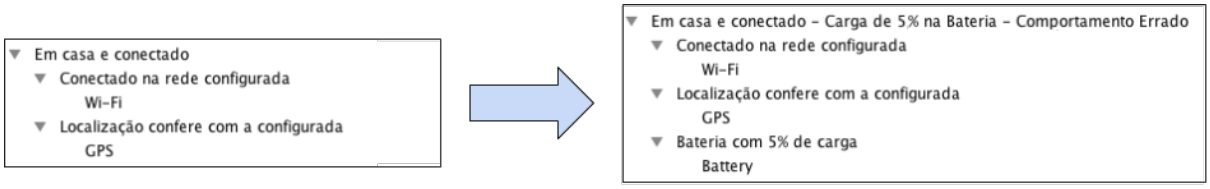

Figura 3.25 Exemplo de transformação de uma situação com o padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções.

No **Passo 5** geramos os casos de teste de acordo com o mostrado na Figura 3.26. Nesta figura mostramos a criação de um dos cenários com o padrão de defeito usando o cenário base da aplicação exemplo. À esquerda temos o cenário base, e à direita temos o cenário derivado após aplicação do padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções para indicar que a bateria atingiu 5% de carga no tempo 1. Como em todos os tempos do cenário existem situações que podem ser alteradas para contemplar o padrão de defeito, geramos casos de teste com o defeito nos tempos 2, 3 e 4.

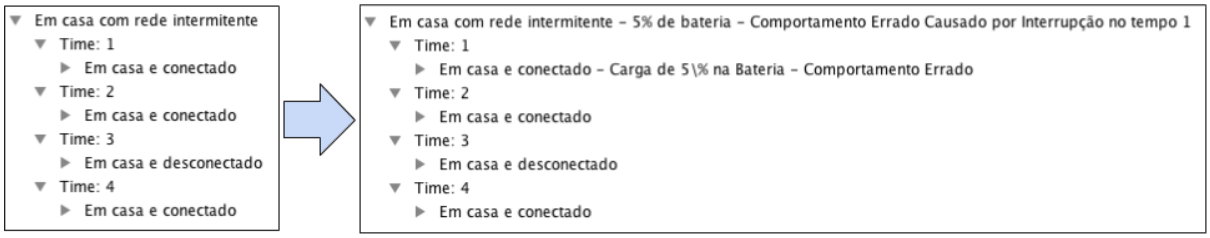

Figura 3.26 À esquerda, o cenário escolhido. À direita, o cenário derivado após aplicação do Padrão de defeito Regra Lógica Problemática/Comportamento Errado Causado por Interrupções para indicar que a bateria atingiu 5% de carga no tempo 1.

3.3.6.4 Aspectos de Implementação do Repositório de Padrões de Defeito

Nesta subseção comentamos alguns aspectos relativos à implementação do repositório de padrões de defeitos. As estratégias explicadas anteriormente foram implementadas nas classes contidas no pacote *org.cemantika.testing.generator.heuristics*. Todas as classes implementam a *interface SensorDefectPatternHeuristic*, a qual especifica o método *deriveTestCases*. Na Listagem 3.2 temos a definição da interface em questão, cujo método especificado recebe os seguintes parâmetros: o cenário base, o sensor afetado pela falha e o padrão de defeito que deve ser aplicado no sensor dentro do cenário base.

Listagem 3.2 Interface que define a implementação do repositório de Padrões de Defeitos.

```
1 public interface SensorDefectPatternHeuristic {
2   List<Scenario> deriveTestCases(Scenario baseScenario,
        PhysicalContext sensor, ContextDefectPattern
        contextDefectPattern);
3 }
```

O método apresentado na Listagem 3.3 implementa a estratégia de geração de casos de teste para o padrão de defeito Incompleto/Indisponível. No método em questão temos no final uma chamada para o método responsável pela redução no conjunto de casos de teste.

Listagem 3.3 Método que implenta o padrão de defeito Incompleto/Indisponível

```
1  @Override
2    public List<Scenario> deriveTestCases(Scenario baseScenario,
         PhysicalContext sensor, ContextDefectPattern
         contextDefectPattern) {
3
4    List<Scenario> scenarios = new ArrayList<Scenario>();
5
6    //generate all timeslots with defect
7    List<TimeSlot> timeSlotsWithIncompleteUnavailable =
         getTimesLotsWithIncompleteUnavailable(baseScenario, sensor,
         contextDefectPattern);
8
9    //derive using always first available and others disabled for
         sensor
10   if (timeSlotsWithIncompleteUnavailable.size() > 0)
11     scenarios.addAll(deriveScenariosWithtimeSlotsWith
           IncompleteUnavailable(baseScenario,
           timeSlotsWithIncompleteUnavailable, sensor));
12
13   return TestSuiteReduction.reducedTestSuite(scenarios);
14 }
```

3.3.6.5 Observações Finais dos Padrões de Defeitos Presentes no Repositório

Dos padrões de defeitos selecionados, Incompleto/Indisponível, Detecção Lenta/Dado Desatualizado e Granularidade Incompatível/Imprecisão são da categoria que depende de um sensor identificado no modelo da aplicação, enquanto que Regra Lógica Problemática/ Comportamento Errado Causado por Interrupções independe da existência de um sensor no modelo da aplicação.

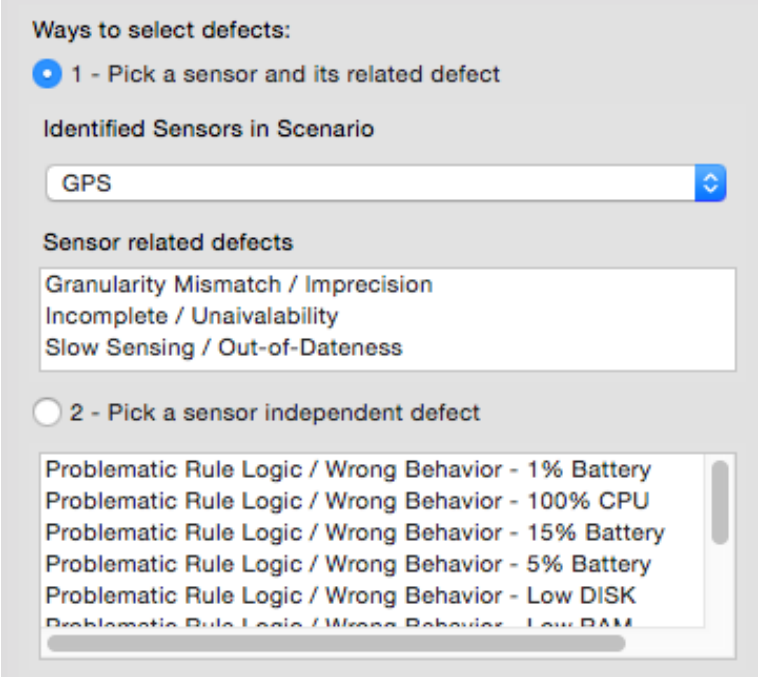

Figura 3.27 Escolhendo padrões de defeitos para gerar os casos de teste na ferramenta CEManTIKA CASE.

Devido a esta diferença, precisamos adotar duas abordagens para gerar os casos de teste. Além disso, o modo como o testador seleciona estes padrões de defeitos também sofre influência desta diferença. A Figura 3.27 mostra a necessidade de escolha de um sensor pelo testador na interface do CEManTIKA CASE como a opção 1, e na opção 2 a escolha direta de um defeito que independe do sensor mapeado na aplicação. Os defeitos listados na opção 2 são alguns dos suportados pela implementação do Context Simulator, sendo que em (Vieira, Holl e Hassel, 2015), a entrada destes defeitos é feita de forma livre e manual, enquanto que implementação fixamos os valores para os sensores (por exemplo, bateria a 5%).

Após aplicar os padrões de defeitos escolhidos no cenário base selecionado, a implementação, antes de exportar para o formato final dos casos de teste, realiza a redução no conjunto de cenários gerados. Explicamos o funcionamento do redutor de casos de teste na Subseção 3.2.5 e explicamos na subseção a seguir a sua implementação.

3.3.7 Redutor de casos de teste

O redutor de casos de casos de teste foi implementado na classe *TestSuiteReduction*. Para calcular a distância entre 2 cenários, utilizamos o índice de Jaccard. Na Listagem 3.4 temos a implementação desta função de distância.

Listagem 3.4 Método que calcula o índice de Jaccard entre 2 cenários.

```
1  private static double getJaccardIndex(Scenario scenarioA, Scenario
       scenarioB) {
2
3    List<LogicalContext> transitionsA = scenarioA.getTransitions();
4    List<LogicalContext> transitionsB = scenarioB.getTransitions();
5
6    Set<LogicalContext> union = new
         HashSet<LogicalContext>(transitionsA);
7    Set<LogicalContext> intersection = new
         HashSet<LogicalContext>(transitionsA);
8
9    union.addAll(transitionsB);
10
11   intersection.retainAll(transitionsB);
12
13   return intersection.size() / union.size();
14
15 }
```

Ao calcular os índices de todos os cenários dentro do conjunto de casos de teste, preenchemos a matriz de similaridade e depois procedemos com a exclusão dos cenários mais similares entre si, como mostra o trecho presente na Listagem 3.5. O cenário de maior tamanho é excluído. Quando os dois cenários possuem o mesmo tamanho escolhe-se um aleatoriamente para exclusão.

Listagem 3.5 Trecho de método que realiza a remoção dos cenários que são similares entre si.

```
1  similarityMatrix[pair[0]] [pair[1]] = 0;
2
3  Scenario scenario1 = scenarios.get(pair[0]);
4  Scenario scenario2 = scenarios.get(pair[1]);
5  Scenario firstChoice = null, secondChoice = null;
6
7  if(scenario1.getTransitions().size() <
       scenario2.getTransitions().size()){
8    firstChoice = scenario2;
9    secondChoice = scenario1;
10 }else if (scenario1.getTransitions().size() >
       scenario2.getTransitions().size()){
11   firstChoice = scenario1;
12   secondChoice = scenario2;
```

```
13 }else{
14   Random random = new Random();
15   int firstChoiceIndex = random.nextInt(2);
16   firstChoice = (firstChoiceIndex == 0)? scenario1 : scenario2;
17   secondChoice = (firstChoiceIndex == 0)? scenario2 : scenario1;
18 }
19
20 int removedTestCaseIndex = -1;
21 if(firstChoice == scenario1)
22   removedTestCaseIndex = pair[0];
23 else{
24   removedTestCaseIndex = pair[1];
25 }
26
27 for(int i = 0 ; i < scenarios.size() ; i++){
28   similarityMatrix[removedTestCaseIndex][i] = 0;
29   similarityMatrix[i][removedTestCaseIndex] = 0;
30 }
31
32 reducedTestSuite.add(firstChoice);
```

3.3.8 Exportação de Casos de Teste

Para que os dados sejam exportados para o Context Simulator, precisamos implementar a classe *TestSuite*, a qual armazena uma lista de cenários. Apresentamos a classe na Listagem 3.6. Esta classe foi implementada tanto no CEManTIKA CASE quanto no Context Simulator. Esta classe é a raiz do arquivo JSON gerado para uso no Context Simulator, conforme mostrado na Listagem 3.7.

Listagem 3.6 Classe TestSuite. Esta classe é responsável por armazenar os dados de contexto de todos os cenários gerados após a solicitação do testador.

```
1  public class TestSuite {
2    private List<AbstractContext> testCases = new
         ArrayList<AbstractContext>();
3
4    public void setTestCases(List<AbstractContext> testCases) {
5      this.testCases = testCases;
6    }
7
8    public List<AbstractContext> getTestCases() {
9      return testCases;
10   }
11 }
```

Listagem 3.7 Arquivo JSON contendo os casos de teste gerados pelo CTC.

```
{
  "testCases":[
    {
      "type":"Scenario",
      "lstContext":[
        {
          "type":"TimeSlot",
          "id":0,
          "lstContext":[
            {
              "type":"Situation",
              "expectedBehavior":"Turn file upload available; ",
              "lstContext":[
                {
                  "type":"LogicalContext",
                  "lstContext":[
                    {
                      "type":"WiFi", "value1":true,
                      "value2":"Home",
                      "name":"Wi-Fi"
                    }
                  ],
                  "name":"Conectado na rede configurada"
                }
              ],
              "name":"Em casa e conectado"
            }
          ],
          "name":"Time: 0"
        }
      ],
      "name":"Em casa com rede intermitente"
    }
  ]
}
```

3.3.9 Interoperabilidade com o Context Simulator

Embora haja compatibilidade na abstração dos dados de contexto entre o gerador de casos de teste CTC e o Context Simulator, não existe um formato de arquivo que garanta a interoperabilidade entre ambos. Outra característica do Context Simulator que limita a ação do testador é a ausência da visualização do comportamento esperado da aplicação numa dada situação. Para resolver ambas limitações precisamos modificar o Context Simulator.

Mostramos o resultado destas implementações na Figura 3.28, que ilustra a versão

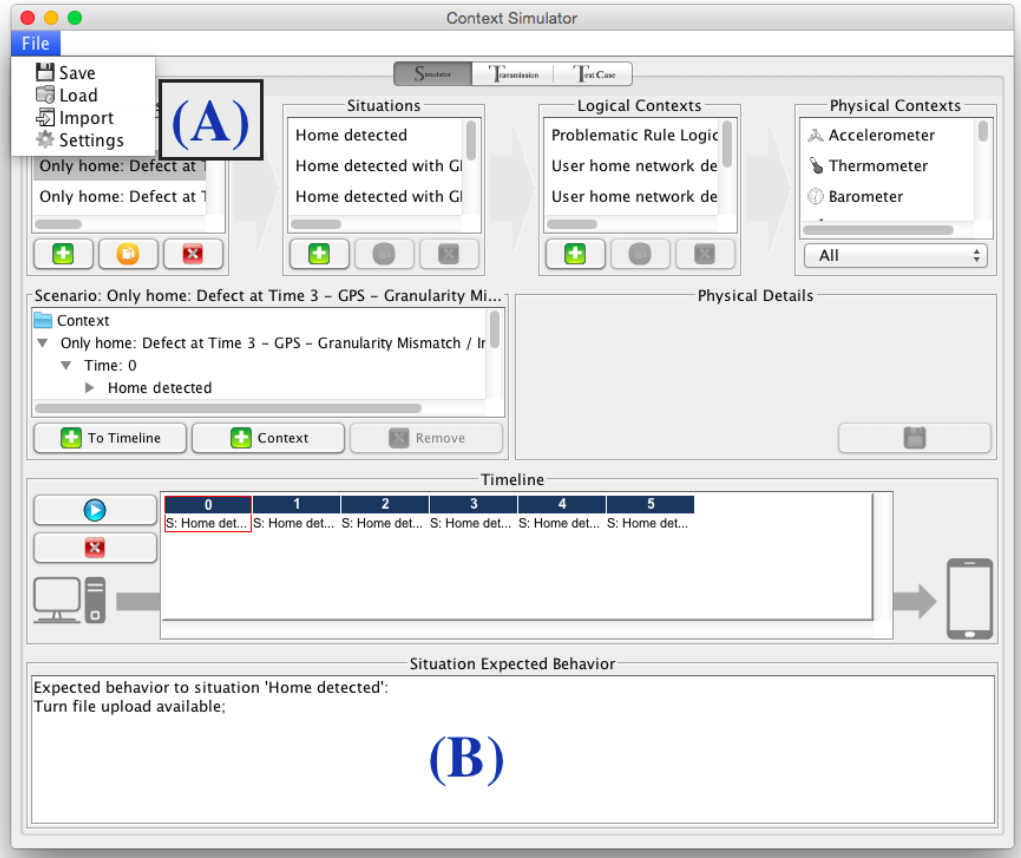

Figura 3.28 Implementação da importação de casos de teste (A) e visualização de comportamento esperado (B) no Context Simulator

modificada do Context Simulator. Uma delas serve para permitir a interoperabilidade entre geradores de casos de teste e o Context Simulator, através da importação de uma suíte de testes contendo uma lista de cenários (A). Esta implementação fez uso do formato JSON, conforme mostrado na Listagem 3.7. Com isto tornou-se possível a geração de casos de teste a partir de ferramentas externas ao Context Simulator.

Para ajudar o testador na detecção dos defeitos na aplicação móvel sob teste, implementamos a visualização do comportamento esperado quando as informações de contexto de uma determinada situação são transmitidas para a aplicação sob teste (B).

3.4 EXECUÇÃO DO EXEMPLO DE USO

Para ilustrar o uso do método CTC vamos exibir a execução do mesmo na aplicação exemplo. Os modelos estrutural (mostrado no diagrama conceitual da Figura 3.14) e comportamental (mostrado no grafo contextual da Figura 3.18) de contexto desta aplicação já foram especificados na etapa de projeto da aplicação. Nesta seção vemos a execução

ilustrada das tarefas definidas no método proposto. Ao final, mostramos como o testador pode detectar defeitos no aplicativo desenvolvido.

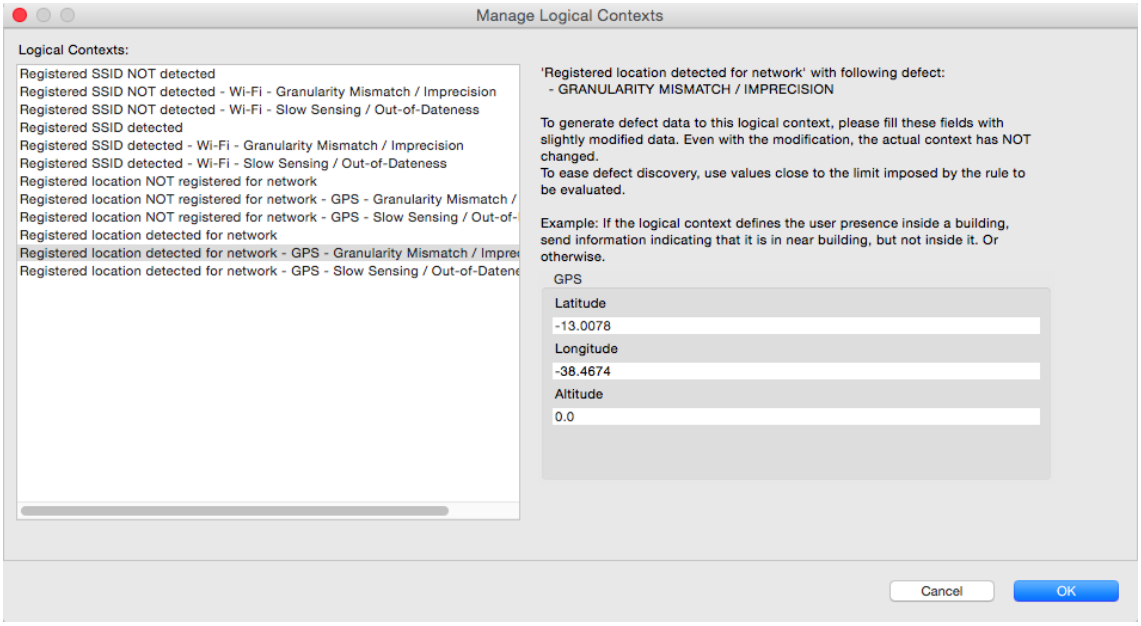

Figura 3.29 Execução da tarefa *Identificar Contextos Lógicos* na aplicação *PersonalUploadManager*. Destaque para a definição do contexto de localização registrada na aplicação com o padrão de defeito Granularidade Incompatível/Imprecisão no sensor de GPS.

Para a geração dos casos de teste o primeiro passo é acessar a CKTB. Depois, é feita a identificação dos contextos lógicos, como mostrado na Figura 3.29. Nesta tarefa informamos os valores dos sensores para cada contexto lógico, inclusive para os casos em que os sensores apresentam falhas.

Em seguida, identificamos as situações e seus comportamentos esperados, como mostrado na Figura 3.30. Note que o comportamento esperado inicialmente é carregado com o valor da ação correspondente ao definido no grafo contextual da aplicação. O testador pode editar o nome da situação e adicionar informações ao campo de texto do comportamento esperado da situação.

Após identificar as situações, geramos então os cenários base, conforme exibido na Figura 3.31. Na referida figura definimos um cenário onde o usuário permanece com uma conexão considerada boa durante todo o tempo. Nesta tarefa o testador pode incluir, alterar e excluir cenários de acordo com a necessidade, e também alterar a ordem das situações dentro da linha do tempo do cenário.

A última etapa da implementação é a geração dos casos de teste. Aqui é o ponto em que executamos a tarefa *Combinar Cenários Base com Padrões de Defeitos* (Figura 3.32). Iniciamos a tarefa ao selecionar um cenário base. Depois, podemos escolher os padrões de defeitos para aplicar ao cenário selecionado. Ao final, solicitamos a geração dos casos de teste, e executamos a tarefa *Exportar Casos de Teste Gerados para Execução*. Antes de realizar a exportação final dos casos de teste, a ferramenta executa o redutor de casos

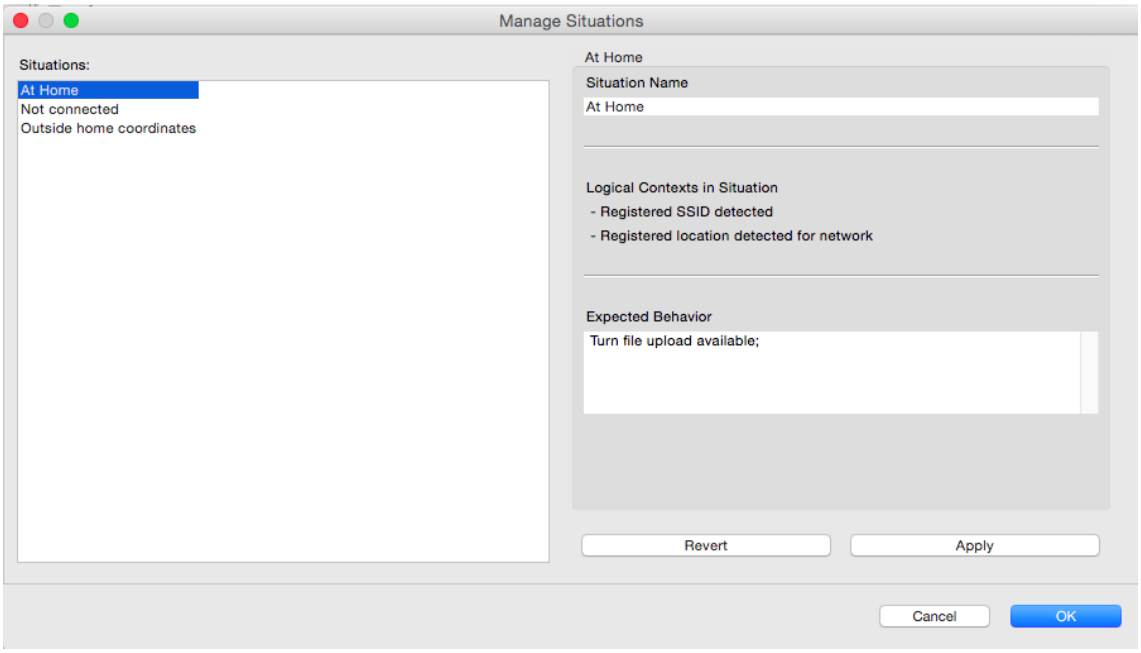

Figura 3.30 Execução da tarefa *Identificar Situações* na aplicação *PersonalUploadManager*. Destaque para definição da situação onde o usuário se encontra na localização e com conectividade coincidentes com o configurado na aplicação.

de teste para diminuir o número de casos de teste a serem exportados. A Figura 3.32 mostra a tela onde ocorrem estes passos. Neste caso combinamos o cenário apenas com o padrão de defeito Granularidade Incompatível/Imprecisão sobre o sensor de GPS.

Com os casos de teste gerados e com a aplicação móvel desenvolvida, procedemos agora com a execução dos testes sobre a mesma. Para executar os testes, configuramos uma localização (latitude entre -13.0080 e -13.0090, longitude entre -38.4670 e -38.4690) e a configuração de rede Wifi com *status* disponível e SSID igual a *Home* para ser considerada pela aplicação como uma conexão boa. Em seguida executamos o simulador de contexto, importamos os casos de teste gerados e então iniciamos a aplicação no emulador Android. A partir de então executamos os casos de teste para descobrir as falhas na adaptação ao contexto. A Figura 3.33 mostra a aplicação funcionando corretamente no tempo 1. No tempo 2 simulamos uma coordenada de GPS imprecisa, como mostrado na Figura 3.34. Note que na Figura 3.34 o comportamento esperado pela aplicação de acordo com a situação é diferente do comportamento obtido ao simular o dado de contexto, o que caracteriza um defeito na aplicação sob teste. Em ambos os casos o usuário está fisicamente no mesmo lugar, que é considerado, em condições normais, como coincidente com a localização configurada na aplicação. No entanto, devido à imprecisão do GPS no tempo 2, a qual deveria ser contornada pela aplicação (pois está em sua especificação), o contexto inferido pela aplicação é de que o usuário está fora da localização configurada na aplicação.

Desta forma temos uma visão do método sendo executado na prática. O exemplo descrito nesta seção foi utilizado como treinamento para a execução do estudo empírico

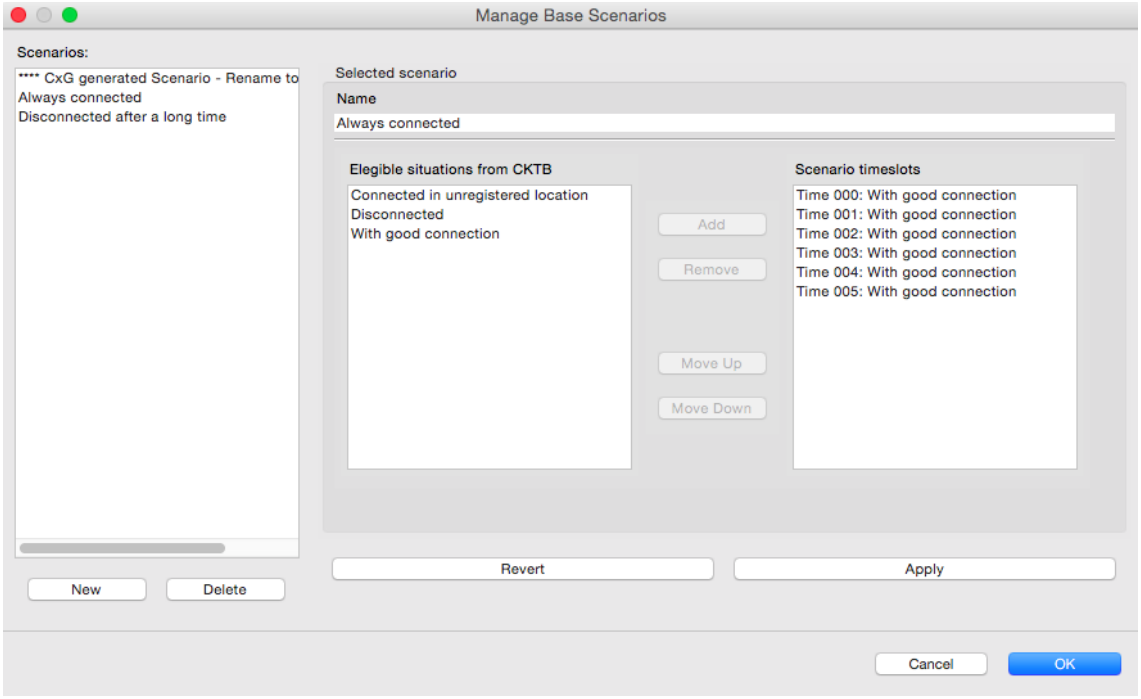

Figura 3.31 Execução da tarefa *Elaborar Cenários Base* para a aplicação *PersonalUploadManager*. Aqui o testador define um cenário onde o usuário se encontra com localização e conectividade coincidentes com o configurado na aplicação durante todo o tempo.

que serviu para avaliar o método proposto por este trabalho.

3.5 RESUMO DO CAPÍTULO

Neste capítulo apresentamos o desenvolvimento do método CEManTIKA Test Creator. Nele temos a definição da arquitetura da solução para a geração de casos de teste, além da especificação das tarefas necessárias para alcançar este objetivo. Para ilustrar estas partes do capítulo apresentamos a aplicação exemplo *PersonalUploadManager*.

Após apresentar a arquitetura e as tarefas envolvidas na geração de casos de teste para aplicações móveis, mostramos como foi feita a implementação do protótipo da proposta nas ferramentas CEManTIKA CASE e no Context Simulator. Além disso, definimos estratégias para geração de casos de teste para quatro padrões de defeitos em fontes de contexto.

Ao final mostramos um exemplo prático da solução desenvolvida em execução no protótipo. Com a aplicação modelada geramos os casos de teste seguindo as orientações contidas nas tarefas que culminam na geração dos casos de teste. Após gerar os casos de teste executamos a aplicação desenvolvida e simulamos dados de contexto de forma a descobrir defeitos na adaptação ao contexto, como foi mostrado na última parte da Seção 3.4. Este exemplo prático foi aproveitado no treinamento realizado no estudo empírico descrito no próximo capítulo.

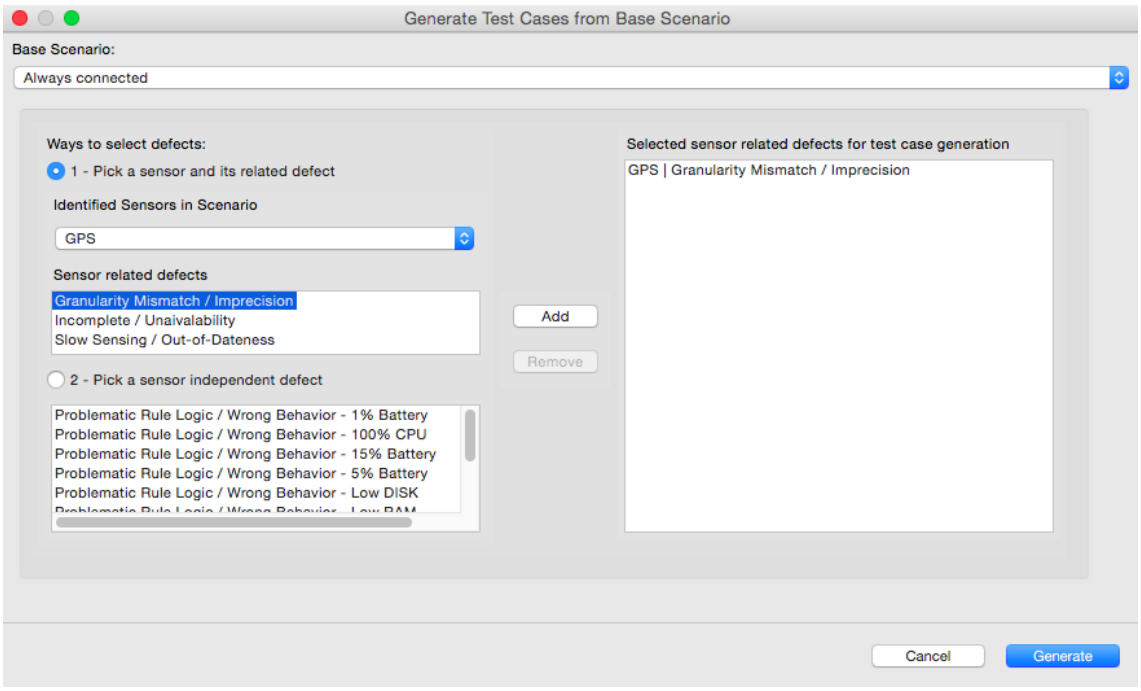

Figura 3.32 Execução da tarefa *Combinar Cenários Base com Padrões de Defeitos* na aplicação *PersonalUploadManager*. O testador combina o cenário em que o usuário está com o contexto que libera a funcionalidade de enviar dados com o padrão de defeito Granularidade Incompatível/Imprecisão no sensor de GPS.

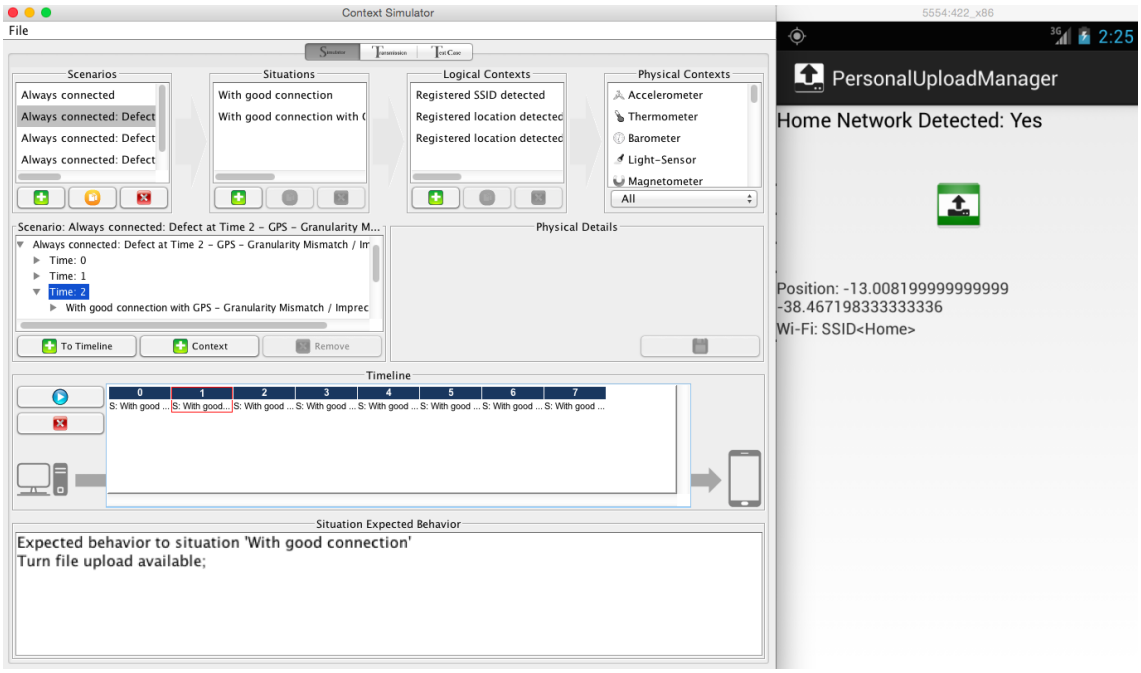

Figura 3.33 Aplicação sob teste funcionando corretamente com o contexto do usuário coincidindo com localização e conectividade configuradas na aplicação.

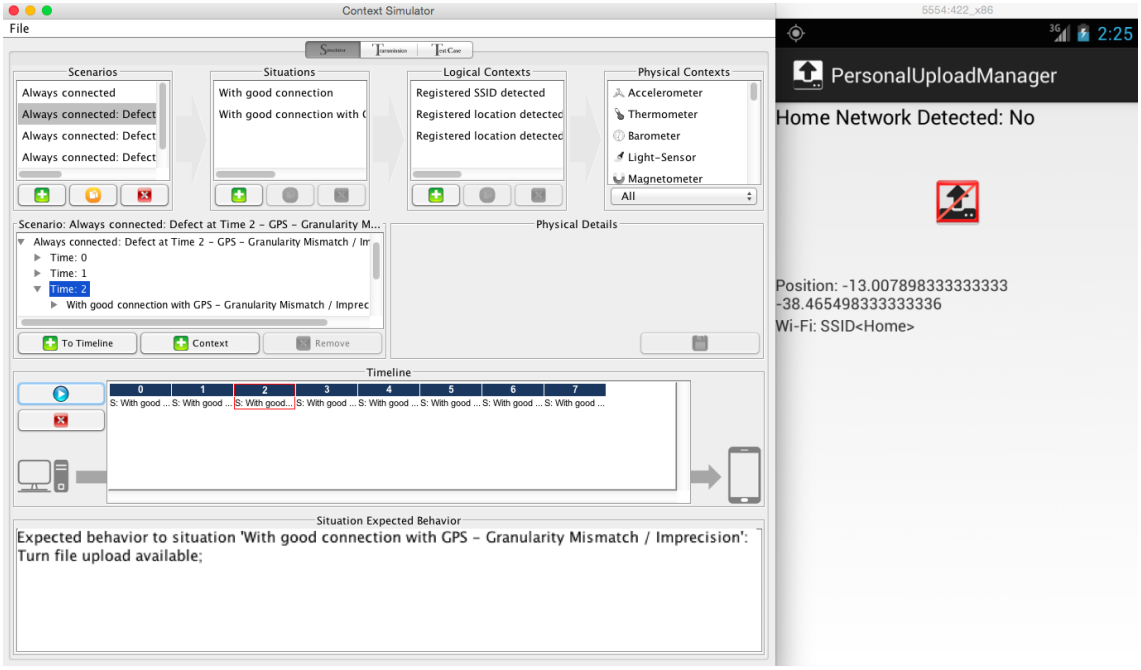

Figura 3.34 Aplicação sob teste com defeito descoberto após leitura de coordenada de GPS imprecisa presente no caso de teste executado.

Capítulo

4

ESTUDO EMPÍRICO

Para verificar a aplicabilidade da proposta, nós realizamos um estudo empírico com desenvolvedores e testadores de aplicativos móveis. A avaliação foi planejada de forma que os participantes do estudo tivessem a oportunidade de usar a implementação da ferramenta para a elaboração de casos de teste para um aplicativo móvel. Após a execução desta atividade pelos participantes, coletamos o seu *feedback*. Para projetar, executar e documentar o estudo empírico utilizamos as práticas recomendadas por (Shull, Singer e Sjøberg, 2007). De forma semelhante ao realizado no estudo experimental em (Brito Junior, 2015) combinamos estas práticas com os métodos qualitativos propostos por (Seaman, 1999), (Cruzes e Dyba, 2011) e (Saldana, 2012).

4.1 OBJETIVOS

A principal meta desta avaliação é verificar se o uso do CEManTIKA Test Creator pode ajudar desenvolvedores e testadores a detectar defeitos de fontes de contexto em aplicativos móveis. Para tanto, usamos o método Meta/Questão/Métrica (Goal/Question/Metric) (Basili, Caldiera e Rombach, 1994), o qual define a seguinte meta:
Analisar o CEManTIKA Test Creator
Com o propósito de Avaliar seu uso
Em relação a utilidade na geração de casos de teste
A partir do ponto de vista de desenvolvedores e testadores de aplicativos móveis

Neste caso, consideramos as métricas de cobertura e agilidade para qualificar o alcance da meta. Desejamos aferir se o CEManTIKA Test Creator é útil para o testador de aplicativos móveis na geração de casos de teste para detecção de defeitos ligados a fontes de contexto levando em consideração a percepção do testador.

4.2 MATERIAIS DO ESTUDO EMPÍRICO

Organizamos a avaliação de modo que a mesma tenha 6 etapas, e cada uma tenha um material específico. Cada etapa possui também um objetivo. Depois de detalhar os

materiais utilizados no estudo empírico, abordamos o processo de criação e refinamento dos mesmos através da realização de estudos piloto.

4.2.1 Detalhamento dos Materiais Elaborados

Para cada uma das 6 etapas do estudo empírico temos um material e uma meta. Ilustramos a relação entre estes componentes através da Tabela 4.1.

Tabela 4.1 Materiais utilizados em cada etapa do estudo empírico

Etapa	Passos do Estudo	Meta
I	Questionário *online*	Identificar perfil do participante
II	Explanação dos conceitos envolvidos no estudo empírico (30 minutos)	Reforçar a contextualização da pesquisa
III	Treinamento (30-40 minutos)	Tornar familiar o uso da ferramenta desenvolvida
IV	Execução do estudo empírico (60 minutos)	Avaliar a execução da CEManTIKA Test Creator pelos participantes
V	Gravação da entrevista (10-30 minutos por participante)	Aferir as impressões do participante no uso de CEManTIKA Test Creator
VI	Transcrição, codificação e interpretação	Analisar as impressões do participante no uso de CEManTIKA Test Creator

Na Etapa I solicitamos o preenchimento de um questionário *online* para identificar o perfil do participante. Na segunda etapa realizamos presencialmente a explicação dos conceitos relacionados ao método proposto, de forma a serem compreendidos. Depois desta apresentação, realizamos um treinamento para uso da ferramenta na geração de casos de teste para a aplicação *PersonalUploadManager*, de forma a preparar o participante para a execução do estudo empírico propriamente dito. Na quarta etapa solicitamos que o participante gerasse e executasse os casos de teste para outra aplicação, a *NoCamInMeeting*, cujo objetivo é gerenciar o acesso à câmera do dispositivo de acordo com o contexto do usuário, de forma que o usuário não consiga realizar gravações com a câmera do dispositivo durante uma reunião. Ao fim do estudo empírico, realizamos uma entrevista com o participante para obter suas impressões sobre a cobertura e agilidade do CEManTIKA Test Creator na geração de casos de teste. Estas entrevistas foram feitas presencialmente de acordo com a disponibilidade de cada participante. Com as entrevistas gravadas, fizemos a transcrição, codificação e interpretação para analisar as impressões passadas pelo participante. A seguir mostramos a construção dos materiais descritos acima.

4.2.2 Execução de Estudos Piloto para Elaboração dos Materiais

Para chegar aos materiais no estado final utilizado no estudo empírico, realizamos três estudos piloto com pessoas integrantes do grupo de pesquisa CEManTIKA, de forma voluntária. Dentro deste grupo selecionamos 3 voluntários que possuem experiência com

desenvolvimento e/ou teste em aplicativos móveis. Por ser um grupo de pesquisa em contexto computacional, estes participantes possuem um conhecimento mais específico sobre o que é contexto.

No primeiro piloto, coletamos o *feedback* do participante relativo às perguntas do Questionário *Online* e da apresentação dos conceitos envolvidos no estudo empírico. Além disso, tivemos diversos problemas na configuração do ambiente do estudo empírico, desde questões relativas à instalação do Java (para execução da IDE Eclipse) até problemas no emulador Android. A partir deste primeiro retorno, ajustamos o Questionário *Online* para o formato final, ajustamos alguns pontos da apresentação e também criamos um pacote para execução das ferramentas com poucos passos. Neste primeiro piloto não realizamos nem a execução da ferramenta nem a coleta dos dados.

No segundo piloto, conseguimos realizar o estudo empírico completo em todas as suas etapas. No entanto, verificamos que algumas perguntas da entrevista precisariam ser modificadas. Além disso, verificamos que faltou um guia com o passo a passo do método para sua execução pelo participante. Com estes retornos, ajustamos as perguntas da entrevista para o formato final e criamos o guia de uso do método, o qual está no Anexo C. Este guia, que foi impresso, contém uma explicação geral da finalidade do estudo empírico, procedimentos pré-estudo, verificação do ambiente de instalação, além do passo a passo a ser seguido na execução do estudo empírico.

No terceiro piloto, não conseguimos instalar o ambiente na máquina do participante. Neste caso cedemos uma máquina para a execução do mesmo pelo participante. O participante ainda mencionou que o formulário de consentimento estava inadequado e que precisava ser refeito. Além disso, a apresentação precisou de ajustes pois não abarcava de forma didática os conceitos. Com este retorno e com os problemas que ocorreram, realizamos as seguintes adequações: Ajuste no pacote de instalação para conter tudo o que é necessário para a execução da ferramenta implementada; Reformulação do formulário de consentimento; e ajustes na apresentação dos conceitos.

Durante os 3 estudos piloto a ferramenta implementada foi de fato testada, e os participantes descobriram erros e apontaram melhorias simples a serem aplicadas, como o uso de máscaras em campos de data e numéricos. Em nenhum dos 3 estudos piloto realizamos as etapas pós-entrevista, como a análise dos dados obtidos. Após a realização do terceiro piloto os documentos presentes no Anexo C chegaram ao seu estado final.

Os participantes desta etapa não foram convidados a participar do estudo empírico propriamente dito, assim como o restante das pessoas que pertencem ao grupo CEMan-TIKA, pois desejamos evitar o enviesamento na avaliação da proposta apresentada neste trabalho, a qual já havia sido apresentada aos integrantes do referido grupo de pesquisa.

4.3 TAREFAS

A primeira tarefa compreende o preenchimento do questionário *online* enviado para o participante. As questões do questionário estão dispostas conforme a Tabela 4.2. As questões entre Q1 e Q5 servem para identificar de maneira geral o participante. As questões Q6 e Q7 qualificam a experiência do participante no processo de desenvolvimento de aplicativos móveis. O restante das questões verificam como o participante encara os

problemas existentes na área teste em aplicativos móveis.

Tabela 4.2 Questionário *online* para caracterizar perfil do participante

#	Questão
Q01	Informe seu nome
Q02	Informe sua data de nascimento
Q03	Informe seu e-mail
Q04	Qual a sua formação acadêmica?
Q05	Sexo
Q06	Quanto tempo de experiência na área de desenvolvimento/teste de aplicativos móveis você tem?
Q07	Você já participou de quais etapas no processo de desenvolvimento de aplicativos móveis?
Q08	Você já utilizou modelos de representação de contexto no desenvolvimento de aplicativos móveis? Se sim, estes modelos foram utilizados na etapa de testes?
Q09	Você utiliza alguma ferramenta para testar aplicativos móveis? Quais?
Q10	Você utiliza alguma técnica ou ferramenta para gerar dados de entrada para sensores em aplicativos móveis? Quais?
Q11	Você já utilizou alguma estratégia para investigar padrões de defeitos em sensores (Ex de técnica: checklists de possíveis problemas em sensores)?

Após responder ao questionário *online*, o participante, já no ambiente de execução do estudo empírico, assiste à apresentação dos conceitos envolvidos, das ferramentas que serão utilizadas e dos objetivos que a proposta visa alcançar. Esta apresentação tem duração aproximada de 30 minutos.

Após isto, realizamos um treinamento de aproximadamente 30-40 minutos, onde exploramos os artefatos relacionados à aplicação utilizada como alvo dos testes: *PersonalUploadManager*. Apresentamos então os artefatos de requisitos e de projeto conceitual e comportamental de contexto desta aplicação. Estes artefatos foram gerados anteriormente dentro da ferramenta CEManTIKA CASE. Após a navegação dos artefatos, iniciamos a exposição da etapa de elaboração de casos de teste. Após a elaboração e exportação dos casos de teste, realizamos uma execução que revele um defeito e então orientamos a forma de preencher o formulário de defeitos.

Após o treinamento solicitamos a execução do estudo empírico pelos participantes, aplicando assim o método de geração de casos de teste no aplicativo *NoCamInMeeting*. Ao fim de 60 minutos encerramos a execução do estudo empírico e recolhemos os dados como os casos de teste gerados e os formulários de defeitos encontrados.

Logo após a execução do estudo empírico realizamos as entrevistas, ou então as agendamos de acordo com a disponibilidade do participante. A entrevista em questão é composta por 10 perguntas abertas, as quais estão listadas na Tabela 4.3. As perguntas seguem este formato mais livre para que a entrevista consiga captar informações que não seriam obtidas com uso de perguntas de escopo bem delimitado, como observado em (Seaman, 1999). Este tipo de pergunta num estudo qualitativo evita que o entrevistador

Tabela 4.3 Orientações para a entrevista pós execução

#	Questão	Tipo de informação esperada
1	O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?	Atendimento ao propósito
2	O que você gostou no uso da ferramenta CEManTIKA Test Creator?	Pontos fortes
3	O que você não gostou no uso do CEManTIKA Test Creator?	Pontos fracos
4	Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?	Utilidade
5	Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?	Ameaças
6	Quais são suas sugestões para mitigar estes riscos?	Possíveis melhorias
7	O que você acha sobre o conjunto de casos de teste gerado?	Atendimento ao propósito
8	Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?	Utilidade
9	O que você acha sobre o uso de modelos da aplicação pelo método proposto?	Ameaças
10	Você gostaria de mencionar algo mais?	Considerações finais

possa dar um possível direcionamento à entrevista. O objetivo das perguntas é obter as impressões do participante no uso do método. Estas impressões, de acordo com as perguntas, podem representar os seguintes temas: atendimento ao propósito, pontos fortes, pontos fracos, cobertura e agilidade na geração de casos de teste na solução experimentada e possíveis melhorias que podem ser aplicadas à mesma.

4.4 PARTICIPANTES

O método proposto tem como público-alvo principal testadores de aplicativos móveis. Além destes, podemos considerar também como púbico-alvo desenvolvedores de aplicativos móveis, profissionais ou não. Portanto, selecionamos e convidamos pessoas que se encaixem neste perfil, como estudantes, pesquisadores e profissionais que já atuaram em algum momento no desenvolvimento e/ou teste de aplicativos móveis.

Um total de 5 pessoas participaram do estudo empírico. Embora todos sejam do sexo masculino, o perfil destes participantes é bem diverso, pois engloba profissionais da indústria e de institutos de pesquisa. Outros fatores que caracterizam a divergência são a área de atuação profissional nos projetos de aplicativos móveis e a experiência com

ferramentas de teste nesta classe de aplicativo. A Tabela 4.4 exibe os dados que são significativos na caracterização dos participantes.

Todos os participantes afirmam que já realizaram alguma atividade de teste em aplicativos móveis, seja na criação dos casos de teste e/ou na execução dos mesmos. Quando indagados sobre as ferramentas que utilizam para testar sensores em aplicativos móveis, os participantes 1, 2 e 3 afirmaram que utilizam o aplicativo Fake GPS para forjar coordenadas GPS, enquanto que o participante 5 utiliza o terminal do emulador Android para executar os casos de teste que envolvam fontes de contexto.

Tabela 4.4 Características dos participantes que participaram do estudo empírico.

Participante	**1**	**2**	**3**	**4**	**5**
Idade	24	25	27	29	37
Sexo	Masculino	Masculino	Masculino	Masculino	Masculino
Formação Acadêmica	Superior Completo	Superior Completo	Superior Completo	Pós-Graduado	Pós-Graduado
Ocupação Profissional	Analista de Sistemas	Engenheiro de Software	Analista de Sistemas	Analista de Qualidade	Analista de Sistemas
Experiência com aplicativos móveis	0-2 anos	2-4 anos	2-4 anos	0-2 anos	>4 anos
Atividade de teste móvel	Criação de CT	Criação de CT	Criação e execução de CT	Criação e execução de CT	Criação e execução de CT

De acordo com o nível de experiência dos participantes no âmbito do processo de desenvolvimento de aplicativos móveis, a explicação de 30 minutos pode ser ajustada para enfatizar conceitos que sejam de pouca familiaridade para o participante.

Para os participantes que participaram do estudo empírico, esta é a primeira abordagem que eles já viram que visa a geração de casos de teste para padrões de defeitos em fontes de contexto (todos os participantes responderam não à pergunta Q11).

4.5 PROJETO DO ESTUDO EMPÍRICO

Nesta seção apresentamos o projeto do estudo empírico, ou seja, como as informações coletadas devem ser tratadas. As entrevistas com os participantes foram gravadas e transcritas. O próximo passo é a realização do processo de codificação das mesmas. Este processo compreende a criação de uma sentença que resume da forma mais sucinta possível a informação contida em uma fala do entrevistado. Após a codificação, agrupamos os códigos encontrados em categorias de acordo com as similaridades entre os códigos.

Além da entrevista, os formulários de defeitos encontrados, os casos de teste gerados e a base de conhecimento de contexto contendo os dados preenchidos pelos participantes na ferramenta também são coletados. Desta forma, nós podemos observar se os participantes conseguiram gerar os casos de teste e se os participantes descobriram defeitos presentes na aplicação sob teste. Em outras palavras, podemos confirmar se os participantes conseguiram executar os passos do método de geração de casos de teste conforme

esperado. As ocorrências relacionadas ao uso do método durante o estudo empírico serão detalhadas na Seção 4.7.

4.6 PROCEDIMENTO DE ANÁLISE

Nesta seção descrevemos como foi realizada a análise das informações coletadas no estudo empírico. Com a coleta das bases de conhecimento criadas e dos formulários de defeitos encontrados, verificamos que todos os participantes conseguiram executar as atividades solicitadas, com um número em torno de 7-60 cenários gerados e 2-7 defeitos de sensores na aplicação detectados por participante. Como o foco do trabalho reside apenas na geração dos casos de teste, não verificamos o número de cenários executados por cada participante. As entrevistas mostram a perspectiva do participante em relação à proposta apresentada.

As entrevistas duraram entre 10 e 30 minutos cada, e para cada 10 minutos de gravação utilizamos 1 hora para transcrever. Depois levamos em torno de 2 horas para extrair e codificar cada transcrição. As 5 entrevistas geraram 9 páginas de texto, com 18-29 códigos para cada entrevista. Para estruturar o dado codificado elaboramos 18 padrões dentro de 5 categorias. Este é o resultado final da tarefa de codificação. Todos os passos foram realizados manualmente.

A execução da tarefa de codificação resulta em um conjunto de códigos, que são os dados mais relevantes extraídos de uma fala e possuem a forma de uma sentença simples. Para obter tal sentença, as falas precisam ser extraídas. O processo de extração compreende a retirada de texto que contenha apenas um tipo de informação dentro da fala. Depois de extraída, sintetizamos a ideia extraída da fala no código. A Figura 4.1 mostra um exemplo de extração e codificação de uma fala.

Para cada código estabelecemos um identificador. Este identificador permite o rastreamento do código, de forma que seja possível relacionar o participante, a pergunta e a sentença extraída da fala. Tal código é definido no seguinte formato: X.Y.Z, onde X é o número do participante, Y é o número da pergunta e Z é o número sequencial do código dentro da relação participante-pergunta. O código 1.1.2 representa o segundo código obtido a partir da resposta da questão 1 do participante 1.

O próximo passo é a identificação de padrões e categorização dos códigos obtidos, cujo resultado mostramos na Tabela 4.5.

As categorias de código que encontramos em relação ao CEManTIKA Test Creator são sobre a expectativa, características, execução, esforço e artefatos. Para obter a disposição final dos códigos agrupamos estes em torno de uma ideia. Para tanto, primeiro agrupamos os códigos de acordo com o padrão de código e depois agrupamos estes padrões em categorias.

Para avaliar o sentido de cada código, classificamos cada um como positivo, negativo ou neutro. Desta forma conseguimos mostrar os pontos fortes e pontos fracos da abordagem proposta. A Figura 4.2 exibe esta classificação com um gráfico.

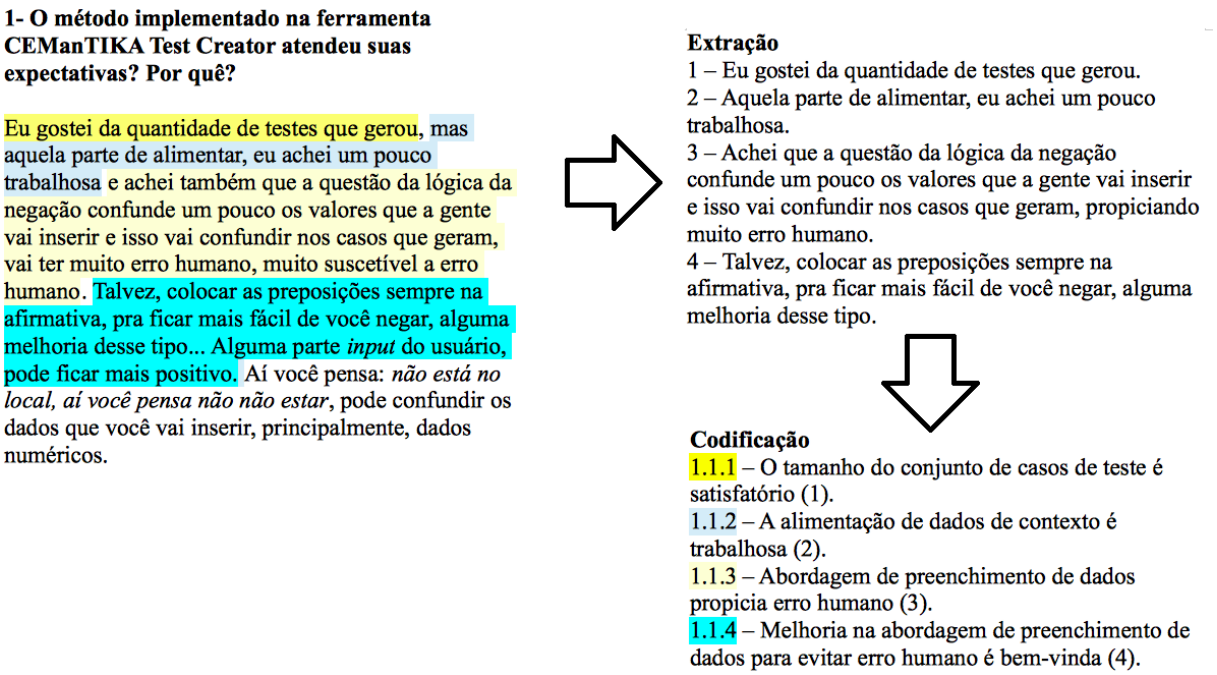

Figura 4.1 Ilustração do processo de codificação em suas fases: Transcrição, Extração e Codificação.

4.7 DISCUSSÃO

Nesta seção detalhamos as descobertas realizadas ao analisar as informações coletadas na codificação das entrevistas. De forma geral, os participantes acreditaram que a solução proposta atende à expectativa de gerar casos de teste para detecção de defeitos em fontes de contexto, ajudando assim o testador de aplicações móveis. A exceção a esta constatação foi de um participante que não soube responder pois não tinha expectativa formada. Alguns acreditavam que a solução proposta aumenta a qualidade da aplicação que está sob teste de alguma forma.

> "Atendeu às expectativas [2.1.1]. O método gerou casos de teste de forma surpreendentemente positiva [1.7.2]. A qualidade da aplicação vai melhorar [4.4.1]."

Um ponto que causou divergência nas respostas coletadas pelos participantes foi em relação à abordagem sistemática utilizada pela solução. A abordagem sistemática compreende os aspectos relativos aos passos necessários para obter os casos de teste. Alguns participantes gostaram da divisão do processo, enquanto outros acharam esta abordagem confusa. Esta dificuldade em compreender os conceitos e passos pode ser creditada à longa curva de aprendizado.

> "As etapas estão bem definidas [3.2.1]. Por delimitar bem as etapas, o método reduz a chance de ocorrer erros humanos [5.5.3]. Alguns passos do processo e alguns termos utilizados estavam confusos [3.7.3]. Possui longa curva de

Tabela 4.5 Categorias e padrões de códigos identificados.

Categorias	**Padrões de Códigos**	**Quantidade de Códigos**
Expectativa	Atendimento ao propósito	22
	Qualidade	4
Característica	Abordagem sistemática	6
	Base de conhecimento	3
	Uso de modelos	7
Execução	Conferência na execução dos Casos de Teste	4
	Entrada de dados de contexto	13
	Apresentação de conceitos	1
	Desempenho	2
	Execução do método	4
	Instalação	2
	Interface gráfica	5
	Usabilidade	5
	Visualização	2
Artefatos	Conjunto de Casos de Teste	18
	Cobertura	1
Esforço	Aprendizado	4
	Economia	1

> aprendizado [2.3.2]. Uma melhoria seria ajustar os termos para tornar os conceitos mais claros, além da sua forma de exibição [3.10.1]."

Como trata-se de um protótipo, aspectos relativos à apresentação dos dados, como visualização, usabilidade, interface gráfica foram criticados pelos participantes. Além disso, o Participante 1 apontou também a instalação das ferramentas contidas no protótipo como um ponto que deve ser melhorado para adoção da implementação da proposta.

> "A usabilidade é um ponto a melhorar [3.3.1]. A visualização dos artefatos criados pode melhorar [2.6.1]. A instalação do ambiente de execução da ferramenta é complexa [1.3.2]."

Outro ponto onde houve divergência de opinião entre os participantes é em relação ao conjunto de casos de teste gerado. A cobertura dos casos de teste gerados foi considerada como um ponto positivo, enquanto que a redundância nos casos de teste foi um contraponto. O participante 5 notou a relação entre a complexidade do grafo contextual (em termos de decisões nos nós contextuais) e o volume de casos de teste gerados. Mesmo com um conjunto de casos de teste considerado redundante, alguns participantes acharam que a abordagem reduz o esforço do testador, trazendo economia de tempo e recursos na fase de testes da aplicação.

> "O tamanho do conjunto de casos de teste é satisfatório [1.1.1]. Os casos de teste possuem boa cobertura, mas existe redundância [1.5.2]. O conjunto de

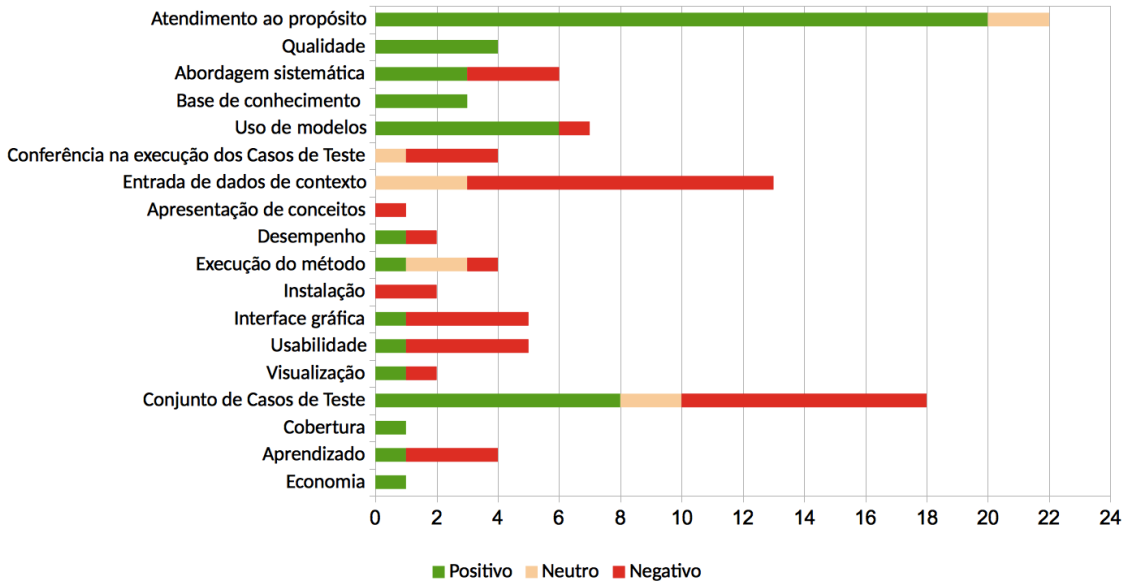

Figura 4.2 Classificação dos códigos de acordo com o sentido avaliado (positivo, neutro e negativo)

> casos de teste gerados é satisfatório [3.7.1]. Informações de natureza complexa são combinadas de maneira complexa entre si pelo gerador de caso de teste [3.8.2]. O conjunto de casos de teste está grande e redundante [5.3.3]. O tamanho do conjunto é influenciado pela complexidade do grafo contextual [5.7.2]."

Quando mencionada, a base de conhecimento foi elogiada. O participante 5 relatou que o uso da base de conhecimento reduz esforços, ao permitir o reaproveitamento de informações de contexto.

> " Salvar os cenários é uma vantagem [5.8.3]. A base de conhecimento favorece a reaplicação dos casos de teste [5.4.2]. O uso de uma base de conhecimento reduz esforço na execução dos testes [5.10.2]."

O uso de modelos foi considerado um ponto positivo pelos participantes. Segundo alguns participantes, o fato dos modelos utilizados pela proposta já terem sido elaborados em uma etapa anterior do desenvolvimento da aplicação é uma vantagem, pois livra o testador da descoberta das possibilidades de combinações de contexto suportadas pela aplicação sob teste.

> "É um ponto positivo da abordagem, e aproveita um trabalho feito em uma etapa anterior do desenvolvimento da aplicação [1.9.1]. O uso de modelagem da aplicação é interessante, pois é uma informação que é confiável e já está disponível [3.9.1]. O uso de modelos abre a cabeça do testador para as possibilidades de casos de teste [2.9.1]."

Um ponto bastante criticado foi o preenchimento dos valores de sensores, os quais se revelaram uma tarefa tediosa e propensa a erros, pois além dos valores de sensores em situação regular, os testadores teriam que preencher dados de sensores em situação de falha. Este era um ponto esperado, pois no próprio exemplo exibido na Figura 3.29 vemos que são 6 contextos lógicos de localização com 3 campos numéricos cada. Preferimos assumir este risco (submeter o protótipo para o uso com esta restrição de uso) para colher sugestões de melhorias. Isto ocorreu, conforme afirmado no código 5.9.4, no qual o participante 5 sugere a inclusão de uma tarefa anterior à identificação dos contextos lógicos. Alguns participantes reclamaram que os textos de ajuda para os casos de sensores afetados por padrões de defeito não cumpriam com o propósito de orientar o preenchimento destes valores excepcionais. Além disso, outros participantes consideraram o preenchimento inadequado de dados de contexto nesta etapa um risco à geração dos casos de teste.

> "A alimentação de dados de contexto é trabalhosa [1.1.2]. Abordagem de preenchimento de dados propicia erro humano [1.1.3]. O trabalho manual propicia erros humanos [5.5.2]. É confusa a entrada de dados de sensores afetados por padrões de defeitos [2.3.3]. O erro de preenchimento de dados de sensores nas etapas iniciais do método pode ocasionar casos de teste que não detectam falhas [1.5.1]. Acho que outra abordagem de preenchimento de dados de sensores poderia para evitar repetição de digitação. Proponho uma etapa anterior ao preenchimento dos contextos lógicos [5.9.4]."

A conferência automática da aplicação durante os testes de forma a detectar os defeitos também foi alvo de sugestão de melhoria de alguns participantes. Para os participantes 2 e 5, uma proposta que englobasse a captura do comportamento da aplicação sob teste seria muito interessante.

> "A conferência do comportamento da aplicação sob teste é feita visualmente pelo testador, e seria interessante se esta etapa fosse automatizada [2.10.1], [5.4.3], [5.8.4]. A automatização da conferência dos testes tornaria o número de casos de teste um problema de impacto relativamente baixo, pois apenas os casos em que ocorressem comportamentos não esperados seriam verificados. O tempo de execução passaria a ser o principal problema [5.8.5]."

A partir dos dados coletados e codificados conseguimos reunir informações relevantes sobre a adoção e uso da abordagem proposta por um testador de aplicativos móveis. Vimos que a abordagem tem potencial de ajudar o testador de aplicativos móveis, pois facilita a tarefa de geração de casos de teste ao definir as tarefas para a criação de dados de contexto em alto nível de abstração. Além disso, existe a possibilidade no ganho de tempo obtido ao combinar os cenários com os padrões de defeitos, o que pode resultar em redução de custos na fase de testes, que pode melhorar ainda mais com a capacidade de reuso dos dados de contexto em outras execuções de testes, a qual é provida pela CKTB. Como a abordagem proposta é nova, e foi disponibilizado um protótipo para avaliação pelos participantes, alguns pontos negativos foram obtidos através da reclamação dos

participantes, como: preenchimento de (muitos) dados de sensores de forma manual; conceitos pouco amigáveis apresentados na interface, fluxos de tela carregados e conjuntos de casos de teste volumosos e redundantes.

4.8 RISCOS À VALIDADE

Para a **validade de conclusão**, o ponto de atenção fica por conta do baixo número de participantes, o que causa risco em relação à generalização dos resultados encontrados.

Em relação à **validade interna**, identificamos o risco na seleção dos participantes, cuja experiência no teste de aplicativos móveis não era conhecida. Assumimos que os participantes tinham um conhecimento básico no teste de aplicativos móveis.

Para a **validade de construção**, identificamos como um risco a escolha das métricas, as quais não conseguem extrair informações de eficiência e eficácia do método avaliado. Uma avaliação comparativa entre o método proposto e outros já existentes poderia elucidar esta lacuna.

No que diz respeito à **validade externa**, consideramos o risco dos participantes serem todos de Salvador/BA, de forma que questões etnográficas podem influenciar no entendimento e execução das atividades do estudo empírico.

4.9 OBSERVAÇÕES FINAIS

A partir da execução do estudo empírico com um protótipo para o CEManTIKA Test Creator conseguimos colher informações que nos permitem dizer que a abordagem proposta pode ajudar os testadores de aplicativos móveis. O projeto do estudo empírico foi formulado de uma maneira que o mesmo possa ser replicado em outros estudos.

O estudo qualitativo que usamos no estudo empírico nos permitiu visualizar os ganhos e problemas da proposta a partir da perspectiva do testador de aplicativos móveis. Como a ferramenta disponibilizada é apenas um protótipo, não sentimos segurança em definir uma hipótese específica (em termos de eficiência/eficácia, por exemplo) para avaliar a proposta. Desta maneira conseguimos coletar informações que podem gerar hipóteses para trabalhos futuros, como por exemplo a melhoria da eficiência dos casos de teste gerados com o uso de outro método para redução de casos de teste.

Capítulo 5

CONCLUSÃO

Nesta dissertação apresentamos o CEManTIKA Test Creator, um método de criação sistemática de casos de teste que visa ajudar testadores de aplicativos móveis na identificação de defeitos na adaptação ao contexto. Para atingir este objetivo, estabelecemos três objetivos específicos, que são: 1) Definir e implementar os artefatos utilizados pelo método para ajudar o testador na geração dos casos de teste; 2) Sistematizar o uso destes artefatos na etapa de preenchimento dos dados de contexto para a geração dos casos de teste; e 3) Verificar como o método proposto pode ajudar o testador.

Para subsidiar o método proposto, escolhemos os seguintes artefatos: modelos de contexto que permitam inferir quais são os sensores que fornecem dados de contexto para um determinado elemento contextual, e que indiquem o comportamento da aplicação na adaptação ao contexto em relação a estes sensores; e padrões de defeitos de contexto, os quais são combinados com casos de teste para revelar defeitos na adaptação ao contexto.

Para sistematizar o uso destes artefatos no preenchimento dos dados de contexto, criamos um conjunto de tarefas visando o preenchimento incremental de dados de contexto, onde o nível de abstração dos dados de contexto informados pelo testador inicia no nível de sensor (baixa abstração) e chega até uma sequência temporal de eventos complexos de contexto (alta abstração). Para suportar esta sequência sistemática de tarefas, elaboramos uma arquitetura que serve para manipular os dados de contexto utilizados na geração dos casos de teste.

Para verificar como o método proposto pode ajudar o testador, primeiro implementamos um protótipo de forma a interligar a ferramenta de projeto de sistemas sensíveis ao contexto CEManTIKA CASE ao simulador de contexto Context Simulator. Utilizamos este protótipo na realização de um estudo experimental, cujo objetivo foi coletar informações que evidenciem se a abordagem proposta ajuda de fato o testador na descoberta de defeitos de fontes de contexto em aplicativos móveis. Para tanto, aplicamos um estudo qualitativo com desenvolvedores e testadores de aplicativos móveis onde estes participantes elaboraram casos de teste para o aplicativo móvel sensível ao contexto *NoCamInMeeting*.

Ao comparar o CEManTIKA Test Creator com as abordagens encontradas na literatura, vimos que a ausência de geração de casos de teste em contexto com alto nível de abstração era uma deficiência comum às abordagens encontradas. O preenchimento de dados de contexto nestas propostas ou não era mencionado ou era feito de forma *ad-hoc*. Acreditamos que com a nossa proposta conseguiríamos ajudar os testadores de aplicativos móveis ao fornecer um método que contemple, além do uso de dados de contexto de alto nível, o uso de modelos da aplicação e de padrões de defeitos para gerar casos de teste simulando defeitos em fontes de contexto.

Seguindo esta linha, a solução proposta conseguiu gerar casos de teste, que embora fossem considerados redundantes em algumas situações, apresentaram boa cobertura para os testadores de aplicativos móveis.

5.1 CONTRIBUIÇÕES

Ao final deste trabalho, fornecemos as seguintes contribuições para a comunidade de pesquisa:

- O método CEManTIKA Test Creator, que permite a geração de casos de teste para aplicativos móveis sensíveis ao contexto. Este método compreende uma sequência sistemática de tarefas, as quais suportam: modelos da aplicação; definições de contexto com alto nível de abstração; padrões de defeitos de fontes de contexto; e uso de método de redução de casos de teste.
- A extensão do metamodelo conceitual presente no framework CEManTIKA, de forma a suportar a definição de sensores como fornecedores de dados de contexto para elementos contextuais, além de sua implementação no CEManTIKA CASE.
- A implementação de uma extensão na ferramenta CEManTIKA CASE para incorporar características relativas à etapa da geração de casos de teste.
- Aumento da interoperabilidade da ferramenta Context Simulator ao implementar a importação de casos de teste gerados externamente, além da funcionalidade de visualizar o comportamento esperado pela aplicação em uma determinada situação.
- A interligação, feita através da implementação de um protótipo, de uma ferramenta de modelagem de contexto com um simulador de contexto, que são respectivamente a CEManTIKA CASE e o Context Simulator. As duas ferramentas em questão não possuíam relação entre si.

5.2 LIMITAÇÕES

Uma limitação da abordagem é que embora tenhamos definido a CKTB como uma base de conhecimento, não especificamos como ocorre a aquisição automática de conhecimento, de forma que a mesma pode ser definida como uma base de dados.

Outra limitação que identificamos após a execução da avaliação é a escolha de um método de redução de casos de teste que possa reduzir de forma mais eficaz o conjunto de casos de teste.

Encaramos como uma limitação desta dissertação o método de avaliação escolhido, o qual não foi capaz de inferir a eficiência e eficácia do método na detecção de defeitos

relacionados a contexto em aplicativos móveis. A avaliação limita-se às impressões dos testadores de aplicativos móveis.

5.3 TRABALHOS FUTUROS

Tendo como ponto de partida os resultados encontrados nesta dissertação, temos os seguintes trabalhos futuros:

- **Uso de outros modelos:** A proposta elaborada utilizou modelos de representação definidos pelo framework conceitual CEManTIKA, como o modelo conceitual, que representa a estrutura da aplicação, e o grafo contextual, que representa o comportamento. Outros tipos de modelo poderiam ser utilizados para a representação de contexto, como ontologias ou redes de Petri.
- **Extensão do Repositório de Padrões de Defeitos em fontes de contexto:** Nesta dissertação definimos e implementamos estratégias para 4 padrões de defeitos de fontes de contexto. Existem outras 11 que não foram cobertas neste trabalho, de acordo com a listagem presente em (Hassel, 2014).
- **Avaliação quantitativa:** Nesta dissertação realizamos um estudo experimental de caráter qualitativo, de forma a coletar as impressões dos participantes na execução do método. Acreditamos na importância da execução de um experimento quantitativo que avalie a eficácia e eficiência do método proposto frente a outras abordagens disponíveis.
- **Automatização da etapa de execução dos casos de teste:** Definimos, neste trabalho, uma sistematização das tarefas relacionadas à etapa de geração de casos de teste. Apenas uma das abordagens encontradas (Griebe e Gruhn, 2014) preocupou-se em relação à execução de casos de teste e identificação de falhas da aplicação de forma automática. Acreditamos que a definição de um método para executar os casos de teste e registrar os defeitos encontrados de forma automatizada seria interessante.
- **Uso de outros métodos de redução de casos de teste:** Mesmo implementando um componente de redução de casos de teste, os casos de teste gerados durante o estudo experimental foram considerados redundantes pelos participantes. Consideramos que um estudo comparativo entre diferentes estratégias/implementações de redução de casos de teste para o método proposto nesta dissertação seria interessante.

REFERÊNCIAS BIBLIOGRÁFICAS

Abowd, G. D.; Dey, A. K.; Brown, P. J.; Davies, N.; Smith, M.; Steggles, P. Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK, UK: Springer-Verlag, 1999. (HUC '99), p. 304–307. ISBN 3-540-66550-1.

Amalfitano, D.; Fasolino, A. R.; Tramontana, P.; Carmine, S. D.; Memon, A. M. Using gui ripping for automated testing of android applications. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2012. (ASE 2012), p. 258–261. ISBN 978-1-4503-1204-2.

Amalfitano, D.; Fasolino, A. R.; Tramontana, P.; Amatucci, N. Considering Context Events in Event-Based Testing of Mobile Applications. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, Ieee, p. 126–133, mar. 2013.

Anand, S.; Burke, E. K.; Chen, T. Y.; Clark, J.; Cohen, M. B.; Grieskamp, W.; Harman, M.; Harrold, M. J.; Mcminn, P. An orchestrated survey of methodologies for automated software test case generation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 86, n. 8, p. 1978–2001, ago. 2013. ISSN 0164-1212.

Baldauf, M.; Dustdar, S.; Rosenberg, F. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 4, p. 263–277, jun. 2007. ISSN 1743-8225.

Basili, V. R.; Caldiera, G.; Rombach, H. D. The goal question metric approach. In: *Encyclopedia of Software Engineering*. [S.l.]: Wiley, 1994.

Bettini, C.; Brdiczka, O.; Henricksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A.; Riboni, D. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Elsevier B.V., v. 6, n. 2, p. 161–180, 2010. ISSN 15741192.

Boehm, B.; Basili, V. R. Software defect reduction top 10 list. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 34, n. 1, p. 135–137, jan. 2001. ISSN 0018-9162.

Bourque, P.; Fairley, R. E. (Ed.). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. ISBN 978-0-7695-5166-1.

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. Human Centered Processes.

Brito junior, J. A. d. *Gamifying user interaction to increase collaboration: The G.A.M.E. conceptual framework*. Dissertação (Mestrado) — Universidade Federal da Bahia e Universidade Estadual de Feira de Santana, July 2015.

Bylund, M.; Espinoza, F. Testing and demonstrating context-aware services with quake iii arena. *Commun. ACM*, ACM, New York, NY, USA, v. 45, n. 1, p. 46–48, jan. 2002. ISSN 0001-0782.

Campillo-sanchez, P.; Serrano, E.; Botía, J. A. Testing context-aware services based on smartphones by agent based social simulation. *J. Ambient Intell. Smart Environ.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 5, n. 3, p. 311–330, maio 2013. ISSN 1876-1364.

Cartaxo, E. G.; Machado, P. D. L.; Neto, F. G. O. On the use of a similarity function for test case selection in the context of model-based testing. *Softw. Test. Verif. Reliab.*, John Wiley and Sons Ltd., Chichester, UK, v. 21, n. 2, p. 75–100, jun. 2011. ISSN 0960-0833.

Chen, T. Y.; Leung, H.; Mak, I. Adaptive random testing. In: *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*. [S.l.]: Springer, 2004. p. 320–329.

Cohen, D. M.; Dalal, S. R.; Fredman, M. L.; Patton, G. C. The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, v. 23, n. 7, p. 437–444, Jul 1997. ISSN 0098-5589.

Cohen, D. M.; Dalal, S. R.; Parelius, J.; Patton, G. C. The combinatorial design approach to automatic test generation. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 13, n. 5, p. 83–88, set. 1996. ISSN 0740-7459.

Copeland, L. *A Practitioner's Guide to Software Test Design*. Norwood, MA, USA: Artech House, Inc., 2003. ISBN 158053791X.

Coutinho, A. E. V. B.; Cartaxo, E. G.; Machado, P. D. D. L. *Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing*. [S.l.: s.n.], 2014. ISSN 0963-9314. ISBN 1121901492.

Cruzes, D. S.; Dyba, T. Recommended steps for thematic synthesis in software engineering. In: *2011 International Symposium on Empirical Software Engineering and Measurement*. [S.l.: s.n.], 2011. p. 275–284. ISSN 1949-3770.

El-far, I. K.; Whittaker, J. A. Model-based software testing. *Encyclopedia of Software Engineering*, Wiley Online Library, 2001.

Elnahrawy, E.; Nath, B. Cleaning and querying noisy sensors. In: *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications*. New York, NY, USA: ACM, 2003. (WSNA '03), p. 78–87. ISBN 1-58113-764-8.

Garzon, S. R.; Hritsevskyy, D. Model-based generation of scenario-specific event sequences for the simulation of recurrent user behavior within context-aware applications (wip). In: *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*. San Diego, CA, USA: Society for Computer Simulation International, 2012. (TMS/DEVS '12), p. 29:1–29:6. ISBN 978-1-61839-786-7.

Graham, D.; Black, R.; Veenendaal, E. V.; Evans, I. *Foundations of Software Testing: ISTQB Certification*. [S.l.]: Thomson, 2006. ISBN 9781844803552.

Griebe, T.; Gruhn, V. A model-based approach to test automation for context-aware mobile applications. ACM, New York, NY, USA, p. 420–427, 2014.

Gu, T.; Pung, H. K.; Zhang, D. Q. A middleware for building context-aware mobile services. In: *In Proceedings of IEEE Vehicular Technology Conference (VTC*. [S.l.: s.n.], 2004.

Hamlet, R. Random testing. *Encyclopedia of software Engineering*, Wiley Online Library, 1994.

Hassel, M. *Context Simulator For Testing Mobile Applications*. Dissertação (Mestrado) — University of Applied Sciences Mannheim, March 2014.

Ieee. Ieee standard for software and system test documentation. *IEEE Std 829-2008*, p. 1–150, July 2008.

Jaccard, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, v. 37, p. 547–579, 1901.

Jensen, C. S.; Prasad, M. R.; Møller, A. Automated testing with targeted event sequence generation. In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2013. (ISSTA 2013), p. 67–77. ISBN 978-1-4503-2159-4.

Jones, N. *Top 10 Mobile Technologies and Capabilities for 2015 and 2016*. 2015. Disponível em: <https://www.gartner.com/doc/2665315>.

Khan, M. E.; Khan, F. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Compututer Science and Applications*, Citeseer, v. 3, n. 6, 2012.

King, J. C. Symbolic execution and program testing. *Commun. ACM*, ACM, New York, NY, USA, v. 19, n. 7, p. 385–394, jul. 1976. ISSN 0001-0782.

Lyu, M. R. (Ed.). *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996. ISBN 0-07-039400-8.

Machado, P.; Vincenzi, A.; Maldonado, J. C. Software testing: An overview. In: _____. *Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 1–17. ISBN 978-3-642-14335-9.

Martin, M.; Nurmi, P. A generic large scale simulator for ubiquitous computing. In: *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*. [S.l.: s.n.], 2006. p. 1–3.

Mcgregor, J. *Test early, test often*. May,June 2007. 7–14 p. Disponível em: <http://www.jot.fm/issues/issue_2007_05/column1>.

Mckinley, P. K.; Sadjadi, S. M.; Kasten, E. P.; Cheng, B. H. C. Composing adaptive software. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 37, n. 7, p. 56–64, jul. 2004. ISSN 0018-9162.

Milner, R. Pure bigraphs: Structure and dynamics. *Information and Computation*, v. 204, n. 1, p. 60 – 122, 2006. ISSN 0890-5401.

Muccini, H.; Francesco, A. D.; Esposito, P. Software testing of mobile applications: Challenges and future research directions. In: *Proceedings of the 7th International Workshop on Automation of Software Test*. Piscataway, NJ, USA: IEEE Press, 2012. (AST '12), p. 29–35. ISBN 978-1-4673-1822-8.

Myers, G. J.; Sandler, C.; Badgett, T. *The art of software testing*. [S.l.]: John Wiley & Sons, 2011.

Naik, K.; Tripathy, P. *Software Testing and Quality Assurance: Theory and Practice*. [S.l.]: Wiley, 2011. ISBN 9781118211632.

Openintents. *Open Intents Sensor Simulator*. 2014. Disponível em: <https://github.com/openintents/sensorsimulator>.

Pacheco, C.; Lahiri, S. K.; Ernst, M. D.; Ball, T. Feedback-directed random test generation. In: *29th International Conference on Software Engineering (ICSE'07)*. [S.l.: s.n.], 2007. p. 75–84. ISSN 0270-5257.

Patricio, R. F. A. *CEManTIKA CASE: uma Ferramenta de Apoio ao Desenvolvimento de Sistemas Sensiveis ao Contexto*. Dissertação (Mestrado) — UFPE, 2010.

Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, v. 16, n. 1, p. 414–454, 2014. ISSN 1553877X.

Püschel, G.; Seiger, R.; Schlegel, T. Test modeling for context-aware ubiquitous applications with feature petri nets. In: *Proc. Workshop Model-based Interactive Ubiquitous Systems (MODIQUITOUS)*. [S.l.: s.n.], 2012.

Raymundo, C. R.; Costa, P. D. CraftContext: A Test Platform for Context-Aware Applications. *Journal of Applied Computing Research*, v. 2, n. 1, p. 11–21, dez. 2012. ISSN 2236-8434.

Saldana, J. *The Coding Manual for Qualitative Researchers*. [S.l.]: SAGE Publications, 2012. ISBN 9781446271421.

Sama, M.; Rosenblum, D. S.; Wang, Z.; Elbaum, S. Multi-layer faults in the architectures of mobile, context-aware adaptive applications: A position paper. In: *Proceedings of the 1st International Workshop on Software Architectures and Mobility*. New York, NY, USA: ACM, 2008. (SAM '08), p. 47–49. ISBN 978-1-60558-022-7.

Samsung. *Samsung Sensor Simulator*. 2011. Disponível em: <http://developer.samsung.com/technical-doc/view.do?v=T000000132>.

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.

Seaman, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 25, n. 4, p. 557–572, jul. 1999. ISSN 0098-5589.

Shan, L.; Zhu, H. Testing software modelling tools using data mutation. In: *Proceedings of the 2006 International Workshop on Automation of Software Test*. New York, NY, USA: ACM, 2006. (AST '06), p. 43–49. ISBN 1-59593-408-1.

Shull, F.; Singer, J.; Sjøberg, D. I. *Guide to Advanced Empirical Software Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN 184800043X.

Statista. *Number of available applications in the Google Play Store from December 2009 to September 2016*. 2016. Disponível em: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.

Statista. *Number of available apps in the Apple App Store from July 2008 to June 2016*. 2016. Disponível em: <https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/>.

Utting, M.; Pretschner, A.; Legeard, B. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, John Wiley and Sons Ltd., Chichester, UK, v. 22, n. 5, p. 297–312, ago. 2012. ISSN 0960-0833.

Vieira, V. *CEManTIKA: a domain-independent framework for designing context-sensitive systems*. Tese (Doutorado) — UFPE, 2008.

Vieira, V.; Holl, K.; Hassel, M. A context simulator as testing support for mobile apps. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2015. (SAC '15), p. 535–541. ISBN 978-1-4503-3196-8.

Vieira, V.; Tedesco, P.; Salgado, A. C. *Modelos e Processos para o Desenvolvimento de Sistemas Sensiveis ao Contexto*. [S.l.], 2009.

Vieira, V.; Tedesco, P.; Salgado, A. C. Designing context-sensitive systems: An integrated approach. *Expert Syst. Appl.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 38, n. 2, p. 1119–1138, fev. 2011. ISSN 0957-4174.

Wang, Z.; Elbaum, S.; Rosenblum, D. S. Automated generation of context-aware tests. In: *Proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (ICSE '07), p. 406–415. ISBN 0-7695-2828-7.

Ye, J.; Mckeever, S.; Coyle, L.; Neely, S.; Dobson, S. Resolving uncertainty in context integration and abstraction: Context integration and abstraction. In: *Proceedings of the 5th International Conference on Pervasive Services*. New York, NY, USA: ACM, 2008. (ICPS '08), p. 131–140. ISBN 978-1-60558-135-4.

Yu, L.; Tsai, W. T.; Jiang, Y.; Gao, J. Generating Test Cases for Context-Aware Applications Using Bigraphs. In: *2014 Eighth International Conference on Software Security and Reliability*. [S.l.]: IEEE, 2014. p. 137–146. ISBN 978-1-4799-4296-1.

Yu, S.; Takada, S. External event-based test cases for mobile application. In: *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*. New York, NY, USA: ACM, 2015. (C3S2E '15), p. 148–149. ISBN 978-1-4503-3419-8.

Yürür, O.; Liu, C. H.; Sheng, Z.; Leung, V. C. M.; Moreno, W.; Leung, K. K. Context-awareness for mobile sensing: A survey and future directions. *IEEE Communications Surveys Tutorials*, v. 18, n. 1, p. 68–93, Firstquarter 2016. ISSN 1553-877X.

Apêndice

A

TRANSCRIÇÕES DAS ENTREVISTAS

Neste apêndice apresentamos as transcrições das entrevistas da avaliação qualitativa realizada a partir da execução da proposta deste trabalho.

A.1 ENTREVISTA COM O PARTICIPANTE 1

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê? Eu gostei da quantidade de testes que gerou, mas aquela parte de alimentar, eu achei um pouco trabalhosa e achei também que a questão da lógica da negação confunde um pouco os valores que a gente vai inserir e isso vai confundir nos casos que geram, vai ter muito erro humano, muito suscetível a erro humano. Talvez, colocar as preposições sempre na afirmativa, pra ficar mais fácil de você negar, alguma melhoria desse tipo... Alguma parte *input* do usuário, pode ficar mais positivo. Aí você pensa: *não está no local, aí você pensa não não estar*, pode confundir os dados que você vai inserir, principalmente, dados numéricos.
2- O que você gostou no uso da ferramenta CEManTIKA Test Creator? Eu gostei dos passos serem bem definidos, eu gosto bastante disso, eu gostei também do resultado final e desempenho um pouco maior, não sei se pelo computador que é bom.
3- O que você não gostou no uso do CEManTIKA Test Creator? Não gostei da navegação dentro de cada formulário que ele abre; aquele primeiro é legal a navegação dele, os outros têm umas partes de incluir e alterar aqui... Aí você abre um detalhe, acho que essa navegação ficou um pouco pesada, embora, se você pensar naquilo ali de uma forma semântica, como a gente fala, você pensar que aqui é um caso, você vai definindo os passos, aí você consegue, mas eu acho que a experiência do usuário em si, na parte de adicionar e de subir e descer... Aí já é outra coisa. Num deles, eu acho que era da criação de quando você define os passinhos, quando você coloca as situações no tempo, não dava para subir e descer elas, mas isso é um detalhe... Tinha um que dava para subir e descer, aí tinha outra coisa que não dava para subir e descer... Eu não lembro agora, mas eu acho que era na segunda lista que não dava para fazer... A instalação da plataforma também... Trabalhosa, não sei se vem ao caso, essa parte da plataforma, eu achei complicado.

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator? Eu acho que a qualidade, com certeza, vai melhorar, principalmente se for uma aplicação que antes disso, não tinha teste formal, ou não tinha teste nenhum, acho melhor com certeza! Até porque contexto, isso já é uma coisa que eu sei, né, porque peguei a matéria... Ainda é muito difícil de você testar então melhora bastante.
5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator? Eu já falei até... Aquela questão daquela carga inicial de coordenada do GPS, porque se o usuário errar ali... Se ele errar em alguns daqueles passos, se você fizer uma combinação, você pode até chegar em algo bom, mas se ele errar naquela parte inicial das faixas de valores, aí vai gerar casos de teste que não detectam os problemas. Agora, uma vez que os dados estejam preenchidos corretamente, que eu imagino que preenchi, porque a gente sempre acha que fez certo, eu acho que os testes estão sendo gerados com uma cobertura boa, inclusive alguns são até redundantes, que é para garantir a cobertura.
6- Quais são suas sugestões para mitigar estes riscos? Tem aquilo de você estruturar aquelas perguntas, você faz aquelas perguntas pra incentivar o usuário a preencher, você estrutura, de uma forma que esteja sempre direta, clara e afirmativa, que aí na hora de fazer uma lógica já é mais simples de entender e, talvez, dar exemplos ou então, por exemplo, na parte de imprecisão, quando é um valor numérico, você podia calcular, você podia colocar um percentual para mais ou para menos, eu acho que isso ajudaria, porque eu acho que esse realmente é um ponto de falha, aquela carga inicial, o restante eu acho que está bem fechado.
7- O que você acha sobre o conjunto de casos de teste gerado? Eu gostei, eu achei os casos de testes bem legais, eu acho que eles têm uma boa cobertura... Eu não sei, porque eu não estudei bem a fundo, mas acho que está gerando testes com uma boa cobertura, só acho que com alguma redundância, você vai ver no meu primeiro cenário não teve muita redundância, mas no meu segundo cenário, teve muita redundância, que eu fiz em um cenário maior, aí eu estaria detectando várias vezes o mesmo defeito, ou não detectando várias vezes o mesmo defeito. Mas eu gostei, eu fiquei até surpreso com o que encontrei.
8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê? Eu acho que não, porque, na verdade, por não ter tanta experiência com teste, eu também não tenho essa experiência formal de padrão de defeito, de saber como gerar o defeito, porque é um defeito, eu não teria, por exemplo, essa ideia de fazer essa combinação de tempo tal com tempo tal... Ia sempre terminar com algo a mais ou a menos.
9- O que você acha sobre o uso de modelos da aplicação pelo método proposto? Eu acho ótimo! Eu acho ótimo! Eu só acho que é um trabalho a mais, mas é um trabalho de projetista, então num processo maduro de desenvolvimento de software, já é um processo que já deverá ter sido feito então, eu acho que não vai trazer ônus nenhum no processo de desenvolvimento e você está dando uma utilidade nova pra algo que não tivesse utilidade, que, às vezes, foi feito só pra ser feito.
10- Você gostaria de mencionar algo mais? Só que eu gostei; Gostei muito dos

casos de testes gerados, do [Context] Simulator, uma vez que ele engatou e começou a funcionar direitinho... E só uma ressalva da plataforma mesmo, nada de mais eu vejo, nada que realmente comprometa o trabalho.

A.2 ENTREVISTA COM O PARTICIPANTE 2

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê? Sim. Talvez, na forma de se fazer e de se criar os artefatos, no Eclipse, poderiam estar um pouco mais claros.
2- O que você gostou no uso da ferramenta CEManTIKA Test Creator? Eu gostei da variedade de opções que você pode aplicar e da automatização, de você colocar vários tempos diferentes para saber um resultado e a interação direta com o emulador, achei bacana, bem simples de fazer também.
3- O que você não gostou no uso do CEManTIKA Test Creator? Talvez poderia ficar uma coisa mais automática, nessa questão de importação e exportação sem que a gente precise fazer muitos passos, achei que no começo a curva de aprendizado foi muito longa, mas depois que você pega o jeito, você consegue fazer os casos de teste, mais tranquilo... Talvez, em um dia, com alguns exemplos... O que pegou mesmo foi na hora da criação dos casos, que eu fiquei um pouco confuso na questão de se colocar o nome do SSID errado, de você forçar o erro e essas coisas; aquele texto que estava na caixa do [Contexto Lógico no] Eclipse também não estava ajudando muito.
4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator? Eu já fiz um estudo sobre essa questão de qualidade, até do projeto daqui do FPC, da prefeitura e eu acho que esses casos de teste, a gente poderia usar, inclusive, no projeto da prefeitura, porque essa questão de GPS, de tempo, acontece muito isso quando os usuários estão usando a aplicação da gente e acontece essa [questão de] granularidade no GPS e eles ficam meio perdidos e a gente também se a gente não rodasse muitos casos de teste. Então, esses casos de teste ajudam o desenvolvedor a saber também que o usuário nem sempre está maluco.
5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator? Na questão da curva de aprendizado, se a pessoa não aprender direito ela pode gerar alguns testes que não vão bater com a realidade e você pode estar gerando um falso positivo, então isso é um risco.
6- Quais são suas sugestões para mitigar estes riscos? Para amenizar os riscos, depois de ter criado o artefato, a gente poderia ter uma melhor visualização do que era para acontecer e do que aconteceu de errado, porque você configura lá que a câmera tem de estar disponível, ou não disponível, mas você poderia visualizar coisas que o GPS está um pouquinho acima e que não deveria estar, uma coisa mais intuitiva para evitar que a pessoa veja como impositivo e ali não ser. No próprio Context Simulator, poderia ter alguma indicação de alguma coisa, que a gente pudesse ver na hora o artefato pronto e o que estivesse dentro dele, de forma mais fácil, sem precisar ficar voltando pra todos aqueles passos, que a gente tinha feito antes, aí tem que olhar o cenário, olhar a situação, tudo isso é cansativo.
7- O que você acha sobre o conjunto de casos de teste gerado? Achei bom...

Achei que gerou ele gerou o teste direitinho, do que tinha pra localizar nos atributos, me satisfez, não precisei fazer muita coisa não.
8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê? Não sei... Eu acho que eu ia precisar de alguma ajuda, apoio, de um especialista ou de alguém com experiência no método, eu achei que ainda não esteja a nível de usuário básico, mas pra gente que é da área ainda consegue se virar, mas ainda precisa facilitar um pouco as coisas.
9- O que você acha sobre o uso de modelos da aplicação pelo método proposto? Achei que é um guia bom para a pessoa usar pra criar, já abre a mente do testador para os casos que tem que ir.
10- Você gostaria de mencionar algo mais? Eu acho que seria bom, ou eu não sei se eu não soube fazer, ou se não tem mesmo, você poderia editar o cenário direto no Context Simulator. O log não fica salvo não? Do *expected behavior*? Seria legal, para não precisar rodar todos os casos novamente.

A.3 ENTREVISTA COM O PARTICIPANTE 3

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê? Por quê? Eu não sei direito... Porque é a primeira vez que eu estava tendo contato com esse tipo de teste e eu ainda estava um pouco confuso enquanto alguns conceitos, aquela questão de configuração da ferramenta, configuração dos testes... Então realmente eu não sei direito, mas eu acho que o resultado final foi interessante, mas eu não entendi muito sobre o processo, um pouco mecânico por minha parte. Então, essa parte se atendeu às minhas expectativas, eu não sei responder agora, porque na verdade, eu não tinha uma expectativa assim formada.
2- O que você gostou no uso da ferramenta CEManTIKA Test Creator? Eu achei que foi...embora, eu não tivesse entendido direito, eu acho ela está bem organizada nos passos e no que tem que gerar, eu imagino que uma pessoa de testes, como eu também que não tive como saber do que tinha sido feito antes, foi bem sequencial o processo de criação das situações e etc. Então, eu acho que essa parte está muito boa.
3- O que você não gostou no uso do CEManTIKA Test Creator? A questão de usabilidade, por exemplo, a função de copiar e colocar, que não estava funcionando direito no Eclipse... Eu estava tentando deletar um cenário, não foi autogerado não, foi um cenário que eu criei, tentei remover, mas não estava removendo...detalhes de usabilidade mesmo.
4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator? Eu acho que sim... Em relação ao teste de contexto, eu conheço poucos, eu achei que o nível de teste está interessante, o tanto de sensores que ele cobre, eu acho que aumenta sim.
5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator? Talvez, a geração de teste dele, pode ocorrer problemas, caso eles não gerem testes, pode-se dizer, corretos, podem até ser muito além de um contexto real, por exemplo, eu acho que o maior risco está

envolvido nisso... Qual o método que ele utiliza para gerar as combinações possíveis e etc. Seriam os riscos envolvidos na utilização dessa ferramenta...como ela gera os testes pra gente executar, eu acho que o maior risco dela, é ela gerar testes, que não sejam reais, por exemplo, que não simulem situações tão reais, ou se ela gerar testes que executem... Se ela não fornecer ao usuário, por exemplo, todas as entradas necessárias para gerar testes que sejam corretos.
6- Quais são suas sugestões para mitigar estes riscos? Agora de cabeça eu não tenho.
7- O que você acha sobre o conjunto de casos de teste gerado? Os testes que eu vi, eu achei interessante... Eu achei que eles estavam dentro de um padrão mais ou menos real, por exemplo, o Wifi parava de funcionar, a bateria baixava, mas eu não consegui entender alguns passos, como eu perguntei na hora: *Por que está dizendo OK aqui se está errado*, de alguns termos que ficaram confusos, não sei se pela minha falta de experiência, ou se realmente estavam confusos.
8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê? Eu acho que seria um pouco mais difícil, pois eu não conheço nenhuma ferramenta que faça algo parecido, são dados complexos de gerar, combinação de dados que são complexas, então, eu acho que daria muito trabalho.
9- O que você acha sobre o uso de modelos da aplicação pelo método proposto? Eu diria que ele é bom, embora eu não tenha um conhecimento tão grande de como ele é feito e criado, mas eu diria que ele é bom, porque você já tem uma modelagem prévia do sistema, que é confiável, gerar dados de uma modelagem é interessante, esse método de geração de conseguir interpretar direito o modelo, eu acho interessante.
10- Você gostaria de mencionar algo mais? No momento não, só essa questão mesmo de melhorar, talvez, os termos utilizados, para ficar mais claro do que está sendo apresentado, mostrar o principal defeito, qual o resultado esperado, qual foi o defeito que foi gerado, esse tipo de coisa.

A.4 ENTREVISTA COM O PARTICIPANTE 4

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê? Sim. Porque realmente consegue verificar o que acontece no cenário de *se você perdeu a internet*, se está sem sinal de GPS, ele realmente mostra o que pode acontecer com o aparelho, porque se ficar com as informações desatualizadas, ele vai ficar dando informação antiga, como se estivesse tudo OK, sem na verdade de estar. Ele consegue mostrar a questão do cenário, eu achei muito interessante isso.
2- O que você gostou no uso da ferramenta CEManTIKA Test Creator? O que eu achei bem interessante foi justamente visualizar cenários, que até então são mais complexos de você identificar, ele consegue simular, eu não sei bem utilizar a ferramenta, mas sim o resultado, de você simular um cenário de desatualização do aplicativo, de um celular por algum motivo, aí ele consegue mostrar essa conjuntura do que pode acontecer, o que eu achei interessante foi isso.
3- O que você não gostou no uso do CEManTIKA Test Creator? Não sei... Es-

tava em inglês. A usabilidade, eu achei interessante, você está falando daquela primeira parte do preenchimento, né? Eu não achei ruim... Como foi a primeira vez usando, não tive nada assim para identificar. A questão do inglês, a questão de tradução. Mas eu não achei difícil o uso. Provavelmente, uma coisa assim que acho que você poderia melhorar em relação a mostrar um executável direto, de não ter que abrir o Eclipse, de ser uma coisa mais futura, mas por ser da área de TI, eu não tive dificuldade em fazer isso, seria só se fosse uma pessoa que não fosse da área de TI para testar o resultado final. Não tenho pontos negativos para dizer.

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator? Eu acho que a qualidade foi muito elevada, eu gostei muito da forma como ele complementa os cenários, a gente consegue fazer cenários, apesar de pré-definidos, mas cenários que você pode mesclar, e aí ele pode mostrar várias situações, a gente consegue abarcar uma maior quantidade de cenários para identificar os defeitos, no caso, eu achei a qualidade muito boa, o nível de cenários que a gente consegue fechar é muito alto, acaba tendo uma qualidade maior na hora de entregar o software.

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator? No momento eu não consegui visualizar nenhum. Basicamente implementar o cenário real, que é o que pode acontecer e verificar como a aplicação se comporta, eu não vi um risco nisso... Não tem como dizer: *ah... a ferramenta deixou de passar uma coisa.* Não vi risco não.

6- Quais são suas sugestões para mitigar estes riscos? [Não respondida por causa da resposta anterior]

7- O que você acha sobre o conjunto de casos de teste gerado? Eu gostei, achei uma quantidade de cenários boa, dos fluxos que eu imaginei, dos cenários possíveis, conseguiu abarcar todos os cenários imagináveis, no caso prático ali foi a questão de estar na reunião ou não estar, então foi feito ele antes da reunião, durante e depois, então nesses três passos, ficou mesclando estas três opções ali. Eu achei bem satisfatório.

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê? De forma independente? Eu acho que não. Porque ali você está desabilitando um sensor. Eu acho que não. Mesmo você usando um aparelho celular normal, se você desabilita o sensor, se você habilita ou desabilita o sensor, na hora que você habilita e desabilita, logo, logo, ele pega a informação. Então, com a ferramenta, ele facilita a questão dos sensores que já tem, acho que facilita no momento de testar por causa disso.

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto? Achei bom. A princípio, eu vi que abarcou todos os cenários que tinham ali no modelo, então achei bom.

10- Você gostaria de mencionar algo mais? Basicamente, que me surpreendeu... Eu gostei muito do que eu vi, a ideia muito boa, interessante mesmo e boa sorte aí... Eu gostei, eu achei fantástica a ideia de como testar, a questão dos sensores, eu achei muito interessante, eu espero ter ajudado.

A.5 ENTREVISTA COM O PARTICIPANTE 5

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê? As minhas expectativas... Acho que poderia ser um pouquinho mais inteligente, por exemplo: os valores de fora e dentro também são valores semânticos, fora e dentro é conceitual. Então, eu poderia definir o que é fora e o que é dentro e apenas ilustrar o que é fora e dentro, para facilitar o caso de teste, sem precisar me preocupar em digitar posições, apenas selecionar o que é fora e o que é dentro.
2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?A economia na hora de gerar os casos de teste, já adianta bastante o processo de geração de casos de teste.
3- O que você não gostou no uso do CEManTIKA Test Creator?Eu não gostei da interface gráfica, pode dar uma melhorada na interface gráfica... Facilitar umas ordenações que não precisam estar ordenadas, selecionar mais e colocar pra facilitar o processo. Em relação aos testes em si, acho que o fator maior pra mim é a usabilidade e a legibilidade das coisas, ficou tudo apertadinho na time line, em vez de ser na horizontal, de repente ser na vertical mesmo, porque daria pra ver o texto inteiro, saber o que eu escrevi ali, isso implica na usabilidade. E a geração de casos de teste, como você falou, gera um volume grande e alguns não são significativos... Talvez, tentar ir pelo caminho crítico, tentar encontrar caminhos críticos para gerar uma quantidade menor.
4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator? Acredito que aumenta a qualidade. Facilita para o testador até reaplicar o teste no momento, que então vai ter uma base de teste para testar de novo, evita estar arrumando, evita o cansaço do testador. Além disso, uma coisa que eu acho que dava para fazer você já diz embaixo: *espero o comportamento tal* e visualmente o testador tem que ver as duas coisas; Mostrar automaticamente que eu espero que a câmera esteja desligada e ela está desligada e não ficar verdinha. Espero que esteja verde, e por algum motivo não está, eu tenho que ver isso visualmente, que a conferência seria automática, aí o teste ficaria mais automatizado; trazer um pouco do *Selenium* para aí.
5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator? Se o cara seguir ali rigorosamente, eu não vejo risco nenhum, mas é um trabalho braçal de gerar, e talvez na geração ele pode esquecer de gerar alguma coisa, mas se ele seguir religiosamente aquilo ali, não vejo nenhum risco de erro humano, você tenta cercar o erro humano ali. Da geração, o risco é o próprio erro humano da geração, não vejo nada.
6- Quais são suas sugestões para mitigar estes riscos? Para mitigar, uma validação. Um segundo momento de validação porque você já gera em quantidade os casos de teste, cobre boa parte das possibilidades. Fazer uma combinatória de tudo que ele colocou lá praticamente. Então, o risco nessas regras que você criou para diminuir os casos de teste, não gerar algum teste, que detecte uma falha, mas se você usou algum critério pra não gerar todos os testes aí, um critério científico, formalizou.
7- O que você acha sobre o conjunto de casos de teste gerado? Grande! Muito

grande! Não dá nem pra avaliar a completude disso, até pelo tamanho da aplicação, e de gerar um caso número muito grande de casos de teste. Se a gente fosse fazer humanamente, cada *if* daria 1,2,3,5,6...6 testes para cobrir todos os *ifs*... 6 testes você quis implicar. Aí a gente tentaria colocar algum tipo de erro em cada um desses 6 testes, gerariam mais de 6 testes, se você botasse isso: em que momento reduzir os *ifs* implicaria em gerar esses erros?

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê? Conseguiria se eu fosse buscar uma técnica de teste, eu ia lá olhar fazer um teste com caminho crítico, analisar as probabilidades dos testes e fazer isso manualmente, com um tempo muito maior. O maior problema é não ser tão repetível, não ter base de teste pra gerar de volta. Imagine que eu criei um cenário muito grande, fiz uma vez, encontrei o erro e corrigi... Como é que faz com aquele cenário lá grandão, você guarda. A vantagem está aí: de você poder executar essa base em qualquer momento, aí aquele cenário que passou legal, qual foi mesmo? Aí você tem os cenários OK e os cenários não OK, você roda o cenário não OK, aí que está o lance da automatização, em todos os cenários OK, eu vou ter que analisar de novo, isso é ruim. Se a conferência fosse automática, eu posso até dar uma analisada se não deu um falso verdadeiro ali, aí o número de casos de teste que você gerou já não iria impactar, só seria o tempo de execução.

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto? O primeiro modelo da estrutura, eu achei um pouco confuso, mas beleza. O testador vai identificar mais ou menos os sensores que ele deve trabalhar, isso é legal e poderia ser substituído por uma lista. Para o cara no papel de testes, uma lista resolveria pra ele, e o outro modelo do grafo contextual, ele ajuda, e pelo que eu percebi ele ajuda a gerar os casos de teste, ele dá uma ideia do que é para fazer, mas isso não impacta tanto, porque querendo ou não o gerador já dá muito na mão todas as possibilidades. Talvez sincronizar isso para ficar mais claro em que ponto do diagrama eu estou, qual foi a condição que me levou a aquilo. Você fala assim: GPS não OK e os outros OK, ou só o GPS que está dando erro, ali é por sensor? Eu vejo isso antes do contexto lógico, tem o contexto do sensor em si, físico, antes do contexto físico, criar simplesmente situações de OK ou não OK, o que é OK e não OK, no contexto lógico eu faço, que é o que você já faz, mas de uma forma que você faz o OK e o não OK dentro, fora, aí já é o contexto lógico, se eu tivesse uma camada antes, que é o contexto físico do que é OK e não OK, no contexto lógico de repente geraria. Esse vai estar com um OK e outro não OK. Assim eu evitaria redigitação de dados de sensores. Eu poderia criar os estados para os sensores, porque lá no contexto [lógico] só mudaria os estados.

10- Você gostaria de mencionar algo mais? Não. Achei legal! Acho que é uma ferramenta que pode ser trabalhada para ajudar realmente os testadores, os caras que fazem um trabalho intelectual, e não ficar testando um a um. Se eu não me engano, lá em Manaus, o pessoal faz uns testes, geram algumas coisas pra testar, tem uma pessoa que fica anotando em uma planilha, lá eles fazem um esquema de rodar um script, abrir e fechar aplicações, umas coisinhas no celular e ficam anotando em uma planilha. Acho que toda automação de teste é bem-vinda, até porque, tem possibilidades de amarrar e rodar isso. Realmente você tem uma base com esses testes e aplicar isso de forma automática.

Quando a gente replica os testes em um contexto maior, vale a pena. Uma coisa que eu vejo de não aplicar só no emulador. Hoje eu vejo aplicativos na web, ele roda em n aparelhos, e em n definições de tela e te dá um relatório final do que é aquilo, ele traz captura dessas telas; ficaria mais interessante se conseguisse aplicar isso em n aparelhos, de forma automática.

Apêndice

B

EXTRAÇÃO E CODIFICAÇÃO DAS TRANSCRIÇÕES

Neste apêndice apresentamos a extração e codificação das transcrições das entrevistas realizadas com os desenvolvedores e testadores de aplicativos móveis.

B.1 EXTRAÇÃO E CODIFICAÇÃO DA ENTREVISTA COM O PARTICIPANTE 1

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?

Extração:

1 - Eu gostei da quantidade de testes que gerou.

2 - Aquela parte de alimentar, eu achei um pouco trabalhosa.

3 - Achei que a questão da lógica da negação confunde um pouco os valores que a gente vai inserir e isso vai confundir nos casos que geram, propiciando muito erro humano.

4 - Talvez, colocar as preposições sempre na afirmativa, pra ficar mais fácil de você negar, alguma melhoria desse tipo.

Codificação:

1.1.1 - O tamanho do conjunto de casos de teste é satisfatório (1).

1.1.2 - A alimentação de dados de contexto é trabalhosa (2).

1.1.3 - Abordagem de preenchimento de dados propicia erro humano (3).

1.1.4 - Melhoria na abordagem de preenchimento de dados para evitar erro humano é bem-vinda (4).

2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?

Extração:

1 - Eu gostei dos passos serem bem definidos, eu gosto bastante disso.

2 - Eu gostei também do resultado final e do desempenho, não sei se pelo computador

que é bom.

Codificação:

1.2.1 - Etapas bem definidas (1).

1.2.2 - Bom desempenho do gerador de casos de teste (2).

3- O que você não gostou no uso do CEManTIKA Test Creator?

Extração:

1 - Não gostei da navegação dentro de cada formulário que ele abre; aquele primeiro é legal a navegação dele, os outros têm umas partes de incluir e alterar. Eu acho que essa navegação ficou um pouco pesada, embora, se você pensar naquilo ali de uma forma semântica, como a gente fala, você pensar que aqui é um caso, você vai definindo os passos.

2 - A instalação da plataforma é trabalhosa, não sei se vem ao caso, essa parte da plataforma, eu achei complicado.

Codificação:

1.3.1 - O fluxo de navegação das telas é pesado (1).

1.3.2 - A instalação do ambiente de execução da ferramenta é complexa (2).

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?

Extração:

1 - Eu acho que a qualidade, com certeza, vai melhorar, principalmente se for uma aplicação que antes disso, não tinha teste formal, ou não tinha teste nenhum, acho melhor com certeza! Até porque contexto já é uma coisa que eu sei. Ainda é muito difícil de você testar então melhora bastante.

Codificação:

1.4.1 - A qualidade da aplicação vai melhorar (1).

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?

Extração:

1 - A carga inicial de coordenada do GPS, porque se o usuário errar ali... Se ele errar em alguns daqueles passos, se você fizer uma combinação. Se ele errar naquela parte inicial das faixas de valores, aí vai gerar casos de teste que não detectam os problemas.

2 - Eu acho que os testes estão sendo gerados com uma cobertura boa, inclusive alguns são até redundantes, que é para garantir a cobertura.

Codificação:

1.5.1 - O erro de preenchimento de dados de sensores nas etapas iniciais do método pode ocasionar casos de teste que não detectam falhas (1).
1.5.2 - Os casos de teste possuem boa cobertura, mas existe redundância (2).

6- Quais são suas sugestões para mitigar estes riscos?

Extração:

1 - Tem aquilo de você estruturar aquelas perguntas, de uma forma que esteja sempre direta, clara e afirmativa, que aí na hora de fazer uma lógica já é mais simples de entender e, talvez, dar exemplos.
2 - Na parte de imprecisão, quando é um valor numérico, você podia calcular, você podia colocar um percentual para mais ou para menos, eu acho que isso ajudaria, porque eu acho que esse realmente é um ponto de falha, aquela carga inicial, o restante eu acho que está bem fechado.

Codificação:

1.6.1 - Tornar mais claro o preenchimento dos valores de sensores, inclusive com exemplos (1).
1.6.2 - Para valores de sensores que são numéricos, poderia utilizar um gerador de números aproximados a partir de um percentual, de forma a reduzir a probabilidade de erro no preenchimento de dados na fase inicial (2).

7- O que você acha sobre o conjunto de casos de teste gerado?

Extração:

1 - Eu gostei, eu achei os casos de testes bem legais, eu acho que eles têm uma boa cobertura, só acho que com alguma redundância, no meu primeiro cenário não teve muita redundância, mas no meu segundo cenário, que era maior, teve muita redundância, aí eu estaria detectando várias vezes o mesmo defeito, ou não detectando várias vezes o mesmo defeito.
2 - Eu gostei, eu fiquei até surpreso com o que encontrei.

Codificação:

1.7.1 - Os casos de teste gerados têm boa cobertura, embora possuam redundância. Esta redundância aumenta quando o cenário possui mais situações (1).
1.7.2 - O método gerou casos de teste de forma surpreendentemente positiva (2).

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?

Extração:

1 - Eu acho que não, porque, por não ter tanta experiência com teste, eu também não tenho essa experiência formal de padrão de defeito, de saber como gerar o defeito, porque é um defeito.

2 - Eu não teria essa ideia de fazer essa combinação de tempo tal com tempo tal, ia sempre terminar com algo a mais ou a menos.

Codificação:

1.8.1 - Pela falta de experiência com testes não seria possível obter os casos de teste gerados de sem apoio da ferramenta (1).
1.8.2 - Caso gerasse os cenários sem apoio, poderia deixar faltar ou sobrar alguma coisa (2).

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto?

Extração:

1 - Eu acho ótimo!
2 - Eu só acho que é um trabalho a mais, mas é um trabalho de projetista, então num processo maduro de desenvolvimento de software, já é um processo que já deverá ter sido feito então, eu acho que não vai trazer ônus nenhum no processo de desenvolvimento.
3 - Você está dando uma utilidade nova pra algo que não tivesse utilidade, que, às vezes, foi feito só pra ser feito.

Codificação:

1.9.1 - É um ponto positivo da abordagem, e aproveita um trabalho feito em uma etapa anterior do desenvolvimento da aplicação (1, 2, 3).

10- Você gostaria de mencionar algo mais?

Extração:

1 - Só que eu gostei; Gostei muito dos casos de testes gerados, do [Context] Simulator, uma vez que ele engatou e começou a funcionar sem sobressaltos.
2 - E só uma ressalva da plataforma mesmo, nada de mais eu vejo, nada que realmente comprometa o trabalho.

Codificação:

1.10.1 - O conjunto de casos de teste gerados é satisfatório (1). 1.10.2 - Após a familiarização com as ferramentas o processo de teste transcorre sem sobressaltos (1).
1.10.3 - A plataforma é de complexa configuração (2).

B.2 EXTRAÇÃO E CODIFICAÇÃO DA ENTREVISTA COM O PARTICIPANTE 2

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?

Extração:

1 - Sim.

2 - Talvez, na forma de se criar os artefatos no Eclipse, poderiam estar um pouco mais claros.

Codificação:

2.1.1 - Atendeu às expectativas (1).

2.1.2 - Falta clareza na criação dos artefatos no Eclipse (2).

2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?

Extração:

1 - Eu gostei da variedade de opções que você pode aplicar e da automatização, de você colocar vários tempos diferentes para saber um resultado e a interação direta com o emulador, achei bacana, bem simples de fazer também.

Codificação:

2.2.1 - Possui boa variedade de opções aplicáveis. Possui boa automatização, além de permitir a interação direta com o emulador (1).

2.2.2 - A execução é simples (1).

3- O que você não gostou no uso do CEManTIKA Test Creator?

Extração:

1 - Talvez poderia ficar uma coisa mais automática, nessa questão de importação e exportação sem que a gente precise fazer muitos passos.

2 - Achei que no começo a curva de aprendizado foi muito longa, mas depois que você pega o jeito, você consegue fazer os casos de teste mais tranquilo.

3- O que pegou mesmo foi na hora da criação dos casos, que eu fiquei um pouco confuso na questão de se colocar o nome do SSID errado, de você forçar o erro e essas coisas.

4 - Aquele texto que estava na caixa do [Contexto Lógico no] Eclipse também não estava ajudando muito.

Codificação:

2.3.1 - Passos para exportação e importação carece de automatização (1).

2.3.2 - Possui longa curva de aprendizado (2).

2.3.3 - É confusa a entrada de dados de sensores afetados por padrões de defeitos (3).

2.3.4 - Os textos de auxilio para os padrões de defeitos não cumprem adequadamente o seu propósito (3, 4).

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?

Extração:

1 - Eu já fiz um estudo sobre essa questão de qualidade e eu acho que esses casos de teste, a gente poderia usar, porque essa questão de GPS, de tempo, acontece muito isso quando os usuários estão usando a aplicação da gente e acontece essa [questão de] granularidade no GPS e eles ficam meio perdidos e a gente também se a gente não rodasse muitos casos de teste.
2 - Esses casos de teste ajudam o desenvolvedor.

Codificação:
2.4.1 - Os casos de teste gerados são aplicáveis para ajudar na descoberta de defeitos (1, 2).
2.4.2 - Os casos de teste gerados com granularidade refletem a realidade de uso típico da aplicação (1).

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?

Extração:
1 - Na questão da curva de aprendizado, se a pessoa não aprender direito ela pode gerar alguns testes que não vão bater com a realidade e você pode estar gerando um falso positivo, então isso é um risco.

Codificação:
2.5.1 - Existe risco de erro humano na geração dos dados de sensores que podem ocasionar em casos de teste inúteis (1).

6- Quais são suas sugestões para mitigar estes riscos?

Extração:
1 - Para amenizar os riscos, depois de ter criado o artefato, a gente poderia ter uma melhor visualização do que era para acontecer e do que aconteceu de errado.

Codificação:
2.6.1 - A visualização dos artefatos criados pode melhorar (1).

7- O que você acha sobre o conjunto de casos de teste gerado?

Extração:
1 - Achei bom.
2 - Achei que gerou ele gerou o teste direitinho, do que tinha pra localizar nos atributos, me satisfez.
3 - Não precisei fazer muita coisa não.

Codificação:
2.7.1 - O conjunto de casos de teste foi satisfatório, e não foi preciso despender muito

esforço para sua geração (1, 2, 3).

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?

Extração:

1 - Não sei. Eu acho que eu ia precisar de alguma ajuda de um especialista ou de alguém com experiência no método.

2 - Eu achei que ainda não esteja a nível de usuário básico, mas pra gente que é da área ainda consegue se virar, mas ainda precisa facilitar um pouco as coisas.

Codificação:

2.8.1 - A geração sem apoio da ferramenta de casos de teste semelhantes poderia acontecer com o auxílio de um especialista, ou de uma pessoa com experiência no método (1, 2).

2.8.2 - O uso da ferramenta requer um esforço não trival para entendimento por parte do usuário (2).

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto?

Extração:

1 - Achei que é um guia bom para a pessoa usar pra criar, já abre a mente do testador para os casos.

Codificação:

2.9.1 - O uso de modelos abre a cabeça do testador para as possibilidades de casos de teste (1).

10- Você gostaria de mencionar algo mais?

Extração:

1 - Seria legal, para não precisar rodar todos os casos novamente, se salvasse o registro dos comportamentos esperados.

Codificação:

2.10.1 - A conferência do comportamento da aplicação sob teste é feita visualmente pelo testador, e seria interessante se esta etapa fosse automatizada (1).

B.3 EXTRAÇÃO E CODIFICAÇÃO DA ENTREVISTA COM O PARTICIPANTE 3

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?

Extração:

1 - Eu não sei, porque é a primeira vez que eu tive contato com esse tipo de teste e eu estava confuso em relação a alguns conceitos, como aquela questão de configuração da ferramenta, configuração dos testes.

2 - Eu acho que o resultado final foi interessante, mas eu não entendi muito sobre o processo, achei um pouco mecânico.

3 - Eu não tinha uma expectativa formada sobre o processo, então não posso dizer se atendeu ou não.

Codificação:

3.1.1 - Não se sabe se atendeu as expectativas (1, 3).

3.1.2 - O resultado final foi interessante (2).

3.1.3 - O processo de geração de casos de teste é mecânico (2).

3.1.4 - Achei confusos os conceitos e o processo (1, 2).

2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?

Extração:

1 - Embora, eu não tivesse entendido direito, eu acho ela está bem organizada nos passos e no que tem que gerar.

2 - Foi bem sequencial o processo de criação das situações e etc. Então, eu acho que essa parte está muito boa.

Codificação:

3.2.1 - As etapas estão bem definidas (1, 2).

3- O que você não gostou no uso do CEManTIKA Test Creator?

Extração:

1 - A questão de usabilidade, e algumas funções que não estavam funcionando no Eclipse, como o copiar e colar.

2 - Tentei e não consegui remover um cenário.

Codificação:

3.3.1 - A usabilidade é um ponto a melhorar (1).

3.3.2 - A ferramenta não apresentou o comportamento esperado (1, 2).

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?

Extração:

1 - Eu acho que aumenta a qualidade.

2 - Eu achei que o nível de teste está interessante e o tanto de sensores cobertos.

Codificação:

3.4.1 - Aumenta a qualidade das aplicações testadas (1).

3.4.2 - Achei que o nível de teste está interessante e possui boa cobertura de sensores (2).

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?

Extração:

1 - Eu acho que o maior risco está envolvido caso possam ocorrer problemas que não se gere o testes corretos, por ir muito além de um contexto real, por exemplo. Eu acho que o maior risco da ferramenta, é ela gerar testes que não sejam reais, por exemplo, que não simulem situações tão reais.

2 - Se ela não fornecer ao usuário, por exemplo, todas as entradas necessárias para gerar testes que sejam corretos.

Codificação:

3.5.1 - Existe o risco de gerar casos de teste que não correspondam à realidade (1).

3.5.2 - Existe o risco de se gerar casos de teste incorretos caso não se preencha todas as entradas necessárias (2).

6- Quais são suas sugestões para mitigar estes riscos?

Extração:

1 - Não tenho a resposta agora.

Codificação:

3.6.1 - Não tenho a resposta agora (1).

7- O que você acha sobre o conjunto de casos de teste gerado?

Extração:

1 - Eu achei interessante.

2 - Eu achei que eles estavam dentro de um padrão mais ou menos real, por exemplo, o Wifi parava de funcionar, a bateria baixava.

3 - Mas eu não consegui entender alguns passos, como eu perguntei na hora: *Por que está dizendo OK aqui se está errado.*

4- Alguns termos ficaram confusos, não sei se pela minha falta de experiência, ou se realmente estavam confusos.

Codificação:

3.7.1 - O conjunto de casos de teste gerados é satisfatório (1).

3.7.2 - Os casos gerados estavam dentro de um padrão próximo ao do mundo real (2).

3.7.3 - Alguns passos do processo e alguns termos utilizados estavam confusos (3, 4).

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?

Extração:

1 - Eu acho que seria um pouco mais difícil, pois eu não conheço nenhuma ferramenta que faça algo parecido, são dados complexos de gerar, combinação de dados que são complexas, então, eu acho que daria muito trabalho.

Codificação:

3.8.1 - Gerar este tipo de caso de teste seria trabalhoso sem o apoio do método implementado (1). 3.8.2 - Informações de natureza complexa são combinadas de maneira complexa entre si pelo gerador de caso de teste (2).

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto?

Extração:

1 - Eu diria que ele é bom, embora eu não tenha um conhecimento tão grande de como ele é feito e criado.
2 - Eu diria que ele é bom porque você já tem uma modelagem prévia do sistema, que é confiável.
3 - Gerar dados a partir de uma modelagem é interessante.

Codificação:

3.9.1 - O uso de modelagem da aplicação é interessante, pois é uma informação que é confiável e já está disponível (1, 2, 3).

10- Você gostaria de mencionar algo mais?

Extração:

1 - Poderia melhorar, talvez, os termos utilizados, para ficar mais claro do que está sendo apresentado, mostrar o principal defeito, qual o resultado esperado, qual foi o defeito que foi gerado etc.

Codificação:

3.10.1 - Uma melhoria seria ajustar os termos para tornar os conceitos mais claros, além da sua forma de exibição (1).

B.4 EXTRAÇÃO E CODIFICAÇÃO DA ENTREVISTA COM O PARTICIPANTE 4

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?

Extração:

1 - Atendeu minhas expectativas, porque realmente consegue verificar o que acontece no cenário se ficar com as informações desatualizadas, ele vai ficar dando informação antiga, como se estivesse tudo OK, sem na verdade de estar. Ele consegue mostrar a questão do cenário, eu achei muito interessante isso.

Codificação:

4.1.1 - Atendeu as expectativas (1).
4.1.2 - As informações sobre os cenários são transparentes (1).

2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?

Extração:

1 - O que eu achei bem interessante foi justamente visualizar cenários, que até então são mais complexos de você identificar, ele consegue simular, eu não sei bem utilizar a ferramenta, mas sim o resultado, de você simular um cenário de desatualização do aplicativo, de um celular por algum motivo, aí ele consegue mostrar essa conjuntura do que pode acontecer, o que eu achei interessante foi isso.

Codificação:

4.2.1 - Achei interessante o modo de visualização dos cenários e seus resultados (1).
4.2.1 - Os casos de teste gerados conseguem simular defeitos em cenários complexos (1).

3- O que você não gostou no uso do CEManTIKA Test Creator?

Extração:

1 - Estava em inglês.
2 - A usabilidade, eu achei interessante, você está falando daquela primeira parte do preenchimento, né? Eu não achei ruim. Como foi a primeira vez usando, não tive nada assim para identificar. A questão do inglês, a questão de tradução. Mas eu não achei difícil o uso.
3 - Provavelmente, uma coisa assim que acho que você poderia melhorar em relação a mostrar um executável direto, de não ter que abrir o Eclipse, de ser uma coisa mais futura, mas por ser da área de TI, eu não tive dificuldade em fazer isso, seria só se fosse uma pessoa que não fosse da área de TI para testar o resultado final.

Codificação:

4.3.1 - A ferramenta estar na língua inglesa pode prejudicar o uso da ferramenta (1,

2).
4.3.2 - A ferramenta ser um plugin de um ambiente de desenvolvimento não familiar ao desenvilvedor/testador pode dificultar sua adoção (3).

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?

Extração:
1 - Eu acho que a qualidade foi muito elevada.
2 - Eu gostei muito da forma como ele complementa os cenários, a gente consegue fazer cenários, apesar de pré-definidos, mas cenários que você pode mesclar, e aí ele pode mostrar várias situações.
3 - Eu achei a qualidade muito boa, o nível de cenários que a gente consegue fechar é muito alto, acaba tendo uma qualidade maior na hora de entregar o software.

Codificação:
4.4.1 - A qualidade da aplicação vai melhorar (3).
4.4.2 - O fator que aumenta a qualidade é o alcance dos cenários gerados (3).
4.4.3 - Os dados gerados possuem qualidade muito boa (1, 2, 3).

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?

Extração:
1 - No momento eu não consegui visualizar nenhum. Basicamente implementar o cenário real, que é o que pode acontecer e verificar como a aplicação se comporta, eu não vi um risco nisso... Não tem como dizer: *ah... a ferramenta deixou de passar uma coisa.* Não vi risco não.

Codificação:
4.5.1 - Não existem riscos aparentes na adoção do método implementado (1).

6- Quais são suas sugestões para mitigar estes riscos?

Extração:
1 - [Não extraído por causa da resposta anterior]

Codificação:
4.6.1 - [Não codificado por causa da resposta anterior]

7- O que você acha sobre o conjunto de casos de teste gerado?

Extração:
1 - Eu gostei, achei uma quantidade de cenários boa, dos fluxos que eu imaginei, dos

cenários possíveis, conseguiu abarcar todos os cenários imagináveis.
2 - Eu achei bem satisfatório.

Codificação:
4.7.1 - O conjunto de casos de teste gerado é satisfatório (1, 2).
4.7.2 - O conjunto de casos de teste abarcou todos os cenários imagináveis (2).

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?

Extração:
1 - Eu acho que não. Porque ali você está desabilitando um sensor.
2 - Mesmo você usando um aparelho celular normal, na hora que você habilita e desabilita, logo, logo, ele pega a informação, e com a ferramenta ele facilita a questão dos sensores que já tem, acho que facilita no momento de testar por causa disso.

Codificação:
4.8.1 - Não acredito que seja possível gerar os casos de teste sem apoio (1).
4.8.2 - O dado de sensor atualiza de forma constante, e o método implementado lida com isso de forma satisfatória (2).

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto?

Extração:
1 - Achei bom.
2 - A princípio, eu vi que abarcou todos os cenários que tinham ali no modelo, então achei bom.

Codificação:
4.9.1 - O uso de modelos da aplicação foi uma boa escolha (1).
4.9.2 - Os casos de teste gerados foram fieis ao indicado no modelo (2).

10- Você gostaria de mencionar algo mais?

Extração:
1 - Basicamente, que me surpreendeu
2 - Eu gostei muito do que eu vi, a ideia muito boa, eu achei fantástica a ideia de como testar, a questão dos sensores, eu achei muito interessante.

Codificação:
4.10.1 - O método gerou casos de teste de forma surpreendentemente positiva (1).
4.10.2 - O teste de sensores de uma aplicação é uma coisa muito interessante (2).

B.5 EXTRAÇÃO E CODIFICAÇÃO DA ENTREVISTA COM O PARTICIPANTE 5

1- O método implementado na ferramenta CEManTIKA Test Creator atendeu suas expectativas? Por quê?

Extração:

1 - As minhas expectativas... Acho que poderia ser um pouquinho mais inteligente, por exemplo: os valores de "fora"e "dentro"também são valores semânticos, são conceitos.
2 - Então, eu poderia definir o que é fora e o que é dentro e apenas ilustrar o que é fora e dentro, para facilitar o caso de teste, sem precisar me preocupar em digitar posições, apenas selecionar o que é fora e o que é dentro.

Codificação:

5.1.1 - Atendeu parcialmente às expectativas (1).
5.1.2 - A entrada de valores de sensores poderia carregar mais significado nos dados (1, 2).

2- O que você gostou no uso da ferramenta CEManTIKA Test Creator?

Extração:

1 - A economia na hora de gerar os casos de teste, já adianta bastante o processo de geração de casos de teste.

Codificação:

5.2.1 - Existe uma economia de esforços e recursos na geração dos casos de teste (1).

3- O que você não gostou no uso do CEManTIKA Test Creator?

Extração:

1 - Eu não gostei da interface gráfica, pode dar uma melhorada na interface gráfica. Facilitar umas ordenações que não precisam estar ordenadas, selecionar mais e colocar pra facilitar o processo.
2 - Em relação aos testes em si, acho que o fator maior pra mim é a usabilidade e a legibilidade das coisas, ficou tudo apertadinho na timeline, em vez de ser na horizontal, de repente ser na vertical mesmo, porque daria pra ver o texto inteiro, saber o que eu escrevi ali, isso implica na usabilidade.
3 - A geração de casos de teste, como você falou, gera um volume grande e alguns não são significativos. Talvez, tentar ir pelo caminho crítico, tentar encontrar caminhos críticos para gerar uma quantidade menor.

Codificação:

5.3.1 - A interface gráfica deixou a desejar (1, 2).
5.3.2 - A usabilidade não está boa (1, 3).

5.3.3 - O conjunto de casos de teste está grande e redundante. (3)
5.3.4 - Alguns casos de teste não são significativos por causa da redundância (3).

4- Qual sua impressão sobre a qualidade de aplicações que são testadas com auxílio do CEManTIKA Test Creator?

Extração:
1 - Acredito que aumenta a qualidade.
2 - Facilita para o testador reaplicar o teste no momento.
3 - O testador vai ter uma base de teste para testar de novo, evita estar arrumando, evita o cansaço do testador.
4 - Além disso, uma coisa que eu acho que dava para fazer: Mostrar automaticamente que eu espero que a câmera esteja desligada e ela está desligada e não ficar verdinha. Espero que esteja verde, e por algum motivo não está, eu tenho que ver isso visualmente, que a conferência seria automática, aí o teste ficaria mais automatizado.
5 - Trazer um pouco do *Selenium* para isso.

Codificação:
5.4.1 - A qualidade da aplicação vai melhorar (1).
5.4.2 - A base de conhecimento favorece a reaplicação dos casos de teste (2).
5.4.3 - A conferência do comportamento da aplicação sob teste é feita visualmente pelo testador, e seria interessante se esta etapa fosse automatizada (4, 5).

5- Quais são os maiores riscos que você enxerga no uso do método implementado pela ferramenta CEManTIKA Test Creator?

Extração:
1 - Se o cara seguir ali rigorosamente, eu não vejo risco nenhum mas é um trabalho braçal de gerar, e talvez na geração ele pode esquecer de gerar alguma coisa. Mas se ele seguir religiosamente aquilo ali, não vejo nenhum risco de erro humano, você tenta cercar o erro humano ali.
2 - Da geração, o risco é o próprio erro humano, não vejo nada além disso.

Codificação:
5.5.1 - O método em si não apresenta riscos, exceto ocorrência a falha humana (1, 2).
5.5.2 - O trabalho manual propicia erros humanos (1). 5.5.3 - Por delimitar bem as etapas, o método reduz a chance de ocorrer erros humanos (1).

6- Quais são suas sugestões para mitigar estes riscos?

Extração:
1 - Para mitigar sugiro uma validação. Um segundo momento de validação porque você já gera em quantidade os casos de teste, cobre boa parte das possibilidades.
2 - Fazer uma combinatória de tudo que ele colocou lá praticamente. Então, o risco

nessas regras que você criou para diminuir os casos de teste, não gerar algum teste que detecte uma falha, mas se você usou algum critério pra não gerar todos os testes aí, um critério científico, formalizou.

Codificação:

5.6.1 - Uma abordagem para mitigar a falha humana é a realização de uma validação nos dados preenchidos (1).

5.6.2 - Uma abordagem para mitigar o risco da não geração de casos de teste é a revisão da técnica de redução (2).

7- O que você acha sobre o conjunto de casos de teste gerado?

Extração:

1 - Grande! Muito grande! Não dá nem pra avaliar a completude disso, até pelo tamanho da aplicação, e de gerar um caso número muito grande de casos de teste. 2 - Se a gente fosse fazer manualmente, cada *if* [saída do nó contextual] daria 1,2,3,5,6... 6 testes para cobrir todos os *ifs*. 6 testes você quis implicar. Aí a gente tentaria colocar algum tipo de erro em cada um desses 6 testes, gerariam mais de 6 testes, se você botasse isso: em que momento reduzir os *ifs* implicaria em gerar esses erros?

Codificação:

5.7.1 - O tamanho do conjunto de casos de teste gerados é muito grande (1).

5.7.2 - O tamanho do conjunto é influenciado pela complexidade do grafo contextual. Quanto mais saídas de nós contextuais, mais casos de teste são gerados. (2).

8- Com a sua experiência, você acredita que conseguiria obter de forma independente os casos de teste que foram gerados com a ferramenta? Por quê?

Extração:

1 - Conseguiria se eu fosse buscar uma técnica de teste, eu ia lá olhar fazer um teste com caminho crítico, analisar as probabilidades dos testes e fazer isso manualmente, com um tempo muito maior. 2 - O maior problema é não ser tão repetível, não ter base de teste pra gerar de volta. Imagine que eu criei um cenário muito grande, fiz uma vez, encontrei o erro e corrigi... Como é que faz com aquele cenário lá grandão, você guarda. 3 - A vantagem está aí: de você poder executar essa base em qualquer momento, aí aquele cenário que passou legal, qual foi mesmo? Aí você tem os cenários OK e os cenários não OK, você roda o cenário não OK, aí está o lance da automatização, em todos os cenários OK, eu vou ter que analisar de novo, isso é ruim. 4 - Se a conferência fosse automática, eu posso até dar uma analisada se não deu um falso verdadeiro ali, aí o número de casos de teste que você gerou já não iria impactar, só seria o tempo de execução.

Codificação:

5.8.1 - Conseguiria gerar os casos de teste de forma independente, mas demandaria

muito mais tempo (1).
5.8.2 - Existe redundância nos casos de teste (2).
5.8.3 - Salvar os cenários é uma vantagem (2, 3).
5.8.4 - A conferência do comportamento da aplicação sob teste é feita visualmente pelo testador, e seria interessante se esta etapa fosse automatizada (3, 4).
5.8.5 - A automatização da conferência dos testes tornaria o número de casos de teste um problema de impacto relativamente baixo, pois apenas os casos em que ocorrem comportamentos não esperados seriam verificados. O tempo de execução passaria a ser o principal problema (4).

9- O que você acha sobre o uso de modelos da aplicação pelo método proposto?

Extração:
1 - O primeiro modelo da estrutura, eu achei um pouco confuso, mas beleza.
2 - O testador dá pra identificar mais ou menos os sensores que ele deve trabalhar, isso é legal e poderia ser substituído por uma lista, pra o cara no papel de testes, uma lista resolveria pra ele.
3 - O modelo do grafo contextual ajuda, e pelo que eu percebi ele ajuda a gerar os casos de teste, ele dá uma ideia do que é para fazer, mas isso não impacta tanto, porque querendo ou não o gerador já dá muito na mão todas as possibilidades.
4 - Talvez sincronizar isso para ficar mais claro em que ponto do diagrama, qual foi a condição que me levou a aquilo.
5 - Eu vejo isso antes do contexto lógico, tem o contexto do sensor em si, físico, e antes do contexto físico, criar simplesmente situações de OK ou não OK, e depois utilizar isto no contexto lógico, uma uma camada antes. Este vai estar com um OK e outro não OK. Eu poderia criar os estados para os sensores, porque lá no contexto [lógico] só mudaria os estados.

Codificação:
5.9.1 - Para o testador, o diagrama conceitual pode parecer confuso. (1, 2)
5.9.2 - Para o testador o grafo contextual passa apenas uma noção de como a aplicação se comporta (3).
5.9.3 - Falta uma interligação do grafo contextual com a ocorrência de defeito (4).
5.9.4 - Acho que outra abordagem de preenchimento de dados de sensores poderia para evitar repetição de digitação. Proponho uma etapa anterior ao preenchimento dos contextos lógicos (5).

10- Você gostaria de mencionar algo mais?

Extração:
1 - Não. Achei legal! Acho que é uma ferramenta que pode ser trabalhada para ajudar realmente os testadores, os caras que fazem um trabalho intelectual, e não ficar testando um a um.
2 - Se eu não me engano, lá em Manaus, o pessoal faz uns testes, geram algumas

coisas pra testar, tem uma pessoa que fica anotando em uma planilha, lá eles fazem um esquema de rodar um script, abrir e fechar aplicações, umas coisinhas no celular e ficam anotando em uma planilha.
3 - Acho que toda automação de teste é bem-vinda, até porque, tem possibilidades de amarrar e rodar isso. Realmente você tem uma base com esses testes e aplicar isso de forma automática. Quando a gente replicar os testes em um contexto maior, vale a pena.
4- Uma coisa que eu vejo de não aplicar só isso no emulador. Hoje eu vejo aplicativos na web, ele roda em n aparelhos, e em n definições de tela e te dá um relatório final do que é aquilo, ele traz captura dessas telas; ficaria mais interessante se conseguisse aplicar isso em n aparelhos, de forma automática.

Codificação:
5.10.1 - A ferramenta tem potencial para auxiliar testadores de fato (1).
5.10.2 - O uso de uma base de conhecimento reduz esforço na execução dos testes (3).
5.10.3 - Uma melhoria que proponho é a execução em dispositivos reais de forma semelhante a abordagens automatizadas já existentes (4).

Apêndice

C

DOCUMENTOS FORNECIDOS AOS PARTICIPANTES

Neste apêndice apresentamos os documentos fornecidos aos desenvolvedores e testadores de aplicativos móveis que foram participantes do estudo experimental.

Execução do experimento

Finalidade

As tarefas que serão executadas neste experimento têm como objetivo gerar de casos de teste para sistemas móveis que usem dados de contexto.
Os casos de teste gerados devem explicitar falhas na adaptação ao contexto da aplicação sob teste.
Os tipos de defeito que devem ser descobertos são os que estão descritos no ANEXO 1.

Atividades a serem executadas

1. Configuração e verificação do ambiente (Pré-experimento)

Esta fase serve para verificar se o ambiente de execução do experimento está devidamente configurado na máquina utilizada pelo participante.
Após a cópia e descompactação do pacote "dist", executar scripts numerados, seguindo a ordem:

1. Executar instalação de cliente telnet (apenas windows) – (1-install_telnet);
2. Executar instalação intel_x86 para o emulador Android – (2-install_intel_x86);
3. Executar o *CEManTIKA CASE* e abrir o *Workspace* exibido – (3-cemantika-case);
4. Navegar no projeto *NoCamInMeetingSpec* e acionar o ícone "C". Deve surgir a tela de navegação do *CEManTIKA CASE*;
5. Fechar *CEManTIKA CASE*;
6. Executar emulador Android (4-emulator);
7. Após o carregamento do sistema Android do emulador, executar instalador das *APKs* – (5-install-apks);
8. Executar, dentro do emulador, o app *SensorSimulator...*;
9. Acionar a tecla *Home* (círculo) do Emulador;
10. Executar o simulador de contexto *Context Simulator* – (6-context-simulator);
11. Executar, dentro do emulador, o app *NoCamInMeeting*;
12. Adicionar um contexto físico à timeline do simulador;
13. Acionar o botão Play no simulador e verificar se o dado de contexto da timeline é passado à aplicação no tempo correto.

2. Explorando Workspace no CEManTIKA CASE para contextualizar

Passos para explorar a documentação do projeto *NoCamInMeetingSpec* no *CEManTIKA CASE*:

1. Executar o *CEManTIKA CASE* e abrir o *workspace* exibido – (3-cemantika-case);
2. Navegar no projeto *NoCamInMeetingSpec* e acionar o ícone "C". Deve surgir a tela de navegação do *CEManTIKA CASE*;
3. Para visualizar o modelo conceitual de contexto (diagrama de classes UML), abrir o diagrama context.model.conceptual.**noCamInMeeting.umlclass**
4. Para visualizar o modelo comportamental (grafo contextual), abrir o diagrama cemantika.contextual.graph.**contextual_graph_0.rf**

3. Preparando os dados de contexto para a geração dos casos de teste

Esta é a etapa principal do experimento. Nela o participante, com apoio do método sob experimento, vai se conectar com uma base de conhecimento de contexto, extrair os dados de contexto presentes do modelo da aplicação, converter estes dados para uma representação de contexto de alto nível e combinar estes dados com padrões de defeitos em fontes de contexto. O resultado desta combinação é o conjunto de casos de teste que deve ser exportado para execução em um simulador de contexto.

Esta etapa é executada na aba *Testing* da ferramenta *CEManTIKA CASE*.

3.1. Conectando à base de conhecimento de contexto para teste

Para armazenar e tratar as associações de contexto com alto nível é preciso criar ou abrir uma base de conhecimento de contexto para teste (*CKTB* na sigla em inglês). Ao final de cada uma das próximas etapas executadas no *CEManTIKA CASE* esta base de conhecimento é enriquecida.

3.2. Identificando os contextos lógicos

Nesta etapa o método sob experimentação obtém os dados dos sensores presentes nas decisões contidas nos nós contextuais (círculos amarelos) do grafo contextual. Estes dados de sensores de baixo nível são combinados em contextos lógicos. Alguns destes sensores são afetados por padrões de defeito e exigem a instanciação de contextos lógicos com estes padrões de defeito.

Nesta etapa o participante deve preencher os dados de sensores para os mais diversos contextos lógicos extraídos do grafo contextual.

3.3. Identificando as situações

Nesta etapa todos os caminhos entre o nó inicial e o nó final do grafo contextual são mapeados. Cada caminho do grafo contextual é convertido em uma situação com os contextos lógicos associados. Cada ação (retângulo verde) do grafo contextual é identificada como parte do comportamento esperado da situação descoberta.

O participante deve renomear as situações obtidas e pode complementar os comportamentos esperados da aplicação.

Obs: Atentar que a ferramenta só altera a situação depois de acionar o botão Apply.

3.4. Gerando os cenários base

Nesta etapa realiza-se a associação entre as situações. Para o método sob experimentação, a relação definida entre as situações é a temporal. Ou seja, o participante deve encadear as diversas situações identificadas na etapa anterior no decorrer do tempo.

O resultado deste encadeamento temporal caracteriza um cenário. Este cenário elaborado conforme o método proposto é denominado cenário base, pois sobre ele serão aplicados padrões de defeitos de fontes de contexto (explicado de forma detalhada na próxima etapa). Uma mesma situação pode ser encadeada diversas vezes num cenário base, de forma repetida ou alternada com outras situações.

Obs: Atentar que a ferramenta só altera o cenário em edição depois de acionar o botão *Apply*.

3.5. Gerando os casos de teste

Nesta etapa ocorre a geração dos casos de teste. Para gerar os casos de teste, o participante deve escolher um cenário base e selecionar os padrões de defeitos que serão combinados com o cenário.

Existem 2 tipos de padrões de defeitos proposto pelo método: 1) O padrão de defeito é aplicado sobre um sensor mapeado no cenário base, ou seja, depende dos sensores mapeados nas etapas anteriores. Ex: Imprecisão no sensor de GPS; 2) O padrão de defeito é aplicado de forma independente dos sensores presentes no modelo da aplicação. Ex: Bateria indica carga baixa.

Após escolher o cenário base e os tipos de padrões de defeitos a serem aplicados ao cenário base o participante gerar e exportar os casos de teste.

Para evitar uma explosão combinatória e inviabilizar a execução dos casos de teste gerados, o método sob experimentação propõe o uso de um método de redução de casos de teste. Sobre o conjunto de casos de testes gerados aplica-se uma função de similaridade. Como resultado, os casos de teste mais semelhantes entre si são removidos, permanecendo na suíte de testes final apenas os casos de teste mais divergentes entre si.

Ao final da redução, os casos de teste gerados são exportados.

4. Preparando emulador Android para testar a aplicação NoCamInMeeting

Após exportar os casos de teste, o participante pode iniciar os procedimentos para executá-los. O primeiro passo para isto é a configuração do emulador Android, a qual ocorre nos passos a seguir:

1. Executar emulador Android(4-emulator);
2. Após o carregamento do sistema Android do emulador, executar instalador das *APKs* – (5-install-apks);
3. Executar, dentro do emulador, o app *SensorSimulator*...;
4. Acionar a tecla Home (círculo) do Emulador para retornar à tela inicial do Android.

5. Executando o simulador de contexto Context Simulator

O participante deve executar o simulador de contexto Context Simulator – (6-context-simulator).

6. Execução dos casos de teste

Com o emulador Android e o simulador de contexto em execução, o participante deve executar, dentro do emulador, o app sob teste.
No Context Simulator, o participante deve importar os casos de teste gerados pelo CEManTIKA CASE, através da opção File → Import.
Após importar os casos de teste, as seções correspondentes aos cenários, situações e contextos lógicos estarão preenchidas.
Para preparar um cenário para execução, basta escolher um destes na seção correspondente e adicioná-lo à *Timeline* do simulador.
Para executar o cenário adicionado à *Timeline*, basta acionar o botão de reprodução (Play). Neste momento o simulador começará a enviar dados de contexto para a aplicação em execução no emulador.
Para remover todos os cenários da *Timeline* do simulador, o participante deve pausar a execução do cenário acionar o botão abaixo do botão de reprodução.

Caso o participante execute um cenário simulando problema de interrupção (Ex: bateria baixa), e depois o mesmo deseje executar outro cenário sem este problema, o participante deve remover o cenário em questão, adicionar à Timeline um contexto físico que reverta a interrupção (Ex: bateria a 50%), executá-lo, removê-lo e partir para o próximo caso de teste.

7. Descobrindo defeitos

Tendo o participante gerado os casos de teste e iniciado a execução de um cenário, como o participante poderá inferir se ocorreu ou não um defeito na aplicação?
A resposta para esta pergunta é simples: basta o participante comparar o comportamento esperado para a situação com o que é apresentado pela aplicação em execução no simulador de contexto. No caso de divergência entre os 2, um defeito de fonte de contexto foi descoberto.
O participante pode pausar a execução do simulador para fixar o dado de contexto enviado para a aplicação sob teste e assim avaliar calmamente as informações apresentadas, tanto pelo simulador quanto pela aplicação sob teste.

8. Registrando defeitos no formulário de defeitos

Na execução do experimento, todo defeito relacionado à aplicação sob teste, ou seja, comportamento diferente do esperado numa determinada situação deve ser relatado.
Para cada defeito encontrado na execução do caso de teste, o participante deve preencher uma linha do Formulário de defeitos encontrados, o qual foi entregue ao participante no início do experimento. As instruções de preenchimento estão contidas no próprio formulário.

Formulário de defeitos encontrados

Relatório de defeitos

Participante:_____

Nome da aplicação: *NoCamInMeeting*

Defeitos			
N°	Sensor	Padrão de defeito[1]	Descrição geral[2]

[1]Assinalar na coluna de Padrão de defeito:
UNV – para Incomplete – Unavailability
GRN – para Granularity Mismatch – Imprecision
DTN – para Slow Sensing – Out-of-Dateness
INT – para Problematic Rule Logic – Wrong Behavior Caused by Interrupts

[2]Preencher coluna Descrição Geral com texto simples:
Ex1: “Aplicação encerrou inesperadamente”
Ex2: “Comportamento esperado era 'apresentar cursor de posição no mapa' e aplicação 'não apresentou o cursor no mapa'”

ANEXO 1 – Defeitos em fontes de contexto

Fault: Incomplete

Failure: Unavailability	
Example	Test Case
Sensor cannot receive data: No network access.	*Precondition*: Context source is available *Action*: Disable context source *Postcondition*: Context source is unavailable

Fault: Granularity Mismatch

Failure: Imprecision	
Example	Test Case
It is almost impossible, to receive same GPS information at one spot within two requests, because GPS is not exactly.	*Precondition*: One context source transmits data *Action*: Change the simulated data slightly and send it again *Postcondition*: Same context source sends updated data, even when the actual context has not changed

Fault: Slow Sensing

Failure: Out-Of-Dateness [45]	
Example	Test Case
An image from a security camera which was taken 2 hours ago is not useful anymore.	*Precondition*: Application requests data from a context source *Action*: Select another (not the excepted) value for the context source, to simulate old values *Postcondition*: Simulator transmit "old" data information

Fault: Problematic Rule Logic

Failure: Wrong Behavior Caused by Interrupts	
Example	Test Case
An application pauses, because battery level reaches a critical level.	*Precondition*: Mobile application is running *Action*: Simulate an interrupt event *Postcondition*: Interrupt influences the mobile application

Fonte das tabelas: Hassel, M., Context Simulator For Testing Mobile Applications. Dissertação de mestrado. University of Applied Sciences Mannheim, Março de 2014

UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE MATEMÁTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

O Sr.(a) está sendo convidado(a) como voluntário(a) a participar da pesquisa "**CEManTIKA Test Creator: Gerando Casos de Teste para Sistemas Sensíveis ao Contexto**". Nesta pesquisa pretendemos **"utilizar gravações de voz e artefatos de teste de software para avaliar um método de geração de casos de teste em aplicativos móveis".** O motivo que nos leva a desenvolver e estudar este método é **"que a elaboração de casos de teste com dados reais de sensores para dispositivos móveis é uma atividade que é cara em termos de consumo de recursos, e o método proposto visa prover estes dados de forma semi-automatizada para execução em ambiente simulado"**. Para esta pesquisa adotaremos os seguintes procedimentos: **Coletaremos artefatos elaborados durante a realização do experimento, como casos de teste e um formulário de defeitos detectados. Além disso, coletaremos o áudio de uma entrevista.** É importante notar que **não há riscos físicos para os participantes pois utilizar uma ferramenta para gerar dados simulados de sensor não é um processo que possa causar dor ou machucar o participante. O que pode ocorrer seria algum desconforto, angústia ou até mesmo ansiedade por parte do participante principalmente com relação ao respeito à privacidade e sigilo das informações.** Neste projeto **não há previsão de ressarcimento, pois o Sr.(a) não terá nenhum custo financeiro para participar desta pesquisa, nem receberá qualquer vantagem financeira.** Terá o esclarecimento sobre o estudo em qualquer aspecto que desejar e estará livre para participar ou recusar-se a participar. Poderá retirar seu consentimento ou interromper a participação a qualquer momento. A sua participação é voluntária e a recusa em participar não acarretará qualquer penalidade ou modificação na forma em que é atendido pelo pesquisador, que tratará a sua identidade com padrões profissionais de sigilo.

Os resultados da pesquisa estarão à sua disposição quando finalizada. Seu nome ou o material que indique sua participação não será liberado sem a sua permissão. O(A) Sr.(a) não será identificado em nenhuma publicação que possa resultar. Este termo de consentimento encontra-se impresso em duas vias originais, sendo que uma será arquivada pelo pesquisador responsável, e a outra será fornecida ao senhor(a). Os dados e instrumentos utilizados na pesquisa ficarão arquivados com o pesquisador responsável. Os pesquisadores tratarão a sua identidade com padrões profissionais de sigilo, atendendo a legislação brasileira (Resolução Nº 466/12 do Conselho Nacional de Saúde), utilizando as informações somente para os fins acadêmicos e científicos.

O(a) participante, _____, portador do Documento de Identidade _____ foi convidado a participar da pesquisa **"CEManTIKA Test Creator: Gerando Casos de Teste para Sistemas Sensíveis ao Contexto"** e foi informado(a) dos

objetivos de maneira clara e detalhada.

No entanto, caso apresente outras dúvidas, em qualquer momento da pesquisa, você pode contatar o responsável (André Luís Monteiro Pacheco dos Santos) pelo telefone 071/98109-1985 ou por e-mail andreluismps@gmail.com.

Salvador, _____ de _____ de 20____.

Nome	Assinatura participante	Data

Nome	Assinatura pesquisador	Data