Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Linha de Produto de Software Dinâmica Ciente de Qualidade

Carolina Sousa Rocha de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Vander Ramos Alves

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Maristela Terto de Holanda

Banca examinadora composta por:

Prof. Dr. Vander Ramos Alves (Orientador) — CIC/UnB
Prof. Dr. Genaina Nunes Rodrigues — CIC/UnB
Prof. Dr. Rodrigo Bonifacio de Almeida — CIC/UnB

CIP — Catalogação Internacional na Publicação

de Oliveira, Carolina Sousa Rocha.

Linha de Produto de Software Dinâmica Ciente de Qualidade / Carolina Sousa Rocha de Oliveira. Brasília : UnB, 2013.
83 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. Linha de Produto de *Software* Dinâmica, 2. Qualidade, 3. Segurança, 4. Eficiência

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Linha de Produto de Software Dinâmica Ciente de Qualidade

Carolina Sousa Rocha de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Vander Ramos Alves (Orientador)
CIC/UnB

Prof. Dr. Genaina Nunes Rodrigues
CIC/UnB

Prof. Dr. Rodrigo Bonifacio de Almeida
CIC/UnB

Prof. Dr. Maristela Terto de Holanda
Coordenador do Bacharelado em Ciência da Computação

Brasília, 6 de julho de 2013

Dedicatória

Dedico aos meus pais que me derem suporte a esta longa, longa, jornada que foi a graduação.

Agradecimentos

Agradeço a todos que estiverem envolvidos nesta minha jornada.

Abstract

Sistemas cientes de contexto são muito influenciados por mudança no seu contexto que não são fáceis de prever. Suas configurações tem que estar constante avaliaçõa para checar se ainda satisfazem os requisitos de qualidade. Técnicas tradicionais de desenvolvimento não são adequadas para este tipo de software por causa da necessidade de adaptação durante o tempo de execução. A monitoração de sinais vitais humanos entra neste contexto, pois dependendo dos dados coletados, o sistema tem que se adequar para auxiliar o paciente de forma correta. Adicionalmente esses sistema tem que ser eficiêntes e garantir a segurança deste. No trabalho de Paula Fernandes [11] foi proposto um método que lida com todas essa variáveis. Este mé todo é baseado em linha de produto de *software* dinâmicas cientes de qualidade, porém nunca foi avaliado nos aspectos de eficiência e segurança. Este trabalho então propõe uma avaliaça das propriedades destacas e ainda constroi uma formalização para o conceito de segurança neste contexto.

Palavras-chave: Linha de Produto de *Software* Dinâmica, Qualidade, Seguranç, Eficiência

Abstract

Context aware systems have their environment influenced by context changes that are not easy to predict. Their configurations have to be in constant evaluation to see if the corresponding functional and non-functional requirements are satisfied. Traditional development techniques are unsuitable for this kind of software, especially because of the great need of adaptation during runtime. Vital data monitoring is a problem that fits this context because it has to respond to all health situations indicated by the data collected and adapt itself during runtime to fulfill all the current needs of the monitored person. Additionally, this monitoring system has to guarantee the safety of the patient and all the requirements established by a domain specialist. In her work, Paula Fernandes [11] developed a method based on quality-aware dynamic software product line. This method was never evaluated in the efficiency and safety aspect, and that is what this work proposes. Besides evaluating the method, it is also formalize the safety property in this context.

Keywords: Dynamic Software Product Line, Quality, Safety, Efficiency

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos dias atuais utilizamos cada vez mais a tecnologia para auxiliar no cotidiano. Seja em pequenas tarefas, como nos fazer lembrar de um compromisso, ou um despertador para acordarmos na hora certa, ou em tarefas de extrema importância, como marcapassos, os sistemas de *software* estão sempre presente, e cada vez mais precisam se adaptar a cada indivíduo e suas necessidades.

Uma das áreas em que a utilização de componentes tecnológico mais cresce é no ambiente de vida assistida. Estes sistemas hoje auxiliam a vida de várias pessoas comoidosos, deficientes, entre outros, e devolvem a liberdade de poderem viver em suas próprias casas e ainda terem uma equipe (sistema + administradores) de plantão para ajudá-los. Porém um dos grandes desafios é que, para cada deficiência, para cada indivíduo, as necessidades se alteram, e se torna inviável desenvolver um sistema única para cada um. A solução é na verdade criar um sistema que se adapta a cada indivíduo que irá suprir todas as necessidade daquele contexto no qual ele irá agir.

Sob esta proposta, foi desenvolvido pela mestranda Paula Fernandes [11], da Universidade de Brasília, um sistema de monitoramento de sinais vitais do corpo humano. Este sistema utiliza diversos sensores para obter informações constantes do paciente. Essas informações são repassadas para um celular que interpreta os dados e reage de forma a sempre buscar o auxilio adequado a pessoa assistida. Por exemplo, se for notado um aumento na pressão, que indique pressão acima do normal, o sistema do celular pode enviar um alerta ao médico responsável pelo paciente, ou dar um aviso ao próprio paciente que pode tomar as medidas necessárias para a situação. O sistema foi desenvolvido utilizando a abordagem de linha de produto de *software* dinâmica, assim ele se adapta em tempo de execução às diferentes situações encontradas ao ler os dados.

1.1 Problema

Na área médica, qualquer sistema deve garantir a segurança do paciente, ou seja, garantir que nada de ruim venha a acontecer durante a execução do programa com os envolvidos. Deve ser também um sistema eficiênte, pois sua resposta deve ser rápida para atender sempre as necessidades do paciente. Nestes aspectos, o método proposto no trabalho de Paula Fernandes [11] nunca foi avaliado.

1.2 Metodologia

Este tabalho propõe e desenvolve uma avaliaç ao sobre o ponto de vista da eficiência e seguranç do método prosposto em [11]. É desenvolvida também a formalização da propriedade de segurança neste contexto.

1.3 Contribuições

A solução proposta tem as seguintes contribuições:

- Formalização da propriedade de segurança;
- Avaliação do método sobre o ponto de vista da eficiência;
- Avaliação do método sobre o ponto de vista da segurança.

1.4 Descrição dos Capítulo

No capítulo 2 é encontrado a fundamentação teórica necessária para o entendimento do trabalho. No capítulo 3 é descrito a atual arquitetura do sistema desenvolvido e no capítulo 4 é feita a formalização da propriedade de segurança. No capítulo 5 é discutido a montagem dos cenários para a avaliação do método. O capťulo 6 discute os resultados obtidos e o capítulo 7 apresenta a conclusão do trabalho.

Capítulo 2

Fundamentação Teórica

Neste capítulo é apresentada a fundamentação teórica para o entendimento deste projeto.

2.1 Linha de Produto de Software

O desenvolvimento de softwares novos sempre foi um desafio para empresas, já que este envolve não somente a codificação de um novo produto, mas sua manutenção e possíveis evoluções. Uma agregação de uma nova funcionalidade a um produto, pode acarretar uma reformulação tão grande de um software, que talvez seja mais natural para empresa recriar o produto, porém esse novo software teria a maioria de suas funções iguais ao anterior. Assim, temos a necessidade de buscar o reuso de códigos e tornar cada vez mais fácil a evolução destes produtos. desta forma, surge o conceito de Linha de produto de software, definido por Borba et. al. [6]:

> **Definição:** Uma Linha de produto de software é um conjunto de produtos relacionados que são gerados a partir de componentes reutilizáveis. Os produtos se relacionam de forma que possuem funcionalidades em comum.

Podemos definir também uma linha de *software* em relação ao seus componentes, chamados de *features* (por Kolesnikov et. al. [15]):

> **Definição:** Uma linha de produto de *software* é uma família de produtos que compartilham artefatos em comum. Os produtos se diferenciam em termos de *features*. A *feature* é um comportamente final, que pode ser visto por um usuário, que satisfaz um requerimento de um *stakeholder*.

Assim, uma linha de produto de software (LPS) é um conjunto de programas que compartilham funcionalidades. Ela é composta por artefatos, que podem ser reutilizados na produção de novos produtos, que representam tanto as comunalidades entre os diferentes produtos a serem gerados como também algum ponto de variação. Cada artefato se relaciona com uma funcionaldade do programa final, assim, temos um mapeamento entre um artefato e uma funcionalidade, que é chamado de *Configuration Knowledge*. Na literatura, as funcionalidades são chamadas de *features*, e a relação entre estas é descritas por meio de um *modelo de features*. Um produto final que pertence a uma linha de

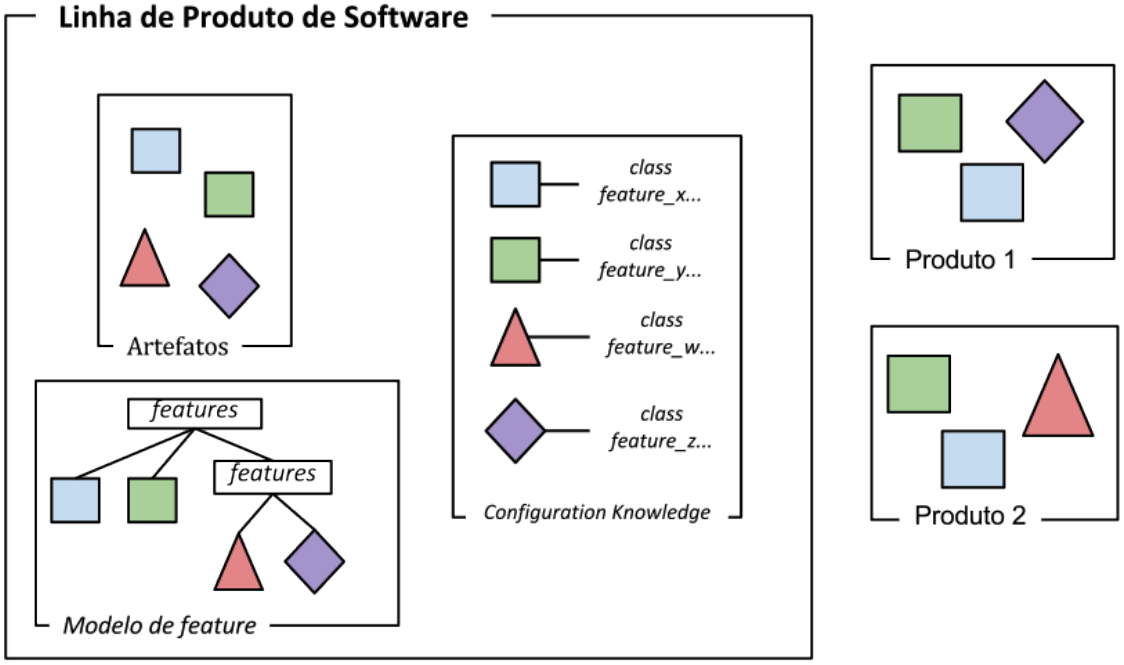


Figura 2.1: Exemplo de formação de uma linha de produto de *software*.

produto de software é, então, a seleção e composição de um conjunto de *features*. Por isso, uma LPS também é chamada de família de software. A figura 2.1 exemplifica uma linha de produto de softwares e dois produtos gerados por ela(artefatos representados por quadrados indicam as funcionalidades em comum dos dois produtos gerados, enquanto os demais representam uma funcionalidade específica).

Defini-se então uma linha de produto de software como um tupla de três componentes: o conjunto de artefatos; o modelo de *features*; e o *configuration knowledge*. E cada produto desta LPS é um configuração distinta de seleção das *features* que a compõe.

2.1.1 Representação do Modelo de *Features*

Uma linha de produto de software é normalmente descrita por meio de suas *features*. Desta forma, o modelo de *features*, que descreve a forma como cada *feature* da LPS se relaciona com as demais e com o produto final, tem grande importância para seu entendimento. Uma das suas representações mais comum é por meio de uma árvore de *features*. Nesta árvore temos o nó raiz que está sempre presente na configuração final de um produto. Os demais nós podem representar uma única *feature* ou um conjunto de *features*. A figura 2.2 traz uma imagem exemplo de uma árvore de *features*. Esta árvore representa o modelo de *feature* da linha de produto de software utilizada neste projeto.

As relações representadas no modelo podem dizer respeito a uma única *feature* em uma configuração de um produto final, como por exemplo, se a *feature* deve pertence ou não ao produto de forma obrigatória. Ou pode ser aplicada à um conjunto de *features* criando

restrições aplicadas as *features* pertencentes a este grupo. Por exemplo, se a *feature1* e *feature2* pertencem a um mesmo grupo podemos aplicar uma relação que indica que a se *feature1* for selecionada, *feature2* não pode ser selecionada. Existem vários modelos diferente com diversas relações, porém, para este trabalho, foram utilizadas somente as quatro descritas abaixo:

Em Relação a uma *feature* temos as seguintes relações:

Relação Opcional (*optional*): Uma *feature* pode ou não ser selecionada em uma configuração.

Relação Obrigatória (*mandatory*): Uma *feature* deve ser selecionada em todas configurações.

Em Relação a um grupo de *features*:

Relação OU (*or*): *Features* pertencente à um mesmo grupo de relação ou possuem a restrição de que pelo menos uma delas deve estar presente na configuração final do produto.

Relação Alternativa (*alternative*): *Features* pertencente à um mesmo grupo de relação alternativa possuem a restrição de que somente uma delas deve estar presente de forma única e obrigatória na configuração final do produto.

Como as *features* podem ser melhor visualizadas na árvore de *features*, foram criadas representações para cada realação na árvore. A representação gráfica de cada relação pode ser vista na figura 2.2. A relação *And* não é utilizada pois ela é substituida pelas relações obrigatórias e opcionais.

Além das relações descritas na árvore, pode-se colocar fórmulas lógicas que adicionam novas restrições ao modelo, chamadas de restrições *cross-tree*. Na figura figura 2.2 temos 3 exemplos, um deles é a linha $Oxygenation \rightarrow SPO2$ que indica que, se a *feature* *Oxygenation* for selecionada, então a *feature* SPO2 também deve ser.

Features Abstratas

Uma *feature* abstrata é uma *feature* que não representa nenhum artefato em uma linha de produto de sfotware. Ela nada mais é do que uma *feature* junta outras em um grupo que possuem uma semelhanća, seja l ógica, funcional, ou outros. Por isso seu objetivo é puramente visual, tornando a interpretação e o entendimento da imagem de uma árvore de *feature* e as realações presentes muito mais fácil.

No exemplo da figura 2.2 somente os nós folhas (aqueles que não possuem nenhum nó filho) são as verdadeiras *feature* que representam um artefato na linha de produto de *software*. Os nós *Monitoring, Storage, SensorInformation, FormInformation* e *Sensor* são *features* abstratas.

2.1.2 Linha de Produto de Software Dinâmica

Uma linha de produto de software padrão constroi um produto final por meio da seleção de um conjunto de *features*, a integração dessas, e por fim a e entrega do software. Porém, alguns sistemas possuem funcionalidades que devem ser mudadas em tempo de execução.

Por exemplo, podemos pensar em um linha de produto de software simples que gera software de conversão de medidas de comprimento. Assim, podemos converter metros para polegadas, pés, ou outras unidades de medidas (esse software poderia ser codificado

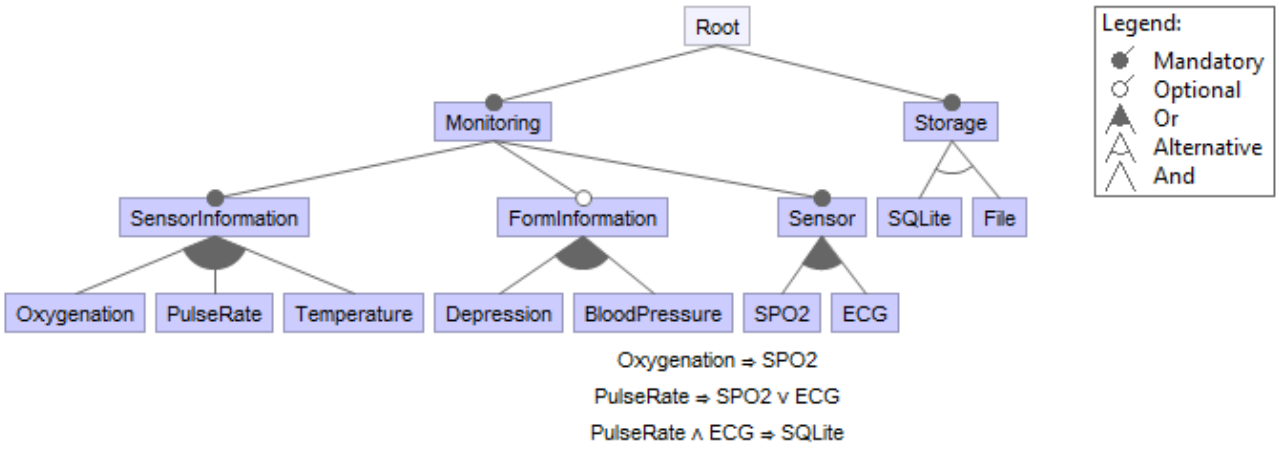


Figura 2.2: Exemplo de um modelo de *features*(em um árvore de *features*). Legenda define os tipos de relações entre as *features* [11].

sem a utilização de uma LPS, porém este é só um exemplo para ilustrar o problema). Então para cada unidade de medida teríamos uma *feature* diferente, e cada software final teria que escolher uma configuração que combinasse sempre uma unidade que deve ser convertida e a unidade para conversão. Dessa forma, para cada par de unidades acabaríamos gerando um programa diferente e se o usuário desejasse converter para um unidade diferente da escolhida, seria necessário utilizar outro programa. Mas nós sabemos que todas as funcionalidades para conversão para outras unidades já estão presentes na LPS, assim, se o programa pudesse se reconfigurar de formar que ele conseguisse utilizar a funcionalidade de conversão de outras unidades, não seria necessário para o usuário ter que utilizar outro programa para o cálculo.

Com este exemplo, percebe-se uma habilidade interessante que uma linha de software pode possuir, que é poder se reconfigurar de forma a se adequar a necessidade atual do usuário. Essa habilidade foi incorporada as LPS surgindo assim as linhas de produto de software dinâmicas (LPSD).

Uma LPSD também é descrita pela mesma tupla que descreve uma LPS, o que irá mudar é justamente o processo de criação do software final, que não estará mais restrito a uma única configuração. Agora, todo software inicia com uma configuração desejada, porém ele pode se reconfigurar, ainda seguindo as regras de relação de *features*, e se adequar aos novos requisitos do sistema. Esta abordagem constrói softwares mais dinâmicos e as vezes mais completos e que satisfazem áreas que a LPS padrão não conseguia preencher. Mas também existem preços a pagar, como a complexidade de programação e o impacto que uma reconfiguração pode levar no sistema, já que temos relações entre features que podem ser extremamente restritivas. A figura 2.4 mostra um exemplo de uma linha de produto de *software* dinâmica onde temos um produto (produto1) derivado dela que quando recebe um evento se reconfigura para atender aos novos requisitos do contexto.

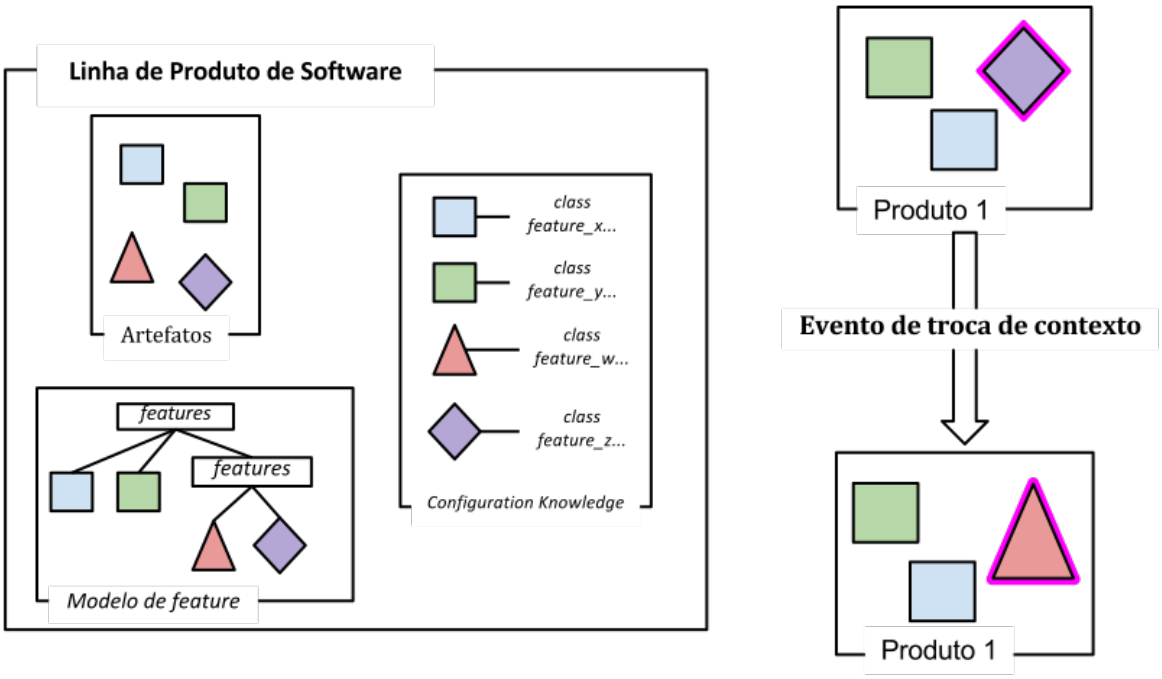


Figura 2.3: Exemplo de um modelo de uma linha de produto de *software* dinâmica onde o produto 1 se reconfigura em tempo de execução.

2.1.3 Reconfiguração de uma LPSD

Como existe um programa que pode se reconfigurar, adicionamos também o problema de encontrar um nova configuração que satisfaça todos os novos requisitos. No exemplo do conversor de medidas, teríamos que achar uma nova configuração que conseguisse fazer a nova conversão, ou seja, selecionar a *feature* que conseguisse realizar o novo cálculo e deselecionar a *feature* antiga. Porém, como dito antes, esse é um exemplo extremamente simples. Para sistemas mais complexos, o modelo de *feature* e suas relações são mais complexas, exigindo assim um estudo sobre como encontrar essa novas configuração de forma rápida e confiável. Umas das abordagens para achar uma configuração é transformar essa busca em um *Satisfiability Problem - SAT* misturado com um *Constraint Satisfaction Problem*, transformando o modelo de *features* em um conjunto de fórmulas proposicionais lógicas e buscando uma valoração que deixa a fórmula verdadeira, obdecendo as restrições impostas no modelo de *features*. Para isso, é preciso entender como se transforma um modelo de *feature* em um conjunto de fórmulas lógicas proposicionais.

2.2 Escolha de uma nova configuração

O problema da escolha de uma nova configuração para o sistema que satisfaça todas as necessidades do contexto atual é um problema grande e deve ser tratado individualmente.

Para cada conjunto de requisitos pode-se encontrar mais de uma configuração que o satisfaça e por isso é necessário adotar estratégias que iram filtrar essas configurações e

por fim decidir qual a melhor. Neste caso, existem diversas abordagens. Se queremos que o sistema tenha o melhor tempo de resposta possível, podemos escolher sempre a primeira solução encontrada e parar a busca. Se quisermos a melhor solução possível, é provável que seja necessário obter todas as configurações possíveis e por fim verificar qual a melhor. Mas vemos que essas buscas podem ser inviáveis. Para o exemplo da tabela **??**, temos somente três variáveis o que nos leva a 2^3 configurações possíveis. Para um modelo de *features* com um número n de *features* teremos um número 2^n possibilidades, que podem ser impossíveis de serem analisadas em tempo aceitável.

E por outro lado existe também o caso de não se encontrar uma configuração aceitável. Se então escolhermos sempre o modelo de a primeira configuração encontrada é a melhor, seria necessário percorrer todas as 2^n configurações para descobrir que não existe uma.

Então é necessário estabelecer um grupo de requisitos que devem levar em consideração o sistema para o qual ele foi desenvolvido. No caso deste projeto, trabalhamos com um sistema de monitoramento de sinais vitais, então o tempo de respota da configuração do sistema é de extrema importância, assim como é necessário que a nova configuração seja a melhor possível para não se perder dados importantes no contexto atual do sistema.

2.3 Sistema de Monitoramento de Sinais Vitais por Sensores

Para este projeto foi utilizado um sistema de monitoramento de sinais vitais por meio de sensores. Este sistema é arquitetado como uma Rede de Sensores, que é um sistema distruibuído onde cada sensor funciona de maneira independente. A rede possui ainda um sensor de recebimento de dados que é encarregado de receber os sinais de todos os outros sensores.

Para garantir o bom funcionamento da Rede, é necessário que o funcionamento de cada sensor individual não interfira com os outros, ou seja, se um sensor para de funcionar, os demais não podem ser prejudicados. E para comunicação ser realizada, são criados protocolos de comunicação interna e externa. O protocolo de comunicação interna serve para ligar os sensores ao sensor de recebimento de dados. Este pode fazer requisições de dados ou enviar comandos que alteram o comportamento dos sensores. Por exemplo, pode ser enviado um comando que aumente a taxa de amostragem de um determinado sensor, ou um comando para desligar um sensor, etc. O protocolo de comunicação externa serve para que um software externo posso ter acesso aos dados obtidos e por fim fazer a análise, assim como enviar para o sensor pedidos de mudanças nos sensores da Rede. Como o software externo não tem acesso direto a cada sensor, o sensor de recebimento de dados trabalha como uma ponte entre os dois.

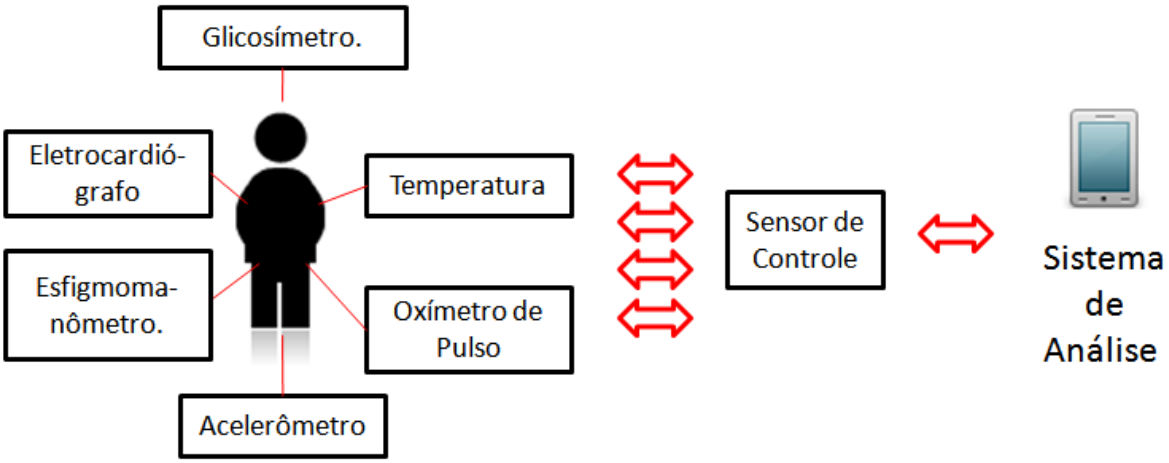


Figura 2.4: Exemplo de uma Rede de Sensores do Corpo Humano [11].

Capítulo 3

Linha de Produto de *Software* Ciente de Qualidade

Neste capítulo é introduzido a arquitetura do sistema de monitoração de sinais vitais do corpo humano desenvolvido por Fernandes [11]. Entra-se em detalhes de como a busca de uma nova configuração é que satisfaça o contexto requisitado é feita. É feita também uma abordagem geral de como funciona o sistema como um todo. Neste capítulo tambe é explicado como o cáculo de qualidade para uma determinada configuraç ao válidade de uma linha de produto de *software* é feita, baseado na técnica desenvolvida por Nunes [18].

3.1 Estados do Sistema

O sistema de monitoração de sinais vitais do corpo humano é orientado à três estados de riscos: Baixo Risco (BR), Médio Risco (MR) e Alto Risco (AR) de risco. Cada estado exige um nível de qualidade específico para que uma configura ccão seja aceitável em seu contexto. Esse nível de qualidade é crescente, ou seja BR $<$ MR $<$ AR.

As transições entre os estados podem ocorrer de todas as formas, como mostra a figura 3.1. A figura consiste em um autômato determinístico finito (ADF) que é definido formalmente como uma 5-tupla $A = (Q, \Sigma, \delta, q_0, F)$. Q é o conjunto de estados do ADF, Σ é o alfabeto, *delta* é a função de transição entre os elementos ($delta : Q \times \Sigma \rightarrow Q$), q_0 é o estados inicial e F é o conjunto de estados finais(Hopcroft et al [12]).

Para o caso do sistema implementado, temos que o ADF é definido, como descrito por Fernandes [11]:

- $\boldsymbol{Q}$: conjunto de estados de risco de um indivíduo monitorado. $Q = BR, MR, AR$
- Σ: conjunto dos eventos identificados pelo contexto que modificam o estado de risco de saúde,
- δ: função de transição que mapeia o relacionamento entre os estados de riscos e eventos.
- q_0: estado inicial – Alto Risco (AR)
- $\boldsymbol{F}$: estados finais – $F = Q$

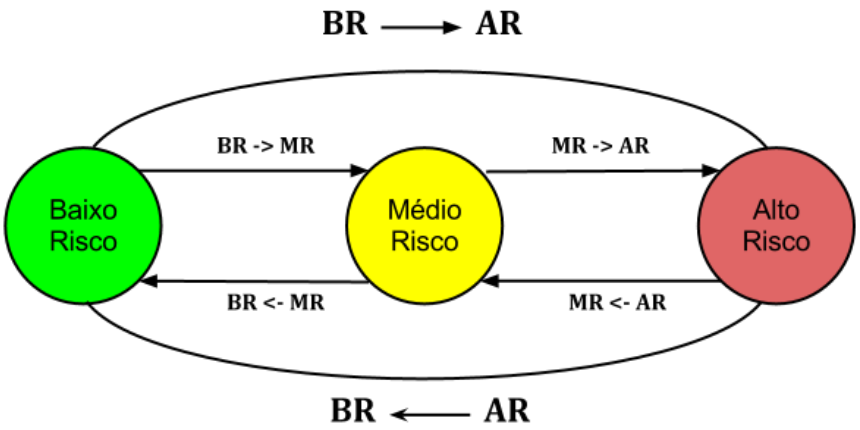


Figura 3.1: Exemplo de formação de uma linha de produto de *software* [11].

Tabela 3.1: Tabela de interpretação para valores de batimentos cardícos para um indivíduo genérico e em repouso [14]

Pulsação Cardíaca (bpm)	**Evento**
$x > 180$	Taquicardia Alta
$140 \leq x \leq 139$	Taquicardia Moderada
$110 \leq x \leq 139$	Taquicardia Baixa
$70 \leq x \leq 109$	Normal

3.2 Gramática

A gramática desenvolvida é usada para interpretação dos dados e geração de eventos. Ela leva em consideração o estado atual do indivíduo e os dados recebidos pelo sistema no dispositivo móvel. Estas interpretações são feitas então em duas etapas: primeiramente se analisa os dados recebidos e quais eventos ele gera; segundo com os eventos se analisa o estado atual do sistema (BR, MR, AR) e gera um novo evento de mudança de estado. As tabelas 3.1 e 3.2 mostram um exemplo das duas etapas de interpretação dos dados. A tabela 3.1 mostra como a gramática interpreta no momneto inicial os dados recebidos pelo coordenador dos sensores. Ele faz uma classificação inicial do evento que este irá gerar. A tabela 3.2 mostra como o sistema leva em consideração o estado atual e o evento recebido para gerar um evento de mudança de estado.

3.3 Rede de Sensores

A rede de sensores do sistemas de monitoração de sinais vitais do corpo humano é composta por 5 sensores e um coordenador.

Tabela 3.2: Tabela de interpretação para evento recebido dependendo do estado atual de risco [7]

Estado Atual	Evento	Novo Estado
Risco Alto	Taquicardia Moderada	Risco Médio
Risco Médio	Braquicardia Baixa	Risco Baixo
Risco Baixo	Taquicardia Alta	Risco Risco Alto

- **Acelerômetro**: São dois sensores que servem para medir a posiç ao do paciente. Ele trabalha para identificar eventos de queda ou posição do indivíduo, como por exemplo deitado ou de pé.
- **Temperatura:** Um único sensor que tem como função medir a temperatura corporal do paciente.
- **Eletrocardiógrafo:** Um único sensor que mede os batimentos cardíacos do paciente.
- **Oxímetro:** Um único sensor que mede a oxigenação do sangue do paciente.
- **Coordenador:** Encarregado de receber o sinal dos sensores e enviar para o sistema no dispositvo móvel. A troca de dados entre o sistema e o coordenador é via *bluethooth* por meio dos protocolos implementados pela empresa desenvolvedora dos sensores.

3.4 Sistema de Monitoramento na Plataforma Android

O sistema desenvolvido para monitoramento dos sinais vitais foi feito para a plataforma *android* para rodar em dispositivos móveis(tablets, celulares, entre outros). Ele se comunica via *bluetooth* com o coordenador de dados dos sensores. Os dados são recebidos na forma bruta, ou seja, sem nenhuma análise realizada. Cabe então ao sistema interpretar os dados, tomar as atitudes baseadas na análise e continuar a receber novos dados do sistema de sensores. A figura 3.3 descreve a arquitetura de referência utilizada no desenvolvimento do sistema.

- **Sensor**: Todos os dados de baixo nível, medidos por sensores são recebidos pela camada de *software*. Dados de baixo nível que não precisam de processamento são enviados diretamente à camada de controle.
- **Percepção**: Interpreta os dados da camada de sensor. Classifica os dados brutos medidos por serviços externos em eventos.
- **Identificação**: Faz a fusão dos dados e a interpretação dos eventos em termos de risco.
- **Planejamento**: Recebe o evento da cama de identificação e verifica se é necessária mudança no sistema.

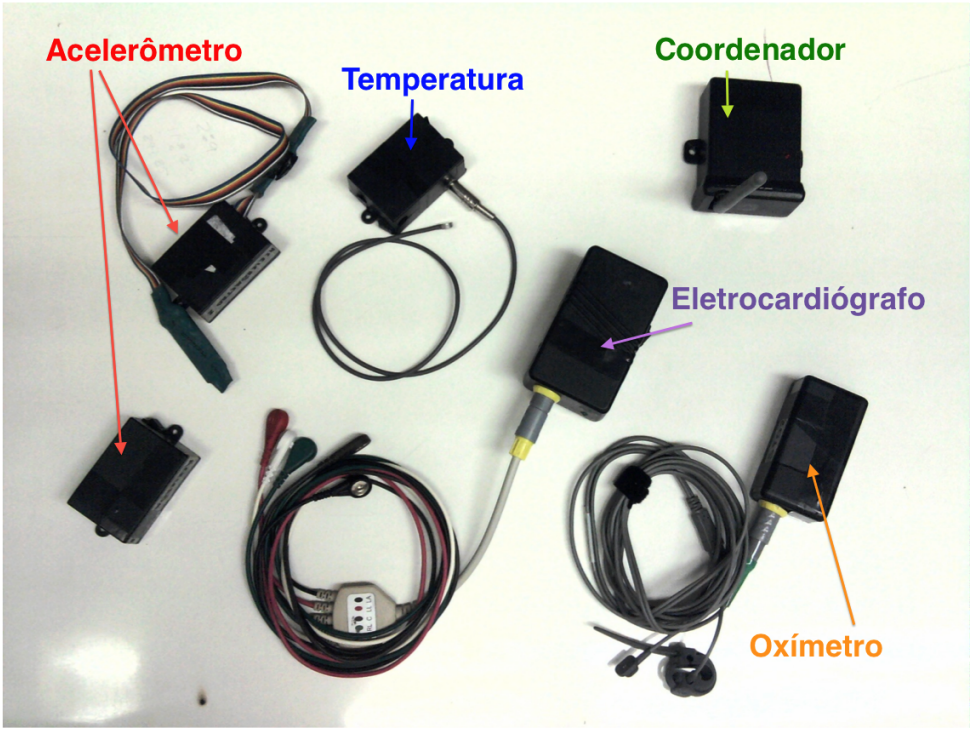


Figura 3.2: Sensores utilizados no sistema [11].

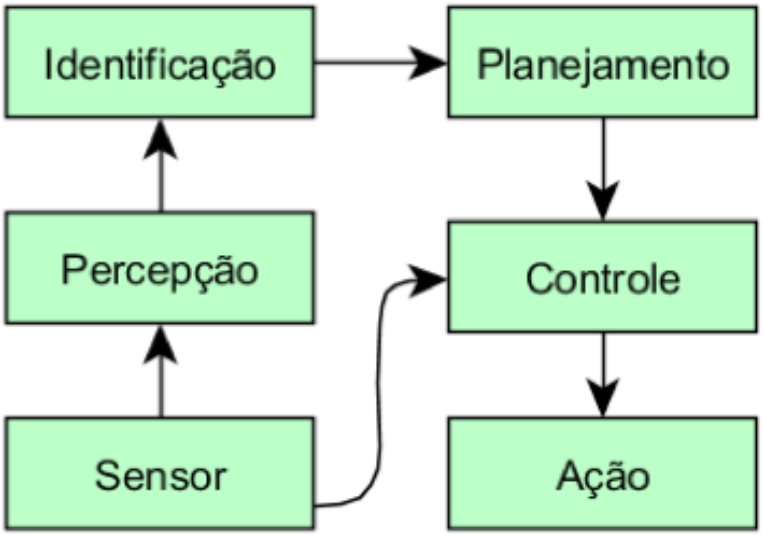


Figura 3.3: Arquitetura de Referência [11].

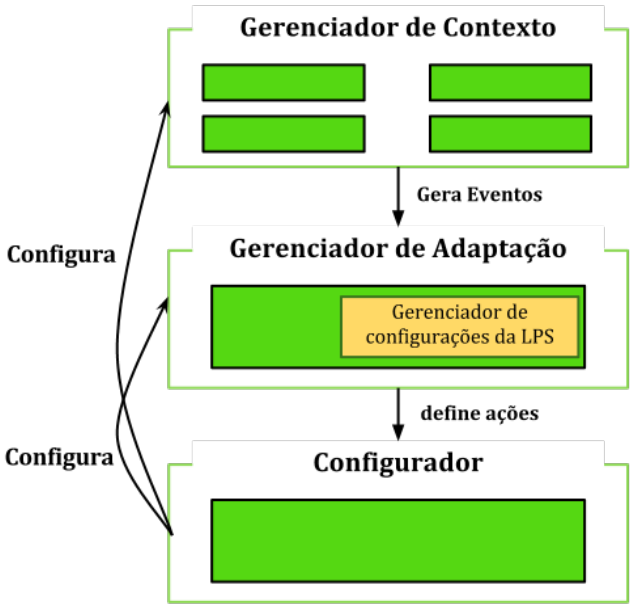


Figura 3.4: Arquitetura em alto nível do sistema. Blocos verdes representam componentes de cada módulo da arquitetura.

- **Controle**: Realiza a modificação do sistema conforme instrução da camada de planejamento.

Para a análise dos dados o sistema de monitoramente possui uma gramática desenvolvida com o auxílio de um especialísta do domínio, neste caso um médico. Esta parceria foi feita pois é o médico que indica quais dados correspondem a cada cenário e é ele que possui o conhecimento para interpretar estes dados para indivíduos diferentes. Para intepretar a gramática existe um componente chamado de gerenciador de contexto que recebe os dados, os interpreta, analisa de acordo com a gramática e percebe a presença de eventos que podem alterar o estado atual do sistema.

Para tratar a mudança de estados o sistema possui um módulo chamado gerenciador de adaptação. Ele é encarregado de, ao receber o evento que pode causar uma mudança no estado atual do sistema, tratá-lo. Caso seja necessária a mudança de estado do sistema o Gerenciador de Configurações da LPS, um dos componentes do gerenciador de adaptação é o encarregado a buscar uma nova configuração válida para o novo contexto.

Dado a nova configuração o componente do sistema chamado Configurador, irá finalmente aplicar as mudanças na configuração atual do sistema.

A figura 3.4 mostra a arquitetura do sistema em alto nível. Os blocos verder representam componentes de maix baixo nível de cada módulo. O componente destado em amarelo, gerenciador de de configurações da LPS é o módulo onde a busca por novas configurações é realizada e é o módulo onde será aplicada o método proposto neste trabalho para otimizar a busca.

Tabela 3.3: Configurações válidas para a Fórmula($storage \leftrightarrow (SQLite \vee File)) \wedge (\neg SQLite \vee \neg File$)

storage	***SQLite***	***File***	$(storage \leftrightarrow (SQLite \vee File)) \wedge (\neg SQLite \vee \neg File)$
0	0	0	1
1	0	1	1
1	1	0	1

3.5 Gerenciador de Configurações da LPS e Gerenciador de Adaptação

O módulo chamado de Gerenciador de Configurações da Linha de Produto de Software (LPS) é encarregar em buscar uma nova configuraç ao válida(CONF) do sistema e repassa-la para o gerenciador de adaptaç para o cálculo de qualidade desta. Após o cálculo, se a configuraç (CONF) possuir um valor de qualidade estiver dentro da faixa de valores definidos para o estado, ela é selecionada como uma possível nova configuração para o sistema. Assim, o gerenciador de configurações faz sua busca até encontrar pelo menso 20 configurações que satisfazem o novo contexto.

Esta busca é realizada por meio de um SAT solver que trabalha em cima do modelo de *features* em forma de fómulas lógicas proposicionais. Esta fórmulas são inseridas via código no programa e são cálculadas assim que o modelo de *features* é definido. Se caso precisem ser alteraas, o programa deve ser alterado e recompilado para se adequar ao novo modelo de *features*.

3.5.1 A Busca

Dados as fómulas lógicas prposicionais, começamos a tratar os possíveis valores das *features* binariamente, sendo que um valor 1 para uma *feature* indica que ela está selecionada na configuração encontrada. Se o valor dela for 0, quer dizer que ela não está selecionada.

Para cada sub-árvore do modelo de *features* temos uma fórmula lógica proposicional, assim, para que uma seleção de features seja válida para aquela sub-árvore, sua fórmula deve retornar o valor VERDADEIRO. Por exemplo, para sub-árvore do exemplo da seção 2.5, encontramos a fórmula $(storage \leftrightarrow (SQLite \vee File)) \wedge (\neg SQLite \vee \neg File)$, que retorna verdadeira para as possíveis configurações listadas na tabela 3.4. Assim, para cada sub-árvore temos que obter uma valoração verdadeira, e por final aplicar a fórmula maior que une todas essas sub- árvores, pois cada nó pai desta subárvore é o nó filho de outro nó ou do próprio nó raiz. Então por fim, a última árvore a ser analisada será a do próprio nó raiz que irá retornar uma configuração válida para o modelo de *feature* completo. No código do programa é assim que as fórmulas são trabalhadas, analisando cada sub-árvore e por fim a árvore do nó raiz.

Decido como as valorações serão interpretadas, podemos começar a construir na ferramenta CHOCO o que é chamado de *constraints*, que são as restrições para que uma fórmula seja verdadeira. É necessário a definicição de uma variável *CPModel* que séra o

```
1  // Define o modelo a ser satisfeito
2  Model m = new CPModel();
3
4  //Define variáveis presentes no modelo. Uma para cada feature.
5  this.varInfoTemperature = Choco.makeIntVar("iT", 0, 1 );
6  this.varInfoOxygenation = Choco.makeIntVar("iO", 0, 1 );
7  this.varSensorTemp = Choco.makeIntVar("sT", 0, 1 );
8  this.varSensorSPO2 = Choco.makeIntVar("sS", 0, 1 );
9  //...
10
11 //Define algumas restrições entre as features
12 Constraint cSensor = Choco.or(varSensorTemp, varSensorECG, varSensorSPO2, varSensorAcc);
13 Constraint temp_IMPLIES_stemp =
14   Choco.ifOnlyIf(Choco.eq(varInfoTemperature, 1), Choco.eq(varSensorTemp, 1));
15 //...
16
17 //Adiciona restrições ao modelo
18 m.addConstraint(cSensor);
19 m.addConstraint(temp_IMPLIES_stemp);
20 //...
21
22 CPSolver s = new CPSolver();
23 s.read(m);
24 if(s.solve()){
25   //busca valoração de features que satisfaz expressão lógica.
26 }
```

Figura 3.5: Trecho do código que utiliza a ferramenta CHOCO [11]

modelo com todas as fórmulas porposicionais e outra do tipo *CPSolver* que é o tipo de dados que resolve as restrições e retorna todas as possíveis configuraç oes para o modelo definido. A figura 3.5 mostra um trecho do código onde é declarado o modelo, o *solver* (resolve as restrições), as variáveis e as restrições.

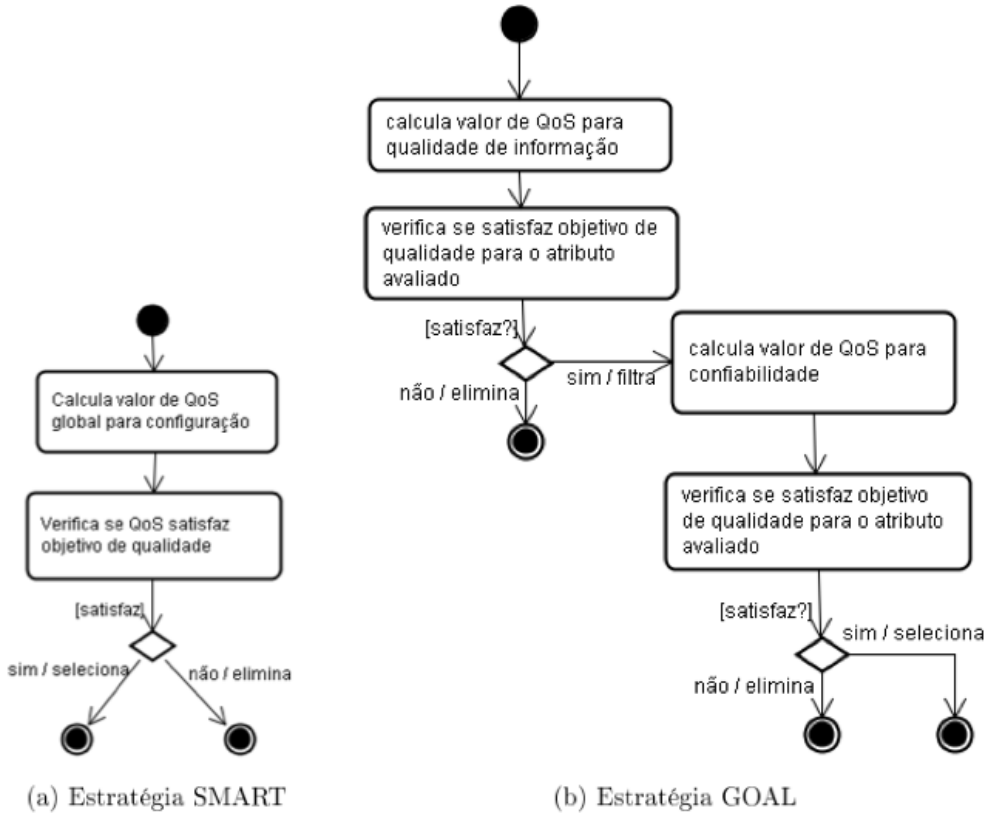
As configurações válidas são acessadas sequencialmente por meio do comando *solver.nextSolution()*, onde *solver* é do tipo *CPSolver*. Assim, temos que verificar todas as configurações até encontrar uma válida e para cada uma calcular seu valor de qualidade. Este tipo de abordagem traz alguns problemas:

- Percorrer grande número de configurações válidas para achar uma com o valor aceitável de qualidade
- Caso não exista uma configuração que satisfaça o valor de qualidade a busca irá percorrer todas as configurações, que pode ser um valor muito grande
- Para achar a configuração com o melhor valor de qualidade, é necessário analisar todas as configurações possíveis.

Hoje o sistema está implementado de forma que as primeiras vinte configurações com valor de qualidade dentro da faixa do estado de risco desejado são selecionadas para análise e dentre estas será selecionada uma configuração para reconfigurar o sistema.

3.5.2 Cálculo do Valor de Qualidade com Vários Atributos

Para cada fórmula encontrada pelo gerenciador de configurações da LPS o gerenciador de adaptação irá calcular seu valor de qualidade associado. Este valor de qualidade é computado a partir de cinco outros atributos de qualidades. São utilizadas duas estratégias



(a) Estratégia SMART (b) Estratégia GOAL

Figura 3.6: Diagrama de Atividades da Estratégia *SMART* e *GOAL* [11]

de cálculo que lidam com vários atributos de qualidade: SMART - *Simple MultiAttribute Rating Techniques* e GOAL - Orientada a Objetivos. Para cada abordagem pode se derivar mais de uma formula.

Atributos de Qualidade

Para o cálculo de qualidade das configurações encontradas neste projeto e por serem analisadas dentro do domínio médico, os seguintes atributos são levados em consideração [11]:

- **Confiança sobre os dados medidos e eventos identificados**: Quanto mais medidas da mesma informação obtidas por instrumentos diferentes, maior é a confiança sobre estes valores.
- **Amostragem da informação**: Quanto maior a frequência de amostragem maior é a chance de o dado mais recente representa melhor a situação real do indivíduo.
- **Disponibilidade de recursos**: Quanto maior o tempo em que um recurso está disponível maior é o controle sobre o estado de saúde do indivíduo.

Tabela 3.4: Tabela que define intervalos aceitáveis de valores de qualidades para cada estado de risco do sistema [11].

Estado Atual	Valor Mínimo	Valor Máximo Desejado
Risco Alto	0.0	< 1.0
Risco Moderado	0.6	< 0.8
Risco Baixo	0.3	< 0.6
Risco Irrelevante	0	< 0.3

- **Quantidade de informação**: Quanto mais informação se tem sobre os dados do estado do indivíduo maior é a precisão de se fazer um diagnóstico sobre o estado atual do indivíduo.
- **Confiabilidade**: Relacionada à probabilidade de execucçcão correta do serviço.

Observe porém que não podemos tratar a busca de um configuração ideal simplesmente tentando achar uma configuração onde o valor de cada atributo é maximizado pois, o aumento de qualidade de um atributo pode necessariamente implicar na diminuição de qualidade de outro. Por exemplo, observe os dois atributos de amostragem e disponibilidade. Como os sensores funcionam à bateria, se aumentarmos a taxa de amostragem de um deles, indica que este trabalhará com mais frequência para obter os dados, consequêntemente, irá gastar mais bateria, diminuindo a chance de este sensor está disponível em um momento futuro. Assim, os atributos de qualidade são trabalhos não para atingir o valor máximo de qualidade, mas sim para atingir um intervalo aceitável de qualidade.

SMART

A estratégia SMART associa a cada atributo de qualidade do sistema um peso. Dessa forma, o valor final da configuração em análise é calculado pela média ponderada destes pesos, e para cada configuração existe um único valor de qualidade associado a ela.

Suas fórmulas variam modificando os pesos associados a cada atributo de qualidade estabelecido e o valor de qualidade cada estado de risco a ser satisfeito. Neste trabalho foram usados seis fórmulas SMART diferentes.

GOAL

A estratégia GOAL verifica para cada atributo de qualidade se a configuração encontrada satisfaz o objetivo de qualidade determinado pelo estado de risco desejado pelo sistema. Desta forma, podemos ter mais de um valor de qualidade final associado a uma configuração, já que é calculado em relação a diferentes atributos de qualidades de cada vez.

Suas fórmulas variam modificando as prioridades associados a cada atributo de qualidade estabelecido e o valor de qualidade cada estado de risco a ser satisfeito. Neste trabalho foram usados quatro fórmulas GOAL diferentes.

Tabela 3.5: Direrentes formula SMART usadas neste trabalho.

	Qualidade	Confiabilidade	Tempo de Vida	Amostragem	Quantidade
SMART 1	4.0	5.0	2.0	1.0	3.0
SMART 2	4.0	5.0	2.0	1.0	3.0
SMART 3	4.0	3.0	1.0	2.0	5.0
SMART 4	3.0	4.0	0.5	0.5	5.0
SMART 5	3.0	5.0	0.5	0.5	4.0
SMART 6	3.0	5.0	0.5	0.5	5.0

	BaixoRisco	MédioRisco	AltoRisco
SMART 1	6.0	7.0	8.0
SMART 2	0.0	6.5	8.5
SMART 3	0.0	5.5	8.0
SMART 4	6.0	7.0	7.5
SMART 5	6.0	7.0	7.5
SMART 6	6.0	7.0	7.5

Tabela 3.6: Fórmulas GOAL

	Qualidade				Confiabilidade			
	Pri.	**BR**	**MR**	**AR**	**Pri.**	**BR**	**MR**	**AR**
Goal 1	1	5.0	7.0	8.0	2	7.0	6.0	4.0
Goal 2	2	5.0	7.0	8.0	1	7.0	6.0	4.0
Goal 3	1	5.0	7.0	8.0	2	7.0	6.0	4.0
Goal 4	1	5.0	7.0	8.0	3	7.0	6.0	5.0

	Tempo de Vida			
	Pri.	**BR**	**MR**	**AR**
Goal 1	3	4.0	7.0	8.0
Goal 2	3	4.0	7.0	8.0
Goal 3	3	4.0	6.5	8.0
Goal 4	2	4.0	7.0	8.0

	Amostragem				Quantidade			
	Pri.	**BR**	**MR**	**AR**	**Pri.**	**BR**	**MR**	**AR**
Goal 1	4	0.0	4.0	7.0	5	0.0	4.0	7.0
Goal 2	4	0.0	4.0	7.0	5	0.0	4.0	7.0
Goal 3	4	1.0	1.0	1.0	5	1.0	1.0	1.0
Goal 4	4	1.0	1.0	1.0	5	1.0	1.0	1.0

3.6 Fórmula para o Cálculo de Confiabilidade

Como proposto por Nunes [18] o cálculo da qualidade de uma configuração válidade de uma linha de produto de *software* é feito por meio de uma fórmula gerada em seu trabalho. Esta fórmula possui constantes e variáveis. As contantes são encontradas fazendo a análise do modelo de *features* e como estas se relacionam. As constantes também representam os diferentes atributos de qualidade que se deseja levar em consideração, como por exemplo confiabilidade, segurança, entre outros. As variáveis irão representar os valores das *features* (1 para presente e 0 para não presente na configuração) e os pesos dos atributos de qualidade.

3.6.1 Características da Fórmula

A fórmula encontrada era extremamente grande, chegando a 2MB quando colocada em um arquivo de texto. Este foi um dos obstáculos encontrados. Para um sistema carregar e calcular uma fórmula desta magnitude compromete seu processamento, principalmente se fosse necessário armazena-lo em memrória ram. Para celulares com *hardware* reduzido, pode ficar inviável. Nunes propôs como diminuir o tamanho da fórmula comparando o impacto que uma variável e uma consntante tem no cumprimento desta e conseguiu reduzi-la. Porém a fórmula ainda é grande, e por isso é carregada no sistema por módulos.

Capítulo 4

Formalização da Propriedade de Segurança (*Safety*)

Para descrever a propriedade de segurança do método proposto, foram feitas primeiramente algumas definições:

- FM: O conjunto de configurações válidas do modelo de *features*
- c : Um configuração derivada do modelo de *features*
- $qos(c)$: Função de qualdiade que recebe uma configuração e calcula seu valor de qualdiade: $qos : FM \rightarrow \mathbb{R}$
- $Q = \{LR, MR, HR\}$: Conjunto dos estados de risco definido na máquina de estados
- s: Um estado da máquina de estados
- $L(s)$: A função que recebe um estado de risco e retorna seu valor de qualidade: $L : Q \rightarrow \mathbb{R}$

Usando estas definições podemos formalizar a propriedade de segurança usando a linguagem CTL [2]:

$$AG(u \rightarrow (EF(p))$$

Onde:

$$c \in FM\text{: predicado } u$$
$$s \in Q\text{: predicado } v$$
$$qos(c) \geq L(s)\text{: predicado } p$$

A fórmula CTL indica que: **para todo**(A) estado s pertencente ao grupo Q de estados de risco do sistema, durante a busca de uma configuração que o satisfaça, **eventualemente**(F) será encontrado **pelo menos uma**(E) configuração c que o satisfaça. E esta propriedade é válida **sempre**(G).

Capítulo 5

Avaliação

Esta seção apresenta a avaliaç do método proposto por Paula Fernandes, focando na eficiência e seguranç no contexto da implementaç feita em Android, chamada Sistema Monitor.

Uma situação de emergência é difícil ser controlado devido a sua característica imprevisível. Por este motivo, foi escolhido conduzir uma simulação para realizar a avaliação do método, mas foi possível usar dados reais coletados no Hospital Beth Israel, MIT.

A seguir eu detalho a avaliação descrevendo o Escopo (5.0.2) e o Planejamento (5.0.3).

5.0.2 Escopo

Para guiar este trabalho foi escolhido usar a técnica de Objetivos, Perguntas e Métricas (*Goal, Questions ans Métricas(GQM)* ([3]). Neste caso, o Objetivo era avaliar a eficiência e a seguranç do sistema monitor, de acordo com a Tabela 5.1. Baseado neste objetivo, foi derivada as perguntas que direcionaram o estudo, descrevendo de maneira mais específica o problema do objetivo. Foram derivadas três perguntas, listadas na Seção 5.0.3. Por último foram desenvolvidos as métricas para avaliar as perguntas.

Para avaliar a eficiência do Sistema Monitor rodando em um dispositivo real com pacientes reais foi considerado o esforç e os recursos gastos pela aplicação. Para este propósito eu decidi medis o tempo médio gasto em cada grande etapa da busca de uma nova configuração que satisfaça o novo requisito de qualidade do estado sobre o ponto de vista da Engenharia da Aplicação, no contexto dos dados reais e simulados e duas abordagens de calculo de qualidade, SMART e GOAL.

5.0.3 Planejamento

Essa seção descreve o planejamento para a aviliaç do objetivo definido. A seção 5.0.3 descreve as perguntas e Métricaas do GQM e justifica brevemente o porque das escolhas. A seção 5.0.3 explica os métodos usados para coletar os dados descritos nas métricas.

Perguntas e Métricas

Para guiar nossos estudos foram definidas três perguntas e suas métricas.

Q1. Qual o tempo gasto pelo Sistema Monitor na troca de contexto?

Proposito	Avaliar
Problema	a eficiênica e segurança do
Objeto	método proposto
Ponto de Vista	Engenharia da Aplicação
Contexto	Sistema Monitor, Calculo de Qualidade e Dados reais

Tabela 5.1: GQM

- Métrica **M1.1** Qual o tempo gasto, em média, no calculo de cada atributo de qualdiade para uma configuração?
- Métrica **M1.2** Qual o tempo gasto, em média, na troca de contexto?

Q2. Quais os recursos usados pelo Sistema Monitor?

- Métrica **M2.1** Qual o tempo médio em horas que a bateria do dispotivo dura?
- Métrica **M2.2** Qual é, em média, o valor de memória consumida?
- Métrica **M2.3** Qual o tamanho médio da aplicação?

Q3. O Sistema Monitor é Seguro?

- Métrica **M3.1** $AG(u \rightarrow (EF(p))$ é satisfeito?

Para medir M1.1 e M1.2 foi decidido avaliar o tempo decorrido entre passos importantes da troca de configuração provocada por uma troca de contexto do sistema. Foi avaliado o tempo gasto em cada atributo de qualidade e o tempo total da troca de configuração. No sistema existem duas abordagens diferentes de calculo de qualidade, SMART e GOAL, e para cada métrica foram rodas simulações diferenciando os tempo com relação as abordagens e suas diferentes formulas. Para métrica M1.1, a simulação foi rodada por 10min para cada fórmula e para M1.2 por 1h.

Métrica M2.2 foi medida durante a execução do programa. M2.3 foi observada em cada instalação diferente no dispositivo, e para M2.1 o sistema foi deixado rodando até que a bateria fosse descarregada por completo.

M3.1 é relacionado a segurança do sistema e representa o seguinte argumento: Para todos os estados da máquina de estado do sistema, existe pelo menos uma configuração que satisfaça o seu valor de qualdade.

Cenários e Contexto

Depois de completo o GQM, os cenários de simulação foram escolhidos. Para rodar a simulação foi usado um computador *Windows* e um dispositivo móvel. No computador rodava, na plataforma eclipse, o programa que simula os sensores e o Sistema Monitor, para coleta de dados da simulação. Os dados eram enviados do computador para o dispositivo via bluetooth, e os dados da simulação eram enviados via conexão USB para o computador, onde eram armazenados.

Os cenários para a avaliação foram construídos usando os dados reais coletados no Hospital Beth Israel. Os dados representam uma senhora idosa de 65 anos com histórico de pressão alta. Porém os dados contém somente eventos de Baixo e Méido risco, então, para

simular o estado de alto risco, foram introduzidos manualmente, por meio do simulador de sensores, os dados que ativam este estado. O simulador de sensores é uma aplicação Java.

Como o Sistemo Monitor contém duas abordagens de calculo de qualidade, e cada uma dessas possui suas formulas específicas, os cenários foram definidos a partir de cada formula diferente. Assim, cada cenário é denominado a partir da formula utilizada, indo de SMART1-SMART6 e GOAL1-GOAL4.

Para coletar os dados da métrica M2.1, o dispositivo foi desconectado do computador, para impedir que ele recarregasse.

Q3 avalia a segurança do sistema, assegurando que a arquitetura dá suporte a todos os seus estados. Durante a execução, o sistema recebe diferentes estados, e seu primeiro passo é selecionar configurações que o satisfaçam. Em seguira, se escolhe dentre uma das que satisfazem, a mais adequada e então o sistema muda sua atual configuração para a nova escolhida. Para garantir M3.1, temos que garantir que a seleção de configurações nunca será vazia.

Para métricas M3.1 novo cenários foram desenvolvidos. Eles variam entre número de sensores ativos e o valor de bateria destes. Com relação a bateria temos: Vazio, Meio Cheio, Cheio. Com relação aos sensores temos: 1 sensor, Metade dos Sensores, Todos os Sensores. Assim, a combiação destes seis quadros, geram os nove cenários possíves. Porém, ainda existe a variação em relação as formulas usadas, por isso, ao final, temos noventa cenários diferentes.

Para roda a simualação foi usado um Motorola Milestone, com 16GB de memória interna, 1GHz dual core processador, 512MB of RAM, 3 anos de uso e Android 2.3 (Gingerbread) instalado.

Procedimentos de Coleta de Dados

Métricas M1.1 and M1.2 são medidas usando a chamada de sistema *System.nanotime()* que retorna o valor mais preciso disponível no contador do sistema, em nanosegundos ([13]). Esta função foi considerada a mais adequada para ser usada, pois é a melhor para computar tempo decorrido.

Para métrica relacionadas a consumo de memória foi usado o *Private Dirty*, obtido pela função da SDK do Android *getProcessMemoryInfo(int[] pid)*. Este parâmetro retorna o valor de memória que é exclusivo do sistema, não compartilhado por mais nenhum processo. Foi avaliado também o parâmetro de *LowMemory* que retorna verdadeiro se o sistema estiver executando com baixa memória disponível.

O espaço em disco e bateria consumida foram monitoradas sem chamadas de sistema. O espaço em disco foi obtido olhando as configurações do sistema no menu de aplicativos. A bateria foi monitorada olhando a porcentagem disponível pelo sistema.

Para a metrica M3.1 a simulação foi rodada com os cenários específicos e o número de configurações encontradas foi anotado.

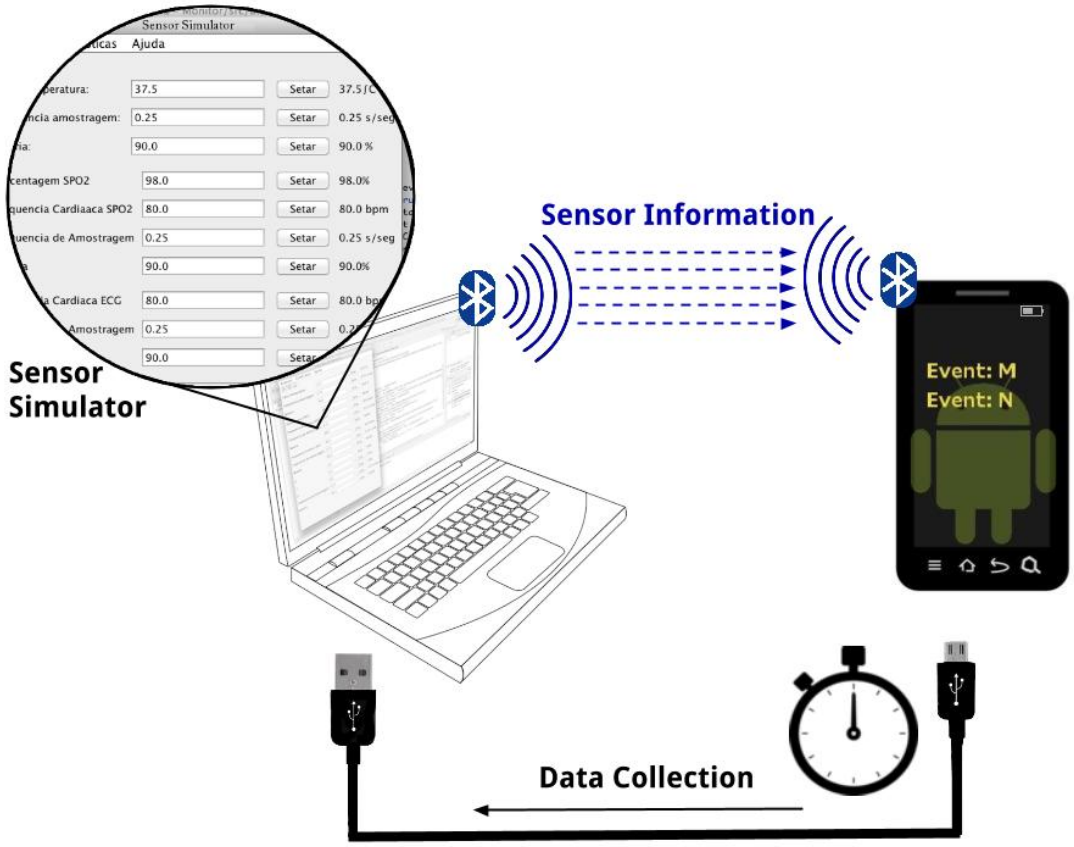


Figura 5.1: Organização do ambiente de simulação.

Capítulo 6

Resultados e Análises

Eficiência

Para M1.1-M1.2 e M2.1-M2.3 foi construida a Tabela 6.1 com os principais dados encontrados.

Analisando a Tabela 6.1, foi descoberto que o tempo para calcular o valor de confiabilidade é o maior dentre todos os cálculos de atributo de qualidade, chegando a 45% do tempo total para calcular todos os valores dos atributos de qualidade. Isso acontece pois a fórmula para cálculo de confiabilidade é muito grande e envolve muitas operações.

A troca de contexto é a segunda parte da avaliação do esforç e envolve o tempo gasto pelo sistema para buscar uma configuração adequada e substituir a atual pela escolhida. Neste período, o os valores de qualidade são calculados para todas as configurações testadas, porém de forma diferentes para cada abordagem escolhida. Para o casa da GOAL, os valores são calculados de acordo com suas prioridades, porém se um valor de um atributo já não satisfizer um objetivo de qualidade, a configuração é descartada imediatamente, assim, somente para algumas configurações será necessário calcular todos os atributos de qualidade. Já para a SMART, para todas as configurações testadas, todos os atributos são calculados. Essa é uma característica muito importante das duas abordagens pois ela justifica os valores encontrados. Para SMART vemos valores similares para todas as diferentes formulas, enquanto para a GOAL, as fórmulas que tem menos prioridade para o cálculo de confiabilidade possuem os menores valores. Assim, na Tabela 6.1 vemos o grande impacto que a formula de confiabilidade tem sobre o tempo total de troca de contexto, e podemos também ressaltar o impacto de ser necessário calcular todos os atributos para todas as configurações testadas, já que a GOAL2, que possui prioridade 1 para confiabilidade e por isso precisa calcular seu valor para todas configurações, ainda obtem um tempo de troca de contexto de quase metade de uma fórmulas SMART.

Concluindo, comparando as duas abordagens podemos dizer que, mesmo no pior caso da GOAL, ou seja, a confiabilidade com prioridade 1, esta abordagem se mostra muito superior a SMART no quesito de eficiência.

O tamanho médio em disco da aplicação, métrica M2.3, encontrado foi de 9.09MB. Considerando que o dispotivo móvel possui 16GB de memória interna, este valor representa somente 0.05%. Ainda eu aparelhos com menor memória interna, chegamos a mínimos de 1GB, e o valor de 9.09MB ainda se mostra insignificante. Assim, por essa avaliação, o sistema foi considerado viável para rodar no mundo real.

Effort	
M1.1 Atributos de Qualidade	**Tempo Médios (ms)**
Qualidade	1.41 ± 0.43
Confiabilidade	166.06 ± 10.99
Tempo de Vida	0.60 ± 0.18
Amostragem	0.60 ± 0.21
Quantidade	0.02 ± 0.06
M1.2 Formulas de Qualidade	**Tempo Médio de Troca de Contexto (ms)**
Goal 1	8245 ± 483
Goal 2	12757 ± 278
Goal 3	9057 ± 151
Goal 4	7734 ± 151
Smart 1	35530 ± 4736
Smart 2	35479 ± 1501
Smart 3	35541 ± 2988
Smart 4	35480 ± 2846
Smart 5	35274 ± 333
Smart 6	35332 ± 838
Recursos	
M2.1 Bateria Gasta(%)	**Tempo (min)**
7	90
26	270
56	420
100	725
M2.2 Memória	**KB**
Private Dirty	7617.98
M2.3 Tamanho em Disco	**MB**
Tamanho Médio	9.08

Tabela 6.1: Resultados dos Dados Coletados

Sobre o consumo de bateria, M2.1, os dados indicam que o Sistema Monitor tem um consumo baixo de energia. Após de 6hs de uso a bateria do dispositivo chegou a metade de sua capacidade, e somente após 12hs de uso que ela veio a se descarregar por completo. Ainda sim, o primeiro aviso de bateria fraca, 15% restante, veio somente depois de 10hs de uso, o que ainda daria ao usuário 2hs para encontrar uma fonte de energia para carregar o dispositivo. Considerados esses dados, nós consideramos o sistema, do ponto de vista da bateria viável para roda em um dispositivo comum. Entretanto, é importante ressaltar que o teste de bateria foi feito sem nenhum outro aplicativo rodando, então a utilização do sistema com outros aplicativos em paralelo pode levar a um consumo maior de energia.

Para a RAM, M2.2, foi analisado duas métricas. O Private Dirty encontrado foi de, em média 7.62MB, que representa somente 1.36% da RAM disponível no dispositvo. O atributo de LowMemory, também analisado, não retornou em nenhum momento positivo, ou seja, o sistema nunca entrou em estado de baia memória disponível. Os dois atributos indicam novamente que o Sistema Monitor poderia rodar em um cenário real.

Segurança

As Tabelas 6.2,6.3 e 6.4 apresentam o número de configurações encontradas para cada formula de qualidade e os cenários envolvendo bateria e o número de sensores, separados por estado de risco.

Observando os dados encotrados, pode-se concluir que nem todas as fórmulas aprovadas pelo especialista do domínio garante a propriedade de seguraça do sistema se usadas sozinhas para avaliação de todos os estados de risco. SMART5 e SMART6 são as únicas fórmulas que, para todos os estado de risco e para todos os cenários disponíveis, foi possível encontra pelo menos uma configuração que satisfaça o estado. Então, são as únicas fórmulas que garante a propriedade de segurança do sistema.

Entretanto, as fórmulas SMART são as que tem o pior tempo de troca de contexto, assim, ao escolher uma dessas fórmulas para rodas em todos os estados de risco, a eficiência do sistema, que por se tratar de um sistema crítico é de extrema importância, é prejudicada. Então para garantir a segurança e a eficiência do sistema, a solução é utilizr as duas fórmulas para gerenciar os estados de risco da máquina de estado. Por exemplo, podemos ver pela Tabela 6.1 que a fórmula GOAL4 é a que possui menor tempo de troca de contexto. Olhando agora a Tabela 6.2, é possível notar que esta fórmula garante q propriedade de seguranç para o estado de baixo risco. Fazendo a mesma análise para GOAL1, temos que esta garante a propriedade de segurança para o estado de médio risco. E como a SMART6 possui tempo menor que a SMART5, podemos usá-la para gerênciar o estado de alto risco. Assim, temos diferentes fórmulas lidando com diferentes estados de risco.

Usando essa mistura, o sistema gataria somente 51.311s contra 105.996s usando somente SMART6, em uma troca de contexto que levaria o sistema de baixo, para médio e depois para alto risco. Temos assim um ganho de 51% na eficiência no cenário total, e no estado de baixo risco um ganho 78% e no de médio risco um ganho de 76%. Considerando o domínio da aplicação, cada segundo é importante, então o uso de diferentes fórmulas para garantir não só a seguranç mas também a eficiência é de extrama importância.

Formula	Profile	Nro de Configurações		
		Baixo	Médio	Alto
GOAL1	1Sensor, Bateria Vazia	3	3	0
	1Sensor, 1/2 Bateria	3	3	0
	1Sensor, Bateria Cheia	3	3	0
	1/2 Sensores, Bateria Vazia	39	37	0
	1/2 Sensores, 1/2 Bateria	39	37	0
	1/2 Sensores, Bateria Cheia	39	37	0
	Todos Sensores, Bateria Vazia	165	163	129
	Todos Sensores, 1/2 Bateria	165	163	129
	Todos Sensores, Bateria Cheia	165	163	129
GOAL2	1Sensor, Bateria Vazia	3	3	0
	1Sensor, 1/2 Bateria	3	3	0
	1Sensor, Bateria Cheia	3	3	0
	1/2 Sensores, Bateria Vazia	39	39	26
	1/2 Sensores, 1/2 Bateria	39	39	26
	1/2 Sensores, Bateria Cheia	33	29	0
	Todos Sensores, Bateria Vazia	72	129	0
	Todos Sensores, 1/2 Bateria	159	165	146
	Todos Sensores, Bateria Cheia	165	165	146
GOAL3	1Sensor, Bateria Vazia	165	165	165
	1Sensor, 1/2 Bateria	165	165	165
	1Sensor, Bateria Cheia	65	0	0
	1/2 Sensores, Bateria Vazia	39	39	39
	1/2 Sensores, 1/2 Bateria	39	39	39
	1/2 Sensores, Bateria Cheia	39	0	0
	Todos Sensores, Bateria Vazia	3	3	3
	Todos Sensores, 1/2 Bateria	3	3	3
	Todos Sensores, Bateria Cheia	3	0	0
GOAL4	1Sensor, Bateria Vazia	165	72	39
	1Sensor, 1/2 Bateria	159	72	0
	1Sensor, Bateria Cheia	72	0	0
	1/2 Sensores, Bateria Vazia	39	3	0
	1/2 Sensores, 1/2 Bateria	39	3	0
	1/2 Sensores, Bateria Cheia	33	0	0
	Todos Sensores, Bateria Vazia	3	0	0
	Todos Sensores, 1/2 Bateria	3	0	0
	Todos Sensores, Bateria Cheia	3	0	0

Tabela 6.2: Resultados dos Cenários envolvendo GOAL formulas para avaliação de Segurança

Formula	Profile	Nro de Configurações		
		Baixo	Médio	Alto
SMART1	1Sensor, Bateria Vazia	4	165	165
	1Sensor, 1/2 Bateria	4	165	152
	1Sensor, Bateria Cheia	4	100	0
	1/2 Sensores, Bateria Vazia	4	39	39
	1/2 Sensores, 1/2 Bateria	4	39	31
	1/2 Sensores, Bateria Cheia	4	20	0
	Todos Sensores, Bateria Vazia	4	3	3
	Todos Sensores, 1/2 Bateria	4	3	2
	Todos Sensores, Bateria Cheia	4	2	0
SMART2	1Sensor, Bateria Vazia	5	165	165
	1Sensor, 1/2 Bateria	5	165	165
	1Sensor, Bateria Cheia	5	165	0
	1/2 Sensores, Bateria Vazia	5	39	39
	1/2 Sensores, 1/2 Bateria	5	39	39
	1/2 Sensores, Bateria Cheia	5	39	0
	Todos Sensores, Bateria Vazia	5	3	3
	Todos Sensores, 1/2 Bateria	5	3	3
	Todos Sensores, Bateria Cheia	5	3	0
SMART3	1Sensor, Bateria Vazia	165	165	157
	1Sensor, 1/2 Bateria	165	165	12
	1Sensor, Bateria Cheia	165	157	0
	1/2 Sensores, Bateria Vazia	39	39	33
	1/2 Sensores, 1/2 Bateria	39	39	0
	1/2 Sensores, Bateria Cheia	39	33	0
	Todos Sensores, Bateria Vazia	3	3	2
	Todos Sensores, 1/2 Bateria	3	3	0
	Todos Sensores, Bateria Cheia	3	2	0

Tabela 6.3: Resultados dos Cenários envolvendo SMART1-SMART3 formulas para avaliação de Segurança

Formula	Profile	Nro de Configurações		
		Baixo	Médio	Alto
SMART4	1Sensor, Bateria Vazia	165	165	164
	1Sensor, 1/2 Bateria	165	163	136
	1Sensor, Bateria Cheia	165	96	0
	1/2 Sensores, Bateria Vazia	39	39	38
	1/2 Sensores, 1/2 Bateria	39	37	27
	1/2 Sensores, Bateria Cheia	39	22	0
	Todos Sensores, Bateria Vazia	3	3	3
	Todos Sensores, 1/2 Bateria	3	3	2
	Todos Sensores, Bateria Cheia	3	2	0
SMART5	1Sensor, Bateria Vazia	165	165	163
	1Sensor, 1/2 Bateria	165	158	137
	1Sensor, Bateria Cheia	165	122	44
	1/2 Sensores, Bateria Vazia	39	39	37
	1/2 Sensores, 1/2 Bateria	39	34	37
	1/2 Sensores, Bateria Cheia	39	26	10
	Todos Sensores, Bateria Vazia	3	3	3
	Todos Sensores, 1/2 Bateria	3	2	2
	Todos Sensores, Bateria Cheia	3	2	2
SMART6	1Sensor, Bateria Vazia	165	165	164
	1Sensor, 1/2 Bateria	165	163	145
	1Sensor, Bateria Cheia	165	110	26
	1/2 Sensores, Bateria Vazia	39	39	38
	1/2 Sensores, 1/2 Bateria	39	37	29
	1/2 Sensores, Bateria Cheia	39	26	10
	Todos Sensores, Bateria Vazia	3	3	3
	Todos Sensores, 1/2 Bateria	3	3	2
	Todos Sensores, Bateria Cheia	3	2	2

Tabela 6.4: Resultados dos Cenários envolvendo SMART4-SMART6 formulas para avaliação de Segurança

Capítulo 7

Conclusão

Neste trabalho foi possível testar a eficiência e a segurança do método proposto por Paula Fernandes. Foi possível também comparar as duas abordagens de calculo de qualdiade: SMART e GOAL.

Foi concluido que, apesar da GOAL se mostrar mais eficiente, sozinha, ela não garante a propriedade de segurança do sistema. Ao contrário, a SMART foi capaz de prover duas fórmulas que garantiam a segurança, mas na troca de contexto obtiveram números muito grandes, pecando na eficiência. Assim, a solução é misturar as duas abordagens melhorando a eficiência e garantindo a segurança do sistema.

Essa solução porém não é perfeita pois, para o estado de alto risco ainda temos um tempo de troca de contexto muito elevado. Analisando os tempo obtidos de cálculo dos atributos de qualidade, vemos que o gargalo maior se encontra no cálculo de confiabilidade, o que sugere a necessidade de uma otptimização da fórmula. Outro ponto de melhora seria a busca de configurações. Reduzindo o espaço de busca, seria possível diminuir o tempo de troca de contexto.

Referências

[1] Germán H. Alférez and Vicente Pelechano. Context-aware autonomous web services in software product lines. *SPLC '11 Proceedings of the 2011 15th International Software Product Line Conference*, pages 100–109, 2011.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. 21

[3] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994. 22

[4] Gabriele Bavota, Rocco Oliveto, Andrea De Lucia, Giuliano Antoniol, and Yann-Gael Gueheneuc. Playing with refactoring: Identifying extract class opportunities through game theory. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–5, Washington, DC, USA, 2010. IEEE Computer Society.

[5] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, September 2010.

[6] Paulo Borba, Leopoldo Teixeira, and Rohit Gheyi. A theory of software product line refinement. In *Proceedings of the 7th International colloquium conference on Theoretical aspects of computing*, ICTAC'10, pages 15–43, Berlin, Heidelberg, 2010. Springer-Verlag. 3

[7] Hervaldo Sampaio Carvalhot. *Data Fusion Implementation in Sensor Networks Applied to Health Monitoring*. PhD thesis, Universidade de Minas Gerais, 2005. vii, 12

[8] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Designing and prototyping dynamic software product lines: Techniques and guidelines. *SPLC '10 Proceedings of the 14th International Conference, SPLC 2010*, 6287:331–345, 2010.

[9] Carlos Cetina, Øystein Haugen, Xiaorui Zhang, Franck Fleurey, and Vicente Pelechano. Strategies for variability transformation at run-time. *SPLC '09 Proceedings of the 13th International Software Product Line Conference*, pages 61–70, 2009.

[10] Magnus Eriksson, Jürgen Börstler, and Kjell Borg. Managing requirements specifications for product lines - an approach and industry case study. *J. Syst. Softw.*, 82(3):435–447, March 2009.

[11] Paula Fernandes. Linha de produto de software dinâmica direcionada por qualidade: o caso de redes de monitoração do corpo humano. Master's thesis, Universidade de Brasília, 2012. iii, iv, vi, vii, 1, 2, 6, 9, 10, 11, 13, 16, 17, 18

[12] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990. 10

[13] Java. Java documentation @MISC, July 2012. 24

[14] WA Knaus, EA Draper, DP Wagner, and JE Zimmerman. Apache ii: a severity of disease classification system. critical care medicine, 1985. http://www.ncbi.nlm.nih.gov/pubmed/3928249. vii, 11

[15] Sergiy S. Kolesnikov, Sven Apel, Norbert Siegmund, Stefan Sobernig, Christian Kästner, and Semah Senkaya. Predicting quality attributes of software product lines using software and network measures and sampling. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 6:1–6:5, New York, NY, USA, 2013. ACM. 3

[16] Marcilio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[17] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

[18] Vinícius Nunes. Gerência de variabilidade em modelos de confiabilidade para linha de produto de software. Master's thesis, Universidade de Brasília, 2012. 10, 20

[19] Norbert Siegmund, Marko Rosenmuller, Christian Kastner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. Scalable prediction of non-functional properties in software product lines. In *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, pages 160–169, Washington, DC, USA, 2011. IEEE Computer Society.