UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Framework para desenvolvimento de LPSD no âmbito de TV Digital Interativa, com suporte a características de ubiquidade

LEANDRO VAGUETTI

ORIENTADOR: PAULO ROBERTO DE LIRA GONDIM

TESE DE DOUTORADO
EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: 094/2015.TD - PPGEE

BRASÍLIA/DF: MARÇO - 2015.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FRAMEWORK PARA DESENVOLVIMENTO DE LPSD NO ÂMBITO DE TV DIGITAL INTERATIVA, COM SUPORTE A CARACTERÍSTICAS DE UBIQUIDADE

LEANDRO VAGUETTI

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.

APROVADA POR:

PAULO ROBERTO DE LIRA GONDIM, Dr., ENE/UNB
(ORIENTADOR)

JORGE HENRIQUE CABRAL FERNANDES., Dr., CIC/UNB
(EXAMINADOR INTERNO)

DÍBIO LEANDRO BORGES., Dr, CIC/UNB
(EXAMINADOR INTERNO)

PAULO HENRIQUE PORTELA DE CARVALHO, Dr., ENE/UNB
(EXAMINADOR INTERNO)

EDISON ISHIKAWA, Dr., CIC/UNB
(EXAMINADOR INTERNO)

Brasília, 04 de março de 2015.

FICHA CATALOGRÁFICA

VAGUETTI, LEANDRO

Framework para desenvolvimento de LPSD no âmbito de TV Digital Interativa, com suporte a características de ubiquidade. [Distrito Federal] 2015.

209p.,(ENE/FT/UnB, Doutor, Engenharia Elétrica, 2015).

Tese de Doutorado - Universidade de Brasília.

Faculdade de Tecnologia.

Departamento de Engenharia Elétrica.

1. TV Digital Interativa
2. Linhas de Produtos de Software
3. Software Orientado a *Features*
4. Software Ubíquo

I. ENE/FT/UnB

II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

VAGUETTI, L. (2015). *Framework* para desenvolvimento de LPSD no âmbito de TV Digital Interativa, com suporte a características de ubiquidade. Tese de Doutorado em Engenharia Elétrica, Publicação 094/2015.TD - PPGEE, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 209p.

DEDICATÓRIA

Este trabalho é dedicado aos meus filhos.

AGRADECIMENTOS

Agradeço à minha família e todas as pessoas que, de alguma forma, compreenderam minha ausência, apoiaram e deram força nos momentos difíceis.
Agradeço ao meu orientador, Prof. Paulo Roberto de Lira Gondim, pela oportunidade, paciência e apoio para realização do trabalho.

Agradeço ao IFB (Instituto Federal de Brasília) pelo concessão de afastamento para qualificação durante os períodos finais de elaboração da tese.

Agradeço a todos que de alguma forma estiveram envolvidos e contribuíram para a realização do trabalho.

RESUMO

***Framework* para desenvolvimento de LPSD (Linhas de Produto de Software Dinâmicas) no âmbito de TV Digital Interativa, com suporte a características de ubiquidade**

Autor: Leandro Vaguetti
Orientador: Paulo Roberto de Lira Gondim
Programa de Pós-Graduação em Engenharia Elétrica
Brasília, Março de 2015

Este trabalho apresenta um *framework*, denominado DSPL2UbiTV, para desenvolvimento de LPS (Linhas de Produto de Software) voltadas ao ambiente de TV Digital interativa - TVDi, aplicadas ao Domínio de Aplicações Ubíquas. O framework DSPL2UbiTV é baseado em uma abordagem de Linhas de Produtos de Software Dinâmicas - LPSD, possibilitando a seleção de produtos de software, em tempo de execução, em um receptor de TV Digital.

O *framework* apresentado busca prover condições adequadas para manutenibilidade e evolução da LPS, e apoia o desenvolvimento de projetos de softwares ubíquos com características como onipresença de serviços, sensibilidade ao contexto, comportamento adaptativo e captura de experiências. Além disso, o *framework* permite a configuração dinâmica do produto de software, com possibilidade de seleção de *features* através de mecanismos de recomendação. Neste contexto, as *features* são recomendadas considerando as interações do usuário com conteúdos interativos de TVDi.

A solução proposta é construída usando as melhores práticas de engenharia de software, permitindo uma melhor manutenibilidade e consequente evolução do software. Experimentos são realizados para demonstrar a capacidade do *framework* DSPL2UbiTV de melhorar a qualidade das LPSs no tocante a atributos de qualidade como complexidade, extensibilidade e reuso de software.

ABSTRACT

FRAMEWORK FOR DEVELOPING INTERACTIVE DIGITAL TV APPLICATIONS, SUPPORTING CHARACTERISTICS OF UBIQUITY.

Author: Leandro Vaguetti
Supervisor: Prof. Paulo Roberto de Lira Gondim
Programa de Pós-graduação em Engenharia Elétrica
Brasília, March of 2015

This work presents a framework for the development of LPS (Software Product Lines) aimed at interactive Digital TV environment - iDTV, applied to the Ubiquitous Applications Domain. The framework is based on an Dynamic Software Product Lines - DSPL approach, allowing the software product selection, at runtime, in a Digital TV receiver.

The presented framework supports the development of ubiquitous software projects with features such as omnipresence of services, context awareness, adaptive behavior and experience capture. Furthermore, the framework allows the dynamic configuration of the product of software, with possibility of selection of features through recommendation engines. In this context, features are recommended considering the user interactions with interactive iDTV content.

The proposed solution is built using best software engineering practices, allowing for better reuse and evolution of software. Experiments are conducted in order demonstrate the ability of the framework to improve the quality of LPS considering the software quality attributes as complexity, extensibility and reuse.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1 INTRODUÇÃO

Nos últimos anos, aplicações para Televisão Digital interativa (TVDi) tem se tornado um recurso importante para a viabilização de novos modelos de negócio, voltados para uma gama bastante variada de serviços. Tais aplicações permitem que mecanismos de interatividade sejam associados aos conteúdos tradicionais de televisão, favorecendo a implantação e a adoção de um novo paradigma de comunicação e de serviços.

Esse novo paradigma, baseado na adoção de uma ou mais formas de interatividade, permite visualizar amplas possibilidades de emprego desse emergente meio de comunicação audiovisual, favorecendo a oferta de serviços em áreas tais como marketing, educação a distância, comércio eletrônico, governo eletrônico, saúde, ensino, votações e entretenimento, dentre outras.

Neste sentido, a progressiva implantação de um novo paradigma de comunicação permitirá disponibilizar ao telespectador um conjunto amplo de novos serviços baseados em mídias interativas (por exemplo: cursos, produtos para compra, serviços governamentais, informações, pesquisas de opinião, etc), de forma associada ao conteúdo assistido.

Observa-se, por outro lado, que mesmo em países de Primeiro Mundo, a TVDi ainda se encontra em fase de adoção relativamente recente, havendo escassez de aplicações e serviços interativos, bem como ocorre ainda a subutilização das bandas de frequência disponíveis, por ainda não se ter emprego usual de multiprogramação [35]. No Brasil, esforços de diversas instituições (por exemplo, PUC-Rio, UFPB e UnB, dentre outras) e a alocação de recursos investidos pelo Governo Federal a partir do ano de 2004, permitiram que se chegasse à construção do sistema ISDB-Tb, para o qual é possível encontrar receptores e conversores com os selos TVD e TVDi, com adoção do *middleware* Ginga de forma progressiva (em termos de percentuais mínimos de novos televisores que incorporam esse recurso produzido em nosso país).

Verifica-se, por outro lado, que a interatividade ainda não foi explorada em sua plenitude, apesar da existência de projetos como o "Brasil 4D"[1], focado em serviços governamentais para famílias de baixa renda e outras aplicações interativas, recente-

mente desenvolvidas [5].

Sem interatividade ou mesmo com interatividade limitada, a TVD possibilita aos seus usuários o usufruto de apenas parte das características e vantagens associadas a TVDi. Dentre tais características e vantagens, incluem-se: a melhoria da qualidade da imagem e do som, facilidades para gravação de programas, a otimização da cobertura e um melhor emprego do espectro eletromagnético.

Alguns entraves podem ser verificados quando se busca uma maior disseminação da TVD em nosso país, em especial quanto ao emprego da interatividade, tais como: questões relativas à regulação dos serviços de comunicação; possível falta de interesse de radiodifusores; escassez de aplicativos em software que aproveitem de forma adequada as possibilidades de interação [6].

Assim, no cenário atual, para que a TVD e a TVDi alcancem maior disseminação e penetração junto a sociedade brasileira (e também em outros países usuários de nosso sistema), um grande esforço de desenvolvimento/manutenção de aplicativos se faz necessário. Para esse fim, cabem ser considerados aspectos como a grande variedade de conteúdos televisivos passíveis de serem produzidos diariamente, a necessidade de personalizar a interação para cada conteúdo disponibilizado e o bastante provável aumento da oferta de serviços nas áreas já citadas. Além disso, a variedade de receptores de TV Digital disponíveis (fixo e móvel, dotado de HDTV e SDTV) aumenta a necessidade de customização da interatividade.

Por outro lado, as características específicas do ambiente de televisão digital levam à necessidade de recursos que facilitem o desenvolvimento de aplicações com qualidade adequada e buscando otimizar o esforço de desenvolvimento, além de facilitar as atividades de manutenção a serem realizadas. Assim, as soluções para desenvolvimento de software devem considerar características peculiares do ambiente de TVDi (por exemplo: transmissão *broadcast*, mecanismos de interatividade e recursos computacionais limitados), tornando-se necessário considerar a necessidade de adaptação do conteúdo interativo no momento da recepção, levando em conta aspectos como a variação de receptores, mecanismos de interatividade disponíveis e perfis de interatividade do telespectador, dentre outros.

Por fim, cabe observar que a necessidade de disponibilidade de conteúdos interativos, em diversos equipamentos e locais, impõe o desenvolvimento de aplicações/serviços

que tratem adequadamente aspectos ligados à ciência de contexto e, de maneira mais ampla, à ubiquidade. Neste sentido, considera-se a possibilidade e a importância de desenvolvimento de softwares para TVDi que atendam as características de aplicações onipresentes, também conhecidas como aplicações Ubíquas.

1.1 Descrição do Problema

Este trabalho considera como principais problemas no tocante ao desenvolvimento de software para TVDi:

1. a necessidade de uma abordagem de desenvolvimento de software que ofereça condições adequadas para a sua manutenção e evolução;
2. a necessidade de uma solução que permita, aos profissionais de desenvolvimento de software, a construção de aplicações capazes de se adaptar, adequadamente, aos ambientes nos quais serão utilizados.

Além dos problemas citados anteriormente é importante considerar algumas observações:

Do ponto de vista funcional, as aplicações ubíquas comumente requerem boa quantidade de adaptações, em função dos diversos ambientes e contextos em que podem ser empregadas [62][16]. Tais aplicações (e os serviços que delas fazem uso) devem estar disponíveis aos usuários, independentemente de fatores tais como o local, momento e ambiente operacional em que o usuário estiver utilizando seu equipamento (celular, tablet, etc). Além disso, os softwares ubíquos possuem diversas características, tais como comportamento adaptável, sensibilidade ao contexto e a onipresença de serviços que devem ser suportadas pelas aplicações [57]. Assim, os softwares ubíquos precisam ser capazes de sofrer adaptações frequentes e, em função do desenvolvimento de novas tecnologias, estarem em constante evolução. Entretanto, atualmente, existe grande dificuldade no desenvolvimento de tais softwares, uma vez que as ferramentas disponíveis (por exemplo: iTVProject [45] e FrameIDTV [51]), para apoio ao desenvolvimento, não dão suporte a todas as características de softwares ubíquos, além de não terem sido submetidas a um processo de avaliação quanto à sua capacidade de manutenção e evolução.

Do ponto de vista de Engenharia de Software, aplicações que são desenvolvidas considerando os diversos ambientes nos quais serão executadas, devem otimizar o custo associado à produção e evolução do software. Neste sentido, diversas abordagens de reuso de software (Desenvolvimento baseado em componentes, *frameworks* de Desenvolvimento, Padrões de Projeto, Engenharia Dirigida a Modelos, etc [56]) são encontradas para atender tal necessidade.

Dentre as abordagens acima referidas, destaca-se a abordagem denominada Linha de Produtos de Software - LPS. LPS refere-se a um conjunto de aplicações (produtos de softwares) que possuem uma arquitetura comum e componentes compartilhados, sendo cada produto especializado para refletir o atendimento a diferentes necessidades dos usuários ou ambientes de execução [56][36][15].

A LPS, em sua proposta original, permite a prévia seleção por parte dos desenvolvedores, das funcionalidades do software em acordo com as necessidades citadas [23], existindo vários casos de sucesso aplicados a ambientes como celulares, carros, aviões, etc. Pode-se destacar, em termos de ambiente de aplicação, a utilização recente de LPS para reconstrução do *middleware* Ginga [52].

Aplicações para TV Digital interativa, como por exemplo programas interativos e aplicações enviadas através do canal *broadcast*, necessitam se ajustar, minimamente, às características do receptor, sob pena de não oferecerem condições adequadas de interatividade. Assim, observa-se que, para a construção de softwares para TVDi, entregues ao telespectador através de um canal *broadcast*, faz-se necessário algum mecanismo que torne dinâmica a construção do software no receptor.

Neste sentido, uma extensão da abordagem de LPS, denominada de Linhas de Produtos de Softwares Dinâmicas (LPSD) [29], prevê a seleção, em tempo de execução, das funcionalidades do produto de software. Neste caso, além da configuração, considera-se a reconfiguração, no caso de mudanças de contexto ou preferência do produto de software em tempo real. Assim, sistemas tratados como adaptativos, tais como aplicações ubíquas, personalizadas e cientes de contexto poderiam ser construídos utilizando-se de LPSD [50][30]. A necessidade de dinamização da construção do software, aliada à possibilidade de seleção de *features*, leva então a considerar-se conveniente a utilização de LPS, dotadas de algumas características especiais (LPSD), para o desenvolvimento aplicações para TVDi.

Finalmente, para que a adaptação do software seja adequada em diversas situações torna-se necessário utilizar um mecanismo capaz de decidir quais *features* devem ser selecionadas dinamicamente. Na literatura existem várias soluções para definição das variabilidades, semelhanças e restrições nas LPS [13][19][20], bem como propostas para configuração e validação dos produtos de software definidos. No entanto, tais soluções não oferecem capacidade de tomada de decisão. Assim, em uma LPSD, torna-se necessário definir e construir os mecanismos para tomada de decisão relativa à seleção das *features*.

1.2 Objetivos

O principal objetivo da tese é o desenvolvimento de uma ferramenta que ofereça suporte ao desenvolvimento de Linhas de Produtos de Software no ambiente de TV Digital interativa, no domínio de aplicações ubíquas, permitindo condições adequadas de manutenibilidade, com foco em modificabilidade voltada aos atributos de qualidade relativos à complexidade, extensibilidade e reuso [33][58].

Para atingir o objetivo citado, os seguintes objetivos intermediários são definidos:

1. Desenvolvimento de um *framework* (versão básica) que facilite o desenvolvimento de LPS para o ambiente de TV Digital, bem como forneça os mecanismos de adaptação e seleção de *features* para construção dinâmica dos produtos de software;
2. Desenvolvimento de extensões, integradas ao citado *framework*, que permitam a construção de aplicações que atendam às principais características de projetos de softwares ubíquos, adequadas ao ambiente de TV Digital.
3. Disponibilizar suporte ao desenvolvimento de mecanismos de tomada de decisão para a seleção dinâmica de *features*.

1.3 Argumento da Tese e Contribuições

Este trabalho propõe o desenvolvimento de um *framework* voltado para o desenvolvimento de Linhas de Produtos de Software no âmbito da TV Digital interativa, no

domínio de aplicações ubíquas, com adequadas condições de manutenibilidade, com foco em modificabilidade das LPS desenvolvidas.

O principal argumento para tese é apresentado a seguir.

O desenvolvimento de aplicações interativas no âmbito de TV Digital, demanda uma considerável necessidade de adaptações aos ambientes nos quais irão executar, devido à diversidade de equipamentos receptores e ao tratamento da interatividade. Além disso, tal desenvolvimento demanda significativa capacidade de manutenção para ajuste de aplicações já existentes aos novos conteúdos produzidos (por exemplo, reuso de parte de uma aplicação de *T-Commerce* para construção de uma loja virtual customizada para um novo filme). Neste sentido, observa-se a necessidade de uma solução em software capaz de permitir a construção de aplicações para TV Digital com características de adaptação a esse ambiente (ubíquas), bem como a definição de uma arquitetura e sua implementação (por meio de um *framework*), de forma a atender a atributos de qualidade de software relativos à manutenibilidade de software (especificamente modificabilidade).

As principais contribuições obtidas são:

- *Framework* (versão básica) para desenvolvimento de LPS no âmbito de TVDi, dotadas de capacidade de adaptação dinâmica, contribuição apresentada através de artigo submetido para a revista *IEICE Transactions on Information and Systems*(Anexo - A).
- *Framework* (versão estendida), denominada DSPL2UbiTV, que permite o desenvolvimento de aplicações ubíquas no âmbito de TV Digital interativa, contribuição apresentada através de artigo submetido à revista *Information and Software Technology, da Elsevier*(Anexo - B).
- Adequação da abordagem FOSD (do inglês, Feature-Oriented Software Development) [10] ao ambiente de TV Digital, para apoio ao desenvolvimento LPS, contribuição apresentada através de aceito para publicação na revista *International Journal of Science and Engineering Applications (IJSEA)*(Anexo - C).

Além das contribuições destacadas anteriormente, pode-se citar algumas contribuições anteriores que ajudaram a definir e aprofundar o problema abordado conforme publicado em [60] [59]. Contribuições futuras podem incluir experimentos dedicados a

demonstrar a capacidade da solução proposta em oferecer e manter aceitáveis características de qualidade de software como usabilidade e eficiência, além de outras características derivadas de manutenibilidade como por exemplo, a estabilidade de software.

1.4 Organização do Trabalho

O presente trabalho é estruturado da seguinte forma:

Capítulo 2 Referencial Teórico relativo às tecnologias utilizadas no desenvolvimento da solução;

Capítulo 3 Trabalhos relacionados que abordam problemas similares aos tratados na tese;

Capítulo 4 Apresentação do *framework* para o desenvolvimento de LPS para TVDi, no domínio de aplicações ubíquas;

Capítulo 4 Descrição dos experimentos considerados, bem como os resultados obtidos;

Capítulo 5 Por fim, são apresentadas as Conclusões e Trabalhos Futuros.

Anexos Os anexos incluem os artigos produzidos até o presente momento, bem como detalhamentos relativos às métricas para avaliação de ALP [47].

1.5 Considerações Finais

Este trabalho apresenta um *framework* para o desenvolvimento de LPS no âmbito da TV Digital interativa, aplicadas ao domínio de aplicações ubíquas. Assim, os principais elementos do *framework* são apresentados, bem como os mecanismos necessários, considerando a necessidade de adaptação para configuração e reconfiguração do produto de software. No tocante à avaliação, considerou-se sob ponto de vista da engenharia de software, a capacidade do *framework* evoluir enquanto software, através da medição de atributos de qualidade de software como complexidade, extensibilidade e reuso. Neste sentido, avaliou-se, inicialmente, a qualidade de software obtida pela solução

proposta, comparando os resultados de aplicações pré-existentes (desenvolvidas utilizando basicamente orientação a objetos) com os resultados obtidos através do *refactoring* das mesmas aplicações contruidas a partir do *framework*. Além disso, avaliou-se a solução quanto aos possíveis impactos gerados pela implementação de aplicações utilizando o *framework*, bem como utilizando as características de ubiquidade necessárias às aplicações. Por fim, validou-se a solução utilizando-a para o desenvolvimento de um estudo de caso, com base em aplicações de TV Digital interativa com suporte a características encontradas em softwares ubíquos.

2 REFERENCIAL TEÓRICO

Inicialmente é necessário destacar alguns conceitos adotados para definição do problema abordado, bem como para a solução proposta. Neste contexto, considerou-se necessário destacar e diferenciar os conceitos utilizados para softwares auto-adaptativos, sensíveis ao contexto, ubíquos e para TV Digital interativa. A seguir são apresentados os conceitos considerados/adotados.

Softwares Auto-Adaptativos são aplicações que modificam seu comportamento em acordo com mudanças ocorridas no ambiente operacional [48].

Softwares Sensíveis ao Contexto são aplicações que se adaptam, tanto para módulos do sistema quanto para informações, em acordo com um conjunto de variáveis de contexto tais como localização, tempo, pessoas e equipamentos [54].

Softwares Ubíquos Weiser [62] define computação ubíqua como hardwares e softwares especializados que são tão onipresentes que nem são percebidos pelos usuários. Complementando a definição, Abowd et al., [7], sugerem que as aplicações ubíquas devem disponibilizar serviços aos usuários de forma transparente e em qualquer lugar ou ambiente a qualquer momento. Assim, pode-se considerar que os softwares sensíveis ao contexto podem estar contidos nos softwares ubíquos. Entretanto, considera-se que o grande diferencial de uma aplicação ubíqua, com relação aos outros conceitos de softwares descritos anteriormente, trata-se da manutenção, de forma onipresente, do serviço e da aplicação ubíqua utilizada.

Softwares para TV Digital interativa Segundo [37], pode-se considerar programas para TV Digital interativa como aplicações contidas no conteúdo televisivo, que possam ser afetadas pela interação do usuário.

Diante dos conceitos apresentados é possível conceituar **Aplicações Ubíquas para TV Digital interativa**. Considerando-se que uma aplicação para TV Digital interativa é disponibilizada através de um canal *broadcast*, e que tal conteúdo pode ser recebido em diversos tipos de receptores, entende-se que a aplicação (limitando-se à

transmissão *broadcast*) deve estar disponível de forma onipresente para todos os receptores (ou seja, se o usuário trocar de receptor continuará tendo acesso ao conteúdo), caso esteja na área de cobertura da emissora pela qual recebia aquele conteúdo.

Entretanto, caso esteja interagindo com algum conteúdo disponibilizado, ao fazer a troca, o usuário perderá as informações relativas ao "status"operacional da interação (ou seja, não será capaz de interagir com a aplicação do ponto que estava antes da troca de receptor, e terá que inicializar toda interação a cada troca de receptor efetuada). Neste caso, apesar do código da aplicação se manter onipresente, a interação não se manterá onipresente. Assim, uma **Aplicação Ubíqua para TV Digital interativa** refere-se a uma aplicação capaz de manter a onipresença quanto ao código e a interação/serviço disponibilizados pela aplicação de forma adequada em todos os ambientes possíveis de recepção.

A seguir são apresentadas seções ligadas a temas que compõem a solução proposta no trabalho, incluindo técnicas consideradas e os desafios encontrados.

2.1 Aplicações para TV Digital interativa

Boa parte das aplicações desenvolvidas para ambientes de TV Digital atualmente se caracterizam pela possibilidade da inclusão de mecanismos de interatividade de forma associada aos conteúdos [37]. Tal característica permite que novos modelos de negócios e informações (como comércio eletrônico e informações de produtos, por exemplo) sejam vinculados aos conteúdos televisivos. Assim, os conteúdos que possuem mecanismos de interatividade tornam-se mais atrativos, ricos e rentáveis.

A interatividade em aplicações para TV Digital permite a inclusão, além de comércio eletrônico, de outras funcionalidades como a busca de informações, ensino à distância, soluções de acessibilidade, jogos, etc. Entretanto, a forma e a quantidade com os quais os recursos de interatividade podem ser apresentados aos telespectadores pode variar de forma significativa, principalmente, considerando conteúdos direcionados a uma diversidade de públicos, regiões, equipamentos, culturas, etc. Recursos de interatividade, quando disponibilizados em excesso ou com limitações, podem tornar os conteúdos desinteressantes, bem como os modelos de negócios relacionados aos mesmos.

Neste contexto, as aplicações que possuem capacidade de adaptação ao ambiente de

recepção tornam-se cada vez interessantes aos telespectadores. Dentre as aplicações que sugerem auto-adaptação pode-se citar as aplicações ubíquas.

2.2 Linhas de Produto de Software

Sendo os conteúdos interativos para TV Digital desenvolvidos com base em soluções de software, e utilizando linguagens de programação variadas (NCL, Lua, Java), é natural buscar soluções para o controle da oferta dos recursos interativos disponíveis nos conteúdos na Engenharia de Software. Através da Engenharia de Software busca-se, além da condição de desenvolvimento, que se obtenham produtos de software de maior qualidade e capazes de, com menor custo, menor incidência de erros e maior velocidade, serem facilmente modificados, expandidos e produzidos.

Considerando a possibilidade de atender telespectadores (usuários) com os mais diversos interesses e necessidades, o desenvolvimento de aplicações interativas para ambientes de TV Digital, e principalmente aplicações ubíquas, torna-se significativamente complexo. Tal complexidade se deve à grande quantidade de módulos de software que devem ser desenvolvidos para suportar a variabilidade de soluções necessárias para atender adequadamente aos usuários e seus terminais de recepção.

Neste sentido, LPS (*Linhas de Produto de Software*) [39] oferecem uma solução, em acordo com a Engenharia de Software, relevante para o planejamento e desenvolvimento de soluções de software, em domínios específicos, que possuem variabilidades na inclusão de módulos de software (*features*) ao produto final disponibilizado ao usuário. SPL consideram que o produto final será definido através da seleção de um conjunto *features* disponíveis na SPL, considerando as *features* mandatórias e um conjunto de *features* opcionais, bem como a relação de dependência entre as *features*, permitindo gerenciar a variabilidade da LPS.

A notação básica de um *Modelo de Features* permite os seguintes tipos de *features* e relações de dependência (Figura 2.1):

Abstract Features Representam as *features* Abstratas, que devem ser definidas através da implementação de *features* Concretas;

Concrete Features representam os módulos de software disponíveis para composição do produto de software.

Figura 2.1: Notação Básica do Modelo de *Features*

Mandatory representa uma *feature* que deve compor, obrigatoriamente, o produto de software extraído a partir da linha de produtos;

Optional representa uma *feature* que pode compor o produto de software extraído a partir da linha de produtos. Neste caso, exige-se a tomada de decisão relativa à inclusão ou não da *feature* no produto final de software;

Alternative Representa que, de todas as *features* envolvidas na relação **Alternative**, apenas uma *feature* pode ser selecionada para compor o produto de software final;

Or Representa que, de todas as *features* envolvidas na relação **Or**, uma ou mais *features* podem ser selecionadas para compor o produto de software final.

Para ambientes de TV Digital interativa, observa-se que a seleção de *features* nem sempre pode ser realizada previamente. Tal seleção deve considerar o ambiente (ou condições) no qual o usuário irá executar a aplicação interativa, ou seja, a seleção de *features* deve ocorrer no momento da execução do software e em acordo com as necessidades e preferências do usuário. Neste sentido, um tipo de LPS que prevê tal comportamento é denominada de LPSD (Linha de Produto de Software Dinâmica) [29][14].

Sendo assim, pode-se utilizar LPSD para definição do produto final disponibilizado ao usuário, uma vez que conteúdos, interativos ou não, em ambientes de TV digital são transmitidos normalmente através de *broadcast*. Tal situação evidencia a necessidade de envio de todo conteúdo (video, voz e dados), independentemente de sua utilização, para todos os usuários. Assim, a utilização (ou execução) de aplicações ubíquas no âmbito de TV Digital pode se beneficiar da utilização de LPSD para a disponibilização de conteúdos interativos.

Finalmente, é importante conceituar o termo "*Framework*" utilizado na solução proposta. Trata-se de um *Framework* de desenvolvimento de software, entendido como um

conjunto cooperativo de classes que compõem um projeto reutilizável para um domínio específico de softwares. O *framework* fornece orientação arquitetural através da divisão do projeto em classes abstratas e definindo suas responsabilidades e colaborações. Um desenvolvedor customisa o *framework* para uma aplicação específica utilizando implementações das classes do *framework* [61].

2.2.1 FOSD - Feature Oriented Software Development

Desenvolvimento de Software Orientado a *Features* (FOSD) é um paradigma para a construção, personalização e síntese de software em sistemas de larga escala. O conceito de *feature* é o cerne de FOSD. Uma *feature* é uma unidade de funcionalidade de um sistema de software que satisfaz uma exigência, representa uma decisão de concepção e proporciona uma potencial opção de configuração. A idéia básica da FOSD é decompor um sistema de software em termos dos recursos que ele oferece. O objetivo da decomposição é construir software bem estruturado que possa ser adaptado às necessidades do usuário e do cenário de aplicação. Normalmente, a partir de um conjunto de *features*, muitos sistemas de software diferentes podem ser gerados, sendo que partilham *features* comuns. O conjunto de sistemas de software gerados a partir de um conjunto de *features* também é chamado de uma linha de produtos de software [10].

2.3 Projetos de Software Ubíquos

Nesta seção são apresentados os principais conceitos e conhecimentos utilizados acerca de computação ubíqua, bem como de projetos de software ubíquos necessários ao entendimento do trabalho proposto.

Mark Weiser [62] definiu os softwares ubíquos como aplicações que deveriam ser onipresentes aos usuários. No entanto, 25 anos se passaram e o desenvolvimento de aplicações ubíquas segue oferecendo desafios [16]. Tais desafios envolvem a capacidade de implementar a onipresença desejada por Mark Weiser, além de mecanismos de desenvolvimento que permitam maior escalabilidade do desenvolvimento, reaproveitamentos dos artefatos de software e obtenção dos atributos de qualidade de software desejados aos usuários.

Assim, analisando o cenário da computação ubíqua sob o ponto de vista da Engenharia de Software, Spinola, R.O. e Travassos, G.H., [57] desenvolveram pesquisas buscando caracterizar os projetos de softwares ubíquos, identificando características funcionais e restritivas encontradas nos diversos projetos de software analisados. Abaixo são listadas as características funcionais identificadas na pesquisa desenvolvida.

Onipresença dos Serviços "*permitir a movimentação física do usuário, dando a ele a percepção física de estar levando consigo os serviços computacionais...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Invisibilidade "*Capacidade de estar presente nos objetos de uso do dia-a-dia, descaracterizando, do ponto de vista do usuário, a utilização de um computador...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Sensibilidade ao Contexto "*Capacidade de coletar informações sobre o ambiente em que está sendo utilizado...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Comportamento Adaptável "*Capacidade de, dinâmicamente, adaptar os serviços disponíveis ao ambiente onde está sendo utilizado...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Captura de Experiências "*capacidade de capturar e registrar experiências para uso posterior...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Descoberta de Serviços "*descobrir serviços proatiamente de acordo com o ambiente que se encontra...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Composição de Funcionalidade "*Capacidade de, a partir de serviços básicos, montar uma determinada funcionalidade requerida pelo usuário...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Interoperabilidade Espontânea "*capacidade interagir com outros dispositivos durante a sua operação e conforme sua movimentação...*;"(Spinola , R.O. e Travassos, G.H.,[57])

Heterogeneidade de Dispositivos "*prover mobilidade da aplicação através de dispositivos hererogêneos...*;"(Spinola, R.O. e Travassos, G.H.,[57])

Além disso, em seu trabalho, Spinola, R.O. e Travassos, G.H., [57], buscaram, através de experimentos, identificar se as características identificadas eram relevantes e pertinentes ao domínio das aplicações ubíquas. Assim, apresentaram resultados (Figura 2.2) que demonstram a relevância e pertinência de tais características sob a visão de especialistas da área.

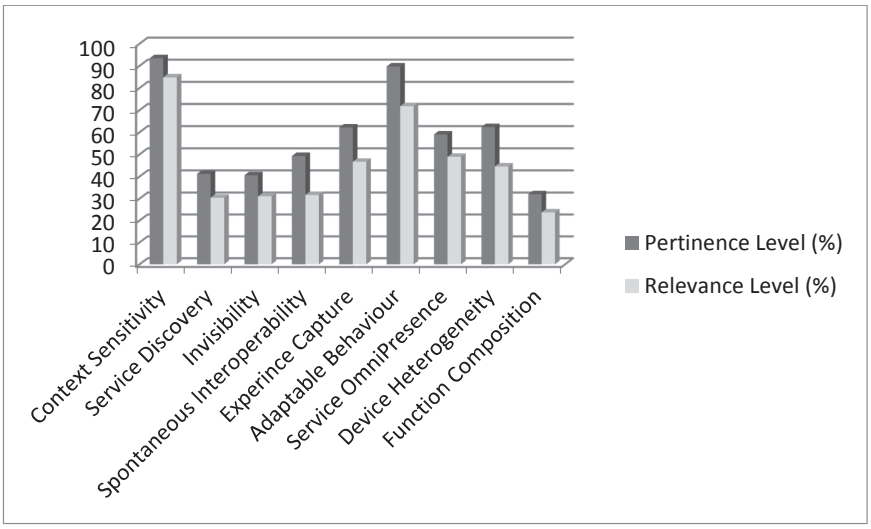

Figura 2.2: Resultados obtidos através das pesquisas de Spinola, R.O. e Travassos, G.H. [57]

É possível observar, através dos resultados obtidos por Spinola, R.O. e Travassos, G.H., [57], que algumas características são consideradas mais ou menos relevantes. Entretanto, todas são consideradas pertinentes por um grupo considerável de avaliadores. Além disso, pode-se observar que, em [57], características como "Onipresença de Serviços"são encontradas em aproximadamente, 50% (cinqüenta por cento) dos projetos avaliados, apesar de a onipresença de serviços ser considerada uma das a principais características de uma aplicação ubíqua.

Assim, este trabalho buscou dar suporte, através de ferramentas para o desenvolvimento de software, a maioria das características funcionais identificadas por Spinola, R.O. e Travassos, G.H., [57] No entanto, devido à complexidade e tempo necessários para desenvolvimento do trabalho, a implementação das características consideradas restritivas (como, por exemplo, Tolerância a Falhas, Escalabilidade, Privacidade e Confiança, etc) são consideradas para trabalhos futuros. Além disso, a característica de "Composição de Funcionalidades"não foi considerada suficientemente relevante e pertinente para projetos ubíquos [57]. Assim, a "Composição de Funcionalidades"não será considerada em todos os experimentos, devido à menor prioridade dada nos projeto e implementação da solução proposta na tese.

Adicionalmente, por razões de escopo deste trabalho, não serão implementados os recursos necessários às seguintes características: descoberta de serviços, invisibilidade e interoperabilidade espontânea.

2.4 Sistemas de Recomendação Cientes de Contexto

A incorporação de informações relativas ao contexto [8] é de grande importância para que os modernos sistemas de recomendação possam se mostrar eficazes.

O modelo multidimensional apresentado em [9] propõe soluções para predição de recomendações em novos contextos (um dos problemas mais importantes quando se trata de *context-aware filtering*) e é utilizado inclusive, em propostas para sistemas de recomendação móveis recentes [31]. Assim, considerando sua relevância, a seguir é apresentado o modelo multidimensional para sistemas de recomendação, bem como suas maiores contribuições relativas às predições de recomendação em novos contextos.

2.4.1 Modelo Multidimensional para Sistemas de Recomendação

O Modelo multimendimensional apresentado inicialmente em [8][9], permite que se forneçam recomendações considerando não apenas um modelo de duas dimensões (modelo clássico em sistemas de recomendação) utilizando a relação $User \times Item$, como também uma relação que pode envolver outros fatores (dimensões) no fornecimento da recomendação. Assim, o modelo multidimensional considera dimensões como usuário, item, tempo, local, etc; que, como pode-se observar, introduz a noção de filtragem ciente de contexto ao modelo.

Formalmente deve-se identificar o conjunto $D = D_1, D_2, D_3, ..., D_n$ de dimensões utilizadas, e utilizar uma função de classificação (*rating function*) (2.1) para obter a classificação ou taxa de recomendação ao espaço multidimensional apresentado.

$$R : D_1 \times D_2 \times ... \times D_n \longrightarrow Rating \tag{2.1}$$

Pode-se então, considerar o exemplo apresentado na (Figura 2.3), onde apresenta-se o modelo espacial, em forma de cubo, para busca e armazenamento da informação desejada. Na Figura 2.3, pode-se observar a utilização de um espaço dimensional composto por $User \times Item \times Time$, na qual $u \in User$, e representa a identificação de um usuário específico; $i \in Item$, representando um item específico e $t \in Time$,

representando um intervalo de tempo específico. Assim, utilizando a função (2.2), obtem-se o valor desejado.

$$R(u, i, t) \longrightarrow Rating \tag{2.2}$$

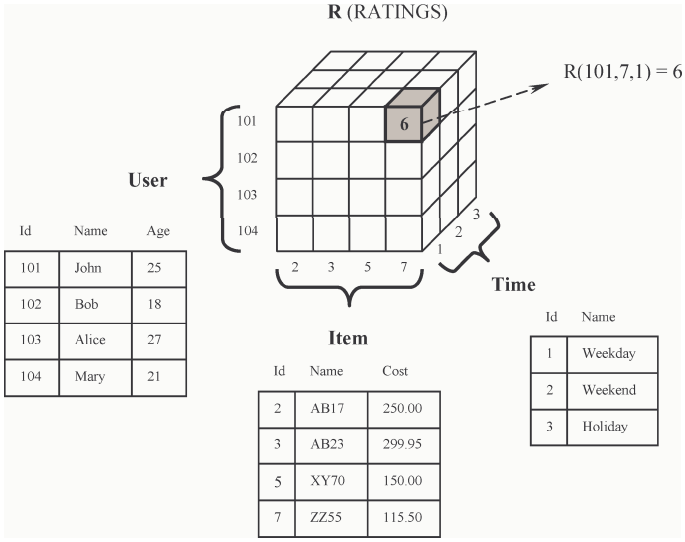

Figura 2.3: Modelo Multidimensional [8]

2.4.1.1 Agregação Hierárquica de Classificações

Também em [8], outra noção relevante foi acrescentada à filtragem ciente de contexto. Trata-se da noção de que pode-se obter classificações baseadas em contextos hierárquicos. É possível, por exemplo, que se tenha classificações (ou taxas) relativas a intervalos de dias da semana, e obter uma classificação relativa somente ao fim-de-semana. Os autores demonstram que tal noção é relevante através de experimentos científicos, e apresentam como é possível representar informações hierárquicas através de agregações de classificações utilizando o modelo multidimensional apresentado anteriormente.

Os experimentos científicos realizados em [8] demonstram também que quanto maior a granularidade da agregação hierárquica, melhor o desempenho da classificação obtida. Em outras palavras, pode-se concluir que quanto mais específico o grau hierárquico utilizado, mais precisos são os resultados.

2.4.1.2 Predição de Classificações

As técnicas utilizadas para determinar a agregação das classificações (ou taxas) de forma hierárquica em [8], evidenciaram um problema relativo à falta de classificação em algumas dimensões utilizadas, fator que tornaria inviável a utilização de tais técnicas em algumas situações. Assim, em [8] e [9] os autores propõem soluções para minimizar o problema.

A solução apresentada inicialmente, utiliza-se de técnicas clássicas como *Collaborative Filtering - CF* combinadas com a proposta de redução de dimensões no modelo multidimensional. Como é de conhecimento da comunidade acadêmica, as técnicas de *CF* são utilizadas para obter classificações (ou taxas) desconhecidas, utilizando a hipótese de que elementos que possuem similaridade entre seus perfis, podem ajudar a construir uma classificação desconhecida aos seus similares.

No entanto, as técnicas tradicionais de *CF* não dão suporte e múltiplas dimensões de informações, sendo capazes apenas, de serem utilizadas em duas dimensões. Assim, os autores propõem uma solução para redução das dimensões em classificações multidimensionais. A *Reduction-based Approach*, reduz o problema de recomendações multidimensionais para o espaço tradicional de recomendação bidimensional. É importante destacar que tais soluções apresentadas, não possuem suporte à predição de classificação em contextos hierárquicos.

A técnica proposta baseia-se na necessidade de obter os valores válidos para duas dimensões, assim a proposta inicial é utilizar a mesma função de agregação utilizada anteriormente com o intuito de filtrar os valores, dados os parâmetros que compõem as dimensões adicionais. Considerando que existem várias entradas no sistema que identificam um usuário, um conteúdo e um intervalo de tempo ($User \times Content \times Time$), e deseja-se obter somente a relação $User \times Content$. Identifica-se o valor (taxa) t desejado, onde $D[Time = t]$, e seleciona-se apenas os registros onde na dimensão de tempo ($Time$) a classificação é igual a t, mantendo os valores de $User$ e $Content$. Então uma função de predição tridimensional (2.3) pode ser reduzida para uma função de predição bidimensional (2.4) como segue:

$$R^{D}_{User \times Content \times Time} : User \times Content \times Time \longrightarrow Rating, \tag{2.3}$$

$$R^{D}_{User \times Content \times Time}(u, c, t) = R^{D[Time=t](User,Content,rating)}_{User \times Content}(u, c) \quad (2.4)$$

Assim, uma vez que possuem duas dimensões, pode-se utilizar das técnicas de *CF* para predição de classificações desconhecidas. Os autores demonstram que a proposta de utilização da *Reduction-based CF Approach* é significativa quando comparada a soluções tradicionais de CF, além de destacarem a necessidade de desenvolvimento de soluções para predição de classificação em contextos hierárquicos.

2.4.1.3 Predição de Classificação em contextos hierárquicos

A predição (ou a inferência) de classificação (ou de probabilidades) utilizando informações contextuais hierárquicas possui proposta de solução apresentada em [27], [49]. Tais propostas, utilizam o modelo multidimensional apresentado anteriormente como base, no entanto, utilizam-se de informações para classificar a hierarquia contextual. Em outras palavras, utilizam-se de um conjunto de atributos para geração de estatísticas preditivas ($Y = X_1, X_2, ...X_n$) de interesse para um usuário, podendo ser estendida para grupos de usuários com interesses similares. Além disso, existe um conjunto K de variáveis contextuais, classificados em q níveis hierárquicos.

A predição pode ser obtida, segundo os autores, através de uma função preditiva, que pode utilizar-se de diferentes métodos de máquinas de aprendizado (*Machine learning*), como regressão, árvores de decisão, redes neurais, etc. Assim para obter-se a predição considerando um conjunto de atributos, sem filtragem contextual é utilizado o seguinte modelo (2.5):

$$W = f(X_1, X_2, ...X_n) \quad (2.5)$$

No entanto, se deseja-se obter a predição filtrando as informações contextuais, o modelo apresentado anteriormente sofre uma pequena variação em sua apresentação (2.6). Deve-se apresentar o valor do atributo contextual (α) a ser filtrado, e em qual nível hierárquico (q) o mesmo está representado. Sendo assim, o modelo considerando a filtragem contextual é apresentado a seguir:

$$W = f_{k^q=\alpha}(X_1, X_2, ..., X_n) \tag{2.6}$$

Entretanto, em algumas situações, é necessário predizer a informação contextual. Neste caso, são apresentados dois modelos:

$$K^q = f(X_1, X_2, ...X_n) \tag{2.7}$$

$$K^{hier} = f(X_1, X_2, ...X_n) \tag{2.8}$$

No modelo representado por (2.7) utiliza-se de uma função preditiva definida como uma *Naïve Bayes Network*, onde apenas um atributo contextual pode ser inferido por vez. Por outro lado, o modelo representado por (2.8) utiliza-se de uma Rede Bayesiana (*Bayesian Network*) como função de inferência, sendo a estrutura hierárquica do qual depende da variável contextual inferida, considerada no momento da predição.

Experimentos foram executados [27] [49], e os autores chegaram a resultados que indicaram a melhoria de performance das recomendações quando utilizaram-se de informações contextuais e hierárquicas. Observaram também melhoria de desempenho quando necessitaram inferir preferências com relação à variável contextual, obtendo melhor resultado utilizando a função baseada em Redes Bayesianas (2.8).

2.5 Qualidade de Software - Manutenibilidade

O trabalho proposto considera a necessidade de melhoria na qualidade dos softwares desenvolvidos com foco na manutenibilidade de software. A manutenibilidade de software trata-se de uma característica de qualidade de software apresentada na ISO/IEC 9126 [33]. A ISO/IEC 9126 e apresenta também algumas sub-características relativas à características de manutenibilidade. As sub-características são citadas a seguir:

- **Analisabilidade** identifica a facilidade em se diagnosticar eventuais problemas e identificar as causas das deficiências ou falhas;

- **Modificabilidade** caracteriza a facilidade com que o comportamento do software pode ser modificado;
- **Estabilidade** avalia a capacidade do software de evitar efeitos colaterais decorrentes de modificações introduzidas;
- **Testabilidade** representa a capacidade de se testar o sistema modificado, tanto pelas novas funcionalidades quanto pelas não afetadas diretamente pela modificação;
- **Conformidade** trata-se da padronização, politicas e normas de um projeto.

Este trabalho considerou o foco na sub-característica de modificabilidade para avaliação da manutenibilidade de software, por considerar tal característica de extrema relevância para a evolução do software. Além disso, este trabalho considera a necessidade de avaliar atributos de qualidade internos ao software, ou seja, atributos que são avaliados através de artefatos de construção do software.

A característica de modificabilidade possui vários atributos de qualidade interna que são definidos na ISO/IEC 9126-3 [58]. Dentre os atributos disponíveis, este trabalho considera, para avaliação dos experimentos, os atributos de complexidade, extensibilidade e reuso. Os atributos de qualidade citados anteriormente foram considerados devido a destacada utilização em outros trabalhos acadêmicos [46] [43], bem como reuso de software ser o principal benefício considerado pelas LPSs.

Uma vez definidos os atributos de qualidade considerados para avaliação do trabalho proposto, é necessário definir as métricas de qualidade que serão utilizadas para medir a qualidade dos atributos.

Como este trabalho aborda a implementação de um *framework* para desenvolvimento de Linhas de Produto de Software construido baseado no paradigma de orientação a objetos, entende-se que as métricas utilizadas devem: (i) avaliar os atributos de qualidade selecionados, (ii) avaliar a arquitetura da LPSs (ALP) e (iii) utilizar métricas relativas a orientação a objetos.

Assim, pode-se encontrar várias métricas capazes de avaliar, de alguma forma, uma LPS [43]. Entretanto, nem todas são capazes de avaliar uma LPS a partir de informações provenientes de artefatos de orientação a objetos. Dentre as possibilidades

consideradas, optou-se pela utilização das métricas de ALP propostas por [47]. Tal abordagem de avaliação foi escolhida por fazer uso, bem como permitir a definição, de métricas consagradas para avalição de classes, no âmbito da orientação a objetos, como WMC, DIT, NOC [18], para construção das métricas de ALP.

2.6 Considerações Finais

A revisão de literatura demonstra que existe uma grande quantidade de problemas a serem tratados exclusivamente nos tópicos abordados como TV Digital, LPS, Aplicações Ubíquas e Sistemas de Recomendação. Entretanto, existe a necessidade de desenvolver tecnologias capazes de evoluir os temas abordados de forma integrada quando os problemas tratados requerem isso.

O trabalho proposto nesta Tese encontra desafios que necessitam de evoluções pontuais em cada um dos temas abordados anteriormente, existindo a necessidade de que as evoluções sejam tratadas de forma integrada para que possibilitem a solução do problema tratado.

3 TRABALHOS RELACIONADOS

Os trabalhos serão aqui tratados com base em soluções relativos a :

- Suporte ao desenvolvimento de aplicações para TVDi.
- Suporte a características de ubiquidade;

3.1 Suporte ao desenvolvimento de aplicações para TVDi

Quando analisada na literatura, a disponibilidade de ferramentas que ofereçam possibilidade de reuso de software para o desenvolvimento de aplicações no âmbito de TV Digital, foram identificadas algumas soluções baseadas em abordagens como :

1. *Frameworks* de desenvolvimento, tais como : TUIG [45], GingaGAME [11], FrameIDTV [51];
2. Soluções baseadas em componentização/APIs, tais como : TUGA [53], iTVProject [45], Athus [55];
3. Por fim, soluções voltadas para necessidades especiais do usuário, tal como o *Framework* GUIDE [34].

Com base no *framework* TUIG [45], é possível gerar interfaces gráficas automaticamente através de um arquivo XML que define os componentes que estarão presentes na tela. Este arquivo pode definir os atributos dos componentes, tais como gráficos, cor, tamanho e eventos a serem lançados a partir do gatilho de um determinado componente. Através do processamento do arquivo XML, as classes responsáveis por gerar as interfaces gráficas são geradas automaticamente. Assim, o *framework* TUIG é responsável por interpretar XMLs específicos e gerar as classes Java correspondentes às aplicações interativas. O *framework* TUIG não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, o *framework* não apresenta suporte à geração de códigos dinâmicos em tempo de execução. Finalmente, o *framework* não foi avaliado

de forma quantitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

O GingaGAME [11] é um *framework* de desenvolvimento de jogos para a TV Digital. O *framework* é subdividido em dois diferentes pacotes Java (GameComponent e GingaGameJavaTV), de forma que as classes que precisam de recursos específicos da plataforma estão em um pacote separado das classes que não têm este tipo de dependência. O pacote GameComponent possui componentes que devem ser adicionados a objetos em uma cena do jogo. O pacote GingaGameJavaTV inclui recursos específicos da plataforma como, por exemplo, o gerenciador de janelas. O GingaGAME não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, o *framework* não apresenta suporte à geração de códigos dinâmicos em tempo de execução. Finalmente, o *framework* não foi avaliado de forma quantitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

O FrameIDTV [51] se propõe a dar suporte aos principais requisitos para aplicações de TV Digital Interativa através de um conjunto de pacotes organizados em uma arquitetura MVC (*Model-View-Controller*). Neste sentido oferece suporte à camada *View* através do Pacote **UI**, suporte à camada *Controller* através dos pacotes **Xlet**, **Util** e **Net**, bem como suporte à camada *Model* através do pacote **Data**. O FrameIDTV não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, o *framework* não apresenta suporte à geração de códigos dinâmicos em tempo de execução. Finalmente, o *framework* não foi avaliado de forma qualitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

O *middleware* TUGA [53] se propõe a oferecer suporte ao desenvolvimento de jogos em TVDI. Esse *middleware*, além da estrutura de execução dos jogos, disponibiliza também uma API de desenvolvimento de aplicações para a TV digital, onde os principais componentes envolvem as seguintes sistemas:

- Sistema Gráfico;
- Sistema Sonoro;
- Sistema de Entrada.

Entretanto, o *middleware* TUGA não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, não apresenta suporte à geração de códigos dinâmicos em

tempo de execução. Finalmente, não foi avaliado de forma quantitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

O iTVProject [45] é uma ferramenta de autoria web, cujo foco é o rápido desenvolvimento de aplicações interativas para TV Digital. A proposta do iTVProject é a criação de um ambiente Web para o desenvolvimento de aplicativos interativos para televisão digital, executável em ambos midllewares das normas europeias e brasileiras - DVBMHP e Ginga-J, respectivamente. Além disso, o ambiente e projetos criados a partir iTVProject são acessíveis de qualquer computador com um navegador e uma conexão para Internet. A ferramenta iTVProject utiliza o *framework* TUIG para geração das classe java. Assim, seu escopo se delimita a gerar os XMLs adequados ao *framework* TUIG. O iTVProject não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, a ferramenta não apresenta suporte à geração de códigos dinâmicos em tempo de execução. Finalmente, não foi avaliado de forma quantitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

O Projeto ATHUS [55] também apresenta-se como uma solução para o desenvolvimento de jogos em TVDi. Por ser mais recente que os outros projetos citados, o ATHUS apresenta suporte a um conjunto maior de funcionalidades. Entretanto, o ATHUS foi criado para desenvolvimento de código declarativo, específico para solução nacional (Ginga-NCL), diferentemente de outras soluções apresentadas que podem ser utilizadas em outros padrões, como o europeu (MHP), através de desenvolvimento procedural. O ATHUS não apresenta suporte ao desenvolvimento de LPS originalmente. Além disso, a ferramenta não apresenta suporte à geração de códigos dinâmicos em tempo de execução, declaradamente. Finalmente, não foi avaliado de forma quantitativa com relação à produção dos softwares desenvolvidos utilizando tal solução, apenas qualitativa através de pesquisa de satisfação feita aos desenvolvedores que testaram a solução.

O *framework* GUIDE [34] é uma solução que permite o desenvolvimento de aplicativos para TVDi interativa, com características de adaptação, voltadas para acessibilidade. Neste contexto a ferramenta oferece suporte à auto-adaptação do software em acordo com as necessidades específicas de cada usuário. O GUIDE não apresenta suporte ao desenvolvimento de LPS originalmente. Entretanto, o *framework* apresenta suporte à geração de códigos dinâmicos em tempo de execução. Finalmente, no *framework* não existe informações sobre avaliação de forma qualitativa com relação à produção dos softwares desenvolvidos utilizando tal solução.

3.1.1 Comparativo entre as soluções

Comparativamente analisamos as soluções com relação a 4 (quatro) aspectos:

- **SRS** - Solução de reuso de software
- **MAS** - Oferta de mecanismos de adaptação de software;
- **AQS** - Análise quanto a qualidade do software produzido com base nas soluções;
- **LPS** - Possibilidade de desenvolvimento de LPS.

Assim, a tabela (3.1) abaixo apresenta um quadro comparativo.

Trabalhos Relacionados	SRS	MAS	AQS	LPS
TUIG	X			
iTVProject	X			
FrameIDTV	X			
TUGA	X			
GingaGAME	X			
ATHUS	X			
GUIDE	X	X		
Framework DSPL2UbiTV	X	X	X	X

Tabela 3.1: Quadro comparativo entre Soluções para o desenvolvimento de aplicações de TV Digital interativa

Conclui-se que as soluções observadas, em sua grande maioria, não oferecem suporte a softwares adaptativos. Além disso, nenhuma proposta foi encontrada com o objetivo de permitir que LPS sejam implementadas em um ambiente de TV Digital. Finalmente, as soluções analisadas não apresentam avaliação da qualidade do reuso oferecido.

Além dos trabalhos citados anteriomente , pode-se citar trabalhos ligados à modelagem de aplicações para TV Digital interativa [44]. Onde o objetivo é organizar e estruturar o processo de desenvolvimento de conteúdos multimídia.

3.2 Suporte a características de ubiquidade

Dentre os trabalhos que possuem maior relação quanto ao tema de desenvolvimento de software ubíquo para TV Digital interativa, principalmente quando considera-se

a necessidade de ferramentas que permitam a manutenibilidade, pode-se destacar as seguintes soluções:

Em Oliveira et al. [22] é proposta uma solução que utiliza MDA (*Model Driven Architecture)* e LPS para o desenvolvimento de aplicações ubíquas. Neste sentido, os autores utilizam o *framework* denominado UCF (*Ubiquitous Computing Framework*) como componente básico de software para o suporte à implementação das funcionalidades relativas à ubiquidade. O UCF é capaz de dar suporte à implementação de características de ubiquidade como: sensibilidade ao contexto, comportamento adaptável e heterogeneidade de dispositivos. Entretanto, tal solução, mesmo em se tratando de uma LPS, não é adequada ao ambiente de TV Digital interativa, uma vez que o produto de software deve ser pré-selecionado antes do envio via *broadcast* ao usuário/telespectador. Além disso, a solução apresentada não oferece suporte ao desenvolvimento da onipresença do serviço oferecido pela aplicação, uma das principais características que diferenciam aplicações ubíquas de aplicações cientes de contexto.

Parra et al. [50] desenvolvem trabalho voltado a oferecer uma linha de produto de software dinâmica, orientada a serviços, para o domínio de Aplicações Sensíveis ao Contexto. O trabalho de Parra et al., possui similaridades com o trabalho proposto devido ao uso de Linhas de produto de Software Dinâmicas e por atender as características de produto de software ubíquos denominadas "Sensibilidade ao Contexto"e "Comportamento Adaptável". No entanto, tal solução não oferece suporte à implementação das demais características de ubiquidade citadas/avaliadas por Spinola, R.O. e Travassos, G.H., [57], bem como não oferece suporte específico à aplicações de TV Digital interativa.

Hallsteinsen et al. [28] desenvolvem um trabalho focado em oferecer um processo de desenvolvimento de aplicações ubíquas, agregado a uma solução baseada em um *middleware* capaz oferecer suporte à sensibilidade de contexto e à auto-adaptação de software. Além disso, oferece um conjunto de componentes reutilizáveis como apoio ao desenvolvimento das aplicações que rodaram sobre o *middleware*. No entanto, tal trabalho diferencia-se da solução proposta, principalmente, quanto ao suporte à aplicações de TV Digital interativa, bem como, por não caracterizar adequadamente o suporte às características de ubiquidade necessárias (apesar de citar o suporte ao desenvolvimento de aplicações ubíquas).

Hussein et al. [32] apresentam um *framework* para desenvolvimento de aplicações

sensíveis ao contexto e também sistemas de recomendação. Entretanto, assim como outras soluções, não oferece suporte a outras características de softwares ubíquos, inclusive "Comportamento Adaptável". Além disso, o *framework* proposto por Hussein et al., não apresenta suporte ao desenvolvimento de aplicações para TV Digital interativa, bem como não utiliza linhas de produto de software como base de desenvolvimento.

Cetina et al. [17] introduzem uma abordagem para projetar LPS pervasivas baseadas em *Model Driven Development*(MDD) e princípios de modelo de variabilidade. O trabalho apresenta solução para reconfiguração em tempo de execução , permitindo que uma SPL Pervasiva se adapte de forma autônoma. Porém, o trabalho não apresenta suporte às aplicações para TV Digital. Além disso, não define claramente o suporte às características de projetos ubíquos.

No tocante às aplicações ubíquas para TV Digital interativa, propriamente ditas, pode-se encontrar apenas alguns estudos de caso, específicos, voltados a integração de aplicações de TV Digital interativa e *Home Networks* como [38], [25], [41] e [21]. No entanto, nenhum dos trabalhos trata de uma solução para o desenvolvimento de aplicações. Além disso, encontra-se trabalho destinado a avaliar à ubiquidade dos conteúdos em ambientes televisivos [24].

3.2.1 Comparativo entre as Soluções

Dentre os elementos utilizados para comparar as soluções apresentadas anteriormente e o trabalho aqui proposto, considerou-se 3 (três) tipos de suporte ao desenvolvimento de software relacionados às contribuições que as soluções comparadas podem oferecer. Cabe ressaltar que a avaliação foi realizada, apenas, através de bases teóricas.

Tratam-se do suporte à:

Manutenibilidade de Software Segundo [26], a manutenibilidade de software é a facilidade com a qual um sistema de software ou componente pode ser modificado para corrigir falhas, melhorar o desempenho ou outros atributos, ou adaptar-se a uma mudança no ambiente operacional. Assim, buscou-se avaliar se as soluções propostas possuem algum tipo de suporte à manutenibilidade de software, do ponto de vista arquitetural.

Características de Projetos Ubíquos Buscou-se avaliar se as soluções propostas oferecem algum suporte a características de projetos de software ubíquos. Assim, as características funcionais identificadas em [57] foram utilizadas como base de avaliação. Cabe destacar que na Tabela (3.2) as principais características de ubiquidade(em termos de relevância e pertinência) possuem sua avaliação destacada de forma individual, devido a variação de suporte apresentado pelas soluções comparadas.

TV Digital interativa Buscou-se identificar suporte à implementação de aplicações para TV Digital interativa, tanto com relação ao suporte a API's específicas quanto ao suporte a características e limitações de execução de softwares interativos para TV Digital.

A listagem abaixo define as siglas utilizadas para cada elemento avaliado na comparação e utilizado na Tabela (3.2):

MS - Suporte à Manutenibilidade de Software

SC - Sensibilidade ao Contexto

CA - Comportamento Adaptável

HD - Heterogeneidade de Dispositivos

CE - Captura de Experiências

OS - Onipresença de Serviços

TVDi - Suporte à TV Digital interativa

Trabalhos Relacionados	MS	SC	CA	HD	CE	OS	TVDi
Oliveira et al., [22]	X	X	X	X			
Parra et al., [50]	X	X	X	X			
Hallsteinsen et al., [28]	X	X	X	X			
Hussein et al., [32]	X	X	X	X			
Cetina et al., [17]	X	X	X	X			
Framework DSPL2UbiTV	X	X	X	X	X	X	X

Tabela 3.2: Elementos considerados para comparação entre soluções relacionadas

Cabe ressaltar que a análise sobre o nível de suporte dos trabalhos relacionados a cada elemento utilizado, para comparação foi significativamente abstrato, sendo baseado apenas na descrição feita nos trabalhos publicados. Além disso, outras características de ubiquidade não foram citadas, pois nenhuma das soluções utilizadas no comparativo, incluindo a solução proposta, apresenta suporte claro a tais características.

Assim, concluiu-se que as soluções relacionadas a este trabalho, voltadas para o tratamento de características de ubiquidade:

- Possuem algum suporte à manutenibilidade de software, entretanto, não apresentam avaliação relativa à qualidade oferecida para manutenção de software;
- Oferecem suporte às principais características de ubiquidade consideradas relevantes e pertinentes a um projeto de software ubíquos, porém, não oferecem suporte à onipresença, bem como não oferecem suporte a outras características importantes como captura de experiências e interoperabilidade espontânea;
- Finalmente, nenhuma das soluções analisadas (relacionadas ao suporte a características de ubiquidade), oferece suporte à implementação de aplicações interativas para TV Digital.

3.3 Considerações Finais

Os trabalhos relacionados foram apresentados de forma a evidenciar a necessidade de solução para o desenvolvimento de software no âmbito de TV Digital, com relação ao desenvolvimento de Linhas de Produtos de Software (LPS), bem como suporte a características de projetos ubíqüos de forma integrada. Além disso, procurou-se analisar junto aos trabalhos relacionados o tipo de avaliação feita nas ferramentas, buscando analisar se avaliações relativas a atributos de qualidade de software foram tratadas.

Observa-se nos trabalhos relacionados que não existe integração entre as soluções que oferecem suporte ao desenvolvimento de aplicações para TV Digital, suporte ao desenvolvimento de LPS, bem como ao desenvolvimento de aplicações ubíquas. Além disso, observa-se que as avaliações são, quando existem, qualitativas e não seguem um método ciêntifico claro, além de não abordarem nenhum atributo de qualidade de software especificamente. Assim, conclui-se que se faz necessária uma solução capaz de

integrar os requisitos de TVDi, LPS e ubiquidade, bem como de avaliação quantitativa da solução.

4 *FRAMEWORK* DSPL2UbiTV

Neste capítulo são apresentadas informações relativas ao contexto de operação e ao modelo de *features* do *framework*, sua arquitetura geral, bem como a adequação do *framework* ao processo FOSD.

4.1 DESCRIÇÃO DO CONTEXTO DE OPERAÇÃO DO *FRAMEWORK*

Os cenários de operação do *framework* devem ser descritos através de dois focos distintos. O primeiro foco é relativo a Linhas de Produtos de Software (LPS), e o segundo foco relativo a aplicações ubíquas. Entretanto, o *framework* foi concebido para um cenário mais abrangente ligado a TV Digital Interativa, no qual LPS e aplicações ubíquas estão integrados.

4.1.1 Linhas de Produtos de Software e TV Digital Interativa

Com relação à proposta do *framework* de auxiliar o desenvolvimento de linhas de produtos de software no âmbito de TV Digital interativa, alguns elementos, relativos à operação do framework em tal ambiente, devem ser destacados.

A seguir são apresentados o contexto de operação e cenários de execução do *framework* em ambientes distintos de TVDi, bem como características e particularidades de seleção de produtos de software no âmbito de TVDi.

4.1.1.1 Contexto de operação e cenários de execução do *framework*

O *framework* foi desenvolvido para operação em um ambiente de transmissão broadcast. Neste sentido o principal problema encontrado trata-se da seleção do produto de software a partir da LPS, uma vez que não é possível identificar, previamente, qual o equipamento será utilizado para recepção do conteúdo interativo. Assim, questões

relativas aos recursos, como por exemplo a memória disponível e resolução de vídeo suportados, para execução não podem ser tratados definitivamente na emissora. Assim, o *framework* deve prover solução para configuração do produto adequado ao software no receptor do conteúdo.

Antes de apresentar detalhes do *framework*, com relação a seleção do produto de software executado no receptor, é importante compreender quais os principais atores considerados para operação do *framework*, bem como quais os principais ambientes de execução, como por exemplo o *middleware* Ginga-J, para os quais o *framework* foi projetado.

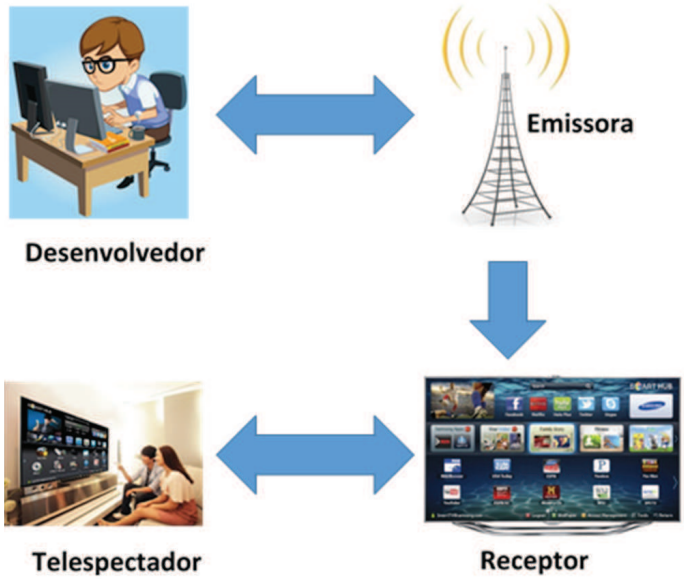

Figura 4.1: Contexto de Uso do *Framework* DSPL2UbiTV

A Figura 4.1 apresenta o contexto de uso do *framework* no âmbito de TV Digital, considerando transmissão via broadcast e controlada pelo gerador de carrossel. Pode-se observar, na figura citada, que dois atores possuem relação com o *framework*:

- O ator denominado Desenvolvedor é o principal usuário do *framework*, pois a partir do *framework* o desenvolvedor construirá as LPS solicitadas/disponibilizadas pelas Emissoras;
- O ator denominado Telespectador, também comumente chamado de usuário, recebe o resultado da operação do *framework*, no sentido de sejam construídas aplicações adequadas a este ator.

Além disso, deve-se destacar o tipo de relação entre os elementos que compõem o contexto de execução:

- O desenvolvedor constrói as LPS atendendo os requisitos de software determinados ou aceitos pela emissora. Neste sentido, além dos requisitos funcionais, o desenvolvedor deve considerar requisitos não funcionais, como por exemplo tipos de receptores (por exemplo TVs, smartphones e *tablets*) e telespectadores (por exemplo perfis de interatividade e preferenciais) atingidos pela transmissão;
- A emissora é responsável por transmitir o conteúdo interativo (no caso: *framework* + LPS) através de um canal *broadcast* e utilizar um gerador de carrossel (responsável pela transmissão dos dados) para controlar a transmissão das aplicações. Dada que a transmissão é feita através de broadcast, a LPS como um todo (uma vez considerada adequada/validada pelo difusor) deve ser transmitida, deixando a tarefa de seleção do produto para ser definida no receptor (STB);
- O receptor (STB) recebe o conteúdo interativo e, ao ser inicializada a execução da aplicação, o *framework* construirá a aplicação, selecionando as *feature*, a partir da LPS, adequadas ao contexto de execução, de acordo com as regras de construção implementadas pelo desenvolvedor;
- Ao telespectador/usuário cabe a interação com o conteúdo interativo disponibilizado.

Assim, as duas principais funções do *framework* com relação à integração com TVDi são:

- Suporte ao desenvolvimento de Linhas de Produtos de Software no âmbito de TV Digital;
- Suporte à seleção e construção do produto de software no receptor.

Por outro lado, para permitir a construção de LPS no âmbito de TV Digital interativa, foi necessário definir qual tecnologia era mais adequada, para desenvolvimento de aplicações, aos propósitos do *framework*. Neste sentido optou-se pela tecnologia Java, mais especificamente, devido à disponibilidade de uma API, denominada JavaTV, que permite a construção de aplicações para TV Digital interativa procedurais, suportada

pelas principais implementações de middlewares para TV Digital. Outras possibilidades de tecnologias, como por exemplo NCL
Lua e DVB-HTML, também foram consideradas. No entanto, outras tecnologias são limitadas, normalmente, a apenas uma implementação de *middleware*, acarretando em uma restrição à portabilidade do *framework*, caso fossem utilizadas.

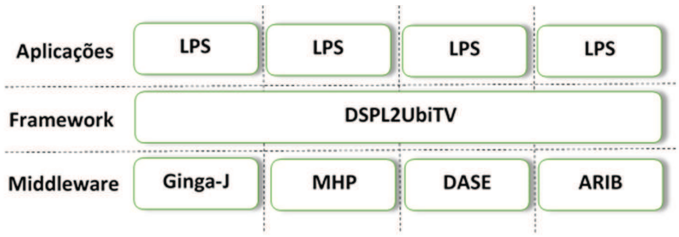

Figura 4.2: Cenário de Execução

A Figura 4.2 apresenta o cenário de execução considerado para o *framework* DSPL2UbiTV. Em tal cenário é possível compreender que o *framework* possui suporte para execução nos principais *middlewares* disponíveis globalmente, devido, principalmente, à construção com base na API JavaTV. Cabe destacar que as funções do *framework* não são afetadas pelas particularidades específicas de cada *middleware*, como por exemplo APIs gráficas suportadas. Assim, cabe ao desenvolvedor organizar/desenvolver as *features* de acordo com os requisitos do *middleware* e da(s) aplicação(ões). Neste sentido, é importante destacar que o *framework* é capaz de executar adequadamente em qualquer *middleware* que suporte JavaTV, porém, as LPS/produtos construídos a partir do *framework* poderão não possuir tal portabilidade.

Alguns tópicos especiais relativos à execução das aplicações no âmbito de TV Digital são considerados e apresentados a seguir:

Canal de Retorno Existem aplicações que necessitam de canal de retorno disponível para funcionar adequadamente ao telespectador. Neste sentido, o desenvolvedor deve implementar os mecanismos de verificação da existência ou não do canal de retorno adequado à execução da aplicação para que, no caso da inexistência de canal de retorno disponível (ou no caso de sua indisponibilidade ou mau funcionamento), as variabilidades implementadas nas LPS permitam a construção de conteúdo interativo adequado ao ambiente ou notifiquem sobre a impossibilidade de execução da aplicação. Destaque-se que o *framework* disponibiliza recursos de software (apresentados na seção 4.3.3) que facilitam a construção de mecanismos para implementação de sensibilidade ao contexto, facilitando a implementação

de códigos para verificação da existência, ou não, de canal de retorno, bem como o tratamento adequado após a verificação.

Persistência Existem dois tipos de persistência a serem consideradas para o ambiente de TV Digital. A persistência necessária para manter a aplicação disponível ao telespectador e a persistência necessária ao funcionamento das aplicações, como por exemplo a necessidade de armazenamento de informações temporariamente em arquivos. O tratamento dado com relação aos problemas oriundos da dificuldade de persistir dados pode ser feito pela emissora, quando o problema for a inexistência de mecanismos de persistência da aplicação, através do controle do tamanho do software disponibilizado através do carrossel. Ou, por outro lado, o desenvolvedor pode, assim como no caso de canal de retorno, implementar mecanismos de avaliação da disponibilidade de mecanismos de persistência adequados ao funcionamento da aplicação e assim disponibilizar, caso seja possível, o conteúdo ao telespectador.

Outros Recursos Operacionais Outros recursos operacionais como, por exemplo, tipo de resolução, capacidade de processamento e mecanismos de entrada, podem/devem ser tratados, no momento da geração do produto, por mecanismos de verificação, implementados pelo desenvolvedor, utilizando, ou não, o suporte oferecido pelo *framework* para auxiliar em tal tarefa.

Para compreender a possibilidade de geração de produtos diferenciados a partir de uma LPS é necessário compreender como é feita a seleção de features através do *framework*. A próxima seção apresenta as considerações necessárias à seleção/geração de produtos de software no âmbito de TVDi.

4.1.1.2 Seleção de produtos de software no âmbito de TVDi

Dentre as principais funções do *framework* está a seleção de *features* para posterior geração do produto. Dada que uma LPS propõe, como principal característica, o reuso de software, pode-se concluir que é possível construir uma LPS que ofereça suporte, através de produtos específicos, a diversos middlewares (como Ginga-J e MHP) e terminais de recepção diferentes (como TV e *Tablets*). Neste sentido, a seleção adequada de features para construção do produto específico torna-se relevante. Como o produto deve ser selecionado no receptor, devido à caraterística de transmissão através de um

canal *broadcast*, a seleção das *features* e a consequente geração do produto devem ser feitas no receptor.

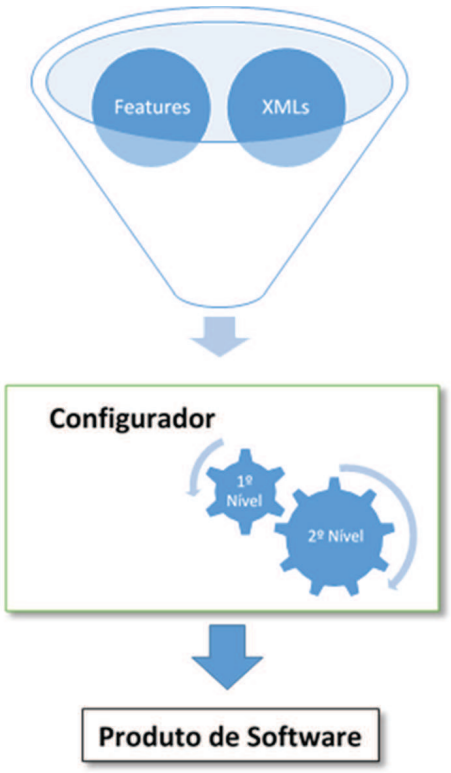

Figura 4.3: Processo de Geração do Produto de Software

O *framework* DSPL2UbiTV trabalha com a geração do produto a partir de uma SPL descrita/organizada a partir de códigos XML, ou seja, o desenvolvedor constrói as *features* (utilizando código Java) e organiza o modelo de *features* através de códigos XML armazenados em um arquivo. O modelo de *features* (códigos XML) e *features* (códigos Java) são tratados pelo Configurador de Produtos (fornecido pelo *framework*) que gera o produto de software (Figura 4.3).

A Figura 4.4 apresenta um diagrama de atividades que identifica o tratamento de seleção de *features*, executado pelo Configurador, feito em dois níveis de seleção distintos para geração do produto de software ao telespectador; tais níveis são descritos a seguir:

- o 1º (primeiro) nível para seleção de *features* possibilita identificar qual modelo de *features* será utilizado para geração do produto. Neste sentido é possível, a partir do *framework*, construir vários arquivos XML, que representem configurações genéricas para alguns produtos de software, nos quais estão descritos os modelos

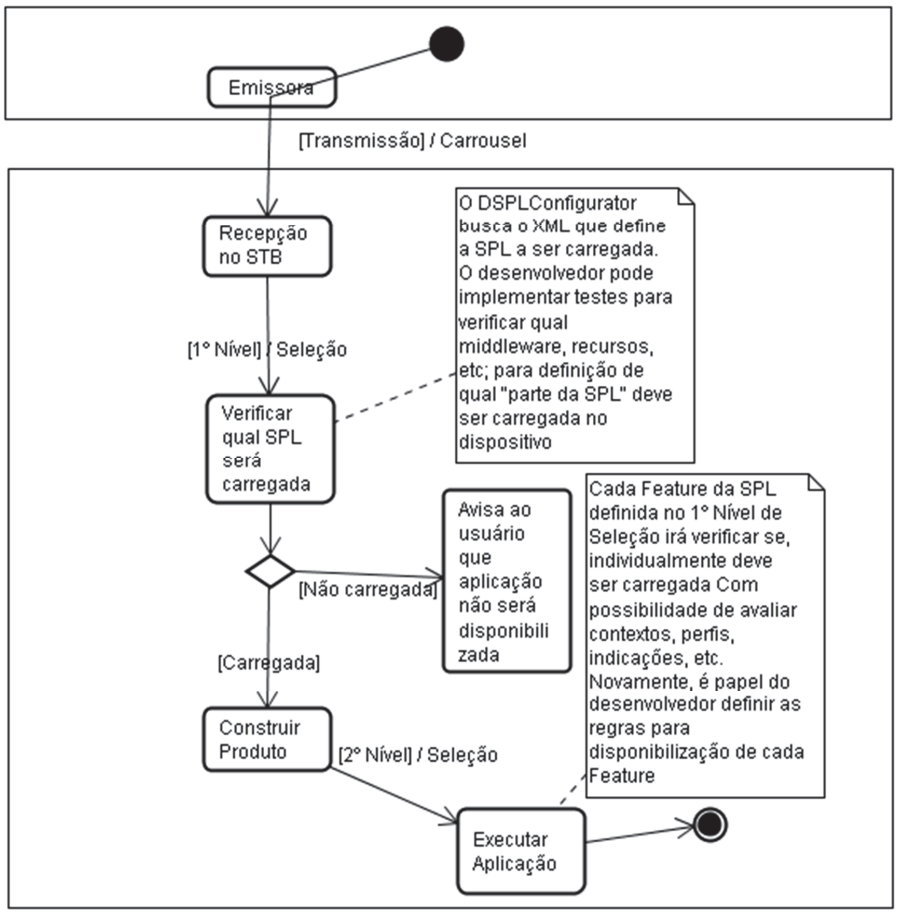

Figura 4.4: Seleção de *Features*

de *features* para ambientes específicos de execução, como por exemplo ambientes sem canal de retorno e perfis de clientes diferenciados (comumente utilizados em TV Fechada). Neste sentido, o *framework* deve avaliar, baseado em regras implementadas pelo desenvolvedor, qual modelo de *features* deve ser utilizado para iniciar a geração do produto de software (conteúdo interativo).

- o 2º (segundo) nível de seleção procura tratar da tomada de decisão a partir das variabilidades presentes na LPS selecionada no 1º Nível. Em ambos os casos, o *framework* oferece mecanismos para auxiliar o desenvolvedor no desenvolvimento dos elementos de tomada de decisão.

4.1.2 Aplicações Ubíquas e TV Digital Interativa

Considera-se que as aplicações ubíquas pressupõem um cenário no qual o usuário/telespectador possa utilizar uma aplicação que forneça um serviço que seja oferecido de forma onipre-

sente, ou seja, considera-se que o usuário, por exemplo, possa se movimentar, trocar de equipamentos e o serviço/funcionalidade deva estar sempre presente e atualizado (Figura 4.5). Obviamente, tal onipresença, no caso de TV Digital, limita-se ao espaço de cobertura disponibilizado pela emissora do conteúdo interativo.

Figura 4.5: Cenário de utilização de aplicações ubíquas

Como trata-se de uma transmissão via *broadcast*, a aplicação não migra de um terminal para o outro (como acontece em alguns casos com aplicações ubíquas), pois a aplicação está de tempos em tempos sendo atualizada pelo gerador de carrossel. Ou seja, se o usuário trocar de terminal de recepção ele receberá a aplicação novamente se sintonizar no canal que estava sintonizado no terminal utilizado anteriormente. Entretanto, no novo terminal, segundo o funcionamento padrão, as informações relativas às interações do usuário com a aplicação no antigo terminal estarão perdidas. Assim, o desafio é manter o serviço onipresente quando o usuário troca de ambiente (terminal ou canal).

Este trabalho considera que uma aplicação ubíqua é composta por dois elementos principais:

- Funcionalidades - que são oferecidas através dos códigos executáveis da aplicação;
- Informações - que representam o status de interação/operação do usuário em relação a aplicação.

No caso de TV digital, as funcionalidades sempre estarão presentes ao usuário desde que o mesmo sintonize a emissora que disponibiliza a aplicação através do gerador

de carrossel, ou possua a aplicação residente no ambiente. Assim, um problema a ser atacado é como manter as informações de interação/operação das aplicações atualizadas enquanto o usuário/telespectador troca de terminal de recepção.

Com relação ao suporte ao desenvolvimento de aplicações ubíquas, o *framework* DSPL2UbiTV oferece suporte às seguintes características, considerando as particularidades do ambiente de TV Digital:

- Onipresença de Serviços;
- Sensibilidade ao Contexto;
- Comportamento Adaptável;
- Captura de Experiências.

4.1.2.1 Onipresença de Serviços

O *framework* disponibiliza um conjunto de objetos que agem como objetos de sessão (onde é possível determinar o tempo de vida que uma informação estará disponível para aquele telespectador, através da aplicação). Neste sentido, o programador define quais informações devem estar presentes nos objetos de sessão e o *framework* busca controlar a disponibilização das informações de forma onipresente de forma local ou remota.

Manter as informações de uma aplicação de forma onipresente/local para um telespectador é relevante quando o usuário resolve navegar entre os canais disponíveis quando está utilizando uma aplicação em determinado canal. Neste caso, como cada emissora/canal controla seu próprio carrossel, as aplicações sempre serão substituídas a cada nova sintonização. Por outro lado, manter as informações de uma aplicação de forma onipresente/remota para um telespectador é relevante quanto o usuário troca de terminal.

A Figura 4.6 apresenta o fluxo das informações, proporcionado pelo *framework*, de uma aplicação ubíqua.

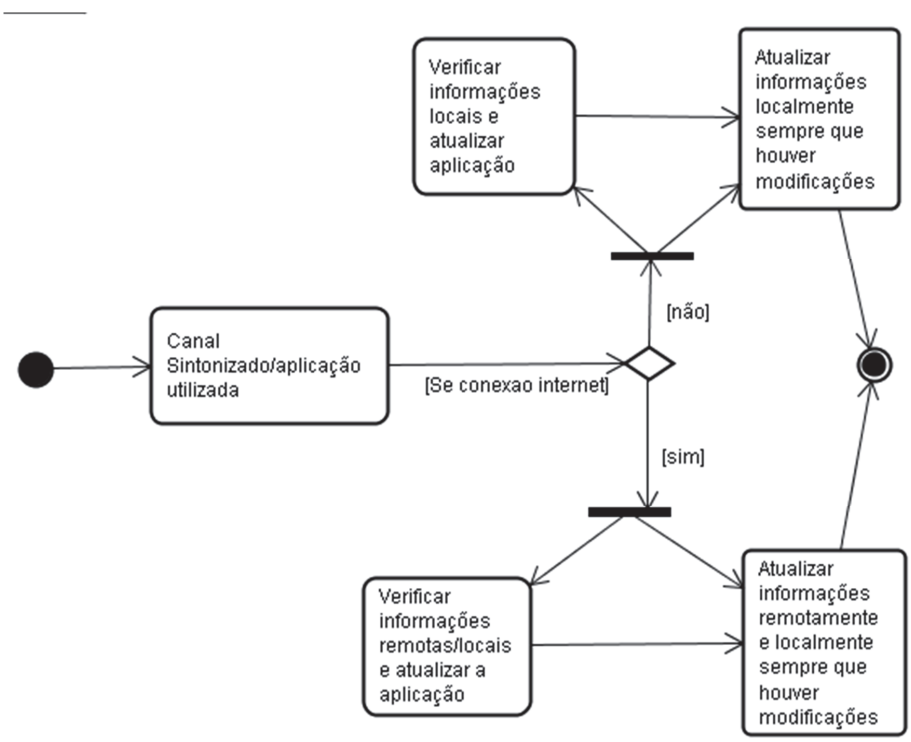

Figura 4.6: Fluxo das informações de um serviço onipresente

De acordo com a Figura 4.6 o *framework* busca armazenar as informações necessárias para a manutenção da onipresença de serviços de forma local e/ou remota. No momento em que uma informação deve ser armazenada (tal momento é representado pela atualização de uma informação na sessão, determinada pelo desenvolvedor) o *framework* verifica a possibilidade de acesso remoto a informação. Caso exista acesso remoto o *framework* armazena as informações local (se possível) e remotamente, caso contrário, procura armazenar as informações apenas localmente. Além disso, ao adicionar uma informação na sessão, o desenvolvedor deve determinar o tempo de vida de cada informação, ou seja, quanto tempo a informação deverá permanecer disponível na sessão independente de trocas de canais ou troca de equipamentos.

4.1.2.2 Sensibilidade ao Contexto

A sensibilidade ao contexto para uma aplicação ubíqua representa a capacidade do sistema coletar informações do ambiente e utilizá-las de forma a manter aplicação adequada ao usuário. Neste sentido, informações relativas ao ambiente, como por exemplo recursos computacionais, localização e informações relativas ao perfil do telespectador; podem ser utilizadas para definir as funcionalidades disponibilizadas pela aplicação.

Como as aplicações construídas a partir do *framework* são baseadas em LPSs, o *framework* permite que a construção do produto de software, ou seja, a seleção das features que constituirão o produto disponibilizado ao usuário; considere informações relativas ao contexto para decisão de utilização, ou não, de determinada *feature* para um produto. Assim, pode-se dizer que sensibilidade ao contexto é direcionada a tomada de decisão relativa a inclusão/exclusão de *features* no produto de software gerado a partir da LPS.

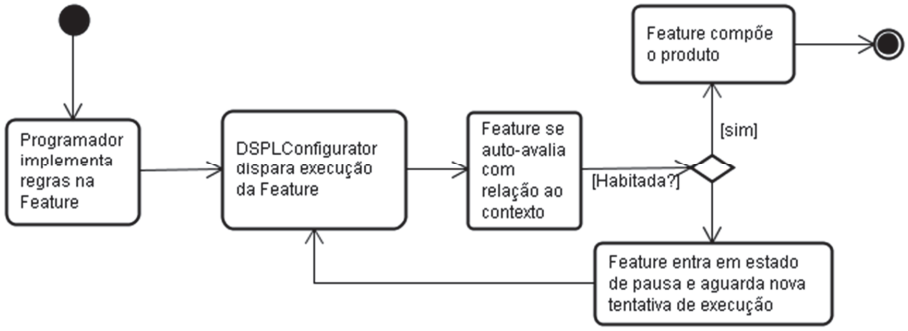

Figura 4.7: Fluxo de atividades para sensibilidade ao contexto

Neste sentido, o *framework* permite ao desenvolvedor a implementação de um método, em cada *feature*, no qual serão definidas as regras para inclusão, ou não, da *feature* no produto de software. Assim, como apresentado na Figura 4.7, o *framework*, através do DSPLConfigurator, utiliza a informação retornada pelo método implementado para geração do produto de software ciente de contexto.

Do ponto de vista da TV Digital interativa a sensibilidade ao contexto é uma característica extremamente relevante para disponibilização dos produtos adequados aos telespectadores em seus ambientes de recepção específicos.

Por outro lado, em algumas situações o contexto pode mudar (por exemplo, em função da mobilidade do usuário) e a *feature*, outrora selecionada, não é capaz de executar adequadamente (como por exemplo situações na qual a *feature* necessita de canal de retorno para executar adequadamente), necessitando uma reavaliação constante do produto disponibilizado. Neste sentido, mecanismos de auto-adaptação da aplicação foram disponibilizados pelo *framework* e descritos a seguir.

4.1.2.3 Comportamento Adaptável

O comportamento adaptável da aplicação representa a necessidade de constante avaliação se as funcionalidades disponibilizadas ao telespectador estão adequadas as necessidades. Neste caso, esperasse que a partir de mudanças de contexto ou preferência do usuário, as aplicações sejam capazes de adaptar seu comportamento funcional para melhor atender ao usuário da aplicação.

Tal característica, do ponto de vista de TV Digital Interativa, é relevante, principalmente, quando os telespectadores se utilizam de equipamentos móveis como smartphones, *tablets* e GPSs para terem acesso ao conteúdo televisivo. Em um cenário de mobilidade a mudança de contexto pode ser constante, principalmente no que diz respeito à existência/qualidade do canal de retorno. Além disso, fatores como localização, entre outros, podem influenciar no perfil de acesso/interatividade do telespectador.

Neste sentido, o *framework* disponibiliza mecanismos de monitoração, que uma vez implementados pelo desenvolvedor, são capazes de monitorar itens relativos ao perfil pessoal do telespectador e contextos. Assim, de acordo com a Figura 4.8, o desenvolvedor deve implementar os monitores de contexto/perfil e vinculá-los às *features* adequadamente. Uma vez vinculados às *features*, toda vez que forem percebidas trocas de contexto, os monitores notificam as *features* vinculadas a necessidade de reavaliação de execução, para adaptação do produto disponibilizado ao telespectador.

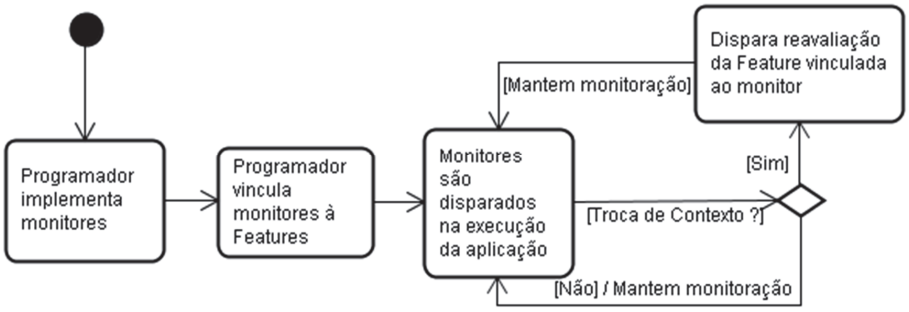

Figura 4.8: Fluxo de atividades para comportamento adaptável das aplicações

4.1.2.4 Captura de Experiências

A captura de experiências é uma característica de software ubíquo que representa a necessidade da própria aplicação aprender com as informações de utilização/operação

do software para futuramente utilizar tais informações em prol do usuário da aplicação. Neste sentido, como exemplo, pode-se citar a possibilidade de observar quais funcionalidades são mais ou menos utilizadas em determinados contexto e, através de mecanismos de sensibilidade ao contexto/auto-adaptação, disponibilizar somente as funcionalidades mais utilizadas, possibilitando, por exemplo, a liberação de recursos de memória e processamento.

Do ponto de vista da TVDi, a captura de experiências se faz interessante, por exemplo, uma vez que existem perfis de interatividade entre os telespectadores. Existem telespectadores que não se sentem confortáveis/capazes de interagir com o conteúdo, preferindo somente assistir. Por outro lado, existem telespectadores que realmente gostam de interagir com o conteúdo sempre que possível. Em ambos os casos citados a ausência ou existência de conteúdo interativo pode tornar o conteúdo como um todo desinteressante ao telespectador.

O *framework* DSPL2UBiTV procura auxiliar o desenvolvedor disponibilizando alguns mecanismos para coleta e tomada de decisão relativas a captura de experiências. Os principais mecanismos são dois métodos, controlados pelo *framework* que, uma vez implementados pelo desenvolvedor, são acionados a cada momento que um determinada *Feature* é disponibilizada/utilizada pelo usuário. Neste sentido, cada vez que a *feature* é executada é possível identificar, por exemplo, em qual contexto ou tipo de conteúdo ela foi acionada, permitindo o aprendizado sobre o uso da funcionalidade. Assim, o *framework* permite a implementação de um método que dispare as verificações/armazenamento de informações para aprendizado das experiências e outro método, utilizado no momento da ativação da seleção da *Feature*, que permite implementar a avaliação sobre a possibilidade de disponibilizar a *Feature* para o telespectador (Figura 4.9).

Além disso, o *framework* disponibiliza uma API que implementa um modelo multidimensional para armazenamento de informações de aprendizado, bem como alguns recursos para busca de informação para tomada de decisão. Entretanto, o desenvolvedor pode se utilizar da tecnologia que entender necessária para definir o perfil de aprendizado do telespectador/aplicação, bem como outras possibilidades de tomada de decisão.

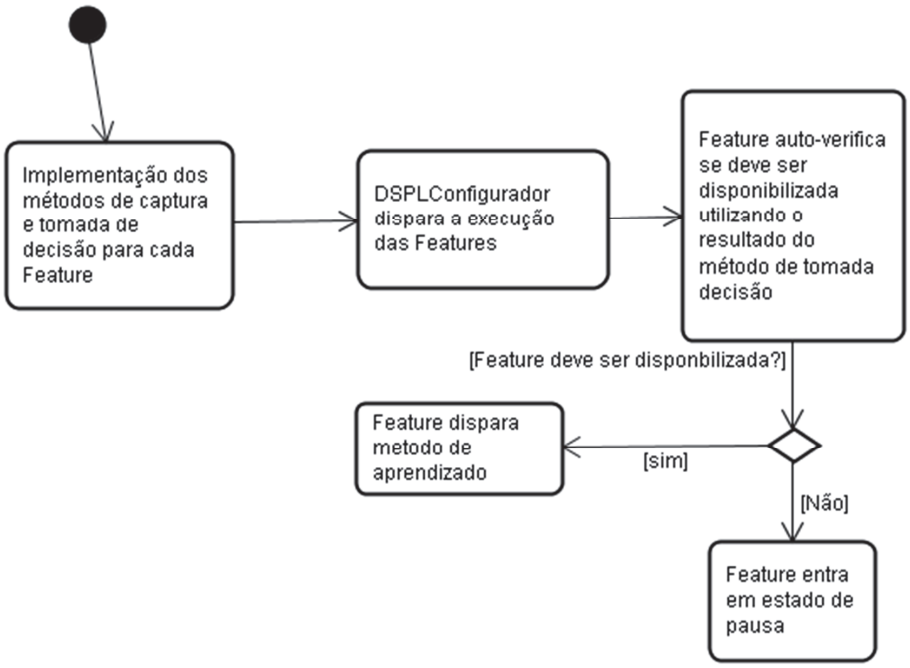

Figura 4.9: Atividades para Captura de Experiências

4.2 MODELO DE *FEATURES* E ARQUITETURA GERAL

A *framework* LPSD para desenvolvimento de softwares ubíquos para TV Digital interativa (TVDi) propõe-se a servir como uma ferramenta de apoio ao desenvolvimento de linhas de produtos de software (LPS), bem como dar suporte a projetos no domínio de aplicações ubíquas para TVDi.

Para identificar as principais características do *framework* é apresentado na Figura 4.10 o modelo geral de *features* do *framework*. Pode-se observar no modelo apresentado 4 (quatro) *features* principais do *framework*, a seguir descritas :

SPLiDTV: *feature* mandatória e abstrata, que representa uma SPL específica para iTVD. Tal SPL para iTVD é construída pelos desenvolvedores com o apoio do *framework* DSPL2UbiTV. A SPLiDTV possui sua execução controlada pela *feature* denominada **DSPLConfigurator**.

UbiquitousFeatures : *feature* opcional que representa a possibilidade de agregar características de ubiquidade às SPLiDTV desenvolvidas através do *framework*. Uma vez selecionadas, as *features* relativas às características de softwares ubíquos, os controles implementados, para as devidas características, serão considerados durante a execução do produto de software gerado;

DSPLConfigurator : *feature* mandatória que representa a implementação do configurador dinâmico de produtos de software gerados a partir das SPLiDTV desenvolvidas sobre o *framework*;

RecommenderFeatures : *feature* opcional que representa a possibilidade do desenvolvedor identificar/implementar quais as técnicas de recomendação utilizados para determinar a seleção de *features* (optativas), em tempo de execução, no caso de indeterminação prévia quanto ao tratamento dado as *features* opcionais presentes na SPLiDTV.

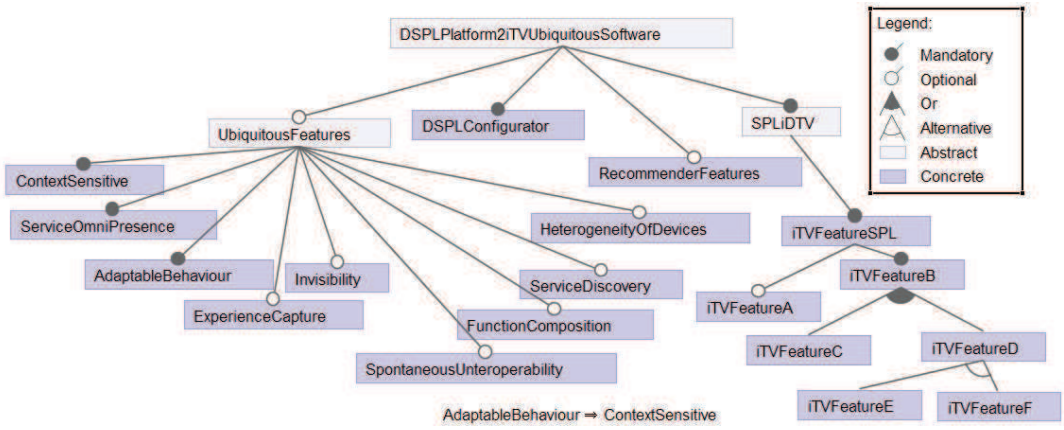

Figura 4.10: Modelo Geral de *Features* do *framework* DSPL

É importante ressaltar que o desenvolvedor, além de implementar a SPLiDTV, pode estender as funcionalidades do *framework* relativas ao suporte às características de softwares ubíquos e técnicas de recomendação utilizadas. Ainda, com relação à seleção de *features* do *framework*, cabe destacar que ela não ocorre de forma dinâmica, ou seja, é determinada pelo desenvolvedor. Apenas a SPLiDTV possui o tratamento de seleção de *features* de forma dinâmica.

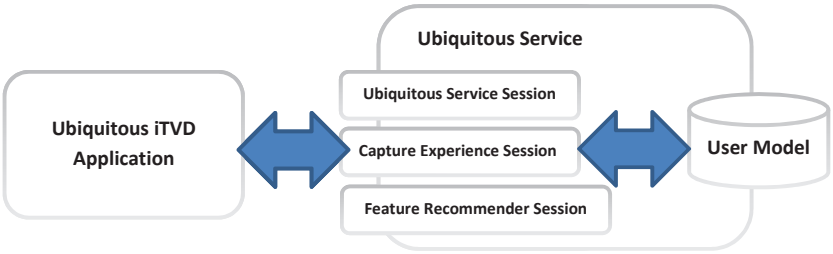

Figura 4.11: Arquitetura Geral DSPL2UbiTV

Além disso, faz-se necessário apresentar a arquitetura geral do *framework*. Tal arquitetura (Figura 4.11) é representada por dois blocos principais:

Ubiquitous Services : Trata-se de um conjunto de funcionalidades responsáveis por manter alguns serviços ubíquos básicos podendo ser extendidos de forma centralizada e remota. Os serviços ubíquos disponibilizados pelo *framework* são apresentados na tabela 4.1:

Serviços Ubíquos	Descrição
Ubiquitous Service Session	Trata-se de uma sessão capaz de manter disponível qualquer informação (como variáveis de controle, objetos, listas, etc) necessária ao software para manutenção da onipresença do serviço de interatividade.
Capture Experience Session	Trata-se de uma sessão capaz de manter disponível as informações relativas ao aprendizado/conhecimento das experiências de interação do usuário com aplicação ubíqua, adicionadas das informações sobre o contexto da interação.
Feature Recommender Session	Trata-se de uma sessão capaz de recomendar a execução de uma *feature* dado seu contexto de execução.

Tabela 4.1: Serviços Ubíquos Básicos disponibilizados na DSPL2UbiTV

Além disso, pode-se identificar as principais informações contidas no Modelo de Usuário (Figura 4.12):

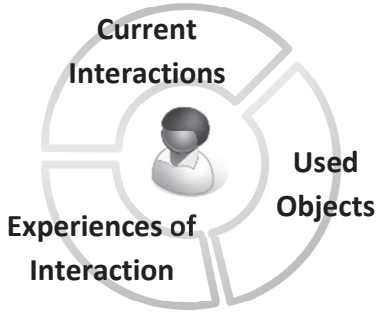

Figura 4.12: Modelo de Informações do Usuário

- Informações relativas ao estado atual de interação com a aplicação (mídias interativas que já foram disponibilizas e descartadas pelo usuário, caso o usuário mude de contexto);
- Informações relativas a variáveis utilizadas pela aplicação (produtos adicionados em um carrinho virtual de compras);

- Informações sobre a interação do usuário com as mídias interativas (*features*), bem como as informações de contexto nas quais foram utilizadas ou não (conhecimento relativo à utilização ou descarte de mídias interativas em ambientes operacionais diferentes).

Ubiquitous iTVD Application : Trata-se do conjunto de funcionalidades responsável por controlar dinâmicamente a execução, em cada ambiente operacional, da SPLiDTV construída pelo desenvolvedor. O bloco **Ubiquitous iTVD Application** interage com o bloco remoto **Ubiquitous Service** com o propósito de buscar informações que permitam manter a onipresença do serviço de interatividade. Entretanto, por depender de conexões com a rede de comunicação para acessar os serviços, o *framework* possui mecanismos para melhor construir a aplicação para o usuário no caso de inexistência de conexão de rede. Alguns mecanismos utilizados serão destacados na seção (4.2) que trata do processo de desenvolvimento FOSD do *framework*.

4.3 PROCESSO DE DESENVOLVIMENTO (FOSD) DA DSPL2UbiTV

O Desenvolvimento de Software Orientado a ***Features*** (FOSD), no qual o *framework* DSPL2UBiTV está baseada, possui um processo de desenvolvimento que envolve: (1) Análise de Domínio, (2) Especificação e Projeto de Domínio, (3) Implementação de Domínio e (4) Configuração e Geração de Produtos de Software.

Assim, as próximas seções apresentarão o desenvolvimento, sob um aspecto prático, do *framework* DSPL2UbiTV em cada uma das fases do processo FOSD.

4.3.1 Análise de Domínio

Na Fase de Análise de Domínio, onde o Modelo de *Features* de uma LPS é definido [10], o *framework* DSPL2UbiTV disponibiliza um conjunto de TAGs (Tabela 4.2) para criação de um arquivo XML no qual o Modelo de *Features* estará representado.

A partir do Modelo de *Features* representado através das TAGs específicas, a *feature* mandatória **DSPLConfigurator** tratará da configuração, geração e controle da SPLiDTV.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<featuremodel>
<features>
   <mainfeature>
     <name>iTVFeatureSPL</name>
     <relationship>MANDATORY</relationship>
   </mainfeature>
   <feature>
     <name>iTVFeatureA</name>
     <relationship>OPTIONAL</relationship>
<parentFeature>iTVFearureSPL</parentFeatures>
   </feature>
   <feature>
     <name>iTVFeatureB</name>
     <relationship>MANDATORY</relationship>
<parentFeature>iTVFeatureSPL</parentFeatures>
   </feature>
   <feature>
     <name>iTVFeatureC</name>
     <relationship>OR</relationship>
        <priority cardinality="1">true</priority>
<parentFeature>iTVFeatureB</parentFeatures>
        <constraints>
            <require>
<nameFeature>iTVFeatureA</nameFeature>
               </require>
        </constraints>
   </feature>
```

```
   <feature>
     <name>iTVFeatureD</name>
     <relationship>OR</relationship>
        <priority cardinality="1">false</priority>
     <parentFeature>iTVFeatureB</parentFeatures>
        <constraints>
            <exclude>
<nameFeature>iTVFeatureA</nameFeature>
            </exclude>
        </constraints>
   </feature>
     <feature>
     <name>iTVFeatureE</name>
     <relationship>ALTERNATIVE</relationship>
        <priority cardinality="1">false</priority>
     <parentFeature>iTVFeatureD</parentFeatures>
   </feature>
     <feature>
     <name>iTVFeatureF</name>
     <relationship>ALTERNATIVE</relationship>
        <priority cardinality="1">true</priority>
     <parentFeature>iTVFeatureD</parentFeatures>
   </feature>
 </features>
</featuremodel>
```

Figura 4.13: Modelo de *Features* em XML

A Figura 4.13 apresenta a representação, em XML, da SPLiDTV apresentada na Figura 4.10.

Assim, uma vez que a estrutura de Modelo de *Features* é definida através de metadados, as mudanças estruturais do Modelo de *Features* tais como relacionamentos e restrições, podem ser modificados e estendidos com facilidade. Além disso, o *framework* permite a definição de sub-modelos de *features* que são utilizados dado o contexto de execução, ou seja, é possível definir modelos de *features* específicos para ambientes de execução distintos, como: HDTV, DTV, smartphones, computadores pessoais, tablets, etc.

TAGS	Descrição
featuremodel	TAG raiz, identifica um Modelo de *Features*.
features	Vinculada à TAG **featuremodel**, identifica o conjunto de *features* do Modelo de *Features*.
mainfeature	Vinculada à TAG **features**, identifica a iTVFeature raiz do Modelo de *Features*.
feature	Vinculada à TAG **features**, identifica uma iTVFeature no Modelo de *Features*.
relationship	Vinculada à TAG **feature**, identifica o relacionamento entre a *feature* e sua *feature* pai (*parentFeature*). Os relacionamentos podem ser MANDATORY, OPTIONAL, ALTERNATIVE e OR.
priority	Vinculada à TAG **feature**, identifica nos casos das *features* optativas (OPTIONAL, ALTERNATIVE e OR) se a *feature* deve ser executada, no caso da ausência de um mecanismo de decisão implementado. A TAG possui uma propriedade denominada ***cardinality*** utilizada quando as *features* optativas (ALTERNATIVE e OR) precisam identificar mais de uma *feature* a ser selecionadas para os relacionamentos.
parentFeature	Vinculada à TAG **feature**, identifica a TAG pai da *feature*.
constraints	Vinculada à TAG **feature**, trata-se da TAG utilizada para identificar restrições ligadas a uma *feature*.
require	Vinculada à TAG **constraints**, e identifica quais *features* serão requeridas pela *feature* que possui a restrição.
exclude	Vinculada à TAG **constraints**, e identifica quais *features* serão excluídas pela *feature* que possui a restrição.
nameFeature	Vincula às TAGs **require** e **exlcude**, identifica o nome da *feature* na qual a restrição (*constraints*) será aplicada.

Tabela 4.2: TAGs utilizadas no Modelo de *Features* no *Framework* DPSL2UbiTV

4.3.2 Especificação e Projeto de Domínio

No contexto do FOSD, na fase de especificação e projeto de domínio as propriedades estruturais e comportamentais essenciais dos recursos envolvidos são especificadas usando uma especificação formal ou informal e / ou uma linguagem de modelagem. Neste trabalho utilizou-se a UML como linguagem de modelagem. Assim, a seguir serão apresentadas as modelagens, utilizando-se diagramas de classe, de alguns dos

principais componentes de software do *framework*.

4.3.2.1 Projeto da Classe *ITVFeature*

A classe *ITVFeature* é a classe mais importante de toda o *Framework* DSPL2UbiTV. Trata-se da base para implementação dos conteúdos televisivos interativos, bem como o elemento que representa as *features* de uma LPS em um desenvolvimento FOSD.

A classe *ITVFeature* (Figura 4.14) implementa a interface Xlet da API javaDTV (API suportada nos principais *middlewares* disponíveis mundialmente, por exemplo: MHP e GINGA). Sendo assim, uma *ITVFeature* possui um ciclo de vida similar a de um Xlet, disponbilizando métodos para inicialização, execução, pausa e destruição de *ITVFeatures*. Além disso, a classe *ITVFeature*, ao implementar a classe *Observer*, age como um observador aguardando notificações relativas às trocas de contextos.

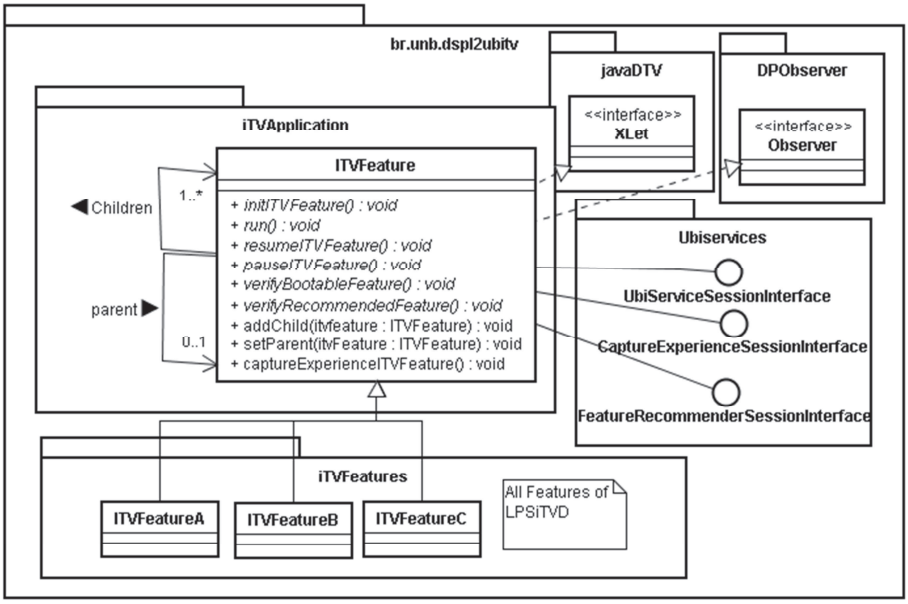

Figura 4.14: Diagrama de Classes - ITVFeature

Quanto ao suporte à construção de modelos de *features*, a Classe *ITVFeature* permite a identificação de classes filhas e também de uma classe pai. Sendo assim, é possível construir um modelo de *features* instanciando e inter-relacionando instâncias da classe *ITVFeature*.

ITVFeature também possui objetos agregados que são a interface para os serviços ubíquos: UbiServiceSession, CaptureExperienceSession e RecommenderFeatureSession.

Além disso, possibilita a implementação por parte do desenvolvedor de software, dos métodos **verifyBootableFeature()**,utilizado para avaliar a possibilidade de execução da *ITVFeature*, e **verifyRecommendedFeature()**,utilizado para avaliar se a execução da *ITVFeature* é recomendável dentro do produto de software configurado.

Finalmente, o desenvolvedor de software deve implementar a herança (extensão) da classe *ITVFeature* para todas as *features* do modelo de *features*.

4.3.2.2 Projeto de Classes de Monitoração

Para que as aplicações desenvolvidas, a partir do *framework* DSPL2UbiTV, sejam capazes de se auto-adaptar aos diferentes ambientes (contextos) de execução possíveis faz-se necessário monitorar as trocas de contexto e avaliar a necessidade de adaptação da aplicação. A Figura 4.15 apresenta um diagrama de classes contendo classes relevantes para execução da tarefa de monitoração.

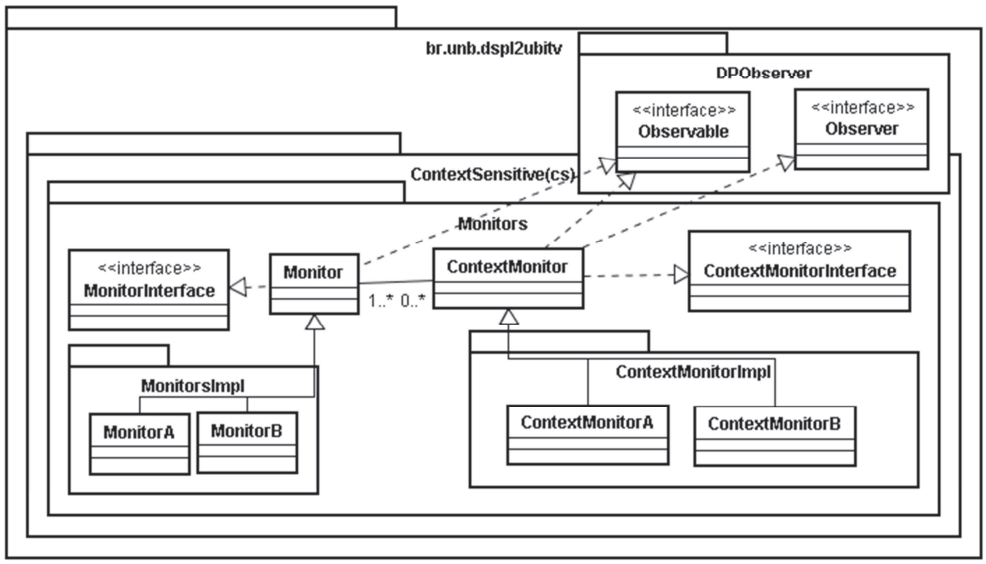

Figura 4.15: Diagrama de Classes - Sensibilidade ao Contexto

Para monitorar as variações de contexto faz-se necessário inicialmente, monitorar os valores presentes nas variáveis contextuais, como por exemplo, identificar qual tecnologia de rede sem fio (WiFi, 3G, 4G, etc) está sendo utilizada no dispositivo em um determinado momento. Assim, o desenvolvedor de software deve estender a classe *Monitor* com a finalidade de avaliar o status de quantas variáveis contextuais forem necessárias. É recomendável ainda criar uma classe de monitoração para cada variável de contexto considerada.

No entanto, várias variáveis contextuais podem ser necessárias, bem como a combinação de seus status, para determinar uma mudança de contexto. Existe a troca dos períodos de tempo matutino para vespertino, por exemplo. No entanto, o usuário permaneceu no local de trabalho. Nesse caso, não se consideraria uma troca de contexto significativa para gerar uma adaptação na aplicação. Assim, faz-se necessária, a partir do desenvolvedor de software, a extensão da classe *ContextMonitor* para cada contexto considerado. Nas classes que herdam de *ContextMonitor* o desenvolvedor deve avaliar, baseado em um conjunto de variáveis contextuais, a troca efetiva de contexto para uma reação da aplicação.

Assim, a classe *Monitor* implementa a interface *Observable*, o que faz com que suas instâncias de objetos possam ser observadas com relação à troca dos valores da variáveis contextuais monitoradas. Na prática, as classes *Observable* notificam as classes *Observer* quando da identificação de alguma mudança. A classe *ContextMonitor*, por sua vez, implementa a interface *Observer*, ou seja, age como classe observadora para as mudanças dos status das variáveis contextuais.

Por fim, a classe *ContextMonitor* também implementa a interface *Observable*, com o intuito de poder notificar a aplicação de uma mudança significativa no contexto. A aplicação, nesse caso, é representada pela classe *ITVFeature* (que implementa a interface *Observer*). Sendo assim, quando de uma mudança significativa de contexto é identificada através das classes de monitoração, as classes que estendem *ITVFeature* são notificadas para que auto-avaliem suas condições de execução.

4.3.2.3 Projeto do Modelo Multidimensional de Contextos

O modelo multidimensional de contextos prevê a contabilização de informações dado um conjunto de dimensões contextuais inter-relacionadas. Assim, se o usuário estiver em um determinado local, período do dia, dia da semana, utilizando um determinado dispositivo, qual seria a probabilidade de utilizar determinado contéudo interativo da TV ? Ou seja, o local, dia da semana, período do dia e tipo de dispositivo são tratados como dimensões no modelo multidimensional.

Por outro lado, existem 7 (sete) dias na semana, 3 (três) períodos no dia, vários locais e vários tipos de dispositivos. Nesse caso, cada dia da semana, por exemplo, é considerado uma variável de contexto. Para persistir/contabilizar tais informações no *framework*

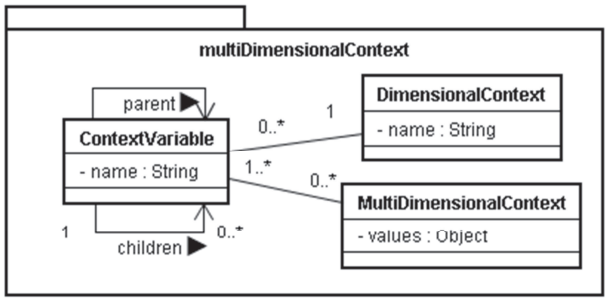

Figura 4.16: Diagrama de Classes - Contexto MultiDimensional

DSPL2UbiTV faz-se necessário representar tais informações através de objetos. A Figura (4.16) apresenta um diagrama de classes simplificado para a representação das classes que implementam o modelo multidimensional. Tais classes são detalhadas a seguir:

DimensionalContext - Esta classe representa a informação de dimensão de contexto. Possui um atributo chamado *name* que pode receber valores como: "período do dia", "local", "ITVFeature", etc;

ContextVariable - o objeto *ContextVariable* possui informações sobre qual dimensão pertence (período do dia, por exemplo), *name* (matutino, por exemplo) e informações hieráquicas (os períodos do dia são vinculados aos dias da semana, que por sua vez são vinculados aos meses, por exemplo);

MultiDimensionalContext - o objeto instanciado, a partir da classe *MultiDimensionalContext*, possui um atributo denominado *values* que possui as informações relativas à contabilização de informações sobre o contexto multidimensional. Além disso, possui a possibilidade de relação com uma ou várias instâncias do objeto *ContextVariable*, para de fato identificar quantas dimensões possui.

Assim, as classes do modelo multidimensional são utilizadas para implementação dos Serviços Ubíquos, com o intuito de aprendizado de informações, bem como recomendações. Neste sentido, as técnicas de predição, agregação hierárquica e classificação apresentadas na seção 2.4 podem ser utilizadas para obter os resultados relativos à tomada de decisão.

4.3.2.4 Projeto de Classes de Sessão e Serviços Ubíquos

Os serviços ubíquos são baseados em 4 (quatro) classes/interfaces principais (Figura 4.17) :

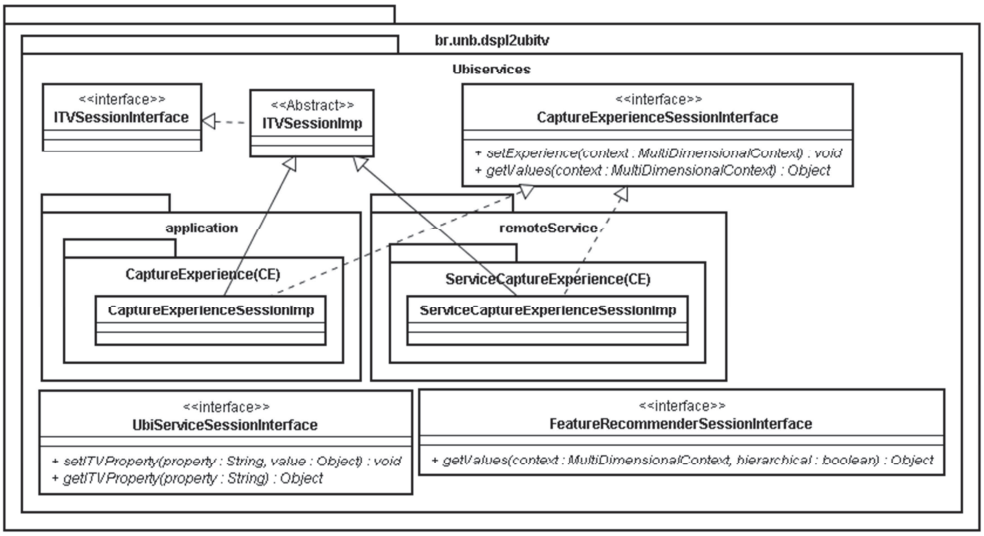

Figura 4.17: Diagrama de Classes - Captura de Experiências

ITVSessionImp - Classe responsável por permitir o controle de sessão para os serviços ubíquos. Deve permitir a utilização da sessão durante o ciclo de vida da aplicação, mesmo durante a troca de dispositivo, se ocorrer em um período pré-determinado.

UbiServiceSessionInterface - Possui os métodos (**setProperty()** e **getProperty()**) que permitem armazenar qualquer objeto em uma sessão controlada em um ambiente remoto. Assim, torna-se possível inicializar uma aplicação em outro dispositivo e recuperar objetos utilizados em interações realizadas em dispositivos utilizados recentemente.

CaptureExperienceSessionInterface - Tal interface disponibiliza o método **setExperience()** utilizado para, uma vez identificado o contexto, armazenar experiências de interação, e disponibiliza também o método **getValues()** utilizado para buscar os valores contabilizados até o momento.

FeatureRecommenderSessionInterface A interface *FeatureRecommenderSessionInterface* disponibiliza um método para buscar objetos com informações relativas à recomendação dado um contexto, disponibiliza também o método *getValues()*, que recebe como parâmetros o contexto multidimensional e uma informação relativa à consideração de hierarquia contextual para geração das recomendações.

É importante destacar que para a implementação dos serviços ubíquos, as classes que implementam as interfaces de sessão também devem estender a classe ITVSessionImp, pois todo controle de sessões, como tempo de vida, criação e invalidação, autenticação, está implementado nas mesmas.

4.3.3 Implementação de Domínio

Na fase de Implementação de Domínio os códigos são implementados. Neste sentido esta seção apresentará as principais classes do *framework*, bem como quais classes podem/devem ser estendidas e agregadas aos códigos. Na prática serão apresentados os passos necessários para implementação desde uma SPLiDTV, até um conjunto de códigos capaz de suportar as principais características de projetos ubíquos consideradas neste trabalho.

4.3.3.1 Primeiro Passo - Definição das *features* da SPLiDTV

O primeiro passo para se utilizar o *framework* é desenvolver as *features* que comporão a SPLiDTV. As *features* são desenvolvidas de forma que as decisões relativas à organização, relacionamentos e restrições do modelo de *features* sejam determinados através de outros mecanismos (Figura 4.13). Sendo assim, para desenvolver uma *feature* aderente ao *framework*, é necessário estender a classe ITVFeature (Tabela 4.3).

```
1 public abstract class ITVFeature implements Xlet, Observer, ITVFeatureInterface,
ActionListener {
...
2 public static UbiServiceSessionInterface ubiTVsession = new UbiServiceSessionImp();
3 public static CaptureExperienceSessionInterface cesession = new
CaptureExperienceSessionImp();
4 public static FeatureRecommenderSessionInterface recommendersession = new
FeatureRecommenderSessionImp();
...
5 public void initITVFeature(){ ...}
6 public void run(){ ...  }
7 public void pauseITVFeature() { ...  }
8 public void resumeITVFeature() { ...  }
...
}
```

Tabela 4.3: ITVFeature Class

A classe ITVFeature possui instâncias de objetos que permitem a utilização das sessões de serviços ubíquos, de aprendizado e de recomendação (Linhas 2, 3 e 4 da tabela 4.3).

Ressalta-se ainda que os objetos de sessão são estáticos, ou seja, são compartilhados por todas as *features* que estendem a classe *ITVFeature*.

Além disso, a classe ITVFeature implementa os métodos necessários relativos à classe Xlet (classe que representa uma mídia interativa no pacote JavaDTV) bem como disponibiliza métodos a serem sobrescritos por suas classes filhas. Os principais métodos que devem ser sobrescritos a partir da classe *ITVFeature* são, com base da Tabela 4.3:

initITVFeature() Na Linha 5, o método **initITVFeature()** deve ser utilizado para inicialização de objetos internos à *feature*;

run() Na linha 6, o método **run()** deve ser utilizado para implementação da interface gráfica e operacional da *feature*. O método **run()** é o único método que deve, obrigatoriamente, ser sobrescrito;

pauseITVFeature() Na linha 7, o método **pauseITVFeature()** deve ser utilizado para determinar ações executadas caso uma *feature* passe ao estado de pausada. Tal situação pode ocorrer em tempo de execução, quando na mudança de contexto existir algum impedimento para execução da *feature*;

resumeITVFeature() Na linha 8, o método **resumeITVFeature()** deve ser utilizado para execução das ações necessárias quando, dada uma mudança contexto uma *feature* seja reinicializada.

Na prática, apenas a *feature* identificada como ***MainFeature*** (raiz da LPS) deve estender a classe ITVFeature. As demais *features* da SPLiDTV devem estender a ***MainFeature***. A ***MainFeature*** (Tabela 4.4) é responsável pela disponibilização do *container* gráfico utilizado na SPLiDTV (linhas 1,3 e 6) e pela inicialização da execução da SPLiDTV (linha 7).

Na Tabela 4.5 pode-se observar um exemplo de implementação de uma *feature* utilizando o *framework* DSPL2UbiTV. Observa-se na linha 1 que a classe *ITVFeatureA* estende a ***MainFeature***. Além disso, pode-se observar a implementação dos métodos **resumeITVFeature()** e **pauseITVFeature()**, onde a implementação de tais métodos é importante para que a reconfiguração dinâmica da LPS seja adequada, bem como o suporte à característica de projetos ubíquos denominada **Comportamento Adaptável**.

```
1 import com.sun.dtv.lwuit.Form;
2 public class ITVFeatureSPL extends ITVFeature {
3 protected static Form form = new Form();
4 @Overide
5 public void run() {
6 form.show();
7 super.startITVFeature(); }
8 }
```

Tabela 4.4: MainFeature Class

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 Button button;
3 public void run() {
4 button = new Button("Old Text");
5 try{ button.setIcon(...);
6 button.setSize(new Dimension(60,60));
7 }catch(Exception e){}
8 button.addActionListener(this);
9 form.addComponent(BorderLayout.WEST, button); }
10 public void resumeITVFeature() { form.repaint(); }
11 public void pauseITVFeature(){
12 form.removeComponent(button); form.repaint();
...  } }
```

Tabela 4.5: ITVFeatureA Class

Enfim, uma vez definidas e implementadas todas as *features* deve-se definir o modelo de *features* segundo o exemplo apresentado na Figura (4.13).

4.3.3.2 Segundo Passo - Implementando Sensibilidade ao Contexto

A implementação da sensibilidade ao contexto deve considerar dois conjuntos de classes:

- Classes utilizadas para implementação da verificação do status das variáveis de contexto. Por exemplo, classes que implementam a interface *ContextTestInterface* e *RulesInterface* (utilizada para implementar as regras de identificação do status);

- Classes utilizadas para manter a verificação do status enquanto as *features* estão executando. Por exemplo, a classe *Monitor* e a classe *ContextMonitor*, que são as classes que devem ser estendidas pelos desenvolvedores.

Os códigos que estendem as classes *Monitor* e *ContextMonitor* devem utilizar, para fins de verificação do status atual de contexto, as implementações das interfaces *ContextTestInterface* e *RulesInterface*.

A classe *Monitor* implementa as interfaces *MonitorInterface* e *Observable*. Assim, as classes que representam monitores verificam temporalmente a mudança de status de variáveis de contexto e notificam as classes observadoras quando da mudança de status. Já a classe *ContextMonitor* implementa as interfaces *ContextMonitorInterface* e *Observer*, ou seja, as classes que herdam de *ContextMonitor* devem observar as mudanças de status ocorridas e identificadas pelas classes que herdam da classe *Monitor*.

Além disso, a classe *ContextMonitor* verifica se as mudanças de status nos monitores observados são significativas a ponto de representarem uma troca de contexto. Caso seja identificada a troca de contexto a classe *ContextMonitor* notifica, devidamente, as classes que estendem ITVFeature. Sendo assim, a classe *ContextMonitor* também implementa a interface *Observable* e a *ITVFeature* implementa a interface *Observer*.

Na prática os desenvolvedores devem estender a classe *Monitor* para desenvolver Monitores de variáves contextuais (por exemplo, períodos do dia: matutino, vespertino e noturno), bem como implementar dois métodos obrigatoriamente. Tratam-se dos métodos **runMonitor()** e **isChangeStatus()**, onde o método **runMonitor()** deve implementar a verificação das variáveis de contexto. A identificação de trocas de status é implementada através do método **isChangeStatus()** (Tabela 4.6).

```
1 public class TimeMonitor extends Monitor{
2 ...
3 public boolean isChangeStatus()
4 { ... }
5 public void runMonitor()
6 { ... }
7}
```

Tabela 4.6: Exemplo de extensão da classe Monitor

Por outro lado, ao estender a classe *ContextMonitor* os desenvolvedores devem implementar apenas um método. Trata-se do método **verifyChangeContext()** responsável por verificar a efetiva troca de contexto para posterior notificiação às extensões de *ITV-Feature* (Tabela 4.7).

```
1 public class ContextMonitorLocalTime extends ContextMonitor{
2 ...
3 public boolean verifyChangeContext()
4 { ... }
5}
```

Tabela 4.7: Exemplo de extensão da classe ContextMonitor

Uma vez identificada uma troca de contexto a extensão de *ITVFeature* notificada entra em um processo de auto-reavaliação do status de execução. Podendo, assim, tornar-se ativa ou inativa.

A organização de quais monitores de variáveis são observados por quais monitores de contexto, bem como a definição de qual *feature* observa um determinado monitor de contexto, é definida através de um metadado (Figura 4.18). Além disso, nesse metadado é definido, através da TAG $< time >$, o tempo de intervalo para verificação da troca de contexto (em milisegundos).

```
 1 public abstract class ITVFeature implements Xlet, Observer, ITVFeatureInterface,
ActionListener {
2 ...
3 public void verifyBootableFeature()
4 { ... this.setBootableFeature(true); ... }
5...
6 }
```

Tabela 4.8: ITVFeature Class - Implementação do Método **verifyBootableFeature()**

Enfim, é necessário destacar a necessidade de implementação do método **verifyBootableFeature()** (Tabela 4.8). Tal método deve identificar, através de verificações em variáveis contextuais, se existe condição (recursos computacionais adequados) de execução da *feature* inicializada. Após a verificação, deve-se setar o método **setBootableFeature()** como verdadeiro ou falso.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<CS>
<monitors>
   <monitor>
     <class>TimetMonitor.class</class>
     <time>12343414</time>
      <observers>
        <observer>ContextMonitorLocalTime.class</observer>
     </observers>
  </monitor>
  <monitor>
     <class>LocalMonitor.class</class>
      <time>12343414</time>
      <observers>
        <observer>ContextMonitorLocalTime.class</observer>
     </observers>
  </monitor>
<monitors>
<contextMonitors>
 <contextMonitor>
  <contextName>LocalTime</contextName>
  <class>ContextMonitorLocalTime.class</class>
           <observers>
    <observer>ITVFeatureB.class</observer>
  </observers>
 </contextMonitor>
<contextMonitors>
</CS>
```

Figura 4.18: Configuração de Monitoração em XML

4.3.3.3 Terceiro Passo - Implementando a OniPresença

A implementação da Onipresença no *framework* DSPL2UbiTV considera a onipresença do serviço de interatividade. Assim, além do possibilidade de controlar quais *features* são mais adequadas ao usuário (dado seu contexto), disponibiliza um mecanismo de sessão onde os objetos utilizados pela aplicação podem ser mantidos em um ambiente remoto e acessados, em seu status mais atual, caso o usuário troque de equipamento receptor.

Desta forma, por exemplo, caso o telespectador esteja utilizando T-Commerce (em sua TV Convencional) e tenha adicionado algum produto ao carrinho de compras, ao migrar para outro receptor (televisor móvel) o *framework* DSPL2UbiTV identificará o novo contexto, configurará a aplicação adequadamente e buscará as informações mantidas na sessão com o objetivo de manter a interatividade do usuário similar a do outro

equipamento.

A Tabela 4.9 exemplifica, nas linhas 4,5 e 6, como a aplicação (*ITVFeature*) pode buscar as informações relativas às interatividades recentes e mantê-las no novo ambiente operacional. É importante destacar que as informações, uma vez buscadas da sessão, são mantidas automaticamente, em caso de atualização das informações (linha 9).

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 ShoppingCart cart; ...
3 public void initITVFeature(){ ...
4 if(!ubiTVsession.isNew()){
5 cart = (ShoppingCart)ubiTVsession.getITVProperty("cart"));...}
6 }
7 public void run() {
8 ...
9 cart.addToCart(book,1);
10 ...
11 } ...  }
```

Tabela 4.9: ITVFeatureA Class - Exemplo de Onipresença

4.3.3.4 Quarto Passo -Implementando a Captura de Experiências

Neste trabalho, a captura de experiências representa o conhecimento sobre quais *features*, o usuário do sistema mais utiliza em determinados contextos. Assim, a configuração e reconfiguração do produto de software, a partir da SPLiDTV, podem tornar-se mais adequadas.

Para que o desenvolvedor de software implemente tal característica é necessário implementar o método **captureExperienceITVFeature()** para cada *feature* na qual objetiva-se obter conhecimento relativo à interatividade. Na versão atual do *framework* DSPL2UbiTV, a *feature* deve representar um elemento gráfico que compreenda os eventos de interação tratados pela interface **ActionListener**, onde a classe **ITVFature** implementa o método necessário (**actionPerformed()**) para o tratamento do evento (Tabela 4.10). Na tabela citada pode observar-se (na linha 11) a contabilização de uma interação para a *feature* a cada utilização da mesma.

Além disso, durante a implementação do método **captureExperienceITVFeature()**,

```
1 public abstract class ITVFeature implements Xlet, Observer, ITVFeatureInterface,
ActionListener {
...
2 @Override
3 public void actionPerformed(ActionEvent arg0) {
4 this.captureExperienceITVFeature();
5 DimensionalContext feature = new DimensionalContext();
6 feature.setName("FEATURE");
7 ContextVariable thisfeature = new ContextVariable();
8 thisfeature.setName(this.getClass().getSimpleName());
9 thisfeature.setContext(feature);
10 this.getCeContext().addContextVariable(thisfeature);
11 this.getCeContext().setValues(((Integer)this.getCeContext().getValues()+ 1));
12 cesession.setExperience(this.getCeContext()); }
...
}
```

Tabela 4.10: ITVFeature Class - Implementação do Método **actionPerformed()**

que representa o tratamento do evento de interação para *feature*, pode-se agregar informações para melhor identificação do contexto de interação (linha 16), pois a classe *ITVFeature* se restringe à compreensão de qual *feature* recebeu interação (Tabela 4.11). Outro detalhe importante, é que a *feature* deve se auto incluir na lista de ações que aguardam um evento (linha 11).

Sendo o contexto identificado através de uma perspectiva multidimensional, pode-se utilizar quantas dimensões se entender necessárias para obter informações relativas à interatividade e preferências do usuário. Além disso, como o serviço de captura de experiência é localizado remotamente, é possível implementar mecanismos de filtragem colaborativa entre usuários do serviço. Considera-se que os usuários serão identificados desde o início do processo de construção do produto de software. Assim, o modelo básico de informações de usuário é baseado nas informações de preferência de interação do usuário dados os contextos.

Finalmente, recomenda-se agregar informações ao contexto da *feature* nos métodos **initITVFeature()** e **resumeITVFeature()**, onde a identificação dos contextos deve ser obtida através das classes de Teste de Contexto encontradas no pacote de Sensibilidade ao Contexto (CS).

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 Button button;
3 public void run() {
4 button = new Button("Old Text");
5 try{ button.setIcon(...);
6 button.setSize(new Dimension(60,60));
7 }catch(Exception e){}
8 button.addActionListener(this);
9 form.addComponent(BorderLayout.WEST, button); }
10 ...
11 public void captureExperienceITVFeature() {
12 DimensionalContextInterface dctime = new DimensionalContextImp();
13 dctime.setName("TIME");
14 ContextVariableInterface cvsunday = new ContextVariableImp();
15 cvsunday.setName("SUNDAY"); cvsunday.setContext(dctime);
16 this.getCeContext().addContextVariable(cvsunday);
17 button.setText("New Text"); }
18 ...  } }
```

Tabela 4.11: ITVFeatureA Class - Implementação do Método **captureExperienceITVFeature()**

4.3.3.5 Quinto Passo - Implementando a Recomendação de *features*

A recomendação de *features* é necessária no mínimo, nas situações onde as *features* não são definidas como mandatórias no Modelo de *features*. Uma vez que a seleção das *features* é feita em tempo de execução e de forma automática, um sistema de recomendação de *features* foi implementado para dar apoio a decisão. Na atual versão do *framework*, o sistema de recomendação é baseado em ciência de contexto [8][9]. Assim, utilizando as informações coletadas através da implementação da característica de ubiquidade denominada **Captura de Experiências**, podem-se obter recomendações sobre quais *features* devem ser selecionadas.

É importante lembrar que o controle sobre se a *feature* será ou não parte do produto de software, em um dado momento, é feito pela própria *feature*. Nesse sentido, assim como no caso da verificação se a *feature* é adequada para um determinado contexto, existe um método a ser implementado nas *features* que identifica se a *feature* é recomendável ou não ao produto de software a ser configurado. Trata-se do método **ge-**

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 ...
3 public IndexRecommended getListRecommender(ITVFeature feature){
4 DimensionalContext dctime = new DimensionalContext();
5 dctime.setName("TIME");
6 ContextVariable cvsunday = new ContextVariable();
7 cvsunday.setName("SUNDAY"); cvsunday.setContext(dctime);
8 this.getCeContext().addContextVariable(cvsunday);
9 this.index = (IndexRecommended)
this.getFrsession().getValues(this.getCeContext(), false);
10 this.index.setCardinality(1);
11 return index;
12 ...  } }
```

Tabela 4.12: ITVFeatureA Class - Implementação do Método **getListRecommender()**

tListRecommender() que deve ser implementado pelos desenvolvedores. O método **getListRecommender()**, exemplificado na Tabela (4.12), retorna um objeto do tipo *IndexRecommended* (linha 3). A classe *IndexRecommended* possui dois atributos principais, sendo o primeiro o índice de recomendação da *feature*, caso existam várias *features* disputando a execução (relacionamentos ALTERNATIVE). O segundo atributo trata-se da cardinalidade, ou seja, quantas *features* podem ser executadas.

O objeto do tipo *IndexRecommended* pode ser retornado também, quando consultada a sessão (**frsession**) utilizada para obter recomendações sobre um determinado contexto (linha 9, Tabela - 4.12). O contexto sempre conterá a informação sobre a *feature* avaliada.

Uma vez obtido o objeto do tipo *IndexRecommended* através da sessão **frsession**, o método **verifyRecommendedFeature()**, implementado na classe **ITVFeature** (Tabela 4.13), verifica-se a *feature* deve ser executada segundo o seguinte critério: se o valor obtido para o atributo de cardinalidade for maior ou igual ao valor do obtido para o atributo de índice (IndexOf) a *feature* deve ser executada.

Assim, cabe ao desenvolvedor implementar as classes de regras que determinam os valores de índice e cardinalidade considerados adequados.

Finalmente, recomenda-se agregar informações ao contexto da *feature* nos métodos **ini-**

```
 1 public abstract class ITVFeature implements Xlet, Observer, ITVFeatureInterface,
ActionListener {
...
2 @Override
3 public void verifyRecommendedFeature(){
4 DimensionalContext feature = new DimensionalContext();
5 feature.setName("FEATURE");
6 ContextVariable thisfeature = new ContextVariable();
7 thisfeature.setName(this.getClass().getSimpleName());
8 thisfeature.setContext(feature);
9 this.getCeContext().addContextVariable(thisfeature);
10 if(noParent)
11 { this.index = this.getListRecommender(this); }
12 else
13 { this.index = parent.getListRecommender(this);}
14 if(this.index.getCardinality() >= this.index.getIndexOf())
15 { this.recommended = true;}
16 else
17 { this.recommended = false; } }
...
}
```

Tabela 4.13: ITVFeature Class - Implementação do Método **verifyRecommendedFeature()**

tITVFeature() e **resumeITVFeature()**, onde a identificação dos contextos deve ser obtida através das classes de Teste de contexto encontradas no pacote de Sensibilidade ao Contexto (CS).

4.3.3.6 Sexto Passo - Implementando a Heterogeneidade de Dispositivos

Uma linha de produtos de software possui como característica principal a possibilidade de configuração de vários produtos de software distintos (mas com similaridades) a partir de uma LPS.

Para atender a demanda de contrução de aplicativos que possuam suporte a diversos dispositivos, a LPS prevê a possibilidade de implementação de *features* específicas para determinados dispositivos e, no momento da configuração do produto para um

determinado dispositivo, selecioná-los para composição do produto adequadamente.

Assim, entende-se que o *framework* atende à característica de projetos de software ubíquos denominada **Heterogeneidade de Dispositivos**, uma vez que as *features* e controles de configuração do produto de software sejam desenvolvidos adequadamente.

4.3.4 Configuração e Geração do Produto de Software

Como já comentado em seções anteriores, a configuração de um produto de software é baseada na seleção de *features* disponibilizadas em uma linha de produtos de software. No caso de aplicações interativas para TV digital a seleção do produto, se feita com antecedência, poderá não ser capaz de executar em alguns equipamentos devido a limitações relativas a processamento, resolução, mecanismos de interação, etc.

Neste sentido, para utilização de Linhas de Produtos de Software no desenvolvimento de aplicações interativas para TV Digital, entende-se que é necessário usar um mecanismo de configuração e geração do produto de software em tempo de execução, ou seja, utilizar Linhas de Produtos de Software Dinâmicas.

Neste trabalho, a configuração e reconfiguração do produto (dada uma SPLiDTV), distribui a tarefa de seleção das *features* por cada *ITVFeature* considerada no modelo de *Features*. Na prática, cada *ITVFeature* controla seu ciclo de vida e de seus dependentes.

Cada *ITVFeature* se auto-avalia com relação a 3 situações que podem impedi-la de executar:

- Auto-verifica se o ambiente possui recursos para execução da *ITVFeature*;
- Auto-verifica se a *ITVFeature* possui sua execução recomendável;
- Auto-verifica se existe alguma restrição que impeça sua execução, como por exemplo, ter sua execução depedendente do status de execução de outra *ITVFeature*.

Assim, o configurador dinâmico de produtos de software inicializa todas as *ITVFeatures* e cada uma delas se auto-avalia quanto a passagem para um possível estado de execução.

A passagem de um estado para outro é baseada na existência de um ciclo de vida para uma *ITVFeature*. Tal ciclo de vida é bastante similar ao ciclo de vida de um *Xlet* (objeto que representa uma mídia interativa em TV Digital, implementado em Java). Assim, uma *ITVFeature* pode se encontrar nos seguintes estados:

Loaded - o método **initITVFeature()** é utilizado na inicialização da *ITVFeature*;

Started - o método **startITVFeature()** é utilizado na execução da *ITVFeature*;

Paused o método **pauseITVFeature()** é utilizado para pausar uma *ITVFeature*;

Destroyed - o método **destroyITVFeature()** é utilizado para encerrar qualquer tipo de atividade de uma *ITVFeature*;

Reloaded - o método **resumeITVFeature()** é utilizado para reiniciar uma *ITVFeature*. Na prática a *ITVFeature* volta ao estado de **Loaded**, momento a partir do qual se auto-avalia com relação à possibidade de mudança para o estado de execução.

A Figura (4.19) apresenta as possibilidades de transição de estados de uma *ITVFeature*.

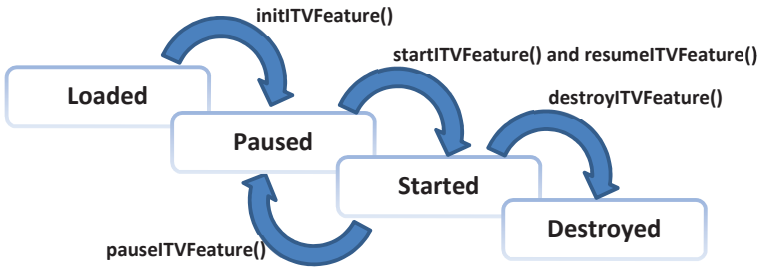

Figura 4.19: Ciclo de Vida - ITVFeature

Considerando que este trabalho utiliza uma programação orientada a *features*, as *ITVFeatures* são organizadas através de um modelo de *features*. Sendo assim, cada *ITVFeature* pode depender de uma *ITVFeature* em uma posição hierárquica mais elevada, bem como pode possuir *ITVFeatures* dependentes. Nesses casos, quando uma *ITVFeature* entra nos estados de pausada, inicializada, reinicializada e destruída todos os dependentes devem entrar no mesmo estado da *ITVFeature* da qual dependem.

Enfim, considerando as características de execução das ITVFeatures e sua organização dentro de um modelo de *features*, entende-se que o *framework* DSPL2UbiTV implementa a característica de projetos de software ubíquos denominada **Comportamento Adaptável**.

5 EXPERIMENTAÇÃO E RESULTADOS

Neste capítulo são apresentados os experimentos definidos e os elementos que tornaram válidas as contribuições propostas no trabalho.

Inicialmente, faz-se necessário destacar que a principal contribuição do trabalho é o desenvolvimento de uma ferramenta que ofereça suporte ao desenvolvimento de Linhas de Produtos de Software no ambiente da TV Digital interativa, permitindo a seleção de produtos de software em acordo com as particularidades apresentadas pelo ambiente de TV Digital. Além disso, considera-se como contribuição relevante o suporte dado pela ferramenta à construção de aplicações com características de softwares ubíquos.

Dado que o ambiente de TV Digital interativa possui características particulares de transmissão e execução dos conteúdos interativos (mencionadas nas seções 1.1 e 2.1), devem ser utilizadas técnicas para implementar a solução, tais como: mecanismos de auto-configuração do software e ciência de contexto.

Devido a ferramenta ser uma solução de apoio ao desenvolvimento de software, deve-se avaliar a ferramenta em acordo com atributos de qualidade de software. No contexto deste trabalho, o atributo de qualidade considerado para avaliação é a manutenibilidade de software, mais especificamente métricas relativas a modificabilidade de.

Sendo assim a validação do trabalho foi realizada através de dois *Case Studies* (Estudos de Caso) [63] distintos, mas que permitem a avaliação das contribuições do trabalho de um forma geral.

1. Validação da capacidade da plataforma de oferecer melhores resultados, relativos a atributos de qualidade de software, a partir do uso da plataforma para reconstruir (*refactoring*) aplicações já existentes. Neste contexto, a plataforma é avaliada em acordo com o estudo experimental I;

2. Validação da capacidade da plataforma de oferecer suporte à modificabilidade de software (onde a modificabilidade de sofware é uma sub-característica do item de

qualidade de software previsto na ISO IEC 9126 [33] denominado de Manutenibilidade de Software). Neste sentido, os atributos de qualidade de complexidade e extensibilidade serão avaliados segundo os estudos experimentais II e III;

O objetivo da validação através de dois estudos de caso, é o de identificar, através do primeiro estudo de caso, se o desenvolvimento através da plataforma apresenta vantagens, relativas à qualidade de software, quando comparadas ao desenvolvimento tradicional. Uma vez que são identificadas melhorias de qualidade, pode-se validar (através do segundo estudo de caso) os impactos qualitativos da utilização da plataforma uma vez que as aplicações, construidas através da plataforma, sofrem manutenção. A tabela 5.1 apresenta um resumo relativo as principais caracterísicas dos estudos experimentais realizados.

Experimento	**Finalidade**	**Métricas**	**Dados Estatísticos**
I	Comparação entre aplicações desenvolvidas usando OO e refactoring utilizando o *framework*	McCabe, WMC, Extensibility, SIX, NOC e DIT	3 aplicações foram refactoradas
II	Impacto nos atributos de qualidade dada a evolução de uma aplicação para TVDi	CompPLA, ExtensPLA, DITPLA, SIXPLA e NOCPLA	População: 48 configurações possíves de produtos; Amostra: 8 configurações de produtos
III	Impacto nos atributos de qualidade dada a evolução de uma aplicação para TVDi utilizando características de ubiquidade	CompPLA, ExtensPLA, DITPLA, SIXPLA e NOCPLA	População: 48 configurações possíves de produtos; Amostra: 8 configurações de produtos

Tabela 5.1: Resumo das características dos estudos experimentais realizados

5.1 DEFINIÇÃO, PLANEJAMENTO E EXECUÇÃO DOS EXPERIMENTOS

5.1.1 Estudo Experimental I

5.1.1.1 Definição do Estudo Experimental I

Com base no *template* GQM [12], o objetivo do experimento I é apresentado a seguir:

Analisar métricas básicas de qualidade de software (relativas à Complexidade e Extensibilidade).

Com propósito de avaliação/comparação.

Referente a resultados obtidos quando da avaliação de aplicações interativas para TV Digital desenvolvidas utilizando o paradigma de Orientação a Objetos (sem apoio de nenhum *framework* de desenvolvimento) quando comparadas à reconstrução de tais aplicações (*refactoring*) utilizando a plataforma para construção de LPSs para TVDi proposta.

Do ponto de vista do Arquiteto/Desenvolvedor de Software.

No contexto de Aluno de Programa de Pós-Graduação em Engenharia de Software ou áreas correlatas.

5.1.1.2 Planejamento do Estudo Experimental I

Considera-se para o planejamento os itens a seguir:

Contexto Global: A avaliação da plataforma proposta quanto a capacidade de melhoria de atributos de qualidade de software, é relevante uma vez que se faz necessário identificar se a ferramenta proposta, quando comparada com soluções tradicionais de desenvolvimento, realmente representa uma evolução qualitativa aos softwares produzidos.

Contexto Local: Avaliação da plataforma proposta quanto a capacidade de melhoria de atributos de qualidade de software, relativos à manutenibilidade, quando comparada com os resultados obtidos por aplicações pré-existentes que não se utilizam da plataforma. Sendo assim, através deste estudo de caso objetiva-se

identificar se a utilização da plataforma para desenvolvimento de aplicações (em forma de LPS) para TV Digital oferece melhorias de qualidade de software.

Projeto Piloto: antes da execução do estudo real foi realizado um projeto piloto com vistas a avaliar a instrumentação que foi utilizada no estudo experimental. Para tanto, somente uma aplicação pré-existente foi considerada. O executor utilizou os mesmos instrumentos do estudo experimental. Os dados obtidos pelo estudo piloto serão considerados para composição das dados e análises do estudo, devido a limitação de aplicações pré-existentes disponíveis no momento da execução do experimento.

Metodologia A seguir são apresentados os passos necessários par a execução do experimento:

1. O executor deve dispor de aplicação(ões) pré-existente(s), procedural(is), para TV Digital interativa, construida(s) com base na API Java TV [2];
2. O executor deve reescrever (refatorar) a(s) aplicação(ões) disponível(eis) de acordo com processo FOSD adaptado à plataforma proposta;
3. O executor pode obter os valores relativos às métricas, consideradas no estudo, para a(s) aplicação(ões) pré-existente(s) e aplicação(ões) reconstruída(s), em forma de SPL, através de cálculo manual. Entretanto, sugere-se a utilização da ferramenta Metrics [3] para automatização dos calculos das métricas;
4. O executor deve rejeitar ou aceitar as hipóteses identificadas.

Instrumentação: O conjunto de elementos necessários para a execução do experimento é composto por:

- uma cópia dos softwares necessários para testar as aplicações para TV Digital (no caso, utilizou-se a set-top-box virtual denominada OpenGinga [4] para execução das aplicações interativas).
- uma cópia das aplicações pré-existentes para serem reconstruidas utilizando a plataforma;
- uma cópia dos códigos relativos à plataforma proposta;
- uma cópia das intruções relativas à instalação e utilização da ferramenta Metrics na IDE Eclipse;

Métricas: Abaixo são apresentadas as métricas básicas consideradas para comparação entre as aplicações para TV Digital interativa.

Métricas relativas a Complexidade:

McCabe McCabe Cyclomatic Complexity - A complexidade ciclomática de uma seção do código fonte é a quantidade de caminhos independentes pelo código. Por exemplo, se o código fonte não contém estruturas de controle senão sequenciais a complexidade é 1, já que há somente um caminho válido através do código. Se o código possui somente uma estrutura de seleção contendo somente uma condição, então há dois caminhos possíveis, aquele quando a condição é avaliada em verdadeiro, e aquele quando a condição é avaliada em falso [42].

Assim, a métrica de **McCabe** é calculada através da seguinte equação (5.1):

$$M = E - N + (2 * P) \tag{5.1}$$

Onde:

- (M) representa a complexidade ciclomática de McCabe;
- (E) representa a quantidade de caminhos possíveis;
- (N) representa o número de nós envolvidos;
- (P) representa a quantidade de nós conectados;

Cabe destacar que quanto maior o valor identificado para métrica de McCabe, maior será a complexidade associada ao software.

WMC Os Métodos ponderados por classe (*Weighted Methods per Class*) são obtidos através do somatório das complexidades dos métodos da classe avaliada [18].

Assim, a métrica **WMC** é calculada através da seguinte equação (5.2):

$$WMC_{Cls} = \sum_{n=1}^{M} Cls \tag{5.2}$$

Onde:

- (Cls) representa a classe avaliada;
- (M) representa o método avaliado da classe;
- (C) representa o valor considerado para complexidade;

Cabe destacar que quanto maior o valor identificado para métrica WMC, maior será a complexidade associada ao software.

Métricas relativas a Extensibilidade:

Extensibility - mede a percentagem de métodos abstratos com relação ao total de métodos (abstratos mais os concretos) de uma classe [46].

Assim, a métrica **Extensibility** é calculada através da seguinte equação (5.3):

$$Extensibility_{Cls} = \frac{\sum_{n=1}^{M_{abst}}}{\sum_{n=1}^{M_{abst}} + \sum_{n=1}^{M_{conc}}} \tag{5.3}$$

Onde:

- (Cls) representa a classe avaliada;
- (M_{abst}) representa os métodos abstratos da classe;
- (M_{conc}) representa os métodos concretos da classe;

Cabe destacar que quanto maior o valor identificado para métrica Extensibility, maior será a extensibilidade associada ao software.

Métricas relativas a Reuso:

DIT - mede é o número máximo de superclasses posicionadas hierarquicamente acima da classe em questão [18].

Assim, a métrica **DIT** é calculada através da seguinte equação (5.4):

$$DIT_{Cls} = Cls_d \tag{5.4}$$

Onde:

- (Cls_d) representa o núnero de superclasses da classe avaliadas;

Cabe destacar que quanto maior o valor identificado para métrica DIT, maior será o reuso associado ao software.

NOC - mede o número máximo de subclasses posicionadas hierarquicamente abaixo da classe em questão [18].

Assim, a métrica **NOC** é calculada através da seguinte equação (5.5):

$$NOC_{Cls} = Cls_n \tag{5.5}$$

Onde:

- (Cls_n) representa o núnero de subclasses da classe avaliadas;

Cabe destacar que quanto maior o valor identificado para métrica NOC, maior será o reuso associado ao software.

SIX Specialization Index - mede a especialização em que as subclasses rescrevem os métodos das classes superiores (superclasses)[40].

Assim, a métrica **SIX** é calculada através da seguinte equação (5.6):

$$SIX_{Cls} = \frac{NMO_{Cls} * DIT_{Cls}}{N} \tag{5.6}$$

Onde:

- (Cls) representa a classe avaliada;
- (NMO) representa o valor da métrica NMO (Number of Methods Overriden) da classe;
- (DIT) representa o valor da métrica DIT (Depth of Inheritance Tree) da classe;
- (N) representa o número de métodos da classe;

Cabe destacar que quanto maior o valor identificado para métrica SIX (Specialization Index), maior será a extensibilidade associada ao software. Tal métrica possui relação com o atributo de extensibilidade uma vez que relaciona métricas ligadas a implementação de métodos abstratos, definidos em interfaces e classes, com métricas indicativas de herança. Assim, sob o ponto de vista de extensibilidade, uma maior quantidade de métodos subescritos em vários níveis hierárquicos apontam para facilidade de extensão/manutenção do software através de polimorfismo.

Formulação de Hipóteses: as seguintes hipóteses são propostas para o estudo experimental em questão:

Hipóteses Relativas à Complexidade Com relação as métricas de complexidade são avaliadas as seguintes hipóteses no estudo:

- **Hipótese Nula H_0:** Obter valores maiores relativos às métricas de **McCabe** e **WMC** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$McCabe(TVDiSPL) > McCabe(TVDiOO) \tag{5.7}$$

$$WMC(TVDiSPL) > WMC(TVDiOO) \tag{5.8}$$

- **Hipótese Alternativa 1 H_1:** Obter valores menores relativos à métrica de **McCabe** e maiores relativos à métrica **WMC** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$McCabe(TVDiSPL) < McCabe(TVDiOO) \tag{5.9}$$

$$WMC(TVDiSPL) > WMC(TVDiOO) \tag{5.10}$$

- **Hipótese Alternativa 2 H_2:** Obter valores maiores relativos à métrica de **McCabe** e menores relativos à métrica **WMC** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$McCabe(TVDiSPL) > McCabe(TVDiOO) \tag{5.11}$$

$$WMC(TVDiSPL) < WMC(TVDiOO) \tag{5.12}$$

- **Hipótese Alternativa 3 H_3:** Obter valores menores relativos às métricas de **McCabe** e **WMC** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$McCabe(TVDiSPL) < McCabe(TVDiOO) \tag{5.13}$$

$$WMC(TVDiSPL) < WMC(TVDiOO) \tag{5.14}$$

Hipóteses Relativas à Extensibilidade Com relação as métricas de extensibilidade são avaliadas as seguintes hipóteses no estudo:

- **Hipótese Nula H_0:** Obter valores maiores relativos às métricas de **Extensibility** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$Extensibility(TVDiSPL) > Extensibility(TVDiOO) \tag{5.15}$$

- **Hipótese Alternativa 1 H_1:** Obter valores menores relativos às métricas de **Extensibility** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$Extensibility(TVDiSPL) < Extensibility(TVDiOO) \quad (5.16)$$

Hipóteses Relativas ao Reuso Com relação as métricas de reuso são avaliadas as seguintes hipóteses no estudo:

- **Hipótese Nula H_0:** Obter valores maiores relativos às métricas **DIT**, **NOC** e **SIX** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$DIT(TVDiSPL) > DIT(TVDiOO) \quad (5.17)$$

$$NOC(TVDiSPL) > NOC(TVDiOO) \quad (5.18)$$

$$SIX(TVDiSPL) > SIX(TVDiOO) \quad (5.19)$$

- **Hipótese Alternativa 1 H_1:** Obter valores menores relativos às métricas de **DIT**, **NOC** e **SIX** para as aplicações reconstruídas utilizando-se da plataforma (**TVDiSPL**) quando comparadas as aplicações tradicionais (**TVDiOO**). Assim, as seguintes situações são avaliadas:

$$DIT(TVDiSPL) < DIT(TVDiOO) \quad (5.20)$$

$$NOC(TVDiSPL) < NOC(TVDiOO) \quad (5.21)$$

$$SIX(TVDiSPL) < SIX(TVDiOO) \quad (5.22)$$

Variáveis Dependentes: As complexidades, extensibilidades e reuso das aplicações de TV Digital avaliadas.

Variáveis Independentes: As métricas McCabe e WMC para complexidade, métrica Extensibility para extensibilidade, e métricas DIT, NOC e SIX para reuso.

Análise Quantitativa: Tem por objetivo avaliar os resultados obtidos neste estudo, por meio de comparações entre os resultados. Objetivando, principalmente, identificar a rejeição, ou não, das hipóteses H_0 apresentada anteriormente.

Mecanismos de Análise: o estudo proposto deve analisar os resultados obtidos para as métricas de complexidade e extensibilidade para os dois tipos de aplicações utilizadas no estudo. Após a obtenção dos resultados deve-se comparar os resultados a fim de identificar qual das aplicações comparadas obtem o melhor resultado dentre as váriavies (métricas) consideradas.

Validade Interna: Para garantir a validade interna do experimento será tomada a seguinte medida:

- As funcionalidades (códigos) das aplicações pré-existentes não serão alteradas, apenas reutilizados na SPL. Assim, apenas o código relativo a estrutura da plataforma será acrescentado.

Validade Externa: A validade externa do estudo experimental, mede sua capacidade de refletir o comportamento em toda a população que utilizará a plataforma. Assim, a validade externa deste estudo é considerada suficiente uma vez que utilizaram-se, para reconstrução, todas as aplicações pré-existentes disponíveis para realização do experimento. Além disso, as aplicações foram desenvolvidas por outros desenvolvedores, que não preocuparam-se com o aspecto comparativo. Sendo assim, as aplicações pré-existentes refletem a realidade do desenvolvimento de aplicações de TVDi.

Validade de Construção: A validade de construção do estudo se refere à relação entre os instrumentos, participantes do estudo e a teoria que está sendo avaliada. Acredita-se que a validade de construção esteja garantida, uma vez que o executor utiliza ferramentas automáticas para coletar as métricas, bem como a análise dos resultados apenas compara o desempenho obtido por cada abordagem considerada.

Validade de Conclusão: A validade de conclusão mede a relação entre os tratamentos e os resultados, determinando a capacidade do estudo de generalizar os resultados. Acredita-se que a validade de conclusão será parcialmente garantida, uma vez que a quantidade de aplicações pré-existentes disponíveis para experimentação não se trata de uma amostra signficativa. Assim, os resultados representam apenas indícios de melhoria ou não na qualidade do software desenvolvido.

5.1.1.3 Execução do Estudo Experimental I

Seguindo a Metodologia definida para execução do experimento foram obtidas três aplicações junto ao projeto OpenGinga. Tais aplicações são distribuidas, de forma livre, juntamente com o *set-top-box* virtual OpenGinga. As Aplicações utilizadas denominam-se Nemo, BBB, e ToV (Torcida Virtual). Todas as aplicações foram desenvolvidas no LAVID - Laboratório de Aplicações de Vídeo Digital, da Universidade Federal da Paraíba.

Assim, após o *refactoring* das aplicações utilizando o *framework* para o desenvolvimento de LPSs os seguintes resultados foram obtidos:

Aplicação Nemo: uma vez coletadas as métricas de qualidade a partir das aplicações, original e refatorada, foi possível comparar os resultados. Os resultados são apresentados através da Tabela 5.2 e Figura 5.1, observando-se que com relação a complexidade, extensibilidade e reuso as hipóteses nulas foram aceitas.

	Nemo	**NemoSPL**
McCabe	1,586	1,507
WMC	7,967	7,785
Extensibility	0	0,793
DIT	1	1,7
NOC	0	0,4
SIX	0	0,37

Tabela 5.2: Resultados comparando a aplicação Nemo original (OO) e a aplicação refatorada através do *Framework*

Figura 5.1: ScreenShot da Aplicação Nemo refatorada

Aplicação BBB: uma vez coletadas as métricas de qualidade a partir das aplicações, original e refatorada, foi possível comparar os resultados. Os resultados são apresentados através da Tabela 5.3 e Figura 5.2, observando-se que com relação à complexidade, extensibilidade e reuso, as hipóteses nulas foram aceitas.

Aplicação ToV uma vez coletadas as métricas de qualidade a partir das aplicações, original e refatorada, foi possível comparar os resultados. Os resultados são apresentados através da Tabela 5.4 e Figura 5.3, observando-se que com relação à complexidade, extensibilidade e reuso, as hipóteses nulas foram aceitas.

	BBB	BBBSPL
McCabe	2,071	1,487
WMC	29	7,722
Extensibility	0	0,116
DIT	2	1,778
NOC	0	0,361
SIX	0	0,585

Tabela 5.3: Resultados comparando a aplicação BBB original (OO) e a aplicação refatorada através do *Framework*

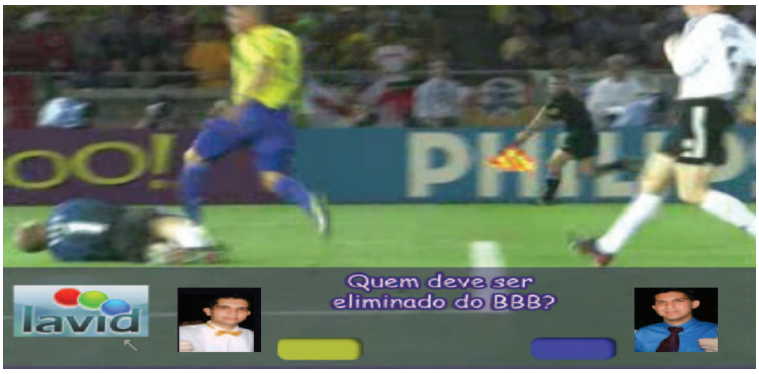

Figura 5.2: ScreenShot da Aplicação BBB refatorada

	ToV	ToVSPL
McCabe	1,685	1,553
WMC	7,955	7,512
Extensibility	0	0,652
DIT	1	1,68
NOC	0	0,38
SIX	0	0,39

Tabela 5.4: Resultados comparando a aplicação ToV original (OO) e a aplicação refatorada através do *Framework*

5.1.1.4 Considerações sobre os resultados obtidos no Estudo Experimental I

É possível concluir que, a partir dos casos utilizados, a utilização do *framework* proposto para o desenvolvimento de LPS é capaz de melhorar a qualidade dos produtos de softwares produzidos no tocante à complexidade, extensibilidade e reuso.

Considera-se os resultados positivos. Entretanto, acredita-se que se o projeto for desen-

Figura 5.3: ScreenShot da Aplicação ToV (Torcida Virtual) refatorada

volvido utilizando LPS desde o início do projeto, os resultados relativos aos atributos de qualidade podem ser ainda mais significativos.

Cabe destacar que o *refactoring* foi feito buscando apenas restruturar a aplicação para uma LPS. Assim, as funcionalidades desenvolvidas originalmente, não foram reescritas, fato este que prejudicou os resultados obtidos pelos atributos de qualidade devido à limitação imposta para definição de *features* na refatoração.

5.1.2 Estudo Experimental II

5.1.2.1 Definição do Estudo Experimental II

Com base no *template* GQM [12], o objetivo do experimento II é apresentado a seguir:

Analisar métricas de complexidade e extensibilidade obtidas a partir da arquitetura dos produtos gerados através de uma SPL desenvolvida com base na Plataforma para desenvolvimento de LPSs proposta.

Com propósito de avaliação

Referente ao impacto causado aos atributos de qualidade de software, uma vez que a LPS seja manutenida (inclusão de novas features e produtos), considerando a abordagem dinâmica (em tempo de execução) de configuração de produtos de Software (LPSD).

Do ponto de vista do Arquiteto/Desenvolvedor de Software.

No contexto de Aluno de Programa de Pós-Graduação em Engenharia de Software ou áreas correlatas.

5.1.2.2 Planejamento do Estudo Experimental II

Considera-se para o planejamento os itens a seguir, em que a itemização adotada segue, em boa parte, a sequência de itens usada em [47], com alteracoes e complementos julgados pertinentes.

Contexto Global: Avaliar a modificabilidade de uma arquitetura de uma Linha de Produtos de Software torna-se relevante pois tal avaliação permite a análise de distintas configurações do núcleo (*Core Assets*), bem como a evolução e o desenvolvimentos dos possíveis produtos [46] [47]. Sendo assim, avaliar a modificabilidade da arquitetura dos possíveis produtos de uma LPS oferece indicador de facilidade ou dificuldade de manter, e até mesmo evoluir os produtos no futuro. Além disso, no que concerne a plataforma proposta, é importante identificar os impactos que as ferramentas, utilizadas para dar suporte ao desenvolvimento de software causam na qualidade dos produtos desenvolvidos, tanto para o desenvolvimento inicial quanto para futuras evoluções.

Contexto Local: este estudo visa avaliar a complexidade, extensibilidade e reuso de um conjunto de produtos de software, desenvolvidos para TV Digital interativa, com base na abordagem de LPS, bem como de LPSD. Uma LPS será desensolvida como estudo de caso utilizando, como ferramenta de apoio para o desenvolvimento da LPS, a plataforma proposta neste trabalho. Ao longo do desenvolvimento da LPS serão coletadas métricas de complexidade, extensibilidade e reuso para observação do impacto que a plataforma causa nos atributos de qualidade à medida que produtos de software, com quantidades diferentes de *features* são considerados, bem como quando funcionalidades relacionadas a abordagem dinâmica da LPS são utilizadas.

Cabe ressaltar que a complexidade, extensibilidade e reuso não são as únicas métricas consideradas para avaliar a modificabilidade de um software. Entretanto, tais métricas foram adotadas na avaliação devido a sua importância para o entendimento sobre o grau de modificabilidade de software, bem como por serem métricas propostas/validadas para análise de modificabilidade da arquitetura de uma LPS em [47].

Metodologia: O experimento deve seguir o seguinte conjunto de tarefas por parte do executor:

1. Deve-se, a partir da plataforma proposta (seguindo o processo FOSD adaptado), construir uma aplicação para TV Digital interativa, baseada em requisitos de software pré-estabelecidos;
2. A partir da implementação (ou projeto), com base na plataforma, deve-se seguir as seguintes etapas previstas no Método SystemPLA [47] (Figura 5.16):

Figura 5.4: Método SystemPLA - processo aplicado

Definir Cenários : Uma vez definido o modelo de classes da aplicação (em tempo de projeto ou implementação), é necessário gerenciar as variabilidades da Linha de Produtos. A abordagem adotada para o gerenciamento de variabilidade é a abordagem denominada ***SMarty*** [47], utilizando particularmente o modelo de variabilidade para classes (os instrumentos necessários (SMarty Profile) para gerenciamento das variabilidades serão apresentados no item relativo à instrumentação);

Criar Configurações de ALP: Uma vez identificadas as varialibilidades da Linha, é necessário definir as configurações dos produtos que serão avaliadas.

Coletar Métricas a partir das Configurações: Neste caso, antes de coletar as métricas relativas aos atributos de qualidade dos produtos configurados, torna-se necessário identificar as métricas básicas da ALP (as definições relativas as métricas básicas de ALP serão apresentados no item relativo à instrumentação). Adicionalmente, é necessário coletar as métricas considerando a abordagem dinâmica de configuração dos produtos.

Análise dos Dados: Os dados coletados devem seguir as orientações apresentadas no item abaixo para análise.

3. Uma vez definidas as configurações dos produtos e coletadas as métricas iniciais é necessário coletar as métricas de complexidade, extensibilidade e reuso a partir de diferentes produtos configurados a partir da LPS. Os valores

obtidos para as métricas devem gerar uma amostra de valores relativos a cada produto configurado.

4. Deve-se avaliar a normalidade das amostras, a fim de identificar qual método de correlação deverá ser utilizado.
5. Deve-se identificar a correlação linear entre as amostras coletadas, para cada métrica considerada, e o número de *features* que compõem o produto configurado, bem como percentual de variabilidades identificada em cada produto considerado. O objetivo de tal correlação é identificar se existe impacto, aos atributos de qualidade, para diferentes quantidades de features utilizadas nos produtos/LPS, bem como a quantidade de variabilidades utilizadas na linha, uma vez que existe a construção dinâmica de produtos e tal controle possui reflexos na construção dos códigos;
6. Supõe-se, a partir das características de extensão da plataforma proposta, que as amostras devem apresentar forte similaridade. Assim, confirmadas as correlações das amostras, deve-se aplicar a técnica de regressão linear as amostras. A partir da amostra selecionada deve-se obter uma função que permita projetar o desempenho da plataforma à medida que novas funcionalidades (*features*) e variabilidades forem adicionadas à LPS.
7. Uma vez definidas as funções deve-se fazer projeções de evolução da LPS e analisar a aceitação ou rejeição das hipóteses consideradas para o experimento.

É importante destacar que classes que devem ser avaliadas neste experimento limitam-se as classes do *Core Asset* da plataforma, bem como das classes extendidas (de fronteira) pelo desenvolvedor. As classes extendidas pelo desenvolvedor são controladas pela plataforma. Assim, isolando a parte relativa à plataforma dos códigos produzidos pelo desenvolvedor (para tratar de requisitos específicos), é possível identificar/projetar o impacto causado pela plataforma aos atributos de qualidade ao longo da evolução da LPS.

Instrumentação: A seguir, são apresentados, resumidamente os instrumentos utilizados/necessários para execução das tarefas previstas na metodologia:

- **Plataforma proposta** - Para executar o experimento, o executor deve ser capaz de desenvolver aplicações a partir dos códigos fornecidos pela plataforma, conforme pode ser observardo no capítulo 4 deste documento, onde o processo FOSD aplicado à plataforma é apresentado com maiores detalhes;

- **Abordagem SMarty** - Antes de obter os valores relativos às métricas é necessário preparar os artefatos de software (ex: Diagramas de Classe) para compreensão relativa à variabilidade existente na arquitetura da LPS avaliada. Para esse fim, este trabalho utiliza a abordagem SMarty, proposta/detalhada em [47]-capítulo 4, a partir da qual é possível identificar junto aos artefatos de software, através de estereótipos, o grau de variabilidade dos elementos considerados.

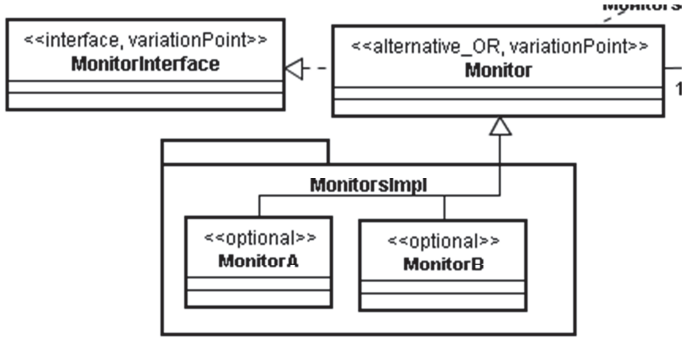

Figura 5.5: Diagrama de Classes (Parcial) - Sensibilidade ao Contexto

A Figura 5.5 apresenta a utilização de três estereótipos considerados pela abordagem SMarty. Tratam-se dos estereótipos:

$<<$ *variationPoint* $>>$ representa que o artefato (abstrato) é um ponto onde existe algum tipo de variabilidade. Nesse caso, o artefato deve possuir, vinculado a ele, artefatos que implementem de forma concreta as funcionalidades previstas de forma abstrata;

$<<$ *alternativeOR* $>>$ representa que o artefato, no caso a classe, pode ser selecionada dentre um conjunto de outras implementações da $<<$ *variationPoint* $>>$;

$<<$ *optional* $>>$ representa que o artefato é uma implementação de uma $<<$ *variationPoint* $>>$ e permite sua seleção opcional para composição de um produto de software.

A abordagem SMarty completa, como definida por [47], especificamente no capítulo 4 da citada referência bibliográfica.

A partir da identificação dos artefatos obrigatórios e variabilidades, de uma LPS, em um artefato de software, é possível calcular as métricas de complexidade e extensibilidade propostas por [47].

- **Ferramenta SDMetrics e Métricas Básicas de Avaliação de ALPS** ([47]) - A Ferramenta SDMetrics permite a análise automática de códigos para obtenção de métricas relativas à avaliação de software. Visando a

automatização da avaliação da arquitetura de uma LPS que se utiliza da abordagem SMarty, alguns passos são necessários:

1. Obter o código em XMI (*XML Metadata Interchange*) de todos os artefatos UML a serem avaliados. É importante destacar que os estereótipos identificados devem estar presentes no código XMI necessário. Também é importante destacar que várias ferramentas de modelagem UML são capazes de gerar o código XMI automaticamente;
2. Para que o SDMetrics compreenda as métricas básicas, para avaliação da arquitetura LPS, propostas/validadas em [47] é necessário configura-lo adequadamente utilizando um documento em XML disponível em [47], especificamente nos apêndices. A Figura 5.6 apresenta um trecho do código XML necessário;
3. Uma vez configurado adequadamente o SDMetrics utiliza-se o código em XMI dos artefatos da arquitetura LPS para calcular as métricas básicas.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<sdmetrics version="2.3" ruleexemption="taggedvalue" exemptiontag="tagname" >
<metric name="CLS_CLS_BAS_VPT_ISA" domain="class" category="variation point" internal="">
<description>Indica se a classe é um ponto de variação</description>
<projection relset="stereotypes" target="stereotype" condition="name='variationPoint'"/>
</metric>
<metric name="CLS_CLS_BAS_INC_ISA" domain="class" category="variant" internal="">
<description>Indica se a classe é uma Variante Inclusiva</description>
<projection relset="stereotypes" target="stereotype" condition="name='alternative_OR'"/>
</metric>
...
```

Figura 5.6: Métricas Básicas para avaliação de ALPS em XML

Cabe destacar que é possível realizar o processo de coleta das métricas básicas da ALP sem o uso de ferramenta de automação. Tal situação é recomendada em caso de dificuldade de gerar os códigos XMI com os esteriótipos necessários. Neste contexto, a geração dos códigos XMI dependerá da capacidade da ferramenta utilizada para projetar as classes da LPS.

- *Construção básica das métricas de ALP* - As métricas de ALP, baseadas em métricas de classe, podem ser construídas de forma genérica a partir do seguinte conjunto de métricas intermediárias e devidas equações (Tabela 5.5).

  Onde :

Métrica	Descrição
Metric_Interface	$Metric\_Interface_{Int} = InterfaceMetricValue$
Metric_Class	$Metric\_Class_{Cls} = ClassMetricValue$
Metric_VarPointClass	$Metric\_VarPointClass_{Cls} = \sum_{i=1}^{NCls} Metric\_Class(Cls_i) + \sum_{i=1}^{NInt} Metric\_Interface(Int_i)$
Metric_VariabilityClass	$Metric\_VariabilityClass_{Vbt} = \sum_{i=1}^{NVpt} Metric\_VarPointClass(Vpt_i)$
Metric_VarComponent	$Metric\_VarComponent_{Vc} = \sum_{i=1}^{NVbt} Metric\_VariabilityClass(Vbt_i)$
Metric_PLA	$Metric\_PLA = \sum_{i=1}^{NVc} Metric\_VarComponent(Vc_i)$

Tabela 5.5: Construção da métricas de ALP Genérica

  - ($Metric$) representa a métrica de classe básica que deseja ser calculada, por exemplo: WMC, DIT e NOC;
  - (Int) representa uma interface em um diagrama UML;
  - (Cls) representa uma classe em um diagrama UML;
  - ($InterfaceMetricValue$) representa o valor de métrica utilizada para uma interface em um diagrama UML;
  - ($ClassMetricValue$) representa o valor de métrica utilizada para uma classe em um diagrama UML;
  - (Vpt) representa uma classe ou interface que é um ponto de variação em uma arquitetura de ALP, dado um diagrama UML;
  - (Vbt) representa uma classe ou interface que é uma variabilidade em uma arquitetura de ALP, dado um diagrama UML;
  - (Vc) representa um componente em uma arquitetura de ALP, dado um diagrama UML;

  Assim, é possível definir novas métricas de ALP a partir de métricas de classe básicas.

- **Cálculo das Métricas de Complexidade e Extensibilidade e Reuso** - uma vez calculadas as métricas básicas utilizando a ferramenta SDMetrics, ou através de cálculo não automatizado, é possível agrupar os resultados e obter os valores relativos às métricas de complexidade, extensibilidade e reuso. A métrica relativa à comlexidade considerada denominasse **CompPLA**, a métrica relatica à extensibilidade considerada denomina-se **ExtensPLA**, as métricas relativas à reuso consideradas denominam-se **DITPLA**,**NOCPLA** e **SIXPLA**. Maiores detalhes relativos às métricas utilizadas são apresentados a seguir.

  O detalhamento, exemplos e fórmulas para cálculo das métricas **CompPLA**

(Equação 5.23) e **ExtensPLA** (Equação 5.24) pode ser obtido em [47] - Capítulo 6. As Tabelas 5.6 e 5.7 podem facilitar a compreensão a respeito de sua obtençao, conforme formulação a seguir apresentadas

$$CompPLA(AP) = \sum_{i=1}^{NCpt} CompVarComponent(Cpt_i) \tag{5.23}$$

Onde:

- (AP) representa a arquitetura do produto configurado;
- $(NCpt)$ representa número de componentes presentes no produto configurado;
- $(CompVarComponent(Cpt_i))$ representa o valor considerado para complexidade do i-ésimo componente da AP;

$$ExtensPLA(AP) = \sum_{i=1}^{NCpt} ExtensVarComponent(Cpt_i) \tag{5.24}$$

Onde:

- (AP) representa a arquitetura do produto configurado;
- $(NCpt)$ representa número de componentes presentes no produto configurado;
- $(ExtensVarComponent(Cpt_i))$ representa o valor considerado para extensibilidade do i-ésimo componente da AP;

As métricas para avaliação de ALP (básicas e de atributos de qualidade), como definidas por [47] são anexadas no ANEXO D deste trabalho.

Além das métricas já propostas, e validadas, é possível derivar outras métricas para avaliação de Arquiteturas de Linhas de Produtos de Software baseadas na abordagem proposta por [47]. Assim, definiu-se as métricas DITPLA, NOCPLA e SIXPLA com o objetivo de avaliar a capacidade de reuso da plataforma. Assim, a equação 5.29 juntamente com a Tabela 5.8 definem a construção da métrica DITPLA; a equação 5.26 juntamente com a Tabela 5.9 definem a construção da métrica NOCPLA; a equação 5.27 juntamente com a Tabela 5.10 definem a construção da métrica SIXPLA;

$$DITPLA(AP) = \sum_{i=1}^{NCpt} DITVarComponent(Cpt_i) \tag{5.25}$$

Onde:

- (AP) representa a arquitetura do produto configurado;

Métrica	Descrição
CompInterface	Possui sempre o valor 0.0, pois não possui métodos concretos para computar a métrica *WMC - Weighted Methods per Class* de McCabe.
CompClass	O valor da métrica WMC para uma determinada classe.
CompVarPointClass	É o valor da métrica *CompClass* da classe que é uma << *variationPoint* >> mais a soma do valor da métrica *CompClass* de cada variante (ex: << *optional* >>) associada à classe.
CompVariabilityClass	É a soma da medida da métrica *CompVarPointClass* de cada ponto de variação de uma determinada variabilidade.
CompVarComponent	É a soma da medida da métrica *CompVariabilityClass* de cada classe que forma o componente.
CompPLA	Soma de todos os *CompVarComponent* da arquitetura

Tabela 5.6: Construção da métrica **CompPLA**

– ($NCpt$) representa número de componentes presentes no produto configurado;

– ($DITVarComponent(Cpt_i)$) representa o valor considerado para a métrica DIT do i-ésimo componente da AP;

$$NOCPLA(AP) = \sum_{i=1}^{NCpt} NOCVarComponent(Cpt_i) \qquad (5.26)$$

Onde:

– (AP) representa a arquitetura do produto configurado;

– ($NCpt$) representa número de componentes presentes no produto configurado;

– ($NOCVarComponent(Cpt_i)$) representa o valor considerado para a métrica NOC do i-ésimo componente da AP;

Métrica	Descrição
ExtensInterface	Possui sempre o valor 1, uma vez que as interfaces possuem 100% dos métodos abstratos.
ExtensClass	Fornece o percentual de métodos abstratos com relação ao total de métodos (abstratos e concretos) de uma classe.
ExtensVarPointClass	É o valor da métrica *ExtensClass* da classe que é uma << *variationPoint* >> mais a soma do valor da métrica *ExtensClass* de cada variante (ex: << *optional* >>) associada à classe.
ExtensVariabilityClass	É a soma da medida da métrica *ExtensVarPointClass* de cada ponto de variação de uma determinada variabilidade.
ExtensVarComponent	É a soma da medida da métrica *ExtensVariabilityClass* de cada classe que forma o componente.
ExtensPLA	Soma de todos os *ExtensVarComponent* da arquitetura

Tabela 5.7: Construção da métrica **ExtensPLA**

$$SIXPLA(AP) = \sum_{i=1}^{NCpt} SIXVarComponent(Cpt_i) \tag{5.27}$$

Onde:

- (AP) representa a arquitetura do produto configurado;
- $(NCpt)$ representa número de componentes presentes no produto configurado;
- $(SIXVarComponent(Cpt_i))$ representa o valor considerado para a métrica SIX do i-ésimo componente da AP;

Projeto Piloto: antes da execução do estudo real foi realizado um projeto piloto com vistas a avaliar a instrumentação que será utilizada no estudo de experimental. Para tanto, somente um executor foi considerado. O executor utilizou os mesmos instrumentos do estudo experimental. Os dados obtidos pelo estudo piloto não foram considerados para composição das dados e análises do estudo.

Métrica	Descrição
DITInterface	Possui sempre o valor 0.0
DITClass	O valor da métrica *DIT - Depth of Inheritance Tree* para uma determinada classe.
DITVarPointClass	É o valor da métrica *DITClass* da classe que é uma << *variationPoint* >> mais a soma do valor da métrica *DITClass* de cada variante (ex: << *optional* >>) associada à classe.
DITVariabilityClass	É a soma da medida da métrica *DITVarPointClass* de cada ponto de variação de uma determinada variabilidade.
DITVarComponent	É a soma da medida da métrica *DITVariabilityClass* de cada classe que forma o componente.
DITPLA	Soma de todos os *DITVarComponent* da arquitetura

Tabela 5.8: Construção da métrica **DITPLA**

Métrica	Descrição
NOCInterface	Possui sempre o valor 0.0
NOCClass	O valor da métrica *NOC - Number of Children* para uma determinada classe.
NOCVarPointClass	É o valor da métrica *NOCClass* da classe que é uma << *variationPoint* >> mais a soma do valor da métrica *NOCClass* de cada variante (ex: << *optional* >>) associada à classe.
NOCVariabilityClass	É a soma da medida da métrica *NOCVarPointClass* de cada ponto de variação de uma determinada variabilidade.
NOCVarComponent	É a soma da medida da métrica *NOCVariabilityClass* de cada classe que forma o componente.
NOCPLA	Soma de todos os *NOCVarComponent* da arquitetura

Tabela 5.9: Construção da métrica **NOCPLA**

Métrica	Descrição
SIXInterface	Possui sempre o valor 0.0
SIXClass	O valor da métrica *SIX - Specialization Index* para uma determinada classe.
SIXVarPointClass	É o valor da métrica *SIXClass* da classe que é uma << *variationPoint* >> mais a soma do valor da métrica *SIXClass* de cada variante (ex: << *optional* >>) associada à classe.
SIXVariabilityClass	É a soma da medida da métrica *SIXVarPointClass* de cada ponto de variação de uma determinada variabilidade.
SIXVarComponent	É a soma da medida da métrica *SIXVariabilityClass* de cada classe que forma o componente.
SIXPLA	Soma de todos os *SIXVarComponent* da arquitetura

Tabela 5.10: Construção da métrica **SIXPLA**

Formulação de Hipóteses: as seguintes hipóteses são propostas para o estudo experimental em questão:

Hipóteses Relativas à Complexidade:

- **Hipótese Nula H_0:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria na complexidade do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS.
- **Hipótese Alternativa 1 H_1:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria na complexidade do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS.
- **Hipótese Alternativa 2 H_2:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora na complexidade do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS.

Hipóteses Relativas à Extensibilidade:

- **Hipótese Nula H_0:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria na extensibilidade do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS.

- **Hipótese Alternativa 1** H_1**:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria na extensibilidade do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS.
- **Hipótese Alternativa 2** H_2**:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora na extensibilidade do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS.

Hipóteses Relativas ao Reuso:

- **Hipótese Nula** H_0**:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria no reuso do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS.
- **Hipótese Alternativa 1** H_1**:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria no reuso do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS.
- **Hipótese Alternativa 2** H_2**:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora no reuso do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS.

Variáveis Dependentes: Os atributos de qualidade de complexidade, extensibilidade e reuso da ALP.

Variáveis Independentes: Métricas de complexidade (**CompPLA**), extensibilidade (**ExtensPLA**) e reuso (**DITPLA**, **NOCPLA** e **SIXPLA**).

Análise Qualitativa: Tem por objetivo avaliar os resultados obtidos neste estudo, por meio de análise estatística. Objetivando, principalmente, identificar a rejeição, ou não, da hipótese nula apresentada anteriormente.

Mecanismos de Análise: o estudo proposto deve analisar a arquitetura proposta com o objetivo de avaliar a capacidade da arquitetura, sob um ponto de vista de modificabilidade de software, de manter níveis aceitáveis dos atributos de complexidade, extensibilidade e reuso, considerando uma amostra significativa de configurações possíveis de produto. Assim, os mecanismos de análise são :

- **Teste de Normalidade** o teste de normalidade de *Shapiro-Wilk* será aplicado aos valores observados para as ditribuições obtidas para cada produto

configurado.

- **Correlação** caso as distribuições de frequência dos valores sejam normais, será aplicada a correlação de *Pearson* para identificar a similaridade entre as váriaveis obtidas para cada produto configurado. Caso contrário utiliza-se a correlação de *Spearman*.
- **Regressão** caso exista correlação entre as váriaveis obtidas, deve-se aplicar a técnica de regressão linear para identificar a função capaz de projetar valores futuros dos atributos de qualidade a medida que a LPS evolua.

Validade Interna: Para garantir a validade interna do experimento será tomada a seguinte medida:

- os códigos utilizados para avaliação dizem respeito somente aos códigos oferecidos pela plataforma e o códigos extendidos pelo desenvolvedor. Isolando ao máximo possível os códigos avaliados de influência de experiência ou conhecimento em desenvolvimento com qualidade por parte do desenvolvedor.
- Uma amostra significativa de configurações (em acordo com a variabilidade da arquitetura) de produto terão suas métricas de complexidade, extensibilidade e resuo coletadas, com o objetivo de se obter confiabilidade estatística sobre os resultados obtidos.

Validade Externa: A validade externa do estudo experimental mede sua capacidade de refletir o comportamento em toda a população que utilizará a plataforma. Assim, a validade externa deste estudo é considerada suficiente uma vez que os métodos estatísticos utilizados observam a significância estatística para geração dos resultados.

Validade de Construção: A validade de construção do estudo se refere à relação entre os instrumentos participantes do estudo e a teoria que está sendo avaliada. Acredita-se que a validade de construção esteja garantida, uma vez que o executor utiliza ferramentas automáticas para coletar as métricas, bem como a análise dos resultados utilizará o apoio de valores pré-determinados e validados em experimentos anteriores.

Validade de Conclusão: A validade de conclusão mede a relação entre os tratamentos e os resultados, determinando a capacidade do estudo de generalizar os resultados. Acredita-se que a validade de conclusão poderá ser assegurada, uma vez que resultados com validade estatística tenham sido obtidos.

5.1.2.3 Execução do Estudo Experimental II

De acordo com a metodologia apresentada anteriormente, após a implementação dos códigos relativos ao estudo de caso (aplicação TVDeals - Figuras 5.7 e 5.8) baseados nos requisitos de software pré-estabelecidos (Tabela 5.11), seguindo o Método SystemPLA, é necessário definir os cenários de variabilidades das classes desenvoldidas.

Requisitos de Usuário	Descrição
RU1	O sistema deve disponibilizar, nos momentos adequados, promoções de produtos relativos ao conteúdo televisivo assistido.
RU2	O sistema deve permitir que o usuário compre o produto ofertado.
RU3	O sistema deve permitir que o usuário busque informações sobre o produto ofertado em dispositivos de alta resolução e mecanimos de interação homem-máquina de alta precisão.
RU4	O sistema deve permitir que o usuário coloque a oferta em espera (*waiting*) caso exista mecanismos de armazenamento remoto ou local.
RU5	O sistema deve permitir que o usuário descarte a oferta, permitindo o entendimento de desinteresse pelo produto.
RU6	O Sistema deve permitir ao usuário acessar o carrinho de compras, no qual deve ser apresentado o produto de interesse e a possibilidade de finalização do negócio.
RU7	O sistema deve permitir que o usuário modifique a quantidade de itens de produto comprado em dispositivos de alta resolução e mecanimos de interação homem-máquina de alta precisão.
RU8	O sistema deve permitir que o usuário receba recomendações de produtos similares ao interesse de compra em dispositivos de alta resolução e mecanimos de interação de alta precisão.
RU9	O sistema deve executar em equipamentos como televisores e computadores (dispositivos de alta resolução e mecanimos de interação homem-máquina de alta precisão), bem como executar em *smartphones*(disposistivos de baixa resolução e mecanimos de interação homem-máquina limitados).

Tabela 5.11: Requisitos de Usuários para o experimento II

Assim, a Figura 5.9 apresenta o diagrama de classes dos códigos, desenvolvidos no experimento, com suas variabilidades gerenciadas através da abordagem ***SMarty***.

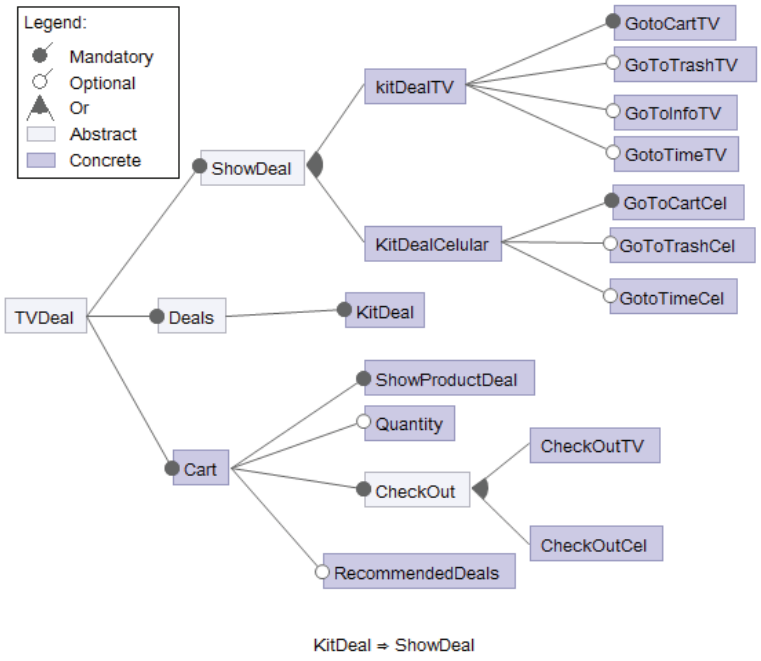

Figura 5.7: Aplicação TVDeals : *Feature Model*

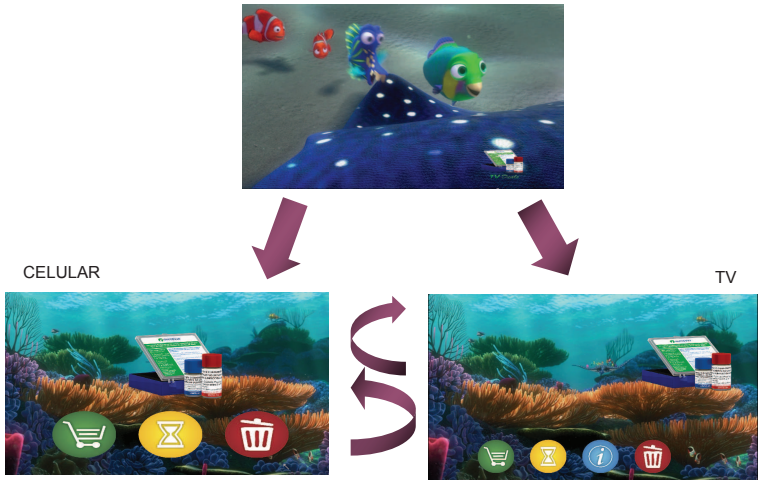

Figura 5.8: Aplicação Interativa TVDEAlS

Seguindo a metodologia proposta, faz-se necessário definir as configurações das ALPs utilizadas no estudo de caso.

No estudo de caso foram consideradas 8 (oito) configurações de produtos (exemplo: P1, P2, etc) a partir da LPS desenvolvida pelo desenvolvedor. As configurações de produtos consideradas são apresentadas nas Tabelas 5.12, 5.13 e 5.14, comunalidades do *framework*, variabilidades do *framework* e *features* da LPS desenvolvida a partir do *framework*, respectivamente.

Um vez definidas as configurações dos produtos de software, faz-se necessário coletar as

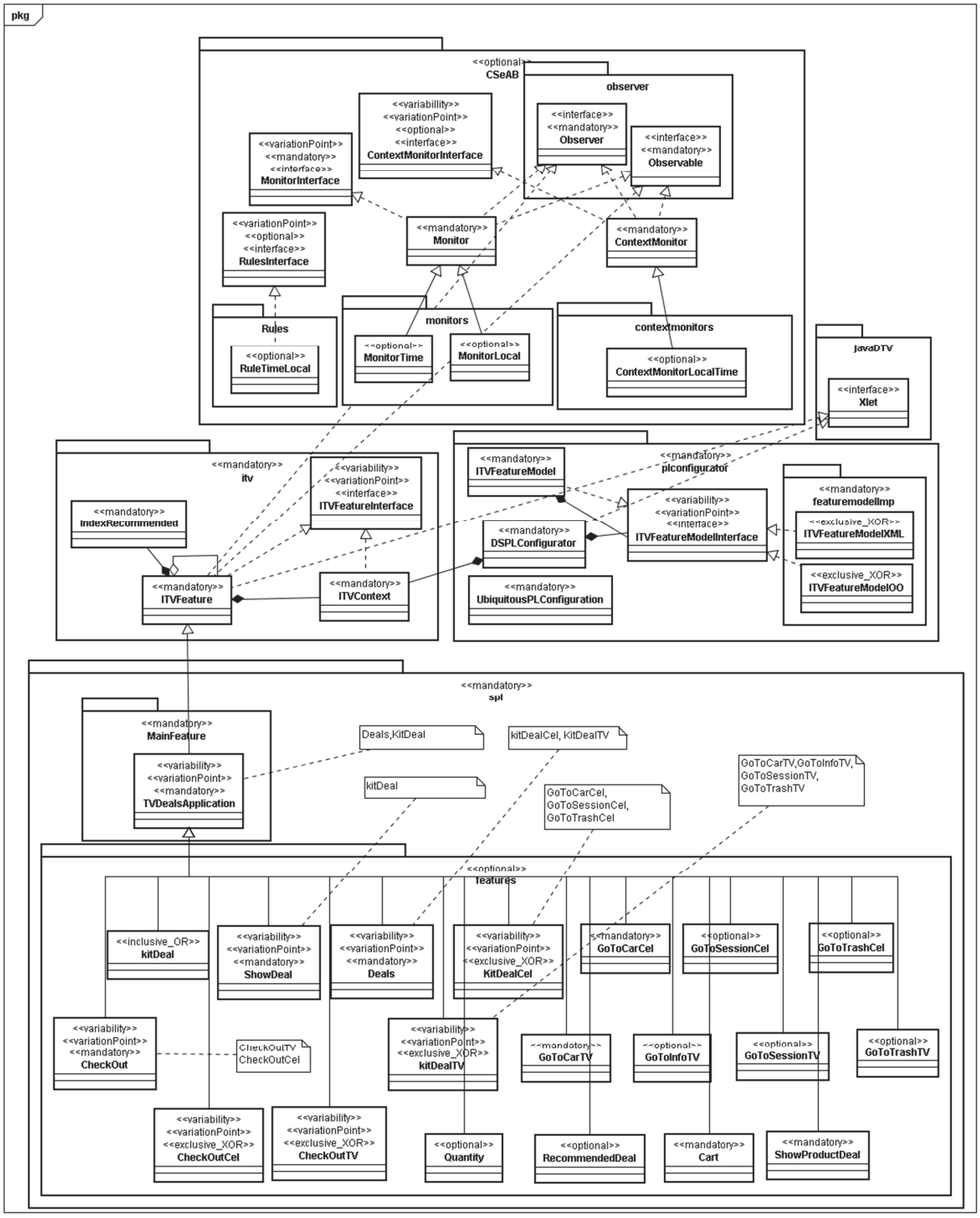

Figura 5.9: Diagrama de Classe Plataforma + TVDeals

métricas de qualidade para cada uma das configurações . A Tabela 5.15 apresenta os resultados obtidos para cada métrica relacionada a configuração do produto (Config.). Adicionalmente, obtem-se o percentual de *features* que são consideradas variabilidades dentro da linha (% VAR) desenvolvida pelo desenvolvedor, bem como a quantidade de

Features	P1	P2	P3	P4	P5	P6	P7	P8
ITVContext	X	X	X	X	X	X	X	X
ITVFeature	X	X	X	X	X	X	X	X
IndexRecommended	X	X	X	X	X	X	X	X
DSPLConfigurator	X	X	X	X	X	X	X	X
ITVFeatureModelInterface	X	X	X	X	X	X	X	X
ITVFeatureModel	X	X	X	X	X	X	X	X
ITVFeatureModelXML	X	X	X	X	X	X	X	X
UbiquitousPLConfiguration	X	X	X	X	X	X	X	X

Tabela 5.12: TVDeals - Core (Comunalidades do *Framework*)

Features	P1	P2	P3	P4	P5	P6	P7	P8
Observer	X	X	X	X	X	X	X	X
Observable	X	X	X	X	X	X	X	X
MonitorInterface	X	X	X	X	X	X	X	X
ContextMonitorInteface	X	X	X	X	X	X	X	X
RulesInteface	X	X	X	X	X	X	X	X
Monitor	X	X	X	X	X	X	X	X
ContextMonitor	X	X	X	X	X	X	X	X
ContextMonitorLocalTime	X	X	X	X	X	X	X	X
MonitorLocal	X	X	X	X	X	X	X	X
MonitorTime	X	X	X	X	X	X	X	X
RuleTimeLocal	X	X	X	X	X	X	X	X
ContextMonitorLocalTime	X	X	X	X	X	X	X	X
MonitorLocal	X	X	X	X	X	X	X	X
MonitorTime	X	X	X	X	X	X	X	X
RuleTimeLocal	X	X	X	X	X	X	X	X

Tabela 5.13: TVDeals - Variabilidades do *Framework*

features consideradas no produto configurado (QFeature).

Seguindo a metodologia proposta, a partir da coleta da métricas, deve-se analisar os dados. Assim, a seguir são analisados os resultados obtidos para cada métrica de qualidade considerada.

Antes de analisar cada métrica individualmente é necessário fazer testes de norma-

Features	P1	P2	P3	P4	P5	P6	P7	P8
TVDealsApplication	X	X	X	X	X	X	X	X
ShowDeal	X	X	X	X	X	X	X	X
kitDeal	X	X	X	X	X	X	X	X
Deals	X	X	X	X	X	X	X	X
kitDealCel					X	X	X	X
kitDealTV	X	X	X	X				
GoToCarCel					X	X	X	X
GoToSessionCel							X	
GoToTrashCel						X	X	
GoToCarTV	X	X	X	X				
GoToInfoTV		X		X				
GoToSessionTV			X	X				
GoToTrashTV		X		X				
Cart	X	X	X	X	X	X	X	X
ShowProductDeal	X	X	X	X	X	X	X	X
Quantity	X	X	X	X				
RecommendedDeals		X		X		X	X	
CheckOut	X	X	X	X	X	X	X	X
CheckOutTV	X	X	X	X				
CheckOutCel					X	X	X	X

Tabela 5.14: TVDeals - LPS desenvolvida a partir do *Framework*

Config.	CompPLA	ExtensPLA	DITPLA	SIXPLA	NOCPLA	% VAR	QFeature
1	225	7,2823	53	36,597	25	33,33	9
2	240	7,2823	62	45,597	25	50	12
3	230	7,2823	56	39,597	25	40	10
4	245	7,2823	65	48,597	25	53,85	13
5	220	7,2823	38	21,597	25	25	8
6	230	7,2823	56	24,597	25	40	10
7	235	7,2823	59	27,597	25	45,45	11
8	220	7,2823	50	21,597	25	33,33	9

Tabela 5.15: Valores das Métricas coletadas a partir da configurações de produtos definidas

lidade nas distribuições relativas aos percentuais de variabilidades (% VAR)), bem como a quantidade de *features* (QFeatures) consideradas em cada configuração de

produto. Sendo assim os resultados apresentaram distribuições normais, utilizando Shapiro-Wilk, segundo as Tabelas 5.16 e 5.17, respectivamente.

Shapiro-Wilk	0,971947347
P-Valor	0,912836096

Tabela 5.16: Teste de Normalidade aplicado a distribuição **% VAR**

Shapiro-Wilk	0,959010888
P-Valor	0,800626352

Tabela 5.17: Teste de Normalidade aplicado a distribuição **QFeature**

Métrica CompPLA: com relação à métrica **CompPLA** a amostra obtida é apresentada na tabela 5.15. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo a Tabela 5.18.

Shapiro-Wilk	0,941482828
P-Valor	0,6257269

Tabela 5.18: Teste de Normalidade aplicado a distribuição **CompPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições obteve-se os resultados apresentados nas tabelas 5.19 e 5.20.

CompPLA	**% VAR**	**QFeature**
1	0,9695302	0,98247214

Tabela 5.19: Matriz de Correlação da Métrica **CompPLA**

CompPLA	**% VAR**	**QFeature**
1	6,91148E-05	1,32862E-05

Tabela 5.20: Matriz de P-Valores da Métrica **CompPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.28 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}9654$.

$$Pd_{CompPLA} = 174,3562 + (\beta_1 * -9,7692) + (\beta_2 * 5,8720) \tag{5.28}$$

Onde:

- $Pd_{CompPLA}$ - representa os valores preditos para a métrica **CompPLA**
- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)
- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de *features*, bem como o percentual de variabilidades utilizado nos produtos, obter a predição da métrica CompPLA. Os resultados obtidos são apresentados nas figuras 5.10 e 5.11.

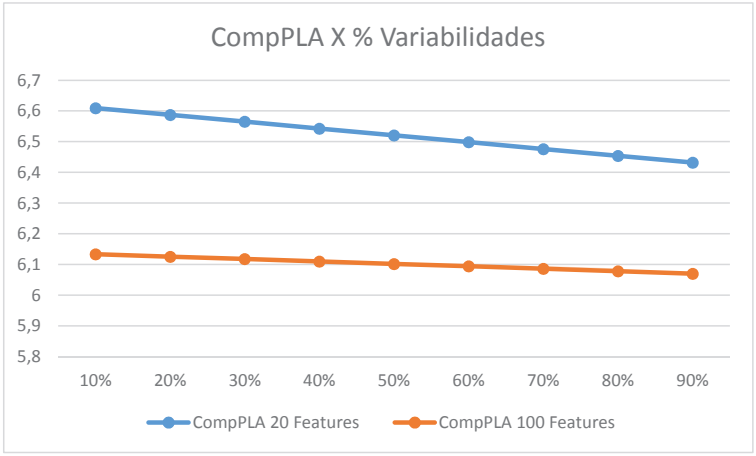

Figura 5.10: Valores da Métrica CompPLA X Percentual de Variabilidades - 20 e 100 Features

Pode-se tirar, através do gráfico representado pela figura 5.10, as seguintes conclusões:

- A quantidade de variabilidades utilizadas nos produtos configurados apresentam uma pequena queda na complexidade produto.
- A quantidade de *features*, quando aumentada, nos produtos configurados representa uma redução significativa na complexidade do produto.

Além disso, observa-se no gráfico representado pela figura 5.11 uma confirmação de que a quantidade de features utilizadas possuem impacto na complexidade do software, bem como pode melhorar a qualidade do atributo a medida que novas features forem acrescidas na LPS.

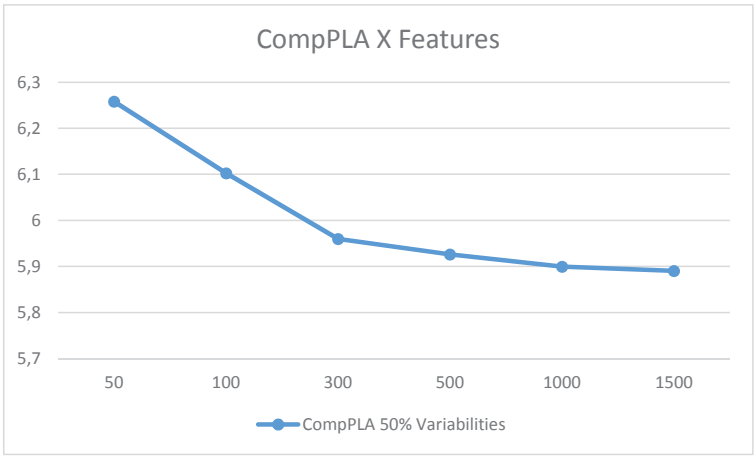

Figura 5.11: Valores da Métrica CompPLA - 50% of Variabilidades

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de complexidade.

ExtensPLA com relação à métrica **ExtensPLA** a amostra obtida é apresentada na Tabela 5.15. Sendo assim os testes de normalidade e correlação foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** Não foi possível realizar o teste de normalidade devido a igualdade dos valores obtidos em todas a configurações avaliadas. Entretanto, com pequenas variações nos valores utilizados, o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição não normal, segundo a Tabela 5.21.

Shapiro-Wilk	0,565940661
P-Valor	6,32298E-05

Tabela 5.21: Teste de Normalidade aplicado a distribuição **ExtensPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Spearman.

2. **Correlação Linear:** Calculando a Correlação de Spearman as distribuições obteve-se os resultados apresentados nas tabelas 5.22 e 5.23.

ExtensPLA	**% VAR**	**QFeature**
1	0,255030685	0,255030685

Tabela 5.22: Matriz de Correlação da Métrica **ExtensPLA**

O resultados apontam para uma fraca correlação entre as váriaveis comparadas, desencorajando a execução de regressão linear.

ExtensPLA	% VAR	QFeature
1	0,542147097	0,542147097

Tabela 5.23: Matriz de P-Valores da Métrica **ExtensPLA**

Apesar disso pode-se observar na tabela 5.15 que a métrica **ExtensPLA**, não varia entre os produtos configurados, ou seja, não sofre impacto da adição de valiabilidades, bem como a adição de features nos produtos derivados da LPS.

Assim, conclui-se que o *framework* avaliado não causa impactos significativos na qualidade de software em se tratando de extensibilidade.

Métrica DITPLA com relação à métrica **DITPLA** a amostra obtida é apresentada na Tabela 5.15. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo tabela 5.24.

Shapiro-Wilk	0,895493111
P-Valor	0,262993195

Tabela 5.24: Teste de Normalidade aplicado à distribuição **DITPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições obteve-se os resultados apresentados nas tabelas 5.25 e 5.26.

DITPLA	% VAR	QFeature
1	0,74268974	0,76610836

Tabela 5.25: Matriz de Correlação da Métrica **DITPLA**

DITPLA	% VAR	QFeature
1	0,034794135	0,02663901

Tabela 5.26: Matriz de P-Valores da Métrica **DITPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.29 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}5973$.

$$Pd_{DITPLA} = -15,2223 + (\beta_1 * -78,8522) + (\beta_2 * 9,4130) \tag{5.29}$$

Onde:

- Pd_{DITPLA} - representa os valores preditos para a métrica **DITPLA**
- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)
- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de features, bem como o percentual de variabilidades utilizado nos produtos, obter a predição da métrica DITPLA. Os resultados obtidos são apresentados nas figuras 5.12 e 5.13.

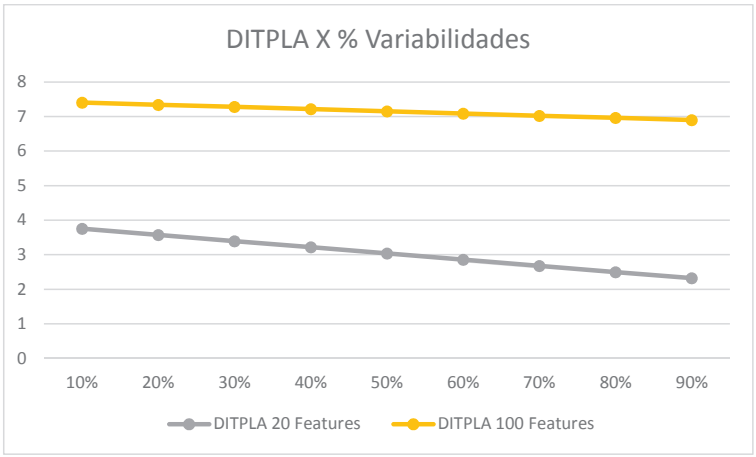

Figura 5.12: Valores da Métrica DITPLA X Percentual de Variabilidades - 20 e 100 Features

Pode-se tirar, através do gráfico representado pela figura 5.12, as seguintes conclusões:

- A quantidade de variabilidades uitilizadas nos produtos configurados apresentam uma pequena queda no reuso do produto, dado o aumento do percentual de variabilidades no produto.
- A quantidade de *features*, quando aumentada, nos produtos configurados, representa um aumento significativo no reuso do produto.

Além disso, observa-se no gráfico representado pela figura 5.13 uma confirmação de que a quantidade de features utilizadas possuem impacto no

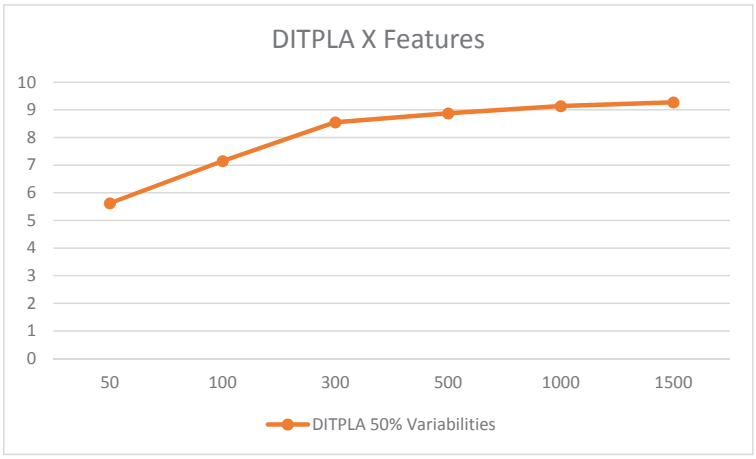

Figura 5.13: Valores da Métrica DITPLA - 50% of Variabilidades

reuso do software, bem como pode melhorar a qualidade do atributo a medida que novas features forem acrescidas na LPS.

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de reuso, no tocante a métrica DIT.

NOCPLA com relação a métrica **NOCPLA** a amostra obtida é apresentada na Tabela 5.15. Sendo assim os testes de normalidade e correlação foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** Não foi possível realizar o teste de normalidade devido a igualdade dos valores obtidos em todas a configurações avaliadas. Entretanto, com pequenas variações nos valores utilizados, o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição não normal, segundo a Tabela 5.27.

Shapiro-Wilk	0,418398447
P-Valor	1,04723E-06

Tabela 5.27: Teste de Normalidade aplicado a distribuição **NOCPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Spearman.

2. **Correlação Linear:** Calculando a Correlação de Spearman as distribuições obteve-se os resultados apresentados nas tabelas 5.28 e 5.29.

   O resultados apontam para uma fraca correlação entre as váriaveis comparadas, desencorajando a execução de regressão linear.

NOCPLA	% VAR	QFeature
1	0,333913548	0,333913548

Tabela 5.28: Matriz de Correlação da Métrica **NOCPLA**

NOCPLA	% VAR	QFeature
1	0,418893883	0,418893883

Tabela 5.29: Matriz de P-Valores da Métrica **NOCPLA**

Apesar disso pode-se observar na Tabela 5.15 que a métrica **NOCPLA** não varia entre os produtos configurados, ou seja, não sofre impacto da adição de valiabilidades, bem como a adição de *features* nos produtos derivados da LPS.

Assim, conclui-se que o *framework* avaliado não causa impactos significativos no reuso do software, em se tratando da métrica NOC.

SIXPLA com relação a métrica **SIXPLA** a amostra obtida é apresentada na Tabela 5.15. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo a Tabela 5.30.

Shapiro-Wilk	0,895493111
P-Valor	0,262993195

Tabela 5.30: Teste de Normalidade aplicado a distribuição **SIXPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições obteve-se os resultados apresentados nas Tabelas 5.31 e 5.32.

SIXPLA	% VAR	QFeature
1	0,74268974	0,76610836

Tabela 5.31: Matriz de Correlação da Métrica **SIXPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

SIXPLA	% VAR	QFeature
1	0,034794135	0,02663901

Tabela 5.32: Matriz de P-Valores da Métrica **SIXPLA**

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.30 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}597392923$.

$$Pd_{SIXPLA} = -31,6253 + (\beta_1 * -78,8522) + (\beta_2 * 9,4130) \tag{5.30}$$

Onde:

- Pd_{SIXPLA} - representa os valores preditos para a métrica **SIXPLA**
- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)
- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de *features*, bem como o percentual de variabilidades utilizando nos produtos, obter a predição da métrica SIXPLA. Os resultados obtidos são apresentados nas Figuras 5.14 e 5.15.

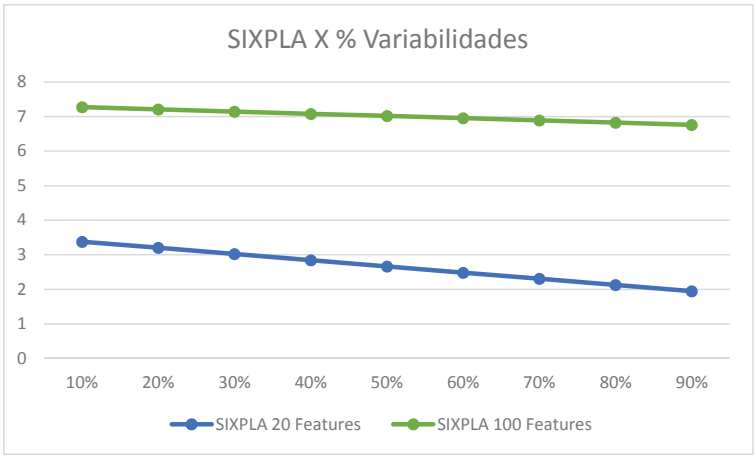

Figura 5.14: Valores da Métrica SIXPLA X Percentual de Variabilidades - 20 e 100 Features

Pode-se tirar, através do gráfico representado pela Figura 5.14, as seguintes conclusões:

- A quantidade de variabilidades utilizadas nos produtos configurados apresentam uma pequena queda no reuso do produto quando existe aumento na quantidade de variabilidades.

- A quantidade de *features*, quando aumentada, nos produtos configurados representa um aumento significativo no reuso do produto.

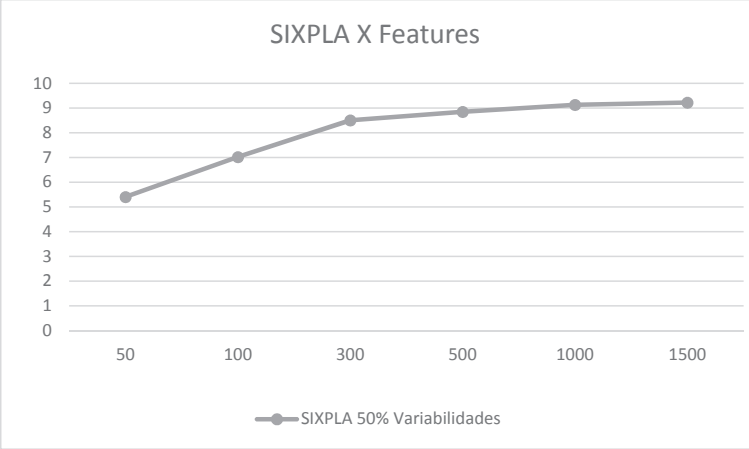

Figura 5.15: Valores da Métrica SIXPLA - 50% of Variabilidades

Além disso, observa-se no gráfico representado pela Figura 5.15 uma confirmação de que a quantidade de *features* utilizadas possuem impacto no reuso do software, bem como pode melhorar a qualidade do atributo a medida que novas features forem acrescidas na LPS.

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de reuso, com base na métrica *Specialization Index*.

5.1.3 Estudo Experimental III

5.1.3.1 Definição do Estudo Experimental III

Com base no *template* GQM [12], o objetivo do experimento III é apresentado a seguir:

Analisar métricas de complexidade, extensibilidade e reuso obtidas a partir da arquitetura dos produtos gerados através de uma SPL desenvolvida (Estudo de Caso) com base na Plataforma para desenvolvimento de SPLs proposta.

Com propósito de avaliação

Referente ao impacto causado aos atributos de qualidade de software , uma vez que a SPL seja manutenida, considerando a utilização de *features* sem recursos adicionais de controle, bem como considerando a utilização de *features* que se utilizam do suporte às características de softwares ubíquos também disponíveis na

plataforma proposta. Sendo que as características consideradas são: ciência de contexto, comportamento adaptável, onipresença e captura de experiências [57].

Do ponto de vista do Arquiteto/Desenvolvedor de Software.

No contexto de Aluno de Programa de Pós-Graduação em Engenharia de Software ou áreas correlatas.

5.1.3.2 Planejamento do Estudo Experimental III

Considera-se para o planejamento os itens a seguir, em que a itemização adotada segue, em boa parte, a sequência de itens usada em [47], com alteracoes e complementos julgados pertinentes.

Contexto Global: Avaliar a modificabilidade de uma arquitetura de uma Linha de Produtos de Software torna-se relevante pois tal avaliação permite a análise de distintas configurações do núcleo (*Core Assets*), bem como a evolução e o desenvolvimentos dos possíveis produtos [46] [47]. Sendo assim, avaliar a modificabilidade da arquitetura dos possíveis produtos de uma LPS oferece indicador de facilidade ou dificuldade de manter e até mesmo evoluir os produtos no futuro. Além disso, no que concerne a plataforma proposta, é importante identificar os impactos que as ferramentas, utilizadas para dar suporte ao desenvolvimento de software, causam na qualidade dos produtos desenvolvidos, tanto para o desenvolvimento inicial quanto para futuras evoluções. Além disso, torna-se importante avaliar, sob o ponto de vista de qualidade de software, se ferramentas que possuem suporte a certas características expecíficas de projeto, por exemplo ubiquidade, e se são capazes realmente de oferecer ambientes de desenvolvimento que permitam a evolução do software com qualidade.

Contexto Local: este estudo visa avaliar a complexidade, extensibilidade e reuso de um conjunto de produtos de software ubíqüos, desenvolvidos para TV Digital interativa, com base na abordagem de LPSD. Uma LPS será desensolvida como estudo de caso utilizando como ferramenta de apoio para o desenvolvimento da LPS, a plataforma proposta neste trabalho. Ao longo do desenvolvimento da LPS serão coletadas métricas de complexidade, extensibilidade e reuso para observação do impacto que a plataforma causa nos atributos de qualidade à medida que produtos de software, com quantidades diferentes de *features* são considerados, bem como quando funcionalidades relacionadas à ubiquidade são construídas no software.

Metodologia: O experimento deve ser realizado em acordo com o seguinte conjunto de tarefas por parte do executor:

1. Deve-se, a partir da plataforma proposta (seguindo o processo FOSD adaptado), construir uma aplicação para TV Digital interativa, baseada em requisitos de software pré-estabelecidos. No caso do experimento deve-se considerar algum serviço ubiquo dentre os requisitos de software;
2. A partir da implementação (ou projeto), com base na plataforma, deve-se seguir as seguintes etapas previstas no Método SystemPLA [47] (Figura 5.16):

Figura 5.16: Método SystemPLA - processo aplicado

Definir Cenários : Uma vez definido o modelo de classes da aplicação (em tempo de projeto ou implementação) é necessário gerenciar as variabilidades da Linha de Produtos. A abordagem adotada para o gerenciamento de variabilidade é a abordagem denominada ***SMarty*** [47], utilizando particularmente o modelo de variabilidade para classes. Os instrumentos necessários (SMarty Profile) para gerenciamento das variabilidades serão apresentados no item relativo à instrumentação;

Criar Configurações de ALP: Uma vez identificadas as varialibilidades da Linha é necessário definir as configurações dos produtos que serão avaliadas.

Coletar Métricas a partir das Configurações: Neste caso, antes de coletar as métricas relativas aos atributos de qualidade dos produtos configurados, torna-se necessário identificar as métricas básicas da ALP (as definições relativas as métricas básicas de ALP serão apresentados no item relativo à instrumentação). Adicionalmente, é necessário coletar as métricas considerando a abordagem dinâmica de configuração dos produtos.

Análise dos Dados: Os dados coletados devem seguir as orientações apresentadas no item abaixo para análise.

3. Uma vez definidas as configurações dos produtos e coletadas as métricas iniciais é necessário coletar as métricas de complexidade, extensibilidade e

reuso a partir de diferentes produtos configurados a partir da LPS. Os valores obtidos para as métricas devem gerar uma amostra de valores relativos a cada produto configurado.

4. Deve-se avaliar a normalidade das amostras, a fim de identificar qual método de correlação deverá ser utilizado.
5. Deve-se identificar a correlação linear entre as amostras coletadas (para cada métrica considerada) e o número de *features* que compõem o produto configurado, bem como percentual de variabilidades identificada em cada produto considerado. O objetivo de tal correlação é identificar se existe impacto, aos atributos de qualidade, para diferentes quantidades de *features* utilizadas nos produtos/LPS, bem como a quantidade de variabilidades utilizadas na linha, uma vez que existe a construção dinâmica de produtos e tal controle possui reflexos na construção dos códigos;
6. Supõe-se, a partir das características de extensão da plataforma proposta, que as amostras devem apresentar forte similaridade. Assim, confirmadas as correlações das amostras, deve-se aplicar a técnica de regressão linear as amostras. A partir da amostra selecionada deve-se obter uma função que permita projetar o desempenho da plataforma a medida que novas funcionalidades (*features*) e variabilidades forem adicionadas à LPS.
7. Uma vez definidas as funções deve-se fazer projeções de evolução da LPS e analisar a aceitação ou rejeição das hipóteses consideradas para o experimento.

Novamente, é importante destacar que as classes que devem ser avaliadas neste experimento limitam-se as classes do *Core Asset* da plataforma, bem como das classes extendidas (de fronteira) pelo desenvolvedor. As classes extendidas pelo desenvolvedor são controladas pela plataforma. Assim, isolando a parte relativa a plataforma dos códigos produzidos pelo desenvolvedor (para tratar de requisitos específicos), é possível identificar/projetar o impacto causado pela plataforma aos atributos de qualidade ao longo da evolução da LPS.

Instrumentação e Projeto Piloto: Os item relativos à Instrumentação e Projeto Piloto são similares aos mesmos itens previamente descritos no planejamento do estudo experimental II.

Formulação de Hipóteses: as seguintes hipóteses são propostas para o estudo experimental em questão:

Hipóteses Relativas à Complexidade:

- **Hipótese Nula H_0:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria na complexidade do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíqüos, consideradas no experimento, contribuem para melhoria da complexidade do software/LPS.
- **Hipótese Alternativa 1 H_1:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria na complexidade do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, não contribuem para melhoria da complexidade do software/LPS.
- **Hipótese Alternativa 2 H_2:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora na complexidade do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíqüos, consideradas no experimento, causam prejuízo à complexidade do software/LPS.

Hipóteses Relativas à Extensibilidade:

- **Hipótese Nula H_0:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria na extensibilidade do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, contribuem para melhoria da extensibilidade do software/LPS.
- **Hipótese Alternativa 1 H_1:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria na extensibilidade do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, não contribuem para melhoria da extensibilidade do software/LPS.
- **Hipótese Alternativa 2 H_2:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora na extensi-

bilidade do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, causam prejuízo à extensibilidade do software/LPS.

Hipóteses Relativas ao Reuso:

- **Hipótese Nula H_0:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam melhoria no reuso do código avaliado e, consequentemente, contribui para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, contribuem para melhoria do reuso do software/LPS.
- **Hipótese Alternativa 1 H_1:** O valores obtidos para a evolução da amostra, bem como para projeções futuras não apresentam melhoria no reuso do código avaliado e, consequentemente, não contribuem para melhoria da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, não contribuem para melhoria do reuso do software/LPS.
- **Hipótese Alternativa 2 H_2:** O valores obtidos para a evolução da amostra, bem como para projeções futuras apresentam piora no reuso do código avaliado e, consequentemente, contribuem para piora da qualidade de toda LPS. Além disso, a utilização dos recursos para implementação das características de software ubíquos, consideradas no experimento, causam prejuízo ao reuso do software/LPS.

Variáveis Dependentes, Independentes e Análise Qualitativa: os itens relativos às variáveis dependentes, variáveis independentes e análise qualitativa são similares aos itens descritos no planejamento do estudo experimental II.

Mecanismos de Análise: o estudo proposto deve analisar a arquitetura proposta com o objetivo de avaliar a capacidade da arquitetura, sob um ponto de vista de modificabilidade de software, de manter níveis aceitáveis dos atributos de complexidade, extensibilidade e reuso, considerando uma amostra significativa de configurações possíveis de produto quando características de softwares ubíquos são consideradas na implementação da LPS. Assim, os mecanismos de análise são:

- **Teste de Normalidade** o teste de normalidade de *Shapiro-Wilk* será aplicado aos valores observados para as ditribuições obtidas para cada produto configurado.
- **Correlação** caso as distribuições de frequência dos valores sejam normais, será aplicada a correlação de *Pearson* para identificar a similaridade entre as váriaveis obtidas para cada produto configurado. Caso contrário, utiliza-se a correlação de *Spearman*.
- **Regressão** caso exista correlação entre as váriaveis obtidas, deve aplicar a técnica de regressão linear para identificar a função capaz de projetar valores futuros dos atributos de qualidade a medida que a LPS evolua.

Validade Interna, Externa, Validade de Construção e Validade de Conclusão: Os itens de validade interna, externa, de construção e conclusão são similares aos itens previamentes descritos no planejamento do estudo experimental II.

5.1.3.3 Execução do Estudo Experimental III

De acordo com a metodologia apresentada anteriormente, após a implementação dos códigos relativos ao estudo de caso (aplicação TVDeals - Figuras 5.7 e 5.8), baseados nos requisitos de software pré-estabelecidos (Tabelas 5.11 e 5.33), seguindo o Método SystemPLA, é necessário definir o cenários de variabilidades das classes desenvoldidas.

Requisitos de Usuário	Descrição
RU10	O sistema deve obter informações de interesse de interatividade do usuário.
RU11	O sistema deve apresentar os elementos de interatividade da aplicação em acordo mecanismos de tomada de decisão.

Tabela 5.33: Requisitos de Usuários para o experimento III

Assim, a Figura 5.17 apresenta o diagrama de classes dos códigos, desenvolvidos no experimento, com suas variabilidades gerenciadas através da abordagem ***SMarty***. Cabe destacar o acréscimo de artefatos relativos ao suporte ao desenvolvimento de softwares ubíquos. Entretanto, as características de sensibilidade ao contexto e comportamento adaptável já estavam presentes no experimento anterior. Devido à necessidade de se avaliar o suporte à contrução de LPS dinâmicas, softwares cientes de contexto ou adaptativos não são considerados softwares ubíquos. No entanto, é requisito extremamente

relevante, aos softwares ubíquos, tais características [57]. Assim, pode-se observar o acréscimo de artefatos relativos à onipresença e captura de experiências no diagrama de classes.

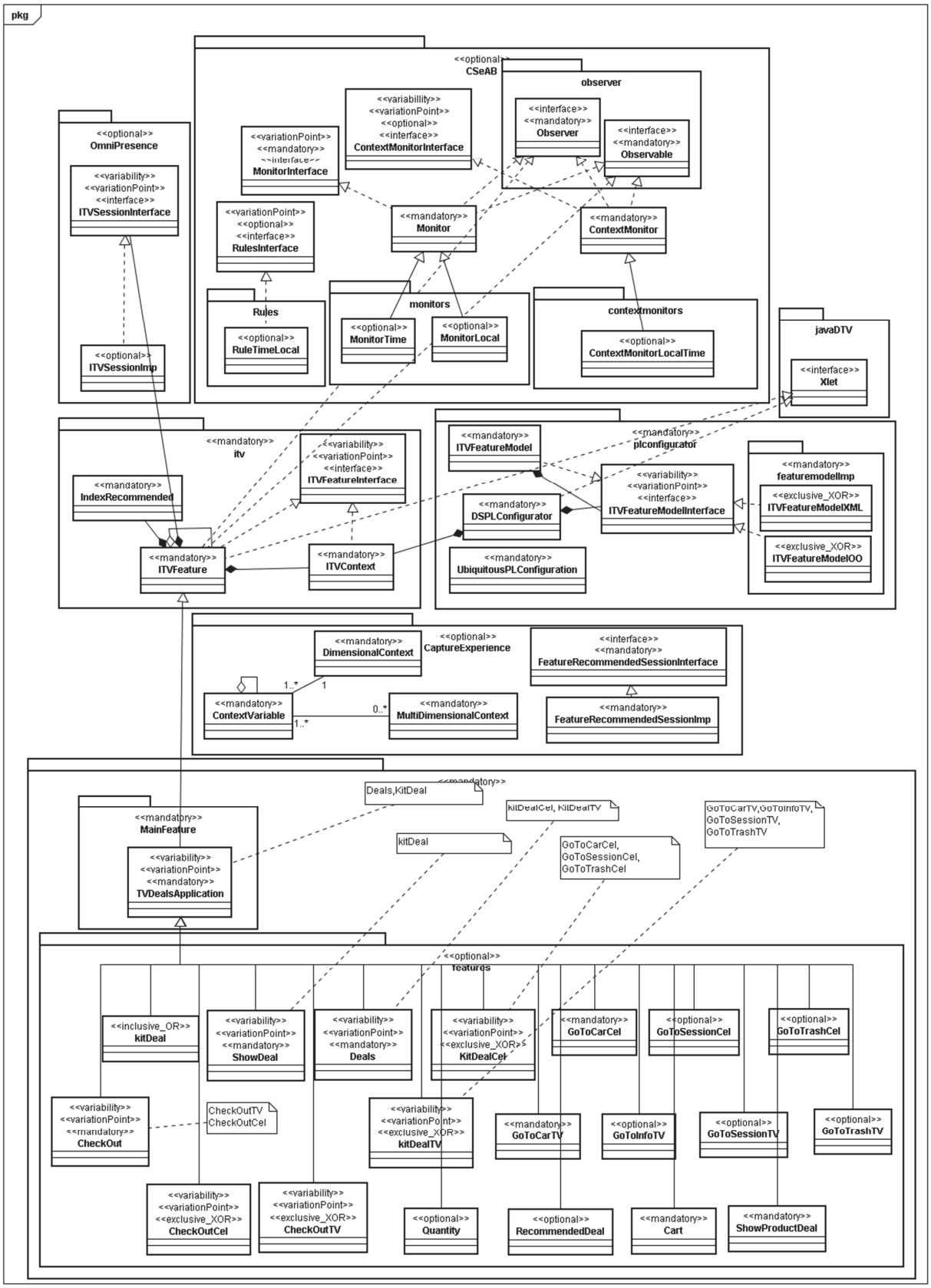

Figura 5.17: Diagrama de Classe Plataforma + TVDeals + Suporte à Ubiquidade

Seguindo a metodologia proposta, faz-se necessário definir as configurações das ALPs utilizadas no estudo de caso.

No estudo de caso foram consideradas 8 (oito) configurações de produtos (P1, P2, etc) a partir da LPS desenvolvida pelo desenvolvedor. As configurações de produtos consideradas são apresentadas nas Tabelas 5.34, 5.35 e 5.36, comunalidades do *framework*, variabilidades do *framework* e *features* da LPS desenvolvida a partir do *framework*, respectivamente.

Features	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
ITVContext	X	X	X	X	X	X	X	X
ITVFeature	X	X	X	X	X	X	X	X
IndexRecommended	X	X	X	X	X	X	X	X
DSPLConfigurator	X	X	X	X	X	X	X	X
ITVFeatureModelInterface	X	X	X	X	X	X	X	X
ITVFeatureModel	X	X	X	X	X	X	X	X
ITVFeatureModelXML	X	X	X	X	X	X	X	X
UbiquitousPLConfiguration	X	X	X	X	X	X	X	X

Tabela 5.34: TVDeals - Core (Comunalidades do *Framework*)

Um vez definidas as configurações dos produtos de software, faz-se necessário coletar as métricas de qualidade para cada uma das configurações . A Tabela 5.37 apresenta os resultados obtidos para cada métrica relacionada à configuração do produto (Config.). Adicionalmente, obtem-se o percentual de *features* que são consideradas variabilidades dentro da linha (% VAR) desenvolvida pelo desenvolvedor, bem como a quantidade de *features* consideradas no produto configurado (QFeature).

Comparando os resultados obtidos nos Experimentos II e III, observa-se que apenas as métricas CompPLA, DITPLA e ExtensPLA apresentarem resultados diferentes. Para o restante das métricas, a modificação do software não causou impacto algum.

Seguindo a metodologia proposta, a partir da coleta da métricas, deve-se analisar os dados. Assim, a seguir são analisados os resultados obtidos para cada métrica de qualidade considerada.

Antes de analisar cada métrica individualmente é necessário fazer testes de normalidade nas distribuições relativas aos percentuais de variabilidades (% VAR)), bem

Features	P1	P2	P3	P4	P5	P6	P7	P8
Observer	X	X	X	X	X	X	X	X
Observable	X	X	X	X	X	X	X	X
MonitorInterface	X	X	X	X	X	X	X	X
ContextMonitorInteface	X	X	X	X	X	X	X	X
RulesInteface	X	X	X	X	X	X	X	X
Monitor	X	X	X	X	X	X	X	X
ContextMonitor	X	X	X	X	X	X	X	X
ContextMonitorLocalTime	X	X	X	X	X	X	X	X
MonitorLocal	X	X	X	X	X	X	X	X
MonitorTime	X	X	X	X	X	X	X	X
RuleTimeLocal	X	X	X	X	X	X	X	X
ContextMonitorLocalTime	X	X	X	X	X	X	X	X
MonitorLocal	X	X	X	X	X	X	X	X
MonitorTime	X	X	X	X	X	X	X	X
RuleTimeLocal	X	X	X	X	X	X	X	X
ITVSessionInteface	X	X	X	X	X	X	X	X
ITVSessionImp	X	X	X	X	X	X	X	X
ITVFeatureInteface	X	X	X	X	X	X	X	X
ContextVariable	X	X	X	X	X	X	X	X
DimensionalContext	X	X	X	X	X	X	X	X
MultiDimensionalContext	X	X	X	X	X	X	X	X
FeatureRecommendedSessionInterface	X	X	X	X	X	X	X	X
FeatureRecommendedSessionImp	X	X	X	X	X	X	X	X

Tabela 5.35: TVDeals - Variabilidades do *Framework*

como a quantidade de *features* (QFeatures) consideradas em cada configuração de produto. Sendo assim os resultados apresentaram distribuições normais, utilizando Shapiro-Wilk, segundo as Tabelas 5.38 e 5.39, respectivamente.

Métrica CompPLA: com relação à métrica **CompPLA** a amostra obtida é apresentada na tabela 5.37. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo tabela 5.40.

Features	P1	P2	P3	P4	P5	P6	P7	P8
TVDealsApplication	X	X	X	X	X	X	X	X
ShowDeal	X	X	X	X	X	X	X	X
kitDeal	X	X	X	X	X	X	X	X
Deals	X	X	X	X	X	X	X	X
kitDealCel					X	X	X	X
kitDealTV	X	X	X	X				
GoToCarCel					X	X	X	X
GoToSessionCel							X	
GoToTrashCel						X	X	
GoToCarTV	X	X	X	X				
GoToInfoTV		X		X				
GoToSessionTV			X	X				
GoToTrashTV		X		X				
Cart	X	X	X	X	X	X	X	X
ShowProductDeal	X	X	X	X	X	X	X	X
Quantity	X	X	X	X				
RecommendedDeals		X		X		X	X	
CheckOut	X	X	X	X	X	X	X	X
CheckOutTV	X	X	X	X				
CheckOutCel					X	X	X	X

Tabela 5.36: TVDeals - LPS desenvolvida a partir do *Framework*

Config.	CompPLA	ExtensPLA	DITPLA	SIXPLA	NOCPLA	% VAR	QFeature
1	256	9,2823	58	36,597	25	33,33	9
2	274	9,2823	67	45,597	25	50	12
3	262	9,2823	61	39,597	25	40	10
4	280	9,2823	70	48,597	25	53,85	13
5	251	9,2823	55	21,597	25	25	8
6	263	9,2823	61	24,597	25	40	10
7	269	9,2823	64	27,597	25	45,45	11
8	251	9,2823	55	21,597	25	33,33	9

Tabela 5.37: Valores das Métricas coletadas a partir da configurações de produtos definidas

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições

Shapiro-Wilk	0,971947347
P-Valor	0,912836096

Tabela 5.38: Teste de Normalidade aplicado a distribuição **% VAR**

Shapiro-Wilk	0,959010888
P-Valor	0,800626352

Tabela 5.39: Teste de Normalidade aplicado a distribuição **QFeature**

Shapiro-Wilk	0,94112883
P-Valor	0,622220487

Tabela 5.40: Teste de Normalidade aplicado a distribuição **CompPLA**

obteve-se os resultados apresentados nas tabelas 5.41 e 5.42.

CompPLA	**% VAR**	**QFeature**
1	0,970305161	0,982968208

Tabela 5.41: Matriz de Correlação da Métrica **CompPLA**

CompPLA	**% VAR**	**QFeature**
1	6,40118E-05	1,21943E-05

Tabela 5.42: Matriz de P-Valores da Métrica **CompPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.31 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}9664$.

$$Pd_{CompPLA} = 197,6140 + (\beta_1 * -9,9785) + (\beta_2 * 6,7940) \tag{5.31}$$

Onde:

- $Pd_{CompPLA}$ - representa os valores preditos para a métrica **CompPLA**
- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)

- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de *features*, bem como o percentual de variabilidades utilizado nos produtos, obter a predição da métrica CompPLA. Os resultados obtidos são apresentados nas Figuras 5.18 e 5.19.

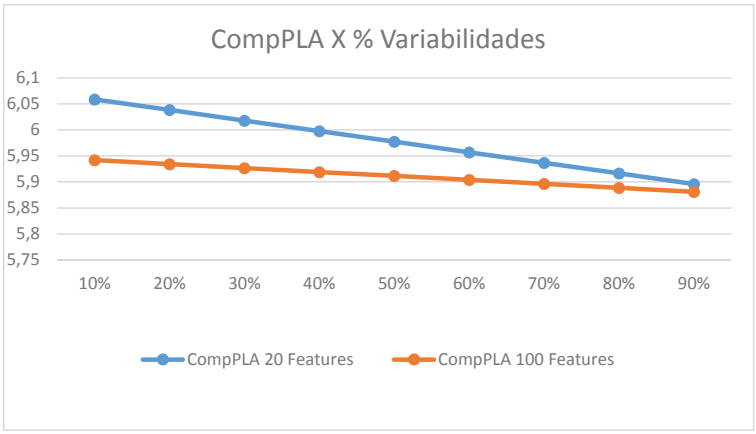

Figura 5.18: Valores da Métrica CompPLA X Percentual de Variabilidades - 20 e 100 Features

Pode-se tirar, através do gráfico representado pela Figura 5.18, as seguintes conclusões:

- A quantidade de variabilidades utilizadas nos produtos configurados apresentam uma pequena queda na complexidade produto.
- A quantidade de *features*, quando aumentada, nos produtos configurados representa uma redução significativa na complexidade do produto.

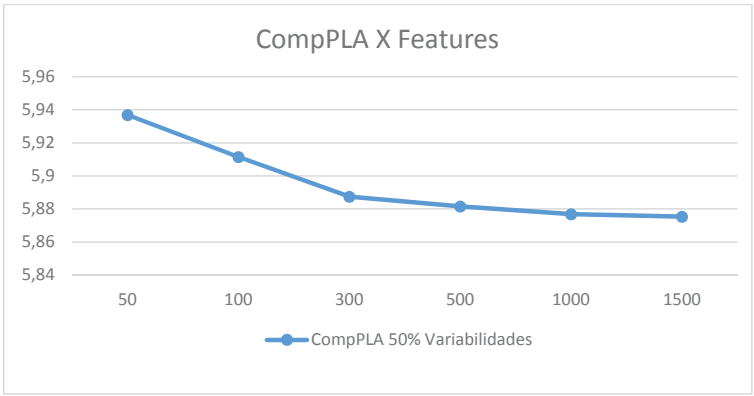

Figura 5.19: Valores da Métrica CompPLA - 50% of Variabilidades

Além disso, observa-se no gráfico representado pela Figura 5.19 uma confirmação de que a quantidade de *features* utilizadas possuem impacto na complexidade do software, bem como pode melhorar a qualidade do atributo a medida que novas *features* forem acrescidas na LPS.

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de complexidade.

ExtensPLA com relação à métrica **ExtensPLA** a amostra obtida é apresentada na Tabela 5.37. Sendo assim os testes de normalidade e correlação foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** Não foi possível realizar o teste de normalidade devido à igualdade dos valores obtidos em todas a configurações avaliadas. Entretanto, com pequenas variações nos valores utilizados, o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição não normal, segundo a Tabela 5.43.

Shapiro-Wilk	0,418398447
P-Valor	1,04723E-06

Tabela 5.43: Teste de Normalidade aplicado a distribuição **ExtensPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Spearman.

2. **Correlação Linear:** Calculando a Correlação de Spearman as distribuições obteve-se os resultados apresentados nas Tabelas 5.44 e 5.45.

ExtensPLA	**% VAR**	**QFeature**
1	0,333913548	0,33391354

Tabela 5.44: Matriz de Correlação da Métrica **ExtensPLA**

ExtensPLA	**% VAR**	**QFeature**
1	0,418893883	0,418893883

Tabela 5.45: Matriz de P-Valores da Métrica **ExtensPLA**

O resultados apontam para uma fraca correlação entre as váriaveis comparadas, desencorajando a execução de regressão linear.

Apesar disso pode-se observar na Tabela 5.37 que a métrica **ExtensPLA** não varia entre os produtos configurados, ou seja, não sofre impacto da adição de variabilidades, bem como a adição de *features* nos produtos derivados da LPS.

Assim, conclui-se que o *framework* avaliado não causa impactos significativos na qualidade de software em se tratando de extensibilidade.

Métrica DITPLA com relação à métrica **DITPLA** a amostra obtida é apresentada na Tabela 5.37. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo a Tabela 5.46.

Shapiro-Wilk	0,941482828
P-Valor	0,625727

Tabela 5.46: Teste de Normalidade aplicado a distribuição **DITPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições obteve-se os resultados apresentados nas Tabelas 5.47 e 5.48.

DITPLA	**% VAR**	**QFeature**
1	0,9695302	0,98247214

Tabela 5.47: Matriz de Correlação da Métrica **DITPLA**

DITPLA	**% VAR**	**QFeature**
1	6,91148E-05	1,32862E-05

Tabela 5.48: Matriz de P-Valores da Métrica **DITPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.32 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}9654$.

$$Pd_{DITPLA} = 27,6137 + (\beta_1 * -5,8615) + (\beta_2 * 3,5232) \tag{5.32}$$

Onde:

- Pd_{DITPLA} - representa os valores preditos para a métrica **DITPLA**
- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)

- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de *features*, bem como o percentual de variabilidades utilizando nos produtos, obter a predição da métrica DITPLA. Os resultados obtidos são apresentados nas Figuras 5.20 e 5.21.

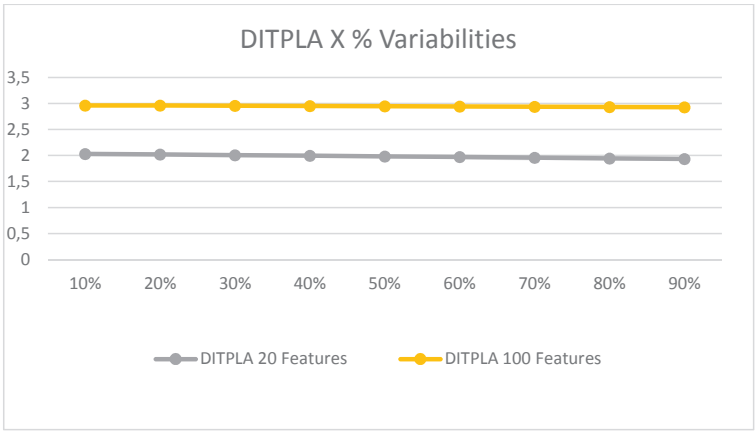

Figura 5.20: Valores da Métrica DITPLA X Percentual de Variabilidades - 20 e 100 Features

Pode-se tirar, através do gráfico representado pela Figura 5.20, as seguintes conclusões:

- A quantidade de variabilidades utilizadas nos produtos configurados apresentam uma pequena queda no reuso do produto, dado o aumento do percentual de variabilidades no produto.
- A quantidade de *features*, quando aumentada, nos produtos configurados representa um aumento significativo no reuso do produto.

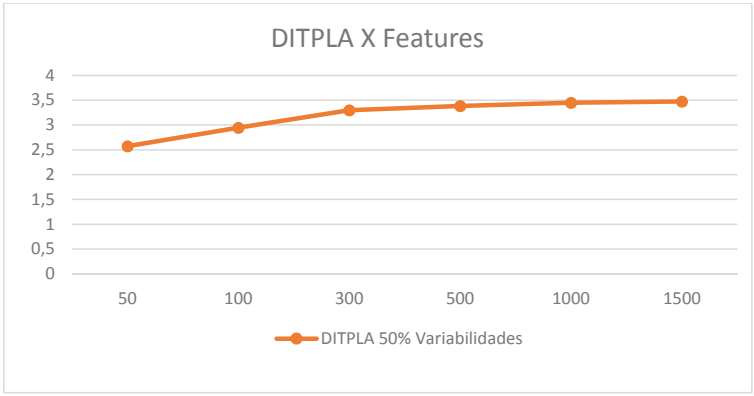

Figura 5.21: Valores da Métrica DITPLA - 50% of Variabilidades

Além disso, observa-se no gráfico representado pela Figura 5.21 uma confirmação de que a quantidade de *features* utilizadas possuem impacto no

reuso do software, bem como pode melhorar a qualidade do atributo a medida que novas *features* forem acrescidas na LPS.

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de reuso, no tocante a métrica DIT.

NOCPLA com relação à métrica **NOCPLA** a amostra obtida é apresentada na Tabela 5.37. Sendo assim os testes de normalidade e correlação foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** Não foi possível realizar o teste de normalidade devido à igualdade dos valores obtidos em todas a configurações avaliadas. Entretanto, com pequenas variações nos valores utilizados, o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição não normal, segundo a Tabela 5.49.

Shapiro-Wilk	0,418398447
P-Valor	1,04723E-06

Tabela 5.49: Teste de Normalidade aplicado a distribuição **NOCPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Spearman.

2. **Correlação Linear:** Calculando a Correlação de Spearman as distribuições obteve-se os resultados apresentados nas Tabelas 5.50 e 5.51.

NOCPLA	**% VAR**	**QFeature**
1	0,333913548	0,333913548

Tabela 5.50: Matriz de Correlação da Métrica **NOCPLA**

NOCPLA	**% VAR**	**QFeature**
1	0,418893883	0,418893883

Tabela 5.51: Matriz de P-Valores da Métrica **NOCPLA**

O resultados apontam para uma fraca correlação entre as váriaveis comparadas, desencorajando a execução de regressão linear.

Apesar disso pode-se observar na Tabela 5.37 que a métrica **NOCPLA** não varia entre os produtos configurados, ou seja, não sofre impacto da adição de variabilidades, bem como a adição de *features* nos produtos derivados da LPS.

Assim, conclui-se que o *framework* avaliado não causa impactos significativos no reuso do software, em se tratando da métrica NOC.

SIXPLA com relação à métrica **SIXPLA** a amostra obtida é apresentada na Tabela 5.37. Sendo assim os testes de normalidade, correlação e regressão foram realizados obtendo-se os seguintes resultados:

1. **Teste de Normalidade:** o teste de normalidade de Shapiro-wilk foi realizado obtendo-se a informação de distribuição normal, segundo a Tabela 5.52.

Shapiro-Wilk	0,895493111
P-Valor	0,262993195

Tabela 5.52: Teste de Normalidade aplicado a distribuição **SIXPLA**

Sendo assim, a correlação linear aplicada deve ser a Correlação de Pearson.

2. **Correlação Linear:** Calculando a Correlação de Pearson as distribuições obteve-se os resultados apresentados nas Tabelas 5.53 e 5.54.

SIXPLA	**% VAR**	**QFeature**
1	0,74268974	0,76610836

Tabela 5.53: Matriz de Correlação da Métrica **SIXPLA**

SIXPLA	**% VAR**	**QFeature**
1	0,034794135	0,02663901

Tabela 5.54: Matriz de P-Valores da Métrica **SIXPLA**

O resultados apontam para uma forte correlação entre as váriaveis comparadas, permitindo assim que a regressão linear seja executada.

3. **Regressão Linear:** a partir da regressão linear executada obteve-se a função apresentada na equação 5.33 com invervalo de confiança $IC = 0{,}95$ e $R^2 = 0{,}597392923$.

$$Pd_{SIXPLA} = -31,6253 + (\beta_1 * -78,8522) + (\beta_2 * 9,4130) \tag{5.33}$$

Onde:

- Pd_{SIXPLA} - representa os valores preditos para a métrica **SIXPLA**

- β_1 - representa o percentual de variabilidades presentes no produto configurado (% VAR)
- β_2 - representa a quantidade de *features* presentes no produto configurado (QFeatures)

Assim, foi possível, variando a quantidade de *features*, bem como o percentual de variabilidades utilizando nos produtos, obter a predição da métrica SIXPLA. Os resultados obtidos são apresentados nas Figuras 5.22 e 5.23.

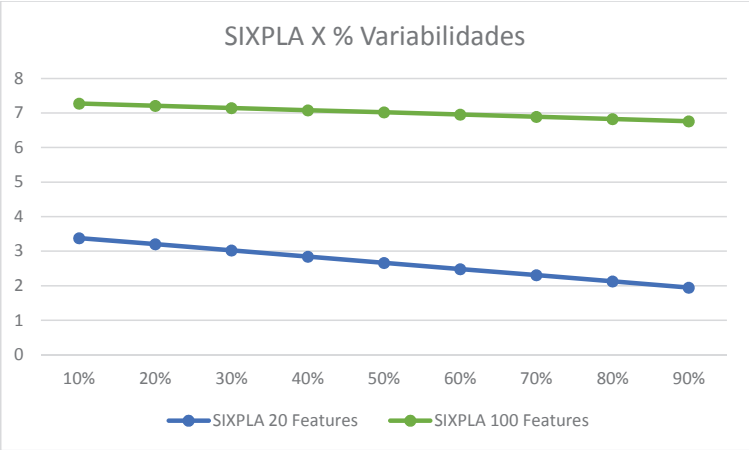

Figura 5.22: Valores da Métrica SIXPLA X Percentual de Variabilidades - 20 e 100 *Features*

Pode-se tirar, através do gráfico representado pela figura 5.22, as seguintes conclusões:

- A quantidade de variabilidades utilizadas nos produtos configurados apresentam uma pequena queda no reuso do produto quando existe aumento na quantidade de variabilidades.
- A quantidade de *features*, quando aumentada, nos produtos configurados representa um aumento significativo no reuso do produto.

Além disso, observa-se no gráfico representado pela Figura 5.23 uma confirmação de que a quantidade de *features* utilizadas possuem impacto no reuso do software, bem como pode melhorar a qualidade do atributo a medida que novas *features* forem acrescidas na LPS.

Assim, conclui-se que o *framework* avaliado é capaz de melhorar a qualidade de software em se tratando de reuso, com base na métrica *Specialization Index*.

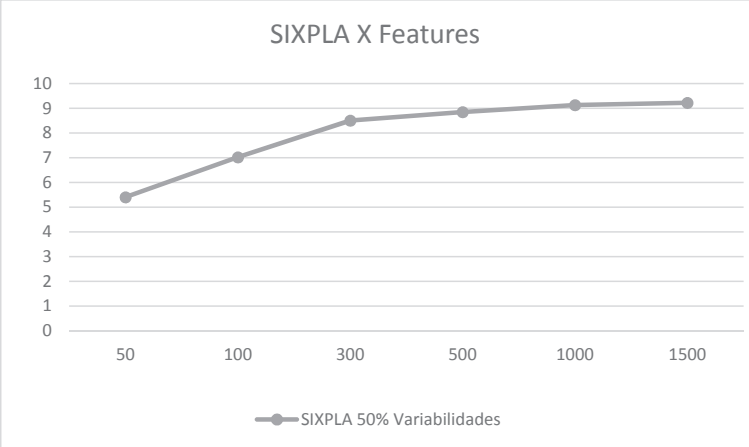

Figura 5.23: Valores da Métrica SIXPLA - 50% of Variabilidades

5.2 Considerações Relativas aos Resultados dos Experimentos

Resultados do Experimento I: os resultados obtidos no experimento I demonstram que utilização do *framework* proposto, bem como a organização de aplicações através de LPS, podem melhorar atributos de qualidade de software como complexidade e extensibiliddade quando comparadas a programação tradicional para TV Digital (usando orientação a objetos).

Resultados do Experimento II: os resultados obtidos no experimento II demonstram que o *framework* proposto é capaz de melhorar a qualidade de software quando atributos de qualidade de complexidade e reuso são considerados durante a evolução da LPS. Além disso, demonstra que dadas as características particulares de projeto do *framework*, o mesmo não interfere no atributo de qualidade relativo à extensibilidade durante a evolução da LPS, apesar de apresentar resultados interessantes relativos à extensibilidade do software avaliado.

Resultados do Experimento III: os resultados obtidos no experimento III apresentam algumas diferenças com relação aos resultados obtidos no experimento II, a partir do momento que o suporte à características de projetos ubíquos são inseridas no estudo de caso. As métricas CompPLA e DITPLA apresentaram um pequena piora com relação aos resultados obtidos no experimento II. Entretanto, ao analisar a evolução da LPS, percebe-se que o comportamento de melhoria dos valores obtidos permanece no experimento III. Enfim, a métrica ExtensPLA apresenta melhora com relação aos resultados obtidos no experimento II, mantendo o mesmo comportamento relativo à evolução da LPS.

Considerações Gerais: Ao final dos experimentos os resultados obtidos demosntra-

ram que o principal objetivo de uma LPS (que é melhorar o reuso de software) pode ser obtido utilizando o *framework* para desenvolvimento de LPSs para TV Digital interativa. Além disso, observa-se que obtem-se ganhos com relação à complexidade do software (fator importante para manutenção da LPS), bem como os resultados relativos à extensibilidade não oferecem prejuízo a evolução/manutenção do software.

Com relação ao suporte a características de software ubíquo, o *framework* apresentou implementação de suporte a algumas características importantes, que foram utilizadas no estudo de caso considerado nos experimentos I e II.

6 CONCLUSÕES E TRABALHOS FUTUROS

6.1 Conclusões

Este trabalho aponta, dentre os principais problemas para o desenvolvimento de TVDi, a necessidade de soluções para o desenvolvimento de software que ofereçam condições adequadas para manutenibilidade e, consequente, evolução do software. Neste sentido este trabalho propos e implementou um *framework* para desenvolvimento de LPS para TVDi de forma que o *framework*, ao ser utilizado, permita que o desenvolvedor possa manutenir o software sem prejuizos ou impactos quanto a atributos de qualidade relativos a manutenibilidade, com foco em modificabilidade.

Além disso, outro problema foi considerado relevante, relativo à necessidade de softwares capazes de se adaptar ao ambiente no qual serão executados (devido à recepção do aplicativo através de um canal *broadcast*), também foi tratado no trabalho. Assim, através de uma abordagem de Linhas de Produtos de Software Dinâmicas (que permite a seleção de *features* em tempo de execução) bem como suporte a implementação de características de softwares ubíquos como sensibilidade ao contexto e comportamento adaptável do software, foi possível apresentar soluções para ambas as necessidades destacadas no início do trabalho.

Para atender as necessidades identificadas no início do trabalho foram traçados alguns objetivos. O primeiro objetivo considerado foi a implementação de uma versão básica do *framework* que permitisse o desenvolvimento de LPSs que se comportassem, quanto à seleção do produto, de forma dinâmica, permitindo assim adaptação do software. Neste sentido, após a implementação da versão básica do *framework*, avaliações quanto a evolução de atributos de qualidade como complexidade, e extensibilidade e reuso foram realizadas a partir de um estudo de caso, no qual obteve-se resultados que evidenciam a capacidade da solução proposta de manter adequado o nível de manutenibilidade da aplicação desenvolvida.

O segundo objetivo traçado foi relativo à extensão do *framework* básico no tocante ao suporte de características de softwares ubíquos, permitindo que a adaptabilidade do

software construído tratasse características básicas para tomada de decisão, principalmente relacionadas ao ambiente de execução. Neste sentido, experimentos avaliando novamente, os atributos de qualidade relacionados a manutenibilidade foram realizados obtendo-se resultados significativos e positivos.

O terceiro objetivo, que trata do aperfeiçoamento da tomada de decisão para seleção de *features*, foi alcançado através da implementação de um modelo multidimensional de informações, que permite que informações contextuais fossem relacionadas com informações de preferência pessoal do usuário. Neste caso, não foram feitas avaliações quanto a acurácia da tomada de decisão, pois o modelo implementado foi proposto e validado anteriormente.

Finalmente, após as implementações e os experimentos realizados, conclui-se que o *framework* permite a construção, bem como a modificabilidade, de Linhas de produtos de software para TV Digital interativa através de um ambiente de execução dinâmica. Os resultados obtidos nos experimentos demonstram que, à medida que novas *features* são adicionadas a LPS, a maioria das métricas de qualidade avaliadas melhoram seu desempenho. Assim, os objetivos e contribuições previstas para tese foram alcançados.

6.2 Trabalhos Futuros

Identificou-se ao longo do desenvolvimento do trabalho um conjunto de novos caminhos a serem dados para pesquisa. Assim, alguns dos principais trabalhos futuros considerados são citados abaixo:

- Com relação a avaliação do *framework*, avaliar outros características de qualidade como, por exemplo, eficiência e usabilidade.
- Com relação a abordagem utilizada para o desenvovimento do *framework*, considerar a integração de orientação a aspectos como parte da solução pode melhorar a qualidade dos produtos construídos.
- Com relação ao processo de desenvolvimento, construir uma solução dirigida a modelos, através da geração e configuração de *features*, bem como da LPS, utilizando ferramentas para definição de modelos de *features*, por exemplo.

- Com relação à TV Digital Interativa, particularmente relacionada ao padrão nacional, integrar do *framework* ao desenvolvimento declarativo, possibilitando assim a contrução de *features* a partir de NCL/Lua.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Brasil 4d - estudo de impacto socio-econÔmico sobre a tv digital publica interativa. url: (http://www.ebc.com.br/sites/default/files/brasil_4d.pdf), acessada em julho de 2014.

[2] Java tv api. url:(http://www.oracle.com/technetwork/java/javame/tech/index-jsp-138011.html), acessado em novembro de 2014.

[3] Metrics software. url:(metrics.sourceforge.net), acessado em novembro de 2014.

[4] Openginga project. url:(gingacdn.lavid.ufpb.br), acessado em novembro de 2014.

[5] Smtvi - serviços multiplataforma de tv interativa. url: (http://www.cpqd.com.br/pesquisa-desenvolvimento/tv-interativa), acessada em julho de 2014.

[6] Wireless brasil. url: (http://wirelessbrasil.org/bloco/websites_tecnologia/tv_digital_interatividade_ginga/interatividade_indice_artigos_noticias.html), acessada em julho de 2014.

[7] Abowd, G. D. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering*, ICSE '99, pages 75–84, New York, NY, USA, 1999. ACM.

[8] Adomavicius, G., Sankaranarayanan, R., Sen, S., e Tuzhilin, A. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, 2005.

[9] Adomavicius, G. e Tuzhilin, A. Extending recommender systems: A multidimensional approach. In *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-01), Workshop on Intelligent Techniques for Web Personalization (ITWP2001*, pages 4–6, 2001.

[10] Apel, S., Batory, D. S., Kästner, C., e Saake, G. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013.

[11] Barboza, D. C. e Clua, E. W. G. Ginga game: A framework for game development for the interactive digital television. *2010 Brazilian Symposium on Games and Digital Entertainment*, 0:162–167, 2009.

[12] Basili, V. R. Software modeling and measurement: the Goal/Question/Metric paradigm. Technical report, Techreport UMIACS TR-92-96, University of Maryland at College Park, College Park, MD, USA, 1992.

[13] Benavides, D., Segura, S., e Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[14] Bencomo, N., Hallsteinsen, S., e Almeida, E. S.de . A view of the dynamic software product line landscape. *Computer*, 45(10):36–41, 2012.

[15] Bosch, J. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 257–271, London, UK, UK, 2002. Springer-Verlag.

[16] Caceres, R. e Friday, A. Ubicomp systems at 20: Progress, opportunities, and challenges. *IEEE Pervasive Computing*, 11(1):14–21, 2012.

[17] Cetina, C., Fons, J., e Pelechano, V. Applying software product lines to build autonomic pervasive systems. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 117–126, 2008.

[18] Chidamber, S. R. e Kemerer, C. F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June de 1994.

[19] Czarnecki, K., Helsen, S., e Eisenecker, U. W. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[20] Czarnecki, K., She, S., e Wasowski, A. Sample Spaces and Feature Models: There and Back Again. *Software Product Line Conference, 2008. SPLC '08. 12th International*, 2008.

[21] Freitas, G. B.de e Teixeira, C. A. C. Ubiquitous services in home networks offered through digital tv. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1834–1838, New York, NY, USA, 2009. ACM.

[22] Oliveira, R.de , Do Prado, A., Souza, W.de , e Biajiz, M. Development based on mda, of ubiquitous applications domain product lines. In *Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on*, pages 1005–1010, 2009.

[23] Deelstra, S., Sinnema, M., e Bosch, J. Product derivation in software product families: a case study. *J. Syst. Softw.*, 74(2):173–194, 2005.

[24] Fleury, A., Pedersen, J. S., e Larsen, L. B. Evaluating ubiquitous media usability challenges: Content transfer and channel switching delays. In Marcus, A., editor, *HCI (10)*, volume 6770 of *Lecture Notes in Computer Science*, pages 404–413. Springer, 2011.

[25] Forno, F., Malnati, G., e Portelli, G. Honey: a mhp-based platform for home network interoperability. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 102–110, 2006.

[26] Geraci, A. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE Press, Piscataway, NJ, USA, 1991.

[27] Gorgoglione, M., Palmisano, C., e Tuzhilin, A. Personalization in context: Does context matter when building personalized customer models? In *ICDM*, pages 222–231. IEEE Computer Society, 2006.

[28] Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., e Papadopoulos, G. A. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *J. Syst. Softw.*, 85(12):2840–2859, 2012.

[29] Hallsteinsen, S., Hinchey, M., Park, S., e Schmid, K. Dynamic software product lines. *Computer*, 41(4):93 –95, Abril de 2008.

[30] Hervás, R. e Bravo, J. Towards the ubiquitous visualization: Adaptive user-interfaces based on the semantic web. *Interact. Comput.*, 23(1):40–56, 2011.

[31] Hosseini-Pozveh, M., Nematbakhsh, M. A., e Movahhedinia, N. A multidimensional approach for context-aware recommendation in mobile commerce. *CoRR*, abs/0908.0982, 2009.

[32] Hussein, T., Linder, T., Gaulke, W., e Ziegler, J. Hybreed: A software framework for developing context-aware hybrid recommender systems. *User Modeling and User-Adapted Interaction (UMUAI)*, 2013.

[33] ISO/IEC,. *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001.

[34] JUNG, C. Guide - adaptive user interfaces for accessible hybrid tv applications. url: (http://www.w3.org/2010/11/web-and-tv/papers/webtv2_submission_55.pdf), acessado em julho de 2014.

[35] Kordus, D. What's on (digital) tv? assessing the digital television broadcasting system, its potential and its performance in increasing media content diversity. *Communication Law and Policy*, 19(1):55–86, 2014.

[36] Krueger, C. W. New methods in software product line practice. *Commun. ACM*, 49(12):37–40, 2006.

[37] Kunert, T. *User-Centered Interaction Design Patterns for Interactive Digital Television Applications.* Human-Computer Interaction Series. Springer, 2009.

[38] Lin, C.-L., Wang, P.-C., e Hou, T.-W. Classification and evaluation of middleware collaboration architectures for converging mhp and osgi in a smart home. *J. Inf. Sci. Eng.*, 25(5):1337–1356, 2009.

[39] Linden, F. J. v. d., Schmid, K., e Rommes, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[40] Lorenz, M. e Kidd, J. *Object-oriented Software Metrics: A Practical Guide.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[41] Maia, O. B., Viana, N. S., e Junior, V. F. d. L. Using the idtv for managing services in the ubiquitous computing environment. In *Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, UIC-ATC '09, pages 143–148, Washington, DC, USA, 2009. IEEE Computer Society.

[42] McCabe, T. J. A complexity measure. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[43] Montagud, S., Abrahão, S., e Insfran, E. A systematic review of quality attributes and measures for software product lines. *Software Quality Control*, 20(3-4):425–486, September de 2012.

[44] NETO, M. C. M. *CONTRIBUIÇÕES PARA A MODELAGEM DE APLICAÇÕES MULTIMÍDIA EM TV DIGITAL INTERATIVA*. Tese de doutorado em ciência da computação, UFBA, Universidade Federal da Bahia, 2011.

[45] Oliveira, M. R. M., Filho, C. B. P., e Fer, A. F. R. itv project: an authoring tool for mhp and ginga-j based on a web environment. In Darnell, M. J., Masthoff, J., Panabaker, S., Sullivan, M., e Lugmayr, A., editors, *UXTV*, ACM International Conference Proceeding Series, pages 179–182, 2008.

[46] Oliveira Junior, E. A., Maldonado, J. C., e Gimenes, I. M. S. Empirical validation of complexity and extensibility metrics for software product line architectures. In *Proceedings of the 2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse*, SBCARS '10, pages 31–40, Washington, DC, USA, 2010. IEEE Computer Society.

[47] OLIVEIRA JUNIOR, E. A. d. *SystEM-PLA: a systematic evaluation method for UML-based software product line architecture*. Tese de doutorado em ciências de computação e matemática computacional, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-07102010-111622/, Setembro de 2010.

[48] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., Wolf, A. L., e Wolf, E. L. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14:54–62, 1999.

[49] Palmisano, C., Tuzhilin, A., e Gorgoglione, M. User profiling with hierarchical context: An e-retailer case study. In Kokinov, B. N., Richardson, D. C., Roth-Berghofer, T., e Vieu, L., editors, *CONTEXT*, volume 4635 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2007.

[50] Parra, C., Blanc, X., e Duchien, L. Context awareness for dynamic service-oriented product lines. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 131–140, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[51] Pequeno, H. S. L., Gomes, G. A. M., Andrade, R. M. C., Souza, J. N.de , e Castro, M. F.de . Frameidtv: A framework for developing interactive applications on digital television environments. *J. Network and Computer Applications*, 33(4):503–511, 2010.

[52] PEREIRA, L. S. *GingaForAll: Linha de Produtos do Middleware Ginga*. Tese de doutorado em sistemas de computação, Centro de Ciências Matemáticas e da Terra, Universidade Federal do Rio Grande do Norte, Dezembro de 2010.

[53] RAFZ, D. F. Tuga game api. url: (http://software.dukitan.com/tuga/), acessada em julho de 2014.

[54] Schilit, B., Adams, N., e Want, R. Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.

[55] SEGUNDO, R. M. C. *Athus: um framework para o desenvolvimento de jogos para TV Digital utilizando Ginga*. Dissertação de mestrado em informática, Centro de Ciências Exatas e da Natureza, Universidade Federal da Paraíba, Setembro de 2011.

[56] Sommerville, I. *Software Engineering (9th Edition)*. Pearson Addison Wesley, 2010.

[57] Spínola, R. O. e Travassos, G. H. Towards a framework to characterize ubiquitous software projects. *Inf. Softw. Technol.*, 54(7):759–785, 2012.

[58] Suryn, W. e Gil, B. Iso/iec 9126-3 internal quality measures: Are they still useful. *Proceedings of HCI International*, 2005.

[59] Vaguetti, L. e Gondim, P. R. L. A web-service-based architecture solution to ubiquitous personalized digital television content. In *5th International Workshop on Ubiquitous User Modeling*, 2008.

[60] Vaguetti, L. e Gondim, P. R. L. Hierarchical Intra-Context Handoff Awareness Approach to Personal Recommender Systems: A mobile DTV case study. In *International Conference on the Applications of Digital Information and Web Technologies*, 2008.

[61] Weinand, A., Wein, A., Gamma, E., e Inc, T. Et++ - a portable, homogenous class library and application framework. computer science research at i/blab. universitatsverlag konstanz, 66-92, 1994.

[62] Weiser, M. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.

[63] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., e Wesslén, A. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

ANEXOS

A PAPER SUBMETIDO (EM REVISÃO) AO *IEICE Transactions on Information and Systems*

LETTER

A Software Product Line Approach for Developing iDTV Applications

Leandro VAGUETTI[†a)] ***and*** **Paulo ROBERTO DE LIRA GONDIM**[†b)]**,**

SUMMARY This paper presents a Dynamic Software Product Line - DSPL approach for the development of interactive Digital TV (iDTV) applications. The solution is based on FOSD (Feature-Oriented Software Development) and generates software products at runtime in the DTV receiver. The proposed solution is built with the best practices of software engineering, which enable reuse and software evolution.

key words: *Interactive Digital Television, FOSD, SPL, DSPL*

1. Introduction

Currently, the interactive applications for interactive Digital Television (iDTV) have become an important resource as a business model for marketing, distance learning, e-commerce, etc. Applications for iDTV enables the association of mechanisms of interactivity to traditional television content. In this sense, the viewer can be provided with interactive media (e.g. products for purchase, information, polls, etc) relating to the content watched.

However, in the current scenario, a great deal of development/maintenance of applications for iTVD is required, due to the large amount of television content produced daily and the need for customizing the interaction with each content available. Moreover, the variety of Digital TV receivers available (e.g. Mobile TV, HDTV, etc.) also increases the need for interactivity customization.

Solutions for the development of applications for iDTV that produce software with sufficient quality attributes so as to minimize the efforts of developing and maintaining applications have been required. Such solutions must consider the particular characteristics of the iDTV environment (e.g. broadcast transmission, limited interactivity mechanisms, limited computational resources, etc).

The adaptation of interactive contents on reception must consider aspects such as variation of receivers, availability of interactivity mechanisms and interactivity profile of the viewer.

In this sense, this paper proposes the use of an FOSD (Feature-Oriented Software Development) approach, based on SPL (Software Product Lines), for minimizing the difficulties faced in the development of interactive applications in a DTV scenario. The solution provides a development framework (DSPL2iDTV) for the building of iDTV features that make up a SPL, as well as, a DSPL (Dynamic SPL) Configurator to select features and dynamically generate the software product suitable for the receiver/viewer. The proposed solution have been validated by a case study.

The paper is organized as follows: Section 2 presents the DSPL approach for software development to be used in the iDTV domain; Section 3 describes a case study, considering an iDTV Software refactoring to SPL; Section 4 discusses some related work; finally, Section 5 addresses our final remarks and future work.

2. FOSD Approach to iDTV

iDTV applications have some characteristics related to the need to adjust the content to the receiving equipment. Among the main features are transmission over broadcast channel, different receiving equipment and different mechanisms of interaction with the iDTV contents (Figure 1 - a).

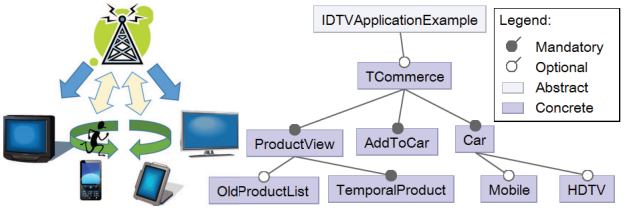

Fig. 1 (a)iDTV Scenario (b)iDTV Application Example

From the point of view of software engineering, interactive content (interactive applications) must be constructed so as to suit the viewer's needs, as well as details of the DTV environment and enable the adequate maintenance of the software developed. Additionally, interactive applications must offer performance and usability at any time.

Applications are developed considering the various execution environments, and should optimize the cost associated with the production and software evolution. Several techniques for software reuse have been developed and one of the most promising approaches is the SPL. SPL is a set of applications (software products) that have a common architecture and shared components, where each product is specialized so as to reflect the different needs of users or execution environments [3]. Figure (1 - b) shows an SPL example of iDTV application for the T-commerce domain, with specific features.

An extension of the SPL approach, called Dynamic

Manuscript received January 1, 2011.
Manuscript revised January 1, 2011.
[†]University of Braslia, Braslia-DF, Brazil
a) E-mail: vaguetti@unb.br
b) E-mail: pgondim@unb.br
DOI: 10.1587/trans.E0.??.1

Software Product Lines (DSPL) [5] enables the selection of software product features in execution time. Besides of configuration, the reconfiguration of the software product in real time is considered if context switches or user's preference must be set.

This paper proposes a FOSD approach for the development of interactive applications for digital television, composed of the following parts (or components): (1) an API for the implementation of features; (2) a mechanism for setting the SPL and constraints; (3) a solution for building dynamic product based SPL; (4) a solution for self-adapting applications.

The proposed solutions are delivered through a FOSD process. In general terms, the FOSD process consists of the following phases: (1) Domain Analysis, (2) Specification and Design Domain, (3) Domain Implementation, and (4) Configuration and Generation of Software Products.

2.1 Domain Analysis

The Feature Model (FM) of an SPL is set in the Domain Analysis phase [1]. The framework provides a set of tags for the creation of a XML file, where the FM will be represented and allowing the configuration, generation and control of SPL.

Once the structure of the Feature Model has been defined through metadata, structural changes involving relationships and constraints can be easily made and extended.

Furthermore, the framework enables the definition of sub-models used in an execution context, i.e., Feature Models specific to different execution environments, such as HDTV, DTV, smartphones, PCs and tablets.

2.2 Domain Design and Specification

In the context of FOSD, the Domain Design and Specification phase involves the treatment of the essential structural and behavioral properties of the resources involved, characterized by a formal or informal specification and/or a modeling language. The UML modeling language was used in our work, and the modeling is presented below by class diagrams for some of the main software components of the framework.

The *ITVFeature* class (Figure 2) implements the XLet JavaDTV API interface (API supported in the main middleware available worldwide, e.g. MHP and Ginga-J). The life cycle of an *ITVFeature* is similar to a XLet, and provides methods for initialization, execution, break and destruction of the *ITVFeature*.

To support the construction of feature models, the *ITVFeature* class also enables identification of a child class from a parent class. Therefore, a feature model instantiating and interrelating instances of the mentioned class can be constructed.

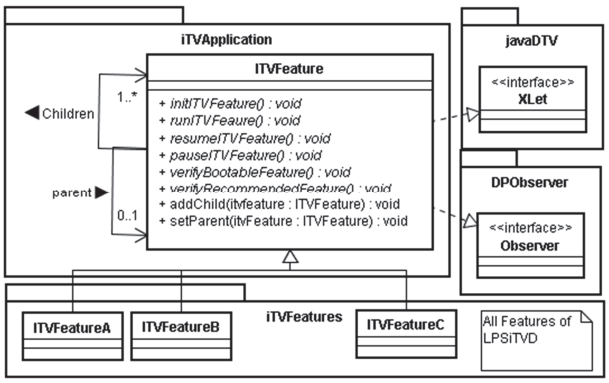

Fig. 2 *ITVFeature* Class Diagram

2.3 Domain Implementation

This section presents the main classes of the framework, since the codes are implemented in the Domain Implementation phase. The first step in this phase is the development of the Features that will comprise the SPL. Such features are developed so that decisions concerning the organization, relationships and constraints of the Feature Model can be determined by other mechanisms (Figure 1 - b). Therefore, the *ITVFeature* class must be extended for the development of an adherent feature framework (Table 1).

```
1 public abstract class ITVFeature implements Xlet, Observer,
2 ITVFeatureInterface, ActionListener {...
3 public void initITVFeature(){ ...}
4 public void runITVFeature(){ ... }
5 public void resumeITVFeature() { ... }...}
```

Table 1 *ITVFeature* Class

Once all features have been defined and implemented, the developer must define the feature model according to the example shown in (Figure 1 - b).

2.4 Product Configuration and Generation

As previously mentioned, the configuration of a product software is based on the selection of features available in a software product line. In the case of interactive Digital TV applications, the product is hardly selected in advance, since it cannot be able to run in some devices due to limitations related to processing, resolution, interaction mechanisms, etc.

When Software Product Lines are used for the development of interactive applications for Digital TV, a configuration mechanism and product generation software must be used at runtime, wich leads to the convenience of using a Dynamic SPL (DSPL).

In this framework, the product configuration and reconfiguration task is distributed between the Features. In practice, each *ITVFeature* controls its life cycle and the life cycle of its dependents.

Each *ITVFeature* is self-assessed regarding three situations that can prevent it from running: it self-checks whether the environment has sufficient resources for the execution of the *ITVFeature*; it self-checks whether the *ITVFeature* has a recommended execution; it self-checks if there is some restriction that prevents its execution, such as having a dependency on the execution status of another *ITVFeature*.

The dynamic product configurator initializes all *ITVFeatures* randomly and each self-assessment for the transition to a possible execution state. The transition from one state to another is based on the life cycle for an *ITVFeature*. This life cycle is very similar to the life cycle of a XLet (object that represents a media on Interactive Digital TV, implemented in Java). Figure (3) shows the possibilities of state transition of an *ITVFeature*.

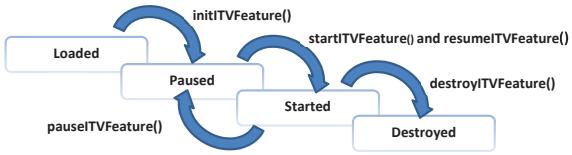

Fig. 3 *ITVFeature* - Life Cycle

Each *ITVFeature* may depend on an *ITVFeature* at a higher hierarchical position, and have *ITVFeatures* dependents. In such cases, when an *ITVFeature* enters the paused, booted, rebooted or destroyed states, their dependents must transition to the same state of the dependent *ITVFeature*.

3. Experiments and Results

The experiment was conducted through a case study [8]. The case study concerns a SPL implemented as shown in Figure 4, involving a T-commerce application. In SPL it can be observed the possibility of building several separate software products.

Fig. 4 TVDeals iDTV Application ScreenShots

The framework evaluation is based on software quality attributes. Thus, based on the GQM approach [2], the goal of the experiment is presented in Table 2. To answer the research question, the following methodology was applied:

1. Set the configuration software products from the SPL developed (Figure 5). In this case study, 8 product settings were generated.

GOAL Purpose	evaluate
Issue	level of complexity and reusability attributes of
Object	DSPL2iTV framework
ViewPoint	Designers and Developers of software
QUESTIONS	The DSPL2iTV framework provides facilities for software development and reuse ?
	The implementation of variability across the framework impacts the evaluated quality attributes?
METRICS	CompPLA and DITPLA

Table 2 Goal of the Experiments

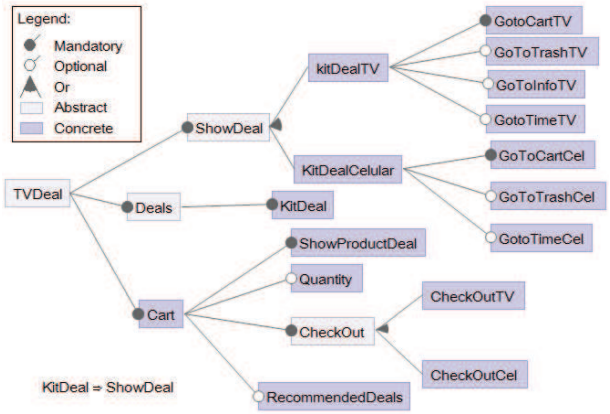

Fig. 5 TVDeals Application: *Feature Model*

2. Collect the CompPLA and DITPLA metrics (Equations 1, 2) for each defined product configuration. The CompPLA metric is defined in [6], being based on $WMC_{McCabeCC}$ (Weighted Methods per Class based on Cyclomatic Complexity of MacCabe) metric, and aims to reflect the complexity of the software. DITPLA is based on DIT (Depth of Inheritance Tree) metric that aims to reflect the degree of software reuse.
These metrics can be expressed as:

$$CompPLA_{PA} = \sum_{i=1}^{NCpt} CompVarComponent_{Cpt_i} \quad (1)$$

$$DITPLA_{PA} = \sum_{i=1}^{NCpt} DITVarComponent_{Cpt_i} \quad (2)$$

Where: $CompPLA$ represents the metric used for evaluation complexity; $DITPLA$ represents the metric used for evaluation reusability; PA represents the architecture of the selected product; $NCpt$ represents the number of components present in the configured product; $CompVarComponent(Cpt_i)$ is the complexity of PA for the i^{th} PA component; $DITVarComponent(Cpt_i)$ is the reusability of PA for the i^{th} PA component;
3. Collect the percentage of features that represent variability as well as the amount of features that make up the product set.
4. Identify the existence of a linear correlation between the values obtained for: CompPLA, DITPLA, percentage of variable features and the number of features used

in each setting. If there is a significant correlation, it is possible to use linear regression to predict future values and identify the impacts that the framework on the evaluated quality attributes.

It should be noted that since the framework is developed using object orientation (Java), the adopted metrics are proposed for evaluation of object oriented codes/designs. Moreover, the codes used in the assessment correspond specifically to the framework and required codes, extended by the developer. Thus, it is possible to isolate the impact of the framework on quality attributes, without the influence of any other values, dependent on developer based codes.

3.1 Results

For the case study, 8 product configurations were constructed. For each configuration, the amount of features involved, the percentage of features variables, CompPLA and DITPLA metrics were collected.

Shapiro-Wilk normality test was applied to the samples, and the distributions of CompPLA and DITPLA were non-normal. The linear correlation of Spearman was applied to the variables collected (Table 3), leading to a strong correlation between the variables considered.

	CompPLA	DITPLA
% Variabilities	0,981707317	0,80607541
Features	0,981707317	0,80607541

Table 3 Linear Correlation of Spearman

Thus, applying multiple linear regression to estimate the CompPLA and DITPLA metrics, we respectively obtain, for a CI = 0,95 (Confidence Interval), R^2 = 0,9654 and R^2 = 0,5973; and following models:

$$Pd_C = 174,3562 + (\beta_1 * -9,7692) + (\beta_2 * 5,8720) \quad (3)$$

$$Pd_D = -15,2223 + (\beta_1 * -78,8522) + (\beta_2 * 9,4130) \quad (4)$$

WherePd_C and Pd_D represent the predicted values of CompPLA and DITPLA metrics, respectively. β_1 and β_2 represent the percentage values of feature variabilities and feature quantities, respectively.

Predictions were made using equations (3) and (4), as shown in figures 6 (for 20 and 100 features) and 7 (considering 50% of variabilities). The results indicated a small impact on quality attributes generated by variable features and a positive impact in the attributes of complexity and reusability using the DSPL2iDTV framework.

4. Related Works

Some of the main solutions for Java-based FOSD are: AHEAD, FeatureHouse and AspectJ. Only AspectJ allows dynamic configuration of products. However, AspectJ has no integration with iDTV middleware.

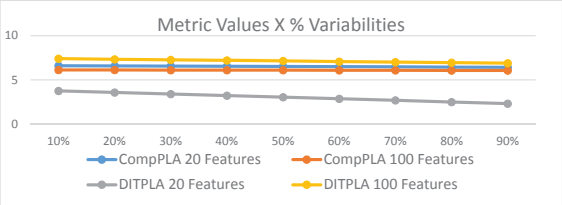

Fig. 6 Metric Values X (%) Variabilities - 20 and 100 Features

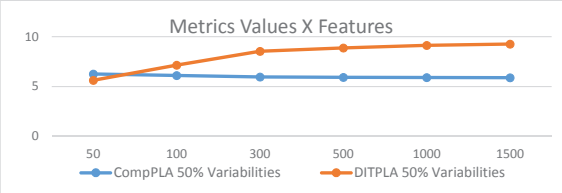

Fig. 7 Metric Values X Features - 50% of Variabilities

In addition, previous works relating SPL and Digital TV are targeted at solutions focused on Digital TV middleware. In [4] the authors consider the MHP (Multimedia Home Platform) middleware, without a detailing of the FOSD process. In [7] the authors uses of SPL in the refactoring of the Ginga middleware.

5. Conclusions and Future Works

This paper has proposed a solution for the development of iDTV applications. Concepts of SPL (particularly of DSPL) were used so as to offer a DSPL framework and enable the development of applications for the Digital Television domain. The main research questions were answered. Future work will consider the extension of the solution for the treatment of characteristics of ubiquitous software projects.

References

[1] Apel, S., Kstner, C.: An overview of feature-oriented software development, journal of object technology - jot, 49-84 (2009)
[2] Basili, V.R.: Software modeling and measurement: the Goal/Question/Metric paradigm. Tech. rep., Techreport UMIACS TR-92-96, University of Maryland at College Park, College Park, MD, USA (1992)
[3] Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. J. Syst. Softw. **74**(2), 173–194 (2005)
[4] Goedicke, M., Köllmann, C., Zdun, U.: Designing runtime variation points in product line architectures: Three cases. Special Issue: Software variability management 53(3), 352-380, Elsevier (2004)
[5] Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. Computer **41**(4), 93 –95 (2008)
[6] Oliveira Junior, E.A., Maldonado, J.C., Gimenes, I.M.S.: Empirical validation of complexity and extensibility metrics for software product line architectures. SBCARS '10, pp. 31–40 (2010)
[7] Saraiva, D., Pereira, L., Batista, T.V., Delicato, F.C., Pires, P.F., Kulesza, U., de Arajo, R.P.M., Freitas, T., Filho, S.M., Souto, A.L.S.: Architecting a model-driven aspect-oriented product line for a digital tv middleware: A refactoring experience. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 166181. Springer (2010)
[8] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)

B PAPER SUBMETIDO À REVISTA *Information and Software Technology - Elsevier*

DSPL2UbiTV: A DSPL Platform for the Ubiquitous Software Development for iDTV Environments☆

Leandro Vaguetti and Paulo Roberto de L. Gondim[1,]

Dept. Electrical Enginneering
Faculdade de Tecnologia
University of Braslia-DF
Tel.Fax: +055-62-3107-5580

Abstract

This paper presents the DSPL2UbiTV platform for the development of ubiquitous applications for Interactive Digital TV - iDTV. The DSPL2UbiTV Platform is based on a Dynamic Product Line Software - DSPL, generating software products, at runtime, in the DTV receiver. The presented platform supports the development of project characteristics of ubiquitous software such as omnipresence of services, context awareness, adaptive behavior, capturing experiences, etc , Spínola and Travassos (2012). Furthermore, the platform allows the dynamic configuration of software products based on the recommendation of features. In this context, the features are recommended considering user interactions with iDTV contents. Moreover, the proposed solution is built using the best practices of software engineering, allowing reuse and software evolution.

Keywords: Ubiquitous Computing Software, DSPL, Recommender Systems, Software Product Lines

1. Introduction

There is an increasing amount of equipment where software can be used. Personal computers, laptops, tablets, smartphones are examples of such equipment. However, the large amount of environments in which applications must be able to run has become a problem for developers' software. Software developed for different environments should consider the environments (contexts) such as processing power, screen resolution, peripheral input and output , support networks, etc. In addition, the software must be built to meet user needs and software evolution.

In this context, applications for Interactive Digital TV - iDTV represent a constant difficulty for developers due to the necessity to build applications that can run on different equipment, with different hardware configurations and serving users with different profiles. Moreover, one should consider the characteristic submission of applications through a broadcast channel and the frequent need for modification or addition of new requirements in applications.

From the functional point of view, the applications that concentrate most amount of need for adaptations are ubiquitous applications , Caceres and Friday (2012), Weiser (1999). Such applications or services should always be available to users, regardless of context (location, time, operating environment, etc.) of the user. In addition, the ubiquitous software has several characteristics (adaptable behaviour, context awareness, ubiquitous service, etc.) that must be realized by applications , Spínola and Travassos (2012). Thus, the ubiquitous software are in constant evolution, due to the development of new technologies. However, currently, there is a great difficulty in the development of ubiquitous software, since tools to support the development do not support all features needed, and do not consider issues relating to the software evolution.

Therefore, when considering the need to develop ubiquitous applications aimed at an environment of interactive Digital TV, it is worth identifying a set of challenges that require solutions for the development of applications, such as the support characteristics of ubiquitous software projects, the capacity for dynamic construction of application, and facilities maintenance and evolution of software.

From the point of view of software engineering, applications are developed considering the various execution environments, and should optimize the cost associated with the production and software evolution. In this sense, several techniques for software reuse are found to meet this need, and one of the most promising approaches is called Software Product Lines - SPL. SPL is a set of applications (software products) that have a common architecture and shared components, where each product is specialized to reflect the different needs of users or execution environments , Bosch (2002), Krueger (2006), Sommerville (2010).

The SPL, in its original proposal, allows a preselection of software functionality in accordance with the needs of the environment and application users , Deelstra et al. (2005). There are several successfully applied cases to environments such as mobile phones, cars, planes, etc. However, some types of applications do not allow preselection functionalities. Some applications for Web and Interactive Digital TV, for example, have

Email address: vaguetti@unb.br and pgondim@unb.br (Leandro Vaguetti and Paulo Roberto de L. Gondim)

characteristics of transmission and construction of software that impose the need of obtention of necessary information for its execution and subsequently, at runtime, determine which features will be selected for the software product.

An extension of the SPL approach called Dynamic Software Product Lines (DSPL) , Hallsteinsen et al. (2008) provides for the selection of software product features in execution time. In this case, in addition to the configuration, reconfiguration of the software product in real time is considered, if context switches or user preference must be set. Thus, adaptive systems based on as ubiquitous, personalized and context aware applications could be built using the DSPL , Parra et al. (2009) , Hervás and Bravo (2011).

In literature there are several solutions for defining variabiltity, similarities and restrictions in SPL , Benavides et al. (2010), Czarnecki et al. (2005), Czarnecki et al. (2008), as well as proposals for configuration and generation of products software. In special, the inherent variability in SPL naturally leads to decisions to be made for the sake of feature selection. Thus, it becomes necessary to define and build the mechanisms for decision making on the selection of features in situations of indecision. Defining which features should be chosen, considering previously defined selection criteria or change.

This work presents a DSPL Platform, based on recommendation for the approach of features for the development of ubiquitous applications in Digital Interactive TV (DSPL2UbiTV) environment. In this sense, the main elements of the architecture are presented, as well as the necessary software product configuration mechanisms. Regarding the evaluation of the solution, we consider the ability to maintain DSPL2UbiTV by measuring quality attributes of software such as extensibility and complexity. Moreover, a case study is presented, aiming to exemplify the use of the platform.

The paper is organized as follows: section 2 presents a background on concepts related to iDTV applications, software product lines and characteristics of ubiquitous software; section 3 addresses related works; section 4 presents the general software architecture of the DSPL2UbiTV Platform and the solution proposed to dynamically solve the SPL variability in the execution environment; section 5 describes the methodology of evaluation of the proposed solution and the metrics of software quality that were considered; finally, section 6 concludes the paper and suggests some future studies.

2. Background

Initially it is necessary to highlight some concepts adopted for definition of the problem addressed as well as for the proposed solution. In this context, it is necessary to consider concepts used for self-adaptive applications, context sensitive applications, ubiquitous applications and Interactive Digital TV, as follows below.

Self-Adaptive Software are applications that modify their behavior according to changes in the operating environment , Oreizy et al. (1999).

Context Sensitive Software are applications that adapt both modules for the system and for information, in accordance with a set of context variables such as location, time, people, equipment, etc; , Schilit et al. (1994). That is, the self-adaptive software can be contained in context-sensitive software.

Ubiquitous Software Weiser et al. , Weiser (1999) define ubiquitous computing as specialized hardware or software that are so ubiquitous that they are not perceived by users. Complementing this definition, Abowd et. al , Abowd (1999) suggests that applications should provide ubiquitous services to users so transparent through the environment anywhere at any time. Thus, it can be considered that the context sensitive software may be contained in the ubiquitous software category.

Software for Digital Interactive TV According to , Kunert (2009) programs to be considered for TV programs Interactive digital as an application or service available in television content, which may be affected by the interaction of user.

Given the concepts presented above, we can define the concept adopted for **Ubiquitous Applications for Interactive Digital TV**. Considering that an application for interactive Digital TV is available through a broadcast channel, and that such content can be received in various kinds of receivers, it is understood that this application is ubiquitously available to all Digital TV receivers (i.e., if the user changes the receiver he/she will continue to have access to content). Moreover, we consider that thus application has function that are limited to the broadcast transmission and the corresponding reception, running on a DTV receiver.

However, currently, if the user is interacting with some content in a receiver, to make the switch to other receiver, the user will lose the information on the operational status of the interaction (i. e., the user will not be able to interact with the application from the point it was before). Thus, the user will have to boot for every interaction and in each exchange of receiver that has been performed. Thus, one **Ubiquitous Application for Interactive Digital TV** is an application that can maintain the omnipresence on the code and interaction or service provided by the application, in an adequate manner (that is, with a good quality of experience) in all possible reception environments.

2.1. Interactive Digital TV Applications

Applications developed for DTV environments are characterized by the possibility of the inclusion of interacting mechanisms to content , Kunert (2009). Such feature enables new business and information models, since interactivity leads to an attractive, rich and portable environment (for example, for t-commerce applications) and a new paradigm of Digital TV applications.

The interactivity in applications for Digital TV allows the inclusion of several services, such as search information, distance learning, accessibility solutions, games, etc. However,

the form and amount in which resources of interactivity can be presented to viewers can vary significantly, especially considering targeted content to a variety of: public, regions, equipment, crops, etc. Thus, interactivity features when provided in excess or limitations may make the uninteresting content and related business models.

In this context, applications that have the ability to adapt to receiving environment become increasingly popular to viewers. Among the applications that suggest self-adaptation, ubiquitous applications can be mentioned.

However, regarding the ubiquitous applications for Digital TV Interactive, one can find only a few case studied (for example specific and targeted integration of applications Digital Interactive TV and Home Networks as , Lin et al. (2009), , Forno et al. (2006), , Maia et al. (2009) and , de Freitas and Teixeira (2009)). However, none of the work presented is a solution for developing applications. Furthermore, studies are designed to evaluate the ubiquity of content in television environments , Fleury et al. (2011).

2.2. Ubiquitous Software Characteristics

In this section, the several characteristics of ubiquitous software are pointed out and their relative importance is discussed. We initially observe that, when using SPL to produce ubiquitous software, some modules that implement the characteristics of ubiquity may be elective and others mandatory. Thus, the selection of optional modules can be determined by either the conditions of the operating environment (e.g. hardware resources and communication) or the efective users application.

The software architecture proposed in this study was designed considering the characteristics of ubiquitous software identified by Spinola et al. , Spínola and Travassos (2012):

Service omnipresence *"it makes users able to move around with the sensation of carrying computing services with them. For instance: an engineer manages several projects and needs to visit the development teams located in different sites. However, he also needs to monitor the other software projects progresses to report their results for the organization. When the software engineer is visiting a specific development site, the local software development environment can connect with the other projects environments. Thus, the engineer will have access to the software projects everywhere as he moves around."*

Invisibility *"it refers to the ability to be present on a daily basis, using objects, weakening, from the users point-of-view, the sensation of explicit use of a computer and enhancing the perception that objects or devices can provide services or some kind of intelligence. For instance: an environment monitor that should be constantly monitoring some comfort variables and adjusting the air conditioning system or asking for maintenance without user intervention."*

Context sensitivity *"it is related to mechanisms present in ubiquitous systems for collecting information from the environment where the system is being used. For instance: a system to control the intensity of light inside a classroom should be constantly monitoring the intensity of light to keep the room in the comfortable configuration for reading."*

Adaptable behaviour *"it represents the dynamic capacity of self adaptation according the environments limitations to the services that should be offered. For instance: by identifying the increasing of throughput to the point of harming the processors due their temperature, the high performance computer management system should command the increase of cooling to reduce the risk of processing failure."*

Experience capture *"it makes the ubiquitous systems able to capture and register experiences for future use. For instance: a software for ambient intelligence can identify common user behaviours, for example: when arriving at work, the employee turns on the office light, computer, air conditioning system at 24C and notifies the team about his presence at the office. The software can manage these activities as soon as it identifies the user arrives at office without repetitive user commands."*

Service discovery *"it represents mechanisms to support proactive discovery of services by the ubiquitous system in accordance with the environment where it is being used, allowing the achievement of some desired target by the finding of new services or information. For instance: a smartphone software based on the location of its user can identify the local restaurants serving the users preferred food and discover those with available seats at the moment."*

Function composition *"it represents the ability to create a service required by the user based on the existent basic services. For instance: a user needs to convert a XML file from one tool to another and this conversion service is not available in the computer. The software can identify the necessary services in other devices and makes them available for use."*

Spontaneous interoperability *"it allows the ubiquitous system to change its partners during operation. For instance: a user is moving and the software, running on a smartphone, is executing a data-intensive process. During the moving, the software can interact with other devices in the environment for temporary allocation of information."*

Heterogeneity of devices *"it makes able the software application to acquire mobility amongst*

heterogeneous devices. Thus, the software application could migrate amongst devices and adjust itself to each device. For instance: an email client that can be used in the workstation at the office or at the smartphone on the road."

Spinola et. al. , Spínola and Travassos (2012) also evaluated the use, pertinence and relevance of the mentioned characteristics, considering opinions expressed by a large group of evaluators (Figure 1).

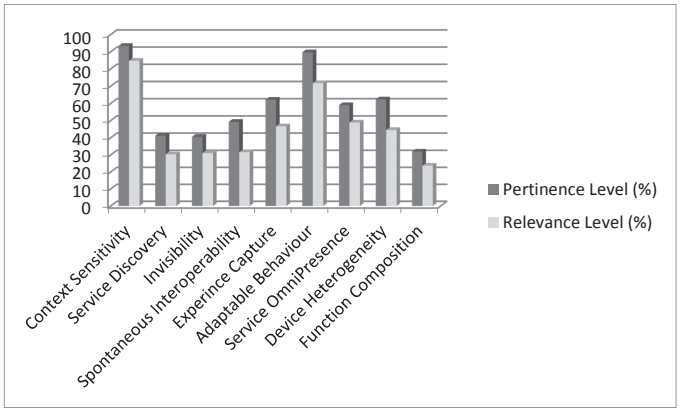

Figura 1: Spinola et al. Research Results

Be observed through the results, some features are considered more or less relevant. However, all are considered relevant by a large group of evaluators. Moreover, one can observe that features like **Omnipresence Service** are found in only about 50% of the projects evaluated, in spite of the main characteristic of a ubiquitous application is the omnipresence of service.

Thus, this work seeks support through software development tools, all the functional features identified by Spinola et al. However, due to the complexity and time required for development work, the implementation of the features considered restrictive (such as: Fault Tolerance, Scalability, Privacy and Trust, etc.) are considered for future work.

2.3. Software Product Lines - SPL

Software Products Line (SPL) is an emerging technology that enables the development of reusable software for specific domains. The software components are arranged so that a software product is obtained from the selection of a set of components among the various components of SPL. A very important part of the definition of a SPL is a feature model, that defines the representation of common and uncommon features, as well as dependency relations between features, allowing to manage the variability of features.

An SPL Platform , Deelstra et al. (2005) is an extensible SPL, in which a set of common features is available and integrated into various applications with similar characteristics (e. g. context-aware management). Conceptually, the characteristics of the SPL Platform are similar to those of a framework for software development.

A Dynamic SPL (DSPL) , Hallsteinsen et al. (2008), Bencomo et al. (2012) allows software products to be defined at runtime of the application. Ubiquitous iDTV applications should be built at runtime in DTV receiver. The iDTV applications are broadcast to millions of recipients irrespective of the ability of receivers; thus, the SPL platform for ubiquitous iDTV applications must build the software dynamically.

2.3.1. FOSD - Feature-Oriented Software Development

Feature-oriented software development (FOSD) is a paradigm for the construction, customization, and synthesis of large-scale software systems. The concept of a feature is at the heart of FOSD. A feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option. The basic idea of FOSD is to decompose a software system in terms of the features it provides. The goal of the decomposition is to construct well-structured software that can be tailored to the needs of the user and the application scenario. Typically, from a set of features, many different software systems can be generated that share common features and differ in other features. The set of software systems generated from a set of features is also called a software product line , Apel et al. (2013).

2.4. Recommender Systems Multidimensional Model

The model originally presented in multidimensional , Adomavicius et al. (2005), Adomavicius and Tuzhilin (2001) allows to provide recommendations considering not only a two-dimensional model (classical model systems recommendation), using the relation $User \times Item$, as also a relationship that may involve other factors (dimensions) in providing the recommendation. Thus, the multidimensional model considers dimensions like user, item, time, location, etc; introducing the concept of filtering aware of the context model.

Should formally identify the set $D = D_1, D_2, D_3, ..., D_n$ of dimensions used, and use a ranking function (*rating function*) (1) for the classification or rate recommendation of the multidimensional space provided.

$$R : D_1 \times D_2 \times ... \times D_n \longrightarrow Rating \quad (1)$$

Then one can consider the example shown in (Figure 2). Where shows the spatial model, cube-shaped, search and store the desired information. The Figure (2), can be observed using a dimensional space consisting of $User \times Item \times Time$, where $u \in User$, and represents the identification of a specific user, $i \in Item$, representing a specific item and $t \in Time$, representing an specific time interval. Thus, using the function (2) to obtain the desired value.

$$R(u, i, t) \longrightarrow Rating \quad (2)$$

3. Related Works

Among the works that have the greatest relationship as the topic of ubiquitous software development for Digital TV interactive we can highlight the following solutions:

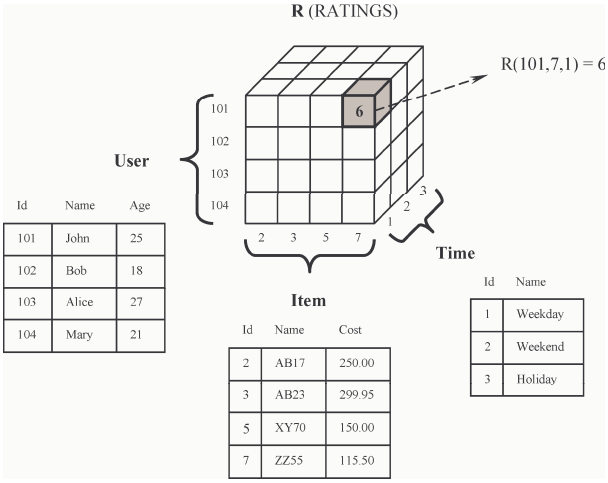

Figura 2: Multidimensional Model , Adomavicius et al. (2005)

Oliveira et. al , de Oliveira et al. (2009), proposes a solution that uses MDA (Model Driven Architecture) and LPS (Software Product Lines) for the development of ubiquitous applications. In this sense use Framework called UCF (Ubiquitous Computing Framework) as a basic component of software to support the implementation of features regarding the ubiquity. The UCF is able to support the implementation of characteristics of ubiquity as: context awareness, adaptive behavior and heterogeneity of devices. However, the solution is not suitable for TV Digital Interactive, given that the software product must be pre-selected before sending broadcast via the user or viewer. Furthermore, the presented solution does not support the development of the ubiquity of the service offered by application of the main features that differs applications of ubiquitous context-aware applications.

Parra et.al, , Parra et al. (2009) developed work aimed to provide a dynamic software product line service oriented to the domain of Context-sensitive applications . The work of Parra et. al, have similarities with the proposed work due to the use of dynamic software product line and meet the characteristics of Ubiquitous software product called "Sensitivity to Context"and "Adaptive Behavior". However, this solution does not offer support to implementation of the other characteristics of ubiquity cited and evaluated by Spinola et. al , Spínola and Travassos (2012), it does not offer specific support of the Interactive Digital TV applications.

Hallsteinsen et. al , Hussein et al. (2012) developed a project focused on providing a process development of ubiquitous applications, added to a solution based on a middleware able to support sensitivity context and self-adaptation software. Also, provides a set of reusable components to support the development applications that executed on the middleware. However, this work differs from the proposed solution, especially as to support interactive digital TV applications, as well as non-adequately characterize the features of the support ubiquity necessary (although cite support the development of ubiquitous applications).

Cetina et. al , Cetina et al. (2008) introduces an approach to designing pervasive SPL based on Model Driven Development (MDD) and Variability Modeling principles. The work presents a solution for runtime reconfiguration, allowing a Pervasive SPL adapts autonomously. However, the work does not have support for Digital TV applications. Moreover, the support does not clearly define the characteristics of ubiquitous projects.

3.1. Comparison of Related Works

Among the factors used to compare the solutions already previously performed and the work proposed here, we considered 3 (three) types of support related to software development contributions that can offer solutions compared. It must be emphasized that the assessment was conducted only through bases theoretical.The types of support were:

Software Maintainability According to , Geraci (1991), the software maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a change in the operating environment. Thus, we sought to assess whether the proposed solutions have some kind of support maintainability of software, from the architectural point of view.

Characteristics of Ubiquitous Project We sought to evaluate whether the proposed solutions offer some support to characteristics ubiquitous software projects. Thus the functional characteristics identified in , Spínola and Travassos (2012) were used as a basis for evaluation. It is noteworthy in Table (1) that the main characteristics of ubiquity (in terms of relevance and pertinence) have highlighted their assessment individually, due to variation introduced by the support solutions compared.

Interactive Digital TV tried to identify support to the implementation of applications for Interactive Digital TV, both to support specific API's related to the characteristics and limitations of the application of interactive software for Digital TV.

The listing below defines the acronyms used for each element evaluated and used in the comparison in Table (1):

A - Software Maintainability

B - Context Sensitivity

C - Adaptable Behaviour

D - Heterogeneity of Devices

E - Experience Capture

F - Service OmniPresence

G - Other Characteristics

H - iDTV Suport

Related Works	A	B	C	D	E	F	G	H
, de Oliveira et al. (2009)	X	X	X	X				
, Parra et al. (2009)	X	X	X	X				
, Hallsteinsen et al. (2012)	X	X	X	X				
, Hussein et al. (2013)	X	X	X	X				
, Cetina et al. (2008)	X	X	X	X				
DSPL2UbiTV Platform	X	X	X	X	X	X	X	X

Tabela 1: Comparison Elements Evaluated

It is noteworthy that the analysis of the level of support from related work for each element used for comparison was significantly abstract, being based only on the description given in the published works.

Thus, it was concluded that solutions related to this work:

- Have some support to maintainability of software, however, do not present review on the quality offered for maintenance software;
- Support the main characteristics of ubiquity considered relevant and pertinent to a project ubiquitous software; however, do not support the omnipresence and do not support other important features like capture experience, spontaneous interoperability, invisibility, etc;
- Finally, none of the analyzed solutions supports the implementation of interactive applications for Digital TV.

4. DSPL Platform for the Ubiquitous Software Development to iDTV Environments - DSPL2UbiTV

This section presents the information on the feature model of the platform, its overall architecture, as well as the adequacy to the FOSD process.

4.1. Feature Model and General Architecture

The DSPL platform for developing software for ubiquitous interactive Digital TV (iDTV) environments is proposed to serve as a tool to support the development of software product lines (SPL), and to support projects in the field of ubiquitous applications for iDTV.

To identify the main features of the platform, it is shown in (Figure 3) its general feature model. Four main features can be observed in the model presented, where:

SPLiDTV: Mandatory and abstract feature, that represents a specific SPL for iDTV. Such SPL for iDTV is built by developers with the support of DSPL2UbiTV platform. The SPLiDTV has its execution controlled by feature called DSPL Configurator;

UbiquitousFeatures: Optional feature which is related to the possibility of aggregating characteristics of ubiquity to SPLiDTV developed through the platform. Once selected the features concerning the characteristics of ubiquitous software, the controls implemented, due to the characteristics will be considered during the implementation of a generated software product;

DSPLConfigurator : Mandatory feature which is related to the implementation of dynamic product configurator developed in SPLiDTV;

RecommenderFeatures: Optional feature, which represents the ability of the developer to identify/implement the recommendation techniques used to determine the selection of features (optional), at run time, given the optional features present in SPLiDTV.

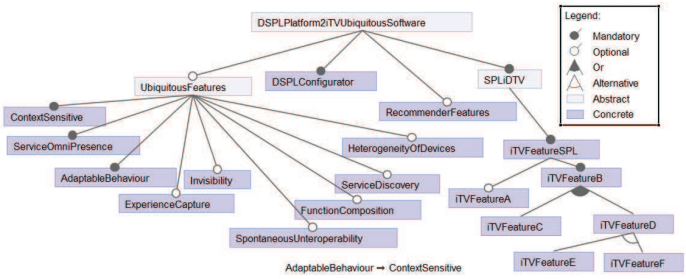

Figura 3: DSPL2UbiTV General Feature Model

The developer can extend the functionality of the platform for the supporting features of ubiquitous software and recommendation techniques used. Yet, regarding to the selection of features of the platform, it is not dynamic, i.e., it is determined by the developer. Only SPLiDTV own treatment of features selection dynamically.

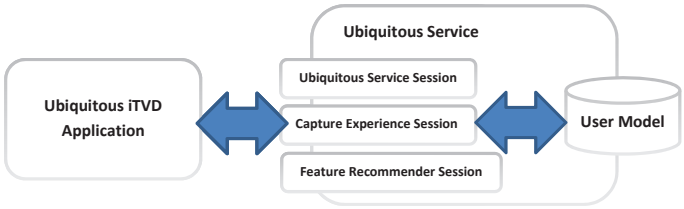

Figura 4: DSPL2UbiTV General Architecture

Moreover, it is necessary to present the general architecture of the platform. This architecture (Figure 4) is represented by two main blocks:

Ubiquitous Services: This is a set of features responsible for keeping some basic ubiquitous services (that can be extended) centrally and remotely. The ubiquitous services provided by the platform are shown in Table (2):

Moreover, one can identify the main information contained in the User Model (Figure 5):

- Information regarding the current state of interaction with the application (such as: interactive media that is already available and discarded by the user, if the user changes context);

Ubiquitous Sessions	Description
Ubiquitous Service	It keeps a session available for any information (as control variables, objects, lists, etc.) required for maintenance of the software ubiquity related to the interactivity service.
Capture Experience	It keeps a session available for any information concerning the learning/knowledge of the experiences of ubiquitous user interaction with the application, added the information about the context of the interaction.
Feature Recommender	This is a session able to recommend the execution of a feature given its execution context.

Tabela 2: DPSL2UbiTV basic Ubiquitous Services

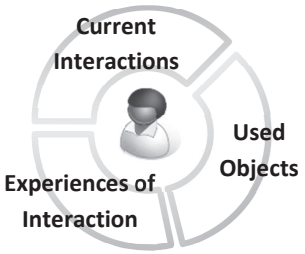

Figura 5: User Modeling

- Information about objects used by the application (e.g.: product added to a virtual shopping cart);
- Information about user's interaction with the interactive media (features), as well as the context in which information was used or not (e.g.: knowledge regarding the use or disposal of interactive media in different operating environments).

Ubiquitous iDTV Application: This is the set of features responsible for dynamically controlling the execution, in each operating environment. The block **Ubiquitous iDTV Application** interacts with the **Ubiquitous Service** remote block in order to seek information related to the presence of the interactive service. However, by relying on connections to the communication networks for access services, the platform has mechanisms to build the best application for the user in the absence of network connection. Some mechanisms used are highlighted in the session that covers the developing process.

4.2. DSPL2UbiTV Development Process (FOSD)

The Feature-Oriented Software Development (FOSD), in which the DSPL2UbiTV platform is based on, has a development process that involves: (1) Domain Analysis, (2) Specification and Design Domain, (3) Domain Implementation (4) Configuration and Generation of Software Products.

The following sections present a discussion, under a practical view, about what the Platform DSPL2UbiTV does in each phase of the FOSD process.

4.2.1. Domain Analysis

During the Domain Analysis phase is where the Feature Model of an SPL is set , Apel et al. (2013). In this regard, the platform DSPL2UbiTV provides a set of tags (Table 3) to create an XML file in which the Feature Model will be represented.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<featuremodel>
<features>
    <mainfeature>
      <name>iTVFeatureSPL</name>
      <relationship>MANDATORY</relationship>
    </mainfeature>
    <feature>
      <name>iTVFeatureA</name>
      <relationship>OPTIONAL</relationship>
<parentFeature>iTVFearureSPL</parentFeatures>
    </feature>
    <feature>
      <name>iTVFeatureB</name>
      <relationship>MANDATORY</relationship>
<parentFeature>iTVFeatureSPL</parentFeatures>
    </feature>
    <feature>
      <name>iTVFeatureC</name>
      <relationship>OR</relationship>
          <priority cardinality="1">true</priority>
<parentFeature>iTVFeatureB</parentFeatures>
          <constraints>
              <require>
<nameFeature>iTVFeatureA</nameFeature>
              </require>
          </constraints>
    </feature>
```

```
    <feature>
      <name>iTVFeatureD</name>
      <relationship>OR</relationship>
          <priority cardinality="1">false</priority>
      <parentFeature>iTVFeatureB</parentFeatures>
          <constraints>
              <exclude>
<nameFeature>iTVFeatureA</nameFeature>
              </exclude>
          </constraints>
    </feature>
      <feature>
      <name>iTVFeatureE</name>
      <relationship>ALTERNATIVE</relationship>
          <priority cardinality="1">false</priority>
      <parentFeature>iTVFeatureD</parentFeatures>
    </feature>
      <feature>
      <name>iTVFeatureF</name>
      <relationship>ALTERNATIVE</relationship>
          <priority cardinality="1">true</priority>
      <parentFeature>iTVFeatureD</parentFeatures>
    </feature>
  </features>
</featuremodel>
```

Figura 6: XML Feature Model

From the Feature Model represented through specific tags, the mandatory feature **DSPLConfigurator** deal with configuration, generation and controll of SPLiDTV.

Figure 6 has the representation in XML of the SPLiDTV shown in Figure 3.

Thus, once the structure is defined Feature Model through metadata, structural changes involving relationships, constraints, etc. can be made and extended with easiness. Furthermore, the platform allows defining sub-models which are used given an execution context, i.e., you can set Feature Models specific to different execution environments such as HDTV, DTV, smartphones, PCs, tablets, etc.

4.2.2. Domain Design and Specification

In the context of FOSD, the Domain Design and Specification involves the treatment of the essential structural and behavioral properties of the involved resources, specified using a formal or informal specification and/or modeling language. This work used the UML modeling language. Thus, the modeling will be presented below, using the class diagrams, for some of the main software components of the platform.

ITVFeature Class Design

The class *ITVFeature* is the most important class of all DSPL2UbiTV platform. This is the basis for implementation of interactive TV content as well as the element that represents the features of a SPL in a FOSD development.

The class *ITVFeature* (Figure 7) implements the XLet JavaDTV API interface (API supported in the main middleware

TAGS	Description
featuremodel	TAG root identifies a Feature Model.
features	Linked to TAG **featuremodel**, identifies the set of Features in the Feature Model.
mainfeature	Linked to TAG **features**, identifies the root iTVFeature in the Feature Model.
feature	Linked to TAG **features**, identifiies the iTVFeature in the Feature Model.
relationship	Linked to TAG **feature**, identifies the relationship between a Feature and the Parent (*parentFeature*). Relationships can be MANDATORY, OPTIONAL, OR and ALTERNATIVE.
priority	Linked to TAG **feature**, features identified in cases of elective (OPTIONAL, ALTERNATIVE and OR) if the Feature should be forfeited in case of absence of an implemented decision mechanism. This TAG has a property called ***cardinality*** used when the optional Features (ALTERNATIVE and OR) need to identify more than one Feature to be selected for relationships.
parentFeature	Linked to TAG **feature**, identifies the Parent Feature TAG.
constraints	Linked to TAG **feature**, it is the TAG used to identify constraints linked to a Feature.
require	Linked to TAG **constraints**, identifies Features required by Feature that has the restrictions
exclude	Linked to TAG **constraints**, identifies Features excluded by Feature that has the restrictions
nameFeature	Linked to TAGs **require** and **exlcude**, identifies the feature in which the constraint is applied.

Tabela 3: Feature Model TAG's

available worldwide, e.g. MHP and Ginga). Thus, one *ITVFeature* has a life cycle similar to a XLet, providing methods for initialization, execution, break and destruction of *ITVFeature*. In addition, the class *ITVFeature*, to implement the Observer class, acts as a observer waiting notifications for exchanging contexts.

For the sake of supporting the construction of feature models, the Class *ITVFeature* also allows identifying a child class from a parent class. Therefore, it is possible to construct a feature model instantiating and interrelating instances of the class *ITVFeature*.

ITVFeature has also aggregated objects that provide the interface to ubiquitous services: UbiServiceSession, CaptureExperienceSession and RecommenderFeatureSession. Furthermore, it allows the deployment, by the software developer, the methods **verifyBootableFeature()** (used to assess the possibility of implementing the *ITVFeature*) and *verifyRecommendedFeature()* (used to assess whether implementation of **ITVFeature** is recommended within the software product configured).

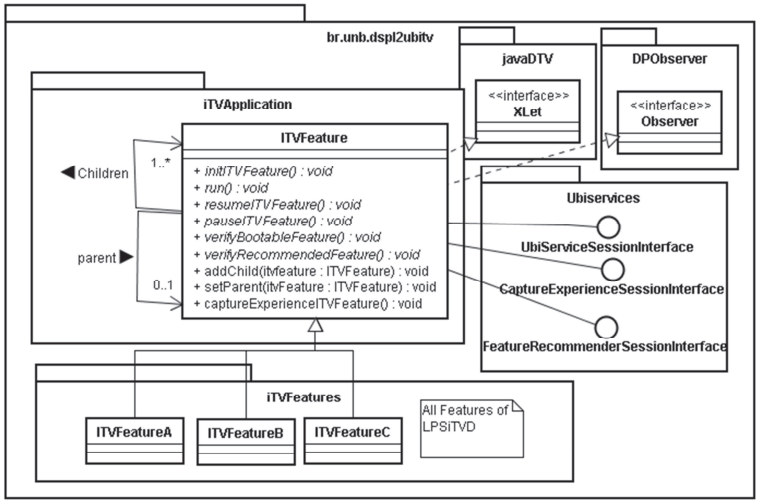

Figura 7: ITVFeature Class Diagram

Finally, the software developer must implement class inheritance *ITVFeature* for all features of the Feature Model.

Monitor Classes Design

Since applications developed from the platform DSPL2UbiTV must be able to self-adapt to different execution environments, it is necessary to monitor context switches and evaluate the need for adaptation the application. Figure 8 shows a Class Diagram containing classes relevant for task execution monitoring.

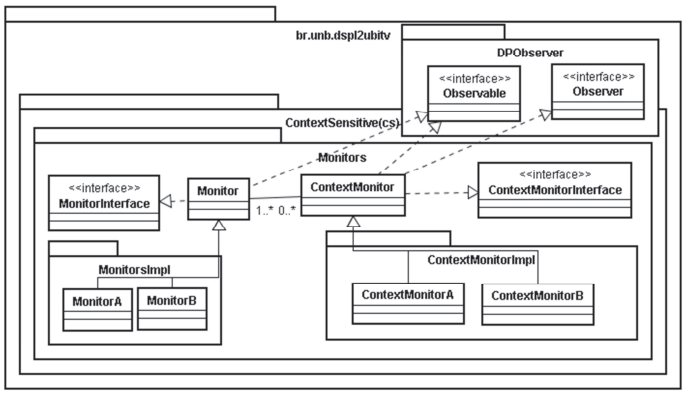

Figura 8: Class Diagram related Context Sensitive

It is necessary to monitor changes in context; in this sense, initially it monitors the values of variables related to context, in order to identify, for example, which wireless networking technology (WiFi, 3G, 4G, etc.) is being used on the device at a certain time. Thus, the software developer must extend the class *Monitor* in order to evaluate the status of several contextual variables (it is recommended the creation of a class of monitoring for each variable context considered).

However, various contextual variables may be needed, and from the combination of their status, to determine a change in context. Then, it is necessary, from the software developer, to extend the class *ContextMonitor* considered for each context. In the classes that inherit from *ContextMonitor* the developer must evaluate, based on a set of contextual variables, the effective exchange of context for a response from the application.

Furthermore, the class *Monitor* implements the *Observable* interface, which represents its instances of objects that can be observed regarding the exchange of the values of monitored contextual variables. In practice, the classes *Observable* notify classes *Observer* when identifying some change. The class *ContextMonitor*, in turn, implements the *Observer*, i.e., acting as an interface class observer to change the status of contextual variables.

Finally, the class *ContextMonitor* also implements the interface *Observable*, in order to be able to notify the application of a significant change in context. The application in this case is represented by the class *ITVFeature* (which implements the interface *Observer*). Therefore, when a significant change is identified by means of the monitoring context classes, classes that extend *ITVFeature* are reported to self-evaluate their execution conditions.

Multidimensional Context Design

The multidimensional model of contexts provides a conceptual structure of information to be recorded and updated, given a set of contextual dimensions interrelated. For example, if the user is in a certain location, time of the day, day of the week, using a particular device, which would be likely to be used to interact with the TV content? I.e., the location, day of the week, time of the day and type of device are treated as dimensions in multidimensional model.

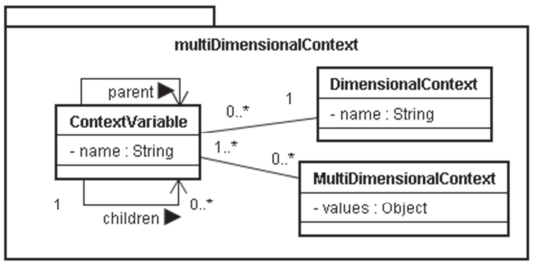

Figura 9: Multidimensional Context - Class Diagram

In order to provide persistence/accounting information in such DSPL2UbiTV Platform, it is necessary to represent such information through objects. Figure (9) shows a diagram of the classes that implement the multidimensional model. Such classes are detailed below:

DimensionalContext - This class represents the information of the dimension of a context. There is an attribute called *name* that can receive values such as ”time of day”, ”local”, ”ITVFeature”, etc;

ContextVariable - each object of this class has information about which dimension it belongs (e.g., time of day), its name (e.g., morning) and hierarchical information (as an example: the periods of the day are linked to days of the week, which in turn are linked to months);

MultiDimensionalContext - the object instantiated from the class *MultiDimensionalContext*, has an attribute called *values* which has accounting information related to multidimensional context. Moreover, it has the possibility of relationship with several instances of the object *ContextVariable* to actually identify how many dimensions is has.

Thus, the classes of the multidimensional model are used to the implementation of Ubiquitous services, with the purpose of learning contextual information and recommendations.

Ubiquitous Service and Session Design

The ubiquitous services are based on four (4) core classes/interfaces (Figure 10):

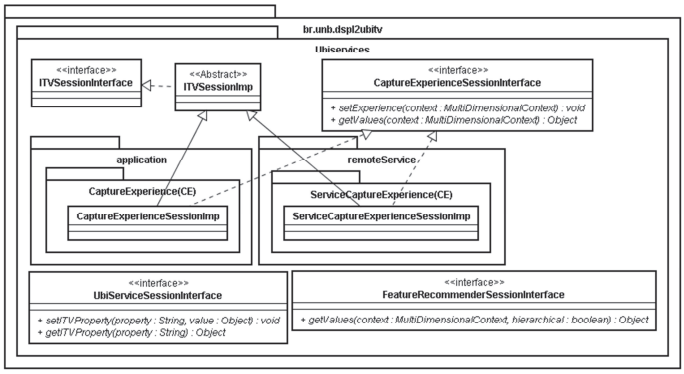

Figura 10: Capture Experiences - Class Diagram

ITVSessionImp - Responsible to allow the session control services for the ubiquitous class, allowing the usage of the session during the life cycle of the application, even if the device is changed, if it occurs at a pre-determined period.

UbiServiceSessionInterface - it contains methods (**setProperty()** and **getProperty()**) that can store any object in a session controlled in a remote environment. Thus, it makes possible to initialize an application on another device and retrieve objects used in devices interactions used recently.

CaptureExperienceSessionInterface - This interface provides the method **setExperience()** used to storing interaction experiences once identified context, besides the method **getValues()** used to fetch the values recorded so far.

FeatureRecommenderSessionInterface - The interface *FeatureRecommenderSessionInterface* provides a method to find objects with information concerning the given recommendation in a context. The method *getValues()* receives

as parameters the multidimensional context and information regarding the consideration of a contextual hierarchy to generate the recommendations.

Importantly for the implementation of ubiquitous services, the class that implements the interfaces session should also extend the *ITVSessionImp* class, since all control sessions (as lifetime, creation and invalidation, authentication) are implemented in this class.

4.2.3. Domain Implementation

During Domain Implementantion phase, the codes are implemented. In this sense, this section will present the main classes of the platform, as well as which classes can/should be extended and added to the codes.

We propose the creation of a SPLiDTV in 6 (Sixth) steps:

- First Step - Defining Features of SPLiDTV
- Second Step - Implementing Context Sensitivity
- Third Step - Implementing OmniPresence Service
- Fourth Step - Implementing Experience Capture
- Fifth Step - Implementing Recommendation of Features
- Sixth Step - Implementing Heterogeneity of Devices

First Step - Defining Features of SPLiDTV

The first step to use the platform is to develop the Features that will comprise the SPLiDTV. These features are developed so that decisions concerning the organization, relationships and constraints of the Feature Model are determined by other mechanisms (Figure 6). Therefore, to develop an adherent feature platform it is necessary to extend the ITVFeature class (Table 4).

```
1 public abstract class ITVFeature implements Xlet,
Observer, ITVFeatureInterface, ActionListener {
...
2 public static UbiServiceSessionInterface ubiTVsession =
new UbiServiceSessionImp();
3 public static CaptureExperienceSessionInterface cesession
= new CaptureExperienceSessionImp();
4 public static FeatureRecommenderSessionInterface
recommendersession = new FeatureRecommenderSessionImp();
...
5 public void initITVFeature(){ ...}
6 public void run(){ ... }
7 public void pauseITVFeature() { ... }
8 public void resumeITVFeature() { ... }
...
}
```

Tabela 4: ITVFeature Class

The ITVFeature class has instances of objects that allow the use of ubiquitous services, learning and recommendation (lines 2, 3 and 4 of Table 4) sessions. Importantly, the Session objects are static, i.e., they are shared by all Features that extend ITVFeature class.

Moreover, ITVFeature class implements the necessary methods for the XLet (class that represents an interactive medium in JavaDTV package) class and provides methods to be overwhelmed by their child classes. The main methods that should be overridden by ITVFeature class are:

initITVFeature() In Line 5, the **initITVFeature()** method should be used for booting from internal objects to Feature;

run() In line 6, the **run()** method should be used to implement the graphical interface and operational requirements of the feature. The method **run()** is the only method that should mandatorily be overwritten;

pauseITVFeature() In line 7, the **pauseITVFeature()** method should be used to determine the actions performed if a Feature pass to the state of paused. This situation can occur at run time when the change of context lead to some impediment to implementing Feature;

resumeITVFeature() In line 8, the **resumeITVFeature()** method should be used for the necessary actions when a change of context occurs, and a Feature is reset.

In practice, only a feature identified as *MainFeature* (SPL root) should extend the ITVFeature class. The rest of the SPLiDTV Features should extend MainFeature. The MainFeature (Table 5) is responsible for providing the chart container used in SPLiDTV (lines 1,3 and 6) and the initialization of the execution of SPLiDTV (line 7).

```
1 import com.sun.dtv.lwuit.Form;
2 public class ITVFeatureSPL extends ITVFeature {
3 protected static Form form = new Form();
4 @Overide
5 public void run() {
6 form.show();
7 super.startITVFeature(); }
8 }
```

Tabela 5: MainFeature Class

In table (6) one can observe an implementation example of a feature using DSPL2UbiTV platform. It is observed on line 1 that the class extends the *MainFeature*. Moreover, one can observe the implementation of the methods **resumeITVFeature()** and **pauseITVFeature()**, where the implementation of such methods is important for the dynamic reconfiguration of the SPL is appropriate, as well as support ubiquitous feature of projects called **Adaptive Behavior**.

Once defined and implemented all features, the developer must define the feature model according to the example shown in Figure (6).

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 Button button;
3 public void run() {
4 button = new Button("Old Text");
5 try{ button.setIcon(...);
6 button.setSize(new Dimension(60,60));
7 }catch(Exception e){}
8 button.addActionListener(this);
9 form.addComponent(BorderLayout.WEST, button); }
10 public void resumeITVFeature() { form.repaint(); }
11 public void pauseITVFeature(){
12 form.removeComponent(button); form.repaint();
... } }
```

Tabela 6: ITVFeatureA Class

Second Step - Implementing Context Sensitivity

The implementation of context awareness must consider two sets of classes:

- classes used for checking the status of implementation of the contextual variables. For example, classes that implement the interface ContextTestInterface and RulesInterface (used to implement the procedures for the identification of status);
- classes used to maintain the status check while Features are running. For example, the Monitor class and ContextMonitor class, which are the classes that should be extended by developers.

The codes that extend the Monitor and ContextMonitor classes should use, for verification purposes of the current status of context, implementations of ContextTestInterface and RulesInterface interfaces.

The Monitor class implements the MonitorInterface and Observable interfaces. Thus, the classes that represent the temporal situation of monitors check the status change of context variables and notify the Observer classes when this status change. On the other hand, the ContextMonitor class implements the Observer and ContextMonitorInterface interfaces, i.e. classes that inherit from ContextMonitor should observe the status changes occurring and identified by classes that inherit from the Monitor class.

Furthermore, ContextMonitor class checks whether the monitors status changes observed are significant to the point to represent a context switch. If identified context switch, this class notifies the feature classes that extend ITVFeature. Thus, ContextMonitor class also implements the interface Observable and ITVFeature implements the Observer interface.

In practice, developers must extend the Monitor class to develop the monitoring functional related to contextual variables (e.g., periods of the day: morning, afternoon and evening) and must implement two methods: **runMonitor()** and **isChangeStatus()**, where **runMonitor()** must implement the verification of context variables. The identification of exchange status is implemented by the method **isChangeStatus()** (Table 7).

```
1 public class TimeMonitor extends Monitor{
2 ...
3 public boolean isChangeStatus()
4 { ... }
5 public void runMonitor()
6 { ... }
7}
```

Tabela 7: Extension of the Monitor class

Moreover, by extending the class ContextMonitor developers should implement only one method. This is the method **verifyChangeContext()** responsible for verifying the effective exchange of context for subsequent notification to extensions ITVFeature (Table 8).

```
1 public class ContextMonitorLocalTime extends ContextMonitor{
2 ...
3 public boolean verifyChangeContext()
4 { ... }
5}
```

Tabela 8: Extension of the ContextMonitor class

Once identified an exchange of context, the notified feature (extension of the ITVFeature), enter into a process of self-reassessment of the status of execution being, then, able to become active or inactive.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<CS>
<monitors>
  <monitor>
    <class>TimetMonitor.class</class>
    <time>12343414</time>
     <observers>
       <observer>ContextMonitorLocalTime.class</observer>
    </observers>
  </monitor>
  <monitor>
    <class>LocalMonitor.class</class>
     <time>12343414</time>
     <observers>
       <observer>ContextMonitorLocalTime.class</observer>
    </observers>
  </monitor>
<monitors>
<contextMonitors>
 <contextMonitor>
  <contextName>LocalTime</contextName>
  <class>ContextMonitorLocalTime.class</class>
          <observers>
   <observer>ITVFeatureB.class</observer>
  </observers>
 </contextMonitor>
<contextMonitors>
</CS>
```

Figura 11: XML Configuration Monitor

The organization which monitors the variables which are observed by monitors of context and the definition of Feature which observes a particular context monitor is defined by a metadata (Figure 11). In addition, the metadata is defined by TAG < *time* > , the time interval for checking the exchange of context (in milliseconds).

```
 1 public abstract class ITVFeature implements Xlet,
Observer, ITVFeatureInterface, ActionListener {
2 ...
3 public void verifyBootableFeature()
4 { ...  this.setBootableFeature(true); ...  }
5 ...
6 }
```

Tabela 9: ITVFeature Class - Implementing **verifyBootableFeature()** method

Finally, it is necessary to highlight the need to implement the method **verifyBootableFeature()** (Table 9). This method should identify, through checks on contextual variables, if there is condition (computational resources) for implementing Feature initialized. After verification, the developer must set the **setBootableFeature()** method to true or false.

Third Step - Implementing OmniPresence Service

The implementation of the DSPL2UbiTV Platform considers omnipresence of the interactive service. Thus, besides the possibility of controlling Features which are most suited to the user (given its context), the platform provides a mechanism for session control where the objects used by the application can be kept in a remote environment and accessed in its most current status in case the user replace the receiver equipment.

Thus, for example, if the viewer is using a T-Commerce application (on a conventional TV receiver) and have added a product to the shopping cart, when migrating to another receiver (Mobile TV) DSPL2UbiTV platform must identify the new context, configure the application properly, and seek information that is maintained in the session in order to maintain user interactivity similar to the previous equipment.

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 ShoppingCart cart; ...
3 public void initITVFeature(){ ...
4 if(!ubiTVsession.isNew()){
5 cart = (ShoppingCart)ubiTVsession.getITVProperty("cart"));
6 ...}...}
7 public void run() {
8 ...
9 cart.addToCart(book,1);
10 ...
11 } ...  }
```

Tabela 10: ITVFeatureA Class - Omnipresence example

Table (10) exemplifies, lines 4,5 and 6, how ITVFeature application can fetch information regarding recent interactivity and automatically (line 9) keep them in the new operating environment.

Fourth Step - Implementing Experience Capture

In this study, the capture of experiences involves the knowledge acquisition about the features that are most used in the different contexts. Thus, the configuration and reconfiguration of the software product from the SPLiDTV may become more appropriate.

For the software developer who wants to implement such a feature, it is necessary to implement the **captureExperienceITVFeature()** method for each Feature in which the goal is to obtain knowledge about interactivity. In the current version, DSPL2UbiTV Platform build a graph that represent the interaction events handled by ActionListener interface, since the class ITVFeature implements the required method (actionPerformed()) for the treatment of the event (Table 11). In the mentioned table it can be seen (in line 11) the counter for interaction with each Feature.

```
1 public abstract class ITVFeature implements Xlet,
Observer, ITVFeatureInterface, ActionListener {
...
2 @Override
3 public void actionPerformed(ActionEvent arg0) {
4 this.captureExperienceITVFeature();
5 DimensionalContext feature = new DimensionalContext();
6 feature.setName("FEATURE");
7 ContextVariable thisfeature = new ContextVariable();
8 thisfeature.setName(this.getClass().getSimpleName());
9 thisfeature.setContext(feature);
10 this.getCeContext().addContextVariable(thisfeature);
11 this.getCeContext().setValues(((Integer)this.getCeContext()
.getValues()+ 1));
12 cesession.setExperience(this.getCeContext()); }
...
}
```

Tabela 11: ITVFeature Class - Implementing method **actionPerformed()**

In addition, during the implementation of **captureExperienceITVFeature()** method (which is the treatment of the interaction event for feature) one can aggregate information to better identify the interaction context (line 16) because the ITVFeature class has limited knowledge about what feature interaction has received (Table 12). Another important detail is that the feature itself should include the list of actions waiting for an event (line 11).

Being the context identified through a multidimensional perspective, it is possible to evaluate how many dimensions would be necessary to obtain information related to interactivity and user preferences. Moreover, as the service of experience capture is remotely located, it is possible to implement collaborative filtering mechanisms between service users and thus progressively improve the learning process about prefe-

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 Button button;
3 public void run() {
4 button = new Button("Old Text");
5 try{ button.setIcon(...);
6 button.setSize(new Dimension(60,60));
7 }catch(Exception e){}
8 button.addActionListener(this);
9 form.addComponent(BorderLayout.WEST, button); }
10 ...
11 public void captureExperienceITVFeature() {
12 DimensionalContextInterface dctime = new
DimensionalContextImp();
13 dctime.setName("TIME");
14 ContextVariableInterface cvsunday = new
ContextVariableImp();
15 cvsunday.setName("SUNDAY"); cvsunday.setContext(dctime);
16 this.getCeContext().addContextVariable(cvsunday);
17 button.setText("New Text"); }
18 ... } }
```

Tabela 12: ITVFeatureA Class - Implementing method **captureExperienceITVFeature()**

rences. It is considered that users preferences will be identified at the outset of the construction process of the software product. Thus, the basic model of user information is based on preference information obtained from user interaction and data contexts.

Finally, it is recommended to add context information to the Feature methods in *initITVFeature()* and *resumeITVFeature()*, where identification of the contexts must be obtained through Context Test classes in the package found in Context Sensitivity (CS).

Fifth Step - Implementing Recommendation of Features

The recommendation of features is required in situations where features are not defined as mandatory in Feature Model. Once the selection of features is made at runtime and automatically, a recommendation system for features has been implemented to support the decision. In the current version of the platform, the recommendation system is based on context awareness , Adomavicius et al. (2005), Adomavicius and Tuzhilin (2001). Thus, using the information collected through the implementation of the characteristic of ubiquity named **Capture Experiments**, one can obtain recommendations on which features should be selected.

Recalling that the control over whether or not the feature will be part of the software product, at a given moment, is made by on feature. In this sense, as in the case of checking whether the feature is appropriate for a given context, there is a method to be implemented in features that identifies the feature is recommendable or not for the software product to be configured. This is the method **getListRecommender()** that must be implemented by developers. The method **getListRecommender()**, exemplified in Table (13) returns an object of type IndexRecommended (line 3). The IndexRecommended class has two main attributes, the first being the index the feature recommendation, if there are several features disputing the execution (e.g. with relationships ALTERNATIVE). The second attribute of cardinality is how many features can be performed.

```
1 public class ITVFeatureA extends ITVFeatureSPL{
2 ...
3 public IndexRecommended getListRecommender(ITVFeature
feature){
4 DimensionalContext dctime = new DimensionalContext();
5 dctime.setName("TIME");
6 ContextVariable cvsunday = new ContextVariable();
7 cvsunday.setName("SUNDAY"); cvsunday.setContext(dctime);
8 this.getCeContext().addContextVariable(cvsunday);
9 this.index = (IndexRecommended)
this.getFrsession().getValues(this.getCeContext(), false);
10 this.index.setCardinality(1);
11 return index;
12 ... } }
```

Tabela 13: ITVFeatureA Class - Implementing method **getListRecommender()**

```
 1 public abstract class ITVFeature implements Xlet,
Observer, ITVFeatureInterface, ActionListener {
...
2 @Override
3 public void verifyRecommendedFeature(){
4 DimensionalContext feature = new DimensionalContext();
5 feature.setName("FEATURE");
6 ContextVariable thisfeature = new ContextVariable();
7 thisfeature.setName(this.getClass().getSimpleName());
8 thisfeature.setContext(feature);
9 this.getCeContext().addContextVariable(thisfeature);
10 if(noParent)
11 { this.index = this.getListRecommender(this); }
12 else
13 { this.index = parent.getListRecommender(this);}
14 if(this.index.getCardinality() >=
this.index.getIndexOf())
15 { this.recommended = true;}
16 else
17 { this.recommended = false; } }
...
}
```

Tabela 14: ITVFeature Class - Implementing method **verifyRecommendedFeature()**

The object of type IndexRecommended may be returned, too, when queried the session (frsession) used to make recommendations on a particular context (line 9, Table - 13). The context will always contain information on the feature assessed.

Once obtained the object of type IndexRecommended through session **frsession**, the method **verifyRecommendedFeature ()**, implemented in ITVFeature (Table 14) class checks

if the feature should be performed according to the following criteria: If the value for the attribute cardinality is greater or equal to the value obtained for the attribute index (IndexOf) the feature must be executed.

Thus, it is up to the developer to implement the classes of rules and to determine the index values and cardinality considered appropriate.

Finally, it is recommended to add context information to the feature methods in **initITVFeature()** and **resumeITVFeature()**, where the identification of contexts must be obtained through the Context Test classes found in package Context Sensitivity (CS).

Sixth Step - Implementing Heterogeneity of Devices

A Software Product Line has as main feature the ability to configure multiple software products distinct (but with similarities) from an SPL.

To meet the demand for construction of applications that have support for multiple devices, SPL provides the possibility of implementing Features specific to certain devices and, at the time of the product for a given device configuration, select them for composition of the product appropriately.

Thus, it is understood that the platform meets the characteristic of ubiquitous software projects called **Heterogeneity of Device**, since the features and controls configuration of the software product to be developed properly.

4.2.4. Product Configuration and Generation

As mentioned previously, configuration of a product software is based on selection of features available in a software product line. In the case of interactive Digital TV applications, the product selection, if it occurs in advance, it couldn't be able to run on some devices due to limitations relating to processing, resolution, interaction mechanisms, etc.

Therefore, when using Software Product Lines in development of interactive applications for Digital TV, there is a necessary of using a configuration mechanism and product generation software at runtime, i.e., using Dynamic Software Product Lines.

In this paper, the configuration and reconfiguration of the product (given a SPLiDTV), distributes the task of selecting the Features. In practice, each *ITVFeature* controls their life cycle and of their dependents.

Each *ITVFeature* self-assesses regarding three situations that can prevent it from running:

- Self-checks whether the environment has sufficient resources for execution of ITVFeature;
- Self-checks whether the ITVFeature has execution recommended;
- Self-checks if there is a restriction that prevents its execution, such as having implementation, depending on the execution status of an other ITVFeature.

Thus, the dynamic product configurator software, initializes all *ITVFeatures* randomly and each self-assessment in order to transition to a possible execution state. When describing the transition from one state to another, it can be seen that there is a life cycle for a *ITVFeature*. This life cycle is very similar to the life cycle of a *XLet* (object that represents a media in Interactive Digital TV, implemented in Java). Thus, a *ITVFeature* can be found in the following states:

Loaded - the method **initITVFeature()** is used in the initialization of *ITVFeature*;

Started - the method **run()** is used in the execution of *ITVFeature*;

Paused the method **pauseITVFeature()** is used to pause a *ITVFeature*;

Destroyed - the method **destroyITVFeature()** is used to shut down any activity of a *ITVFeature*;

Reloaded - the method **resumeITVFeature()** is used to restart an *ITVFeature*. In real practice, an *ITVFeature* back to the **Loaded** state, time in which self-assessment considers a possibility to change to the running state.

Figure (12) presents the possibilities of state transition of an *ITVFeature*.

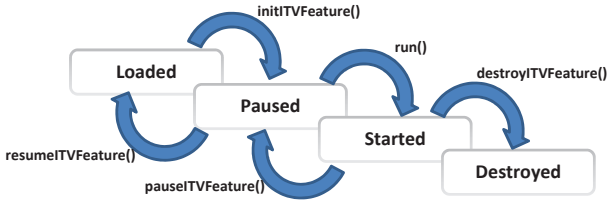

Figura 12: ITVFeature - Life Cycle

Whereas this work uses an feature oriented programming, the *ITVFeatures* are organized through a Feature Model. Thus, each *ITVFeature* may depend on a *ITVFeature* at a higher hierarchical position, and may have *ITVFeatures* dependent. In such cases, when a *ITVFeature* enters to the paused, booted, rebooted and destroyed state, all dependents must enter the same state of *ITVFeature* which are depended.

Anyway, considering the characteristics of implementation of *ITVFeatures* and their organization within a Feature Model, it is understood that the DSPL2UbiTV Platform implements the characteristic of projects **Adaptive Behavior** that must be presented in an ubiquitous software.

5. Experiments and Results

Initially, it is necessary to emphasize that the main contribution of this work is to offer a platform for developing ubiquitous applications in the Interactive Digital TV environment, considering that interactive Digital TV has particular characteristics of the transmission and execution of interactive content.

Moreover, mechanisms such as recommendation systems, self-configuration and user modeling should be used to implement the solution.

Thus, considering the tool as a solution to support the development of software, weÍl evaluate the tool in accordance with attributes and metrics of software quality. In the context of this work, the quality attribute evaluation is focused on software maintenance.

This section presents the methodology used for the experimentation, as well as, software quality metrics considered in the experiments and the results.

5.1. Methodology

The experiment was conducted through a case study , Wohlin et al. (2000). The case study concerns a SPL implemented as shown in Figure 13, involving a Ubiquitous T-commerce application. In SPL it can be observed the possibility of building several separate software products.

Figura 13: TVDeals iDTV Application ScreenShots

The framework evaluation is based on software quality attributes. Thus, based on the GQM approach , Basili (1992), the goal of the experiment is presented in Table 15. To answer the research question, the following methodology was applied:

GOAL Purpose	evaluate
Issue	level of complexity and reusability attributes of
Object	DSPL2iTV framework
ViewPoint	Designers and Developers of software
QUESTIONS	The DSPL2iTV framework provides facilities for software development and reuse ? The implementation of variability across the framework impacts the evaluated quality attributes?
METRICS	CompPLA and DITPLA

Tabela 15: Goal of the Experiments

1. Set the configuration software products from the SPL developed (Figure 14). In this case study, 8 product settings were generated.
2. Collect the CompPLA and DITPLA metrics (Equations 3, 4) for each defined product configuration. The CompPLA metric is defined in , Oliveira Junior et al. (2010), being based on $WMC_{McCabeCC}$ (Weighted Methods per Class based on Cyclomatic Complexity of MacCabe) metric, and aims to reflect the complexity of the software. DITPLA is based on DIT (Depth of Inheritance Tree) metric that aims to reflect the degree of software reuse.
These metrics can be expressed as:

$$CompPLA_{PA} = \sum_{i=1}^{NCpt} CompVarComponent_{Cpt_i} \quad (3)$$

$$DITPLA_{PA} = \sum_{i=1}^{NCpt} DITVarComponent_{Cpt_i} \quad (4)$$

Where: $CompPLA$ represents the metric used for evaluation complexity; $DITPLA$ represents the metric used for evaluation reusability; PA represents the architecture of the selected product; $NCpt$ represents the number of components present in the configured product; $CompVarComponent(Cpt_i)$ is the complexity of PA for the i^{th} PA component; $DITVarComponent(Cpt_i)$ is the reusability of PA for the i^{th} PA component;
3. Collect the percentage of features that represent variability as well as the amount of features that make up the product set.
4. Identify the existence of a linear correlation between the values obtained for: CompPLA, DITPLA, percentage of variable features and the number of features used in each setting. If there is a significant correlation, it is possible to use linear regression to predict future values and identify the impacts that the framework on the evaluated quality attributes.

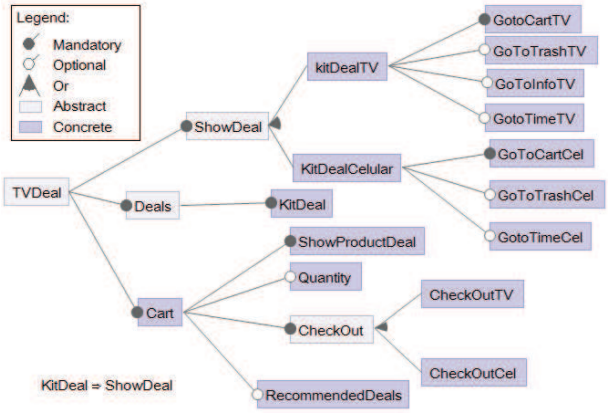

Figura 14: TVDeals Application: *Feature Model*

It should be noted that since the framework is developed using object orientation (Java), the adopted metrics are proposed for evaluation of object oriented codes/designs. Moreover, the codes used in the assessment correspond specifically to the framework and required codes, extended by the developer. Thus, it is possible to isolate the impact of the framework on quality attributes, without the influence of any other values, dependent on developer based codes.

5.2. Results

For the case study, 8 product configurations were constructed. For each configuration, the amount of features involved, the percentage of features variables, CompPLA and DITPLA metrics were collected.

Shapiro-Wilk normality test was applied to the samples, and the distributions of CompPLA and DITPLA were non-normal. The linear correlation of Spearman was applied to the variables collected (Table 16), leading to a strong correlation between the variables considered.

	CompPLA	**DITPLA**
% Variabilities	0,970305	0,969530
Features	0,982668	0,982472

Tabela 16: Linear Correlation of Spearman

Thus, applying multiple linear regression to estimate the CompPLA and DITPLA metrics, we respectively obtain, for a $CI = 0{,}95$ (Confidence Interval), $R^2 = 0{,}9664$ and $R^2 = 0{,}9654$; and following models:

$$Pd_C = 197,6140 + (\beta_1 * -9,9785) + (\beta_2 * 6,7940) \quad (5)$$

$$Pd_D = 27,6137 + (\beta_1 * -5,8615) + (\beta_2 * 3,5232) \quad (6)$$

WherePd_C and Pd_D represent the predicted values of CompPLA and DITPLA metrics, respectively. β_1 and β_2 represent the percentage values of feature variabilities and feature quantities, respectively.

Predictions were made using equations (5) and (6), as shown in figures 15 (for 20 and 100 features) and 16 (considering 50% of variabilities). The results indicated a small impact on quality attributes generated by variable features and a positive impact in the attributes of complexity and reusability using the DSPL2iDTV framework.

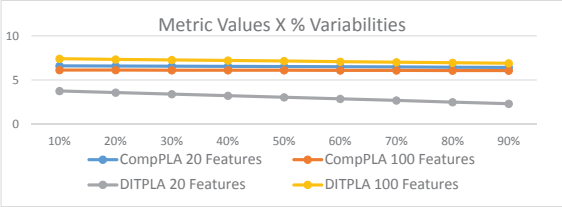

Figura 15: Metric Values X (%) Variabilities - 20 and 100 Features

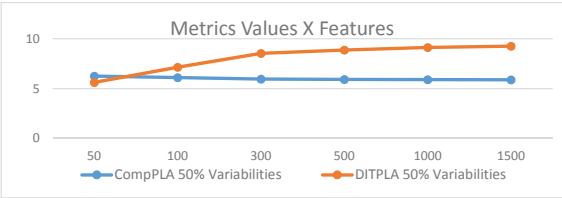

Figura 16: Metric Values X Features - 50% of Variabilities

6. Conclusions and Future Works

This paper has proposed a solution for development of ubiquitous application for iDTV. The solution supports extensibility as well as most of the functional features needed for the software ubiquitous second Spinola et al. (, Spínola and Travassos (2012)) Additionally, this work make use of the concepts of SPL (particularly of DSPL) to offer a DSPL Platform, allowing the development applications with good support to maintainability (complexity) and evolution (extensibility and reuse).

The experiments show that the evaluated architectures provided good results for complexity and reuse, being ranked as low complexity and high reusability. However, the results are close to the thresholds considered normal for the quality attributes evaluated and it can be improved in future versions of the solution by using a greater amount of design patterns so that only Observer, Factory and Singleton standards was used initially.

From the point of view of ubiquity characteristics, the solution can evolve towards a support for non-functional software requirements (integrating aspect orientation to the solution) software such as security, trust computing, etc. From the viewpoint of evaluation solution, new experiments can be conducted in order to evaluate attributes of software quality such as: performance, portability, etc. It would be also interesting to develop experiments to validate the quality of recommendations for the products obtained in various usage scenarios. Finally, from the point of view of iDTV, since the platform has been developed in JAVA (considering the procedural vertent of GINGA, GINGA-J), it is relevant to implement part of the solution (DSPL Application) using the NCL / Lua , Soares et al. (2010) languages , Ierusalimschy (2006) (considering the declarative vertent of GINGA, GINGA-NCL) for full adaptation to the Brazilian middleware Ginga.

7. References

Abowd, G. D., 1999. Software engineering issues for ubiquitous computing. In: Proceedings of the 21st international conference on Software engineering. ICSE '99. ACM, New York, NY, USA, pp. 75–84.
URL http://doi.acm.org/10.1145/302405.302454

Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A., 2005. Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst. 23 (1), 103–145.
URL http://doi.acm.org/10.1145/1055709.1055714

Adomavicius, G., Tuzhilin, A., 2001. Extending recommender systems: A multidimensional approach. In: In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-01), Workshop on Intelligent Techniques for Web Personalization (ITWP2001. pp. 4–6.

Apel, S., Batory, D. S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Basili, V. R., 1992. Software modeling and measurement: the Goal/Question/-Metric paradigm. Tech. rep., Techreport UMIACS TR-92-96, University of Maryland at College Park, College Park, MD, USA.
URL http://portal.acm.org/citation.cfm?id=137076

Benavides, D., Segura, S., Ruiz-Cortés, A., 2010. Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35 (6), 615–636.
URL http://dx.doi.org/10.1016/j.is.2010.01.001

Bencomo, N., Hallsteinsen, S., de Almeida, E. S., 2012. A view of the dynamic software product line landscape. Computer 45 (10), 36–41.

Bosch, J., 2002. Maturity and evolution in software product lines: Approaches, artefacts and organization. In: Proceedings of the Second International Conference on Software Product Lines. SPLC 2. Springer-Verlag, London, UK,

UK, pp. 257–271.
URL http://dl.acm.org/citation.cfm?id=645882.672252

Caceres, R., Friday, A., 2012. Ubicomp systems at 20: Progress, opportunities, and challenges. IEEE Pervasive Computing 11 (1), 14–21.
URL http://dx.doi.org/10.1109/MPRV.2011.85

Cetina, C., Fons, J., Pelechano, V., 2008. Applying software product lines to build autonomic pervasive systems. In: Software Product Line Conference, 2008. SPLC '08. 12th International. pp. 117–126.

Czarnecki, K., Helsen, S., Eisenecker, U. W., 2005. Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10 (1), 7–29.

Czarnecki, K., She, S., Wasowski, A., 2008. Sample Spaces and Feature Models: There and Back Again. Software Product Line Conference, 2008. SPLC '08. 12th International.

de Freitas, G. B., Teixeira, C. A. C., 2009. Ubiquitous services in home networks offered through digital tv. In: Proceedings of the 2009 ACM symposium on Applied Computing. SAC '09. ACM, New York, NY, USA, pp. 1834–1838.
URL http://doi.acm.org/10.1145/1529282.1529691

de Oliveira, R., Do Prado, A., de Souza, W., Biajiz, M., 2009. Development based on mda, of ubiquitous applications domain product lines. In: Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on. pp. 1005–1010.

Deelstra, S., Sinnema, M., Bosch, J., 2005. Product derivation in software product families: a case study. J. Syst. Softw. 74 (2), 173–194.
URL http://dx.doi.org/10.1016/j.jss.2003.11.012

Fleury, A., Pedersen, J. S., Larsen, L. B., 2011. Evaluating ubiquitous media usability challenges: Content transfer and channel switching delays. In: Marcus, A. (Ed.), HCI (10). Vol. 6770 of Lecture Notes in Computer Science. Springer, pp. 404–413.

Forno, F., Malnati, G., Portelli, G., 2006. Honey: a mhp-based platform for home network interoperability. In: Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on. Vol. 2. pp. 102–110.

Geraci, A., 1991. IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. IEEE Press, Piscataway, NJ, USA.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., Papadopoulos, G. A., 2012. A development framework and methodology for self-adapting applications in ubiquitous computing environments. J. Syst. Softw. 85 (12), 2840–2859.
URL http://dx.doi.org/10.1016/j.jss.2012.07.052

Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K., Abril 2008. Dynamic software product lines. Computer 41 (4), 93 –95.

Hervás, R., Bravo, J., 2011. Towards the ubiquitous visualization: Adaptive user-interfaces based on the semantic web. Interact. Comput. 23 (1), 40–56.
URL http://dx.doi.org/10.1016/j.intcom.2010.08.002

Hussein, T., Linder, T., Gaulke, W., Ziegler, J., 2012. Hybreed: A software framework for developing context-aware hybrid recommender systems. In: User modeling and user adapted interaction.
URL http://dx.doi.org/10.1007/s11257-012-9134-z

Hussein, T., Linder, T., Gaulke, W., Ziegler, J., 2013. Hybreed: A software framework for developing context-aware hybrid recommender systems. User Modeling and User-Adapted Interaction (UMUAI).

Ierusalimschy, R., 2006. Programming in Lua, Second Edition. Lua.Org.

Krueger, C. W., 2006. New methods in software product line practice. Commun. ACM 49 (12), 37–40.
URL http://doi.acm.org/10.1145/1183236.1183262

Kunert, T., 2009. User-Centered Interaction Design Patterns for Interactive Digital Television Applications. Human-Computer Interaction Series. Springer.
URL http://dx.doi.org/10.1007/978-1-84882-275-7

Lin, C.-L., Wang, P.-C., Hou, T.-W., 2009. Classification and evaluation of middleware collaboration architectures for converging mhp and osgi in a smart home. J. Inf. Sci. Eng. 25 (5), 1337–1356.

Maia, O. B., Viana, N. S., Junior, V. F. d. L., 2009. Using the idtv for managing services in the ubiquitous computing environment. In: Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing. UIC-ATC '09. IEEE Computer Society, Washington, DC, USA, pp. 143–148.
URL http://dx.doi.org/10.1109/UIC-ATC.2009.61

Oliveira Junior, E. A., Maldonado, J. C., Gimenes, I. M. S., 2010. Empirical validation of complexity and extensibility metrics for software product line architectures. In: Proceedings of the 2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse. SBCARS '10. IEEE Computer Society, Washington, DC, USA, pp. 31–40.
URL http://dx.doi.org/10.1109/SBCARS.2010.13

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., Wolf, A. L., Wolf, E. L., 1999. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems 14, 54–62.

Parra, C., Blanc, X., Duchien, L., 2009. Context awareness for dynamic service-oriented product lines. In: Proceedings of the 13th International Software Product Line Conference. SPLC '09. Carnegie Mellon University, Pittsburgh, PA, USA, pp. 131–140.
URL http://dl.acm.org/citation.cfm?id=1753235.1753254

Schilit, B., Adams, N., Want, R., 1994. Context-aware computing applications. In: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications. IEEE Computer Society, Washington, DC, USA, pp. 85–90.
URL http://portal.acm.org/citation.cfm?id=1439278.1440041

Soares, L., Moreno, M., De Salles Soares Neto, C., Moreno, M., 2010. Ginga-ncl: Declarative middleware for multimedia iptv services. Communications Magazine, IEEE 48 (6), 74–81.

Sommerville, I., 2010. Software Engineering (9th Edition). Pearson Addison Wesley.

Spínola, R. O., Travassos, G. H., 2012. Towards a framework to characterize ubiquitous software projects. Inf. Softw. Technol. 54 (7), 759–785.
URL http://dx.doi.org/10.1016/j.infsof.2012.01.009

Weiser, M., 1999. The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. 3 (3), 3–11.
URL http://doi.acm.org/10.1145/329124.329126

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA.

C PAPER ACEITO PARA PUBLICAÇÃO NO *IJSEA - International Journal of Science and Engineering Applications*

International Journal of Science and Engineering Applications
Volume *– No.*, _____ 2014

A FOSD approach for the development of iDTV Software

Leandro Vaguetti
Electrical Engineering Dept.
University of Brasília
Brasília-DF, Brazil
vaguetti@unb.br

Paulo R. L. Gondim
Electrical Engineering Dept.
University of Brasília
Brasília-DF, Brazil
pgondim@unb.br

Abstract: This paper presents the FOSD (Feature-Oriented Software Development) approach for the development of iDTV applications. The solution is based on a Dynamic Software Product Line - DSPL, and generates software products at runtime in the DTV receiver. The proposed solution is built with the best practices of software engineering, which enable reuse and software evolution.

Keywords: Software Engineering, Interactive Digital Television, FOSD, SPL, DSPL

1. INTRODUCTION

Currently, the interactive applications for interactive Digital Television (iDTV) have become an important resource as a business model for marketing, distance learning, e-commerce, etc. Applications for iDTV allows to associate mechanisms of interactivity to traditional television content. In this sense, it is possible to provide the viewer interactive media (e.g. products for purchase, information, polls, etc) relating to the content watched.

However, in the current scenario, a great deal of development/maintenance of applications for iTVD is required, due to the large amount of television content produced daily, and the need to customize the interaction to each content available. Moreover, the variety of Digital TV receivers available (e.g. Mobile TV, HDTV, etc.) also increases the need for customization of interactivity.

Therefore solutions for the development of applications for iDTV that produce software with sufficient quality attributes so as to minimize the efforts of developing and maintaining applications is required. The solutions for the software development must consider the particular characteristics of the iDTV environment (e.g. broadcast transmission, limited interactivity mechanisms, limited computational resources, etc).

Thus, it becomes necessary to consider the adaptation of interactive contents on reception, considering aspects such as the variation of receivers, the availability of interactivity mechanisms and the interactivity profile of the viewer.

In this sense, this paper proposes the use of an FOSD (Feature-Oriented Software Development) approach, based on SPL (Software Product Lines), to minimize the difficulties faced in the development of interactive applications in DTV scenario. The solution provides a development Platform (Framework) for the building of iDTV features that make up a SPL.

Furthermore, the solution provides a DSPL (Dynamic SPL) Configurator to select features and dynamically generate the software product suitable for the receiver/viewer. The proposed solution is validated by a case study.

This paper is organized as follows. Section 2 presents the FOSD approach for software development to be used in the iDTV domain. Section 3 describes a case study, considering a T-commerce application, while Section 4 discusses some related work. Section 5 presents our final remarks and future work..

2. FOSD Approach to iDTV

iDTV applications have some characteristics linked the need to adjust the content to the receiving equipment. Among the main features are transmission over broadcast channel, different receiving equipment and different mechanisms of interaction with the equipment/contents (Figure 1).

Figure. 1 iDTV Scenario

From the point of view of software engineering, interactive content (interactive applications) must be constructed so as to suit the viewer's needs, as well as details of the DTV environment and enable the adequate maintenance of the software developed. Additionally, interactive applications must offer appropriate performance and usability at any time.

Applications are developed considering the various execution environments, and should optimize the cost associated with the production and software evolution. In this sense, several techniques for software reuse can be found, and one of the most promising approaches is the Software Product Line - SPL. SPL is a set of applications (software products) that have a common architecture and shared components, where each product is specialized so as to reflect the different needs of users or execution environments [1] [2]. Figure (2) shows an

SPL example of iDTV application for the T-commerce domain, with specific features.

An extension of the SPL approach called Dynamic Software Product Lines (DSPL) [3] provides for the selection of software product features in execution time. In this case, in addition to the configuration, reconfiguration of the software product in real time is considered, if context switches or user preference must be set.

This work proposes a FOSD approach for developing interactive applications for digital television, composed of the following parts (or components): (1) an API for the implementation of features; (2) a mechanism for setting the SPL and restrictions; (3) a solution for building dynamic (DSPL) product based software LPS; (4) a solution for self-adapting context-aware applications.

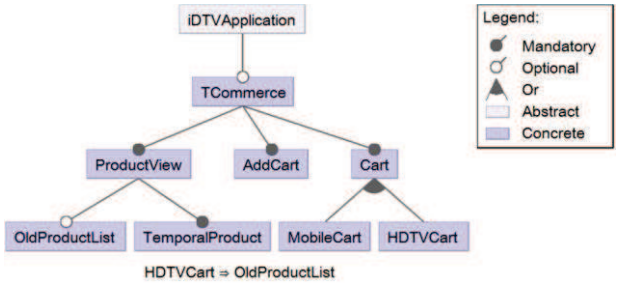

Figure. 2 iDTV Application Example

The proposed solutions are delivered through a FOSD process. The process determines, during the time of application development, the components of the proposed method which should be used. In general terms, the FOSD process consists of the following phases: (1) Domain Analysis, (2) Specification and Design Domain, (3) Domain Implementation, and (4) Configuration and Generation of Software Products.

2.1 Domain Analysis

The Feature Model of an SPL is set in the Domain Analysis phase [4]. The platform provides a set of tags for the creation of a XML file where the will be represented and allowing the configuration, generation and control of SPL.

Once the structure of the Feature Model has been defined through metadata, structural changes involving relationships and constraints can be easily made and extended (Figure 3).

Furthermore, the platform enables the definition of sub-models used in an execution context, i.e., Feature Models specific to different execution environments such as HDTV, DTV, smartphones, PCs and tablets can be set.

2.2 Domain Design and Specification

In the context of FOSD, the Domain Design and Specification phase involves the treatment of the essential structural and behavioral properties of the involved resources, characterized using a formal or informal specification and/or a modeling language. The UML modeling language was used in our work, and the modeling is presented below by class diagrams, for some of the main software components of the platform.

The *ITVFeature* class (Figure 4) implements the XLet JavaDTV API interface (API supported in the main middleware available worldwide, e.g. MHP and Ginga-J). An *ITVFeature* has a life cycle similar to a XLet, providing methods for initialization, execution, break and destruction of *ITVFeature*.

To support the construction of feature models, the *ITVFeature* class also allows identifying a child class from a parent class. Therefore, a feature model instantiating and interrelating instances of the mentioned class can be constructed.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<featuremodel>
<features>
   <mainfeature>
     <name>iTVFeatureSPL</name>
     <relationship>MANDATORY</relationship>
   </mainfeature>
   <feature>
     <name>iTVFeatureA</name>
     <relationship>OPTIONAL</relationship>
<parentFeature>iTVFearureSPL</parentFeatures>
   </feature>
   <feature>
     <name>iTVFeatureB</name>
     <relationship>MANDATORY</relationship>
<parentFeature>iTVFeatureSPL</parentFeatures>
   </feature>
   <feature>
     <name>iTVFeatureC</name>
     <relationship>OR</relationship>
        <priority cardinality="1">true</priority>
<parentFeature>iTVFeatureB</parentFeatures>
        <constraints>
            <require>
<nameFeature>iTVFeatureA</nameFeature>
            </require>
        </constraints>
   </feature>
   <feature>
     <name>iTVFeatureD</name>
     <relationship>OR</relationship>
        <priority cardinality="1">false</priority>
     <parentFeature>iTVFeatureB</parentFeatures>
        <constraints>
            <exclude>
<nameFeature>iTVFeatureA</nameFeature>
            </exclude>
        </constraints>
   </feature>
     <feature>
     <name>iTVFeatureE</name>
     <relationship>ALTERNATIVE</relationship>
        <priority cardinality="1">false</priority>
     <parentFeature>iTVFeatureD</parentFeatures>
   </feature>
     <feature>
     <name>iTVFeatureF</name>
     <relationship>ALTERNATIVE</relationship>
        <priority cardinality="1">true</priority>
     <parentFeature>iTVFeatureD</parentFeatures>
   </feature>
 </features>
</featuremodel>
```

Figure 3. XML Feature Model

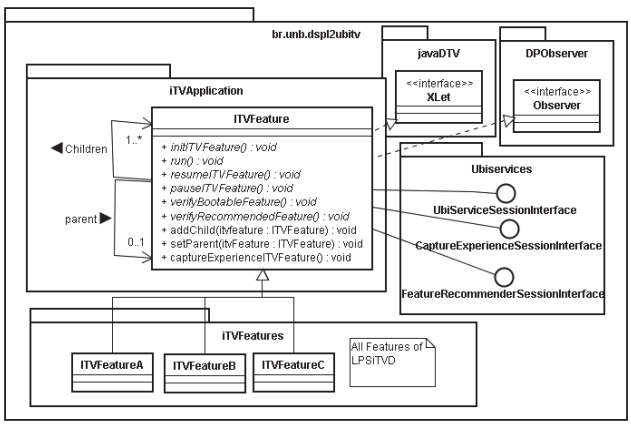

Figure. 4 *ITVFeature* Class Diagram

2.3 Domain Implementation

The codes are implemented in the Domain Implementation phase. This section presents the main classes of the platform. The first step for the use of the solution is the development of the Features that will comprise the SPL. Such features are developed so that decisions concerning the organization, relationships and constraints of the Feature Model can be determined by other mechanisms (Figure 3). Therefore, the *ITVFeature* class must be extended for the development of an adherent feature platform (Figure 5).

```
1 public abstract class ITVFeature implements Xlet,
Observer, ITVFeatureInterface, ActionListener { ...
2 public void initITVFeature(){ ... }
3 public void run() { ... }
4 public void pauseITVFeature() { ... }
5 public void resumeITVFeature() { ... }
6 public void verifyBootableFeature() {... }
7 public void addChild(ITVFeatue itvfeature ) {... }
8 public void setParent(ITVFeatue itvfeature) {... } ... }
```

Figure 5. *ITVFeature* Class Code

Once all features have been defined and implemented, the developer must define the feature model according to the example shown in Figure 3.

2.4 Product Configuration and Generation

As previously mentioned, the configuration of a product software is based on the selection of features available in a software product line. In the case of interactive Digital TV applications, the product selection commonly doesn't occurs in advance, since it couldn't be able to run on some devices due to limitations relating to processing, resolution, interaction mechanisms, etc.

When Software Product Lines are used for the development of interactive applications for Digital TV, a configuration mechanism and product generation software must be used at runtime, thus leading to the convenience of using a Dynamic SPL (DSPL).

In this paper, the configuration and reconfiguration of the product distribute the task of selecting the Features. In practice, each *ITVFeature* controls its life cycle and the life cycle of its dependents.

Each *ITVFeature* self-assesses regarding three situations that can prevent it from running: it self-checks whether the environment has sufficient resources for execution of the *ITVFeature*; it self-checks whether the *ITVFeature* has execution recommended; it self-checks if there is some restriction that prevents its execution, such as having its implementation, depending on the execution status of another *ITVFeature*.

The dynamic product configurator initializes all *ITVFeatures* randomly and each self-assessment for the transition to a possible execution state. When the transition from one state to another is being described, it can be seen that there is a life cycle for an *ITVFeature*. This life cycle is very similar to the life cycle of a XLet (object that represents a media on Interactive Digital TV, implemented in Java). Therefore, an *ITVFeature* can be found in the following states:

- **Loaded** - the **initITVFeature()** method is used in the initialization of *ITVFeature*;
- **Started** – the **run()** method is used in the execution of an *ITVFeature*;
- **Paused** - the **pauseITVFeature()** method is used to pause an *ITVFeature*.
- **Destroyed** - the **destroyITVFeature()** method is used to shut down any activity of an *ITVFeature*;
- **Reloaded** - the **resumeITVFeature()** method is used to restart an *ITVFeature*. In real practice, an *ITVFeature* backs to the **Loaded** state, time at which the self-assessment considers a possibility of changing to the running state.

Figure 6 presents the possibilities of state transition of an *ITVFeature*.

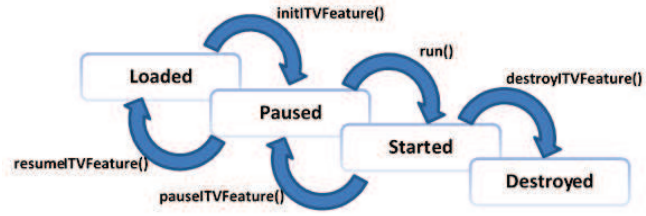

Figure 6. *ITVFeature* Life Cycle

Whereas this work uses a feature-oriented programming, the *ITVFeatures* are organized through a Feature Model. Thus, each *ITVFeature* may depend on an *ITVFeature* at a higher hierarchical position, and have *ITVFeatures* dependents. In such cases, when an *ITVFeature* enters the paused, booted, rebooted or destroyed states, their dependents must transition to the same state of the *ITVFeature* which are dependent.

3. CASE STUDY

The case study concerns a SPL implemented as shown in Figure 2, involving a T-commerce application. In SPL it can be observed the possibility of building two separate software products where, minimally, the application must provide (temporarily) the viewer to buy the products in accordance with your video presentation and, additionally, should present options add products to the shopping cart, as well as the option to access the shopping cart for checkout.

The shopping cart must be presented to the viewer after checking the existence of available resources. So, if the user can load the *iTVFeature* **HDTVCart** (if the TV supports HDTV resolution), the *iTVFeature* **OldProductList** must be performed (offering the products that were presented earlier for shopping). The checks on the possibility of execution should be implemented through **verifyBootableFeature()** method (Figure 4). The dependencies control is done automatically, in accordance with the settings of the SPL (Figure 3).

4. RELATED WORKS

Previous work relating SPL and Digital TV are targeted at solutions aimed at the Digital TV middleware. In [5] and [6] it was considered MHP (Multimedia Home Platform) middleware. [7] and [8] considered the use of SPL in the construction and refactoring of the middleware Ginga

From the viewpoint of tools for developing applications for interactive Digital TV, some other solutions can be mentioned, using software reuse techniques alternative to SPL. For example, TUIG [9], iTVProject [9] and FrameIDTV [10] are approaches based on development frameworks. TUGA [11], GingaGame [12] and Athus [13] are component-based approaches and APIs. Moreover, one can cite solutions for user accessibility, such as the Framework GUIDE [14].

Our work differs from others by offering a solution for construction of iDTV applications running over Digital TV middleware, using a SPL approach for software reuse and enabling feature selection at runtime.

5. CONCLUSIONS AND FUTURE WORKS

This paper has proposed a solution for development of iDTV applications. Concepts of SPL (particularly of DSPL) were used so as to offer a DSPL Platform (Framework) and enable the applications development to Digital Television domain.

Future work consider the extension of the solution for the treatment of characteristics of ubiquitous software projects, as well as the experimental evaluation of the maintainability of the platform.

6. REFERENCES

[1] J. Bosch, Maturity and evolution in software product lines: Approaches, artefacts and organization, in: Proceedings of the Second International Conference on Software Product Lines, SPLC 2, 2002, pp. 257–271.

[2] S. Deelstra, M. Sinnema, J. Bosch, Product derivation in software product families: a case study, J. Syst. Softw. 74 (2) (2005) 173–194.

[3] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid, Dynamic software product lines, Computer 41 (4) (2008) 93 –95.

[4] S. Apel, C. Kästner. An overview of feature-oriented software development, in Journal of Object Technology, Vol. 8, Nº 4, ETH Zurich, July-August (2009).

[5] M. Goedicke, C. Kõllmann, U. Zdun, Designing runtime variation points in product line architectures: Three cases, in Science of Computer Programming, Vol. 53, Issue 3, pag. 353-380, Elsevier, July (2004).

[6] M. Goedicke, K. Pohl, U. Zdun, Domain-specific runtime variability in product line architectures, in: Object-Oriented Information Systems, 2002 OOIS 8th International Conference on, France, Vol. 2425, pag. 384-396 Springer, (2002).

[7] E. A. Barbosa, T. V. Batista, A. F. Garcia, E. Silva, PL-aspectual acme: An aspect-oriented architectural description language for software product lines. in: ECSA, Springer, 2011.

[8] D. Saraiva, L. Pereira, T. V. Batista, F. C. Delicato, P. F. Pires, U. Kulesza, R. P. M. de Araujo, T. Freitas, S. M. Filho, A. L. S. Souto, Architecting a model-driven aspect-oriented product line for a digital TV middleware: A refactoring experience, in Software Architecture, Lecture Notes in Computer Science, ECSA 2010 4th European Conference on, Vol. 6285, pag .166-181,Springer, August (2010) .

[9] Oliveira, M. R. M., Filho, C. B. P., e Fer, A. F. R. ITV project: an authoring tool for MHP and Ginga-J based on a web environment. In Darnell, M. J., Masthoff, J., Panabaker, S., Sullivan, M., e Lugmayr, A., editors, UXTV, ACM International Conference Proceeding Series, pages 179–182, 2008.

[10] Pequeno, H. S. L., Gomes, G. A. M., Andrade, R. M. C., Souza, J. N.de , e Castro, M. F.de . FrameIDTV: A framework for developing interactive applications on digital television environments. J. Network and Computer Applications, 33(4):503–511, 2010.

[11] Rafz, D. F. TUGA game API. url: (http://software.dukitan.com/tuga/), accessed july 2014.

[12] Barboza, D. C. e Clua, E. W. G. GingaGame: A framework for game development for the interactive digital television. 2010 Brazilian Symposium on Games and Digital Entertainment, 0:162–167, 2009.

[13] Segundo, R. M. C. ATHUS: Um framework para o desenvolvimento de jogos para TV Digital utilizando Ginga. Master's degree dissertation on Informatics, Centro de Ciências Exatas e da Natureza, Universidade Federal da Paraíba, September, 2011.

[14] Jung, C. GUIDE - Adaptive user interfaces for accessible hybrid tv applications. url: (http://www.w3.org/2010/11 /web-andtv/ papers/webtv2 submission 55.pdf), accessed july 2014.

D Métricas para Avaliação de ALP [47]

Capítulo

6

Métricas para Avaliação de Arquitetura de LP

"*A verdadeira medida de um homem não é como ele se comporta em momentos de conforto e conveniência, mas como ele se mantém em tempos de controvérsia e desafio.*"

M. Luther King Jr. (1929 - 1968), Ativ. Político, Prêmio Nobel (1964)

O método *SystEM-PLA* fornece um conjunto de métricas baseadas em UML, de apoio à avaliação de ALP. Tais métricas são aplicadas a modelos de LP com o objetivo de coletar dados que auxiliam na realização de análises quantitativas e qualitativas, além de *trade-offs*. Para tanto, dois tipos de métricas são considerados pelo método *SystEM-PLA*, sendo elas:

- **Métricas Básicas:** visam medir elementos UML essenciais aos modelos de LP como, por exemplo, classes e componentes, com o objetivo de apoiar a definição de métricas compostas para atributos de qualidade. Este tipo de métrica é apresentado na seção 6.1; e
- **Métricas de Atributos de Qualidade:** são compostas com base nas métricas básicas e visam medir atributos de qualidade de uma ALP. A apresentação das métricas desse tipo é feita Seção 6.2.

A nomenclatura das métricas do *SystEM-PLA* é composta por cinco partes, separadas pelo caracter "_", sendo elas, respectivamente:

1. a **sigla referente à metaclasse** UML que está sendo medida;
2. a **sigla do modelo UML**, usado na medição da primeira parte;
3. a **sigla do tipo de métrica** que está sendo medida;
4. a **sigla do tipo de elemento de variabilidade** que está sendo medido; e
5. a **sigla do tipo de medida** que está sendo realizada.

Por exemplo, a métrica básica (BAS) $CLS\_CLS\_BAS\_VPT\_ISA$ indica se uma classe (CLS) é um (ISA) ponto de variação (VPT). A métrica $CLS\_ITF\_BAS\_INC\_NUM$ é básica (BAS) e mede o número (NUM) de interfaces (ITF), que são variantes inclusivas (INC), implementadas por uma classe (CLS).

A Tabela 6.1 apresenta as siglas usadas para formar o nome das métricas baseadas em UML do método *SystEM-PLA*. O conjunto de siglas não é definitivo, permitindo que novas siglas sejam adicionadas à medida que novas métricas vão sendo definidas. Note que a sigla MDL refere-se ao conjunto de todos os modelos UML de uma LP.

Tabela 6.1: Siglas Usadas para Formar o Nome das Métricas Baseadas em UML.

Sigla	Modelo UML	Tipo de Métrica	Elemento de Variabilidade	Tipo de Medida
ITF	Interface	—	—	—
CLS	Classe	—	—	—
CPT	Componente	—	—	—
DGM	Diagrama	—	—	—
MDL	Modelo	—	—	—
BAS	—	Básica	—	—
CLX	—	Complexidade	—	—
EXT	—	Extensibilidade	—	—
VBT	—	—	Variabilidade	—
VPT	—	—	Ponto de Variação	—
VTN	—	—	Variação	—
INC	—	—	Variante Inclusiva	—
EXC	—	—	Variante Exclusiva	—
OPT	—	—	Variante Opcional	—
MND	—	—	Variante Obrigatória	—
ISA	—	—	—	É um/uma
HAS	—	—	—	Possui um/uma
NUM	—	—	—	Número/Quantidade
TOT	—	—	—	Quantia Total

A Seção 6.1 apresenta as 52 métricas básicas fornecidas pelo *SystEM-PLA*, a Seção 6.2 enumera as métricas para os atributos de qualidade complexidade e extensibilidade, enquanto a Seção 6.3 mostra como as métricas são automatizadas usando a ferramenta *SDMetrics*. Um exemplo completo de aplicação e coleta das métricas de atributos de qualidade, com base nas configurações da LP AGM (Apêndice A) geradas como resultado do estudo piloto para a validação experimental de tais métricas, é apresentado na Seção 6.4. A validação experimental de tais métricas é apresentada no Capítulo 7.

6.1 Métricas Básicas

As métricas básicas visam medir modelos UML representados pelas seguintes metaclasses do metamodelo padrão da UML: *Interface*, *Class*, *Component*, *Dependency* e *Comment*. As métricas indicam quais modelos são pontos de variação e quais indicam variantes inclusivas, exclusivas, opcionais e obrigatórias. As métricas também medem o número de pontos de variação e variantes em cada diagrama dos modelos de LP e da ALP em geral Oliveira Junior *et al.* (2008).

As métricas básicas podem ser usadas para compor novas métricas para atributos de qualidade de uma ALP. Exemplos de composição de métricas para atributos de qualidade são apresentados na Seção 6.2 e em Oliveira Junior *et al.* (2008), que discorrem sobre as métricas e apresentam exemplos de coleta para uma LP de *Workflow Management Systems*.

Cada métrica está classificada em uma das seguintes categorias:

- ***variation point*:** indica métricas específicas à medição de pontos de variação;
- ***variant*:** indica métricas específicas à medição de variantes; e
- ***variability*:** indica métricas específicas à medição de variabilidades.

Os itens a seguir apresentam a descrição de cada uma das métricas básicas fornecidas pelo método *SystEM-PLA*, divididas em: métricas para classes e interfaces, diagramas, componentes e modelos de LP. O código XML de cada uma delas é apresentado no Apêndice B. Vale lembrar que o conjunto de métricas básicas aqui apresentado não é restrito e, portanto, permite que novas métricas sejam definidas e incorporadas às atuais.

6.1.1 Métricas Básicas de Classes e Interfaces

É definido um subconjunto de 32 métricas, sendo 16 para classes e 16 para interfaces. As 16 métricas básicas de classes são apresentadas nos itens a seguir:

1. ***CLS_CLS_BAS_VPT_ISA*:** indica se uma classe é um **Ponto de Variação**. Esta métrica possui valor 1 para classes marcadas com o estereótipo ≪**variationPoint**≫ e valor 0, caso contrário;

2. ***CLS_CLS_BAS_INC_ISA*:** indica se uma classe é uma **Variante Inclusiva**. Esta métrica possui valor 1 para classes marcadas com o estereótipo ≪**alternative_-OR**≫ e valor 0, caso contrário;

3. ***CLS_CLS_BAS_EXC_ISA*:** indica se uma classe é uma **Variante Exclusiva**. Esta métrica possui valor 1 para classes marcadas com o estereótipo ≪**alternative_-XOR**≫ e valor 0, caso contrário;

4. ***CLS_CLS_BAS_OPT_ISA*:** indica se uma classe é uma **Variante Opcional**. Esta métrica possui valor 1 para classes marcadas com o estereótipo ≪**optional**≫ e valor 0, caso contrário;

5. ***CLS_CLS_BAS_MND_ISA*:** indica se uma classe é uma **Variante Obrigatória**. Esta métrica possui valor 1 para classes marcadas com o estereótipo ≪**mandatory**≫ e valor 0, caso contrário;

6. ***CLS_CLS_BAS_INC_NUM*:** número de subclasses, que são **Variantes Inclusivas**, de uma classe que é **Ponto de Variação**. Esta métrica é representada pela Equação (7.1):

$$\left.\begin{array}{l} \boldsymbol{CLS\_CLS\_BAS\_INC\_NUM(C_{vp})} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_INC\_ISA(SC_i), \\ \text{tal que:} \\ n = \textit{número de subclasses (SC) da classe } C_{vp} \textit{ que é ponto de variação} \end{array}\right\} \quad (6.1)$$

7. ***CLS_CLS_BAS_EXC_NUM*:** número de subclasses, que são **Variantes Exclusivas**, de uma classe que é um **Ponto de Variação**. Esta métrica é representada pela Equação (7.2):

$$\left.\begin{array}{l} \boldsymbol{CLS\_CLS\_BAS\_EXC\_NUM(C_{vp})} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_EXC\_ISA(SC_i), \\ \text{tal que:} \\ n = \textit{número de subclasses (SC) da classe } C_{vp} \textit{ que é ponto de variação} \end{array}\right\} \quad (6.2)$$

8. ***CLS_CLS_BAS_OPT_NUM*:** número de classes, que são **Variantes Opcionais**, associadas a uma determinada classe. Esta métrica é representada pela Equação (7.3):

$$\left.\begin{array}{l} \boldsymbol{CLS\_CLS\_BAS\_OPT\_NUM(C)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_OPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à classe C} \end{array}\right\} \quad (6.3)$$

9. ***CLS_CLS_BAS_MND_NUM*:** número de classes, que são **Variantes Obrigatórias**, associadas a uma determinada classe. Esta métrica é representada pela Equação (7.4):

$$\left.\begin{array}{l} \boldsymbol{CLS\_CLS\_BAS\_MND\_NUM(C)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_MND\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à classe C} \end{array}\right\} \quad (6.4)$$

10. ***CLS_CLS_BAS_VPT_NUM*:** número de classes, que são **Pontos de Variação**, associadas a uma determinada classe. Esta métrica é representada pela Equação (7.5):

$$\left.\begin{array}{l} \boldsymbol{CLS\_CLS\_BAS\_VPT\_NUM(C)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_VPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à classe C} \end{array}\right\} \quad (6.5)$$

11. ***CLS_ITF_BAS_INC_NUM*:** número de interfaces, que são **Variantes Inclusivas**, associadas a uma determinada classe. Esta métrica é representada pela Equação (6.6):

$$\left.\begin{array}{l} \boldsymbol{CLS\_ITF\_BAS\_INC\_NUM(C)} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_INC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à classe } C \end{array}\right\} \quad (6.6)$$

12. ***CLS_ITF_BAS_EXC_NUM*:** número de interfaces, que são **Variantes Exclusivas**, associadas a uma determinada classe. Esta métrica é representada pela Equação (6.7):

$$\left.\begin{array}{l} \boldsymbol{CLS\_ITF\_BAS\_EXC\_NUM(C)} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_EXC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à classe } C \end{array}\right\} \quad (6.7)$$

13. ***CLS_ITF_BAS_OPT_NUM*:** número de interfaces, que são **Variantes Opcionais**, associadas a uma determinada classe. Esta métrica é representada pela Equação (6.8):

$$\left.\begin{array}{l} \boldsymbol{CLS\_ITF\_BAS\_OPT\_NUM(C)} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_OPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à classe } C \end{array}\right\} \quad (6.8)$$

14. ***CLS_ITF_BAS_MND_NUM*:** número de interfaces, que são **Variantes Obrigatórias**, associadas a uma determinada classe. Esta métrica é representada pela Equação (6.9):

$$\left.\begin{array}{l} \textbf{\textit{CLS\_ITF\_BAS\_MND\_NUM(C)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_MND\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à classe C} \end{array}\right\} \quad (6.9)$$

15. ***CLS_ITF_BAS_VPT_NUM*:** número de interfaces, que são **Pontos de Variação**, associadas a uma determinada classe. Esta métrica é representada pela Equação (6.10):

$$\left.\begin{array}{l} \textbf{\textit{CLS\_ITF\_BAS\_VPT\_NUM(C)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_VPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à classe C} \end{array}\right\} \quad (6.10)$$

16. ***CLS_CLS_BAS_VBT_NUM*:** número de variabilidades de uma determinada classe. Esta métrica é representada pela Equação (6.11):

$$\left.\begin{array}{l} \textbf{\textit{CLS\_CLS\_BAS\_VBT\_NUM(C)}} = \sum_{i=1}^{n} Cmt_i, \\ \text{tal que:} \\ n = \textit{número de comentários (Cmt), com o estereótipo} \ll \textit{variability} \gg \textit{, associados à classe C} \end{array}\right\} \quad (6.11)$$

A Tabela 6.2 apresenta as métricas definidas para classes e suas respectivas categorias.

Tabela 6.2: Métricas Básicas para Classes.

Ord.	Métrica	Categoria
01	CLS_CLS_BAS_VPT_ISA	*variation point*
02	CLS_CLS_BAS_INC_ISA	*variant*
03	CLS_CLS_BAS_EXC_ISA	*variant*
04	CLS_CLS_BAS_OPT_ISA	*variant*
05	CLS_CLS_BAS_MND_ISA	*variant*
06	CLS_CLS_BAS_INC_NUM	*variant*
07	CLS_CLS_BAS_EXC_NUM	*variant*
08	CLS_CLS_BAS_OPT_NUM	*variant*
09	CLS_CLS_BAS_MND_NUM	*variant*
10	CLS_CLS_BAS_VPT_NUM	*variation point*
11	CLS_ITF_BAS_INC_NUM	*variant*
12	CLS_ITF_BAS_EXC_NUM	*variant*
13	CLS_ITF_BAS_OPT_NUM	*variant*
14	CLS_ITF_BAS_MND_NUM	*variant*
15	CLS_ITF_BAS_VPT_NUM	*variation point*
16	CLS_CLS_BAS_VBT_NUM	*variability*

As 16 métricas básicas de interfaces são apresentadas nos itens a seguir:

17. ***ITF_ITF_BAS_VPT_ISA*:** indica se uma interface é um **Ponto de Variação**. Esta métrica possui valor 1 para interfaces que estão marcadas com o estereótipo ≪**variationPoint**≫ e valor 0, caso contrário;

18. ***ITF_ITF_BAS_INC_ISA*:** indica se uma interface é uma **Variante Inclusiva**. Esta métrica possui valor 1 para interfaces que estão marcadas com o estereótipo ≪**alternative_OR**≫ e valor 0, caso contrário;

19. ***ITF_ITF_BAS_EXC_ISA*:** indica se uma interface é uma **Variante Exclusiva**. Esta métrica possui valor 1 para interfaces que estão marcadas com o estereótipo ≪**alternative_XOR**≫ e valor 0, caso contrário;

20. ***ITF_ITF_BAS_OPT_ISA*:** indica se uma interface é uma **Variante Opcional**. Esta métrica possui valor 1 para interfaces que estão marcadas com o estereótipo ≪**optional**≫ e valor 0, caso contrário;

21. ***ITF_ITF_BAS_MND_ISA*:** indica se uma interface é uma **Variante Obrigatória**. Esta métrica possui valor 1 para interfaces que estão marcadas com o estereótipo ≪**mandatory**≫ e valor 0, caso contrário;

22. ***ITF_ITF_BAS_INC_NUM*:** número de interfaces, que são **Variantes Inclusivas**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.12):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_ITF\_BAS\_INC\_NUM(I)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_INC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à interface I} \end{array}\right\} \quad (6.12)$$

23. ***ITF_ITF_BAS_EXC_NUM*:** número de interfaces, que são **Variantes Exclusivas**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.13):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_ITF\_BAS\_EXC\_NUM(I)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_EXC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à interface I} \end{array}\right\} \quad (6.13)$$

24. ***ITF_ITF_BAS_OPT_NUM*:** número de interfaces, que são **Variantes Opcionais**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.14):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_ITF\_BAS\_OPT\_NUM(I)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_OPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à interface I} \end{array}\right\} \quad (6.14)$$

25. ***ITF_ITF_BAS_MND_NUM*:** número de interfaces, que são **Variantes Obrigatórias**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.15):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_ITF\_BAS\_MND\_NUM(I)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_MND\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à interface I} \end{array}\right\} \quad (6.15)$$

26. ***ITF_ITF_BAS_VPT_NUM*:** número de interfaces, que são **Pontos de Variação**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.16):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_ITF\_BAS\_VPT\_NUM(I)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_VPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) associadas à interface I} \end{array}\right\} \quad (6.16)$$

27. ***ITF_CLS_BAS_INC_NUM*:** número de classes, que são **Variantes Inclusivas**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.17):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_CLS\_BAS\_INC\_NUM(I)}} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_INC\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à interface I} \end{array}\right\} \quad (6.17)$$

28. ***ITF_CLS_BAS_EXC_NUM*:** número de classes, que são **Variantes Exclusivas**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.18):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_CLS\_BAS\_EXC\_NUM(I)}} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_EXC\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à interface I} \end{array}\right\} \quad (6.18)$$

29. ***ITF_CLS_BAS_OPT_NUM*:** número de classes, que são **Variantes Opcionais**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.19):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_CLS\_BAS\_OPT\_NUM(I)}} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_OPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à interface I} \end{array}\right\} \quad (6.19)$$

30. ***ITF_CLS_BAS_MND_NUM*:** número de classes, que são **Variantes Obrigatórias**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.20):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_CLS\_BAS\_MND\_NUM(I)}} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_MND\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à interface I} \end{array}\right\} \quad (6.20)$$

31. ***ITF_CLS_BAS_VPT_NUM*:** número de classes, que são **Pontos de Variação**, associadas a uma determinada interface. Esta métrica é representada pela Equação (6.21):

$$\left.\begin{array}{l} \textbf{\textit{ITF\_CLS\_BAS\_VPT\_NUM(I)}} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_VPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) associadas à interface I} \end{array}\right\} \quad (6.21)$$

32. ***ITF_ITF_BAS_VBT_NUM*:** número de variabilidades de uma determinada interface. Esta métrica é representada pela Equação (6.22):

$$\left.\begin{array}{l} \boldsymbol{ITF\_ITF\_BAS\_VBT\_NUM(I)} = \sum_{i=1}^{n} Cmt_i, \\ \text{tal que:} \\ n = \textit{número de comentários (Cmt), com o estereótipo} \ll \textit{variability} \gg, \textit{associados à interface I} \end{array}\right\} \quad (6.22)$$

A Tabela 6.3 apresenta as métricas definidas para interfaces e suas respectivas categorias.

Tabela 6.3: Métricas Básicas para Interfaces.

Ord.	Métrica	Categoria
17	ITF_ITF_BAS_VPT_ISA	*variation point*
18	ITF_ITF_BAS_INC_ISA	*variant*
19	ITF_ITF_BAS_EXC_ISA	*variant*
20	ITF_ITF_BAS_OPT_ISA	*variant*
21	ITF_ITF_BAS_MND_ISA	*variant*
22	ITF_ITF_BAS_INC_NUM	*variant*
23	ITF_ITF_BAS_EXC_NUM	*variant*
24	ITF_ITF_BAS_OPT_NUM	*variant*
25	ITF_ITF_BAS_MND_NUM	*variant*
26	ITF_ITF_BAS_VPT_NUM	*variation point*
27	ITF_CLS_BAS_INC_NUM	*variant*
28	ITF_CLS_BAS_EXC_NUM	*variant*
29	ITF_CLS_BAS_OPT_NUM	*variant*
30	ITF_CLS_BAS_MND_NUM	*variant*
31	ITF_CLS_BAS_VPT_NUM	*variation point*
32	ITF_ITF_BAS_VBT_NUM	*variability*

6.1.2 Métricas Básicas de Diagramas

As métricas de diagrama visam medir os totais relacionados aos elementos que pertencem a um determinado tipo de diagrama. Por exemplo, em um diagrama de classes, esse tipo de métrica mede a quantidade total de interfaces e classes que são pontos de variação ou variantes. Em um diagrama de componentes, mede quantos componentes são variáveis.

As 13 métricas básicas de diagramas são apresentadas nos itens a seguir:

33. ***DGM_CLS_BAS_VPT_TOT*:** número total de classes que são **Pontos de Variação** em um diagrama de classes. Esta métrica é representada pela Equação (6.23):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_VPT\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_VPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.23)$$

34. ***DGM_CLS_BAS_INC_TOT*:** número total de classes que são **Variantes Inclusivas** em um diagrama de classes. Esta métrica é representada pela Equação (6.24):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_INC\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_INC\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.24)$$

35. ***DGM_CLS_BAS_EXC_TOT*:** número total de classes que são **Variantes Exclusivas** em um diagrama de classes. Esta métrica é representada pela Equação (6.25):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_EXC\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_EXC\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.25)$$

36. ***DGM_CLS_BAS_OPT_TOT*:** número total de classes que são **Variantes Opcionais** em um diagrama de classes. Esta métrica é representada pela Equação (6.26):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_OPT\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_OPT\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.26)$$

37. ***DGM_CLS_BAS_MND_TOT*:** número total de classes que são **Variantes Obrigatórias** em um diagrama de classes. Esta métrica é representada pela Equação (6.27):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_MND\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_MND\_ISA(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.27)$$

38. ***DGM_CLS_BAS_VBT_TOT*:** número total de **Variabilidades** em classes em um diagrama de classes. Esta métrica é representada pela Equação (6.28):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CLS\_BAS\_VBT\_TOT(DC)} = \sum_{i=1}^{n} CLS\_CLS\_BAS\_VBT\_NUM(Cls_i), \\ \text{tal que:} \\ n = \textit{número de classes (Cls) do diagrama DC} \end{array}\right\} \quad (6.28)$$

39. ***DGM_ITF_BAS_VPT_TOT*:** número total de interfaces que são **Pontos de Variação** em um diagrama de classes. Esta métrica é representada pela Equação (6.29):

$$\left.\begin{array}{l} \boldsymbol{DGM\_ITF\_BAS\_VPT\_TOT(DC)} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_VPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.29)$$

40. ***DGM_ITF_BAS_INC_TOT*:** número total de interfaces que são **Variantes Inclusivas** em um diagrama de classes. Esta métrica é representada pela Equação (6.30):

$$\left.\begin{array}{l} \textbf{\textit{DGM\_ITF\_BAS\_INC\_TOT(DC)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_INC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.30)$$

41. ***DGM_ITF_BAS_EXC_TOT*:** número total de interfaces que são **Variantes Exclusivas** em um diagrama de classes. Esta métrica é representada pela Equação (6.31):

$$\left.\begin{array}{l} \textbf{\textit{DGM\_ITF\_BAS\_EXC\_TOT(DC)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_EXC\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.31)$$

42. ***DGM_ITF_BAS_OPT_TOT*:** número total de interfaces que são **Variantes Opcionais** em um diagrama de classes. Esta métrica é representada pela Equação (6.32):

$$\left.\begin{array}{l} \textbf{\textit{DGM\_ITF\_BAS\_OPT\_TOT(DC)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_OPT\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.32)$$

43. ***DGM_ITF_BAS_MND_TOT*:** número total de interfaces que são **Variantes Obrigatórias** em um diagrama de classes. Esta métrica é representada pela Equação (6.33):

$$\left.\begin{array}{l} \textbf{\textit{DGM\_ITF\_BAS\_MND\_TOT(DC)}} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_MND\_ISA(Itf_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.33)$$

44. ***DGM_ITF_BAS_VBT_TOT*:** número total de **Variabilidades** em interfaces em um diagrama de classes. Esta métrica é representada pela Equação (6.34):

$$\left.\begin{array}{l} \boldsymbol{DGM\_ITF\_BAS\_VBT\_TOT(DC)} = \sum_{i=1}^{n} ITF\_ITF\_BAS\_VBT\_NUM(Cmt_i), \\ \text{tal que:} \\ n = \textit{número de interfaces (Itf) do diagrama DC} \end{array}\right\} \quad (6.34)$$

45. ***DGM_CPT_BAS_VBT_TOT*:** número total de **Variabilidades** em componentes em um diagrama de componentes. Esta métrica é representada pela Equação (6.35):

$$\left.\begin{array}{l} \boldsymbol{DGM\_CPT\_BAS\_VBT\_TOT(DC)} = \sum_{i=1}^{n} CPT\_CPT\_BAS\_VBT\_NUM(Cpt_i), \\ \text{tal que:} \\ n = \textit{número de componentes (Cpt) do diagrama DC} \end{array}\right\} \quad (6.35)$$

A Tabela 6.4 apresenta as métricas definidas para diagramas e suas respectivas categorias.

Tabela 6.4: Métricas Básicas para Diagramas.

Ord.	Métrica	Categoria
33	DGM_CLS_BAS_VPT_TOT	*variation point*
34	DGM_CLS_BAS_INC_TOT	*variant*
35	DGM_CLS_BAS_EXC_TOT	*variant*
36	DGM_CLS_BAS_OPT_TOT	*variant*
37	DGM_CLS_BAS_MND_TOT	*variant*
38	DGM_CLS_BAS_VBT_TOT	*variability*
39	DGM_ITF_BAS_VPT_TOT	*variation point*
40	DGM_ITF_BAS_INC_TOT	*variant*
41	DGM_ITF_BAS_EXC_TOT	*variant*
42	DGM_ITF_BAS_OPT_TOT	*variant*
43	DGM_ITF_BAS_MND_TOT	*variant*
44	DGM_ITF_BAS_VBT_TOT	*variability*
45	DGM_CPT_BAS_VBT_TOT	*variability*

6.1.3 Métricas Básicas de Componentes e Modelos

As 7 métricas básicas de componentes e modelos são apresentadas nos itens a seguir:

46. ***CPT_CPT_BAS_VTN_HAS*:** indica se um componente possui **Variação**. Esta métrica possui valor 1 para componentes marcados com o estereótipo ≪**variable**≫, e valor, 0 caso contrário.

47. ***MDL_CLS_BAS_VPT_TOT*:** número total de **Pontos de Variação** em classes de uma ALP. Esta métrica é representada pela Equação (6.36):

$$\left.\begin{array}{l} \boldsymbol{MDL\_CLS\_BAS\_VPT\_TOT(LP)} = \sum_{i=1}^{n} DGM\_CLS\_BAS\_VPT\_NUM(DC_i), \\ \text{tal que:} \\ n = \textit{número de diagramas de classes (DG) da linha de produto LP} \end{array}\right\} \quad (6.36)$$

48. ***MDL_ITF_BAS_VPT_TOT*:** número total de **Pontos de Variação** em interfaces de uma ALP. Esta métrica é representada pela Equação (6.37):

$$\left.\begin{array}{l} \textbf{\textit{MDL\_ITF\_BAS\_VPT\_TOT(LP)}} = \sum_{i=1}^{n} DGM\_ITF\_BAS\_VPT\_NUM(DC_i), \\ \text{tal que:} \\ n = \textit{número de diagramas de classes (DG) da linha de produto LP} \end{array}\right\} \quad (6.37)$$

49. ***MDL_MDL_BAS_VPT_TOT*:** número total de **Pontos de Variação** em uma ALP. Esta métrica é resultado da soma da equação 6.36 com a equação 6.37, representado pela Equação (6.38):

$$\left.\begin{array}{l} \textbf{\textit{MDL\_MDL\_BAS\_VPT\_TOT(LP)}} = MDL\_CLS\_BAS\_VPT\_TOT(LP) + \\ MDL\_ITF\_BAS\_VPT\_TOT\ (LP) \end{array}\right\} \quad (6.38)$$

50. ***MDL_CLS_BAS_VBT_TOT*:** número total de **Variabilidades** em classes de uma ALP. Esta métrica é representada pela Equação (6.39):

$$\left.\begin{array}{l} \textbf{\textit{MDL\_CLS\_BAS\_VBT\_TOT(LP)}} = \sum_{i=1}^{n} DGM\_CLS\_BAS\_VBT\_NUM(DC_i), \\ \text{tal que:} \\ n = \textit{número de diagramas de classes (DG) da linha de produto LP} \end{array}\right\} \quad (6.39)$$

51. ***MDL_ITF_BAS_VBT_TOT*:** número total de **Variabilidades** em interfaces de uma ALP. Esta métrica é representada pela Equação (6.40):

$$\left.\begin{array}{l} \textbf{\textit{MDL\_ITF\_BAS\_VBT\_TOT(LP)}} = \sum_{i=1}^{n} DGM\_ITF\_BAS\_VBT\_NUM(DC_i), \\ \text{tal que:} \\ n = \textit{número de diagramas de classes (DG) da linha de produto LP} \end{array}\right\} \quad (6.40)$$

52. ***MDL_MDL_BAS_VBT_TOT*:** número total de **Variabilidades** em uma ALP. Esta métrica é resultado da soma das Equações 6.39 e 6.40, representada pela Equação (6.41):

$$\left.\begin{array}{l} \textbf{\textit{MDL\_MDL\_BAS\_VBT\_TOT(LP)}} = \textit{MDL\_CLS\_BAS\_VBT\_TOT(LP)} + \\ \textit{MDL\_ITF\_BAS\_VBT\_TOT (LP)} \end{array}\right\} \quad (6.41)$$

A Tabela 6.5 apresenta as métricas e suas categorias definidas para componentes e modelos.

Tabela 6.5: Métricas Básicas para Componentes e Modelos.

Ord.	**Métrica**	**Categoria**
46	CPT_CPT_BAS_VTN_HAS	*variant*
47	MDL_CLS_BAS_VPT_TOT	*variation point*
48	MDL_ITF_BAS_VPT_TOT	*variation point*
49	MDL_MDL_BAS_VPT_TOT	*variation point*
50	MDL_CLS_BAS_VBT_TOT	*variability*
51	MDL_ITF_BAS_VBT_TOT	*variability*
52	MDL_MDL_BAS_VBT_TOT	*variability*

6.2 Métricas para Atributos de Qualidade

Esta seção apresenta as métricas definidas para os atributos de qualidade complexidade e extensibilidade, utilizadas para ilustrar o método *SystEM-PLA*. Essas métricas consideram as variabilidades representadas em modelos UML de LP e representam um exemplo de como métricas para atributos de qualidade podem ser definidas, formalizadas, aplicadas, coletadas e validadas para que avaliações de ALP possam ser realizadas.

As subseções a seguir apresentam as métricas de complexidade e extensibilidade para ALP. A Seção 6.4 apresenta um exemplo de aplicação e coleta de tais métricas, tomando como base o estudo piloto realizado para a validação experimental (Capítulo 7) de tais métricas.

6.2.1 Métricas de Complexidade

A compreensão da **complexidade de uma LP** é fundamental do ponto de vista de adoção da abordagem de LP, uma vez que o gerente de LP pode fazer uma análise da complexidade potencial de uma LP e dos tipos de produtos que podem ser produzidos.

Organizações que possuem o núcleo de artefatos já desenvolvido para um determinado domínio podem analisar a complexidade de configurações distintas do núcleo para o desenvolvimento e evolução de seus possíveis produtos e, dessa forma, optar por uma ou mais configurações viáveis para o desenvolvimento de uma LP.

As métricas compostas a partir das métricas básicas para classes, visam medir a complexidade de uma ALP e são definidas com base na complexidade ciclomática (CC) de McCabe (1976). A CC mede a quantidade de lógica de decisão de um módulo de software representada pelo número de caminhos que devem ser testados. Outra métrica relacionada à complexidade ciclomática, porém específica da abordagem orientada a objetos, é o número de Métodos Ponderados por Classe (*Weighted Methods per Class* - WMC) (Chidamber e Kemerer, 1994), que indica a soma da CC de todos os métodos de uma classe.

As métricas compostas de complexidade para classes e componentes são apresentadas nos itens a seguir, assim como suas descrições e definições formais.

- ***CompInterface*:** tem sempre o valor 0.0, pois não possui métodos concretos para computar a métrica WMC de McCabe.
- ***CompClass*:** é o valor da métrica WMC para uma determinada classe. Esta métrica é representada pela Equação (6.42):

$$\left.\begin{array}{l} \textbf{CompClass(Cls)} = WMC(Cls) = \sum_{i=1}^{n} WMC(Mtd_i), \\ \text{tal que:} \\ \quad - \ n = \textit{número de métodos (Mtd) da classe Cls} \end{array}\right\} \quad (6.42)$$

- ***CompVarPointClass*:** é o valor da métrica *CompClass* (6.42), da classe que é um ponto de variação mais a soma do valor da métrica *CompClass* (6.42) de cada variante associada à classe. Esta métrica é representada pela Equação (6.43):

$$\left.\begin{array}{l} \textbf{CompVarPointClass(Cls)} = CompClass(Cls) + \sum_{i=1}^{n} CompClass(ClsAss_i), \\ \text{tal que:} \\ \quad - \ n = CLS\_CLS\_BAS\_INC\_NUM(Cls) + CLS\_CLS\_BAS\_EXC\_NUM(Cls) \\ \quad + \ CLS\_CLS\_BAS\_OPT\_NUM(Cls) + CLS\_CLS\_BAS\_MND\_NUM(Cls) \\ \quad + \ ITF\_ITF\_BAS\_INC\_NUM(Cls) + ITF\_ITF\_BAS\_EXC\_NUM(Cls) + \\ \quad ITF\_ITF\_BAS\_OPT\_NUM + ITF\_ITF\_BAS\_MND\_NUM(Cls) \end{array}\right\} \quad (6.43)$$

- ***CompVariabilityClass*:** é a soma da medida da métrica *CompVarPointClass* (6.43), de cada ponto de variação de uma determinada variabilidade. Esta métrica é representada pela Equação (6.44):

$$\left.\begin{array}{l} \textbf{CompVariabilityClass(Vbt)} = \sum_{i=1}^{nVP} CompVarPointClass(Cls_i), \\ \textit{tal que:} \\ \quad - \ nVP = CLS\_CLS\_BAS\_VPT\_NUM + CLS\_ITF\_BAS\_VPT\_NUM \end{array}\right\} \quad (6.44)$$

- ***CompVarComponent*:** é a soma da medida da métrica *CompVariabilityClass* (6.44), de cada classe que forma um componente. Esta métrica é representada pela Equação (6.45):

$$\left.\begin{array}{l} \textbf{CompVarComponent(Cpt)} = \sum_{i=1}^{nCls} CompVariabilityClass(Cls_i), \\ \textit{tal que:} \\ \quad - \ nCls = \textit{número de classes que formam o componente Cpt} \end{array}\right\} \quad (6.45)$$

- ***CompPLA*:** é a soma dos valores da métrica *CompVarComponent* (6.45) para cada componente de uma ALP. Esta métrica é representada pela Equação (6.46):

$$\left.\begin{array}{l} \textbf{CompPLA(ALP)} = \sum_{i=1}^{nCpt} CompVarComponent(Cpt_i), \\ \textit{tal que:} \\ \quad - \ nCpt = \textit{número de componentes da ALP} \\ \quad - \ Cpt_i \textit{ é o i-ésimo componente da ALP} \end{array}\right\} \quad (6.46)$$

6.2.2 Métricas de Extensibilidade

Um dos conceitos mais importantes e de maior impacto em uma aplicação que segue o paradigma de orientação a objetos, é a herança entre classes. A herança tem por característica permitir a extensão de uma aplicação em termos de suas funcionalidades com base nas interfaces e implementações de suas classes (Batory *et al.*, 2002; Freeman *et al.*, 2004; Nystrom *et al.*, 2004; Shalloway e Trott, 2002). Porém, um dos problemas da herança é a necessidade de a cada nova extensão, criar-se novos tipos de classes, acarretando em mudanças estruturais da aplicação. Para amenizar o impacto da herança na estrutura de uma aplicação durante a sua manutenção, utiliza-se o conceito de classes abstratas (Sane e Birchenough, 1999; Woolf, 1997). Tais classes possuem uma interface, formada por métodos abstratos, e uma implementação-padrão, formada por métodos concretos. Isso faz com que a aplicação não possua uma estrutura fixa voltada somente às classes concretas, mas define um conjunto de classes que representam pontos de extensão da aplicação, além dos *hot spots* de um arcabouço (Arango *et al.*, 1988).

Os conceitos apresentados até aqui, nesta seção, formam a base para se entender como extensibilidade é medida, no nível de classes e componentes.

As métricas compostas de extensibilidade para classes e componentes, são apresentadas nos itens a seguir, assim como suas descrições e definições formais.

- ***ExtensInterface*:** é o nível de extensibilidade de uma interface, o qual é sempre o valor 1.0 já que interfaces são compostas 100% por métodos abstratos. Esta métrica é representada pela Equação (6.47):

$$\left.\begin{array}{l} \textbf{ExtensInterface(Itf)} = \frac{ITF\_ITF\_EXT\_MTD\_NUM(Itf)}{ITF\_ITF\_EXT\_MTD\_NUM(Itf)} = 1.0, \\ \\ \text{tal que:} \\ \\ \quad - \textit{ITF\_ITF\_EXT\_MTD\_NUM(Itf) = número de métodos da interface Itf} \end{array}\right\} \quad (6.47)$$

- ***ExtensClass*:** é o nível de extensibilidade de uma classe. Fornece a porcentagem de métodos abstratos com relação ao total de métodos (abstratos mais os concretos) de uma classe. Esta métrica é representada pela Equação (6.48):

$$\left.\begin{array}{l} \textbf{ExtensClass(Cls)} = \frac{CLS\_CLS\_EXT\_ABM\_NUM(Cls)}{CLS\_CLS\_EXT\_MTD\_NUM(Cls)}, \\ \\ \text{tal que:} \\ \\ \quad - \textit{CLS\_CLS\_EXT\_ABM\_NUM(Cls) = número de métodos abstratos da classe Cls} \\ \quad - \textit{CLS\_CLS\_EXT\_MTD\_NUM(Cls) = número de métodos da classe Cls} \end{array}\right\} \quad (6.48)$$

- ***ExtensVarPointClass*:** é o valor da métrica *ExtensClass* (6.48), da classe que é um ponto de variação, mais a soma do valor da métrica *ExtensClass* (6.48) de cada variante associada à classe. Esta métrica é representada pela Equação (6.49):

$$\left.\begin{array}{l} \textbf{ExtensVarPointClass(Cls}_{VP}\textbf{)} = ExtensClass(Cls_{VP}) + \\ \sum_{i=1}^{n} ExtensClass(ClsAss_i), \\ \\ \text{tal que:} \\ \\ \quad - \; n = CLS\_CLS\_BAS\_INC\_NUM(Cls_{VP}) + \\ \quad CLS\_CLS\_BAS\_EXC\_NUM(Cls_{VP}) + CLS\_CLS\_BAS\_OPT\_NUM(Cls_{VP}) \\ \quad + CLS\_CLS\_BAS\_MND\_NUM(Cls_{VP}) + ITF\_ITF\_BAS\_INC\_NUM(Cls_{VP}) \\ \quad + ITF\_ITF\_BAS\_EXC\_NUM(Cls_{VP}) + ITF\_ITF\_BAS\_OPT\_NUM(_{VP}) + \\ \quad ITF\_ITF\_BAS\_MND\_NUM(Cls_{VP}) \end{array}\right\} \quad (6.49)$$

- ***ExtensVariabilityClass*:** é a soma da medida da métrica *ExtensVarPointClass* (6.49) de cada ponto de variação, de uma determinada variabilidade. Esta métrica é representada pela Equação (6.50):

$$\left.\begin{array}{l} \textbf{ExtensVariabilityClass(Vbt)} = \sum_{i=1}^{nVP} ExtensVarPointClass(Cls_i), \\ \textit{tal que:} \\ \quad - \ nVP = CLS\_CLS\_BAS\_VPT\_NUM + CLS\_ITF\_BAS\_VPT\_NUM \end{array}\right\} \quad (6.50)$$

- ***ExtensVarComponent*:** é a soma da medida da métrica *ExtensVariabilityClass* (6.50) de cada classe que forma um componente. Esta métrica é representada pela Equação (6.51):

$$\left.\begin{array}{l} \textbf{ExtensVarComponent(Cpt)} = \sum_{i=1}^{nCls} ExtensVariabilityClass(Cls_i), \\ \textit{tal que:} \\ \quad - \ nCls = \textit{número de classes que formam o componente Cpt} \end{array}\right\} \quad (6.51)$$

- ***ExtensPLA*:** é o nível geral de extensibilidade da ALP, sendo a soma dos valores da métrica *ExtensVarComponent* (6.51) de cada componente da ALP. Esta métrica é representada pela Equação (6.52):

$$\left.\begin{array}{l} \textbf{ExtensPLA(ALP)} = \sum_{i=1}^{nCpt} ExtensVarComponent(Cpt_i), \\ \text{tal que:} \\ \quad - \ nCpt = \textit{número de componentes da ALP} \\ \quad - \ Cpt_i \textit{ é o i-ésimo componente da ALP} \end{array}\right\} \quad (6.52)$$