UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

RAFAEL DE FARIA SCHEIDT

UM ESTUDO APLICADO DE LINHA DE PRODUTOS DE SOFTWARE EM UM AMBIENTE COMPUTACIONAL DISTRIBUÍDO

FLORIANÓPOLIS
2012

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

RAFAEL DE FARIA SCHEIDT

UM ESTUDO APLICADO DE LINHA DE PRODUTOS DE SOFTWARE EM UM AMBIENTE COMPUTACIONAL DISTRIBUÍDO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Mario Antônio Ribeiro Dantas

FLORIANÓPOLIS
2012

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

S318e Scheidt, Rafael de Faria
Um estudo aplicado de Linha de Produtos de Software em um ambiente computacional distribuído [dissertação] / Rafael de Faria Scheidt ; orientador, Mário Antônio Ribeiro Dantas. – Florianópolis, SC, 2012.
112 p.: il., grafs., tabs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da computação. 2. Engenharia de software. 3. Sistema distribuído. I. Dantas, Mário Antônio Ribeiro. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

RAFAEL DE FARIA SCHEIDT

UM ESTUDO APLICADO DE LINHA DE PRODUTOS DE SOFTWARE EM UM AMBIENTE COMPUTACIONAL DISTRIBUÍDO

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Curso de Pós-Graduação em Ciência da Computação, Programa de Pós-Graduação em Ciência da Computação, da Universidade Federal de Santa Catarina.

Florianópolis, 28 de fevereiro de 2012.

Orientador: Dr. Mario Antônio Ribeiro Dantas, UFSC	Dr.a Patrícia Vilain, UFSC
Dr. Roberto Willrich, UFSC	ph.D. Eduardo Santana de Almeida, UFBA

AGRADECIMENTOS

Gostaria de agradecer, primeiramente, ao meu orientador e professor, Mario Dantas, um excelente profissional motivado e motivador, por ter acreditado em mim e nas minhas ideias e, além disso, por ter me acolhido como orientador. Agradeço também ao Matheus Anversa Vieira, por ter me ajudado muito a entender o funcionamento do processo do LaPeSD e dos trabalhos realizados nesse laboratório. Sua ajuda e incentivo foram fundamentais para a conclusão deste trabalho.

Aos pesquisadores do LaPeSD, em especial ao Gabriel, ao Camilo, à Juliana Candia e ao Izaias, por compartilharem suas experiências e terem me ajudado no desenvolvimento e na aplicação do meu trabalho.

Não poderia deixar de agradecer aos meus amigos de mestrado Katreen e Urian, com os quais cursei disciplinas e fiz vários trabalhos.

Ao professor Edson A. Oliveira Junior, por ter criado a abordagem SMarty e por tirar minhas dúvidas sempre que possível.

Aos meus amigos da empresa Suntech, por entenderem a importância do mestrado em minha vida e me liberarem para os estudos.

À Nadine e à Maria Helena, minha esposa e minha mãe, respectivamente, agradeço de coração o entendimento da ausência e do esforço empreendido para concluir este trabalho. A vocês que são fontes de força na minha vida o meu muito obrigado.

Escolha um trabalho que você ame e não terás que trabalhar um único dia em sua vida.
Sigmund Freud

RESUMO

Projetos de software em geral tendem a buscar cada vez mais a reutilização e a componentização, visando à economia de tempo, custo e recursos de novos produtos. Sendo assim, a necessidade de técnicas e ferramentas para organizar projetos de maior qualidade em menor tempo é um dos grandes desafios da Engenharia de Software. Com isso, a Linha de Produtos de Software (LPS) se propõe a organizar e auxiliar sistematicamente o desenvolvimento de novos produtos em série em um mesmo domínio. Nesse contexto, o presente trabalho de pesquisa objetiva aplicar a abordagem de Linha de Produtos de Software em um ambiente computacional distribuído, visto que, em projetos envolvendo ambientes distribuídos, novas versões de um produto com evolução de suas características no mesmo domínio repetem e não reutilizam os principais artefatos, tais como arquitetura e componentes. A Linha de Produtos de Software pode evidenciar através de pontos de variação quais serão os locais de evolução bem como quais farão parte da arquitetura principal. Assim, o objetivo da abordagem levantada nesta dissertação é analisar um processo atual utilizado no Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) e propor uma nova abordagem utilizando Linha de Produtos de Software para desenvolver projetos. Dessa forma, busca-se uma nova abordagem para desenvolver o projeto reutilizando toda uma arquitetura, componentes e documentos já prontos, partindo de uma base sólida e criando novos produtos com foco nas novas funcionalidades. Como resultado dessa proposta, apresentam-se uma arquitetura e componentes reutilizáveis, além de maior organização e visibilidade, pois se entende que, com a aplicação dessa abordagem, se atinge com sucesso o desafio de aplicar o uso de Linha de Produtos de Software no Ambiente Computacional Distribuído.

Palavras-chave: Linha de Produtos de Software. Engenharia de Domínio. Ambientes Computacionais Distribuídos.

ABSTRACT

Software Projects, in a common sense, seek to use more the concepts of reutilization and componentization, in order to save time, costs, and resources for new products. Thus, the need for tools and techniques for organizing high quality projects in less time is one of the most challenging in Software Engineering. Said that, The Software Product Line proposes to organize and to help systematically the serial development of new products in a same domain. In that context, the present work proposes to apply the approach of The Software Product Line on Distributed System Environments, since, on projects involving Distributed Systems, new versions of a product that evolves its characteristics in the same domain, repeat and don't reuse the main artifacts, like architecture and components. The Software Product Line, can show through variation points, which ones will be evolution points, as well which ones will be part of the main architecture. Therefore, the objective of this work is to analyze the current process used in the Laboratory of Research Distributed Systems (LaPeSD) and propose a new approach using The Software Product Line to develop projects. Thus, we seek a new approach for developing the project, reusing an entire architecture, components and documents already existent, stating from a solid base and creating new products, focusing new features. As result of this proposal, it is presented an architecture and reusable components, as well a greater visualization, because it is understandable that, with the application of this approach, it reaches successfully the challenge of applying The Software Product Line on Distributed System Environments.

Keywords: Software Product Line. Domain Engineering. Distributed Computing Environment.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGM	Arcade Game Maker
API	Application Programming Interface
CMMI	Capability Maturity Model Integration
CSF	Community Scheduler Framework
DLL	Dynamic-Link Library
DOS	Disk Operating System
DRMAA	Distributed Resource Management Application API
EA	Engenharia de Aplicação
ED	Engenharia de Domínio
EJB	Enterprise JavaBeans
FAE	Family Architecture Evaluation
FODA	Feature-Oriented Domain Analysis
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HP	Hewlett-Packard
HTTP	HyperText Transfer Protocol
JME	Java Micro Edition
LAN	Local Area Network
LaPeSD	Laboratório de Pesquisa em Sistemas Distribuídos
LP	Linha de Produto
LPS	Linha de Produtos de Software
LSF	Load Sharing Facility
McMs	Meta Scheduler Multi-Cluster
MPS.BR	Melhoria de Processo do Software Brasileiro
OMG	Object Management Group
OSGI	Open Grid Services Infrastructure
PBS	Public Broadcasting Service
PC	Personal Computer
PHP	Personal Home Page
PLP	Product Line Practice
QADA	Quality-driven Architecture Design and Analysis
QoS	Quality of Service
RF	Requisito Funcional
RMS	Resource Management System
RNF	Requisito Não Funcional

SEI	Software Engineering Institute
SGE	Sun Grid Engine
SMarty	Stereotype-based Management of Variability
SPL	Software Product Line
UFSC	Universidade Federal de Santa Catarina
UML	Unified Modeling Language
WAN	Wide Area Network
WS	Web Service
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Este capítulo visa fornecer uma visão geral do trabalho, sendo composto de contextualização do problema, objetivo geral e objetivos específicos, bem como de delimitações, questões de pesquisa e principais contribuições.

1.1 MOTIVAÇÃO

Com o mercado de software aquecido no Brasil, com crescimento anual de 15%, tendo o mercado mundial crescido apenas 5,5% (MCT, 2009), empresas, academia e indústria já reconhecem a importância da reutilização de software, visando a uma maior qualidade no produto.

O número de projetos de software cancelado antes da entrega em 2005 foi de 15,52% e em 2007 chegou a 11,54% (EMAM; KORU, 2008), um volume relativamente grande de dinheiro que poderá não mais ser recuperado. Partindo da premissa de que há chances não remotas de projetos falharem, empresas estão cada vez mais interessadas nos processos que garantem ou tendem a aumentar o sucesso do projeto (KORNILOVICZ, 2006).

Cada vez mais, através da Engenharia de Software e da Programação Orientada a Objetos, procura-se a reutilização de elementos para o reaproveitamento em projetos do mesmo domínio (NAKAGAWA, 2007). Essa prática está diretamente relacionada ao melhoramento no custo de projetos, no tempo e na produtividade (SEI, 2012). Partindo desse princípio, muitas técnicas foram propostas, tais como engenharia de domínio (ED), frameworks, padrões, arquitetura de software e desenvolvimento baseado em componentes (GIMENES; TRAVASSOS, 2002).

Nesse contexto, a Linha de Produtos de Software surge para organizar sistematicamente e tornar previsível a reutilização de software baseada em um mesmo domínio de produtos. Segundo Aline Vasconcelos (2007), a Linha de Produtos de Software atua sobre produtos ou pesquisa similares e no mesmo domínio referente a uma área comum de negócio. O Software Engineer Institute (SEI, 2012) diz que Linha de Produtos representa sistemas de software que compartilham um conjunto de características comuns e controladas que satisfazem necessidades de um segmento de mercado em particular e

são desenvolvidas a partir de artefatos-chave (*core assets*), de forma predefinida (CLEMENTS; NORTHROP, 2001).

A Linha de Produtos de Software compreende as atividades de desenvolvimento do núcleo de artefatos, que, além dessa atividade, engloba também o desenvolvimento do produto e o gerenciamento da Linha de Produtos (CLEMENTS; NORTHROP, 2001; GIMENES; TRAVASSOS, 2002; OLIVEIRA, 2006). Segundo Clements e Northrop (2001) e Northrop (2002), a atividade de desenvolvimento do núcleo de artefatos também pode ser conhecida como Engenharia de Domínio e a atividade de desenvolvimento do produto como Engenharia de Aplicação.

Para formar uma linha de produtos é necessário representar as variações que podem ocorrer em cada artefato que faz parte desse domínio. Produtos em uma linha de produto existem simultaneamente e podem se diferenciar em termos de comportamento (conjunto de ações), atributos de qualidade, plataforma, configuração física, fatores de escala, entre muitos outros (CLEMENTS; NORTHROP, 2001).

O presente trabalho apresenta uma abordagem para Linha de Produtos de Software na área de Ambientes Computacionais Distribuídos, visando ao desenvolvimento sistemático de um conjunto de aplicações. Através dos componentes da Linha de Produtos de Software, pretende-se criar uma engenharia de domínio que servirá de base para o desenvolvimento de produtos, através da Engenharia de Aplicação na área de Grade Computacional e Computação Distribuída.

1.2 PROBLEMA

As organizações em geral constroem sistemas de software específicos em um domínio de aplicação, constantemente entregando variantes de produtos pela adição de novas características (SUGUMARAN; PARK; KANG, 2006). Nesse sentido, cada vez mais essas organizações buscam processos e formas de reaproveitar as características principais encontradas nos seus produtos.

Muitas técnicas que favorecem a reutilização têm sido propostas ao longo dos últimos anos, entre essas estão frameworks, padrões, arquitetura de software e desenvolvimento baseado em componentes. Porém, o que falta nesse contexto é uma maneira sistemática e previsível de realizar a reutilização em todas as fases do desenvolvimento (GIMENES; TRAVASSOS, 2002).

O problema principal deste trabalho foi encontrado em projetos do Laboratório de Pesquisa em Ambientes Distribuídos (LaPeSD) da Universidade Federal de Santa Catarina (UFSC), em que a cada nova versão de um mesmo produto são gerados novamente os mesmos artefatos em um produto que evolui suas características, porém em todas as versões existe uma arquitetura principal com variações de componentes.

Foram encontrados problemas frequentes de indecisão do cliente, que solicitava a implementação de um novo módulo e depois de um período pedia sua retirada. Após um tempo, o cliente identificava a necessidade de implantar novamente aquele módulo e, na maioria das vezes, a arquitetura não o suportava.

Com a utilização de uma linha de produtos de software, a adição ou a remoção de um elemento na arquitetura é facilmente realizada devido às suas características de reutilização, evitando a situação descrita anteriormente. Além disso, não existia a cultura da reutilização dos elementos, dos componentes nem mesmo de arquitetura.

1.3 OBJETIVOS

A partir da apresentação da motivação e do contexto do problema, são assim definidos na sequência os objetivos geral e específicos deste trabalho.

1.3.1 Objetivo geral

O objetivo geral do presente trabalho é o desenvolvimento de uma nova abordagem sistemática de Linha de Produtos de Software para um Ambiente Computacional Distribuído, bem como sua aplicação em um projeto real.

1.3.2 Objetivos específicos

Refinando o objetivo geral, podem-se apresentar como objetivos específicos os seguintes itens:

- entender e escolher uma abordagem para analisar e identificar variabilidades em Linha de Produtos de Software;
- desenvolver uma engenharia de domínio no que tange a requisitos da computação distribuída a fim de alimentar um repositório de Linha de Produtos de Software visando à engenharia de aplicações;
- criar uma arquitetura mais modular para ambientes distribuídos através da Linha de Produtos de Software;
- criar componentes para arquitetura principal desenvolvida e, com isso, fornecer maior visibilidade e flexibilidade ao ambiente; e
- avaliar através da Engenharia de Aplicação a abordagem em um projeto de estudo de caso e apresentar os resultados encontrados.

1.4 QUESTÕES DE PESQUISA

O presente trabalho pretende dar sua contribuição à comunidade científica na área de Linha de Produtos de Software, analisando e aplicando uma abordagem criada para Ambientes Computacionais Distribuídos, com vistas a uma melhor reutilização dos produtos instanciados a partir do domínio. Nesse contexto definiram-se as seguintes questões de pesquisa:

1. É possível criar uma linha de produtos de software para Ambientes Computacionais Distribuídos?
2. É possível utilizar a abordagem de Linha de Produtos de Software criada para suportar os trabalhos já feitos pelo LaPeSD e resolver os problemas de indecisão do cliente visando à produtividade, à reutilização e à visibilidade?

1.5 CONTRIBUIÇÃO DA PESQUISA

Visando aumentar o suporte da Engenharia de Software nos Ambientes Distribuídos, o presente trabalho deixa como contribuição os seguintes itens:

- entendimento de um ambiente de escalonamento distribuído real; e
- utilização da abordagem de Linha de Produtos de Software no cenário de Computação Distribuída.

1.6 ESTRUTURA DO TRABALHO

O desenvolvimento da abordagem para a criação de uma linha de produtos de software para um ambiente computacional distribuído compreende um estudo da literatura, a proposta do trabalho, o desenvolvimento, a avaliação e a aplicação através de estudos de caso. Assim, a dissertação é composta de seis capítulos.

No Capítulo 2, apresenta-se um estudo de Ambientes Computacionais Distribuídos visando entender cada uma das diferentes abordagens de cluster, multi-cluster, grid, middleware.

Na sequência e ainda na parte de revisão bibliográfica no Capítulo 3, aborda-se a questão da Linha de Produtos de Software, como é seu processo genérico de criação de Engenharia de Domínio e Engenharia de Aplicação. É apresentado também o Processo SMarty.

No Capítulo 4, apresenta-se o laboratório de pesquisa LaPeSD, no qual está sendo aplicada a abordagem proposta, bem como sua família de produtos. Adicionalmente, como era utilizada a arquitetura do Ambiente Computacional Distribuído e baseava-se nos problemas, é sugerida uma nova arquitetura para melhorar o reúso dos elementos do ambiente.

Um estudo de caso e resultados experimentais são apresentados no Capítulo 5, no qual se apresentam as telas do ambiente atual e seu funcionamento, os pontos fortes e fracos. Em seguida, exibem-se as telas do ambiente proposto, os artefatos e a Engenharia de Aplicação.

No último capítulo, apresentam-se as conclusões relativas a este trabalho, apontando direções através de trabalhos futuros relacionados à Linha de Produtos de Software aplicada a outros contextos e à continuação do trabalho.

2 AMBIENTES COMPUTACIONAIS DISTRIBUÍDOS

Na área de sistemas distribuídos o foco principal é a utilização de recursos distribuídos em configurações, tais como os clusters, os grids e as nuvens computacionais (DANTAS, 2005). Os Ambientes Computacionais Distribuídos podem ser interpretados através de duas visões. Na primeira sob o ponto de vista de arquitetura de computadores; já na segunda sob o aspecto orientado a software. As duas abordagens podem ser encontradas em Dantas (2005) e De Rose e Ferreto (2003). Nesta seção, abordam-se os paradigmas de conceitos necessários para o entendimento do ambiente computacional distribuído aplicado à Linha de Produtos de Software.

2.1 CLUSTER

Clusters são computadores interconectados também conhecidos como agregadores computacionais. Segundo Dantas (2005), um cluster pode ser definido como uma agregação de computadores de uma forma dedicada (ou não) para a execução de aplicações específicas de uma organização ou de uma instituição.

Já para Buyya (2011), cluster é definido como configurações de sistema distribuído paralelo formadas por um conjunto de computadores convencionais, independentes, mas interconectados por redes, muitas vezes de alto desempenho e trabalhando juntos, como um único recurso integrado para solução de problemas computacionais.

Um nó do cluster pode ser composto e compreendido por uma máquina ou um computador conectado na rede com um único processador, ou por um sistema multiprocessado. Dessa forma, é possível alcançar processamentos semelhantes aos supercomputadores, reaproveitando computadores de uma mesma rede, contudo por um custo mais baixo.

Segundo Colvero, Dantas e Cunha (2005), a classificação e as principais características de uma configuração de cluster ou ainda daquelas que se diferenciam de outras configurações, tal como grid, são:

- oferece alocação de recursos dentro de um único domínio que é centralizado;

- a segurança de processamento e recursos é dispensável, visto se tratar de um único domínio;
- permite milhares de nós na rede (quantidade limitada se for comparada às outras configurações detalhadas nas próximas seções);
- objetiva solução de problemas de granularidade grossa; e
- tem homogeneidade com relação ao sistema operacional.

Alguns dos principais objetivos desses sistemas são:

- realizar tarefas complexas em paralelo pela rede, oferecendo alto desempenho;
- diminuir custos por meio do uso de componentes de fácil disponibilidade e heterogêneos, ou seja, independentes de fornecedor;
- permitir diferentes configurações, adaptando a disponibilidade do sistema às requisições das aplicações e ao orçamento do cliente;
- oferecer escalabilidade, ou seja, facilidade em adicionar novos recursos ao sistema;
- obter alta disponibilidade de recursos para a execução das tarefas por meio de tolerância a falhas. A independência das máquinas escravas impede que falhas afetem o sistema; e
- oferecer transparência das complexidades do gerenciamento dessas máquinas para o usuário final, formando um sistema de imagem única (SSI). Ou seja, permitir o manuseio dos recursos de forma global e transparente, criando a ilusão de estar sendo usada uma única máquina, em vez da rede.

2.2 MULTI-CLUSTER

Esta função permite a distribuição das cargas, conhecidas em inglês como *workloads*, para a execução em nodos computacionais ociosos localizados em qualquer lugar da organização, e trabalha com prioridades de recursos e tarefas (JAVADI; AKBARI; ABAWAJY, 2006).

Diferentemente de cluster, que são conjuntos de computadores independentes interconectados por uma rede local LAN (Local Area Network), a estrutura de multi-clusters são sistemas computacionais

formados por conjuntos de clusters independentes conectados por uma rede WAN (Wide Area Network), permitindo o compartilhamento de recursos em diferentes domínios através da conexão de ambientes de cluster geograficamente dispersos.

Essa configuração forma um sistema integrado de imagem única, ou seja, o usuário tem uma visão global dos recursos e os utiliza como se estivessem em uma única máquina, independentemente de onde estão fisicamente associados (YEO et al., 2006).

2.3 GRADES COMPUTACIONAIS (GRID)

Grades computacionais podem ser definidas como uma variedade de recursos distribuídos geograficamente, paralelos e colaborativos, voltados ao compartilhamento, à seleção e à agregação de serviços e recursos autônomos em larga escala, para solucionar problemas em organizações virtuais dinâmicas multi-institucionais (FOSTER; KESSELMAN; TUECKE, 2001).

Por ser um ambiente heterogêneo de computadores e recursos, tanto no sentido de hardware quanto de software, não há centralização de recursos e gerenciamento, portanto se tem uma melhor escalabilidade no ambiente. O ambiente escalável demanda melhores políticas de gerenciamento a fim de evitar qualquer tipo de perda de performance ou degradação de desempenho no sistema (BOTE-LORENZO; DIMITRIADIS; GÓMEZ-SÁNCHEZ, 2004).

2.4 MIDDLEWARE

O middleware serve como um mediador em uma estrutura de Ambiente Distribuído. Existe a possibilidade de aplicar inteligência nos middlewares através de ontologias que podem ser previamente configuradas. Tais ontologias são desenvolvidas a fim de orientar as requisições processadas. Esse componente central da arquitetura de uma grade computacional tem como principal função ocultar a natureza heterogênea do sistema, assim como a complexidade inerente à distribuição dos recursos que compõem a grade.

Os protocolos utilizados nos gerenciadores de recursos são muito específicos e não seguem um padrão, o que não permite uma maior

heterogeneidade entre clusters. A maioria não integra recursos distribuídos sem um controle central, ou seja, os clusters devem estar sob controle de um gerenciador específico suportado pelo gerenciador (FOSTER, 2006).

Um metaescalonador provê interfaces para acesso virtual aos recursos e adota políticas globais para os provedores de recursos e consumidores. Dessa forma, são definidas regras importantes em um ambiente de grid.

A composição de um middleware (AIKEN et al., 2000) é formada por uma camada de software situada logicamente entre uma camada de nível mais alto (composta de usuários e aplicações) e outra camada subjacente que consiste em sistemas operacionais e facilidades básicas de comunicação. Os seguintes serviços geralmente são propostos pelo middleware: identificação, autenticação, autorização, diretórios, certificados digitais e outras ferramentas para segurança. A Figura 1 ilustra a lógica de um middleware perante todo o sistema de uma grade computacional (OLIVEIRA, 2010).

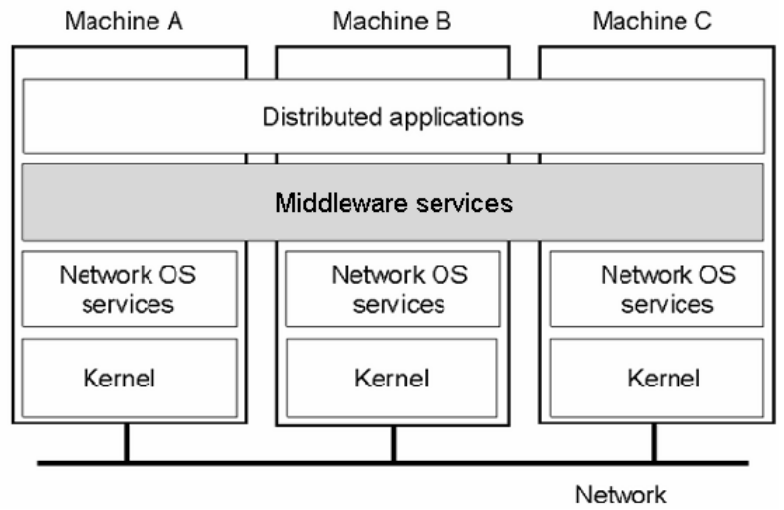


Figura 1: Localização do middleware na arquitetura de grade
Fonte: OLIVEIRA, 2010.

Nessa arquitetura o middleware é um componente-chave para a estrutura de grade computacional executar as tarefas de modo eficiente, uma vez que a transparência proporcionada por ele permite que o cliente

final se preocupe apenas em colher os resultados de sua atividade, abstraindo, assim, as questões secundárias.

2.5 COMMUNITY SCHEDULING FRAMEWORK (CSF)

A abordagem conhecida como Community Scheduling Framework (CSF, 2012) é um metaescalonador, de código aberto (*open-source*), capaz de realizar a submissão e o gerenciamento de tarefas, reservas antecipadas de recursos e o cumprimento de políticas requeridas pelas aplicações. A versão atual desse ambiente é conhecida como CSF4 (DING et al., 2008).

O CSF disponibiliza ao usuário final uma interface de acesso a recursos virtualizados e aplica políticas globais tanto para os provedores de recursos quanto para os consumidores desses recursos. Algumas características do CSF são:

- permite integração com diversos escalonadores locais (SGE, LSF, Condor, PBS);
- mantém independência dos escalonadores locais;
- possui visão global dos recursos; e
- permite controle de jobs em ambientes heterogêneos e escalonadores diferentes.

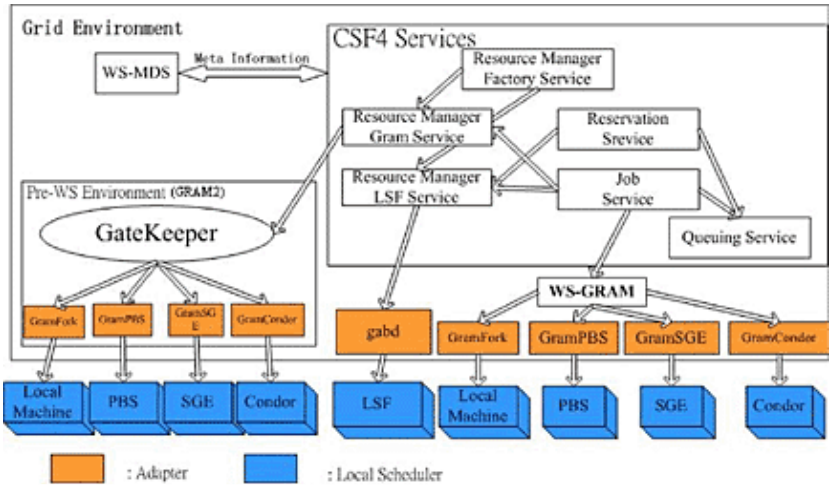


Figura 2: Visão do CSF4 e sua interação com gerenciadores de recursos
Fonte: CSF, 2012.

A Figura 2 ilustra uma visão geral de metaescalonamento CSF4 como ambiente. Dessa figura é possível entender que a interface de comunicação dos usuários em um ambiente distribuído, no caso uma configuração de grid, é realizada através do CSF4. Esse ambiente, utilizando-se dos componentes Resource Manager, Reservation Service, Job Service e Queuing Services, faz uma interação com o LSF, o PBS, o SGE e o Condor.

2.6 DESAFIO NO DESENVOLVIMENTO DE LINHA DE PRODUTOS DE SOFTWARE NO AMBIENTE COMPUTACIONAL DISTRIBUÍDO

Cada vez mais, organizações vêm utilizando Linha de Produtos de Software como solução para *time-to-market*, produtividade, flexibilidade e necessidade de customização massiva. A Hewlett-Packard, a Cummins Inc., a General Motors, a CelsiusTech, a Rockwell-Collins, a Motorola, a Philips, a Nokia, a Boeing, a Raytheon e a Salion Inc. têm uma abordagem de Linha de Produtos de Software que explora a semelhança entre os sistemas, a assertividade em custos, o melhoramento no cronograma e a diminuição do risco técnico (SEI, 2012).

Segundo SEI (2012), nas empresas citadas, a abordagem de Linha de Produtos de Software pode ser encontrada em várias finalidades, tais como

- a Nokia foi capaz de aumentar sua produção de telefones celulares de 5 a 10 modelos novos por ano para mais de 30 novos modelos por ano;
- a Hewlett-Packard reportou um aumento de produtividade de 400% de melhoria no tempo de colocação no mercado de uma linha de produtos de impressoras; e
- a Motorola sentiu uma melhoria no ciclo de tempo de 4x na sua linha de produtos de *pagers*.

Podem-se encontrar várias referências (MIETZNER, 2010; RUEHL; ANDERLFINGER, 2011) de aplicações em Linha de Produtos de Software, porém até o momento não se encontrou uma forma de aplicá-la em um Ambiente Computacional Distribuído. Encontra-se a

aplicação de abordagens especificamente em alguns elementos, tal como no middleware (OLIVEIRA; ROSA, 2009). Nesse sentido, torna-se um desafio a aplicação da abordagem de Linha de Produtos de Software em tal área.

A aplicação da abordagem se encontra em projetos do Laboratório de Pesquisa em Sistemas Distribuídos da Universidade Federal de Santa Catarina, em que a cada nova versão de um mesmo produto são gerados os mesmos artefatos repetidamente em um produto que evolui suas características, porém, no fundo, existe uma arquitetura principal com variações de componentes. Novos componentes podem ser incorporados na arquitetura principal e a cada novo trabalho existir uma retroalimentação nos seus componentes, servindo de referência para novos trabalhos e posteriormente sendo aplicada aos clientes.

No próximo capítulo serão explanadas abordagens de Linha de Produtos de Software bem como atividades essenciais e seus principais conceitos.

3 CONCEITOS DE LINHA DE PRODUTOS DE SOFTWARE

Para um melhor entendimento do desenvolvimento do trabalho, nesta seção são apresentados conceitos e abordagens de Linha de Produtos de Software, métodos de analisar e delimitar variabilidades, como também trabalhos corretados comparando as áreas de atuação de cada um com a presente pesquisa.

3.1 INTRODUÇÃO À LINHA DE PRODUTOS DE SOFTWARE

A abordagem de Linha de Produtos de Software vem se consolidando com o passar dos anos como uma forma efetiva de reutilização de artefatos de software tanto na indústria quanto na academia (ETXEBERRIA; SAGARDUI, 2008; LINDEN; SCHMID; ROMMES, 2007; OLIVEIRA JUNIOR, 2010a; SEI, 2012). Na Engenharia de Software, para desenvolver produtos com alta qualidade e que sejam economicamente viáveis, são necessários conjuntos sistemáticos de processos, técnicas e ferramentas em que o reúso de artefatos está entre as principais ideias (GIMENES; TRAVASSOS, 2002).

Nesse sentido, a Linha de Produtos de Software surge como uma proposta de construção sistemática de software baseada em uma família de produtos com vistas principalmente ao reúso (GIMENES; TRAVASSOS, 2002). Uma linha de produto representa um conjunto de sistemas que compartilham características comuns e gerenciáveis que satisfazem as necessidades de um segmento particular do mercado ou mesmo domínio (CLEMENTS; NORTHROP, 2001; NORTHROP, 2002).

O SEI (2012) define Linha de Produtos de Software como um conjunto de sistemas intensivos de software que, compartilhando em comum um conjunto de características, satisfaçam as necessidades específicas de determinado segmento de mercado ou missão e que sejam desenvolvidos a partir de um conjunto de bens essenciais de forma prescrita.

Segundo Oliveira Junior (2010a), a motivação para utilizar a abordagem de Linha de Produtos de Software é baseada nos seguintes itens (LINDEN; SCHMID; ROMMES, 2007; POHL; BOCKLE; LINDEN, 2005):

- redução de custos no desenvolvimento: através da atividade de geração de um núcleo de artefatos, pode-se alcançar a reutilização desses artefatos em diferentes produtos, consequentemente diminuindo o valor do produto final;
- melhoria na qualidade: a cada novo produto criado, os artefatos são testados e aperfeiçoados, em seguida podem retornar ao núcleo de artefatos para buscar cada vez mais qualidade;
- redução no tempo de produção (*time-to-market*): inicialmente o tempo de produção em uma linha de produto é mais alto devido ao desenvolvimento de todos os seus artefatos comuns. Porém, uma vez construídos esses artefatos, o tempo de desenvolvimento é reduzido às diferenças de cada novo produto;
- redução no esforço de manutenção: os reparos são feitos sempre no núcleo e propagados para os produtos;
- contribuição para a evolução: a organização baseada no núcleo oferece a oportunidade de os produtos estarem sempre evoluindo de acordo com o seu amadurecimento;
- contribuição para a redução da complexidade: a própria divisão de núcleo e produtos contribui para a redução da complexidade através de interfaces, componentização e reaproveitamento maior dos artefatos;
- melhoria da estimativa de custo: como os produtos possíveis são previsíveis, torna-se mais clara a criação de orçamentos. As diferenças ou os novos produtos podem ser vendidos por preços mais altos e ainda pode ser agregado o produto se for passível a sua reutilização por outros clientes; e
- benefícios específicos ao cliente: primeiramente o cliente escolhe o produto com suas necessidades selecionando as principais funcionalidades que irão atender às suas expectativas. O produto criado será de qualidade, uma vez que faz parte de um núcleo em comum em constante atualização.

A principal diferença entre Engenharia de Software convencional e Linha de Produtos de Software são a presença de variação em alguns ou até todos os requisitos de software e o enfoque. Nos sistemas convencionais foca-se em um objetivo único, ou seja, produzir um

produto específico. A mudança de enfoque é fundamentalmente de estratégia de negócios. Enquanto o desenvolvimento de sistemas convencionais funciona de maneira *ad hoc*, ou seja, orientado a contratos, o desenvolvimento de Linhas de Produto tem uma visão estratégica do nicho de mercado (LINDEN; SCHMID; ROMMES, 2007).

A implementação da abordagem de Linha de Produtos de Software exige esforços nos principais artefatos no ciclo de desenvolvimento. A Tabela 1 apresenta as principais características dos artefatos essenciais para a Linha de Produtos de Software.

Tabela 1: Custos de implementação de uma linha de produtos de software

Núcleo de Artefatos	**Custo de Implementação**
Arquitetura	Deve suportar variação herdada da Linha de Produto
Componentes de Software	Devem ser projetados para serem gerais, sem perda de desempenho; devem ter suporte para pontos de variação
Planos de teste, Casos de teste e Dados de teste	Devem considerar pontos de variação e múltiplas instâncias de Linha de Produto
Caso de Negócio e Análise de Mercado	Devem abordar uma família de produtos de software, e não apenas um produto
Planos de Projeto	Devem ser genéricos ou extensíveis para acomodar as variações do produto
Ferramentas e Processos	Devem ser mais robustos
Pessoas, Habilidades e Treinamentos	Devem envolver treinamento e experiência sobre os artefatos e os procedimentos associados à Linha de Produto

Fonte: SEI, 2012.

A seguir serão apresentadas as principais atividades para a aplicação da Linha de Produtos de Software.

3.1.1 Atividades essenciais à Linha de Produtos de Software

Segundo Clements e Northrop (2001) e Gimenes e Travassos (2002), as atividades essenciais da construção de uma linha de produto são Engenharia de Domínio, que tem como objetivo desenvolver o núcleo de artefatos, e Engenharia de Aplicação, na qual será instanciado o núcleo de artefatos para gerar diferentes produtos conhecidos também como Desenvolvimento do Produto (POHL; BOCKLE; LINDEN, 2005).

Segundo Pohl, Bockle e Linden (2005) e SEI (2012), o processo de desenvolvimento de uma linha de produtos de software pode ser visto como dois modelos de ciclo de vida, conforme mostra a Figura 3.

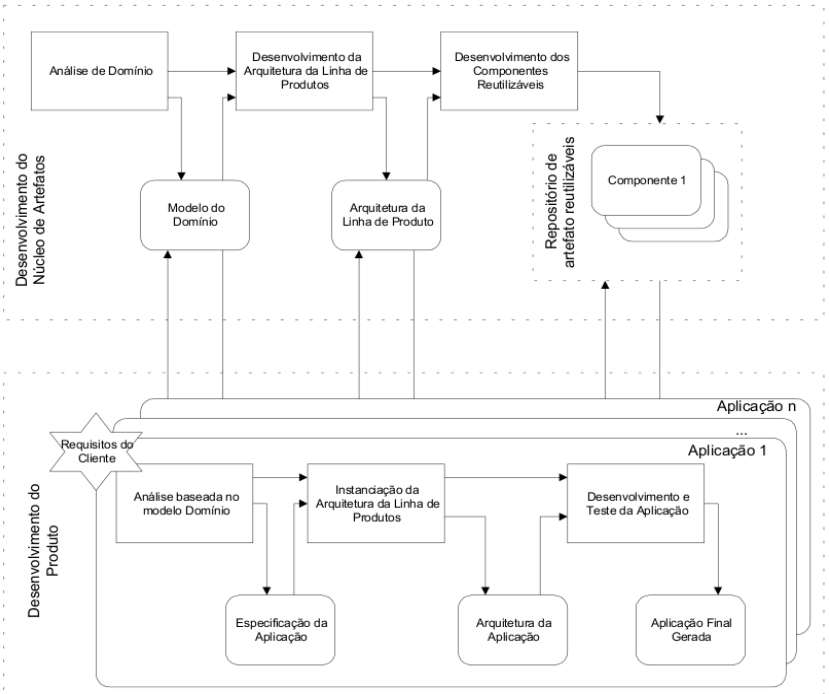


Figura 3: O processo de desenvolvimento de uma linha de produto
Fonte: POHL; BOCKLE; LINDEN, 2005; SEI, 2012.

O primeiro ciclo do Desenvolvimento do Núcleo de Artefatos (Engenharia de Domínio) tem como objetivo desenvolver o domínio

principal, que envolve essencialmente o desenvolvimento da arquitetura e dos componentes reutilizáveis (GIMENES; TRAVASSOS, 2002; POHL; BOCKLE; LINDEN, 2005; SEI, 2012).

O segundo ciclo de Desenvolvimento do Produto (Engenharia de Aplicação) busca basicamente gerar um novo produto com base nos artefatos construídos no primeiro ciclo que será conhecido como um membro da família de produtos.

O SEI (2012), por meio da iniciativa Product Line Practice (PLP), estabeleceu as atividades essenciais de uma abordagem de Linha de Produto (LP). Essas atividades são o Desenvolvimento do Núcleo de Artefatos, que corresponde à Engenharia de Domínio; o Desenvolvimento do Produto, que corresponde à Engenharia de Aplicação; e o Gerenciamento de LP. A Figura 4 ilustra as interações entre essas atividades.

Figura 4: Atividades essenciais de uma linha de produtos de software
Fonte: NORTHROP, 2002; SEI, 2012.

Os círculos da imagem representam a iteratividade das atividades bem como suas interligações. Os sentidos das flechas indicam que, além dos artefatos gerados no desenvolvimento do produto do núcleo, podem ser realizadas revisões dos artefatos gerados e, se for necessário, ser incluídos esses artefatos no núcleo (OLIVEIRA JUNIOR, 2010a).

Essas três atividades estão intrinsecamente relacionadas de tal forma que a alteração em uma delas implica analisar o impacto nas

demais (GIMENES; TRAVASSOS, 2002). As atividades essenciais de Desenvolvimento de Linha de Produtos de Software são explanadas em detalhes a seguir.

3.1.2 Desenvolvimento do Núcleo de Artefatos

O Desenvolvimento do Núcleo de Artefatos, ou originalmente do inglês *core assets,* é um conjunto de *assets* prontos para serem reusados no desenvolvimento de novos produtos. Os *core assets* podem ser componentes de software, padrões de projeto, documentos utilizados no desenvolvimento, na arquitetura, em cronogramas e em outros artefatos que servirão como blocos de construção na Linha de Produto. Os *core assets* encontram-se presentes em todos os produtos da linha. A arquitetura é um desses *core assets* e carrega consigo as possibilidades de variabilidade da linha. A arquitetura do *core asset*, em particular, é vista como ponto-chave de uma linha de produtos; caso seja mal projetado esse produto, pode inviabilizar todo o projeto (CLEMENTS; KAZMAN; KLEIN, 2002).

Também conhecida como Engenharia de Domínio (ED), objetiva que a reutilização seja atingida desde o início do ciclo de vida do software e, nesse sentido, os métodos de ED, em geral, cobrem as etapas de planejamento, análise, projeto e implementação do domínio. Alguns dos métodos de ED conhecidos na literatura são FODA (KANG et al., 1990), FORM (LEE; KANG; LEE, 2002), FODACom (VICI; ARGENTIERI, 1998), FeatuRSEB (GRISS; FAVARO; D'ALESSANDRO, 1998), ODM (SIMOS; ANTHONY, 1998), dyssey-DE (BRAGA, 2000) e CBD-Arch-DE (BLOIS, 2004).

Outra visão da Engenharia de Domínio, tratada por Almeida (2007), diz que é a atividade de coletar, organizar e armazenar experiências passadas na construção de sistemas ou parte de sistemas em um domínio particular na forma de artefatos reutilizáveis, bem como proporcionar um meio adequado para reutilização de ativos na construção de novos sistemas (ALMEIDA apud CZARNECKI; EISENECKER, 2000).

Ainda segundo Almeida (2007), um processo de Engenharia de Domínio deveria definir três passos importantes: Análise de Domínio (DA), Design de Domínio (DD) e Implementação de Domínio (DI). Em geral, o principal objetivo da Análise de Domínio é definir o escopo do domínio e um conjunto de reutilizáveis, requisitos configuráveis para os

sistemas no domínio. Em seguida, o Design de Domínio desenvolve uma arquitetura comum para o sistema no domínio e elabora o plano do produto. Finalmente, no terceiro passo são implementados os ativos reutilizáveis, por exemplo, componentes reutilizáveis, linguagens específicas de domínio, geradores e um processo de produção (ALMEIDA apud CZARNECKI; EISENECKER, 2000).

Conforme a Figura 5 apresentada pelo SEI (2012), as atividades de entrada para o Desenvolvimento do Núcleo de Artefatos são:

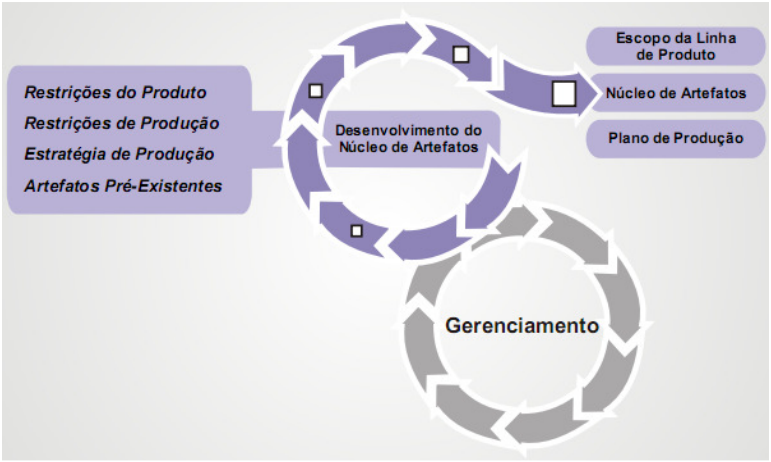


Figura 5: Desenvolvimento do Núcleo de Artefatos
Fonte: SEI, 2012.

- restrições do produto (estilos arquiteturais, padrões e arcabouços);
- restrições de produção;
- estratégia de produção; e
- repositório de artefatos preexistentes.

Como principais saídas tem-se as seguintes atividades:

- escopo da Linha de Produto;
- núcleo de arquitetos da Linha de Produto; e
- plano de produção dos produtos.

3.1.2.1 Desenvolvimento do Produto

O Desenvolvimento do Produto tem como principal objetivo a geração de produtos de uma linha de produto. Corresponde à instanciação de aplicações no ciclo dois, conforme a Figura 3, e, uma vez encontrados novos requisitos que não haviam sidos especificados, inicia-se uma retroalimentação entre essa atividade e a atividade de Desenvolvimento do Núcleo de Artefatos (Engenharia de Domínio), numa abordagem evolutiva (OLIVEIRA JUNIOR, 2010a; VASCONCELOS, 2007).

Tradicionalmente, a entrada para essa atividade é a saída da atividade anterior (Desenvolvimento do Núcleo de Artefatos). O ciclo de desenvolvimento de produto está representado pela Figura 6.

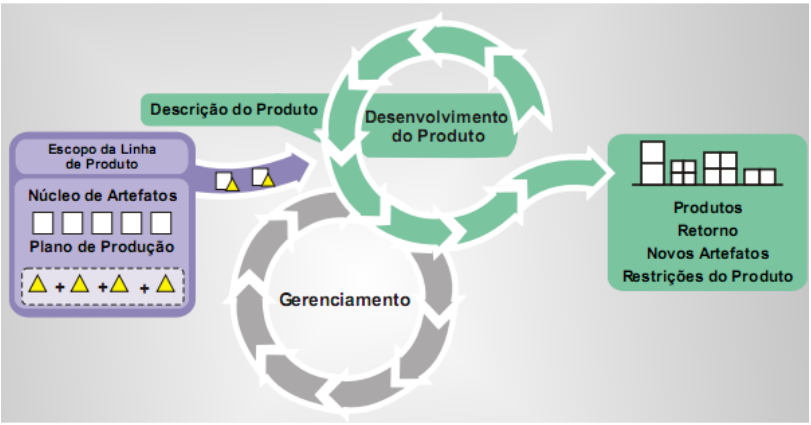


Figura 6: Desenvolvimento do Produto
Fonte: Adaptado de: OLIVEIRA JUNIOR, 2010a; SEI, 2012.

As direções das flechas indicam iteração e relacionamentos intrínsecos como, por exemplo, a existência e a disponibilidade de um produto específico podem afetar os requisitos de um novo produto que venha a ser derivado do núcleo de artefatos.

3.1.2.2 Gerenciamento da Linha de Produto

A atividade de Gerenciamento da Linha de Produto deve controlar para que todas as atividades técnicas sejam realizadas de acordo com um planejamento coordenado (OLIVEIRA JUNIOR, 2010a). Assim, é importante que a organização estabeleça um plano de adoção da Linha de Produtos que descreva o estado desejado e as estratégias para atingir esse estado (CLEMENTS; KAZMAN; KLEIN, 2002; CLEMENTS; NORTHROP, 2001).

É essencial que haja um sincronismo entre o grupo que desenvolve os artefatos e o grupo que gera os produtos. Adicionalmente, a organização deve garantir que as unidades organizacionais recebam os recursos corretos em quantidades suficientes.

3.2 VARIABILIDADE EM LINHA DE PRODUTOS DE SOFTWARE

Para formar uma família de software, ou seja, um conjunto de sistemas com características diferentes em um mesmo contexto, através da Linha de Produtos de Software, torna-se necessária a identificação das possíveis variações que podem ocorrer nos artefatos produzidos. Os produtos gerados na fase de desenvolvimento de produtos podem ser diferenciados em termos de comportamentos, atributos de qualidade, configurações físicas, fatores de escala, entre outros (CLEMENTS; NORTHROP, 2001; NORTHROP, 2002).

Variações são as diferenças tangíveis entre os produtos da Linha de Produtos em qualquer artefato, tais como arquitetura, componentes, interfaces entre componentes e conexões entre componentes. As variações podem ser identificadas em qualquer fase do desenvolvimento da Linha de Produtos (GIMENES; TRAVASSOS, 2002). Um exemplo de variações é a existência de duas abordagens distintas que podem ser aplicadas ao mesmo problema. Um exemplo simples de ser representado é a escolha de diferentes componentes para uma possível arquitetura. Um exemplo prático pode ser citado na linguagem JAVA (ORACLE, 2011): a aplicação do ArrayList e a LinkedList. Outro exemplo poderia ser uma variação de conexão sem fio, com dois pontos de variação: *bluetooth* e/ou GPRS.

Uma definição importante para variações em Linha de Produtos de Software é de que a escolha de todos os componentes possíveis em

uma arquitetura deve ser consistente e possuir um valor semântico que faça sentido no contexto do produto (GIMENES; TRAVASSOS, 2002).

Segundo Van Gurp e Bosch (2001), as variações podem ser classificadas por meio do conceito de *feature*. Esse conceito tem origem na Engenharia de Domínio (KANG et al., 1990). *Feature* é uma característica de um produto que pode ser definida como relevante e visível para o usuário final, que considera importante a descrição e a distinção de membros de uma família de produtos (KANG et al., 1990; SIMONS et al., 1996).

Segundo Kang et al. (1990) e Simons et al. (1996),

> Uma *feature* pode ser um requisito específico, uma seleção entre os requisitos opcionais e alternativos; ou pode estar relacionada a certas características do produto como funcionalidade, usabilidade e desempenho, ou pode estar relacionado às características de implementação como o tamanho, a plataforma de execução ou compatibilidade com certos padrões.

Os tipos de *features* e seus relacionamentos que podem ocorrer em uma família de produtos são organizados em modelos de *feature*. Atualmente, existem várias propostas de representações de *features* na literatura, porém todas tomam como base a representação em árvore (SOCHOS; PHILIPPOW; RIEBISCH, 2004) ou representações por grafos, em que os nós contêm as *features* e seus atributos (GIMENES; TRAVASSOS, 2002).

Em suma, *features* podem ser representadas como opcionais (podem ou não estar presentes em um produto), obrigatórias (estão sempre presentes) e composição de *features ou* variáveis (possuem um conjunto de *features* relacionadas).

A composição de pontos de variação e variantes forma a variabilidade, ou seja, um ponto de variação diz respeito a uma decisão de projeto que ainda não foi resolvida e a cada ponto de variação está associada a uma ou a várias variantes. Cada variante, posteriormente, poderá dar origem à instância de determinada variabilidade. Por exemplo, em uma linha de produtos para um projeto de Computação Distribuída, produtos poderiam ser gerados com dois tipos de *middleware*: com reserva antecipada ou sem reserva antecipada. O ponto de variação nesse caso seria o *middleware* e as variantes associadas seriam com reserva antecipada ou sem reserva antecipada.

3.2.1 Formas e representações de variabilidade

Para o desenvolvimento de uma linha de produtos, tornam-se necessárias a identificação e a representação de uma abordagem de variabilidade. Segundo Oliveira Junior (2010b), não existe até o momento um consenso sobre qual forma de identificar e representar a variabilidade em Linha de Produtos, por ser um assunto recente. Cada técnica possui vantagens e desvantagens e pode ser. também composta, porém um formato conhecido de representar e identificar pode ser através de Unified Modeling Language (UML) (OMG, 2012), por ser um padrão de mercado na Engenharia de Software (OLIVEIRA JUNIOR, 2010b).

Segundo Gimenes e Travassos (2002) e Griss, Favaro e D'Alessandro (1998), o Modelo de Features apresenta relações com o modelo de Casos de Uso da UML (OMG, 2012). Estas são as principais diferenças: o modelo de Casos de Uso é orientado para o usuário e o Modelo de Features é orientado para o reutilizador; o modelo de Casos de Uso descreve os requisitos do usuário em termos de funcionalidades do sistema e o Modelo de Features organiza o resultado da análise dos aspectos comuns e variáveis, preparando uma base para a reutilização; o modelo de Casos de Uso deve cobrir todos os requisitos de um sistema individual do domínio, enquanto o Modelo de Features deve incluir apenas aquelas características que o analista do domínio considera importantes.

Para a identificação e a representação de variabilidades em Linha de Produtos, o presente trabalho irá explorar os modelos de casos de uso, as classes e os componentes disponibilizados pelo SEI (2012). O SEI (2012) disponibiliza um exemplo simples mas compreensivo de Linha de Produtos caracterizado como Arcade Game Maker (AGM), criado para suportar aprendizados e experimentos.

A Figura 7 apresenta uma visão do Caso de Uso para Linha de Produtos da AGM.

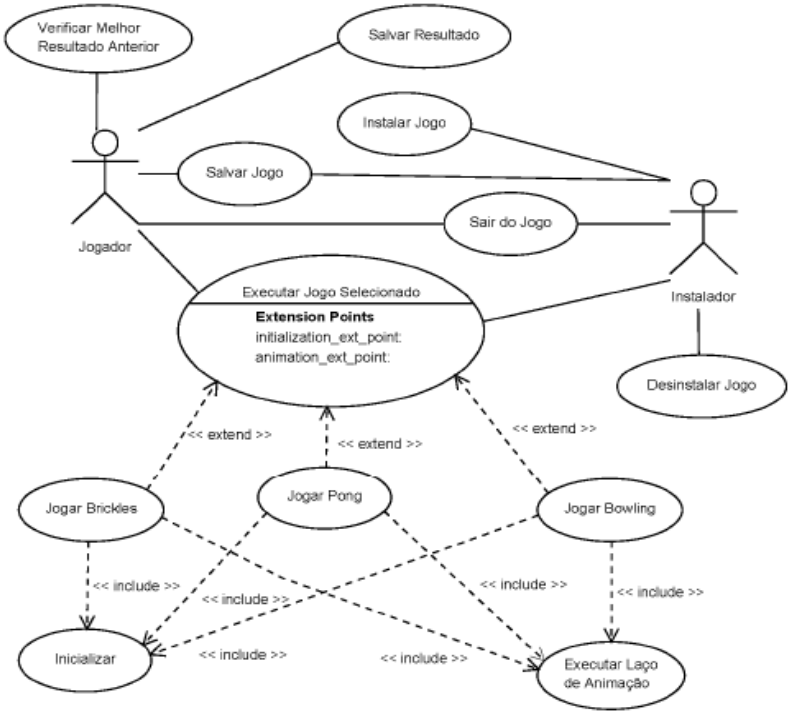


Figura 7: Diagrama de Caso de Uso da AGM

Os elementos de modelos de casos de uso relacionados aos mecanismos de extensão e de pontos de extensão sugerem pontos de variação com variantes associadas, as quais podem ser inclusivas, uma ou mais variantes podem ser selecionadas para fazer parte de um produto, ou exclusivas, somente uma variante pode ser selecionada. Assim, os casos de uso estendidos (<<extend>>) representam pontos de variação, enquanto os casos de uso que estendem representam variantes. Na Figura 7, por exemplo, os casos de uso Jogar Brickles, Jogar Pong e Jogar Bowling podem ser considerados variantes inclusivas associadas ao ponto de variação Executar Jogo Selecionado (caso de uso estendido).

Assim como os casos de uso estendidos têm sua representatividade, os casos de uso relacionados com associação de inclusão (<<include>>) possuem variantes obrigatórias. Exemplificando melhor através da Figura 7, o caso de uso Inicializar pode ser considerado uma variante obrigatória; por outro lado, numa associação

normal (sem seta), o caso de uso Verificar Melhor Resultado Anterior pode ser representado por uma variante opcional.

Outra forma de representação de variabilidade é apresentada na Figura 8, exibida conforme o modelo de classes, os pontos de variação e suas variantes, identificados nos seguintes elementos da UML (OMG, 2012):

- generalização: no exemplo da Figura 8, os classificadores mais gerais são os pontos de variação (exemplo MovebleSprite), enquanto os mais específicos são as variantes (Puck e Paddle);
- interface: no caso de a superclasse ser uma interface (realização), essa será ponto de variação e as subclasses serão as variantes;
- agregação: as instâncias tipadas com losangos não preenchidos são os pontos de variação e as instâncias associadas são as variantes; e
- composição: as instâncias tipadas com losangos preenchidos são os pontos de variação e as instâncias associadas são as variantes.

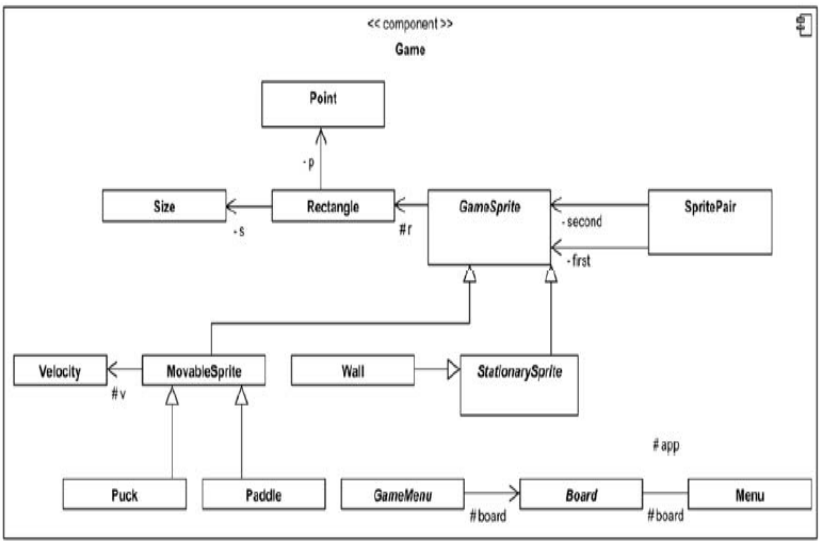


Figura 8: Modelo de Classes da AGM

As classes mais relevantes do modelo de classes da Figura 8 são o GameSprite, que faz o papel dos elementos de um jogo, o StationarySprite, que são os itens estáticos no jogo que não apresentam

movimento, e o MovableSprite, que representa justamente o contrário, ou seja, os elementos que possuem movimentos (SEI, 2012). Pode-se observar que, assim como no modelo de caso de uso, os elementos que envolvem extensão terão representações de pontos de variação e os que estendem serão variantes.

3.3 A ABORDAGEM SMARTY

Com base nas atividades e nos conceitos de variabilidade, Oliveira Junior (2010a) criou a proposta de uma abordagem Stereotype-based Management of Variability (SMarty) para gerenciar as variabilidades compostas de um perfil UML (OMG, 2012) chamado de SMartyProfile e um processo nomeado SMartyProcess. O principal objetivo da abordagem SMarty é permitir que as variabilidades de uma linha de produtos sejam gerenciadas de forma efetiva, suportando modelos UML (OMG, 2012).

O perfil SMartyProfile pertence à "abordagem para o gerenciamento de variabilidade", Systematic Management of Variability (SMarty) pode ser aplicado a qualquer ferramenta UML, sendo incorporado à modelagem padrão e exportado junto aos modelos UML (OMG, 2012) na forma de arquivos XMI (XML Model Interchange). Esse recurso permite que ferramentas automatizadas desenvolvidas por terceiros possam fazer uma extração no conteúdo do arquivo XMI e, por exemplo, coletar métricas a partir de modelos UML (OLIVEIRA JUNIOR, 2010b; OMG, 2012).

De acordo com Pohl, Bockle e Linden (2005) e Linden, Schmid e Rommes (2007), o gerenciamento de variabilidade é relacionado a todas as atividades de uma abordagem de Linha de Produtos e deve compreender as seguintes atividades: identificação de variabilidade (consiste em identificar as diferenças entre produtos e onde esses se localizam dentro dos artefatos da Linha de Produtos); delimitação de variabilidade (define o tempo de ligação e a multiplicidade das variabilidades); implementação de variabilidade (é a seleção dos mecanismos de implementação); e gerenciamento de variantes (controla as variantes e os pontos de variação). Essas atividades e os conceitos relacionados formam a base para a abordagem SMarty (OLIVEIRA JUNIOR; GIMENES; MALDONADO, 2010).

3.3.1 O perfil SmartyProfile

O SMartyProfile representa a relação dos conceitos mais importantes da Linha de Produtos e diz respeito ao gerenciamento de variabilidade. Existem quatro conceitos principais: variabilidade (BOSCH, 2004), pontos de variação (POHL; BOCKLE; LINDEN, 2005), variante (POHL; BOCKLE; LINDEN, 2005) e restrições de variante (BOSCH, 2004), os quais são descritos na sequência.

Variabilidade: segundo Weiss e Lai (1999), caracteriza a forma como os membros de uma família de produtos podem se diferenciar entre si. Apesar de ela poder ocorrer em diferentes graus de abstração de artefatos, a abordagem SMarty leva em consideração apenas artefatos da UML (OMG, 2012) que derivam de atividades de Desenvolvimento de Linha de Produtos (OLIVEIRA, 2006).

Pontos de variação: os pontos em que ocorrem as variabilidades possíveis, ou seja, os locais de variabilidade de uma linha de produto, segundo Oliveira (2006). Segundo Pohl, Bockle e Linden (2005), um ponto de variação deve responder à seguinte pergunta: O que varia em uma linha de produtos?

Variante: simboliza um artefato que foi escolhido para ser aplicado em um ponto de variação e, em alguns casos, pode fazer referência direta e resolver uma variabilidade associada. Com a seguinte pergunta pode-se identificar uma variante: Como uma variabilidade ou um ponto de variação varia em uma linha de produto? (POHL; BOCKLE; LINDEN, 2005).

Restrições de variante: definem os relacionamentos entre variantes visando equacionar um ponto de variação ou uma variabilidade. Por exemplo, pode-se decidir não oferecer certas combinações entre variantes mutuamente exclusivas para um conjunto de produtos, gerando uma restrição exclusiva entre tais variantes (OLIVEIRA JUNIOR, 2010a).

A Figura 9 representa a forma como são relacionados os conceitos de gerenciamento de variabilidade com os elementos de interesse do metamodelo UML (BOOCH; RUMBAUGH; JACOBSON, 1999; OMG, 2012). Na UML (OMG, 2012) existem classificações para os modelos, tais como Modelo Comportamental ou Modelo Estrutural.

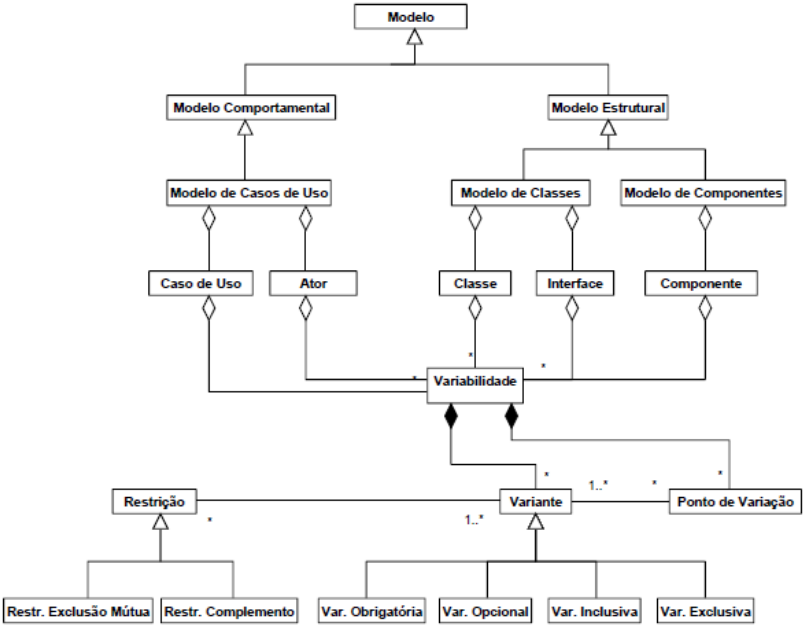


Figura 9: Relacionamento entre conceitos de variabilidades e UML

Como mostra a Figura 9 e seguindo os conceitos de relacionamento da UML (OMG, 2012), uma variabilidade pode ter zero ou várias variantes, as quais podem ser do tipo obrigatório, opcional, inclusivo ou exclusivo.

Segundo Oliveira Junior (2010b), variabilidade é composta de zero ou várias variantes, que podem ser inclusivas (representadas a partir de um conjunto de alternativas) ou exclusivas (representam a escolha de uma única alternativa a partir de um conjunto). E a variabilidade é composta de zero ou mais pontos de variação, tendo esta uma ou mais variantes associadas.

Ainda segundo o exemplo de Oliveira Junior (2010b), será exemplificada a forma de aplicação dos conceitos em uma linha de produtos que contém variabilidades relacionadas à ordenação de itens de coleção, denominada “Ordenação de elementos”. As variabilidades dessa característica podem ser realizadas por dois pontos de variação:

- o tipo de elemento a ser ordenado: variantes possíveis tipos String e Numérico; e

- o algoritmo de ordenação a ser aplicado: variantes possíveis *buble sort, quick sort, heap sort, selection sort, insertion sort* e *shell sort*.

Tais variantes podem ser selecionadas para resolver os pontos de variação visando gerar diferentes configurações de características para "Ordenação de elementos". As configurações selecionadas ou geradas são utilizadas para montar o possível produto da Linha de Produtos.

As figuras a seguir mostram o diagrama de caso de uso (Figura 10) e o diagrama de classes (Figura 11) da característica da "Ordenação de elementos".

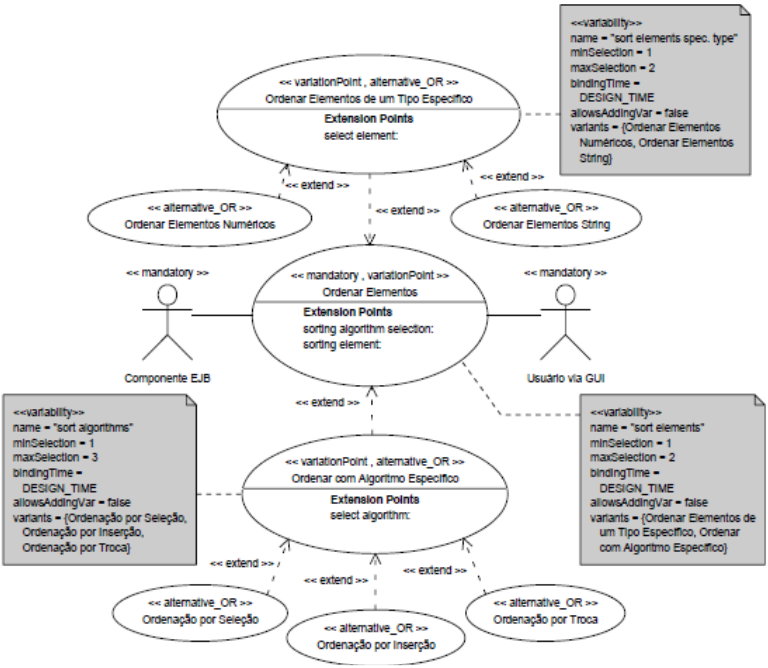


Figura 10: Modelo de caso de uso da característica "Ordenação de elementos"

Descrevendo a Figura 10, percebe-se que a ordenação de elementos é invocada pelo usuário via Graphical User Interface (GUI) ou por outro componente chamado Enterprise JavaBeans (EJB). Um desses atores dispara o caso de uso para que ordene elementos contendo duas extensões ou possibilidades: Ordenar Elementos de Um Tipo Específico

(define o tipo dos elementos a serem ordenados) e Ordenar com Algoritmo Específico (ordena o tipo de ordenação apropriado).

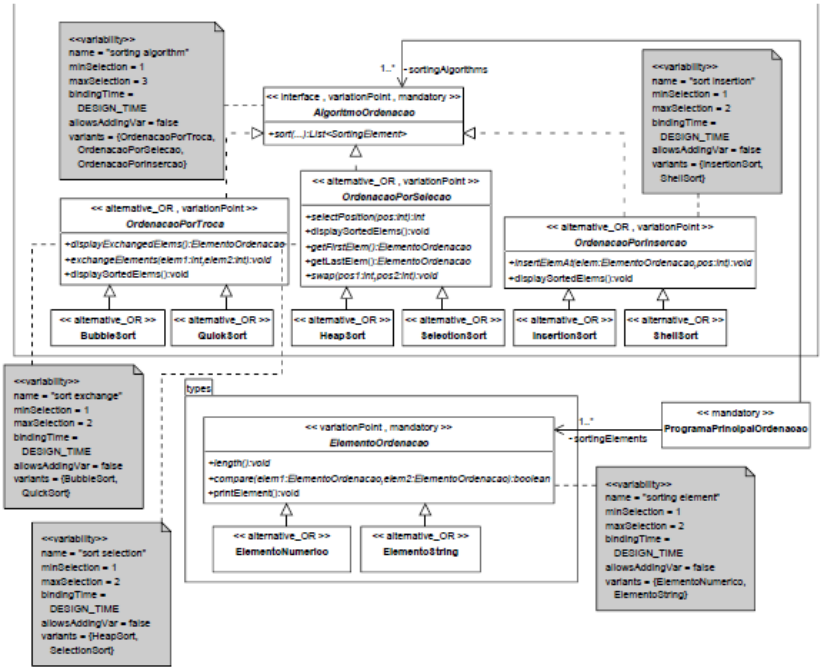


Figura 11: Modelo de classes da característica "Ordenação de elementos"

Na Figura 11 a superclasse ElementoOrdenacao possui duas subclasses – ElementoNumerico e ElementoString –, através das quais é possível ordenar por elementos numéricos ou *string*. Através da interface e do ponto de variação AlgoritmoOrdenacao, é possível realizar as seguintes classes: OrdenacaoPorTroca, OrdenacaoPorSelecao e OrdenacaoPorInsercao.

Com base nesses conceitos de gerenciamento de variabilidade, a Figura 12 representa o SMartyProfile, composto dos seguintes estereótipos e das respectivas metas-atributo (*tagged values*) (OLIVEIRA JUNIOR, 2010b):

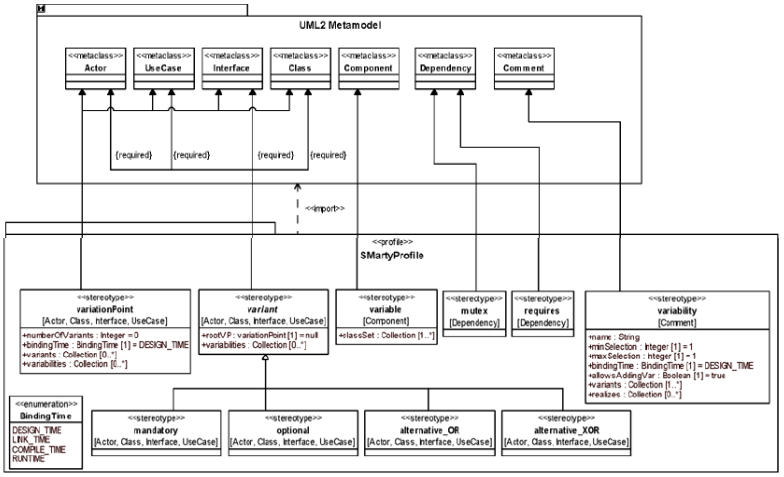


Figura 12: SMartyProfile: perfil UML para representar variabilidade em modelos de Linha de Produtos de Software
Fonte: OLIVEIRA JUNIOR, 2010b.

- **«variability»** representa o conceito Variabilidade e é uma extensão da metaclasse Comment. Isso significa que esse estereótipo pode ser aplicado somente em notas UML (OMG, 2012). Ele possui os seguintes meta-atributos:
    - **name** – o nome pelo qual uma variabilidade é referenciada;
    - **minSelection** – representa o número mínimo de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
    - **maxSelection** – representa o número máximo de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
    - **bindingTime** – é o momento no qual uma variabilidade deve ser resolvida. Esse tempo é representado pela classe de enumeração BindingTime;
    - **allowsAddingVar** – indica se é possível incluir novas variantes após uma variabilidade ser resolvida;
    - **variants** – representa a coleção de instâncias associada à variabilidade; e
    - **realizes** – representa a coleção de variabilidades de modelos de menor nível que realiza a variabilidade.

- **«variationPoint»** representa o conceito Ponto de Variação e é uma extensão das metaclasses Actor, UseCase, Interface e Class. Isso significa que este estereótipo pode ser aplicado somente a atores, casos de uso, interfaces e classes. Ele possui os seguintes meta-atributos:
    - **numberOfVariants** – indica o número de variantes associadas que podem ser selecionadas para resolver este ponto de variação;
    - **bindingTime** – o momento no qual um ponto de variação deve ser resolvido. Esse tempo é representado pela classe de enumeração BindingTime;
    - **variants** – representa a coleção de instâncias das variantes associadas a este ponto de variação; e
    - **variabilities** – representa a coleção de variabilidades com as quais este ponto de variação está associado.

- **«variant»** representa o conceito Variante e é uma extensão abstrata das metaclasses Actor, UseCase, Interface e Class. Por ser abstrato, este estereótipo não pode ser aplicado em nenhum elemento UML, porém suas especializações não abstratas podem ser aplicadas em atores, casos de uso, interfaces e classes. Este estereótipo é especializado em outros quatro estereótipos não abstratos, sendo eles: «mandatory», «optional», «alternative_OR» e «alternative_XOR». O estereótipo «variant» possui os seguintes meta-atributos:
    - **rootVP** – representa o ponto de variação ao qual está associado; e
    - **variabilities** – é a coleção de variabilidades com as quais essa variante está associada.

- **«mandatory»** representa uma variante obrigatória que está presente em todos os produtos de uma LP.
- **«optional»** representa uma variante que pode ser escolhida para resolver uma variabilidade ou um ponto de variação.
- **«alternative_OR»** representa uma variante que faz parte de um grupo de variantes inclusivas. Diferentes combinações das variantes inclusivas podem resolver pontos de variação de diferentes maneiras, gerando, assim, produtos distintos.
- **«alternative_XOR»** representa uma variante que faz parte de um grupo de variantes exclusivas. Isso significa que apenas

uma variante do grupo pode ser selecionada para resolver um ponto de variação.

- **«mutex»** representa o conceito de restrição Exclusão Mútua e é um relacionamento mutuamente exclusivo entre variantes. Isso significa que, para uma variante ser selecionada, a variante relacionada não pode ser selecionada.
- **«requires»** representa o conceito de restrição Complemento e é um relacionamento entre variantes no qual a variante escolhida requer a escolha da variante relacionada.
- **«variable»** é uma extensão da metaclasse Component. Este estereótipo indica que um componente é formado por um conjunto de classes com variabilidades explícitas. Este estereótipo possui o meta-atributo classSet, que é a coleção de instâncias das classes variáveis que formam o componente.

3.3.2 O processo SMartyProcess

Conforme apresentado na tese de Oliveira Junior (2010a), o processo SMartyProcess representa na sua essência que suas atividades estão diretamente relacionadas às atividades genéricas do processo de Desenvolvimento de Linha de Produtos (POHL; BOCKLE; LINDEN, 2005; SEI, 2012).

A seguir, na Figura 13, apresenta-se o processo SMartyProcess através do diagrama de atividade da UML, comparando o processo genérico de Desenvolvimento de Linha de Produtos, do lado esquerdo, com o processo SMartyProcess, que se encontra no retângulo à direita. Torna-se importante destacar que nem todos os autores tomam esse processo como o genérico padrão.

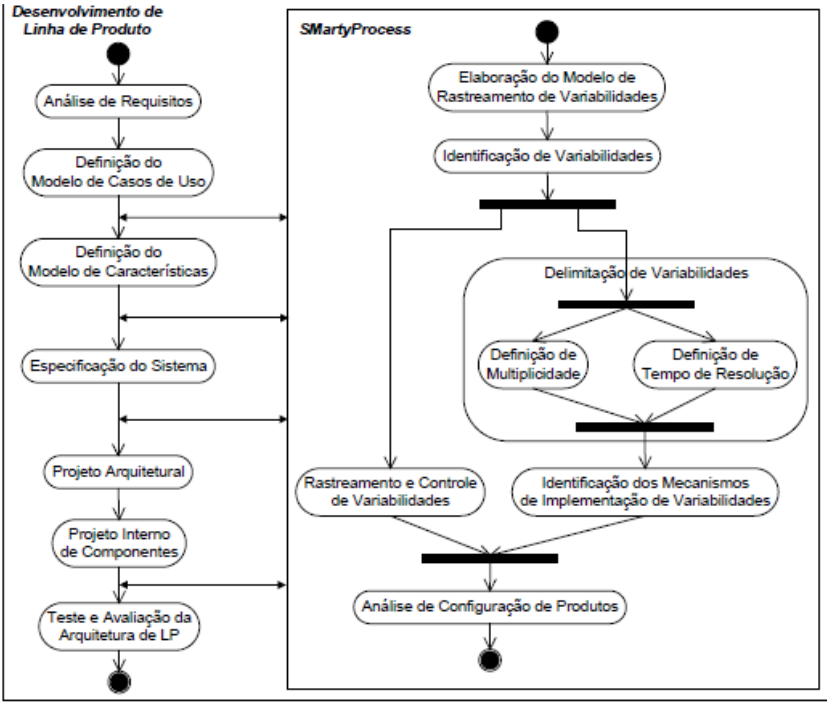


Figura 13: Apresentação de uma atividade de Desenvolvimento de Linha de Produtos vs. SMartyProcess
Fonte: OLIVEIRA JUNIOR, 2010a.

O processo SMartyProcess é realizado de forma iterativa, pois, após a execução de cada atividade do Desenvolvimento da Linha de Produtos, o SMartyProcess usa tais atividades como entrada para autoalimentar suas atividades e também se caracteriza por ser incremental, em que o número de variabilidades tende a crescer à medida que suas atividades são executadas. Todo o processo SMartyProcess é executado pelo engenheiro da Linha de Produtos, sendo desenvolvido em paralelo com o Desenvolvimento da Linha de Produtos (OLIVEIRA JUNIOR, 2010a).

A Figura 14 apresenta o processo da entrada e saída das atividades do Desenvolvimento da Linha de Produtos vs. SMartyProcess e como eles estão relacionados.

Ord.	*Desenvolvimento de Linha de Produto*				*SMartyProcess*		
	Atividade	→	Saída	→	Atividade	→	Saída
1	Definição do Modelo de Casos de Uso	gera como	Modelo de Casos de Uso	usado(a) como entrada para a(o)	Elaboração do Modelo de Rastreamento de Variabilidades	gera como	Modelo de Rastreamento
					Identificação de Variabilidades		Modelo de Variabilidade para Casos de Uso
					Delimitação de Variabilidades		
2	Definição do Modelo de Características		Modelo de Características		Elaboração do Modelo de Rastreamento de Variabilidades		Modelo de Rastreamento
					Rastreamento e Controle de Variabilidades		Instância do Metamodelo de Variabilidade
					Análise de Configurações de Produtos		-----
3	Especificação do Sistema		Modelo de Classes		Identificação de Variabilidades		Modelo de Variabilidade para Classes
					Delimitação de Variabilidades		
					Identificação dos Mecanismos de Implementação de Variabilidades		Modelo de Implementação de Variabilidades
4	Projeto Interno dos Componentes		Arquitetura Lógica de Componentes		Identificação de Variabilidades		Modelo de Variabilidade para Componentes
					Delimitação de Variabilidades		
					Identificação dos Mecanismos de Implementação de Variabilidades		Modelo de Implementação de Variabilidades

Figura 14: Relação de entrada e saída das atividades do Desenvolvimento de Linha de Produtos vs. SMartyProcess

Com base na Figura 14, na linha 1 a atividade de Definição do Modelo de Casos de Uso do Desenvolvimento de Linha de Produto tem a saída Modelo de Casos de Uso, que é usado como entrada para a atividade de Identificação de Variabilidades, que, por conseguinte, tem como saída o Modelo de Variabilidade para Casos de Uso. As demais linhas seguem o mesmo fluxo de entrada e saída de atividades.

3.3.3 Comentários da abordagem SMarty

Cada vez mais a abordagem de Linha de Produtos de Software vem se consolidando como uma forma efetiva de reutilização de artefatos de software tanto na indústria quanto na academia (ETXEBERRIA; SAGARDUI, 2008; LINDEN; SCHMID; ROMMES, 2007; SEI, 2012).

Neste capítulo foram apresentados conceitos de Linha de Produtos de Software, tais como ponto de variação, variante e restrições entre variantes, juntamente com suas características. Em suma, foram explorados o perfil SMartyProfile e o processo SMartyProcess, que, dentre outras abordagens que existem no mercado, visa estender as

principais metaclasses UML para representar variabilidades em casos de uso, classes e componentes.

3.4 TRABALHOS RELACIONADOS

Nesta seção serão apresentados trabalhos encontrados na comunidade científica que estão de alguma forma relacionados com o tema desta dissertação no que diz respeito à Linha de Produtos de Software.

3.4.1 UBÁ

A UBÁ (OLIVEIRA; ROSA, 2009) se propõe a construir uma arquitetura de Linha de Produtos de Software para auxiliar o desenvolvimento de sistemas de middleware para Computação em Grade. A utilização da abordagem leva a computação em grade a minimizar a complexidade latente, tratando de algumas características como coordenação de recursos em grande escala, provimento do QoS, grande número de dispositivos heterogêneos, além da questão geograficamente distribuída.

O processo de desenvolvimento de middleware com UBÁ segue as fases de definição da Engenharia de Domínio, por meio da utilização da metodologia Quality-driven Architecture Design and Analysis (QADA), que já contém serviços responsáveis pela execução de tarefas tradicionais, tais como gerência de recursos, serviço de segurança e descoberta de recursos. Após essa fase, é implementada a arquitetura usando a linguagem JAVA (ORACLE, 2012) e, posteriormente, são construídos os testes.

A fase de aplicação do trabalho foi realizada através da instanciação da Linha de Produtos de Software proposta, gerando, assim, um middleware para computação em Grade contendo as principais funcionalidades. Além desse, foi construída uma estrutura que fez uso da API do middleware gerado e foram feitos testes em diferentes cenários. A aplicação submetida teve o objetivo de calcular e verificar números primos.

A avaliação da abordagem deu-se através da aplicação Family Architecture Evaluation (FAE), um método de avaliação de arquitetura

de Linha de Produtos de Software em que as respostas relativas aos artefatos incluindo arquitetura são mapeadas através da observação e pontuadas em níveis de maturidade, particularmente parecendo uma avaliação subjetiva.

3.4.2 Reúso sistematizado de software e Linhas de Produto de Software no setor financeiro

A tese de doutorado de Reinehr (2008) trata do reúso de sistemas através de Linha de Produtos de Software na área de sistemas financeiros. Tal pesquisa apresenta o mapeamento do cenário de reúso de software nesse setor no Brasil, com estudos de caso nos cinco maiores bancos nacionais. A questão principal do trabalho é como o reúso de software é praticado nas organizações do setor financeiro e como, por sua vez, essas práticas contribuem para o sucesso dos projetos de software.

Um exemplo clássico da falta de reutilização é uma situação em que um dos bancos precisou alterar uma regra na conta-corrente e foi necessário promover alterações em 72 sistemas. Esses problemas foram consequentemente causados por falta de planejamento de reúso nos softwares em questão.

Da pesquisa realizada nas entidades financeiras, foram levantados vários pontos de análise ligados ao reúso de software, tais como iniciativa organizacional de reúso; processo orientado a reúso; habilidades, competências e motivação para reúso; medição para reúso; iniciativas setorizadas de reúso; reúso não sujeito a planejamento e gerenciamento; Engenharia de Domínio e Engenharia de Aplicação; Gerenciamento da Variabilidade; fatores favoráveis à Linha de Produtos de Software (LPS) relacionados à organização; fatores favoráveis à SPL relacionados ao pessoal; fatores favoráveis à SPL relacionados ao processo; fatores favoráveis à SPL relacionados ao produto; tipos de artefatos reutilizados; e finalização com sucesso dos projetos com reúso qualitativa e quantitativamente.

Especificamente, a avaliação das práticas de Linha de Produtos de Software (REINEHR, 2008) buscou informações nas organizações sobre a existência da utilização de tais práticas mesmo sem utilizar fortemente essa denominação. A resposta do setor financeiro a respeito da existência do conceito de Engenharia de Domínio e de Engenharia de Aplicação foi satisfatória, compreendendo-se que três das cinco

empresas possuem como artefato de reutilização a arquitetura das aplicações, um dos elementos principais da Linha de Produtos de Software. Por outro lado, deixaram a desejar no quesito Gerenciamento de Variabilidade, visto que apenas três das organizações tinham uma noção de utilização da abordagem.

A conclusão de Reinehr (2008) a respeito das práticas de Linha de Produtos de Software no setor financeiro é de que eles já entendem a importância de tal disciplina e é verdadeira a aplicabilidade no setor. A Engenharia de Domínio principalmente já pode ser identificada nos bancos que estão amadurecendo mais suas iniciativas de reúso.

3.4.3 UbiComSPL

A dissertação de Oliveira (2009) realizou o trabalho chamado de UbiComSPL. Trata-se de uma abordagem para desenvolvimento baseada em MDA, de Linha de Produtos de Software orientada a domínios de aplicações ubíquas. Com foco no domínio do problema, desenvolve-se Core Asset, núcleo da Linha de Produtos de Software que é utilizado na Engenharia de Aplicação da linha de produtos.

A abordagem é construída em duas fases: Engenharia de Domínio e Engenharia de Aplicação. Em cada fase tem-se as disciplinas de requisitos, projetos e implementação.

3.4.4 Linha de Produtos de Software no processo de geração de sistemas web de apoio à gestão de fomento de projetos

A dissertação de mestrado de Carromeu (2007) teve como objetivo a automatização de um processo usando Linha de Produtos de Software orientado à família de produtos do domínio de sistemas web de apoio à gestão de fomento de projetos utilizados por agências estaduais, tal como a Federação de Amparo à Pesquisa (FAP).

O processo de Linha de Produtos de Software usado nesse trabalho foi a abordagem PLUS. Adicionalmente, foi integrado com uma ferramenta denominada Fênix e com o framework Titan para automatizar o processo de geração da Engenharia de Aplicação.

3.4.5 Discussão

A abordagem UBÁ trabalha mais focada no desenvolvimento de uma engenharia de domínio para middleware em grade computacional em que esse não abrange o contexto da Engenharia de Domínio do Ambiente Distribuído.

Já a abordagem UbiComSPL se preocupou em desenvolver uma engenharia de domínio com a ajuda do MDA para facilitar na Engenharia de Aplicação, no contexto de computação ubíqua.

O trabalho de Carromeu (2007) teve foco na criação de uma linha de produtos de software para automatização da Engenharia de Aplicação no que tange a software de governo federal no padrão web.

Por outro lado, a proposta da pesquisa de reúso em sistemas financeiros de Reinehr (2008) mostra a falta, a importância e a busca das organizações da Linha de Produtos de Software para amenizar o problema de acoplamento e manutenção de aplicações.

Ao contrário das abordagens apresentadas, a proposta do presente trabalho é aplicar Linha de Produtos de Software no Ambiente Computacional Distribuído, abrangendo toda a parte de domínio e instanciando aplicações completas distribuídas na fase do desenvolvimento de produto.

O que faz a abordagem proposta ser diferente e não ter sido encontrada nos trabalhos até o momento é o fato de estar sendo sugerida uma nova abordagem para a construção de Ambientes Distribuídos com suporte da Linha de Produtos de Software, buscando uma maior reusabilidade dos artefatos atuando nas camadas de GUI, metaescalonamento e RMS.

4 PROPOSTA

Nesta seção apresentam-se a ideia principal do trabalho e o que será desenvolvido, bem como formas de aplicação do processo de Linha de Produtos de Software no Ambiente de Computação Distribuída. Também são explanados os projetos desenvolvidos pelo LaPeSD e como esses podem ser transformados em componentes e inseridos na arquitetura padrão.

4.1 CONSIDERAÇÕES INICIAIS

A abordagem de Grade Computacional em geral apresenta uma estrutura complexa e distribuída envolvendo hardware e software que permite acesso de baixo custo para recursos computacionais de alto desempenho (SILVA; DANTAS, 2007). No contexto dessa abordagem, podem-se encontrar vários elementos de rede, tais como interfaces gráficas de acesso ao meio, middleware, meta-escalonadores, entre outros, e, por consequência, várias áreas do conhecimento, podendo derivar daí um conjunto de outros elementos.

Nesse sentido, torna-se um desafio organizar sistematicamente tais elementos e arquitetura, visando ao planejamento de reúso sistemático e à rápida customização para o desenvolvimento de produtos que completem outras necessidades e possuam a mesma estrutura.

Um dos objetivos deste trabalho é aplicar o processo SMarty de Linha de Produtos de Software em Ambientes Computacionais Distribuídos, especificamente no projeto que vem sendo desenvolvido pelo laboratório LaPeSD, contribuindo para o desenvolvimento de técnicas para a sistematização de um processo e a reutilização dos artefatos, da arquitetura e dos componentes.

Em meio às abordagens de Linha de Produtos de Software existentes e pesquisadas, a abordagem SMarty foi escolhida devido à sua capacidade de modelagem dos artefatos e de integração com a UML.

Além disso, a abordagem SMarty leva em consideração trabalhos relacionados aos seguinte temas: gerenciamento de atividades, notação de artefatos, variabilidade de atributos e modelagem de metamodelos. O gerenciamento das atividades definidas para SMartyProcess está alinhado com os requisitos essenciais de Linha de Produtos (LINDEN;

SCHMID; ROMMES, 2007). A gestão de variabilidade é considerada um subprocesso da Linha de Produtos no gerenciamento de atividades, portanto ela tem uma estreita interação com atividades da Engenharia de Domínio e Aplicação (OLIVEIRA JUNIOR; GIMENES; MALDONADO, 2010).

As atividades definidas na abordagem SMartyProcess foram inicialmente baseadas em sugestões de atividades de gerenciamento de variabilidade (LINDEN; SCHMID; ROMMES, 2007; POHL; BOCKLE; LINDEN, 2005). No entanto, os trabalhos citados apenas listam atividades dentro de um contexto de um processo bem definido, com papéis, entradas e saídas. As atividades da SMartyProcess e sua associação de papéis e artefatos são totalmente especificadas (OLIVEIRA JUNIOR; GIMENES; MALDONADO, 2010).

SMarty usa a SMartyProcess como base de representação para os artefatos de Linha de Produtos em termos de conceito de variabilidade e seus atributos (OLIVEIRA JUNIOR; GIMENES; MALDONADO, 2010).

Optou-se por utilizar como ferramenta case (para modelar os diagramas da UML) a TTool, que é um conjunto de ferramentas dedicadas à edição de diagramas UML e SysML, além de simulação e validação formal desses diagramas. TTool suporta vários perfis da UML. A ferramenta foi escolhida por ser um ambiente livre de modelagem e possuir vários parceiros acadêmicos e da indústria.

Na sequência apresenta-se uma visão geral de todos os sistemas da família de produtos em Ambientes Computacionais Distribuídos produzidos e utilizados pelo laboratório LaPeSD. Esses sistemas são analisados com a intenção de encontrar similaridades e variabilidades para especificar a arquitetura e os componentes da Linha de Produtos.

4.1.1 Sobre o LaPeSD

O Laboratório de Pesquisa em Sistemas Distribuídos, também conhecido como LaPeSD, aplica seus esforços no desenvolvimento de uma linha de produtos no Ambiente Computacional Distribuído. Sua área de atuação, especificamente no Ambiente Distribuído, gira em torno de:

- bancos de dados distribuídos;
- bibliotecas digitais/serviços personalizados;

- computação de alto desempenho: redes, clusters e grids computacionais;
- middleware para Computação Distribuída;
- qualidade de serviço/acordo de nível de serviço;
- sistemas multimídias distribuídos; e
- web services.

A ideia de trabalhar com Engenharia de Software nos produtos do LaPeSD, através da Linha de Produtos de Software, é uma forma de organizar e sistematizar o desenvolvimento de software que até o momento vinha sendo feito sem nenhum auxílio, em que se encontravam problemas de alto acoplamento, falta de reúso, entre outros.

4.1.2 Família de produtos do LaPeSD

Pensando em recuperar e documentar as pesquisas realizadas pelo LaPeSD no domínio de Ambientes Computacionais, realizou-se um estudo sistemático e classificatório de cada trabalho produzido. O processo realizado procurou identificar as principais características dos trabalhos a fim de encontrar similaridades e variabilidades entre eles, área de atuação na arquitetura, pontos de variação, entre outros. Os trabalhos descritos nas próximas seções são relativos aos componentes de middleware, uma vez que estão focados nesse elemento.

4.1.2.1 Reserva dinâmica e antecipada de recursos para configurações multi-clusters utilizando ontologias e lógica difusa

A pesquisa realizada por Denise Janson Ferreira (2010) teve por objetivo a proposta de uma abordagem diferenciada de reservas antecipadas no gerenciamento de recursos distribuídos em grades para maximizar o nível de utilização desses recursos. Foi desenvolvida uma arquitetura de middleware para grades computacionais chamada GRADI, com a implementação para correserva visando ao melhoramento do nível de utilização dos recursos e do tempo de resposta total do sistema. O GRADI possui sistema de descrição e

seleção de recursos, e utiliza ontologias para a associação de recursos entre sistemas heterogêneos. Em suma, apresenta uma abordagem dinâmica para reservas imediatas e antecipadas, e um sistema de coalocação baseado em lógica difusa.

A Figura 15 apresenta os principais componentes do ambiente proposto por Ferreira (2010). No lado do cliente há uma interface interativa que recebe informações sobre restrições estabelecidas por um usuário.

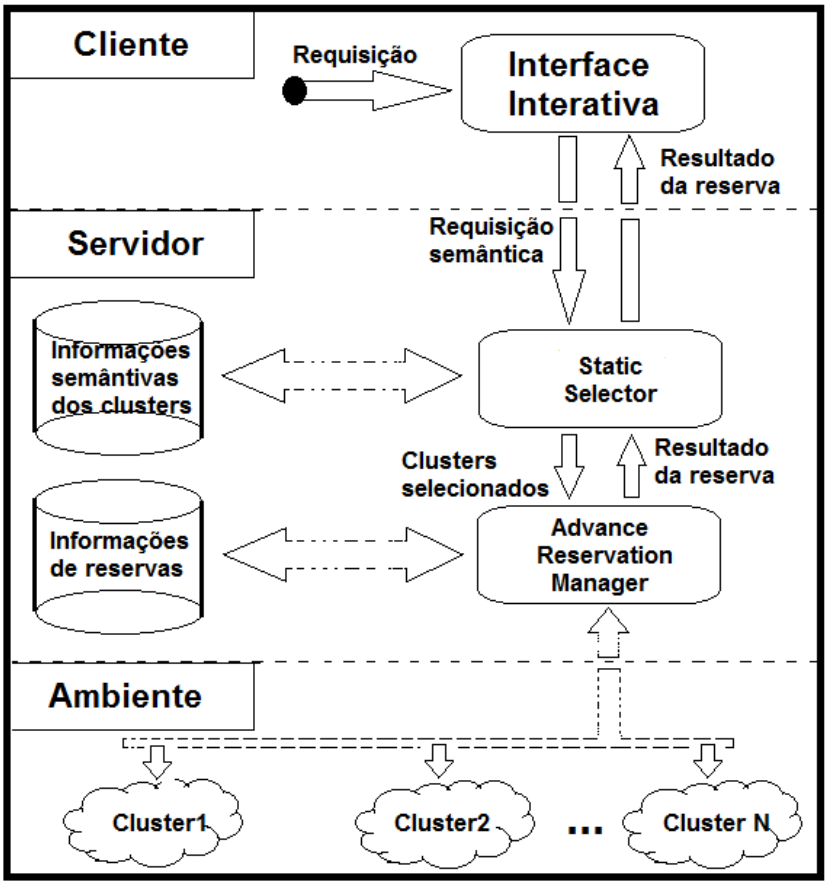


Figura 15: Arquitetura e componentes propostos
Fonte: Adaptado de: FERREIRA, 2010.

As restrições estabelecidas pelo usuário dizem respeito aos tipos e às quantidades de recursos necessários para a execução de uma aplicação. Assim que a requisição é enviada pelo cliente, passa pelo servidor, que possui informações semânticas estáticas a respeito dos recursos de cada cluster do ambiente. O Selector é responsável por fazer a comparação da requisição com as informações estáticas armazenadas. Todos os cenários possíveis para responder à requisição são encontrados nas informações estáticas.

Seguindo o fluxo, no componente Advance Reservation Manager, o AR_MANAGER compara resultados envolvendo também as reservas já realizadas e tenta encontrar resultados conflitantes. E então se pode definir o ambiente que será alocado para requisição, podendo, entretanto, dependendo da quantidade de processadores solicitados, não existir um cluster que consiga atender a essa requisição. Nesse caso, são realizadas reservas distribuídas entre clusters, controladas por um algoritmo baseado em lógica difusa. Além das funcionalidades citadas, o AR_MANAGER também conta com um mecanismo de conferência dinâmica da disponibilidade do ambiente dos recursos selecionados no momento da reserva. Esse componente é a principal contribuição do trabalho de Ferreira (2010).

4.1.2.2 Seleção de recursos baseada nas características de aplicações em ambientes de grade de multiagregados

O trabalho de Rodrigo Grumiche Silva (2011) trata da proposta e da avaliação de uma abordagem para seleção de recursos computacionais baseada em heurísticas, em cima de características estruturais e comportamentais de uma aplicação, e da relação de adequação dessas às características dos agregados computacionais num ambiente multiagregado.

Tem-se como objetivo diminuir a dificuldade da tarefa de seleção de recursos computacionais para o usuário, pois esse não terá que descrever em detalhes de recursos computacionais para que ocorra o processo de seleção. Tal facilidade se dá através do uso de ontologias para descrição semântica das características da aplicação e dos recursos computacionais baseados no trabalho de Silva e Dantas (2007).

Uma visão geral de como funciona o mecanismo de seleção dos recursos computacionais baseados nas características da aplicação é descrita pelo diagrama de atividade na Figura 16. Nesse diagrama, o

usuário informa as características da aplicação por meio de uma ontologia de descrição de características de uma aplicação.

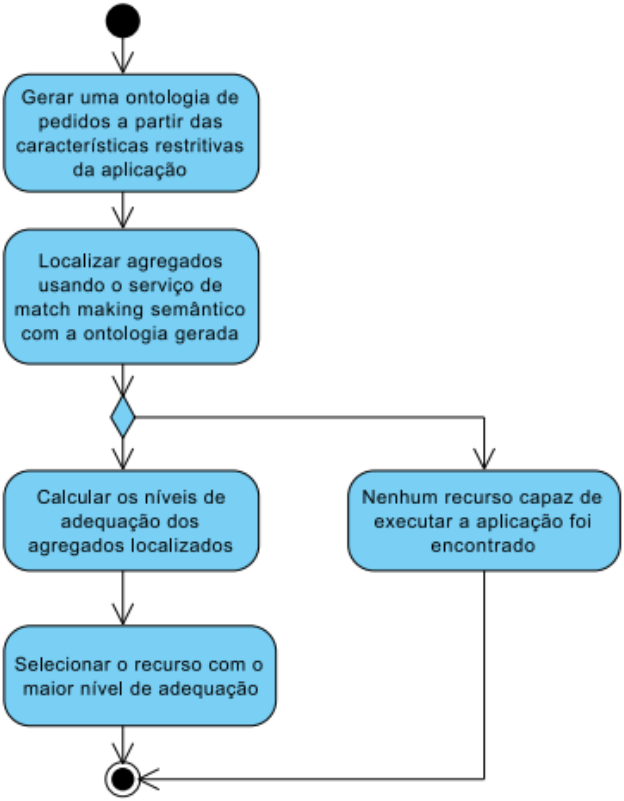


Figura 16: Diagrama de atividades do mecanismo de seleção
Fonte: SILVA, 2011.

Na primeira etapa do mecanismo de seleção, as características da aplicação são restritivas, ou seja, definem-se os requisitos mínimos a que um recurso deverá atender para ser candidato à execução e esses são transformados em uma ontologia.

Já na segunda etapa a descrição da ontologia de pedidos gerada é usada junto ao mecanismo de seleção. Então é recebida uma lista de recursos computacionais com respectivas descrições ontológicas, as quais são compatíveis com as características restritivas da aplicação.

Na terceira etapa um sistema de controle difuso avalia o nível de adequação de cada um dos agregados recebidos na segunda etapa. O nível de adequação vai de 0 a 100; quanto maior o nível, maior será o nível da adequação.

Na quarta e última etapa o agregador que tem o maior nível de adequação é selecionado.

4.1.2.3 Uma abordagem para reserva antecipada de recursos em ambientes de grades computacionais móveis

Segundo Vieira (2011), para um melhor reaproveitamento dos recursos da grade computacional, uma arquitetura para reserva antecipada de recursos, considerando as características da aplicação, é um fator determinante para reserva.

Nesse sentido, sua abordagem procura melhorar a qualidade das reservas, buscando o melhor nível de adequação dos recursos com base nas particularidades requisitadas pelo usuário. Além disso, sua proposta envolve uma interface móvel de acesso para os usuários interagirem através de dispositivos móveis.

De forma geral, pode-se visualizar a arquitetura dessa pesquisa na Figura 17, na qual o usuário do sistema interage com o ambiente através de um disposto móvel na GUI Graphical User Interface.

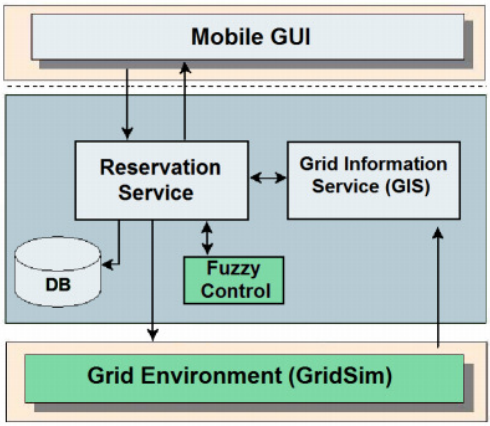


Figura 17: Visão geral dos componentes do sistema
Fonte: VIEIRA, 2011.

Através desse ambiente, o usuário insere informações relacionadas à reserva de recursos que deseja realizar, a qual pode ser antecipada ou imediata.

4.1.2.4 Conclusão e comparação das pesquisas

A tabela a seguir apresenta a síntese das pesquisas realizadas pelo LaPeSD. Tais trabalhos, mapeados abaixo, serão transformados em componentes de forma que poderão servir de variáveis para outros trabalhos. A tabela mostra as principais características entre os trabalhos.

Tabela 2: Comparação família de produtos LaPeSD

Autor	Camada de Atuação	Características
FERREIRA, 2010	Middleware	- Reserva antecipada. - Reserva imediata. - Correserva de recursos; considera carga atual da rede como critério de reserva.
SILVA, 2011	Middleware	- Seleção de recursos em nível de usuário final baseada nas características.
VIEIRA, 2011	Middleware	- Reserva antecipada e reserva imediata através de dispositivos móveis. - Seleção dos recursos a serem reservados baseados nas características da aplicação.

4.2 ARQUITETURA ATUAL

Uma linha de produtos de software representa um conjunto de sistemas compartilhando características e funcionalidades em comum

que satisfazem a necessidade de um mercado particular ou segmento específico (SEI, 2012). Tal conjunto de sistemas também pode ser chamado de família de produtos (NORTHROP, 2002).

Os membros da família de produtos são produtos específicos desenvolvidos sistematicamente pela Linha de Produtos de Software que também podem ser classificados como *core assets*. O *core asset* é representado por um conjunto de funcionalidades variáveis que indica uma decisão tardia de design de projeto. A escolha e a configuração dos *assets* compõem um produto específico (OLIVEIRA JUNIOR; GIMENES; MALDONADO, 2010).

Na família de produtos do LaPeSD, cada elemento apresentado na GUI Interface, no Metaescalonador e nos Escalonadores pode ser componentizado, fazendo parte de uma arquitetura comum chamada Engenharia de Domínio. E, posteriormente, a cada novo produto far-se-ia uma análise para a instância dessa arquitetura de Engenharia de Aplicação (EA) e da especialização das partes não comuns.

Por exemplo, podem-se aplicar ontologias voltadas para descrever as configurações existentes nos recursos disponíveis na rede, especializando apenas um dos componentes da arquitetura. Podem-se também adicionar novos elementos ou componentes à arquitetura, tais como uma camada superior para substituir a GUI por um aplicativo móvel com uma ontologia própria (em que esse componente faria parte do Domínio e seria reaproveitado posteriormente em outros produtos).

Todos os produtos da família de produtos LaPeSD podem ser desenvolvidos de maneira planejada, pensando em reúso e com pontos de variação (obrigatórios, opcionais e alternativos) com vistas à criação de uma linha de produtos de software no Ambiente Computacional Distribuído.

Cada componente na arquitetura do Ambiente Computacional Distribuído tem seu valor e propósito. A Figura 18 mostra como era a arquitetura padrão que se identificou utilizada em todos os projetos desenvolvidos pelo LaPeSD.

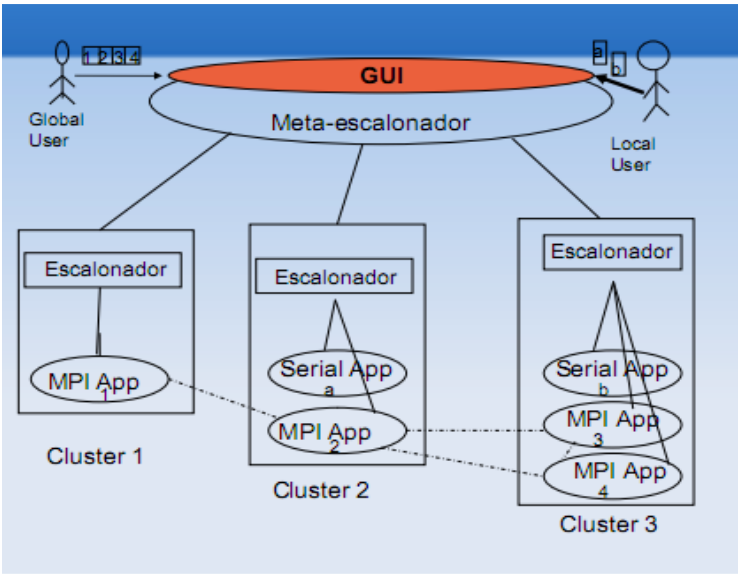


Figura 18: Arquitetura atual do Ambiente Computacional Distribuído

- GUI: nesta camada o usuário se utiliza de uma interface gráfica para configurar os parâmetros do middleware e da rede de forma amigável. Pode ser classificada com a camada de configuração do usuário.
- Metaescalonador: na arquitetura atual é no metaescalonador onde se concentra o maior número de trabalhos da família de produtos do LaPeSD. Como já descrito em capítulos anteriores, o middleware tem a função de conhecer e se comunicar com todos os clusters da rede. Ontologias e comunicação com escalonadores (uma implementação para cada escalonador) são função deste módulo também.
- Sistemas distribuídos: o nível mais baixo é aquele onde existem os recursos (podendo existir vários sistemas operacionais diferentes), como, por exemplo, nodos computacionais e áreas de armazenamento. Cada um deles tem seu próprio escalonador local. Este, em um ambiente real, poderia ser composto de gerenciadores, tais como SGE, LSF, TORQUE/PBS e Condor, dentre outros.

Especificamente, o componente de metaescalonador tem um papel fundamental na arquitetura, sendo o intermediário entre os clusters

(conhecido também como middleware) e conhecendo todos os da rede. Porém, como cada cluster pode ter um distribuidor e uma tecnologia diferente, o metaescalonador tem que conhecer a API bem como a forma de comunicação entre eles.

A cada novo cluster conectado na rede, o middleware deve ser informado e deve existir um cliente de comunicação com informações de localização, portas, componentes e descrições gerais, formando um novo componente que será agregado ao middleware com a informação de como será realizada a conversação entre tais pontas.

A intenção do presente trabalho é oferecer, através da Linha de Produtos de Software, uma forma sistemática, reutilizável e previamente planejada de adicionais, tais como componentes na arquitetura, sem precisar realizar grandes alterações de operações e do funcionamento das atividades dos clusters. Para isso, torna-se necessário realizar alterações na arquitetura padrão.

Este trabalho apresenta uma nova forma de arquitetura reutilizável e componentizada, com pontos de variações previamente definidos pensando em futuras novas funcionalidades, componentes e escalonadores.

Analisando a arquitetura atual de Ambientes Distribuídos, a seguir serão apresentadas as etapas de mapeamento das fases de análise de cada uma delas.

4.3 ARQUITETURA PROPOSTA

A arquitetura atual apresentada na seção anterior não atende às necessidades do cliente, dá suporte ao projeto de diferentes aplicações, mas existem dificuldades e problemas em manter e alterar a estrutura. A abordagem proposta através da Linha de Produtos de Software oferece uma nova forma reutilização da arquitetura e de componentes, bem como presta suporte no que diz respeito à manutenção e ao funcionamento do Ambiente Computacional Distribuído.

O processo genérico de Desenvolvimento de Linha de Produtos de Software compreende as etapas ilustradas na Figura 19 (GOMAA, 2005). A cada fase do modelo são produzidos e armazenados no repositório de Linha de Produtos de Software os artefatos produzidos.

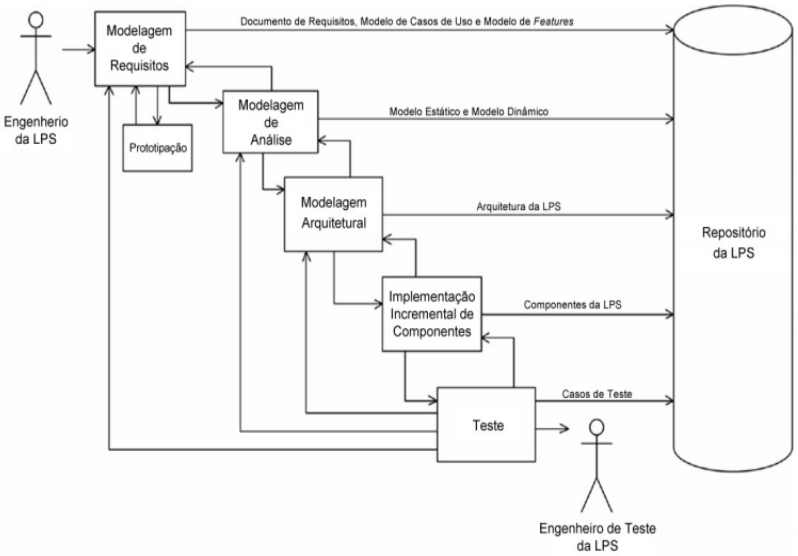


Figura 19: Fases da Engenharia de Linha de Produtos de Software
Fonte: GOMAA, 2005.

Os objetivos das etapas da Figura 19 são levantar requisitos e os casos de uso, derivar componentes e detectar a arquitetura e suas variabilidades.

Nas próximas seções será descrita com detalhes cada uma das fases citadas, iniciando-se pelos modelos de requisitos e casos de uso, seguidos da especificação técnica, da arquitetura e do componente do sistema, e finalizando com a etapa de testes e a integração do todo.

4.3.1 Modelagem de Requisitos

A Modelagem de Requisitos compreende a elaboração do modelo e do documento de requisitos, um modelo de casos de uso e um modelo de features. O modelo de casos de uso define requisitos funcionais da Linha de Produtos de Software em termos de atores e casos de uso. No caso deste trabalho, pode ser também realizado o modelo de similaridade e variabilidade a partir da abordagem SMarty. No Modelo de Features cada item é um requisito que define uma similaridade ou uma variabilidade que pode ser parametrizada.

4.3.1.1 Análise de requisitos

Na fase de análise de requisitos são especificados os principais requisitos funcionais e não funcionais para a Linha de Produtos de Software. Tais requisitos nas próximas fases serão analisados visando verificar a relação entre eles. Dessa forma, é necessário estabelecer restrições através de pré-requisitos. Os requisitos levantados irão guiar o restante do desenvolvimento, tanto para a construção da arquitetura quanto para a sua validação.

Inicialmente, para a atividade de levantamento de requisitos, o presente trabalho baseou-se na família de produtos do LaPeSD. Além dos requisitos do LaPeSD, utilizou-se também a avaliação comparativa que foi montada baseada nos artigos de Gaj et al. (2003), nas documentações mais recentes dos respectivos projetos, nas experiências relatadas pelos usuários e em avaliações pessoais sobre os conteúdos. Com essa tabela comparativa disponível no Apêndice A, conseguiram-se extrair as principais funções de um gerenciador de recursos (escalonamento e políticas de alocação em sistemas distribuídos heterogêneos), as quais são apresentadas a seguir.

É importante ressaltar que a escolha de um gerenciador de recursos depende do tipo de projeto, do workflow e das necessidades do usuário. Isso mostra que existe uma variabilidade já na escolha do gerenciador de recurso.

A) REQUISITOS FUNCIONAIS

1. **RF001 –** Reserva antecipada de recursos

Descrição: no sistema deve ser possível realizar reserva antecipada para que determinado job rode futuramente com os recursos previamente alocados.

2. **RF002 –** Reserva imediata de recursos

Descrição: no sistema deve ser possível realizar reserva imediata de recursos.

3. **RF003 –** Seleção de recursos baseada em características da aplicação

Descrição: deve ser possível no sistema realizar seleção de recursos baseada nas características da aplicação que será submetida.

4. **RF004 –** Submissão de aplicações ao processamento no ambiente através de interface com o usuário

Descrição: o sistema deve possuir uma interface amigável para submissão de aplicações que serão processadas de forma distribuída no ambiente através da *web-browser*.

5. **RF005 –** Submissão de aplicações ao processamento no ambiente através de XML

Descrição: o sistema deve possuir uma forma de submissão de aplicações via XML.

6. **RF006 –** Submissão de aplicações ao processamento no ambiente através de celular

Descrição: o sistema deve possuir uma interface amigável para submissão de aplicações que serão processadas de forma distribuída no ambiente através do celular.

7. **RF007 –** Autenticação no sistema

Descrição: o sistema deve possuir uma proteção de usuário e senha para submissão das aplicações.

8. **RF008 –** Gerenciamento de usuários

Descrição: o sistema deve possuir um local de gerenciamento de usuários através do cadastramento com dados pessoais, usuário e senha.

9. **RF009 –** Migração de processo para dispositivos móveis

Descrição: o sistema deve ter suporte à migração de módulos do metaescalonador para a abordagem de computação móvel, ou seja, deve existir uma forma de implementar uma versão para dispositivos móveis.

10. **RF010 –** Balanceamento de carga no escalonador de recursos

Descrição: o sistema deve possuir a funcionalidade de balanceamento de carga entre as máquinas.

11. **RF011 –** Realização de *checkpointing* no escalonador de recursos

Descrição: o sistema deve possuir a capacidade de realizar *checkpoints* periódicos.

12. **RF012 –** Tolerância a falhas

Descrição: o sistema deve possuir a capacidade de permitir que jobs migrem para execução em outros nós, evitando que se percam computações acumuladas caso haja falhas no sistema.

B) REQUISITOS NÃO FUNCIONAIS

1. **RNF001 –** Adicionar/Remover melhorias ou funcionalidades sem impactar o funcionamento da estrutura atual

Descrição: a estrutura deve suportar adição ou remoção de melhorias nas funcionalidades existentes na arquitetura, ou mesmo adição de funcionalidades e de componentes sem que a relação entre os serviços e os usuários sofra qualquer prejuízo, como, por exemplo, problemas de comunicação entre os serviços.

2. **RNF002 –** Serviços heterogêneos

Descrição: a arquitetura deve suportar qualquer sistema operacional e hardware.

3. **RNF003 –** Gerenciador de recurso *open-source*

Descrição: o sistema deve suportar gerenciadores de recursos, principalmente as versões *open-source*.

4. **RNF004 –** Linguagem de programação do gerenciador de recurso

Descrição: o sistema deve possuir linguagens de programação conhecidas no mercado, tais como C, C++ e Java.

4.3.1.2 Definição do Modelo de Casos de Uso segundo SMartyProcess

Partindo dos requisitos funcionais e não funcionais, podem-se identificar os casos de uso existentes do modelo de domínio. Antes de mostrar os diagramas de caso de uso, torna-se necessário também identificar os possíveis ATORES do sistema, bem como suas responsabilidades.

A) ATORES

- **Gerenciador da Arquitetura**
  - Instanciar a arquitetura para determinado cliente e situação.
  - Adicionar componentes na arquitetura principal.
  - Remover componentes da arquitetura principal.
- **Gerenciador do Ambiente**
  - Responsável por conhecer e gerenciar os gerenciadores de recursos disponíveis.
  - Responsável por conhecer os gerenciadores e o metaescalonador.
  - Responsável por configurar as ontologias.

- **Usuário Global ou Local**
  - Responsável por gerenciar o sistema de usuários e logar na aplicação.
  - Responsável por submeter os aplicativos ao processamento distribuído.

A Figura 20 mostra os três possíveis atores do sistema, bem como seus quatro casos de uso de negócio.

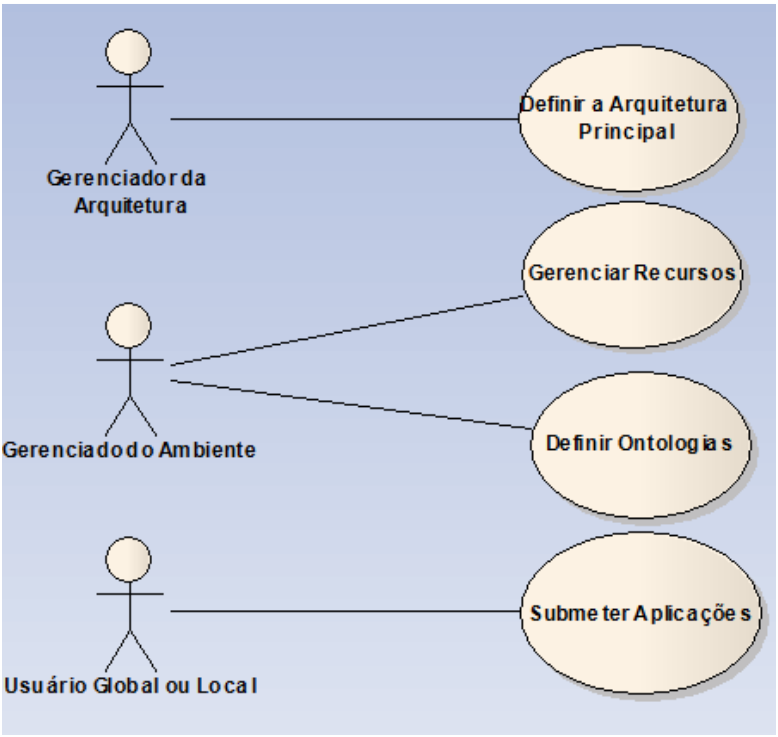


Figura 20: Atores e Modelo de Casos de Uso do domínio para Ambientes Distribuídos

B) CASOS DE USO

Tais casos de uso apresentados na Figura 20 devem ser refinados, explanados e transformados em outro(s) específico(s) que cada ator realiza em seus casos de uso, como se pode observar nas Figuras 21, 22 e 23.

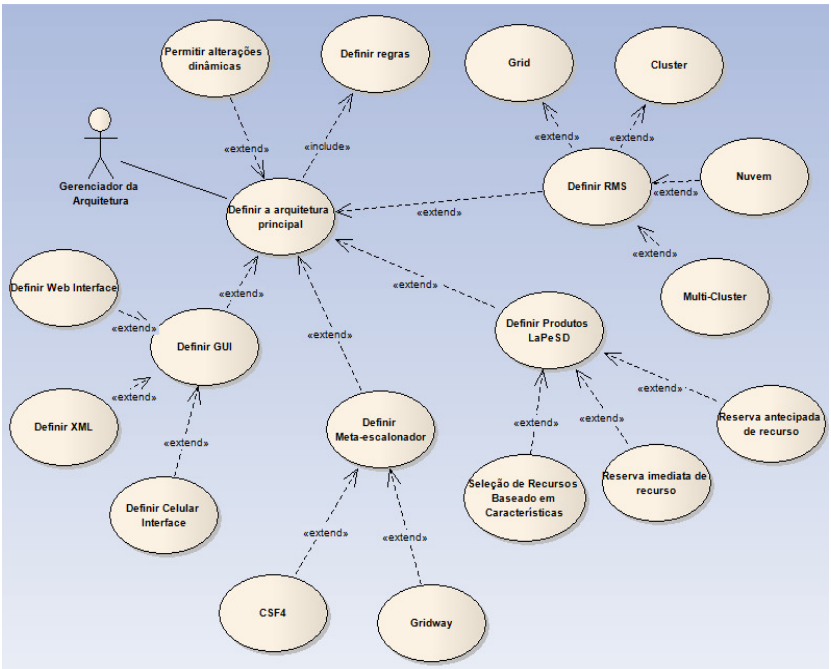


Figura 21: Diagrama de Caso de Uso para ator Gerenciador da Arquitetura

O diagrama de caso de uso descreve as funcionalidades propostas para um sistema que será desenvolvido. Segundo Booch, Rumbaugh e Jacobson (2005), podemos dizer que um caso de uso é um "documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo". Um caso de uso representa uma unidade discreta da interação entre um usuário (humano ou máquina) e o sistema (BOOCH; RUMBAUGH; JACOBSON, 2005); é uma unidade de um trabalho significante. Por exemplo, "login para o sistema", "registrar no sistema" e "criar pedidos" são todos casos de uso. Cada caso de uso tem uma descrição da funcionalidade que será construída no sistema proposto. Um caso de uso pode "usar" outra funcionalidade de caso de uso ou "estender" outro caso de uso com seu próprio comportamento.

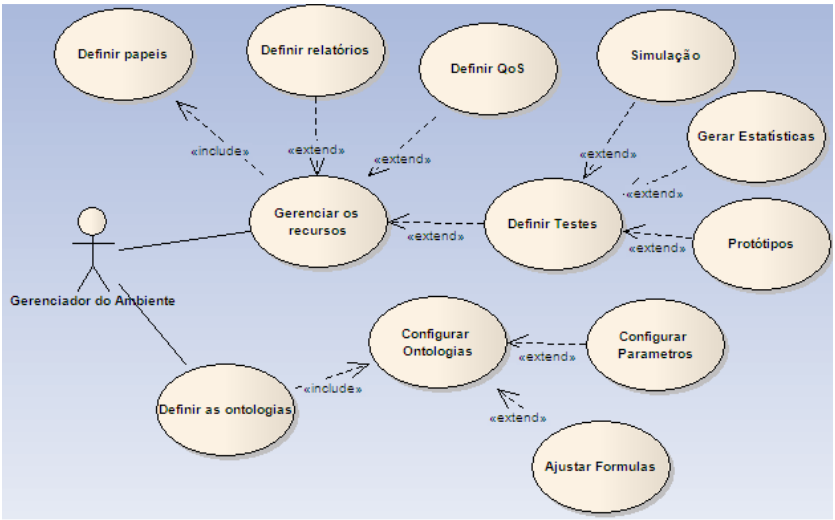


Figura 22: Diagrama de Casos de Uso para o ator Gerenciador do Ambiente

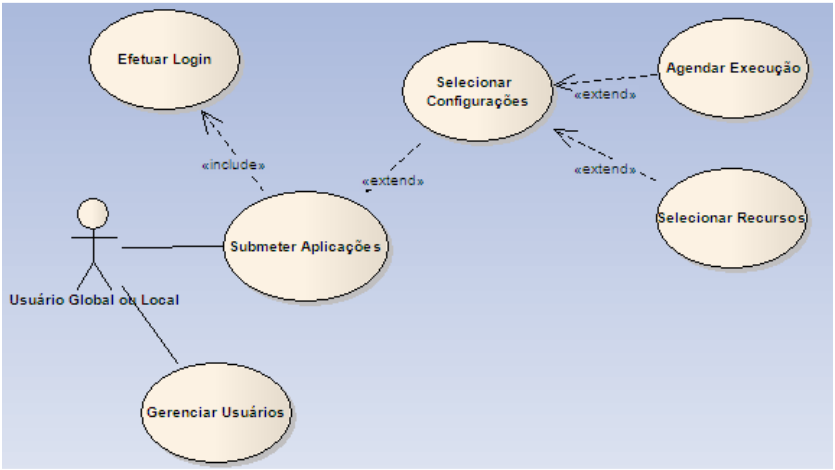


Figura 23: Diagrama de Casos de Uso para o ator Usuário Global ou Local

Seguindo o processo SMartyProcess (OLIVEIRA JUNIOR, 2010a) para identificação e gerenciamento de variabilidades, conforme a subseção 3.3.2, após a execução da atividade da elaboração do modelo de casos de uso, mapeada na execução da atividade do Desenvolvimento da Linha de Produtos, o SMartyProcess usa essa atividade como entrada para autoalimentar de forma iterativa a próxima atividade. Segundo o

processo, a tendência é de que as variabilidades cresçam à medida que suas atividades são executadas.

Entretanto, nessa primeira iteração da base de elaboração de casos de uso, ainda não há possibilidade de realizar a atividade de elaboração do modelo de rastreamento de variabilidades definida no processo, pois o *input* de tal atividade é o Modelo de Features que será realizado apenas na próxima etapa do processo de Desenvolvimento da Linha de Produtos de Software.

Partindo então diretamente para a atividade de identificação das variabilidades, serviram-se como artefatos de entrada os diagramas de casos de uso resultantes da atividade de elaboração do modelo de casos de uso. O artefato resultante da atividade de identificação das variabilidades é a evolução dos diagramas de casos de uso. Tais diagramas possuem agora as suas variabilidades identificadas.

Por consequência, a atividade de delimitação das variabilidades teve como artefatos de entrada os diagramas de casos de uso resultantes da atividade anterior. O artefato resultante dessa atividade é a evolução dos diagramas de casos de uso, com as variabilidades identificadas, porém essas, agora, apresentam-se delimitadas, conforme mostram as Figuras 24, 25 e 26. Essas figuras apresentam os diagramas de casos de uso com as variabilidades identificadas, representadas e delimitadas.

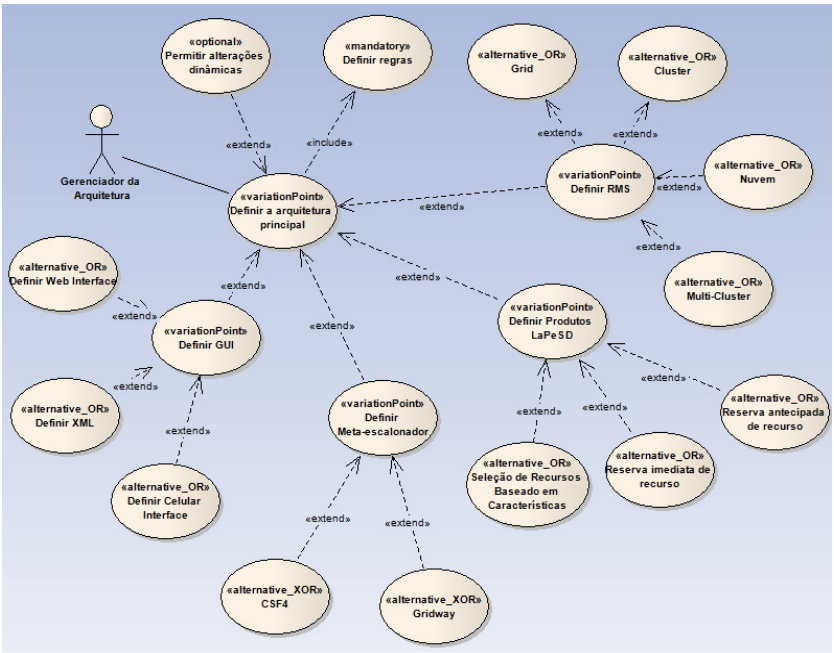


Figura 24: Diagrama de Caso de Uso para ator Gerenciador da Arquitetura com variabilidades identificadas e delimitadas

O Caso de Uso evoluído da Figura 24 apresenta vários pontos de variações, tais como “Definir a arquitetura principal”, “Definir GUI”, “Definir Metaescalonador”, “Definir Produtos LaPeSD” e “Definir RMS”. Outro fato importante a ser considerado é a ocorrência de atividades obrigatórias – “Definir regras” – e atividades opcionais – “Permitir alterações dinâmicas”.

Adicionalmente ao ponto de variação “Definir Metaescalonador”, existem duas variantes com estereótipo de “alternative_XOR”, isso indica que só uma delas pode ser escolhida. Tal especificação consta como relação de alternativa exclusiva na abordagem SMartyProcess.

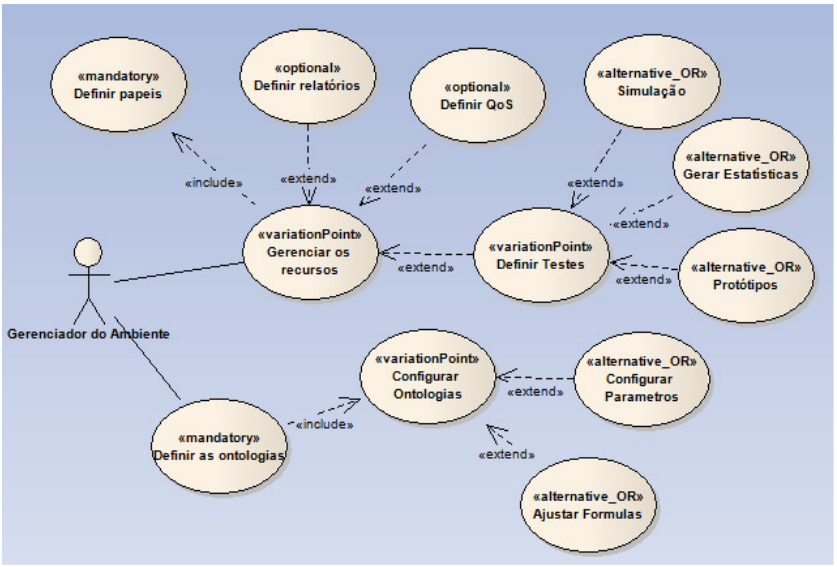


Figura 25: Diagrama de Casos de Uso para o ator Gerenciador do Ambiente com variabilidades identificadas e delimitadas

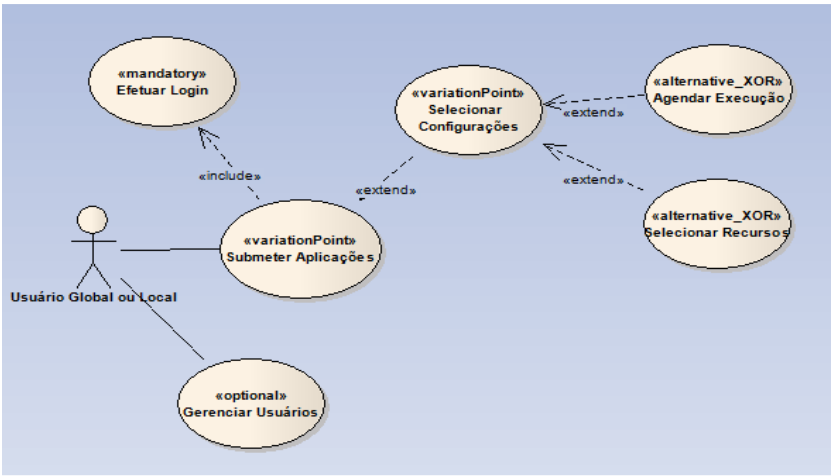


Figura 26: Diagrama de Casos de Uso para o ator Usuário Global ou Local com variabilidades identificadas e delimitadas

Na esfera da abordagem SMartyProcess a atividade de escolha dos mecanismos de implementação de variabilidades ainda não pode ser realizada devido à falta do diagrama de classes de componentes que

serão realizados nas próximas fases. Em suma, também não é possível realizar as atividades de rastreamento e controle de variabilidade, análise e configuração de produto, pois essas só existirão ao final da atividade de elaboração do Modelo de Features.

4.3.1.3 Features com abordagem SMarty

Também conhecido como Modelo de Características, a atividade de elaboração do Modelo de Features tem como entrada o artefato produzido na fase anterior do diagrama de Casos de Uso.

Segundo Oliveira Junior (2010a), nessa iteração ainda não é possível realizar a atividade de escolha dos mecanismos de implementação de variabilidade, por não ter passado ainda a base de Modelagem de Análise responsável pela geração dos artefatos de diagrama de classes e diagrama de componentes.

Uma das funções do Modelo de Features é detectar as dependências existentes entre as *features*. Tal relação de dependência é representada pelo estereótipo <<requires>>. É importante ressaltar que o Modelo de Features representado pela Figura 27 tem como nós iniciais os grupos correspondentes às *features* identificadas a partir do caso de uso apresentado na Figura 18 e seus refinamentos.

Além do Modelo de Features, nessa iteração também são elaboradas as atividades de rastreamento e controle de variabilidades, e análise de configurações de produtos.

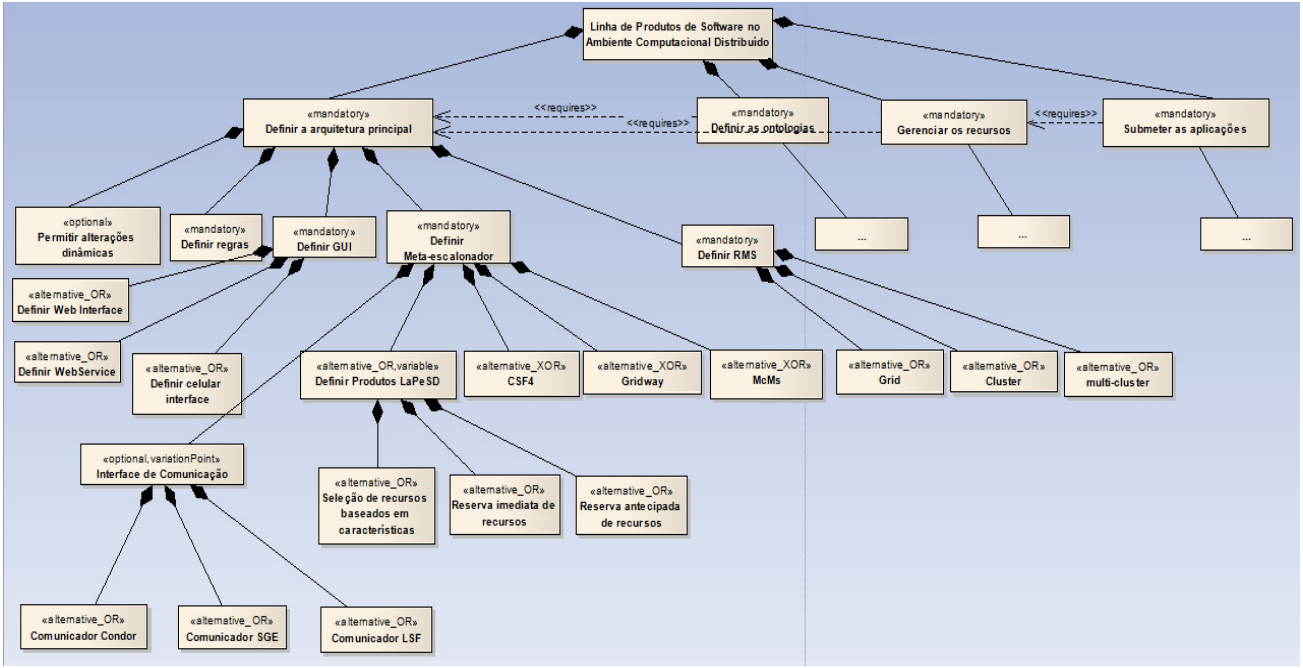


Figura 27: Modelo de Features para Ambientes Distribuídos

4.3.2 Modelagem de Análise

Na fase de Modelagem de Análise o principal modelo de desenvolvimento é o modelo estático, que define as relações estruturais entre as classes do domínio. Analisando as ações do sistema, são identificados os tipos e as ações relacionados (NISHIMURA, 2004).

Nos diagramas da Figura 27, são apresentados os componentes relacionados ao componente principal do caso de uso e todos os componentes que apresentaram variabilidade, juntamente com seus componentes opcionais e alternativos relacionados. Nesse sentido, para esse modelo são eliminados os componentes que não se enquadram na redação acima.

4.3.3 Modelagem de Componentes

Na fase de modelagem de componentes um subconjunto da Linha de Produtos de Software é selecionado para ser implementado em cada iteração. Com base nos casos de uso, o componente será gerado alinhado à arquitetura e adicionado no repositório de Linha de Produtos.

4.3.4 Modelagem Arquitetural

A fase mais importante da Linha de Produtos de Software é a Modelagem Arquitetural, pois diz respeito à especificação e à definição da arquitetura baseada nos componentes. Nesse caso o Modelo de Análise é endereçado para o Modelo Arquitetural, levando em consideração os padrões do software.

A Figura 28 representa a nova proposta de arquitetura utilizando Linha de Produtos de Software no Ambiente Computacional Distribuído. Chegou-se a tal arquitetura através do processo executado nos capítulos anteriores.

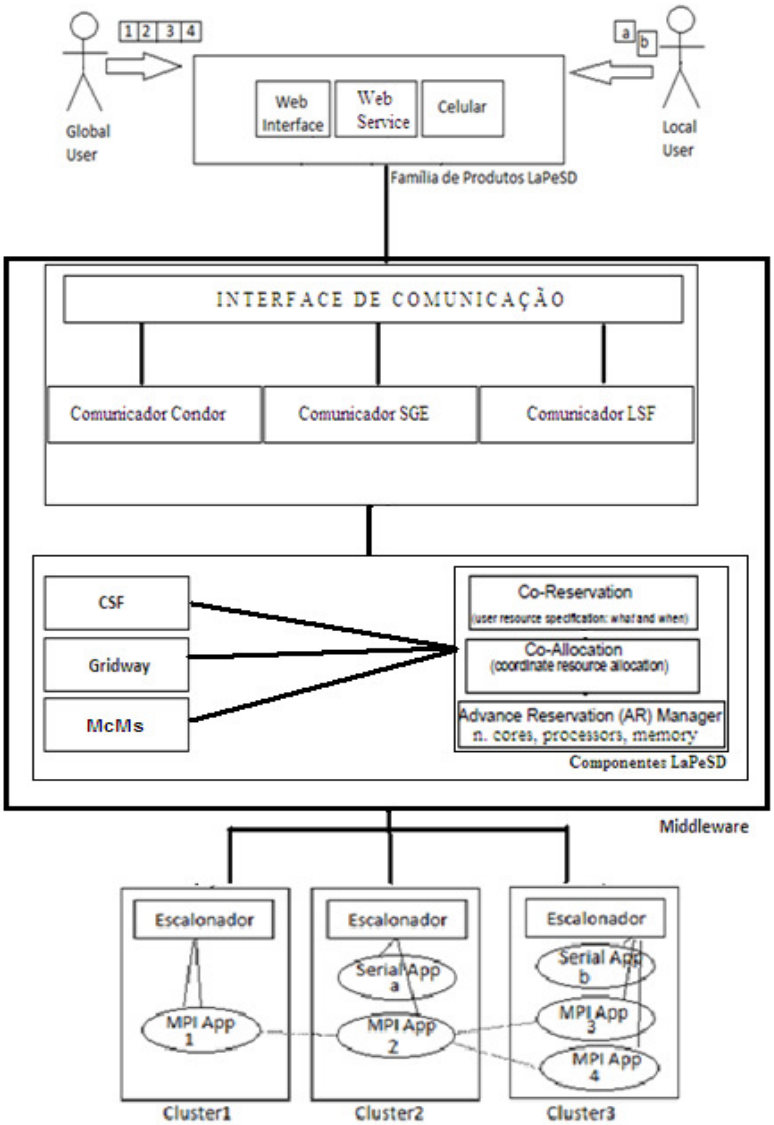


Figura 28: Arquitetura proposta

As variações encontram-se na arquitetura proposta nas três camadas de interface, middleware e RMS, as quais são descritas abaixo.

Além disso, muitos dos elementos viraram componentes da arquitetura, quais sejam:

- Interface: nesta primeira camada encontra-se um ponto de variação com três variações de possíveis escolhas opcionais:
    - web: apresentação visual web para o cliente manipular o ambiente;
    - celular: apresentação visual restrita a algumas funcionalidades em dispositivos móveis; e
    - client web service: conjunto de classes para integração com outros sistemas e criação de novas variantes de interface.
- Drivers de comunicação: ponto de variação que abstrai a interface do middleware com seus possíveis comandos. Existem implementados atualmente três pontos de variação:
    - Condor: variante que pode ser composta de outros pontos de variação, tais como alocação de recursos que não são implementados pelo condor;
    - SGE: variante. Neste caso o SGE já implementa reserva antecipada e não precisa de tal componente; e
    - LSF: não implementado no trabalho, porém passível de implementação utilizando como base o middleware.
- Middleware: ponto de variação composto de variantes ou componentes do LaPeSD:
    - middleware McMs: variante, middleware implementado pelo LaPeSD;
    - middleware externo: variantes externas;
    - reserva antecipada: variante e componente implementados no LaPeSD; e
    - alocação de recursos: variante e componente implementados no LaPeSD.
- RMS: ponto de variação. Duas variantes foram implementadas e testadas no LaPeSD:
    - SGE: variante com versão *free* e comercial mais completa; e
    - Condor: variante com versão *free* e *open-source*.

Essa figura ilustra, de forma geral, o funcionamento do ambiente. Nesse ambiente, usuários com aplicações paralelas ou sequenciais podem requerer que essas sejam executadas, utilizando-se da mesma interface. Na figura, a aplicação paralela é representada pela aplicação

com quatro processos (processos 1, 2, 3 e 4). Por outro lado, as aplicações sequenciais são representadas pelas aplicações *a* e *b*.

O nível de interação dos usuários é com uma interface gráfica (GUI). Já o metaescalonador na parte inferior do módulo da figura se comunica com nodos para que as aplicações e suas partes (i.e., *1, 2, 3, 4, a* e *b*) sejam executadas nas configurações mais apropriadas. É importante observar que as *partes* podem ser enviadas para um ambiente em que não exista um gerenciador de recursos, como, por exemplo, um conjunto de PCs executando o sistema operacional Windows.

A camada de visão é composta de interfaces de comunicação e submissão de jobs, bem como o usuário pode selecionar os recursos e os atributos necessários. Logo em seguida vem o principal ponto de variação da arquitetura que é a **interface de comunicação,** deixando como pontos de variação os variantes Comunicador Condor, Comunicador SGE e Comunicador LSF. Tais itens, se instanciados na Engenharia de Domínio, devem ser implementados conforme a especificação de cada RMS. Podem existir inclusive mais elementos como eles, dependendo dos RMSs adicionados na arquitetura. É importante ressaltar que a implementação das interfaces faz parte da família de produtos LaPeSD.

4.3.5 Testes de integração e avaliação da arquitetura

Esta etapa é necessária para realizar os testes de integração entre componentes e arquitetura, bem como as funcionalidades da Linha de Produtos. Tal etapa será executada no capítulo de experimentos e Engenharia de Aplicação.

4.4 DISCUSSÃO GERAL

Com foco no Ambiente Computacional Distribuído, o presente trabalho se propõe a organizar sistematicamente os elementos e a arquitetura, visando ao planejamento de reúso sistemático e à rápida customização para desenvolvimento de novos produtos que completem outras necessidades e possuam a mesma estrutura.

Para alcançar esse objeto, este trabalho aplica o processo SMarty especificamente no projeto que vem sendo desenvolvido pelo

laboratório LaPeSD da Universidade Federal de Santa Catarina, contribuindo para o desenvolvimento de técnicas de sistematização de um processo para reutilização dos artefatos, da arquitetura e de componentes de Linha de Produtos de Software em ambientes computacionais distribuídos.

Em meio às abordagens de Linha de Produtos de Software existentes e pesquisadas, a abordagem SMarty foi escolhida devido à sua capacidade de modelagem dos artefatos e de integração com a UML.

A arquitetura atual tem como características:

- camada de GUI e Metaescanolador fortemente acoplados;
- dificuldade de manutenção em ambas as partes. A cada novo RMS adicionado na arquitetura principal, deve-se reimplementar e gerar uma elevada manutenção nas camadas de GUI e Metaescalonador;
- a família de produtos do LaPeSD não está componentizada, não sendo possível a rápida implementação dos produtos nos clientes;
- ausência de uma abordagem de reúso, além da ausência de documentação e de padronização no processo de desenvolvimento; e
- ausência de abstração na API da GUI e Metaescalonador perante os RMSs.

Já a arquitetura proposta

- possui um Modelo de Domínio bem definido, explicitando a arquitetura principal e seus componentes, bem como a família de produtos do LaPeSD;
- possui delimitação e gerenciamento de variabilidades através de pontos de variação e variantes;
- padroniza e separa os requisitos da arquitetura e seus componentes, bem como cria um caminho padrão para as aplicações;
- possui documentação e artefatos suficientes para o entendimento do todo; e
- busca, através da Linha de Produtos de Software, aumentar o *time-to-market*, a produtividade e a qualidade; e diminuir o acoplamento e o curto.

Para que um processo de engenharia de aplicação seja realizado, é necessária a existência de um conjunto de componentes de software disponíveis para reutilização que foram definidos nesta seção. Na próxima seção, será abordada a questão da geração de aplicações no contexto do desenvolvimento, baseado em reutilização através da engenharia de aplicação.

5 ESTUDO DE CASO E RESULTADOS EXPERIMENTAIS

Na seção anterior dedicou-se a detalhar uma nova abordagem de criação de Ambientes Distribuídos apoiada na Linha de Produtos de Software. Neste capítulo, por sua vez, dedica-se a aplicar a abordagem proposta à luz do referencial teórico e dos conceitos discutidos e propostos até então.

O principal objetivo nesta seção é a engenharia de aplicação em um projeto real do LaPeSD, seguindo a engenharia de domínio definida na seção anterior. O nome da empresa será preservado devido a sigilo de contratos. Entretanto, as informações sobre arquitetura, componentes e artefatos em geral são reais.

Pelos motivos expostos e discutidos neste trabalho em torno de Ambientes Distribuídos, é importante que a abordagem seja projetada e desenvolvida em nível de software, visando efetuar essa agregação cooperativa para atender às solicitações das mais diversas aplicações.

Com vistas a abstrair a complexidade inerente ao Ambiente Distribuído, a primeira camada modelada na arquitetura é a interface gráfica, na qual o usuário submete de fato seus jobs. O presente trabalho está focado no desenvolvimento, na implementação e na validação dessa camada, uma vez que outros pesquisadores estão contribuindo para a pesquisa, implementando e validando as outras camadas.

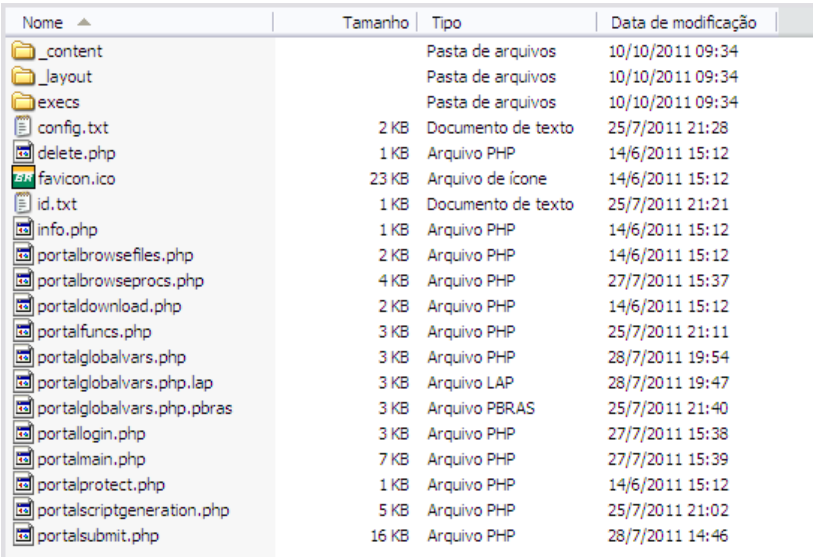
A aplicação dar-se-á através do caso de uso especificamente na camada da interface, usando os pontos de variação e as variâncias delimitados através da Linha de Produtos de Software, usando a abordagem SMartyProcess: Interface Web, Interface Celular e Web Service. Tais elementos formarão a família de produtos LaPeSD em nível de interface com o usuário.

Adicionalmente às interfaces, conforme apresentado na Engenharia de Domínio, criou-se uma abstração da interface de comunicação com os vários RMSs que a arquitetura pode vir a utilizar. Cada comunicador e variância do Condor, SGE e LSF foi implementado e será descrito a seguir. Os comunicadores especificam a forma de conversação com cada RMS, visto que cada um tem seus comandos.

Antes de apresentar o ambiente proposto, é exibido como funciona hoje o ambiente atual para posteriormente comparar com o que foi realizado de melhorias.

5.1 AMBIENTE ATUAL

As classes não puderam ser modeladas, pois o projeto da interface encontra-se implementado em PHP4, sem suporte de programação orientada a objetos. Foi encontrado um total de mais de 40 arquivos entre php, html e js. A Figura 29 mostra os arquivos do portal (interface gráfica GUI) do ambiente atual.



Nome	Tamanho	Tipo	Data de modificação
_content		Pasta de arquivos	10/10/2011 09:34
_layout		Pasta de arquivos	10/10/2011 09:34
execs		Pasta de arquivos	10/10/2011 09:34
config.txt	2 KB	Documento de texto	25/7/2011 21:28
delete.php	1 KB	Arquivo PHP	14/6/2011 15:12
favicon.ico	23 KB	Arquivo de ícone	14/6/2011 15:12
id.txt	1 KB	Documento de texto	25/7/2011 21:21
info.php	1 KB	Arquivo PHP	14/6/2011 15:12
portalbrowsefiles.php	2 KB	Arquivo PHP	14/6/2011 15:12
portalbrowseprocs.php	4 KB	Arquivo PHP	27/7/2011 15:37
portaldownload.php	2 KB	Arquivo PHP	14/6/2011 15:12
portalfuncs.php	3 KB	Arquivo PHP	25/7/2011 21:11
portalglobalvars.php	3 KB	Arquivo PHP	28/7/2011 19:54
portalglobalvars.php.lap	3 KB	Arquivo LAP	28/7/2011 19:47
portalglobalvars.php.pbras	3 KB	Arquivo PBRAS	25/7/2011 21:40
portallogin.php	3 KB	Arquivo PHP	27/7/2011 15:38
portalmain.php	7 KB	Arquivo PHP	27/7/2011 15:39
portalprotect.php	1 KB	Arquivo PHP	14/6/2011 15:12
portalscriptgeneration.php	5 KB	Arquivo PHP	25/7/2011 21:02
portalsubmit.php	16 KB	Arquivo PHP	28/7/2011 14:46

Figura 29: Arquivos do ambiente atual: interface gráfica GUI

No portal atual existe uma tela inicial de segurança na qual o usuário fornece suas informações de acesso ao ambiente através dos pares *username* e *password,* representados pela Figura 30.

Figura 30: Tela de acesso ao ambiente atual

Assim que o usuário obtiver acesso ao sistema, a primeira tela do ambiente mostra uma listagem contendo todos os jobs que estão sendo executados na Figura 31. Na listagem são mostrados os campos Comando: mostra o arquivo que está sendo executado; Processos: mostra detalhes do job em execução em nível de log; Submissão: apresenta a data e a hora da submissão do job; Estado: mostra o *status* de conclusão do job; S.O: é o sistema operacional que está sendo executado; e Apagar: fornece ao usuário a possibilidade de abortar a execução do job. Importante ressaltar que a listagem fica sendo atualizada de cinco em cinco minutos.

Figura 31: Jobs em execução no ambiente atual

No canto superior esquerdo existe um menu com a opção de submissão de jobs. Ao clicar nesse item, o usuário tem a possibilidade de escolha e submissão de um job específico (Figura 32). Existem três regiões na tela de submissão, são elas:

- Executável: o campo “Upload” é utilizado para enviar um executável armazenado localmente e o campo “Comando” para indicar um comando já instalado nos nodos de execução. Ao enviar arquivos via Upload, deve-se ter certeza de que os nodos de execução contêm todas as bibliotecas ou DLLs necessárias já instaladas. Se necessário, deve-se utilizar o campo “Argumentos” para indicar qualquer número de parâmetros necessários à execução. Os argumentos podem conter a variável $range desde que essa tenha sido definida no campo apropriado (ver Arquivos de Entrada). O campo “Servidor” encontra-se inoperante nesta versão do portal;
- Arquivos de Entrada: utiliza-se o campo “Locais” caso seja necessário enviar arquivos de dados armazenados localmente para utilização pelo comando a ser executado. Se for necessário enviar mais de um arquivo, pode-se utilizar o botão "+". Cada arquivo enviado corresponderá a uma execução do comando e os resultados (impressões para a tela) dessa execução estarão disponíveis em um arquivo com o mesmo nome do arquivo enviado, com extensão ".txt". Para enviar arquivos de entrada localizados remotamente – ou para indicar onde colocar arquivos de saída –, utiliza-se o campo “Servidor” para indicar o nome do servidor Windows/Samba (no formato Windows. ex.: \\servidor\diretório). Na sequência, pode-se usar o campo “Remotos” para indicar o(s) arquivo(s). Esse campo aceita uma lista (separada por ";") de nomes de arquivo, máscaras em formato DOS (ex.: *.txt ou arq??.dat) e a utilização da variável $range. Os arquivos indicados neste campo terão o mesmo funcionamento dos indicados no campo “Locais” quanto à execução e a eventuais impressões para a tela. Se necessário, pode-se indicar no campo “Range” uma faixa numérica no formato N:M (ex.: 001:050). O campo “Comuns” deve ser usado quando for necessário deslocar qualquer número de arquivos (identificados por uma lista separada por ";" e aceitando nomes ou máscaras DOS) para cada nó antes da execução do comando (ex.: usa-se este campo quando o comando requer múltiplos arquivos de entrada ou bibliotecas não presentes nos nodos de execução); e
- Arquivos de Saída: todos os arquivos de saída gerados em cada execução são automaticamente recuperados pelo sistema e disponibilizados para visualização. Utiliza-se o campo

"Download" para indicar uma lista (separada por ";" e aceitando nomes e máscaras DOS) de arquivos. Se o destino desses arquivos for "Local", esses serão colocados em um único arquivo (.tar) para posterior download via browser. Se o destino for Servidor Remoto, esses serão deslocados para o servidor e o diretório indicados em Arquivos de Entrada.

Figura 32: Tela de submissão de jobs do ambiente atual

A submissão e a listagem de jobs são feitas pelo sistema operacional diretamente ao Condor, que é o único RMS instalado e disponível no ambiente atual, conforme demonstram a Figura 33 e a Figura 34.

```
414  $retorno = shell_exec("( export CONDOR_CONFIG=/etc/condor/condor_config;
415      ".$condor_home."bin/condor_submit " . $file_sub . ") 2>>
416          ".$target_path_error."/output.err");
417  $retorno ? portalLog($retorno) : errorLog(join("", file($target_path_error."output.err")));
418  $retorno=shell_exec($chmod_bin." -R a+rw ".$target_path_execs_job." 2>> "
419      .$target_path_error."output.err");
```

Figura 33: Código de submissão do ambiente atual ao Condor

```
128 function updateStatus() {
129     global $condor_home;
130     $_POST['console'] = shell_exec("(export CONDOR_CONFIG=/etc/condor/condor_config; " .
131         $condor_home . "bin/condor_q )");
132 }
```

Figura 34: Código de listagem do ambiente atual ao Condor

A seguir na Tabela 3 são listados os pontos fortes e fracos encontrados no ambiente atual.

Tabela 3: Pontos fortes e fracos do ambiente atual

Pontos Fortes	Pontos Fracos
Fácil utilização do usuário	Não é possível escolher o RMS e a execução
Simplicidade de instalação	Fortemente acoplado ao RMS utilizado
Possibilidade de executar um arquivo que se encontra em outro computador da rede	Não fornece possibilidade de integração com outras interfaces
	Não fornece possibilidade de execução de jobs em sistemas operacionais Linux
	Não existe uma interface padrão com a definição dos possíveis métodos
	Autenticação de usuários fixos no código-fonte
	Não é possível listar os RMSs do ambiente
	Ausência de programação orientada a objetos
	Ausência de reúso dos artefatos
	Utiliza apenas o Condor como RMS

Através da Linha de Produtos de Software e da abordagem proposta no capítulo anterior, serão observados os pontos fracos visando melhorar sistematicamente o desenvolvimento dos produtos do contexto Ambiente Computacional Distribuído e, principalmente, fornecer um maior reúso entre os artefatos do projeto.

5.2 AMBIENTE PROPOSTO

É apresentado nas subseções que seguem o processo de Engenharia de Aplicação baseado na modelagem do ambiente proposto no Capítulo 4.

5.2.1 Engenharia de Aplicação

Na Engenharia de Aplicação é construído um produto específico de software (seleção de produto baseada na Engenharia de Domínio) através do reúso sistemático e organizado dos artefatos de domínio.

Dessa forma, os requisitos para a criação do novo produto em questão podem ser descritos como

- interação com os usuários para a obtenção de informações de execução da aplicação;
- utilização de um metaescalonador aberta (*open-source*) que permite que novas facilidades sejam consideradas, tais como reserva antecipada, QoS, uso de vários gerenciadores de recursos;
- uso de ontologia, ou seja, a adoção de uma abordagem cultural na empresa em reservas e reservatórios pode ser um importante fator na descrição do tipo de recursos (processador, memória e/ou armazenamento) necessários para a execução da aplicação;
- migração de um módulo do metaescalonador para a abordagem de computação móvel. Em outras palavras, o desenvolvimento de uma versão para dispositivos móveis deve permitir a utilização do ambiente sob o paradigma de qualquer lugar e a qualquer momento;
- adoção de orientação a contexto, que deverá permitir que a localização do usuário possibilite que recursos mais apropriados sejam alocados devido à sua localização. Esse fato pode ser mais bem visualizado quando enlaces de comunicação locais são mais rápidos do que aqueles para conexão em ambientes mais distantes ou ainda em configurações computacionais que requeiram privilégios especiais que o usuário não tem; e
- adoção do paradigma de software aberto (*open-source*) no tocante a ambientes em que a não existência de gerenciadores de recursos é fato. De outra forma, ambientes, como por exemplo, Windows, que não disponham de um gerenciador de recursos possam ser considerados para a execução de aplicações de reserva e reservatórios, sem que os usuários locais sejam perturbados. Em outras palavras, o sistema deverá fazer a submissão da aplicação para uma (ou mais) máquina(s)

Windows, sem que seus usuários locais percebam essa carga extra de processamento.

Um caminho para que esses requisitos sejam atendidos é prover para os usuários das aplicações uma forma semântica fácil para que suas requisições sejam traduzidas em solicitações de recursos disponíveis nas configurações dos clusters, de grids e até mesmo de nuvens computacionais. O uso de ontologias voltadas para descrever os recursos existentes nas configurações é uma vertente de pesquisa bastante considerada por vários pesquisadores de inúmeras instituições no Brasil e no exterior.

A instanciação a partir da Engenharia de Domínio visando à geração de um produto que se denomina Engenharia de Aplicação utilizou os elementos descritos na Figura 35. A intenção é derivar um produto com a capacidade de alternar RMS sem a necessidade da reimplementação da camada superior, na qual não era possível com a estrutura anteriormente implementada.

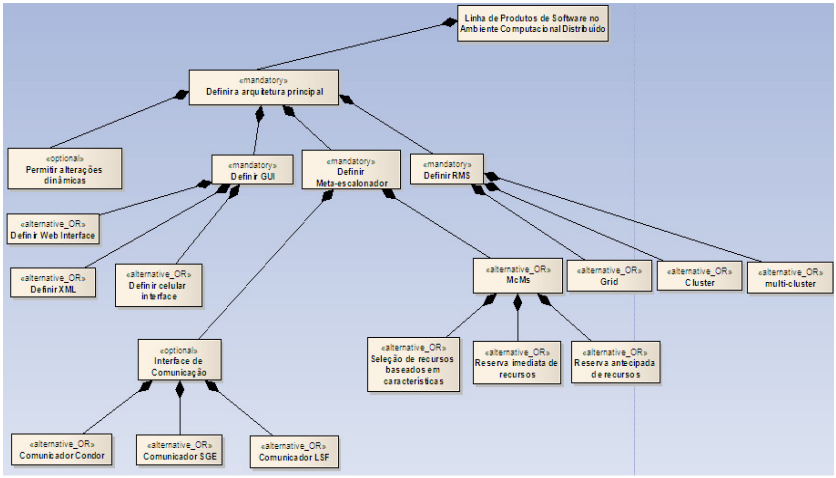


Figura 35: Elementos implementados na Engenharia de Aplicação

Primeiramente foram implementados os componentes apontados pela Engenharia de Domínio no ponto de variação “Definir GUI”. As variantes definidas como componentes “Definir Web Interface”, “Definir Celular Interface” e “Definir Web Service” foram derivadas seguindo a definição da arquitetura principal, que inclusive se encontra em aberto para a criação de novos componentes de interface GUI, como,

por exemplo, a integração da arquitetura com outros sistemas através de Web Services.

A garantia da abstração e da possibilidade de adição e remoção entre os possíveis RMSs da arquitetura se dá através do ponto de variação “Interface de Comunicação” e suas variantes “Comunicador Condor”, “Comunicador SGE” e “Comunicador LSF”. As possíveis mensagens de comunicação entre os RMSs são definidas na “Interface de Comunicação”, porém a implementação concreta de cada uma delas encontra-se em suas variantes.

Como não é escopo do presente trabalho a implementação do metaescalonador, o ponto de variação “Definir Metaescalonador” e “McMs”, bem como suas variantes foram “apenas” modelados, desacoplados e desenhados na arquitetura principal. Sua implementação é trabalho de outro pesquisador do laboratório LaPeSD. É importante ressaltar que este trabalho forneceu a visibilidade necessária para a possível implementação do McMs, podendo-se, se necessário for, utilizar outro metaescalonador de terceiros.

Entretanto, o metaescalonador proposto deverá se encarregar de conectar as melhores opções de gerenciamento de recursos RMS (por exemplo, LSF, SGE e Condor) e fazer com que a aplicação seja alocada no melhor ambiente disponível e ocioso possível. Deve-se entender que o sistema deverá fazer alocação inclusive para ambientes em que não existe um gerenciador de recursos, como o caso dos PCs rodando Windows.

5.2.2 Definir GUI

As interfaces GUI implementadas têm como objetivo fornecer ao usuário final, definido no presente trabalho como ator “Usuário Global ou Local”, uma forma para submeter os jobs à arquitetura que será executa de forma distribuída. Conforme demonstrado na Figura 26, o Diagrama de Casos de Uso para o ator Usuário Global ou Local deverá ser implementado: “Efetuar Login”, “Gerenciar Usuário”, “Submeter Aplicações ou Jobs”, “Agendar Execução” e “Selecionar Recursos”.

A Figura 36 apresenta a interação do usuário com os componentes que serão implementados e deverão possuir as características descritas no caso de uso citado no texto anterior.

Figura 36: Componentes de interface da Linha de Produtos LaPeSD

De fato, essa fase trata da abordagem de construção dos produtos da Linha de Produtos de Software utilizando os *core assets* definidos na Engenharia de Domínio. Devido ao que foi desenvolvido dos *core assets,* a fase de Engenharia de Aplicação se torna uma tarefa mais simples e consequentemente demanda menos tempo e esforço (LINDEN; SCHMID; ROMMES, 2007).

5.2.2.1 Interface web

Tanto na interface web como na interface celular a primeira tela será a de segurança, devendo o usuário entrar com seu login e senha para ter acesso ao sistema. A tela inicial do ambiente mostra uma listagem contendo todos os jobs que estão sendo executados na Figura 37. Na listagem são apresentados os seguintes campos: "Id": identificador do job que está sendo executado. Este campo serve para resgate do job para uma possível ação de deleção, edição ou mais informações; "Owner": o usuário local ou global que submeteu o job ao ambiente; "Priority": o nível de prioridade de processamento dos jobs. Tendo como 0 o mais prioritário, 1 o menos prioritário, e assim por diante; "ProgramName": diz respeito ao nome do job que está em execução bem como sua extensão, como, por exemplo, .class feito pelo programa em Java; "runTime": tempo de execução do job*;* "Status": se o job está em execução, hibernado ou finalizado; "SubmittedDate e SubmittedHour": data e hora de submissão do job ao ambiente; "RMS": em qual RMS está sendo executado o job*;* e, por último, porém não menos importante, o botão de remover um job do ambiente.

É importante ressaltar também os filtros existentes nessa tela. A qualquer momento o usuário pode filtrar apenas os jobs entre determinado período para examinar melhor seus jobs. Além disso, existe o campo "Tecnologia", que é preenchido com os RMSs que estão

instalados no ambiente. Essas informações vêm do Web Service, que armazena (configurado pelo ator Gerenciador do Ambiente) todos os RMSs configurados e instalados em uma tabela de configuração no banco de dados. Na hipótese de um usuário leigo submeter um job e não possuir experiência necessária para entender o que é um RMS, esse campo pode ficar em branco e assim o Web Service escolhe o RMS que tem mais disponibilidade para executar tal job. A aplicação Web foi desenvolvida utilizando a IDE Eclipse (ECLIPSE, 2011).

Os diagramas de classe da aplicação Web encontram-se no Apêndice B deste trabalho. Eles foram divididos pela quantidade de classes existentes.

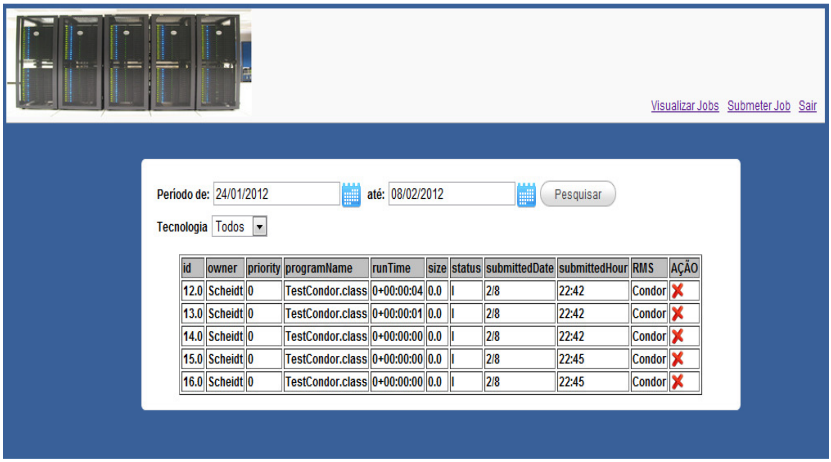


Figura 37: Jobs em execução apresentados na web

5.2.2.2 Interface celular

O desenvolvimento da abordagem móvel para que os usuários possam executar a qualquer hora e em qualquer lugar suas aplicações dá-se através da interface implementada utilizando JME (ORACLE, 2012), com as mesmas características e campos da interface web, porém com a comodidade de ser uma tecnologia móvel. A mesma implementação foi realizada com a tecnologia Android (ANDROID, 2012). A Figura 38 mostra a mesma listagem da web, porém no celular. É importante ressaltar que, por ser um dispositivo restrito, as

funcionalidades e o número de campos também são restritos. A manipulação completa da aplicação se dá através da interface web.

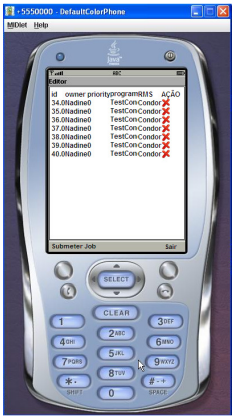


Figura 38: Jobs em execução apresentados no celular

5.2.2.3 Interface Web Service – Integração

A interface Web Service foi implementada para mostrar que existe também uma forma de integração da interface direta com outros possíveis sistemas já existentes nos clientes.

Independentemente da interface gráfica escolhida, a comunicação deve ser feita através da fronteira Web Service exportada pela interface de comunicação, expondo esta um WSDL com os métodos disponíveis. Cada uma das interfaces GUI deve implementar essa fronteira e gerar os Stubs de acesso ao Web Service.

5.2.3 Interface de comunicação

Uma vez que a forma de comunicação foi desacoplada da interface e criada uma estrutura para ela na borda do middleware, isso faz com que os componentes de interface sejam componentizados e fiquem livres para criação e evolução de outros futuros trabalhos de interface. Por exemplo, através dos comandos especificados na interface de comunicação, um futuro trabalho poderia desenvolver uma interface

gráfica que suporte um levantamento dinâmico utilizando lógica de *fuzzy* da configuração de hardware e software.

Adicionalmente, a Figura 39 mostra os pontos de variação e as variantes de comunicação em que, através da camada "Interface de Comunicação" representada por um ponto de variação, podem existir futuras implementações de novos comunicadores com novos RMSs adicionados na arquitetura. No caso da Figura 39, os seguintes elementos são variações: Comunicador Condor, Comunicador SGE e Comunicador LSF.

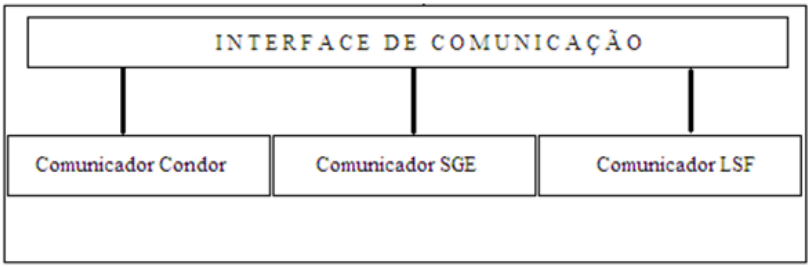


Figura 39: Ponto de variação e variantes de comunicação

Foi realizado um levantamento junto aos especialistas de Computação Distribuída do laboratório LaPeSD e mapearam-se as possíveis mensagens que serão disponibilizadas e implementadas na "Interface de Comunicação" para as interfaces gráficas, bem como implementadas concretamente em cada comunicador.

Tabela 4: Mensagens existentes na "Interface de Comunicação" ligadas aos RMSs

Mensagem	**Função**	**Comando no SGE**	**Comando no Condor**
submitJobSynch()	Este método é chamado para submeter um job à execução de forma síncrona.	Qsub	Condor_submit
submitJobAsynch()	Este método é chamado para submeter um job à execução de forma assíncrona. E retorna um job ID.	Qsub	Condor_submit

Mensagem	Função	Comando no SGE	Comando no Condor
delJob()	Usado para cancelar/deletar um job em execução.	Qdel	Condor_rm
getJobs()	Lista os jobs que estão em execução.	Qstat	Condor_q
listNodes()	Mostra *status* e informações sobre a execução dos *hosts* e nodos de execução.	Qhost	Condor_status

Com base nas mensagens descritas na Tabela 4, foram implementados em linguagem de programação JAVA (ORACLE, 2011) os comunicadores "Comunicador Condor" e "Comunicador SGE". No caso do presente trabalho, foram implementados o Condor e o SGE, pois esses fazem parte do projeto em que estão sendo aplicados.

Além dos comandos para a manipulação dos jobs, foi encontrada a necessidade de acrescer novos comandos para o gerenciamento de usuários e a manipulação dos RMSs existentes cadastrados. Para esse gerenciamento foi criada uma estrutura local com base de dados própria. A Tabela 5 lista os possíveis comandos fornecidos na interface para o gerenciamento local.

Tabela 5: Mensagens existentes na "Interface de Comunicação" ligadas ao gerenciamento local

Mensagem	Função
Authentication()	Verifica se um usuário está devidamente cadastrado no sistema local através de usuário e senha.

Mensagem	Função
getRMSs()	Retorna quais RMSs existem cadastrados e implementados no sistema. Serve para apresentar ao usuário as opções de onde ele deseja executar seus jobs.

Para expor os métodos das Tabelas 4 e 5 de modo que os atores "Global User" e "Local User" possam submeter jobs, gerenciar usuários e RMSs, foi criado um Web Service a fim de tornar possível qualquer implementação na camada superior ou de GUI. Dessa forma, os métodos foram expostos através da classe "Communication", como mostra a Figura 40.

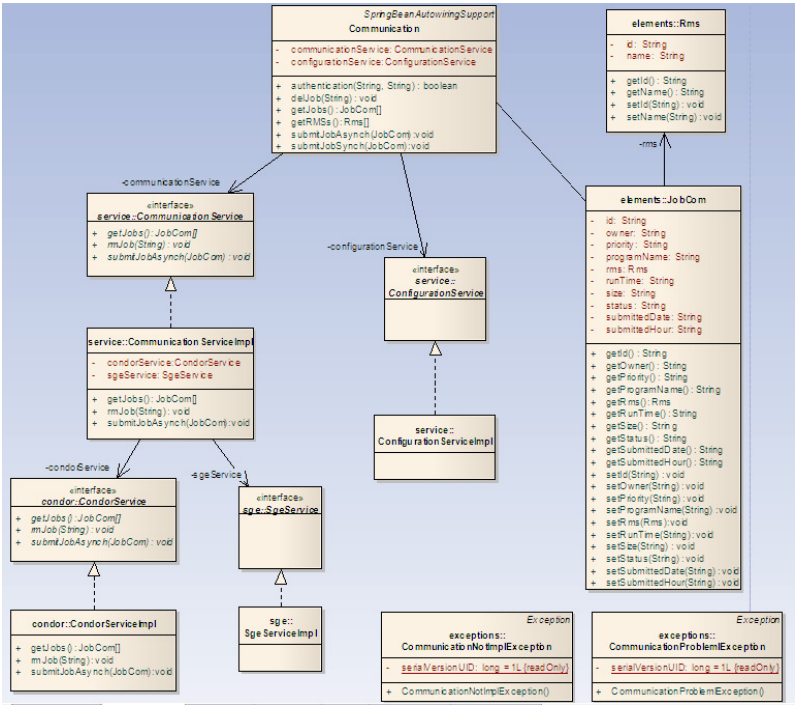


Figura 40: Implementação da Camada de Comunicação

Assim que a interface de comunicação representada pela classe “Communication” no Web Service recebe uma requisição, ela é encaminhada para a classe “CommunicationServiceImpl”, que irá verificar qual RMS deve ser manipulado. Por exemplo, a submissão de um job está condicionada à escolha do usuário na interface de GUI sobre qual RMS ele quer executar esse job. Assim que o Web Service receber a requisição, ele sabe se vai enviar para a classe “CondorService” ou “SgeService”.

A classe “CondorService” fará acesso ao RMS local instalado do Condor e tratará suas requisições e especificações apenas para ele. Por exemplo, o comando de submissão de jobs no comando é submit_condor. Nesse sentido, existe um conjunto de classes específicas para realizar o tratamento de entrada e saída das informações no Condor, conforme mostra a Figura 41:

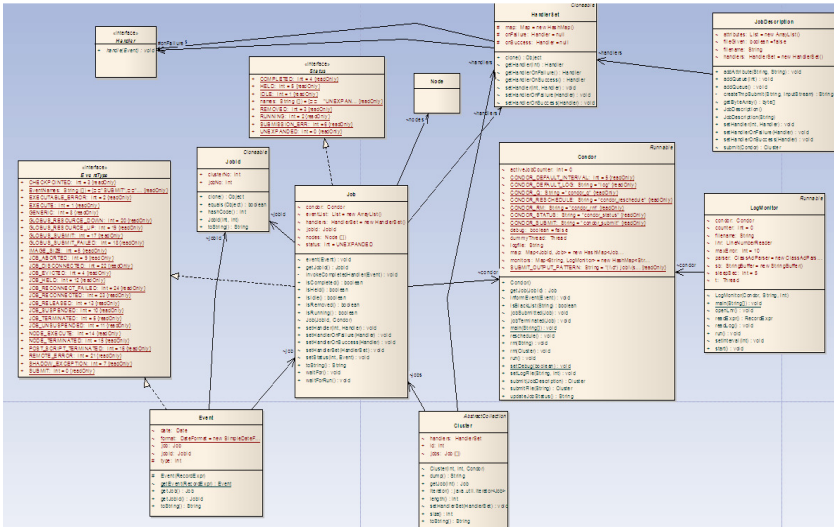


Figura 41: Conjunto de classes para a manipulação do Condor

O outro RMS local implementado no presente trabalho é o SGE, representado pela classe “SgeService”. Da mesmo forma que o Condor, ele trata suas requisições de entrada e saída com um conjunto de classe específico de seu entendimento, implementado no presente trabalho.

Como é proposto e objetivado neste trabalho, se um dia for necessário adicionar um novo RMS, deve-se criar uma nova interface,

como “NovoRmsService”, e criar o conjunto de classes necessárias para manipular tal RMS. Em suma, deve-se adicioná-lo na tabela de RMSs da base local para ser apresentada na interface GUI ao usuário como um RMS válido de submissão e manipulação de jobs.

5.3 CONSIDERAÇÕES FINAIS

Realizou-se engenharia reversa no código do antigo portal a fim de fazer uma comparação entre as arquiteturas. Porém, como foi implementado em PHP e sem o auxílio de programação orientada a objetos, não foi possível criar o diagrama de classes do projeto. Conforme apresentado, o ambiente atual possui uma estrutura muito acoplada e de difícil manutenção e extensão para adição de outros RMSs.

O problema do ambiente atual é que os gerenciadores de recursos de grades computacionais possuíam diversas funcionalidades, mas tinham de ser adaptados ao que os clientes necessitavam. Para isso, novos componentes deveriam ser adicionados e não existia nenhum padrão, documentação ou forma de reutilização dos componentes e da arquitetura.

A forma de implementação do ambiente atual deixou a arquitetura completamente dependente do Condor, não sendo possível adicionar outras formas de comunicação pela interface nem adicionar novos RMSs. Utilizando a abordagem proposta, deixou-se a interface componentizável e criou-se uma camada paralela de comunicação com a implementação de cada RMS possivelmente adicionado, abstraindo, assim, os elementos plugados na arquitetura.

É importante ressaltar que, para o desenvolvimento do trabalho, foi essencial a aplicação da Linha de Produtos de Software nos Ambientes Distribuídos, tornando possível a componentização dos elementos, bem como a criação de variações e variantes entre eles.

Conforme a aplicação da nova abordagem, agora é possível remover o Condor e adicionar outros gerenciadores, tal como o SGE, em que existe uma forma transparente de funcionamento, sem a necessidade de grandes alterações na arquitetura e no ambiente.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo apresenta um sumário do trabalho realizado nesta dissertação, juntamente com a sumarização dos trabalhos futuros.

6.1 CONCLUSÃO

Cada vez mais através da Engenharia de Software procura-se a reutilização de elementos para o reaproveitamento em projetos do mesmo domínio. Essa prática está diretamente relacionada ao melhoramento no custo dos projetos, no tempo e na produtividade (SEI, 2012). A Linha de Produtos de Software vem para contribuir com todas essas características e cada vez mais aumentar a qualidade do produto final, a produtividade, entre outros.

Neste trabalho foi proposta uma linha de produtos de software para o Ambiente Computacional Distribuído. Para alcançar esses objetivos, foi realizada a análise do funcionamento atual e em seguida foram desenvolvidas uma engenharia de domínio e a engenharia de aplicação baseada no domínio. O estudo de caso foi realizado no laboratório LaPeSD, simulado em um projeto real, e continua em desenvolvimento utilizando a abordagem realizada neste trabalho.

A disciplina de Computação Distribuída no que tange à Engenharia de Software é pouco amparada ou pouco utilizada. Através da Linha de Produtos de Software busca-se uma forma aplicada de documentar e buscar a reusabilidade sistemática dos componentes distribuídos.

A abordagem SMarty foi utilizada e se mostrou de extrema importância para, de forma iterativa, encontrar, definir e gerenciar variabilidades no ambiente, distribuindo de acordo com o desenvolvimento genérico da Linha de Produtos de Software e influenciando cada uma das etapas a fim de encontrar as variabilidades de forma fácil.

A adoção de conceitos de Linha de Produtos de Software fornece um caminho para uma melhor integração dos componentes do ambiente de forma rápida e segura, além de a abordagem também suportar uma maciça customização.

A reutilização de artefatos e do produto final diminui o número de erros, aumenta a conformidade do software produzido, facilita a manutenção e reduz o tempo de desenvolvimento.

A abordagem está sendo aplicada agora em projetos reais, com a abordagem já conseguindo resolver problemas práticos relacionados à indecisão do usuário sobre quais elementos na sua arquitetura gostaria de ter, podendo rapidamente adicionar e remover elementos sem grandes problemas.

6.2 TRABALHOS FUTUROS

A Linha de Produtos de Software é uma área pouco explorada e com grande capacidade de aplicação na Engenharia de Software. Prova disso são os poucos trabalhos existentes na literatura e na rede. No decorrer do desenvolvimento do trabalho foram identificadas algumas possibilidades de trabalhos futuros, quais sejam:

- aplicar Linha de Produtos de Software na computação ubíqua ou computação das coisas. Tais disciplinas parecem trabalhar bem, uma vez que é notória uma arquitetura padrão com várias variabilidades em torno de um ambiente ubíquo;
- relacionar Linha de Produtos de Software ao processo de Engenharia do CMMI e/ou MPS.BR, respectivamente, no nível 4, E e C dos processos;
- especificar e implementar WorkFlow na aplicação web para facilitar todos os componentes da LP, visando identificar todas as possíveis combinações e facilitar a usabilidade do usuário leigo;
- implementar o metaescalonador citado McMs e seus componentes, conforme Linha de Produtos do LaPeSD;
- aplicar métricas de Linha de Produtos de Software para validar quantitativamente a abordagem;
- construir uma ferramenta automatizada de apoio à geração de engenharia de aplicações através do domínio;
- estudar teste e avaliação de arquiteturas de LP, uma vez que essa atividade foi incorporada ao processo de desenvolvimento de LP existente. Para tanto, é necessário um estudo minucioso sobre as técnicas de teste e avaliação existentes na literatura, como, por exemplo, os conceitos de engenharia de software experimental e a avaliação empírica de métodos e ferramentas.

REFERÊNCIAS

AIKEN, R. et al. Network Policy and Services: A Report of a Workshop on Middleware. *Technical Report (RFC 2768)*. IETF, 2000.

ALMEIDA, Eduardo Santana de. *RiDE*: The RiSE Process for Domain Engineering. 2007. 278 f. Tese (Doutorado em Ciência da Computação) – Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2007.

ANDROID. Disponível em: <http://www.android.com/>. Acesso em: 10 jan. 2012.

BLOIS, A. P. T. B. *Uma proposta de projeto arquitetural baseado em componentes no contexto de Engenharia de Domínio*. Exame de Qualificação – Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modelling Language User Guide*. Addison Wesley Longman, Inc., 1999.

_____. *Unified Modeling Language User Guide*. 2nd Edition. Addison-Wesley Professional, 2005. (Addison-Wesley Object Technology Series).

BOSCH, J. Preface. In: The 2nd GRONINGEN WORKSHOP ON SOFTWARE VARIABILITY MANAGEMENT: SOFTWARE PRODUCT FAMILIES AND POPULATIONS, 2004, Groningen, The Netherlands. *Proceedings…* Groningen, The Netherlands, 2004.

BOTE-LORENZO, M.; DIMITRIADIS, Y.; GÓMEZ-SÁNCHEZ, E. Grid Characteristics and Uses: A Grid Definition. *Grid Computing*, p. 291-298, Springer 2004.

BRAGA, R. M. M. *Busca e recuperação de componentes em ambientes de reutilização de software*. Tese (Doutorado em Engenharia de Sistemas e Computação) – Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2000.

BUYYA, R. *Grid Computing Info Centre (GRID Infoware)*. Disponível em: <http://gridcomputing.com>. Acesso em: 10 maio 2011.

CARROMEU, Camilo. *Linha de Produtos de Software no processo de geração de sistemas web de apoio à gestão de fomento de projetos*. 2007. 100 f. Dissertação (Mestrado) – Universidade Federal do Mato Grosso do Sul, Mato Grosso do Sul, 2007.

CLEMENTS, P.; KAZMAN, R.; KLEIN, M. *Evaluating Software Architectures*: Methods and Cases Studies. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

CLEMENTS, P.; NORTHROP, L. *Software Product Lines*: Practices and Patterns. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

COLVERO, T. A.; DANTAS, M.; CUNHA, D. P. da. *Ambientes de clusters e grids computacionais:* características, facilidades e desafios. Criciúma: Unesc, 2005.

CONDOR HIGH THROUGHPUT COMPUTING. Disponível em: <http://www.cs.wisc.edu/condor/>. Acesso em: 13 jun. 2011.

CSF. COMMUNITY SCHEDULER FRAMEWORK. Disponível em: <http://www.globus.org/grid_software/computation/csf.php>. Acesso em: 7 jan. 2012.

CZARNECKI, K.; EISENECKER, U. W. *Generative Programming*: Methods, Tools, and Applications. Addison-Wesley, 2000.

DANTAS, M. *Computação distribuída de alto desempenho*: redes, clusters e grids computacionais. Rio de Janeiro: Axcel Books, 2005.

DE ROSE, C. A. F.; FERRETO, T. C. *Improving Performance Analysis Using Resource Management Information*. Springer Berlin/Heidelberg, 2003.

DE ROURE, D. et al. The Evolution of the Grid. *Grid Computing: Making the Global Infrastructure a Reality*, v. 13, p. 14-15, 2003.

DING, Z. et al. Implement the Grid Workflow Scheduling for Data Intensive Applications with CSF4. In: FOURTH IEEE INTERNATIONAL CONFERENCE ON E-SCIENCE, 2008, Indianápolis. p. 563-569.

ECLIPSE PROJECT. Disponível em: <http://www.eclipse.org>. Acesso em: 20 dez. 2011.

EDSON JUNIOR, E. A. Variabilidade em Linha de Produtos de Software: como identificar, delimitar e representar variabilidade nos principais modelos UML de uma linha de produtos de software. *Java Magazine: Engenharia de Software*, Rio de Janeiro, p. 1-15, 2010.

EMAM, Khaled El; KORU, A. Günes. *A Replicated Survey of it Software Project Failures*. Los Alamitos, Ca, Usa: Ieee Computer Society Press, 2008.

ETXEBERRIA, L.; SAGARDUI, G. Evaluation of Quality Attribute Variability in Software Product Families. In: THE ANNUAL IEEE INTERNACIONAL CONFERENCE AND WORKSHOP ON THE ENGINEERING OF COMPUTER BASED SYSTEMS, 2008, Washington, DC, USA. *Proceedings*... Washington, DC, USA: IEEE Computer Society, 2008. p. 255-264.

FERREIRA, Denise Janson. *Reserva dinâmica e antecipada de recursos para configurações Multi-Clusters utilizando ontologias e lógica difusa*. 2010. 114 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2010.

FOSTER, I. Globus Toolkit Version 4: Software for Serviceoriented Systems. In: INTERNATIONAL CONFERENCE ON NETWORK AND PARALLEL COMPUTING, 2006, Springer-Verlag LNCS. p. 2-13.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, v. 15, n. 3, p. 200-222, 2001.

GAJ, K. et al. Effective Utilisation and Reconfiguration of Distributed Hardware Resources using Job: Management Systems. In: THE PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2003. *Proceedings*... 2003.

GIMENES, I. M. S.; TRAVASSOS, G. H. O enfoque de Linha de Produto para Desenvolvimento de Software. In: XXI JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI), evento integrante do XXII Congresso da SBC, julho 2002, Florianópolis. p. 1-31.

GOMAA, H. *Designing Software Product Lines with UML*: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, 2005.

GRISS, M. L.; FAVARO, J.; D'ALESSANDRO, M. Integrating Feature Modeling with the RSEB. In: FIFTH INTERNACIONAL CONFERENCE ON SOFTWARE REUSE – ICSR5, 1998, Victoria, British Columbia, Canada. *Proceedings...* Victoria, British Columbia, Canada, 1998. p. 76-85.

GURP, J. V.; BOSCH, J.; SVAHNBERG, M. On the Notion of Variability in Software Product Lines. In: THE WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2001, Washington, DC, USA. *Proceedings...* Washington, DC, USA, 2001.

JAVADI, B.; AKBARI, M.; ABAWAJY, J. A Performance Model for Analysis of Heterogeneous Multi-Cluster Systems. *Parallel Computing*, v. 32, n. 11/12, p. 831-851, 2006.

KANG, K. C. et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Relatório Técnico CMU*. Pittsburgh, USA: Carnegie-Mellon University (SEI), 1990. SEI-90-TR-21.

KORNILOVICZ, Karen. *Adesão das empresas brasileiras ao MPS.BR em 2006 supera expectativas da SOFTEX*. Disponível em: <http://www.softex.br/mpsbr/_noticias/noticia.asp?id=1069>. Acesso em: 21 dez. 2006.

LEE, Kwanwoo; KANG, Kyo C.; LEE, Jaejoon. *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. Springer Berlin/Heidelberg, 2002.

LINDEN, F. J. V. D.; SCHMID, K.; ROMMES, E. *Software Product Lines in Action*: The Best Industrial Practice in Product Line

Engineering. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

MCT. Ministério da Ciência e Tecnologia. *Qualidade e Produtividade no Setor de Software Brasileiro em 2009*. Brasília: MCT, 2009.

MIETZNER, R. *A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing*. ph.D. Thesis, July 2010.

NAKAGAWA, Elisa Yumi. *Uma contribuição ao projeto arquitetural de ambientes de Engenharia de Software*. 2006. 250 f. Tese (Doutorado em Ciências da Computação) – Curso de Ciências de Computação e Matemática Computacional, Universidade de São Paulo, São Carlos, 2007.

NISHIMURA, R. T. *Geração de produto em uma abordagem de Linha de Produto para Sistemas Gerenciadores de Workflow*. 2004. 125 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática, Universidade Estadual de Maringá, Maringá, 2004.

NORTHROP, L. M. SEIs Software Product Line Tents. *IEEE Software*, v. 19, n. 14, p. 32-41, 2002.

OLIVEIRA, Paulo H. M. A. *Comparação de sistemas de middleware para grades e clusters computacionais*. São Paulo: Unesp, 2010.

OLIVEIRA, R. *Formalização e verificação de consistência na representação de variabilidades*. Dissertação (Mestrado em Engenharia de Sistemas e Computação) – Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

OLIVEIRA, Raphael Pereira de. *UbiComSPL*: desenvolvimento baseado em MDA, de Linha de Produtos de Software no Domínio de Aplicações Ubíquas. 2009. 78 f. Dissertação (Mestrado em Ciência da Computação) – Curso de Ciência da Computação, Universidade de São Carlos, São Carlos, 2009.

OLIVEIRA, D.; ROSA, N. Ubá: A Software Product Line Architecture for Grid-Oriented Middleware. In: 33rd ANNUAL IEEE INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2009, Informatic Center, Fed. Univ. of Para, Santarem, Brazil.

OLIVEIRA JUNIOR, E. A. *SystEM-PLA*: um método sistemático para avaliação de arquitetura de Linha de Produtos de Software baseada em UML. 2010. 305 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos, 2010a.

_____. Variabilidade em Linha de Produtos de Software: como identificar, delimitar e representar variabilidade nos principais modelos UML de uma linha de produtos de software. *Java Magazine: Engenharia de Software*, Rio de Janeiro, p. 1-15, 2010b.

OLIVEIRA JUNIOR, E. A.; GIMENES, Itana M. S.; MALDONADO, José C. Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science*, p. 1-20, 2010.

OMG. Disponível em: <http://www.omg.org>. Acesso em: 15 jan. 2012.

ORACLE. Disponível em: <http://www.oracle.com/>. Acesso em: 10 dez. 2011.

_____. Disponível em: <http://www.oracle.com/>. Acesso em: 1 jan. 2012.

POHL, K.; BOCKLE, G.; LINDEN, F. J. V. D. *Software Product Line Engineering:* Fundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag, 2005.

REINEHR, Sheila dos Santos. *Reúso sistematizado de software e Linhas de Produto de Software no setor financeiro*: estudo de caso no Brasil. 2008. 327 f. Tese (Doutorado em Engenharia de Produção) – Escola Politécnica, Programa de Pós-Graduação em Engenharia de Produção, São Paulo, 2008.

RUEHL, Stefan T.; ANDERLFINGER, Urs. Applying Software Product Lines to Create Customizable Software-as-a-Service Applications, SPLC '11. In: 15th INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 2011. *Proceedings…* 2011. v. 2.

SEI. Software Engineering Institute. CMMI for Development. Version 1.2: CMMIDEV. USA: SEI, 2010.

_____. *Software Product Line Overview*. Disponível em: <http://www.sei.cmu.edu/productlines/>. Acesso em: 14 dez. 2012.

SILVA, Rodrigo Grumiche. *Seleção de recursos baseado nas características de aplicações em ambientes de grade de multi-agregados*. 2011. 164 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2011.

SILVA, A. P. C.; DANTAS, M. A. R. An Efficient Approach for Resource Set-Matching in Grid Computing Configurations. In: 19th INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD'07), 2007. p. 143-150.

SIMONS, M. et al. Organization Domain Modeling (ODM) Guidebook, version 2.0. *Technical Report*. Lockheed Martin Tactical Defence Systems, 1996. STARS-VC-A025/001/00.

SIMONS, M.; ANTHONY, J. Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering. In: 5th INTERNATIONAL CONFERENCE OF SOFTWARE REUSE (ICSR-5), 1998.

SOCHOS, P.; PHILIPPOW, I.; RIEBISCH, M. *Feature-Oriented Development of Software Product Lines*: Mapping Feature Models to the Architecture. Springer: LNCS 3263, 2004.

SUGUMARAN, V.; PARK, S.; KANG, K. C. Software Product Line Engineering. *Communications of the ACM*, v. 49, n. 12, 2006.

VAN GURP, J.; BOSCH, J. On the Notion of Variability in Software Product Lines. In: THE
WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2001, Amsterdam. *Proceedings…* Amsterdam, 2001.

VASCONCELOS, A. *Uma abordagem de apoio à criação de arquiteturas de referência de domínio baseada na Análise de Sistemas Legados*. Tese (Doutorado em Engenharia de Sistemas e Computação) – Programa de Engenharia de Sistemas e Computação, Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2007.

VICI, A. D.; ARGENTIERI, N. FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. In: FIFTH INTERNATIONAL CONFERENCE ON SOFTWARE REUSE (ICSR5), Apr. 1998. p. 166-175.

VIEIRA, Matheus Anversa. *Uma abordagem para reserva antecipada de recursos em ambientes de grades computacionais móveis*. 2010. 120 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2011.

WEISS, D. M.; LAI, C. T. R. *Software Product-Line Engineering*: a Family-Based Software Development Process. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

YEO, C. S. et al. *Cluster Computing*: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. 2006.

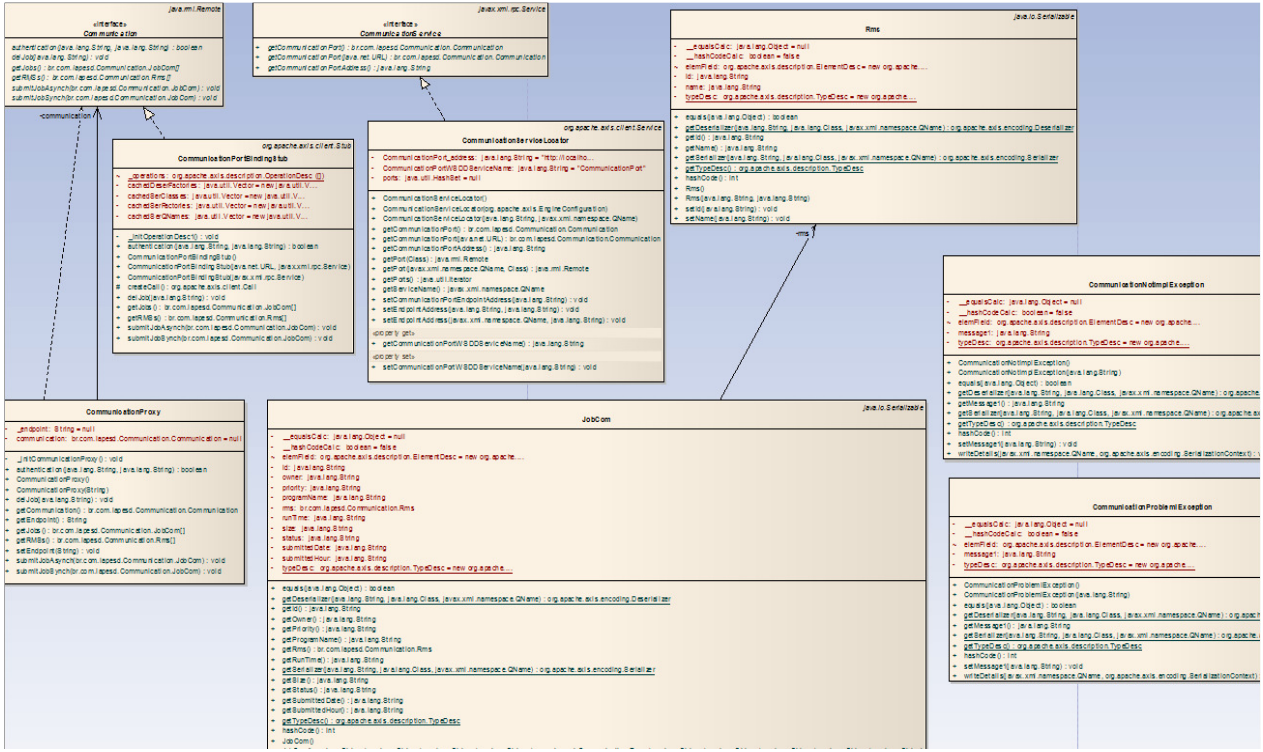
APÊNDICES

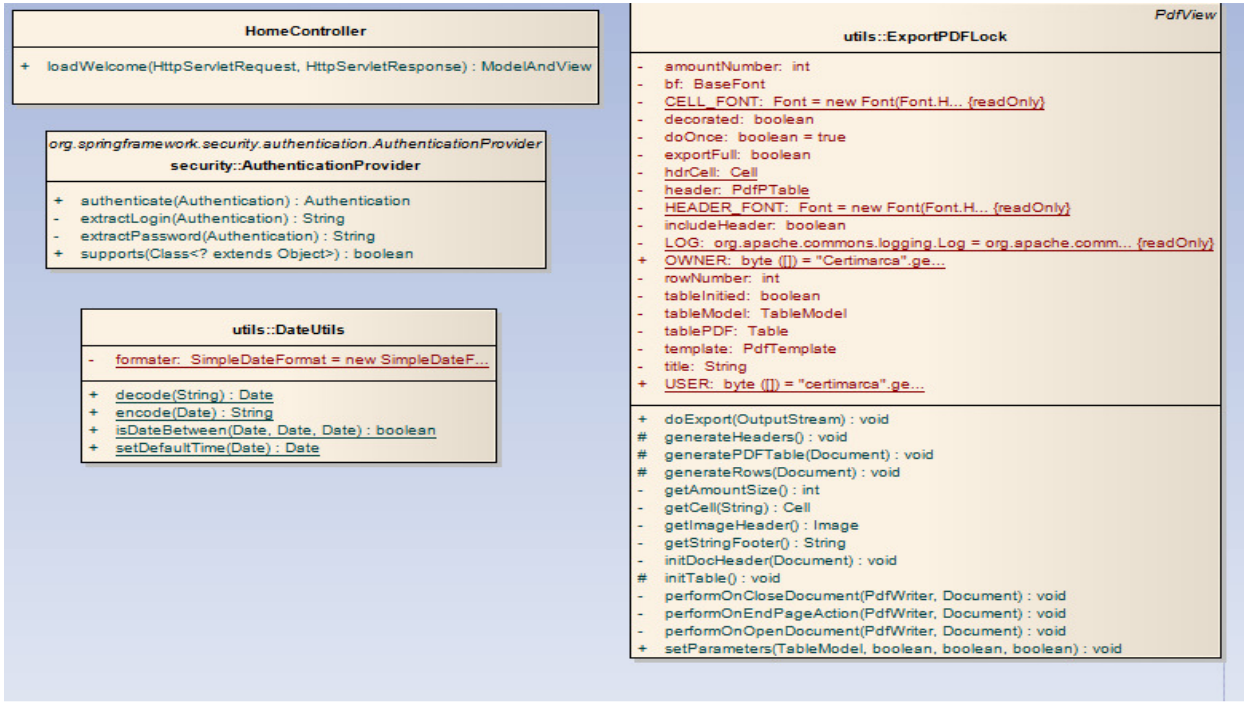
APÊNDICE A – COMPARAÇÃO DE CARACTERÍSTICAS ENTRE LSF, SGE, TORQUE E CONDOR

Características	LSF	SGE	Torque/PBS	Condor
Distribuição	Comercial	Versão comercial: N1 Grid Engine - N1GE versão pública open-source: Grid Engine	Versão comercial: PBS Pro versão pública: Torque	Público e open-source
Ambientes paralelos	PVM, OpenMPI, MPICH	PVM, OpenMPI, MPICH	PVM, OpenMPI, MPICH	Suporta PVM. OpenMPI MPICH com limitações
API	Interna, várias linguagens	DRMAA (C e Java)		DRMAA (C e Java)
Migração de Processo	Sim	Sim	Não	Sim
Balanceamento de Carga Dinâmico	Sim	Sim	Não	Não
Checkpointing	Sim	por meio de bibliotecas externas	Em nível de kernel	Sim
Tolerância a Falhas	Mestre e hosts em execução	Mestre e hosts em execução	Hosts em execução	Hosts em execução
job preemptivo	Sim	Sim	Não	Sim
Jobpreemptivos com multiprocessos	Sim	Sim	Não	Não
job interativo	Sim	Sim	Sim	Sim
job interativo com multiprocessos	Sim	Sim	Não	Não
job preemptivo	Sim	Sim	Não	Sim

Características	LSF	SGE	Torque/PBS	Condor
interativo				
job preemptivo interativo com multiprocessos	Sim	Sim	Não	Não
Documentação	Excelente	Excelente	Boa	Boa

APÊNDICE B – DIAGRAMA DE CLASSES DA APLICAÇÃO WEB E INTERFACE DE COMUNICAÇÃO




HomeController
+ loadWelcome(HttpServletRequest, HttpServletResponse) : ModelAndView
org.springframework.security.authentication.AuthenticationProvider
security::AuthenticationProvider
+ authenticate(Authentication) : Authentication
- extractLogin(Authentication) : String
- extractPassword(Authentication) : String
+ supports(Class<? extends Object>) : boolean
utils::DateUtils
- formater: SimpleDateFormat = new SimpleDateF...
+ decode(String) : Date
+ encode(Date) : String
+ isDateBetween(Date, Date, Date) : boolean
+ setDefaultTime(Date) : Date
PdfView
utils::ExportPDFLock
- amountNumber: int
- bf: BaseFont
- CELL_FONT: Font = new Font(Font.H... {readOnly}
- decorated: boolean
- doOnce: boolean = true
- exportFull: boolean
- hdrCell: Cell
- header: PdfPTable
- HEADER_FONT: Font = new Font(Font.H... {readOnly}
- includeHeader: boolean
- LOG: org.apache.commons.logging.Log = org.apache.comm... {readOnly}
+ OWNER: byte ([]) = "Certimarca".ge...
- rowNumber: int
- tableInitied: boolean
- tableModel: TableModel
- tablePDF: Table
- template: PdfTemplate
- title: String
+ USER: byte ([]) = "certimarca".ge...
+ doExport(OutputStream) : void
# generateHeaders() : void
# generatePDFTable(Document) : void
# generateRows(Document) : void
- getAmountSize() : int
- getCell(String) : Cell
- getImageHeader() : Image
- getStringFooter() : String
- initDocHeader(Document) : void
initTable() : void
- performOnCloseDocument(PdfWriter, Document) : void
- performOnEndPageAction(PdfWriter, Document) : void
- performOnOpenDocument(PdfWriter, Document) : void
+ setParameters(TableModel, boolean, boolean, boolean) : void