UNIVERSIDADE da MADEIRA

APLICAÇÃO DE UMA LINHA DE PRODUTOS DE SOFTWARE (LPS) NO CONTEXTO DE UMA PME

Vitorino Augusto Gonçalves Gouveia

(Licenciado em Engenharia Informática e Computadores pelo Instituto Superior Técnico)

Tese Submetida à Universidade da Madeira para a
Obtenção do Grau de Mestre em Engenharia Informática

Orientador

Professor Doutor António Rito Silva

Professor Associado do Departamento de Engenharia Informática do Instituto Superior Técnico,
Universidade Técnica de Lisboa.

Funchal – Portugal

Setembro de 2007

CITMA
CENTRO DE CIÊNCIA E TECNOLOGIA DA MADEIRA

UNIÃO EUROPEIA
FUNDO SOCIAL EUROPEU

PROGRAMA OPERACIONAL
PLURIFUNDOS DA REGIÃO
AUTÓNOMA DA MADEIRA

APLICAÇÃO DE UMA LINHA DE PRODUTOS DE SOFTWARE (LPS) NO CONTEXTO DE UMA PME

Vitorino Augusto Gonçalves Gouveia

(Licenciado em Engenharia Informática e Computadores pelo Instituto Superior Técnico)

Tese Submetida à Universidade da Madeira para a
Obtenção do Grau de Mestre em Engenharia Informática

Orientador

Professor Doutor António Rito Silva

Professor Associado do Departamento de Engenharia Informática do Instituto Superior Técnico, Universidade Técnica de Lisboa.

Funchal – Portugal

Setembro de 2007

ABSTRACT

A software product line (SPL), is a set of products which shares common *features* and are developed in a systematic way, from a set of software elements. The software development approaches based on SPL, have revolutionized the way that organizations make software engineering. Getting economies of scale, in the conception and distribution of new products, from the reuse of the SPL's software base elements and the instantiation of respective variants, is one of the main benefits in adopting this approach.

In a SPL, the reference software architecture goes besides the dichotomy design/ code of traditional software architectures. Its documentation includes the SPL's software architecture representation and respective variability points, as well as the description of the products instantiation process.

In a small and middle enterprise (SME), the human, technical and financial resources are scarce. The viability of the SPL implementation must be seen in a context of operational costs reduction and efficiency improvement of the software products production process.

This work goal is the development and application of a methodology for SPL implementation and management, which is appropriate to the reality of a SME.

The main contributions of the work done include: a) a methodology for the SPL implementation and management, appropriate to a SMB, which provides the definition of SPL software architecture based on a set of already existent products, b) the representation of SPL software architecture, supported by UML models and extended through an UML Profile, based on a *3+1* architecture perspectives: features, implementation and execution views. The *(+1)* view or "products view" is an instantiation of the other three views, in the particular context of the SPL or a specific product development, at a determined moment in the time.

The proposed methodology was applied to ARQUO ™ solution, a real solution, in production in several organizations.

Key words (in alphabetical order): application engineering, domain engineer, software architecture, software architecture documentation, software configuration management, software product lines, variability management.

RESUMO

Uma linha de produtos de software (LPS), é um conjunto de produtos que partilham funcionalidades comuns, desenvolvidos de forma sistemática a partir de um conjunto de elementos de software base da LPS. As abordagens de desenvolvimento baseado em LPS revolucionaram a forma como as organizações realizam a engenharia de software. A obtenção de economias de escala, na concepção e distribuição de novos produtos, pela reutilização dos elementos de software base da LPS e instanciação dos variantes respectivos, é um dos principais benefícios na adopção desta abordagem.

Numa LPS, a arquitectura de software de referência vai para além da dicotomia desenho/ codificação da arquitectura de software tradicional. A sua documentação, inclui a representação da arquitectura de software da LPS e respectivos pontos de variabilidade, bem como a descrição do processo para instanciação dos produtos.

Numa pequena e média empresa (PME), os recursos humanos, técnicos e financeiros são escassos. A viabilidade da implementação de uma LPS adequa-se num contexto de redução de custos operacionais e eficiência do processo de produção dos produtos de software.

O objectivo deste trabalho é o desenvolvimento e aplicação de uma metodologia para a gestão e implementação de uma LPS, adequada à realidade de uma PME.

As principais contribuições do trabalho incluem: a) uma metodologia para a implementação e gestão de uma LPS adequada a uma PME, que prevê a definição da arquitectura de software da LPS com base num conjunto de produtos já existentes, b) a representação da arquitectura de software de uma LPS, suportado por modelos UML, estendidos através de um perfil UML, baseado em 3+1 perspectivas: dos requisitos, da implementação e dos componentes de execução, sendo que a vista (+1) ou "vista produtos" é uma instanciação das restantes três vistas no contexto particular da LPS ou de um produto, num determinado momento no tempo.

A metodologia proposta foi aplicada à solução ARQUO™, uma solução real e em produção em diversas organizações.

Palavras Chave (por ordem alfabética): arquitectura de software, documentação de uma arquitectura de software, engenharia aplicacional, engenharia de domínio, gestão de configurações de software, gestão de variabilidades, linha de produtos de software.

À Patrícia,

À família da Ribeira e do Janeiro,

Aos meus amigos

AGRADECIMENTOS

Em primeiro lugar agradeço ao Professor Doutor António Rito Silva, orientador do trabalho, pela disponibilidade, e apoio demonstrado ao longo deste percurso. A aprendizagem e troca de experiências foram muito para além do conteúdo desta tese.

Ao Filipe Velosa, director geral da TIE e In-Formar pelo apoio incondicional e por ter incentivado e tornado possível a aplicação prática deste trabalho à solução ARQUO™ da TIE.

Ao António Sargento, director técnico e responsável técnico pela arquitectura de software do ARQUO™, pela colaboração e elevada disponibilidade, na análise e aplicação prática da metodologia à solução ARQUO™. O apoio na revisão do trabalho foi precioso.

Ao Marco Erra e restantes colegas da In-Formar pelo apoio diário e incentivo, mesmo nos momentos de elevado volume de trabalho.

À Universidade da Madeira e Departamento de Matemática e Engenharia, pela eficiência e pelas boas condições disponibilizadas para a realização do trabalho, mesmo para alunos cujo orientador não se encontrava na Região Autónoma da Madeira.

Aos funcionários da secretaria do departamento de Matemática e Engenharia, em especial a Sr.ª Maria Encarnação, pela simpatia e grande eficiência demonstrada sempre que necessário.

Aos colegas de mestrado pelo apoio e companheirismo, durante estes últimos anos em que estivemos juntos por este objectivo, e pela amizade criada, a qual ficará para a vida.

A todos um muito obrigado!

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

ABREVIATURAS E ACRÓNIMOS

ACM – Association for Computing Machinery

ANSI – American National Standards Institute

API – Application Programming Interface

ARQUO™ – Solução de Arquivo Documental Óptico

C&C – Componente e Conector

C4ISR – Command, Control, Comunications, Computers, Intelligence, Surveillance and Reconnaissance

COM – Component Object Model

CORBA – Common Object Request Broker Architecture

CRUISE – Component Reuse In Software Engineering

DBMS – DataBase Management System

DSL – Domain Specific Language

FAST – Family-Oriented Abstraction, Specification, and Translation

FODA – Feature-Oriented Domain Analysis

FORM – Feature-Oriented Reuse Method

GCS – Gestão de Configurações de Software (em inglês Software Configuration Management – SCM)

I&D – Investigação e Desenvolvimento

IBM - International Business Machines Corporation

IDL – Interface Definition Language

IEEE – Institute of Electrical and Electronics Engineers

ISO – International Standardization Organization

LPS – Linha de Produtos de Software (em inglês: SPL – Software Produto Line)

MDA – Model Drives Architecture

OMG – Object Management Group

OO – Orientado a Objectos

ORB – Object Request Broker

PME – Pequena e Média Empresa

PuLSE – Product Line Software Engineering

QADA – Quality-driven Architecture Design and quality Analysis

RiSE – Reuse in Software Engineering Group

RMI – Remote Method Invocation

RM-ODP – ISO Reference Model for Open Distributed Processing

RUP – Rational Unified Process

SEI – Software Engineering Institute

SGBD – Sistema Gestor de Base de Dados

SOAP – Simple Object Access Protocol, ou Service Oriented Architecture Protocol.

SPL – Software Product Line

TIE – Tecnologia de Integração Empresarial, Lda.

UCM – IBM Unified Change Managent

UML – Unified Modeling Language

WBS – Estruturação de Trabalho Padrão (em inglês Work Beakdown Structure)

Win32 – Windows 32 bits

XML – Extended Metamodel Language

XOR – Exclusive "OR"

XSLT – e**X**tensible **S**tylesheet **L**anguage **T**ransformation

I. INTRODUÇÃO

Neste capítulo, é apresentada a contextualização do problema a resolver (ver I.1). Em particular são identificados os objectivos e motivação na base da selecção deste tema como tese no âmbito do mestrado em engenharia informática da Universidade da Madeira realizado entre Outubro de 2005 e Setembro de 2007 (ver I.2).

A definição do contexto específico do problema a endereçar no âmbito da aplicação de uma metodologia de LPS à realidade de uma PME está descrito em I.3.

A organização do trabalho realizado e respectivo enquadramento nos vários capítulos deste documento é apresentado em I.4. A metodologia de trabalho utilizada está descrita em I.5.

I.1. CONTEXTUALIZAÇÃO

A abordagem de desenvolvimento de software através de uma LPS tem hoje uma enorme relevância estratégica para as empresas de desenvolvimento de software. Em particular no contexto da globalização, o colapso das barreiras geográficas e temporais faz com que hoje as empresas, tenham ao seu dispor um mercado global com necessidades semelhantes. Por outro lado, em cada região ou país existem particularidades, e.g. culturais, linguísticas, legais, sociais, que necessitem ser endereçadas de forma particular, aquando do desenvolvimento de software.

A concepção de uma LPS permite a obtenção de economias de escala na distribuição dos produtos de software, pois é utilizada a mesma base de concepção para todos os produtos, sendo apenas necessário instanciar as variabilidades específicas de cada produto, no momento da instalação.

O planeamento, definição, representação e implementação desses pontos de variabilidade nos elementos de software é a chave para o sucesso de uma LPS. O "segredo" está em conseguir aferir no momento da concepção se um determinado elemento de software será base na LPS ou se por outro lado é específico de um projecto em particular e não mais será reutilizado.

Hoje existem várias abordagens para a implementação de LPS, no entanto a sua materialização é deveras complexa e exigem das organizações que as implementam um elevado investimento de recursos humanos e financeiros, face ao tempo necessário para a concretizar.

Numa PME, os recursos financeiros e humanos são escassos. Nesse contexto, as metodologias actuais para a implementação de uma LPS, obrigariam a um investimento elevado, podendo colocar em risco a estabilidade e continuidade da operação da organização.

A existência de uma abordagem "mais leve" para a implementação de uma LPS, que herdasse as boas práticas dos modelos existentes por um lado e que optimizasse o tempo e esforço exigido à gestão e implementação da LPS por outro, teria um enorme sucesso junto das PMEs.

I.2. MOTIVAÇÃO E OBJECTIVOS

A motivação principal na base do desenvolvimento deste trabalho é a criação de uma filosofia de trabalho para o desenvolvimento de software, rapidamente assimilada por todos os elementos da organização e suportada numa metodologia para a concepção, implementação e gestão de uma LPS.

A optimização do esforço necessário à concepção e colocação no mercado de produtos de software flexíveis, reduzindo o investimento e o tempo de implementação dos projectos é um objectivo na base da utilização de LPS.

Este trabalho tem como objectivo o desenvolvimento de uma metodologia para a gestão e implementação de uma LPS, adequada à realidade de uma pequena equipa de trabalho, nomeadamente no contexto de uma PME.

Sempre que aplicável, devem ser utilizados os "*standards*" da indústria e boas práticas existentes nas principais abordagens para a implementação de LPS.

A representação e descrição da arquitectura de software de uma LPS, deve incluir não só a descrição dos elementos de software, como dos mecanismos para instanciação dos produtos resultantes.

O caso prático de aplicação da metodologia proposta será realizada sobre a gama de produtos de software ARQUO™, uma solução de arquivo e gestão documental de engenharia portuguesa, concebida pela empresa TIE – Tecnologias de Integração Empresarial[1], com uma base de produtos instalada em Portugal, em empresas de várias áreas da indústria e com uma estratégia para a internacionalização dos produtos.

A descrição detalhada do contexto do problema o qual motivou a realização deste trabalho é apresentada na secção I.3.

I.3. CONTEXTO DO PROBLEMA

A decisão estratégica para a implementação de uma LPS numa PME tem de ser cuidadosamente analisada e ponderada face ao impacto operacional que a decisão poderá ter na organização. Neste contexto, o resultado de uma má decisão pode por em causa a sustentabilidade do negócio e viabilidade da organização.

A TIE pretende adoptar uma abordagem ao desenvolvimento de software baseada em LPS, para a sua gama de produtos de software ARQUO™. Os produtos ARQUO™ são especializados na captura, arquivo e gestão de documentos electrónicos, servindo organizações de vários sectores de mercado, nomeadamente o financeiro, indústria, telecomunicações e sector da saúde.

O desafio em causa prende-se com a selecção de uma metodologia para a implementação e gestão de uma LPS, adequada à dimensão e necessidades da organização, equipa de desenvolvimento e tecnologias utilizadas, minimizando o impacto operacional na sua implementação.

A Tabela 1 apresenta uma breve caracterização da organização TIE, da sua equipa de desenvolvimento bem como das características pretendidas para a sua LPS.

Características da organização TIE e LPS ARQUO™
PME com menos de 10 elementos na equipa de desenvolvimento.
LPS concebida com base de mais de 5 produtos instalados.
Arquitectura de Software inclui elementos de software desenvolvidos externamente à organização.
Criação da LPS baseada na arquitectura e elementos de software dos produtos ARQUO existentes.
Elevada experiência dos elementos seniores da equipa (+ 5 anos de desenvolvimento), na tecnologia e produtos.
Desenvolvimento dos elementos de software base na LPS baseado em linguagens orientadas a objectos (Java e C++).
Maior foco no desenvolvimento e menos na produção de documentação associada.
Utilização da linguagem UML para a análise e desenho de software.
Elevada reutilização de código nos produtos de software ARQUO™.

Tabela 1 – Enquadramento Organizacional da TIE e LPS ARQUO™

De forma a explicitar as principais necessidades em termos de metodologia para a implementação e gestão da futura LPS, foram identificados os principais requisitos a considerar pelo processo de implementação e gestão da LPS bem como para a representação da arquitectura de software, descritos na Tabela 2.

Ref. Requisito	Descrição	Genérico LPS / Específico TIE.
Aspectos Gerais		
G1	Processo de desenvolvimento "leve", existindo vários níveis de exigência na documentação e comunicação face à criticidade dos elementos de software a desenvolver.	Genérico LPS
G2	Os elementos seniores poderão assumir mais que um papel na implementação da LPS, isto é (i.e.) o responsável pela arquitectura da LPS poderá acumular outro papel na equipa de desenvolvimento.	Genérico LPS
G3	Abordagem ao desenvolvimento baseada em objectos (Java, C++) e componentes distribuídos (via *Common Object Request Broker Architecture* - CORBA , Web Services). A modelação da arquitectura e desenho de software deverá realizar-se em UML.	Específico TIE
Documentação da Arquitectura de Software da LPS		
D1	A representação da arquitectura da LPS deverá satisfazer as necessidades dos vários interlocutores no processo de desenvolvimento dos elementos de software da LPS, nomeadamente a representação numa perspectiva a) requisitos, b) dos módulos de código, c) dos componentes de execução e d) dos elementos de configuração que correspondem à LPS ou produto em particular.	Genérico LPS
D2	O elemento principal de identificação na arquitectura da LPS é o requisito. A arquitectura da LPS deverá conter uma vista de requisitos da LPS, contendo todos os requisitos que esta satisfaz. Deverá ser possível identificar em qualquer elemento da arquitectura, seja um módulo de código ou componente de execução, o(s) requisito(s) que este implementa.	Específico TIE
D3	Contemplar de forma explícita a representação dos pontos de variabilidade e variantes na LPS. Identificar igualmente e de forma explícita os elementos de software que foram desenvolvidos por uma entidade externa à organização.	Genérico LPS
D4	No proposta de modelo a apresentar deverá ser garantida a correspondência entre os artefactos que constituem a arquitectura da LPS, desde a definição dos requisitos aos elementos de software e binários para instalação de um produto.	Genérico LPS
Processo de Implementação e Gestão da LPS		
P1	Abordagem à concepção da LPS contemplar a análise e reengenharia dos elementos de software existentes, i.e. a definição do âmbito da LPS deverá passar por uma análise prévia dos elementos de software existentes e funcionalidades respectivas.	Específico TIE
P2	Contemplar a possibilidade da distribuição dos produtos instanciados da LPS ser realizada externamente à organização, através de uma rede de parceiros.	Genérico LPS
P3	Segregação clara das actividades de engenharia, gestão da LPS e operação e instalação dos produtos instanciados.	Genérico LPS
P4	Actividades de controlo, validação e verificação reduzidas ao essencial, sendo o foco o desenvolvimento e reutilização de elementos de software.	Específico TIE
P5	Foco na gestão de configurações de software, transversal a todos os artefactos da LPS produzidos, por exemplo (e.g.) arquitectura, análise e desenho, elementos de software, ficheiros de configuração, casos de teste.	Genérico LPS
P6	Gestão de configurações multi-produto para a LPS. Capacidade de analisar as configurações dos artefactos no tempo, i.e. a evolução das versões de um componente de configuração ao longo do tempo, e no espaço, i.e. as diferentes versões do mesmo componente de configuração nos produtos da LPS, num determinado momento.	Genérico LPS

Tabela 2 – Requisitos a satisfazer pela Metodologia de Implementação e Gestão da LPS

I.4. ORGANIZAÇÃO DO TRABALHO

Este documento está organizado em sete capítulos principais, os quais apresentam o resultado do trabalho no âmbito da "Aplicação de uma LPS no contexto de uma PME".

No capítulo I. é apresentada uma contextualização do tema da LPS e principais conceitos. Este capítulo, apresenta o trabalho de síntese do "estado da arte", dando especial ênfase à representação da arquitectura da LPS e o processo para a implementação e gestão da LPS. Como extensão do trabalho de síntese é apresentado em apêndice (Apêndice I. , Apêndice II. e Apêndice III.)um resumo comparativo das principais abordagens existentes bem como uma análise da sua adequação ao contexto do problema definido em I.3.

No capítulo III. é apresentada a proposta para uma metodologia de implementação de uma LPS (SPL-LIGHT), adequada à realidade de uma PME. A metodologia proposta enquadra a concepção da LPS para organizações que pretendem reutilizar os produtos de software existentes.

No capítulo IV. é apresentada a proposta para o método de representação da arquitectura de uma LPS (SPL-LIGHT: 3+1), a qual engloba três perspectivas na documentação de uma LPS, nomeadamente a perspectiva funcional, a perspectiva de implementação dos elementos de software, a perspectiva de execução e ligação desses elementos de software no ambiente final. A perspectiva (+1), dos produtos, é transversal às restantes e apresenta a perspectiva da linha de configuração da LPS e produtos instanciados da LPS.

A aplicação prática da metodologia proposta em III. e método para a documentação de uma LPS proposto em IV. , à solução ARQUO™, uma solução real e em produção em diversas organizações, é apresentada no capítulo V.

As principais conclusões do trabalho realizado e próximos passos encontram-se descritos no capítulo VI.

Uma análise detalhada dos aspectos referenciados neste trabalho pode ser obtida através das referências bibliográficas apresentadas em VII.

I.5. METODOLOGIA DE TRABALHO

A descrição dos objectivos e contexto do problema a resolver no âmbito da tese foi o primeiro passo do trabalho realizado (ver I.2 e I.3).

O tema das LPS suscita, neste momento um particular interesse por diversas organizações, tanto da área de investigação como da indústria de software. A quantidade de material e referências sobre o tema é vasto. Nesse sentido, foi realizado uma análise do trabalho existente ao contexto do problema.

Resultante do trabalho de levantamento realizado, foram identificadas por um lado, um conjunto de boas práticas adequadas que endereçavam partes do contexto do problema e por outro, um conjunto de limitações nas abordagens existentes para o satisfazer.

A definição da metodologia proposta para satisfazer o contexto do problema, embebe algumas dessas boas práticas e normas da indústria.

A definição da metodologia proposta e aplicação prática à solução ARQUO, foi realizada de forma iterativa, permitindo a sua validação e adequação de forma progressiva.

II. IMPLEMENTAÇÃO E GESTÃO DE UMA LINHA DE PRODUTOS DE SOFTWARE

Neste capítulo, são apresentados os principais conceitos e trabalho de síntese ao "estado da arte" no contexto da investigação e aplicação prática dos conceitos base na implementação e gestão de uma Linha de Produtos de Software (LPS).

No trabalho de síntese foram tomados em consideração os itens identificados no contexto do problema (ver I.3), a serem satisfeitos pela proposta de metodologia a apresentar e modelo para a documentação da arquitectura de LPS.

A abordagem proposta utiliza os conceitos aqui apresentados, pelo que a sua percepção ajudará a melhor contextualizar e identificar os aspectos inovadores do trabalho proposto.

II.1. CONCEITOS

II.1.1.Arquitectura de Software

"*A arquitectura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, as quais incluem os elementos de software, as propriedades externamente visíveis desses elementos e as relações entre eles Por "propriedades externamente visíveis" referimo-nos aquelas assumpções que outros elementos podem fazer de um elemento, tais como os serviços providenciados, características de performance, tolerância a faltas, utilização de recursos partilhados.*"[2].

A engenharia de software "tradicional" tem como base a construção de um elemento de software ou um produto, no prazo definido. Um elemento de software é um bloco de construção da arquitectura de software ("*bluiding block*") que possui uma implementação interna, interfaces para ligação com outros elementos de software bem como documentação associada.

A arquitectura de software preocupa-se com a definição dos elementos de software e do seu comportamento, na perspectiva da relação com outros elementos e não tanto com os detalhes da implementação de cada elemento. Neste contexto, se o comportamento de um determinado elemento influenciar a implementação de outros elementos que com ele pretendem relacionar-se, o seu comportamento deve ser parte integrante da arquitectura.

A arquitectura de um sistema de software pode ser observada de diversas perspectivas (ou vistas) [3], segundo as necessidades dos interlocutores na engenharia de software, e.g. funcionalidades, implementação, execução, instalação. Alguns exemplos dessas abordagens (analisadas em detalhe na secção II.2.1 e descritas em Apêndice I.) são: as "quatro mais uma" vistas de Krutchen [4], presente no *Rational Unified Process* (RUP)[5, 6], as quatro vistas apresentadas pela *Siemens Corporate Reseach* [7], as cinco vistas da RM-ODP (*The ISO Reference Model for Open Distributed Processing*) [8], as três vistas da C4ISR (*Command, Control, Comunications, Computers, Intelligence, Surveillance and Reconnaissance*) [9], as quatro vistas arquitecturais para o desenvolvimento baseado em componentes de Herzum & Sims [10] e a framework de Zackman [11].

Associado ao conceito de elemento de software é importante apresentar outros conceitos associados, no contexto da documentação das várias perspectivas da

arquitectura de software (implementação e execução). A semântica dos conceitos é idêntica à apresentada por Clements et. al. em [3]:

- Módulo: Um módulo é a implementação do código fonte de um elemento de software (ou parte dele), das suas funcionalidades e comportamentos, descritos pelos requisitos funcionais, satisfazendo as decisões de desenho necessárias para satisfazer os atributos de qualidade definidos para o sistema, e.g. uma classe ou pacote (*package*) contendo várias classes;
- Componente: Um componente é uma unidade de processamento e armazenamento de dados que um sistema executa. São tipos de componentes: servidor, cliente, filtro, objecto, base de dados. Os componentes expõem interfaces para o exterior, de forma a poderem ser utilizados ou invocados por outros componentes, essas interfaces são descritos através de portos. Um elemento de software poderá conter "n" componentes;
- Conector: Um conector é uma ligação para a partilha de dados e controlo entre componentes, e.g. chamadas remotas de procedimentos entre um cliente e um servidor, ou entre dois objectos (numa abordagem Orientada a Objectos – OO); mensagens assíncronas, divulgação de mensagens mediante subscrição. Os conectores possuem uma ou mais papeis que descrevem a forma como os componentes deverão estabelecer a comunicação através do conector.

Durante a definição da arquitectura devem ser envolvidos todos os intervenientes principais, desde o cliente da solução, utilizadores de negócio, arquitecto de software, responsáveis pelas equipas de desenvolvimento e gestor de projecto. Em conjunto, deverão definir os requisitos para a arquitectura: a) requisitos de comportamento, b) atributos de qualidade (e.g. desempenho, disponibilidade, funcionalidade, usabilidade, testabilidade, segurança), c) interfaces com outros sistemas, d) objectivos de negócio, na base da definição da arquitectura e f) origem dos elementos de software da arquitectura (desenvolvimento, reutilização, aquisição a entidades externas).

Existem várias abordagens para a definição de uma arquitectura de software, as quais deverão ser enquadradas no contexto aplicacional existente. São exemplos destas abordagens:

- Abordagens topo-para-detalhe: Iniciam-se com o levantamento de requisitos até à modelação da arquitectura. Vantagem: Completude e mapeamento dos requisitos. Desvantagem: demora em obter um sistema, mesmo para teste pelos utilizadores.
- Abordagens detalhe-para-topo: Utilizadas quando a arquitectura é suportada por elementos de software existentes. Neste cenário há lugar à reengenharia de software e de requisitos, como primeiro passo para a concepção da arquitectura.

- Infra-estrutura primeiro: É desenhado em primeiro lugar a infra-estrutura tecnológica e de ligação dos componentes de execução, com base nos atributos de qualidade e depois o desenho das funcionalidades.
- Funcionalidades primeiro: Para sistemas onde os atributos de qualidade não são críticos e o arquitecto conhece a infra-estrutura de suporte / necessita de obter requisitos funcionais o mais rápido possível.

A arquitectura de software determina a estrutura do produto e condiciona o projecto de desenvolvimento bem como do sistema resultante, i.e. uma boa definição da arquitectura é chave para o sucesso do sistema resultante, uma má arquitectura compromete decisivamente todo o processo de engenharia subsequente.

II.1.2. Reutilização de Software

"*Reutilização de software é o processo de criação de sistemas de software a partir de software existente em vez de o desenvolver de raiz*" [12]

A reutilização de software [13, 14] na concepção de produtos com características semelhantes, a um menor custo, é um factor de vantagem competitiva na engenharia de software.

São exemplos de reutilização de software: compiladores, bibliotecas de classes (em desenvolvimento OO), padrões de desenho, sistemas distribuídos (e.g. COM, CORBA), interfaces gráficas com utilizador, *frameworks* de desenvolvimento, sistemas de gestão de bases de dados (SGDB) (em inglês: *Database Management Systems* – DBMS).

O desenvolvimento de software para promover a reutilização é contudo mais oneroso em termos de esforço dias/homem e prazo de implementação, pois existem um conjunto de princípios e regras a respeitar pela equipa de desenvolvimento. Neste sentido, Bosh [15] apresenta uma abordagem para reutilização de software em que os elementos de software a reutilizar são planeados a um nível arquitectural e desenvolvidos com um grau de abstracção adequado.

A reutilização de software depende da existência de processos sistemáticos de desenvolvimento e suporte de ferramentas adequado. Esta tese é defendida em *Component Reuse In Software Engineering* (C.R.U.I.S.E.), produzido pelo *Reuse in Software Engineering (RiSE) Group*. [16].

A reutilização de software é uma das motivações para a utilização de uma LPS, esta ideia é defendida em Bosh [17], CAFÉ [18], ESAPS [19], PuLSE [20]. Cada elemento de software base de uma LPS é reutilizado tantas vezes, quantas as instâncias de produtos forem concretizadas.

II.1.3.Variabilidade

"*A variabilidade é a capacidade de alterar ou adaptar um sistema. Aumentar a variabilidade de um sistema implica torná-lo mais fácil a certos tipos de alterações.*" [21]

A gestão de variabilidades no desenvolvimento de software sempre existiu, embora aplicada a cada produto em particular. A implementação das variabilidades (desde a análise, desenho e desenvolvimento) era realizada produto a produto, replicando todo o esforço necessário.

A introdução de mecanismos de variabilidade no desenvolvimento de software tem como principal objectivo permitir a reutilização dos elementos de software, abstraindo as variações que estes possam conter em pontos concretos da arquitectura, face às necessidades específicas de cada projecto ou implementação.

Os autores Gurp, Bosch and Svahnberg [21], defendem que as variabiliades podem ser inicialmente identificadas, com base no conceito de *feature* (i.e. unidade lógica de comportamento que corresponde a um conjunto de requisitos funcionais e de qualidade). É igualmente apresentado um conjunto de mecanismos para a inclusão de variabilidades na arquitectura, ao longo do processo de desenvolvimento e instalação dos produtos.

Com a abordagem de LPS as variabilidades são modeladas na arquitectura de referência da LPS (através da representação dos *Pontos de Variabilidade* e *Variantes* respectivos) e resolvidas antes da instalação dos produtos.

- Um *Ponto de Variabilidade* corresponde a um aspecto de variação funcional num elemento de software base. O ponto de variabilidade define o conjunto de possíveis variantes, o mecanismo de variabilidade a utilizar para os instanciar e o tempo de activação dos variantes, e.g. instanciação da arquitectura de software do produto, tempo de compilação, execução.
- Um *Variante* corresponde a uma opção do conjunto de possíveis instâncias de variação que um ponto de variabilidade poderá originar.

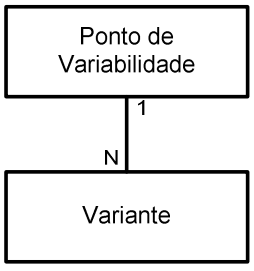


Figura 1 – Conceitos na variabilidade

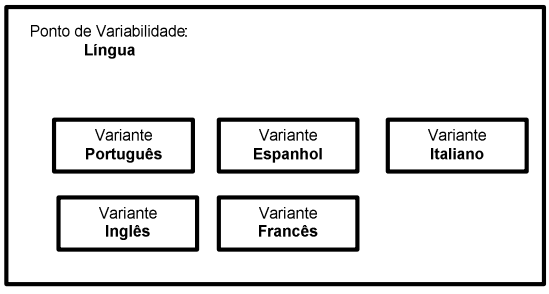


Figura 2 – Exemplo de ponto de variabilidade e variantes

A Figura 1 e Figura 2 ilustram a relação entre estes conceitos, bem como um exemplo prático de utilização, em que para o ponto de variabilidade "Língua" estão implementados cinco variantes.

De forma a podermos identificar variabilidades no software devemos utilizar as seguintes questões:

- Onde variam os vários produtos do mesmo software? A resposta a esta questão permite identificar pontos de variabilidade.
- Como varia? Ou seja qual o conjunto de variantes detectados para um ponto de variabilidade.
- Porque é que varia? A resposta permite aferir das razões para a variação e identificar o variante que melhor a satisfaz.

A gestão de variabilidades aplicada as LPS, é uma área de investigação em crescendo, no entanto, existem já publicadas algumas técnicas, procedimentos e ferramentas. Algumas abordagens utilizam linguagens específicas de domínio [22, 23] na concepção e desenho de LPS. Outras abordagens utilizam a linguagem UML e adicionam extensões específicas (Perfis UML, ver II.2.3 para mais detalhe) para a representação das variabilidades, quer para os aspectos estruturais da arquitectura [24, 25], de comportamento[24-27] ou modelos de requisitos [28, 29].

II.1.4.Linha de Produtos de Software

"*Uma linha de produtos de software é um conjunto de sistemas de software, que partilham um conjunto de funcionalidades comuns e que satisfazem as necessidades de um segmento de mercado particular ou missão e que são desenvolvidos de forma sistemática, a partir de um conjunto de activos base.*" [30]

O conceito de Linha de Produtos de Software (LPS) (em inglês: *Software Product Line* - SPL) está intrinsecamente ligado às áreas de arquitectura, engenharia e reutilização de software.

As abordagens baseadas em LPS tiveram início no final da década de 90 [31], face à necessidade de colmatar algumas das falhas das abordagens existentes (nomeadamente baseadas em engenharia de domínio, ver II.1.5). Segundo Bayer et. al. [20] este tipo de abordagens tinha como principais limitações: a) excluía a área de engenharia aplicacional, b) falta de suporte operacional e c) demasiado enfoque nos aspectos organizacionais.

O desenvolvimento baseado em LPS veio revolucionar a forma como as organizações, grandes ou pequenas, concebem e executam as tarefas de engenharia de software. A produção e reutilização de elementos de software base, os quais poderão incorporar variantes é específico do desenvolvimento de software suportado por uma LPS.

- Um elemento de software base da LPS é um elemento de software resultado da engenharia de domínio da LPS, que foi desenvolvido de forma a poder ser reutilizado.

Um elemento de software base da LPS poderá ser "obrigatório", "opcional" ou "externo" (i.e. desenvolvido por uma entidade externa) no contexto da instanciação em novos produtos. Um elemento de software base do tipo "obrigatório" ou "opcional" poderá conter um ou mais pontos de variabilidade, contendo variantes.

Numa LPS, existem duas perspectivas de engenharia de software: engenharia de domínio e engenharia aplicacional.

- A engenharia de domínio foca o desenvolvimento dos elementos de software base reutilizáveis da LPS. O objectivo da construção destes elementos de software base é permitir reutilizá-los na produção de produtos da LPS. Neste sentido, devem ser igualmente produzidos os "plano de produção" para a LPS. Um plano de produção descreve a forma como podem ser instanciados os produtos a partir dos elementos de software base. A gestão de variabilidades (pontos de variabilidade e variantes) nos elementos de software base é realizada durante a engenharia de domínio.
- A engenharia aplicacional tem como objectivo a geração de novas aplicações (ou produtos da LPS) utilizando os elementos de software base desenvolvidos na engenharia de domínio e os "plano de produção" respectivos. Na engenharia aplicacional, o enquadramento dos requisitos do projecto no âmbito da LPS é crítico, pois poderão existir requisitos específicos que estejam fora do âmbito da LPS o que implica que os elementos de software sejam desenvolvidos "à medida". Os elementos de software resultantes do desenvolvimento específico são classificados como elementos de software "aplicacionais" e não são considerados elementos de software base da LPS.

Esta segregação é defendida pelas abordagens SEI Framework [32], KobrA [33], FAST [23] e FORM [34]. O mesmo não ocorre comas abordagens para a reutilização de software baseado exclusivamente em engenharia de domínio, como FODA [35].

Cada LPS possui uma arquitectura de software própria (Arquitectura de Software da LPS). A arquitectura de software de uma LPS (descrita em II.1.5) permite a instanciação de implementações dos elementos de software base da LPS em organizações com realidades específicas. As variabilidades dos produtos são obtidas pela instanciação dos pontos de variabilidade, definidos nos elementos de software base da LPS.

No contexto da reutilização de software, a utilização de pontos de variabilidade e variantes é um factor diferenciador das abordagens baseadas em LPS, evitando desta forma a duplicação de código bem como do esforço de implementação e manutenção

associado. Durante a instanciação de um produto específico da LPS, para cada ponto de variabilidade é realizada a selecção dos variantes que melhor satisfazem os requisitos específicos do produto.

II.1.5.Modelos de Domínio e Funcionalidade

"*Um modelo de domínio é uma representação explícita das propriedades comuns e variáveis de um sistema, a semântica das propriedades e dos conceitos de domínio e as dependências entre as propriedades variáveis*" [36].

Como descrito na secção II.1.2, são vários os benefícios na reutilização de software. Para tal, no desenvolvimento de software orientado à reutilização é importante que: a) todos os elementos da equipa de desenvolvimento partilhem dos mesmos conceitos e semântica do domínio a resolver pelo sistema e b) existam restrições que limitem o desenvolvimento de elementos de software a reutilizar, sob o risco de serem construídos elementos de software reutilizáveis, mas sem utilidade futura. É nesse contexto que surge a necessidade de limitar o domínio de abrangência do software, através da definição de modelos de domínio, incluindo os conceitos, propriedades e âmbito do domínio.

Estes modelos suportam a análise e tomada de decisão em relação ao desenvolvimento de novas aplicações que se enquadrem dentro do âmbito do domínio. A definição dos requisitos da LPS e o modelo de domínio constituem o âmbito da LPS.

- Um requisito da LPS é "*uma unidade lógica de comportamento que é especificada por um conjunto de funcionalidades e requisitos de qualidade*" [17]. Um requisito satisfeito pela LPS é um conjunto de características funcionais ou não funcionais que são suportadas por um ou mais elementos de software da arquitectura da LPS.

Na definição do domínio, Kang et al. [35] utiliza um modelo de decomposição de *features* em árvore, onde as *features* genéricas são colocados no topo e as detalhadas em baixo. A utilização de modelos hierárquicos de *features* pode ser enquadrada nos seguintes contextos:

- modelação de vastos domínios,
- modelação de variabilidades em LPS,
- encapsulamento dos requisitos do sistema,
- contextualização e orientação no desenvolvimento da LPS,
- planeamento e suporte à evolução da LPS e
- comunicação entre os vários intervenientes no desenvolvimento do sistema.

Os métodos FODA [35], FORM [34] e Jansen et al.[37], utilizam os modelos de *features* para suportar a análise de requisitos e definição do modelo de domínio.

A representação do modelo de domínio pode ser realizado utilizando uma linguagem padrão, e.g. UML ou uma Linguagem Específica do Domínio (*Domain Specific Language - DSL*) [23]. São várias as abordagens que utilizam DSL na representação e automatização do processo de instanciação dos produtos e respectivas variabilidades numa LPS, e.g. *Family-Oriented Abstraction, Specification, and Translation* (FAST) [23].

II.1.6. Arquitectura de Software de uma LPS

"*Uma arquitectura de uma linha de produtos de software é uma arquitectura de software que satisfaz as necessidades da linha de produto em geral e os produtos individuais em particular, explicitamente admitindo um conjunto de pontos de variabilidade necessários para suportar o espectro dos produtos no âmbito da linha de produtos. Contém a arquitectura dos produtos e as especificações das interfaces para os componentes que constituirão os activos nucleares.*" [30]

Numa LPS, a arquitectura de software de referência vai para além da dicotomia desenho / codificação da arquitectura de software tradicional. A arquitectura de uma LPS permite a instanciação de várias implementações seguindo um conjunto de pontos de variabilidade definidos explicitamente.

Na implementação dos pontos de variabilidades e variantes poderão ser utilizados diferentes padrões e mecanismos de inclusão de variabilidade, desde a definição da arquitectura à compilação e ligação dos elementos de software (ver II.2.4).

Em sistemas orientados a objectos (OO), a implementação de pontos de variabilidade e respectivos variantes pode realizar-se pela especialização de classes particulares a partir de uma classe abstracta. As classes "variantes" devem ser reescritas, nos pontos de variabilidade, para os variados produtos de forma a satisfazer a funcionalidade específica. Neste contexto, a inclusão de variantes num ponto de variabilidade pode ser realizada de forma incremental. Herança e delegação são técnicas OO que podem ser utilizadas para permitir as variações. Em Jansen et al. [37] é apresentado um protótipo para a instanciação dos pontos de variabilidade de uma LPS utilizando uma técnica de composição funcional, suportada na linguagem Java.

A documentação e implementação dos atributos de qualidade da arquitectura, tais como desempenho, disponibilidade, fiabilidade, usabilidade e testabilidade na arquitectura de uma LPS é idêntica à arquitectura de software tradicional. Neste contexto, L. Zhu e I. Gorton [38] apresentam um perfil UML para a representação das decisões de desenho e cenários de atributos de qualidade da arquitectura baseado no *framework* para representação dos cenários de atributos de qualidade definido por L. Bass, P. Clements e R. Kazman (SEI) [2].

A documentação da arquitectura de uma LPS, inclui a descrição da arquitectura da LPS e respectivos pontos de variabilidade bem como a descrição do processo para

instanciação dos produtos. Ao longo do tempo, vários autores e organizações desenvolveram abordagens para descrever a arquitectura de um sistema de software. Mas estarão estas abordagens preparadas para ser utilizadas no contexto da implementação e gestão de uma LPS? Na secção II.1.1 é apresentada uma análise destas abordagens face ao contexto do problema definido em I.3.

Numa LPS é fulcral existir uma definição do processo de instanciação dos produtos a partir dos elementos base. Os processos de integração do produto, instalação e testes devem estar igualmente definidos. Em Clements e Northrop [30] estes procedimentos encontram-se descritos no "plano de produção", um documento nuclear da arquitectura, onde está descrito o processo de instanciação de produtos a partir dos elementos base da LPS, bem como das métricas e controlos para a validação e verificação do processo de produção.

II.1.7. Desenvolvimento de Software Baseado em LPS

A adopção de uma LPS por organizações que concebem produtos de software não é um processo imediato. Esta exige uma alteração substancial na cultura da equipa de desenvolvimento, e das práticas de desenho da arquitectura, desenvolvimento e instalação dos produtos de software. São várias as abordagens existentes para a implementação de uma LPS [20, 23, 30, 32, 39, 40], descritas em maior detalhe em Apêndice III. Não foi objectivo neste trabalho a realização de uma análise exaustiva das várias abordagens existentes.

Hoje, são vários os casos de estudo de organizações que adoptaram LPS para o desenvolvimento dos seus produtos, tendo obtido com isso um aumento da produtividade e redução do tempo de resposta e evolução dos produtos face a imperativos de negócio ou solicitações do mercado. Um exemplo de aplicações diversas de LPS na industria é apresentado em [32].

Mas será que qualquer organização que desenvolva software beneficia com a adopção de uma LPS? A resposta é não. De facto, nem todo o desenvolvimento de software se enquadra no conceito de LPS. São vários os exemplos de situações que de forma isolada não se enquadram numa lógica de LPS:

- Reutilização fortuita de pequenos elementos de software;
- Reutilização no desenvolvimento de um sistema isolado;
- Desenvolvimentos baseados em elementos de software, os quais não permitem variação;
- Existência de uma sequência de versões de um produto unitário;
- Existência de normas ou constrangimentos para o desenvolvimento.

II.1.8.Implementação e Gestão de uma LPS

A tomada de decisão em relação à implementação de uma LPS numa organização exige uma avaliação prévia da capacidade da sua organização para a adoptar, bem como do contexto da sua aplicação.

À questão: "O que é necessário para a minha organização adoptar uma LPS?" deve ser precedida de uma outra questão: "Faz sentido a minha organização adoptar uma LPS?". De facto, nem todas as organizações que concebem produtos de software terão vantagens na adopção de uma LPS.

Das principais abordagens para a implementação e gestão de uma LPS, algumas foram concebidas para uma implementação de raiz da LPS, tal como a *Framework* para Implementação LPS, desenvolvida pelo *Software Engineering Institute* (SEI) [30, 41] e *Family-Oriented Abstraction, Specification, and Translation* (FAST) [23].

A abordagem *Product Line Software Engineering* (PuLSE) [20] prevê a incorporação de elementos de software de arquitecturas já existentes na implementação da LPS. A abordagem KobrA [33], desenvolvida no *Fraunhofer Institute for Experimental Software Engineering* (IESE) é uma especialização do PuLSE para o desenvolvimento baseado em componentes, com base numa abordagem OO, sendo a linguagem de modelação o UML.

Como descrito em II.1.5, algumas metodologias para a implementação e gestão de LPS utilizam os modelos de *features* para suportar a análise de requisitos e modelo de domínio. São exemplos os métodos FODA [35], FORM [34] e Jansen et al.[37].

II.1.9.Gestão de Configurações de Software numa LPS

Na engenharia de software, as alterações aos elementos de software ocorridas ao longo do tempo são controladas através de um processo de gestão de configurações de software, sendo a gestão de conflitos no desenvolvimento em paralelo sobre o mesmo artefacto um dos principais problemas a resolver pela equipa de desenvolvimento. Uma análise mais aprofundada das actividades de engenharia de software e gestão de configurações pode ser obtida em Pressman [42] e SEI [43].

Numa LPS, para além deste aspecto é importante gerir as diferentes instâncias de variantes para os mesmos pontos de variabilidade, contida nos produtos da LPS.

A gestão de configurações numa LPS pode ser enquadrada como sendo multi-dimensional, pois entra em linha de conta com a variação: a) no tempo (corresponde à gestão de configurações da LPS consoante a variação no tempo) e b) no espaço (corresponde à gestão de variações entre produtos da LPS num particular momento no tempo).

Numa LPS, durante a engenharia de domínio poderão ocorrer variações ao nível dos elementos base da arquitectura de referência, com impacto em todos os produtos dessa LPS. Em simultâneo e durante a engenharia aplicacional, ocorrem desenvolvimentos específicos em cada produto.

A identificação, representação e gestão dos conflitos numa LPS é complexa, mas decisiva na eficiência e eficácia da gestão e operação da LPS, pois cada alteração ao nível de um elemento de software base da LPS tem influência nas configurações de cada um dos produtos existentes, nomeadamente em termos de actualizações futuras.

Como gerir as configurações, de forma que, em cada momento seja possível a obter as configurações de um determinado elemento base pelos vários produtos, bem como o seu historial de alterações ao longo do tempo?

No contexto da gestão de configurações numa LPS são importantes os seguintes conceitos:

- Versão. Sempre que as alterações a um artefacto sob controlo de configurações sejam "submetidas", é criada uma nova versão do artefacto contendo o estado actual. O histórico das versões anteriores é mantido e poderá ser recuperado.
- Elemento de configuração. Um elemento de configuração é um artefacto sob gestão de versões. São exemplos de artefactos, os modelos de desenho, código fonte, ficheiros de configuração, documentação da arquitectura e plano de produção.
- Componente de configuração. Um componente de configuração é um agrupamento de elementos de configuração, que representa um elemento básico de construção de um produto de configuração. Um componente possui um conjunto de *baselines*.
- *Baseline*. Um *baseline* é um conjunto de versões de elementos de configuração, num determinado instante no tempo, realizados sob uma linha de configuração.
- Produto de configuração. Um produto de configuração é um conjunto de componentes de configuração que integrados formam uma solução de software, que satisfaz o âmbito e requisitos definidos num projecto.
- *Release*. Uma *release* é um conjunto de *baselines*, sob a qual foram realizados testes de integração e de sistema, garantindo-se estabilidade da solução como um todo.
- Linha de Configuração. Uma linha de configuração (*branch* em inglês) é um repositório físico de elementos de configuração. Durante a engenharia de domínio e engenharia aplicacional poderão ser criados várias linhas de configuração para segregar as diferentes configurações dos elementos de configuração do ambiente de trabalho dos diferentes intervenientes e nos diferentes ambientes, e.g. integração, qualidade e produção. Tanto a LPS como os projectos de engenharia aplicacional deverão possuir uma linha de configuração, denominada "linha de configuração principal".

II.2. ESTADO DA ARTE

II.2.1. Abordagens para a Documentação da Arquitectura de Software

"*Documentar uma arquitectura, consiste na documentação das vistas relevantes, e de seguida adicionar documentação que se aplique a mais do que uma vista*." [3]

A documentação da arquitecturas de software necessita de ser adequada às necessidades particulares dos interlocutores no processo de desenvolvimento e utilização que estes lhe vão dar, seja o responsável pelo projecto, gestor projecto, gestor LPS, arquitecto de software, programador, elemento da equipa de testes ou responsável pela instalação. Este princípio é consensual e está definido nas práticas ANSI/IEEE 1471/2000 [44], para a documentação de arquitecturas de sistemas.

As principais abordagens para a documentação de arquitecturas de software utilizam o conceito de vistas arquitecturais. A arquitectura de um sistema de software pode ser observada sob diversas perspectivas (ou vistas), cada qual representando aspectos específicos do sistema segundo um determinado prisma.

O conjunto de vistas e estilos arquitecturais corresponde assim à documentação da arquitectura de software. O número e tipo de vistas a incluir na documentação varia segundo a abordagem seguida. O reconhecimento de que as arquitecturas devem conter o número de vistas que sejam úteis para o sistema em questão, qualquer que seja esse número. É o princípio fundamental da aproximação "para além das vistas" – (*views and beyond)* de Clements et al [3].

Na Tabela 3, está apresentada uma comparação de um conjunto de abordagens, descritas em detalhe em Apêndice I.

As abordagens analisadas, apresentam diferentes perspectivas para a documentação de arquitecturas de software. A utilização de uma abordagem em detrimento de outra é uma decisão que pode ser tomada, dependendo da cultura e dimensão da equipa de desenvolvimento, tecnologia utilizada e enquadramento no processo de desenvolvimento existente.

Não obstante as diferentes perspectivas apresentadas, existe correspondência entre as vistas dos diferentes modelos, na cobertura dos vários aspectos alvo de documentação numa arquitectura de software. A Tabela 3 ilustra essa correspondência.

Cobertura Arquitectural	**"4+1 vistas Kruchten" / RUP**	**Siemens**	**ISO RM-ODP**	**Paul Clements et al.**	**Framework Zachman**
Funcionalidades	Vista Lógica	Conceptual	Organizacional	Diagramas de Contexto	Âmbito Modelo Organizacional
Requisitos Qualidade	Vista Desenvolvimento			Documentação para além das vistas: Rational	
Estruturas Informação			Informação		Coluna "Dados"
Módulos de Código Fonte	Vista Desenvolvimento	Código	Computacional	Módulo	Modelo Tecnológico Representação Detalhada
Componentes e Interfaces	Vista Processos	Módulo	Engenharia	Componente & Conector	Modelo Sistema Modelo Tecnológico
Instalação	Vista Instalação	Execução	Tecnologia	Afectação	Representação Detalhada
Afectação Pessoas	Vista Desenvolvimento			Afectação	Coluna "Pessoas"
Variabilidades				Documentação para além das vistas: Variabilidades.	

Tabela 3 – Correspondência entre as abordagens para a documentação de arquitecturas de software

Foi realizada uma análise comparativa das abordagens: "4+1 vistas Kruchten" / RUP[4, 5], Siemens [7], ISO RM-ODP[8], Paul Clements et al. [3] e Framework Zachman[45] face às necessidades descritas no contexto do problema (ver I.3), para uma arquitectura de software de uma LPS. Da análise realizada, cujo detalhe é apresentado em Apêndice I.6, verifica-se o seguinte:

- Nenhuma das abordagens representa a perspectiva de uma LPS, nomeadamente a distinção entre os elementos base da LPS e específicos de cada produto;
- A representação da arquitectura da LPS de forma a satisfazer as necessidades dos vários interlocutores é realizada nas abordagens "4+1 vistas Kruchten" [4] / RUP [6] e Paul Clements et al. [3]. Por outro lado, o modelo RM-ODP[8] não identifica os interlocutores alvos de cada uma das vistas que define e o Siemens [7] foca no desenho da arquitectura, segundo o ponto de vista do arquitecto e não tanto na documentação para a comunicação da arquitectura a outros intervenientes;

- Em relação à representação dos requisitos, a abordagens "4+1 vistas Kruchten" [4] / RUP apresenta a vista casos de utilização (vista "+1") para a identificação das funcionalidades e requisitos. Em ISO RM-ODP[8] a vista Organizacional utiliza casos de utilização, e categorias para representar e classificar as funcionalidades do sistema. Em Paul Clements et al. [3], não é apresentada uma vista para a representação dos requisitos, no entanto o modelo é flexível e poderão ser utilizados os diagramas de contexto (os quais possuem uma utilização abrangente) apresentados como anexos às restantes vistas;
- Apenas a abordagem de Paul Clements et al. [3], aborda a representação de variabilidade, permitindo descrever a variabilidade e dinamismo nas vistas e estilos da arquitectura de software. Os diagramas de contexto anexos às vistas podem identificar diferentes variantes;
- Em ISO RM-ODP[8], é dado foco à reutilização dos elementos de software da arquitectura;
- A *Framework* de Zachman [45] é aplicável à documentação de um sistema de informação como um todo, i.e. incluindo as pessoas, infra-estrutura tecnológica de suporte e arquitectura de software. O custo da criação e manutenção da documentação (a *framework* possui trinta células), pode ser superior ao benefício que daí advém;
- Em relação à necessária correspondência entre os artefactos que constituem a arquitectura de software, desde a definição dos requisitos aos elementos de software e binários para instalação de um produto, a abordagem "4+1 vistas Kruchten" [4] / RUP[6] apresenta um conjunto de regras e heurísticas para a ligação entre elementos de vistas diferentes. A implementação RUP [6] da IBM o processo UCM (*Unified Change Management*)[46] permite a gestão integradas de alterações e configurações nos artefactos da arquitectura de software. Siemens inclui fluxos de informação entre as vistas (estruturas), suportando a correspondência dos elementos da vista conceptual com os elementos das restantes vistas da arquitectura de software. Em, Paul Clements et al. [3] os diagramas de contexto anexos às vistas podem representar a relação entre elementos.

II.2.2. Utilização de UML na Documentação de Arquitecturas de Software numa LPS

O *Unified Modeling Language* (UML), lançado em 1995 e adoptado pelo *Object Management Grou*p (OMG) [47] em 1997. Desde então tem vindo a ser adoptado e estendido pela indústria e hoje é a linguagem de modelação de software mais utilizada.

O UML é um standard para a análise e desenho de arquitecturas de software, nomeadamente as orientadas a objectos (OO). A linguagem é utilizada para

representar as várias perspectivas do sistema (por exemplo: casos de utilização, classes, sequência, colaboração, instalação), para os vários intervenientes no processo de desenvolvimento, quer sejam utilizadores de negócio, analistas, programadores, ou membros da equipa de instalação e suporte.

Em simultâneo com a publicação das especificações para o UML 2.0 [48-51], o OMG lançou a iniciativa *Model Drives Architecture* (MDA) [52]. O MDA define uma *framework* conceptual para uma abordagem ao desenvolvimento de software baseado em modelos utilizando linguagens padrão da indústria. Este facto teve relevância na definição da superstrutura UML 2.0 pois o MDA utiliza as linguagens de modelação muito para além da documentação ou desenho de software de "alto-nível". Com o MDA é dado ênfase à capacidade de geração automática de software com base nos modelos e possibilidade de análise da qualidade e validade do desenho.

Em particular no contexto da modelação de arquitecturas para uma LPS, a versão UML 2.0 inclui algumas melhorias que permitem aproximar a linguagem das necessidades de modelação da arquitectura de software de uma LPS:

- Aumento da precisão na descrição semântica dos conceitos base. A semântica e sintaxe dos conceitos de propriedade, espaço de nomes (*namespaces*), classificador (*classifier)* foram redefinidas de forma separada. Este facto agiliza e permite a inclusão de extensões aos conceitos, importantes no contexto de uma LPS.
- Melhoria na capacidade para modelar e descrever arquitecturas de sistemas de software. Em particular o UML 2.0 é possível descrever o comportamento da estrutura interna da classe bem como das interfaces que esta expõe a instâncias de outras classes, através da identificação dos seus elementos: parte, conector e porta. A representação da arquitectura para a reutilização de software é um dos objectivos principais de uma LPS, a qual é possível pela capacidade de conhecer/invocar os pontos de conexão das classes, sem conhecer a sua estrutura interna.
- Melhorias na capacidade de modelação de processos de negócio e aplicações orientadas às funcionalidades. Numa LPS a identificação das funcionalidades é nuclear, não só na definição do âmbito da LPS, como na referência aos módulos de software que a implementam.
- Notação para expressar padrões
- Inclusão na descrição de um componente UML, a identificação da sua interface e porta(s). Importante é a inclusão de portos e pontos de interacção nos componentes UML e regras e pontos de interacção com os conectores na metalinguagem UML 2.0. A inclusão do conceito de conector na ligação dos componentes, adaptando o protocolo e formato da mensagem de um componente UML para outro não existia até à versão 2.0.

- O metamodelo UML 2.0 apresenta ao nível dos diagramas de sequência um conjunto de novos operadores de interacção tais como: alternativa (*alt*), sequência (*seq*) e ciclo (*loop*) que permitem a representação de uma alternativa, sequência e repetição de interacções. A metaclasse interacção (*interaction*) refere-se à unidade de comportamento baseada na troca de informação entre objectos no diagrama de sequência. O elemento interacção permite referenciar no contexto de um diagrama de sequência um conjunto de interacções associadas, bem como diferentes comportamentos ou alternativos, consoante a lógica da interacção em causa.

II.2.3. Representação de variabilidades com Perfis UML

Um perfil UML (UML *profile*) é um conjunto de extensões ao meta-modelo UML, composto pelos estereótipos, atributos especiais (*tagged values)* e constrangimentos necessários à representação de um determinado conceito, por exemplo a representação de variabilidades nas LPS.

No contexto da representação das variabilidades numa LPS através de um perfil UML, são vários os contributos existentes. os vários autores apresentam diferentes abordagens para a representação de variabilidades usando perfis UML.

- Ao nível da representação dos requisitos (*features*) da arquitectura da LPS, Clauß [28] introduz uma extensão UML 1.4 para suportar os diagramas de *features* e adiciona elementos que descrevem a variabilidade nos diagramas UML standard. A abordagem apresentada estende o modelo *Feature-Oriented Reuse Method* (FORM) [34] extensão do método *Feature-Oriented Domain Analysis* (FODA) [35], na medida em que além dos três tipos de *features*: «optional», «alternative» e «mandatory" é adicionado o tipo «*external*», que representa as características ou requisitos externos ao sistema. Incluído um modelo UML de características (*feature model*) que é utilizado para representar os requisitos comuns e variáveis numa LPS. Utilizado um modelo UML de pacotes para agrupar os casos de utilização que pertencem à mesma *feature*.
- Gomaa, em [53], inclui um modelo UML de requisitos (*feature model*) que é utilizado para representar os requisitos comuns e variáveis numa LPS. Neste contexto é utilizado um modelo UML de pacotes para agrupar os casos de utilização que pertencem à mesma *feature* e um conjunto de extensões UML para representar as variabilidades ao nível das LPS, nomeadamente os estereótipos «*common feature*», «*optional feature*», «*alternative feature*»,«*default feature*» e «*parameterized feature*». A especificação e representação do comportamento de cada requisito (*features*) em casos de utilização UML, e relação entre os pontos de variabilidade das funcionalidades e a sua representação através das relações de extensão e inclusão dos casos de utilização é um dos aspectos inovadores da abordagem.

- A representação de variabilidades da LPS ao nível dos diagramas de classes UML é apresentada por Gomaa [53], no entanto no contexto da representação do modelo de domínio da LPS e incluem os seguintes estereótipos: «*kernel*», «*optional*», «*variant*» e «*alternative*». Por outro lado, Em Ziadi et al [24], é apresentada uma extensão aos diagramas de classes UML 2.0 para representar os módulos de software da LPS, através da inclusão dos estereótipos: «*optional*» (funcionalidade opcional), «*variation*» (para representar os pontos de variabilidade) e «*variant*» (para representar os variantes).
- A representação de variabilidades ao nível do comportamento dos componentes e conectores é proposta por Ziadi et al [24] através de extensões aos diagramas de sequência: «*optionallifeline*» e «*optionalintereaction*», «virtual» (indica que a interacção é virtual), «*variation*» e «*variant*».

II.2.4. Padrões e mecanismos para gestão de variabilidades numa LPS

Contrariamente ao que ocorre na engenharia de software tradicional, onde qualquer alteração do produto implica usualmente alteração de código fonte, o desenvolvimento de produtos suportados por uma LPS partilha de um conjunto de elementos base da arquitectura, sendo que as novas funcionalidades ou requisitos de qualidade são adicionadas em fases posteriores do processo de desenvolvimento, muitas vezes após a compilação do produto.

Neste contexto, a existência de uma definição de mecanismos de desenho para a inclusão de variabilidades numa LPS, que possam ser instanciadas nos desenvolvimentos baseados em LPS tem uma grande importância quer para as equipas de desenvolvimento, quer para o processo de desenvolvimento pois passa a existir um *framework* com padrões comuns para a inclusão de variabilidades numa LPS, em situações definidas.

O trabalho desenvolvido por Jan Bosch et al. [21], resumido nesta secção, apresenta um conjunto de padrões que permitem a inclusão de variabilidades numa LPS ao longo das várias fases de desenvolvimento, segundo um conjunto de mecanismos definidos.

Os mecanismos de inclusão de variabilidades objectivam a criação de um ou mais variantes em diferentes tempos de construção do sistema, desde a definição de requisitos à execução. Neste processo podem ser abstraídos alguns padrões recorrentes no que se refere à introdução, gestão e definição dos limites de variabilidades.

A Tabela 4, identifica um conjunto de mecanismos para a implementação de variabilidade, identificados por classe de padrão (Entidades Variantes – EV, Entidades

Opcionais – EO e Entidades Múltiplas Coexistentes – EMC), bem como identifica as etapas onde são implementadas e activadas.

Tipo Padrão			Mecanismo de Variabilidade	Fase em que a Variabilidade toma lugar	Fase em que a Variabilidade é adicionada
EV	EO	EMC			
X			Reorganização da Arquitectura	Derivação da Arquitectura do Produto	Desenho da Arquitectura
X			Inclusão de pontos de variabilidade e variantes nos Elementos de Software da Arquitectura		Desenho da Arquitectura Desenho Detalhado
	X		Elemento de Software Opcional da Arquitectura		
X			Especialização dos Elemento de Software com Pontos de Variabilidade.		Desenho Detalhado
	X		Especialização dos Elemento de Software Opcionais		
x	X		Condição em Constante	Compilação	Implementação
X			Imposição de fragmentação de código		Compilação
X			Substituição de binários – Directivas de ligação	Ligação	Ligação
X			Substituição de binários – Física		
X			Arquitectura Centrada na Infra-estrutura	Tempo de Execução	Desenho da Arquitectura
		X	Implementação dos Elemento de Software Múltiplos coexistentes		Desenho Detalhado
		X	Especialização dos Elemento de Software Múltiplos coexistentes		
X	X	X	Condição em Variável		Implementação

Tabela 4 – Mecanismos para a implementação de variabilidade.

O detalhe dos aspectos a relevar de cada um dos mecanismos de variabilidade, contendo a motivação, descrição da solução e consequências da aplicação e exemplos cujo detalhe pode ser obtido em [21].

II.2.5. Da gestão de configurações à gestão de variabilidades numa LPS

A abordagem de Krueger [54] para a gestão de variabilidades em LPS utiliza como base o processo de gestão de configurações para a engenharia de software e adiciona procedimentos e técnicas para a cobrir os aspectos da gestão de variabilidades em LPS. A técnica utilizada é a de "dividir para conquistar" sendo que a gestão de variabilidades é dividida em nove partes, cada qual correspondendo uma célula de uma tabela:

Tipo Variação Artefactos	Tempo Sequencial	Tempo em Paralelo	Espaço do Domínio
Ficheiros	Gestão versões	Gestão ramos	Gestão pontos variabilidade
Componentes	Gestão *baselines*	Gestão *baselines* pelos ramos	Gestão de adaptações
Produtos	Gestão composição	Gestão composição pelos ramos	Gestão composta de adaptações

Tabela 5 – Técnica de "dividir para conquistar" par a a gestão de variabilidades, adaptado de [54]

Os tipos de variação: tempo sequencial e tempo em paralelo de ficheiros e componentes de configuração são usualmente abrangidos pelas ferramentas de gestão de configurações se software básicas.

A abordagem propõe a composição de ficheiros em componentes de configuração para colmatar as dificuldades na gestão de variações no tempo ao nível dos produtos da LPS e adaptação em massa de software para os pontos de variabilidade no espaço do domínio.

Esta abordagem "*detalhe-para-topo*", aplica-se essencialmente na concepção de LPS com base em produtos já existentes. No entanto, esta abordagem não foca a formalização e definição dos modelos de representação das variabilidades ou na relação entre funcionalidades e artefactos (ficheiros, componentes de configuração, produtos) na LPS.

II.2.6. Abordagens para a Implementação e Gestão de uma LPS

Existem vários autores e instituições que têm dedicado especial interesse em desenvolver o tema da implementação de LPS nas organizações, no entanto a sua aplicação ao contexto de uma PME, não é abordada. Nesta secção são apresentados alguns dos modelos de referência, bem como uma análise destes face à cobertura das necessidades definidas no contexto do problema (ver I.3), para o processo de implementação e gestão de uma LPS.

Para as várias abordagens enumeradas em II.1.7, descritas em detalhe em Apêndice III.1, é apresentada uma análise dos aspectos que correspondem aos requisitos definidos em I.3 ao nível do processo de implementação e gestão da LPS (P1 a P6) bem como os aspectos particulares em que as abordagens não satisfazem o contexto do problema (ver Tabela 6).

Abordagem	Aspectos que satisfazem requisitos	Aspectos não satisfazem requisitos
SEI *Framework*	A abordagem do SEI, é muito abrangente na perspectiva do processo de implementação e gestão de uma LPS, dando ênfase às actividades de engenharia, gestão técnica e organizacional, incluindo a melhoria contínua do processo ao longo do tempo	Orientação da abordagem à concepção de LPS de raiz. Práticas de gestão organizacional, são dificilmente adaptáveis à realidade das PMEs onde os recursos humanos estão essencialmente focados nas actividades operacionais e onde o grau de formalismo nos processos está reduzido ao essencial.
Family-Oriented Abstraction, Specification, and Translation (FAST)	A abordagem faz uma diferenciação das actividades de definição de requisitos (domínio do problema) da implementação respectiva (domínio da solução).	Utiliza uma linguagem específica de domínio (DSL), para a geração dos produtos numa LPS. Orientação da abordagem à concepção de LPS de raiz.
Product Line Software Engineering (PuLSE)	Permite contemplar na actividade de concepção da arquitectura de referência da LPS elementos de software de produtos já existentes.	Esta abordagem para a implementação de uma LPS é genérica, não focando em particular o desenvolvimento baseado em objectos.
Feature-Oriented Domain Analysis (FODA)	O método FODA objectiva a identificação das funcionalidades (i.e. aspectos, qualidades ou características do sistema) durante a análise de domínio. São utilizados modelo hierárquicos em árvore de funcionalidades para visualizar as variabilidades e dependências de funcionalidades.	Abordagem não foca práticas e técnicas para o desenho e implementação do modelo de funcionalidades na arquitectura de referência e elementos de software da LPS.
Feature-Oriented Reuse Method (FORM)	A abordagem FORM apresenta dois processos de engenharia para as LPS: engenharia de domínio e engenharia aplicacional. O FORM utiliza no processo de engenharia alguns princípios importantes na concepção de LPS: separação de conceitos e encapsulamento de informação, configuração dos artefactos da arquitectura de software consoante as diferentes funcionalidades, separação entre componentes e conectores.	O processo de correspondência dos requisitos particulares em elementos de software da arquitectura não é descrito explicitamente pelo método FORM. Apresenta uma relação unária entre as funcionalidades e os módulos de software da arquitectura de software, não sendo adequado ao desenvolvimento baseado em objectos onde a relação entre as funcionalidades e os módulos que as implementam são n-árias.

Tabela 6 – Satisfação dos requisitos pelas abordagens para a implementação e gestão de LPS

II.3. CONSIDERAÇÕES FINAIS

O tema do desenvolvimento baseado em Linhas de Produtos de Software é hoje base de investigação em diversas instituições e organizações ligadas à engenharia e arquitecturas de software.

De facto, os benefícios pela adopção de desenvolvimento suportado em LPS são imensos, nomeadamente no aumento de eficiência e rentabilidade da produção de software. Por outro lado, as dificuldades que se colocam às organizações que pretendem adoptar esta abordagem são reais e é essencial, de entre as várias possíveis abordagens e metodologias a selecção da que melhor se enquadra na realidade e cultura da organização.

O trabalho de síntese apresentado nesta secção deu especial ênfase a duas das mais críticas áreas na gestão e implementação de uma LPS: a representação da arquitectura da LPS e o processo para a implementação e gestão da LPS.

No contexto particular de uma PME, onde os recursos humanos, técnicos e financeiros são escassos, o processo de implementação e gestão de uma LPS deve ser o mais "leve" possível, focando-se nas actividades essenciais de análise dos produtos existentes, gestão, engenharia dos elementos de software e distribuição dos produtos da LPS.

Nos capítulos seguintes, é apresentada uma proposta para o processo de implementação de uma LPS adequado à realidade de uma PME (III.) e uma proposta para a representação da arquitectura da LPS (IV.) tendo como base a definição da arquitectura da LPS, com recurso à reengenharia dos elementos de software de produtos já existentes e gerindo as variabilidades e configurações ao nível da engenharia de domínio e engenharia aplicacional. As propostas apresentadas embebem as boas práticas das referências sintetizadas neste capítulo.

O caso prático apresentado no capítulo V. permite obter um exemplo da implementação do trabalho proposto, numa LPS real, em produção em diversas organizações.

Uma análise detalhada dos aspectos sintetizados neste capítulo pode ser obtida através das referências bibliográficas apresentadas em VII.

III. PROPOSTA DE METODOLOGIA PARA IMPLEMENTAÇÃO E GESTÃO DE UMA LPS

Nesta secção é apresentada uma proposta de metodologia para a implementação e gestão de uma LPS, adequando-se ao contexto de uma PME.

Numa PME, os recursos humanos, técnicos e financeiros são escassos. A viabilidade da implementação de uma abordagem de desenvolvimento baseado em LPS numa PME justifica-se num contexto de redução de custos operacionais e eficiência do processo de produção dos produtos de software.

Nesse contexto, a adopção de uma LPS por uma empresa de desenvolvimento com produtos já existentes, pode ser enquadrada como uma medida estratégica para a optimização do processo de desenvolvimento e consolidação da experiência acumulada ao longo do tempo.

Não sendo um requisito imprescindível para a adopção da metodologia proposta, esta poderá ser mais assertiva em organizações que possuem alguma maturidade no processo de engenharia aplicacional no âmbito da LPS pretendida.

A metodologia proposta nesta secção pretende servir de veículo para a implementação e gestão de uma LPS em organizações que já possuam produtos desenvolvidos e instalados em diversos clientes, e pretendam utilizá-los como base para definir uma LPS.

III.1. CONTEXTO DA IMPLEMENTAÇÃO DE UMA LPS NUMA PME

Numa pequena ou média empresa (PME), ao contrário das grandes empresas ou multinacionais, as limitações orçamentais, de estrutura orgânica e recursos disponíveis estão sempre presentes. Neste contexto, a implementação de uma linha de produtos de software numa PME, dificilmente ocorrerá numa abordagem de desenvolvimento da LPS "de raiz", face ao esforço operacional, comercial e financeiro necessário e rápido retorno no investimento pretendido.

Em organizações que se dedicam à engenharia aplicacional, com produtos instalados em diversas organizações, é usual existir um elevado esforço de manutenção dos produtos instalados face a solicitações dos clientes ou evolução do produto. Neste contexto, nas PMEs é usual ocorrer um problema comum: a equipa de engenharia dedica demasiado tempo à manutenção correctiva e evolutiva em cada um dos produtos instalados (duplicando esforço em cada produto), em detrimento do tempo necessário para fazer evoluir as funcionalidades do produto e torná-lo comercialmente mais atractivo. Neste contexto, a abordagem proposta permitirá obter benefícios a curto prazo.

Na adopção de uma LPS as organizações de uma forma geral enfrentam os seguintes desafios, os quais para as PMEs poderão por em causa a viabilidade da organização:

- Investimento em I&D necessário à implementação da LPS, não deve comprometer as necessidades de operação diária da organização e orçamento financeiro;
- Gestão adequada da relação com os clientes actuais no sentido de não criar falsas expectativas em relação à evolução dos produtos instalados (poderão conter elementos de software não considerados no âmbito da LPS);
- Gerir as expectativas e reais necessidades de novos clientes, em especial após a apresentação de um protótipo contendo todas as funcionalidades da LPS;
- Afectação dos recursos humanos à reengenharia dos elementos de software existentes de forma a enquadrá-los na arquitectura de software da LPS;
- Redução do esforço de manutenção associado aos produtos da LPS e criação de um plano de migração da base instalada para as versões no âmbito da LPS;
- Optimizar o tempo de desenvolvimento, mesmo contando com o facto de existirem constrangimentos arquitecturais que a LPS assim o obriga;

- Alterações que impliquem modificações a elementos base são complexas e usualmente devem ser bem pensadas, sob o risco de comprometer a LPS. Os requisitos específicos dos clientes que não se enquadrem no âmbito da LPS devem ser tratados como tal e desenvolvidos "à medida";
- Uma LPS endereça flexibilidade, no entanto limitada. Nem todas as alterações solicitadas pelos clientes podem ser satisfeitas pelos variantes existentes. É necessário um elevado nível de conhecimento e experiência no desenvolvimento dos elementos base, de forma a definir pontos de variabilidade que enderecem as questões de forma adequada.

A implementação de uma LPS exige da equipa afecta uma maior responsabilização, nomeadamente nos seguintes aspectos chave:

- Produção de documentação completa e elevada qualidade;
- Necessidade de amadurecer o desenvolvimento antes da 1ª entrega (se possível);
- Elevada comunicação entre os intervenientes da equipa e gestão do projecto;
- Possuir um sistema de registo dos erros e correcções partilhado por todos os intervenientes;
- Documentar e satisfazer os requisitos específicos das plataformas e sistemas operativos onde as instâncias dos produtos irão ser instaladas.

Numa PME, dada a usual reduzida dimensão da equipa, a utilização de ferramentas de suporte é fundamental, diminuído o esforço e "burocracia" usual nos processos de documentação da arquitectura, gestão de alterações e testes.

Não existe uma receita para o sucesso da implementação de uma LPS numa PME, pois são vários os factores externos que influem nesse objectivo, não obstante, como referido em [30] existem algumas boas práticas que poderão contribuir positivamente:

- Existir um responsável pela LPS, com a visão e objectivo de incutir a consciencialização para a LPS nos restantes elementos.
- Pressão para evitar enveredar pelo desenvolvimento tradicional.
- Vasta experiência e conhecimento nos domínios chave da aplicação.
- Foco na arquitectura
- Envolvimento da administração da organização
- Disciplina na implementação e instanciação dos produtos

III.2. PROCESSO DE IMPLEMENTAÇÃO E GESTÃO DE UMA LPS: SPL-LIGHT

III.2.1. Estrutura Orgânica e Papeis

Numa PME, os recursos e competências encontram-se usualmente dispersos por poucas pessoas. Este facto constitui um risco operacional, pela dependência da organização num conjunto reduzido de pessoas chave. Técnicas para a criação de bases de conhecimento são hoje muito frequentes para registar, partilhar e optimizar o conhecimento dentro das organizações. A utilização de *wikis*, bases de dados de conhecimento e soluções de gestão documental são exemplos destas técnicas que visam a diminuição do risco de dependência do conhecimento mantido nos recursos humanos.

Nestas organizações, tão (ou mais) importantes como as áreas administrativas, financeiras e comerciais, a área de implementação e gestão da LPS é essencial. De facto, enquanto que a gestão financeira e administrativa pode ser subcontratada a uma empresa externa, a estrutura comercial pode incluir parcerias com empresas externas distribuidoras dos produtos, a gestão e implementação da LPS é nuclear na organização e não pode/deve ser delegada em terceiros.

A Figura 3 ilustra uma possível representação orgânica para os vários papéis necessários para a gestão e implementação de uma LPS numa organização.

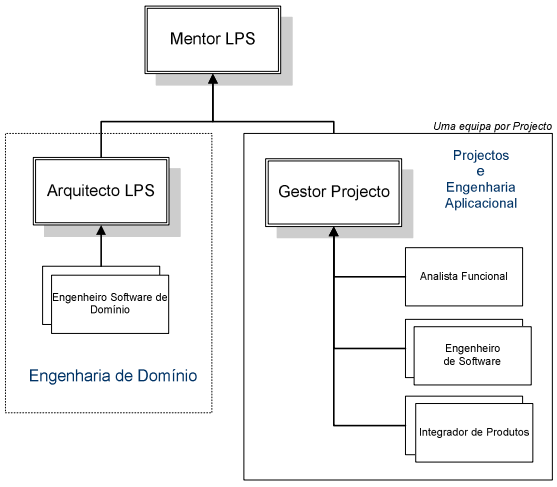


Figura 3 – Estrutura Orgânica para a Gestão e Implementação de uma LPS numa PME

Na definição da estrutura organizacional, é importante a segregação entre as duas equipas de desenvolvimento: Engenharia de Domínio, ou seja, dos elementos base da LPS e Engenharia Aplicacional e Projectos, ou seja, a instanciação e implementação dos produtos da LPS em clientes.

A Tabela 7, apresenta um conjunto indispensável de papéis que a equipa de gestão e implementação da LPS deverá possuir. No contexto de uma PME, é muito usual alguns dos papeis serem acumulados pelos recursos seniores, pelo que a coluna "Possível Acumulação de Papeis" demonstra possíveis conjugações para a acumulação de papeis, garantindo uma segregação mínima de funções entre o desenvolvimento de domínio e o desenvolvimento aplicacional.

Papel Principal	**Responsabilidades**	**Possível Acumulação de Papeis**
Mentor da LPS	Definição do plano de evolução (*roadmap*) para a LPS Decisão de integração de produtos externos na LPS Estratégia de marketing e oferta a clientes	Arquitecto da LPS Analista Funcional
Arquitecto da LPS	Concepção de uma arquitectura refactorizada com base nos produtos existentes Definição da Arquitectura da LPS Enquadramento de produtos existentes na arquitectura LPS Enquadramento de novos desenvolvimentos na LPS Documentação da Arquitectura da LPS Definição e optimização das ferramentas de suporte à gestão e operação da LPS.	Engenheiro(s) de Software de Domínio
Engenheiro(s) de Software de Domínio	Refactorização de elementos de software existentes. Desenvolvimento e alteração dos elementos bases. Implementação dos pontos de variabilidade da LPS.	Analista Funcional Engenheiro(s) de Software Aplicacional
Gestor(es) de Projecto	Responsável pela gestão e controlo dos projectos em curso. Responsável pela afectação dos recursos às tarefas do projecto.	Analista Funcional
Analista Funcional	Levantamento dos requisitos do cliente Enquadramento dos requisitos na LPS	
Engenheiro(s) de Software Aplicacional	Desenvolvimento de software específico no âmbito de um projecto.	Integrador de Produtos
Integrador de Produtos	Responsável pela instalação e configuração dos produtos finais nas instalações dos clientes.	Engenheiro(s) de Software Aplicacional

Tabela 7 – Papéis necessários numa equipa de gestão e implementação de uma LPS

Em algumas organizações, a actividade de instanciação e implementação dos produtos da LPS nos clientes, bem como dos desenvolvimentos específicos necessários é delegada a uma entidade externa ou parceiro de negócio. Esta actividade

por não ser nuclear para a evolução da LPS pode não justificar a criação de uma estrutura comercial e de desenvolvimento para a distribuição dos produtos pelo mercado alvo. Esta estratégia é utilizada pelas grandes multinacionais de desenvolvimento de software com base numa LPS, são exemplos a SAP [55], Oracle [56], mas também por empresas mais pequenas com enfoque no desenvolvimento de uma LPS como a TIE – Tecnologias de Integração Empresarial [57]e a LPS ARQUO™.

III.2.2.O processo SPL-LIGHT

O processo de implementação e gestão de uma LPS, SPL-LIGHT (SPL do inglês *Software Product Line* e *Light* por ambicionar a definição de um processo "leve"), foi concebido tomando em consideração os seguintes princípios:

- Adequação do processo à dimensão de uma pequena equipa de desenvolvimento
- Capacidade de iniciar a implementação da LPS a partir de um conjunto de produtos ou elementos de software já existentes.
- Separação clara entre o desenvolvimento dos elementos base da LPS e os elementos específicos de cada instalação (engenharia de domínio v.s. engenharia aplicacional).
- Rastreabilidade entre a definição de requisitos e a implementação dos elementos de software

O processo SPL-LIGHT, ilustrado pela Figura 4, é constituído por quatro áreas principais, as quais estão completamente interligadas no âmbito de uma LPS: Análise de Produtos Existentes, Gestão, Engenharia e Operação e Gestão de Produtos.

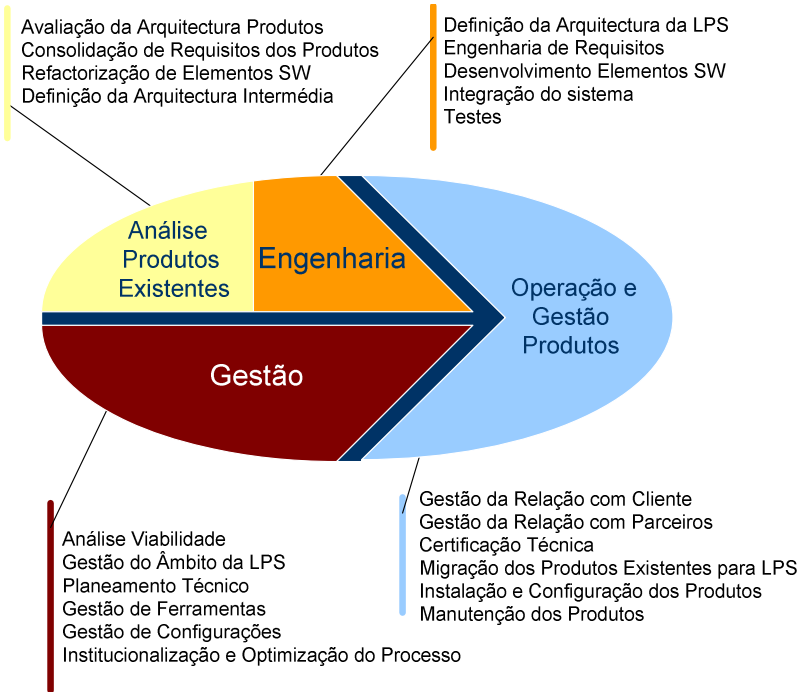


Figura 4 – Áreas do Processo SPL-LIGHT

Em particular esta abordagem adequa-se a qualquer organização que objective a inclusão de elementos de software dos produtos existentes como base para a definição da arquitectura de software da LPS.

III.2.3.SPL-LIGHT: Análise de Produtos Existentes

A decisão de avançar para a concepção de uma LPS é na grande maioria das organizações precedida de experiência prévia na área de negócio alvo da LPS. Esta experiência, poderá já estar consolidada caso a empresa de desenvolvimento de software possua um produto ou vários produtos de software já desenvolvidos e instalados em clientes.

Por outro lado, a adopção de uma LPS por uma empresa de desenvolvimento com produtos já existentes, pode ser enquadrada na necessidade de optimização do processo de desenvolvimento e consolidação da experiência acumulada ao longo do tempo.

A área de actividades "Análise de Produtos Existentes" da metodologia SPL-LIGHT visa a identificação dos elementos comuns entre os vários produtos existentes, que possam ser reutilizados na concepção da LPS.

A utilização de técnicas de refactorização dos elementos de software e da arquitectura de software, permite a transformação incremental da estrutura dos elementos de software, sem afectar a funcionalidade exposta e comportamento nos produtos já instalados.

Esta categoria tem igualmente como objectivo a consolidação de requisitos (funcionais e atributos de qualidade) dos produtos existentes para a definição do âmbito da LPS, com base na experiência obtida no passado.

A Tabela 8 resume as actividades da categoria "Análise de Produtos Existentes".

Actividade	Tarefas Principais	Descrição
Avaliação da Arquitectura dos Produtos	Análise da arquitectura técnica dos produtos existentes. Análise dos elementos de software comuns e específicos. Identificação de padrões de arquitectura e desenho comuns. Avaliação do esforço de refactorização dos elementos de software versus reengenharia.	Mentor da LPS Arquitecto da LPS
Consolidação de Requisitos dos Produtos	Definição da matriz de requisitos transversal aos vários produtos existentes. Identificação dos requisitos comuns e específicos para cada produto. Abstracção e generalização na definição dos requisitos. Definição do modelo de domínio consolidado.	Mentor da LPS Arquitecto da LPS Analista Funcional

Actividade	Tarefas Principais	Descrição
Refactorização de Elementos de Software	Identificação de dependências e interfaces entre elementos de software. Avaliar a reabilitação de elementos de software das arquitecturas existentes para a arquitectura da LPS. Realizar a refactorização dos elementos de software elegíveis para a arquitectura de software intermédia (resultado da refactorização dos produtos existentes).	Mentor da LPS Arquitecto da LPS Integrador de Produtos
Definição da Arquitectura Intermédia	Refinamento "*topo-para-detalhe*" do modelo de requisitos da arquitectura. Definição de abstracções para elementos de software genéricos e candidatos à inclusão de pontos de variabilidade. Consolidação dos cenários de atributos de qualidade e decisões de desenho dos vários produtos. Refactorização "*detalhe-para-topo*" dos elementos de software e enquadramento no modelo de requisitos da arquitectura. Definição do plano de migração dos produtos existentes para a arquitectura intermédia.	Mentor da LPS Arquitecto da LPS

Tabela 8 – Actividades da categoria "Análise de Produtos Existentes"

III.2.4.SPL-LIGHT: Gestão

A categoria "Gestão" engloba todas as actividades necessárias para suportar o planeamento, gestão e evolução dos elementos de software base da LPS e dos produtos resultantes, bem como das ferramentas e técnicas de suporte.

Numa fase inicial de adopção da LPS, com base na análise de produtos existentes ou não, é realizada uma análise de viabilidade na adopção da LPS pela organização e adequação da estrutura orgânica para a gestão e implementação da LPS.

A construção do modelo de domínio contendo os conceitos e âmbito da LPS é importante pois caso contrário, a organização ficará muito dependente e vulnerável à saída dos seus técnicos mais experientes (o que usualmente tem um elevado impacto nas PMEs).

Numa LPS são extremamente importantes as actividades de gestão de configurações dos artefactos da LPS, a instanciação respectiva nos produtos, a gestão de alterações aos elementos base e a geração e instalação dos produtos resultantes.

Em organizações que possuam um processo de desenvolvimento e gestão de projectos instituído, este deverá ser adoptado e adequado à realidade do desenvolvimento baseado em LPS, causando o menor impacto possível na operação.

As actividades de *Gestão do Âmbito* e *Planeamento Técnico* definem o quê e como será desenvolvida a linha de produtos, a qual poderá ter por base a refactorização dos elementos de software dos produtos existentes. A *Institucionalização e Optimização de Processos*, acompanha a adopção dos processos LPS pela equipa interna e optimização respectiva ao longo do tempo. As restantes actividades acompanham de forma transversal todo o esforço de desenvolvimento.

A Tabela 9 resume as actividades da categoria "Gestão", tarefas principais e responsabilidades.

Actividade	Tarefas Principais	Responsabilidade
Análise de Viabilidade	Análise da viabilidade na implementação de uma abordagem de desenvolvimento baseada em LPS. Análise da viabilidade de refactorização dos elementos de software dos produtos existentes Definição do mercado alvo e oportunidades para a LPS. Definição de parcerias e estratégia de distribuição dos produtos.	Mentor da LPS Responsável Financeiro Responsável Comercial
Gestão do Âmbito da LPS e Compreensão dos Domínios Relevantes	Definição do modelo de domínio da LPS, tendo por base a análise dos produtos existentes. Definição do que estará "dentro" e "fora" do âmbito da LPS. Aferir se um produto será membro ou não da linha de produtos. Tomada de decisão sobre o desenvolvimento interno de um elemento de software versus aquisição produto ou subcontratação do desenvolvimento.	Mentor da LPS Arquitecto da LPS
Gestão de Ferramentas	Seleccionar ferramentas de suporte à implementação e gestão da LPS, que permitam: - Suportar o processo de engenharia de domínio e engenharia aplicacional. - Suportar a definição do âmbito da LPS e gestão de requisitos multi-produto. - Representar os pontos de variabilidade nos elementos de software base da LPS nos vários sítios onde estes são articulados, e.g. requisitos, arquitectura, módulos de software, testes, planos. - Realizar a gestão de configurações de software da LPS. - Disponibilizar informação de suporte à tomada de decisão técnica ou à gestão.	Mentor da LPS Arquitecto da LPS
Planeamento Técnico	Definição do plano integrado (multi-projectos) e do seu conteúdo. Definir uma estrutura de trabalho padrão (em inglês *Work Breakdown Structure* – WBS) para um projecto típico. Estabelecer uma estimativa dos recursos requeridos para executar o plano.	Gestor(es) de Projecto

Actividade	Tarefas Principais	Responsabilidade
	Estabelecer o envolvimento dos recursos com o projecto Rever o plano de forma a garantir fiabilidade face ao envolvimento dos recursos identificados. Efectuar o controlo de execução do plano e tomar medidas preventivas face a não cumprimentos.	
Gestão de Configurações	Gerir versões no desenvolvimento em paralelo sobre os mesmos artefactos Gerir conflitos de versões na engenharia aplicacional distribuída, com ambientes de desenvolvimento heterogéneos. Gestão de *baselines* dos componentes de configuração. Gestão de diferentes linhas de configuração, existindo uma por produto, podendo existir linhas de configuração temporárias no âmbito dos projectos. Gestão das configurações no espaço de trabalho. Criação do ambiente de trabalho individual de cada elemento da equipa de desenvolvimento, incluindo a criação de vistas e controlo de acesso aos elementos de configuração.	Arquitecto da LPS Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional
Institucionalização e Optimização do Processo	Formação e consciencialização da equipa de engenharia de domínio. Institucionalização dos novos padrões e técnicas de desenvolvimento a adoptar. Formação nas ferramentas de suporte. Recolher indicadores das actividades de implementação e gestão da LPS. Documentar e optimizar os processos ao longo do tempo. Exemplos: a) Gestão de configurações de software, b) Operação da LPS, c) Gestão Integrada de Projectos, d) Colecção e análise de métricas e e) Gestão da Relação com o utilizador Identificar melhorias ao nível da comunicação e colaboração dos intervenientes no processo de desenvolvimento da LPS. Identificar melhorias e recomendações ao nível das ferramentas e técnicas utilizadas.	Mentor da LPS

Tabela 9 – Actividades da categoria "Gestão "

Na Tabela 9, a coluna das responsabilidades não está incluída a administração pois foi assumido que todas as decisões tomadas durante a fase de Preparação da LPS são aprovadas pela Administração e/ou Direcção Geral.

III.2.5.SPL-LIGHT: Engenharia

A engenharia de software tem como base a construção de um elemento de software ou produto de software. Numa LPS esta categoria objectiva a concepção e evolução dos elementos base da LPS por uma lado, e desenvolvimento de elementos de software específicos de um produto em particular por outro lado.

A construção da LPS poderá fazer-se a partir de uma arquitectura de software intermédia, resultado da análise dos produtos existentes na organização. Neste caso é necessário definir um plano de migração dos produtos da arquitectura intermédia para a arquitectura da LPS.

Para a actividade de engenharia é fundamental a construção do modelo de domínio, onde está expresso o conhecimento. A definição dos requisitos é alimentada pelo modelo de domínio e justifica a arquitectura a ser desenvolvida. A arquitectura é composta por um conjunto de elementos de software resultantes do desenvolvimento ou aquisição a entidades terceiras.

No caso da engenharia ser precedida da análise de produtos existentes, deveram ser considerados a arquitectura intermédia e processo de desenvolvimento existente em todas as actividades de engenharia da LPS.

A Tabela 10 resume as actividades da categoria "Engenharia", tarefas principais e responsabilidades.

Actividade	Tarefas Principais	Responsabilidade
Definição da Arquitectura	Definição da arquitectura de software da LPS e instanciação da abordagem de desenvolvimento. Definição dos estilos e padrões da arquitectura. Definir o "plano de produção" para instanciar cada um dos produtos a partir da LPS Definir o "plano de migração" para migrar os produtos da arquitectura intermédia (caso exista) para a arquitectura da LPS. Definir os mecanismos para implementar a variabilidade na arquitectura da LPS	Arquitecto da LPS
Engenharia de Requisitos	Definir os produtos e as funcionalidades/características dos produtos na LPS. Identificar pontos de variabilidade que podem ser instanciados para satisfazer requisitos específicos dos produtos. Para cada produto, refinar o âmbito e requisitos da LPS acrescentando os aspectos específicos a implementar.	Arquitecto da LPS Analista Funcional

Actividade	Tarefas Principais	Responsabilidade
Desenvolvimento de Elementos de software	Desenvolvimento de elementos base da LPS, com flexibilidade para suportar os pontos de variabilidade definidos na LPS. Implementação de pontos de variabilidade e variantes nos elementos de software base da LPS. Desenvolvimentos das interfaces entre os elementos de software base. Desenvolvimento ou instanciação de elementos de software específicos para produtos em particular.	Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional
Integração do Sistema	Combinação dos elementos de software de forma a perfazer um sistema como um todo. Integração do sistema no momento da instalação dos elementos base e instanciação dos variantes. Integração do sistema na compilação do produto.	Engenheiro(s) de Software de Domínio Integrador de Produtos
Testes	Teste aos activos base, os activos específicos de cada produto e as interacções entre eles. Validação dos modelos de análise e desenho Testes unitários Testes de integração dos subsistemas Testes de integração do sistema Testes de regressão Testes de conformidade Testes de aceitação	Arquitecto da LPS Analista Funcional Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional Integrador de Produtos

Tabela 10 – Actividades da categoria "Engenharia"

III.2.6.SPL-LIGHT: Operação e Gestão de Produtos

A Operação e Gestão de Produtos engloba as actividades de preparação da equipa, pré-venda, distribuição, adaptação e instalação dos produtos da LPS e gestão e acompanhamento dos clientes pós entrada em produção.

Em particular, as grandes empresas de desenvolvimento de LPS procuram estabelecer parcerias com empresas nos mercados alvo, nas quais delegam a responsabilidade pela distribuição e configuração dos produtos às especificidades dos mercados locais.

Nestes casos, o controlo de qualidade do processo de implementação dos produtos e certificação técnica dos profissionais responsáveis pela implementação dos produtos é fundamental para a correcta implementação do produto e satisfação dos clientes.

A boa comunicação entre as equipas de distribuição e a equipa de implementação da LPS é chave para a transmissão de novos requisitos a serem incluídos na LPS ou das

dificuldades ou problemas encontrados durante a implementação e que poderão implicar a alteração de elementos base na LPS. A Tabela 11 resume as actividades da categoria "Operação e Gestão de Produtos", tarefas principais e responsabilidades.

Actividade	Tarefas Principais	Responsabilidade
Gestão da Relação com o Cliente	Identificação das necessidades específicas e enquadramento das necessidades no plano de evolução dos produtos da LPS Envolvimento do cliente na especificação de requisitos e adaptação do produto à sua realidade. Gestão das reclamações de clientes pós implementação	Gestor(es) de Projecto
Gestão da Relação com Parceiros	Definição de protocolo de cooperação com parceiros tecnológicos Definição de contratos de parceria com organizações distribuidoras dos produtos Envolvimento dos parceiros na especificação dos requisitos e integração com a sua oferta específica Gestão de pedidos e suporte especializado aos parceiros certificados nos produtos da LPS.	Mentor da LPS Arquitecto da LPS Gestor(es) de Projecto
Certificação Técnica	Formação dos elementos da equipa de implementação na arquitectura técnica dos produtos e processo de implementação / instanciação de variabilidades. Formação técnica nos procedimentos e ferramentas para desenvolvimentos específicos integrados com os elementos base da LPS.	Engenheiro(s) de Software Aplicacional Integrador de Produtos
Migração dos Produtos existentes para LPS	Aplicação do plano de migração aos produtos existentes (já instalados), anteriores à concepção da LPS. Instalação da última *release* dos produtos de configuração. Instanciação dos variantes que satisfazem requisitos específicos de cada instalação. Migração e integração dos desenvolvimentos específicos existentes.	Integrador de Produtos
Instalação e Configuração dos Produtos	Instanciação dos pontos de variabilidade na LPS, face aos requisitos específicos do produto. Configurações adicionais específicas face aos requisitos do cliente. Instalação do produto no ambiente de produção do cliente.	Integrador de Produtos
Manutenção dos Produtos	Apoio ao cliente pós produção. Instalação de novas versões do produto ou correcções lançadas ao longo do tempo. Manutenção correctiva e pequenos desenvolvimentos solicitados pelo cliente.	Integrador de Produtos

Tabela 11 – Actividades da categoria "Operação e Gestão de Produtos"

No cenário em que a adopção da LPS é realizada sobre uma base de produtos instalados há a necessidade de realizar uma migração destes para uma *release* resultado do novo processo de engenharia.

Após a entrada de um produto em produção, é usual ser activado um contrato de manutenção que garante ao cliente a disponibilização das actualizações futuras do produto, segundo o plano de evolução definido. Cabe ao gestor de relação/conta com o cliente aferir de novas necessidades nos clientes actuais face à evolução natural do seu negócio. Nestes casos, poderá ser necessário o desenvolvimento de elementos de software específicos para o cliente ou caso seja aprovado, serem adicionados á LPS.

III.3. OPERACIONALIZAÇÃO DA METODOLOGIA SPL-LIGHT

O modelo SPL-LIGHT para a implementação e gestão de uma LPS proposto deve servir como referência para a instanciação das actividades de análise dos produtos existentes, gestão, engenharia e operação da LPS.

Cada organização, pela sua dimensão, cultura de desenvolvimento e estratégia de distribuição dos produtos pode adoptar apenas as actividades do modelo que mais se enquadrem à sua realidade. Por exemplo, uma organização que ela própria realiza a instalação dos produtos e consultoria junto dos clientes finais não necessita de incluir no seu modelo a actividade "*Gestão da relação com Parceiros*", da categoria "*Operação da LPS*". O modelo deve assim servir de modelo de suporte à adopção da abordagem ao desenvolvimento de software baseada em LPS.

Este modelo prevê a concepção da arquitectura de referência da LPS tendo por base a reengenharia dos elementos de software existentes (actividades da categoria "*Análise de Produtos Existentes*"). Em organizações que pretendam a implementação de uma LPS de raiz esta categoria não deve ser considerada.

IV. MODELO DE REPRESENTAÇÃO DA ARQUITECTURA DE UMA LPS: SPL-LIGHT

Neste capítulo, é apresentado o modelo para a representação da arquitectura de uma LPS: SPL-LIGHT.

Na concepção do modelo apresentado, foram tidos em consideração alguns pressupostos, enquadrados no contexto da sua utilização prática numa PME (ver IV.1). Na secção IV.2 estão documentadas as motivações na base da definição e implementação do modelo proposto, face às várias abordagens já existentes.

O modelo SPL-LIGHT contendo 3+1 vistas arquitecturais, encontra-se ilustrado em IV.3, sendo cada uma das suas vistas descritas em detalhe na secção IV.4. A proposta para um perfil UML com vista a facilitar a representação das variabilidades na descrição da arquitectura de uma LPS é apresentado em IV.6.

IV.1. PRESSUPOSTOS

A representação da arquitectura de uma LPS deverá ser simples e inteligível para facilitar a comunicação e compreensão da mesma pelos interlocutores alvo. Por outro lado, deverá ser completa de forma a focar as vistas que interessam aos diferentes tipos de interlocutores.

O esforço dispendido em documentar a arquitectura da LPS deve ser bem ponderado, de forma a ser adequado aos benefícios que se obtêm com a sua existência. Em particular, no contexto de uma PME pretende-se que os elementos mais seniores da equipa estejam afectos às tarefas mais críticas, onde rentabilizam os seus conhecimentos e experiência. No modelo proposto, a concretização dos vários modelos e decomposição dos elementos nos mesmos deve ser realizada na medida das necessidades, e.g. para um determinado requisito, poderá não haver necessidade da descrição dos cenários de comportamento funcional utilizando os casos de utilização.

As técnicas e linguagens utilizadas para a documentação da arquitectura da LPS deverão seguir as normas da indústria, de forma a serem conhecidos universalmente por qualquer novo elemento na equipa de implementação e gestão da LPS. No modelo proposto as vistas são representadas através da notação UML 2.0., estendida através de um *Perfil UML* específico para LPS, apresentado em IV.6.

Neste capítulo é proposto um modelo para a representação da arquitectura de uma LPS, segundo as perspectivas dos vários intervenientes, e.g. arquitecto da LPS, analista funcional, engenheiro de software, integrador produtos, utilizador final. O conjunto de etiquetas definidas para os elementos da arquitectura permitem não só os identificar univocamente na arquitectura (promovendo a referência entre elementos) como os caracterizar no contexto das LPS.

A descrição das vistas e estilos do modelo proposto será apresentada utilizando uma adaptação da estrutura definida por Clements et al [3]: a) Descrição Geral, b) elementos, relações e propriedades, c) âmbito de utilização, d) notação, e) relação com outras vistas e f) exemplos.

As secções seguintes, descrevem em detalhe a abordagem para a representação da arquitectura da LPS, segundo cada uma das vistas do modelo proposto.

IV.2. RACIONAL PARA O MODELO PROPOSTO

Alguns dos modelos de documentação de arquitecturas de software analisados (ver II.2.1), tal como o Siemens [7], não incluem a perspectiva funcional e organizacional do sistema correspondendo à descrição do problema no âmbito da LPS, perdendo-se com isso a expressividade e legibilidade dos modelos para intervenientes não técnicos usualmente, e.g. utilizador final, gestor de projecto, analista funcional. Abordagens como o RUP [5, 6] ou o RM-ODP[8] abordam esta necessidade.

A identificação unívoca dos requisitos da LPS e respectivos pontos de variabilidade e variantes, associando-os de forma explícita através de etiquetas UML, aos módulos de software na vista módulo, nos componentes de execução e conectores permite a identificação clara de quais os requisitos e quais os módulos de software implementados por um determinado elemento de software. Esta informação é de vital relevância para os gestores das equipas de desenvolvimento, na perspectiva da análise de impacto que uma alteração a um requisito na arquitectura de software existente, sendo mais fácil fazer uma estimativa de tempo e custo da alteração. A reutilização de software é igualmente potenciada pois através da análise das funcionalidades do sistema é possível identificar os elementos de software base que as implementam.

No modelo proposto, cada requisito é classificado por um conjunto de atributos, sendo o comportamento dos requisitos funcionais representado por casos de utilização e os requisitos não funcionais são especificados por cenários de atributos de qualidade.

A representação dos requisitos não funcionais ou atributos de qualidade é também uma das lacunas encontradas nas abordagens existentes. A especificação dos requisitos de qualidade permite a tomada de decisão sobre a arquitectura do sistema e selecção dos variantes arquitecturais definidos na LPS. Em Bass et. al.[2] é definida uma representação de cenários de atributos de qualidade, a qual foi adaptada de forma a satisfazer a realidade de uma LPS.

O modelo proposto apresenta uma perspectiva da representação multi-dimensional das vistas que descrevem a arquitectura da LPS e dos produtos instanciados a partir dela. A denominada "Vista Produtos" é apresentada como uma instanciação das restantes vistas no contexto particular de um produto, num determinado momento no tempo.

IV.3. MODELO SPL-LIGHT: REPRESENTAÇÃO DA ARQUITECTURA DE UMA LPS

O modelo proposto para a representação da arquitectura de uma LPS: Modelo SPL-LIGHT visa a representação das várias perspectivas da arquitectura de uma LPS, adequando-se ao perfil e necessidades dos intervenientes no processo de gestão e implementação da LPS.

Nesse sentido, o modelo é composto por (3+1) vistas principais sendo que a vista requisitos descreve o contexto do problema, i.e. as funcionalidades, comportamento, cenários de atributos de qualidade e âmbito da LPS, as vistas módulo e componente e conector (C&C) descrevem o contexto da solução através da identificação dos módulos de software, componentes e interfaces entre esses componentes.

A vista produtos (+1) é transversal às restantes e representa a instanciação das três vistas anteriores em artefactos no contexto de cada produto particular instanciado da LPS.

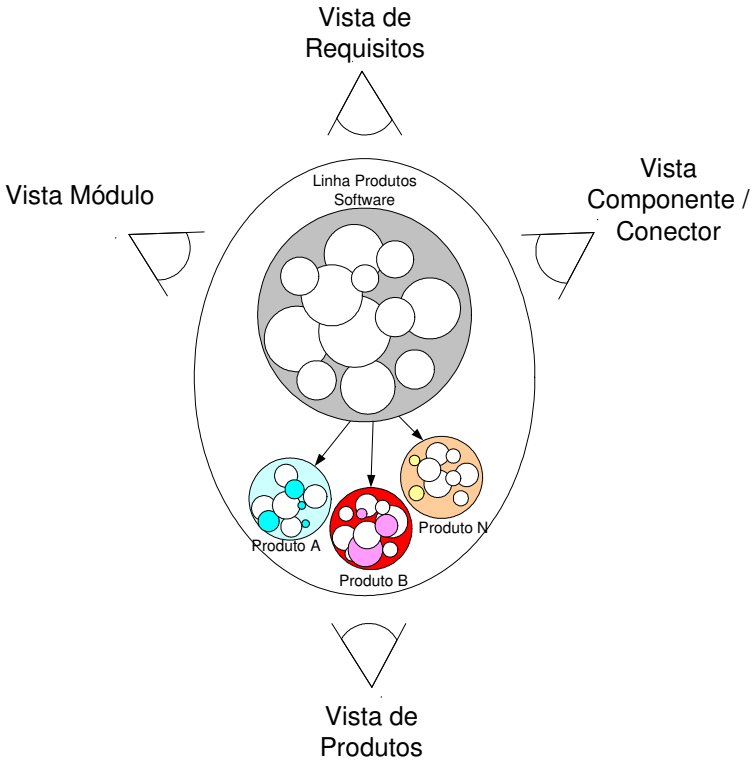


Figura 5 – Modelo SPL-LIGHT para a representação da arquitectura de uma LPS

A Tabela 12 apresenta uma descrição de cada uma das vistas do modelo SPL-LIGHT, considerando os objectivos e contexto da sua utilização no âmbito da implementação de uma LPS:

Vista	**Descrição**	**Intervenientes Alvo**
Vista de Requisitos	A vista de requisitos descreve o contexto do problema da LPS, através da representação dos requisitos funcionais e não funcionais no âmbito da LPS. Os pontos de variabilidade e tipos de variantes são representados de forma explícita em cada requisito de forma a distinguir as características (funcionalidades e atributos de qualidade) nucleares do sistema e as alternativas ou opcionais na instanciação dos produtos. Cada requisito funcional (ou funcionalidade) é descrito por um conjunto de atributos, sendo o seu comportamento representado por casos de utilização. Cada requisito não funcional (ou atributo de qualidade) é especificado por um cenário de atributo de qualidade. A representação da vista requisitos (nomeadamente dos requisitos funcionais) é uma óptima ferramenta de trabalho e comunicação entre os analistas funcionais e os utilizadores chave no sentido da elegibilidade e especificação dos requisitos funcionais. A inclusão de cenários de atributos de qualidade é uma via privilegiada para suportar a tomada de decisão em relação ao desenho da arquitectura e selecção de padrões de desenho de software a utilizar.	Mentor da LPS Gestor Projecto Arquitecto da LPS Analista Funcional Utilizadores Chave
Vista Módulo	A vista módulo tem como objectivo a descrição dos elementos de software da LPS e as suas dependências. A representação da vista módulo segue uma técnica de decomposição, desde o sistema como um todo até ao módulo unitário de código de um método de uma classe ou função que implementa uma determinada funcionalidade. A representação dos pontos de variabilidade e variantes é explicita, sendo utilizado o padrão fábrica para a sua implementação. Os módulos opcionais, externos e aplicacionais são igualmente representados de forma explícita. Os módulos são áreas de responsabilidade funcional, e são atribuídos ás equipas de implementação do domínio da LPS ou de um produto em particular. Existe uma ligação forte entre os elementos da vista módulo e os elementos da vista de requisitos, sendo possível a rastreabilidade entre uma funcionalidade e o módulo que a implementa.	Arquitecto da LPS Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional

Vista	Descrição	Intervenientes Alvo
Vista Componente & Conector	A vista componente & conector (C&C) descreve os elementos de software e respectivas ligações, em tempo de execução. Nesta vista são representados os componente, i.e. objectos (caso sejam linguagens OO), os serviços servidor, clientes bem como os conectores que expõem as interfaces dos componentes. Em particular são representadas as ligações entre os portos dos componentes e papeis dos conectores. Em cada projecto a instanciar, a definição e selecção dos conectores a utilizar dependerá dos atributos de qualidade pretendidos para o sistema. Os elementos da vista C&C estão associados aos elementos da vista módulo de forma a garantir a correspondência do código com o serviço, conector ou processo de execução.	Arquitecto da LPS Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional Integrador de Produtos
Vista de Produtos	A vista produtos objectiva a visualização dos elementos de configuração que constituem a vista requisitos, vista módulo e vista componente & conector, numa perspectiva da linha de configuração base da LPS ou de cada um dos produtos instanciados ao longo do tempo. Um produto é constituído pela documentação das funcionalidades, atributos de qualidade, e módulos de software, bem como do resultado da sua implementação num conjunto de ficheiros separados, e.g.: código fonte, ficheiros de configuração, *makefiles*, e executáveis. Esta vista pretende a visualização dos elementos de configuração, não só na perspectiva da evolução de versões ao longo do tempo, mas sobretudo na perspectiva de cada produto da LPS, i.e. identificação das versões de cada elemento de configuração que estão na linha de evolução base da LPS e instanciadas nos vários produtos da LPS.	Analista Funcional Engenheiro(s) de Software de Domínio Engenheiro(s) de Software Aplicacional Integrador de Produtos

Tabela 12 – Descrição das vistas do modelo SPL-LIGHT

IV.4. MODELO SPL-LIGHT: VISTAS ARQUITECTURAIS

IV.4.1.Vista de Requisitos

IV.4.1.1.Descrição Geral

A vista requisitos do modelo SPL-LIGHT representa a descrição do contexto do problema na LPS, i.e. descreve o âmbito da LPS, o que o sistema deve fazer e os atributos de qualidade que deve respeitar. A representação do modelo de domínio e comportamento funcional é também descrito nesta vista.

Os pontos de variabilidade e variantes são representados de forma explícita em cada requisito de forma a distinguir os requisitos obrigatórios que são satisfeitos pelo sistema e os que são alternativos ou opcionais na instanciação dos produtos.

As representações da vista de requisitos são óptimas ferramentas de trabalho e comunicação entre os analistas funcionais e os utilizadores chave no sentido da identificação e especificação dos requisitos funcionais.

A descrição do contexto do problema não ficaria concluído se não fossem representadas as todas as entidades que fazem parte do modelo de domínio da solução. Neste sentido a vista de requisitos inclui a definição do modelo de domínio.

IV.4.1.2.Âmbito de utilização

O âmbito de utilização da vista de requisitos foca as actividades de preparação e definição do âmbito da LPS, suporte às actividades de engenharia da LPS. Em particular, inclui os seguintes cenários:

- Comunicação pouco técnica do âmbito, funcionalidades do sistema e alternativas de implementação de forma inteligível, aos utilizadores de negócio, gestores de projecto ou potenciais clientes da solução.
- Definição do âmbito da LPS, respectivos pontos de variabilidade e variantes.
- Definição dos principais módulos funcionais da LPS. Descrição e classificação das funcionalidades.
- Descrição das dependências entre módulos funcionais e do comportamento espectável de cada funcionalidade num cenário normal e de excepção.

- Descrição dos atributos de qualidade do sistema ou requisitos não funcionais, tais como desempenho, segurança, disponibilidade, robustez ou características de usabilidade, que suportam decisões de desenho da arquitectura da LPS.
- Suporte à criação dos planos de teste funcionais e de aceitação para a engenharia de domínio ou no contexto de um projecto de implementação de um produto.
- Suporte ao desenho detalhado e implementação dos módulos de código, com base na descrição detalhada do comportamento espectável de cada funcionalidade.

IV.4.1.3.Elementos e relações

Os elementos da vista de requisitos estão representados pela Tabela 13:

Elementos	Descrição
Requisito	Um requisito representa uma característica do sistema, funcional ou um atributo de qualidade (e.g. desempenho, disponibilidade, segurança).
Caso de utilização	Um caso de utilização é a representação do comportamento de uma funcionalidade num contexto de execução.
Actor	Um actor representa um papel que uma pessoa, sistema ou processo automático representa na interacção com os casos de utilização.
Cenário Atributo de Qualidade	Um cenário atributo de qualidade descreve a ocorrência de estímulos associados a um determinado atributo de qualidade e a reacção espectável do sistema face a essa ocorrência.
Decisão Arquitectural (ou Táctica Arquitectural)	Uma decisão arquitectural ou táctica é uma medida tomada pelo arquitecto da LPS para satisfazer as repostas pretendidas em um ou vários estímulos descritos nos cenários de atributos de qualidade.
Entidade	Uma entidade representa um conceito no âmbito da LPS. O conjunto das entidades representa o âmbito da LPS.

Tabela 13 – Elementos da Vista de Requisitos

O Perfil UML apresentado em IV.6, inclui estereótipos com base nos elementos acima apresentados, importantes no contexto da gestão de variabilidades numa LPS. Para os cenários de atributos de qualidade e decisões arquitecturais é utilizado o perfil UML definido por Zhu [38].

Na vista requisitos, as relações entre os elementos são representadas, tal como descrito pela Tabela 14.

Elementos	Relação	Descrição
Entre Requisitos	Composição (*é-composto-por*)	A relação de composição entre os requisitos aplica-se em requisitos funcionais. Esta relação implica que a funcionalidade "pai" ou módulo funcional é composta por um conjunto de funcionalidades de menor granularidade. Este tipo de relação permite a representação da decomposição funcional de um sistema desde os módulos funcionais principais ao detalhe das funcionalidades individuais.
	Dependência (*depende-de*)	Um requisito depende de outro, quando para a sua satisfação necessita que essa tenha sido satisfeita em primeiro lugar. A um nível macro esta relação é representada nos modelos de requisitos. Em detalhe é representada pelos casos de utilização dessa funcionalidade ou cenários de atributos de qualidade.
	Generalização (*é-uma*)	Um requisito é uma generalização de outro quando a sua implementação herda as propriedades e comportamentos do requisito pai, estendidos com comportamento e propriedades próprias.
Entre Requisito e Caso de Utilização	Descrição (*descrito por*)	A relação entre os elementos requisito (do tipo funcional) e caso de utilização é uma relação implícita do tipo "descrito por". Em que o comportamento de um requisito funcional pode ser descrito através de um ou mais casos de utilização.
Entre Requisito e Cenário de Atributo de Qualidade	Caracterização (*caracterizado por*)	A relação entre os elementos requisito (do tipo não funcional) e cenário atributo de qualidade é uma relação implícita do tipo "caracterizado por". Em que um requisito não funcional pode ser caracterizado por pares estímulo/resposta do sistema.
Entre Casos de Utilização	Inclusão (*inclui*)	A relação de inclusão entre um caso de utilização A e um caso de utilização B verifica-se quando A inclui o comportamento de B.
	Extensão	A relação de extensão entre um caso de utilização A e um caso de utilização B verifica-se quando A poderá opcionalmente estender o comportamento de B.
	Generalização (*é-uma*)	Um caso de utilização é uma generalização de outro quando a sua implementação herda as propriedades e comportamentos do caso de utilização pai, estendida com comportamento e propriedades próprias.
	Associação	A relação de associação associa um actor a um caso de utilização.
Entre Cenário(s) de atributos de qualidade e táctica	Realização (*controla-resposta*)	Uma táctica está relacionada com um ou mais cenários de atributos de qualidade na medida em que propõe uma solução para controlar as respostas espectáveis do sistema face aos estímulos descritos.
Entre Entidades	Generalização (*é-uma*)	Uma entidade é uma generalização de outra quando esta herda as propriedades da entidade pai utilização pai, estendida com propriedades próprias.
	Associação	A relação de associação associa duas entidades. Uma associação pode ser opcional ou ocorrer várias vezes entre duas entidades.

Tabela 14 – Tipos e relações da Vista de Requisitos

O Perfil UML apresentado em IV.6 inclui extensões e redefinições das relações acima apresentadas, importantes no contexto da gestão de variabilidades nas LPS.

IV.4.1.4.Atributos

Cada elemento da vista de requisitos (funcionalidade e caso de utilização) possui um conjunto de metadados, i.e. atributos ou propriedades que identificam e classificam o requisito na arquitectura da LPS.

A Tabela 15 identifica os principais atributos dos elementos da vista de requisitos.

Elemento	**Atributo**	**Descrição**
Requisito	ID	Identificador unívoco da funcionalidade
	Estereótipo	Identificação do estereótipo instanciado.
	ID Caso Utilização	Identificador unívoco dos casos de utilização que implementam a funcionalidade.
	Objectivo	Descrição do Objectivo da Funcionalidade
	Tipo de Requisito	Tipo de funcionalidade, e.g. funcional, segurança, desempenho e usabilidade.
	Valores possíveis	Identificação da lista de valores ou restrições existentes.
	Critério de satisfação	Critério para satisfação dos testes à funcionalidade.
	Rational	Motivo pelo qual a funcionalidade existe
	Prioridade	Prioridade de implementação da funcionalidade, segundo o impacto para o negócio e dependências com outras funcionalidades.
	… Tagged values	Cada estereótipo do *Perfil* UML apresentado em IV.6 poderá incluir alguns atributos adicionais específicos para a gestão de variabilidades nas LPS.
Caso de Utilização	ID	Identificador unívoco do caso de utilização
	Nome	Nome do caso de utilização
	ID Requisito	Identificador unívoco dos requisitos funcionais cujo comportamento está totalmente ou parcialmente representada pelo caso de utilização.
	Estereótipo	Identificação do estereótipo instanciado.
	… Tagged values	Cada estereótipo do *Perfil UML* apresentado em IV.6 poderá incluir alguns atributos adicionais específicos para a gestão de variabilidades nas LPS.
Actor	ID	Identificador unívoco de actor na LPS
	Tipo	Tipo de actor: pessoa ou máquina.

Elemento	Atributo	Descrição
Cenário Atributo de Qualidade	ID	Identificador unívoco do cenário de actor na LPS
	ID Requisito	Identificados unívoco do requisito não funcional o qual este cenário caracteriza.
	Fonte do Estimulo	Entidade que gera o estímulo (poderá ser uma pessoa ou sistema).
	Estímulo	Condição que deve ser considerada quando ocorre no sistema.
	Ambiente	Condições do ambiente de execução aquando da ocorrência do estimulo.
	Artefactos	Artefactos alvo do estimulo. Poderão ser alguns elementos de software do sistema ou o sistema na totalidade.
	Resposta	Actividade que deverá ser levada a cabo quando o estimulo ocorre.
	Valores Espectáveis	Forma de validar a adequação da resposta ao estímulo, de forma a esta poder ser alvo de testes.
Decisão de Desenho (ou Táctica Arquitectural)	ID	Identificador unívoco da táctica.
	Tipo de Táctica	Tipificação da táctica, e.g. alto desempenho, recuperação de falha, gestão de recursos do sistema, interoperabilidade.
	Descrição da Táctica	Descrição da táctica e medidas pretendidas para a implementação da táctica.
	Participantes na Decisão	Lista de participantes na decisão arquitectural
	Racional para a Táctica	Motivações para a criação da táctica
	Limitações da Táctica	Limitações ou constrangimentos associados à implementação da táctica.
Entidade	ID	Identificador unívoco da entidade
	Âmbito	Enquadramento num dos conceitos principais no âmbito da LPS.
	ID Requisitos	Lista de requisitos associados com a entidade
	Nome	Nome da Entidade
	Esteriótipo	Identificação do estereotipo do módulo.
	… Tagged values	Cada estereótipo do Perfil UML apresentado em IV.6 poderá incluir alguns atributos adicionais específicos para a gestão de variabilidades nas LPS.

Tabela 15 – Atributos dos elementos da Vista de Requisitos

Para a identificação dos atributos para a classificação do elemento requisito foram utilizados vários modelos de referência, nomeadamente a classificação dos requisitos apresentada pelo modelo de especificação de requisitos de Volere [58]. Na definição

das propriedades do elemento cenário atributo de qualidade, foi seguida a classificação dos atributos de qualidade apresentada por Bass et. al. [2].

IV.4.1.5.Notação

A notação utilizada para a representação da vista de requisitos, tem como base a linguagem UML 2.0. O Perfil UML, apresentado em IV.6, apresenta uma extensão para a implementação de LPS, nomeadamente a inclusão de estereótipos e etiquetas que permitem explicitar o papel de um elemento na LPS, i.e. se é um ponto de variabilidade, variante, opcional, externo ou aplicacional.

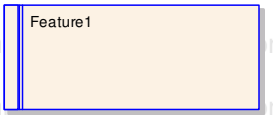
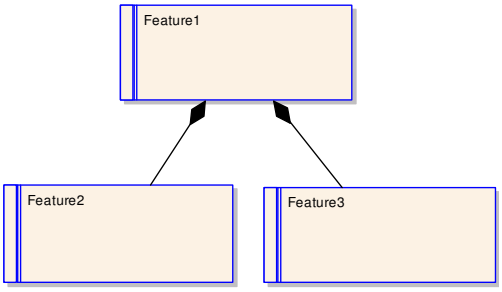
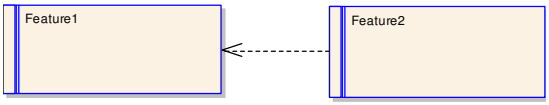
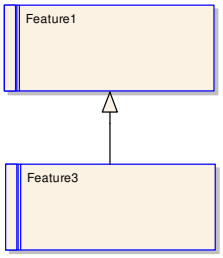
Item	**Representação UML**
Requisito	Extensão ao UML: *Feature* 
Relação "composto por"	Composição UML entre requisitos um pai e outro(s) filho(s): 
Relação "depende de	Dependência UML entre requisitos: 
Relação "é uma	Herança UML entre dois requisitos, um pai e um filho. 

Tabela 16 – Notação para elementos e relações do modelo de requisitos da vista requisitos

A tabela anterior ilustra a notação utilizada para representar os elementos e relações nos modelos da vista de requisitos, para o modelo de requisitos. Na Tabela 17 é apresentada a notação para o modelo de casos de utilização da vista requisitos.

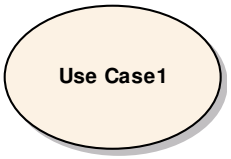
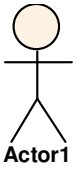
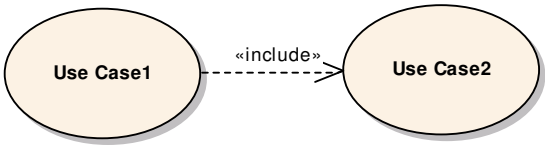
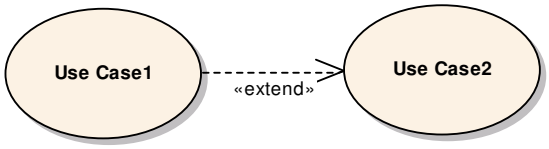
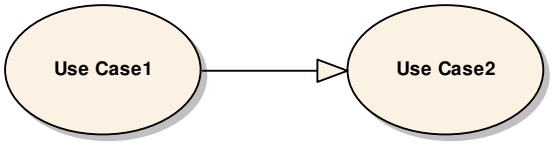
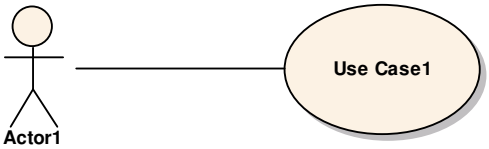
Item	Representação UML
Caso de Utilização	Caso de Utilização UML 
Actor	Actor UML 
Relação Inclusão "*inclui*"	Relação inclusão UML para casos de utilização 
Relação Extensão "*estende*"	Relação extensão UML para casos de utilização 
Relação Generalização "*é-uma*"	Relação Herança UML para casos de utilização 
Relação de Associação	Relação Associação UML para casos de utilização 

Tabela 17 – Notação para elementos e relações do modelo de casos de utilização da vista requisitos

Na Tabela 18 é apresentada a notação para o modelo de cenários de atributos de qualidade da vista requisitos.

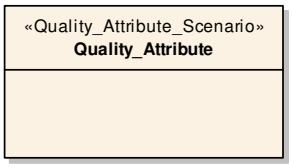
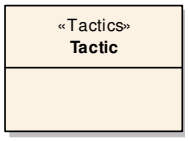
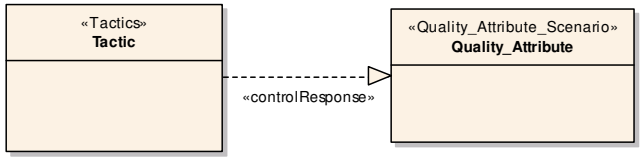
Item	Representação UML
Cenário Atributo Qualidade	Elemento UML cenário atributo de qualidade (extensão Class UML).  Nota: No Perfil UML apresentado em IV.6 são apresentados estereótipos que especificam os cenários de atributos de qualidade, e.g. desempenho, fiabilidade, segurança, robustez, facilidade modificação, interoperabilidade.
Decisão de Desenho (ou Táctica Arquitectural)	Elemento UML decisão de desenho (ou táctica) (extensão *Class* UML). 
Relação Realização "*controla resposta*"	Relação Realização UML (extensão *Realization* UML). 

Tabela 18 – Notação para elementos e relações do modelo de cenários de atributos de qualidade vista requisitos

Na Tabela 19 é apresentada a notação para o modelo de domínio da vista requisitos.

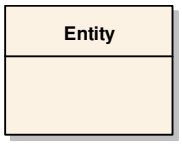
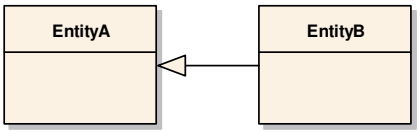
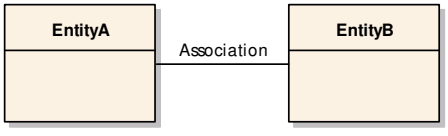
Item	Representação UML
Entidade	Elemento UML Entidade (extensão *Class* UML). 
Relação Generalização "*é-uma*"	Relação Herança UML para Entidades 
Relação Associação	Relação Associação UML entre Entidades 

Tabela 19 – Notação para a representação dos elementos e relações do modelo de domínio da Vista de Requisitos

IV.4.1.6.Relação com outras vistas

A vista de requisitos está intrinsecamente ligada às restantes vistas do modelo SPL-LIGTH. Em particular existe uma relação explícita entre os elementos da vista requisitos (requisitos do tipo funcional e entidades) e a sua implementação em elementos da vista módulo (módulos). Por outro lado, existe uma relação explícita entre os elementos da vista requisitos (requisitos não funcionais, atributos de qualidade e tácticas arquitecturais) e a implementação dos conectores da vista C&C.

Os artefactos produzidos nos modelos da vista de requisitos são alvo de gestão de configurações da LPS e estarão associados ao ramo de configuração da LPS ou de um produto em particular, visíveis pela vista produto.

IV.4.2.Vista Módulo

IV.4.2.1.Descrição Geral

A vista módulo consiste na decomposição funcional dos elementos de software da arquitectura da LPS em módulos de software, desde os módulos funcionais "*alto-nível*" até ao módulo unitário que implementa uma determinada funcionalidade. A decomposição pode ser realizada em sub módulos desde que estes possam ser entregues a um programador como uma unidade de código, capaz de ser desenvolvida e testada de forma isolada.

Os módulos são áreas de responsabilidade funcional, e são atribuídos pelo responsável da equipa aos engenheiros de software de domínio da LPS ou de um produto em particular.

Os módulos são implementações das funcionalidades e comportamentos descritos pelos requisitos funcionais e casos de utilização, satisfazendo as decisões de desenho necessárias para satisfazer os atributos de qualidade definidos para o sistema.

IV.4.2.2.Âmbito de utilização

O âmbito de utilização da vista módulo foca essencialmente as actividades de engenharia da LPS. Em particular, inclui os seguintes cenários:

- Criação ou alteração do código fonte que implementa uma funcionalidade base da LPS, pela equipa de engenharia do domínio.
- Criação ou alteração do código fonte que implementa uma funcionalidade específica de um produto, pela equipa de engenharia aplicacional.

- Análise de impacto de uma alteração nos elementos de software existentes. Esta análise é muito importante para poder estimar e definir prioridades face ao impacto da implementação/alteração de uma funcionalidade (identificadas pela etiqueta *Req_ID* em cada módulo).
- Comunicação da arquitectura técnica e implementação das funcionalidades a novos interlocutores. A utilização de uma notação padrão objectiva a diminuição das barreiras à aprendizagem.

IV.4.2.3.Elementos e relações

O elemento da vista módulo é o módulo.

Elemento	Descrição
Módulo	Um módulo representa uma unidade de implementação funcional do sistema. i.e. dependendo do grau de decomposição um módulo pode representar um pacote de funcionalidades ou uma funcionalidade particular. Uma funcionalidade pode estar implementada por mais que um módulo de software.

Tabela 20 – Elementos da Vista Módulo

São exemplos da concretização de um módulo os seguintes artefactos: código fonte, ficheiros de configuração, *makefiles* e bibliotecas.

A vista módulo pressupõe a possibilidade de representar as relações entre módulos da seguinte forma:

Relação	Descrição
Generalização (*é-um*)	Representa uma relação hierárquica entre dois módulos: um módulo pai e o módulo filho. Numa linguagem OO esta relação é representada pela relação de herança entre classes.
Decomposição (*é-parte-de*)	A relação um módulo *é-parte-de* outro módulo implica que ocorreu uma decomposição de um módulo principal em vários outros sub módulos, cada um *é-parte-de* o módulo principal.
Utilização (*usa*)	Um módulo "usa" outro quando está implícita uma relação de dependência entre do primeiro módulo no segundo.

Tabela 21 – Tipos de relações da Vista Módulo

IV.4.2.4.Atributos

Cada módulo possui um conjunto de metadados, i.e. atributos ou propriedades que identificam e classificam o módulo na arquitectura da LPS.

A Tabela 22 identifica os principais atributos do elemento módulo.

Atributo	Descrição
ID Módulo	Identificador unívoco do módulo
Nome	Nome do módulo
Objectivo	Enquadramento do módulo no sistema. Que funcionalidades implementa.
ID Requisitos	Lista de requisitos implementados pelo módulo.
Restrições à Implementação	Identificação das restrições de implementação ou requisitos não funcionais implementados.
Primeira Versão	Versão em que o módulo foi adicionado, e.g. 2.0.
Última Versão	Última versão em que o módulo em causa sofreu alterações.
Estereótipo	Identificação do estereotipo do módulo.
… *Tagged values*	Cada estereótipo do Perfil *UML* apresentado em IV.6 poderá incluir alguns atributos adicionais específicos para a gestão de variabilidades nas LPS.

Tabela 22 – Atributos dos elementos da Vista Módulo

IV.4.2.5.Notação

A notação utilizada para a representação da vista módulo, tem como base a linguagem UML 2.0. O Perfil UML, apresentado em IV.6, apresenta uma extensão para a implementação de LPS.

A Tabela 23 ilustra a notação utilizada para representar os elementos e relações nos modelos da vista módulo.

IV.4.2.6.Relação com outras vistas

A vista módulo está intrinsecamente ligada à vista de requisitos na medida em que cada módulo implementa um ou mais requisitos funcionais de forma parcial ou total, materializando o comportamento definido pelos casos de utilização e entidades do modelo de domínio. A relação entre os elementos módulos e requisitos é explícita, pela inclusão no módulo da lista de requisitos implementada.

A relação da vista módulo com a vista componente & conector é explícita na medida em que os módulos na vista módulo correspondem a componentes de execução na vista componente & conector.

Quer na construção dos elementos base, quer em cada produto instanciado, os artefactos resultantes da implementação dos módulos da vista módulo bem como os modelos da vista módulo são mantidos sobre gestão de configurações, no ramo de configuração do produto respectivo.

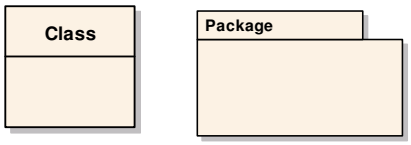
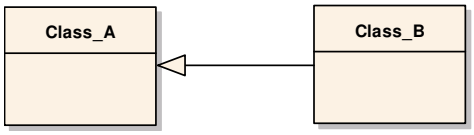
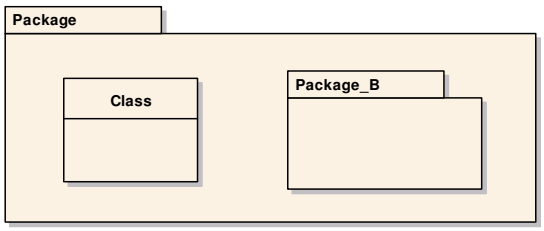
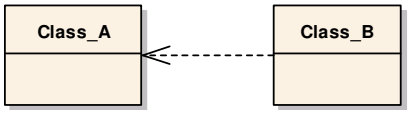
Item	Representação UML
Módulo	Classe ou Pacote UML (para os módulos principais "*alto-nível*" dos sistema). 
Relação "*é-uma*"	Herança entre classe pai e classe filho. 
Relação "*é-parte-de*"	Os módulos funcionais "*alto-nível*" são representados por pacotes UML. Os sub módulos são classes ou pacotes UML pertencentes a esse pacote UML. 
Relação "*usa*"	Relação dependência UML (B "usa" A). 

Tabela 23 – Notação para a representação dos elementos e relações da Vista Módulo

IV.4.3.Vista Componente & Conector

IV.4.3.1.Descrição Geral

A vista Componente & Conector (C&C) descreve as instâncias de execução dos elementos de software da arquitectura LPS e respectivas ligações, em tempo de execução. Nesta vista são representados os componentes, i.e. objectos (caso sejam linguagens OO), os serviços servidor, clientes e conectores que expõem interfaces para os componentes e poderão ser invocados em tempo de execução.

Os componentes são instâncias de execução que podem incluir as funcionalidades de um ou mais elementos da vista módulo (dependendo do processo de compilação), e.g. uma classe, elemento da vista módulo poderá corresponder a vários objectos numa vista C&C. Um outro exemplo é a criação de um ficheiro executável ou biblioteca com base num pacote (*package*) contendo diversas classes.

De forma a garantir a correspondência do código que implementa cada componente ou conector, cada componente deverá conter o(s) identificador(es) do módulo(s) de origem e funcionalidade(s) que implementa. Do mesmo modo, um conector deverá referenciar o(s) atributos de qualidade e tácticas arquitecturais associadas, bem como o(s) módulo(s) que o implementa, e.g. uma *IDL Corba* ou *WebService*.

Consoante a natureza e arquitectura do sistema, a descrição da arquitectura segundo uma vista C&C pode ser instanciada através de um estilo arquitectural específico, e.g. *cliente-servidor*, *par-a-par*, processos comunicantes, dados partilhados. Existem padrões definidos para a representação de cada um destes tipos de arquitecturas, que instanciam os elementos e relações da vista C&C.

IV.4.3.2.Âmbito de utilização

O âmbito de utilização da vista C&C está estritamente relacionado com a análise e percepção dos atributos de qualidade do sistema, e.g. o desempenho, disponibilidade, segurança, fiabilidade. A utilização da descrição da arquitectura segundo a vista C&C obtém benefícios nos seguintes cenários:

- Identificar a carga computacional que um componente está sujeito. Definir melhorias na estrutura do sistema, optimizando o desempenho de execução;
- Identificar os componentes críticos da arquitectura em tempo de execução e que poderão justificar a implementação de balanceamento de carga, e.g. através de uma infra-estrutura de execução em *cluster*;
- Identificar os recursos de dados partilhados pelos componentes em execução.
- Perceber os fluxos de controlo do sistema, de forma a facilitar a depuração de erros.

IV.4.3.3.Elementos e relações

Os elementos da vista C&C são os componentes e os conectores.

Elemento	Descrição
Componentes	Os componentes são unidades de processamento e armazenamento de dados que um sistema executa. Exemplos: servidor, cliente, filtro, objecto, base de dados. Os componentes expõem interfaces para o exterior, a serem invocados por outros componentes, essas interfaces são descritos através de portos.
Conectores	Os conectores são as ligações para a partilha de dados e controlo entre componentes, e.g. chamadas remotas de procedimentos entre um cliente e um servidor, ou entre dois objectos; mensagens assíncronas, divulgação de mensagens mediante subscrição. Os conectores possuem um ou mais papeis que descrevem como os componentes deverão estabelecer a comunicação através do conector.

Tabela 24 – Elementos da Vista C&C

Numa vista C&C as relações entre os componentes e conectores é representada da seguinte forma:

Relação	Descrição
Ligação	A relação de ligação entre componentes e conectores existe se existir uma associação entre os portos de um componente e os papeis de um conector, i.e. um porto de um componente *po* está ligado a um papel de um conector *pa* se o componente interage através do conector, utilizando a interface descrita por po e estando conforme com as regras definidas por *pa*.

Tabela 25 – Relações entre elementos da Vista C&C

IV.4.3.4.Atributos

Cada elemento possui um conjunto de metadados, i.e. atributos ou propriedades que identificam e os classificam na arquitectura da LPS.

A Tabela 26 identifica os principais atributos dos elementos: componente e conector.

Elemento	Atributo	Descrição
Componente	ID Componente	Identificador unívoco do componente
	Nome	Nome do componente
	ID Requisitos	Lista de requisitos implementado pelo componente
	ID Módulo	Lista de módulo(s) que estão contidos no componente de execução.
Conector	ID Conector	Identificador unívoco do conector
	Nome	Nome do conector
	ID Requisitos	Lista de requisitos não funcionais implementados pelo conector.
	ID Módulo	Lista de módulo(s) que estão contidos no componente de execução.
	ID Táctica	Lista de tácticas de arquitectura que justificam as características do conector.

Tabela 26 – Atributos dos elementos da Vista C&C

IV.4.3.5.Notação

A notação utilizada para a representação da vista módulo, tem como base a linguagem UML 2.0, a qual teve uma evolução significativa para a representação de componentes e conectores, face à versão 1.4.[59].

A Tabela 27 ilustra a notação utilizada para representar os elementos e relações nos modelos da vista C&C.

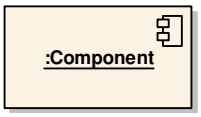
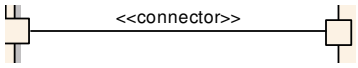
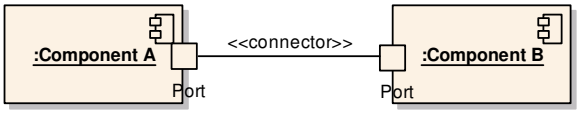
Item	Representação UML
Tipo de Componente	Componente UML. 
Componente	Instância de Componente UML. 
Porto	Porto UML 
Conector	Associação UML entre dois portos de componentes  OU Classe UML separada 
Relação "anexação"	Associação UML como conector:  OU Classe UML como conector: 

Tabela 27 – Notação UML para representar a vista C&C

Ao nível da representação dos conectores, no contexto da vista C&C, apesar de existir um novo conceito de conector, a sua expressividade é pobre para poder ser utilizada, sem restrições, para representar o conceito conector. Assim, na notação proposta a representação deste conceito é apresentada em duas perspectivas, que devem ser utilizadas consoante exista necessidade ou não de representar o comportamento do conector:

- Associação simples UML, caso seja pretendido apenas representar os tipos de conectores que o sistema possui, sem detalhar as suas propriedades ou comportamento.

- Instância de uma Classe UML, permite a representação do comportamento do conector, através da inclusão de atributos ou valores etiquetados. Permite igualmente a representação dos seus papéis, através do elemento *Port* UML associado à instância da classe.

IV.4.3.6.Relação com outras vistas

A vista C&C está directamente ligada à vista de requisitos em particular com a implementação dos atributos não funcionais (ou de qualidade do sistema), e.g. desempenho, disponibilidade, fiabilidade. A selecção do tipo de conector e tipo de interface via conectores depende das tácticas arquitecturais definidas. A relação entre os modelos faz-se pelo atributo "ID Requisitos", tanto nos componentes como nos conectores, e "ID Táctica" nos conectores.

A relação dos elementos módulo e componentes poderá ser complexa, na medida em que a um módulo poderão corresponder vários componentes de execução da vista C&C e vice-versa.

Os conectores poderão corresponder a uma implementação ou configuração dos atributos de qualidade, contendo igualmente um ou mais módulos da vista módulo associado. Por outro lado, em elementos de software adquiridos a entidades externas em que apenas estão disponíveis os binários, estes correspondem a componentes de execução que não têm módulos associados.

Os modelos da representação da vista C&C são mantidos sobre gestão de configurações, no ramo de configuração principal da LPS e de cada produto na vista produto. A gestão de configurações dos componentes de execução poderá ou não ser realizada, dependendo do ambiente de execução e da necessidade de compilação no ambiente de execução final.

IV.4.4.Vista Produto

IV.4.4.1.Descrição Geral

A vista produto permite obter uma perspectiva dos artefactos que compõem um produto em particular ou da linha principal de evolução da LPS.

Esta vista pretende a visualização dos elementos de configuração, não só na perspectiva da evolução das versões dos seus elementos de configuração ao longo do tempo, mas sobretudo na perspectiva da LPS e de cada produto instanciado (incluindo os variantes e desenvolvimentos específicos).

A concepção de um produto de configuração específico no âmbito da engenharia aplicacional é realizada pela simples selecção do *baseline* pretendido (criado na linha

de configuração principal da LPS) para cada um dos componentes de configuração no âmbito do projecto. Os desenvolvimentos específicos e instanciação dos variantes são realizados sobre a linha de configuração principal do projecto, logo sem interferir com a linha de configuração principal da LPS.

IV.4.4.2.Âmbito de utilização

O âmbito de utilização da vista produto é transversal às várias actividades do processo de implementação e gestão da LPS. Os itens seguintes identificam cenários de utilização prática da vista produto:

- Obtenção das últimas configurações dos elementos base da LPS de forma a iniciar a instanciação de um novo produto específico.
- Comparação das versões do mesmo componente de configuração em várias implementações de produtos específicos (e com a LPS) de forma a aferir da sua estabilidade e ponderar a alteração dos elementos base respectivos.
- Obtenção dos artefactos que implementam um determinada funcionalidade, agregados num componente de configuração, e.g. documentação, código fonte, ficheiros de configuração e binários.
- Análise do impacto que a alteração de um elemento base tem na LPS e produtos instalados, face à possibilidade de existirem diferentes versões desse componente de configuração nos vários produtos instanciados.

IV.4.4.3.Elementos e relações

O elemento principal da vista produto é o elemento de configuração. As definições dos conceitos apresentados estão igualmente apresentadas em II.1.9.

Elemento	Descrição
Elemento de configuração	Um elemento de configuração é um artefacto que pode ser alvo de gestão de versões individualmente. São exemplos de artefactos, os modelos de análise e desenho, código fonte, ficheiros de configuração, *makefiles*, documentação da arquitectura e plano de produção. Durante o processo de implementação e gestão da LPS serão produzidos artefactos, comuns à LPS ou específicos de um produto particular. Esses artefactos são alvo de gestão de versões, nas linhas de configuração dos produtos e/ou LPS.
Componente de Configuração	Um componente de configuração é um agrupamento de elementos de configuração, que representa um elemento básico de construção de um produto de configuração. Um componente possui um conjunto de *baselines*. Um elemento de configuração pertence a um e apenas um componente de configuração. Numa LPS um componente de configuração representa uma parte do produto que pode ser entregue a um elemento ou equipa de desenvolvimento e pode ser testada separadamente das restantes.

Elemento	Descrição
Baseline	Um *baseline* é um conjunto de versões de elementos de configuração para um componente de configuração, num determinado instante no tempo. A realização de *baselines* permite como que obter uma "fotografia" das configurações actuais, de forma a que esta possa ser reposta ou reutilizada num momento posterior.
Produto de Configuração	Um produto de configuração é um conjunto de componentes de configuração que integrados formam uma solução de software, que satisfaz o âmbito e funcionalidades definidas. Um produto de configuração é o resultado da actividade de engenharia aplicacional.
Release	Uma *release* representa uma versão do produto de configuração. Uma *release* é composta por um conjunto de *baselines*, sob a qual foram realizados testes de integração e de sistema, garantindo-se estabilidade da solução como um todo.
Linha de Configuração	Uma linha de configuração é um repositório físico de elementos de configuração. A inclusão de um *baseline* na linha de configuração implica a disponibilização física das versões dos elementos de configuração presentes no *baseline*. Durante a engenharia de domínio e engenharia aplicacional poderão ser criados várias linhas de configuração para segregar as diferentes configurações dos elementos de configuração do ambiente de trabalho dos diferentes intervenientes nos diferentes ambientes, e.g. integração, qualidade e produção. Tanto a LPS como cada produto de configuração da LPS deverá possuir uma linha de configuração, denominada "linha de configuração principal".
Projecto	Um projecto no âmbito de uma LPS corresponde à instanciação de um ou mais produtos de configuração da LPS no contexto de uma instalação específica. O desenvolvimento e evolução dos elementos base da LPS no contexto do plano de evolução da LPS, é considerado um projecto. Cada projecto possui uma linha de configuração principal e poderá incluir outras linhas de configuração secundárias, e.g. integração, desenvolvimento.

Tabela 28 – Elementos da Vista Produto

A vista produto pressupõe a possibilidade de representar as relações entre os seus elementos da seguinte forma:

Relação	Descrição
Relação "*é-parte-de*"	A relação "é parte de" descreve a decomposição das configurações de um produto em componentes de configuração e deste em elementos de configuração.
Relação "*versionamento*"	A gestão de configurações assenta numa relação de "*versionamento*" dos elementos de configuração. Os elementos *baseline* e *release* possuem uma relação de "*versionamento*" com os elementos de configuração de um componente de configuração, numa vista produto.

Tabela 29 – Relações entre elementos da Vista Produtos

A Figura 6 ilustra os elementos e respectivas relações da vista produtos:

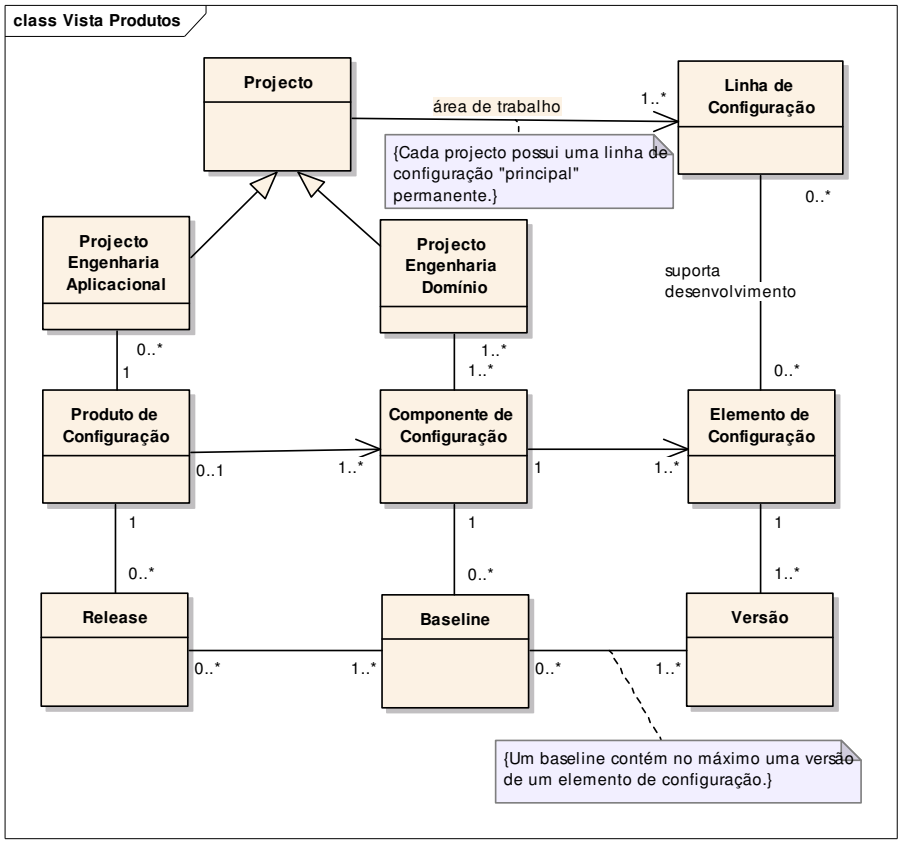


Figura 6 – Elementos e Relações entre elementos da Vista Produtos

IV.4.4.4.Atributos

Cada elemento possui um conjunto de metadados, i.e. atributos ou propriedades que identificam e classificam o módulo na arquitectura da LPS. A Tabela 30 identifica os principais atributos dos elementos da vista produtos.

Elemento	Atributo	Descrição
Elemento de Configuração	Nome	Nome do elemento de configuração
	Versão Actual	Versão actual do elemento de configuração
	Data Versão	Data em que foi criada a versão actual
	ID Componente	Identificador unívoco do componente ao qual pertence o elemento de configuração
Componente de Configuração	ID Componente	Identificador unívoco do componente de configuração
	Nome	Nome do componente de configuração
	ID Módulo	Identificador do(s) módulo(s) de software respectivo(s)
Produto de Configuração	ID Produto	Identificador de produto
	ID Componente	Lista de componentes de configuração do produto

Elemento	Atributo	Descrição
Projecto	ID Projecto	Identificador de Projecto
	Nome Projecto	Nome do Projecto
	ID Produtos	Lista de Produtos envolvidos no projecto
Baseline	ID *Baseline*	Identificador unívoco do *baseline*
	ID Componente	Identificador do componente de configuração alvo do *baseline*
	Data *Baseline*	Data do *baseline*
	Lista Elementos	Lista de elementos de configuração e respectivas versões alvo do *baseline*
	Tipo de *Baseline*	Tipo de *Baseline*: integração, qualidade, principal (i.e. realizado na linha de configuração principal)
Release	ID *Release*	Identificador da *release* do produto
	ID Produto	Identificador do produto alvo da *release*
	Lista *Baselines*	Lista de *Baselines* que constituem a *release*
	Data da *Release*	Data de criação da *release*
Linha de Configuração	ID Linha Configuração	Identificador unívoco da linha de configuração
	ID Projecto	Identificador do Projecto
	Tipo de Linha	Tipo de Linha: Principal, Integração ou Desenvolvimento.
	ID *Release*	Lista de *releases* dos produtos alvo.

Tabela 30 – Atributos dos elementos da Vista Produtos

IV.4.4.5.Notação

A vista produto não possui uma notação específica para a representação dos seus elementos e relação, ao nível da LPS ou em cada produto instanciado. A visualização dos elementos e relações entre os elementos desta vista dependerá essencialmente da ferramenta de gestão de configurações utilizada.

IV.4.4.6.Relação com outras vistas

A vista produto é transversal a todas as restantes vistas no modelo SPL-LIGTH, para a implementação e gestão de LPS. Os artefactos geridos pelas vistas de requisitos, módulo e C&C corresponderão a elementos de configuração na vista produto, sendo que sobre eles serão armazenados metadados (ver atributos em IV.4.4.4) que permitirão obter os artefactos e respectivas versões: a) por linha de configuração, b) por componentes de configuração, c) por elemento de configuração.

IV.5. MODELO SPL-LIGHT – RELAÇÃO ENTRE OS ELEMENTOS DAS VÁRIAS VISTAS

O processo de desenvolvimento de uma LPS suportado no modelo SPL-LIGHT pressupõe a correspondência entre os elementos das várias vistas. A Figura 7, ilustra a relação entre os elementos das várias vistas do modelo SPL-LIGHT. Em particular é visível a relação dos vários elementos com o elemento requisito.

A implementação das funcionalidades em módulos de software é identificada pela relação "implementação" na Figura 7. Por outro lado, caso exista necessidade de especificar o comportamento de um requisito, serão utilizados os casos de utilização. A modelação das interfaces entre componentes de execução é realizada com conectores que poderão implementar diferentes cenários de atributos de qualidade (e.g. interoperabilidade).

De forma transversal a todo o processo, a implementação destes elementos é materializada em vários ficheiros (artefactos) os quais são alvo de gestão de configurações. Os elementos de configuração, incluem a gestão do histórico de versões de um artefacto e estão enquadrados no contexto de um componente de configuração da LPS e produto de configuração instanciado a partir da LPS.

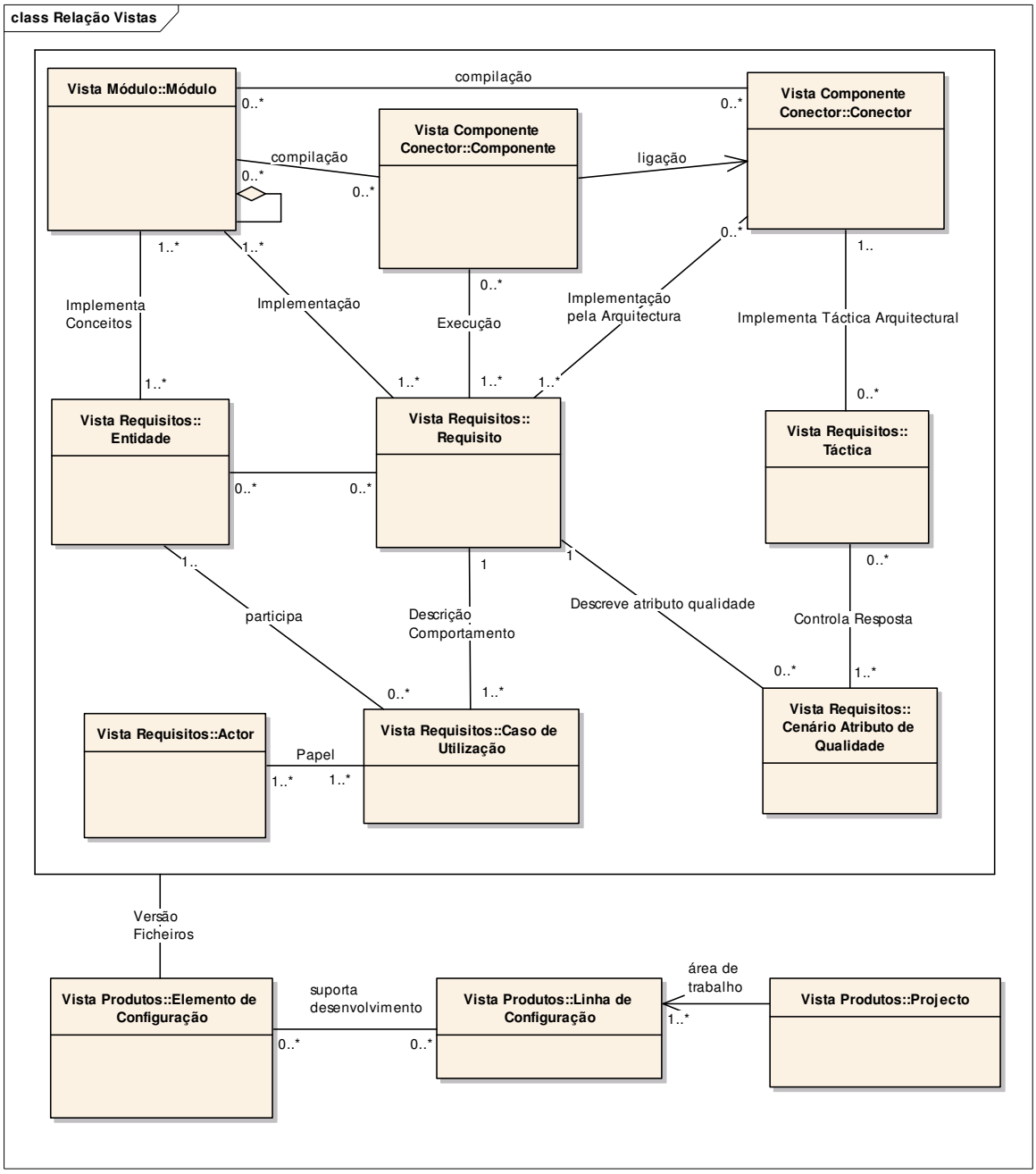


Figura 7 – Relação entre os elementos das vistas do modelo SPL-LIGHT

IV.6. PERFIL UML PARA REPRESENTAR VARIABILIDADES NUMA LPS

IV.6.1.Motivação

O UML é a linguagem padrão mais utilizada pela indústria, na representação da análise e desenho de software, contendo um conjunto de notações que permitem a modelação das diferentes perspectivas do sistema, e.g. casos de utilização, interacção, classes, componentes e instalação.

A abordagem de LPS para o desenvolvimento de software acarreta novos desafios à forma como as variabilidades são representadas e implementadas. No desenvolvimento tradicional de produtos, as variabilidades entre produtos semelhantes são identificadas e representadas após a implementação. Técnicas como a herança, limites de cardinalidade e classes abstractas permitem agilizar a representação desta necessidade.

Numa LPS, a inclusão das variabilidades é decidida à priori, i.e. no momento da concepção e durante o desenho dos elementos de software. Nesta abordagem ao desenvolvimento, é relevante a existência de uma representação única da arquitectura da LPS, independentemente do número de produtos que possam ser gerados. A especificação da superstrutura UML 2.0., apesar de conter uma maior aproximação à necessidade de documentação de uma arquitectura de software (ver II.2.2), possui lacunas ao nível da representação de variabilidades numa arquitectura de LPS.

Nesta secção é apresentado um perfil UML, contendo um conjunto de extensões ao metamodelo UML 2.0. [49], composto por estereótipos, etiquetas (*tagged values)* e constrangimentos necessários à representação das variabilidades nos modelos das vistas arquitecturais de uma LPS.

Em particular são apresentadas extensões a um conjunto de metaclasses UML utilizadas nos diagramas de requisitos, cenários de atributos de qualidade, modelo de domínio e diagrama de classes.

IV.6.2. Perfil UML: SPL-LIGHT

IV.6.2.1.Objectivo

O objectivo deste Perfil UML (*UML Profile*) é facilitar a representação das variabilidades na descrição da arquitectura de uma LPS.

Em concreto este perfil permite a representação dos elementos definidos nas vistas do modelo SPL-LIGHT (ver Figura 5) e sua extensão através de estereótipos e etiquetas (*tagged values*). A Figura 8, ilustra o modelo conceptual dos conceitos que compõem o modelo

IV.6.2.2.Modelo Conceptual

A Figura 8 identifica os principais conceitos presentes em cada um dos modelos do perfil UML proposto. Estes conceitos são extensões ao meta-modelo UML para representar as variabilidades numa LPS.

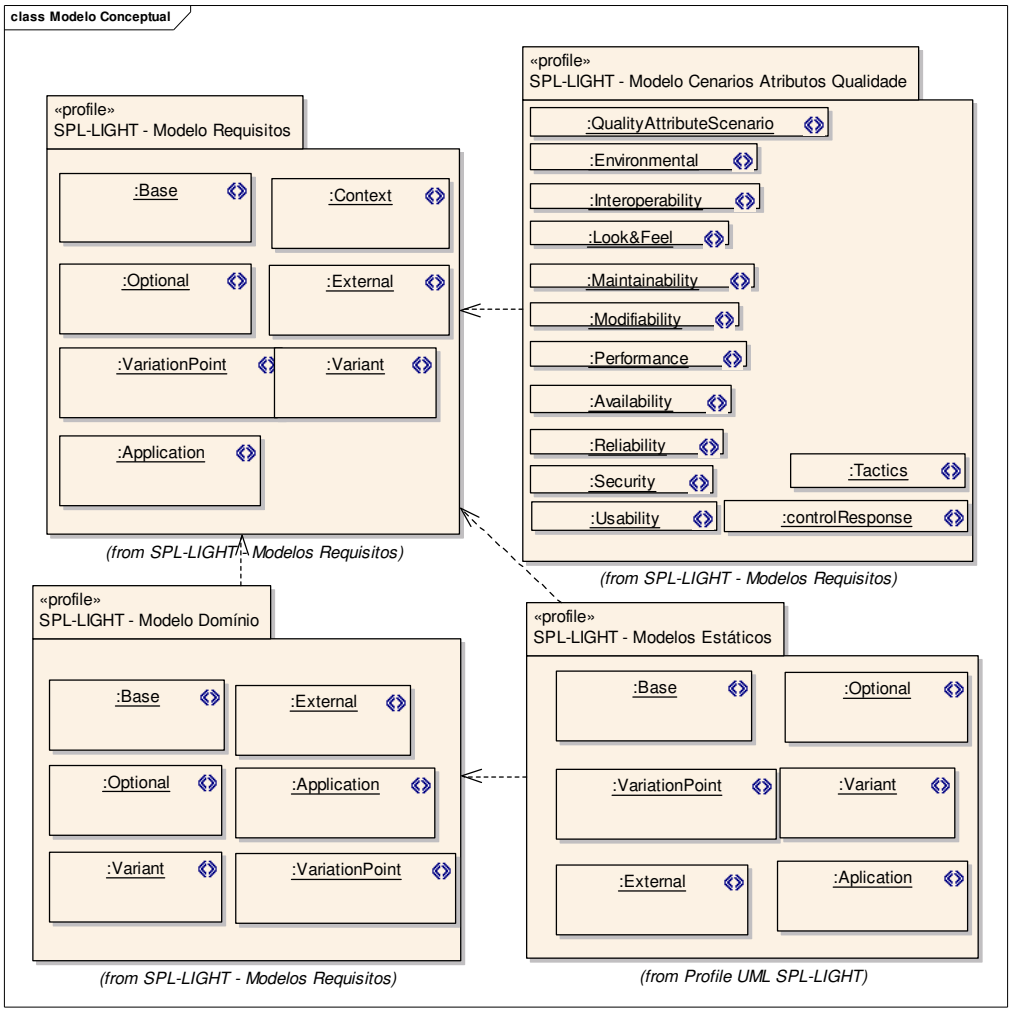


Figura 8 – Modelo conceptual do Perfil UML SPL-LIGHT

IV.6.2.3.Extensões ao Meta Modelo UML 2.0.

A Tabela 31 apresenta as extensões ao meta-modelo UML 2.0. utilizadas no perfil UML proposto, para cada elemento das vistas SPL-LIGHT.

Elemento das Vistas SPL-LIGHT	Metaclass UML 2.0	Estereótipos Perfil SPL-LIGHT
Vista de Requisitos		
Requisito	*Feature*	*Context, Mandatory, Optional, External, Application, Variation Point, Variant*
	Package	*Mandatory, Optional, External, Application*
Caso de Utilização	*UseCase*	
	Package	
Cenário Atributo de Qualidade	*Class*	*QualityAttributeScenario, Reliability, Security, Usability, Availability, Modifiability, Interoperability, Look&Fell, Environmental, Maintainability, Performance*
Táctica Arquitectura	*Class*	*Tactics*
Entidade	*Class*	*Mandatory, Optional, External, Application, Variation Point, Variant*
Vista Módulo		
Módulo	*Class*	*Mandatory, Optional, External, Application, Variation Point, Variant*
	Package	*Mandatory, Optional, External, Application*
Vista Componente & Conector		
Componente	*Component*	
Conector	*Association*	
	Class	

Tabela 31 – Extensões ao Meta-Modelo UML 2.0

IV.6.2.4.Modelos UML

De forma a facilitar a implementação de cada vista arquitectural, as extensões ao meta-modelo UML 2.0 são agrupadas em três tipos de modelos, tendo sido adicionado um tipo de modelo funcional, aos dois modelos padrão do UML: estáticos e dinâmicos. Assim, os modelos propostos são:

- Modelos de Requisitos, onde estão definidos os elementos dos diagramas de requisitos e modelo de domínio utilizados na vista requisitos (ver IV.4).
- Modelos Estáticos, onde estão definidos os elementos dos diagramas de pacotes e de classes da vista módulo (ver IV.4.2), e componentes da vista C&C (ver IV.4.3).

- Modelos Dinâmicos, onde estão definidos os elementos dos diagramas de interacção da vista C&C (ver IV.4.3)

As secções seguintes descrevem em detalhe os elementos e extensões UML no âmbito do perfil UML proposto, para cada um destes modelos.

IV.6.3.Modelos de Requisitos

Este modelo inclui os elementos necessários à concretização da vista de requisitos, nomeadamente na definição dos requisitos funcionais e atributos de qualidade do sistema. Neste contexto é apresentado um diagrama UML: Diagrama de Requisitos que inclui a representação dos requisitos e variabilidades respectivas para uma LPS.

A análise e descrição do comportamento em cada requisito funcional, é descrito por intermédio de casos de utilização.

IV.6.3.1.Diagrama de Requisitos

O modelo de requisitos representa os elementos e estereótipos que compõem o Perfil SPL-LIGHT, para a representação do contexto do problema e âmbito da LPS.

A Figura 9, ilustra os estereótipos e valores etiquetados que compõem o modelo de requisitos. São estendidas as metaclasses UML: *feature*, *package* para os elementos do modelo e as metaclasses UML: *composition*, *dependency* e *generalization* para as relações entre os elementos.

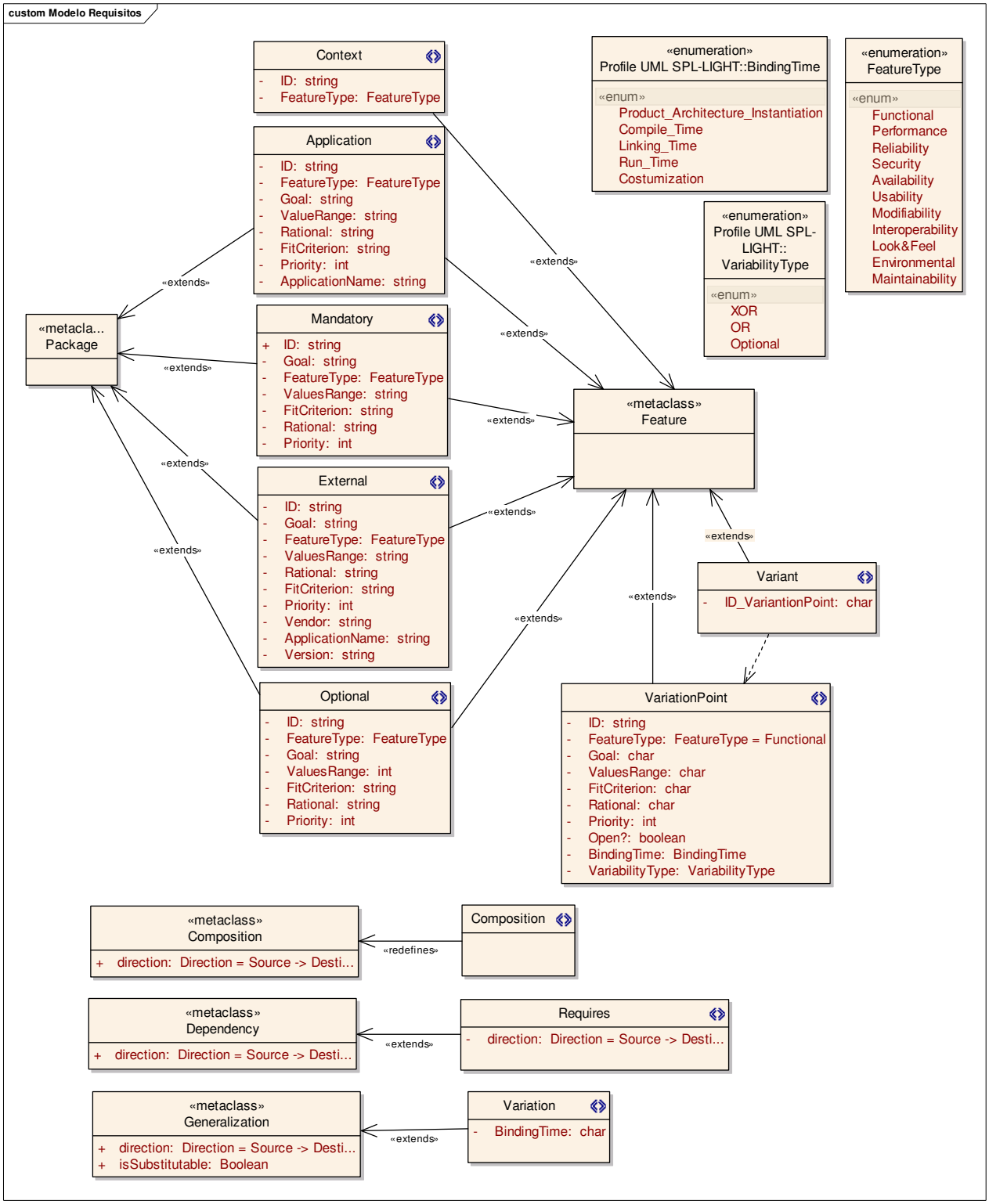
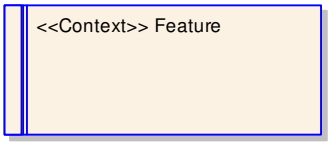
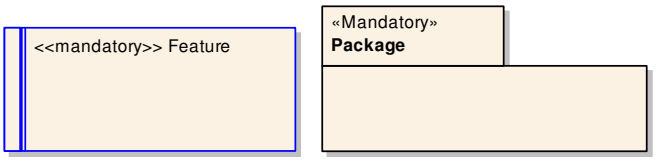


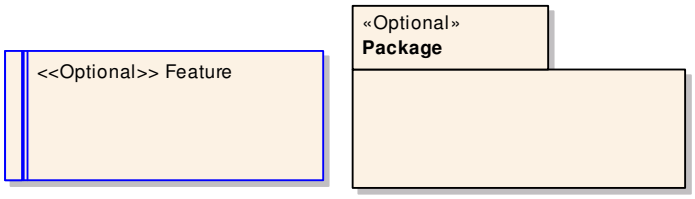
Figura 9 - SPL-LIGHT: Perfil UML para Modelo de Requisitos

Estereótipos do elemento Requisito da Vista de Requisitos:

Estereótipo	***Context***
Extensão Meta-Modelo UML	*Feature*
Semântica	Requisito *Context* representa o nó principal da árvore de funcionalidades. Este nó não tem implementação própria, sendo utilizado como contexto para a composição. O nome do requisito contexto é igual ao nome do pacote de funcionalidade que ele descreve.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista de Requisitos (ver Tabela 15).
Notação	
Limitações	Apenas poderá ser utilizado caso a árvore de funcionalidade tenha mais que uma folha. O nome do requisito de contexto é igual ao nome do pacote de funcionalidade que ele descreve.

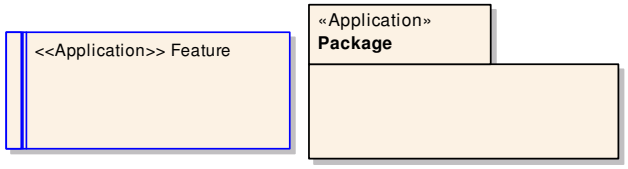
Estereótipo	***Mandatory***
Extensão Meta-Modelo UML	*Feature; Package*
Semântica	Requisito *Mandatory* na LPS. Este estereótipo é utilizado para identificar que um requisito que é comum a todos os produtos gerados na LPS. A alteração de um requisito *Mandatory* implica a alteração de código em todos os produtos resultantes.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista Requisitos (ver Tabela 15).
Notação	
Limitações	Entre requisitos apenas podem ser utilizadas as relações de composição e dependência.

Estereótipo	***Optional***
Extensão Meta-Modelo UML	*Feature; Package*
Semântica	Um requisito *optional* significa que a sua instanciação num produto da LPS é opcional, segundo os requisitos específicos.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista de Requisitos (ver Tabela 15).

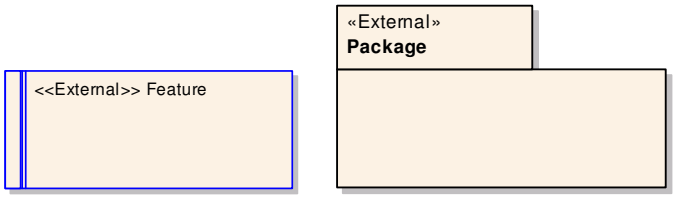
Notação	
Limitações	Um requisito só poderá ser opcional se não existir um requisito *mandatory* que dependa da sua implementação.

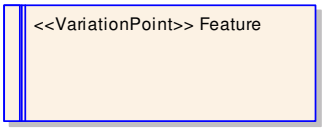
Estereótipo	***Application***
Extensão Meta-Modelo UML	*Feature; Package*
Semântica	A representação através de um requisito *application* deve ser utilizada quando os requisitos não pertencem à LPS, sendo específicos de produto em particular. A relevância deste estereótipo está na capacidade de diferenciar logo numa fase de especificação dos requisitos os requisitos que não farão parte da LPS, logo dispensam o cuidado particular no desenvolvimento dos elementos de software da LPS.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista Requisitos (ver Tabela 15). Este estereótipo inclui a seguinte etiqueta:

Nome	Tipo	Descrição
ApplicationName	*String*	Nome da Aplicação ou elemento de software.

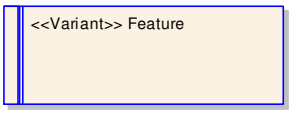
Notação	
Limitações	Nenhum requisito da LPS poderá depender de um requisito do tipo *aplicational*. Caso um requisito tenha dependência de uma funcionalidade aplicacional então esta deve ser transformada em requisito "*mandatory*" ou "*optional*".

Estereótipo	***External***
Extensão Meta-Modelo UML	*Feature; Package*
Semântica	Requisito *External* deve ser utilizado para representar um requisito do sistema, cuja implementação é externa à equipa de engenharia da LPS. Utilizado em casos onde são utilizados elementos de software desenvolvidos por uma entidade externa na LPS.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista Requisitos (ver Tabela 15). Este estereótipo inclui as seguintes etiquetas específicas:

	<table><tr><th>Nome</th><th>Tipo</th><th>Descrição</th></tr><tr><td>*ApplicationName*</td><td>*String*</td><td>Nome da Aplicação ou elemento de software.</td></tr><tr><td>*Vendor*</td><td>*String*</td><td>Fornecedor do elemento de software.</td></tr><tr><td>*Version*</td><td>*String*</td><td>Versão da aplicação externa utilizada.</td></tr></table>
Notação	
Limitações	

Estereótipo	***Variation Point***
Extensão Meta-Modelo UML	*Feature*
Semântica	Requisito *VariantionPoint* é utilizado para identificar os pontos de variabilidade da LPS. Sempre que numa LPS existirem várias alternativas de implementação de um requisito, este é representada como *VariationPoint*. A representação dos pontos de variabilidade é realizada pela relação explícita "*variation*", uma extensão da relação *generalization* UML.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Requisito" da Vista de Requisitos (ver Tabela 15). Este estereótipo inclui as seguintes etiquetas específicos: <table><tr><th>Nome</th><th>Tipo</th><th>Descrição</th></tr><tr><td>*Open?*</td><td>*Lógico*</td><td>O ponto de variabilidade está aberto a novos variantes?</td></tr><tr><td>*BindingTime*</td><td>*String*</td><td>Tempo de Ligação da variabilidade, e.g. derivação da arquitectura do produto, compilação, tempo de execução.</td></tr><tr><td>*VariabilityType*</td><td>*String*</td><td>Tipo de variabilidade: XOR (apenas poderá ser seleccionado um dos variantes), OR (poderá ser seleccionado pelo menos um variante).</td></tr></table>
Notação	
Limitações	Um *VariationPoint* tem de possuir no mínimo um variante.

Estereótipo	***Variant***
Extensão Meta-Modelo UML	*Feature*
Semântica	Requisito *Variant* é utilizada para identificar uma alternativa de variabilidade da LPS para um ponto de variabilidade em particular.

<table>
<tr><td></td><td>Um variante pode ser seleccionado ou não para ser instanciado na arquitectura de um produto em particular. Um Variant é uma generalização de um ponto de variabilidade.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Requisito” da Vista de Requisitos (ver Tabela 15). Este estereótipo inclui as seguintes etiquetas específicos:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>ID_VaritionPoint</td><td>ID</td><td>Identificador unívoco do ponto de variabilidade ao qual o variante pertence.</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>Um ponto de variabilidade pode conter vários variantes, no entanto um variante apenas pode pertencer a um ponto de variabilidade.
Um requisito “mandatory” não pode depender de um variante da LPS em particular pois em tempo de implementação este pode não se instanciado na arquitectura do produto.</td></tr>
</table>

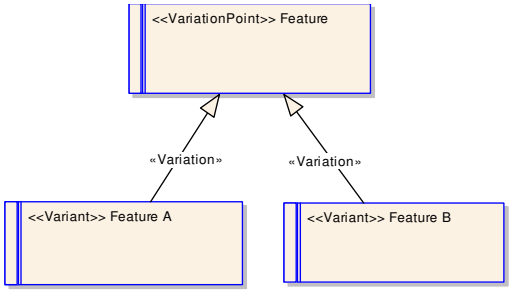
<table>
<tr><td>Estereótipo</td><td>Variantion</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Generalization</td></tr>
<tr><td>Semântica</td><td>A representação da relação entre os pontos de variabilidade e os seus variantes é realizada pela relação explícita “variation”, uma extensão da relação generalization UML.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todas as propriedades da meta-classe UML Generalization. Este estereótipo inclui a seguinte etiqueta específica:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>BindingTime</td><td>String</td><td>Tempo de Ligação da variabilidade, e.g. derivação da arquitectura do produto, compilação, tempo de execução.</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>A relação Variant tem sempre como origem um Variant e destino um VariationPoint. Numa LPS um variant apenas poderá possuir um variantionPoint e um VariationPoint tem obrigatoriamente possuir pelo menos um Variant.</td></tr>
</table>

Tabela 32 – Estereótipos para representação do elemento Módulo da Vista Módulo:

IV.6.3.2.Modelo de Domínio

O modelo de domínio representa os elementos e estereótipos que compõem o Perfil UML SPL-LIGHT, para a representação das principais entidades do domínio do problema no âmbito da LPS.

A Figura 10, ilustra os estereótipos e valores etiquetados que compõem o modelo de domínio. É estendida a metaclasse UML *class* para os elementos do modelo e as metaclasses UML *association e generalization* para as relações entre os elementos.

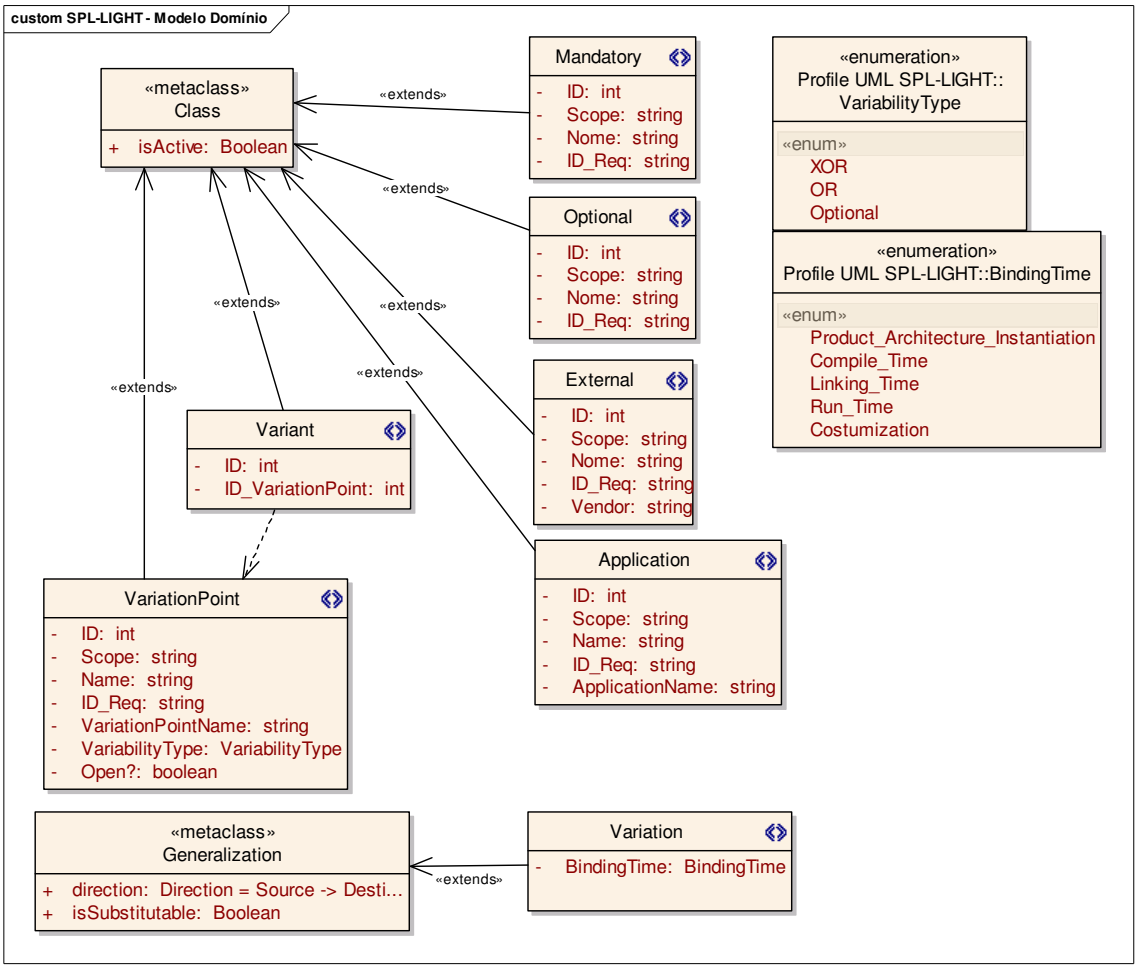
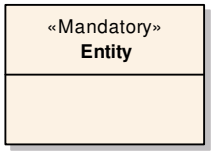
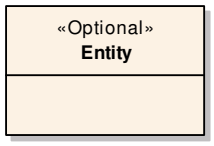
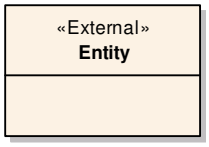


Figura 10 – SPL-LIGHT: Perfil UML para Modelo de Domínio

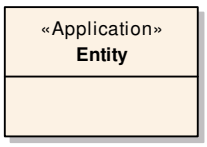
Estereótipos para representação do elemento Entidade do Modelo de Domínio:

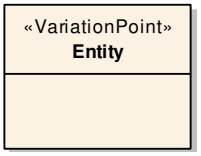
Estereótipo	***Mandatory***
Extensão Meta-Modelo UML	*Class*
Semântica	Uma entidade *Mandatory* é um conceito que pertence ao âmbito da LPS definido, comum em todas as instâncias da LPS. A implementação de uma entidade faz-se não só ao nível dos módulos de software mas também ao nível da estrutura de dados e do fluxo de informação.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Entidade" da Vista Requisitos (ver Tabela 15).
Notação	
Limitações	Uma entidade *Mandatory* não pode depender de uma entidade que não seja *Mandatory*.

Estereótipo	***Optional***
Extensão Meta-Modelo UML	*Class*
Semântica	Uma entidade *Optional* é um conceito da LPS que poderá pertencer ou não ao âmbito dos produtos instanciados da LPS, no contexto dos projectos específicos. A implementação de uma entidade do modelo de domínio opcional implica que todos os módulos e estrutura de dados associados possam ser desacoplados, sem prejuízo de disponibilidade ou robustez do sistema.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Entidade" da Vista Requisitos (ver Tabela 15).
Notação	
Limitações	Uma entidade *Optional* não possui dependência de uma entidade *Mandatory*.

<table>
<tr><td>Estereótipo</td><td>External</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class</td></tr>
<tr><td>Semântica</td><td>Uma entidade External é um conceito que embora pertença ao âmbito da LPS, a sua implementação será externa. Esta opção é muito frequente em elementos de software onde o grau de especialização é muito elevado e o esforço de desenvolvimento versus comprar não justifica o desenvolvimento interno.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Entidade” da Vista Requisitos (ver Tabela 15). Este estereótipo inclui a seguinte etiqueta:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>Supplier</td><td>Char</td><td>Nome do fornecedor ou representante do elemento de software</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>A comunicação entre as entidades External e as restantes entidades deverá ser explícita através de conectores que utilizem tecnologia padrão.</td></tr>
</table>

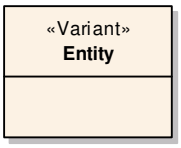
<table>
<tr><td>Estereótipo</td><td>Application</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class</td></tr>
<tr><td>Semântica</td><td>Uma entidade Application é conceito que embora esteja no contexto da LPS, foi adicionado para um projecto em particular e não faz parte do âmbito da LPS.

A promoção de uma entidade Application para ser incluída no âmbito da LPS implica que os elementos de software que a implementam sejam alvo de um processo de adequação à realidade de uma LPS (nomeadamente a possível inclusão de pontos de variabilidade e variantes).</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Entidade” da Vista Requisitos (ver Tabela 15). Este estereótipo inclui a seguinte etiqueta:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>ApplicationName</td><td>Char</td><td>Nome da aplicação de software desenvolvida especificamente.</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>Uma entidade Mandatory não poderá depender de uma entidade Application.</td></tr>
</table>

<table>
<tr><td>Estereótipo</td><td>Variation Point</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class</td></tr>
<tr><td>Semântica</td><td>Uma entidade Variation Point é utilizada para representar um ponto de variabilidade no âmbito da LPS. Na modelação da arquitectura da LPS estes pontos de variabilidade dos produtos são identificados e representados através deste estereótipo. Cada entidade Variation Point descreve as características do ponto de variabilidade, nomeadamente o tipo de variação, e.g. XOR, OR, e se o ponto de variabilidade ainda se encontra aberto a novos variantes.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Entidade” da Vista Requisitos (ver Tabela 15). Este estereótipo inclui as seguintes etiquetas:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>Open?</td><td>Boolean</td><td>Identifica se o ponto de variabilidade encontra-se aberto a novos variantes ou se pelo contrário já possui o domínio de variantes definido.</td></tr>
<tr><td>VariationType</td><td>String</td><td>Tipo de variabilidade: XOR (apenas poderá ser seleccionado um dos variantes), OR (poderá ser seleccionado pelo menos um variante).</td></tr>
<tr><td>VariationPointName</td><td>String</td><td>Nome do ponto de variabilidade que será estendido pelos variantes.</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>Uma entidade VariationPoint deve incluir no mínimo um variante.</td></tr>
</table>

<table>
<tr><td>Estereótipo</td><td>Variant</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class</td></tr>
<tr><td>Semântica</td><td>Uma entidade Variant representa uma variação particular do ponto de variabilidade da LPS.

Um variante poderá ser opcional e/ou conjugado com outro(s) variantes(s) para cada instanciação de produto da LPS, seguindo o tipo de variabilidade definida pelo ponto de variabilidade.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Entidade” da Vista Requisitos (ver Tabela 15). Este estereótipo inclui a seguinte etiqueta:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>ID_VariationPoint</td><td>Int</td><td>Identificador unívoco do ponto de variabilidade ao qual o variante pertence</td></tr>
</table></td></tr>
</table>

Notação	
Limitações	Uma entidade *Variant* pertence a uma única entidade *Variation Point*, sendo o ponto de variabilidade identificado pelo Identificador unívoco da entidade *VariationPoint*.

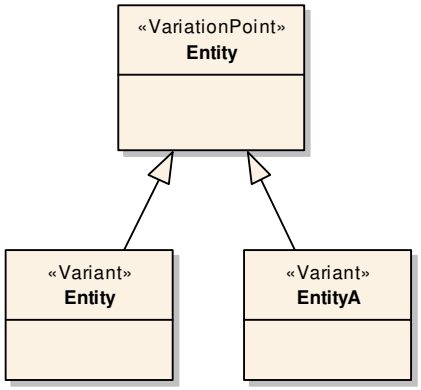
Estereótipo	***Variantion***
Extenção Meta-Modelo UML	*Generalization*
Semântica	A representação da relação entre os pontos de variabilidade e os seus variantes é realizada pela relação explícita *"variation"*, uma extensão da relação *generalization* UML.
Propriedades	Este estereótipo inclui todas as propriedades da meta-classe UML *Generalization*. Este estereótipo inclui a seguinte etiqueta específica: Nome: *BindingTime* — Tipo: *String* — Descrição: Tempo de Ligação da variabilidade, e.g. derivação da arquitectura do produto, compilação, tempo de execução.
Notação	
Limitações	A relação *Variant* tem sempre como origem um *Variant* e destino um *VariationPoint*. Numa LPS um *variant* apenas poderá possuir um *variantionPoint* e um *VariationPoint* tem obrigatoriamente possuir pelo menos um *Variant*.

Tabela 33 – Estereótipos para representação do elemento Entidade do Modelo de Domínio

IV.6.4.Modelos Estáticos

O modelo estático representa os elementos e estereótipos que compõem o Perfil UML SPL-LIGHT, para a representação da arquitectura de implementação dos elementos de software de uma LPS.

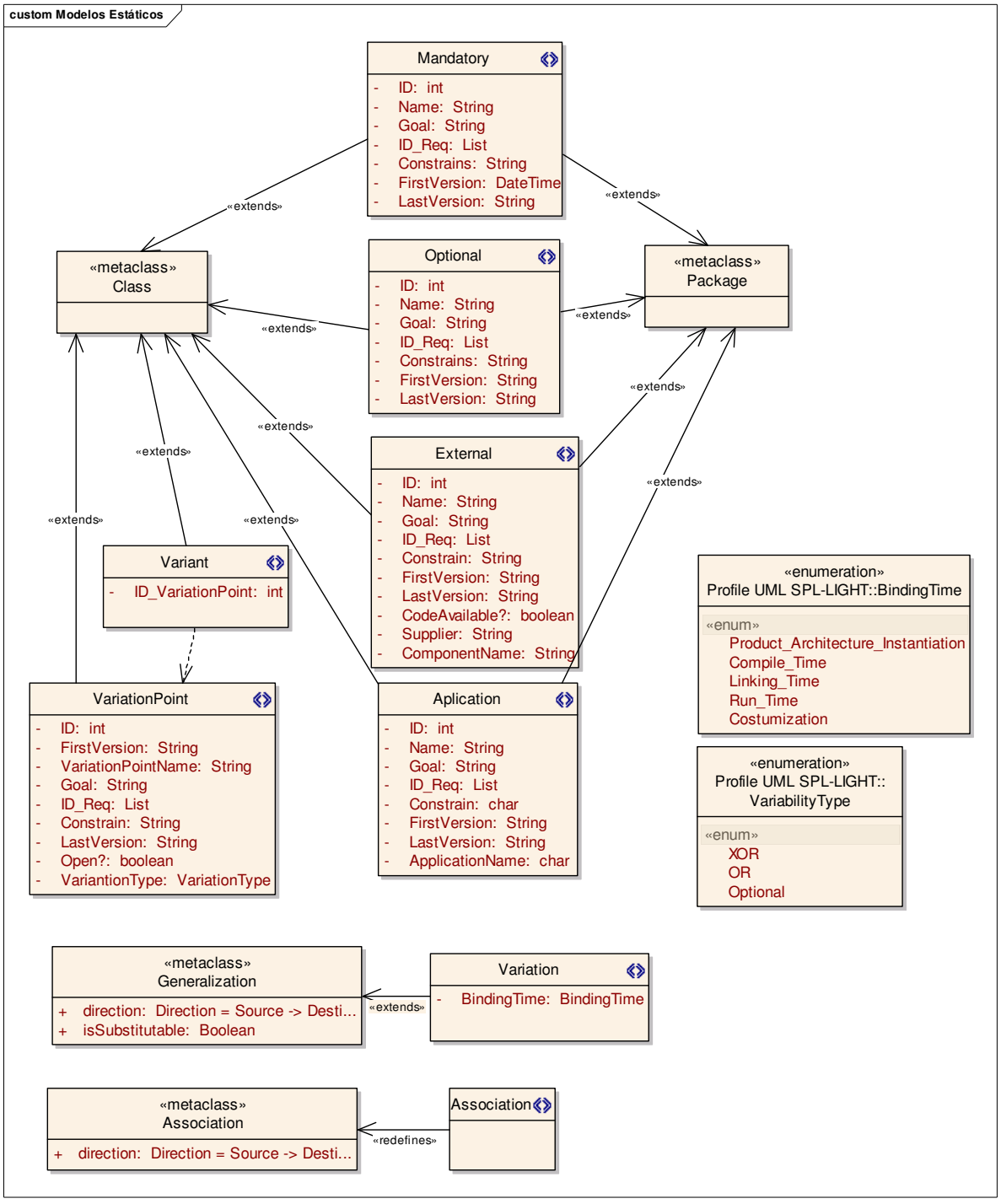
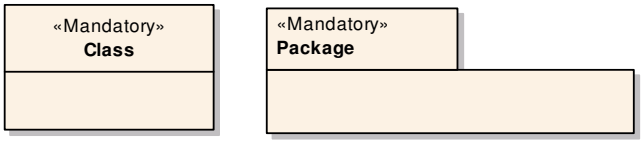
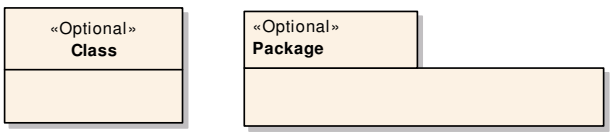


Figura 11 – SPL-LIGHT: Perfil UML para Modelo Estático

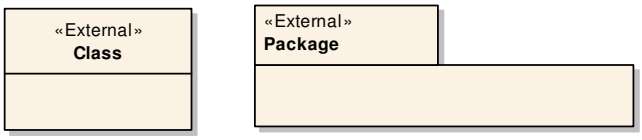
As tabelas seguintes ilustram os estereótipos e valores etiquetados que compõem o modelo estático, nomeadamente para a representação dos diagramas de classes. São estendidas as metaclasses UML: *class* e *package* para os elementos do modelo e a metaclasse UML: *generalization* para as relações de variabilidade entre o ponto de variabilidade e os variantes respectivos.

Estereótipos para representação do elemento Módulo da Vista Módulo:

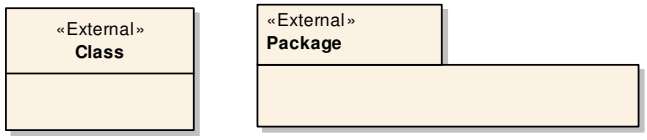
Estereótipo	***Mandatory***
Extensão Meta-Modelo UML	*Class e Package*
Semântica	Uma classe ou pacote *Mandatory* é um módulo de software que pertence a um elemento base da LPS e que implementa parcialmente ou totalmente uma ou mais funcionalidades. Qualquer alteração a uma classe Mandatory terá impacto na necessária actualização em todos os produtos derivados da LPS. Um pacote *Mandatory* corresponde a um elemento base da LPS que pode ser desenvolvido e testado de forma isolada. A implementação de um elemento de software base deverá respeitar os padrões de desenho e arquitectura definidos para a LPS.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Módulo" da Vista Módulo (ver Tabela 22).
Notação	
Limitações	Uma classe ou pacote *Mandatory* não pode depender de uma classe ou pacote que não seja *Mandatory*.

Estereótipo	***Optional***
Extensão Meta-Modelo UML	*Class e Package*
Semântica	Uma classe ou pacote *Optional* é um módulo de software que pertence a um elemento base da LPS e que implementa parcialmente ou totalmente uma ou mais funcionalidades. No entanto, a sua implementação é opcional para os produtos instanciados a partir da LPS. Na implementação de um elemento de software opcional da LPS deve ser garantido que este não gera dependência para qualquer outro elemento base.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Módulo" da Vista Módulo (ver Tabela 22).
Notação	
Limitações	Uma classe ou pacote *Optional* não pode gerar dependência de uma classe ou pacote *Mandatory*.

<table>
<tr><td>Estereótipo</td><td>External</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class e Package</td></tr>
<tr><td>Semântica</td><td>Uma classe ou pacote External é um módulo de software desenvolvido externamente à equipa de engenharia da LPS e cuja evolução não é controlada por esta. No entanto este elemento de software implementa parcialmente ou totalmente uma ou mais funcionalidades.

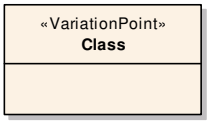
O módulo de software pode ser disponibilizado em código fonte ou apenas os binários ou componentes de execução. No segundo caso, a representação do diagrama de classes detalhado não faz sentido. Dependendo da relação com a entidade externa e adequação do código fonte, os elementos de software poderão ser promovidos a elementos base da LPS, eventualmente através da aquisição de uma licença ou direitos de acesso ao código fonte.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Módulo” da Vista Módulo (ver Tabela 22). Este estereótipo inclui as seguintes etiquetas específicas:
<table>
<tr><th>Nome</th><th>Tipo</th><th>Descrição</th></tr>
<tr><td>ComponentName</td><td>Char</td><td>Nome do elemento de software externo</td></tr>
<tr><td>Supplier</td><td>Char</td><td>Nome do fornecedor ou representante do módulo de software</td></tr>
<tr><td>CodeAvailable?</td><td>Boolean</td><td>O código fonte está disponível ou apenas os componentes de execução?</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>Os módulos External não incluem a representação de variabilidade, tal como os elementos base da LPS.</td></tr>
</table>

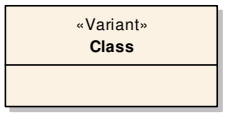
<table>
<tr><td>Estereótipo</td><td>Application</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class e Package</td></tr>
<tr><td>Semântica</td><td>Uma classe ou pacote Application é um módulo de software desenvolvido especificamente para satisfazer os requisitos particulares de um produto, não fazendo parte da LPS. Neste caso, a implementação dos módulos de software dispensa a conformidade com os padrões e procedimentos obrigatórios para o desenvolvimento dos elementos base da LPS.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Módulo” da Vista Módulo (ver Tabela 22). Este estereótipo inclui as seguintes etiquetas específicas:
<table>
<tr><th>Nome</th><th>Tipo</th><th>Descrição</th></tr>
<tr><td>ApplicationName</td><td>Char</td><td>Nome da aplicação de software desenvolvida especificamente.</td></tr>
</table></td></tr>
</table>

Notação	
Limitações	Os módulos *Application* não incluem a representação de variabilidade, tal como os elementos base da LPS, sendo cada funcionalidade representada de forma isolada. Um módulo *Mandatory* da LPS não poderá depender da implementação de um módulo *Application*.

Estereótipo	***Variation Point***
Extensão Meta-Modelo UML	*Class*
Semântica	Um módulo *Variation Point* é utilizado para representar um ponto de variabilidade particular da LPS. Na modelação da arquitectura da LPS estes pontos de variabilidade dos produtos são identificados e representados através deste estereótipo. Cada módulo *Variation Point* descreve as características do ponto de variabilidade, nomeadamente o tipo de variabilidade, se todas as possibilidades já se encontram representadas ou o tempo em que os variantes são instanciados em cada produto resultante da LPS.
Propriedades	Este estereótipo inclui todos os atributos do elemento "Módulo" da Vista Módulo (ver Tabela 22). Este estereótipo inclui as seguintes etiquetas específicas:

Nome	Tipo	Descrição
Open?	*Boolean*	Identifica se o ponto de variabilidade encontra-se aberto a novos variantes ou se pelo contrário já possui o domínio de variantes definido.
VariationType	*String*	Tipo de variabilidade: XOR (apenas poderá ser seleccionado um dos variantes), OR (poderá ser seleccionado pelo menos um variante).
VariationPointName	*String*	Nome do ponto de variabilidade que será estendido pelos variantes.

Notação	
Limitações	Um módulo *Variation Point* deve incluir no mínimo um variante.

<table>
<tr><td>Estereótipo</td><td>Variant</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Class</td></tr>
<tr><td>Semântica</td><td>Um módulo Variant representa uma variação particular do ponto de variabilidade da LPS. Um variante poderá ser opcional e/ou conjugado com outro(s) variantes(s) para cada instanciação de produto da LPS, seguindo o tipo de variabilidade definida pelo ponto de variabilidade.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todos os atributos do elemento “Módulo” da Vista Módulo (ver Tabela 22). Este estereótipo inclui as seguintes etiquetas específicas:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>ID_VariationPoint</td><td>Int</td><td>Identificador unívoco do ponto de variabilidade ao qual o variante pertence</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>Um módulo Variant pertence a um único módulo Variation Point.</td></tr>
</table>

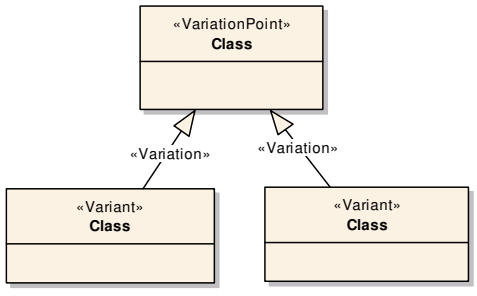
<table>
<tr><td>Estereótipo</td><td>Variantion</td></tr>
<tr><td>Extensão
Meta-Modelo UML</td><td>Generalization</td></tr>
<tr><td>Semântica</td><td>A representação da relação entre os pontos de variabilidade e os seus variantes é realizada pela relação explícita “variation”, uma extensão da relação generalization UML.</td></tr>
<tr><td>Propriedades</td><td>Este estereótipo inclui todas as propriedades da meta-classe UML Generalization. Este estereótipo inclui a seguinte etiqueta específica:
<table>
<tr><td>Nome</td><td>Tipo</td><td>Descrição</td></tr>
<tr><td>BindingTime</td><td>String</td><td>Tempo de Ligação da variabilidade, e.g. derivação da arquitectura do produto, compilação, tempo de execução.</td></tr>
</table></td></tr>
<tr><td>Notação</td><td> 
</td></tr>
<tr><td>Limitações</td><td>A relação Variant tem sempre como origem um Variant e destino um VariationPoint. Numa LPS um variant apenas poderá possuir um variantionPoint e um VariationPoint tem obrigatoriamente possuir pelo menos um Variant.</td></tr>
</table>

Tabela 34 – Estereótipos para representação do elemento Módulo da Vista Módulo:

IV.7. CONSIDERAÇÕES FINAIS

O modelo SPL-LIGHT proposto permite a documentação da arquitectura uma LPS, de diferentes perspectivas satisfazendo as necessidades de diferentes interlocutores na engenharia de domínio e aplicacional, em especial, para os utilizadores chave, analistas funcionais e demais intervenientes envolvidos na definição do contexto do problema.

O ênfase do modelo na vista requisitos e descrição do problema permitirá numa segunda fase (contexto da solução), os elementos da equipa de engenharia e integradores de produtos herdarem informação preciosa do âmbito, objectivos e comportamento expectável do sistema, bem como da informação se um determinado elemento a implementar é base da LPS ou específico de um projecto em particular. Este último aspecto é relevante na medida em que o desenvolvimento de um elemento base necessita cumprir com os padrões de desenho e testes definidos para a LPS, ao passo que um desenvolvimento específico não exige este esforço adicional.

A documentação das tácticas arquitecturais que satisfazem os atributos de qualidade previstos e que poderão ocorrer no âmbito de cada projecto, permite antever as características necessárias para o ambiente de execução. A implementação de atributos de qualidade poderá ter impacto na selecção dos variantes que deverão ser instanciados ao nível dos conectores existentes para componentes de execução, e.g. diferentes tipos de interorperabiliade.

A documentação da LPS só terá valor se a sua actualização acompanhar a evolução dos desenvolvimentos realizados sobre os artefactos que constituem o produto, e.g. código fonte, ficheiros de configuração, *makefiles* e documentação de análise e desenho. A vista produtos apresenta uma perspectiva de gestão de configurações de software multi-produto que permite também os modelos da arquitectura possam ser instanciados em cada projecto e tal como o código fonte e restantes elementos de implementação, alvo de gestão de versões. Por exemplo, a evolução das versões do modelo de requisitos e código fonte para um determinado elemento de software, pode em qualquer momento ser analisada face ao mesmo modelo e código fonte pertencente á linha de configuração de um outro projecto ou na linha de configuração principal da LPS. O mesmo se aplica a qualquer outro artefacto gerado no âmbito da LPS.

V. CASO PRÁTICO – APLICAÇÃO DA METODOLOGIA À LPS ARQUO™

Tal como descrito na secção contexto do problema (I.3), a metodologia proposta neste trabalho será aplicada à LPS ARQUO™, desenvolvida pela empresa de engenharia de software TIE, Lda.

A TIE tomou a decisão estratégica de adaptar uma abordagem ao desenvolvimento de software baseado numa LPS. O resultado deste trabalho permitirá dotar a organização de uma metodologia para a implementação e gestão da LPS bem como um modelo para a definição e gestão da arquitectura da LPS.

O ARQUO™ é uma marca registada, pelo que não será realizada uma representação completa da arquitectura. Não obstante, são apresentados vários exemplos que permitem ilustrar a aplicabilidade do modelo de representação da arquitectura SPL-LIGHT no contexto da LPS ARQUO™.

Este capítulo pretende ilustrar um caso prático de implementação de metodologia LPS, tendo como exemplo a linha de produtos de software ARQUO™.

V.1. A LINHA DE PRODUTOS DE SOFTWARE ARQUO™

O ARQUO™ é um sistema integrado de captura e arquivo óptico de documentos, gestão documental e processos (*workflow*). O arquivo digital permite armazenar qualquer documento ou objecto em formato digital, nomeadamente imagens, música, vídeos, textos, e-mail, etc.

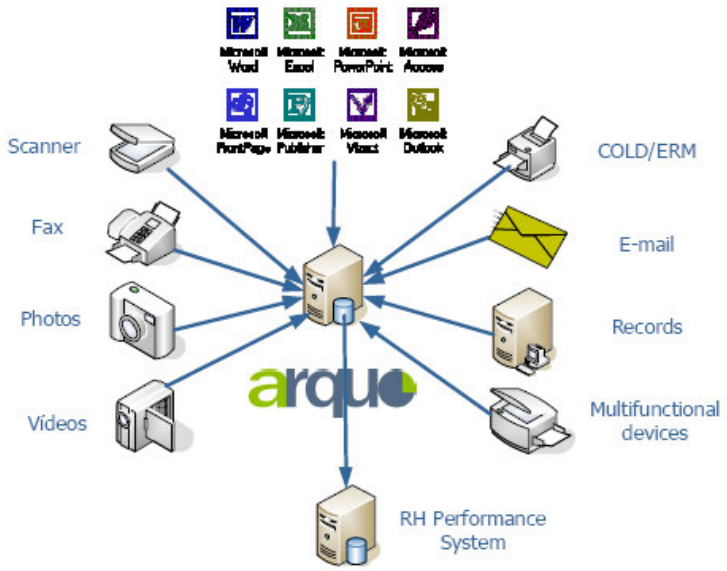


Figura 12 – Sistema ARQUO™: Solução de Arquivo e Gestão Documental Global

A arquitectura do sistema foi desenhada tendo como objectivos a garantida de desempenho, escalabilidade e capacidade para a importação, arquivo e consulta de um grande volume de conteúdos e documentos ópticos, num reduzido espaço de tempo.

O sistema disponibiliza uma API de comunicação através dos protocolos de comunicação CORBA, SOAP (*Web Services*) e RMI, sendo através dela que as interfaces do utilizador ou aplicações externas, comunicam com o sistema.

A Figura 13, ilustra os principais blocos funcionais e técnicos do sistema ARQUO™ actual.

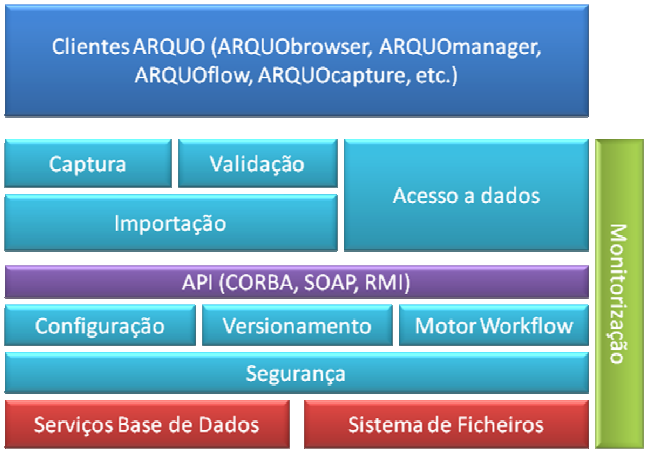


Figura 13 – Sistema ARQUO™ - Principais Blocos funcionais e técnicos

O sistema ARQUO™ evoluiu ao longo da última década, tendo sido adicionadas funcionalidades específicas em cada implementação, em clientes dos vários sectores de negócio.

O Problema:

Verificou-se que a abordagem existente ao desenvolvimento de software não satisfazia as necessidades presentes e desafios futuros da organização, essencialmente pelas seguintes razões:

- A gestão das configurações e *releases* multi-produto e reutilização de elementos de software tornou-se complexa.
- Necessidade de redução dos custos e esforço dispendido em engenharia aplicacional (implementação e manutenção dos produtos instalados) e foco da equipa de engenharia no desenvolvimento de novas funcionalidades, seguindo o plano de evolução do produto.
- A necessidade de internacionalizar o produto em outros mercados, reutilizando as funcionalidades do sistema, mas incluindo necessidades específicas.

A Solução:

Adopção de uma abordagem ao desenvolvimento baseado em LPS, adequada à realidade de uma PME, tendo como base os produtos existentes.

V.2. ADOPÇÃO DA ABORDAGEM LPS

Na adopção da abordagem LPS, foi seguida a metodologia proposta em III. e instanciadas as actividades da metodologia, adequadas a satisfazer as necessidades da implementação de uma LPS na organização.

Nesta secção, mais do que mostrar a documentação da arquitectura da LPS (descrita em V.3) é focado o processo de adopção da abordagem e actividades de análise da arquitectura e refactorização dos produtos existentes com vista à concepção da arquitectura intermédia.

V.2.1.Estrutura Orgânica e Papeis

Antes da adopção pelo desenvolvimento baseado em LPS, e estrutura orgânica da TIE estava organizada por uma equipa técnica em que todos os recursos eram afectos a projectos de engenharia aplicacional, conforme as necessidades.

Estrutura Orgânica e Papeis (Antes Adopção LPS)

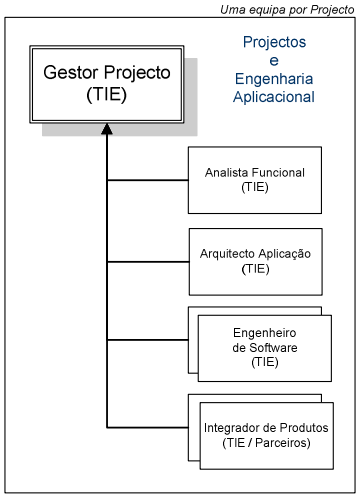


Figura 14 – Estrutura Orgânica e Papeis antes adopção LPS

Estrutura Orgânica e Papeis (Após Adopção LPS)

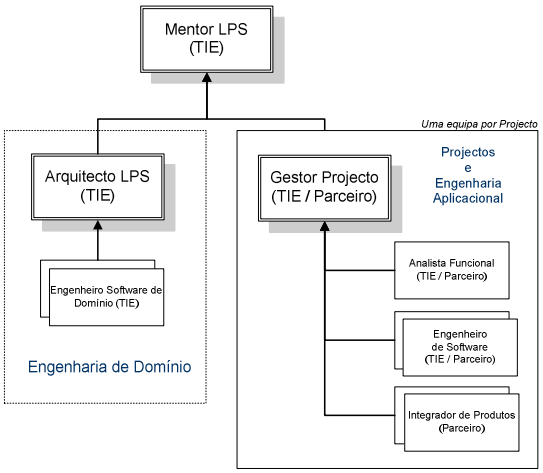


Figura 15 – Estrutura Orgânica e Papeis após adopção LPS

A adopção de uma abordagem LPS permite reduzir o esforço de evolução dos vários produtos instalados, o qual até ao momento era multiplicado pelo número de produtos existentes. Com uma LPS e definição de elementos base e variabiliades o

esforço está concentrado em menos pessoas, permitindo a definição e execução de um *roadmap* para a LPS, em paralelo com a actividade de engenharia aplicacional.

V.2.2.Tomada de Decisão

A tomada de decisão na adopção de uma abordagem LPS teve por base a realização das actividades da metodologia: "Avaliação das Arquitecturas dos Produtos Existentes" e "Análise de Viabilidade".

V.2.2.1.Avaliação das Arquitecturas dos Produtos Existentes

Passo 1. Análise da arquitectura técnica dos produtos existentes

A Tabela 35 apresenta uma lista dos produtos de software desenvolvidos e instalados em diversos clientes.

Produtos	Arquitectura Técnica Tipo
ARQUO	Arquivo Documental Óptico, incluindo a importação, armazenamento, validação e controlo de acessos.
Config App	Ferramenta de gestão da configuração dos documentos do ARQUO.
ArchiveApp	Ferramenta de administração e configuração das importações
Export App	Aplicação responsável pela geração da informação a exportar para os aderentes
VolumeApp	Ferramenta de administração dos volumes
FileMover	Ferramenta que permite mover os contentores para a sua localização definitiva
QualityControl	Ferramenta que permite efectuar o controlo de qualidade das imagens e dos contentores
eArquo	Interface de pesquisa e visualização do conteúdo do arquivo, permitindo efectuar pesquisas sobre a informação de indexação e visualizar documentos.
ARQUOManager	Ferramenta de administração e configuração do ARQUO.
ARQUOMonitor	Interface de monitorização do ARQUO a qual permite visualizar a informação referente à validação das actividades.
WebServices	Interface web service do ARQUO (SOAP)

Tabela 35 – ARQUO™: Produtos existentes antes da adopção abordagem LPS

Passo 2. Análise dos elementos de software comuns e específicos.

A Tabela 36 ilustra a implementação dos produtos existentes anteriormente à adopção pela LPS, para os cinco principais clientes à data.

De forma a satisfazer as necessidades específicas dos clientes, dos produtos acima descritos existem alguns elementos de software que foram desenvolvidos especificamente para um cliente em particular. Por outro lado no produto principal ARQUO™ as versões existentes em diferentes clientes tiverem uma evolução

independente ao longo do tempo o que faz com que estas tenham muitas funcionalidades comuns, mas algumas específicas.

Pela análise da Tabela 36, verifica-se que os produtos *ARQUO, ConfigApp, ArchiveApp, eArquo, ARQUOmanager e WebServices* estão presentes em mais que uma instalação, logo são candidatos para poderem ser incluídos no âmbito da LPS.

Produtos	Arquitectura Técnica Tipo	Cliente1	Cliente2	Cliente3	Cliente4	Cliente5
ARQUO	*JAVA, CORBA, C++*	✔	✔	✔	✔	✔
Config App	*C++, CORBA*	✔	✔	✔	✔	✔
Archive App	*C++, CORBA*	✔		✔	✔	
VolumeApp	*C++, CORBA*	✔	✔	✔		
ExportApp	*JAVA, CORBA*					✔
QualityControl	*JAVA, CORBA*	✔				
FileMover	*C++, CORBA*	✔				
eArquo	*JSP, JAVASCRIPT, JAVA*	✔	✔		✔	✔
ARQUOManager	*JSP, JAVASCRIPT, JAVA*	✔	✔		✔	
ARQUOMonitor	*JSP, JAVASCRIPT, JAVA*	✔				
WebServices	*JAVA, SOAP, CORBA*	✔	✔		✔	

Tabela 36 – ARQUO™: Instalação dos diferentes produtos para os cinco principais clientes à data.

O produto ARQUO™ inclui um conjunto vasto de funcionalidades pelo que carece de uma análise mais detalhada aos seus elementos de software, de forma a aferir dos aspectos comuns e dos específicos para clientes em particular. Este trabalho é um primeiro passo na construção da arquitectura intermédia e refactorização dos elementos de software.

A Tabela 37 ilustra uma análise aos elementos de software do produto ARQUO™ para a funcionalidade de importação no ARQUO™ (o trabalho para os outros elementos de software é idêntico).

Pela análise da Tabela 37, podemos verificar que em diferentes instalações, a mesma funcionalidade (importação de documentos) está implementada, utilizando distintos elementos de software e contendo diversas modificações de forma a satisfazer requisitos de importação específicos dos clientes, e.g. formatos ficheiros, índices, automatização da importação. Estes elementos de software são candidatos a refactorização no âmbito da implementação da LPS.

Funcionaliade	Elemento Base	Descrição	Produto	Dependências	Tecnologia
Importação	*DocImport Server*	Importa documentos e a sua informação de indexação para o ARQUO	Cliente1 (Específico) Cliente2 Cliente4 Cliente5 (Comum)	*SecurityServer* *ConfigServer* *ValidationServer* *Base de Dados*	*C++* *CORBA*
	ImportServer	Importa documentos e a sua informação de indexação para o ARQUO através da aplicação ArchiveApp	Cliente3 (Específico)	*SecurityServer* *ConfigServer* *ValidationServer* *Base de Dados*	*C++* *CORBA*
	AutoFileServer	Importação automática no ARQUO Pesquisa nas directorias de importação por ficheiros a importar	Cliente1 Cliente5 (Específico) Cliente2 Cliente4 (Comum)	*SecurityServer* *ConfigServer* *DocImportServer*	*C++* *CORBA*
	StagingServer	Gestão das *Stages* Move os contentores de *staging* para localização definitiva	Cliente1 (Específico)	*Base de Dados*	*JAVA* *CORBA*

Tabela 37 – ARQUO™: Elementos de Software do produto ARQUO, para as funcionalidades de Acesso a Dados e Importação.

Passo 3. Identificação de padrões de arquitectura e desenho comuns.

A arquitectura do ARQUO™ é orientada a serviços (*Service Oriented Architecture – SOA*) e permite a gestão e evolução autónoma dos elementos de software que a constituem. As interfaces entre os elementos de software do produto ARQUO estão descritas através de IDL e são expostos através de interfaces CORBA.

Ao nível dos produtos com interface com o utilizador existem soluções web (*eArquo, ArquoManager*) e soluções *win32* (*ConfigApp, ArchiveApp, ExportApp, FileMover*).

Passo 4. Avaliação do esforço de refactorização dos elementos de software comuns *versus* reengenharia.

Pela análise dos vários produtos e respectivos elementos de software verifica-se que ao nível do produto ARQUO™, é desejável a realização de refactorização dos seus elementos base, essencialmente porque:

- As funcionalidades existentes satisfazem grande parte das as necessidades dos clientes
- A arquitectura SOA adequa-se à realização da refactorização dos elementos de software, pois as dependências entre estes estão bem definidas e os elementos de software são auto-contidos.

Ao nível das aplicações com interfaces para o utilizador é desejável a realização de uma reengenharia. As principais razões para esta decisão são:

- Necessidade de inclusão de novas funcionalidades
- Limitações da tecnologia utilizada (Win32) em algumas aplicações.
- Necessidade de uniformização da imagem do produto pelas várias interfaces com o utilizador.

V.2.2.2.Análise de Viabilidade

Passo 1. Análise da viabilidade na implementação de uma abordagem de desenvolvimento baseada em LPS.

A decisão de avançar para a implementação desta abordagem tem como principais fundamentos:

- Agilizar a gestão das configurações e *releases* multi-produto e reutilização de elementos de software, à data muito complexa.
- Necessidade de internacionalizar o produto em mercados onde as principais funcionalidades do sistema são reutilizáveis, existindo no entanto variabilidades que deverão ser endereçadas, e.g. língua, legislação.
- Diminuir o tempo de resposta a solicitações de mercado.
- Optimização do tempo de engenharia aplicacional, aumentando a competitividade em termos de preço e prazos de projecto.

Passo 2. Análise da viabilidade de refactorização dos elementos de software dos produtos existentes

A análise da arquitectura dos produtos existentes permite aferir da viabilidade da refactorização dos elementos de software do produto ARQUO.

As aplicações com interfaces com utilizador, serão alvo de reengenharia.

V.2.3.Definição da Arquitectura de Software Intermédia

A arquitectura intermédia ARQUO™ será o resultado do trabalho de consolidação dos elementos de software das arquitecturas dos vários produtos existentes. Será com base nesta arquitectura que será definida a arquitectura de software da LPS, incluindo as novas funcionalidades segundo o plano de evolução para o produto.

V.2.3.1.Consolidação de Requisitos dos Produtos

Modelo de Requisitos Intermédio

A Tabela 38 resume os principais grupos de requisitos satisfeitos pelos produtos existentes e instalados nos principais clientes.

Grupos de Requisitos	Descrição	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Cliente 5
Arquivo Documental	Arquivo de documentos e objectos associados.	✔	✔	✔	✔	✔
Configuração e Administração do Arquivo	Gestão dos tipos de documentos, índices respectivos, utilizadores, perfis e contentores de objectos.	✔	✔	✔	✔	✔
Consulta de Documentos	Consulta dos índices e objectos dos documentos arquivados	✔	✔	✔	✔	✔
Importação de Documentos	Importação e classificação dos índices e objectos associados.	✔	✔	✔	✔	✔
Captura e Indexação de Documentos	Digitalização, reconhecimento e captura automática de índices de tipos de documentos.	✔			✔	
Gestão de Clientes e Subscrições	Gestão das fichas de clientes e subscrições por tipos de documentos e temas de interesse.					✔
Compensação de Cheques	Gestão e compensação de cheques interbancários.				✔	
Exportação de Documentos	Exportação de documentos a pedido ou em lote.					✔

Tabela 38 – Implementação dos grupos de requisitos nos cinco principais clientes à data.

Pela análise dos produtos existentes e segundo o modelo de requisitos definido na metodologia SPL-LIGHT (ver IV.4.1), foram tomadas as seguintes decisões:

- Os grupos de requisitos "*Compensação de Cheques*", "*Gestão de Clientes e Subscrições*" foram considerados requisitos "*Application*", por terem sido desenvolvidos para satisfazer necessidades específicas dos clientes, ficando fora do âmbito da LPS;
- O grupo de requisitos "*Captura e Indexação de Documentos*" foi classificado como "*External*" pois é desenvolvido por uma entidade externa à TIE;
- Os grupos de requisitos "*Exportação de Documentos*" e Captura e Indexação de Documentos foram considerados no âmbito da LPS, no entanto poderão ocorrer ou não nos produtos instanciados da LPS, logo são classificados como "*Optional*";
- Os grupos de requisitos "*Arquivo Documental*", "*Importação de Documentos*", "*Configuração e Administração ARQUO*" e "*Consulta de Documentos*" foram classificados como "*Mandatory*" pois fazem parte de todos os produtos instalados ARQUO™;

A Figura 16, apresenta um modelo de requisitos "*alto-nível*" consolidado face aos requisitos satisfeitos por cada um dos produtos existentes.

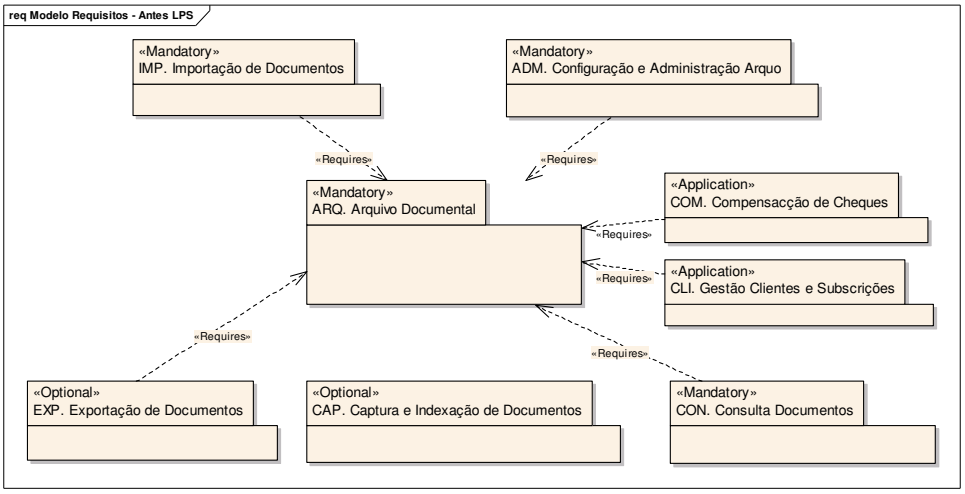


Figura 16 – Modelo de Requisitos Consolidado "*Alto-Nível*" para os produtos existentes ARQUO

O levantamento detalhado dos requisitos para cada um dos pacotes de requisitos "*alto-nível*" é necessário para suportar o trabalho de refactorização necessário.

Na Figura 17, é apresentado um levantamento dos requisitos para o requisito "*alto-nível*" de importação de documentos.

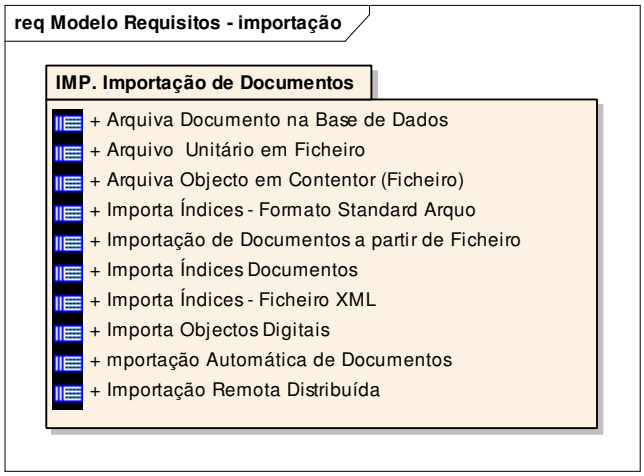


Figura 17 – Requisitos cobertos pelos produtos existentes no âmbito da Importação de Documentos

Modelo de Domínio Intermédio

O levantamento dos conceitos e entidades do domínio, com base nos produtos existentes permite a definição de um modelo de domínio consolidado para os vários produtos. Nesta fase é possível identificar desde já as entidades que serão obrigatórias, opcionais, aplicacionais, externas ou variantes na LPS futura. A Figura 18, ilustra o modelo de domínio da arquitectura intermédia da LPS.

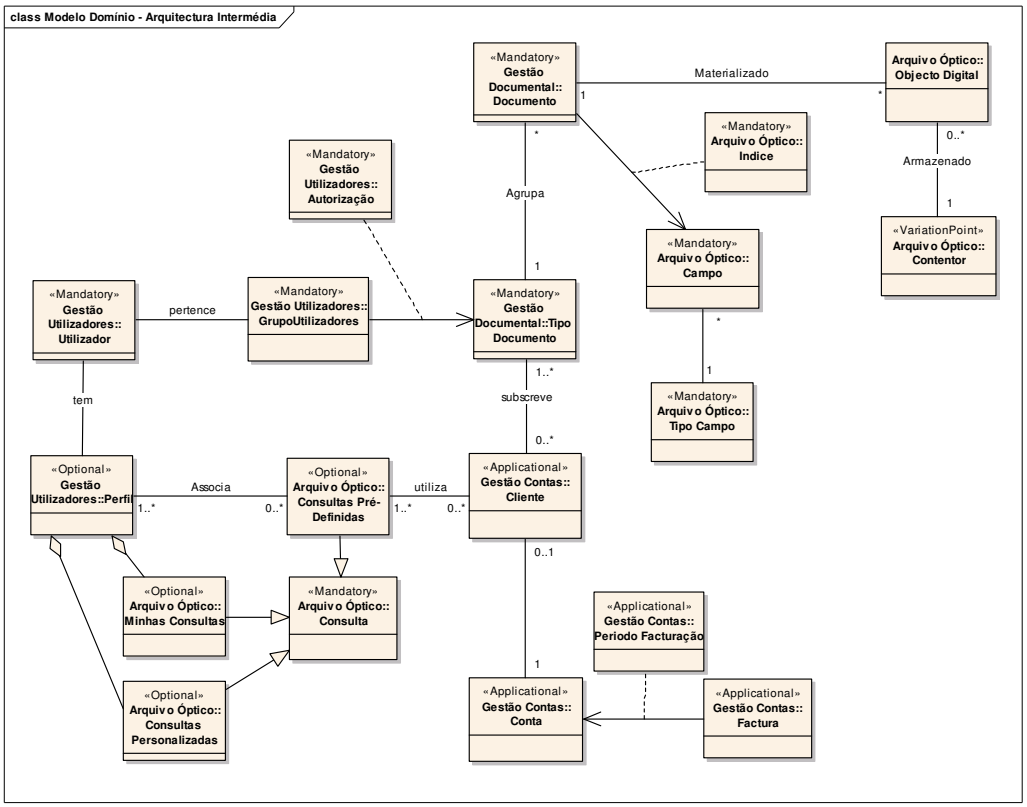


Figura 18 – Modelo de Domínio Consolidado para os produtos existentes ARQUO

V.2.3.2.Refactorização de Elementos de Software

Modelo de Classes Intermédio

Pela análise da arquitectura dos produtos existentes, concluiu-se que os elementos de software que compõem o produto ARQUO™, deverão ser alvo de uma refactorização de forma a poderem ser reutilizados numa abordagem de LPS.

A título de exemplo, iremos apresentar o trabalho de refactorização necessário para transformar o elemento de software "*Importação de Documentos*" o qual possui diferentes implementações nos produtos existentes, num elemento de software base, no âmbito de uma LPS.

A Figura 19 ilustra os principais módulos de software (Vista Módulo) que implementam a funcionalidade "*Importação de Documentos*" nos vários produtos existentes. Na figura, são visíveis as diferentes implementações e extensões existentes, motivadas pelo desenvolvimento de requisitos específicos. São exemplos as classes *CFileSession, CFileParser e CDocImport Server*.

O trabalho de refactorização pretende identificar pontos de variabilidade e variantes para cada um dos elementos de software candidatos a serem elementos de software

base da LPS, de forma a poder definir uma estrutura única, reutilizável para todos os produtos no futuro.

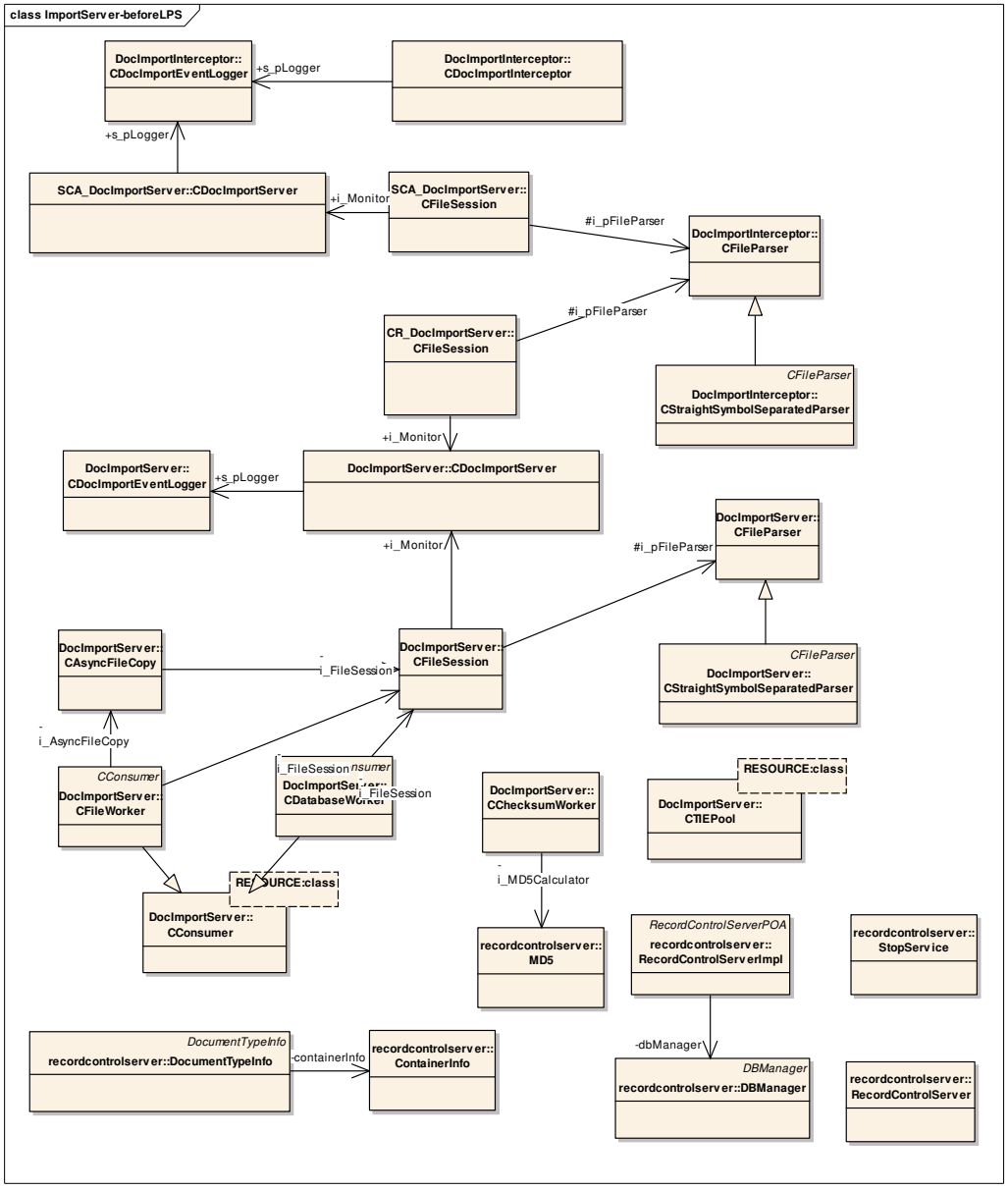


Figura 19 – Diagrama de Classes, antes da refactorização à lógica de uma LPS, para o módulo "*Importação de Documentos*"

Na Figura 20, estão apresentados os módulos, após a refactorização. Para os módulos que contenham pontos de variabilidade com diferentes possibilidades de implementação, estes são apresentados como variantes. A relação entre um *VariationPoint* e um *Variant* é uma relação de generalização, tal como definido no modelo SPL.LIGHT.

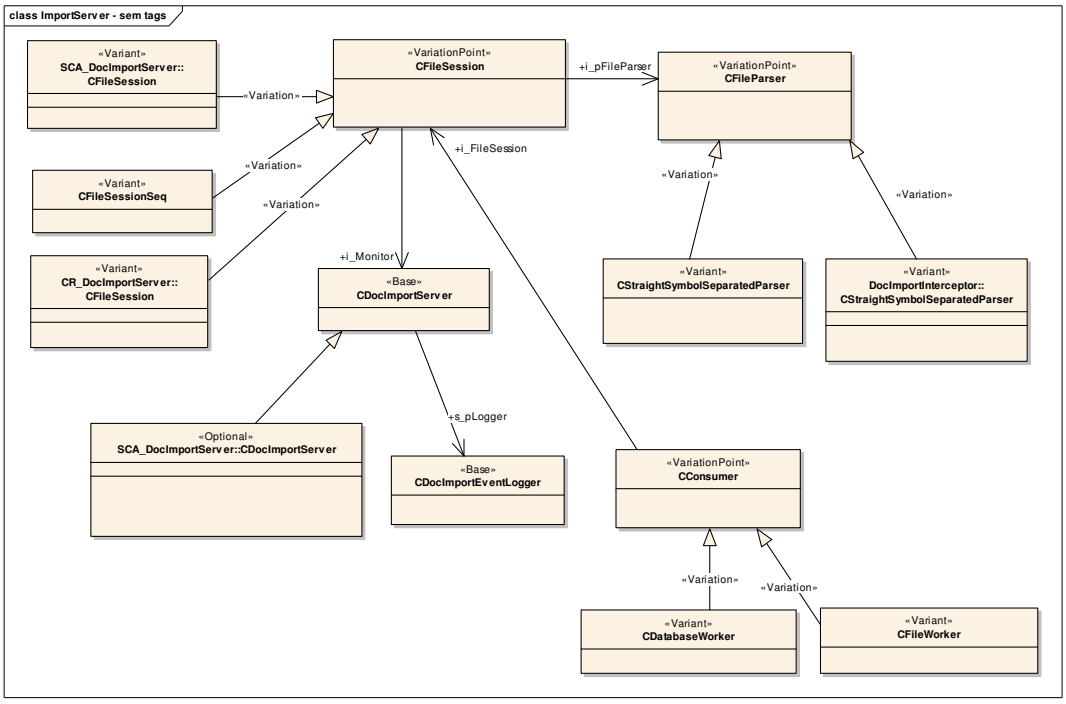


Figura 20 – Diagrama de Classes utilizando o Perfil UML SPL-LIGHT para elemento de software: "*Importação de Documentos*"

A introdução dos estereótipos "*variation point*" e "*variant*", permite ao engenheiro de software da LPS poder adicionar ou alterar funcionalidades de um variante, sem que com isto tenha que alterar os módulos que implementam os restantes variantes, logo sem impacto nos produtos já instalados.

V.2.3.3.Definição da Arquitectura Intermédia

A análise dos produtos existentes e refactorização dos elementos de software tem como objectivo a definição de uma arquitectura única partilhada pelos vários produtos existentes. A arquitectura intermédia será a base para a definição da arquitectura da LPS, garantindo assim uma migração progressiva em duas fases dos produtos existentes: 1) migração para a arquitectura intermédia resultante da refactorização dos elementos de software dos vários produtos existentes e 2) migração para a arquitectura da LPS.

Após a análise e avaliação dos produtos existentes, consubstanciada pela definição do modelo de requisitos, modelo de domínio e diagramas de classes consolidados, foram definidos os principais produtos que irão constituir a LPS ARQUO™ bem como a relação com os produtos existentes.

Durante a refactorização dos elementos de software poderá existir a necessidade de representar o comportamento dos requisitos funcionais (utilizando casos de utilização) para melhor perceber os cenários de interacção e identificar possíveis

pontos de variabilidade. Esta análise é semelhante à realizada no âmbito da engenharia de domínio apresentada na secção V.3.1.2.

Produtos ARQUO™ Existentes	Produto LPS ARQUO™	Descrição	Técnica de Desenvolvimento
ARQUO	*ARQUO ArchivingSystem*	Contém os módulos funcionais principais do sistema de arquivo documental: importação, consulta, exportação de metadados e objectos associados aos documentos digitais. Possui ainda vários elementos de software transversais auxiliares, e.g. parsing de ficheiros, stagging contentores de objectos e gestão de offline.	Refactorização dos elementos base do produto ARQUO
Config App	*ARQUOManager*	Solução Web de suporte à configuração e administração da solução ARQUO. Possui a definição de utilizadores, autorizações, configurações de documentos, monitorização SNMP, e gestão de repositório de objectos.	Reengenharia
ARQUOManager			
ARQUOMonitor			
VolumeApp			
eArquo	*ARQUOBrowser*	Solução Web de consulta e tipos de documentos arquivados no ARQUO.	Reengenharia
WebServices	*WebServices*	Interface web service do ARQUO (SOAP)	Refactorização
Archive App	Fora âmbito LPS		
ExportApp	Fora âmbito LPS		
QualityControl	Fora âmbito LPS		
FileMover	Fora âmbito LPS		

Tabela 39 – Estratégia para a evolução da LPS ARQUO™ a partir dos produtos existentes.

V.2.4.Definição da Arquitectura da LPS

A definição da arquitectura da LPS tem como base a arquitectura intermédia definida, com base nos produtos existentes. Além disso, esta actividade inclui a definição dos elementos de software que permitirão que a arquitectura satisfaça o âmbito futuro da LPS, segundo o plano de evolução definido.

A definição da arquitectura da LPS é uma actividade contínua que acompanha as actividades de engenharia de domínio e engenharia aplicacional, suportada pelos procedimentos e processos de gestão definidos na categoria de gestão do modelo SPL-LIGHT, e.g. gestão de configurações de software, planeamento técnico, optimização do processo de desenvolvimento.

A definição e evolução da arquitectura da LPS ARQUO™ são suportadas pelo modelo definido em IV. Nesse sentido, na secção V.3 é apresentada a documentação da arquitectura de software da LPS ARQUO™.

V.3. DESCRIÇÃO DA ARQUITECTURA DA LPS ARQUO™

A descrição da arquitectura de software da LPS ARQUO apresentada nesta secção segue o modelo de representação de uma arquitectura LPS proposto em IV.

Este modelo suporta a descrição da arquitectura, desde a tomada de decisão na adopção de uma LPS, reengenharia e refactorização necessária para a definição da arquitectura da LPS com base em produtos existentes e suporte à engenharia de domínio e engenharia aplicacional.

V.3.1.Vista de Requisitos

A vista de requisitos consiste na representação do contexto do problema e âmbito da LPS. Neste contexto, segundo o modelo SPL-LIGHT a representação desta vista é complementada pelos seguintes diagramas:

- Modelo de requisitos: Apresentação dos requisitos funcionais e não funcionais do sistema. A representação dos requisitos funcionais é realizada através de uma árvore de requisitos funcionais (funcionalidades). Os requisitos não funcionais não têm representação em árvore e são representados ao longo dos nós da árvore funcional do sistema. E.g. sistema como um todo, sub-sistema ou funcionalidade particular.
- Modelo de casos de utilização, onde para cada requisito funcional é representado o comportamento e cenários de interacção dos utilizadores com o sistema.
- Modelo de cenários de atributos de qualidade, para a especificação dos estímulos / resposta expectável do sistema no contexto de um requisito não funcional.
- Modelo de domínio, onde são expressas as entidades principais do domínio da LPS e respectivas relações.

V.3.1.1.Modelo de Requisitos

A Figura 21, é o diagrama de contexto da LPS ARQUO, representado através de um diagrama de pacotes UML e ilustra os principais módulos do sistema ARQUO.

Na representação dos requisitos "*alto-nível*" da LPS são utilizados os diagramas de pacotes UML, tal como definido no modelo SPL-LIGHT. Um pacote agrupa um conjunto de requisitos (funcionais e não funcionais) associados a um módulo principal ou subsistema na LPS.

A relação de dependência entre os pacotes é representada com a relação "requires", o que traduz que um determinado pacote de requisitos depende da implementação de outro para poder ser implementado.

A Figura 21 é uma evolução do modelo da arquitectura intermédia (inclui o grupo de requisitos "*Gestão Documental e Workflow*". Ilustra os principais blocos funcionais da LPS, identificando os que são obrigatórios (estereotipo "*mandatory*"), opcionais (estereotipo "*optional*") e externos (estereotipo "*external*") na LPS e ainda os pacotes aplicacionais implementados para satisfazer necessidades específicas e que não pertencem à LPS.

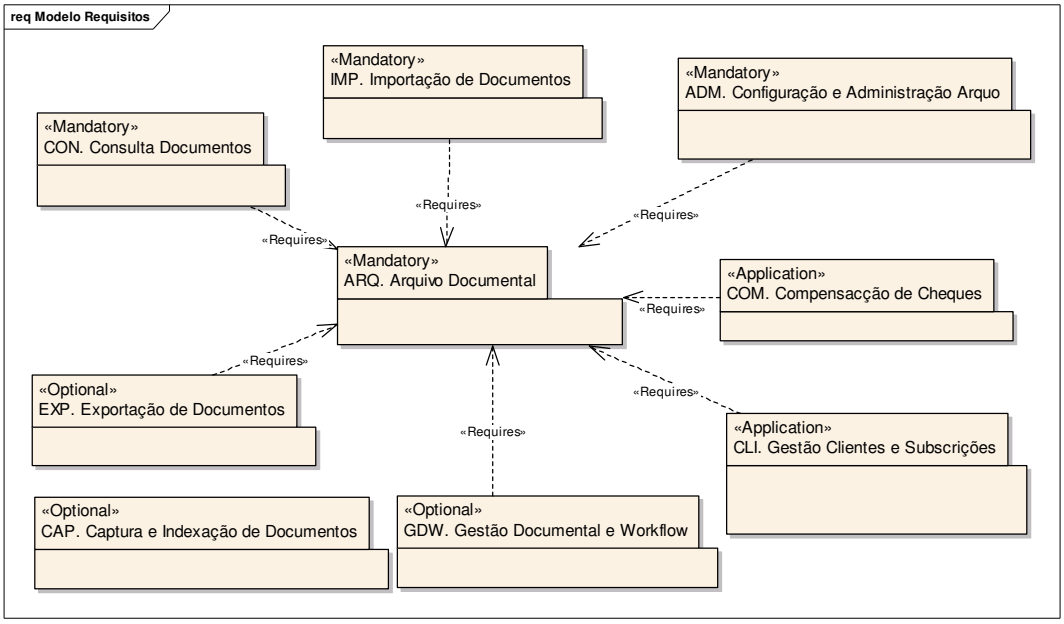


Figura 21 – Modelo de Requisitos: Módulos Principais sistema ARQUO

Pela análise da figura poderemos concluir que:

- Todas as instâncias resultantes da engenharia aplicacional no âmbito da LPS, conterão obrigatoriamente requisitos dos grupos "*Importação de Documentos*", "*Arquivo Documental*", "*Consulta de Documentos*" e "*Administração e Configuração do ARQUO*";
- Poderão existir produtos resultantes da engenharia aplicacional que não incluam requisitos de "*Exportação de Documentos*", "*Gestão Documental*" e "*Workflow e Captura e Indexação de Documentos*";
- À excepção dos requisitos de "*Captura e Indexação de Documentos*", todos os restantes grupos de requisitos dependem dos requisitos definidos em "*Arquivo Documental*";
- Os grupos de requisitos "*Compensação de Cheques*" e "*Gestão de Clientes e Subscrições*" foram resultado de engenharia aplicacional e não fazem parte do âmbito da LPS.

Exemplo 1: Importação de Documentos

De forma a exemplificar a aplicabilidade do modelo de requisitos, nomeadamente na decomposição dos requisitos funcionais e não funcionais, irá ser representado um dos subsistemas da LPS ARQUO™: "*Importação de Documentos*".

A Figura 22, ilustra a árvore de requisitos incluídos no grupo de requisitos "*Importação de Documentos*", identificado como "*IMP*" no modelo. Na figura são visíveis os estereótipos instanciados para cada um dos requisitos, no âmbito de uma LPS.

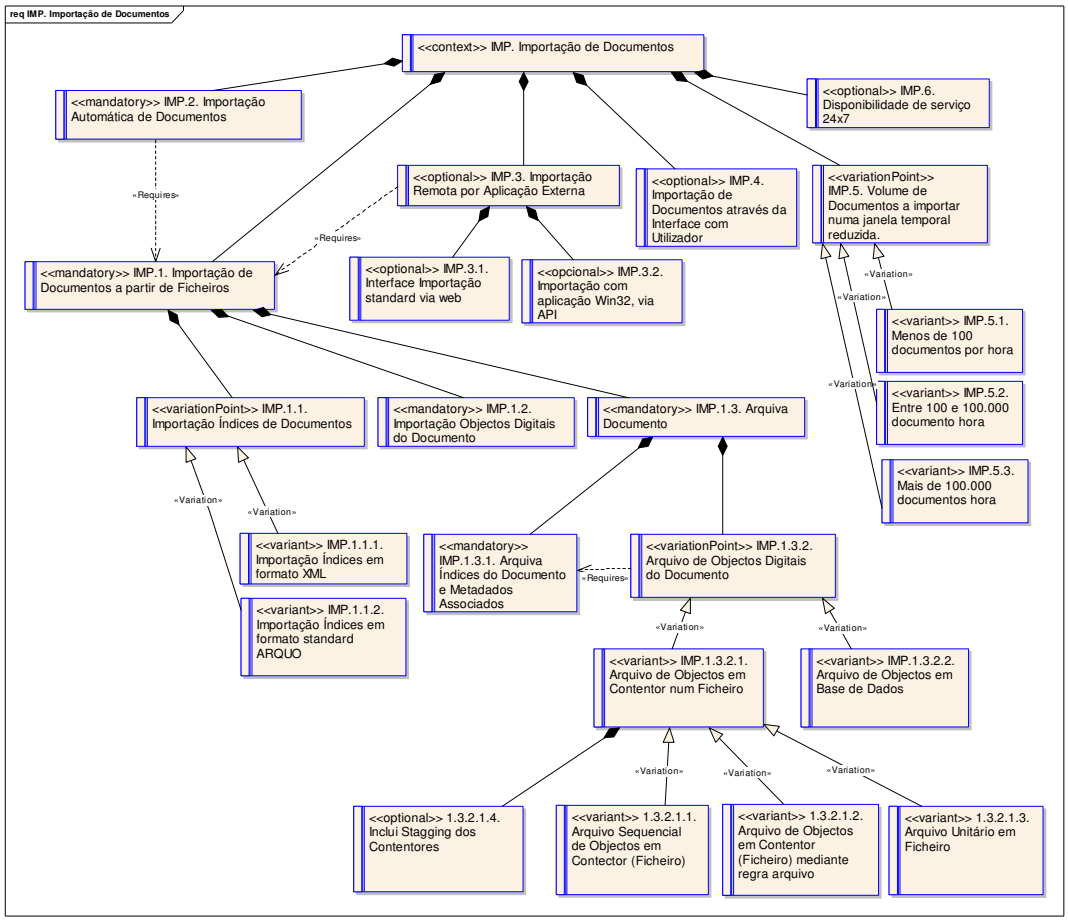


Figura 22 – Modelo de Requisitos para o requisito "*IMP. Importação de Documentos*" da LPS ARQUO™

Exemplo 2: Captura e Indexação de Objectos

O módulo Captura inclui alguns requisitos externos à LPS e requisitos enquadrados no âmbito aplicacional. A Figura 23, ilustra estes requisitos.

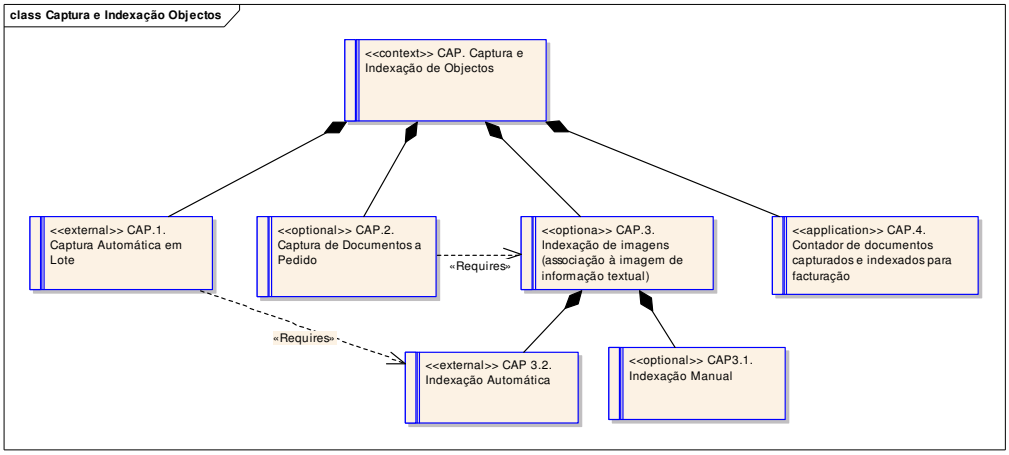


Figura 23 – Modelo de Requisitos para o subsistema "*Captura e Indexação de Documentos*" da LPS ARQUO

Tal como definido na vista de requisitos modelo SPL-LIGHT a classificação dos requisitos identificar e classificam um requisito na arquitectura da LPS, de forma a ser mais fácil a percepção do seu âmbito e promover a reutilização durante a engenharia aplicacional.

A classificação de cada requisito do modelo de requisitos faz-se através das etiquetas específicas definidas no perfil UML (ver IV.6.3.1), para cada um dos estereótipos. Em Tabela 40, Tabela 41 e Tabela 42, são ilustrados os valores atribuídos para um exemplo de cada um dos estereótipos presentes no perfil UML. As figuras seguintes ilustram a classificação de alguns dos requisitos contidos no pacote de requisitos "*Importação de Documentos*".

Tipo de Requisito	Exemplo Representação Requisito e Etiquetas
Context	req IMP. Importação de Documentos (tagged values) <<context>> IMP. Importação de Documentos tags FeatureType = Functional ID = IMP.
Optional	req IMP. Importação de Documentos (tagged values) <<optional>> IMP.3. Importação Remota por Aplicação Externa tags FeatureType = Functional FitCriterion = Documento importado com sucesso. Resposta de sucesso pelo sistema à aplicação externa. Goal = Importação de um documento por solicitação ID = IMP.3. Priority = N/A Rational = Permitir a invocação de uma interface com o ARQUO para importação distribuída de documentos ValuesRange = Importação individual de documento e índices em cada invocação

Tabela 40 – Exemplos de classificação para requisitos do tipo "*Context*" e "*Optional*".

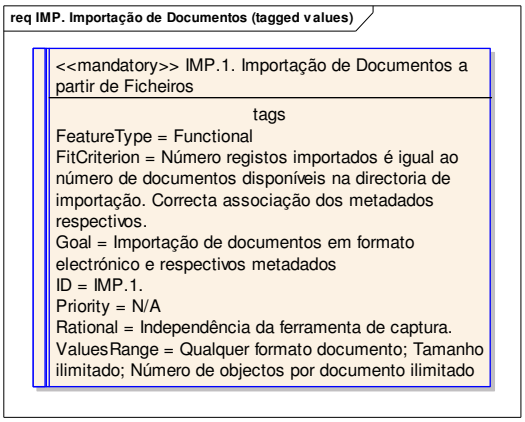
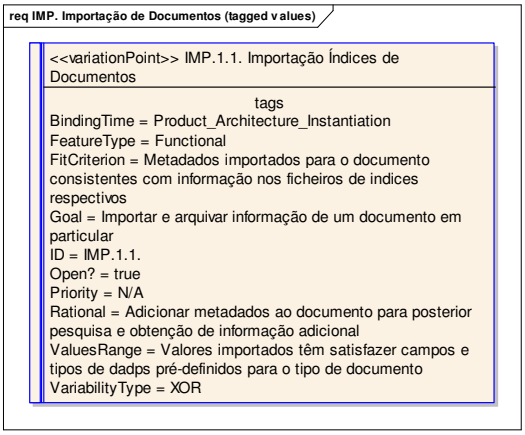
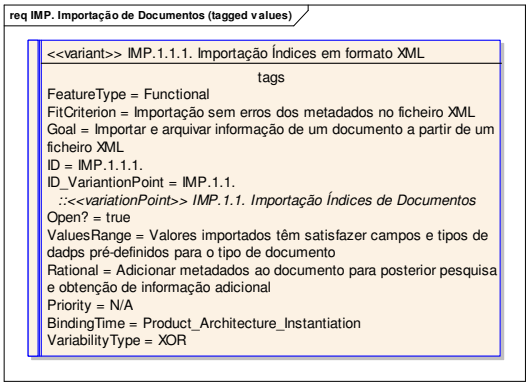
Tipo de Requisito	Exemplo Representação Requisito e Etiquetas
Mandatory	
VariationPoint	
Variant	

Tabela 41 – Exemplos de classificação para requisitos do tipo *"Mandatory"*, *"VariationPoint"* e *"Variant"*.

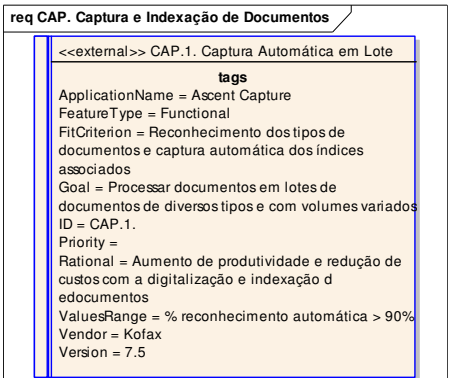
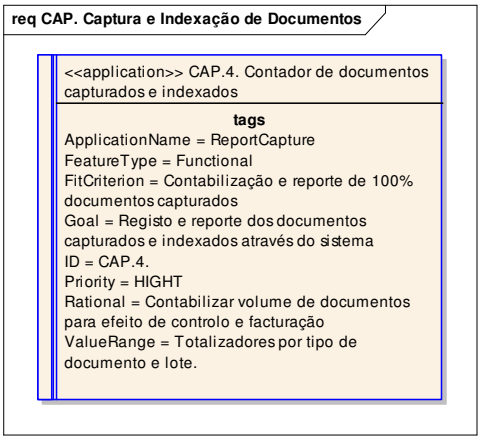
Tipo de Requisito	Exemplo Representação Requisito e Etiquetas
External	
Application	

Tabela 42 – Exemplos de classificação para requisitos do tipo "*External*" e "*Application*".

V.3.1.2. Modelo de Casos de Utilização

A especificação do comportamento dos requisitos funcionais descritos no modelo de requisitos é realizada utilizando para o efeito diagramas de casos de utilização.

A representação através de casos de utilização deve ser utilizada quando se pretenda explicitar os cenários de interacção com os utilizadores e o comportamento esperado do sistema. A utilização dos casos de utilização é uma opção do analista funcional, aquando da especificação de requisitos no contexto do problema.

Para os requisitos cuja implementação ou refactorização não exija este nível de detalhe na análise, não necessitam desta representação. Por outro lado, esta é uma ferramenta privilegiada na localização e representação dos pontos de variabilidade, no contexto da interacção, utilizando as relações de inclusão e extensão dos casos de utilização. A Figura 24, ilustra um exemplo de um caso de utilização para o requisito "*IMP.1.1. Importação de Índices de Documentos*".

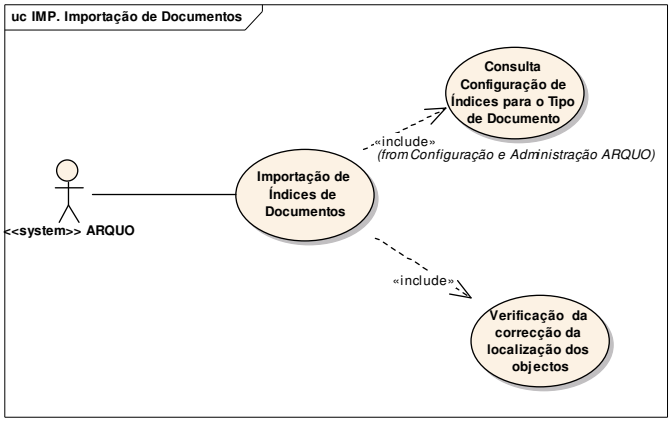


Figura 24 – Caso de Utilização para o requisito "*IMP.1.1. Importação de Índices de Documentos*"

A relação entre um ponto de variabilidade e os seus variantes descrita ao nível dos requisitos poderá ser representada através das relações de inclusão e de extensão dos casos de utilização.

O caso de utilização "*Importação de Índices de Documentos*", descrito na Tabela 43, possui o seguinte cenário principal (caminho básico). No passo 2, está descrito um ponto de extensão no caso de utilização, que corresponde à implementação da variabilidade descrita no modelo de requisitos para o requisito IMP.1.1.

Caso de utilização: Importação de Índices de Documentos	
Etiquetas	ID: 12 (identificador unívoco) REQ_ID: IMP.1.1. (referência ao requisito cujo comportamento implementa)
Caminho Principal	1- Abrir ficheiro 2- Ponto de extensão: **<interpreta ficheiro de índices>** 3- Identifica tipo de documento a importar 4- Include Caso de Utilização: "Consulta Configuração de Índices para o Tipo de Documento" 5- Para cada índice do documento: verifica correcção dos tipos de dados face aos dados importados. 6- Include Caso de Utilização: "Verificação da correcção da localização dos objectos". 7- Processa importação do documento e localização respectiva dos objectos.

Tabela 43 – Caso de Utilização "*Importação de Índices de Documentos*"

Modelação das variabilidades utilizando a relação de "extensão"

A descrição do comportamento de cada um dos variantes faz-se utilizando os mecanismos de extensão existentes nos casos de utilização. A Figura 25, ilustra a representação do requisito variante "*IMP.1.1.1. Importa índices em formato XML*".

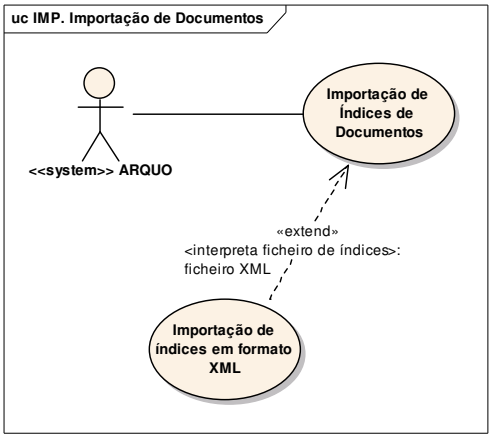


Figura 25 – Caso de Utilização para o requisito "*IMP.1.1.1. Importação de Índices em formato XML*"

Pena análise da figura, verifica-se que o caso de utilização "*Importação de Índices em Formato XML*" é um variante do ponto de variabilidade "interpreta ficheiro de índices" do caso de utilização "*Importação de Índices de Documentos*"

Caso de utilização: : Importação de Índices em formato XML	
Dependência	*Extends* "*Importação de Índices de Documentos*" no ponto de extensão <interpreta ficheiro de índices>.
Etiquetas	ID: 13 (identificador unívoco) REQ_ID: IMP.1.1.1. (referência ao requisito cujo comportamento implementa)
Caminho Principal	Obtém formato ficheiro 2- Aplica XSLT 3- Obtém *metadados* do documento 4- Retorna *metadados* do documento ao serviço de importação

Tabela 44 – Caso de Utilização "*Importação de Índices de Documentos*"

Modelação das variabilidades utilizando a relação de "inclusão"

A relação de "inclusão" é adequada à representação de casos de utilização opcionais, i.e. que representam o comportamento de um requisito funcional opcional na LPS.

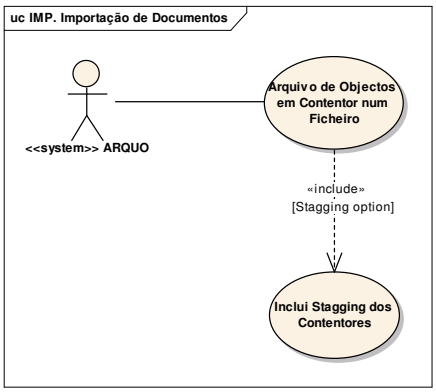


Figura 26 – Caso de Utilização para o requisito "*IMP.1.3.2.1. Arquivo de Objectos em Contentor num Ficheiro*"

Os casos de utilização opcionais são representados por uma relação de inclusão com uma condição lógica. Caso a condição seja verdadeira para uma instância do produto no âmbito de um projecto, o caso de utilização está disponível. A figura anterior ilustra a relação de inclusão como implementação de um ponto de variabilidade opcional da LPS. O mecanismo de variabilidade (ver II.2.4) a aplicar neste caso poderá ser um parâmetro de configuração do sistema (configurado durante a instalação) que indica se o arquivo de objectos em contentores será alvo de "*stagging*" ou não.

Caso de utilização: Arquivo de Objectos em Contentor num Ficheiro	
Dependência	*Include* "*Inclui Stagging dos Contentores*"
Condição de inclusão	<stagging>
Etiquetas	ID: 17 (identificador unívoco) REQ_ID: IMP.1.3.2.1 (referência ao requisito cujo comportamento implementa)
Caminho Principal	1- Obtém contentor de objectos aberto 2- Adiciona novo objecto 3- Retorna localização do objecto no contentor 4- Se contentor atingiu máximo de tamanho ou número de objectos, fecha o contentor 5- <stagging>: Include "*Inclui Stagging de Contentores*"

Tabela 45 – Caso de Utilização "*Arquivo de Objectos em Contentor num Ficheiro*"

A Tabela 46 descreve o caso de utilização "*Inclui Stagging de Contentores*".

Caso de utilização: Inclui Stagging de Contentores	
Etiquetas	ID: 18 (identificador unívoco) REQ_ID: IMP.1.3.2.1.4 (referência ao requisito cujo comportamento implementa)
Caminho Principal	1- Altera estado do contentor para "stagging" 2- *Include* caso de utilização "*Controlo de Qualidade de Contentor*"

Tabela 46 – Caso de Utilização "*Inclui Stagging de Contentores*"

V.3.1.3.Modelo de Cenários de Atributos de Qualidade

O objectivo deste modelo é a documentação das decisões arquitecturais realizadas sobre o ARQUO, com base na análise de um conjunto de requisitos não funcionais.

A especificação dos requisitos não funcionais ou atributos de qualidade definidos para a LPS é detalhada por intermédio de cenários de atributos de qualidade.

Esta representação tem a vantagem de caracterizar de forma explícita os atributos de qualidade face aos cenários estímulo/reacção associada, associando-os a uma decisão arquitectural a ter em consideração para satisfazer os atributos de qualidade respectivos. A Figura 27, apresenta um subconjunto dos requisitos não funcionais no âmbito do subsistema "Importação de Documentos". Os requisitos IMP. e IMP.3. são requisitos funcionais que contextualizam o âmbito de aplicação dos atributos de

qualidade. A especificação destes requisitos em cenários de atributos de qualidade está apresentada em Figura 28 e Figura 29.

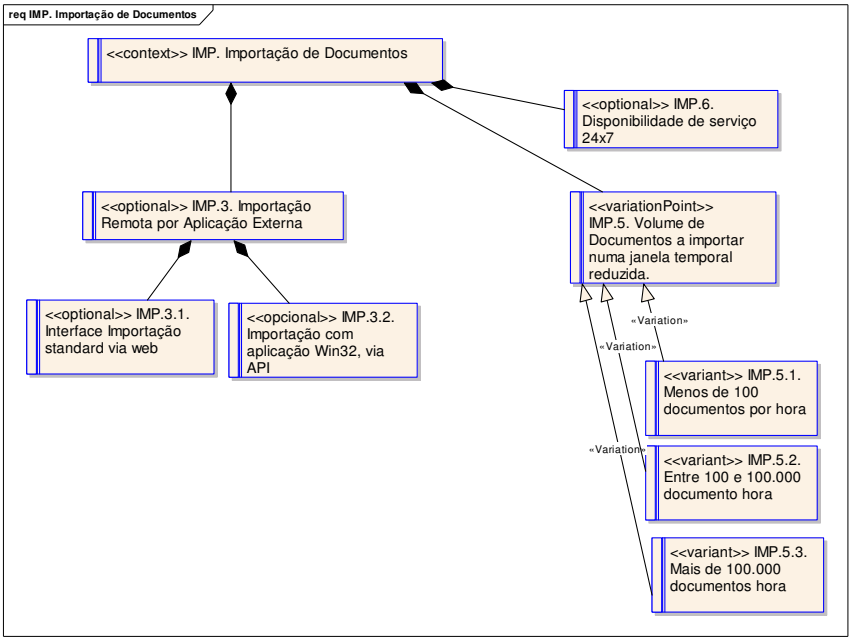


Figura 27 – Requisitos não funcionais do subsistema "*IMP. Importação de Documentos*"

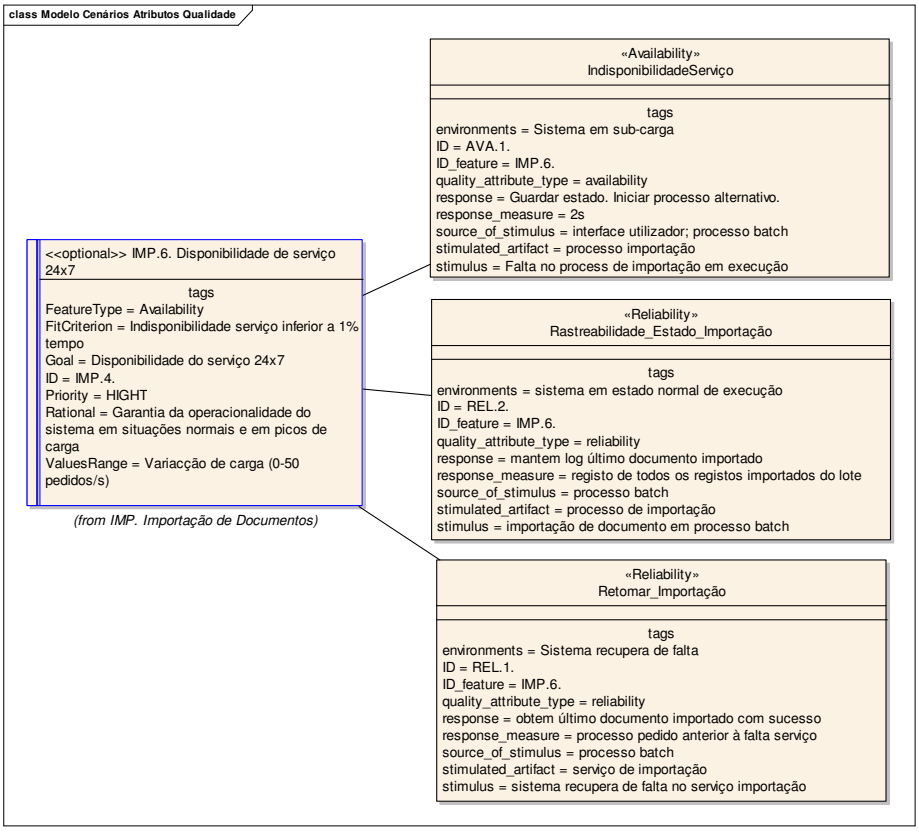


Figura 28 – Cenários de Atributos de Qualidade para o Requisito: "*IMP.4. Disponibilidade de Serviço de 24x7*"

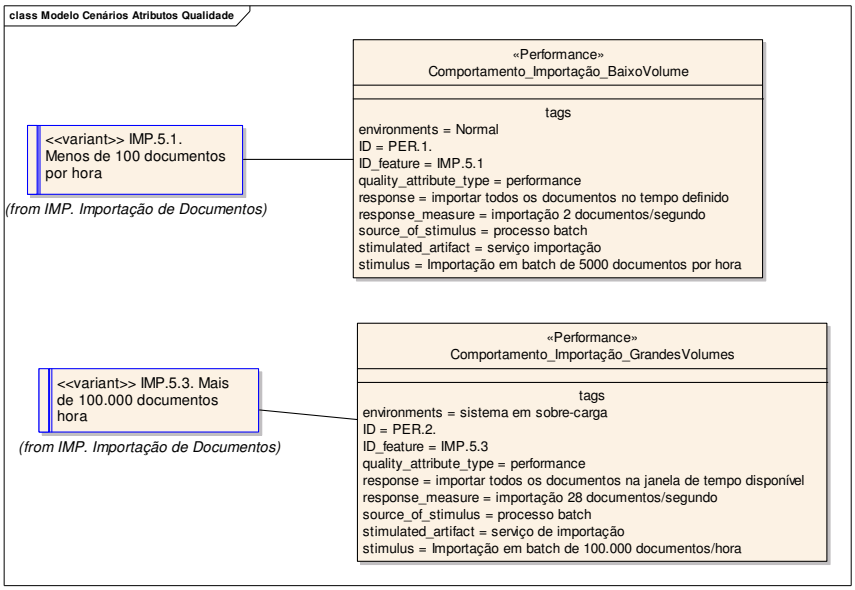


Figura 29 – Cenários de Atributos de Qualidade para os Requisitos: "*IMP.5.1 e IMP.5.3*"

Aquando da instanciação dos produtos da LPS, o arquitecto de software opta pelas decisões arquitecturais que melhor satisfazem os atributos de qualidade adequados à realidade do projecto de engenharia aplicacional.

A Figura 30, ilustra uma táctica prevista na LPS e utilizada pelo arquitecto de software na instanciação de um produto que satisfaz os cenários de atributos de qualidade referenciados.

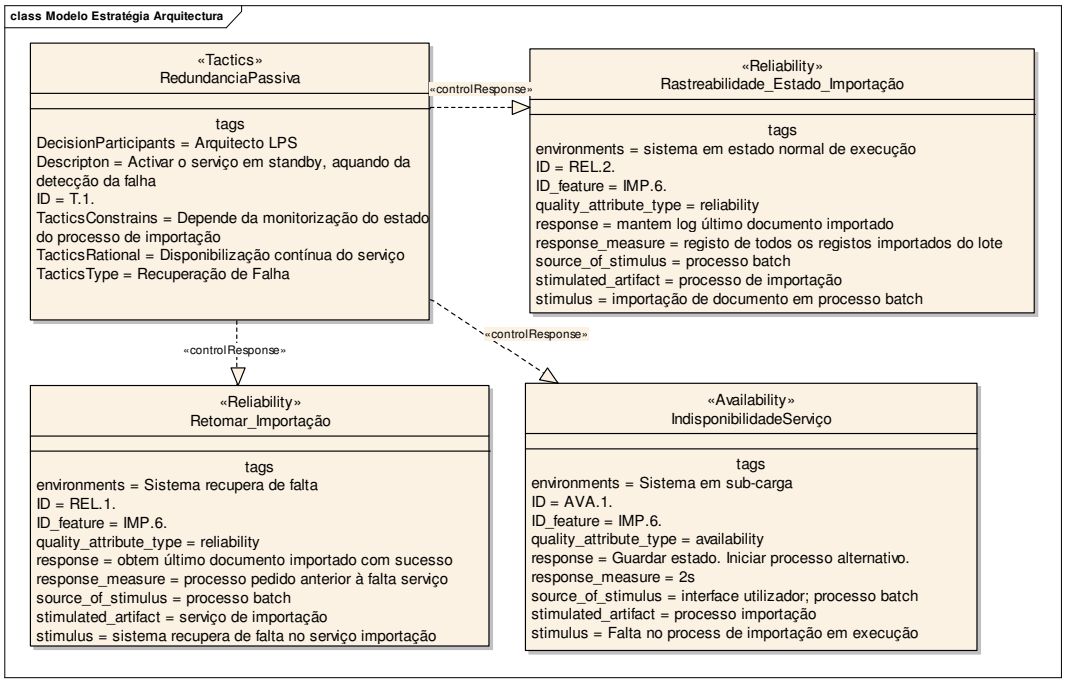


Figura 30 – Táctica "*Redundância Passiva*"

Na Figura 31, é apresentada uma táctica que pretende alcançar a resposta adequada face ao estímulo de importação de 100.000 documentos por hora.

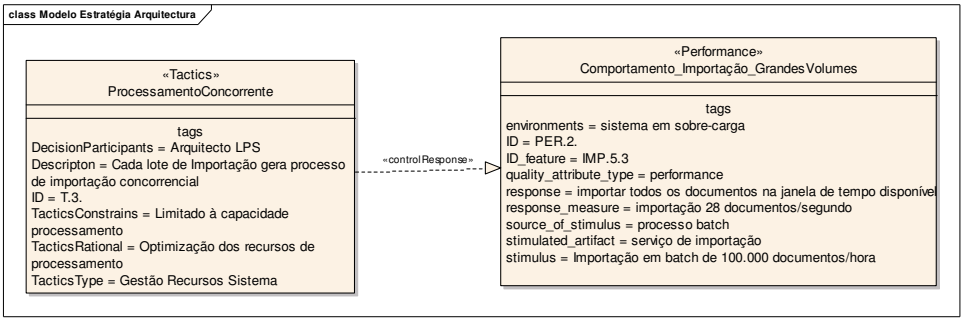


Figura 31 – Táctica "*Processamento Concorrente*"

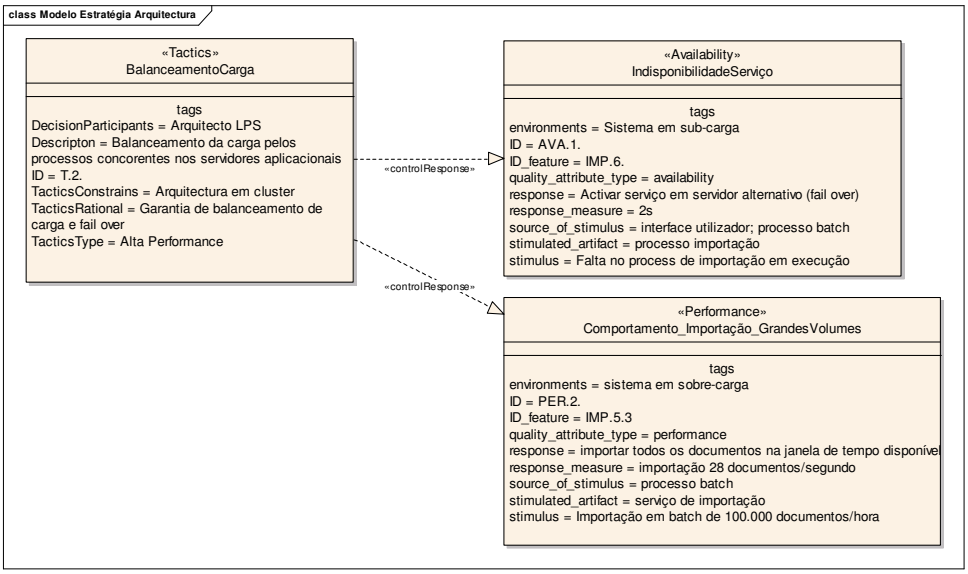


Figura 32 – Táctica "*Balanceamento de Carga*"

A implementação das tácticas, como forma de controlar as respostas aos estímulos descritos nos cenários de atributos de qualidade poderá passar pela instanciação de variantes arquitecturais definidos ao nível dos módulos de software e/ou conectores.

V.3.1.4.Modelo de Domínio

O modelo de entidades, ilustrado pela Figura 33, traduz os principais conceitos e entidades no âmbito da LPS ARQUO™. Estes podem ser agrupadas em quatro áreas principais:

- Gestão Documental
- Arquivo Óptico
- Gestão Utilizadores
- Gestão de Contas

O modelo de domínio apresentado é o resultado da integração do modelo de domínio da arquitectura intermédia com a incorporação de novos conceitos e entidades entretanto adicionados ao domínio da LPS ARQUO™. A gestão documental e *workflow* é uma das novas funcionalidades da LPS, estando representada no modelo de domínio pelas entidades: "*Processo*", "*Tarefa*" e relações com as entidades "*Documento*" e "*Utilizador*".

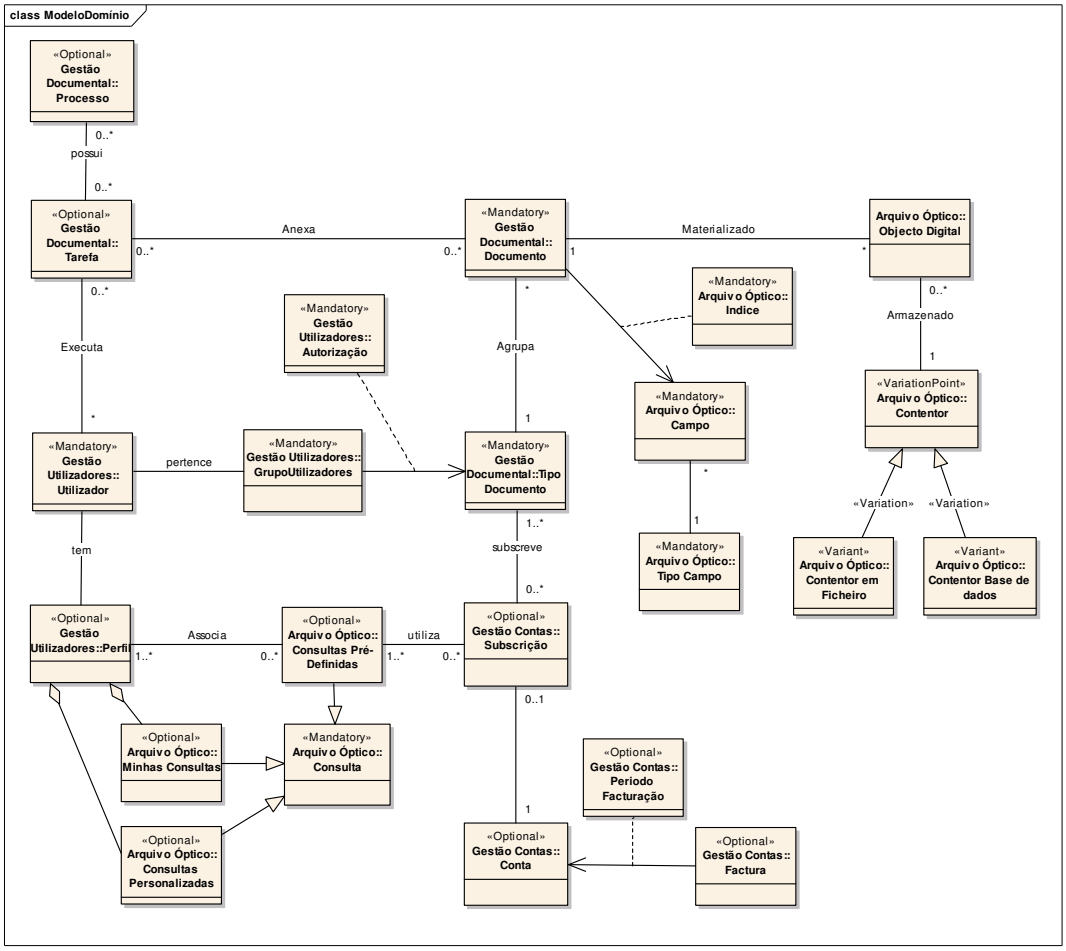


Figura 33 – Modelo de Domínio da LPS ARQUO™

As entidades "*mandatory*" constam de todas as instâncias dos produtos ARQUO™. As "*optional*" são entidades que poderão, ou não, fazer parte do âmbito das instâncias dos produtos ARQUO™. A identificação dos pontos de variabilidade e variantes no modelo de domínio permite explicitar que para diferentes instâncias da LPS, o mesmo conceito pode ter especificações e âmbitos diferentes.

A conjunção do modelo de domínio, requisitos funcionais e especificação destes em cenários de interacção com os utilizadores define o domínio do problema que a LPS pretende endereçar. A representação do modelo de classes que define o espaço da solução é apresentada através da vista módulo, descrita na secção seguinte.

V.3.2. Vista Módulo

A vista módulo apresenta a decomposição funcional dos módulos de software da LPS ARQUO™, desde os módulos funcionais "*alto-nível*", e.g. "*Importação Documentos*" ou "*Consulta Documentos*", até ao módulo unitário (e.g. classe ou interface) que implementa uma determinada funcionalidade.

A vista módulo decompõe o contexto da solução, tanto na perspectiva "*alto-nível*" de enquadramento da implementação das funcionalidades, como na perspectiva de implementação unitária necessária ao engenheiro de software.

A título ilustrativo nesta secção será apresentado a decomposição do módulo "Importação de Documentos".

A Figura 34, apresenta o diagrama de contexto da vista módulo, para o módulo "*ImporServer*" da LPS ARQUO™. Este módulo é constituído por um conjunto de outros sub-módulos, e.g. "*RecorControlServer*", "*ImportServer*", "*AutoFileServer*".

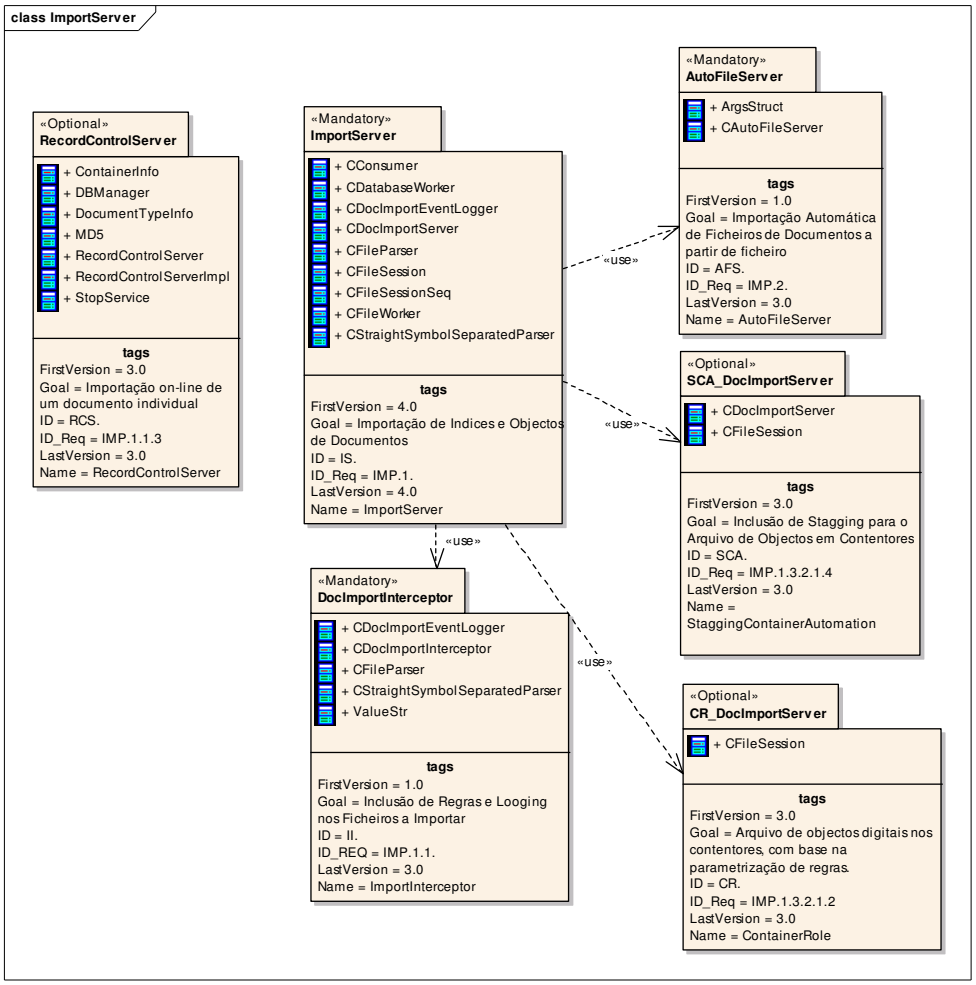


Figura 34 – Vista Módulo - Decomposição do módulo "*ImportServer*" da LPS ARQUO™

Na Figura 34, estão descritas as relações de decomposição e de utilização entre os módulos da vista: a) decomposição do módulo "*ImportServer*" nos módulos (classes) que o compõem (relação "*é-parte-de*") e b) relação "*usa*" entre módulos. Cada módulo é caracterizado por um conjunto de etiquetas, tal com definido no modelo SPL-LIGHT. O módulo "*RecordControlServer*" é opcional pois implementa o requisito "*optional*" "*IMP 4. Importação de Documentos através da Interface com Utilizador*", que poderá estar, ou não, presente nos produtos a instanciar da LPS.

Na Figura 35, estão apresentados os sub-módulos do módulo "*ImportServer*", após a refactorização (ver V.2.3.2), e respectivos estereótipos SPL-LIGHT.

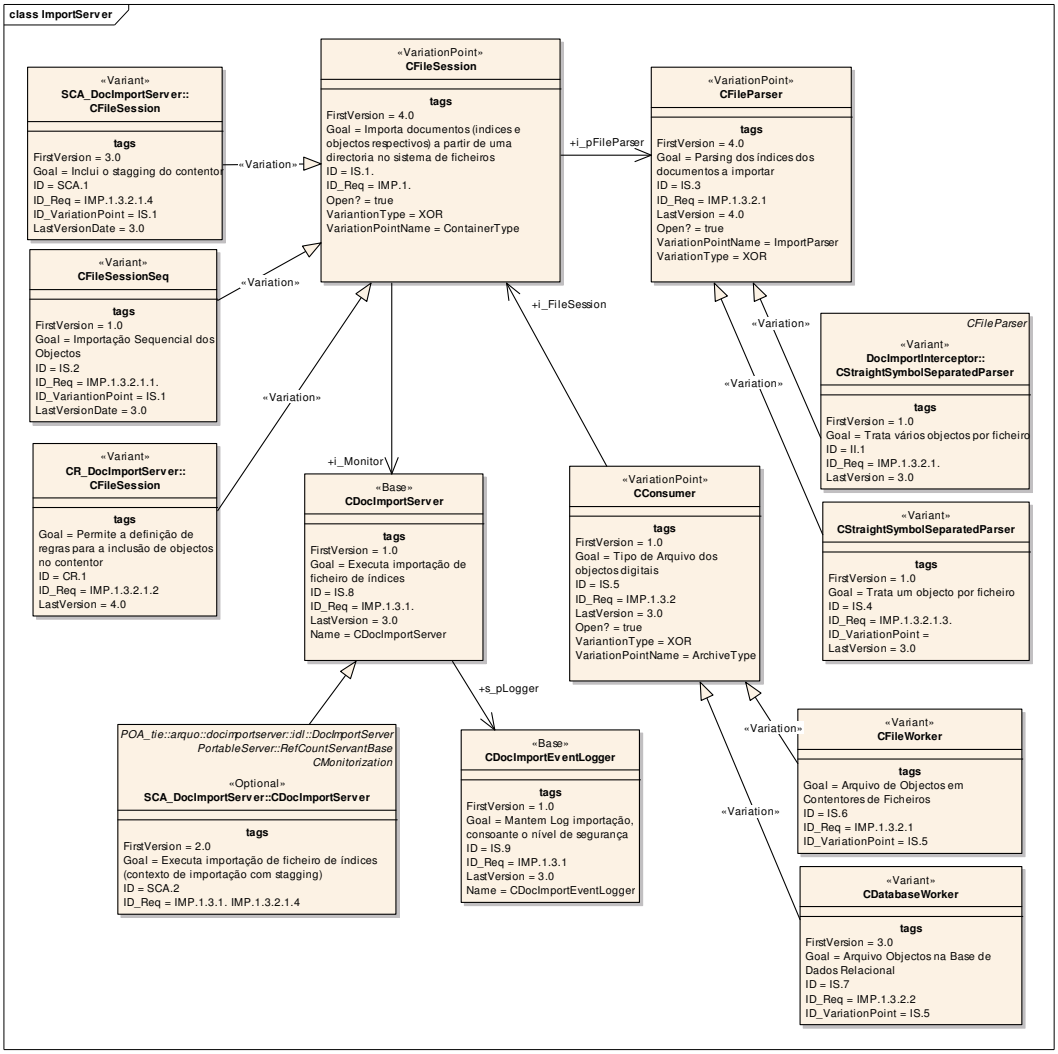


Figura 35 – Diagrama de Classes utilizando Perfil UML SPL-LIGHT para o módulo "*Importação de Documentos*"

A introdução dos estereótipos "*Variation Point*" e "*Variant*", permite ao engenheiro de software da LPS poder adicionar ou alterar funcionalidades de um variante, sem que com isto tenha que alterar os módulos que implementam os restantes variantes, logo sem impacto nos produtos já instalados.

Uma das formas de implementação dos pontos de variabilidade e variantes faz-se através da utilização do padrão de desenho "Fábrica", onde o ponto de variabilidade é implementado como uma classe abstracta e os variantes estão associados através de uma relação de generalização.

```
package ImportServer.ImportServer;
public class CFileParser {
        public CFileParser(){
        }
        public void finalize() throws Throwable {
        }
}
```

Figura 36 – Estrutura da Classe Java gerada para Ponto de variabilidade "*CFileParser*"

```
package ImportServer.ImportServer;
public class CStraightSymbolSeparatedParser extends CFileParser {
        public CStraightSymbolSeparatedParser(){
        }
        public void finalize() throws Throwable {
                super.finalize();
        }
}
```

Figura 37 – Estrutura da Classe Java gerada para Variante "*CStraightSymbolSeparatedParser*"

V.3.3. Vista Componente & Conector

A documentação desta vista tem como objectivo representar os componentes de execução da LPS ARQUO bem como as possibilidades de ligação entre elas ou com aplicações externas.

A arquitectura do ARQUO™ pretende satisfazer os requisitos de desempenho, escalabilidade e robustez mesmo em picos de carga elevados (ver atributos de qualidade do ARQUO™ em V.3.1.1), pode ser caracterizada por ser orientada à disponibilização de serviços (*Service Oriented Archiecture* - SOA). Assim, os componentes de execução do ARQUO™ estão disponíveis como serviços de execução que interagem entre si ou com aplicações externas, através de conectores padrão.

De uma forma geral, os conectores utilizados na LPS foram desenvolvidos utilizando o padrão de interacção *Object Request Broker* (ORB) presente na arquitectura CORBA.

A integração do ARQUO™ em portais corporativos e aplicações *web* de entidades externas, para os serviços de consulta, importação e exportação de documentos de e para o arquivo é realizada através de conectores implementados com *web services.*

A opção pela selecção dos conectores (CORBA ou *Web Services*) dependerá dos requisitos específicos de cada produto, nomeadamente a necessidade de interacção com aplicações externas e/ou utilização das aplicações "*cliente*" ARQUO™.

A necessidade de implementar redundância e/ou balanceamento de carga na arquitectura de um produto em particular, poderá implicar alterações na configuração dos conectores. Alguns dos serviços base do ARQUO™, e.g. importação de documentos poderão ser executados de forma concorrente na mesma máquina (com nomes de serviços diferentes) ou em máquinas separadas (em *cluster activo-passivo*).

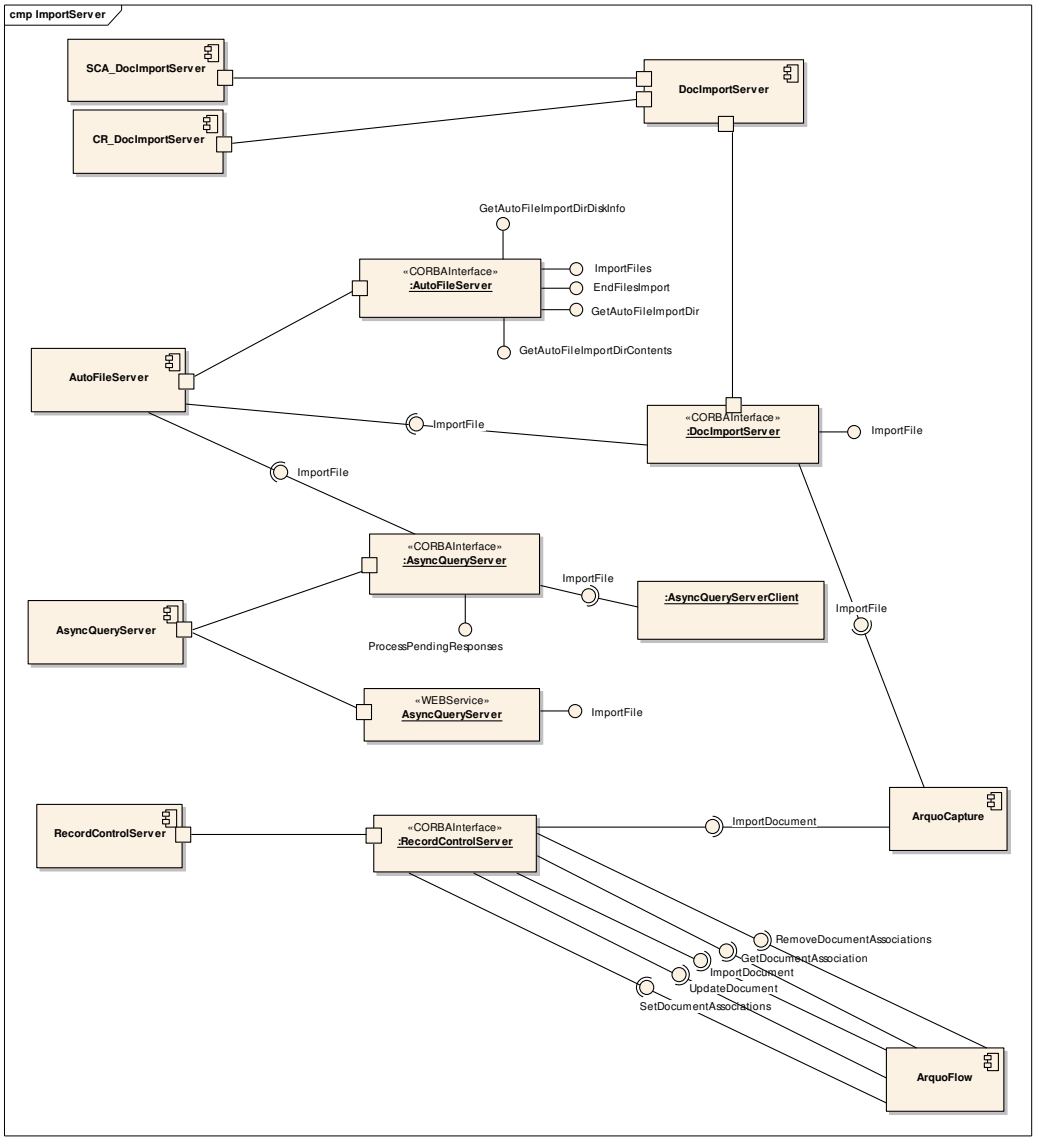


Figura 38 – Componentes e Conectores do módulo "*Importação de Documentos*" do ARQUO™

A Figura 38, ilustra os componentes e conectores do módulo de Importação de Documentos do ARQUO™, e respectivas ligações.

Da análise da Figura 38, verificamos como poderão ser distribuídos os componentes que implementam as funcionalidades de importação de documentos pelo ambiente de execução. Na instanciação dos produtos da LPS, apenas os componentes e conectores que executam os requisitos pretendidos serão instalados na infra-estrutura tecnológica de execução.

Na Figura 40, é ilustrada a vista C&C que executa as funcionalidades base "IMP.1. Importação de Documentos a partir de ficheiros" e funcionalidade opcional "IMP.4. Importação de Documentos via Interface de Utilizador".

No produto final, apenas constarão os componentes e conectores que implementam requisitos no âmbito do projecto. Por exemplo, o componente de execução "*RecordControlServer*" fará parte do produto caso seja necessária a importação de documentos a pedido, via interface com o utilizador (requisito *IMP.4.*), neste caso a aplicação *web ARQUOflow*.

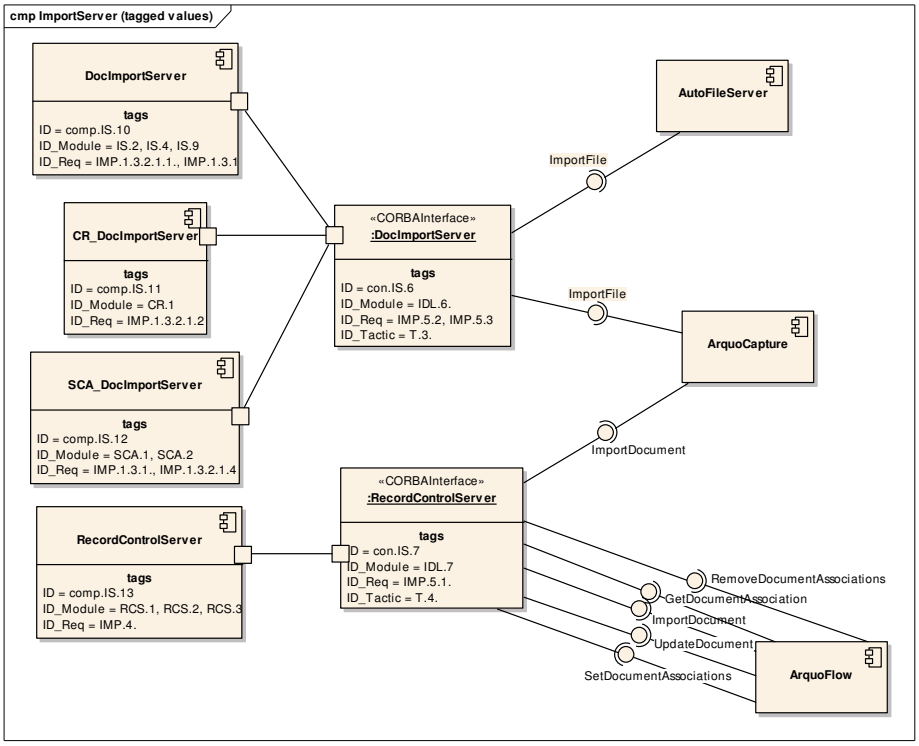


Figura 39 – Componentes & Conectores do elemento de software "*Importação Documentos*" e respectivas etiquetas

A relação entre componentes e módulos de software é explícita e é materializada no momento da compilação do código fonte (*c++* ou *Java*) e ligação e geração do componente de execução (*.exe* ou *.jar*).

De uma forma geral, os elementos de software ARQUO™ implementam funcionalidades do sistema e os conectores são o resultado da implementação das decisões arquitecturais tomadas face à especificidade dos atributos de qualidade de

cada produto. A Figura 40, ilustra as duas possíveis implementações de conectores para o componente de execução *AsyncQueryServer*: via interface CORBA ou *Web Services*, face aos cenários de requisitos de interoperabilidade apresentados.

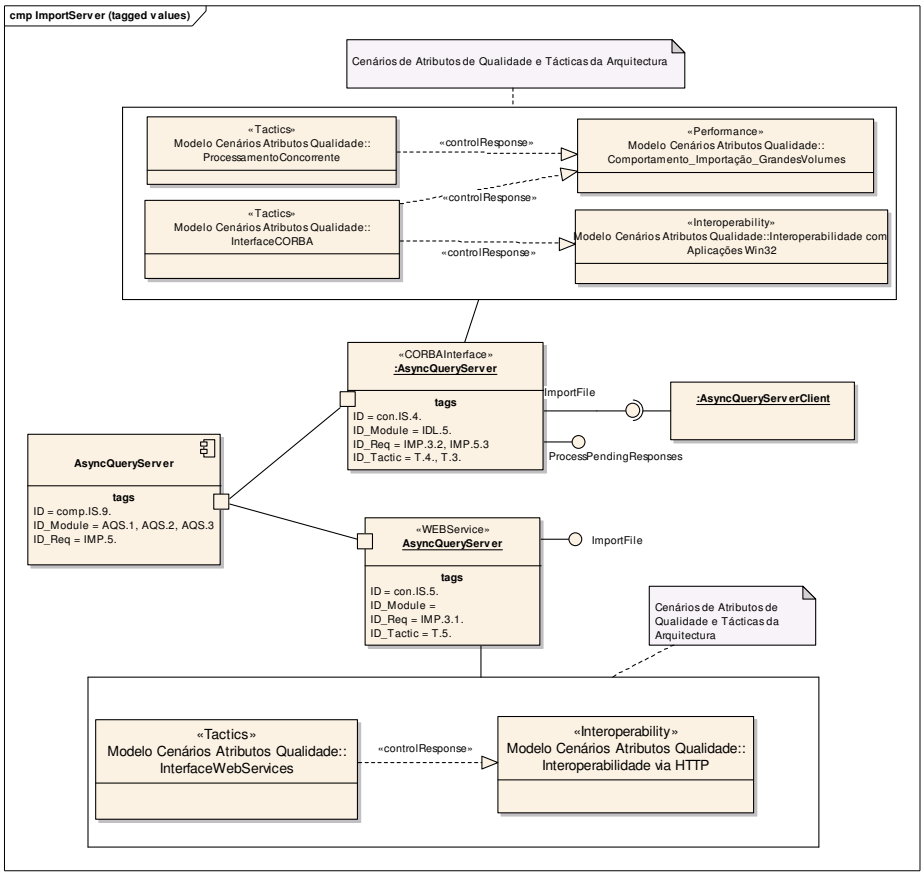


Figura 40 – Representação de duas diferentes implementações para conectores do mesmo componente de execução, face aos requisitos não funcionais dos produtos.

Na figura é possível identificar a correspondência entre o conector, os requisitos que são satisfeitos e as decisões de arquitectura (ou tácticas) na base da sua concepção. A identificação do relacionamento entre os vários elementos é conseguida através dos identificadores unívocos dos elementos (*ID_Req, ID_Module, ID_Tactic*).

A decisão pela utilização de conectores ORB (conector CORBA *con.IS.4.* na figura) teve como origem:

- A necessidade de garantia de alto desempenho (requisito desempenho *IMP.5.2* e *IMP.5.3.*) e
- Expor para outras aplicações *Win32* uma API (requisito de interoperabilidade *IMP.3.2*) para importar os metadados e objectos dos documentos.

V.3.4.Vista Produtos

A vista produtos na LPS ARQUO™ requer a implementação de uma ferramenta de Gestão de Configurações de Software (GCS) que satisfaça os requisitos específicos da gestão de uma LPS.

Neste momento na TIE é utilizada a ferramenta de GCS *Perforce da* Perforce Software[60] para a Gestão de Configurações de Software, no entanto esta possui algumas limitações no contexto da sua aplicação a uma LPS, nomeadamente na gestão de configurações multi-produto, gestão integrada de alterações e *baselines* ao nível dos componentes de configuração.

Nesta secção os exemplos apresentados, não estão associados a uma ferramenta em particular, constituindo um bom exemplo de cenários de utilização para a selecção de uma nova ferramenta ou adaptação da ferramenta existente de forma a ser adequada à gestão de configurações de software em LPS.

V.3.4.1. Produtos da LPS

A LPS ARQUO é constituída por uma família de produtos, que partilham dos mesmos princípios arquitecturais. Para cada produto existem um conjunto de elementos base, os quais incluem pontos de variabilidade, variantes que poderão ser instanciados, bem como desenvolvimentos específicos de forma a satisfazer as necessidades dos clientes. A Tabela 47, ilustra os diferentes produtos da LPS ARQUO, e dá exemplos de entidades onde estes produtos se encontram instalados.

Produtos	Descrição	Entidade Associada
ARQUOArchivingSystem	Contém os módulos funcionais principais do sistema de arquivo documental: importação, consulta, exportação de metadados e objectos associados aos documentos digitais. Possui ainda vários elementos de software transversais auxiliares, e.g. parsing de ficheiros, stagging contentores de objectos e gestão de offline.	SIBS Processos MillenniumBCP Caixa Central de Crédito Agrícola ENSA – Seguros de Angola Cabo TV Madeirense Investimentos e Gestão da Água (IGA) Sociedade Portuguesa de Cardiologia FaxInForme
ARQUOBrowser	Solução Web de consulta a tipos de documentos arquivados no ARQUO.	
ARQUOManager	Solução Web de suporte à configuração e administração da solução ARQUO. Possui a definição de utilizadores, autorizações, configurações de documentos, monitorização SNMP, e gestão de repositório de objectos.	SIBS Processos MillenniumBCP Caixa Central de Crédito Agrícola ENSA – Seguros de Angola Cabo TV Madeirense Investimentos e Gestão da Água (IGA) Sociedade Portuguesa de Cardiologia FaxInForme

Produtos	Descrição	Entidade Associada
ARQUOKofaxConector	Conector com a solução da Kofax Accent Capture para a importação automática de documentos digitalizados e respectivos metadados.	Cabo TV Madeirense Investimentos e Gestão da Água (IGA)
ARQUOFlow	Solução de gestão documental e workflow.	Investimentos e Gestão da Água (IGA)
ARQUOCapture	Solução Web de captura e digitalização de documentos.	Investimentos e Gestão da Água (IGA)

Tabela 47 – LPS ARQUO™: Produtos e Instalações

V.3.4.2. Projectos

Aplicando a metodologia SPL-LIGHT, na LPS existe um projecto permanente de suporte à engenharia de domínio, i.e. para fazer evoluir os produtos no âmbito do plano de evolução (*roadmap*) definido. Fazem parte deste projecto o arquitecto da LPS e engenheiros de domínio. Este projecto tem uma linha de configuração "principal" denominada: LCP_LPSARQUO™.

A instalação de um ou mais produtos da LPS ARQUO™ num cliente em particular implica a criação de um projecto onde será construída a solução como um todo (denominado produto de configuração). Cada projecto no âmbito da engenharia aplicacional tem uma linha de configuração "principal", referenciando o nome do cliente, e.g. LCP_ClienteXPTO. Após a entrada em produção, este projecto mantém-se permanente pois será utilizado para suporte à actividade de manutenção evolutiva e correctiva dos produtos instalados.

V.3.4.3.Linhas de Configuração

Na LPS ARQUO poderão ser utilizadas três tipos de linhas de configuração, as quais poderão estar limitadas com controlos de acessos:

- Linha de Configuração Principal (denominada LCP). É a linha de configuração permanente do projecto, sob a qual são realizados os testes de sistema.
- Linha de Configuração de Integração (denominada LCI). É uma linha de configuração criada com um determinado propósito, e.g. integração de elementos de configuração de diferentes produtos, produzidos em paralelo, testes de integração, testes em ambiente de qualidade. É usual existir uma linha de configuração para cada ambiente de execução, e.g. integração, qualidade.
- Linha de Configuração de Desenvolvimento (denominada LCD). Cada engenheiro de software poderá igualmente criar uma LCD específica para gerir as versões dos elementos de configuração no âmbito do seu trabalho e suportar os testes unitários.

Engenharia de Domínio

Na LPS ARQUO™, as linhas de configuração são utilizadas pelo arquitecto da LPS e engenheiros de domínio para desenvolvimento e teste ao nível dos componentes de configuração da LPS e respectivas ligações. A Tabela 48 ilustra as diferentes linhas de configuração e as actividades de engenharia de domínio que são executadas.

Linha de Configuração	Tipo	Principais actividades executadas
LCP_LPSARQUO	Linha de configuração principal da LPS ARQUO	Suporte os testes de integração entre os componentes de configuração da LPS. Realização de *baselines* estáveis para cada componentes de configuração a reutilizar nas instâncias dos produtos da LPS. Gestão das *releases* para os componentes de configuração da LPS ARQUO™.
LCI_LPSARQUO	Linha de configuração de integração e testes da LPS ARQUO.	Suporta a integração e testes de integração dos componentes de configuração desenvolvidos pela equipa de engenharia de domínio. Suporta o *merge* de versões com novos desenvolvimentos realizados nos projectos em curso e que podem ser "promovidos" a elementos base da LPS.
LCD_ <NomeEng.Domínio>	Linha de configuração temporária de desenvolvimento na engenharia de domínio.	Suporta desenvolvimentos e testes unitários aos componentes de configuração, nomeadamente a inclusão de variabilidades e novas funcionalidades base presentes na linha de evolução do produto.

Tabela 48 – Linhas de Configuração: Engenharia de Domínio

A Tabela 49 exemplifica os passos necessários à execução da actividade de alteração de um componente de configuração da LPS (ilustrado na Figura 41).

Linha de Configuração Principal: LCP_LPSAEQUO	Linha de Configuração de Integração: LCI_LPSAEQUO
Engenheiro de Domínio Sénior 1. Realiza baseline dos componentes de configuração após testes de integração inter componentes de configuração. Atribui o nome: LCP_LPSARQUO_31052007 … 7. Realiza a comparação entre o último baseline estável na LCP e o baseline LCI_LPSARQUO_02062007. Faz o "merge", aceitando as alterações. 8. Realiza testes (com sucesso). Realiza baseline dos componentes de configuração após testes de integração inter componentes de configuração. Atribui o nome: LCP_LPSARQUO_03062007. 9. Realiza uma release para todos os componentes de configuração na LPS, criando um baseline por componente de configuração (do tipo "principal"), os quais podem ser utilizados para a instanciação da LPS em novos produtos.	Engenheiro de Domínio 2. Actualiza a linha de configuração integração com o baseline LCP_LPSARQUO_31052007. 3. Reserva os elementos a alterar (*check-out*). 4. Realiza a alteração dos elementos de configuração necessários à satisfação do requisito de inclusão de um novo variante (código fonte e modelos das vistas arquitecturais). 5. Realiza testes (com sucesso). Submete ("check-in") das novas versões dos elementos de configuração e atribui uma etiqueta contendo o ID_Requisito a alterar e o ID_Tarefa (registado na ferramenta de gestão de trabalho). 6. Realiza um baseline com as versões estáveis: LCI_LPSARQUO_02062007 e informa o engenheiro de domínio sénior que a tarefa foi terminada com sucesso.

Tabela 49 – Detalhe das actividades realizadas pela equipa de engenharia de domínio nas linhas de configuração da LPS.

Apesar de não estar representado na figura, poderiam existir outras linhas de configuração de desenvolvimento, uma para cada elemento da equipa de engenharia de domínio. Nesse caso, as alterações resultantes passariam primeiro pela linha de configuração de integração para serem testadas antes de serem promovidas para a linha de configuração principal da LPS.

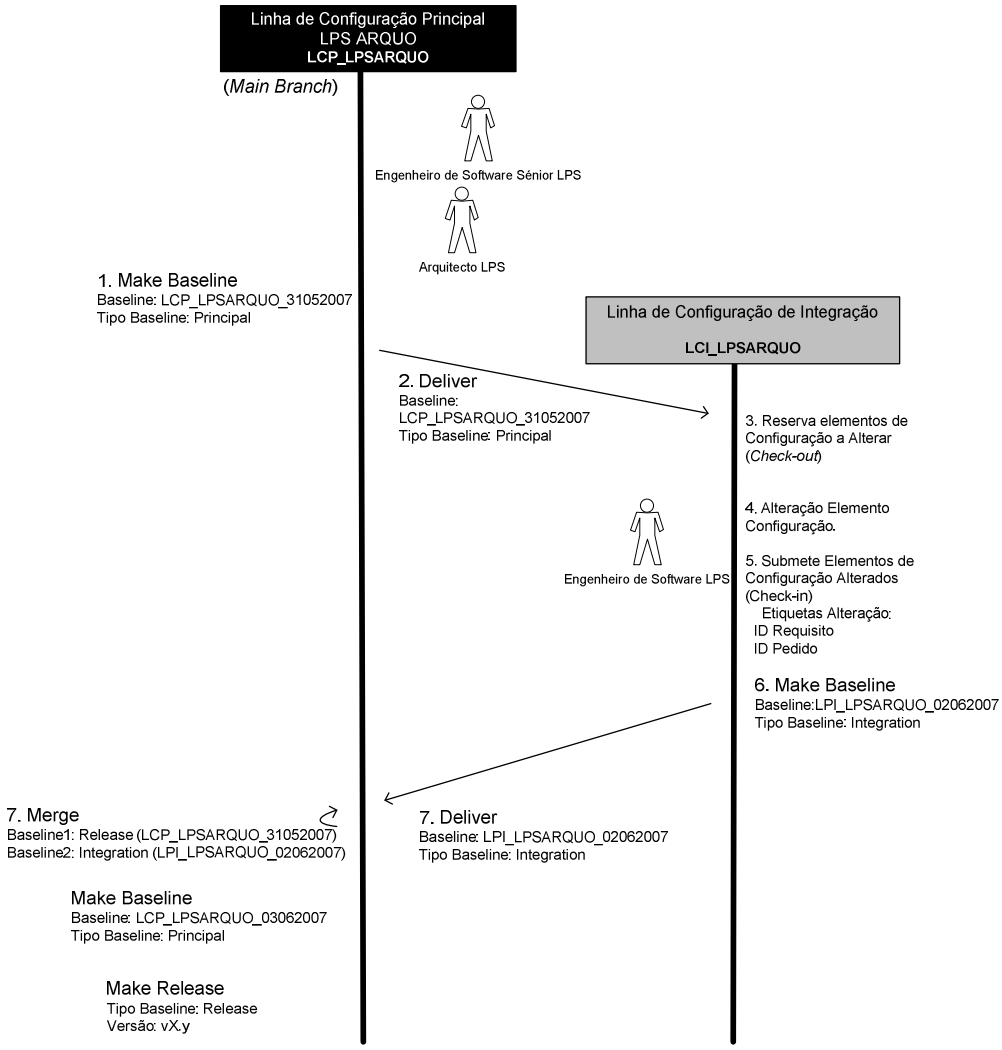


Figura 41 – Linhas de Configuração para LPS ARQUO™ e fluxo de configurações associado: Projecto Engenharia de Domínio

Engenharia Aplicacional

Em cada projecto de engenharia aplicacional, existirá uma linha de configuração principal: LCP_<*NomeCliente*>, podendo ser criadas igualmente linhas de configuração de integração e desenvolvimento, dependendo da existência de um ambiente de desenvolvimento/qualidade no cliente e número de engenheiros aplicacionais afectos ao projecto. A Tabela 50 ilustra as diferentes linhas de configuração e as actividades de engenharia de domínio que são executadas.

Linha de Configuração	Tipo	Principais actividades executadas
LCP_<*NomeCliente*>	Linha de configuração principal da instanciação dos produtos e elementos de software no âmbito do projecto.	Realização de testes de integração dos vários componentes de configuração. Realização de *baselines* estáveis do tipo *principal* do projecto (que poderão conter elementos de configuração candidatos a promoção para a LPS). Realização de *releases* ao nível do produto de configuração.
LCI_<*NomeCliente*>	Linha de configuração de integração e testes do Projecto do cliente "NomeCliente".	Suporta as alterações aos componentes de configuração da LPS, instanciação de variantes, correcção de erros e desenvolvimento de funcionalidades específicas do cliente. Suporta o *merge* de novas *releases* para os produtos instalados, antes da passagem a produção e integração no ramo de configuração LCP_<NomeCliente>.
LCD_<*NomeEng.Domínio*> (Se for previsto desenvolvimento em paralelo no âmbito do projecto)	Linha de configuração de desenvolvimento do Projecto do cliente "*Client*".	Suporta as alterações aos componentes de configuração da LPS, instanciação de variantes, correcção de erros e desenvolvimento de funcionalidades específicas do cliente.

Tabela 50 – Linhas de Configuração: Engenharia Aplicacional

As actividades de engenharia aplicacional poderão resumir-se aos seguintes cenários:

- Instanciação de um produto da LPS, instanciação dos variantes e realização de desenvolvimentos específicos.
- Alteração ou correcção de erros no âmbito do projecto.
- Desenvolvimentos específicos após a entrada em produção. Os desenvolvimentos poderão ser posteriormente "promovidos" para elementos base da LPS.
- Actualização de novas *releases* da LPS no produto de configuração existente.

No início do projecto de engenharia aplicacional é criada a linha de configuração principal do projecto. Por omissão deverá ser utilizada a última *release* realizada na linha de configuração principal da LPS. Após a entrada em produção, é recomendado a criação de uma linha de configuração de integração onde serão realizadas as alterações ou novos desenvolvimentos e testes aos componentes de configuração da linha de configuração principal do produto instalado. As migrações para novas versões deverão ser realizadas também nesta linha de configuração.

A Figura 42, ilustra a realização de desenvolvimentos específicos no âmbito de um projecto de engenharia de aplicacional (Cliente1), e posterior promoção para a linha de configuração da LPS, exemplificado em detalhe pela Tabela 51.

Linha de Configuração Principal: LCP_Cliente1	Linha de Configuração de Integração: LCI_Cliente1
Engenheiro Aplicacional Sénior … 6. Realiza a comparação entre o último *baseline* estável na LCP do projecto e o *baseline* LCI_Cliente1_15062007. Faz o "*merge*", aceitando as alterações. 7. Realiza testes (com sucesso). Realiza *baseline* dos componentes de configuração após testes de integração inter componentes de configuração. Atribui o nome: LCP_Cliente1_16062007. 8. Realiza uma *release* para o produto de configuração do Cliente1, criando um *baseline* por componente de configuração (do tipo "principal"). (mais tarde o engenheiro de domínio poderá adoptar os desenvolvimentos e "promover" o elemento de software a elemento de software base da LPS).	Engenheiro Aplicacional 1. Actualiza a linha de configuração integração com o último *baseline* da LCP do projecto para o componente de configuração a alterar: LCP_Cliente1_20070110. 2. Reserva os elementos a alterar (*check-out*). 3. Realiza a alteração dos elementos de configuração necessários à satisfação do requisito de implementação de uma nova funcionalidade específica (código fonte e modelos das vistas arquitecturais). 4. Realiza testes (com sucesso). Submete ("*check-in*") das novas versões dos elementos de configuração e atribui uma etiqueta contendo o ID_Requisito a alterar e o ID_Pedido (registado na ferramenta de gestão de trabalho). 5. Realiza um *baseline* com as versões estáveis: LCI_Cliente1_15062007 e informa o engenheiro aplicacional sénior que a tarefa foi terminada com sucesso.

Tabela 51 – Detalhe das actividades realizadas pela equipa de engenharia aplicacional nas linhas de configuração do projecto.

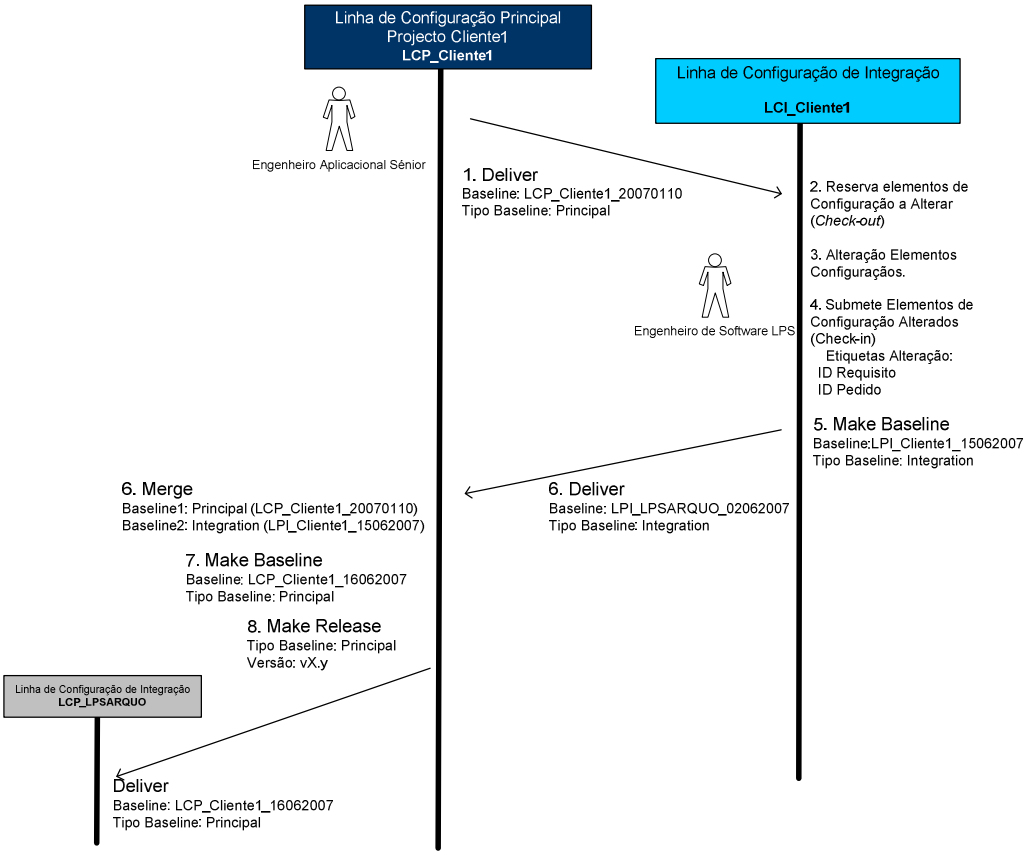


Figura 42 – Actividades e Linhas de Configuração durante uma actividade de engenharia aplicacional

V.3.4.4.Componentes de configuração

No ARQUO™, cada elemento de software representa um componente de configuração (ver definição em II.1.9) a ser gerido pela ferramenta de GCS.

A Tabela 52 ilustra alguns dos principais componentes de configuração do ARQUO e respectiva ligação com os módulos de software "*alto-nível*" da vista módulo.

Nome Componente Configuração	Nome Modulo (Vista Módulo)	ID Modulo (Vista Módulo)
ARQUOArchitecture	Não Aplicável	Não Aplicável
ImportServer	*ImportServer*	IS
RecordControlServer	*RecordControlServer*	RCS
SecurityServer	*SecurityServer*	SS
ConfigServer	*ConfigServer*	CS
DocumentQueryServer	*DocumentQueryServer*	DQS
ObjectQueryServer	*ObjectQueryServer*	OQS
MonitorInfoServer	*MonitorInfoServer*	MIS
ARQUOcapture	*ARQUOcapture*	AC
ARQUOflow	*ARQUOflow*	AF
ARQUOmanager	*ARQUOmanager*	AM
ARQUObrowser	*ARQUObrowser*	AB

Tabela 52 – Componentes de Configuração da LPS ARQUO™

Os componentes de configuração são definidos ao nível da LPS e são usados em qualquer projecto subsequente, i.e. caso tivéssemos criado pela primeira vez, por exemplo, o componente de configuração *ARQUOflow* (com os elementos de configuração respectivos, e.g. código fonte, documentos análise e desenho, ficheiros de configuração) e realizado um *baseline*, a partir desse momento seria possível em qualquer outra linha de configuração utilizar esse *baseline* com as versões de todos os elementos de configuração do componente de configuração *ARQUOflow*.

O componente de configuração *ARQUO_Architecture* corresponde à documentação da arquitectura global da LPS ARQUO™. Neste componente, encontram-se os elementos de configuração correspondentes à documentação da metodologia para a implementação e gestão da LPS ARQUO™, Arquitectura "*alto-nível*" do sistema (incluindo as vistas requisitos, módulo e componente & conector) e Plano de Testes global do sistema.

A título ilustrativo, na Tabela 53 estão apresentadas possíveis versões do mesmo elemento de configuração em três ramos de configuração do ARQUO™.

		Linha de Configuração		
Elemento de Configuração	Tipo de ficheiro	LCP_LPSARQUO	LCP_Cliente1	LCP_Cliente2
Componente de Configuração : ImportServer				
\code\CConsumer.cpp	*Source C++*	V.2.2	V.2.1	V.2.2
\code\CDocImportServer.cpp	*Source C++*	V.3.0	V.2.2	V.3.0
\code\CFileParser.cpp	*Source C++*	V.3.0	V.2.2	V.2.5
\code\CFileSession.cpp	*Source C++*	V.4.0	V.3.4	V.3.5
\code\CDocImportInterceptor.cpp	*Source C++*	-	V.1.0	-
\code\CImportServer.cpp	*Source C++*	V.1.0	-	-
\header\CConsumer.h	*Source Header C++*	V.2.2	V.2.1	V.2.2
\header\CDocImportServer.h	*Source Header C++*	V.3.0	V.2.7	V.3.0
\header\CImportServer.h	*Source Header C++*	V.1.0	-	-
\header\CFileParser.h	*Source Header C++*	V.3.0	V.2.2	V.2.5
\header\CFileSession.h	*Source Header C++*	V.4.0	V.3.4	V.3.5
\code\CDocImportInterceptor.h	*Source Header C++*	-	V.1.0	-
\interface\docimportserver.idl	*Sorce IDL CORBA*	V.2.0	V.2.0	V.2.0
\interface\makefile	*Makefile*	V.2.0	V.2.0	V.2.0
\design\vm\MC_ImportServer.xml	Modelo Classes: Ficheiro XML	V.1.0	-	-
\design\vcc\MCC_ImportServer.xml	Modelo C&C: Ficheiro XML	V.1.0	-	-
\design\vr\MR_ImportServer.xml	Modelo Requisitos: Ficheiro XML	V.1.0	-	-
\design\vr\MCU_ImportServer.xml	Modelo Casos Utilização: Ficheiro XML	V.1.0	-	-
\design\vr\MCAQ_ImportServer.xml	Modelo Cenários Atributos Qualidade: Ficheiro XML	V.1.0	-	-
\design\vr\MD_ImportServer.xml	Modelo Dominio: Ficheiro XML	V.1.0	-	-

Tabela 53 – Elementos de Configuração do Componente "*ImportServer*" da LPS ARQUO™ nas LCP da LPS, Cliente 1 e Cliente2

Por estarem a ser desenvolvidas de raiz, na tabela estão algumas versões de elementos de configuração apenas no ramo de configuração LCP_LPSARQUO, nomeadamente o elemento de configuração "*CImportServer*" (classe abstracta C++, que mapeia um ponto de variabilidade) e os artefactos resultantes da implementação das vistas arquitecturais: vista requisitos, módulo e C&C. Esta situação ocorrerá sempre que sejam desenvolvidos novos elementos de configuração ao nível da LPS e ainda não

constam de nenhuma *release* da LPS e\ou não foram actualizados nos projectos Cliente1 ou Cliente2. Na próxima actualização da *release* da LPS nos ramos de configuração LCP_Cliente1 e LCP_Cliente2, os novos elementos de configuração serão reflectidos.

Por outro lado, na linha de configuração LCP_Cliente1 existem igualmente dois elementos de configuração (*CDocImportInterceptor.cpp* e *CDocImportInterceptor.h*) que não estão na LCP_LPSARQUO e LCP_Cliente2. Este facto significa que este desenvolvimento foi realizado especificamente para o projecto em causa e não faz parte da LPS (o módulo de software respectivo está classificado nestes casos como <<*application*>>).

V.3.4.5.Gestão de Configurações Multi-Produto para a LPS ARQUO™

Um dos pressupostos para a realização de gestão de configurações multi-produto na LPS ARQUO™ é a existência de uma única ferramenta de gestão de configurações sob a qual são geridas as linhas de configurações dos projectos de engenharia de domínio e engenharia aplicacional.

A ferramenta de gestão de configurações de software deverá permitir a analise das configurações dos elementos de configuração no tempo, i.e. a evolução das versões de um componente de configuração ao longo do tempo, e no espaço, i.e. as diferentes versões do mesmo componente de configuração nos produtos da LPS, num determinado momento (ver requisito P6 em I.3.).

Gestão de configurações no Tempo

No ARQUO™, a obtenção das diferentes configurações de um elemento de configuração no tempo realiza-se pela análise do histórico de versões desse elemento de configuração, bem como dos *baselines* em que este foi alvo ao longo do tempo.

A Figura 43, ilustra uma possível representação da gestão de configurações de um elemento de configuração no tempo.

Como é possível verificar pela Figura 43, um elemento de configuração pode sofrer alterações ao longo do tempo, tanto na linha de configuração de integração (ou desenvolvimento) da LPS como nos projectos onde o componente de configuração faz parte do âmbito do projecto (Client1 ou Cliente2 na figura).

As alterações deverão ficar documentadas com a referência ao requisito (ID_REQ) ou correcção de erro (ID_BUG) na origem da necessidade de alteração. Em cada versão deverá ficar armazenada igualmente a data de alteração e a linha de configuração na qual foi criada uma nova versão.

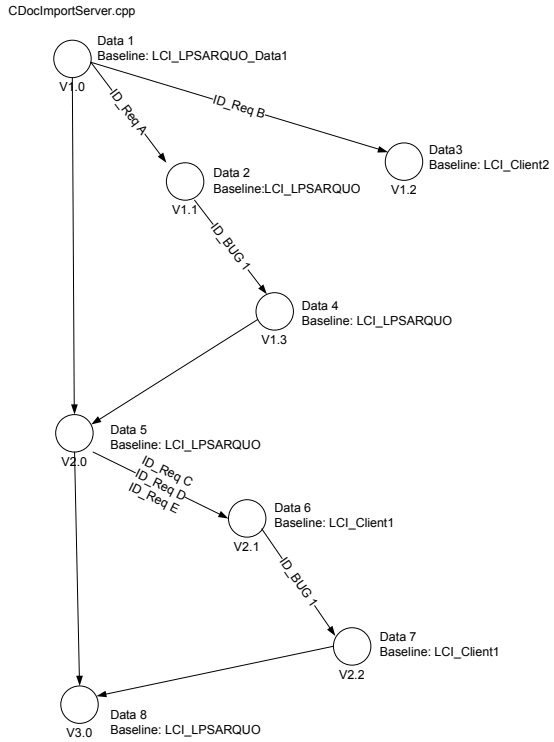


Figura 43 – Evolução de versões de um elemento de configuração no Tempo

Gestão de configurações no Espaço

A obtenção das versões de um elemento de configuração unitário, em simultâneo nas linhas de configuração dos vários produtos existentes e LPS é possível pela análise dos *baselines* associados a cada linha de configuração.

Como um *baseline* de um componente de configuração contém uma versão de cada elemento de configuração, realizado num determinado momento do tempo sobre uma linha de configuração, é sempre possível identificar:

- As versões de elementos de configuração que estão presentes num determinado *baseline* (pode ser realizada uma pesquisa pelos atributos do elemento *baseline* em IV.4.4.4) e linha de configuração.
- A versão actual de um elemento em cada linha de configuração.

V.3.4.6. Rastreabilidade entre os elementos das vistas arquitecturais e elementos de configuração

Na vista produto do ARQUO™ são geridas as versões de todos os elementos de configuração, sejam os modelos de requisitos, atributos de qualidade, módulos ou C&C, código fonte associado, *makefiles* ou ficheiros de configuração. O principal desafio é saber num determinado momento ou para uma determinada instalação:

- Qual o impacto que uma alteração de um requisito tem no código existente?

- Quais os módulos de software que implementam um requisito em particular?
- Que módulos de software inclui um determinado componente de execução?
- Qual o impacto na arquitectura do produto, provocado por uma decisão táctica?

Os elementos das vistas do modelo SPL-LIGHT, são classificados com base num conjunto de etiquetas específicas. A relação entre os elementos dos modelos, e.g. requisito, caso de utilização, cenário de atributo de qualidade, módulo, componente, conector é definida explicitamente pela associação de identificadores unívocos a cada elemento e nas relações com outros elementos, como o ID Requisito.

A vantagem de utilizar metadados associados a cada elemento na vista produto permite, para além do registo de versões, armazenar no repositório informação que classifica uma determinada alteração, no tempo (atributos de data), espaço (tipo de linha de configuração e projecto), grau de estabilidade (tipo de *baseline*) ou âmbito (*ID_Requisito, ID_Pedido* associado aquando do "*Check-in*"). Toda esta meta-informação pode ser pesquisada permitindo fornecer aos elementos da equipa de engenharia um maior contexto das alterações aos elementos de software base da LPS. Na Tabela 54, é apresentado um exemplo de uma pesquisa por todos os elementos da arquitectura que implementem o requisito "*IMP.1.3. Arquiva Documento*" do ARQUO™.

ID	Descrição	Tipo de Elemento	Elemento de Configuração
IMP.1.1.	Importação de Índices de Documentos	Requisito	*MR_ImportServer.XML*
IMP.1.1.1	Importação de índices em formato XML	Requisito	*MR_ImportServer.XML*
IMP.1.1.2.	Importação de índices em formato standard ARQUO™	Requisito	*MR_ImportServer.XML*
12	Importação de Índices de Documentos	Caso de Utilização	*MCU_ImportServer.XML*
13	Importação de índices em formato XML	Caso de Utilização	*MCU_ImportServer.XML*
14	Importação de índices em formato standard ARQUO™	Caso de Utilização	*MCU_ImportServer.XML*
15	Verificação da correcção da localização dos objectos	Caso de Utilização	*MCU_ImportServer.XML*
II.	*DocImportInterceptor*	Módulo	*cDocImportINterceptor.cpp cDocImportInterceptor.h cFileParser.cpp, FileParser.h StraightSymbolSeparated Parser.cpp, main.cpp, aightSymbolSeparatedPar ser.h, DocImportEventLogger.cp p DocImportEventLogger.h*
IDL.4	*DocImportInterceptor*	Módulo	*DocImportInterceptor.IDL*
con.IS.1	*IDL CORBA DocImportInterceptor*	Conector	*DocImportInterceptor.IDL*
com.IS.1	*DocImportInterceptor*	Componente	*DocImportInterceptor.exe* (poderá não ser alvo de gestão de versões)

Tabela 54 – Elementos da Arquitectura da LPS associados com o requisito "*IMP.1.1. Importação de Índices de Documentos*".

V.4. CONSIDERAÇÕES FINAIS

A LPS ARQUO™ é adequada à aplicação de uma abordagem de desenvolvimento orientada a LPS. A sua arquitectura orientada a serviços permite a gestão e evolução autónoma dos elementos de software que constituem cada produto da LPS.

O processo de implementação da LPS está em curso, sendo o objectivo da primeira fase a definição da arquitectura da LPS e respectivas vistas (requisitos, módulo e componente e conector), reflectindo sempre que adequado os pontos de variabilidade existentes ao nível dos vários produtos.

Em paralelo e numa segunda fase está a ser realizada a reengenharia dos elementos base da LPS, de forma a introduzir os pontos de variabilidade e agilizar o processo de instanciação dos produtos às necessidades específicas dos clientes.

A gestão de configurações multi-produto necessária à implementação da vista produto é um dos aspectos a melhorar na LPS ARQUO™, de forma a tirar o máximo partido da visão integrada das versões dos elementos de configuração ao longo do tempo e no espaço (i.e. para as várias instâncias dos produtos instalados nos clientes).

O objectivo da consciencialização dos elementos da equipa de engenharia de software e arquitecto de software foi alcançado com o trabalho na medida em que hoje a organização "pensa" numa perspectiva de LPS, sendo os desenvolvimentos aplicacionais específicos tratados como tal e os desenvolvimentos sobre os elementos base alvo de um especial cuidado, com supervisão pelo arquitecto da LPS.

VI. CONCLUSÕES

Neste capítulo estão apresentadas as principais conclusões do trabalho realizado face aos objectivos propostos no âmbito da aplicação de uma LPS ao contexto de uma PME.

Na secção VI.1, são apresentadas as principais contribuições do trabalho para a empresa TIE no contexto da aplicação prática da metodologia ao ARQUO™ bem como para a comunidade de PMEs em geral, com necessidades semelhantes.

A identificação de áreas de trabalho futuro, na sequência do trabalho realizado, é apresentada em VI.2.

VI.1. CONTRIBUIÇÕES DO TRABALHO

Este trabalho teve como objectivo o desenvolvimento de uma metodologia para a gestão e implementação de uma LPS, adequada à realidade de uma PME, nomeadamente no contexto da empresa TIE e da linha de produtos ARQUO™ (ver I.2).

Nesse contexto, foi desenvolvida uma metodologia para a implementação e gestão de uma LPS, que prevê a definição da arquitectura de software da LPS com base num conjunto de produtos já existentes. A definição de uma arquitectura intermédia resultante da refactorização e reengenharia dos elementos de software existentes, verificou-se ser adequada pois permite definir um alvo para a migração dos produtos existentes bem como um ponto de partida para a evolução dos elementos de software base da LPS. A optimização do esforço de migração dos produtos existentes e definição do âmbito da LPS logo numa fase inicial são dois dos principais benefícios verificados.

O caso prático demonstra que o método para a representação da arquitectura de software de uma LPS (SPL-LIGHT) proposto, baseado em 3+1 vistas arquitecturais, endereça os requisitos descritos no contexto do problema (I.3). O método satisfaz os requisitos específicos da TIE, mas que poderão ser endereçados de forma genérica para organizações com necessidades semelhantes.

Em particular, é de realçar algumas das características do modelo proposto no contexto da representação da arquitectura de uma LPS:

- Representação da arquitectura de uma LPS, segundo as perspectivas dos vários intervenientes, e.g. arquitecto da LPS, analista funcional, engenheiro de software, integrador produtos, utilizador final.
- Representação explícita e classificação dos elementos de software base, respectivos pontos de variabilidade e variantes da LPS, através de estereótipos e etiquetas UML definidas num perfil UML proposto.
- Representação de cenários de atributos de qualidade que justificam as decisões arquitecturais ou tácticas, com impacto na selecção dos variantes arquitecturais.
- Identificação unívoca de cada requisito da LPS, respectivos pontos de variabilidade e variantes, associando-os de forma explícita através de etiquetas UML, aos casos de utilização, módulos de software na vista módulo, componentes de execução e conectores.

- Vista produtos, com a perspectiva da representação multi-dimensional dos componentes de configuração (agregam os artefactos das restantes três vistas) que compõem a arquitectura da LPS e os produtos de configuração instanciados a partir dela.

Para a TIE, a aplicação da metodologia proposta ao produto ARQUO™ teve como primeiro benefício o facto de ter colocado a organização a "pensar" numa lógica de desenvolvimento baseado em LPS. Hoje, na TIE existe uma clara segregação entre a engenharia de domínio e a engenharia aplicacional, o que num curto espaço de tempo permitiu a definição de um plano de evolução ambicioso para o ARQUO™ (realizado pela equipa de engenharia interna) e delegação em parceiros, grande parte da actividade de engenharia aplicacional.

O processo de adopção de uma LPS pelo ARQUO™ será progressivo, sendo que o caso prático focou essencialmente as actividades de análise dos produtos existentes, definição da arquitectura intermédia e definição da arquitectura da LPS.

A metodologia proposta no âmbito deste trabalho teve aplicação prática o ARQUO™ da TIE, no entanto, esta metodologia poderá ser aplicada a qualquer outra organização e LPS candidata, que se enquadre no contexto do problema (I.3).

VI.2. TRABALHOS FUTUROS

O trabalho realizado aborda diversas áreas no contexto da engenharia de software, arquitecturas de software, implementação e gestão de LPS e gestão de variabilidades. A abrangência dos temas é muito grande pelo que o foco do trabalho foi restrito ao considerado essencial, estritamente útil para a definição de uma LPS no contexto do problema a resolver.

Neste contexto, poderá ser realizado um trabalho de maior detalhe na definição de procedimentos, padrões e ferramentas para a aplicação prática da metodologia SPL-LIGHT. Em especial esse trabalho teria especial interesse:

- Na refactorização dos elementos de software dos produtos existentes com vista a torná-los reutilizáveis no contexto da arquitectura intermédia de software da LPS;
- Na implementação dos vários mecanismos de variabilidade, ao longo do processo de desenvolvimento, i.e. definição da arquitectura, desenvolvimento, compilação, tempo de execução;
- Na instanciação dos produtos e pontos de variabilidade dos elementos de software, a partir da linha de configuração principal da LPS.

Ao nível do modelo SPL-LIGHT para a documentação da arquitectura de software da LPS seria interessante realizar como trabalho futuro:

- O desenvolvimento de uma ferramenta ou módulo adicional integrada com a ferramenta de modulação utilizada (no âmbito do trabalho foi realizado a ferramenta *Enterprise Architect* da *Sparx Systems*, mas poderia ter sido outra), que facilitasse a pesquisa e navegação entre os elementos das várias vistas arquitecturais, relacionados através de etiquetas UML (*tagged values), e.g*. requisito, módulo de software e componente respectivo;
- Incluir uma definição formal, não ambígua, das limitações e constrangimentos aplicáveis a cada estereótipo do perfil UML proposto, substituindo a linguagem natural utilizada, por uma linguagem de especificação formal, e.g. OCL.

VII. REFERÊNCIAS BIBLIOGRÁFICAS

[1] TIE, "TIE - Tecnologias de Integração Empresarial. www.tie.co.pt."

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, Second Edition*: Addison Wesley, 2003.

[3] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2002.

[4] P. Kruchten, "Architectural Blueprints—The "4+1" View Model of Software Architecture," in *IEEE Software 12 (6)*, November 1995 ed, 1995, pp. 42-50.

[5] R. Software, "Rational Unified Process (RUP)," in *http://www-306.ibm.com/software/rational/*, IBM, Ed.

[6] "Rational Unified Process (RUP)," in *http://www.wthreex.com/rup/*, Wthreex, Ed.

[7] D. Soni, R. L. Nord, and C. Hofmeister, "Software architecture in industrial applications," presented at 17th international conference on Software Engineering Seattle, Washington, United States 1995.

[8] K. Farooqui, L. Logrippo, and J. d. Meer, "The ISO Reference Model for Open Distributed Processing - An Introduction," vol. Introduction into the ODP Reference Model,, 1996.

[9] DoD, *C4ISR Architecture Framework*, vol. Version 2.0: AWG - Arquitectures Working Group, 1997.

[10] P. Herzum and O. Sims, "A Comprehensive Overview of Component-Based Development for the Enterprise," in *Business Component Factory*: Herzum Software, 1999.

[11] J. A. Zachman, "A framework for information systems architecture," *IBM Systems Jornal*, vol. 25, pp. 454-470, 1987.

[12] C.W.Krueger, *Software Reuse*, vol. 24, 1992.

[13] Jacobson, *Software Reuse: Architecture, Process and Organization for Business Sucess*, 1997.

[14] Poulin, *Measuring Software Reuse: Principles, Pratices and Economic Models*, 1997.

[15] J. Bosch, "Software Architecture, Components and Reuse," 2001.

[16] E. S. d. Almeida, A. Alvaro, V. C. Garcia, J. C. C. P. Mascena, V. A. d. A. Burégio, L. M. d. Nascimento, D. Lucrédio, and S. L. Meira, *C.R.U.I.S.E. Component Reuse In Software Engineering*, 2007.

[17] J. Bosh, *Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach*, 2000.

[18] CAFE, "From Concepts to Application in System-Family Engineering," Eureka ! 2023 Programme, ITEA project ip00004, 2001.

[19] ESAPS, "Engineering Software Architectures, Processes and Platforms for System-Families," I. p. Eureka ! 2023 Programme, Ed., 2001.

[20] J. Bayer, O. Flege, P. Knauber, Roland Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "PuLSE: A Methodology to Develop Software Product Lines," 1999.

[21] J. Bosch, M. Svahnberg, and J. v. Gurp, "On the Notion of Variability in Software Product Lines," 2001.

[22] G. J. Chastek and J. D. McGregor, "Integrating Domain Specific Modeling into the Production Method of a Software Product Line," 2005.

[23] D. Weiss, "Family-Oriented Abstraction, Specification, and Translation: The FAST Process," 1995.

[24] T. Ziadi, L. Hélouet, and J.-M. Jézéquel, "Towards a UML Profile for Software Product Lines," 2006.

[25] L. Dobrica and E. Niemelä, "UML Notation Extensions for Product Line Architectures Modeling," 2005.

[26] M. V. Cengarle, P. Graubmann, and S. Wagner, "From Feature Models to Variation Representation in MSCs," 2005.

[27] M. V. Cengarle, P. Graubmann, and S. Wagner, "Semantics of UML 2.0 Interactions with Variabilities," 2005.

[28] M. Clauß, "Modeling variability with UML," 2001.

[29] S. ROBAK, B. FRANCZYK, and K. POLITOWICZ, "EXTENDING THE UML FOR MODELLING VARIABILITY FOR SYSTEM FAMILIES," *Int. J. Appl. Math. Comput. Sci.,*, vol. 12, pp. 285–298, 2002.

[30] P. Clements and L. Northrop, *Software Product Lines - Practices and Patterns*, vol. 1, 3rd ed: Addison-Wesley, 2005.

[31] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison Wesley, 2001.

[32] P. Clements and L. Northrop, "Software Product Lines," Carnegie Mellon University, Pittsburgh 2003.

[33] C. Atkinson, J. Bayer, and D. Muthig, "Component-Based Product Line Development: The KobrA Approach," presented at First Software Product Line Conference, 2000.

[34] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), 1998.

[35] Kang, Cohen, J. Hess, Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1990.

[36] U. E. Krzysztof Czarnecki, "Generative Programming: Methods, Tools, and Applications."

[37] A. Jansen, R. Smedinga, J. v. Gurp, and J. Bosch, "Feature-Based Product Derivation: Composing Features," 2002.

[38] L. Zhu and I. Gorton, "UML Profiles for Design Decisions and Non-Functional Requirements," presented at ICSE2007, 2007.

[39] L. M. Northrop, "SEI's Software Product Line Tenets," *IEEE Jornal July/August 2002*, 2002.

[40] I. Philippow, D. Streitferdt, and M. Riebisch, "Design Pattern Recovery in Architectures for Supporting Product Line Development and Applica t ion," 2001.

[41] SEI, "Software Engineering Institute - A Framework for Software Product Line Practice," 2006.

[42] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed: McGraw-Hill, 2005.

[43] SEI, "Software Configuration Management (SCM). ," C. M. U. S. E. I. (SEI), Ed., 2007.

[44] I. C. Society, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," The Institute of Electrical and Electronics Engineers, Inc., New York,USA 2000.

[45] J. A. Zachman, "Zachman Framework," http://www.zifa.com/framework.pdf.

[46] Rational, "Unified Change Management from Rational Software: An Activity-Based Process for Managing Change," 2004.

[47] O. M. G. (OMG), "Object Management Group (OMG). www.omg.org."

[48] O. M. G. (OMG), "UML OCL Specification," in *http://www.omg.org/cgi-bin/doc?formal/06-05-01*, version 2.0 ed: formal/06-05-01 2006.

[49] O. M. G. (OMG), "UML 2.0 Superstructure Specification," vol. formal/05-07-04: http://www.omg.org/cgi-bin/doc?formal/05-07-04, 2005.

[50] O. M. G. (OMG), "UML 2.0 Infrastructure Specification," vol. formal/05-07-05 http://www.omg.org/cgi-bin/doc?formal/05-07-05, 2006.

[51] O. M. G. (OMG), "UML Diagram Interchange Specification," vol. formal/06-04-04 http://www.omg.org/cgi-bin/doc?formal/05-07-05, 2006.

[52] OMG, "MDA Guide Version 1.0.1," *OMG*, 2003.

[53] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based* Addison Wesley, 2004.

[54] C. W. Krueger, "Variation Management for Software Production Lines," *SPLC02*, 2003.

[55] SAP, "www.sap.com."

[56] Oracle, "http://www.oracle.com/index.html."

[57] T.-T. d. I. Empresarial, "www.tie.co.pt."

[58] J. Robertson and S. Robertson, "Volere Requirements Specification Template - Edition 11—February 2006," 2006.

[59] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. R. O. Silva, "Documenting Component and Connector Views with UML 2.0," Software Architecture Technology Initiative April 2004 2004.

[60] Perforce, "Perforce Software. www.perforce.com."

[61] G. Booch, *Object-Oriented Analysis and Design with Applications*,, vol. 2º Edition. Redwood City, California: Benjamin-Cummings Pub. Co.,, 1993.

[62] M. Matinlassi, E. Niemela, and L. Dobrica, "Quality-driven architect design and quality analysis method," vol. ESPOO2002, 2002, pp. 141.

[63] H. Gomaa and M. E. Shin, "Variability in Multiple-View Models of Software Product Lines," presented at Intl Workshop on Software Variability Management,, Oregon, 2003.

APÊNDICES

No capítulo I. é apresentada uma contextualização do tema da LPS e principais conceitos. Este capítulo, apresenta o trabalho de síntese do "estado da arte", dando especial ênfase à representação da arquitectura da LPS e o processo para a implementação e gestão da LPS.

Como extensão do trabalho de síntese é apresentado nos apêndices seguintes um resumo comparativo das principais abordagens existentes bem como uma análise da sua adequação ao contexto do problema definido em I.3. Não foi no entanto preocupação deste trabalho, a análise exaustiva das várias abordagens.

Assim, em Apêndice I. é apresentado uma análise a um conjunto de abordagens para a documentação de arquitecturas de software, em Apêndice II. um levantamento de Perfi*s* UML para a representação de variabilidades numa LPS e no Apêndice III. é apresentado uma análise de várias abordagens para a implementação e gestão de uma LPS.

Apêndice I. ABORDAGENS PARA A DOCUMENTAÇÃO DE ARQUITECTURAS DE SOFTWARE

Apêndice I.1 "4+1" vistas de Rational Unified Process/ Kruchten

Em 1995, Philippe Kruchten, publicou um artigo no qual apresentava as suas "quatro mais uma" vistas para as arquitecturas de software [4]. Esta aproximação foi adoptada no modelo conceptual da metodologia *Rational Unified Process* (RUP) [6].

Vista	Descrição
Vista Lógica	A vista lógica suporta os requisitos funcionais do sistema, ou seja as funcionalidades que o sistema deve dispor aos utilizadores. A identificação dos objectos, ou classes de objectos da vista lógica é obtida pela abstracção dos conceitos chave obtidos do domínio do problema. Notação: diagramas de classes do UML (com definido no RUP), ou por outra notação informal, como a definida por Booch [61].
Vista de Processamento	A vista de processamento foca na representação os requisitos não funcionais ou de qualidade do sistema, nomeadamente a desempenho e disponibilidade da execução num ambiente de produção. Aspectos como a concorrência, distribuição e tolerância a falhas são correspondidos na instanciação dos objectos da vista lógica em tarefas, comunicações e recursos computacionais dos processos em execução. Notação: diagramas de componentes UML ou a notação para a representação dos processos de Booch [61].
Vista de Implementação	A vista de implementação focaliza-se na organização dos módulos do sistema no ambiente de desenvolvimento do software. A vista de desenvolvimento suporta a afectação de requisitos e de equipas de trabalho, suporta também a avaliação de custos, o planeamento, a monitorização do progresso de projecto, e o estudo sobre a reutilização, portabilidade e segurança. Notação: diagramas de pacotes e componentes UML ou notação Booch. [61]
Vista de Instalação	A vista de instalação tem em conta os requisitos do sistema, tais como desempenho, disponibilidade e fiabilidade (tolerância a falhas). Esta vista faz a correspondência dos vários elementos identificados na vista lógica, de processamento e de implementação – redes, processos, tarefas, e objectos - com os nós de comunicação e de processamento no ambiente de execução. Notação: diagramas de componentes e de instalação UML.
Vista de casos de utilização	A aproximação "quatro mais uma" foi acolhida desde a sua apresentação como uma peça fundamental do *Rational Unified Process*, enquanto metodologia conduzida por casos de utilização. Esta vista tem em consideração os casos de utilização importantes, de forma a definir, por um lado, a arquitectura nas fases iniciais do processo e, por outro, validar mais tarde a percepção das restantes vistas. Notação: casos de utilização UML.

Tabela 55 – "4+1" vistas de *Rational Unified Process/ Kruchten*

Apêndice I.2 As quatro vistas da Siemens

Soni, Nord, Hofmeister da Siemens Corporate Research [7], em 1995 apresentaram igualmente uma abordagem para a documentação da arquitectura de software, nomeadamente para aplicações da indústria. A abordagem da Siemens consiste na documentação da arquitectura através de quatro vistas (vista conceptual, vista módulo, vista de código e vista de execução).

Na concepção de cada vista são tidos em consideração várias actividades: análise global, desenho central e desenho final. Na análise global são identificados e analisados os factores que influenciam a arquitectura de forma a delinear a arquitectura. Desta actividade resulta um documento de análise dos factores que influenciam a arquitectura e o racional sobre as decisões de desenho que foram tomadas. Nas actividades de desenho são definidos os elementos, as relações entre eles e as propriedades mais importantes da arquitectura.

Vista	Descrição
Vista conceptual	Representa as funcionalidades do sistema que são correspondidas no que diz respeito a componentes e conectores; esta vista é a mais próxima do domínio da aplicação, porque é a menos limitada pelo software ou hardware.
Vista módulo	Representa, os componentes, conectores, portas e papéis e as suas interfaces. O sistema é decomposto em módulos e em subsistemas. Um módulo pode também ser atribuído a uma camada, a qual limita as dependências com outros módulos.
Vista de execução	Representa a afectação dos componentes de execução do sistema na plataforma de execução, nomeadamente através de processos e bibliotecas partilhadas.
Vista de código	Nesta vista o foco principal é a representação do software que implementa o sistema e a forma como este está organizado

Tabela 56 – As quatro vistas da Siemens

Uma das virtudes deste modelo é a ligação apresentada entre os elementos (estruturas) das várias vistas. As estruturas conceptuais (da vista conceptual) são "*implementadas-por*" estruturas módulo (da vista módulo), e "*atribuídas-a*" estruturas de execução (da vista de execução). As estruturas módulo podem estar "*localizadas-em*" ou "*implementadas-por*" estruturas código (da vista código). As estruturas código podem configurar estruturas de execução.

Apêndice I.3 ISO RM-ODP (Reference Model of Open Distributed Processing)

O RM-ODP (sigla de *Reference Model of Open Distributed Processing*) da *International Organization for Standardization* (ISO) [8] foi criado para servir como modelo de referência para a descrição de sistemas distribuídos abertos e é aceite actualmente como um padrão de facto. Este modelo apresenta cinco vistas que devem ser seguidas

para que o desenvolvimento de um sistema de informações seja compatível com este padrão, na Tabela 57 são apresentadas resumidamente essas cinco vistas.

Vista	Descrição
Vista Organizacional	Descreve o sistema de informações em termos de o que o mesmo deve fazer. As necessidades e especificações administrativas e técnicas que guiam e justificam o projecto do sistema também são capturadas neste vista. A vista Organizacional do RM-ODP captura os principais objectivos e restrições do sistema em desenvolvimento. Notação: Casos de Utilização UML.
Vista de Informação	Descreve o sistema de informação em termos de estruturas, fluxo de informações e restrições relacionados com a manipulação das mesmas Os diagramas de Classes UML podem representar de forma adequada esta vista do RM-ODP. Os diagramas de estado e transição complementam a representação interna do comportamento dos objectos, mostrando as diferentes fases de seu ciclo de vida e os eventos específicos (métodos) que podem causar a transição de um estado para outro. Notação: Diagramas de Classes, Estado e Transição.
Vista Computacional	Descreve o sistema de informação em termos de operações e características computacionais do processo de transformação de informação. Notação: Diagramas de sequência e de actividades UML.
Vista Tecnológica	Descreve o sistema de informação em termos dos elementos de software que o sistema é construído. Na especificação da Vista Tecnológica, os componentes de execução, hardware, redes, etc. devem ser incluídos como implementação dos elementos de software do Modelo de Engenharia, incluindo componentes de localização, infra-estruturas, funções ODP, interceptores, etc. Notação: Não existe uma notação formal para este tipo de vista.
Vista de Engenharia	Descreve o sistema de informação em termos de recursos de engenharia para suportar a natureza distribuída do processamento. A Vista de Engenharia pode ser representada através do Modelo de Engenharia da ODP [8]. Este é um modelo abstracto para expressar os conceitos do tipo de Vista Tecnológica. Envolve conceitos tais como sistemas operativos, sistemas de comunicação (protocolos, redes), processadores, armazenamento, entre outros. O modelo foca-se essencialmente nos serviços que devem ser fornecidos ás aplicações e quais os mecanismos que devem ser usados para obter esses serviços. Existem tipos diferentes de objectos de engenharia no modelo de engenharia correspondendo as diferentes funções de distribuição (permitir, regular, esconder) necessárias num ambiente distribuído.

Tabela 57 – Vistas do modelo ISO RM-ODP

A Figura 44, mostra uma representação geral dos diagramas UML dentro da estrutura do RM-ODP. Esta imagem foca apenas as três primeiras vistas do RM-ODP uma vez que, nestas vistas, se encontram os principais aspectos relativos à análise e projecto de um sistema de informações. Os Casos de Utilização e Categorias aparecem como principais elementos na representação da Vista Organizacional, guiando as vistas

subsequentes através de objectos, que são os principais elementos das vistas 2 e 3, Informação e Computacional, respectivamente. Na Vista de Informações são utilizados os diagramas de Classes e diagramas de Estados e Transição, enquanto que na Vista Computacional são utilizados os diagramas de Sequência e diagramas de Actividades.

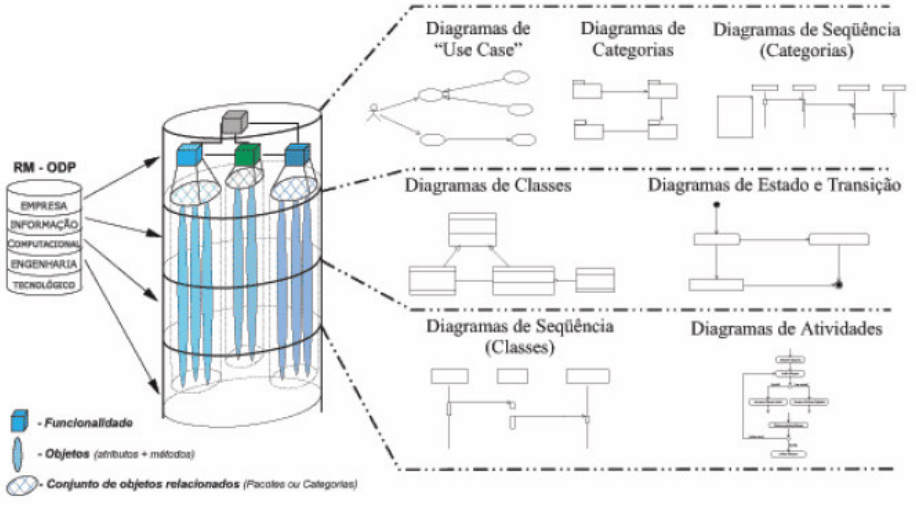


Figura 44 – Notação UML aplicada ás três primeiras vistas do RM-ODP, retirado de [8]

Apêndice I.4 Abordagem Paul Clements et al

A abordagem de Paul Clements et al [3], do *Software Engineering Institute* (SEI) para a documentação da arquitectura de software baseia-se na identificação das vistas relevantes para os intervenientes e a documentação para além das vistas que se aplica a mais que uma vista. Nesta abordagem, as vistas estão agrupadas em vista: Módulo, Componente & Conector e Afectação.

Cada tipo de vista poderá conter mais que um estilo arquitectural para a descrever. Um estilo especializa os elementos e tipos de relações de um tipo de vista, e.g., o tipo de vista módulo pode ser representado através de vários estilos: decomposição, usa, generalização, cada qual especializa o elemento "módulo" e relações específicas.

As vistas do tipo de vista Módulo, documentam as principais unidades de código (módulos). As vistas do tipo de vista Componente & Conector documentam a interacção dos elementos de execução do sistema (componentes e conectores). E as vistas do tipo de vista Afectação documentam as relações entre o software do sistema e os ambientes de desenvolvimento e de execução. Na Tabela 58, são apresentadas em mais pormenor estes três tipos de vista, os seus elementos, relações e estilos e é apresentada a notação usada para os representar.

Tipo de Vista	Descrição
Tipo de Vista Módulo	Este tipo de vista consiste na decomposição do sistema em módulos, ou seja, unidades de código / implementação. Os módulos são áreas de responsabilidade funcional, e são atribuídos ás equipas de implementação. As relações entre módulos incluem "*é-uma*" (define uma relação de generalização entre um módulo mais especifico - o filho A - e um módulo mais geral - o pai B), "*é-parte-de*" (de uma forma geral a relação simplesmente indica agregação) e "*depende-de*" (define a relação dependência entre A e B). São identificados quatro estilos pertencentes a este tipo de vista: camadas, usa, decomposição e generalização. Notação: Representações informais, que podem ser gráficas ou textuais ao uso do UML. Diagramas de classes, componentes, pacotes ou estereótipos UML como o subsistema ou interface. As relações podem ser representadas através da generalização, composição ou dependência UML.
Tipo de Vista Componente & Conector	Na vista Componente & Conector os elementos são os componentes (principais unidades de processamento e de armazenamento de dados que um sistema executa) e os conectores (que são as ligações de comunicação entre componentes). As vistas C&C descrevem os elementos durante a execução, e.g. os processos, os objectos, os clientes, os servidores e as interacções entre eles, e.g. ligações de comunicação, protocolos ou fluxos de informação. A principal relação na vista C&C é a ligação entre componentes e conectores. Os portos dos componentes são associados a papéis dos conectores. Um porto *po* de um componente pode ser ligado a um papel *pa* de um conector se o componente interage através do conector, utilizando a interface descrita por *po* e conforme as expectativas descritas por *pa*. No tipo de vista C&C são vários os estilos que representam o tipo de comunicação entre os componentes num ambiente de execução: estilo *cliente-servidor*, *canais-e-filtros*, *publicação-subscrição*, *par-a-par*, dados partilhados e processos comunicantes. Notação: diagramas de "*caixas e linhas*" informais, Acme , Classes e pacotes UML. A relação entre classes e objectos é igual á relação entre os tipos arquitecturais e as suas instâncias
Tipo de Vista Afectação	As vistas pertencentes ao tipo de vista afectação mostram a relação entre os elementos de software e os elementos num ou mais ambientes externos onde o software é criado e executado. O tipo de vista afectação apresenta a correspondência dos elementos de qualquer um dos estilos dos tipos de vista módulo ou componente & conector em elementos do ambiente de execução. São identificados três estilos mais comuns: o estilo instalação que descreve a correspondência entre os componentes e os conectores no hardware onde é executado o software; o estilo implementação que descreve a correspondência dos módulos num ficheiro de sistema que contém esses módulos; e o estilo trabalho que descreve a correspondência dos módulos nas pessoas, grupos ou equipas que têm como tarefa o desenvolvimento dos módulos. Notações: estilo Instalação: notações informais ou diagrama de instalação; estilo de Implementação Qualquer notação para o estilo de implementação deve conter módulos, os itens de configuração e a correspondência entre eles; estilo trabalho: tabela que relaciona as pessoas ou equipas com o módulo que lhe foi atribuído.

Tabela 58 – Os tipos de vista do modelo do SEI (Paul Clements et al)

Apêndice I.5 *Framework* de Zackman

A *framework* de Zachman [45], da autoria de John A. Zachman, foi adoptado por organizações de TI como a *framework* para identificar e disciplinar as várias perspectivas envolvidas numa arquitectura de software empresarial. Na prática, a *framework* de Zachman apresenta um conjunto de perspectivas da arquitectura de um sistema. Estas perspectivas são descritas numa tabela bidimensional onde as linhas contêm as vistas dos diferentes interlocutores (ver Tabela 59) e as colunas os aspectos da arquitectura (ver Tabela 60).

Vistas (Linhas da Framework de Zachman)	**Descrição**
Âmbito (Vista do Resp. Planeamento)	Define a direcção e orientação do sistema, definindo o contexto e os limites da arquitectura a ser representada.
Modelo Organizacional (Vista Resp. Negócio)	Define em termos do negócio a natureza da organização incluindo a estrutura, processos e organização.
Modelo de Sistema (Vista do Arquitecto)	Define a organização com um maior nível de detalhe que o Modelo Organizacional. Esta linha foi inicialmente definida por vista do desenho do sistema de informação.
Modelo Tecnológico (Vista do Resp. Desenho)	Define a forma como a tecnologia será aplicada de forma a endereçar as necessidades definidas pelas vistas anteriores.
Representação Detalhada (Vista do Resp. Desenvolvimento)	Define o desenho detalhado, segundo a linguagem de implementação, armazenamento em base de dados e considerações de *middleware*.

Tabela 59 – Vistas da *Framework* de Zahman

A Tabela 60 descreve os Aspectos da Arquitectura descritos pela *framework* de Zachman.

Aspectos da Arquitectura (Colunas da Framework de Zachman)	**Descrição**
Estrutura (O Quê)	Foco nas entidades/objectos/elementos de software com significado para a arquitectura e relação entre eles no contexto da organização.
Actividades (Como)	Foco no que a organização faz para suportar os processos.
Localizações (Onde)	Foca na distribuição geográfica das actividades da organização.
Pessoas (Quem)	Foca nas pessoas envolvidas no negócio da organização.
Tempo (Quando)	Foco no impacto que o planeamento e eventos no tempo têm na organização.
Motivação (Porquê)	Foco na tradução dos objectivos de negócio, estratégia e constrangimentos nas implementações específicas.

Tabela 60 – Aspectos da Arquitectura, segundo a *Framework* de Zahman

Cada célula na *framework* de Zachman resulta na representação de um aspecto da arquitectura para um interlocutor em particular. Numa mesma coluna, os modelos vão evoluindo de forma a reflectir as vistas dos interlocutores para cada linhas. Os modelos na mesma linha devem ser consistentes entre eles.

A *framework* de Zachman não define um processo para a definição da arquitectura, define as várias perspectivas que a arquitectura deve incorporar. A framework de *Zachman* é um modelo (*template)* ou um guia que deve ser seguido pelo processo de desenvolvimento de software implementado por cada organização.

Apêndice I.6 Adequação das Abordagens ao contexto do problema

Nesta secção é apresentada uma análise comparativa das várias abordagens, nomeadamente face à cobertura dos requisitos identificados no contexto do problema (I.3). São igualmente apresentados os resultados da análise à cobertura das vistas de cada abordagem face às necessidades descritas no contexto do problema, para uma arquitectura de software de uma LPS.

D1 - A representação da arquitectura da LPS deverá satisfazer as necessidades dos vários interlocutores no processo de desenvolvimento dos elementos de software da LPS, nomeadamente a representação numa perspectiva a) funcional, b) dos módulos de código, c) dos componentes de execução e d) dos elementos de configuração que correspondem à LPS ou produto em particular.		
Abordagem	**Aspectos que satisfazem requisitos**	**Aspectos não contemplados**
"4+1 vistas Kruchten" / RUP	A abordagem apresenta várias vistas que documentam a arquitectura com vista a vários intervenientes, funcionais, técnicos, gestores e equipa de testes.	Não representa a perspectiva de uma LPS, e instanciação dos produtos a partir da arquitectura de referência.
Siemens	Inclui fluxos de informação entre as vistas, suportando a correspondência dos elementos das vistas (estruturas) pelo processo de desenvolvimento.	Foco no desenho da arquitectura, segundo o ponto de vista do arquitecto e não tanto na documentação para a comunicação da arquitectura a outros intervenientes. Não representa a perspectiva de uma LPS, e instanciação dos produtos a partir da arquitectura de referência.
ISO RM-ODP	Inclusão da Vista Organizacional que descreve o sistema de informações em termos de o que o mesmo deve fazer e especificações que guiam e justificam o projecto. Foco na reutilização dos elementos de software da arquitectura.	Não representa a perspectiva de uma LPS, e instanciação dos produtos a partir da arquitectura de referência. O modelo RM-ODP não identifica os interlocutores alvos de cada uma das vistas que define. Este modelo foi desenhado de forma a satisfazer as necessidades da equipa de desenvolvimento.

Abordagem	Aspectos que satisfazem requisitos	Aspectos não contemplados
Paul Clements et al.	A abordagem apresenta várias vistas que documentam a arquitectura com vista a vários intervenientes, funcionais, técnicos, gestores e equipa de testes. Permite criar pacotes de documentação / vistas independentes para cada interveniente.	Não representa a perspectiva de uma LPS, e instanciação dos produtos a partir da arquitectura de referência.
Framework Zachman	Inclui uma vista para descrição dos aspectos organizacionais e das estruturas de informação. Enquadramento de um conjunto vasto de aspectos relevantes para a construção de uma arquitectura. Reconhecida popularidade na comunidade de arquitectos de sistemas de informação. Permite a conjunção desta abordagem com qualquer metodologia de desenvolvimento de software.	Não representa a perspectiva de uma LPS, e instanciação dos produtos a partir da arquitectura de referência. Poderá implicar um elevado esforço dos intervenientes na concepção dos artefactos necessários a cada uma das trinta células do modelo. Dificuldade em abstrair os artefactos relevantes para cada situação. O custo da criação e manutenção da documentação pode ser superior ao benefício que daí advém.

D2 - O elemento principal de identificação na arquitectura da LPS é a funcionalidade. A arquitectura da LPS deverá conter uma vista de requisitos da LPS, contendo todos os requisitos que esta satisfaz. Deverá ser possível identificar em qualquer elemento da arquitectura, seja um módulo de código ou componente de execução, a(s) funcionalidade(s) que este implementa.

Abordagem	Aspectos que satisfazem requisitos	Aspectos não contemplados
"4+1 vistas Kruchten" / RUP	Importância da vista Casos de Utilização, na identificação das funcionalidades e requisitos numa fase inicial e mais tarde na percepção das restantes vistas e validação do sistema. Inclui fluxos de informação entre as vistas, suportando a correspondência dos elementos das vistas pelo processo de desenvolvimento, baseados nos casos de utilização.	
Siemens		Não inclui uma vista de requisitos com a correspondência das funcionalidades ou requisitos com os elementos de software.
ISO RM-ODP	A vista Organizacional do RM-ODP utiliza casos de utilização, e categorias para representar e classificar as funcionalidades do sistema.	Não é explícito a correspondência entre os elementos das vistas.

Abordagem	Aspectos que satisfazem requisitos	Aspectos não contemplados
Paul Clements et al.	Utilização de diagramas de contexto para representar a relação entre módulos e descrição do âmbito funcional implementado. Poderão ser usados diagramas de casos de utilização UML para o efeito. O diagrama de contexto descreve o âmbito do módulo em desenvolvimento e envolvimento de entidades externas (caso existam). O diagrama de contexto "*alto-nível*" poderá ser utilizado para descrever as funcionalidades do sistema como um todo.	Para além dos diagramas de contexto que possuem uma utilização abrangente (variando o contexto de utilização dependendo da vista em causa) e são apresentados como anexos às restantes vistas, não é dado ênfase à representação funcional / vista requisitos da arquitectura de software.
Framework Zachman	Inclui uma vista para descrição dos aspectos organizacionais e das estruturas de informação.	Não inclui explicitamente as técnicas ou ferramentas para concretizar os vários artefactos. Complexidade do modelo de controlo e correspondência necessário para garantir a interligação dos artefactos nas linhas e colunas da *framework* de Zachman.

D3 - Contemplar de forma explícita a representação dos pontos de variabilidade e variantes na LPS. Identificar igualmente e de forma explícita os módulos de software que foram desenvolvidos por uma entidade externa à organização.

Abordagem	Aspectos que satisfazem requisitos	Aspectos não contemplados
"4+1 vistas Kruchten" / RUP		Não inclui a representação de pontos de variabilidade e variantes na arquitectura de software
Siemens		
ISO RM-ODP		
Framework Zachman		
Paul Clements et al.	Contempla a representação de variabilidade e dinamismo nas vistas e estilos da arquitectura de software. identificação da origem de um módulo de software pode ser representado através da inclusão de um atributo nos estilos a utilizar. Os diagramas de contexto anexos às vistas podem identificar os variantes base, opcionais e externos numa LPS.	

D4 - No proposta de modelo a apresentar deverá ser garantida a correspondência entre os artefactos que constituem a arquitectura da LPS, desde a definição dos requisitos aos elementos de software e binários para instalação de um produto.		
Abordagem	**Aspectos que satisfazem requisitos**	**Aspectos não contemplados**
"4+1 vistas Kruchten" / RUP	A abordagem apresenta um conjunto de regras e heurísticas para a ligação entre elementos de vistas diferentes. A implementação RUP da IBM o processo UCM (*Unified Change Managent*)[46] suportado por um conjunto de ferramentas que permite a gestão integradas de alterações e configurações nos artefactos da arquitectura de software.	Não apresenta a distinção clara entre os artefactos que fazem parte da LPS e os artefactos instanciados para os produtos em particular.
Siemens	Inclui fluxos de informação entre as vistas (estruturas), suportando a correspondência dos elementos da vista conceptual com os elementos das restantes vistas da arquitectura de software.	
ISO RM-ODP		O RM-ODP não define quais as ferramentas ou técnicas que devem ser utilizadas para a representação de cada vista. Não apresenta a distinção clara entre os artefactos que fazem parte da LPS e os artefactos instanciados para os produtos em particular.
Paul Clements et al.	Diagramas de contexto anexos às vistas podem representar a relação entre elementos.	Não apresenta a distinção clara entre os artefactos que fazem parte da LPS e os artefactos instanciados para os produtos em particular.
***Framework* Zachman**		A *framework* é adequada para a compreensão do domínio do problema e fazer o planeamento da arquitectura. A implementação da arquitectura e garantia de interligação dos artefactos nas linhas e colunas da *framework* de Zachman é abordada superficialmente.

Tabela 61 – Grau de satisfação dos requisitos das abordagens para a documentação de arquitecturas de software

Apêndice II. PERFIS UML PARA A REPRESENTAÇÃO DE VARIABILIDADES NUMA LPS

Apêndice II.1 Ziadi et al.

Em Ziadi et al [24], é apresentada uma extensão aos diagramas de classes e diagramas de sequência UML 2.0, através da inclusão de estereótipos e constrangimentos:

- Diagrama de Classes: adicionado o estereótipo «*optional*» para representar as funcionalidade do software que são opcionais em termos de instalação nos produtos alvo; «*variation*» para representar os pontos de variabilidade e «*variant*» para os variantes respectivos. A Figura 45, ilustra um diagrama de classes para a SPL camera.

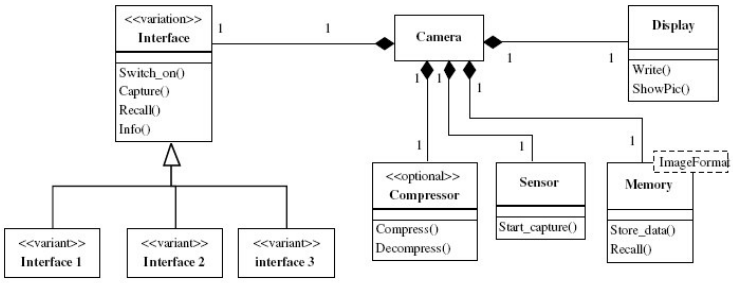


Figura 45 – Diagrama de Classes da SPL Camera (retirado de [24]).

- Diagrama de Sequência: adicionados os estereótipos para definir características opcionais: «*optionallifeline*» e «*optionalintereaction*», «virtual»: interacção é virtual, «*variation*» e «*variant*»: a interacção é um ponto de variabilidade e variante respectivamente. No estereótipo «*variant*» é igualmente incluído o *tagged value*: {*variation=Variation*}: identifica o ponto de variabilidade associado a cada variante.

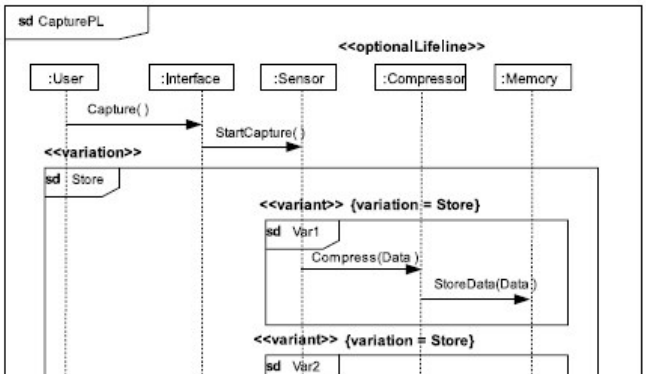


Figura 46 – Diagrama de Sequencia Captura (retirado de [24]).

Apêndice II.2 Matthias Clauß

Clauß [28] introduz uma extensão UML 1.4 para suportar os diagramas de *features* e adiciona elementos que descrevem a variabilidade nos diagramas UML standard. A abordagem apresentada estende o modelo *Feature-Oriented Reuse Method* (FORM) [34] extensão do método *Feature-Oriented Domain Analysis* (FODA) [35], na medida em que além dos três tipos de *features*: "*optional*", "*alternative*" e "*mandatory*" é adicionado o tipo "*external*", que representa as características ou requisitos externos ao sistema. A modelação dos diagramas de *features* em UML é um dos contributos mais importantes deste autor.

- As relações hierárquicas entre *features* são representadas pela relação de composição UML. A exclusividade é definida pelas anotações à relação: {xor}.
- Inclusão de atributos especiais (*tagged values*) para descrever as razões pela selecção de um variante ou tempo de ligação do ponto de variabilidade {desenvolvimento, instalação, execução}.
- Os pontos de variabilidade são classificados através da utilização do estereótipo «*variantionPoint*» e um conjunto de atributos que descrevem o ponto de variabilidade.
- Os estereótipos «*context*», «*optional*», «*mandatory*» e «*external*» são utilizados para identificar as *features* de contexto (topo hierarquia), opcionais, obrigatórias e externas.
- Utilização dos constrangimentos '*mutex*' e '*requires*' para adicionar alguma semântica e descrever as relações de dependência entre *features*.

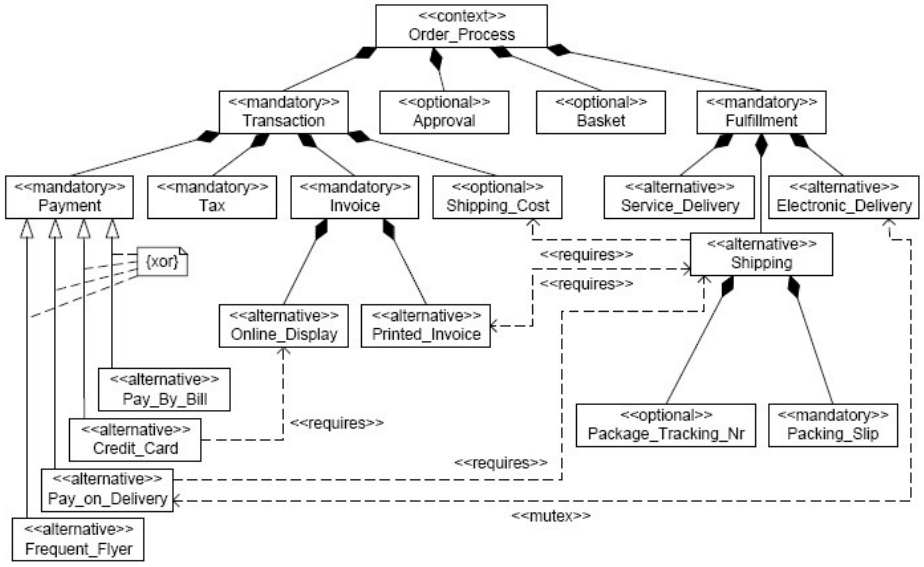


Figura 47 – Exemplo de um modelo de funcionalidades, usando extensões ao UML 1.4. (retirado de [28])

Apêndice II.3 Dobrica e Niemelä

Dobrica e Niemelä [25], no contexto da análise e desenho de arquitecturas em LPS, adoptaram um perfil UML para desenhar arquitecturas de software baseadas no método *Quality-driven Architecture Design and quality Analysis* (QADA) [62], específico para a modelação de arquitecturas de serviços *middleware*.

- Estereótipos UML para a vista estrutural QADA, com objectivo de representar as variabilidades a um nível estrutural: «*mandatoryLeaf*» ou «*mandatorySubsystem*», «*alternativeLeaf*» ou «*alternativeSubsystem*», «*optionalAlternativeLeaf*» ou «*optionalAlternativeSubsystem*», «*optionalLeaf*» ou «*optionalSubsystem*».
- São usados os seguintes estereótipos UML para as relações de variabilidade entre elementos: a) associação entre sistemas obrigatórios: «*control*», «*data*», «*uses*», b) associação entre dois subsistemas, sendo que pelo menos um deles é opcional: «*control (opt)*», «*data (opt)*», «*uses (opt)*», c) associação entre dois subsistemas, sendo que pelo menos um deles é opcional e alternativo: «*control (optAlt)*», «*data (optAlt)*», «*uses (optAlt)*».
- Para a vista estrutural concreta são utilizados os seguintes estereótipos, que estendem os componentes UML, subdivididos em componentes abstractos: «*topCapsule*», composição/decomposição: «*subsystemCapsule*», «*component1..NCapsule*» e concretos: «*concreteComponent*». Cada componente concreto possui uma letra de A a Z que identifica o produto ao qual ele pertence.

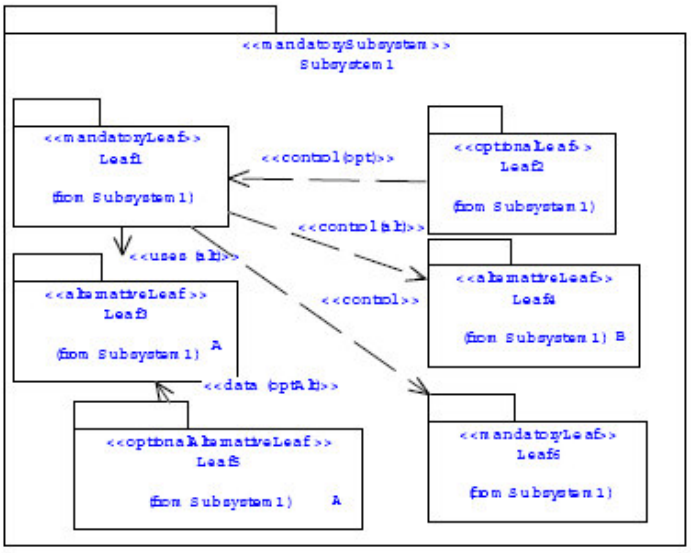


Figura 48 – Pontos de Variabilidade estrutural incluídos na vista conceptual (retirado de [25])

Apêndice II.4 Gomaa et al.

O método apresentado por Gomaa et al. [63], permite a modelação explícita dos aspectos comuns e variabilidade pelos membros de uma LPS ou em combinações LPS. Várias vistas do método são estendidas (em particular a vista de casos de utilização e a vista estática) para comportarem LPS.

- São introduzidos estereótipos na vista casos de utilização.
- Incluído um modelo UML de características (*feature model*) que é utilizado para representar os requisitos comuns e variáveis numa LPS. Utilizado um modelo UML de pacotes para agrupar os casos de utilização que pertencem à mesma *feature*.
- Os diagramas de classes são utilizados para representar o modelo de domínio da LPS e incluem os seguintes estereótipos para representar as variabilidades: «*kernel*», «*optional*», «*variant*» e «*alternative*». Hierarquias de agregação e generalização/especialização são utilizadas para representar a vista estática.
- Gomaa, em [53], apresenta igualmente um conjunto de extensões UML para representar as variabilidades ao nível das LPS, nomeadamente os estereótipos «*common feature*», «*optional feature*», «*alternative feature*»,«*default feature*» e «*parameterized feature*».

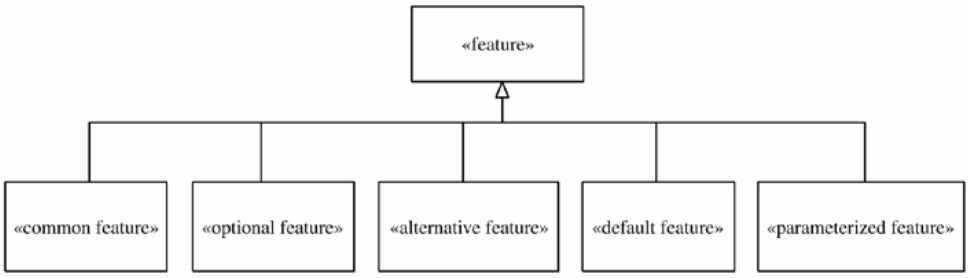


Figura 49 – Classificação de funcionalidades numa LPS utilizando estereótipos UML, segundo Gomaa (retirado de [53]).

A especificação e representação do comportamento de cada funcionalidade (*features*) em casos de utilização UML, e relação entre os pontos de variabilidade das funcionalidades e a sua representação através das relações de extensão e inclusão dos casos de utilização é um dos aspectos inovadores da abordagem proposta por Gomaa em [53].

Apêndice III. ABORDAGENS PARA A IMPLEMENTAÇÃO E GESTÃO DE UMA LPS

Apêndice III.1 SEI *Framework* para Implementação LPS

O *Software Engineering Institute* (SEI) desenvolveu um programa específico para o tema das LPS: "*Product Line Systems Program*", segundo o qual surgiram um conjunto de práticas e ideias pioneiras para a implementação de uma abordagem de LPS no desenvolvimento de software [41]. Os objectivos da *framework* prendem-se com a definição de um conjunto de práticas (ao nível da engenharia, gestão organizacional e tecnológica), que permitam as organizações diminuir o risco na adopção das LPS.

Segundo a *framework* do SEI, a implementação de uma linha de produtos de software, na sua essência, pode ser dividida em três actividades essenciais, as quais são por sua vez materializadas através de um conjunto de práticas:

- Desenvolvimento dos activos base
- Desenvolvimento dos produtos
- Gestão (técnica e organizacional)

Figura 50 – Actividades Essenciais na Implementação e Gestão de uma LPS, adaptado de [30]

Para a implementação da *framework* foram definidas várias áreas de prática, as quais cada organização deve seguir para implementar a LPS. As áreas de prática são actividades específicas para a execução de tarefas concretas, e.g. "desenvolver elemento base".

As áreas de prática do modelo foram agrupadas nas seguintes categorias:

- Engenharia de Software. Aplicar a tecnologia para criar e evoluir os elementos base e produtos;
- Gestão Técnica. Técnicas de gestão, a aplicar na criação e evolução dos elementos base e produtos;
- Gestão Organizacional. Práticas para a orquestração das actividades de construção da linha de produtos de software;

A Tabela 62 apresenta as várias áreas de prática para cada categoria. Informação adicional pode ser obtida em [30].

Engenharia de Software	**Gestão Técnica**	**Gestão Organizacional**
Definição Arquitectura	Gestão de Configurações	Criação do Business Case
Avaliação Arquitectura	Colecção de Dados, Métricas e Tracking	Gestão da Interface com o Cliente
Desenvolvimento dos Elementos de software	Definição processo	Estratégia de Aquisição e Desenvolvimento
Utilização de Pacotes SW	Definição âmbito	Gestão de Fundos
Engenharia de Requisitos	Planeamento técnico	Arranque e Institucionalização
Integração do sistema	Gestão de risco técnico	Análise de Mercado
Testes	Suporte utilizando ferramentas	Operações
Percepção dos Domínios Relevantes		Planeamento Organizacional
		Gestão de Risco Organizacional
		Estruturação da Organização
		Evolução da Tecnologia
		Formação

Tabela 62 – Áreas de Prática da *Framework* Implementação de LPS do SEI

Apêndice III.2 Family-Oriented Abstraction, Specification, and Translation (FAST)

O FAST é um processo de definição de LPS e de desenvolvimento de uma linguagem específica do domínio (*Domain Specific Language - DSL*) de suporte à geração de produtos da LPS [23]. Os principais objectivos são:

- Prever as semelhanças e variabilidades entre os elementos de software base da LPS;
- Identificar abstracções que possam ser comuns na LPS;
- Definir um processo de produção dos produtos da LPS;
- Desenhar uma linguagem para especificar os produtos da LPS;
- Geração rápida do software a partir das especificações.

A abordagem FAST prevê um maior investimento do esforço no processo de engenharia do domínio nomeadamente na construção de um ambiente comum de engenharia aplicacional, permitindo obter uma maior rapidez na geração dos produtos. Os principais componentes da abordagem FAST são:

- Engenharia aplicacional. Existência de um processo iterativo para a construção de produtos de software, com base na definição de requisitos e na existência de uma linguagem de especificação;
- Ambiente de engenharia aplicacional. Conjunto de ferramentas que suporta o processo de engenharia aplicacional;
- Engenharia de domínio. Processo iterativo para o desenho e desenvolvimento tanto da LPS como do processo de engenharia aplicacional para a LPS;
- Modelo de Domínio. Especificação para um ambiente de engenharia aplicacional.

Processo FAST:

1 Definir a Família.
   - Definir assumpções, termos e abstracções que são comuns à LPS;
   - Identificar variações que caracterizem os membros da família.

2. Desenhar o ambiente de engenharia aplicacional
   - Desenhar a linguagem de modelação aplicacional (AML);
   - Implementar a AML.

3. Gerar os Produtos da LPS, com base no ambiente de engenharia aplicacional.

Apêndice III.3 Product Line Software Engineering (PuLSE)

O objectivo principal do *Product Line Software Engineering* (PuLSE) [20] é permitir a concepção e desenvolvimento de LPS numa vasta variedade de contextos empresariais. Os métodos de engenharia baseados no domínio cobrem muitos dos aspectos do PuLSE, no entanto sem a facilidade de adaptação e instalação dos produtos específicos.

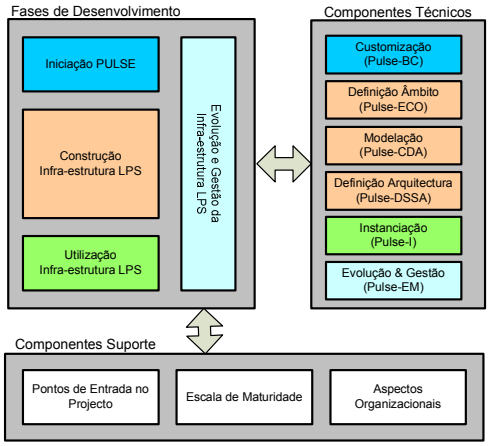


Figura 51 – Fases de desenvolvimento e componentes técnicos e de suporte do PuLSE. Adaptado de [20].

O PuLSE é composto por três elementos principais: a) Fases de desenvolvimento, b) Componentes técnicos e c) Componentes de suporte

A Figura 51, ilustra a decomposição dos componentes e fases do PuLSE. As cores pretendem ilustrar a relação entre cada uma das fases e os componentes técnicos usados para a sua concretização.

Fases de desenvolvimento

No PuLSE, as fases de desenvolvimento são as actividades para a concepção e utilização da linha de produtos de software são:

- Iniciação: Obter o diagnóstico da organização e adaptar o PULSE à sua realidade
- Construção da Infra-estrutura: Definição do âmbito, modelar e definir a arquitectura da infra-estrutura da LPS.
- Utilização da Infra-estrutura: Utilização da infra-estrutura para a criação de produtos da LPS
- Evolução e Gestão: Evolução e gestão da infra-estrutura da LPS ao longo do tempo.

Componentes técnicos

Os componentes técnicos são as disciplinas ou áreas de conhecimento que suportam as actividades de cada uma fase de desenvolvimento. Os componentes técnicos são:

- Customização: Suportar a fase de iniciação
- Definição do âmbito: Suporta a definição do âmbito da infra-estrutura focada na definição dos produtos
- Modelação: Suporta a modelação das características dos produtos no âmbito da LPS
- Definição Arquitectura: Suporta a definição da arquitectura de referência e rastreio do modelo. Um dos métodos mais utilizados para a construção de arquitecturas de referência é o PuLSE-DSSA, um método iterativo, baseado em cenários. A base deste método é o desenvolvimento da arquitectura de referência de forma incremental, aplicando os cenários por ordem decrescente de importância para a arquitectura. Outro aspecto relevante é a contínua avaliação durante a criação da arquitectura.
- Instanciação: suporta a fase de utilização da LPS.
- Evolução e Gestão: integração de novos elementos de software na LPS e aspectos de gestão de configurações nos produtos ao longo do tempo.

Componentes de suporte

Os componentes de suporte são guias e informação que permite uma melhor adaptação, evolução e implementação da LPS:

- Pontos de entrada no projecto: identifica os aspectos específicos de reutilização e integração de elementos de software no projecto.
- Escala de maturidade: apresenta o caminho de evolução e maturação da LPS, utilizando o PULSE.

- Aspectos de organizacionais: dispõe informação para assegurar a correcta estrutura orgânica para desenvolver e manter a LPS.

Esta abordagem para a implementação de uma LPS é genérica, não focando em particular o desenvolvimento baseado em objectos.

Apêndice III.4 Abordagem KobrA

O método KobrA [33], desenvolvido no *Fraunhofer Institute for Experimental Software Engineering* (IESE), apresenta-se como uma adaptação do método PuLSE, onde foi dado ênfase à definição de um processo de desenvolvimento capaz de ser utilizado em organizações cujo grau de maturidade dos seus processos seja reduzido.

A abordagem KobrA é uma especialização do PuLSE para o desenvolvimento baseado em componentes, com base numa abordagem OO, sendo a linguagem de modelação o UML (ver Figura 52). Este método tem como principais desvantagens a sua complexidade em termos de implementação (no contexto de uma PME) e o facto de não serem conhecidas referências da sua implementação com sucesso na indústria.

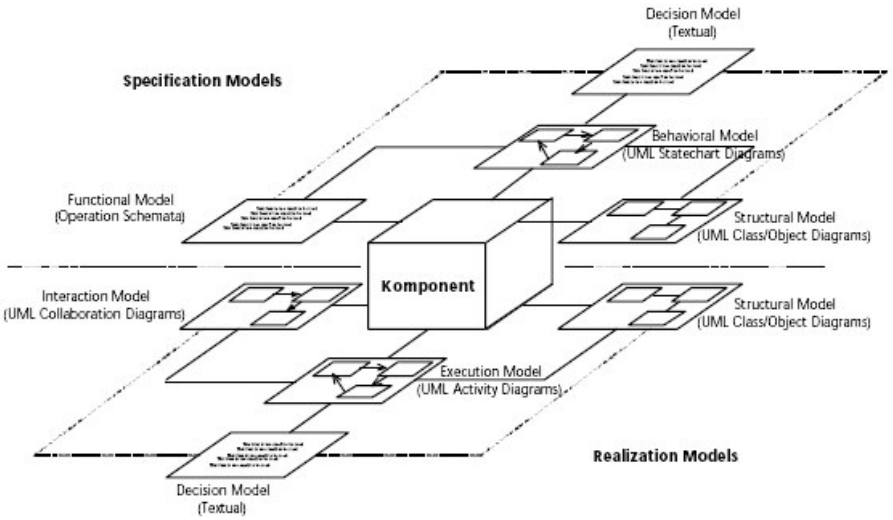


Figura 52 – Modelos de Especificação e Realização de Komponents (retirado de [33])

Apêndice III.5 Feature-Oriented Domain Analysis (FODA)

O método *Feature-Oriented Domain Analysis* (FODA) [35] introduz a definição do modelo de domínio, através de uma estrutura hierárquica de funcionalidades (*features*) obtida durante a engenharia e análise do domínio.

A engenharia de domínio é um processo que visa o desenho não só de um sistema como de uma família de sistemas de software. A análise de domínio é o primeiro passo da engenharia da engenharia de domínio com vista à criação de um modelo

descrevendo o vocabulário, as características comuns e variáveis bem como as dependências existentes entre elas.

O método FODA objectiva a identificação das funcionalidades (i.e. aspectos, qualidades ou características do sistema) durante a análise de domínio. São utilizados modelo hierárquicos em árvore de funcionalidades para visualizar as variabilidades e dependências de funcionalidades. O método FODA compreende quatro elementos: a) diagramas de funcionalidades, b) definições de funcionalidades, c) regras de composição para as funcionalidades e d) racional para as funcionalidades.

Um diagrama de funcionalidades, ver Figura 53, descreve a decomposição hierárquica de funcionalidades em sub-funcionalidades. Cada sub-funcionalidade pode ser descrita como sendo "*opcional*", "*obrigatória*" ou "*alternativa*".

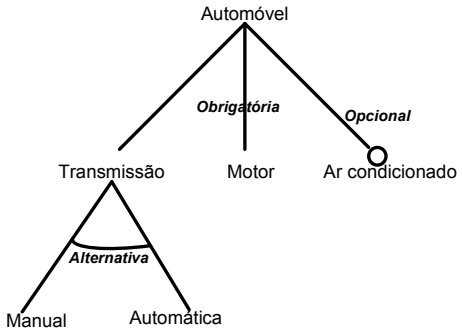


Figura 53 – Exemplo de um diagrama de funcionalidades FODA, para descrever um automóvel

Segundo o método FODA, uma funcionalidade pode ser classificada quanto ao seu propósito como: a) de contexto, i.e. descrevem o contexto da aplicação, padrões de utilização ou requisitos de desempenho; b) de representação, i.e. representam as entradas e saídas de dados da aplicação; c) de operação, i.e. descrevem as funções da aplicação. Um outro contributo do FODA é a classificação das funcionalidades segundo o tempo de ligação (*binding*) da mesma no sistema: a) tempo de compilação, b) tempo de activação, i.e. são activadas no início da execução da aplicação e permanecem estáveis durante o tempo de execução e c) tempo de execução, activadas dinamicamente. No FODA a composição de funcionalidades pode ser realizada de duas formas: a) regra requer (*requires*) e b) regra "*mutuamente-exclusiva-com*" (implementa a condição XOR).

Apêndice III.6 Feature-Oriented Reuse Method (FORM)

O método *Feature-Oriented Reuse Method* (FORM) é uma concretização do modelo FODA. O FORM apresenta um conjunto de procedimentos para o desenho e implementação do modelo de *features* na arquitectura de referência e elementos de software reutilizáveis da LPS. A filosofia do FORM é a de que "as funcionalidades

caracterizam cada produto variante do domínio e o código que o implementa pode ser empacotado, gerido e utilizado como módulos de software" [34].

O primeiro passo do método FORM é a definição do modelo de funcionalidades, que resulta da análise dos aspectos semelhantes entre aplicações de um domínio em particular nomeadamente em termos de funções, ambientes operacionais, tecnologias utilizadas e técnicas de implementação. Este modelo é usado para a definição da arquitectura de referência e elementos de software reutilizáveis que serão instanciados durante o desenvolvimento aplicacional.

As arquitecturas são definidas segundo três vistas principais: subsistema, processo e módulo. A arquitectura do subsistema materializa um conjunto de funcionalidades que podem ser executadas em diferentes computadores de forma distribuída. Cada subsistema é decomposto em processos segundo as funcionalidades do ambiente operacional. Os módulos são definidos com base nas funcionalidades de tecnologia e técnicas de implementação.

O método FORM apresenta na sua abordagem uma correspondência entre as funcionalidades e os elementos de software da arquitectura se software, através da representação de cada funcionalidade ou respectiva especialização, como um elemento de software aplicacional distinto. Nesta perspectiva, a instanciação de um produto far-se-ia pela instanciação dos elementos de software que correspondem às funcionalidades que caracterizam esse produto.

A abordagem FORM apresenta dois processos de engenharia para as LPS: engenharia de domínio e engenharia aplicacional. O FORM utiliza no processo de engenharia alguns princípios importantes na concepção de LPS: separação de conceitos e encapsulamento de informação, configuração dos artefactos da arquitectura de software consoante as diferentes funcionalidades, separação entre componentes e conectores.

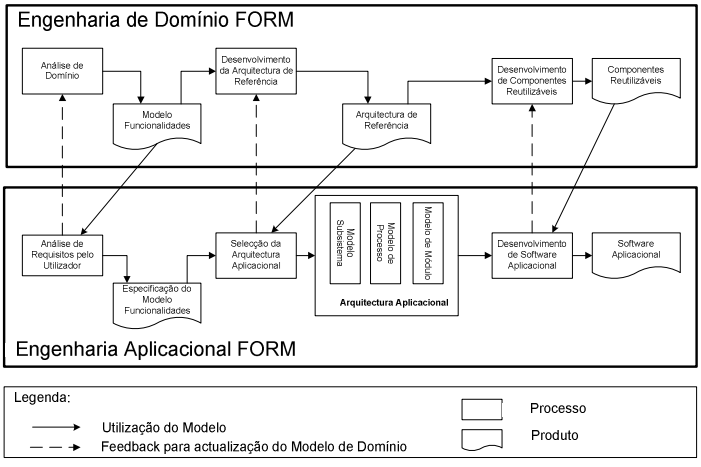


Figura 54 – Processo de engenharia FORM. Adaptado de [34]

UNIÃO EUROPEIA
FUNDO SOCIAL EUROPEU

PROGRAMA OPERACIONAL
PLURIFUNDOS DA REGIÃO
AUTÓNOMA DA MADEIRA