Universidade da Madeira

Documentação da Arquitectura de um Sistema de Software

Ana Cristina Teixeira da Silva

Março 2008

Universidade da Madeira

Documentação da Arquitectura de um Sistema de Software

Ana Cristina Teixeira da Silva

Relatório Final

Mestrado em Engenharia Informática

Orientador na Universidade da Madeira: Eng. Pedro Campos
Orientador na Informar: Eng. Vitorino Gouveia

Março 2008

Resumo

Este documento é o relatório final de um projecto de Mestrado, da Universidade da Madeira em colaboração com a empresa Informar. O objectivo deste projecto consistiu na documentação da arquitectura de software do sistema ARQUO. O pacote de documentação, resultado final deste projecto, consiste na documentação das várias vistas aplicadas ao sistema e á documentação que se aplica a mais do que uma dessas vistas. A escolha do "*template*" a usar para documentar o sistema, a escolha das vistas, o estudo sobre a forma como modelar a informação de cada vista e a forma de apresentação do pacote de documentação foram os passos mais importantes do projecto.

Palavras Chave

ARQUO
Arquitectura
Pacote de Documentação
Vistas
Estilos
UML

Agradecimentos

Ao Engenheiro Vitorino Gouveia pela orientação e apoio.

Ao Engenheiro Pedro Campos pela orientação.

Aos Engenheiros António Sargento e Sérgio Soares da TIE, pela colaboração e disponibilidade.

Aos Engenheiros Filipe Velosa e Marco Erra e a todas as pessoas da Informar.

Aos Engenheiros Miguel Gouveia, Lúcio Quintal, Davide Sousa, Manuel Coelho, Rui Henriques, Pedro Valente e Filipe Pereira pela preciosa colaboração no projecto através da resposta ao inquérito.

Índice

Indice

Índice de Figuras

Indice

Índice de Tabelas

1. Introdução

À medida que os sistemas de software aumentam a sua complexidade e dimensão torna-se cada vez mais importante compreendê-los a níveis de abstracção mais elevados, seja durante o projecto de engenharia, na fase de análise ou na reengenharia.
A arquitectura de um sistema de software descreve a sua estrutura global em termos dos seus componentes, das propriedades externas desses componentes e das suas inter-relações.

A arquitectura de um sistema de software é um factor crucial na determinação do sucesso ou insucesso do mesmo. Sem uma arquitectura adequada, que descreva as funções requeridas bem como os atributos de qualidade, o projecto irá falhar. No entanto, mesmo a melhor arquitectura não terá qualquer valor se não for eficientemente comunicada ás pessoas que necessitam dela: a equipa de desenho, a equipa de implementação, a equipa de testes, os analistas, de entre um conjunto de muitos outros; daí que a sua documentação seja tão importante para a compreensão do sistema. Tendo em conta que, essas pessoas ou grupos de pessoas – *stakeholders* – que têm necessidades de informação sobre o sistema, desempenham diferentes papéis, e que cada *stakeholder* vê o sistema através de diferentes perspectivas é necessário que a documentação da arquitectura do sistema satisfaça as necessidades de informação para cada *stakeholder* em particular. Assim, a documentação da arquitectura deve ser suficientemente abstracta de forma a fornecer uma rápida compreensão do sistema a novos stakeholders; deve ser suficientemente detalhada de forma a ser utilizada como *blueprint* durante a construção, e ao mesmo tempo, deve conter informação suficiente para servir como base á análise do sistema.

O uso de um formato predefinido para a documentação de uma arquitectura, ajuda a evitar que a documentação seja incompleta, inadequada, vaga ou demasiado detalhada. A documentação e o planeamento da mesma deve estar a um nível proporcional á sua importância e aos fins a que se destina. Os arquitectos de software e as pessoas a quem a arquitectura tem de ser comunicada vão beneficiar, pelo facto da informação estar disponível e acessível de forma rápida e precisa. Assim, os custos, em termos de tempo e recursos, dispendidos durante a tarefa de documentação de uma arquitectura serão colmatados.
Existem actualmente várias abordagens sobre a forma de documentar uma arquitectura. A escolha de uma abordagem em detrimento de outra vai depender do tipo de sistema de software e dos fins a que se destina essa documentação.

No caso particular deste projecto, o sistema a documentar já está em uso no mercado, e a documentação do mesmo tem como finalidade ajudar na sua manutenção e reengenharia. Documentar a arquitectura de um sistema deste tipo, tem dificuldades acrescidas que podem ser por exemplo, a ausência de documentação, ou no caso de existir que esta não esteja actualizada ou correctamente documentada; alguns dos stakeholders envolvidos no desenvolvimento do sistema podem já não estar ligados ao sistema; ou a ausência de especificações técnicas. A solução para minimizar estas dificuldades será um estudo aprofundado do sistema em questão, conciliado com a comunicação com os stakeholders que usam o sistema actualmente com o objectivo de produzir documentação adequada á manutenção e reengenharia do sistema.

1.1 Sistema ARQUO

O ARQUO é um sistema de arquivo óptico digital. Desenvolvido pela empresa Tecnologias de Integração Empresarial e adoptado por várias empresas portuguesas e não só. Essas empresas são na sua maioria Instituições Bancárias e Companhias de Seguros, sendo assim os critérios de segurança, desempenho e armazenamento assumem um papel muito importante no desenvolvimento do sistema. O ARQUO é um sistema que está em constante expansão, seja para responder ás necessidades dos clientes actuais, ou para adaptação a clientes futuros. Assim, a existência de uma documentação como suporte a essa expansão é essencial.

1.2 1.2 Organização deste documento

Este documento está organizado em cinco capítulos.
O primeiro capítulo, onde se insere esta secção, consiste na introdução ao projecto e uma pequena apresentação sobre o sistema usado, no caso prático da documentação de uma arquitectura.
O segundo capítulo expõe os objectivos deste projecto e mostra a sequência de passos que foram executados para atingir esses mesmos objectivos.
O terceiro capítulo, apresenta os estilos que foram usados para documentar a arquitectura do sistema ARQUO, tendo em conta a escolha das vistas que se encontra no Anexo D. Neste capítulo é também especificado cada um desses estilos, no que diz respeito á definição dos seus elementos e relações, a sua relação com os outros estilos e a notação usada para representar os elementos e as relações em cada um desses estilos.
O capítulo quatro apresenta alguns excertos, daquele que foi o resultado final da aplicação dos conceitos e metodologias definidos no capítulo três e anexos A e D, na documentação da arquitectura de software do sistema ARQUO.
O capítulo cinco apresenta um estudo realizado com base num inquérito feito a várias organizações regionais e nacionais, sobre a forma como vêem a documentação do software.

2. Apresentação do problema

O objectivo do presente projecto é a definição de uma metodologia para a documentação da arquitectura de sistemas de software, aplicado ao caso prático do sistema ARQUO. O resultado dessa documentação será então um pacote de documentação completo, fiável, de fácil leitura e que corresponda ás necessidades dos *stakeholders* envolvidos no sistema.

2.1 Trabalho Realizado

O cumprimento desse objectivo passou por várias fazes de execução. Inicialmente foi feito o estudo de uma solução para a documentação da arquitectura. Esse estudo foi feito sobre o livro "Documenting Software Architectures" [1]. O resultado desse estudo apresentado no Anexo A, foi a definição da estrutura do pacote de documentação. Assim sendo, a arquitectura do sistema em estudo seria representada através de vistas arquitecturais, que seriam documentadas individualmente seguindo cada uma um *Template* definido. De forma a que a documentação das vistas fosse de fácil compreensão e leitura seria anexado ao pacote de documentação um conjunto de informações, que consistiam na descrição do formato de documentação das vistas, na apresentação do sistema a documentar e na descrição da organização do pacote de documentação como um todo. Adoptava assim o principio fundamental da aproximação "para além das vistas" [1] – views and beyond, que diz o seguinte:

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

Definido o *Template* para o pacote de documentação, o passo seguinte consistiu na escolha das vistas a documentar. Foi feito um estudo sobre algumas abordagens existentes, [Anexo D] e daí resultou um conjunto de vistas que seriam usadas para documentar a arquitectura do sistema ARQUO.

Após a escolha das vistas e estilos a usar, o problema residia na notação a usar para representar a informação em cada uma das vistas. A solução definida para a notação e a descrição dos estilos a usar é apresentada no capítulo seguinte: Apresentação da Solução. Definidas as notações, o passo a seguir foi a análise do sistema ARQUO em geral e a identificação e análise dos vários módulos que constituem o sistema. Toda a notação (diagramas) foram desenvolvidos na ferramenta Enterprise Architect, devido ás suas características de flexibilidade no que diz respeito á modelagem em UML.

Um factor muito importante na documentação do sistema ARQUO prendeu-se com a forma como essa documentação seria apresentada. Foram estudadas algumas opções tendo em vista os dois principais requisitos: a navegabilidade (a documentação teria que ser fácil de percorrer, permitindo assim que um stakeholder encontre facilmente a informação que necessita) e a actualização da documentação (este ultimo factor é muito importante, tendo em conta que o ARQUO é um sistema em constante expansão). Foram então analisadas duas soluções: a documentação automática fornecida pelo Enterprise Architect, que correspondia plenamente ao requisito de actualização, no entanto falhava no que diz respeito á navegabilidade. Esta opção foi afastada devido a essa falha e por não ser possível adaptar a metodologia de documentação

definida no anexo A ao *template* que o Enterprise Architect disponibiliza. A segunda solução foi a solução adoptada; a utilização do Tikiwiki. O Tikiwiki tem a vantagem de permitir, que o pacote de documentação seja fácil de consultar e de navegar, através da utilização de hiper ligações (hipertexto); permite também controlo de acessos assegurando assim a confidencialidade e o acesso restrito á informação através da criação de grupos de utilizadores. No entanto, tem a desvantagem de não permitir uma fácil actualização no que diz respeito aos diagramas, para minimizar este problema a solução adoptada foi a criação de uma hierarquia de directórios, que contêm as imagens referentes aos diagramas que modelam a arquitectura, semelhante a hierarquia de directórios usada para organizar o código do sistema ARQUO. Possibilitando assim, que aquando da actualização de um modelo, seja fácil encontrar o ficheiro que o contém e actualiza-lo.

O último passo consistiu então na aplicação do resultado dos passos anteriores, ao caso prático da documentação do sistema ARQUO.

3 Apresentação da Metodologia

Este capítulo apresenta os estilos que foram usados para documentar a arquitectura do sistema ARQUO, tendo em conta a escolha das vistas que se encontra no Anexo D.
A secção 3.2 contém a especificação de cada um desses estilos, no que diz respeito á definição dos seus elementos e relações, a sua relação com os outros estilos e a notação usada para representar os elementos e as relações em cada um desses estilos. A notação utilizada em cada estilo segue a metodologia que é apresentada na secção 3.1 deste capítulo.

3.1 Estilos Utilizados para documentar a o sistema ARQUO

Os estilos surgem através da especialização dos elementos e relações das vistas arquitecturais. As vistas seleccionadas como sendo as mais apropriadas para documentar o sistema ARQUO (Anexo D) já possuem, cada uma, um conjunto de estilos predefinidos. De seguida, são apresentados os estilos pertencentes ás vistas Módulo, Componente&Conector, Afectação e Organizacional que foram utilizados neste projecto. São apresentados também dois novos estilos, o estilo Agregação e o estilo Casos de Utilização, criados no âmbito deste projecto, e a combinação entre dois estilos – o estilo Utilização e o estilo Agregação.

3.1.1 - Vista Módulo

São definidos quatro estilos para a vista Módulo, o estilo Decomposição, o estilo Generalização, o estilo Utilização e o estilo Camadas (definidos no Anexo D). Este ultimo, estilo camadas, não foi usado na documentação do sistema ARQUO, porque a definição de máquina virtual não se aplica a este sistema. Os restantes estilos foram usados para documentar o sistema.
O estilo Decomposição foi usado para mostrar a decomposição dos módulos do sistema ARQUO. Os níveis de decomposição servirão as diferentes necessidades dos stakeholders. Por exemplo o gestor de projecto terá interesse em consultar os níveis mais altos de decomposição, por exemplo para distribuição de trabalho. Ao passo que um stakeholder da equipa de desenvolvimento necessitará dos níveis de mais detalhe para desenvolver o seu trabalho.
O estilo Utilização foi usado para mostrar a dependência entre os vários módulos do sistema. Permitindo assim, por exemplo, que sempre que seja necessário fazer alterações a um módulo seja possível ter a informação de quais os módulos dos quais o módulo a alterar depende.
O estilo Generalização foi usado porque relaciona módulos mostrando como um é a generalização ou especialização do outro, este tem como base o conceito da programação orientada a objectos o qual foi usado no desenvolvimento do sistema ARQUO.
Além dos estilos predefinidos da vista Módulo foi necessário criar um outro estilo – o estilo Agregação. Este surgiu, da necessidade de representar informação sobre os módulos do sistema que nenhum outro módulo representava. Tal como o estilo Generalização, o estilo Agregação tem como base os conceitos da programação orientada a objectos usada no sistema ARQUO. Essa base, faz com que a definição de módulo neste estilo seja mais restrita do que na vista Módulo.

3.1.2 – Vista Componente & Conector

Na vista Componente e Conector são conhecidos muitos estilos. De entre eles os dois escolhidos para usar neste projecto. Para representar a interacção entre os componentes do sistema ARQUO, quando essa interacção é do tipo pedido-resposta, foi escolhido o estilo Cliente-Servidor. Para mostrar, como é que os componentes de execução (em especial os objectos do sistema) comunicam através da troca de mensagens foi escolhido o estilo Processos Comunicantes.

3.1.3 – Vista Afectação

A vista afectação define três estilos, Instalação, Implementação e Trabalho. Para a documentação do sistema ARQUO, foram usados os estilos Implementação e Instalação. O estilo Instalação, porque fornece uma visão de como é que os componentes do sistema estão afectados à plataforma de execução. Este estilo permite mostrar também quais os requisitos dos elementos de software para que seja mais fácil, afectá-los aos elementos de hardware que satisfaçam esses requisitos. O estilo Implementação foi usado para fornecer a informação de como é que os módulos da vista Módulo estão afectados ás directorias e ficheiros. Permitindo assim, que os stakeholders do sistema associem mais facilmente os ficheiros e directórios ás unidades de implementação, tratadas como módulos.

3.1.4 – Vista Organizacional

A vista Organizacional, uma das vistas propostas pela RM-ODP (sigla de Reference Model of Open Distributed Processing) da ISO, descreve o sistema de informações em termos do que o mesmo deve fazer e descreve também as necessidades e especificações administrativas e técnicas que guiam e justificam o projecto do sistema também são capturadas neste vista.
Esta vista captura os principais objectivos e restrições do sistema. Desta forma, os Casos de Utilização, que representam as funcionalidades do sistema ("o que o sistema deveria fazer"), enquadram-se como um importante elemento para representação desta vista, daqui surge então o estilo Casos de Utilização. Que vem completar a informação que as outras vistas, Módulo, Afectação e Componente e Conector documentam.

3.1.5 – Combinação dos estilos Utilização e Agregação

A combinação dos estilos Agregação e Utilização surgiu, porque, após a conclusão da modulação destes dois estilos, foi detectada uma forte relação entre ambos. Ou seja, a informação representada no estilo Agregação de um módulo específico estava também representada no estilo Utilização desse mesmo módulo; variando apenas na relação. Por esse motivo e para evitar o excesso de informação que os dois estilos separados iriam provocar, foram combinados estes dois estilos num só, denominado Dependência. O nome do estilo, surge das relações que este estilo suporta, são elas: “usa” e “agrega” e porque estas duas relações implicam uma dependência entre módulos. Este novo estilo herda as restrições dos estilos que o compõem, por esse motivo, o estilo dependência só será aplicado nos módulos cuja implementação siga o conceito da programação orientada a objectos (restrição esta imposta pelo estilo Agregação), nos restantes módulos será usado o estilo Utilização para mostrar a dependência entre módulos.
Esta combinação foi feita segundo o conceito de Estilo híbrido, isso implica que o estilo Dependência, tem as mesmas obrigações de documentação que os outros estilos apresentados

anteriormente, assim sendo, a especificação deste estilo e dos outros definidos nesta secção são especificados na secção 3.3 do presente capitulo.

3.2 Nomenclatura para Definição da Arquitectura

A modelação é a arte e ciência de criar modelos de uma determinada realidade. È uma técnica bem aceite e adoptada pela generalidade das disciplinas de engenharia conhecidas. Permite a partilha de conhecimento entre diferentes grupos de intervenientes (técnicos e não técnicos), facilita e promove a comunicação entre todos [Alberto Silva [7]].

Como vimos anteriormente a documentação da Arquitectura é um factor crucial para a compreensão do sistema. Vimos também que essa documentação deve seguir um formato predefinido de modo a facilitar a sua leitura e evitar que a mesma seja incompleta, inadequada, vaga ou demasiado detalhada. São então de realçar aqui duas expressões importantes "compreensão do sistema" e "facilitar a sua leitura". Se estes são os dois objectivos principais da documentação de uma arquitectura de software, para alcançá-los, a forma como será representada a informação, dentro do formato de documentação predefinido, assume uma importância relevante.
O uso de modelos (notação gráfica) tem inúmeras vantagens sobre o uso de linguagem descritiva, entre elas a maior facilidade de compreensão e leitura. Assim sendo, no caso da documentação da arquitectura do sistema ARQUO será usada sempre que possível a notação gráfica.
E mais uma vez, voltando aos dois principais objectivos da documentação de uma arquitectura, a compreensão e facilidade de leitura estarão dependentes do tipo de notação utilizada. Daí que a escolha da notação a usar seja um factor muito importante para que o resultado final da documentação da arquitectura atinja os resultados esperados. Tendo isto em conta a notação escolhida para a documentação do sistema ARQUO será a Unified Modelling Language – UML.
O motivo da escolha desta linguagem prende-se com as características da mesma e que contribuíram no desenvolvimento do trabalho:

- Padronização: a UML torna padrão toda a especificação do sistema, contribuindo assim para uma melhor compreensão da documentação do sistema, por qualquer pessoa envolvida no mesmo.
- Independência: é independente do domínio de aplicação e das ferramentas de modelação.
- Expansível: apresenta mecanismos potentes de expansão.
- Adaptável: agrega um conjunto muito significativo de diferentes diagramas/técnicas dispersos por diferentes linguagens.

A estrutura da UML é basicamente dividida em: elementos de modelo (por exemplo: classes, casos de uso, componentes), diagramas e relacionamentos. Possui, ainda, mecanismos de extensão, que fornecem a flexibilidade de estender os modelos.
Para a documentação do sistema ARQUO foram usados vários modelos (classes, interfaces, componentes, *nós*, artefactos, *packages* e outros), vários diagramas (diagrama de Classes, diagrama de Componentes, diagrama de Instalação) e vários relacionamentos (Associação, Agregação, entre outros) e ainda foram estendidos alguns modelos. Na secção seguinte são apresentados os modelos, diagramas e relações UML usadas para modelar cada estilo.

3.3 Especificação dos Estilos

Nesta secção são especificados os estilos usados para documentar as vistas do sistema ARQUO. A especificação dos estilos segue o *template* do Guia de Estilos apresentado no Anexo A, figura 35.

3.3.1 - Vista Módulo - Estilo Decomposição

Visão geral

O estilo Decomposição obtém-se a partir dos elementos e propriedades da vista Módulo, e da especialização da relação "*é-parte-de*". Este estilo é usado para mostrar como as responsabilidades do sistema estão repartidas nos módulos que implementam o sistema e como é que esses módulos estão decompostos em sub-módulos.
O estilo Decomposição deve ser usado para os stakeholders obterem uma visão abrangente do sistema; os níveis de decomposição a consultar dependem das necessidades de cada um dos stakeholders.

Elementos, relações e propriedades

A tabela seguinte mostra as características do estilo Decomposição. Os elementos são módulos, que podem ser unidades de implementação ou "*namespaces*" usados para organizar os vários módulos; que se relacionam através da relação "*é-parte-de*". Esta relação restringe os ciclos, ou seja, um módulo não pode conter na sua decomposição nenhum dos seus antecessores. Nenhum módulo no estilo Decomposição pode ter mais do que um pai.

Elementos	Módulos, como definidos no tipo de vista Módulo.
Relações	A relação "*é-parte-de*", como especialização da relação "*é-parte-de*" definida na vista Módulo.
Propriedades	As definidas na vista Módulo.
Topologia	Um módulo não pode fazer parte de mais do que um módulo numa vista

Tabela 1: Resumo das características do estilo Decomposição

Para que serve o estilo Decomposição e para que não serve

O estilo Decomposição apresenta as funcionalidades do sistema em vários níveis de detalhe, e por isso, a sua leitura é essencial no processo de conhecimento e aprendizagem de um sistema.
Este estilo pode ser usado como suporte á análise dos efeitos de alterações ao nível da implementação. No entanto, como este estilo não representa todas as dependências entre módulos, terá que ser usado juntamente com o estilo Utilização, Generalização e Agregação de forma a obter uma análise mais fiável.

Notações

A notação usada para representar os módulos do estilo Decomposição baseia-se nos modelos UML. No estilo Decomposição os módulos podem ser Classes ou *Packages*, como mostra a figura seguinte:

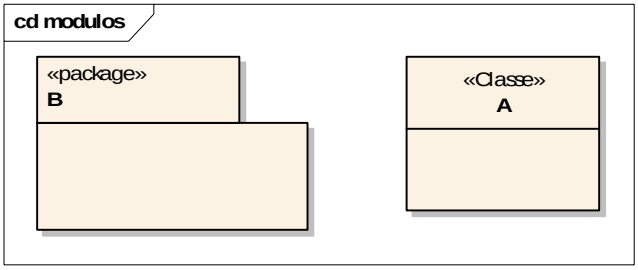


Figura 1: Notação para os elementos do estilo Decomposição

As classes e os *packages* podem assumir vários estereótipos, dependendo da linguagem em que está implementado esse módulo. As figuras seguintes, mostram esses estereótipos, com a indicação de qual a linguagem de programação a que correspondem.

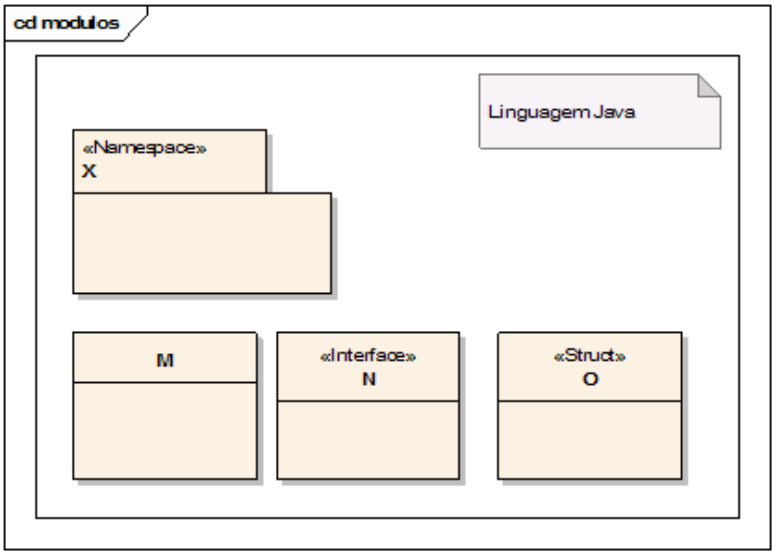


Figura 2: Estereótipos dos módulos do estilo Decomposição - na linguagem Java

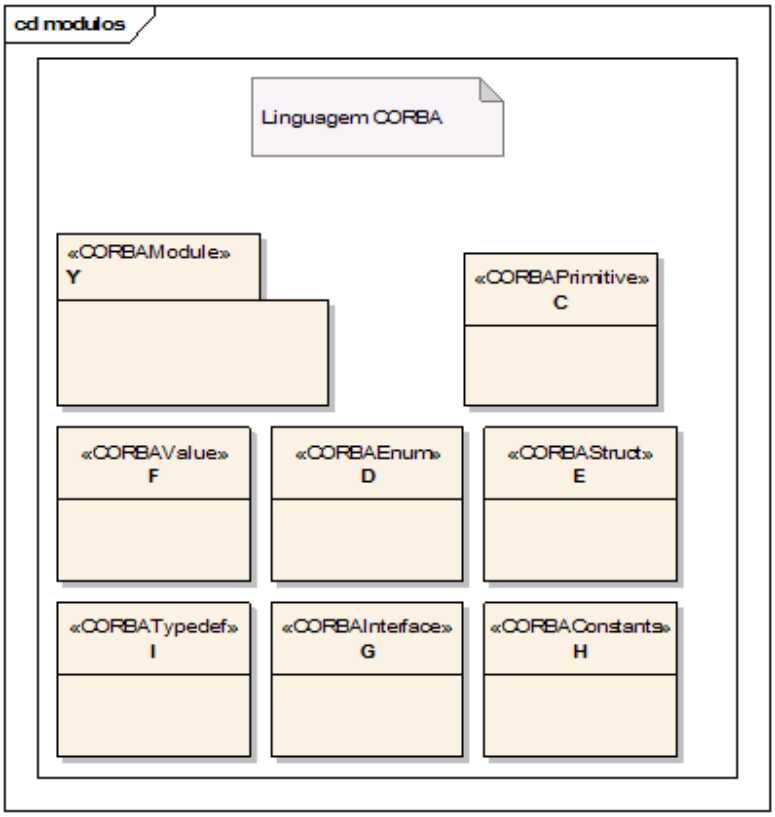


Figura 3: Estereótipos dos módulos do estilo Decomposição – JSP e JavaScript

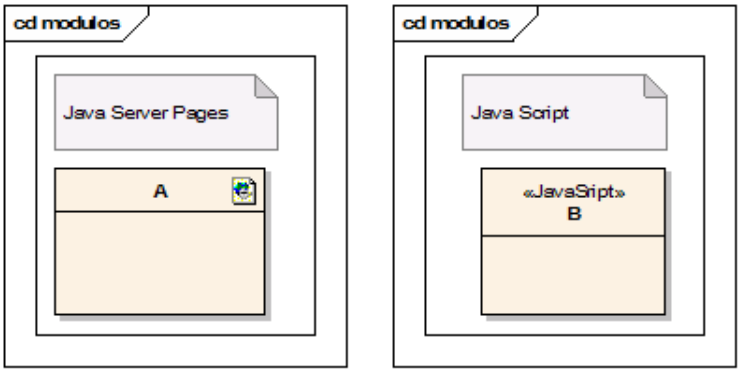


Figura 4: Estereótipos dos módulos do estilo Decomposição – na linguagem CORBA

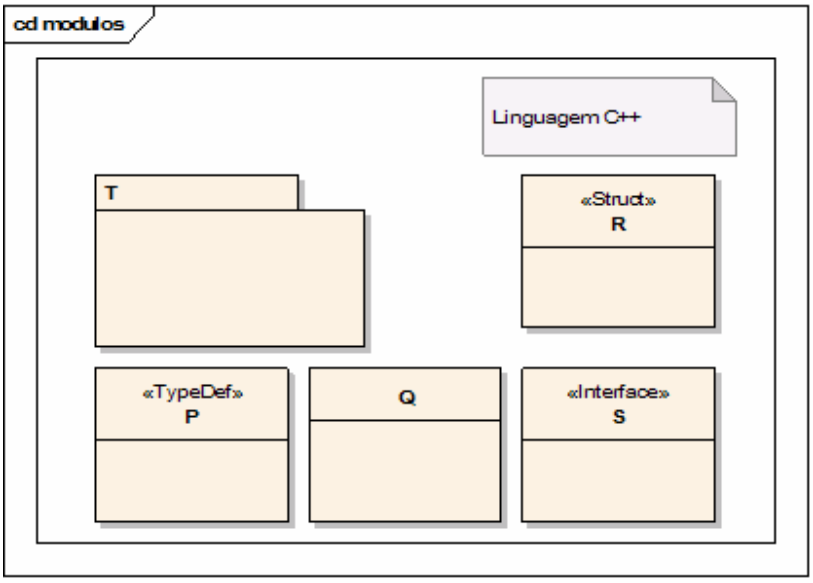


Figura 5: Estereótipos dos módulos do estilo Decomposição – da linguagem C++

A figura 6 mostra a relação "*é-parte-de*" entre os módulos do estilo Decomposição.

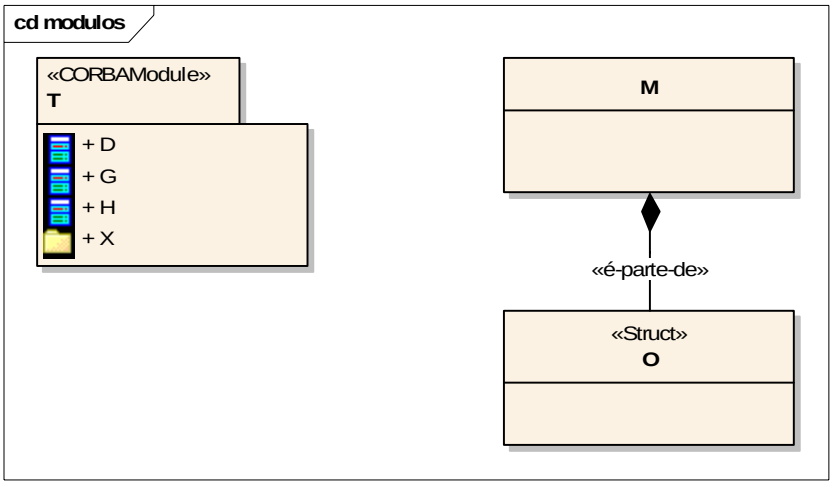


Figura 6: Notação para a relação "*é-parte-de*"

Os módulos D, G, H e X têm uma relação "*é-parte-de*" com o módulo T. O módulo O é parte do módulo M.
Um módulo do tipo *Package* pode conter módulos do tipo Classe ou também módulos do tipo *Package*. A relação "*é-parte-de*" entre duas classes é representada através de uma linha com um losango preenchido na extremidade da classe que é composta pela outra classe.

O comportamento dos elementos, no estilo Decomposição, será descrito através da informação sobre os atributos e métodos dos módulos em estudo, caso os módulos sejam implementados na linguagem Java, C++ ou Corba, ver figura 7. No caso das Páginas Jsp e JS o comportamento dos elementos será feito através de uma breve descrição do módulo.

A notação será a definida por UML para representação dos atributos e métodos de uma classe, como mostra a figura seguinte.

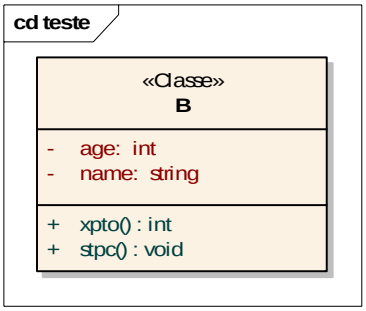


Figura 7: Notação para o comportamento dos elementos do estilo Decomposição

O comportamento da classe B reflecte-se nos dois atributos “age” e “name” e nos dois métodos “xpto” e “stpc”.

Relação com os outros estilos

Existe uma ligação entre o estilo Decomposição e o estilo Implementação da vista Afectação. Visto que o estilo Implementação documenta a forma como os módulos do estilo Decomposição estão mapeados em ficheiros ou directorias.
O estilo Decomposição está também relacionado com a vista Componente e Conector, na medida em que as unidades de implementação estão relacionadas com as estruturas de execução.

Exemplos do Estilo Decomposição

Como exemplo do estilo Decomposição, considere-se as seguintes linhas de código:

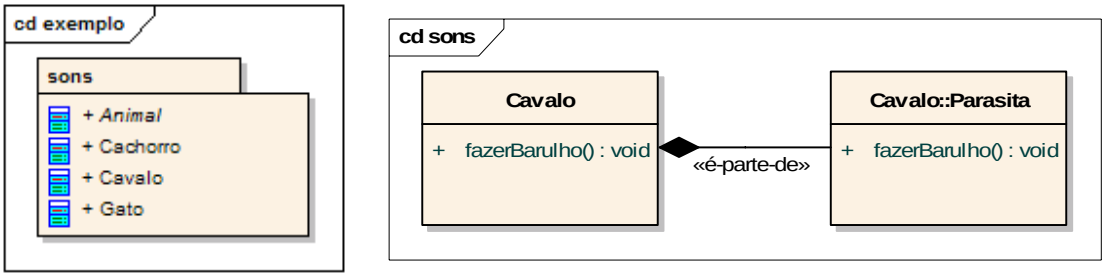
```
package sons;

public abstract class Animal {
  public abstract void fazerBarulho();
}

public class Cachorro extends Animal {
  public void fazerBarulho() {

      System.out.println("AuAu!");
  }
}

public class Gato extends Animal {
  public void fazerBarulho() {
     System.out.println("Miau!");
  }
}
```

```
public class Cavalo extends Animal {
  public void fazerBarulho() {
     System.out.println("RIINCH!");
  }
     /*Classe interna e privada.
      Existe só no contexto do cavalo.*/
  private class Parasita extends Animal {
     public void fazerBarulho() {
        System.out.println("SQRRT");
     }
  }
}
```

Figura 8: Exemplo de um ficheiro de código para exemplificar o estilo Decomposição.
(Exemplo adaptado da referência [21])

Nesta porção de código são identificados seis módulos, são eles o *package* Sons e as classes Animal, Cachorro, Gato, Cavalo e Parasita.

As figuras seguintes mostram a primeira apresentação do estilo Decomposição para o exemplo de código anteriormente apresentado.



Os sub módulos Animal, Cachorro, Cavalo e Gato têm uma relação "*é-parte-de*" com o módulo Sons.
O sub módulo Parasita tem uma relação "*é-parte-de*" com o sub módulo Cavalo que por sua vez faz parte do módulo Sons.

3.3.2 – Vista Módulo - Estilo Generalização

Visão geral

Através da especialização da relação "*é-uma*" da vista Módulo obtemos o estilo Generalização. Os módulos que têm uma relação de generalização entre eles, podem ser chamados de pais e filhos, onde o módulo pai é uma versão mais geral do módulo filho. Os termos pai e filho são partilhados com o estilo Decomposição, mas assumem um significado distinto em cada estilo. No estilo Decomposição, o pai é composto pelos seus filhos. No estilo Generalização, os pais e filhos têm comportamentos em comum. Os módulos do estilo Generalização estão definidos de tal forma que capturam as variações e as analogias. O módulo pai contém as analogias e as variações são manifestadas no módulo filho. Podem ser feitas extensões, adicionando, removendo ou alterando os filhos; as alterações no módulo pai vão alterar automaticamente todos os filhos que lhe são inerentes.

Elementos, relações e propriedades

A tabela seguinte mostra características do estilo Generalização. Os elementos do estilo Generalização são classes, uma especialização dos módulos definidos na vista Módulo, porque este estilo tem como base a orientação a objectos. Por exemplo, não faz sentido o estilo Generalização aplicado a páginas JSP.
A especialização da relação "*é-uma*" da vista Módulo é a relação da vista Generalização, a relação entre dois módulos significa que um módulo é a generalização do outro e o outro a especialização do primeiro.

Elementos	Classes, uma especialização dos módulos da vista Módulo.
Relações	A relação "*é-uma*" especializada da vista Módulo.
Propriedades	As definidas na vista Módulo.
Topologia	Um módulo pode ter múltiplos pais, no entanto não é de boa prática que isso aconteça. E um módulo pode ter múltiplos filhos. Os ciclos não são permitidos, um módulo do estilo Generalização não pode ter uma relação "*é-uma*" com ele próprio.

Tabela 2: Resumo das características do estilo Generalização.

No estilo Generalização o módulo que herda informação é referido como o módulo descendente e o módulo que fornece a informação é o ascendente. Os ciclos não são permitidos, ou seja, um módulo não pode ser ascendente ou descendente dele próprio.

Para que serve o Estilo Generalização e para que não serve

O estilo Generalização pode ser usado para suportar evoluções e extensões, tendo em conta que este estilo mostra o que há de comum e de distinto entre dois módulos e que é muito mais fácil perceber o comportamento de dois módulos sabendo o que têm em comum e em que diferem do que tentar percebe-los de raiz. È usado também para facilitar a reutilização de código e as alterações ao mesmo.

Notações

A notação usada para representar os módulos do estilo Generalização baseia-se nos modelos UML. No estilo Generalização os módulos são Classes. Tal como no estilo Decomposição as classes do estilo Generalização podem ter vários estereótipos, mas não tantos como no estilo Decomposição. Os estereótipos usados no estilo Generalização são mostrados nas figuras seguintes:

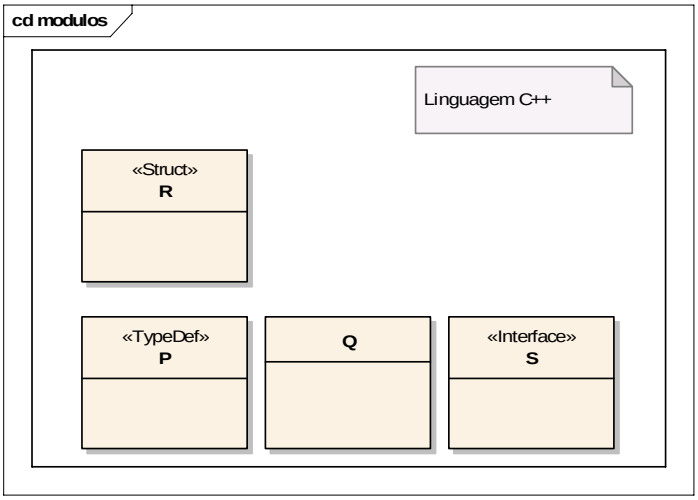


Figura 9: Estereótipos dos módulos do estilo Generalização – linguagem C++

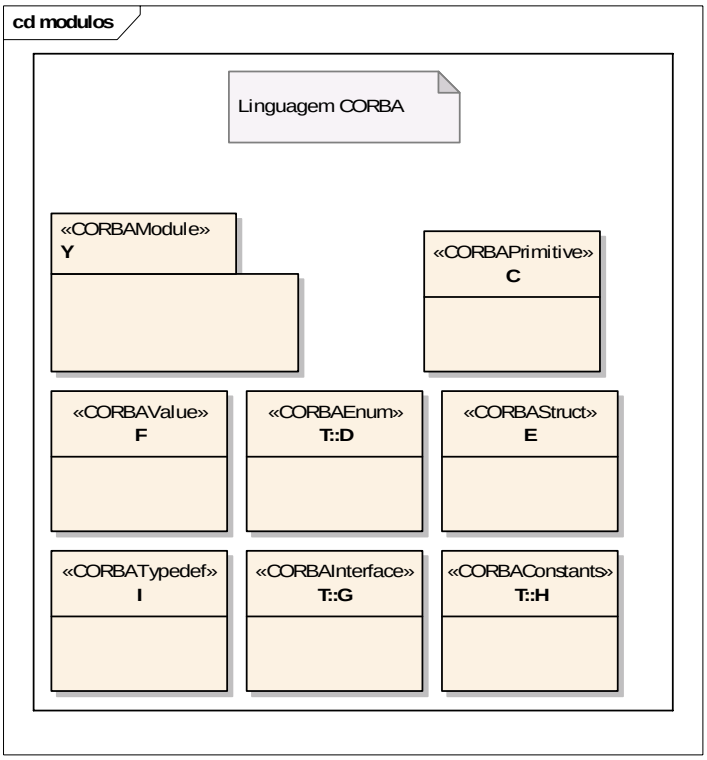


Figura 10: Estereótipos dos módulos do estilo Generalização –Linguagem CORBA

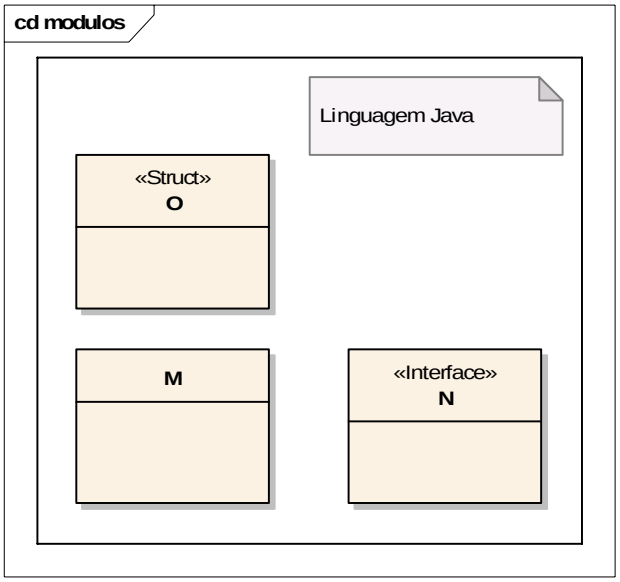


Figura 11: Estereótipos dos módulos do estilo Generalização – Linguagem Java

A relação "*é-uma*" é representada através de uma linha que liga a classe especializada á classe generalizada e que contém um triângulo na extremidade junto a classe que generaliza. A figura 12, mostra essa representação.

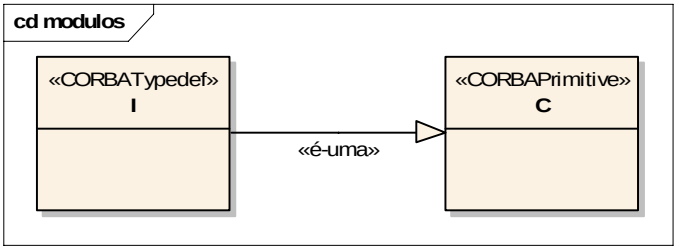


Figura 12: Relação "é-uma"

O comportamento dos elementos, no estilo Generalização, será descrito da mesma forma que no estilo Decomposição. Ver figura7.

Relação com os outros estilos

O estilo Generalização complementa os outros estilos da vista Módulo, nomeadamente o estilo Decomposição, o estilo Utilização e o estilo Agregação.

Exemplos do Estilo Generalização

Voltando ao exemplo apresentado no estilo Decomposição. E á análise do código apresentado na figura 8, podemos identificar a relação "*é-uma*" através da palavra-chave "*extend*" na declaração da classe.

A figura seguinte mostra como seria a Primeira Apresentação do estilo Generalização para este exemplo.

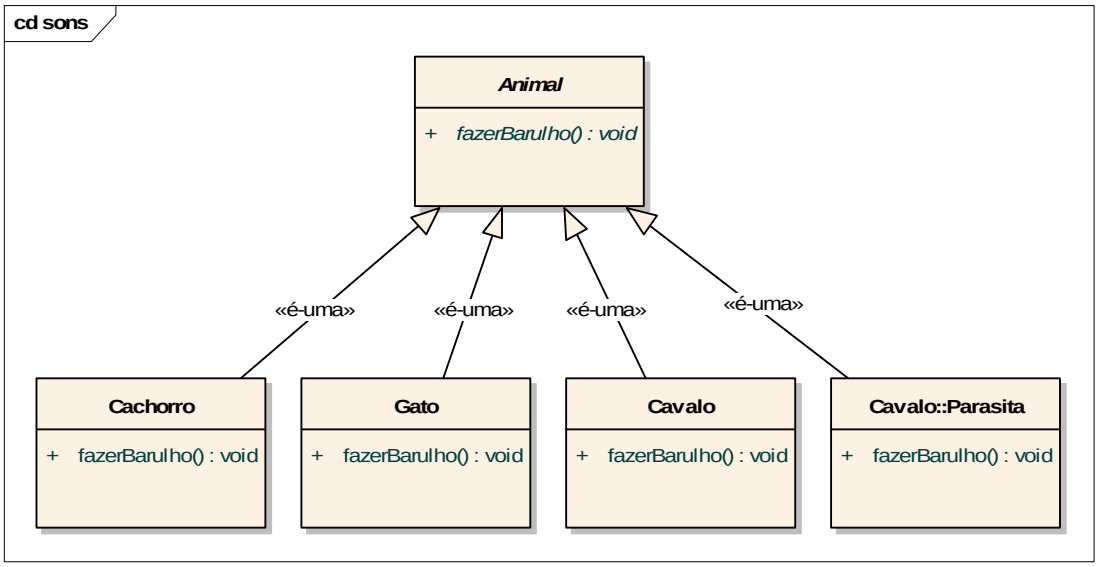


Figura 13: Exemplo do estilo Generalização
(Exemplo adaptado da referência [21])

Os módulos Cachorro, Gato, Cavalo e Parasita têm uma relação "*é-uma*" com o módulo Animal.

3.3.3 – Vista Módulo -Estilo Utilização

Visão geral

O estilo Utilização surge da especialização da relação "*depende-de*"da vista Módulo, na relação "usa". Este estilo mostra como é que os módulos dependem uns dos outros através da relação "usa". Este estilo, serve essencialmente á equipa de desenvolvimento, visto que contém a informação de que outros módulos devem existir para que a sua porção de sistema funcione correctamente.

Elementos, relações e propriedades

A tabela seguinte mostra as características do estilo Utilização. Os elementos são Módulos, como definidos na vista Módulo; que se relacionam através da relação "usa".
Um módulo A usa o módulo B se A depende do correcto funcionamento de B de forma a satisfazer os seus requisitos.

Elementos	Módulos, como definidos no tipo de vista Módulo.
Relações	A relação "usa", como refinamento da relação "*depende-de*" definida na vista Módulo.
Propriedades	As definidas na vista Módulo.
Topologia	Este estilo não tem restrições topológicas

Tabela 3: Resumo das características do estilo Utilização

Para que serve o estilo Utilização e para que não serve

O estilo Utilização é útil para facilitar o planeamento de desenvolvimento incremental e extensões do sistema e de subconjuntos, para suportar testes e a eliminação de erros e para ajudar a prever efeitos de alterações específicas.

Notações

A notação usada para representar os módulos do estilo Utilização baseia-se nos modelos UML. No estilo Utilização os módulos podem ser Classes ou *Packages*, tais como definidos no estilo Decomposição.

A figura 14 mostra a relação "usa" entre os módulos do estilo Utilização. A relação "*usa*" pode ser usada entre dois ou mais *packages*, entre duas ou mais classes e entre *Packages* e classes no entanto esta última relação não é muito conveniente pois pode causar ambiguidades de interpretação.

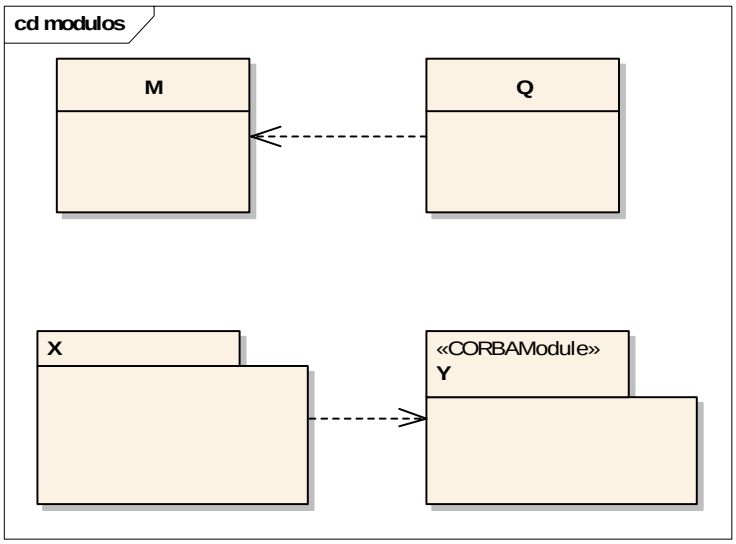


Figura 14: Notação para a relação "usa"

A relação "usa" entre dois módulos é representada através de uma linha a tracejado com uma seta na extremidade do módulo usado.

O comportamento dos elementos, no estilo Utilização, será descrito através da informação sobre os atributos e métodos dos módulos em estudo, caso os módulos sejam implementados na linguagem Java, C++ ou Corba, como definido no estilo Decomposição, ver figura 7. No caso das Páginas Jsp e JS o comportamento dos elementos será feito através de uma breve descrição do módulo.

A notação usada para representar as interfaces será a representada na figura 15. Os modelos de notação usados são as classes, interfaces e portas definidas pelo UML.
A interface entre os módulos será notada através de interfaces UML que vão representar os métodos através dos quais é feita a interacção entre os módulos. A figura seguinte mostra essa representação:

Figura 15:Notação para as interfaces dos elementos no estilo Utilização

Os métodos são representados como interfaces que a classe usada disponibiliza e a classe que usa requer. A interface é feita através de portas.
Através do exemplo da figura podemos ver que a interface entre as classes A e B é feita através dos métodos Xpto e Sptc. Métodos esses que são disponibilizados pela classe B e requeridos pela classe A.
Para cada um dos métodos de implementação disponibilizados pela classe B é feita uma representação de informação sobre a definição dos mesmos, nomeadamente as excepções, os parâmetros de entrada e o retorno.

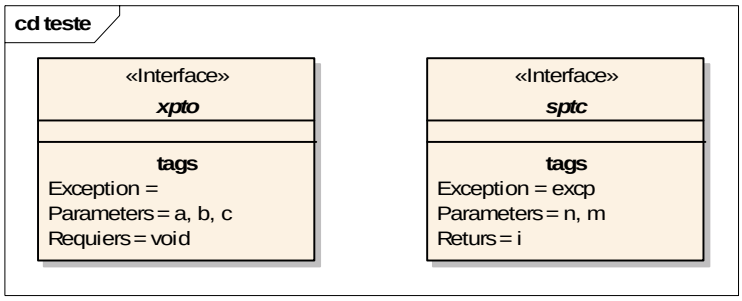


Figura 16: Notação para representar os métodos usados como interface na relação "usa"

A figura seguinte mostra a definição dos métodos que fazem a interacção entre os módulos A e B.
O comportamento dos elementos será descrito através da informação sobre os atributos e métodos dos módulos em estudo. Será a mesma utilizada para documentar o comportamento dos elementos do estilo Decomposição. Ver figura 7.

Relação com os outros estilos

Uma ligação entre o estilo Utilização, o estilo Decomposição e o estilo Agregação permite uma análise do sistema.

Exemplos do Estilo Utilização

Como exemplo do estilo Utilização, considere-se o seguinte diagrama da figura 17. Este é obtido através da análise do código apresentado nas figuras 18 e 19 (fonte [19] – Curso de C++).

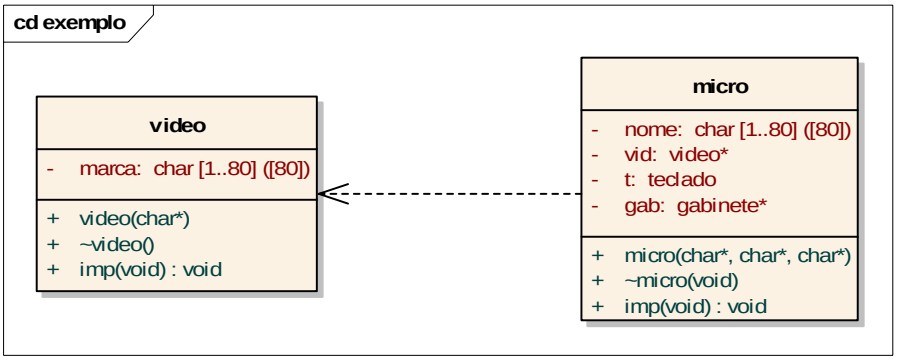


Figura 17: Exemplo do estilo Utilização

Os módulos deste estilo serão as classes Micro e Vídeo. O módulo Micro tem uma relação "usa" com o módulo Video. No exemplo, esta relação surge do facto da implementação da classe Micro depender do correcto funcionamento da classe Video, visto que a primeira inclui (através da instrução "*include*") o ficheiro de declaração da classe Video, video.h.

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>

#include video.h;

class micro
{
char nome[80];
video *vid;
public:
micro(char *nome,char *tipo_cpu,char
*marca_video);
~micro(void);
void imp(void);
};

// Implementacao da classe micro
micro::micro(char *n,char *tipo_cpu,char
*marca_video)
:t("Teclado comum - 128 teclas")
{
strcpy(nome,n);
vid = new video(marca_video);
}
micro::~micro()
{
delete vid;
printf(""Destruiu micro:" << nome << "\n";
}
void micro::imp()
{
printf(""\nDescricao do micro:" << nome <<
"\n";
vid->imp();
printf(""\n";
}
// Rotina principal
void main()
{
micro *micro1;
micro micro2("Micro2","386 DX 25","Nec
MultiSync 3D");
micro1 = new micro("Micro1","486 DX
66","Samsung SyncMaster3");
micro1->imp();
micro2.imp();
delete micro1;
}
```

Figura 18: Exemplo -Implementação da classe Micro.

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>

class video
{
char marca[80];
public:
video(char *m) { strcpy(marca,m);}
~video() { printf("Destruiu o video\n"); }
void imp(void);
};

// Implementacao da classe video
void video::imp()
{
printf(""Marca do video: " << marca <<
"\n");
}
```

Figura 19: Exemplo- Implementação da classe vídeo.h

3.3.4 –Vista Módulo - Estilo Agregação

Visão geral

O estilo Agregação surge da necessidade de especialização da relação "*é-parte-de*" definida no tipo de vista Módulo. Sendo que, o estilo Decomposição já é uma especialização dessa mesma relação, o estilo Agregação difere do mesmo na forma como a relação "*é-parte-de*" é especializada. No estilo Decomposição a relação "*é-parte-de*" representa os módulos que são decompostos em sub módulos; por exemplo uma classe "*é-parte-de*" num *namespace;* ou seja o estilo Decomposição é usado mais no sentido de organização dos módulos como unidades de implementação. No caso do estilo Agregação a relação "*é-parte-de*" tem um significado diferente, e por isso terá outro nome: "agrega". A relação "agrega" é então um tipo de associação entre dois ou mais módulos, e indica que um dos módulos participantes precisa desempenhar o papel do controlador do(s) outro(s) módulo(s) associado(s) e que os dois (ou mais) módulos comportam-se como um só; introduz-se assim o conceito "agregação" da programação Orientada a Objectos que é a base do estilo Agregação.

Elementos, relações e propriedades

A tabela seguinte mostra características do estilo Agregação. Os elementos são módulos, unidades de implementação; que se relacionam através da relação "*agrega*".
O estilo Agregação só pode ser usado quando os módulos associados são do tipo Classes ou Estruturas. Não faz sentido usar a relação "*agrega*" nos outros tipos de módulos.

Elementos	Módulos, que podem ser classe ou estruturas.
Relações	A relação "*agrega*", como especialização da relação "*é-parte-de*" definida na vista Módulo.
Propriedades	As definidas na vista Módulo.
Topologia	Um módulo pode ser "agregado" por mais do que um módulo.

Tabela 4: Resumo das características do estilo Agregação.

Para que serve o Estilo Agregação e para que não serve

Este estilo pode ser usado para fazer uma análise ao sistema visto que, mostra algumas dependências entre módulos. No entanto não mostra todas as dependências, assim a análise ao sistema será mais completa se usarmos além do estilo Agregação, os estilos Decomposição e Utilização.

Notações

A notação usada para representar o estilo Agregação é a notação definida para o conceito Agregação da programação Orientada a Objectos, o diagrama de classes com a relação agregação, como mostra a figura seguinte. Os módulos no estilo agregação são classes.

A figura seguinte mostra a notação para a relação "*agrega*". Os módulos M e N estão relacionados através da relação "agrega" que é representada com uma linha com um losango na extremidade junto á classe controladora.

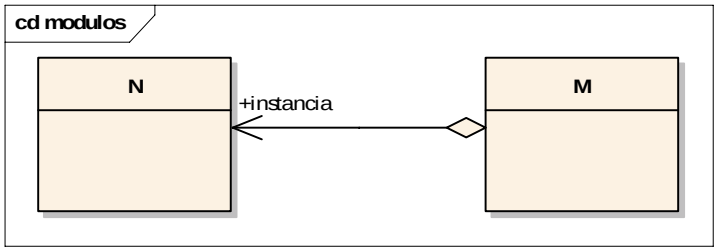


Figura 20: Notação para a relação "agrega"

A notação usada para representar as interfaces será a representada na figura 21. Os modelos de notação usados são as classes, interfaces e portas definidas pelo UML.

A interface entre os módulos será notada através de interfaces UML que vão representar os métodos através dos quais é feita a interacção entre os módulos. A figura seguinte mostra essa representação:

Figura 21: Notação para as interfaces entre módulos com a relação "agrega"

Os métodos são representados como interfaces que a classe controlada disponibiliza e a classe controladora requer. A interface é feita através de portas.

Através do exemplo da figura podemos ver que a interface entre as classes A e B é feita através dos métodos Xpto e Sptc. Métodos esses que são disponibilizados pela classe B e requeridos pela classe A. Na classe A a porta que faz a interacção entre as duas classe é o nome da instância (objecto).

Para cada um dos métodos de implementação disponibilizados pela classe B é feita uma representação de informação sobre a definição dos mesmos, nomeadamente as excepções, os parâmetros de entrada e o retorno. A figura seguinte mostra a definição dos métodos que fazem a interacção entre os módulos A e B.

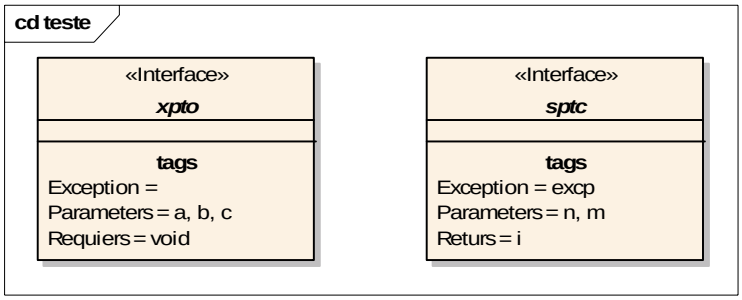


Figura 22: Representação dos métodos que fazem a interface na relação "agrega"

O comportamento dos elementos será descrito através da informação sobre os atributos e métodos dos módulos em estudo. Será a mesma utilizada para documentar o comportamento dos elementos do estilo Decomposição. Ver figura 7.

Relação com os outros estilos

O estilo Agregação está relacionado com o estilo Utilização, pois os módulos que se relacionam através da relação "agrega" definida anteriormente também se relacionam através da relação "usa" do estilo Utilização.

Exemplo do Estilo Agregação

Na figura seguinte podemos ver um exemplo do estilo Agregação. Os módulos deste estilo serão as classes Micro e Vídeo. A classe Micro tem uma relação "*agrega*" com a classe vídeo. Esta relação é obtida através da análise do código apresentado nas figuras 18 e 19.

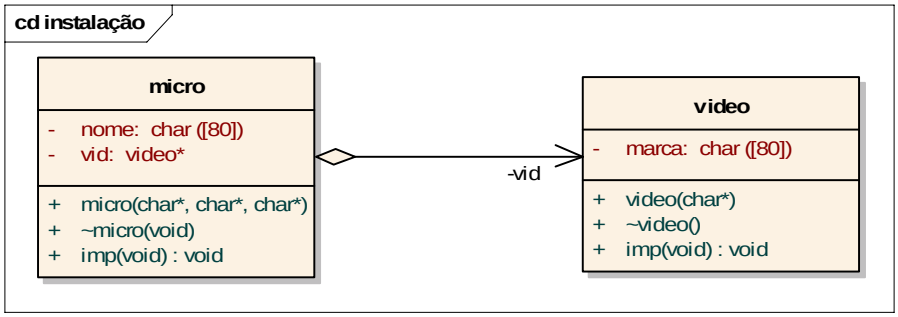


Figura 23: Exemplo da relação "agrega"
(fonte [19] – Curso de C++)

Esta relação surge do facto de um dos atributos da classe Micro se uma instância da classe Vídeo.

3.3.9 – Vista Módulo - Estilo Dependência

Visão geral

Este estilo surge através da combinação híbrida dos estilos Utilização e Agregação da vista Módulo. Este estilo surge após a análise das relações "usa" e "agrega". O resultado dessa análise mostrou que todos os elementos do estilo Agregação de um módulo específico, eram também elementos do estilo Utilização para esse mesmo módulo. A relação "*agrega*" pode então passar a ser vista como uma especialização da relação "usa". Desta combinação surge o estilo Dependência, o nome do estilo tem praticamente o mesmo significado do nome do estilo Utilização, que é um dos estilos base desta combinação, isso deve-se ao facto de que a relação mais importante desta combinação é a relação "usa"; e que a relação "agrega" passa a ser uma especialização da mesma.
Assim, sempre que uma classe utilizar objectos de outras classes como atributos seus a relação a usar será a relação "agrega". Para os restantes casos de dependência entre módulos a relação será "usa".

Elementos, relações e propriedades

A tabela seguinte mostra características do estilo Dependência. Os elementos desta combinação, assumem as restrições da vista Agregação, ou seja, este estilo só pode ser usado quando os módulos associados são do tipo Classe. Estes elementos relacionam-se através da relação "usa" ou "agrega", consoante o tipo de utilização.

Elementos	Módulos, como definidos no tipo de vista Módulo.
Relações	A relação "agrega", como especialização da relação "*usa*". A relação "usa", como especialização da elação "*depende-de*" da vista Módulo.
Propriedades	As definidas na vista Módulo.
Topologia	Um módulo pode ser "agregado" por mais do que um módulo.

Tabela 5: Resumo das características do estilo Dependência.

Para que serve o Estilo Dependência e para que não serve

Este estilo pode ser usado para fazer uma análise ao sistema pois, mostra algumas dependências entre módulos. No entanto não mostra todas as dependências, assim a análise ao sistema será mais completa se usarmos além do estilo Dependência, os estilos Decomposição e Generalização.

Notações

A notação usada para representar o estilo Dependência é o diagrama de classes com a relação agregação para o caso da relação "agrega" e uma relação de dependência para a relação "usa", como mostra a figura seguinte. Os módulos no estilo Dependência são classes.

O módulo M tem uma relação “agrega” com o módulo N, relação esta que é representada tal como no estilo Agregação, definido anteriormente, com uma linha com um losango na extremidade junto á classe controladora.

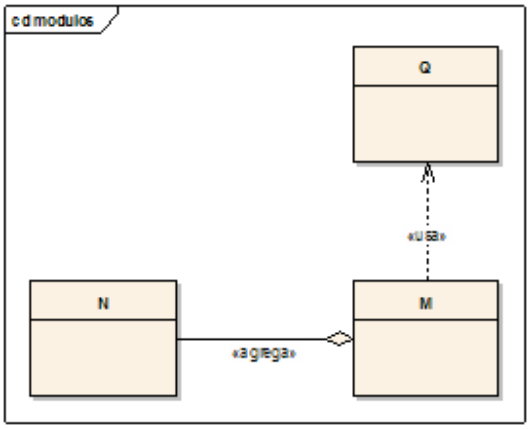


Figura 24: Notação dos elementos e relações do estilo Dependência

O módulo M tem também uma relação “usa” com o módulo Q, relação esta que é representada tal como no estilo Utilização, definido anteriormente, com uma linha a tracejado com uma seta na extremidade junto ao módulo usado.

A notação usada para representar as interfaces será a representada na figura 21. Os modelos de notação usados são as classes, interfaces e portas definidas pelo UML.
A interface entre os módulos será notada através de interfaces UML que vão representar os métodos através dos quais é feita a interacção entre os módulos. Tal como definido no estilo Utilização.

Os métodos são representados como interfaces que a classe controlada disponibiliza e a classe controladora requer. A interface é feita através de portas. A representação das interfaces é também igual á definida nos estilos Agregação e Utilização.
Para cada um dos métodos de implementação disponibilizados pela classe B é feita uma representação de informação sobre a definição dos mesmos, nomeadamente as excepções, os parâmetros de entrada e o retorno, como definido nos estilos base desta combinação.

O comportamento dos elementos será modelado como definido nos estilos Agregação e Utilização.

Relação com os outros estilos

O estilo Dependência tem uma relação com o estilo Decomposição, visto que os módulos que usam e são usados são os descritos no estilo Decomposição.

Exemplos do Estilo Dependência

Na especificação dos estilos Utilização e Agregação podemos ver, através dos exemplos apresentados para cada um dos estilos, que entre as classe Micro e Vídeo existem duas relações “usa” e “agrega”. A relação mais correcta neste caso, seria a relação “agrega” visto

que contém mais informação do que a relação “usa”, no entanto a relação “usa” tem todo o sentido de existir no conceito do estilo Utilização.

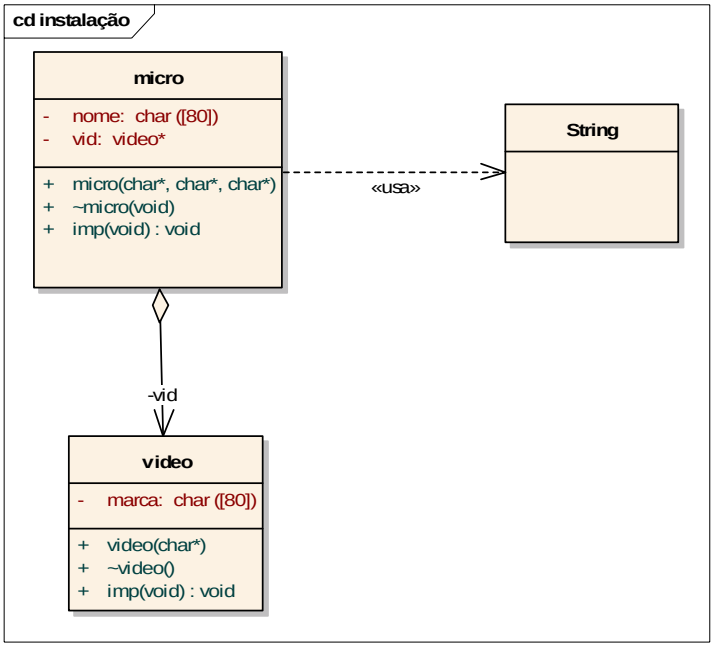


Figura 25: Exemplo da notação do estilo Dependência
(fonte [19] – Curso de C++)

A opção de combinar estes dois estilos permite que ao relacionar os dois módulos através da relação “agrega” subentende-se que o módulo Micro “usa” o módulo “Vídeo”, visto que a relação “agrega” aqui definida é um especialização da relação “usa” dentro do conceito agregação da programação OO. Evitando assim, informação repetida, que possa ser interpretada como inconsistente, e ainda com a vantagem de diminuir o número de vistas, facilitando assim a leitura e compreensão do pacote de Documentação.

3.3.5- Vista Componente & Conector - Estilo Cliente Servidor

Visão geral

No estilo Cliente-Servidor da vista C&C os componentes assumem dois tipos de comportamento: o de Cliente que requisita serviços e o de Servidor que fornece os serviços de resposta ás requisições dos Clientes. A particularidade deste estilo é que a comunicação é feita por pares e é sempre iniciada pelo componente Cliente. Os Servidores neste estilo fornecem um conjunto de serviços através de interfaces, que podem ser requeridos pelos clientes.

Elementos, relações e propriedades

No estilo Cliente-Servidor, os componentes são do tipo clientes e servidores. O principal tipo de conector no estilo Cliente-Servidor é o conector pedido-resposta (ou seja, os conectores serão as interfaces que cada componente disponibiliza e que os outros componentes podem requisitar para comunicar com esse componente) usado para invocar

serviços. Os servidores podem também assumir o comportamento de clientes através da requisição de serviços a outros servidores.

Elementos	Clientes e servidores.
Relação	A relação "interligação", como definida na vista C&C
Propriedades	As mesmas definidas para o estilo C&C
Topologia	Não existem restrições no que diz respeito á topologia.

Tabela 6:Resumo das características do estilo Cliente Servidor

Para que serve o Estilo Cliente Servidor e para que não serve

O estilo cliente servidor apresenta-se como uma visão do sistema que relaciona as aplicações cliente com os serviços que estas usam. O agrupamento de funcionalidades deste estilo fornece uma base importante para determinar a afectação do sistema com a plataforma de hardware.

Notações

A notação usada para este estilo é representada na figura 24. Os elementos deste estilo são, como referidos anteriormente, Clientes e Servidores e a notação usada para os mesmos são os Componentes - modelo da UML. Cada componente tem a representação da interface que fornece ou requer.

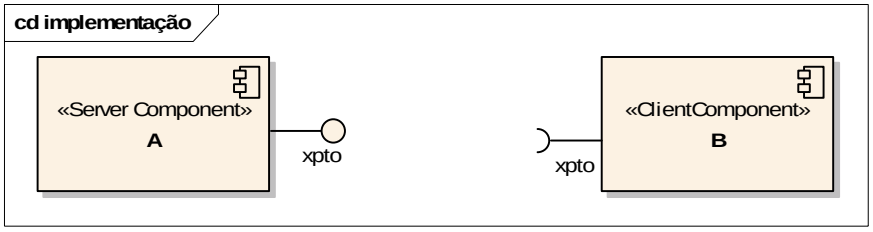


Figura 26: Notação para o estilo Cliente Servidor

O componente A, tem o comportamento de Servidor, fornece a interface Xpto através da qual ira fornecer os serviços que são requeridos pelo Cliente B.
A interface entre estes dois componentes é representada através da especificação dos serviços que o servidor fornece e o Cliente requer. Como mostra a figura seguinte:

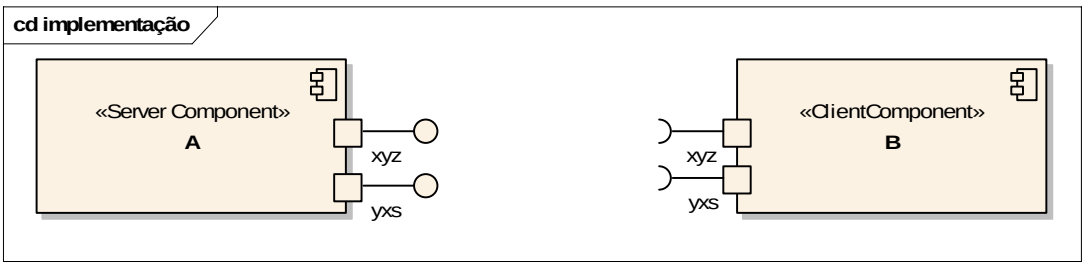


Figura 27: Representação dos serviços que fazem a interacção entre clientes e servidores

O servidor A fornece dois serviços, xyz e yxs que são requeridos pelo Cliente B. As portas aqui representam a interface Xpto.

Relação com os outros estilos

O estilo Cliente Servidor está relacionado com o estilo Instalação, porque os componentes deste estilo fornecem uma base importante para determinar a afectação do sistema com a plataforma de hardware.
O estilo Cliente Servidor está também relacionado com o estilo Decomposição, visto que as unidades de implementação, módulos, estão mapeadas com as estruturas de execução nomeadamente com os componentes.

Exemplos do Estilo Cliente Servidor

Suponha-se, a título de exemplo, (adaptado da referência [18]) de reservas on-line de um restaurante. O sistema é composto pelos seguintes componentes, uma aplicação web - Restaurant Service, o sistema de reservas –OrderSystem- um sistema de pagamentos – TaxSystem e o sistema do restaurante – RestaurantSystem. As interfaces requeridas e disponibilizadas serão os conectores deste estilo.

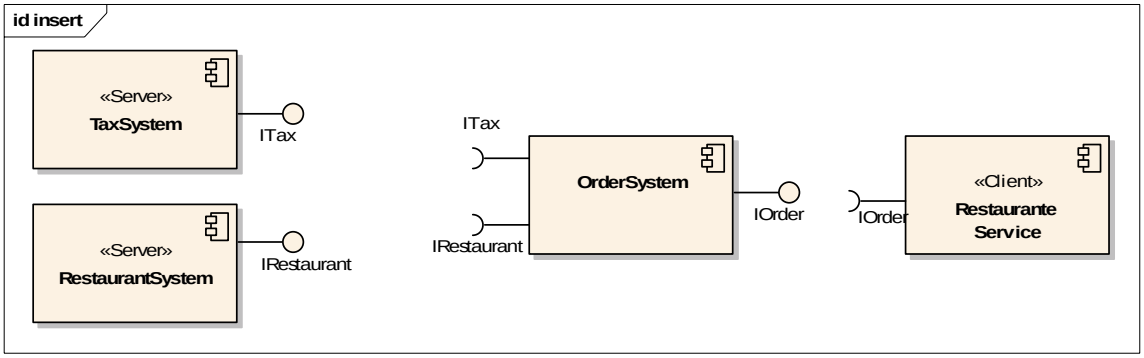


Figura 28: Exemplo do estilo Cliente Servidor

O sistema OrderSystem assume o papel de Servidor, quando a aplicação RestaurantServices, cliente, requer os serviços do mesmo. Essa interacção é feita através da interface IOrder que o sistema OrderSystem disponibiliza. Por outro lado, o sistema OrderSystem assume o papel de Ciente quando interage com o sistema de pagamentos TaxSystem e com o sistema RestaurantSystem. Os servidores sistemas TaxSystem e RestauranteSystem interagem com o cliente OrderSystem através das interfaces que disponibilizam, ITax e IRestaurant, respectivamente. Como mostra a figura 26.

3.3.6 - Estilo Processos Comunicantes

Visão geral

O estilo Processos Comunicantes é caracterizado através da interacção entre componentes de execução através de vários mecanismos de conexão. Um exemplo de um mecanismo de conexão é a troca de mensagens.

Elementos, relações e propriedades

Os elementos do estilo Processos Comunicantes são unidades de execução. O estilo representa um sistema como um conjunto de unidades concorrentes de execução, juntamente com as suas interacções. Os conectores permitem essa interacção através da troca de dados entre as unidades de execução.

Elementos	Componentes de Software e Componentes de Execução.
Relação	A relação “interligação”, como definida na vista C&C
Propriedades	As mesmas definidas para o estilo C&C
Topologia	Não existem restrições no que diz respeito á topologia.

Tabela 7: Resumo das características do estilo Processos Comunicantes

Para que serve o Estilo Processos Comunicantes e para que não serve

O estilo Processos Comunicantes aqui definido é útil para a equipa de implementação poder analisar a forma como os processos comunicam entre si, e a troca de mensagens que é feita entre os mesmos.

Notações

A notação usada para o estilo Processos Comunicantes foi o diagrama de sequência da UML. Os diagramas de sequência descrevem a forma como as unidades de execução (neste caso: objectos) colaboram em algum comportamento ao longo do tempo. Essa colaboração é demonstrada através da troca de mensagens entre os mesmos.

Os diagramas de sequência, da UML, estão sempre associados a um caso de utilização, o que implica que associada a esta vista esteja um levantamento dos casos de utilização do sistema em estudo, representado na vista Casos de Utilização.

A notação usada para descrever as interfaces entre os componentes do estilo Processos Comunicantes será a mesma que usada no estilo Decomposição para descrever os métodos que fazem a interacção entre os módulos (figura 16).

Relação com os outros estilos

Este estilo está relacionado com o estilo Cliente Servidor na medida em que mostra a sequência de troca de mensagens dos componentes do sistema. Está relacionado também com a vista Casos de Utilização, visto que a interacção entre os componentes existe tendo em conta uma funcionalidade do sistema. Existe uma relação entre o estilo Processos Comunicantes e o estilo Decomposição da vista Módulo, visto que é este ultimo que apresenta as unidades que implementam os componentes da vista Processos Comunicantes.

Exemplos do Estilo Processos Comunicantes

O exemplo seguinte (fonte: [18]) mostra um diagrama de sequências de um sistema de JuKeBox. Este diagrama de sequência está associado ao caso de utilização: Remover Musica da PlayList.

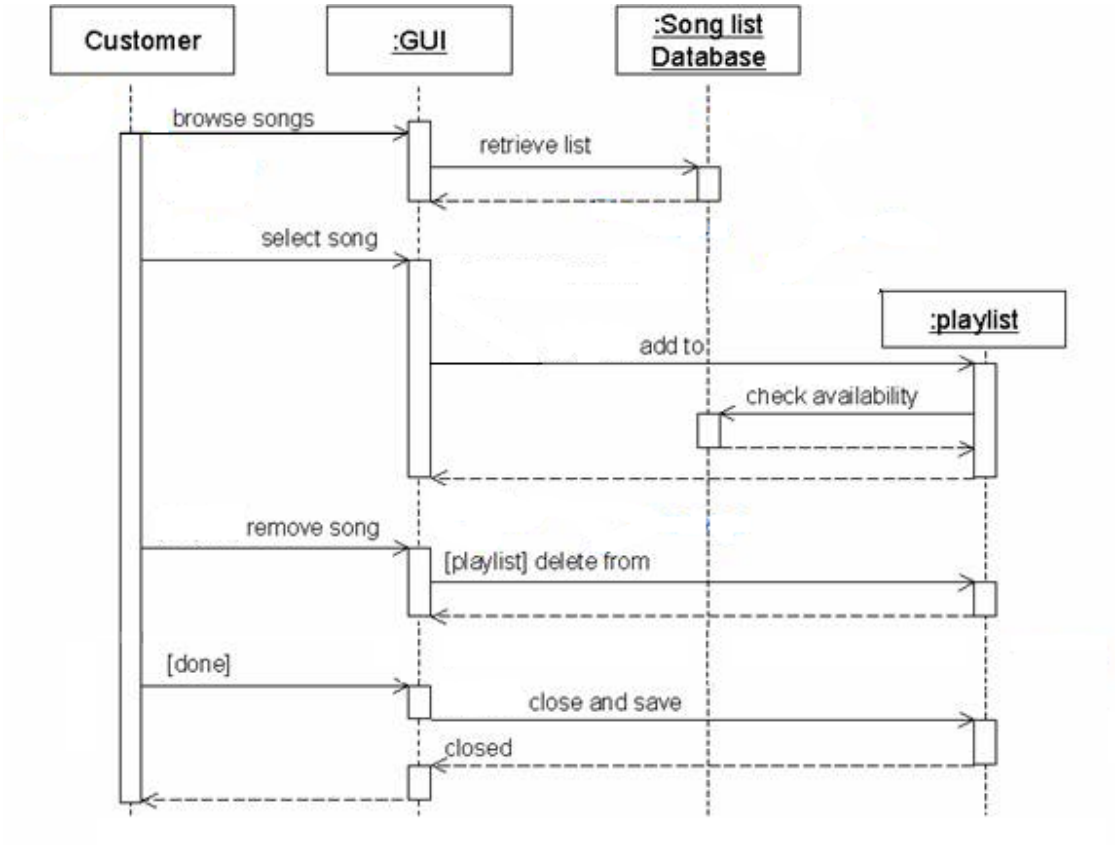


Figura 29: Exemplo de notação para um estilo Processos Comunicantes.
(fonte: [18] Fitting the UML into Your Development Process)

Este diagrama mostra os vários objectos que interagem uns com os outros. Esses objectos serão os componentes deste estilo e as mensagens serão o conector. Apresenta também a troca de mensagens entre os mesmos objectos.

3.3.7 – Vista Afectação - Estilo Implementação

Visão geral

O estilo Implementação faz o mapeamento entre os módulos da vista Módulo e a estrutura de desenvolvimento. Desta forma, este estilo é extremamente útil para a equipa de desenvolvimento e de instalação. A implementação de um módulo resulta num conjunto de ficheiros separados, por exemplo os que contêm o código fonte, ficheiros que indicam como construir os executáveis, ficheiros que resultam da execução e da compilação dos módulos entre outros. Esses ficheiros necessitam estar bem organizados para que não se perca o controlo e integridade do sistema.

Elementos, relações e propriedades

Os elementos do estilo Implementação, são directórios com fins organizacionais e ficheiros que implementam os módulos de qualquer um dos estilos da vista Módulo e que estão organizados nos directórios. A relação deste estilo é "contém" um directório contém directório(s) e ficheiros.

Elementos	Elementos de software - módulos. Ficheiros e Directórios.
Relações	A relação "contém".
Propriedades	Indica as características fornecidas pelo ambiente de desenvolvimento
Topologia	Itens de configuração Hierárquica – "está contido em"

Tabela 8: Resumo das características do estilo Implementação.

Para que serve o estilo Implementação e para que não serve

O estilo Implementação pode ser usado pela equipa de desenvolvimento, de forma a gerir, a manter e a testar os ficheiros que correspondem aos elementos de software. Este estilo pode também ser usado para especificar as diferenças entre as várias versões de um sistema.

Notações

Para representar os elementos do estilo Implementação foram usados modelos UML. No caso dos directórios foi usado o modelo "*package*" e para os ficheiros foi usado o modelo "artefacto".
A relação "contém" foi representada como um linha entre os elementos relacionados. A figura seguinte mostra a notação para a primeira apresentação do estilo Implementação.

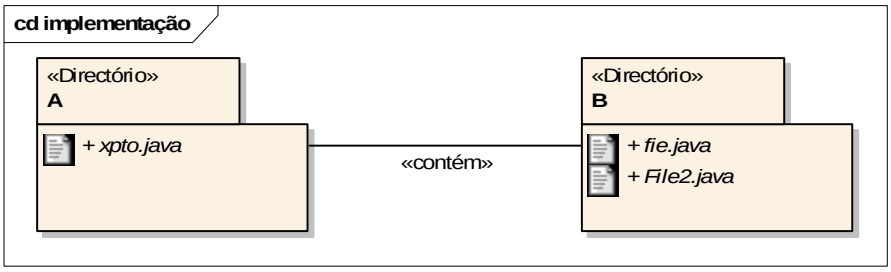


Figura 30: Notação para os elementos e relações do estilo Implementação

O directório A contém o ficheiro xpto.java e o directório B. Este último por sua vez contém os ficheiros file.java e file2.java.

Relação com os outros estilos

O estilo Implementação está relacionado com os estilos da vista Módulo, visto que este estilo faz o mapeamento dos módulos no ambiente de desenvolvimento. Ou seja, este estilo apresenta a organização dos ficheiros que contêm o código que implementa os módulos da vista Módulo.

Exemplos do estilo Implementação

Considere-se o exemplo do estilo Utilização, os módulos Micro e Vídeo estão declarados nos ficheiros Micro.h e Vídeo.h e implementados nos ficheiros Micro.cpp e Vídeo.cpp, respectivamente. Estão guardados numa directoria chamada Micro. Supondo que seriam desenvolvidos no Microsoft Visual Studio C++ existiria também o ficheiro de projecto Micro.dsp. Assim, a primeira apresentação do módulo Micro seria a seguinte:

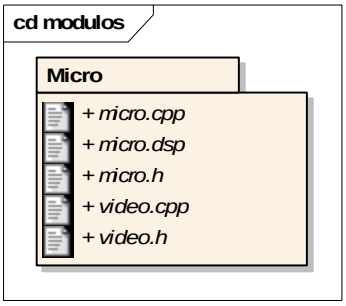


Figura 31: Exemplo da notação para o estilo Implementação
(fonte [19] – Curso de C++)

A figura 31 mostra a relação "contém" entre o Directório Micro e os ficheiros Micro.cpp, Micro.h, Micro.dsp, Vídeo.Cpp e vídeo.h.

3.3.8 – Vista Afectação - Estilo Instalação

Visão geral

O estilo Instalação descreve como é que os componentes do estilo Componente & Conector estão afectados ao hardware onde é executado o software. Descreve também as restrições que se aplicam á instalação dos mesmos componentes.

Elementos, relações e propriedades

Os elementos de software neste estilo são normalmente derivados dos elementos da vista Componente e Conector. O estilo Instalação representa como é que os elementos de software devem ser executados num computador com o suporte de um sistema operativo.

A relação típica do estilo Instalação é uma especialização da relação "*afectado-a*" que mostra as unidades físicas onde residem os elementos de software.

Elementos	Componentes e elementos de hardware tais como servidores.
Relações	A relação "*afectado-a*".
Propriedades	Os aspectos de hardware mas significativos que influenciam as decisões de afectação.
Topologia	Não existem restrições no que diz respeito á topologia.

Tabela 9: Resumo das características do estilo Instalação.

As propriedades mais importantes dos elementos, físicos e de software, do estilo Instalação são aqueles que influenciam a afectação do software nos elementos físicos. A forma como

um elemento físico satisfaz os requisitos do elemento de software são determinados pelas propriedades dos dois. Por exemplo, se um elemento de software requer um mínimo de capacidade de armazenamento, qualquer elemento físico que forneça pelo menos essa capacidade mínima de armazenamento é um candidato para uma afectação concluída com sucesso, entre esses dois elementos.

Para que serve o Estilo Instalação e para que não serve

O estilo Instalação deve ser usado essencialmente pela equipa de instalação, testes e manutenção. Este estilo é usado para análises de performance, confiabilidade e segurança. É também usado para estimação de custos.

Notações

A notação usada para este estilo é representada na figura 32. Os elementos de software são representados como componentes, os mesmos do estilo Cliente Servidor. Os elementos de hardware são representados através do modelo UML "nó", que representa os servidores que "afectam" os elementos de software.

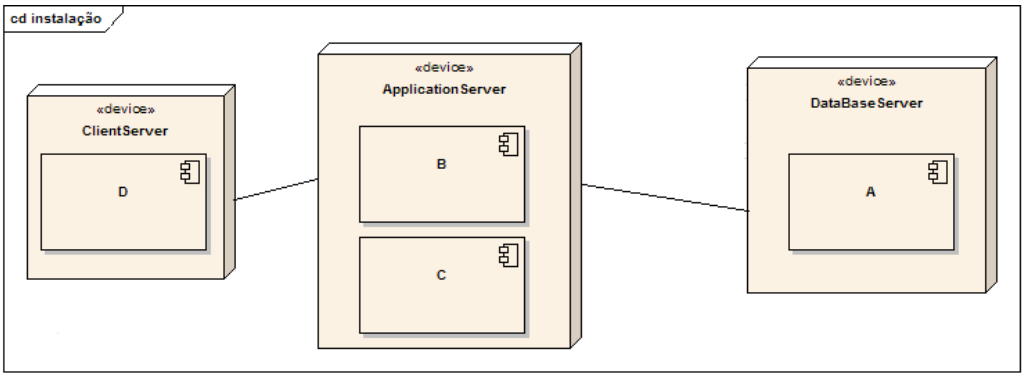


Figura 32: Notação para o estilo Instalação

Os componentes B e C estão "afectados" ao Servidor ApplicationServer. O componente A está "afectado" ao servidor DataBaseServer e o componente D está "afectado" ao componente ClientServer.

Relação com os outros estilos

O estilo Instalação está relacionado com os estilos da vista Componente e Conector. Essencialmente com o estilo Cliente Servidor, visto que este estilo mostra a afectação dos componentes do estilo Cliente Servidor, nos elementos de hardware.

Exemplos do Estilo Instalação

Neste exemplo, temos o diagrama de Instalação para um sistema de Jukeboxe. Este sistema permite aos utilizadores gerir as suas listas pessoais de músicas através do componente PlayListSupport. Essas listas são armazenadas no WebRepository.

Existem três nós, neste diagrama, um que representa um PC ao qual está afectado o componente PlayList Support, que permite aos utilizadores criar listas pessoais de música. Essas listas pessoais são guardadas no nó WebRepository.

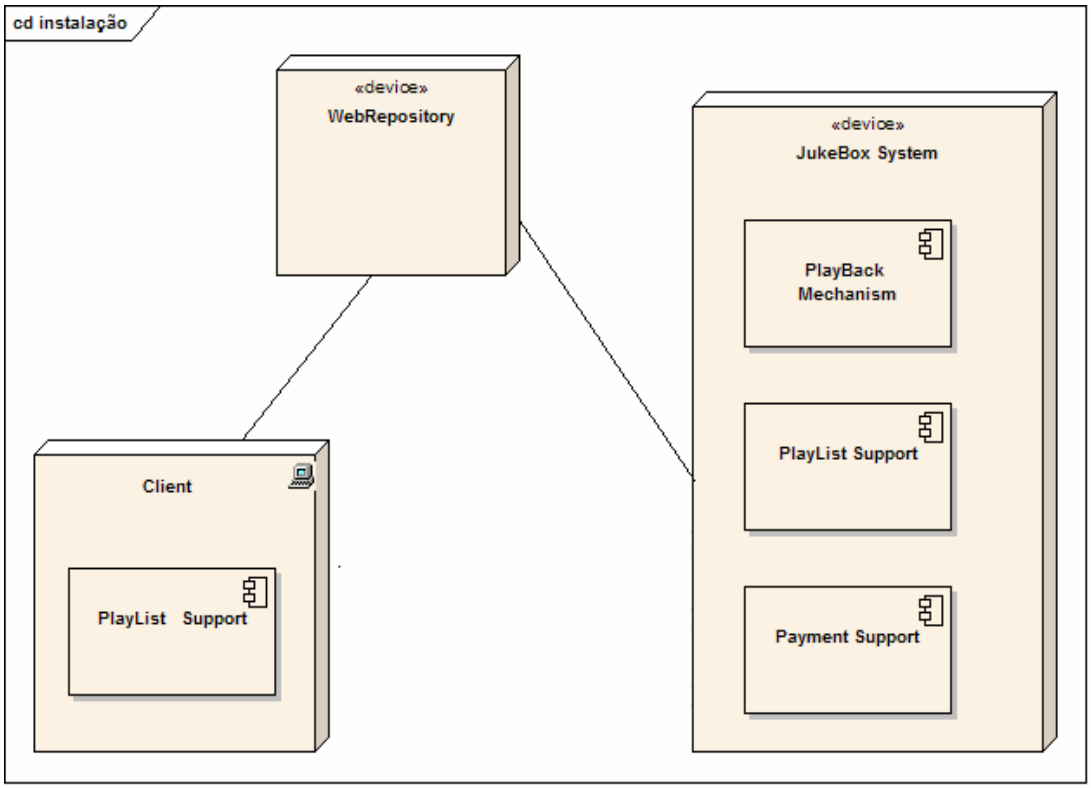


Figura 33: exemplo da notação estilo Instalação
(fonte: [18] Fitting the UML into Your Development Process)

O sistema de JukeBox, estará localizado em restaurantes e lojas, as principais funcionalidades consistem em suportar cobranças e exibir e executar as listas pessoais do Web Repository. Os componentes afectados a este nó são o PlayBack Mechanism, PlayBack Support

3.3.10 - Vista Organizacional - Estilo Casos de Utilização

Visão geral

O estilo Casos de Utilização surge da necessidade de documentar a interacção entre os componentes de execução do sistema ARQUO. Visto que esses componentes interagem para executar uma funcionalidade do sistema e que essas funcionalidades são os casos de utilização (ou de uso), é então necessário que essas funcionalidades sejam identificadas e documentadas.

Elementos, relações e propriedades

Os elementos no estilo Casos de Utilização são os Actores do sistema (que são entidades que participam no sistema através dos casos de utilização) e os casos de utilização (principiais funcionalidades do sistema). A relação entre os actores do sistema e os casos de uso é “participa”. Entre os casos de utilização existem três relações: “generalização”,

"inclusão" e "extensão". A relação "generalização" é similar á generalização entre classes da UML, o caso de utilização filho herda tanto o significado do seu pai, quanto o seu comportamento. A relação "inclusão" significa que o caso de utilização base incorpora explicitamente o comportamento de outro caso de utilização num ponto específico. A relação "extensão" significa que o caso de utilização base incorpora implicitamente o comportamento de outro caso de utilização.

Elementos	Actores e Casos de Utilização
Relação	As relações "participa", "generalização", "inclusão" e "extensão"

Tabela 10: Resumo das características do estilo Casos de Utilização.

Para que serve o estilo Casos de Utilização e para que não serve

O estilo Casos de Utilização aqui definido é útil para a equipa de implementação poder identificar as funcionalidades do sistema e assim implementá-las. È útil também para os novos stakeholders perceberem o sistema.

Notações

A notação usada para o estilo Casos de Utilização foi o diagrama de Casos de Uso da UML. Os diagramas de Casos de Uso mostram um conjunto de casos de utilização, os actores do sistema e os seus relacionamentos. A notação para as relações é também a definida pela UML nos diagramas de Casos de Uso.

O comportamento dos elementos, neste estilo, será apresentado através da descrição dos cenários e dos actores de cada caso de utilização

Relação com os outros estilos

Este estilo está relacionado com o estilo Processos Comunicantes na medida em que este descreve uma funcionalidade do sistema através da representação da forma como os objectos do sistema interagem entre si, ao longo do tempo.

Exemplos do Estilo Casos de Utilização

Imagine-se, a título de exemplo, um sistema de vendas, onde os actores principais serão os clientes e os operadores. Estes actores participam nos casos de utilização seguintes: Comprar Produto, Efectuar Pagamento, Efectuar Troca e Devolver Produto.

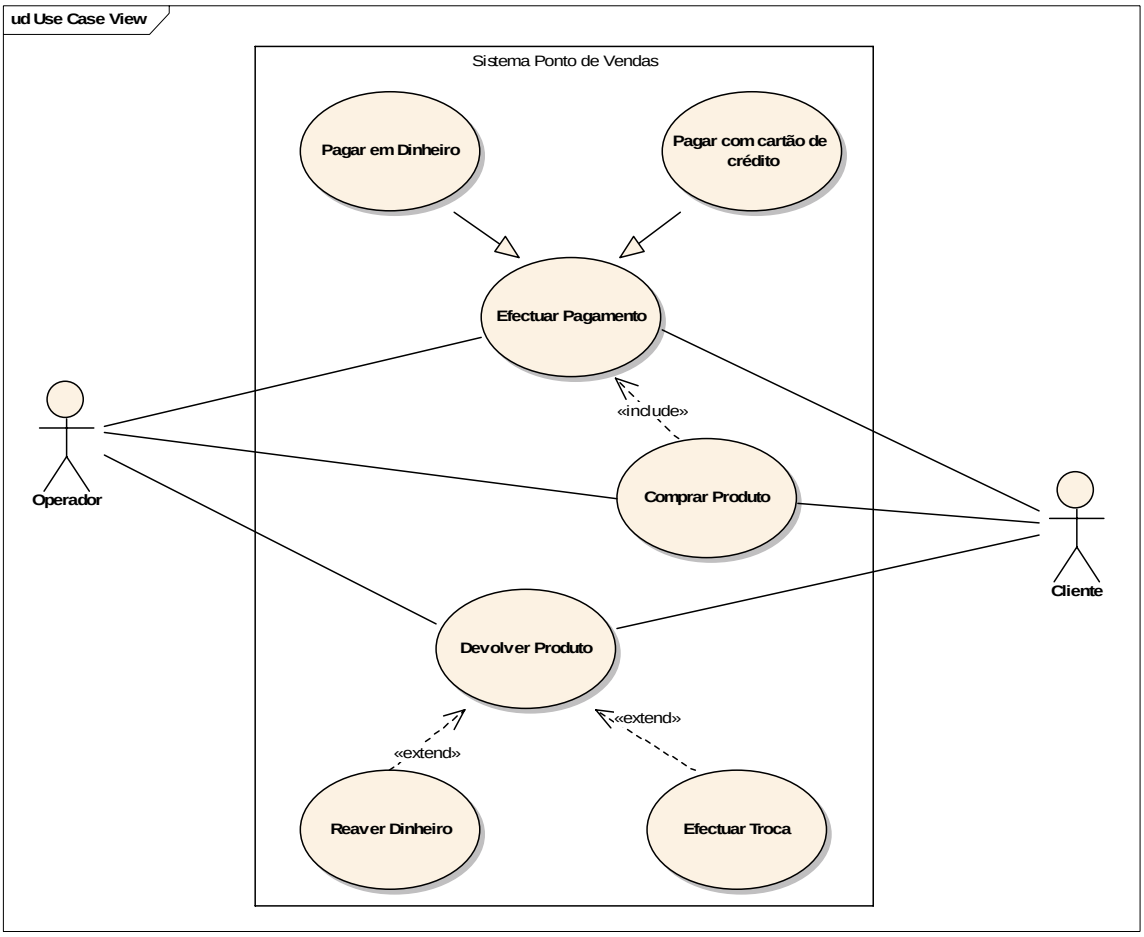


Figura 34: Exemplo do estilo Casos de Utilização
(exemplo adaptado da referência nº 20)

O caso de utilização Efectuar Pagamento é uma generalização dos casos de uso Pagar em Dinheiro e Pagar com Cartão de Crédito. Ou seja o pagamento de um produto pode ser efectuado de duas formas ou a dinheiro ou com cartão de crédito, sendo assim os casos de uso Pagar em Dinheiro e Pagar com Cartão de Crédito são uma especialização do caso de uso Efectuar Pagamento. A realização da compra de um produto inclui efectuar pagamento. Devolução de um produto, pode lavar a dois comportamentos distintos, ou é efectuada a troca por outro produto, ou em vez disso o cliente pode optar por reaver o dinheiro.

4. Apresentação da solução - Caso Pratico sistema ARQUO

Este capítulo apresenta alguns excertos, daquele que foi o resultado final da aplicação dos conceitos e metodologias atrás definidos, na documentação da arquitectura do sistema ARQUO.

4.1 - O pacote de documentação

O pacote de documentação do sistema ARQUO é composto por dois volumes. O volume I que contém a informação arquitectural do sistema ARQUO que se aplica a mais do que uma vista. Este volume inclui o guia de documentação, o *template* utilizado para representar cada uma das vistas, uma visão geral sobre o sistema ARQUO, o mapeamento entre as várias vistas, um directório com a lista dos elementos arquitecturais do sistema, um glossário com os termos e abreviaturas mais comuns e uma apresentação dos "porquês" que estão por detrás das decisões mais importante sobre o desenho do sistema em estudo. O volume II contém as vistas arquitecturais que documentam o sistema ARQUO, são elas a vista Módulo, a vista Componente e Conector, a vista Afectação e a vista Organizacional.

Por questões de confidencialidade da informação contida neste pacote de documentação, que se prendem com o facto do sistema ARQUO ser propriedade intelectual da empresa que o desenvolveu - a TIE, neste relatório serão apresentados apenas alguns excertos do pacote de documentação. O volume I será apresentado quase integralmente com excepções para o directório, devido á extensão dos dados contidos no mesmo (o directório contém 1262 elementos) e para o Glossário, pelos mesmos motivos.
O volume II porque apresenta o sistema ARQUO segundo as várias vistas arquitecturais implica uma maior confidencialidade. Sendo assim dos 32 módulos principais do sistema ARQUO, foi seleccionado um desses módulos – o AutoFileServer - cuja documentação abrange todos os estilos usados neste pacote de documentação, permitindo assim, mostrar pelo menos um exemplo de cada um desses estilos. Ainda pelas mesmas questões de confidencialidade, e devido ao teor público do conteúdo deste relatório, a apresentação do volume II será feita no CD que acompanha este relatório.

4.1.1 - Volume I - Documentação para Além das Vistas da Arquitectura de Software ARQUO

O volume I contém a informação arquitectural do sistema ARQUO que se aplica a mais do que uma vista.

4.1.1.1 - Capítulo I1 - Guia de Documentação

Descrição do pacote de documentação da Arquitectura de Software ARQUO

Este capítulo descreve a estrutura do pacote de documentação da arquitectura de software do sistema ARQUO. Este pacote de documentação está organizado em dois volumes. O Volume I contém a informação que se aplica a mais do que uma vista, onde está incluído este guia de documentação. O Volume II apresenta as vistas arquitecturais do sistema ARQUO.

O Volume I - a documentação para além das vistas da Arquitectura de Software ARQUO - contém os seguintes capítulos:

Capítulo I1 - Guia de Documentação: o Guia de Documentação é a introdução da informação sobre o sistema ARQUO, seleccionada para incluir na documentação. Este guia consiste em duas secções. A primeira secção consiste na descrição das partes - uma breve descrição de cada parte do pacote de documentação, a parte mais importante do guia é a descrição das vistas incluídas no pacote de documentação. O guia prossegue com a descrição sobre como é que os vários stakeholders devem aceder ao pacote de documentação de forma a alcançar os seus propósitos.

Capítulo I2 - Template da Vista: O Template de Vista corresponde à organização padrão de uma vista, ou seja, apresenta a estrutura padrão do documento de um pacote de vista. O objectivo do Template de Vista consiste em ajudar o leitor a navegar rapidamente para uma secção de interesse, e ajudar o escritor a organizar a informação e a estabelecer um critério para saber o andamento do trabalho.

Capítulo I3 - Visão geral do sistema: É uma pequena descrição da função do sistema ARQUO, dos seus utilizadores, e toda a informação de background e restrições considerada importante. A Visão Geral do Sistema não é uma parte da arquitectura e não contem informação arquitectural, no entanto, é indispensável para compreender a arquitectura.

Capítulo I4 - Mapeamento entre vistas: Porque todas as vistas de uma arquitectura descrevem o mesmo sistema, existem algumas vistas que têm muito em comum. O objectivo do Mapeamento entre vistas é ajudar um leitor ou um outro consumidor da documentação a perceber a relação entre as vistas, permitindo assim a esse leitor ter uma visão da arquitectura como um todo.

Capítulo I5 – Directório: O Directório é um índice de todos os elementos, relações e propriedades que aparecem nas vistas, juntamente com um apontador para a sua definição.

Capítulo_I6 - Glossário e Lista de Acrónimos: O Glossário e a Lista de Acrónimos definem os termos usados na documentação da arquitectura, que têm um significado especial.

Capítulo_I7 - Porquê que a arquitectura é como é: Esta secção documenta uma fundamentação cruzada; isto é, documenta o raciocínio por detrás das decisões que se aplicam a mais do que uma vista. Inclui a documentação do background e restrições organizacionais que levaram a decisões importantes do sistema.

O Volume II, as Vistas da Arquitectura de Software ARQUO, contém os seguintes capítulos:

Capítulo II1 – Estilo Decomposição da Vista Módulo: este estilo mostra como as responsabilidades do sistema estão repartidas ao longo dos módulos e como estes módulos são decompostos em sub-módulos. O estilo Decomposição mostra como o sistema ARQUO é decomposto em unidades de implementação e, simultaneamente, como é que as funcionalidades do sistema estão afectadas a essas unidades de implementação.

Capítulo II2 - Estilo Generalização da Vista Módulo: este estilo relaciona os módulos mostrando como um é a generalização ou especialização do outro.

Capítulo II3 - Estilo Utilização da Vista Módulo: o estilo utilização mostra como os módulos estão relacionados uns com os outros através da relação "usa", uma especialização da relação "depende-de". Este estilo informa a equipa de desenvolvimento que outros módulos devem existir para que a sua porção do sistema funcione.

Capítulo II4 - Estilo Dependência da Vista Módulo: este estilo surge através da combinação híbrida dos estilos Utilização e Agregação da vista módulo.

Capítulo II5 - Estilo Cliente Servidor da Vista Componente e Conector: o estilo Cliente-Servidor mostra como os componentes interagem através da requisição e prestação de serviços a outros componentes.

Capítulo II6 - Estilo Processos Comunicantes da Vista Componente e Conector: o estilo Processos Comunicantes representa o sistema como um conjunto de unidades de execução juntamente com as suas interacções.

Capítulo II7 - Estilo Instalação da Vista Afectação: o estilo Instalação mostra como os processos e outros elementos de software estão afectados ás plataformas de execução, ás unidades físicas de armazenamento, á transmissão ou ao processamento de dados.

Capítulo_II8 - Estilo Implementação da Vista Afectação: a vista de Implementação mostra como as unidades de código ou módulos estão organizados em ficheiros e directórios.

Capítulo_II9 - Estilo Casos de Utilização da vista Organizacional: esta vista apresenta os principais casos de utilização do sistema, e serve como base para o estilo Processos Comunicantes da vista Componente e Conector.

Como é que os stakeholders podem usar a documentação

Esta secção apresenta os stakeholders do sistema ARQUO com mais interesse neste pacote de documentação, e mostra também a forma como estes devem usar o pacote de documentação de forma a atingir os seus interesses:

O Gestor do Projecto: de forma a ajudar no planeamento do projecto, dar especial atenção ao estilo Decomposição (Capítulo II1), em especial aos níveis mais altos de decomposição, de forma a definir a organização e atribuição do trabalho pela equipa de trabalho.

A Equipa de Manutenção: Ler o pacote de vista instalação (Capítulo_II7) de forma a perceber como é que as unidades de software estão afectadas ao hardware. Ler o pacote de vista Implementação (Capítulo_II8) para perceber como é que as unidades de implementação estão afectadas e organizadas no ambiente de desenvolvimento.

A equipa de desenvolvimento: Ler o estilo Decomposição (Capítulo_II1) para perceber as unidades de implementação do sistema, o estilo Utilização (Capitulo_II3) de forma a perceber a dependência entre os módulos, o estilo Implementação (Capítulo_II8) para saber onde é que o software reside no ambiente de desenvolvimento. Dar especial atenção ao *rationale* em cada vista e ás interfaces. Ler o estilo Casos de Utilização (Capitulo II9) para perceber quais as principais funcionalidades do sistema. Ler o estilo Cliente Servidor (Capítulo_II5) e Processos Comunicantes (Capítulo_II6), de forma a saber como é que os componentes de execução se relacionam através da requisição e prestação de serviços e como é que estes interagem entre si.

Alguém novo no projecto: ler o Guia de Documentação (Capítulo_I1) de forma a compreender a organização do pacote de documentação e o Template de Vista (Capítulo_I2), para perceber como é que as vistas estão documentadas. Ler a Visão Geral do Sistema (Capítulo_I3) . Examinar o estilo Decomposição (Capítulo_II1) de mais alto nível e o estilo Instalação (Capítulo_II7). Usando o Mapeamento entre Vistas (Capítulo_I4), ler as partes relevantes das outras vistas que revelem como é que as suas unidades são instaladas no hardware ou como é que se manifestam enquanto componentes de execução. Estes são os Capítulos que todos os novos stakeholders devem ler para conhecerem o sistema, no entanto, depois consoante a sua área de trabalho devem ler então a documentação que mais se adequa a essa área.

4.1.1.2 - Capítulo_I2 - Template da Vista

Este capítulo descreve a organização da documentação padrão de um pacote de vista a qual é usada no volume II para descrição do pacote de documentação do sistema ARQUO. A documentação é feita em sete partes:

1 - Primeira Apresentação: A primeira apresentação mostra os elementos e as relações entre eles que povoam esta porção de vista apresentada neste pacote de vista. Esta primeira apresentação deve conter a informação sobre o sistema que queremos transmitir primeiro. Deve conter os elementos primários e as suas relações mas devido a certas circunstâncias não os deve incluir a todos.
Esta primeira apresentação é usualmente gráfica, neste caso deve conter uma chave que explique e descreva cada símbolo utilizado. Mas por vezes, a primeira apresentação é textual e deve também apresentar um sumário que descreva a informação mais importante no pacote de vista.

2 - Catálogo de Elementos: O catálogo de elementos detalha a informação sobre os elementos descritos na primeira apresentação. Se na primeira apresentação foram omitidos elementos ou relações relevantes ao pacote de vista em questão, então estes devem ser introduzidos e explicados no catálogo. O catalogo é composto por:

a) Os elementos e as suas propriedades: esta secção apresenta cada elemento do pacote de vista e lista as propriedades; desse elemento.
b) As relações e as suas propriedades: cada vista tem tipos de relações específicas que são descritas entre os elementos dessa vista;
c) As interfaces dos elementos: uma interface é uma fronteira através da qual os elementos interagem e comunicam uns com os outros. Esta secção documenta as interfaces dos elementos;

d) O comportamento dos elementos: alguns elementos têm interacções complexas com o seu ambiente. Para fins de compreensão ou análise, é necessário que o arquitecto especifique o comportamento dos elementos;

3 - Diagrama de Contexto: que mostra como o sistema ou parte dele descrito no pacote de vista em questão, se relaciona com o seu ambiente.

4 - Um guia de variabilidade: que mostra como lidar com os pontos de variação que sejam parte da arquitectura descrita no referente pacote de vista.

5 - O background da arquitectura: explica como é que o desenho reflectido no pacote de vista é. O objectivo desta secção consiste em explicar porque é que o desenho é como é e fornecer um argumento convincente de que é a melhor opção. O background da arquitectura inclui:

a) Rationale. O arquitecto explica porque foram tomadas as decisões referentes ao desenho reflectido no pacote de vista e apresenta uma lista de alternativas rejeitadas e o motivo da rejeição.
b. Resultados da análise. O arquitecto deve documentar os resultados das análises que foram feitas, (por exemplo os resultados da análise de desempenho ou segurança) bem como a lista do que deve ser mudado em caso de haver um certo tipo de modificação no sistema.
c. Assumpções. O arquitecto deve documentar qualquer assumpção que faça na criação do desenho. As assumpções sobre o ambiente documentam o que o arquitecto assume ser válido no ambiente e o que pode ser utilizado pelo sistema que está a ser desenhado. Podem ser feitas também assumpções sobre invariantes do ambiente. Por fim, as assumpções sobre o ambiente podem pertencer ao ambiente de desenvolvimento, por exemplo os níveis de perícia da equipa de implementação ou o conjunto de ferramentas disponíveis. As assumpções assumem um papel crucial na validação de uma arquitectura. O desenho produzido por uma arquitectura é feito em função destas assumpções. E documentá-las explicitamente torna muito mais fácil revê-las em termos de exactidão e solidez do que tentar descobri-las examinando o desenho.

6 - Outra informação fornecida: varia de acordo com as práticas padrão da organização ou das necessidades de um projecto em particular. É conveniente guardar a informação não arquitectural "perto" da arquitectura, esse é o propósito desta secção.

7 - Pacotes de vista relacionados: Esta secção fornece um apontador para o pacote de vista pai, irmão, filho ou outro (caso existam).

4.1.1.3 - Capítulo_I3 - Visão geral do sistema

O ARQUO é o resultado dos esforços de cinco anos de investigação e desenvolvimento da TIE – Tecnologias de Integração Empresarial.
É um sistema de arquivo digital de conteúdos de elevado desempenho e fiabilidade, especialmente vocacionado para lidar com os documentos essenciais ao negócio, mantendo-os sempre disponíveis, onde quer que o utilizador esteja.

O Arquo tem uma utilidade abrangente, no que diz respeito ao formato dos objectos arquivados, que vai desde os cheques de um banco, contratos duma seguradora, processos clínicos de um hospital ou conteúdos numa mediateca. O ARQUO permite guardar, de forma segura e persistente, toda a informação resultante da actividade de uma organização.
O acesso aos conteúdos é feito via web browser, através de uma interface simples e intuitiva, especialmente desenhada para proporcionar aos utilizadores um nível elevado de produtividade desde o primeiro instante.
O ARQUO foi desenhado para crescer com o negócio. A sua arquitectura distribuída permite escalar de forma transparente de soluções mono-servidor para soluções multi-servidor, bem como suportar arquivos de milhões de documentos, cumprindo os mais exigentes requisitos de desempenho e disponibilidade.

As principais funcionalidades do sistema ARQUO são:

Consulta Remota – feita através de uma interface web e permite pesquisa simultânea em múltiplos tipos de documentos**.**

Arquivo de imagens – O arquivo de imagens permite a captura de imagens em lote (através de scanning) e o controlo de qualidade das mesmas. A captura pode ser centralizada ou distribuída. A cada uma dessas imagens é-lhe associado um ou mais índices (informação textual associada á imagem) de forma a ser possível posteriormente efectuar pesquisas ao arquivo, e a reaver ou aceder as imagens arquivadas. A importação de ficheiros pode ser desencadeada de forma automática ou manualmente.

Gestão do arquivo off-line – responsável pela monitorização e satisfação de pedidos pendentes.

Gestão do sistema – permite a definição de novos tipos de documento, a definição de segurança e controlo de acessos, contabilização de operações e o registo de informação para auditoria.

Exportação de dados (reporting) – permite a segmentação automática por ano e mês, a criação automática de bases de dados e tabelas, a possibilidade de agregação de dados e a exportação de documentos e dados estatísticos em XML

Os utilizadores do sistema ARQUO são:

Administradores do Sistema: são os responsáveis pela configuração, facturação, monitorização e manutenção do sistema;
Operadores de arquivo: são os responsáveis pelos processos de alimentação do histórico – processamento de ficheiros e processamento de documentos diversos, não tratados pelos centros de captura e leitura;
Operadores de consulta: consultam os documentos arquivados no sistema;
Operadores Off-line: são responsáveis pela catalogação e disponibilização de unidades off-line.

4.1.1.4 - Capítulo_I4 - Mapeamento entre vistas

Porque todas as vistas da arquitectura descrevem o mesmo sistema, existem algumas vistas que têm muito em comum.
De seguida é apresentado o mapeamento entre os estilos que documentam o sistema ARQUO.

Mapeamento entre o estilo Decomposição da vista Módulo e o estilo Cliente-Servidor da vista Componente e Conector

O estilo Decomposição está relacionado com o estilo Cliente-Servidor. Visto que os módulos (unidades de implementação) do estilo Decomposição implementam os componentes e conectores do estilo Cliente-Servidor.
A tabela seguinte, mostra essa relação.

Estilo Decomposição	**Estilo Cliente-Servidor**
ArchiveApp	ArchiveApp_ClienteServidor
ARQUOManager	ARQUOManager_ClienteServidor
ARQUOMonitor	ARQUOMonitor_ClienteServidor
AsyncQueryServer	AsyncQueryServer_ClienteServidor
AutoFileServer	AutoFileServer_ClienteServidor
ConfigApp	ConfigApp_ClienteServidor
ConfigurationServer	ConfigServer_ClienteServidor
DocImportServer	DocImportServer_ClienteServidor
DocumentQueryServer	DocumentQueryServer_ClienteServidor
eARQUO	eARQUO_ClienteServidor
ExportApp	ExportApp_ClienteServidor
FileServer	FileServer_ClienteServidor
MonitorInfoServer	MonitorInfoServer_ClienteServidor
ObjectQueryServer	ObjectQueryServer_ClienteServidor
OfflineServer	OfflineServer_ClienteServidor
SecurityServer	SecurityServer_ClienteServidor
UploadApp	UploadApp_ClienteServidor
ValidationServer	ValidationServer_ClienteServidor
VolumeApp	VolumeApp_ClienteServidor
WebInterface	WebInterface_ClienteServidor
WebServicesInterface	WebServicesInterface_ClienteServidor

Mapeamento entre a estilo Decomposição da vista Módulo e o estilo Implementação da vista Afectação

Existe uma ligação entre o estilo Decomposição e o estilo Implementação. Visto que o estilo Implementação documenta a forma como os módulos do estilo Decomposição estão mapeados no que diz respeito a ficheiros ou directorias.
A tabela seguinte, mostra essa relação:

Estilo Implementação	Estilo Decomposição
ArchiveApp	ArchiveApp_Implementacao
ARQUO	ARQUO_Implementacao
ARQUOAgent	ARQUOAgent_Implementacao
ARQUOManager	ARQUOManager_Implementacao
ARQUOMonitor	ARQUOMonitor_Implementacao
AsyncQueryServer	AsyncQueryServer_Implementacao
AutoFileServer	AutoFileServer_Implementacao
ConfigApp	ConfigApp_Implementacao
ConfigurationServer	ConfigServer_Implementacao
DocImportInterceptor	DocImportInterceptor_Implementacao
DocImportServer	DocImportServer_Implementacao
DocumentQueryServer	DocumentQueryServer_Implementacao
eARQUO	eARQUO_Implementacao
ExportApp	ExportApp_Implementacao
FileServer	FileServer_Implementacao
Master	Master_Implementacao
MonitorInfoServer	MonitorInfoServer_Implementacao
ObjectQueryServer	ObjectQueryServer_Implementacao
OfflineServer	OfflineServer_Implementacao
SecurityServer	SecurityServer_Implementacao
StagingServer	StagingServer_Implementacao
Tester	Tester_Implementacao
UploadApp	UploadApp_Implementacao
UtilApps	UtilApps_Implementacao
ValidationServer	ValidationServer_Implementacao
VolumeApp	VolumeApp_Implementacao
WebInterface	WebInterface_Implementacao
WebServicesInterface	WebServicesInterface_Implementacao
WebServicesPrototype	WebServicesPrototype_Implementacao

Mapeamento entre o estilo Instalação da vista Afectação e o estilo Cliente-Servidor da vista Componente e Conector

Existe um mapeamento entre o estilo Instalação e o estilo Cliente-Servidor, porque o estilo Instalação mostra como os componentes do estilo Cliente-Servidor estão afectados á plataforma de hardware.

Vista Instalação	Vista Cliente-Servidor
ARQUO_Instalacao	ArchiveApp_ClienteServidor
	ARQUOManager_ClienteServidor
	ARQUOMonitor_ClienteServidor
	AsyncQueryServer_ClienteServidor
	AutoFileServer_ClienteServidor
	ConfigApp_ClienteServidor
	ConfigServer_ClienteServidor
	DocImportServer_ClienteServidor
	DocumentQueryServer_ClienteServidor
	eARQUO_ClienteServidor
	ExportApp_ClienteServidor
	FileServer_ClienteServidor
	MonitorInfoServer_ClienteServidor
	ObjectQueryServer_ClienteServidor
	OfflineServer_ClienteServidor
	SecurityServer_ClienteServidor
	UploadApp_ClienteServidor
	ValidationServer_ClienteServidor
	VolumeApp_ClienteServidor
	WebInterface_ClienteServidor
	TIESecurity_ClienteServidor

Mapeamento entre o estilo Casos de Utilização e o estilo Processos Comunicantes

O mapeamento entre o estilo Casos de Utilização e o estilo Processos Comunicantes deve-se ao facto do estilo Casos de Utilização ser a base para o estilo Processos Comunicantes.

Estilo Casos de Utilização	Estilo Processos-Comunicantes
ARQUO_CasosDeUso	Login
	Acesso
	Consultar-Objectos
	Importar-Objectos
	Logoff

4.1.1.5 - Capítulo_I5 – Directório

Este capítulo consiste num índice de todos os elementos que aparecem nas vistas, juntamente com a especificação do tipo de elemento (por exemplo: módulo, componente, conector) apontador para a sua definição. Por motivos de confidencialidade expostos anteriormente, e por questões de espaço é apenas apresentado um pequeno excerto do índice dos elementos, a título de amostra.

Elemento	Tipo	Apontador para o pacote de vista correspondente
3Party	Elemento do Módulo ARQUO	ARQUO_Utilizacao
ActionCopy	Elemento do Módulo ARQUOManager	ARQUOManager_jsp
ActionEdit	Elemento do Módulo ARQUOManager	ARQUOManager_jsp
ActionEdit_121	Elemento do Módulo ARQUOManager	ARQUOManager_jsp
ActionEdit_2	Elemento do Módulo ARQUOManager	ARQUOManager_jsp
ActionEntities	Elemento do Módulo ARQUOManager	ARQUOManager_jsp
ActionEntity	Elemento do Módulo ARQUOManager	ARQUOManager_jsp

4.1.1.6 - Capítulo_I6- Glossário e Lista de Acrónimos

O Glossário e a Lista de Acrónimos definem os termos usados na documentação da arquitectura, que têm um significado especial.

Glossário

Arquivo óptico digital - repositório de documentos, cujo formato dos documentos é digital e foi na maioria dos casos resultante de captura óptica, foi transformado em formato digital através dum digitalizador, vulgarmente conhecido como scanner. Além dos documentos o arquivo óptico digital também armazena a informação de onde são mantidos esses documentos (ou melhor, a informação de como chegar a estes documentos).

Índice - informação de indexação dos objectos e que permitirá mais tarde recuperá-los, para além de possibilitar a execução de pesquisas de várias ordens.

Documento ou Objecto – Um documento ou objecto é uma instância de um tipo de documento. A informação mantida no sistema relativa a um documento pode incluir diferentes versões dos dados textuais (registo) e diferentes imagens.

Captura Óptica - Transformação dum documento físico no seu formato digital com o auxílio dum digitalizador (scanner).

Controlo de Qualidade - Controlo efectuado por um operador humano para avaliar a qualidade das imagens armazenadas e a soma de controlo dos contentores onde as imagens estão armazenadas.

(…)

Lista de Acrónimos

ARQUO – Sistema de Arquivo Óptico
CORBA - Common Object Request Broker Architecture
(…)

4.1.1.7 - Capítulo_I7 - Porquê que a arquitectura é como é?

O ARQUO é um sistema de arquivo óptico digital. Adoptado por várias empresas portuguesas e não só. Essas empresas são na sua maioria Instituições Bancárias e Companhias de Seguros, sendo assim os critérios de segurança, desempenho, armazenamento e outros assumem um papel muito importante no desenvolvimento do sistema. A forma com a arquitectura responde a esses critérios é a seguinte:

- **Desempenho** – a plataforma permite a vários processos correr em simultâneo para eliminar engarrafamentos e reduzir tempos de espera, e apresenta também capacidade de gerar contentores de imagem.

- **Fiabilidade –** a plataforma de arquivo não permite apagar documentos do arquivo, mantendo assim a fiabilidade dos objectos arquivados.

- **Escalabilidade -** uma solução desenhada para crescer com as necessidades de cada cliente, permitindo incorporar novos tipos de documentos, incluir novos utilizadores sempre com elevado desempenho.

- **Flexibilidade –** o ARQUO é uma plataforma aberta que permite a configuração de novos tipos de documentos, utilizadores, grupos de utilizadores e perfis de utilização através da disponibilização de ferramentas para o efeito.

- **Disponibilidade –** arquitectura distribuída que em caso de falha consegue sempre assegurar os serviços, este tipo de arquitectura serve também de suporte a outras características tais como, escalabilidade e fiabilidade.

- **Facilidade de Integração -** o ARQUO é composto por um conjunto de componentes que se articulam segundo uma filosofia de Arquitectura Orientada a Serviços (SOA) com interfaces para o exterior bem definidos de modo a facilitar a sua integração com outros sistemas.

- **Acessibilidade –** o módulo de consulta é web, permitindo assim que os objectos arquivados estejam sempre disponíveis.

- **Qualidade** - uma solução que dispõe de controlos de qualidade automáticos, supervisionando as diferentes actividades do processo, assegurando a qualidade das imagens e a correcta validação dos dados.

- **Segurança -** controlo de acessos ao sistema, permissões, identificação de utilizadores, assegurando a autenticidade dos dados e imagens. Disponibilização de mecanismos de registo de todas as operações efectuadas no sistema.

- **Universalidade -** a plataforma de captura permite a aquisição de imagens e dados a partir dos mais diversos dispositivos: scanners, leitores de cheques, servidores de fax, correio electrónico, formulários, etc.

4.1.2 - Volume II - Vistas da Arquitectura de Software ARQUO

O volume II contém as vistas arquitecturais que documentam o sistema ARQUO. O volume II é composto pelos seguintes capítulos:

Capítulo II1 - Vista Módulo - Estilo Decomposição
Capítulo II2 - Vista Módulo - Estilo Generalização
Capítulo II3 - Vista Módulo - Estio Utilização
Capítulo II4 - Vista Módulo - Estilo Dependência
Capítulo II5 - Vista Componente e Conector - Estilo Cliente Servidor
Capítulo_II6 - Vista Componente e Conector - Estilo Processos Comunicantes
Capítulo II7 - Vista Afectação - Estilo Instalação
Capítulo II8 - Vista Afectação - Estilo Implementação
Capítulo II9 - Vista Organizacional – Estilo Casos de Utilização

Pelas questões de confidencialidade, referidas anteriormente, da informação que o volume II descreve, o mesmo é apresentado no CD que acompanha o relatório.

Capitulo 5 – Estudo sobre a realidade das organizações no que diz respeito a Documentação do Software

Com vista a uma análise sobre a forma como a documentação do software é vista e como é posta em prática em várias organizações, foi feito um inquérito a 8 organizações regionais e não só. A amostra apesar de ser pequena é significativa na medida em que cobre Organizações com abordagens diferentes nos seus processos de desenvolvimento e manutenção do software, são elas três empresas de engenharia de software – TIE, In-formar e Expedita; uma empresa de engenharia de software de gestão – Infos; um centro de investigação e Tecnologia - Madeira Tecnopolo; dois departamentos de informática - Sector de Comunicações e Informática da UMA e a Direcção Regional de Informática; e uma empresa de energia - Empresa de Electricidade da Madeira.

O anexo E apresenta o inquérito que foi enviado às organizações atrás referidas. As respostas a este inquérito podem ser consultadas no ficheiro inquérito.pdf, contido no CD que acompanha este relatório.

5.1 - Apresentação dos resultados

Com base nos resultados do inquérito, em particular á questão número 1, constata-se que 71% das empresas contêm um pacote de documentação técnica, como mostra o gráfico 1. No entanto, em análise as respostas da questão número 2.1 (ver gráfico 2), pode-se constatar que apenas uma das organizações segue uma metodologia em particular.

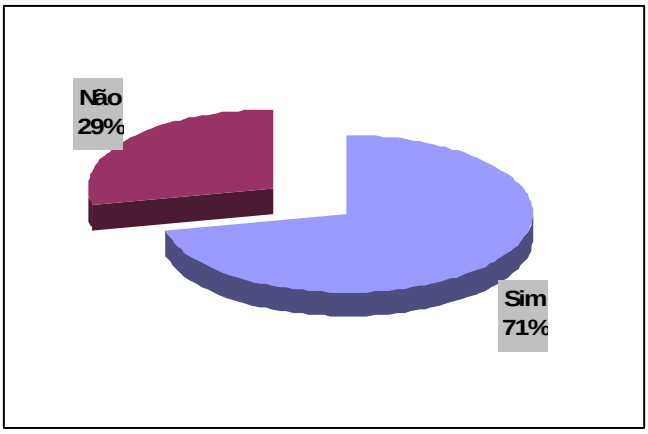


Gráfico 1 : Resultado das respostas da questão número 1 do inquérito

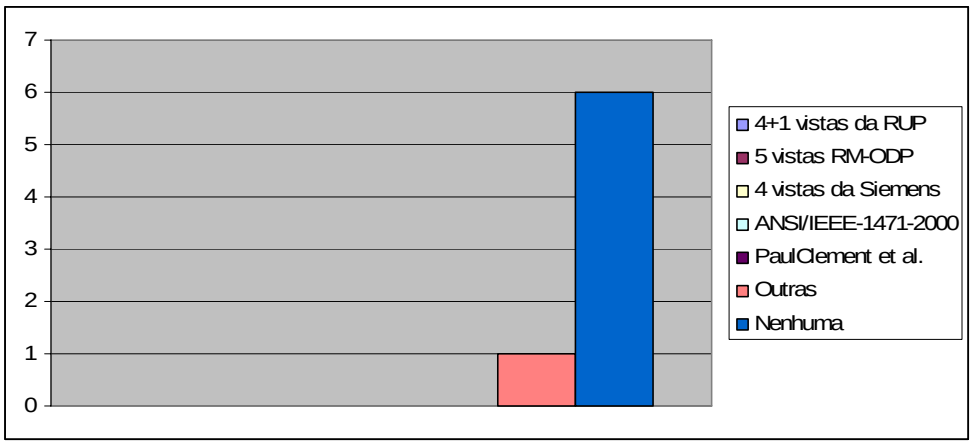


Gráfico 2: Resultado das respostas da questão número 2.1 do inquérito

O gráfico número 3, sintetiza as respostas á questão número 2.2 do inquérito, sobre o formato dos ficheiros que compõem o pacote de documentação e que são na sua maioria o formato Word e html.

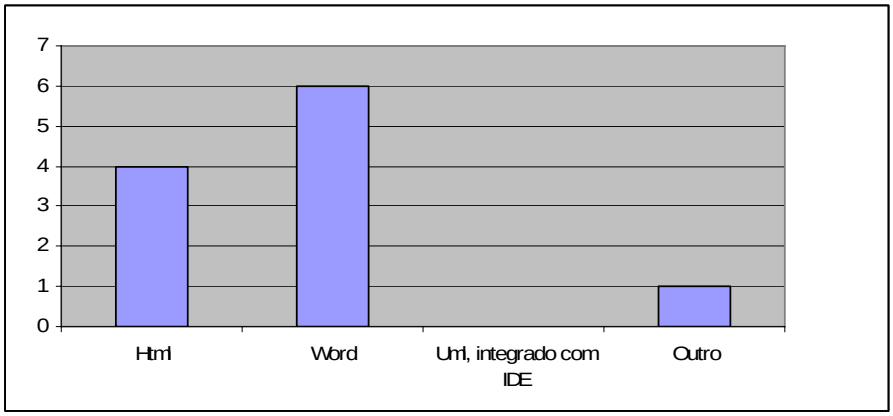


Gráfico 3: Formato dos ficheiros que compõem o pacote de documentação

A frequência com a qual o pacote de documentação é consultado, é um factor determinante para a importância do mesmo. No caso das empresas em análise o pacote de documentação é consultado com muita frequência, como mostra o gráfico seguinte.

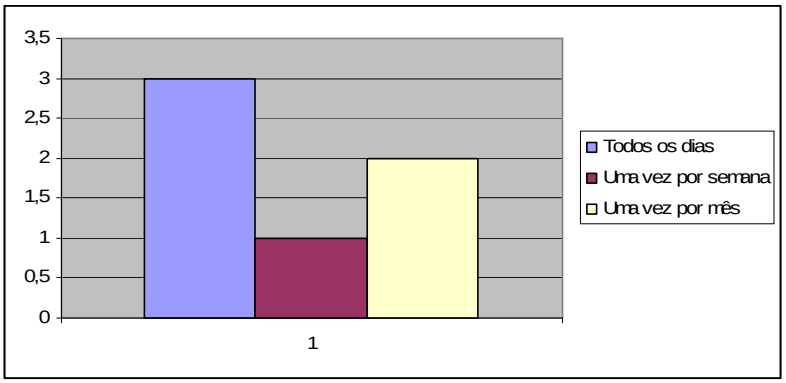


Gráfico 4: Frequência com a qual o pacote de documentação é consultado

Estudo sobre a realidade das organizações no que diz respeito a Documentação do Software

Foi pedido aos inquiridos uma análise classificativa do pacote de documentação segundo os seguintes parâmetros: fiabilidade, navegabilidade, facilidade e grau de actualização, conteúdo e organização do mesmo. O gráfico 5 mostra essa classificação.

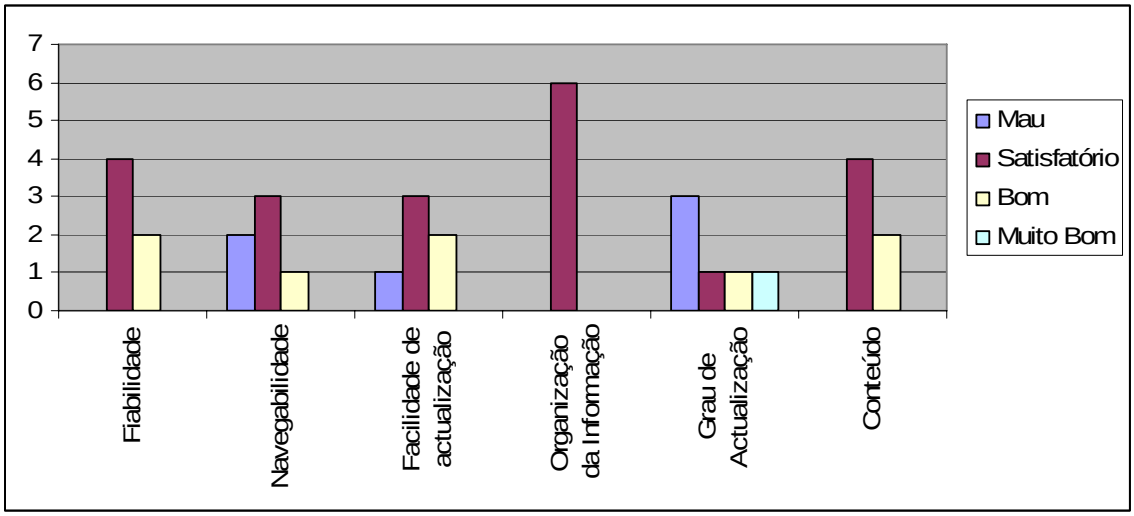


Gráfico 5: Classificação do pacote de documentação

Na resposta á pergunta: "Quais as principais desvantagens do pacote de documentação?" as respostas incidiram praticamente todas na questão da navegabilidade e facilidade de actualização.
A actualização do pacote de documentação, nas organizações aqui em estudo, é feita regra geral pela equipa de desenvolvimento.
Na questão número 3 foi pedido aos inquiridos que dessem a sua opinião sobre requisitos mais importantes de um pacote de documentação, as respostas voltaram a incidir sobre a facilidade e grau de actualização, a navegabilidade e a forma como a informação está organizada.
A produtividade e eficiência são as principais vantagens apontadas pelos inquiridos, como resposta á questão número 4.

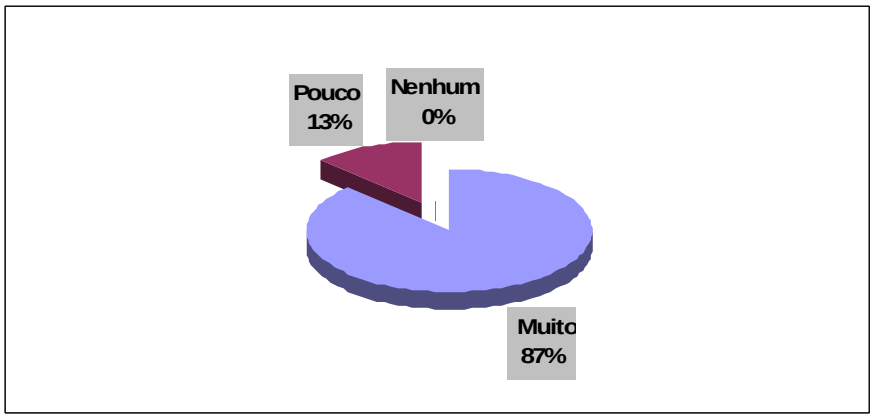


Gráfico 6: Síntese do interesse demonstrado pelos inquerido em conhecer a metodologia desenvolvida neste projecto

Estudo sobre a realidade das organizações no que diz respeito a Documentação do Software

Todas as organizações, que responderam a este inquérito mostraram interesse em conhecer a metodologia para documentação de uma arquitectura de software desenvolvida neste projecto de fim de curso.

5.2 - Análise dos resultados

Nesta secção serão identificados os principais requisitos de um pacote de documentação de uma arquitectura de software, através da análise dos resultados das respostas do inquérito. E mostrando como a metodologia desenvolvida neste projecto responde, ou não, a esses mesmos requisitos.

Uma análise geral as respostas deste inquérito mostrou que, apesar da maioria das organizações alvo do estudo conterem um pacote de documentação técnica, esse pacote de informação não corresponde ás necessidades nem as expectativas das pessoas que o responderam. Sendo que as desvantagens apontadas ao pacote de documentação existente (questão número 2.5) correspondem quase sempre aos principais requisitos (questão número 3) indicados para elaboração de um pacote de documentação.
Um dos requisitos mais importantes de um pacote de documentação, e que foi também um dos mais referidos nas respostas do inquérito é a facilidade de actualização da documentação. Este requisito, foi realizado parcialmente. Visto que a informação está organizada segundo um formato comum e está toda relacionada, torna-se fácil localizar e consequentemente actualizar a informação, isto no que diz respeito á informação descritiva. No que diz respeito aos diagramas, a sua actualização é mais complexa. O estudo de uma solução para este requisito faz parte do trabalho futuro deste projecto.
Um dos principais requisitos, também identificados neste inquérito, foi a navegabilidade. A resposta seguinte (no contexto da questão 4) mostra a importância deste requisito: "*Se porventura estiver organizado, e com um bom indice, em muito pouco tempo posso encontrar e ficar com o conhecimento que me levaria muito tempo e ineficiência, se fosse efectuado doutro modo*." Este requisito é valorizado tendo em conta também a frequência com que os inquiridos consultam a documentação. Para responder a este requisito o formato da apresentação do pacote de documentação assume um papel importante. Neste projecto o formato utilizado foi o Tikiwiki que responde plenamente a questão da navegabilidade.
Ainda na questão do formato do pacote de documentação (a organização da informação), o uso de uma metodologia de documentação comum ao pacote de documentação, é fundamental. Uma das respostas a questão número 3 ilustra este requisito: "*regras pré-definidas para que toda a documentação tenha o mesmo formato*". Este requisito foi também muito importante neste projecto, para concretizá-lo foi usado um único "*template*" para a documentação de cada vista (e estilo) e em termos de notação foi utilizado sempre a mesma metodologia, ou seja, o UML.
Esta análise revelou também que dois outros requisitos importantes para o pacote de documentação de uma arquitectura de software, são o nível de detalhe e a forma como essa documentação cobre as várias perspectivas do sistema, de salientar aqui duas respostas, a primeira, no contexto da questão número 3 e a segunda no contexto da questão 2.5:. "*Rapidez na obtenção de informação adequada à necessidade do técnico ou gestor de projecto." e "Descreve apenas pequenos módulos e nunca uma visão geral de todo o sistema*".

Na metodologia desenvolvida neste projecto esses requisitos são cobertos, através da divisão da documentação sobre várias perspectivas (vistas) com diversos níveis de detalhe, adequadas as necessidades dos stakeholders.
Concluindo esta análise, a metodologia desenvolvida neste projecto constitui uma solução de documentação que responde aos requisitos identificados pelas empresas analisadas.

4.Conclusão

Neste documento foi descrito o trabalho desenvolvido no âmbito deste PFC, com ênfase nos conceitos e técnicas empregues e o modo como estas foram usadas de modo a resolver os problemas propostos.

A documentação de um sistema já existente acarreta inúmeras desvantagens em relação á documentação de um sistema a desenvolver. Essas vantagens podem ser por exemplo, a falta de documentação base que ajude a perceber o sistema em causa, o facto de as pessoas que desenvolveram o sistema já não estarem envolvidas no mesmo, o facto do sistema estar em constante mutação, com surgimento de novas versões, entre outras. Neste caso específico essas desvantagens tiveram alguma importância, em especial o facto das pessoas que desenvolveram o sistema na sua fase inicial já não estarem envolvidas no mesmo. O que dificultou a aquisição de informação para alguns aspectos da documentação, nomeadamente o *rationale*. No entanto, de salientar a colaboração, sempre que solicitada, das pessoas que neste momento estão envolvidas no sistema.

O sistema ARQUO é um sistema, grandioso e em expansão. A sua documentação completa, como objectivo deste projecto, consistiu num estudo aprofundado de cada um dos módulos, desde a identificação das funcionalidades de cada um, até ao código que as implementa. O resultado foi um pacote de documentação extenso, mas devido a certas opções tomadas no desenvolvimento deste projecto, o resultado final é uma documentação de fácil leitura e compreensão. Essas opções são, por exemplo, o uso de um *template* comum para descrição de todas as vistas; a documentação para além das vistas que permite a cada stakeholder do sistema identificar facilmente as áreas do pacote de documentação que respondem às suas necessidades, bem como identificar os mapeamentos entre as vistas que lhe dão uma visão de como é que a informação está relacionada e o uso do Tikiwiki que permite uma interligação de toda a informação relacionada, facilitando a navegabilidade dentro do pacote de documentação.

4.1 - Trabalho Futuro

O trabalho apresentado neste relatório apresenta algumas limitações. A principal limitação identificada prende-se com a actualização da informação documentada. Esta limitação não é geral, a informação documentada pode ser facilmente alterada, desde que a pessoa que o pretende fazer tenha permissões para isso, isto no que diz respeito á informação apresentada na forma descritiva. No que diz respeito á actualização dos diagramas a solução apresentada neste projecto não é a mais funcional. Visto que, a actualização de um diagrama tem de ser feita na ferramenta CASE, usada para modelar o sistema e posteriormente é necessário aceder ao directório de imagens do Tikiwiki e substituir a imagem com o diagrama antigo pelo diagrama actualizado. Mais uma vez não é mais funcional, para minimizar esta limitação o directório de imagens, no Tikiwiki, foi organizado da mesma forma que o código que implementa os módulos do sistema ARQUO, permitindo assim que aquando de uma actualização de um diagrama seja fácil encontrar a imagem correspondente ao mesmo.

O trabalho futuro, consiste então em encontrar uma solução para esta limitação. Essa solução pode passar pela integração do Tikiwiki com o Enterprise Architect (ferramenta usada para edição de modelos). Esta solução já foi estudada, no decorrer deste projecto, e é uma potencial solução.

Não foram identificadas outras limitações.

5. Referências

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

[2] Kruchten, P. *The Rational Unified Process: An Introduction*, Second Edition. Addison-Wesley, Boston, 2001.

[3] IEEE 2000. IEEE Product No.: SH94869-TBR: Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Standard No. 1471-2000. Available at http://shop.ieee.org/store/.

[4] Hammer M, Champy J. *Re-engineering the corporation: a manifesto for business revolution.* London, Nicholas Brearley, 1995.

[5] Bredmeyer Consultin, *Software Architecture Documentation - Action Guides for Architecture Specification*, www.bredemeyer.com/architecture_documentation_action_guides.htm

[6] Rich Hilliard, "IEEE Std 1471 and Beyond", http://egov.alentejodigital.pt/Page10549/Arquitectura/IEEE_1471_and_Beyond.pdf

[7] Silva, Alberto, C. Videira. *UML Metodologias e Ferramentas CASE. Edições* Centro Atlântico, Lisboa, 2001.

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

[9] Especificação UML versão 2.0: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

[10] Bass, Len, Clements, Paul, Kazman, Rick. *Software Architecture in Practice, 2ª edição.* Addison-Wesley, 2003.

[11] The ISO Reference Model for Open Distributed Processing - An Introduction, http://www.enterprise-architecture.info/Images/Documents/RM-ODP2.pdf

[12] C4ISR Architecture Framework, http://www.fas.org/irp/program/core/fw.pdf

[13] Architectural Blueprints—The "4+1" View Model of Software Architecture, http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

[14] G. Booch: *Object-Oriented Analysis and Design with Applications*, 2ª edição, Benjamin-Cummings Pub. Co., Redwood City, California, 1993.

[15] Standard ISO/IEC 10746-1,
http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm

[16] The Process Analyses of Technological Production with ISO standard STEP/EXPRESS-P.
http://martin.feld.cvut.cz/~molhanec/VaV/files/publik/2003/expressp.pdf

[17] A Survey of Software Architecture Viewpoint Models, Nicholas May,
http://mercury.it.swin.edu.au/ctg/AWSA05/Papers/may.pdf

[18] Fitting the UML into Your Development Process, Paul Gustavson, SimVentions, Inc.
http://bdn1.borland.com/borcon2004/article/paper/0,1963,32151,00.html

[19] CURSO DE C++; Alceu Heinke Frigeri, Bernardo Copstein, Carlos Eduardo Pereira,
http://twiki.im.ufba.br/pub/Main/AlexSantana/cursocpp.pdf

[20] Modelagem de casos de uso.
http://www.ic.unicamp.br/~ariadne/inf301/Cap02-alunos.pdf

[21] WikiPédia – Java (Linguagem de Programação)
http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)

Anexos

Anexo A - Documentação de Arquitecturas de Software

1. Vistas e Estilos

A prática moderna das arquitecturas de software utiliza o conceito de vistas arquitecturais. Uma vista é uma representação do conjunto de elementos do sistema e das relações a eles associadas.
As vistas são representações das muitas estruturas do sistema que estão presentes simultaneamente nos sistemas de software. Os sistemas modernos são demasiado complexos para explica-los e percebe-los de uma só vez como um todo, assim, ao separar diferentes aspectos em vistas distintas é uma forma de gerir a complexidade. As vistas permitem reduzir a quantidade de informação que o arquitecto trata num dado momento. Cada vista descreve diferentes conceitos da Engenharia.
Alguns autores recomendam um conjunto fixo de vistas de forma a desenvolver e a comunicar uma arquitectura. A *Rational Unified Process [2]*, por exemplo, baseia-se na aproximação das 4+1 vistas de *Kruchten.* O modelo das quatro vistas da Siemens [1] é um outro exemplo. Uma recente tendência é o reconhecimento de que as arquitecturas devem produzir o número de vistas que sejam úteis para o sistema em questão, qualquer que seja esse número. O IEEE 1471 [3] exemplifica esta filosofia; através da afirmação de que a descrição de uma arquitectura consiste num conjunto de vistas, cada uma delas seguindo um *ponto de vista*, o que por sua vez é a realização das necessidades de um ou mais *stakeholders*.
Esta perspectiva sobre as vistas leva ao princípio fundamental da aproximação "para além das vistas" – *views and beyond:*

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

Estilos

Mesmo dentro dos limites de um tipo de vista, os elementos e as relações podem ser especializados em formas conhecidas, resultando em estilos. Os estilos representam aproximações ao desenho da arquitectura, já conhecidas.
Os estilos são documentados através de um guia de estilos. O guia de estilos é a descrição de um estilo arquitectural que especifica o vocabulário de desenho (conjunto de tipos elementos e tipos relação) e as regras (conjuntos de restrições topológicas e semânticas) e como o vocabulário pode ser usado. Este documento é composto por seis secções. A primeira secção – vista geral – explica porque é que este tipo de vista/estilo é útil para documentar uma arquitectura de software. A secção dois, apresenta os elementos (blocos d construção da arquitectura que são nativos para o tipo de vista/estilo), as relações (determinam como os elementos cooperam para alcançar o objectivo do sistema) e as propriedades (informação adicional acerca dos elementos, por exemplo o nome de uma camada). As secções seguintes descrevem, respectivamente, o tipo de racional suportado e o que não é suportado (quando se aplica e quando não se aplica), as notações, que podem ser gráficas ou textuais, as relações com outros estilos e tipos de vista semelhantes, e um exemplo de um sistema documentado com o estilo.
Quando um estilo é ligado a um sistema em particular, o resultado é uma vista.

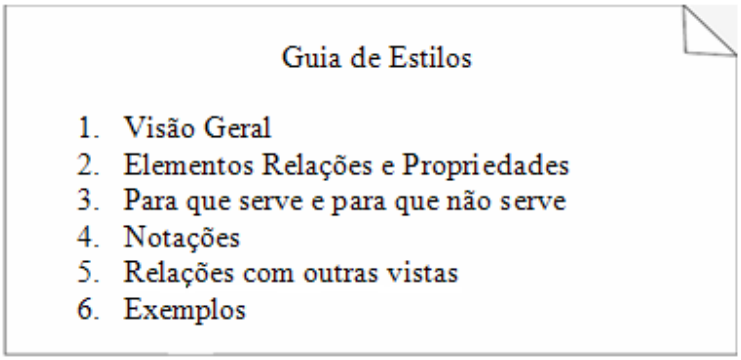


Figura 35: *Template* do documento do Guia de Estilos

Mesmo nos limites de um estilo, devem ser feitas escolhas: como é que os tipos de elementos e de relações num estilo estão ligados a um elemento ou relação no sistema. No contexto dos tipos de vista e estilos, uma vista pode ser tida como um estilo que está ligado a um determinado sistema.

2. Pacotes de vistas

As vistas de grandes sistemas de software podem conter centenas ou mesmo milhares de elementos. Mostrar todos esses elementos numa única apresentação, juntamente com as relações entre eles, pode resultar num "bombardeamento" de informação que será impossível decifrar, além de conter demasiada informação para qualquer *stakeholder*, que apenas requer parte da informação do sistema. Os arquitectos necessitam então de uma forma de apresentar a informação de uma vista em porções perceptíveis. A essas porções é dado o nome de **pacotes de vista**. Cada um destes pacotes mostra um fragmento do sistema. A documentação de uma vista consiste então num conjunto de pacotes de vista, mostrando, cada um, uma pequena parte do sistema. Um pacote de vista é o menor grão de documentação que é coeso e pode ser dado a um *stakeholder*.

Pacotes da mesma vista

Os pacotes de vista da mesma vista estão relacionados uns com os outros da mesma forma que se relacionam os nós de uma estrutura em árvore; eles documentam os elementos do sistema que são irmãos e filhos uns dos outros.
Sendo que os pacotes irmãos documentam diferentes partes do mesmo sistema e os pacotes filho documentam a mesma parte do sistema mas com um maior detalhe.

Pacotes de diferentes vistas

Os pacotes de vista de diferentes vistas podem estar também relacionados uns com os outros. Partindo do princípio de que nenhum sistema é desenvolvido a partir de um único estilo, podemos ver como o conceito dos pacotes de vistas lida com este princípio através do estudo de três casos.

Caso 1 - diferentes áreas do sistema possuem diferentes estilos. Por exemplo, um sistema pode usar um estilo canais-e-filtros para processar dados de entrada e fazer o encaminhamento do

resultado para uma base de dados que é acedida por múltiplos elementos. Este sistema será uma mistura dos estilos canais-e-filtros e dados-partilhados. A documentação deste sistema irá incluir uma vista canais-e-filtros que apresentará uma parte do sistema e uma vista dados-partilhados que apresentará a outra parte. Um ou mais elementos podem ocorrer em ambas as vistas e ter propriedades de ambos os tipos de elemento (de outra forma, as duas partes do sistema não poderiam comunicar entre si). Estes elementos são chamados de ponte e o seu objectivo é dar uma continuidade do conhecimento de uma vista para a outra. A figura 36, ilustra o exemplo anteriormente descrito. Onde o elemento ponte, ou o elemento comum aos dois estilos, é o conector que traduz o fluxo dos dados desde o ultimo filtro até á base de dados. O conector tem um papel que lida com o porto do filtro e outro que lida com o porto da base de dados.

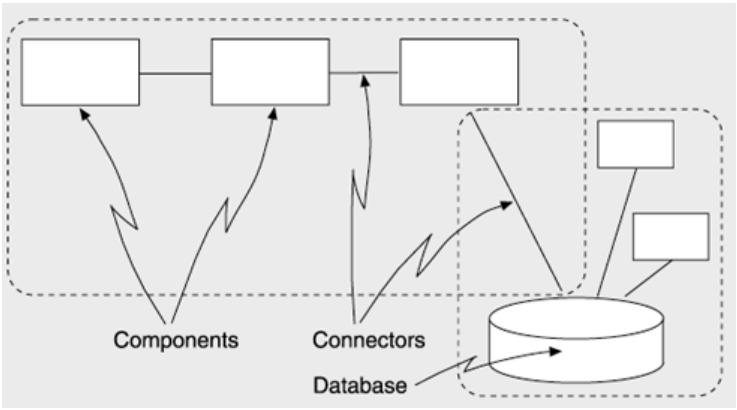


Figura 36: Um sistema com os estilos *canais-e-filtros* e *dados-partilhados*

Caso 2 - um elemento de um estilo é composto por elementos de outro estilo. Por exemplo, um servidor num sistema cliente-servidor pode ser implementado através do uso do estilo canais-e-filtros. A documentação para este sistema deve incluir uma vista cliente-servidor mostrando todo sistema, bem como uma vista canais-e-filtros documentando esse servidor, como mostra a figura 37. Para documentar este sistema, um pacote de vista na vista cliente-servidor vai mostrar todo o sistema, cujos elementos serão os clientes, os servidores, e os conectores que fazem a ligação entre eles. Um pacote de vista na vista canais-e-filtros vai documentar o servidor. A vista canais-e-filtros não vai mostrar todo o sistema mas sim a(s) parte(s) a que este se aplica. O pacote de vista cliente-servidor irá fazer referência ao pacote de vista na vista canais-e-filtros e vice versa.

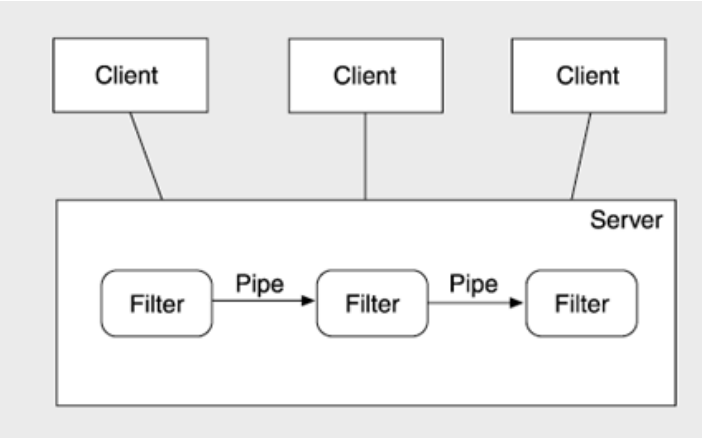


Figura 37: A decomposição de um elemento num estilo revela a substrutura num estilo diferente.

Caso 3 - o mesmo sistema pode ser visto em diferentes perspectivas. Por exemplo, num sistema canais-e-filtros em particular, os filtros acedem todos a um repositório de dados comum durante o seu processamento, como mostra a figura 38. Se olharmos para este sistema através de uns óculos que escondessem os canais, o sistema resultante seria do estilo dados-partilhados. Mas, se olharmos através de uns óculos que escondessem o repositório de dados e os conectores, o sistema seria claramente do estilo canais-e-filtros. Os óculos que se escolhe determinam o estilo que se "vê".

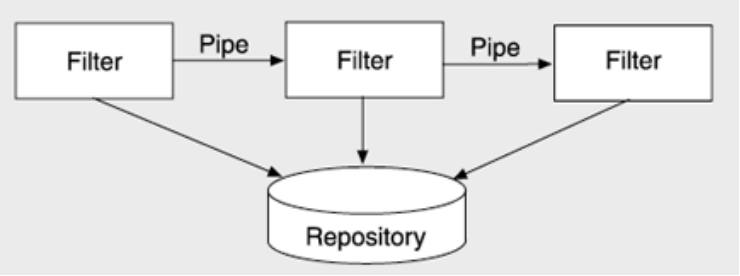


Figura 38: Um sistema do tipo canais-e-filtros com uma base de dados partilhada pode ser visto como a representação de qualquer um dos dois estilos.

2.1 - Refinamento

Os pacotes de vista que representam uma operação de *zoom-in* são refinamentos dos seus irmãos. Um refinamento é o processo de gradualmente mostrar mais informação, através de uma série de descrições.

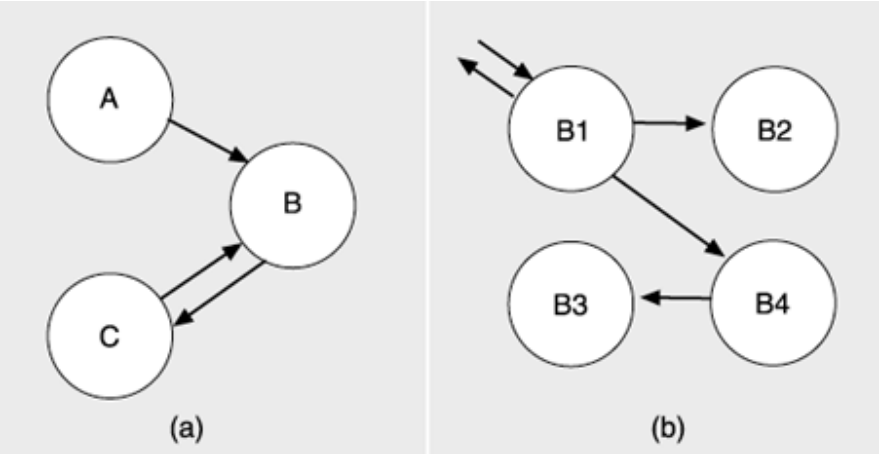


Figura 39: Exemplo de um sistema com três elementos A, B e C relacionados

Existem dois tipos de refinamento, o de Decomposição e o de Implementação. O refinamento de Decomposição é um refinamento no qual um elemento é elaborado para mostrar a sua estrutura interna e o mesmo se aplica recursivamente a cada elemento da estrutura interna. Usar o refinamento de Decomposição numa vista acarreta a obrigação de manter a consistência no que diz respeito a relação nativa dessa vista. Supondo, por exemplo, que a relação apresentada na figura 39 (a) é o envio de dados. Como o elemento B apresenta o envio e a recepção de dados, o refinamento de B na figura 39 (b) deve representar a entrada e a saída de dados: neste caso, isso acontece através de B1.
O refinamento de implementação é um refinamento no qual muitos, se não todos, os elementos e relações são substituídos por outros mais orientados para a implementação.

2.2 - Completude Descritiva

O conceito de completude descritiva, está relacionado com o refinamento, que diz como os elementos estão relacionados uns com os outros nos pacotes de vista. Os pacotes de vista com a propriedade da completude descritiva mostram todos os elementos e relações dentro da vista que documentam. A figura 40 mostra um diagrama arquitectural de um sistema imaginário. O elemento

A está relacionado com o elemento B de alguma forma – não é possível definir qual através do diagrama – o elemento B está relacionado com o elemento C e este por sua vez esta relacionado com B. O que pode ser concluído sobre A e C, existe relação entre eles?

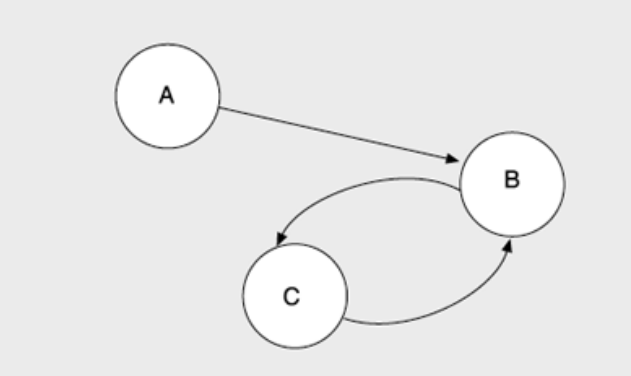


Figura 40: Relação entre os elementos A, B e C

Entre algumas respostas possíveis a esta questão, que representariam diferentes estratégias de documentação, interessa apenas a que diz respeito a completude descritiva. Ou seja, segundo esta estratégia os elementos A e C não estão relacionados porque não há nenhuma seta entre eles. Esta estratégia é usada em pacotes que querem transmitir instruções e restrições ás equipas de desenho e de implementação. Por exemplo, a vista de camadas mostra aos programadores que elementos podem usar – e, por extensão, que elementos não podem usar – quando estão a codificar o trabalho que lhes foi atribuído; assim se a figura 40 fosse dada ao codificador do elemento A, a ausência de uma seta entre A e C teria que ser interpretada com um significado: a proibição de A usar C.

3. Combinar vistas

O princípio básico de documentar uma arquitectura como um conjunto de vistas separadas, traz todas as vantagens do lema "dividir e conquistar" á tarefa de documentação. No entanto se as vistas forem irrevogavelmente diferentes, sem qualquer relação umas entre elas, será impossível compreender o sistema como um todo. Gerir a forma como as vistas estão relacionadas é uma tarefa importante no trabalho de um arquitecto. Por vezes a forma mais conveniente de mostrar uma forte relação entre duas vistas é *colapsá-las* numa vista combinada. Uma vista combinada é uma vista que contém elementos e relações que provêm de duas ou mais vistas; as combinações podem ser entre vistas ou entre estilos.
A opção de combinar um conjunto de estilos ou vistas é parcialmente dependente da importância, persistência, e utilidade futura do novo estilo ou vista resultante da combinação. Para diferenciar estes dois casos de vistas combinadas, são apresentados dois conceitos **estilo híbrido** e **sobreposição**.

Estilo Híbrido

Um estilo híbrido é a combinação de dois ou mais estilos existentes. Introduz as mesmas obrigações de documentação que qualquer outro estilo, deve seguir o formato de documentação de estilos apresentado anteriormente. Desta forma, fica garantida a fácil compreensão da documentação, no caso do estilo ser usado, num sistema em particular, por vários *stakeholders* que têm de estar familiarizados com esse estilo. No entanto, por vezes uma vista combinada é criada para um fim único e de curto prazo: para análise ou comunicação, por exemplo. Para estes fins de curto prazo, a criação da documentação requerida para um novo estilo é totalmente

desnecessária. Adicionalmente, a aplicação entre os estilos que constituem o estilo híbrido deve ser também documentada.

Sobreposição

Uma sobreposição é a combinação das apresentações primárias de duas ou mais vistas. Tem como objectivo a utilização de curto prazo. Introduz as mesmas obrigações de documentação que uma vista normal. Adicionalmente, a aplicação entre as vistas que constituem a sobreposição deve ser também documentada.

O arquitecto têm então três formas para estabelecer o mapeamento entre estilos ou vistas, que de outra forma estariam separados:

- Documentar a aplicação entre vistas separadas
- Criar uma sobreposição que combina a informação
- Criar um estilo híbrido combinando os estilos existentes e criando um guia do estilo

3.1 - Quando combinar vistas?

Produzir uma vista combinada que seja útil depende da compreensão do mapeamento entre as vistas e estilos constituintes. Uma vista combinada pode ser produzida a partir de diferentes vistas do mesmo tipo de vista.

Quando a arquitectura está a ser comunicada aos *stakeholders*, são necessárias múltiplas vistas para fornecer a informação suficiente de modo a permitir uma compreensão total do sistema. Quando a documentação da arquitectura serve como *blueprint*, um membro da equipa de desenvolvimento deve perceber bem as prescrições incluídas nas várias vistas de modo a implementar o sistema correctamente. Mantendo estes princípios separados facilita a apresentação clara da informação e aumenta a compreensão. Por outro lado, a equipa de desenvolvimento fica sujeita a um trabalho mental superior por ter que usar documentos separados e manter as suas relações correctas. Para além das queixas, com justificação, da equipa de desenvolvimento sobre o trabalho mental acrescido ainda existe o custo que implica manter múltiplas vistas. As múltiplas vistas tendem a tornar-se confusas á medida que evoluem, e a modificação de uma das vistas implica alterações em três documentos: um de cada vista, e o do mapeamento entre elas. A hiper ligação dos documentos ajuda, mas mesmo assim a equipa de desenvolvimento ainda terá que percorrer todos os documentos e fazer as adaptações apropriadas. Estas são boas razões para manter o número de vistas reduzido ao mínimo.

È necessário ter vários aspectos em atenção antes de decidir combinar vistas:

- Quando considerar uma vista combinada, ter a certeza de que o mapeamento entre os seus constituintes é claro e integro. De outra forma, o resultado pode ser uma vista confusa e complexa.
- Quanto maior for o sistema mais sentido faz manter vistas separadas, visto que um erro causado por uma combinação terá um impacto muito maior que num sistema de menor dimensão.
- Demasiados conceitos complicam as vistas combinadas.
- Antes de se fazer uma vista combinada assegurar que existe um stakeholder para ela. A escolha das vistas deve ser feita tendo em conta as necessidades dos stakeholders.

- Quanto mais sofisticada for a ferramenta de suporte mais vistas podem ser suportadas

Tipos de mapeamento

No mapeamento muitos-para-um (ver figura 41), múltiplos elementos numa vista são mapeados para um único elemento noutra vista. Os módulos são frequentemente mapeados para processos desta forma. O mapeamento deve deixar bem claro qual o módulo mapeado para um processo.

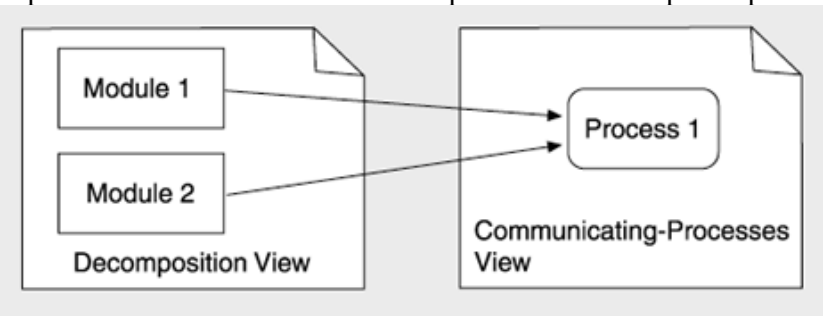


Figura 41: Mapeamento muitos-para-um

Num mapeamento de um-para-muitos (ver figura 42), um único elemento pode ser mapeado de uma vista para múltiplos elementos noutra vista. Por exemplo, um módulo de comunicações pode ser mapeado em múltiplos processos num estilo cliente-servidor de *n* camadas (n-tier).

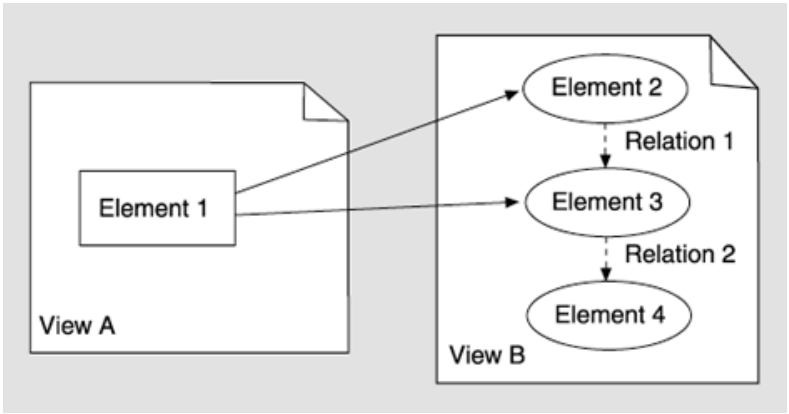


Figura 42: Mapeamento um-para-muitos

Num mapeamento de muitos-para-muitos é frequente ter de se "partir" os elementos. Uma aplicação complexa sugere que se deve manter as vistas isoladas e a aplicação deve ser documentada separadamente.

3.2 - Documentar vistas combinadas

A figura 42 mostra como múltiplos módulos podem ser mapeados para único processo. A figura 43 mostra como esse mapeamento pode ser documentado usando uma vista combinada.

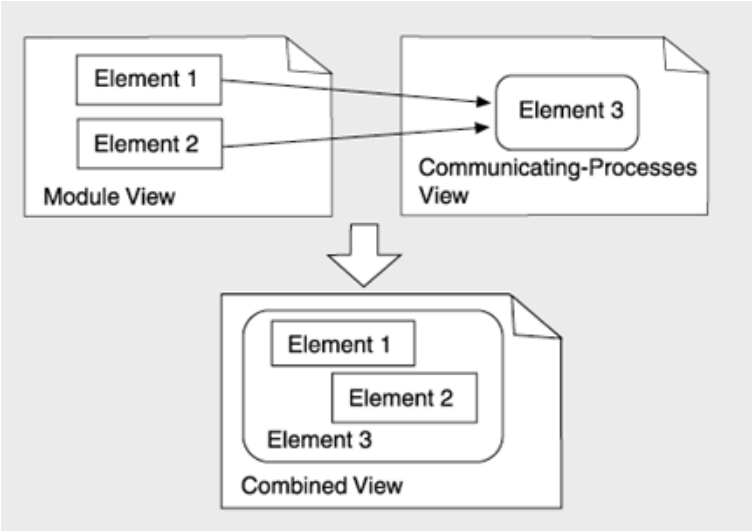


Figura 43: Documentação do mapeamento muitos-para-um usando um vista combinada

Na figura 44 mostra-se como um elemento de uma vista é mapeado para mais do que um elementos na segunda vista. A figura 44, mostra a representação disso num estilo híbrido.

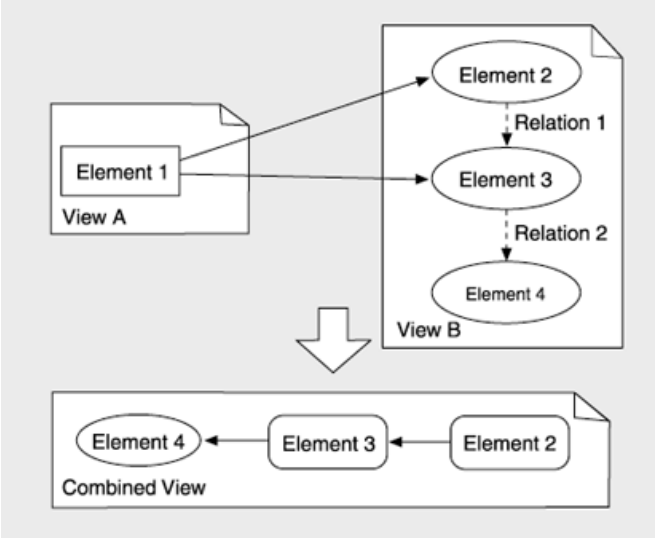


Figura 44: representação do mapeamento um-para-muitos através do uso do estilo híbrido

4. Variabilidade e Dinamismo

Há situações em que as decisões sobre alguns aspectos da arquitectura ainda não foram tomadas, mas, as opções disponíveis necessitam ser documentadas. São então distinguidos dois casos: variabilidade e dinamismo.

Variabilidade

A variabilidade refere as decisões que serão tomadas por um elemento da equipa de desenvolvimento antes da instalação do sistema.
A variabilidade ocorre porque:

- Algumas decisões não foram tomadas mas foram exploradas diferentes opções
- A arquitectura é para uma família de sistemas e as opções a tomar dependem da especificidade do particular membro da família
- A arquitectura é uma estrutura para uma colecção de sistemas e contém pontos explícitos (pontos de extensão) onde as extensões a essa estrutura podem ocorrer

Um ponto de variação é o local na arquitectura onde uma decisão específica foi limitada a um conjunto de opções, mas em que a opção a ser escolhida para um particular sistema foi deixada em aberto.
Para os dois primeiros casos apresentados anteriormente, três tipos de informação tem de ser documentada:

- Pontos de variação. O ponto de variação é o local na arquitectura onde ocorre a variação.
- Os elementos que são afectados por cada opção. Cada opção vai afectar a existência de um elemento, a sua interface, as propriedades desse elemento ou as relações entre os elementos.
- O tempo de concretização de cada opção – tempo de desenho, tempo de compilação, tempo de ligação, tempo de execução (início do programa, re-início do programa, início do contentor da opção, outro momento)

No caso da documentação da estrutura, os pontos de extensão têm de ser documentados. Um ponto de extensão é o local na arquitectura onde podem ser anexados elementos adicionais. Cada ponto de extensão é documentado através de uma interface de descrição sobre o que estrutura fornece e o que a extensão requer.

Dinamismo

O dinamismo refere-se ás decisões que serão tomadas ou reconsideradas pelo sistema durante a execução.
A arquitectura muda durante execução em resposta aos requisitos do utilizador ou para permitir uma melhor forma de atingir algum atributo de qualidade em particular. Uma arquitectura pode mudar dinamicamente através da criação ou "destruição" de componentes e conectores. Por exemplo, quando um novo utilizador faz a entrada num ambiente e requer novos serviços, os componentes que vão fornecer esses serviços são criados. Quando o utilizador sai desse ambiente, os componentes serão apagados.
Os componentes e conectores criados devem ser uma réplica ou então exclusivos. De qualquer forma o número de réplicas permitidas, as condições sobre as quais a criação ou destruição ocorrem e os componentes ou conectores criados devem ser documentados.
Uma outra forma de mudar uma arquitectura dinamicamente é através da re-alocação de recursos ou responsabilidades. Os componentes podem ser deslocados de um processo para outro de forma a fornecer um melhor desempenho.

Documentar a informação

O trabalho envolvido na documentação da variabilidade e do dinamismo numa arquitectura é igual. De seguida são apresentadas algumas regras para documentar arquitecturas com essas características:

- Escolher os estilos usuais
- Mostrar o que é constante e o que não é
- Indicar quando as alterações ocorrem
- Documentar dependências entre as opções
- Usar cenários de construção de diferentes tipos do sistema no guia de variabilidade

O guia de variabilidade deve identificar os pontos de variação - através dos seguintes dados: nome, descrição, opcional ou obrigatório, número de possíveis ocorrências das variantes, entre outros, - deve conter os fundamentos para a escolha de uma opção em detrimento de outra(s) e deve também descrever o que é necessário ser feito – desde a edição de um ficheiro de configuração á implementação de novos módulos - para que a escolha de uma alternativa cumpra uma determinada interface.

5. Escolha das vistas

Antes de um vista ser documentada, esta tem de ser previamente escolhida pelo arquitecto. E esse é o tópico que será abordado neste ponto: como é que um arquitecto decide que vistas deve incluir no pacote de documentação.
O conhecimento sobre as vistas a produzir, o momento e o nível de detalhe só pode ser obtido tendo em conta o contexto do projecto.
Para que o desenvolvimento de um projecto seja bem sucedido, apenas podemos determinar que vistas são necessárias, quando cria-las e o nível de detalhe para a descrição das mesmas, se soubermos:

- Quais as competências das pessoas disponíveis
- Qual o orçamento disponível
- Qual é a calendarização
- Quais são os *stakeholders* mais importantes

De forma a escolher o conjunto de vistas apropriado para um projecto, é necessário identificar os *stakeholders* que dependem da documentação da arquitectura de software. È também importante perceber as necessidades de informação de cada *stakeholder*. O número de stakeholders varia, dependendo do tipo de organização e do tipo de projecto.
Por exemplo, o gestor de projecto poderá não ter interesse em nenhuma vista do tipo C&C. Mas, no caso de termos um produto com uma arquitectura cliente–servidor baseada na Web este deverá ter uma vista do tipo C&C mostrando partes da arquitectura cliente-servidor que é do interesse do gestor de projecto.

De seguida, é apresentado um procedimento de três passos para a escolha das vistas.
O primeiro passo consiste na produção de uma lista de vistas candidatas, através da produção de uma tabela stakeholder/vista com diferentes níveis de detalhe (genérico, detalhado ou muito detalhado) dependendo das necessidades. O anexo B apresenta um exemplo de uma tabela stakeholder/vista para um caso geral.
È uma boa estratégia para o sucesso do próprio projecto que o arquitecto promova, pelo menos, uma reunião com os *stakeholders* a fim de perceber quais as necessidades de informação de cada um.
O segundo passo consiste na redução da lista de vistas para um tamanho razoável. De seguida são analisadas as vistas de modo a ver quais as que podem ser combinadas, esta combinação pode ser feita através da escolha das vistas partilhadas por *stakeholders*, ou por exemplo combinando a

vista trabalho com a vista implementação, ou da vista instalação com outra qualquer vista do tipo C&C.
Depois de concluído o passo dois o resultado final deve ser um conjunto mínimo de vistas necessárias de modo a satisfazer as necessidades da comunidade de stakeholders. O passo seguinte será a decisão do que fazer primeiro, ou seja, priorizar. Essa decisão depende dos detalhes específicos do projecto em questão, no entanto há alguns aspectos a ter em consideração: quais os *stakeholders* mais importantes, se a arquitectura ainda não foi avaliada dar importância ás vistas que ajudam a avaliar (ver anexo C) e, muito importante, não descurar a descrição do rationale.

6. Documentar uma vista

Como referido anteriormente uma vista consiste num conjunto de pacotes que estão relacionados através de relações pai, irmão e filho. Documentar uma vista, torna-se então na tarefa de documentar uma série de pacotes de vista. Independente da vista, a documentação para um pacote de vista pode ser colocado numa organização padrão que consiste em sete partes.

1 **A primeira apresentação**. A primeira apresentação mostra os elementos e as relações entre eles que povoam esta porção de vista apresentada neste pacote de vista. Esta primeira apresentação deve conter a informação sobre o sistema que queremos transmitir primeiro. Deve conter os elementos primários e as suas relações mas devido a certas circunstâncias não os deve incluir a todos.
   Esta primeira apresentação é usualmente gráfica, neste caso deve conter uma chave que explique e descreva cada símbolo utilizado. Mas por vezes, a primeira apresentação é textual e deve também apresentar um sumario que descreva a informação mais importante no pacote de vista.

2 O **catálogo de elementos** detalha a informação sobre os elementos descritos na primeira apresentação. Se na primeira apresentação foram omitidos elementos ou relações relevantes ao pacote de vista em questão, então estes devem ser introduzidos e explicados no catálogo. O catalogo é composto por:
   a. Os elementos e as suas propriedades. Esta secção apresenta cada elemento do pacote de vista e lista as propriedades desse elemento.
   b. As relações e as suas propriedades. Cada vista tem tipos de relações específicas que são descritas entre os elementos dessa vista.
   c. As interfaces dos elementos. Uma interface é uma fronteira através da qual os elementos interagem e comunicam uns com os outros. Esta secção documenta as interfaces dos elementos.
   d. O comportamento dos elementos. Alguns elementos têm interacções complexas com o seu ambiente. Para fins de compreensão ou análise, é necessário que o arquitecto especifique o comportamento dos elementos.

3 Um **diagrama de contexto** que mostra como o sistema ou parte dele descrito no pacote de vista em questão, se relaciona com o seu ambiente.

4 Um **guia de variabilidade** que mostra como lidar com os pontos de variação que sejam parte da arquitectura descrita no referente pacote de vista.

5 O ***background* da arquitectura** explica como é que o desenho reflectido no pacote de vista será. O objectivo desta secção consiste em explicar porque é que o desenho é como é e fornecer um argumento convincente de que é a melhor opção. O background da arquitectura inclui:
   a. *Rationale*. O arquitecto explica porque foram tomadas as decisões referentes ao desenho reflectido no pacote de vista e apresenta uma lista de alternativas rejeitadas e o motivo da rejeição.
   b. Resultados da análise. O arquitecto deve documentar os resultados das análises que foram feitas, (por exemplo os resultados da análise de desempenho ou segurança) bem como a lista do que deve ser mudado em caso de haver um certo tipo de modificação no sistema.
   c. Assumpções. O arquitecto deve documentar qualquer assumpção que faça na criação do desenho. As assumpções sobre o ambiente documentam o que o arquitecto assume ser valido no ambiente e o que pode ser utilizado pelo sistema que está a ser desenhado. Podem ser feitas também assumpções sobre invariantes do ambiente. Por fim, as assumpções sobre o ambiente podem pertencer ao ambiente de desenvolvimento, por exemplo os níveis de perícia da equipa de implementação ou o conjunto de ferramentas disponíveis.
      As assumpções assumem um papel crucial na validação de uma arquitectura. O desenho produzido por uma arquitectura é feito em função destas assumpções. E documentá-las explicitamente torna muito mais fácil revê-las em termos de exactidão e solidez do que tentar descobri-las examinando o desenho.

6 **Outra informação** fornecida, varia de acordo com as práticas padrão da organização ou das necessidades de um projecto em particular. É conveniente guardar a informação não arquitectural “perto” da arquitectura, esse é o propósito desta secção.

7 **Pacotes de vista relacionados**. Esta secção fornece um apontador para o pacote de vista irmão e filho (caso existam).

A figura 45 apresenta sumariamente o *template* do documento de uma vista.

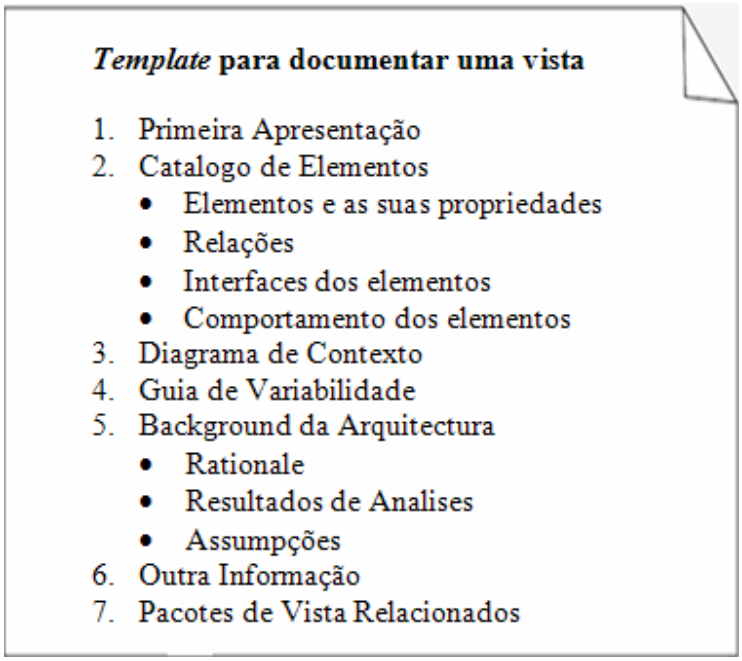


Figura 45: *template* para documentar uma vista

Cada pacote de vista consiste então, numa primeira apresentação, normalmente gráfica, e a documentação de suporte que explica as figuras. Para realçar a forma como a documentação de suporte complementa a primeira apresentação, passou a chamar-se *cartoon* arquitectural á parte gráfica do pacote de vista. E a definição é a mesma que a do mundo da arte: um *cartoon* é um esboço preliminar do trabalho final; tem como objectivo lembrar que a figura, embora tendo a maioria das atenções, não é a descrição completa mas só um esboço. De facto ela deve ser considerada como uma mera introdução á informação ou como um sumário da mesma, fornecido pela informação de suporte.

7. Documentar para além das vistas

Podemos considerar que uma arquitectura é para um sistema o que um mapa do mundo é para o mundo. O foco principal até ao momento foi dado á captura das várias vistas de um sistema, o que nas palavras de Miles Harvey, são os “continentes, oceanos, montanhas, lagos, rios e fronteiras politicas” do sistema completo que está a ser desenhado.
A fase final da documentação da arquitectura consiste na captura de informação que se aplica a mais do que uma vista e ao pacote de documentação como um todo. Este junta as vistas, e fornece uma imagem completa do desenho. A documentação para além das vistas consiste em três grandes aspectos, que podem ser resumidos a “como - o quê - porquê”:

1. Como é que a documentação está disposta e como está organizada para que o *stakeholder* da arquitectura possa encontrar a informação que necessita com eficiência e com garantia de confiança. Esta parte consiste num guia de documentação e num *template* de vista.
2. O que a arquitectura é. A informação que resta para ser capturada para além das próprias vistas é uma curta revisão para esclarecer qualquer leitor sobre o propósito do sistema, a

forma como as vistas estão relacionadas umas com as outras, uma lista dos elementos com a sua localização e um glossário e uma lista de acrónimos para toda a arquitectura.
3. Porquê que a arquitectura é desta forma: o *background* para o sistema, restrições externas que tenham sido impostas para moldar a arquitectura de alguma forma, e o *rationale* para decisões em larga escala.

A figura 46 apresenta sumariamente o *template* da documentação para alem das vistas.

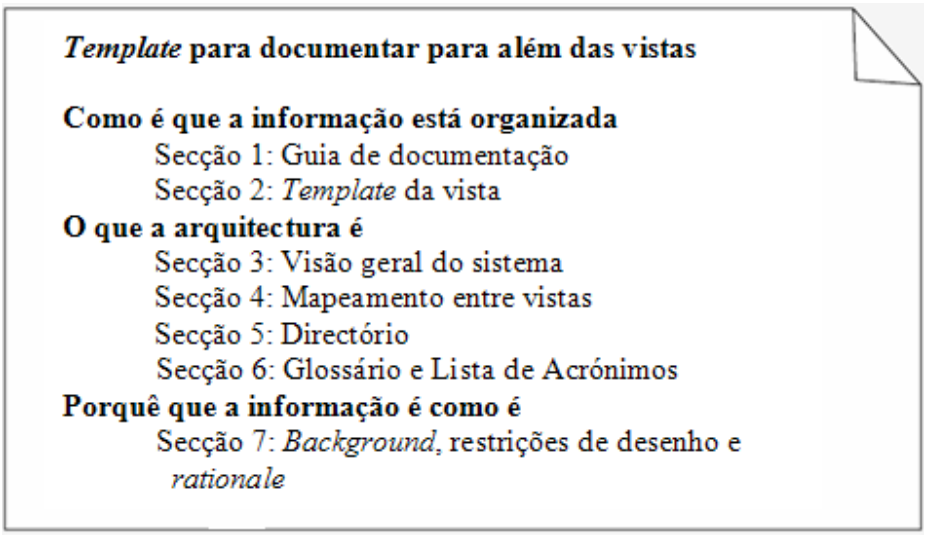


Figura 46: *template* para a documentação além das vistas

7.1 - Como é que a informação esta organizada de forma a servir o stakeholder

Cada parte da documentação da arquitectura necessita de uma introdução que explique a forma como está organizada a um novo *stakeholder*, e de forma a ajudar um *stakeholder* a aceder á informação que lhe interessa. Há duas formas de "como" a informação pode ajudar um *stakeholde*r de uma arquitectura: um guia de documentação e um *template* de vista.

Guia de Documentação

O guia de documentação é a introdução da informação que o arquitecto seleccionou para incluir na documentação. Quando a documentação é usada como meio de comunicação, é necessário que um novo leitor consiga determinar facilmente onde pode encontrar uma determinada informação. Quando a documentação é usada como base para a análise, é necessário saber que vistas contêm a informação necessária a uma análise em particular. Um guia vai conter esta informação. Um guia consiste em apenas duas secções:

1. Descrição das partes. O guia começa com uma breve descrição de cada parte do pacote de documentação. A parte mais importante do guia é a descrição as vistas que um arquitecto incluiu no pacote. O guia apresenta, para cada vista:
   a. O nome da vista e qual o estilo que esta instancia.

   b. Uma descrição dos tipos de elementos da vista, dos tipos de relações e dos tipos das propriedades.
   c. Uma descrição sobre para que é a vista destinada.
   d. Uma descrição da linguagem, técnicas de modelação ou métodos analíticos usados na construção da vista.
2. Como é que os *stakeholders* devem usar este pacote. O guia prossegue descrevendo como é que os vários *stakeholders* devem aceder ao pacote de forma a alcançar os seus propósitos.

***Template* da vista**

Um *template* de vista corresponde à organização padrão de uma vista. O ponto 6 "Documentar uma vista" forma uma base para um *template* de vista definindo como padrão as partes do documento de uma vista bem como o conteúdo e as regras para cada uma dessas partes. O objectivo do *template* de vista consiste em ajudar o leitor a navegar rapidamente para uma secção de interesse, e ajudar o escritor a organizar a informação e a estabelecer um critério para saber o andamento do trabalho.

7.2 - O que a arquitectura é

Esta secção apresenta a visão geral do sistema, no mapeamento entre vistas, no directório, e no glossário e lista de acrónimos do projecto.

Visão geral do sistema

É uma pequena descrição da função do sistema, dos seus utilizadores, e toda a informação de *background* e restrições que seja importante.
A visão geral do sistema não é uma parte da arquitectura e não contem informação arquitectural, no entanto, é indispensável para compreender a arquitectura.

Mapeamento entre vistas

Porque todas as vistas de uma arquitectura descrevem o mesmo sistema, podemos concluir que haverá duas vistas quaisquer que terão muito em comum. Ao ajudar um leitor ou um outro consumidor da documentação a perceber a relação entre vistas, vai ajudar esse leitor a ter uma visão da arquitectura como um todo. Clarificar as relações através do mapeamento entre vistas é a chave para aumentar a compreensão e diminuir a confusão ao leitor.

Directório

O directório é um índice de todos os elementos, relações e propriedades que aparecem nas vistas, juntamente com um apontador para a sua definição.

Anexo A

Glossário e Lista de Acrónimos

O glossário e a lista de acrónimos definem os termos usados na documentação da arquitectura, que têm um significado especial. Estas listas, se existirem como parte da documentação do sistema ou projecto, devem ser representados como apontadores no pacote de arquitectura.

7.3 - Porquê que a arquitectura é como é

Esta secção documenta uma fundamentação cruzada; isto é, documenta o raciocínio por detrás das decisões que se aplicam a mais do que uma vista. Os primeiros candidatos para a fundamentação cruzada incluem a documentação do *background* ou restrições organizacionais que levam a decisões importantes do sistema.

8. O compromisso com o Futuro

As empresas e organizações estão inseridas num ambiente, em constante mudança, onde variáveis como o crescimento de mercado, a procura, o ciclo de vida dos produtos, a tecnologia e a natureza da competição são imprevisíveis.

A necessidade de reengenharia e de gestão nas empresas está crescendo rapidamente, tornando-se factores chave de sobrevivência e competitividade das companhias. Nesse caso a organização precisa desenvolver uma forma de organizar todo o conhecimento requerido, para identificar a necessidade de mudança na organização, e para cumprir essas mudanças de maneira conveniente e profissional.

Tendo isso em conta é necessário fomentar a manutenção da documentação da arquitectura, tornando-se assim imperioso adicionar informação aos documentos anteriormente referidos. Essa informação pode ser:

- as oportunidades de aperfeiçoamento (erros, alto custo, atrasos, má qualidade) identificadas
- as medições no processo e metas de desempenho que sejam desenvolvidas;
- as auditorias ao sistema;
- as revisões de qualificação realizadas;
- as avaliações do impacto de mudanças no negócio e nos clientes;
- a comparação do sistema com outros de igual ou superior desempenho;
- entre outras.

Esta informação pode ser documentada na secção 5 do documento de pacotes de vista ou na secção 6 que diz respeito a outras informações relevantes.

9. Sumário

As arquitecturas são demasiado complexas para serem comunicadas de uma só vez, tal como um objecto de quinta dimensão, por exemplo, não pode ser visto nem compreendido na sua plenitude no nosso mundo tridimensional. Como uma forma de dividir e conquistar a complexidade, as vistas são de longe o meio de comunicação arquitectural mais eficaz conhecido, até ao momento.

Anexo A

As vistas, tal como os estilos e os padrões, estabelecem um vocabulário especializado e partilhado, permitem a reutilização do conhecimento técnico de um sistema para o próximo e facilitam a análise e prognósticos.
Relacionar as vistas umas com as outras e tornar a documentação acessível aos seus *stakeholders* completa as obrigações de comunicação com os mesmos.
Conhecer o fundamento (*rationale*) e o porquê das decisões tomadas completa as obrigações de comunicação com o futuro.
Esta é a essência da documentação: reconhecer e libertar o arquitecto das obrigações de comunicação á comunidade de *stakeholders*, ao presente e ao futuro e a quem possa necessitar da arquitectura.

Anexo B - Tabelas Stakeholder/Vista para um caso geral

Vistas / Stakeholder	Vista Módulo				Vista C&C				
	Decomposição	Utilização	Generalização	Camadas	Canais-e-filtros	Dados-partilhados	Cliente-servidor	Par-a-Par	Processos-Comunicantes
Gestor de Projectos	***	***		***	*	***	***	***	*
Membro da equipa de desenvolvimento	*****	*****	*****	*****	*	*****	*****	*****	*****
Membro da equipa de testes e de integração	*****	*****	*****	*****	*	*****	*****	*****	***
Designer de outros sistemas									
Membro da equipa de manutenção	*****	*****	*****	*****	*	*****	*****	*****	*****
Cliente					*	*	*	*	*
Utilizador Final									
Analista	*****	*****	***	*****	*	*****	*****	*****	*****
Pessoal da insfraestrutura de suporte	***	***							
Novo stakeholder	X	X	X	X					
Actual/Futuro Arquitecto	*****	*****	*****	*****	***	*****	*****	*****	*****

Key: ***** detalhada, *** menos detalhada, * visão geral, x qualquer coisa

Tabela 11: Tabela Stakeholder/Vista Módulo e C&C

Anexo B

Vistas / Stakeholder	Vista Afectação			Outras				
	Instalação	Implementação	Trabalho	Diagramas de Contexto	Mapeamento entre vistas	Guia de Variabilidade	Analise de Resultados	Rationale
Gestor de Projectos	*****		*****	*				***
Membro da equipa de desenvolvimento	***	***		*****	*****	*****		***
Membro da equipa de testes e de integração	***	***		*****	***	*****		***
Designer de outros sistemas				*				
Membro da equipa de manutenção	***	***		*****	*****	*****		*****
Cliente	*		*	*			***	
Utilizador Final	***						***	
Analista	*****			*****		***	*****	***
Pessoal da insfraestrutura de suporte	***	*****				*****		
Novo stakeholder	X	X	X	X	X	X	X	X
Actual/Futuro Arquitecto	*****	***	***	*****	*****	*****	*****	*****

Key: ***** detalhada, *** menos detalhada, * visão geral, x qualquer coisa

Tabela 12: Tabela Stakeholder/Vista Afectação e outras

Anexo C - Vistas envolvidas na avaliação dos requisitos do sistema

Vistas / Requisitos	Vista Módulo				Vista C&C				
	Decomposição	Utilização	Generalização	Camadas	Canais-e-filtros	Dados-partilhados	Cliente-servidor	Par-a-Par	Processos-Comunicantes
Portabilidade									
Desempenho	*****	***	*****	***	*	*****	*****	*****	*****
Segurança	*****	***	*****	*****	*	***	*****	*****	*****
Usabilidade	*****								*****
Modificabilidade	*****	*****	***	***		*	*	*	*****
Disponibilidade	*****	***	*****	*****				***	***
Integridade do dados	***	***	***	*****	*	*****	*****	*****	*****
Tempo de Rresposta									

Key: ***** detalhada, *** menos detalhada, * visão geral

Tabela 13: Vistas Módulo e C&C envolvidas na avaliação dos requisitos

Anexo C

Vistas / Requisitos	Vista Afectação		
	Instalação	Implementação	Trabalho
Portabilidade	*****	***	***
Desempenho	*****	*****	
Segurança	*****	*	*
Usabilidade			
Modificabilidade	*	*	
Disponibilidade	*****		*
Integridade do dados	*****		
Tempo de Resposta	***	***	***

Key: ***** detalhada, *** menos detalhada, * visão geral

Tabela 14: Vistas Afectação envolvidas na avaliação dos requisitos

Anexo D – Escolha das vistas Arquitecturais

Introdução

Existem vários conceitos ou abordagens sobre os tipos de vistas necessárias para representar uma arquitectura de software. Vários autores, por sua conta própria ou sobre a tutela de grandes organizações, desenvolveram diferentes modelos com conjuntos de vistas específicos para a arquitectura de um sistema de software. Alguns exemplos desses modelos são: as "quatro mais uma" vistas da Rational Unified Process/Kruchten [2,13], as quatro vistas da Siemens [1], o padrão para a documentação de arquitecturas ANSI/IEEE-1471-2000 [1], as cinco vistas da RM-ODP (Reference Model for Open Distributed Processing) [1,11], as três vistas da C4ISR (Command, Control, Comunications, Computers, Intelligence, Surveillance and Reconnaissance) [1,12], entre outras. Uma recente tendência é o reconhecimento de que as arquitecturas devem produzir o número de vistas que sejam úteis para o sistema em questão, qualquer que seja esse número.
De seguida são apresentadas com mais pormenor as várias perspectivas sobre as vistas arquitecturais.

1. As "quatro mais uma" vistas da Rational Unified Process/ Kruchten

Em 1995, Philippe Kruchten, publicou um artigo muito influente no qual apresentava as suas "quatro mais uma" vistas para as arquitecturas de software. Esta aproximação popularizou-se e foi institucionalizada como a base conceptual da Rational Unified Process [2].
De seguida são brevemente apresentadas as quatro mais uma vistas de Kruchten.

A vista Lógica

A vista lógica suporta principalmente os requisitos funcionais – os serviços que o sistema deve fornecer aos seus utilizadores finais. A equipa de desenho decompõe o sistema num conjunto de abstracções chave, retiradas do domínio do problema. Estas abstracções podem ser objectos ou classes de objectos. Adicionalmente á ajuda na análise funcional, a decomposição identifica mecanismos e elementos de desenho que são comuns ás várias partes do sistema.

Vista de Processos

A Vista de processos faz o endereçamento da cooperação e da distribuição, da integridade do sistema, e da tolerância a falhas. A vista de processos especifica também que linha de controlo executa cada operação de cada classe identificada na vista lógica.

Vista de desenvolvimento

A Vista de desenvolvimento focaliza-se na organização dos módulos do sistema no ambiente de desenvolvimento do software. As unidades desta vista são pequenas porções de software – bibliotecas ou subsistemas – que podem ser desenvolvidas por um ou mais membros da equipa de desenvolvimento. A vista de desenvolvimento suporta a afectação de requisitos e 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.

Vista física

A Vista física tem em conta os requisitos do sistema, tais como desempenho, disponibilidade e fiabilidade (tolerância a falhas). Esta vista faz o mapeamento dos vários elementos identificados na vista lógica, de processos e de desenvolvimento – redes, processos, tarefas, e objectos - com os nós de comunicação e de processamento.

Cenários

Finalmente, Kruchten mostra, através do uso de um pequeno subconjunto de cenários importantes – instâncias de casos de utilização – que os elementos destas quatro vistas trabalham juntos e de forma semelhante. Esta é a "mais uma" vista, redundante em relação as outras mas servindo dois objectivos distintos:

- Como um condutor para descobrir os elementos da arquitectura durante o desenho da arquitectura.
- Como uma representação da validação e ilustração depois de completo o desenho da arquitectura.

2. As quatro vistas da Siemens

Pela mesma altura, 1995, Dilip Soni, Robert Nord, e Christine Hofmeister da Siemens Corporate Research, fizeram uma observação semelhante sobre as vistas arquitecturais.

A abordagem da Siemens consiste na documentação da arquitectura através de quatro vistas. As quatro vistas e as tarefas de desenho associadas a elas estão representadas na figura 47. A primeira tarefa para cada vista é uma análise global. O segundo e terceiro grupos de tarefas são as tarefas de desenho centrais e finais, que definem os elementos, as relações entre eles e as propriedades mais importantes da arquitectura.

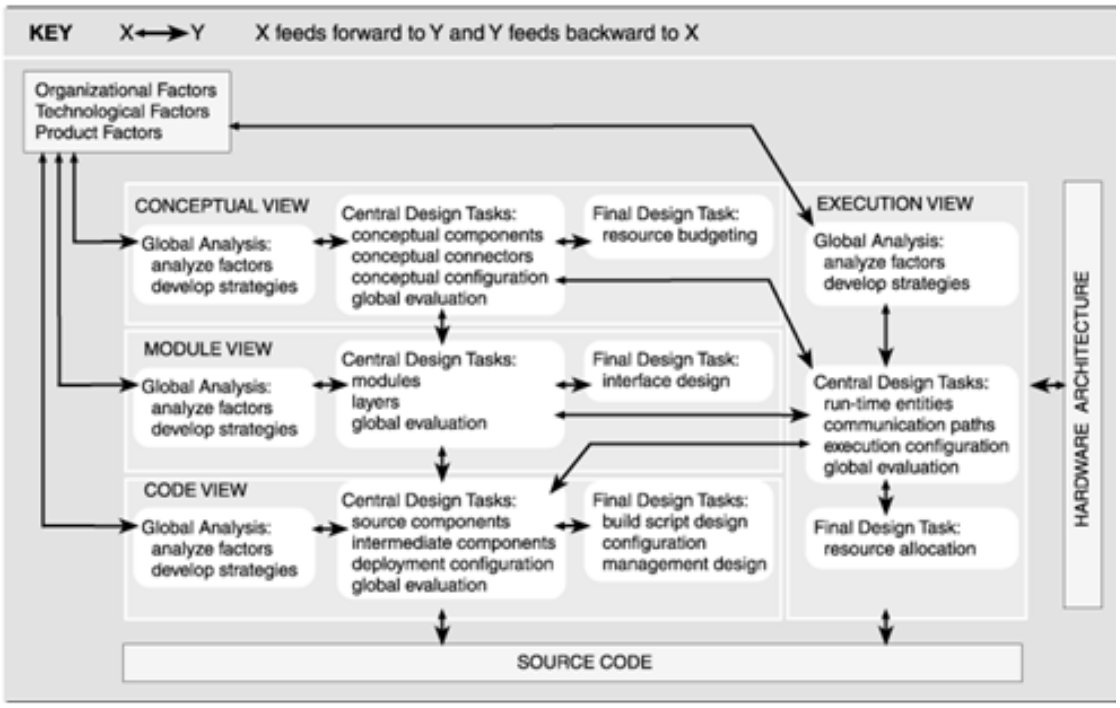


Figura 47: As quatro vistas da Siemens para as arquitecturas de software (" Documenting Software Architectures" [1])

Na análise global são identificados e analisados os factores que influenciam a arquitectura, de forma a obter as estratégias para desenhar a arquitectura. Fornecendo assim, documentação de suporte contendo a análise dos factores que influenciam a arquitectura e o rationale sobre as decisões de desenho que foram tomadas.

Vista conceptual – explica como é que as funcionalidades do sistema estão mapeadas 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 de interligação de módulos – explica como é que os componentes, conectores, portas e papéis estão mapeados com os módulos abstractos e as suas interfaces; O sistema é decomposto em módulos e em subsistemas. Um módulo pode também ser atribuído a uma camada, o qual limita então as suas dependências a outros módulos.

Vista de execução – explica como é que as funcionalidades do sistema estão mapeadas com os elementos da plataforma de execução, tais como processos e bibliotecas partilhadas.

Vista de código – explica como é que o software que implementa o sistema está organizado.

O foco principal deste modelo é o desenho tendo em conta as necessidades do arquitecto de software. Sendo assim, este modelo falha no que diz respeito aos interesses dos outros

stakeholders. São definidos alguns mapeamentos importantes das estruturas, no que diz respeito ao desenho. As estruturas conceptuais são “implementadas-por” estruturas módulo, e “atribuídas-a” estruturas de execução. As estruturas módulo podem estar “localizadas-em” ou ser “implementadas-por” estruturas código. As estruturas código podem configurar estruturas de execução.
O modelo Siemens ignora as necessidades explicitas dos stakeholders, excepto, como referido anteriormente, os arquitectos de software. No entanto, os outros stakeholders podem ser implicitamente endereçados através das necessidades que são satisfeitas por cada uma das vistas. Isto reflecte o foco do modelo no desenho do ponto de vista do arquitecto e não na documentação para a comunicação.

3. C4ISR

O Departamento de Defesa (DoD) dos Estados Unidos da América definiu uma aproximação coordenada, isto é, definiu uma estrutura para o desenvolvimento, apresentação e integração de arquitecturas para ser usada nos serviços militares e organizações de defesa. Esta estrutura define uma aproximação coordenada para os domínios militares: Comando, Controlo, Computadores, Comunicação, Inteligência, Vigilância e Reconhecimento (Command, Control, Computers, Communication, Intelligence, Surveillance, and Reconnaissance - C4ISR).
O objectivo da estruturaC4ISR é promover sistemas informáticos integrados, inter operáveis e rentáveis dentro e através das organizações do DoD.
A motivação que levou ao desenvolvimento desta estrutura foi a falta de uma aproximação comum para o desenvolvimento e uso das arquitecturas do DoD. As arquitecturas que são desenvolvidas pelas várias organizações do DoD diferem significativamente no conteúdo e formato, levando ao desenvolvimento de produtos “fielded” que não são inter operáveis. Estas diferenças impedem que sejam feitas comparações e contrastes ao analisar as várias arquitecturas. A Estrutura C4ISR tem como função fornecer uma “*língua franca*” para os vários comandos, serviços e agências do DoD, para descrever as suas diversas arquitecturas.

O modelo C4ISR fornece direcções sobre a forma como descrever uma arquitectura; não fornece orientação sobre como desenhar ou implementar uma arquitectura específica ou como desenvolver e obter sistemas-de-sistemas. Esta estrutura diferencia a descrição da implementação da arquitectura. A descrição de uma arquitectura é a representação das partes, o que essas partes fazem, como é que elas se relacionam umas com as outras, e sob que regras e restrições funcionam. A Estrutura C4ISR tem interesse apenas nesta representação, não com a implementação. Esta é a maior diferença entre a Estrutura C4ISR e as outras abordagens aqui apresentadas.

Os componentes mais importantes da Estrutura C4ISR são os seguintes:

- Definição de vistas arquitecturais comuns
- Guia para desenvolvimento das arquitecturas
- Definição de produtos comuns
- Recursos de referência relevantes.

Anexo D

O interesse aqui será dado apenas ás vistas e produtos comuns apresentadas por esta ferramenta.

Vistas Arquitecturais da Estrutura C4ISR

A C4ISR define três perspectivas ou vistas principais, que são logicamente combinadas para descrever uma arquitectura. Estas três vistas são a vista operacional, a vista sistema e a vista técnica.
Cada uma destas vistas arquitecturais tem implicações sobre as características da arquitectura que serão consideradas e/ou mostradas, apesar de, geralmente, existir entre as vistas algum grau de redundância nas características que são apresentadas.
Porque cada vista fornece diferentes perspectivas da mesma arquitectura, é esperado que, na maioria dos casos, a descrição mais útil da arquitectura será uma descrição integrada, isto é, uma descrição que consista em múltiplas vistas. Comparado com a descrição de uma vista, a descrição integrada de uma arquitectura fornece uma relação mais próxima e informação mais útil para o processo de planeamento, programação e aquisição.

Definição da Vista Operacional

A vista operacional descreve (usualmente através de gráficos) as actividades, os elementos operacionais, e os fluxos de informação necessários para alcançar ou suportar as operações. Define os tipos de informação trocada, a frequência da troca de informação, que tarefas e actividades são suportadas pela troca de informação e a natureza da informação trocada em detalhe suficiente de forma a verificar requisitos de interoperabilidade específicos.

Definição da Vista Sistema

A vista sistema descreve como é que os vários sistemas se relacionam e inter operam, e deve descrever a construção interna e as operações de um sistema em particular da arquitectura. Para um sistema individual, a vista de sistema inclui a conexão física, a localização, a identificação dos nós chave, dos circuitos, das redes, etc., e especifica os parâmetros de desempenho do sistema e dos componentes.
A vista sistema associa os recursos físicos e os seus atributos de desempenho á vista operacional e os seus requisitos de acordo com os critérios definidos na vista técnica.

Definição da Vista Técnica

A vista técnica é um conjunto mínimo de regras que comandam as interacções, os arranjos e as interdependências entre as partes do sistema. Esta vista fornece as linhas de orientação técnicas para a implementação dos sistemas, sobre as quais as especificações de engenharia são baseadas, os blocos de construção são estabelecidos e as linhas de produtos são desenvolvidas. A vista técnica inclui uma colecção de padrões, convenções, regras e critérios técnicos, organizados em perfis que comandam os serviços de sistema, as interfaces e as relações para as vistas de sistema em particular e que se relacionam com as vistas operacionais.

Anexo D

Notação - Produtos

Os produtos arquitecturais são os itens gráficos, textuais e tabulares que são desenvolvidos durante a construção da descrição de uma arquitectura e que descrevem as características pertinentes do seu propósito. Quando completos, este conjunto de produtos constitui a descrição da arquitectura. Os produtos arquitecturais a serem desenvolvidos são classificados em duas categorias:

Produtos essenciais - estes produtos constituem o conjunto mínimo de produtos necessários para desenvolver arquitecturas que possam ser facilmente compreendidas e integradas dentro e através das fronteiras organizacionais do DoD e entre o DoD e elementos multinacionais. Estes produtos devem ser desenvolvidos para todas as arquitecturas.

Produtos de Suporte - estes produtos fornecem dados que serão necessários para um propósito em particular e objectivos de um esforço arquitectural especifico. Produtos apropriados deste conjunto de produtos de suporte serão desenvolvidos de acordo com os propósitos e objectivos da arquitectura.

As tabelas seguintes resumem os produtos essenciais e de suporte, respectivamente, definidos pela Estrutura C4ISR.

Architecture Product	C4ISR Architecture View
Overview and Summary Information	All views
Integrated Dictionary	All views
High-Level Operational Concept Graphic	Operational
Operational Node Connectivity Description	Operational
Operational Information Exchange Matrix	Operational
System Interface Description	Systems
Technical Architecture Profile	Technical

Tabela 15: Produtos Essenciais da Estrutura C4ISR (“ Documenting Software Architectures” [1])

Architecture Product	C4ISR Architecture View
Command Relationships Chart	Operational
Activity Model	Operational
Operational Activity Sequence and Timing Descriptions	Operational
Operational Activity Sequence and Timing Descriptions—Operational State Transition Description	Operational
Operational Activity Sequence and Timing Descriptions—Operational Event/Trace Description	Operational
Logical Data Model	Operational
Systems Communications Description	Systems
Systems^2 Matrix	Systems
Systems Functionality Description	Systems
Operational Activity to System Function Traceability Matrix	Systems
System Information Exchange Matrix	Systems
System Performance Parameters Matrix	Systems
System Evolution Description	Systems
System Technology Forecast	Systems
System Activity Sequence and Timing Descriptions	Systems
Systems Activity Sequence and Timing Descriptions—Systems Rules Model	Systems
Systems Activity Sequence and Timing Descriptions—Systems State Transition Description	Systems
Systems Activity Sequence and Timing Descriptions—Systems Event/Trace Description	Systems
Physical Data Model	Systems
Standards Technology Forecast	Technical

Tabela 16: Produtos de Suporte da Estrutura C4ISR (“ Documenting Software Architectures” [1])

È importante referir que o C4ISR está quase exclusivamente dedicado á documentação da arquitectura do sistema. Nenhuma das suas vistas e nenhum dos seus produtos essenciais ou de suporte prescreve qualquer coisa que se assemelhe remotamente à arquitectura do software. A arquitectura operacional refere-se em termos das operações visíveis ao utilizador. A arquitectura de sistema direcciona-se a forma como essa operações são realizadas nos recursos físicos, isto é, hardware. E a arquitectura técnica impõe regras e restrições, o que na prática significa apenas a escolha de um conjunto de interfaces padrão. O software do sistema não é descrito de forma alguma, no que diz respeito á sua estrutura, comportamento e interrelacionamento dos elementos da sua estrutura.

4. RM-ODP

O RM-ODP (sigla de Reference Model of Open Distributed Processing) da ISO foi criado para servir como modelo de referência para a descrição de sistemas distribuídos abertos e é aceite actualmente como um padrão de facto. Estão definidos cinco vistas neste modelo e devem ser seguidas para que o desenvolvimento de um sistema de informações seja compatível com este padrão (Figura 48), de seguida são apresentadas resumidamente essas cinco vistas.

Figura 48: Vistas arquitecturais do RM-ODP

Vista Organizacional ou Empresa (*Enterprise Viewpoint*): 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;

Vista de Informações (*Information Viewpoint*): descreve o sistema de informações em termos de estruturas de informações, fluxo de informações e restrições relacionados com a manipulação das mesmas;

Vista Computacional (*Computational Viewpoint*): descreve o sistema de informações em termos de operações e características computacionais do processo de mudança de informações;

Vista Tecnológica (*Technological Viewpoint*): descreve o sistema de informações em termos dos componentes que o sistema é construído, e

Vista de Engenharia (*Engineering Viewpoint*): descreve o sistema de informações em termos de recursos de engenharia para suportar a natureza distribuída do processamento.

Apesar do RM-ODP apresentar uma descrição essencial dos conteúdos de cada vista, que deve ser considerada e assim facilitar a comparação entre sistemas alternativos, aderir ao mesmo não resolverá todas as questões práticas envolvidas no projecto e implementação de sistemas de *software*. O RM-ODP não determina como o sistema de informações deve ser projectado e implementado, bem como não define quais as ferramentas ou técnicas que devem ser utilizadas para a representação de cada vista. Tal tarefa é deixada a critério do utilizador. Por esta razão, além do RM-ODP, torna-se necessária a aplicação de métodos formais que possam guiar o projecto e implementação efectiva e consistente de sistemas de *software* através de cada uma dos seus vistas.

O modelo RM-ODP não identifica quaisquer stakeholders com as vistas que define. Este modelo refere que foi desenhado de forma a satisfazer as necessidades da equipa de desenvolvimento, e os “standards writers”. No entanto, os outros stakeholders podem ser implicitamente endereçados através das necessidades que são satisfeitas por cada uma das vistas.

5. Paul Clements et al.

De modo a trazer alguma ordem a um conjunto das possíveis vistas das várias abordagens, estas foram agrupadas, tendo em conta o tipo de informação que acarretam.

Os arquitectos desenvolvem a sua tarefa criativa através da análise do sistema de três formas distintas:

- Como é que o sistema será estruturado em termos de unidades de código?
- Como é que o sistema será estruturado em termos da interacção de elementos de execução?

- Como é que o sistema se irá relacionar com as estruturas de não software?

Pensar na vistas através destas três categorias ajuda os arquitectos a pensar nos sistemas em termos bem estruturados, e ajuda os consumidores da documentação a distinguir de entre os conceitos separados que a arquitectura apresenta.

Tipos de vista

Estas categorias são chamadas de tipos de vista – *viewtypes*- e são distinguidas pelos seguintes nomes: Módulo, Componente & Conector, Afectação.
As vistas do tipo de vista Módulo, documentam as principais unidades de código. As vistas do tipo de vista Componente & Conector documentam a interacção dos elementos de execução do sistema. E as vistas do tipo de vista Afectação documentam as relações entre o software do sistema e os ambientes de desenvolvimento e de execução. De seguida 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. Todas as figuras e exemplos são retirados do livro *"Documenting software Architecture" [1]*. Os exemplos são essencialmente sobre o sistema ECS e arquitectura de Software A-7E que são usados como casos de estudo no mesmo livro.

Tipo de Vista Módulo

Este tipo de vista consiste na decomposição do sistema em módulos, ou seja, os seus elementos são módulos, que por sua vez são unidades de implementação. Os módulos representam uma forma de considerar o sistema baseado no código.
Os módulos são áreas de responsabilidade funcional, e são atribuídos ás equipas de implementação. Não é dado muito ênfase ao desempenho do software durante a execução.

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. Através da especialização da relação entre módulos *"pode usar"* e através da imposição de uma ordem estrita na relação, obtém-se o estilo **camadas**, onde cada camada representa uma máquina virtual. Quando a relação "*depende-de"* é especializada na *utilização,* surge o estilo **utilização**. Um arquitecto pode utilizar este estilo para restringir ou delimitar a implementação da arquitectura. Este estilo informa a equipa de desenvolvimento que outros módulos devem existir para que a sua porção do sistema funcione. Ao tirarmos os elementos e propriedades do tipo de vista Módulo e focando na relação "*é-parte-de"*, obtemos o estilo **Decomposição**, que permite mostrar como as responsabilidades do sistema estão repartidas ao longo dos módulos e como estes módulos são decompostos em sub-módulos. A aplicação da relação *"é uma"* e outras restrições origina o estilo **Generalização**, que é a base para relações de herança nos sistemas orientados a objectos; relaciona módulos mostrando como um é a generalização ou especialização do outro.

Anexo D

Tipo de Vista Componente & Conector

As vistas C&C definem modelos que consistem em elementos que existem durante a execução, como por exemplo os processos, os objectos, os clientes, os servidores. Adicionalmente esses modelos incluem também como elementos as interacções, como por exemplo as ligações de comunicação, os protocolos, os fluxos de informação. Resumindo, 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).

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 muitos estilos são bem conhecidos. Através da restrição dos componentes que interagem via um conector cliente – servidor do tipo pedido-resposta, e da restrição dos caminhos de comunicação entre os elementos, emerge o estilo ***cliente-servidor***. Restringindo os componentes de modo a funcionarem como repositórios de dados e os conectores a fornecer os mecanismos de comunicação necessários para aceder (para leitura e escrita) dos dados, obtém-se o estilo ***dados-partilhados.*** Outros estilos C&C bem conhecidos incluem ***canais-e-filtros***, ***publicação-subscrição***, ***Par-a-Par*** e ***Processos-Comunicantes.***

Qualquer componente ou conector pode ser documentado em diversas vistas C&C usando diversos estilos, visto que o mesmo pode interagir com outros componentes e conectores de várias formas. Por exemplo, um componente pode ser um servidor num sistema cliente-servidor; mas o mesmo componente pode também actuar como um repositório para dados. Assim, tendo em conta que tanto o estilo cliente-servidor como o estilo dados-partilhados são importantes para perceber o sistema, a documentação das vistas nos dois estilos vai incluir o mesmo componente, mostrando diferentes padrões de interacção.

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. A relação na vista afectação é *afectado-a* que representa a forma como um elemento de software é afectado a um elemento do contexto. Um elemento de software pode ser *afectado-a* múltiplos elementos do contexto e múltiplos elementos de software podem ser *afectados-a* um único elemento do contexto. As afectações podem mudar ao longo do tempo.

O tipo de vista afectação apresenta o mapeamento dos elementos de qualquer um dos estilos dos tipos de vista Módulo ou componente & conector em elementos do ambiente de execução. Assim, são identificados três estilos mais comuns: o estilo **instalação** que descreve o mapeamento entre os componentes e os conectores no hardware onde é executado o software; o estilo **implementação** que descreve o mapeamento dos módulos

num ficheiro de sistema que contém esses módulos; e o estilo **trabalho** que descreve o mapeamento dos módulos nas pessoas, grupos ou equipas que têm como tarefa o desenvolvimento dos módulos.

Estilo Instalação

No estilo instalação, os elementos da vista Componente&Conector, são afectados á plataforma de execução. As restrições para cada afectação têm a ver com os requisitos expressados pelos elementos de software e a forma como esses requisitos são satisfeitos pelas características dos elementos de hardware.

Estilo de Implementação

O estilo de Implementação mapeia os módulos do tipo de vista Módulo com a infra-estrutura de desenvolvimento. A implementação de um módulo resulta sempre num conjunto de ficheiros separados, tais como os ficheiros que contém o código fonte, os ficheiros que têm que ser incluídos e usualmente contém também definições, ficheiros que descrevem como construir os executáveis, e ficheiros que são o resultado de traduções, ou compilações do módulo. Esses ficheiros necessitam ser organizados de forma a não se perder o controlo e integridade do sistema.

Estilo Trabalho

Não existem notações especiais para o estilo arquitectural Trabalho. Normalmente essa notação é feita através de uma tabela que relaciona as pessoas ou equipas com o módulo que lhe foi atribuído para, por exemplo, desenvolver, testar ou analisar.

6. Relação entre as abordagens

As várias abordagens podem ser comparadas através da cobertura que dão a alguns conceitos. A cobertura é uma medida da extensão a que os elementos de um domínio são dirigidos. Cada um dos modelos apresentados pode ser avaliado através da cobertura que dá aos satkeholders (por exemplo, arquitectos, equipa de implementação, gestores de projecto), aos interesses (por exemplo, desempenho, implementação, privacidade) e ás estruturas da arquitectura (por exemplo, decomposição, camadas, processos).
Os modelos que têm uma maior cobertura no que diz respeito aos stakeholders são o modelo “4+1 de Kruchten” e o modelo de Paul Clements et al. Os modelos RM-ODP e Siemens têm uma pequena cobertura sobre os stakeholders. O foco do RM-ODP é em definir uma linguagem para a arquitectura e não em definir os stakeholders. O modelo Siemens foca-se no desenho da arquitectura de software, e então as suas vistas são tratadas da perspectiva do arquitecto de software. Nenhum destes modelos relaciona explicitamente as suas vistas com os stakeholders, no entanto, estas satisfazem implicitamente as necessidades dos stakeholders através dos interesses a que as suas vistas se dirigem.
A cobertura de interesses (ver gráfico 1) e estruturas (ver gráfico 2) mostra uma variação inferior á dos stakeholders porque todos os modelos reconhecem que a arquitectura de software é composta por estruturas e restrita por interesses. Como resultado, cada modelo inclui vistas suficientes para descrever uma arquitectura de software completa.

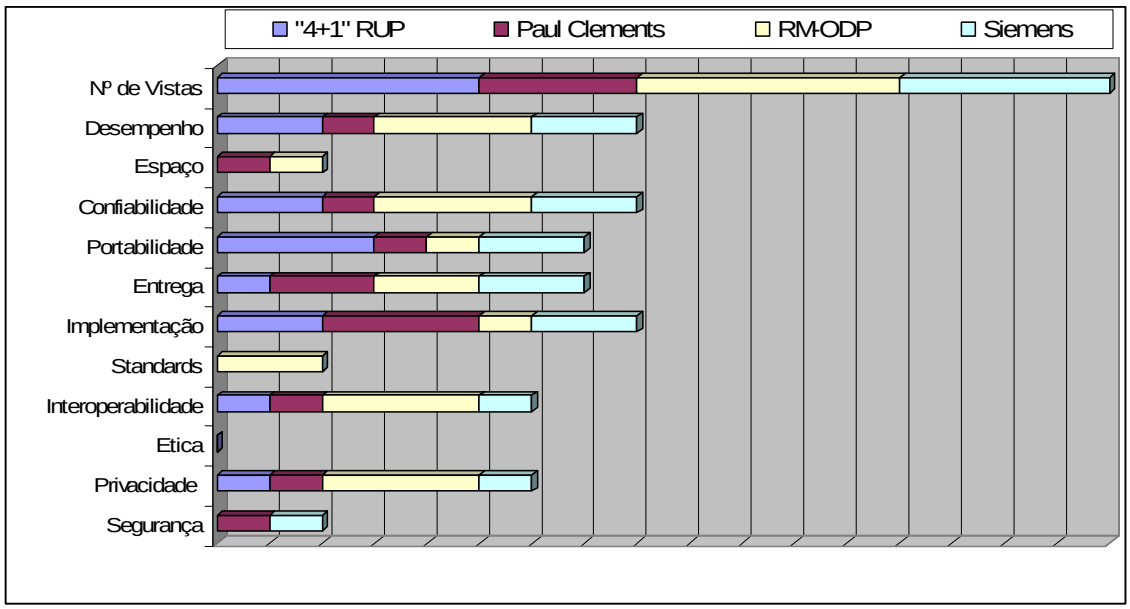


Gráfico 7: Cobertura dos vários modelos em termos de necessidades

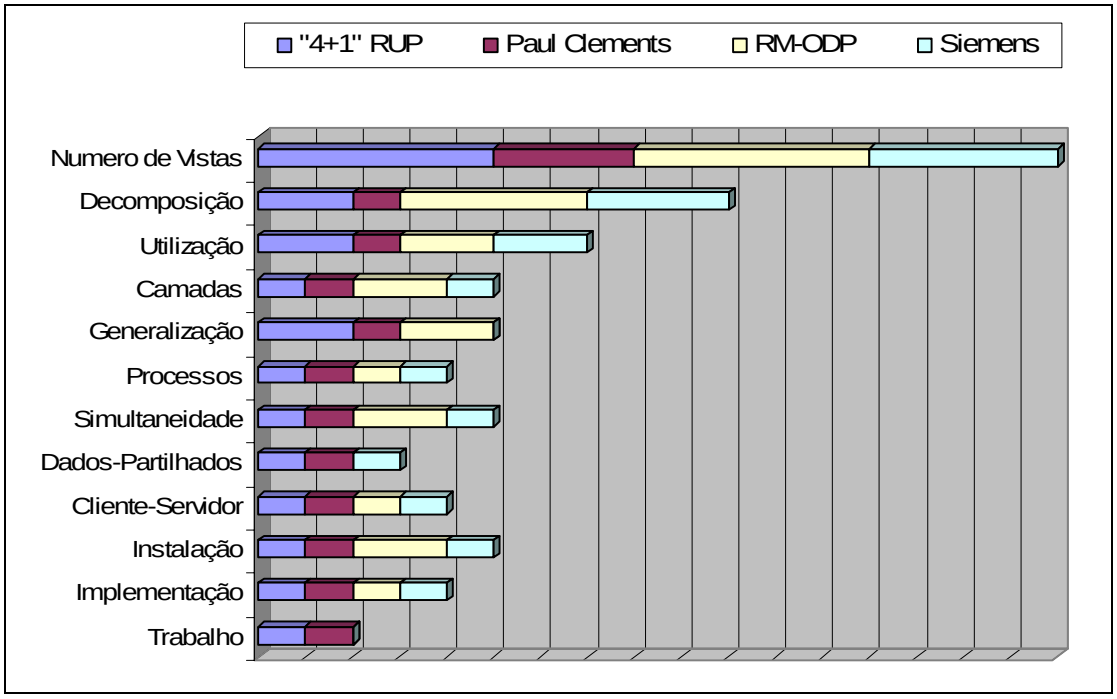


Gráfico 8: Cobertura dos vários modelos em termos de estruturas

Correspondência entre as vistas

Existe alguma correspondência entre as vistas dos diferentes modelos, e é mais notável na distribuição das estruturas endereçadas pelas vistas. A tabela seguinte ilustra essa correspondência (mais precisamente, entre os estilos das vistas do modelo de Paul

Clements e tal. e os outros modelos), segundo o livro "*Documenting Software Architectures*" [1].

Paul Clements et al		RUP	Siemens	C4ISR	RM-ODP
Módulo	**Decomposição**	Lógica, Desenvolvimento	Módulo	Técnica	Informação, Computacional
	Utilização	Lógica, Desenvolvimento			
	Camadas				
	Generalização	Lógica, Desenvolvimento			
C&C		Lógica, Processos	Conceptual, Execução	Técnica	Engenharia
Afectação	**Instalação**	Física	Execução	Sistema	
	Implementação		Código		Tecnologia
	Trabalho				
		Cenários		Operacional	Organizacional

Tabela 17: Relação entre as várias abordagens de vistas arquitecturais

Divergência dos Modelos

Os modelos variam ente si em focos distintos, que reflectem os seus principais propósitos. O modelo de Paul Clements et al. está orientado para a comunicação, os modelos "4+1 de Kruchten" e Siemens estão orientados ao desenho, o modelo RM-ODP é orientado á reutilização, e o C4ISR está orientado á comunicação, mas de alto nível de abstracção.

O modelo de Paul Clements et al. consiste em vistas relativamente independentes, comparadas com os outros modelos. Isto permite que os pacotes de documentação sejam criados independentemente para os diferentes stakeholders. Os modelos "4+1 de Kruchten" e Siemens incluem fluxos de informação entre as vistas, suportando assim as aproximações de desenho sugeridas por esses modelos.

O RM-ODP dirige-se especificamente à estrutura da informação dentro do sistema e das políticas de negócio relevantes ao sistema. A inclusão destas duas vistas mostra a grande abrangência de descrição que esta vista provê. A profundidade da linguagem das vistas permite que este modelo seja usado como um dicionário padrão para descrever as arquitecturas de software em detalhe.

As três vistas do modelo C4ISR são totalmente dependentes, visto que a vista sistema associa os recursos físicos e os seus atributos de desempenho á vista operacional e os seus requisitos de acordo com os critérios definidos na vista técnica.

7. Escolha das vistas

Para que o desenvolvimento de um projecto seja bem sucedido, apenas podemos determinar que vistas são necessárias, quando cria-las e o nível de detalhe para a descrição das mesmas, se soubermos:

- Quais as competências das pessoas disponíveis
- Qual o orçamento disponível
- Qual é a calendarização
- Quais são os *stakeholders* mais importantes

De forma a escolher o conjunto de vistas apropriado para um projecto, é necessário identificar os *stakeholders* que dependem da documentação da arquitectura de software. È também importante perceber as necessidades de informação de cada *stakeholder*. O número de stakeholders varia, dependendo do tipo de organização e do tipo de projecto.

De seguida são apresentados alguns dos stakeholders que têm necessidades de informação mais específicas e que são os mais comuns no processo de desenvolvimento de um projecto de software. A tabela 17 mostra a forma como os vários stakeholders se relacionam com as várias vistas dos modelos apresentados.

Gestor de Projecto

O gestor de projecto necessita de informação sobre a calendarização, atribuição de recursos, e planos de contingência de forma a libertar algum subconjunto do sistema por motivos de negócio. Este stakeholder não está interessado nos detalhes de desenho de nenhum dos elementos nem nas especificações das interfaces apenas tem interesse em saber se essas tarefas estão completas. O gestor está também interessado nos objectivos e restrições gerais do sistema; a sua interacção com outros sistemas, o que pode sugerir uma interface entre organizações que o gerente terá que estabelecer; e o ambiente de hardware, que o gestor deverá obter.

Cliente

Os clientes são os stakeholders que pagam o desenvolvimento de projectos especialmente feitos para eles. Os clientes estão interessados no custo e progresso do desenvolvimento do sistema, bem como em argumentos convincentes de que a arquitectura e o sistema resultante vão de encontro aos requisitos de qualidade e funcionais. Os clientes têm que suportar o ambiente onde o sistema irá ser executado, e tem também interesse em saber se o sistema ira inter operar com os outros sistemas desse ambiente. O cliente tem interesse nos resultados das análises feitas ao sistema.

Anexo D

Equipa de Desenvolvimento

Um membro da equipa de desenvolvimento, para quem a arquitectura fornece as "ordens de marcha", recebe orientações sobre como fazer o seu trabalho. Um membro da equipa de desenvolvimento necessita informação sobre:

- A ideia geral (objectivos e missão) sobre sistema. Esta informação pode ser-lhe dada através de um diagrama de contexto de alto nível de abstracção.
- Que elementos lhe foram atribuídos para desenvolver, isto é, onde é que as funcionalidades serão implementadas.
- Os detalhes desse elemento e as suas propriedades.
- Os elementos com os quais a parte que lhe foi atribuída se relaciona, e que relações são essas.
- Os recursos em termos de código que pode utilizar.
- As restrições têm de ser tidas em conta, tais como atributos de qualidade, o orçamento.

Equipa de Instalação

Um membro da equipa de instalação, tem interesse em saber a forma como os componentes e conectores se ligam no hardware onde o software é executado. Este tem como função o carregamento e instalação do sistema no lado do cliente.
Após identificar os stakeholders e as necessidades de informação de cada um, o próximo passo é então escolher as vistas que melhor satisfaçam essas necessidades em termos de informação.

	Paul Clements et al.	**RUP**	**Siemens**	**C4ISR**	**RM-ODP**
Gestor de Projectos	Afectação	Desenvolvimento	Módulo Execução	Operacional Técnica	Organizacional Tecnológica
Cliente	Afectação C&C	Lógica	Execução Conceptual	Operacional	Organizacional Tecnológica
Equipa de Desenvolvimento	Módulo C&C Afectação	Física Processos	Módulo Código Conceptual	Sistema Técnica	Computacional Informação Engenharia Tecnológica
Equipa de Instalação	Afectação	Lógica Física	Execução Código	Sistema Técnica	Tecnológica

Tabela 18: Relação entre as várias perspectivas arquitecturais e os interesses dos Stakeholders

Anexo D

Conclusão

O resultado ideal para a escolha das vistas seria determinar o conjunto mínimo de vistas, de entre os vários modelos, com a maior cobertura dos conceitos analisados anteriormente – stakeholders, estruturas e interesses. Uma aproximação a esse resultado, seria determinar quais as vistas dos diferentes modelos que podem ser combinadas num novo conjunto, e identificar o conjunto mais pequeno que forneça a máxima cobertura.
A maior barreira para combinar vistas dos diferentes modelos é a inter relação que existe entre as vistas do mesmo modelo. O modelo "4+1 de kruchten" inclui um grande fluxo de dados de uma vista para a outra, começando pelos cenários, como parte de uma aproximação iterativa. Existe um fluxo de dados semelhante entre as vistas do modelo Siemens, apesar de não serem tão fortemente acoplados. O modelo C4ISR também apresenta uma forte dependência entre as vistas que o compõem. As vistas dos modelos de Paul Clements et al. e RM-ODP são relativamente independentes, e definir todas as traduções entre as vistas é adiado para o nível da descrição arquitectural. Como consequência, as vistas destes dois últimos modelos são as melhores candidatas para serem combinadas.
Para fornecer um conjunto mínimo de vistas, é necessário ter em conta a sobreposição em termos de cobertura entre as vistas. Inicialmente deve ser seleccionado um modelo base e depois então adicionar-lhe vistas que cubram os conceitos que faltam. A melhor opção para um modelo base será o modelo de Paul Clements et al. porque este contém vistas independentes e porque sobrepõe perfeitamente os modelos "4+1 de Kruchten", Siemens e C4ISR. O modelo de Paul Clements et al. fornece uma cobertura completa das estruturas, mas, por outro lado, não satisfaz as necessidades dos stakeholders utilizadores finais nem dos "standard writers". Esta lacuna pode ser satisfeita através da inclusão da vista organizacional do RM-ODP. Esta é a vista de mais "alto" nível do modelo e por isso não é dependente das outras vistas. O resultado será um conjunto de vistas com a máxima cobertura dos conceitos, composto por:

- Vista Organizacional, do RM-DP
- Vista Módulo, do modelo Paul Clements et al.
- Vista C&C, do modelo Paul Clements et al.
- Vista Afectação, do modelo Paul Clements et al.

Este conjunto não será o ideal para todos os casos, no entanto tendo em conta o estudo e comparação entre as abordagens (em termos da cobertura de interesses, stakeholders e estruturas) será de longe o mais adequado.

Anexo E - Documentação de Sistema Informáticos

Nome da Empresa:	
Dados Pessoais:	
Nome:	
Função:	

Este inquérito surge no âmbito de um projecto de fim de curso da Universidade da Madeira, projecto esse, que consiste na documentação da arquitectura de software de um sistema Informático. As suas respostas serão muito importantes para avaliar a realidade das empresas de desenvolvimento de software, no que diz respeito as práticas de documentação. Agradeço desde já a sua preciosa colaboração.

1. O(s) sistema(s) que desenvolve possui um pacote de documentação técnica?

   ☐Sim ☐Não

2. Caso exista um pacote de documentação:

   O pacote de documentação segue alguma metodologia específica?

   ☐ as 4+1 vistas da RUP ☐ as 5 vistas RM-ODP
   ☐ as 4 vistas da Siemens ☐ ANSI/IEEE-1471-2000
   ☐ PaulClement et al. ☐Outro
   ☐Nenhum

   Qual o formato dos ficheiros que compõem o pacote de documentação?

   ☐ Html ☐Word
   ☐UML, Integrado com IDE ☐Outro

   Com que frequência costuma consultar esse pacote de documentação?

   Todos os dias

   Classifique o pacote de documentação consoante os seguintes parâmetros:

	Mau	Satisfatório	Bom	Muito Bom
Fiabilidade	☐	☐	☐	☐
Navegabilidade	☐	☐	☐	☐
Facilidade de actualização	☐	☐	☐	☐
Organização da Informação	☐	☐	☐	☐
Grau de Actualização	☐	☐	☐	☐
Conteúdo	☐	☐	☐	☐

   Na sua opinião, quais as principais desvantagens, desse pacote de documentação?

Quem é o responsável por actualizar a documentação?

3 Na sua opinião, quais os requisitos mais importantes num pacote de documentação?

4 Em que medida um bom pacote de documentação pode influenciar o seu trabalho?

5 Qual o seu interesse em conhecer a proposta de metodologia para documentação de uma arquitectura de software desenvolvida neste projecto?

☐Muito ☐Pouco ☐Nenhum