

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**APLICAÇÕES DE ESTILOS ARQUITETURAIS NO DESENVOLVIMENTO DE
PROJETOS DE SOFTWARE**

Rodrigo Carlo Gurgel Martiniano

**NATAL/RN
2003**

RODRIGO CARLO GURGEL MARTINIANO

**APLICAÇÕES DE ESTILOS ARQUITETURAIS NO DESENVOLVIMENTO DE
PROJETOS DE SOFTWARE**

Monografia apresentada ao curso de graduação em Bacharelado em Sistemas de Informação da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Orientadora:

Bartira Paraguaçu F. Dantas Rocha, M.Sc.

**NATAL/RN
2003**

Martiniano, Rodrigo Carlo Gurgel
Aplicação de estilos arquiteturais no desenvolvimento de projetos de software/
Rodrigo Carlo Gurgel Martiniano – Natal: FARN, 2003.

49p.

Monografia (Graduação em Bel. Sistemas de Informação) – Faculdade Natalense
Para O Desenvolvimento do Rio Grande do Norte.

Orientador: Bartira Paraguaçu F. Dantas Rocha(Msc)

1. Engenharia de Software – Monografia 2. Estilos Arquiteturais de Software –
Monografia 1. Título.

RN/FARN/BC

CDU 004.41



**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

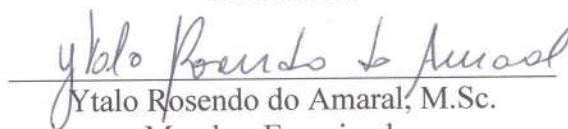
**APLICAÇÕES DE ESTILOS ARQUITETURAIS NO DESENVOLVIMENTO DE
PROJETOS DE SOFTWARE**

Rodrigo Carlo Gurgel Martiniano

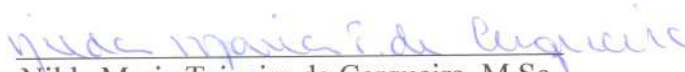
Monografia apresentada e aprovada em ____ de ____ de ____, pela banca examinadora composta pelos seguintes membros.



Bartira Paraguaçu F. Dantas Rocha, M.Sc.
Orientadora



Ytalo Rosendo do Amaral, M.Sc.
Membro Examinador



Nilda Maria Teixeira de Cerqueira, M.Sc.
Membro Examinador

Natal, ____ de ____ de ____

A DEUS por ser a razão de toda a minha vida e por estar sempre presente em todos os meus momentos.

Aos meus pais dedico este trabalho, como resultado de meu esforço e também do apoio que deles recebi desde de meus primeiros passos de vida e em todos os meus projetos.

A minha namorada, Alessandra, pelo apoio constante.

AGRADECIMENTOS

Este trabalho foi construído com o apoio de várias pessoas e instituições. Por isso, venho agradecer a todos com as mais sinceras palavras, esperando atingido os fins pretendidos neste empenho conjunto e atendido as expectativas de todos que contribuíram para sua construção.

Assim, agradeço a todos os meus professores, pelos ensinamentos ministrados e pelo estímulo aos desafios profissionais e acadêmicos que deles recebi.

Aos meus professores revisores e orientadores, em especial a Prof. Bartira Paraguaçu F. Dantas Rocha, que incondicionalmente me distinguiu com sua atenção, inteligência e soube perceber os erros e acertos deste trabalho, fazendo-o sempre melhor.

À Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, particularmente nas pessoas de seus Diretores, Professores e Funcionários, pela presteza e suporte que me ofereceram para a realização e conclusão desta oportunidade produtiva e valorosa que é a absorção do conhecimento.

E a todos que, de alguma forma, contribuíram na elaboração deste trabalho.

SUMÁRIO

Capítulo 1 – INTRODUÇÃO	10
1.1 – APRESENTAÇÃO	10
1.2 – PROBLEMA DE PESQUISA	11
1.3– OBJETIVOS	11
1.3.1 – Objetivo Geral.....	11
1.3.2 – Objetivos Específicos	11
1.4 – JUSTIFICATIVA	12
1.5 – METODOLOGIA	12
1.6 – ORGANIZAÇÃO DO TRABALHO.....	14
Capítulo 2 – FUNDAMENTAÇÃO TEÓRICA	14
2.1 – ENGENHARIA DE SOFTWARE	15
2.2 – ARQUITETURA DE SOFTWARE	19
Capítulo 3 – ESTILOS ARQUITETURAIS	19
3.1 – INTRODUÇÃO	24
3.3 – ARQUITETURAS DE FLUXO DE DADOS	25
3.4 – ARQUITETURAS DE MÁQUINAS VIRTUAIS	26
3.5 – ARQUITETURAS DE CHAMADA-E-RESPOSTA.....	30
3.6 – ARQUITETURAS DE COMPONENTES INDEPENDENTES	31
3.7 – ESTILOS HETEROGÊNEOS	33
3.8 - ORGANIZANDO ESTILOS ARQUITETURAIS.....	33
3.9 – TIPOS DE CARACTERÍSTICAS.....	34
3.9.1 - Partes de um estilo: Componentes e Conectores.	34
3.9.2 - Questões de Controle.	35
3.9.3 - Questões de dados.	36
3.9.4 - Questões de interação de Dados/ Controle.....	36
3.9.5 - Tipos de Computação.	36
3.10 – Usando estilos em projetos de sistemas.....	42
Capítulo 4 – ESTUDO DE CASO	46
Capítulo 5 – ANÁLISE DOS RESULTADOS	47
Capítulo 6 – CONCLUSÃO	48
REFERENCIAS BIBLIOGRÁFICAS	49
BIBLIOGRAFIA	

LISTA DE FIGURA

FIGURA 1: Catálogos de estilos.....	22
FIGURA 2: Estilos Centrados em Dados.....	24
FIGURA 3: Estilos Seqüenciais.....	25
FIGURA 4: Estilos de Máquinas Virtuais.....	26
FIGURA 5: Estilos de Programa Principal e Sub-Rotinas.....	28
FIGURA 7: Estilo em Camadas.....	29
FIGURA 6: Estilo Orientado a Objeto.....	29
FIGURA 8: Heterogeneidade Hierárquica.....	32
FIGURA 9: Visão Geral da Arquitetura- <i>Cliente/servidor</i>	44
FIGURA 10: Arquitetura de Chamada e Resposta- <i>Orientadoa Objetos</i>	44
FIGURA 11: Arquitetura de Chamada e Resposta- <i>Orientadoa Objetos</i>	45

LISTA DE TABELAS

TABELA 1: Notação de Componentes e Conectores usada em Arquiteturas de Software	18
TABELA 2: Classificação de Estilos Arquiteturais - Fluxo de Dados	38
TABELA 3: Classificação de Estilos Arquiteturais - Chamada e Resposta	39
TABELA 4: Classificação de Estilos Arquiteturais - Componentes Independentes	40
TABELA 5: Classificação de Estilos Arquiteturais - Centrados em Dados	41
TABELA 6: Máquinas Virtuais	41

RESUMO

Ao longo das últimas décadas o software teve grande evolução, tornando-se parte essencial nos sistemas de computação, sendo visto como um elemento chave na concepção de sistemas. Tão grande foi a evolução dos sistemas computacionais que as técnicas, métodos, ferramentas e princípios da engenharia que foram adotados pela engenharia de software para tornar os softwares cada vez mais baratos e corretos, estão se tornando obsoletos, pois, com a evolução dos softwares vieram também fatores de preocupação para os desenvolvedores, como desempenho, qualidade, confiabilidade, manutenção e economia.

Tais, fatores estão associados à organização global dos sistemas de softwares, ou seja, desenvolvimento de software orientado para arquitetura – *Arquitetura de Software*.

Segundo Mendes (2002) a Arquitetura de Software procura analisar as propriedades dos softwares no nível de subsistema ou módulo, atuando como uma estrutura a fim de atender os requisitos de sistemas, além de ser utilizada como aspecto técnico para projetos de sistemas, bem como suporte na estimação de custos e na gerência do processo. Também serve de base para análise da consistência fornecendo suporte ao reuso.

Este trabalho tem como objetivo apresentar estilos de arquiteturas de software desenvolvidos e utilizados na construção de diversos tipos de arquiteturas de software, procurando deixar claro qual o estilo mais adequado à uma arquitetura específica. Será apresentado também um estudo de caso onde é especificado e desenvolvido a arquitetura de um software de comércio eletrônico através de estilos que mais se adequam ao sistema.

Palavras Chaves: Software, Sistema, Engenharia de Software, Arquitetura de Software, Estilo.

ABSTRACT

Along the last decades the software had great evolution, becoming leaves essential in the computation systems, being seen as a key element in the conception of systems. Such it was the evolution of the systems computacionais that the techniques, methods, tools and beginnings of the engineering that were adopted more and more by the software engineering to turn the softwares cheap and correct, they are becoming obsolete, because, with the evolution of the softwares they also came concern factors for the developers, as acting, quality, reliability, maintenance and economy.

Such, factors are associated to the global organization of the systems of softwares, in other words, development of oriented software for architecture.

According to Mendes (2002) the Architecture of Software tries to analyze the properties of the softwares in the subsystem level or module, acting as a structure in order to assist the requirements of systems, besides being used as technical aspect for projects of systems, as well as it supports in the estimate of costs and in the management of the process.

This work has as objective presents styles of software architectures developed and used in the construction of several types of software architectures, trying to leave clear which the most appropriate style to a specific architecture. It will also be presented a study of case of a system where it is specified and developed the architecture through styles that more they are adapted to the system.

Key words: Software, System, Software engineering, Architecture of Software, Style.

Capítulo 1 – INTRODUÇÃO

1.1 – APRESENTAÇÃO

Ao longo das últimas décadas o software teve grande evolução, tornando-se parte essencial nos sistemas de computação, sendo visto como um elemento chave na concepção de sistemas.

Contudo, ao mesmo tempo em que os sistemas computacionais tornam-se onipresentes em inúmeras aplicações, o tamanho e a complexidade do software tem crescido tanto que as técnicas utilizadas já não são suficientes para lidar com projetos de software com o nível de detalhes inerentes ao sistema. O crescimento em escala dos sistemas de software traz fatores de preocupação para seus desenvolvedores, como desempenho, qualidade, confiabilidade e manutenção. A engenharia de software apresenta diversas técnicas para construção de sistemas computacionais de uma forma segura e clara.

De acordo com Mendes (2002) a Engenharia de Software busca usar os princípios de engenharia com o fim de produzir a baixo custo software que funcionem de forma correta e eficaz. A necessidade de trabalhar com sistemas de software cada vez maiores e mais complexos trouxe a técnica de abstração, que é utilizada com o objetivo de reconhecer padrões, analisar e especificar sistemas. Todavia, o crescimento contínuo do tamanho dos sistemas de software juntamente com o aumento da complexidade tem requerido mais confiabilidade, economia e desempenho. Tais requisitos estão associados à organização global dos sistemas de software, ou seja, desenvolvimento de software orientado para arquitetura – *Arquitetura de Software*.

Segundo Mendes (2002) a Arquitetura de Software procura analisar as propriedades dos softwares no nível de subsistema ou módulo, atuando como uma estrutura a fim de atender os requisitos de sistemas, além de ser utilizada como aspecto técnico para projetos de sistemas, bem como suporte na estimação de custos e na gerência do processo. Também serve de base para análise da consistência fornecendo suporte ao reuso.

Um dos principais aspectos do projeto arquitetural é o uso dos padrões de organização de sistemas (padrões de arquitetura de software) ou *estilos arquiteturais*. Tais padrões buscam desenvolver projetos seguindo tipos definidos de arquiteturas que mais se adequam as características de um sistema específico.

Este trabalho tem como objetivo apresentar estilos de arquiteturas de software desenvolvidos e utilizados na construção de diversos tipos de arquiteturas de software, procurando deixar claro qual o estilo mais adequado a uma arquitetura específica. Será apresentado também um estudo de caso onde é especificada e desenvolvida a arquitetura de um software de comércio eletrônico através de estilos que mais se adequam ao sistema..

1.2 – PROBLEMA DE PESQUISA

Porque utilizar *Arquiteturas de Software* na criação de sistemas computacionais? Qual importância da utilização de *Estilos Arquiteturais* para criação de sistemas?

1.3 – OBJETIVOS

1.3.1 – Objetivo Geral

Analisar padrões de Arquitetura de Software no desenvolvimento de projetos, mostrando a importância, usando como referência estudos de casos.

1.3.2 – Objetivos Específicos

Apresentar estilos de arquitetura de software como alternativa na construção de sistemas de softwares como características bem definidas.

Justificar o uso de estilos arquiteturais através de estudos de casos.

Solucionar e aplicar um determinado estilo em um sistema de comércio eletrônico via WEB.

1.4 – JUSTIFICATIVA

Esse trabalho procura mostrar aos profissionais da área de desenvolvimento de software a importância da Arquitetura de Software para o desenvolvimento de sistemas e projetos, como também deixar claro a importância de se utilizar os estilos de Arquitetura de Software. Para isso foi realizado um estudo de caso de um sistema de comércio eletrônico, onde foi feita uma análise da Arquitetura de Software desenvolvida e suas características, para assim se chegar a determinar o grupo de Arquitetura de Software, seu estilo geral ou principal

existente nesse sistema, bem como seus subestilos. A análise foi baseada em sistemas de computação já existentes e sua aplicabilidade no sistema de comércio eletrônico via WEB.

1.5 – METODOLOGIA

Trata-se de uma pesquisa Aplicada, pois destina-se a utilizar a análise dos resultados de um estudo de caso de um sistema de comércio eletrônico via WEB, para assim mostrar a importância do uso de estilos arquiteturais. É teórica por buscar através de estudos realizados em diferentes teorias, informações a respeito de arquitetura de software, especificamente estilos arquiteturais, mostrando porque se utilizar arquitetura de software, estilos arquiteturais para projetar sistemas computacionais. Trata-se de uma pesquisa bibliográfica baseada em autores como Garlan, Shawn, Mendes, entre outros com o intuito de obter a fundamentação teórica necessária para realização de uma breve análise de estilos de arquitetura de software em fase de concepção e, estilos já consolidados. É Exploratória, pois, restringi-se a definir e analisar um estudo de caso, buscando a solução de problemas, e não há hipóteses para serem testadas.

1.6 – ORGANIZAÇÃO DO TRABALHO

Neste capítulo foi abordada uma breve apresentação sobre Arquitetura de Software, mais especificamente Estilos Arquiteturais, a colocação do problema, os objetivos a serem alcançados, a justificativa da pesquisa e a metodologia utilizada.

A estrutura dessa monografia será dividida da seguinte maneira:

- **Capítulo 2:** Neste capítulo foram abordados conceitos básicos inerentes a Engenharia de Software e Arquitetura de Software, assim como uma breve introdução aos Estilos Arquiteturais.
- **Capítulo 3:** Neste capítulo será abordado detalhadamente um catálogo contendo tipos de arquitetura de software, exemplos de estilos de arquitetura de software, bem como suas características, vantagens e desvantagens.

- **Capítulo 4:** Neste capítulo será apresentado o estudo de caso referente a um sistema de comércio eletrônico via WEB que é modelado a partir dos estilos que foram considerados mais adequados.
- **Capítulo 5:** Neste capítulo apresentaremos a análise dos resultados obtidos a partir de estudos abordados por diferentes autores e através de um estudo de caso, da arquitetura de um sistema de comércio eletrônico.
- **Capítulo 6:** Neste capítulo apresentaremos a conclusão obtida a partir da análise do estudo de caso e dos estilos apresentados.

Capítulo 2 – FUNDAMENTAÇÃO TEÓRICA

2.1 – ENGENHARIA DE SOFTWARE

Durante a década de 80, inovações e o crescimento acelerado da microeletrônica possibilitou um aumento em grande escala no poder computacional a custos cada vez menores. Porém, os softwares desenvolvidos não atendiam a requisitos, como custos que excediam o previsto, cronogramas que não eram cumpridos, desempenho, qualidade, confiabilidade e manutenção. Com o intuito de resolver e atender a esses requisitos criou-se e introduziu-se uma disciplina de desenvolvimento de software, que propunha utilizar-se de princípios da engenharia para produzir softwares a baixos custos e que funcionassem de forma correta e eficaz. Tal disciplina ficou conhecida como *Engenharia de Software*. Carvalho e Chiossi (2001).

Podemos definir Engenharia de Software como:

Uma disciplina que reúne metodologias, métodos e ferramentas a ser utilizadas, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolver problemas inerentes ao processo de desenvolvimento e ao produto de software. Carvalho e Chiossi (2001).

O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcionem eficientemente em máquinas reais. Bauer apud Pressman (1995).

A engenharia de software tem por objetivo auxiliar na produção de software de maneira que dê origem a produtos de alta qualidade, em um menor tempo e a um custo cada vez menor. A engenharia de software trata de vários problemas, onde muitos atributos devem ser considerados para que se tenha sucesso no software desenvolvido, como, a complexidade, a visibilidade, a aceitabilidade, a confiabilidade, a manutenibilidade, a segurança etc. (CARVALHO E CHIOSSI, 2001).

A engenharia de software herdou da engenharia o conceito de disciplina de produção de software, através de *metodologias*, que por sua vez seguem *métodos* que se utilizam ferramentas e princípios como, formalidade, abstração, decomposição, generalização, flexibilidade entre outros para englobar as principais atividades do processo de produção de software. Alguns métodos focalizam as funções do sistema; outros se concentram nos objetos

que povoam; outros, ainda, baseiam-se nos eventos que ocorrem durante o funcionamento do sistema (CARVALHO E CHIOSSI, 2001).

Contudo, as técnicas, métodos e ferramentas utilizadas pela Engenharia de Software para desenvolver projetos de software já não são mais suficientes para lidar com softwares cada vez mais complexos e com níveis de detalhes inerentes ao sistema.

A *Arquitetura de Software* propõe o desenvolvimento de uma estrutura arquitetural baseada em componentes e conectores que permite a especificação de propriedades relevantes ao sistema. A seguir serão apresentadas as principais definições de Arquitetura de Software com mais detalhes.

2.2 – ARQUITETURA DE SOFTWARE

Segundo Mendes (2002) *Arquitetura de Software* é uma área nova dentro da Engenharia de Software. Arquitetura de software é o estudo da organização global dos sistemas de software bem como do relacionamento entre subsistemas e componentes. Desde suas origens, quando descrições qualitativas de organização de sistemas eram consideradas úteis, a arquitetura de software tem amadurecido ao longo da última década, buscando englobar e exportar notações, ferramentas e técnicas de análise. Também se tem observado sobreposição e interação da pesquisa de arquitetura de software com trabalhos sobre reuso com base em componentes, projeto de software, classes de componentes, análise de programas e linha de produto. Embora não se considere muito produtivo tentar fazer distinções entre essas áreas, sempre que oportuno às diferenças serão salientadas.

Várias definições são dadas para arquitetura de software. Por ser uma área nova dentro da Engenharia de Software, definições são propostas e algumas delas são apresentadas a seguir.

Arquitetura de software é a estrutura dos componentes de um programa/sistema, seus inter-relacionamentos, princípios e diretrizes que guiam um projeto e suas evoluções ao longo do tempo. Garlan e Perry *apud* Bredemeyer (2003).

Arquitetura de software se refere à descrição dos elementos com os quais um sistema é construído, as interações entre estes elementos, os padrões que guiam a sua composição e as restrições nesses padrões. Garlan e Shaw (1994).

Uma arquitetura é o conjunto de decisões significativas sobre a organização de um sistema de software, a seleção dos elementos estruturais e suas interfaces, junto com o comportamento deles,

através da interação entre esses elementos. A composição destes elementos estruturais e do comportamento em subsistemas progressivamente maiores é definido como estilo arquitetural que orienta a organização de elementos e as interfaces deles, os relacionamentos entre eles, e a composição deles. Booch, Rumbaugh, e Jacobson *apud* Bredemeyer (2003).

Arquitetura de software é o estudo da ampla estrutura e desempenho de sistemas de software. Aspectos importantes da arquitetura de um sistema incluem a divisão de funções entre módulos de sistemas, os meios de comunicação entre módulos, e a representação de informação compartilhada. Pista *apud* Bredemeyer (2003).

Arquitetura de software é definida como sendo uma estrutura organizacional de um sistema de software incluindo componentes e conexões. Componentes podem ser pedaços pequenos de código, como módulos, ou pedaços maiores, tais como programas de administração de banco de dados. Conexões em uma arquitetura são abstrações para que componentes interajam em um sistema como, por exemplo, chamadas de procedimento, pipes, e chamadas de procedimento remotas. Clements *apud* Bredemeyer (2003).

Uma arquitetura de software é representada pelos seguintes conceitos: 1. Componente: Um objeto com existência independente, por exemplo, um módulo, processo, procedimento, ou variável. 2. Interface: ponto lógico de interação entre um componente e seu ambiente. 3. Conector: um objeto, que relaciona pontos de interfaces entre componentes. 4. Configuração: é uma coleção de restrições que englobam objetos dentro de uma arquitetura específica. 5. Mapeamento: relação entre os vocabulários e as fórmulas de uma arquitetura abstrata e uma arquitetura concreta. O mapeamento das fórmulas é necessário porque duas arquiteturas podem estar escritas em estilos diferentes. A semântica de componentes não é considerada parte de uma arquitetura, mas a semântica de conectores é. Moriconi *apud* Bredemeyer (2003).

Uma arquitetura de sistema está tipicamente definida no contexto das "exigências, projeto, implementação" se referindo ao último nível de estágio de projeto "onde os assuntos do projeto envolvem associação geral de capacidades de sistemas com componentes." Também designa um nível mais alto de abstração, codificação, e padronização, enquanto leva a melhoria de design de sistema e faz o sistema ser intelectualmente tratável. Para se caracterizar uma arquitetura de sistema temos os seguintes tópicos: as relações que ligam uma arquitetura de sistema ao processo de desenvolvimento correspondente; as decisões importantes a serem feitas (ao nível correspondente de abstração); as questões a serem solucionadas, as propriedades a serem garantidas; as relações para o modelo de informação empregado; e as ferramentas usadas.. O conjunto de conceitos construtivos- elementos arquiteturais – assim como notações para eles, que podem ser usados para construir a descrição de sistemas ao nível apropriado de abstração. A partir desta perspectiva é definido uma arquitetura de software como um modelo

*de projeto de sistema que captura comportamento e organização de sistema em termos de componentes e interações estáticas e configurações dinâmicas. Lawson **apud** Bredemeyer (2003).*

A Arquitetura de Software é resultado de uma série de influências exercidas pelas diversas pessoas envolvidas na concepção e utilização do sistema. Essas diversas visões, e muitas vezes divergentes, podem gerar uma série de conceitos ambíguos ou até mesmo contraditórios. Arquitetos devem identificar e ativamente comprometer os diversos envolvidos para solicitar suas necessidades e expectativas, além de gerenciar expectativas e negociar prioridades através de diplomacia, negociação e comunicação.

A criação de uma arquitetura envolve diversas atividades:

1. Entender os requisitos do sistema (Uso de técnicas como cenários, casos de uso, máquinas de estados finitos, criação e protótipos, etc)
2. Criar ou selecionar a arquitetura (Uso de estilos e Linguagens de Descrição de Arquiteturas)
3. Representar e comunicar a arquitetura
4. Analisar e avaliar a arquitetura
5. Implementar baseando-se na arquitetura
6. Manutenção

A utilização de estilos é o objeto de estudo deste trabalho, onde os principais estilos e suas principais características serão apresentados no próximo capítulo.

A TABELA 1 apresenta a notação utilizada para criação da arquitetura de um software.

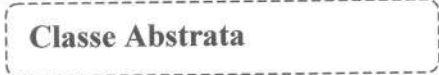
Componentes	
	Processo
	Componente Computacional/Objeto
	Componente de Dados Ativo
	Componente de Dados Passivo
	
	
	
Conexões	
	Fluxo de Controle uni e bi-direcional
	Fluxo de dados uni e bi-direcional
	Fluxo de dados e controle
	Implementação
	Agregação
	Herança

TABELA 1: Notação de Componentes e Conectores usada em Arquiteturas de Software

Capítulo 3 – ESTILOS ARQUITETURAIS

3.1 – INTRODUÇÃO

Ao caracterizar as arquiteturas de software que são utilizadas nos sistemas, identificamos seus componentes, mecanismos de interação e propriedades, podendo dessa forma classificá-las. Como resultado, isso nos possibilita o reuso desse conhecimento. No decorrer da década de 1990, houve uma crescente preocupação em compreender a organização dos sistemas de software, culminando com o surgimento da área de arquitetura de software. Assim, começou-se a identificar famílias de arquiteturas, isto é, classes de arquiteturas que possuíam aspectos e componentes peculiares a elas bem como formas de combiná-los. Isso caracteriza um estilo arquitetural (MENDES, 2002).

Assim como a arquitetura de software os estilos de arquiteturas de software caracterizam uma área nova dentro da engenharia de software, por isso há várias definições. Destacam-se algumas delas:

Segundo Mendes (2002), estilos arquitetais são as formas pelas quais se caracterizam, uma arquitetura de software que é utilizada em sistemas e/ou softwares. É a maneira de identificar os componentes, mecanismos de interação e propriedades, modo de compreender a organização dos sistemas de software. São famílias de arquiteturas, isto é, classes de arquiteturas semelhantes que possuem aspectos e componentes peculiares a elas, bem como formas de combiná-las.

Para Garlan e Shaw *apud* Bredemeyer(2003), *define-se estilos arquitetais como sendo uma família de sistemas em torno de um padrão de organização estrutural. Especialmente, um estilo arquitetural determina o conjunto de componentes e conectores que podem ser usados e combinados.*

Define-se estilo arquitetural como uma abstração de tipos de elementos e aspectos formais de várias arquiteturas específicas, concentrando talvez em um só aspecto de uma arquitetura. Um estilo arquitetural encapsula decisões importantes nos elementos e as relações entre eles. Esta definição permite que estilos só focalizem os conceitos de uma arquitetura, ou em um aspecto específico das interfaces de componentes.

*Estilo arquitetural: Um estilo consiste em um vocabulário de elementos de projeto, um conjunto de restrições bem formuladas que devem ser satisfeitas por qualquer arquitetura existente naquele estilo e uma interpretação dos componentes, interfaces e conectores são tratados como objetos de primeira classe, isto é eles tem um nome e são refinados. Objetos arquiteturais abstratos podem ser decompostos, agregados ou eliminados em uma arquitetura concreta. A semântica de componentes não é considerada parte de uma arquitetura, mas a semântica de conectores é. Moriconi **apud** Bredemeyer (2003).*

Um estilo arquitetural permite que um profissional (engenheiro, projetista ou arquiteto de software) determine a classe a qual pertence a organização de um sistema. Características dos componentes (subsistemas) e conectores dos sistemas, topologia da arquitetura, restrições semânticas e mecanismos de interação *entre os componentes ajudam a identificar o estilo que retrata a arquitetura de software do sistema* (MENDES, 2002).

Qual a utilidade de termos estilos arquiteturais? É fácil perceber que possuir estilos arquiteturais bem definidos com suas características específicas nos permite fazer um menor esforço para entender o projeto de sistemas desenvolvido por outra pessoa e, portanto, reduz a quantidade de informações a serem assimiladas num novo projeto bem como traz benefícios, como maior confiabilidade, qualidade, manutenção, redução de custos, cronograma e tornando o sistema ou software eficiente e eficaz.

Para BASS et al. (1998) um estilo arquitetural de software é de alguma forma análogo a estilos arquitetônicos em edifícios, como gótico ou grego. Ele consiste de algumas características chaves e regras para combinar essas características de forma que a integridade arquitetural seja preservada. Um estilo de software é determinado pelo seguinte:

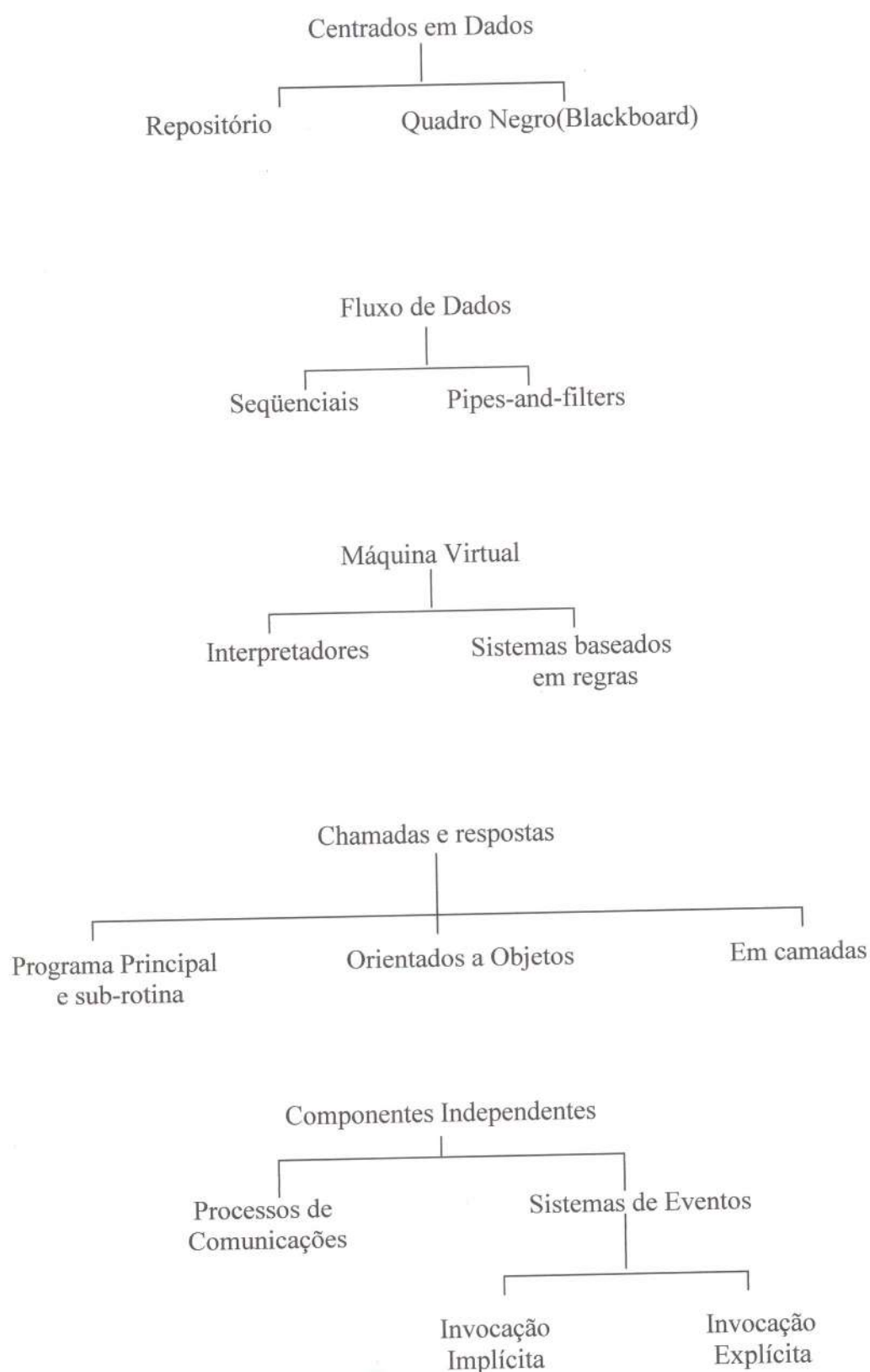
- Um conjunto de tipos de componentes (por exemplo, repositório de dados, ou procedimento) que executam alguma função em tempo de execução;
- Um layout topológico destes componentes que indicam o relacionamento entre eles em tempo de execução;
- Um conjunto de restrições semânticas (por exemplo, em um repositório de dados não é permitido mudar os dados armazenados nele);
- Um conjunto de conectores (por exemplo, chamada de procedimento remoto) que mediam a comunicação, coordenação ou cooperação entre componentes.

Estilos são normalmente ambíguos (intencionalmente) sobre o número de componentes envolvidos. Por exemplo, um fluxo de pipe-and-filters pode ter dois filters conectados por um pipe ou 20 filters conectados por através de 19 pipes. Estilos podem ser ambíguos sobre o mecanismo pelo qual os componentes interagem, embora alguns estilos abordam isso explicitamente. Estilos são sempre ambíguos sobre a função do sistema: um dos componentes pode ser um banco de dados, por exemplo, mas o tipo de dados pode variar (BASS et al. 1998).

Mary Shaw e David Garlan catalogaram um conjunto de estilos arquiteturais que eles observaram em um conjunto de sistemas complexos existentes. A motivação deles era a observação de abstrações para sistemas complexos existentes, mas não haviam sido estudados ou catalogados, como é comum em outras disciplinas da engenharia (BASS et al. 1998,).

Estilos não só acontecem regularmente em projetos de sistemas, mas eles ocorrem periodicamente em formas ligeiramente diferentes. *Catálogos de estilos* podem ajudar a reconhecer quando dois estilos são semelhantes, como mostra a FIGURA 1. Nesta figura estilos são categorizados em grupos relacionados. Por exemplo, um sistema de evento é um sub-estilo do estilo de componentes independentes. Sistemas de evento têm dois sub-estilos: invocação implícita e explícita. Catálogos de estilos apresentam as circunstâncias nas quais é apropriado aplicar um estilo (BASS et al, 1998).

A seguir é apresentado em detalhes estilos e sub-estilos que foram catalogados e classificados a partir das características comuns entre eles.

**FIGURA 1: Catálogo de Estilos**

3.2 –ARQUITETURAS CENTRADAS EM DADOS

Arquiteturas centradas em dados têm como objetivo obter a integração de dados, onde estes poderão ser acessados e atualizados por componentes do sistema. Um esqueleto deste estilo é mostrado na FIGURA 2. Um cliente é executado como uma linha independente de controle. Os dados compartilhados que todo o cliente acessa (e atualiza), podem ser um repositório passivo (como um arquivo) ou um repositório ativo (como um *Quadro-Negro*(blackboard)). Embora o estilo centrado em dados mostrado tem um repositório passivo (você pode verificar isto porque nenhum controle é encontrado), ele pode se apresentar de uma forma ligeiramente diferente. O principal é que não há nada além de um repositório de dados centralizados que serve de comunicação entre vários clientes. O significado da comunicação (às vezes chamado de *modelo de coordenação*) distingue os dois subtipos: *repositório*, que está sendo apresentado, e *quadro-negro*(blackboard). Um *quadro-negro* envia notificação a subscritores quando dados de seus de interesses são mudados. Um *quadro-negro* difere da figura 2 no sentido que apresentaria setas de controle¹ que emanam dos dados compartilhados (BASS et al.1998).

Conforme Bass et al. (1998), estilos centrados em dados estão se tornando importantes porque eles oferecem uma solução estrutural à integrabilidade. Muitos sistemas, especialmente aqueles construídos de componentes pré-existent, estão obtendo integração de dados através do uso de mecanismos de quadro-negro. Eles têm a vantagem de que os clientes são relativamente independentes com relação a um outro e o repositório de dados é independente dos clientes. Portanto, este estilo é escalável: novos clientes podem ser adicionados. Também é modificável com respeito à mudança da funcionalidade de qualquer cliente particular porque este não é afetado por outros clientes. Agrupar clientes dessa forma pode melhorar o desempenho, visto que, por serem independentes, podem ter sua funcionalidade incrementada sem ser afetada e nem afetar os outros.

Note que quando os clientes são construídos como processos independentes, o que nós temos é um estilo de cliente-servidor que pertence a seção de componentes independentes do catálogo de estilo apresentado anteriormente. Pode-se observar que os estilos não são rigidamente separados uns dos outros.

¹ Notação utilizada apresentada na tabela TABELA 1

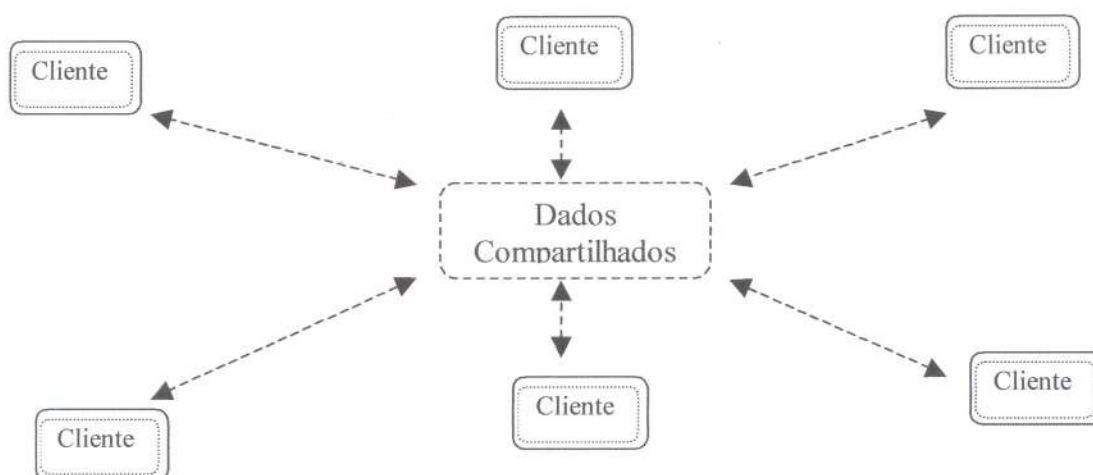


FIGURA 2: Estilo Centrado em Dados

3.3 – ARQUITETURAS DE FLUXO DE DADOS

Arquiteturas de fluxo de dados têm a o objetivo de alcançar as qualidades de reuso e modificabilidade. O estilo fluxo de dados é caracterizado por visualizar o sistema como uma série de transformações em pedaços sucessivos de entradas de dados. Dados entram no sistema e então fluem pelos componentes, um de cada vez até que eles são direcionados a algum destino final (saída ou depósito de dados). O estilo tem dois subtipos, *sequencial* e *pipe-and-filter*. No estilo *sequencial*, componentes são programas independentes, e é assumido que cada componente é executado por completo antes do próximo componente começar. Cada grupo de dados é transmitido totalmente entre os componentes. A aplicação típica para este estilo é o processamento de dados clássico (BASS et al. 1998). Este estilo está ilustrado na FIGURA 3.

Segundo Bass et al. (1998), o estilo *pipe-and-filter* enfatiza a transformação incremental de dados através de componentes sucessivos. Este é um estilo típico da família de sistemas operacionais Unix.. *Filters* são responsáveis pelas transmissões de fluxo de dados. Eles transformam dados, usam pouca informação contextual, e não retem nenhuma informação de estado entre as instâncias. *Pipes* não possuem estado, simplesmente existem para mover dados entre os *filters*.

Ambos *pipes* e *filters* são executados até que não haja mais computação ou transmissões possíveis. Restrições no estilo *pipe-and-filters* indicam a maneira em que eles podem ser agrupados. Um *pipe* tem uma porta de entrada que só pode ser conectado à saída

de um *filter*, e uma porta de saída que só pode ser conectada a entrada de um *filter* (BASS et al.1998).

Sistemas *pipe-and-filters*, como todos os outros estilos, tem várias vantagens e desvantagens. Suas vantagens provem de sua simplicidade - os modos limitados nos quais eles podem interagir com o ambiente. Esta simplicidade significa que a função de um sistema de *pipe-and-filter* é nem mais e nem menos a composição das funções de suas primitivas. Não há nenhuma interação de componentes complexos para administrar. O estilo *pipe-and-filter* também simplifica a manutenção do sistema e aumenta o reuso pela mesma razão - *filters* podem ser tratados como caixas pretas. Tanto os *pipes* quanto os *filters* podem ser hierarquicamente compostos: qualquer combinação de *filters*, conectados por *pipes*, podem ser empacotados e serem vistos pelo ambiente externo como um simples *filter*. Finalmente, um *filter* pode processar sua entrada de forma isolada do resto do sistema, um sistema de *pipe-and-filters* é facilmente colocado em paralelo ou distribuído, permitindo um aumento no desempenho do sistema sem modificá-lo (BASS et al. 1998).

Sistemas *pipe-and-filters* também têm algumas desvantagens. Por causa do modo que um problema é modelado neste estilo, o modelo *batch* está implicitamente definido. Assim, aplicações interativas são difíceis de ser criadas neste estilo. Ordenação de *filters* também podem ser difícil: não há meios de *filters* cooperativamente interagirem para solucionarem problemas (BASS et al. 1998).

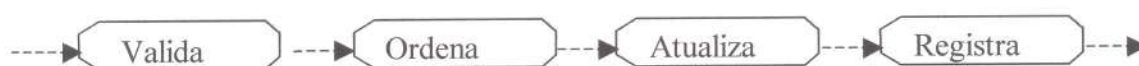


FIGURA 3: Estilo Sequencial

3.4 – ARQUITETURAS DE MÁQUINAS VIRTUAIS

Arquiteturas de máquina virtuais têm o objetivo de alcançar a qualidade de portabilidade. Máquinas virtuais são estilos de software que simulam alguma funcionalidade que não é nativa ao software e/ou hardware no qual é implementado. Isto pode ser útil de vários modos: pode permitir a simulação (e teste) de plataformas que ainda não tenham sido construídas (como um hardware novo), e pode simular modelos de "desastre" (como é comum

em simuladores de voo e sistemas críticos) que seria muito complexo, caro, ou perigoso para testar com um sistema real (BASS et al. 1998).

Exemplos comuns de máquinas virtuais são interpretadores, sistemas baseados em regras, e processadores de linguagem de comando. Por exemplo, a linguagem de Java é construída para rodar em cima da máquina virtual de Java que permite a linguagem ser independente de plataforma. Máquinas virtuais têm a estrutura mostrada na FIGURA 4. A figura mostra três tipos de dados: *o programa sendo interpretado*, *os dados do programa* (como os valores de variáveis nomeados na execução do programa), e *o estado interno do interpretador* (como os valores de registros ou a requisição atual sendo executados). *O mecanismo de interpretação* seleciona uma instrução do programa sendo interpretado, atualiza seu *estado interno* e, baseado na instrução, atualiza os dados do programa (BASS et al.1998).

Permite a execução de um programa através de um interpretador, adiciona flexibilidade através da habilidade de interromper e examinar o programa, e introduz modificações em tempo de execução. Mas há um custo de desempenho por causa da computação adicional envolvida na execução (BASS et al.1998).

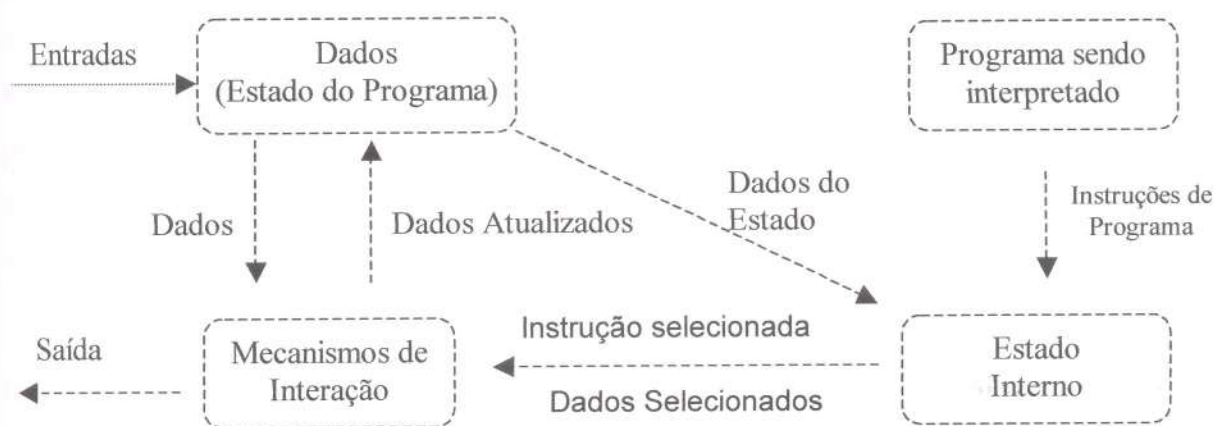


FIGURA 4: Estilos de Máquinas Virtuais

3.5 – ARQUITETURAS DE CHAMADA-E-RESPOSTA

Arquiteturas de chamada e resposta têm o objetivo de alcançar as qualidades de modificabilidade e escalabilidade. Arquiteturas de chamadas e resposta tem sido o estilo arquitetônico dominante em grandes sistemas de software durante os últimos 30 anos. Assim como esse estilo, um número de sub-estilos com características interessantes têm surgido.

Arquitetura de *Programa principal e sub-rotinas*, como mostrado na FIGURA 5 representa o paradigma de programação clássico. O objetivo é decompor um programa em pedaços menores hierarquicamente. Há uma única linha de controle e cada componente tipicamente na hierarquia adquire este controle (opcionalmente junto com alguns dados) de programas que deram origem passando isto para seus filhos (BASS et al. 1998).

Sistemas de chamada de procedimento remotos são sistemas de *Programa principal e sub-rotinas* que são decompostos em partes que se mantêm em computadores conectados por uma rede. O objetivo é aumentar o desempenho distribuindo as computações e tirar proveito de múltiplos processadores (BASS et al. 1998).

Conforme Bass et al. (1998), *Sistemas Orientados a Objetos* ou de *Tipos de Dados Abstratos*, como mostrado na FIGURA 6 é a versão moderna de arquiteturas de chamada-e-resposta. O paradigma orientado a objetos, assim como paradigma de dados abstratos do qual ele evoluiu, enfatiza o empacotamento de dados e o conhecimento de como manipular e acessar aquele dado. O principal objetivo é a modificabilidade. Este encapsulamento esconde seu estado interno do ambiente. O acesso ao objeto só é permitido por operações específicas, tipicamente conhecidas como *métodos*, que são chamadas de procedimento restritas. Este encapsulamento promove reuso e modificabilidade, principalmente porque promove separação de conceitos: o usuário de um serviço não sabe, e não precisa saber, qualquer coisa sobre como aquele serviço é implementado. As principais características que distinguem o paradigma orientado a objetos de tipos de dados abstratos são herança (o compartilhando hierárquico de definições e código) e o polimorfismo (a habilidade para determinar a semântica de uma operação em tempo de execução).

Quando as abstrações de objetos vistos como caixas-pretas formam componentes que provêem serviços, e outros componentes que requerem esses serviços, trata-se do *estilo Cliente-Servidor baseado em Camadas* (ao invés de um estilo de *Cliente-servidor baseado em Processos* que será visto na próxima família) (BASS et al. 1998).

Para Bass et al. (1998), *Sistemas em camadas*, como mostrado em FIGURA 7, são aqueles nos quais os componentes são colocados em camadas distintas de acordo com a sua funcionalidade. Neste estilo cada nível se comunica com os vizinhos imediatos. A meta é alcançar as qualidades de modificabilidade, usualidade e portabilidade. A camada mais baixa provê um pouco de funcionalidade a nível físico, como hardware, ou um Núcleo de sistema operacional. Cada camada é construída a partir de sua predecessora, omitindo a camada mais baixa das superfícies e fornecendo alguns serviços que as camadas superiores fazem uso. As

camadas superiores fornecem uma máquina virtual: conjuntos completos de funcionalidade coerentes no qual uma aplicação, ou uma máquina virtual mais complexa, pode ser construída. Na prática, sistemas em camadas não são freqüentemente "puros", funções em uma camada pode se comunicar com funções em *camadas diferentes* de seus vizinhos imediatos. Isto é chamado camadas com pontes, e esta prática é usada onde eficiência de tempo de execução é preocupante (o overhead de enviar um pedido por muitas camadas de software não pode ser absorvido). Mas camadas com pontes comprometem o modelo. Por outro lado, se a formação de camadas é pura, aumentar o sistema envolve replantar somente a camada de portabilidade, e isto pode ser feito uma vez para todos os sistemas que são construídos na máquina virtual em questão. Por exemplo, um programa escrito em Java requer nenhum esforço para ser executado em uma grande variedade de hardware e plataformas de software, porque Java apresenta uma abstração de máquina virtual uniforme em muitas plataformas. Os desenvolvedores da linguagem Java são quem tem que se preocupam com essa portabilidade, não o usuário.

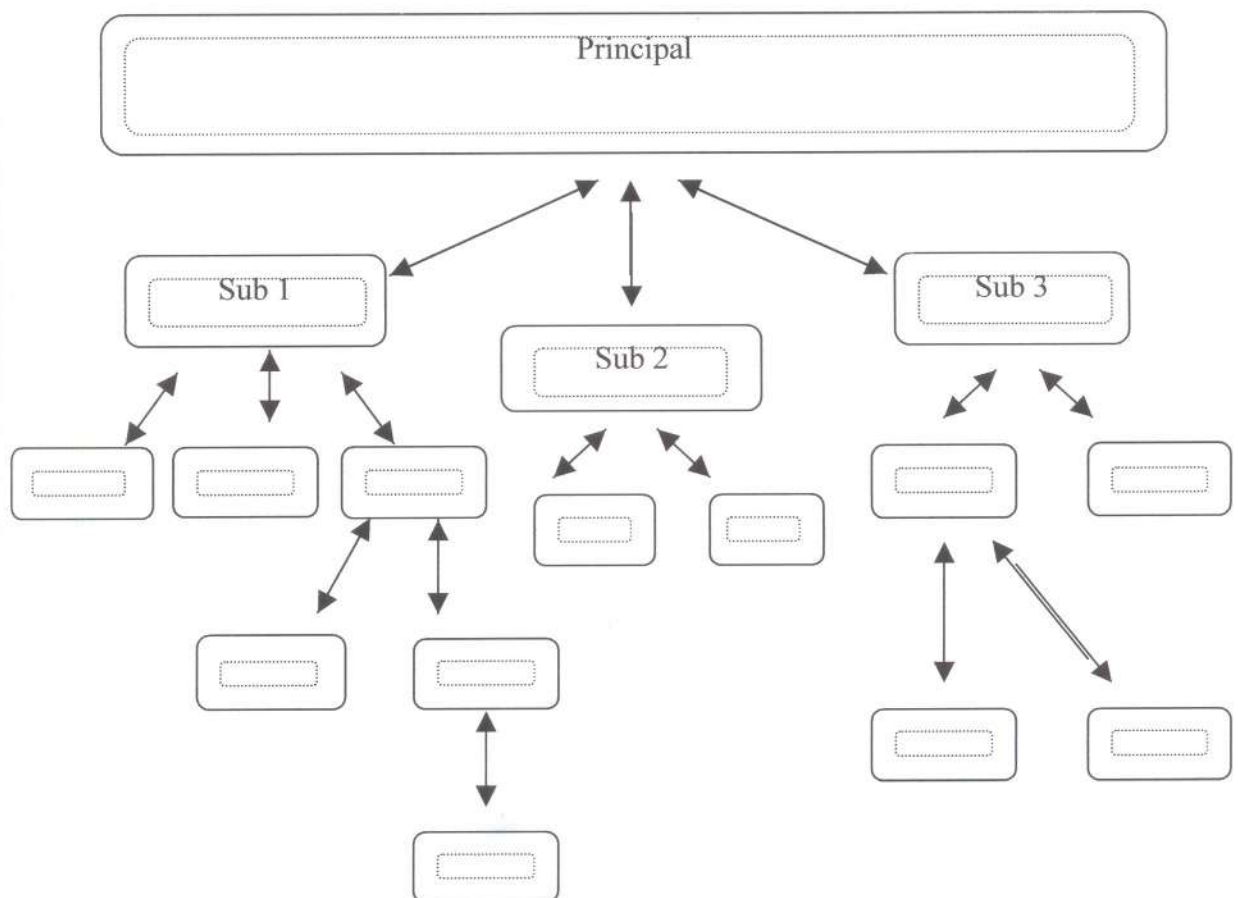


FIGURA 5: Estilo de Programa Principal e Sub-Rotinas

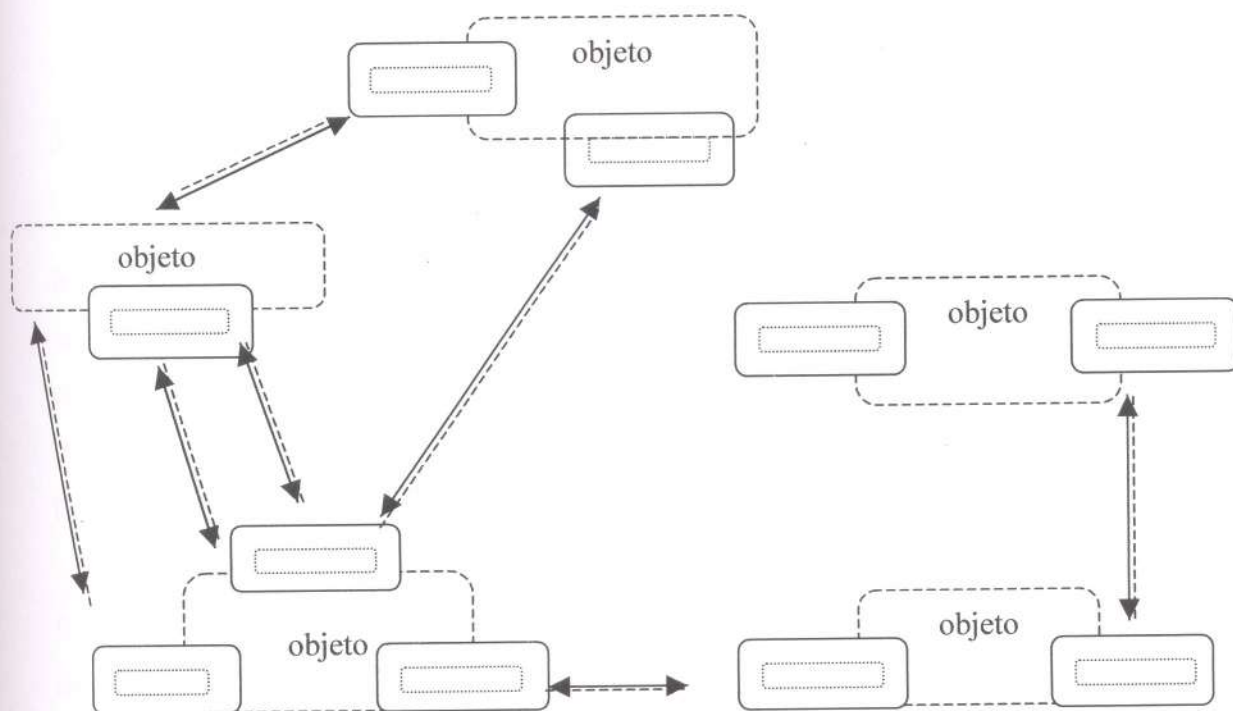


FIGURA 6: Estilo Orientado a Objeto

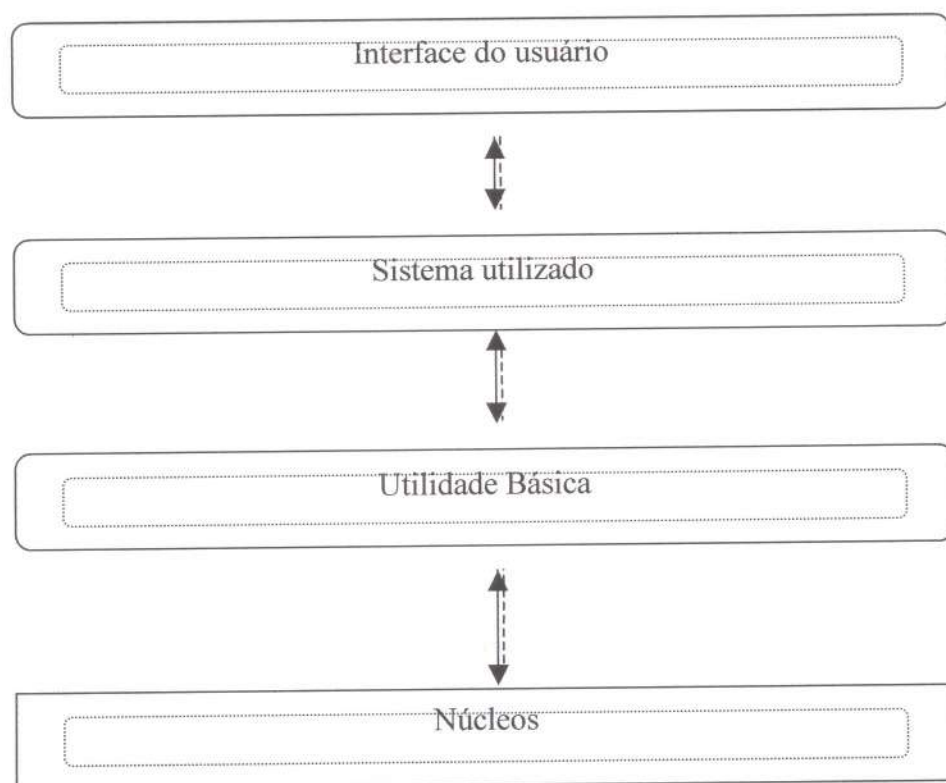


FIGURA 7: Estilo em Camadas

3.6 – ARQUITETURAS DE COMPONENTES INDEPENDENTES

Arquiteturas de Componente Independentes consistem em vários processos ou objetos independentes que se comunicam por mensagens. Todas estas arquiteturas têm o objetivo de alcançar modificabilidade através da decomposição da computação em vários pedaços. Eles enviam dados um ao outro, mas tipicamente não controlam um ao outro diretamente. As mensagens podem ser passadas a participantes nomeados ou, usar o paradigma de publicar/subscrever, entre participantes não nomeados (BASS et al.1998).

Sistemas de eventos são um subestilo no qual controle faz parte do modelo. Componentes individuais anunciam dados que eles desejam compartilhar (publicar) com o ambiente - um conjunto de componentes não nomeados. Estes outros componentes podem registrar um interesse nesta classe de dados (subscrever). Se isto ocorre, quando o dados são escritos, os componentes são invocados e recebem estes dados (BASS et al. 1998).

Tipicamente, sistemas baseados em eventos fazem uso de um gerente de mensagens que administra a comunicação entre os componentes, invocando-os e controlando-os quando uma mensagem é recebida. Neste paradigma de publicar/subscrever, um gerente de mensagens pode ou não controlar os componentes para os quais remete mensagens. Componentes informam os tipos de informação que eles fornecem e que eles desejam receber. Eles publicam informação enviando isto ao gerente de mensagens que remete a mensagem, ou, em alguns casos, uma referência de objeto, para todos os participantes interessados (BASS et al. 1998).

De acordo com Bass et al. (1998), além de sistemas baseados em eventos, o outro subestilo de componentes independentes é o *Estilo de Processos de Comunicação*, que representam os sistemas de multiprocessos clássicos. Destes, *Cliente-servidor* é um subtipo mais conhecido. A meta é alcançar a qualidade de escalabilidade. Um servidor existe para fornecer dados a um ou mais clientes que estão tipicamente localizados em uma rede. O cliente origina uma chamada ao servidor que trabalha, de forma síncrona ou assíncrona, para responder a requisição do cliente. Se o servidor trabalhar de forma síncrona, ele retorna o controle ao cliente ao mesmo tempo que retorna os dados. Se o servidor trabalha de forma assíncrona ele retorna somente os dados ao cliente, que tem sua própria linha de controle.

3.7 – ESTILOS HETEROGÊNEOS

Para Bass et al. (1998), sistemas são raramente construídos a partir de um único estilo sendo tais sistemas considerados heterogêneos. Há três tipos de heterogeneidade que são mostradas a seguir:

1. *Localmente heterogêneo*: significa que sua estrutura em tempo de execução irá revelar padrões de diferentes estilos. Por exemplo, alguns sistemas de programas principal e subrotinas pode ter um repositório de dados compartilhados.
2. *Hierarquicamente heterogêneo*: significa que um componente de um estilo, quando decomposto, é estruturado de acordo com as regras de um estilo diferente, como a FIGURA 7 ilustra.
3. *Simultaneamente heterogêneo*: significa que qualquer um de vários estilos pode definir o sistema.

Esta última forma de heterogeneidade reconhece que estilos não dividem arquitetura de software em categorias distintas. O estilo *Centrado em Dados* era composto de clientes independentes, como se fosse uma arquitetura de componentes independentes. As camadas em um *Sistema em Camadas* pode incluir *objetos ou componentes independentes* ou até mesmo *subrotinas* em um sistema de programas principal e subrotinas. Os componentes em um sistema *pipe-and-filters* normalmente são implementados como processos que operam independentemente, enquanto esperam a entrada em sua porta; novamente, isto é semelhante a sistemas de *componentes independentes* cuja ordem de execução é predeterminada (BASS et al. 1998).

Estilo: Sistemas de Eventos

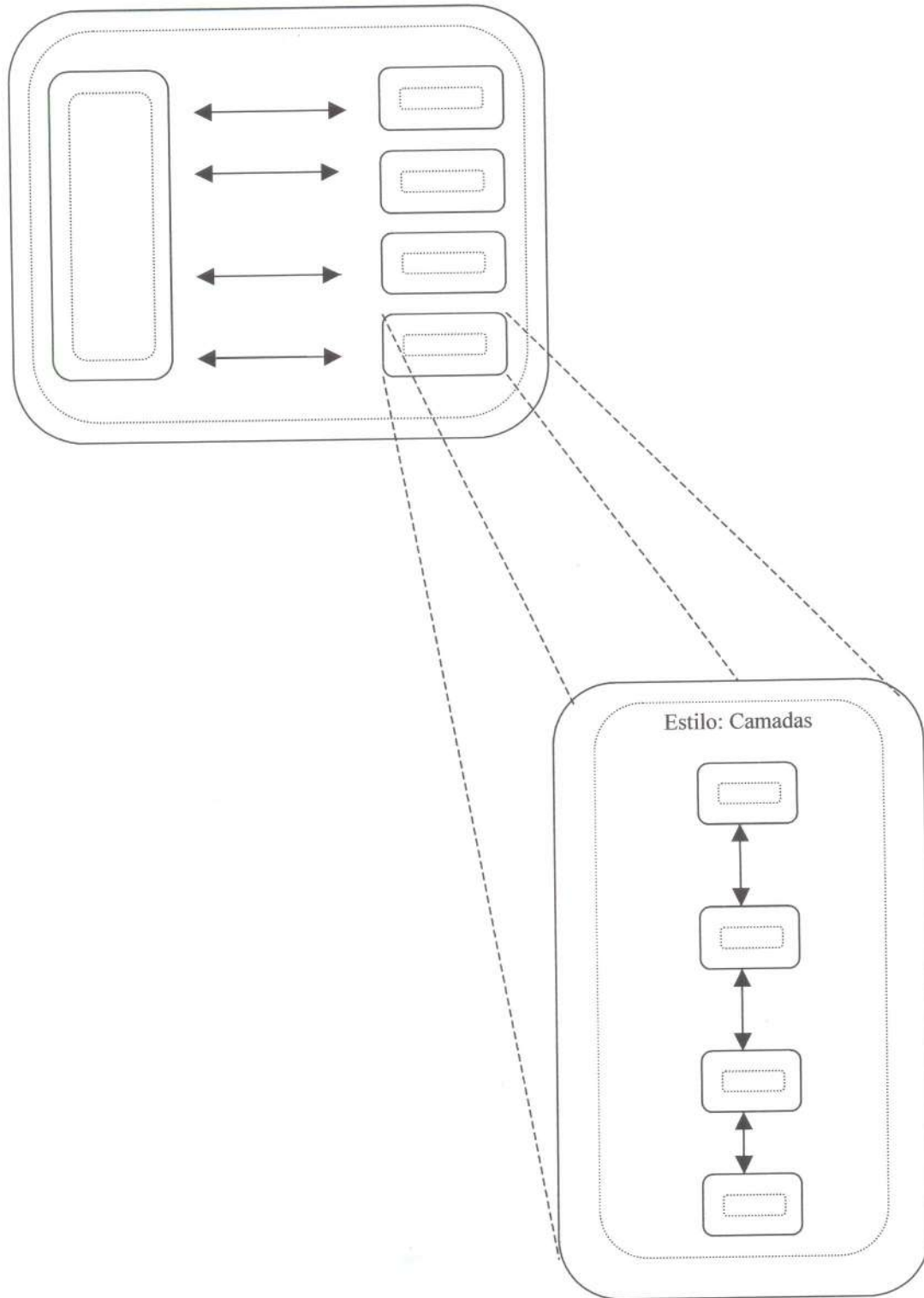


FIGURA 8: Heterogeneidade Hierárquica

3.8 - ORGANIZANDO ESTILOS ARQUITETURAIS.

Devido ao fato de que uma arquitetura quase nunca é completamente construída a partir de um único estilo, um arquiteto precisa entender o relacionamento entre estilos (Bass et al. 1998).

Segundo Bass et al. (1998), em geral, entender que a ramificação de uma arquitetura é resultado da combinação de estilos requer um processo de análise arquitetural. Nós podemos analisar o conjunto de estilos apresentados para ver como eles são descritos, o que eles têm em comum, e como eles poderiam ser estendidos para admitir outras possibilidades de projeto. Nesta seção é organizado e classificado alguns dos estilos que aparecem em descrições de software, muitos dos quais foram introduzidos na seção anterior.

A motivação dessa organização/classificação é apresentada a seguir:

- Apresentar diferenças significantes que afetam a adequabilidade de um estilo para propósitos específicos, capacitando o arquiteto a fazer melhores seleções de estilos;
- Mostrar quais estilos são variações de outros, permitindo ao arquiteto fazer combinações apropriadas de estilos;
- Fornecer uma lista de tópicos a serem considerados com o objetivo de possibilitar que o arquiteto se concentre nas questões importantes do projeto.

As características usadas para descrever e discriminar estilos são apresentadas a seguir.

3.9 – TIPOS DE CARACTERÍSTICAS

Para Bass et al. (1998), a primeira atitude de um projetista, de uma arquitetura, é freqüentemente focalizada no caráter das interações entre componentes. Assim, os principais critérios de classificação são o controle e as interações de dados entre componentes. Portanto, discriminações melhores são feitas dentro destas dimensões através dos seguintes questionamentos:

- Como controle é compartilhado, alocado, e transferido entre os componentes?
- Como os dados são comunicados pelo sistema?
- Como dados e controle interagem?

- Qual tipo de computação que o estilo permite?
- Quais tipos de componentes e conectores que são usado no estilo?

3.9.1 - Partes de um estilo: Componentes e Conectores.

Os tipos permitidos de *componentes e conectores* são formas importantes de distinção entre estilos. *Componentes e Conectores* são os blocos primários de construção de arquiteturas. Nesta categoria, um *componente* é uma unidade de software que executa alguma função em tempo de execução. Exemplos incluem programas, objetos, processos, e filtros. Um *conector* é um mecanismo que media a comunicação, coordenação, ou cooperação entre componentes. Implementações de conectores podem ser feitas em cima de muitos componentes, freqüentemente eles não correspondem a elementos do sistema executando. Exemplos incluem representações compartilhadas, chamadas de procedimento remoto, protocolos de passagem de mensagens e fluxos de dados (Bass et al.1998).

Selecionar os tipos de partes constituintes não identifica exclusivamente o estilo, existindo outros tipos de características a serem consideradas.

3.9.2 - Questões de Controle.

Conforme Bass et al. (1998), questões de controle descrevem como o controle passa entre componentes e como os componentes trabalham temporariamente juntos. Essas questões são definidas a seguir.

- *Topologia*. Qual forma geométrica representa o fluxo de controle do sistema? Um pipeline tem uma topologia linear, ou pelo menos uma topologia de controle acíclica; um estilo de programa principal e sub-rotinas se caracterizam por uma topologia hierárquica (em forma de árvore); alguns sistemas servidores tem topologia estrela; um estilo composto de processos seqüenciais de comunicação podem ter uma topologia arbitrária. Cada topologia pode ser útil para estipular a direção no qual o controle flui. Uma topologia pode ser estática ou dinâmica.
- *Sincronismo*. Quão dependentes são as ações de componentes sobre cada um dos estados de controle? Em um sistema fechado o estado de qualquer componente implica o estado de todos os outros; por exemplo, componentes de sistemas seqüenciais em batch estão “fechados” uns com os outros porque não podem

começar a execução até que seu predecessor termine a sua execução. Em sistemas *síncronos*, componentes sincronizam regularmente e freqüentemente, mas outras relações são imprevisíveis. Componentes *assíncronos* são amplamente imprevisíveis na interação ou sincronização, e componentes oportunistas, como agentes autônomos, trabalham completamente independentes um ao outro e em paralelo. Sistemas fechados podem ser seqüenciais ou paralelos dependendo de quantas linhas de controle são executados através deles. Outras formas de sincronismo implicam paralelismo.

- *Tempo de conexão*. Quando a identidade de um padrão em uma operação de transferência de controle é estabelecida? Algumas transferências de controle são predeterminadas no momento em que um programa é escrito (i.e código fonte), tempo de compilação, ou tempo de invocação (i.e quando o sistema operacional inicializa o processo). Outros são dinamicamente incorporados enquanto o sistema está executando.

3.9.3 - Questões de dados.

Descrevem como dados se movem em um sistema; elas definem o seguinte:

- *Topologia*. Topologia de dados descreve a forma geométrica do gráfico de fluxo de dados de sistema. As alternativas são as mesmas para topologia de controle.
- *Continuidade*. Quão *contínuo* é o fluxo de dados ao longo do sistema? Um sistema de fluxo contínuo tem dados novos disponíveis a toda hora; um sistema de fluxo *esporádico* tem dados novos gerados a tempos discretos. Transferência de dados também pode ser de *alto volume*.
- *Modo*. Modo de dados descreve como dados são disponibilizados ao longo do sistema. No estilo de objeto, eles são passados de componente a componente, enquanto que em sistemas de dados compartilhados eles são compartilhados através de um lugar acessível a todos os componentes. Alguns dados podem ser repassados a componentes específicos.
- *Tempo de Conexão*. Quando é que a identidade de um padrão em uma operação de transferência de controle é estabelecida? É a mesma analogia das questões de controle.

3.9.4 - Questões de interação de Dados/ Controle.

Questões de interação descrevem relacionamento entre certas questões de dados e controle.

- *Forma.* As topologias de fluxo de dados e fluxo de controle são substancialmente isomórficas.
- *Direção.* Se as formas são substancialmente as mesmas, as direções do fluxo de controle e de dados são as mesmas ou possuem direções opostas? Em um sistema de fluxo de dados, como pipe-and-filters, controle e dados passam junto de componente a componente. Porém, em um estilo de cliente-servidor, controle tende a fluir nos servidores e dados fluem nos clientes.

3.9.5 - Tipos de Computação.

Diferentes classes de arquiteturas podem sofrer diferentes tipos de análise. Um sistema de componentes operando de forma assíncrona e em paralelo podem ter abordagem computacionais diferentes de um sistema que é executado como uma sequência fixa de passos. Muitas técnicas de análise compõem seus resultados a partir de análises de subestruturas, mas isto depende da habilidade para combinar sub-análise. Assim, estilos arquiteturais diferentes podem ser analisados por tipos de análise diferentes.

Um resumo das principais características dos estilos apresentados no início desse capítulo é apresentado na tabela.

3.10 – USANDO ESTILOS EM PROJETOS DE SISTEMAS.

De acordo com Bass et al. (1998) Qual estilo você deveria escolher para projetar um sistema? A resposta (é claro) depende das características do sistema a ser projetado.

Se você pedir para um arquiteto que lhe fale sobre a arquitetura de um sistema, a resposta será expressa em termos de soluções arquiteturais. Se o sistema tem que ser super confiável, isto poderia ser problemático, onde provavelmente se ouviria falar primeiro dos aspectos de tolerância a falhas e redundância da arquitetura, ou talvez o sistema tenha que ter alto desempenho, o que ocasionaria outras respostas do arquiteto. Se o arquiteto soubesse que o sistema ia durar por muito tempo, crescendo e sendo modificado, você pensaria sobre

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados			Interação de Dados/Controle		
	Componentes	Conectores	Topologia ^a	Sincronismo ^b	Tempo de Conexão ^c	Topologia ^a	Continuidade ^d	Modo ^e	Tempo de Conexão	Formas Isomórficas	Direções de fluxo
Fluxo de Dados: dominado por fluxo de dados no sistema. Tipo de computação: composição funcional.											
Sequencial	Programas	dados	Linear	Seq	r	Linear	Spor. Invol.	Passado, Compar-tilhado	r	Sim	Mesma
Pipes and Filters	Transdutores	Stream de dados	Linear	Assíncrono	i,r	Linear	Cont, Ivol ou hvol	Passado	i, r	--	--

TABELA 2: Classificação de Estilos Arquiteturais - Fluxo de Dados

- a. Hier. (Hierárquico), Arb. (Arbitrário), Star, Linear (um caminho), Fixo (Determinado pelo Estilo).
b. Seq. (Seqüencial, Uma Linha De Controle) Síncrono, Assíncrono, Opp (Oportunista).
c. w (Tempo de escrita - Código fonte), c (tempo de compilação), i (tempo de invocação), r (Tempo de Execução).
d. Spor (Esporádico), Cont.(Continuo), Hvol(Alto Volume), Ivol (Baixo Volume).
e. Mcast (Multicast).

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados			Interação de Dados/Controle		
	Componentes	Conectores	Topologia ^a	Sincronismo ^b	Tempo de Conexão ^c	Topologia ^a	Continuidade ^d	Modo ^e	Tempo de Conexão	Formas Isomórficas	Direções de fluxo
Chamadas eRespostas: dominado por ordem de computação, normalmente com única linha de controle. Tipo de Computação: Hierarquia											
Programa Principal/subrotinas	Procedi-mentos, dados	Chamadas de procedi-mento	Hier	Seq	w, c	Arb.	Spor, Ivol	Passado, Compar-tilhado	w, c, r	Não	--
Tipos de dados Arbstratos	Gerentes	Chamadas estáticas	Arb.	Seq..	w, d	Arb.	Spor. Ivol.	Passado	w, c, r	Sim	Mesma
Objetos	Gerentes (objetos)	Chamadas dinâmicas	Arb.	Seq.	w, c, r	Arb.	Spor., Ivol.	Passado	w, c, r	Sim	Mesma
Chamadas baseadas em cliente servidor	Programas	Chamadas ou RPC	Estrela	Síncrono	w, c, r	Estrela	Spor. Ivol.	Passado	w, c, r	Sim	Oposta
Camadas	Vários	Vários	Hier.	qualquer	qualquer	Hier.	Spor, Ivol, cont.	qualquer	W, c, i, r	Fre- quente	Mesma ou oposta.

TABELA 3: Classificação de Estilos Arquiteturais - Chamada e Resposta

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados				Interação de Dados/Controle		
	Componentes	Conectores	Topologia ^a	Sincronismo ^b	Tempo de Conexão ^c	Topologia ^a	Continuidade ^d	Modo ^e	Tempo de Conexão	Formas Isomórficas	Direções de fluxo	
Componentes												
Independentes: domínio por padrões de comunicação entre processos independentes. Tipo de Argumento: não determinado												
Sistemas baseados em eventos	Proce-ssos	Sinais	Arb.	Assíncrono	w, c, r	Arb.	Spor. Ivol.	Multi-Cast	w, c, r	Sim	Mesmo	
Processos de comunicação	Proce-ssos.	Protocolo de mensagem	Arb.	Qualquer	w, c, r	Arb.	Spor. Ivol.	Qualquer	w, c, r	Possível	Se isomórfica, outro	

TABELA 4: Classificação de Estilos Arquiteturais - Componentes Independentes

FARN

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados			Interação de Dados/Controle		
	Componentes	Conectores	Topologia ^a	Sincronismo ^b	Tempo de Conexão ^c	Topologia ^a	Continuidade ^d	Modo ^e	Tempo de Conexão	Formas Isomórficas	Direções de fluxo
Centrados em Dados : dominação por dados centrados armazenados e manipulados por processos independentes. Tipo de Computação : Interação de dados.											
Repositório	Memória	Consultas	Estrela	Assíncrono, Opp.	w	Estrela	Spor. Ivol.	Compartilhado, passado	w	Possível	Se isomórfico, Oposto
Black-board	Memória	Acesso direto	Estrela	Assíncrono, Opp.	w	Estrela	Spor. Ivol.	Compartilhado, passado	w	não	n/a

TABELA 5: Classificação de Estilos Arquiteturais - Centrados em Dados

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados			Interação de Dados/Controle		
	Componentes	Conectores	Topologia ^a	Sincronismo ^b	Tempo de Conexão ^c	Topologia ^a	Continuidade ^d	Modo ^e	Tempo de Conexão	Formas Isomórficas	Direções de fluxo
Máquinas Virtuais: Caracterizado por transformar uma instrução em outra											
Tipo de Computação: Níveis de Serviço											
Interpretores	Memória/Máquina de estado	Acesso direto a dados	Fixo, Hier.	Seq.	w, c	Hier.	Cont.	Compartilhado	w, c	não	n/a

TABELA 6: Máquinas Virtuais

Capítulo 4 – ESTUDO DE CASO

O sistema a ser modelado trata-se de um sistema de comércio eletrônico via WEB, onde foram analisadas características específicas do sistema em questão. O sistema envolve a parte de vendas de determinados produtos por parte de um cliente, assim como a administração destes produtos. Fazendo uma análise dos componentes que fazem parte do sistema, identificamos os componentes *Clientes*, *Admin* (Administrador do Sistema), *Produtos*, *Usuário* e *Compras*. O relacionamento existente ocorre entre *Admin* e *Produto*, *Cliente* e *Compras*, *Compras* e *Produtos*, e entre *Compras* e *Usuário*. O estilo identificado para modelar o sistema é heterogêneo, tendo como sub-estilo principal o estilo de arquitetura *Cliente-Servidor*. Os sub-estilos *Orientado a Objeto* e *Chamada e Resposta* também são usados.

As figuras FIGURA 9, FIGURA 10, e FIGURA 11, apresentam os componentes e o relacionamento entre eles.

O estudo de caso tem por finalidade mostrar a importância da Arquitetura de Software, em específico o uso de Estilos de Arquitetura de Software, através das características do sistema e as características já definidas por algumas arquiteturas e estilos.

Na FIGURA 9 o estilo utilizado é o *cliente/servidor*, onde os componentes ADMIN e CLIENTE são vistos como clientes e o SISTEMA é visto como o servidor de serviços. Podemos observar que o relacionamento entre ADMIN-SISTEMA envolve *fluxo de controle* e *fluxo de dados*, enquanto o relacionamento entre CLIENTE e SISTEMA só envolve *fluxo de dados*. O componente SISTEMA é visto como uma arquitetura criada, não é modelada a partir de nenhum dos estilos apresentados, composta pelos componentes PRODUTOS, USUÁRIO e COMPRAS, o relacionamento entre eles é apresentado nas figuras FIGURA 10 e FIGURA 11.

Na FIGURA 10 podemos observar o relacionamento entre o componente ADMIN e PRODUTOS, que faz parte de SISTEMA. O componente ADMIN representa o administrador do sistema que tem como principais atividades *incluir*, *excluir* e *alterar* produtos do sistema. O componente PRODUTO é visto como um objeto que possui os métodos *incluir*, *excluir* e *alterar*. O estilo utilizado é o *Orientado a Objetos*.

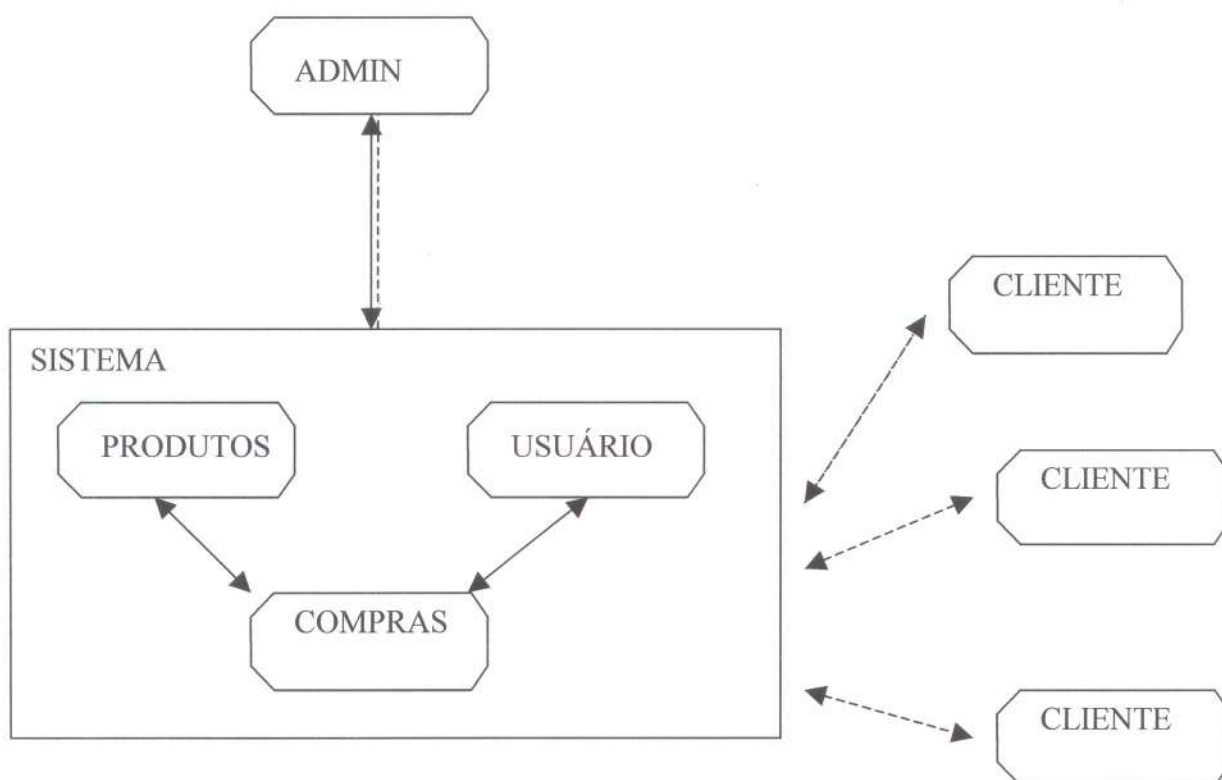


FIGURA 9: Visão Geral da Arquitetura – *Cliente/Servidor*

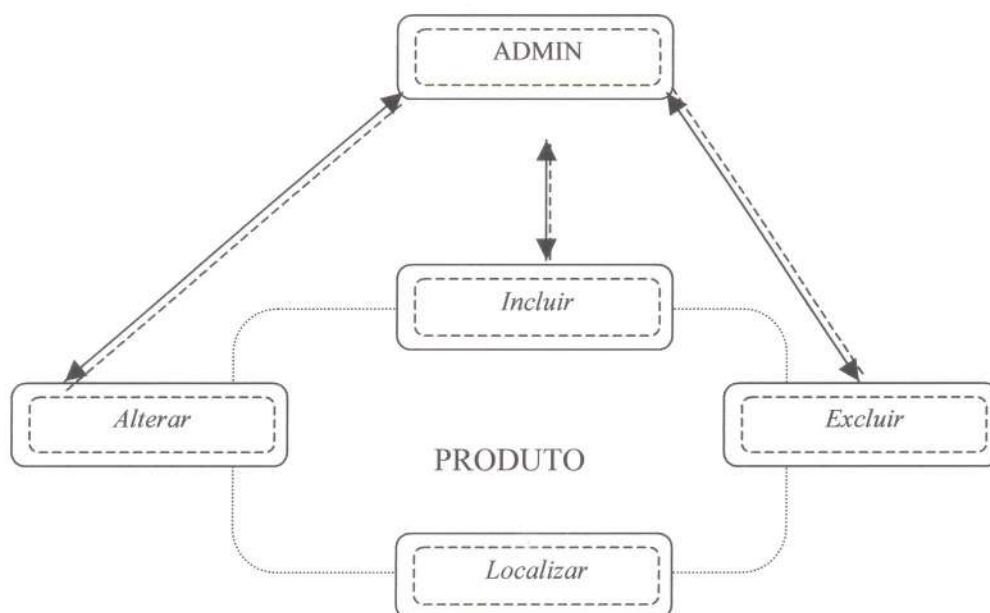


FIGURA 10: Arquitetura de Chamada e Resposta - *Orientado a Objetos*

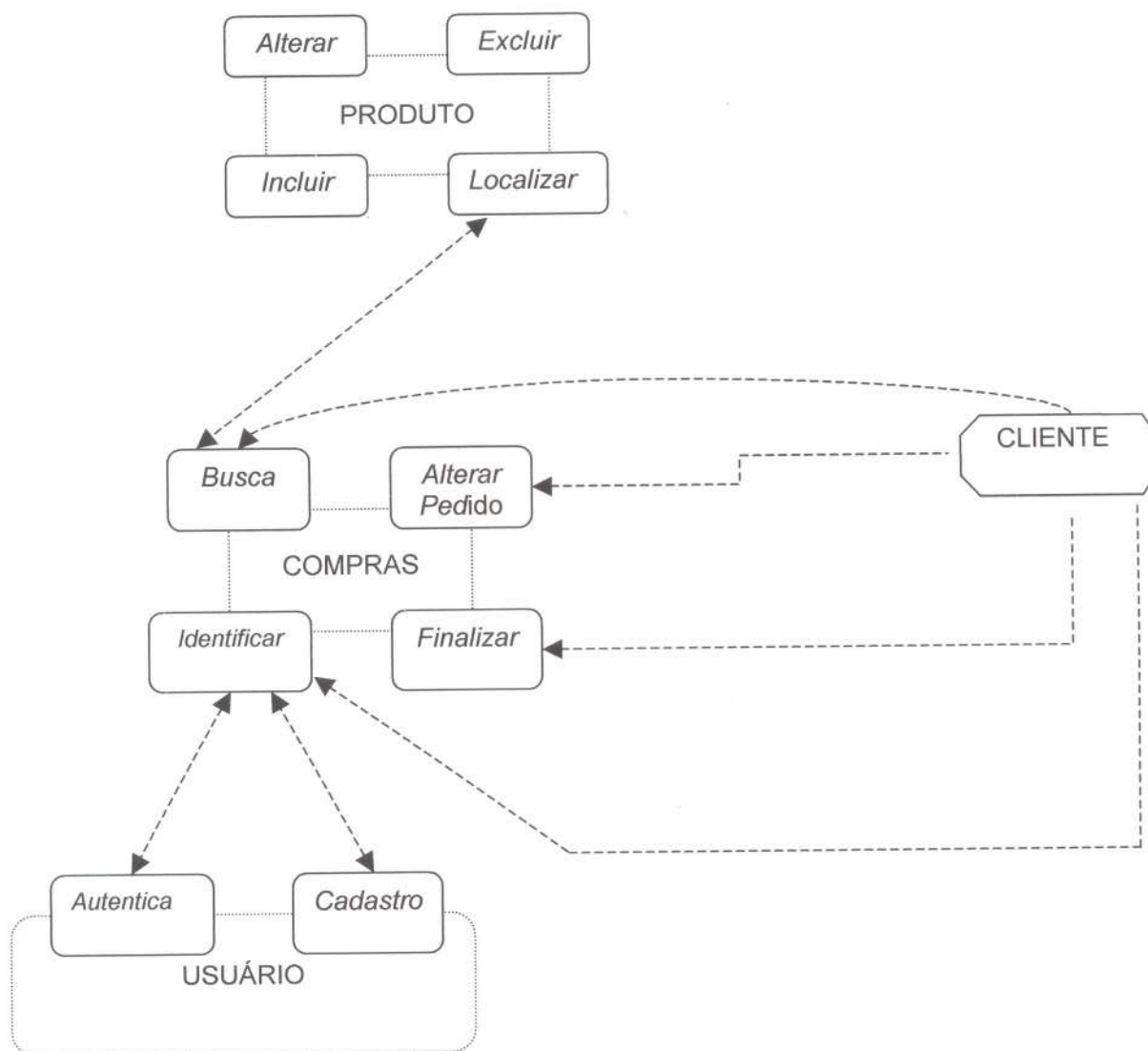


FIGURA 11: Arquitetura de Chamada e Resposta - *Orientado a Objetos*

Capítulo 5 – ANÁLISE DE RESULTADOS

A Arquitetura de Software permite a concepção de sistemas a partir de seus componentes e conexões, abstração essa que permite o projetista dedicar-se a estrutura do sistema sem se preocupar com detalhes de implementação. A idéia de estilos de arquitetura de software pré-definidos viabiliza e agiliza a concepção do sistema como um todo. A escolha desses estilos é feita de acordo com características específicas do sistema que estar sendo projetado. Vários estilos são propostos e a partir deles é possível escolher qual o mais adequado em cada sistema em particular.

O estudo de caso de um sistema de comércio eletrônico permite a descrição de seus componentes e conectores, onde a identificação das características dos componentes e conectores envolvidos no sistema são de fundamental importância para a definição dos estilos utilizados.

A distinção clara entre os diversos estilos apresentados e das características do sistema permitem ao arquiteto desenvolver uma arquitetura clara e específica que servirá como estrutura geral na fase de implementação do sistema, evitando dessa forma ambigüidades e redundâncias de projeto.

Capítulo 6 – CONCLUSÃO

A aplicação e uso de estilos arquiteturais geram confusão e dúvidas por parte dos projetistas de software visto que a utilização dessas técnicas é pouco definida. A apresentação de um estudo de caso de um comércio eletrônico, bem como a apresentação de alguns estilos já definidos, mostra que as características de cada sistema são vitais pois, de acordo com estas características pode-se definir o estilo mais adequado.

O uso de estilos arquiteturais de software proporciona para os projetistas de software maior segurança, reusabilidade, manutenção, confiabilidade e redução nos custos de projetos de software. Permite também uma maior interação entre as várias pessoas envolvidas no projeto do sistema tais como, cliente, projetista, usuário, etc, onde eles podem possuir diferentes conceitos sobre o sistema. A arquitetura de software fornece um mecanismo de comunicação comum entre essas pessoas.

Os estilos arquiteturais de software permitem ao projetista uma melhor visão de como os componentes e conectores poderão se comportar no sistema e essa visão pode evitar inúmeros problemas no desenvolvimento e implementação do sistema.

REFERENCIAS BIBLIOGRÁFICAS

BASS, L.; CLEMENTS, P.; KAZMAN, Rick. Software Architecture In Practice. EUA: Addison Wesley, 1998.

BREDEMEYER, Dama. Software Architect. Disponível em: <http://www.bredemeyer.com> . Acesso em: 15/04/2003.

CARVALHO, Ariadne M. B. Rizzzone; CHIOSSI, Thelma C. dos Santos. INTRODUÇÃO À ENGENHARIA DE SOFTWARE. Campinas, SP: Editora da Unicamp, 2001.

GARLAN, David; SHAW, Mary. An Introduction Software Architecture. School of Computer Science Carnegie Mellon University Pittisburgh, Pittisburgh.

MENDES, Antônio. Arquitetura de Software: Desenvolvimento Orientado para arquitetura. Rio de Janeiro: Editora Campos LTDA, 2002.

PRESSMAN, Roger S. ENGENHARIA DE SOFTWARE.

BIBLIOGRAFIA

BASS, L.; CLEMENTS, P ; KAZMAN, Rick. Software Architecture In Practice. EUA: Addison Wesley, 1998.

THAIS, Batista. Arquitetura de Software. Disponível em: <http://dimap.ufrn.br/~thais/arquitetura/home.html> . Acesso em: 22/05/2003.

VAROTO, Ane C. Visões em Arquitetura de Software. 2002. Dissertação (Mestrado em Ciências da Computação). Universidade de São Paulo.