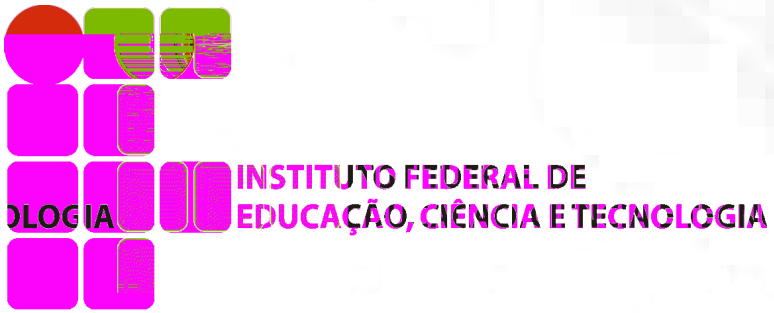
INF016 – Arquitetura de Software

04 – Projetando Arquiteturas

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas



Objetivos

- Como os projetos arquiteturais são criados ?
- Qual abordagem é ensinada aos engenheiros e arquitetos ?
- Alguns acreditam que projeto arquitetural não pode ser ensinado como um método. “Observe o mestre e aprenda por osmose … “
- Outros acreditam ser um dom com o qual você nasce ou não
- Discorda-se dessas duas visões

Objetivos

- Todos temos a habilidade de projetar …
- O projeto está sujeito a análises rigorosas e metódicas. Ele não é incapaz de ser estudado
- O resultado deste estudo é um conjunto de abordagens e técnicas de projeto que podem ser descritas, ensinadas, avaliadas e refinadas
- Investigar projetos existentes ajuda a identificar boas estratégias
- Pode-se ensinar ferramentas e métodos simples de projeto
- Existe espaço para a criatividade, entretanto ...

Objetivos

- Apresentar abordagens que ajudam o estudante a aprender como projetar arquiteturas:
  - Fundamentos do processo de projeto
  - Ferramentas conceituais básicas
  - Padrões e estilos arquiteturais
  - Recuperação de projeto
  - Projeto *r enrf il nos*
  - Processo de projeto revisitado

Processo de Projeto

- Fases (Jones, 1970):
  1) Viabilidade: identificação de um conjunto de conceitos viáveis. Um conjunto de arranjos alternativos para o projeto é rapidamente derivado
  2) Projeto Preliminar: seleção e desenvolvimento do melhor conceito, o qual é dividido e processado, em paralelo, nas próximas fases
  3) Projeto Detalhado: desenvolvimento das descrições de engenharia do conceito
  4) Planejamento: avaliação e alteração do conceito de modo a adequá-lo aos requisitos de produção, distribuição, consumo e descarte

Processo de Projeto

- O modelo de Jones é tão amplamente utilizado que é difícil perceber que outras abordagens são possíveis:
  - Ele pode não funcionar sempre ou, mesmo funcionando, não produzir os melhores resultados:
    - Impossibilidade de identificação do conjunto de conceitos viáveis para a estrutura geral do sistema (fase 1)
    - A fase 1 é geralmente realizada por um único arquiteto. Para sistemas grandes e complexos torna-se difícil. Utilizar uma equipe de arquitetos traz novos problemas
    - De forma similar, a abordagem não é adequada quando deseja-se projetar uma arquitetura para um **conjunto** de produtos. Novamente tem-se maior complexidade

Processo de Projeto

- Diversas estratégias de projeto (modelos de processo aplicados no nível de projeto):
  - Padrão: modelo linear de Jones
  - Cíclico: se problemas ou abordagens inviáveis são encontrados nas fases 2-4 o processo retorna a uma fase anterior
  - Paralelo: após a (ou na) fase 1 alternativas independentes são exploradas em paralelo
  - Adaptativo: a estratégia de projeto a ser utilizada na próxima fase é decidida no fim da fase atual, com base em *feedback* obtidos nesta fase
  - Incremental: projeta-se à medida em que o desenvolvimento é realizado, melhorando de forma incremental qualquer projeto existente após a fase anterior

Concepção Arquitetural

- **Como** identificar um (ou vários) arranjos viáveis para o projeto ?
  - Resposta fácil 1: aplique as ferramentas fundamentais de projeto ensinadas pela Engenharia de Software: abstração e modularidade
    - São necessárias e úteis mas são apenas ferramentas. Onde e como manejá-las ?
  - Resposta fácil 2: inspiração …
    - Criatividade é necessário mas não é uma varinha mágica
    - Pode-se minimizar e isolar as partes do projeto que requerem maior criatividade e aplicar técnicas mais prosaicas e previsíveis nas outras partes
    - Precisa-se saber, entretanto, onde é necessário a criatividade e onde não é

Concepção Arquitetural

- **Como** identificar um (ou vários) arranjos viáveis para o projeto ?
  - Resposta mais comum, efetiva e apropriada: use sua experiência aplicada
  - Não é uma resposta superficial, existe uma sofisticação considerável no uso de experiência
  - Não é infalível e as vezes não é suficiente, entretanto

Concepção Arquitetural

- Ferramentas conceituais fundamentais:
  - Separação de *off one h*
  - Abstração
  - Modularidade
  - Antecipação de mudanças
  - Projeto voltado para generalidade

Concepção Arquitetural
Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Abstração é a seleção de um conjunto de conceitos como representantes de um todo mais complexo
  - Pode ser *bottom-up*, partindo dos detalhes e chegando aos conceitos sumários
  - Entretanto, quando relacionado a projeto, geralmente é aplicado *top down*: define-se um arranjo de partes abstratas (conceitos) que compõem a solução ainda em alto nível e então reifica-se estes conceitos em estruturas mais completas até chegar no código-fonte
  - Poderia ser usado os termos refinamento ou dedução, porém utiliza-se abstração pois ela será uma caracterização precisa e útil do código-fonte

Concepção Arquitetural
Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Mas a pergunta persiste: “Quais conceitos devem ser escolhidos no início de um projeto ?”
  - Encontre uma “máquina” simples que serve como abstração de um potencial sistema que irá realizar a tarefa desejada
    - Ex: planilha de cálculo: grafo de relacionamentos onde, quando um nó é modificado, os relacionamentos são todos re-avaliados para manter a planilha sincronizada
    - Ex: *hbio c e* para máquina de fax: uma máquina de estados pequena
    - Ex: sistema embarcado em aviões: lê-se os sensores, calcula-se as leis de controle, envia valores para os *s h d h* e atuadores e então repete-se tudo novamente

Concepção Arquitetural
Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Embora possam parecer triviais ou inadequadas elas apresentam uma primeira concepção plausível sobre como a aplicação poderia ser construída

Domain	Simple Machines
Graphics	Pixel arrays Transformation matrices Widgets Abstract depiction graphs
Word processing	Structured documents Layouts
Industrial process control	Finite state machines
Income tax return preparation	Hypertext Spreadsheets Form templates
Web pages	Hypertext Composite documents
Scientific computing	Matrices Mathematical functions
Financial accounting	Spreadsheets Databases Transactions

Concepção Arquitetural
Ferramentas Fundamentais

- Escolhendo o nível e os termos do discurso:
  - Precisa-se definir o escopo e os termos do discurso
  - A abordagem padrão (linear) demanda que o sujeito inicial de discurso seja o sistema inteiro
  - Entretanto, existem duas alternativas:

  1) Trabalhar em um nível mais baixo que o da aplicação como um todo:
     - Trabalha-se em algo que assume-se ser parte da solução. Projeta-se neste nível com a esperança que várias soluções de várias partes componham a solução total
     - Geralmente não requer muita experiência, simplesmente quebrar o problema em partes torna a manipulação do problema total mais fácil
     - Particularmente apropriada quando já existe um conjunto de componentes reutilizáveis. Conhecendo o que eles fazem e como podem ser compostos traz *[illegible]* sobre a estruturação da aplicação como um todo

Concepção Arquitetural
Ferramentas Fundamentais

- Escolhendo o nível e os termos do discurso:

  2) Trabalhar em um nível mais alto que o da aplicação como um todo:

  - Pode significar resolver um problema mais genérico
  - Ex: tratamento de entradas complicadas de dados. Melhor utilizar uma ferramenta genérica de *ehne* (lex, ANTLR, etc) do que desenvolver um componente específico
  - Embora mais poderoso que o necessário, sua maturidade tecnológica e disponibilidade de pacotes reutilizáveis que implementam a solução resultam na utilização de um componente de menor tamanho, mais rápido e *boi uc nuht nid* ( )
  - Ex: projeto de sistema para declaração de imposto de renda onde as regras mudam de ano para ano. Construir um sistema específico ou utilizar uma planilha de cálculo parametrizada ?

Concepção Arquitetural
Ferramentas Fundamentais

- Separação de *obf onef h*
  - Sub-divisão do problema em partes independentes
  - Ex: a interface de usuário de uma é independente da lógica utilizada para controlar as operações bancárias
  - Se os conceitos são inerentemente independentes a separação é trivial
  - É mais difícil quando os problemas estão aparentemente (ou de fato) entrelaçados
  - Estes entrelaçamentos ocorrem por:
    - Abuso de linguagem: termo único (ex: conta) descrevendo diferentes partes da aplicação
    - Questões de eficiência: ex: *h eabr on n h*em bancos de dados

Concepção Arquitetural
Ferramentas Fundamentais

- Separação de *oblf onfe h*
  - Um exemplo importante é a divisão da estrutura do sistema em componentes (*dbod* de computação) e conectores (*dbod* de comunicação)
  - Frequentemente envolve muitos *œ snubi h*
  - Independência total de *oblf onfe h* pode ser impossível, requerendo uma análise dos *œ snubi h* de desempenho, custo, aparência e funcionais das concepções alternativas

Concepção Arquitetural
Ferramentas: Experiência Refinada

- Ao mesmo tempo em que as ferramentas fundamentais são de uso inegável geralmente representam um guia insuficiente para o projetista
- Experiência pode ser um poderoso aliado para manejar as ferramentas fundamentais de modo efetivo
- Realizar projetos por si só não produz a maturidade necessária para lidar com novos problemas
- É necessário uma reflexão e refinamento posteriores de modo a compreender os problemas essenciais e lições aprendidas

Concepção Arquitetural
Ferramentas: Experiência Refinada

- A meta é a escolha criteriosa de técnicas apropriadas ao contexto em questão
- Inclui análise de lições de sucesso mas também de falhas
- As lições não se restringem àquelas do próprio projetista:
  - Ampla leitura técnica e investigação de boas soluções
  - Conferências técnicas
- Não há virtude alguma em reinventar a roda

Concepção Arquitetural
Ferramentas: Experiência Refinada

- Quando se possui experiência técnica:
  - É mais fácil encontrar o conjunto inicial de arranjos viáveis alternativos
  - Não se gasta tempo em pontos para os quais já existe uma solução validada
  - Facilita a comunicação na equipe de desenvolvimento
- Entretanto não é livre de riscos:
  - “Quando se está com o seu martelo favorito tudo passa a se parecer com um prego”. Gera-se sistemas que funcionam, mas deficientes ou sub-ótimos
  - As limitações também são importadas
  - Inibe a inovação

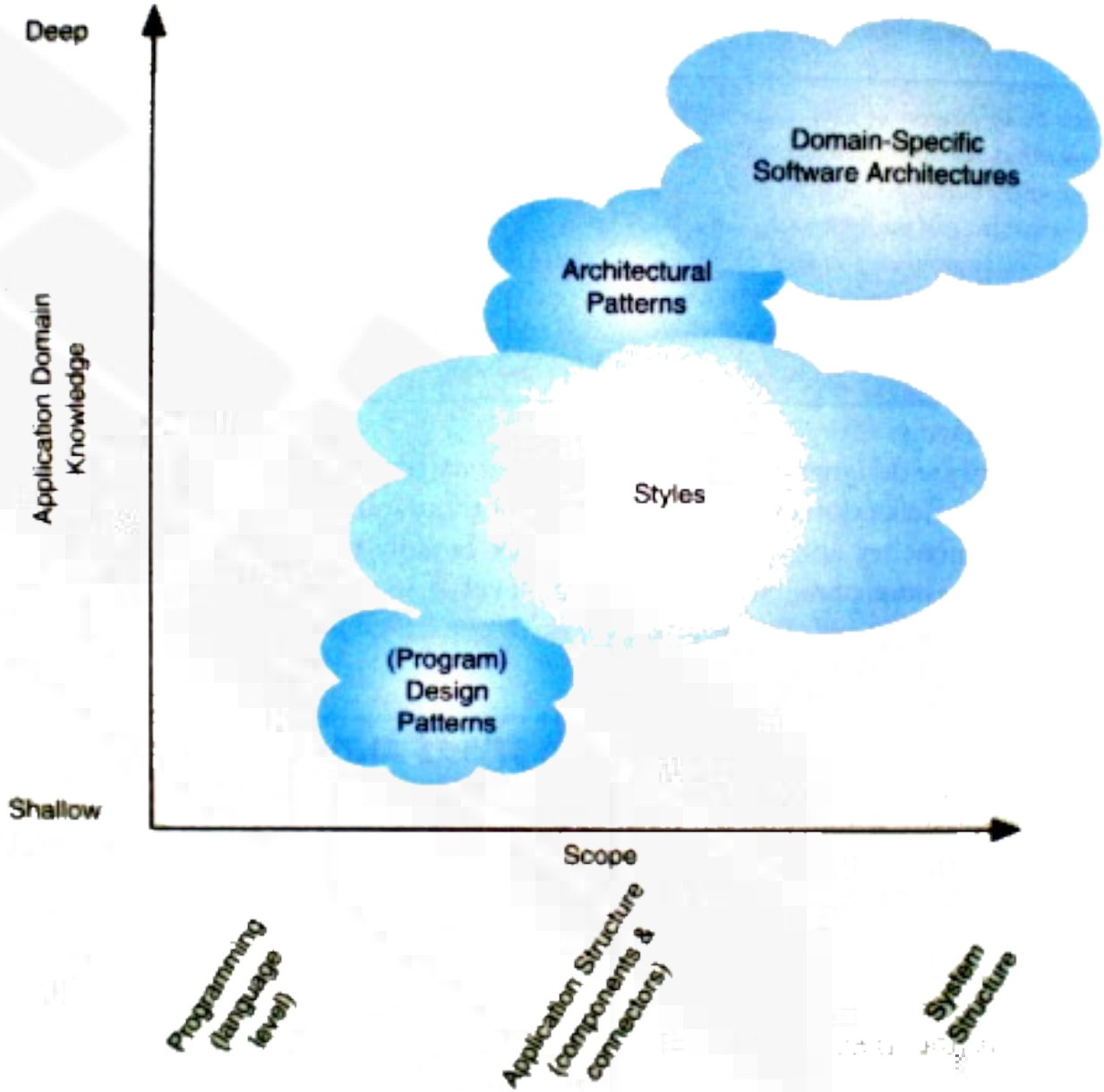
Estilos e Padrões Arquiteturais

Estilos e padrões arquiteturais são projetados para capturar o conhecimento de projetos efetivos no alcance de metas específicas dentro de um contexto particular de aplicações

- Exemplo na construção civil:
  - Uma determinada combinação de metas e contexto pode demandar a construção de uma casa para uma família pequena numa região de clima mediterrâneo utilizando pedras e telhas como materiais básicos
  - O estilo apropriado seria *[illegible]*
- Exemplo em *[illegible]*
  - Metas e contexto demandam o desenvolvimento de um sistema de *[illegible]* operando entre *[illegible]* remotos de uma empresa
  - O estilo apropriado seria *[illegible]*

Estilos e Padrões Arquiteturais

- Estilos e padrões podem ser caracterizados tanto para problemas pequenos quanto mais complexos



Estilos e Padrões Arquiteturais

- As bordas do diagrama anterior não são precisamente definidas
- O eixo *hob* mão é genuinamente linear ou totalmente ordenado
- O que um arquiteto denomina *se b e l anc e d* pode ser chamado de *hcdb e l anc e p* por outro

Estilos e Padrões Arquiteturais
Domain-Specific Software Architectures

- É uma combinação de:
  - Uma *e l cnc e sn ei nef d* para o domínio de aplicação
  - Uma biblioteca de componentes de *hbi c e,n* para esta arquitetura de referência, contendo partes reutilizáveis da *n nec m* no domínio
  - Um método para escolher e configurar componentes para trabalhar em uma *lf hcf d* da arquitetura de referência
- Representam a forma mais útil de experiência na descoberta dos arranjos alternativos de projeto

Estilos e Padrões Arquiteturais
Domain-Specific Software Architectures

- Exemplo na construção civil:
  - Projeto genérico de casas populares:
    - São instanciadas dezenas ou centenas de vezes
    - Podem variar em relação à marcenaria, estilos das colunas, cores, tapeçaria, material do telhado, etc
    - Entretando são casas similares em relação à sua planta baixa, localização das janelas e todos os outros elementos estruturais principais
    - Pode-se instanciar esta planta genérica seguindo as preferências e escolhas do cliente, gerando uma estrutura que além de viável financeiramente é, de alguma forma, personalizada

Estilos e Padrões Arquiteturais
Domain-Specific Software Architectures

- Exemplo em *hbic en*
  - *bic* para eletrônicos de consumo:
    - Aplicações para diferentes dispositivos eletrônicos podem ser criadas através da parametrização ou especialização da arquitetura central
    - Isto pode implicar na adição de componentes nunca antes utilizados, porém estas adições são realizadas em partes da arquitetura que foram previamente projetadas para este fim
  - Embora as empresas desenvolvam aplicações com grandes similaridades de estrutura, infelizmente estas não são documentadas, permanecem latentes na memória dos engenheiros

Padrões Arquiteturais

- Padrões arquiteturais são semelhantes às *h*, porém aplicados em um escopo bem mais específico

Padrão Arquitetural: coleção identificada de decisões arquiteturais de projeto que são aplicáveis a um problema recorrente de desenvolvimento e parametrizadas de modo a serem aplicadas em qualquer contexto de desenvolvimento de *hbic* *e* mo qual o problema aparece

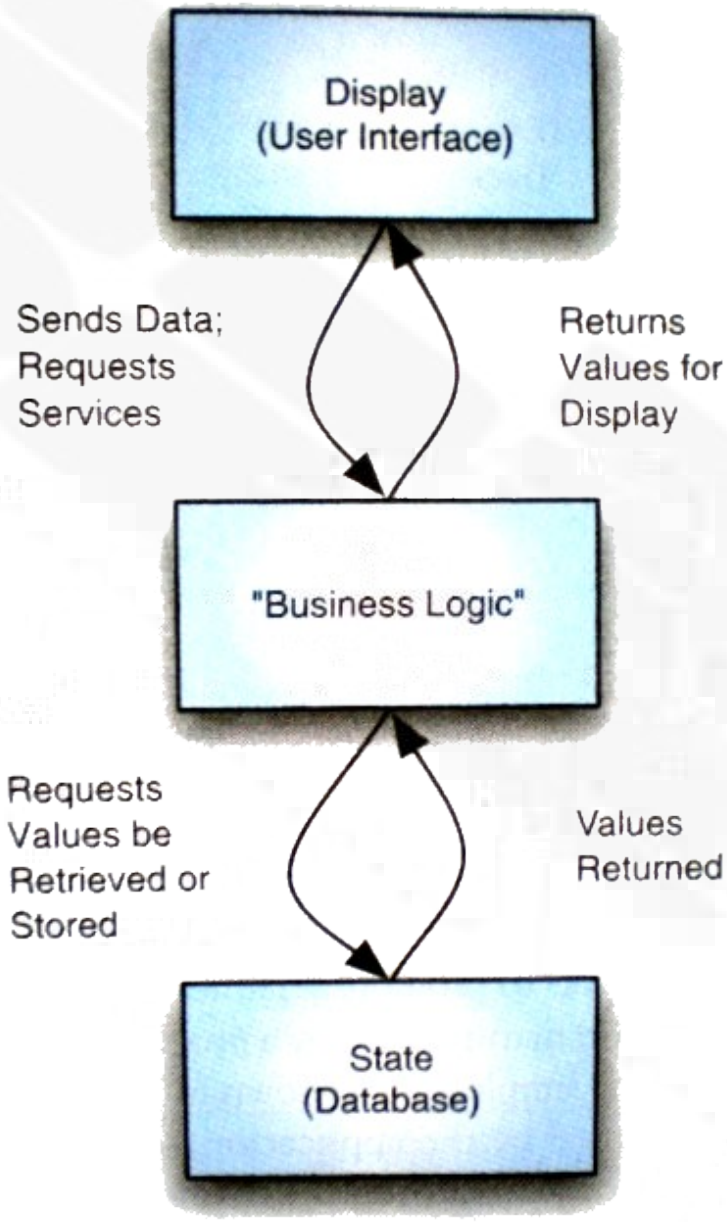
Padrões Arquiteturais

State-Logic-Display (Three-Tier)

- Comumente empregado em sistemas de informação:
  - *c cbon* + Lógica de Negócio + Interface de Usuário
- É facilmente mapeado em uma implementação distribuída com comunicação via *en bon ebons en o odl*
- Jogos - *dtu d n* em rede
- Sistemas *n m*

Padrões Arquiteturais

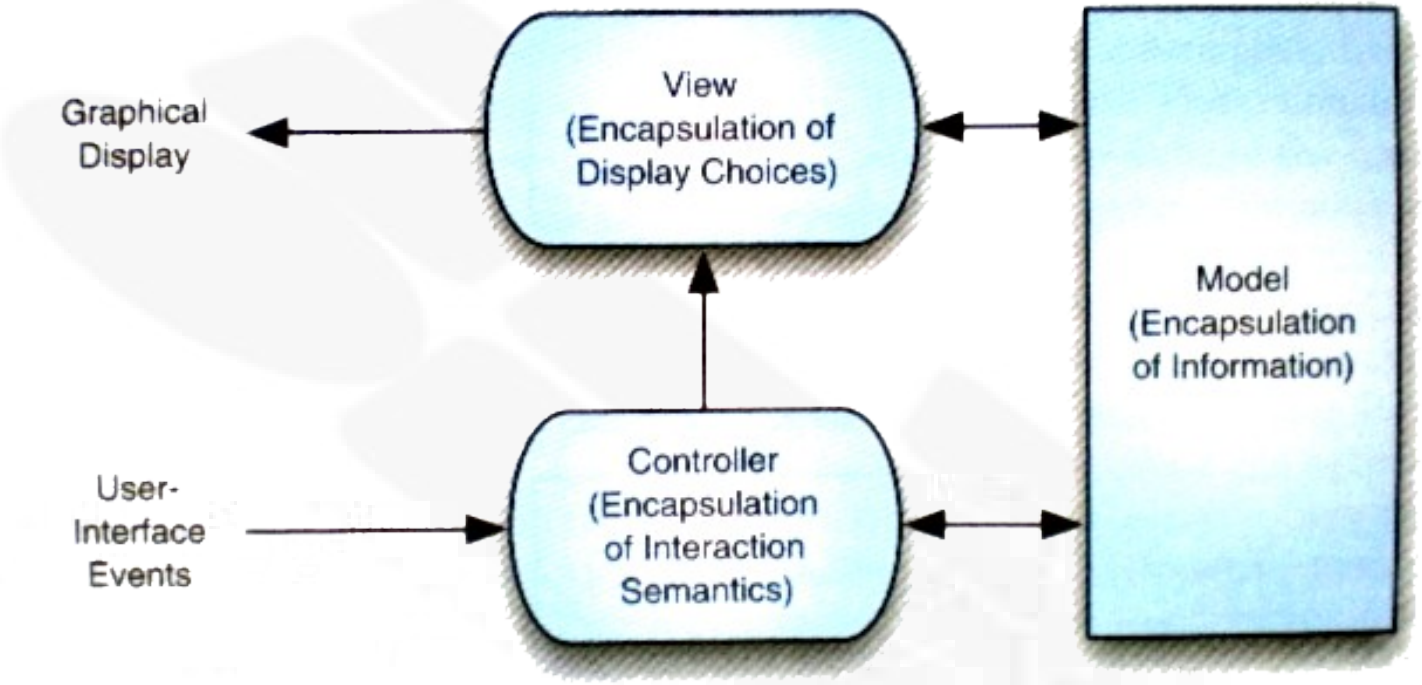
State-Logic-Display (Three-Tier)



Padrões Arquiteturais
Model-View-Controller (MVC)

- Utilizado desde a década de 80 para o projeto de interfaces gráficas de usuário
- Pode também ser visto como um *snlhrf conf*
- Promove a separação e, a consequente independência de desenvolvimento, da informação manipulada pelo programa e interações do usuário com esta informação

Padrões Arquiteturais
Model-View-Controller (MVC)



- *bsnd* encapsula a informação usada pela aplicação
- *l n* : encapsula artefatos necessários à descrição gráfica da informação
- *lf œbodh* encapsula a lógica necessária à manutenção da consistência entre o - *bsnd* o *l n* . É responsável pelo processamento dos eventos do usuário

Padrões Arquiteturais
Model-View-Controller (MVC)

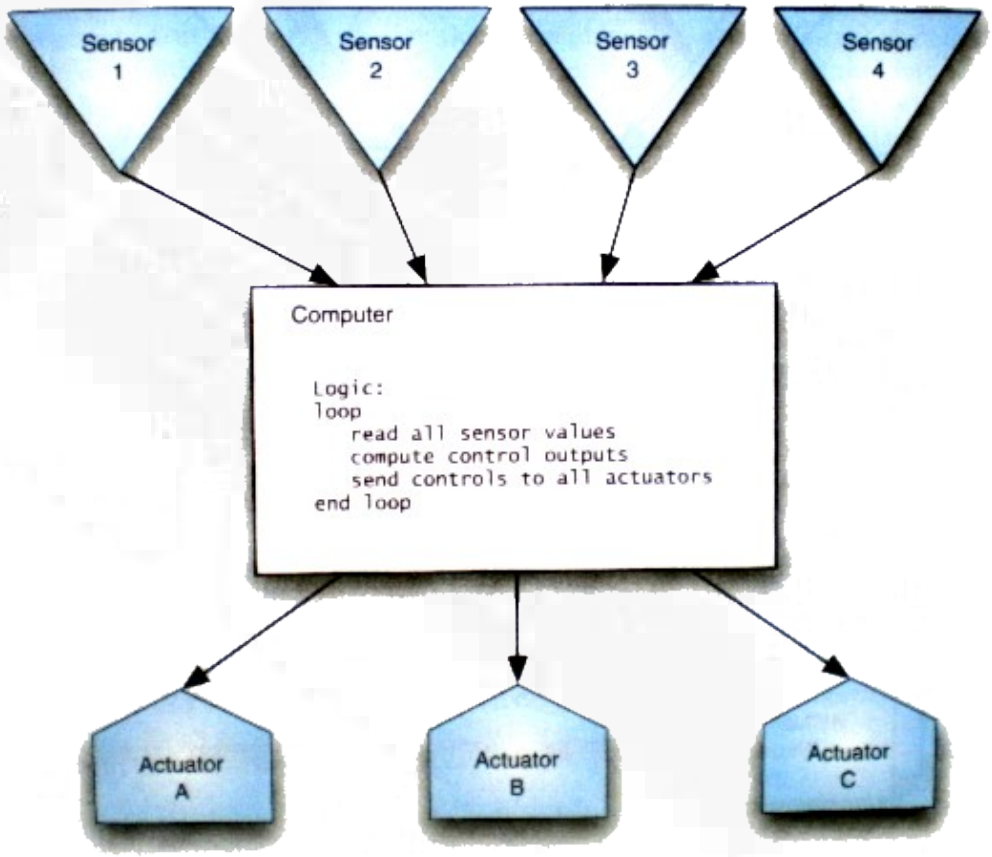
- Colaborações entre os componentes (variações são consideradas na prática):
  - Quando a aplicação modifica um valor no *bsnd* uma notificação é enviada à(s) *l n h* de modo que a representação gráfica seja atualizada e re-exibida
  - Notificações também podem ser enviadas ao *obf œbdh e* que pode modificar a *l n* se necessário
  - A *l n* pode solicitar dados adicionais ao *bsnd*
  - O sistema de janelas envia os eventos do usuário ao *obf œbdh e* que pode consultar a *l n* , obtendo informações que ajudam a determinar a ação a ser tomada
  - Como consequência o *obf œbdh e* atualiza o *bsnd*

Padrões Arquiteturais
Model-View-Controller (MVC)

- Geralmente existe um acoplamento forte entre as ações da *l n* e do *olf cebdhe*, eventualmente justificando um *- ne n* destes componentes
- na *becls l s n n:m*
  - *bs nd* recursos *n m*
  - *l n*: agente de renderização HTML do *neb hne*
  - *lf cebdhe*: parte do *neb hne* que responde aos eventos do usuário e que interaje com o servidor *n m* ou modifica o que é exibido no *neb hne*. Pode também agregar código obtido do servidor *n* (ex: *ole*)*c*

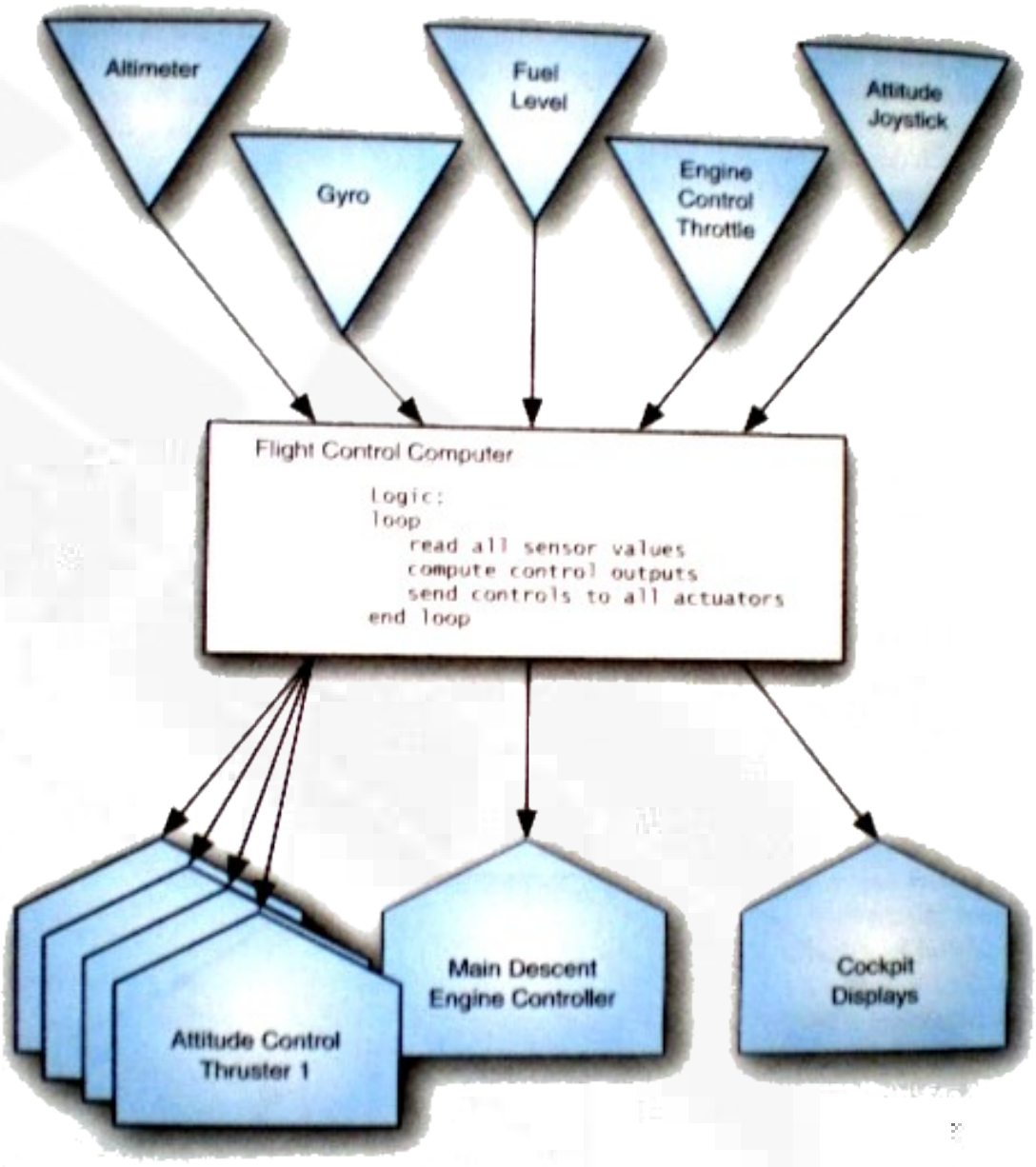
Padrões Arquiteturais
Sense-Compute-Control (ou *Sensor-Control-Actuator*)

- Tipicamente utilizado na estruturação de aplicações embarcadas de controle:
  - Exemplos: forno de micro-ondas, *d* *ne*, sistemas automotivos e robôs



Padrões Arquiteturais
Sense-Compute-Control (ou *Sensor-Control-Actuator*)

- Jogo de pouso lunar:
  - *nns mo* implícito através do ambiente



Introdução aos Estilos Arquiteturais

- Principal forma de caracterizar experiências em projeto de *hbio c en*
- Elemento chave no desenvolvimento da concepção inicial ou detalhada da arquitetura do sistema
- São amplamente aplicados, refletindo menos conhecimento de domínio do que os padrões arquiteturais
- A fronteira entre estilos e padrões arquiteturais pode não ser clara, entretanto

Introdução aos Estilos Arquiteturais

- Serão estudados:
  - As decisões e restrições que compõem o estilo arquitetural

Introdução aos Estilos Arquiteturais

- Estilos tradicionais influenciados por linguagens de programação
  - *lf edr e- f s nedo lf nh*
  - *mmocu lerf cns*
- Estilos em Camadas:
  - *l ecc d dlf nh*
  - *ldrf cu ne ne*
- Estilos Baseados em Fluxo de Dados
  - *cdo u n rf lc d*
  - *l nuf sul odne*

Introdução aos Estilos Arquiteturais

- Estilos com Memória Compartilhada:
  - d o rb es
  - ohu hns nec hon-
- Estilos Baseados em Interpretadores:
  - lh o f one enone
  - b hmoh bsn
- Estilos Baseados em Invocação Implícita:
  - hdht u rhoe m
  - rf ou hns
- nneudbu nne

Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

- Linguagens tais como C, C++, Java e Pascal podem ser utilizadas para implementar arquiteturas de qualquer estilo
- Alguns estilos, entretanto, refletem os relacionamentos básicos de organização e controle de fluxo entre componentes disponibilizados por estas linguagens:
  - *lf edr e- f s nebo tf nh*
  - *mmoou terf ons*

Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

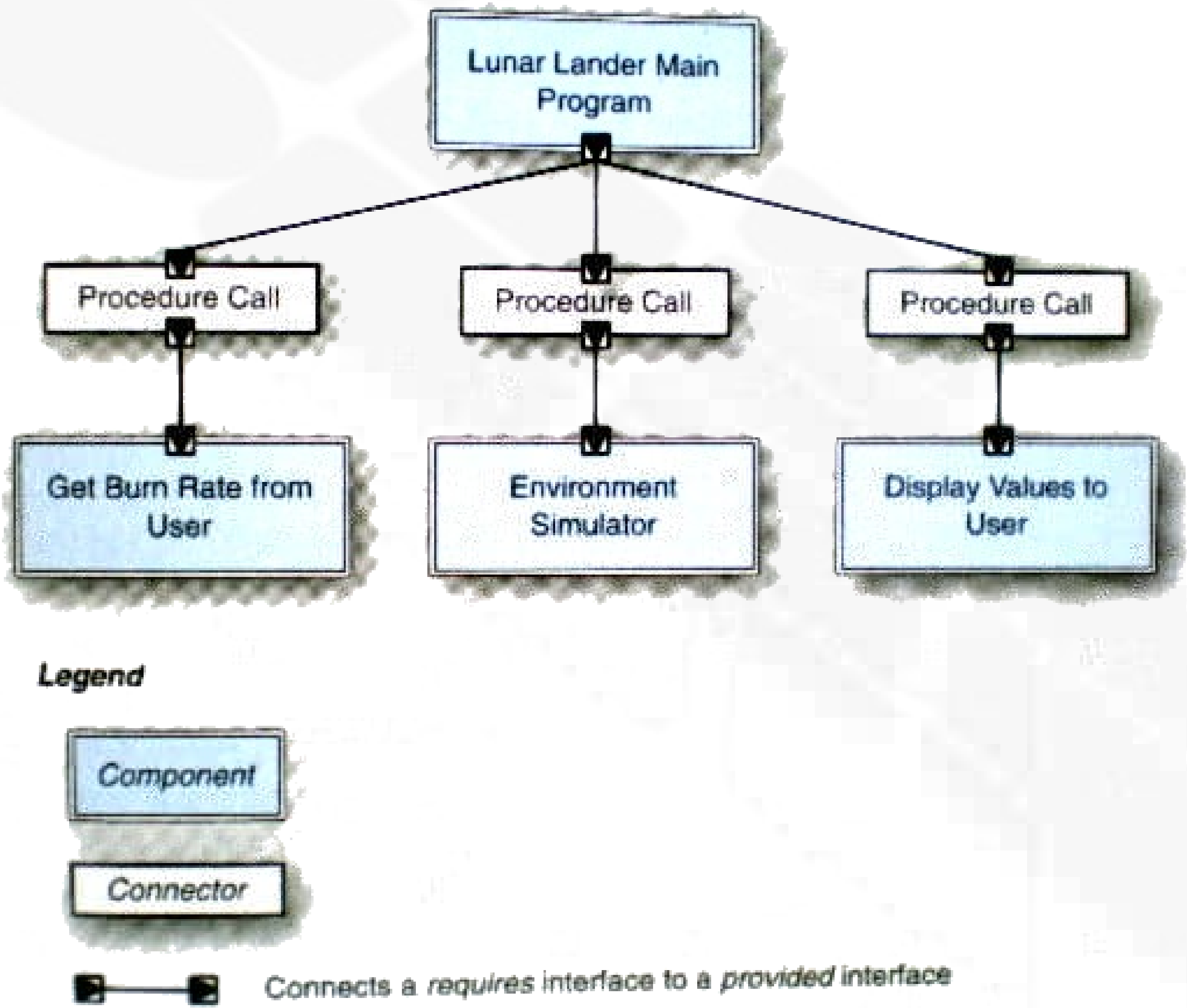
- *lf edr e- f s ndo lf nh*

Resumo: decomposição baseada na separação de passos funcionais de processamento
Componentes: programa principal e sub-rotinas
Conectores: *i f oclf ebons en o odh*
Elementos de Dados: parâmetros e valores de retorno utilizados nas sub-rotinas
Topologia: organização estática e hierárquica de componentes; grafo direcionado
Restrições Adicionais: nenhuma
Qualidades Induzidas: modularidade – sub-rotinas podem ser substituídas por outras com implementação diferente, desde que a semântica da interface não mude
Usos Típicos: programas pequenos e de propósito pedagógico
Precauções: não é escalável para grandes aplicações; atenção inadequada às estruturas de dados; imprevisibilidade na determinação do esforço necessário para acomodar novas mudanças
Relacionamento com Linguagens de Programação e Ambientes: linguagens de programação imperativas, tais como BASIC, Pascal ou C

Estilos Arquiteturais

Tradicionais influenciados por linguagens de programação

- *lf ebr e- f s neb tf nh*(pouso lunar):



Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

- *mmoau lerf ons* (OO):
  - A única estrutura disponibilizada é um conjunto de objetos cujo tempo de vida varia de acordo com os seus usos
  - Compreender um programa OO requer entender os numerosos relacionamentos estáticos e dinâmicos entre os objetos

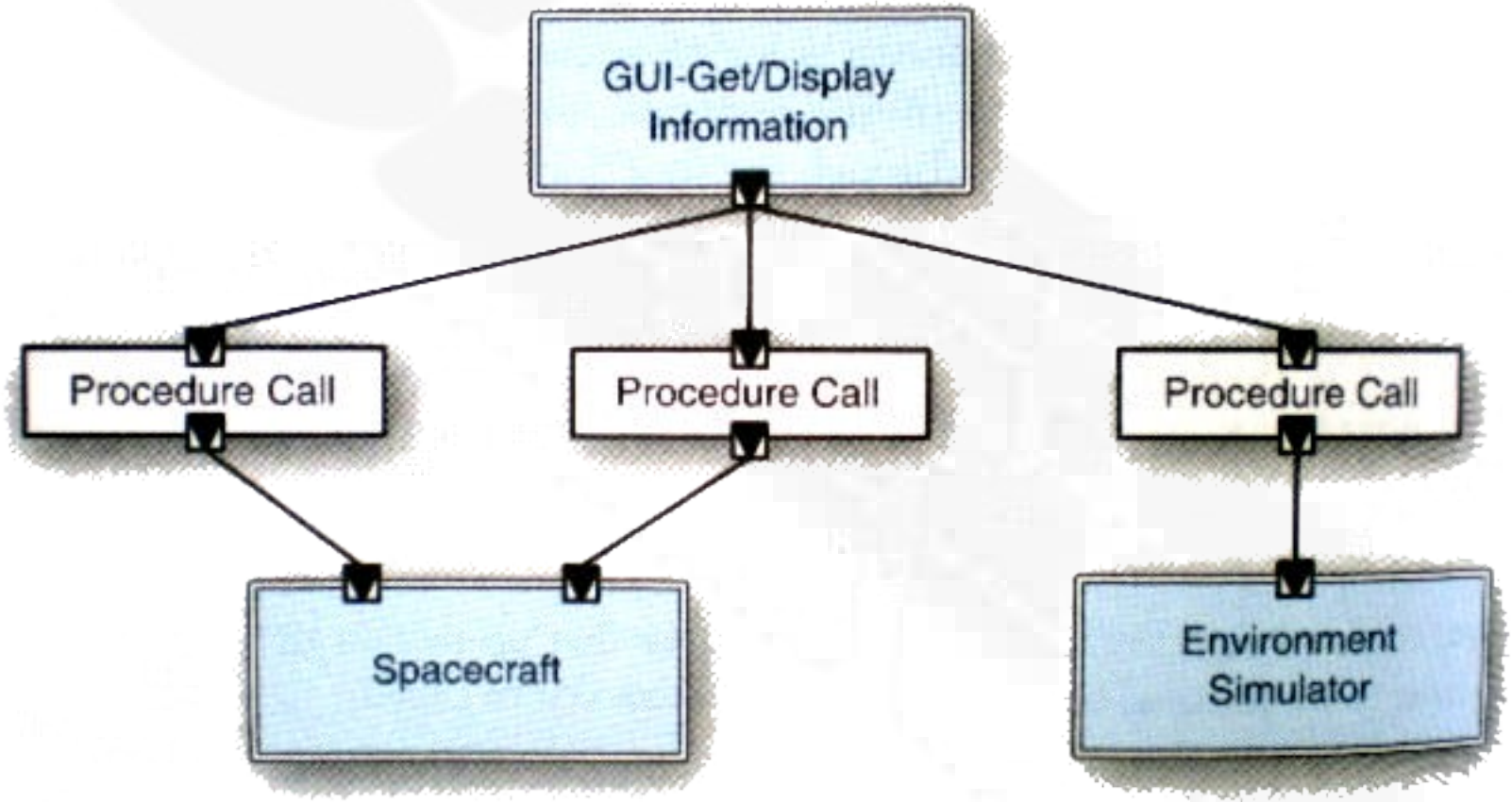
Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

- *mmoau lerf ons*(OO):

Resumo: estado fortemente encapsulado com funções que operam neste estado, sob a forma de objetos. Objetos devem ser instanciados antes que seus métodos sejam invocados
Componentes: objetos (instâncias de uma classe)
Conectores: invocações de métodos (*ebons en o dh*que manipulam estado)
Elementos de Dados: argumentos de métodos
Topologia: pode variar arbitrariamente; componentes podem compartilhar dados e interfaces de funções através de hierarquias de herança
Restrições Adicionais: geralmente memória compartilhada (para ponteiros) e *lhf r dhuc en sns*
Qualidades Induzidas: integridade de operações nos dados – dados são manipulados somente por funções apropriadas. Abstração – detalhes de implementação estão ocultos
Usos Típicos: quando deseja-se um relacionamento forte entre entidades do mundo físico e do programa; propósitos pedagógicos; sistemas com estruturas de dados complexas e dinâmicas
Precauções: uso em sistemas distribuídos requer alguma solução de *- l ssdh e*,relativamente ineficiente para aplicações de alto-desempenho com grandes estruturas de dados; ausência de princípios estruturantes adicionais pode resultar em aplicações altamente complexas
Relacionamento com Linguagens de Programação e Ambientes: Java, C++

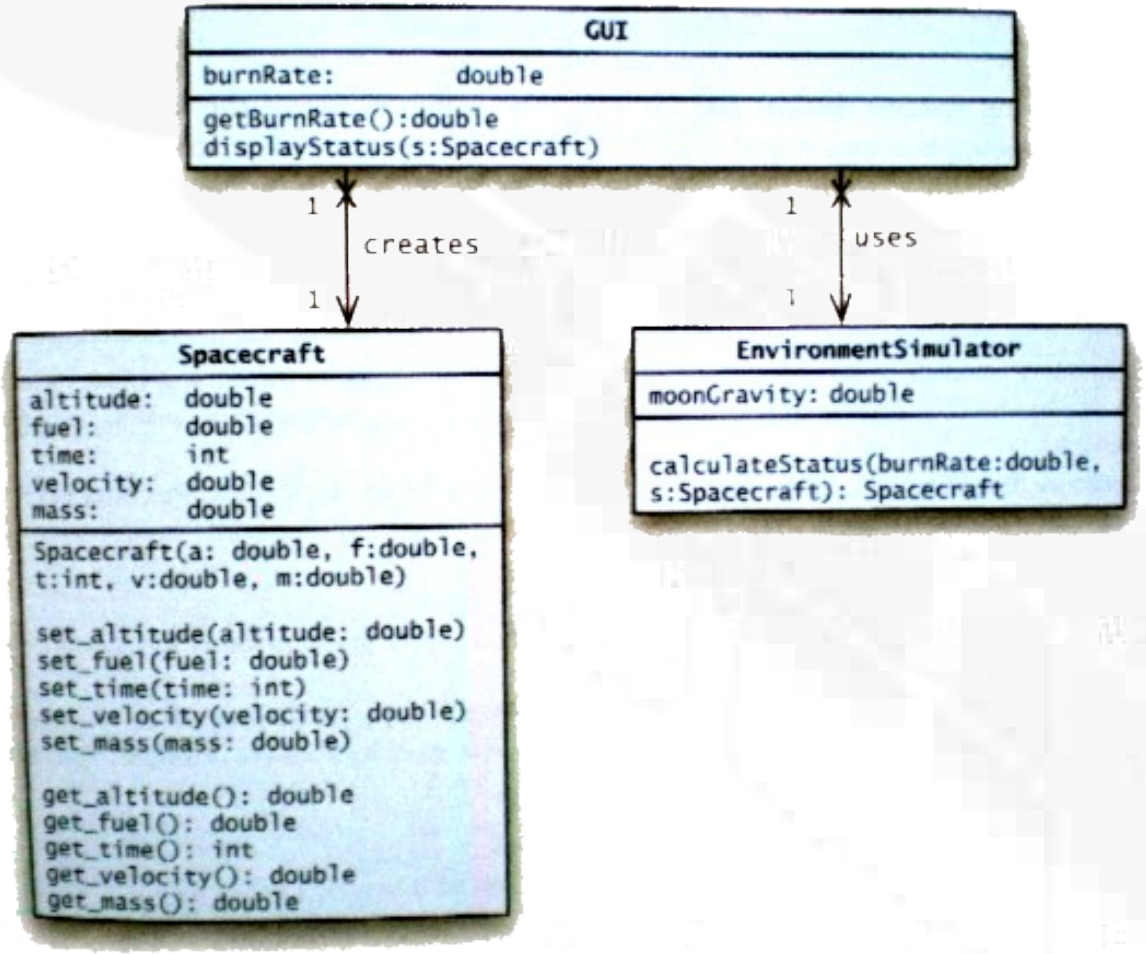
Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

- *mmoau lerf ons* (OO) (pouso lunar):



Estilos Arquiteturais
Tradicionais influenciados por linguagens de programação

- *mmoau lerf ons* (OO) (pouso lunar):



Estilos Arquiteturais
Em Camadas

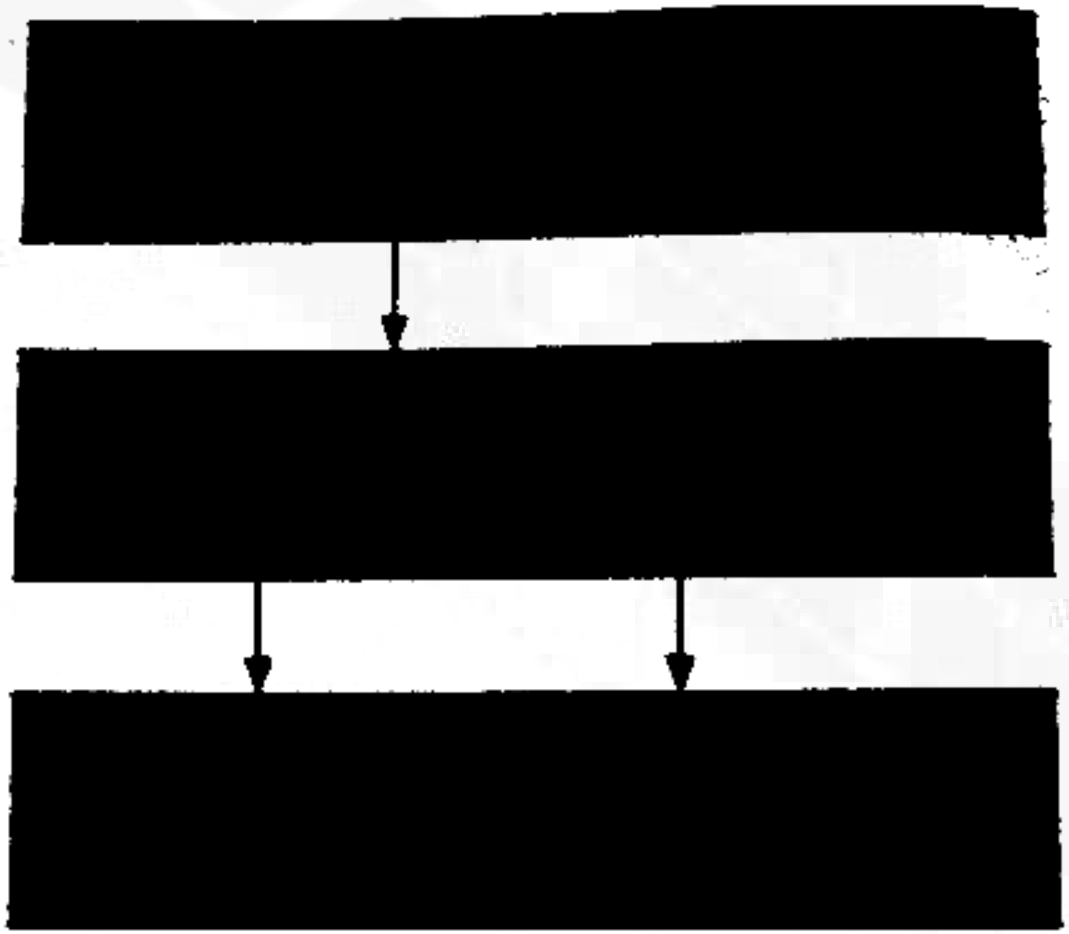
- A arquitetura é separada em camadas ordenadas, onde um programa de uma camada pode solicitar serviços de uma camada inferior
- Exemplos:
  - Arquiteturas de sistemas operacionais
  - *ldrf au ne ne*

Estilos Arquiteturais
Em Camadas

- *l e c d o l f n h*
  - Uma camada oferece um conjunto de serviços *e o l s n h l f c n i e o n* que podem ser utilizados por programas que residem na(s) camada(s) acima
  - Os serviços podem ser implementados por diversos programas dentro da camada porém, para os clientes destes serviços, tal distinção não é aparente

Estilos Arquiteturais
Em Camadas

- *l e c d o l f n h* (exemplo):



Estilos Arquiteturais
Em Camadas

- *l ec d o lf nh* (classificação):
  - Máquina Virtual Estrita: programas de um determinado nível somente podem acessar serviços providos pela camada imediatamente inferior
  - Máquina Virtual Não-Estrita:programas de um determinado nível podem acessar serviços de qualquer cada abaixo do nível em questão
- Exemplo 1: camadas de um sistema operacional:
  - Aplicações do usuário (nível 1)
  - Serviço de manipulação de arquivos e diretórios (nível 2)
  - *e nd* de disco e gerenciamento de volume (nível 3)
- Exemplo 2: protocolos de comunicação em rede

Estilos Arquiteturais
Em Camadas

- *l ec d o lf nh*

Resumo: sequência ordenada de camadas; cada camada (ou máquina virtual) oferece um conjunto de serviços que podem ser acessados por programas (sub-componentes) de uma camada acima
Componentes: camadas oferecendo serviços para outras camadas, tipicamente compostas de vários programas (sub-componentes)
Conectores: tipicamente *ebons en o odh*
Elementos de Dados: parâmetros que transitam entre as camadas
Topologia: linear para máquinas virtuais estritas e grafo direcionado acíclico em interpretações mais fracas
Restrições Adicionais: nenhuma
Qualidades Induzidas: estrutura de dependência clara; componentes em uma camada superior são imunes a modificações das camadas inferiores desde que as especificações do serviço não mudem; componentes em uma camada inferior são totalmente independentes de camadas superiores
Usos Típicos: projeto de sistemas operacionais, pilhas de protocolos de rede
Precauções: máquinas virtuais estritas com muitos níveis podem ser relativamente ineficientes

Estilos Arquiteturais
Em Camadas

- *l œc d ø lf nh* (pouso lunar):

Estilos Arquiteturais
Em Camadas

- *ldrf cu ne ne*
  - Máquina Virtual de duas camadas com conexões em rede
  - O servidor é a máquina virtual abaixo dos clientes
  - Múltiplos clientes podem acessar o servidor
  - Os clientes são independentes
  - Pode-se utilizar *tclf* (*tcl o*) *oddrf dh*
  - Exemplo: máquina de saque eletrônico ( )

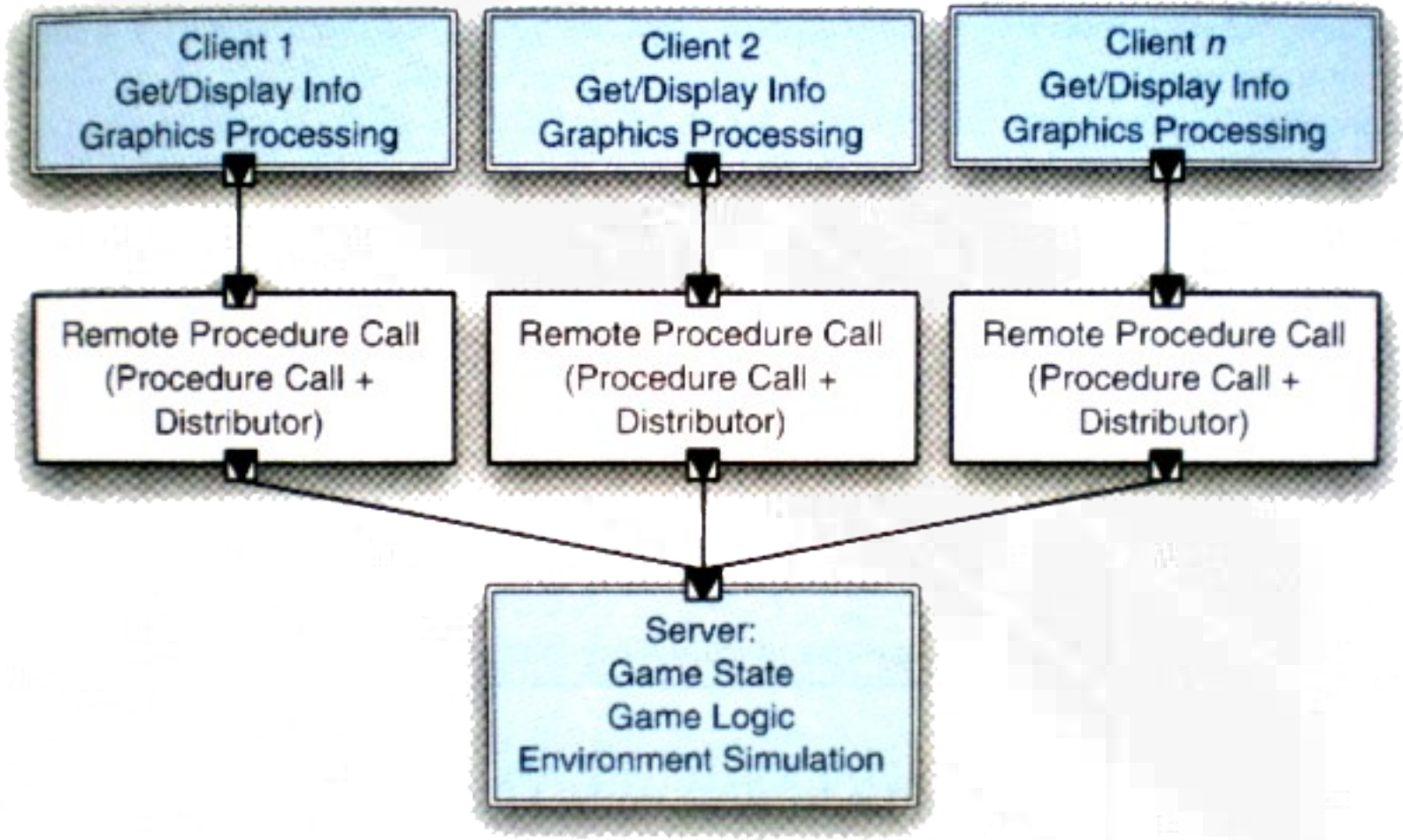
Estilos Arquiteturais
Em Camadas

- *ldrf cu ne ne.e*

Resumo: clientes enviam requisições de serviço ao servidor, que realiza as operações requeridas e responde conforme necessário com as informações solicitadas. Toda comunicação é iniciada pelos clientes
Componentes: clientes e servidor
Conectores: *en- bon ebons en o dh* protocolos de rede
Elementos de Dados: parâmetros e valores de retorno conforme enviados pelos conectores
Topologia: em dois níveis, com múltiplos clientes realizando requisições ao servidor
Restrições Adicionais: não existe comunicação entre clientes
Qualidades Induzidas: centralização da computação e dos dados no servidor, que torna a informação disponível para os clientes. Um único servidor poderoso pode servir muitos clientes
Usos Típicos: onde é necessário centralização de dados; onde processamento e armazenamento de dados se beneficiam de uma máquina de alta capacidade; e onde clientes realizam geralmente tarefas simples de interface de usuário, tais como em diversos sistemas de informação
Precauções: quando a largura de banda é limitada e existe um grande número de clientes

Estilos Arquiteturais
Em Camadas

- *ldrf au ne n*(pouso lunar):

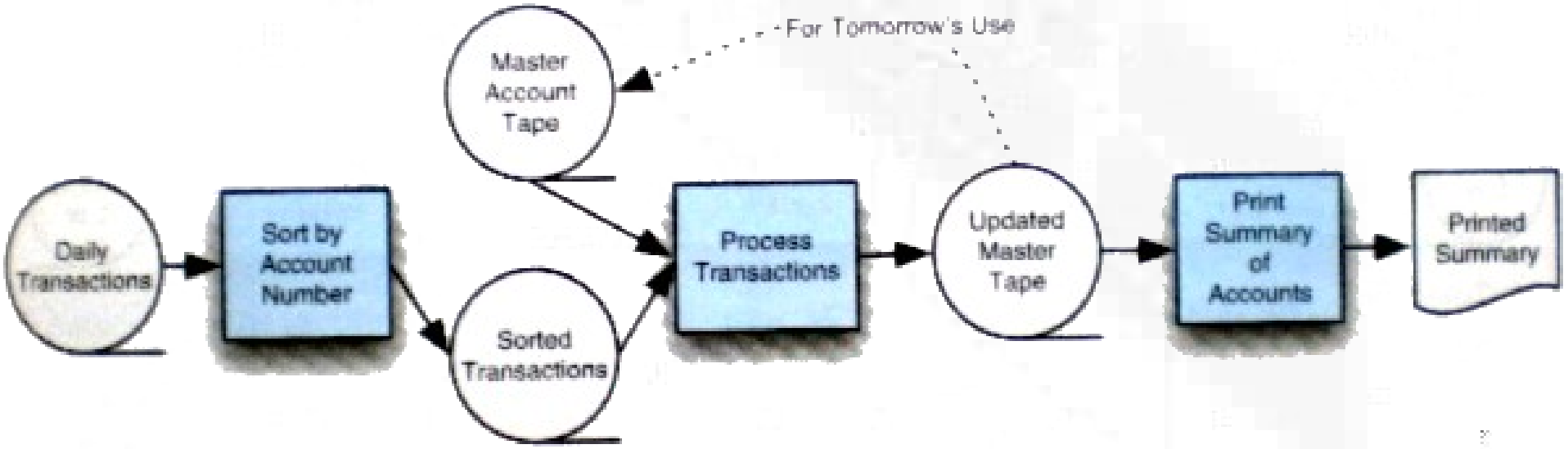


Estilos Arquiteturais
Baseados em Fluxo de Dados

- Caracterizado pelo movimento de dados entre elementos independentes de processamento

Estilos Arquiteturais
Baseados em Fluxo de Dados

- [illegible]
  - Um dos estilos arquiteturais mais antigos: as limitações dos equipamentos exigiam que o problema fosse sub-dividido em componentes que se comunicavam através da transferência de fitas magnéticas
  - Exemplo: atualizar um registro bancário de todas as contas



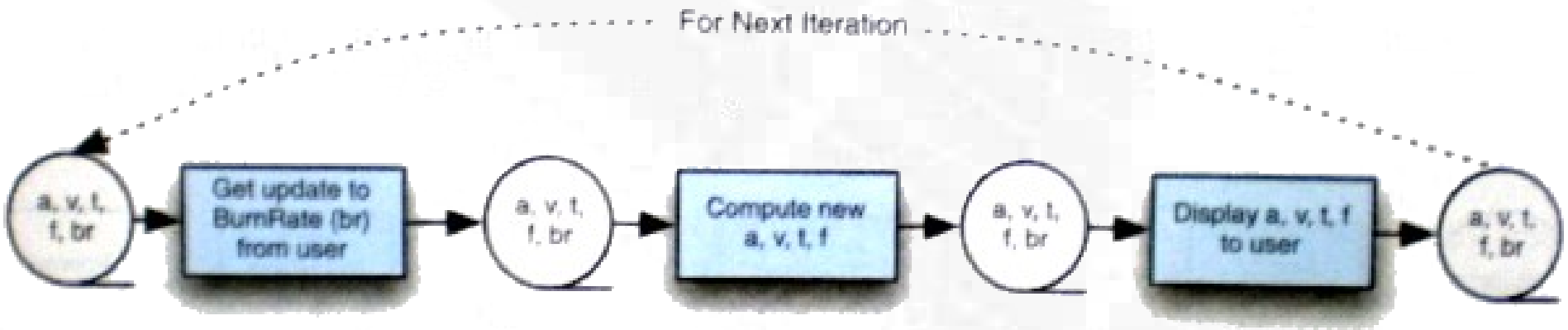
Estilos Arquiteturais
Baseados em Fluxo de Dados

- *cotunnflc:d*

Resumo: programas distintos são executados em ordem; os dados são passados, sob a forma de blocos (agregados), de um programa para o próximo
Componentes: programas independentes
Conectores: mãos humanas que carregam as fitas entre os programas (*hinnefunc*)
Elementos de Dados: elementos agregados explícitos que, após o término da execução de um componente, são repassados deste para o próximo componente
Topologia: linear
Restrições Adicionais: um único programa executa por vez até o seu término
Qualidades Induzidas: execuções separáveis; simplicidade
Usos Típicos: processamento de transações em sistemas financeiros
Precauções: quando interação entre componentes é requerida; quando concorrência entre componentes é possível ou necessário
Relacionamento com Linguagens de Programação e Ambientes: nenhum

Estilos Arquiteturais
Baseados em Fluxo de Dados

- Exemplo (pouso lunar):
  - Indica quão inapropriado é este estilo para a aplicação



Estilos Arquiteturais
Baseados em Fluxo de Dados

- *Pipe and filter*
  - Os filtros podem operar de forma concorrente, não é necessário aguardar o término do produtor para que o componente que consome a saída do produtor inicie o seu funcionamento

Estilos Arquiteturais
Baseados em Fluxo de Dados

- *I nuf sul odn.e*

Resumo: programas distintos são executados, potencialmente de forma concorrente; os dados são passados, sob a forma de fluxo, de um programa para o próximo
Componentes: programas independentes (filtros)
Conectores: roteadores explícitos de fluxos de dados (*I nh*), possivelmente é um serviço disponibilizado pelo sistema operacional ou pela linguagem de programação
Elementos de Dados: não definidos explicitamente, porém devem ser *hoen - h*ineares de dados
Topologia: *I nkdf n*embora bifurcações sejam possíveis
Qualidades Induzidas: filtros são mutualmente independentes. Estruturas simples de chegada e saída de fluxos de dados facilitam novas combinações de filtros
Usos Típicos: programação de aplicações baseadas em primitivas de sistemas operacionais
Precauções: quando estruturas complexas de dados precisam ser transferidas entre componentes; quando interatividade entre os programas é necessário
Relacionamento com Linguagens de Programação e Ambientes: Unix shell

Estilos Arquiteturais
Com Memória Compartilhada

- Caracterizado pela presença de múltiplos componentes que acessam o mesmo repositório de dados (*shared-data*) e se comunicam através deste repositório
- Semelhante ao uso de dados globais porém, nestes estilos, o centro de atenção no projeto é explicitamente direcionado para este repositório
- O repositório é bem ordenado e cuidadosamente gerenciado

Estilos Arquiteturais
Com Memória Compartilhada

- *d o rb es*
  - Tem sua origem nas aplicações de Inteligência Artificial
  - Analogia:
    - Diversos peritos (*n ne*) sentados ao redor de uma mesa (*s c h* *be*) tentando cooperar na solução de um problema grande e complexo
    - Quando um perito reconhece que pode resolver alguma parte do problema que está na mesa ele recolhe este sub-problema, vai embora e trabalha na sua solução
    - Quando concluída a solução, o perito retorna e disponibiliza a solução na mesa
    - A disponibilização desta solução pode habilitar outro perito a resolver uma outra parte do problema
    - O processo continua até que todo o problema esteja resolvido

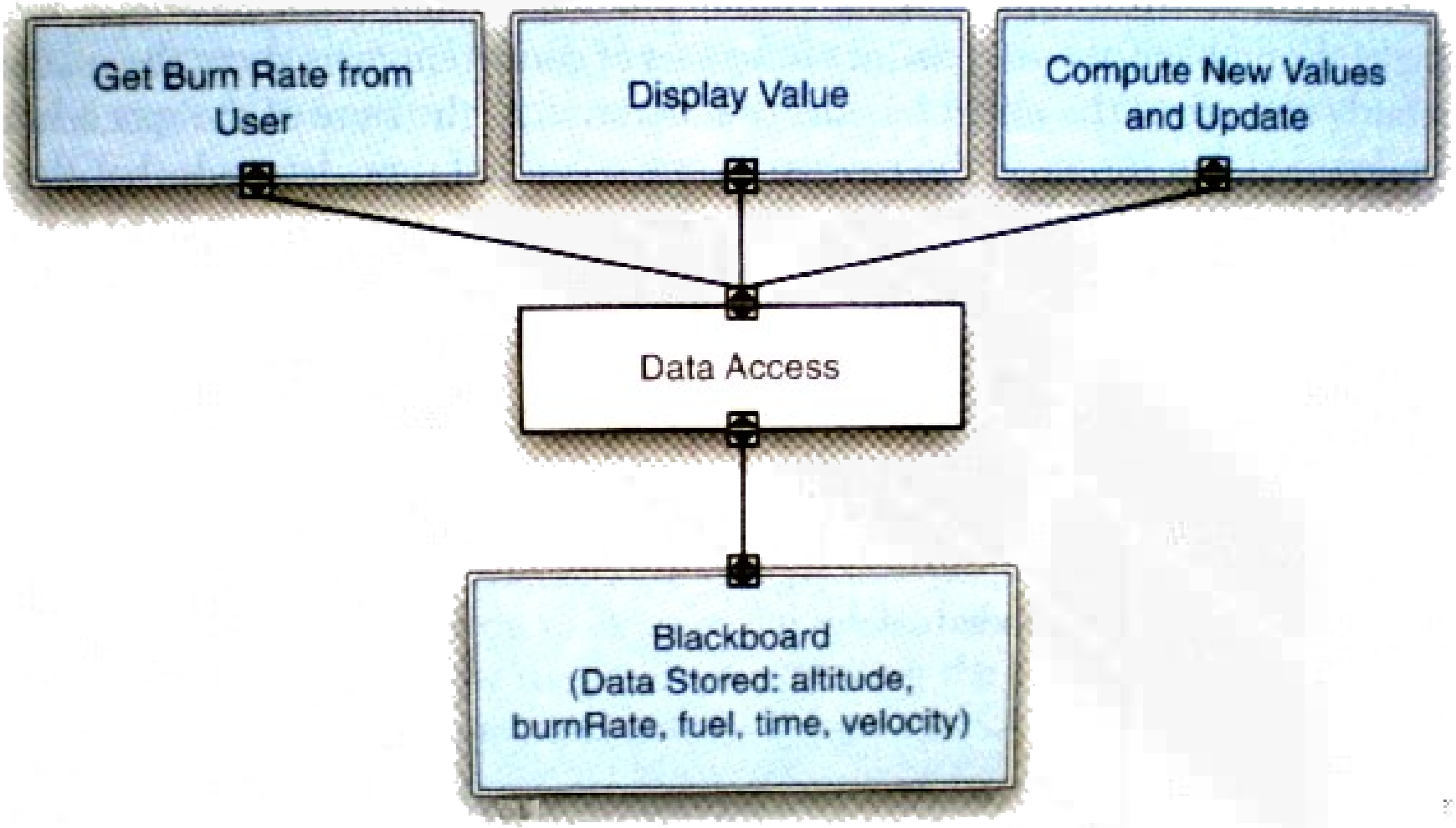
Estilos Arquiteturais
Com Memória Compartilhada

- *d o rb es*

Resumo: programas independentes acessam e se comunicam exclusivamente através de um repositório global de dados, conhecido como *rd o rb es*
Componentes: programas independentes (*f b dhsr n hb e*on)he *rd o rb es*
Conectores: acesso ao *rd o rb es*pode ser através de referência direta a memória, *ebons en o od*u uma consulta em um banco de dados
Elementos de Dados: dados armazenados no *rd o rb es*
Topologia: em estrela, com o *rd o rb es*no centro
Variações:) programas consultam o *rd o rb es*para verificar se algum valor de seu interesse mudou;) um *rd o rb es - f r n*e notifica atualizações do *rd o rb es*aos componentes interessados
Qualidades Induzidas: estratégias completas de solução para problemas complexos não precisam ser pré-planejadas, pois são determinadas por visões, em constante mudança, dos dados/problema
Usos Típicos: solução heurística de problemas em aplicações de Inteligência Artificial
Precauções: quando existe uma estratégia bem-estruturada de solução; quando as interações entre os programas independentes requerem coordenações complexas; quando as representações dos dados do *rd o rb es*mudam frequentemente (requerendo modificações em todos os participantes)
Relacionamento com Linguagens de Programação e Ambientes: versões que permitem concorrência entre os programas requerem primitivas para gerenciar o *rd o rb es*compartilhado

Estilos Arquiteturais
Com Memória Compartilhada

- *d o rb es*(pouso lunar :



Estilos Arquiteturais
Com Memória Compartilhada

- *dhu hns nœc hcn+:*
  - Tipo altamente especializado de arquitetura com memória compartilhada
  - A memória compartilhada funciona como uma base de conhecimento que contém fatos e regras de produção (cláusulas *li t rf* sobre o conjunto de variáveis)
  - A interface gráfica de usuário disponibiliza duas operações básicas:
    - Entrada de novos fatos e regras de produção
    - Entrada de consultas (*r b d*
  - Uma máquina de inferência opera na base de conhecimento em resposta às entradas do usuário

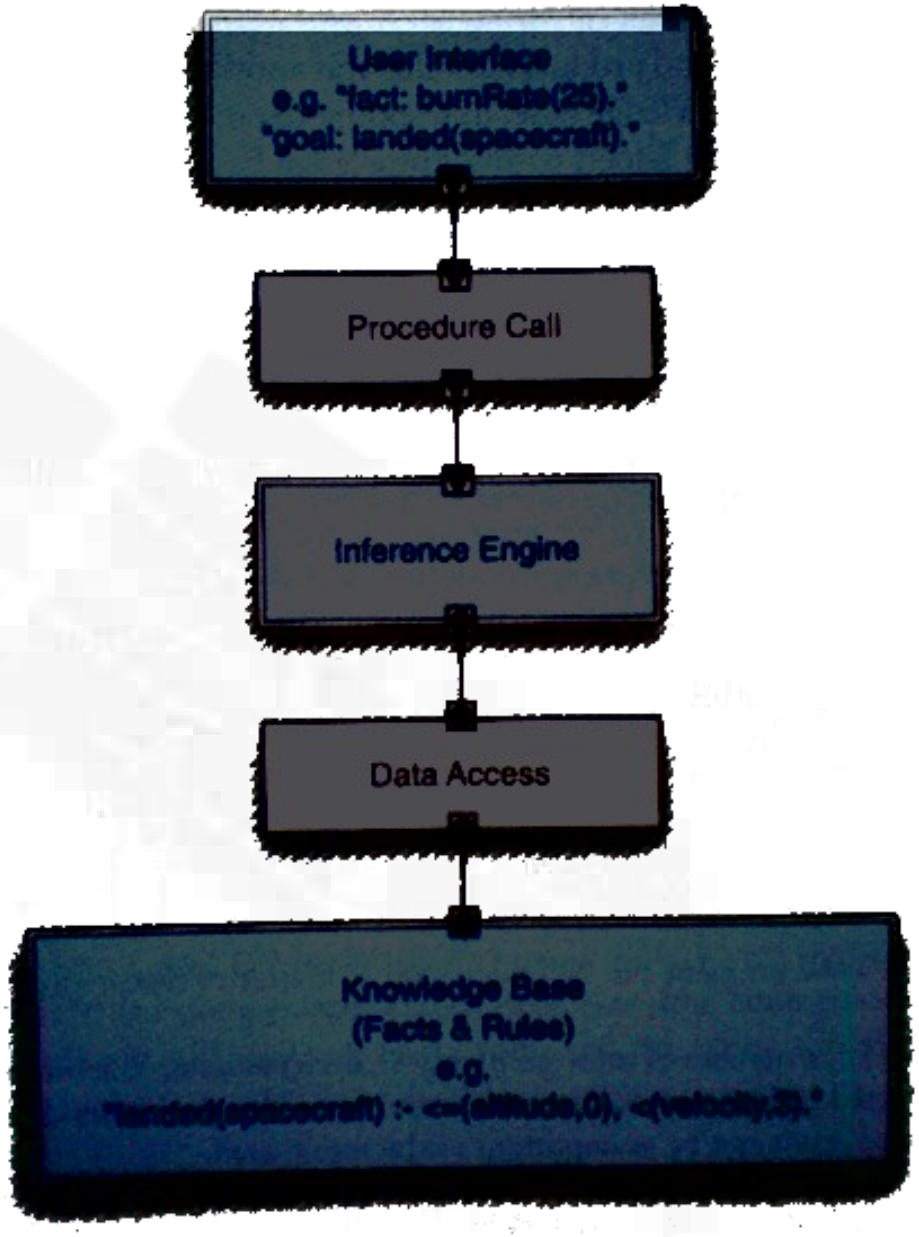
Estilos Arquiteturais
Com Memória Compartilhada

- *dhu hns nec hcn+:*

Resumo: a máquina de inferência analisa a entrada do usuário e determina se é um fato/regra ou consulta. Se for um fato/regra, esta entrada é adicionada à base de conhecimento. Caso contrário, é realizada uma consulta à base, buscando regras aplicáveis, com o objetivo de resolver a consulta
Componentes: interface de usuário, máquina de inferência e base de conhecimento
Conectores: componentes são fortemente inter-conectados com *ebons en o dhou s c oonhh*
Elementos de Dados: fatos e consultas
Topologia: três camadas fortemente acopladas
Qualidades Induzidas: o comportamento da aplicação pode ser facilmente modificado através da adição ou remoção dinâmica de regras na base de conhecimento; sistemas pequenos podem ser rapidamente prototipados; útil para explorar iterativamente problemas cuja abordagem para uma solução genérica não é clara
Usos Típicos: quando o problema pode ser visto como uma resolução sucessiva de predicados
Precauções: quando muitas regras estão envolvidas pode ser difícil entender as interações entre múltiplas regras afetadas pelos mesmos fatos; entender a base lógica do resultado gerado pode ser tão importante quanto o próprio resultado
Relacionamento com Linguagens de Programação e Ambientes: Prolog é comumente utilizado para construir sistemas baseados em regras

Estilos Arquiteturais
Com Memória Compartilhada

- dhu hns nœc hcn- (pouso lunar):



Estilos Arquiteturais
Baseados em Interpretadores

- Caracterizado pela interpretação dinâmica, *bf uc niud*, de comandos
- Comandos são sentenças explícitas, possivelmente criados momentos antes da sua execução, geralmente representados por um texto que pode ser compreendido e editado por humanos
- Comandos são construídos a partir de um comandos primitivos pré-definidos

Estilos Arquiteturais
Baseados em Interpretadores

- Execução:
  1) Inicia-se no estado inicial de execução
  2) Obtém-se o primeiro (próximo) comando a ser executado
  3) Executa-se o comando sobre o estado atual
  4) Avança-se para um novo estado
  5) Procede-se à execução do próximo comando (goto 2)
- A identificação do próximo comando pode ser afetada pelo resultado da execução do comando anterior

Estilos Arquiteturais
Baseados em Interpretadores

- *Interpreter*
  - Similar ao Baseado em Regras / Sistema Especialista porém utiliza um interpretador de comandos no lugar da máquina de inferência
  - Este interpretador executa comandos mais genéricos (do que inferências em regras) e a interpretação de um único comando pode envolver várias operações primitivas
  - A base de conhecimento é similarmente mais genérica visto que estruturas de dados arbitrárias podem estar envolvidas
  - Exemplos: fórmulas de uma planilha de cálculo são interpretadas pela máquina de execução (interpretador) do sistema de planilhas; máquinas CNC; programação de trajetórias de robôs

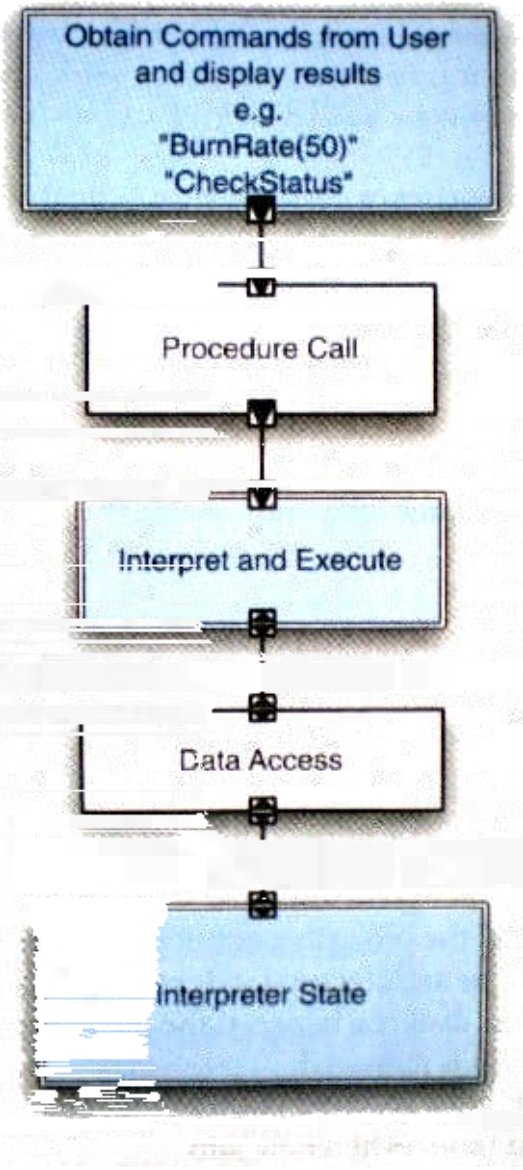
Estilos Arquiteturais
Baseados em Interpretadores

- *lh o f one enone*

Resumo: o interpretador analisa e executa comandos de entrada, atualizando o estado por ele mantido
Componentes: interpretador de comandos, estado do programa/interpretador e interface de usuário
Conectores: tipicamente o interpretador de comandos, interface de usuário e estado são fortemente acoplados via *ebons en o odhou ht ens hc on*
Elementos de Dados: comandos
Topologia: três camadas altamente acopladas; o estado pode estar separado do interpretador
Qualidades Induzidas: possibilidade de comportamentos altamente dinâmicos, onde o conjunto de comandos é dinamicamente modificado; a arquitetura do sistema permanece inalterada enquanto novas funcionalidades são criadas com base nas primitivas existentes
Usos Típicos: excelente para permitir a programação pelo usuário final; para suportar mudança dinâmica do conjunto de funcionalidades
Precauções: quando processamento rápido é necessário (código interpretado é executado de forma mais lenta que código compilado); gerenciamento de memória pode se tornar um problema, especialmente quando múltiplos interpretadores são simultaneamente executados
Relacionamento com Linguagens de Programação e Ambientes: *l h* e *ø n+ n* são linguagens interpretadas e são eventualmente utilizadas para construir outros interpretadores; macros

Estilos Arquiteturais
Baseados em Interpretadores

- *Lunar Lander* (pouso lunar):



Estilos Arquiteturais
Baseados em Interpretadores

- *b hmdh bsn*
  - Permite que um código seja transmitido a um *t bhc* remoto e por este *t bhc* interpretado
  - Um elemento de dado (representação de um programa) é dinamicamente transformado em um componente de processamento de dados
- Motivações para a transmissão: falta de poder computacional, falta de recursos ou conjunto extenso de dados localizados remotamente
- Classificações:
  - *bsn lf n- f s*
  - *n- bcn d clf*
  - *b hmdh r rf c*

Estilos Arquiteturais
Baseados em Interpretadores

- *bhmd bsn*
  - *bsn bf n+ f s*
    - Possui recursos e estado porém o código é obtido de um *t bhc* remoto e executado localmente
  - *n+ bcn d lcbf*:
    - Possui o código mas não os recursos para execução do código (ex: o interpretador)
    - O código é transmitido a um *t bhc* remoto para processamento (ex: *r les*) e os resultados enviados de volta
  - *bhmd r nf :c*
    - Possui o código e o estado mas parte dos recursos estão em outro *t bhc*
    - O código + estado + alguns recursos *r nf)c* é transferido para o *t bhc* remoto
    - Os resultados não necessariamente precisam ser enviados de volta ao *t bhc* original

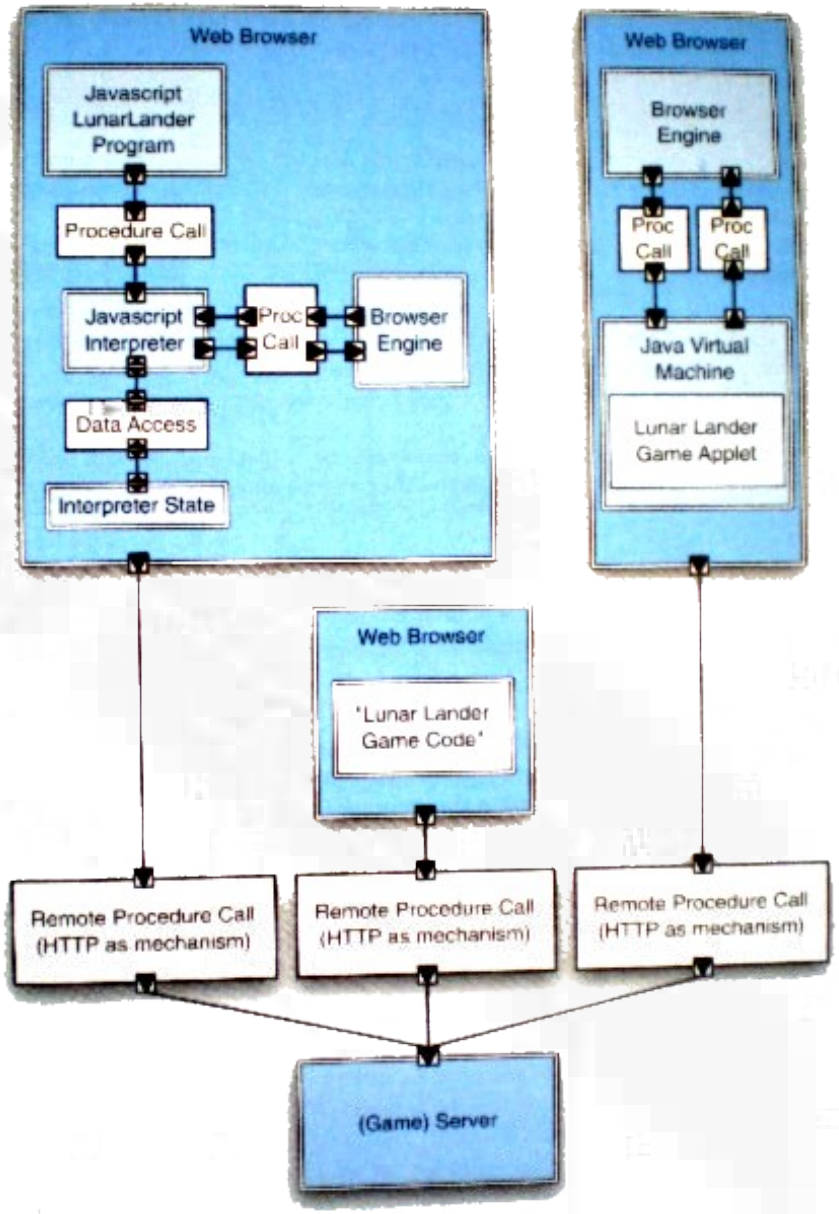
Estilos Arquiteturais
Baseados em Interpretadores

- *b hmdh bsn..*

Resumo: o código se desloca com o objetivo de ser interpretado em outro *t bhc*dependendo da variação do estilo o estado pode também se deslocar
Componentes: doca de execução (que trata o recebimento e implantação do código e do estado) e o interpretador/compilador de código
Conectores: elementos e protocolos de rede que empacotam código e dados para fins de transmissão
Elementos de Dados: representações de código sob a forma de dados; estado do programa e dados
Topologia: em rede
Variações: *obsn lf sn- f s, en- bcn n d tclf e - b hmdh r rf c*
Qualidades Induzidas: adaptabilidade dinâmica; se beneficia do poder computacional agregado nos *t bhcd*disponíveis; *sn rf s hbdc* melhorada em função da provisão de migração para novos *t bhch*
Usos Típicos: quando deseja-se processar um conjunto extenso de dados localizados remotamente; quando deseja-se configurar dinamicamente um nó através da inclusão de código externo
Precauções: aspectos de segurança (códigos maliciosos); quando precisa-se alto controle sobre as diferentes versões do *hbi c* eimplantadas; quando o custo de transmissão é maior que o de processamento; quando as conexões de rede não são confiáveis
Relacionamento com Linguagens de Programação e Ambientes: linguagens de *hoe tf r* (ex: *oe*); computação em *r tes*

Estilos Arquiteturais
Baseados em Interpretadores

- *b lmodh bsn* (pouso lunar):



Estilos Arquiteturais
Baseados em Invocação Implícita

- Caracterizados por chamadas que são invocadas indiretamente e implicitamente em resposta a uma notificação ou a um evento
- Esta interação indireta entre componentes fracamente acoplados facilita a adaptação e melhora a escalabilidade do sistema

Estilos Arquiteturais

Baseados em Invocação Implícita

- *mdht u rhoe m*

Estilos Arquiteturais
Baseados em Invocação Implícita

- *ndht u rhoe rs* simples:
  - O *ndht ne* mantém uma lista de *h rhoe meh*
  - Para cada *h rhoe me* uma *ebons en o odl* disparada sempre que uma nova informação estiver disponível
  - *rhoe meh* realizam suas assinaturas com o *ndht ne*, informando a interface de *ebons en*(*o odmo*) a ser utilizada quando a informação for publicada
  - *rhoe meh* podem cancelar suas assinaturas e ter seus respectivos *o odmo h* removidos da lista de assinantes do *ndht ne*

Estilos Arquiteturais
Baseados em Invocação Implícita

- *ndh u rhoe m*
  - Em aplicações baseadas em rede e de larga escala algumas modificações são necessárias:
    - *ndh ne* precisam anunciar (no *hc eu* do sistema, periodicamente ou sob demanda) a existência de recursos de informação que podem ser assinados
    - Assinaturas não são mais representadas por *ebons enh* de *o odmo* e passam a envolver protocolos de rede
    - Aspectos de desempenho impedem o uso de conexões ponto-a-ponto entre *ndh ne* e *h rhoe meh*, demandando *eb l nh* ou *o ot nh* intermediários

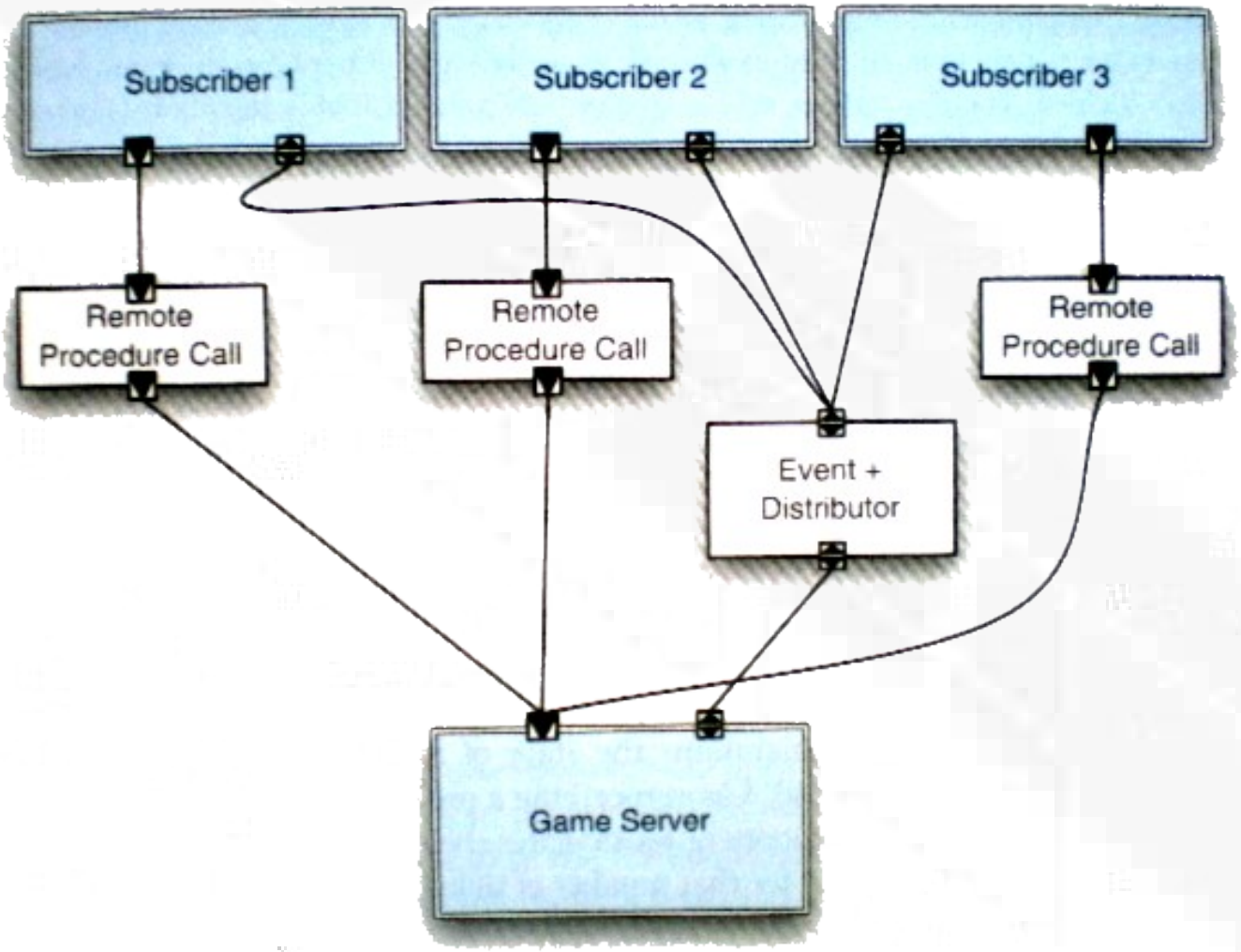
Estilos Arquiteturais
Baseados em Invocação Implícita

- *ndh u rhoe m*

Resumo: *h rhoe me* solicitam/cancelam o recebimento de mensagens específicas; *ndh neh* mantêm uma lista de assinantes e a eles enviam mensagens de forma síncrona ou assíncrona
Componentes: *ndh neh h rhoe meh eb l nh* para gerenciamento da distribuição
Conectores: *ebons en o odi* (dentro de programas) ou protocolos de rede; assinaturas baseadas em conteúdo requerem conectores mais sofisticados
Elementos de Dados: assinaturas, notificações e informações publicadas
Topologia: *h rhoe me* se conectam diretamente aos *ndh neh* ou através de intermediários
Variações: usos específicos do estilo podem requerer passos particulares na assinatura e cancelamento; suporte a correspondências complexas entre interesses de assinatura e informação disponível pode ser realizada por intermediários
Qualidades Induzidas: disseminação (*lf nu*) eficiente de informação; baixo acoplamento
Usos Típicos: disseminação de notícias; programação de interfaces gráficas de usuário; jogos - *dt d ne* baseados em rede
PrecaGGes

Estilos Arquiteturais
Baseados em Invocação Implícita

- *Middleware* (pouso lunar):



Estilos Arquiteturais
Baseados em Invocação Implícita

- *rf au hns*
  - Caracterizado por componentes independentes que se comunicam somente através de eventos transmitidos por um barramento (conector)
  - Na sua forma mais pura componentes emitem eventos para o barramento que, por sua vez, os re-transmite para todos os outros componentes
  - Componentes podem reagir em resposta ao recebimento de um evento ou ignorá-lo
  - Embora aparentemente caótico e imprevisível é similar à forma com a qual humanos se comportam em sociedade

Estilos Arquiteturais
Baseados em Invocação Implícita

- *rf au hns*
  - Por razões de eficiência a forma pura deste estilo raramente é utilizada
  - É mais eficiente distribuir os eventos somente para aqueles componentes que demonstraram interesse por eles
  - Com esta modificação o *rf au hns* se torna similar ao *ndh rhoe m*, entretanto não há distinção entre produtores e consumidores
  - A replicação e otimização de distribuição dos eventos (ex: registro de interesse em um evento particular) é responsabilidade somente dos conectores

Estilos Arquiteturais
Baseados em Invocação Implícita

- *rf au hns*
  - Pode funcionar em modo:
    - *dl bbkdf r* : receptores de eventos consultam o conector (de forma síncrona ou assíncrona) para verificar se algum novo evento está disponível
    - *ht*: o conector replica e re-transmite os eventos aos possíveis interessados
  - Altamente indicado para sistemas com componentes concorrentes altamente desacoplados onde, em um determinado momento, um componente pode estar ou criando ou consumindo informação
  - Ex: mercado financeiro / bolsa de valores

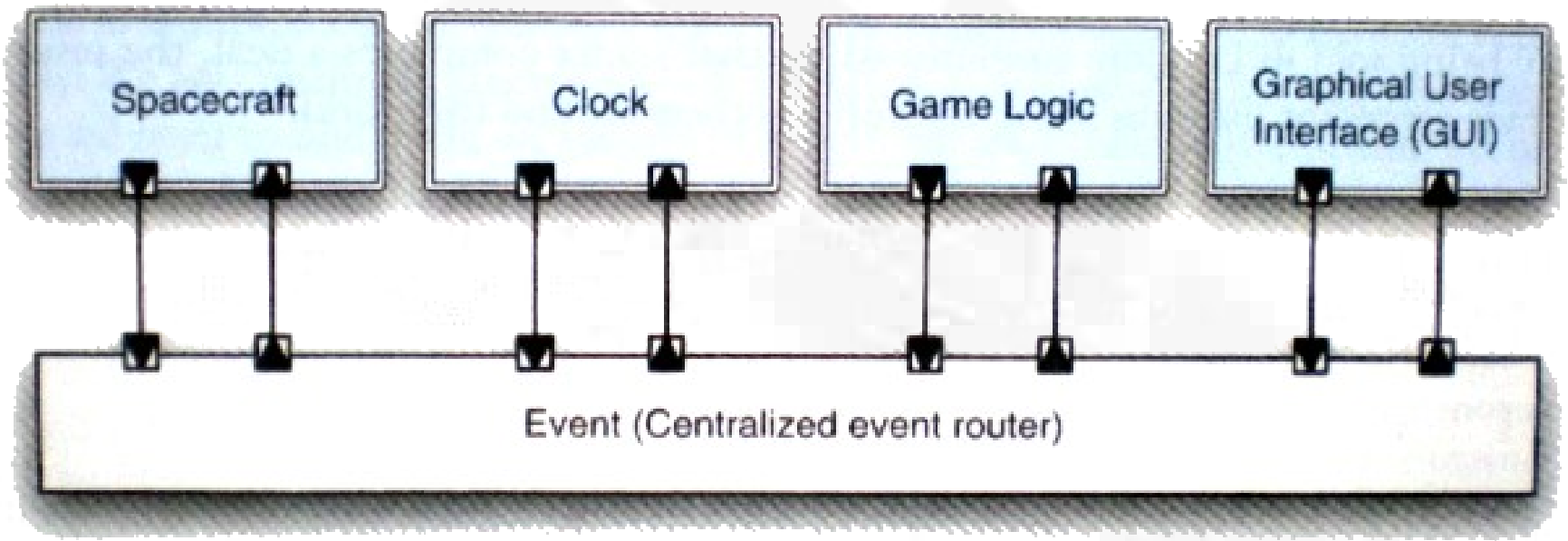
Estilos Arquiteturais
Baseados em Invocação Implícita

- *rf au hns*

Resumo: componentes independentes emitem e recebem, de forma assíncrona, eventos transmitidos por barramentos de eventos
Componentes: produtores e consumidores independentes e concorrentes de eventos
Conectores: barramento de eventos; em certas variações, mais de um conector pode ser utilizado
Elementos de Dados: eventos – dados enviados como entidades de primeira-ordem através de barramentos de eventos
Topologia: componentes se comunicam somente com os barramentos de eventos
Variações: a comunicação dos componentes com os barramentos pode acontecer em modo *h* ou *dl*
Qualidades Induzidas: altamente escalável; fácil de evoluir; efetivo para aplicações altamente distribuídas e heterogêneas
Usos Típicos: interfaces gráficas de usuário; aplicações *l snu en* envolvendo partes independentes (mercado financeiro, logística, redes de sensores)
Precauções: não há garantia se ou quando um evento particular será processado
Relacionamento com Linguagens de Programação e Ambientes: tecnologias de *- l ssdh en* orientado a mensagens (, *rf c nel on,*)

Estilos Arquiteturais
Baseados em Invocação Implícita

- *rf au hns*(pouso lunar):



Estilos Arquiteturais

Peer-to-Peer

- *nneu*: *cbu nne* ( ):
  - Consiste em uma rede de *nneh* autonômos fracamente acoplados
  - Cada *nne* atua tanto como cliente quanto servidor
  - *nneh* se comunicam utilizando um protocolo de rede, provavelmente especializado para comunicação (ex: *hcn*, *e f cnd*)
  - Descentraliza tanto informação quanto controle, fazendo com que a descoberta de recursos seja um aspecto importante

Estilos Arquiteturais

Peer-to-Peer

- *nnecdbu nn*( ):
  - Descoberta de recursos em sistemas puros:
    - A solicitação da informação é lançada na rede como um todo
    - A requisição se propaga até que a informação seja descoberta ou algum limite de propagação (ex: número de *t b*) seja alcançado
    - Se a informação é encontrada o *nn* obtém o endereço direto do outro *nn* e o contacta diretamente
  - É limitado pelo algoritmo distribuído utilizado para consultar o sistema e pela largura de banda disponível

Estilos Arquiteturais
Peer-to-Peer

- *nne* *ud* *bu* *nne* ( ):
  - Descoberta de recursos em sistemas híbridos:
    - O processo é otimizado através da presença de *nne* *h* especiais, especializados na localização de outros *nne* *h* e/ou disponibilização de diretórios que localizam as informações
    - Ex: *hc* *ne* – utilizava um servidor centralizado para indexação das músicas e localização de outros *nne* *h*
  - Embora o estilo tenha se tornado popular nas aplicações de compartilhamento de arquivos é frequentemente utilizado em *ob* - - *ne* *on* *ot* *,c* colaboração remota e redes de sensores

Estilos Arquiteturais

Peer-to-Peer

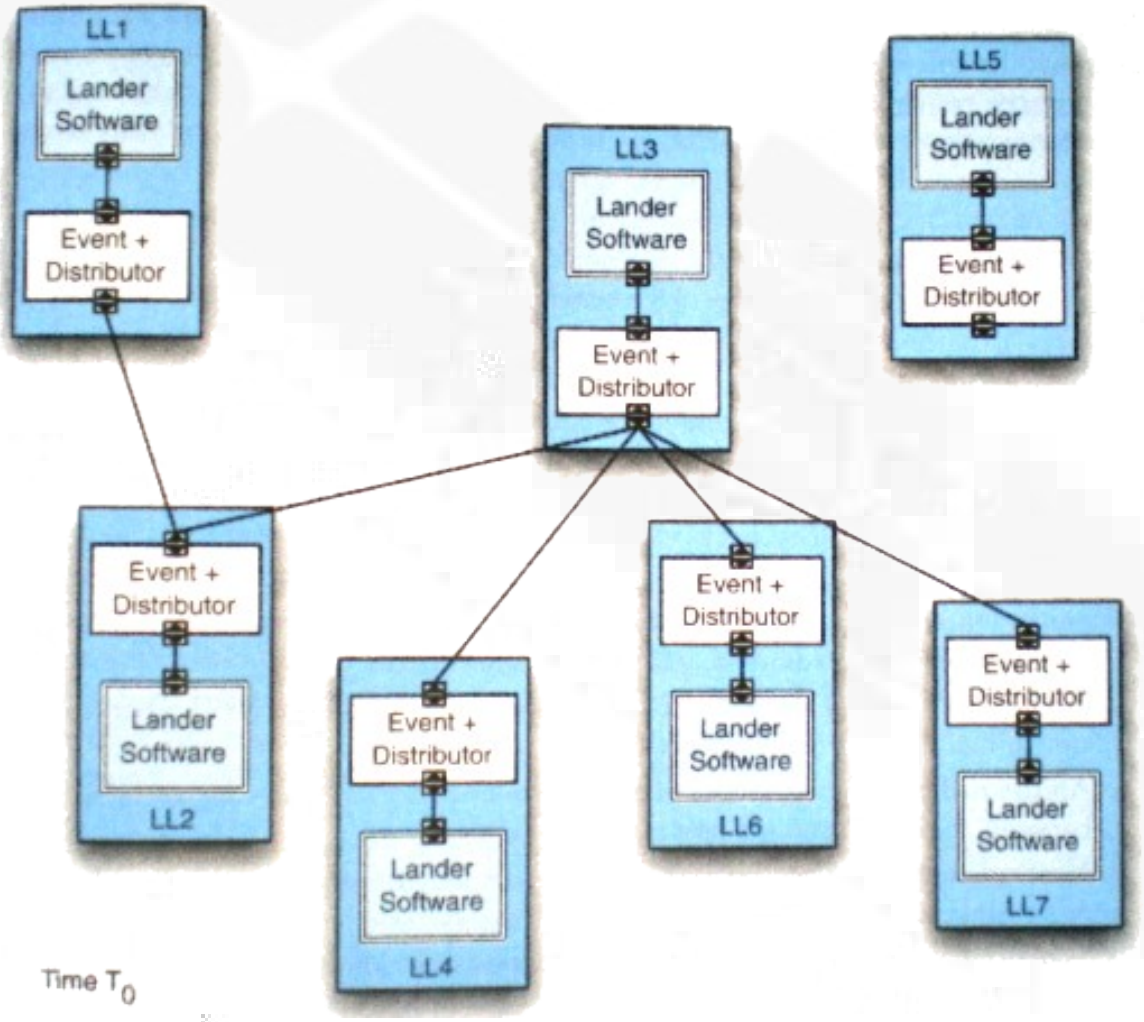
- *nneacdbu nne*( ):

Resumo: estado e comportamento estão distribuídos entre *nneh* que podem atuar tanto como clientes quanto como servidores
Componentes: *nneh* – componentes independentes com seu estado e *tc en s* de controle próprios
Conectores: protocolos de rede, frequentemente especializados
Elementos de Dados: mensagens de rede
Topologia: em rede (com possibilidade de conexões redundantes entre *nneh*); pode variar arbitrariamente e dinamicamente
Qualidades Induzidas: computação descentralizada com fluxo de controle e recursos distribuídos entre os *nneh* altamente robusto na presença de falhas em qualquer nó; escalável em relação ao acesso a recursos e poder computacional
Usos Típicos: onde as operações e fontes de informação estão distribuídas e a rede é *stu bo*
Precauções: quando o tempo necessário para recuperação da informação é importante e é inviável lidar com a latência imposta pelo protocolo; segurança (deve-se detectar *nneh* maliciosos e prover meios para gerenciar a confiança – *œ he* – em ambientes abertos)

Estilos Arquiteturais

Peer-to-Peer

- *nneacdbu nne(*) (pouso lunar):

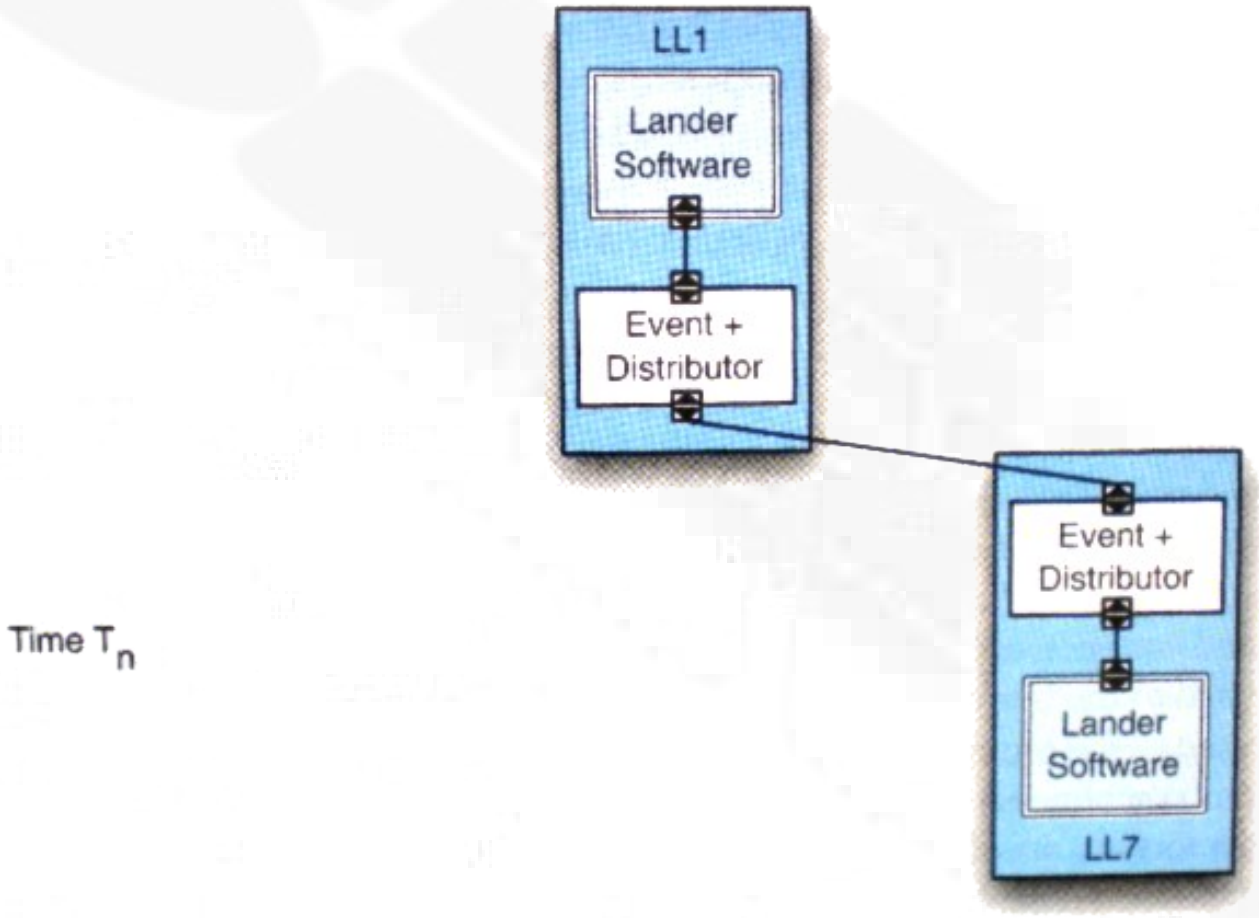


Estilos Arquiteturais
Peer-to-Peer

- *nneu cbu nne* ( ) (pouso lunar):
  - Obtenção da informação pelo *f e f sne* (LL1):
    1) No tempo T_0, LL1 contacta todas as naves presentes no raio de comunicação
    2) Somente LL2 responde
    3) LL1 pergunta a LL2 se ela possui a informação desejada
    4) Visto que LL2 não possui esta informação ela repassa a pergunta para o seu nó de comunicação adjacente – LL3 (assume-se que LL2 e LL3 já se conhecem)
    5) Visto que LL3 não possui esta informação ela repassa a pergunta para LL4, LL6 e LL7
    6) LL7 informa, a LL3, que possui a informação e a envia para ela
    7) LL3 passa a informação de volta a LL2 e, subsequentemente, a LL1
  - Em um tempo T_n LL7 adentra o raio de comunicação de LL1 e elas agora podem se contactar diretamente

Estilos Arquiteturais
Peer-to-Peer

- *nneacdbu nne(*) (pouso lunar):



Discussão: Padrões e Estilos

- Tanto padrões quanto estilos refletem experiências e codificam conhecimentos obtidos através destas experiências
- Criar um bom estilo exige refletir sobre a experiência vivida, abstrair o conhecimento dos detalhes irrelevantes e possivelmente generalizar a solução empregada

Discussão: Padrões e Estilos

- Porque então ainda existe resistência no uso de estilos ?
  - Porque exige que o projetista trabalhe dentro do conjunto de restrições que compõem o estilo e isto “parece” improdutivo ou limitante
  - Entretanto, uma característica curiosa e não-intuitiva do projeto com estilos arquiteturais é que, com a adoção destas restrições, obtém-se grande liberdade e efetividade no projeto

Discussão: Padrões e Estilos
A Liberdade das Restrições

- Estilos limitam o projetista de diversas formas:
  - A quantidade de detalhes e conceitos possíveis de serem utilizados pode ser limitado (ex: somente três camadas)
  - A forma de interação entre os elementos pode ter restrições (ex: comunicação somente através de eventos)
  - Podem existir obrigações a serem cumpridas (ex: comunicação mediada sempre por conectores de primeira-classe)
  - Podem fazer com que a solução de algum sub-problema seja sub-ótima (ex: exigir comunicação por *hœn -)n*
    - Neste caso, entretanto, os benefícios de facilidade de compreensão e reuso induzidos pelo estilo superam qualquer outro benefício limitado

Discussão: Padrões e Estilos
A Liberdade das Restrições

- De onde vêm então a liberdade e efetividade ?
  - Estilos restringem o foco, criando um espaço de problemas com o qual o projetista não precisa se preocupar
  - Estilos garantem a aplicabilidade de ténicas particulares de análise
  - As restrições possibilitam a utilização de estratégias de geração automática de código e uso de *i e- n be h*
  - Estilos disponibilizam invariantes: “desde que eu siga estas regras eu sei que isto sempre será verdade”
  - Estilos tornam a comunicação mais eficiente e facilitam a compreensão das decisões em um tempo futuro

Discussão: Padrões e Estilos
Combinação, Modificação e Criação de Estilos e Padrões

- A lista de estilos apresentada não é exaustiva
- Cada domínio de aplicação poderá ter um estilo específico associado
- Entretanto, é comum que este estilo específico seja uma agregação ou combinação de estilos mais simples, motivado pelo desejo de obtenção de múltiplos benefícios
- Exemplo:
  - C2 = MVC + Camadas + Eventos
  - REST = Camadas + Interpretadores + Replicação + ...

Discussão: Padrões e Estilos

Combinação, Modificação e Criação de Estilos e Padrões

- Síndrome do (*not invented here*)
  - Deve-se evitar a criação desnecessária ou imprudente de novos estilos pois:
    - Este novo estilo provavelmente representa a experiência de um projetista particular e provavelmente não foi comparado com experiências de outros projetistas da empresa ou do domínio de aplicação
    - É uma negação de uma das vantagens dos estilos – promoção de comunicação efetiva entre projetistas
    - A conveniência e perfeccionismo raramente superam os benefícios genuinos da aplicação cuidadosa de estilos já estabelecidos e bem conhecidos

Discussão: Padrões e Estilos
Combinação, Modificação e Criação de Estilos e Padrões

- Se o novo estilo for realmente necessário deve-se focar no que é verdadeiramente central
- Deve-se pensar que “menos significa mais” até que a nova abordagem seja adequadamente validada
  - Tal validação inclui a clara articulação de quais invariantes são obtidas como consequência das restrições adicionadas

Design Recovery

- Em um cenário ideal assume-se que a arquitetura é a reificação da concepção técnica do sistema e que está fielmente refletida na implementação do sistema
- Infelizmente, não é isso que acontece:
  - Sistemas são criados e modificados sem consideração e documentação dos aspectos arquiteturais
  - Quando um projeto e documentação arquitetural estão presentes as futuras modificações são geralmente realizadas sem muita preocupação em manter a integridade intelectual, resultando em erosão arquitetural:
    - A evolução do *hbi c er*requer conhecimento profundo da aplicação, da sua complexidade, da sua arquitetura geral, dos componentes principais e suas interações e dependências

Design Recovery

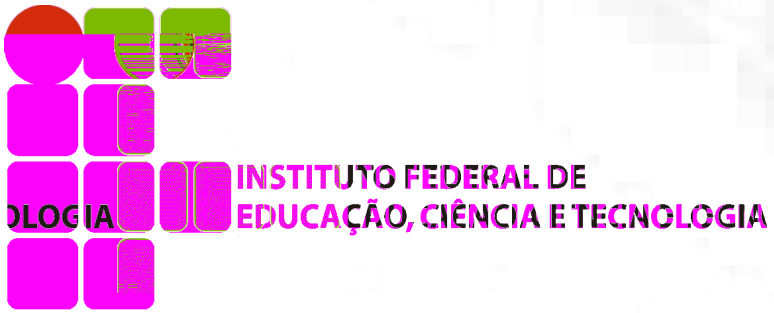
- Para lidar com este problema geralmente realiza-se recuperação arquitetural (*architecture recovery*):
  - Processo onde o projeto arquitetural do sistema é extraído a partir dos seus outros artefatos, atualizados de forma mais confiável que o projeto arquitetural
  - Tais artefatos podem ser: requisitos, especificações formais, planos de testes, etc
  - A implementação do sistema, entretanto, é frequentemente utilizada como ponto de partida para a recuperação

Design Recovery

- Uma abordagem comum para recuperação arquitetural é o *clustering* de entidades de implementação em elementos arquiteturais
- Técnicas de *clustering*, a depender da abordagem utilizada para agrupar entidades de código (classes, procedimentos, etc), podem ser classificadas em:
  - Sintáticas:
    - Foca exclusivamente no relacionamento estático entre entidades de código-fonte. Podem utilizar métricas de coesão e acoplamento
  - Semânticas:
    - Inclui informações sobre a similaridade comportamental das entidades de código-fonte. Pode avaliar, por exemplo, a frequência de interação ou o método efetivamente invocado no caso de polimorfismo + ligação dinâmica

INF016 – Arquitetura de Software

04 – Projetando Arquiteturas

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas