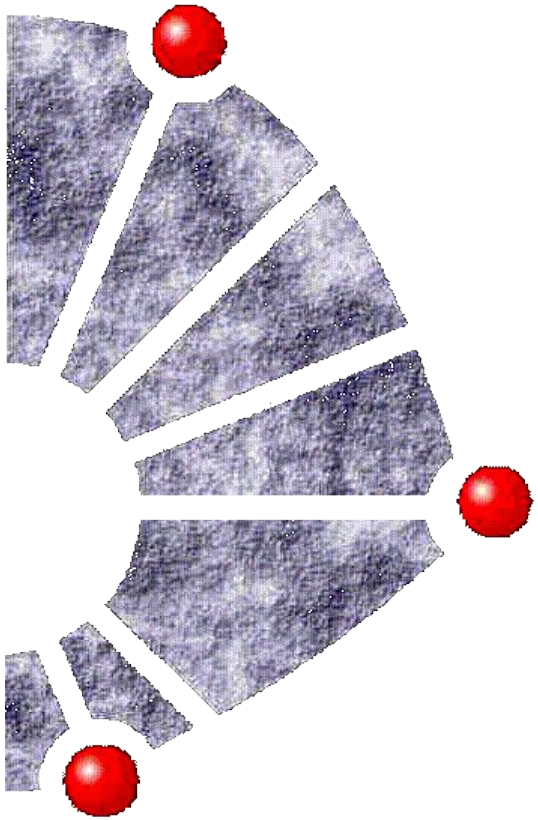
O Processo Unified e a Fase de Design



FEEC-UNICAMP

Ricardo Gudwin

Design

- **Fase de Análise:**
  - enfatiza a compreensão dos requisitos, conceitos e operações relacionadas ao sistema - **QUAIS** os processos, conceitos, etc ...
- **Fase de Design:**
  - inicia o desenvolvimento de uma solução lógica baseada no paradigma da orientação a objetos - **COMO** os processos, conceitos, etc... são implementados
- Objetivos do Design
  - Aquisição de uma compreensão profunda dos fatores relacionados a requisitos não-funcionais e restrições relacionadas a:
    - linguagens de programação, reuso de componentes, sistemas operacionais, tecnologias de distribuição e concorrência, tecnologias de bancos de dados, tecnologias de interface com o usuário, tecnologias de gerenciamento de transações, etc ...

Design

- Objetivos do Design
  - Criação dos elementos lógicos necessários para as atividades de implementação subsequentes, por meio da definição de subsistemas, interfaces e classes
  - Planejamento do trabalho de implementação decomposto em pedaços que possam ser trabalhados por diferentes equipes de trabalho, possivelmente ao mesmo tempo
  - Captura das principais interfaces entre subsistemas, úteis no projeto da arquitetura do software, principalmente para a sincronização entre diferentes equipes de trabalho
  - Permitir a especificação de elementos lógicos a serem implementados por meio de uma notação uniforme
  - Criar uma abstração objetiva da implementação do sistema, de tal forma que a implementação seja um mero refinamento para o design, permitindo e.g. a geração automática de código

Design

- Papel do Design no Ciclo de Vida do Software
  - Design é enfatizado nos ciclos finais da fase de elaboração e começo da fase de construção
  - Ao final da fase de construção, a ênfase maior acaba sendo para a implementação
- Artefatos do Design
  - Modelo de Design
  - Classes de Design
  - Realização dos Casos de Uso - Design
  - Subsistemas de Design
  - Interfaces
  - Descrição da Arquitetura (Visão do Modelo de Design)
  - Modelo de Distribuição
  - Descrição da Arquitetura (Visão do Modelo de Distribuição)

Patterns e Design Patterns

- Patterns
  - desenvolvedores de software orientado a objetos experientes acumularam um repertório de princípios gerais e soluções que os guiam frequentemente em suas decisões no desenvolvimento de novos softwares
  - esses princípios são formalizados/compilados, dando origem aos chamados **Patterns (padrões)**
  - **patterns** codificam um conhecimento comum sobre princípios de como resolver problemas que aparecem repetidamente
- Formato
  - par problema/solução que pode ser aplicado em novos contextos, acompanhados de conselhos sobre como devem ser aplicados
  - **nomes sugestivos:** enraízam o conceito do pattern em nossa memória e promovem seu uso, sempre que possível

Patterns e Design Patterns

- Origem dos Patterns
  - "Design Patterns: Elements of Reusable Object-Oriented Software" de Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides (Gang of Four - GoF)
  - Vários diferentes domínios:
    - organizações, processos, ensino, arquitetura, etc….
  - Atualmente: arquitetura e design de software, outras fases do desenvolvimento de software (análise, etc…)
  - Outros livros:
    - Pattern-Oriented Software Architecture: A System of Patterns" (POSA book), de Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Somerlad e Michael Stal (Siemens Gang of Five - GoV)
    - Pattern Languages of Program Design and PLPD 2 - selected papers from 1st and 2nd conferences on Patterns Languages of Program Design

Patterns e Design Patterns

- Termo "Pattern"
  - derivado dos trabalhos do arquiteto Christopher Alexander, que escreveu vários livros em tópicos relacionados ao planejamento urbano e arquitetura de construção civil
- História dos Patterns
  - Em 1987, Ward Cunningham e Kent Beck decidiram utilizar algumas das idéias de Alexander no desenvolvimento de conselhos para a geração de estruturas eficientes de código em Smalltalk, que se repetiam em diversos programas
  - Em 1991, Jim Coplien publicou "Advanced C++ Programming Styles and Idioms" (idiomas são um tipo especial de pattern)
  - De 1990 a 1992, membros da GoF compilaram um catálogo de patterns
  - Em 1993 e 94, novas conferências sobre patterns surgiram e logo a seguir apareceu o livro de Design Patterns da GoF

Patterns e Design Patterns

- O que é um Pattern, afinal ?
  - Dirk Riehle e Heinz Zullighoven definem:
    - "Um Pattern corresponde a uma abstração de uma forma concreta que aparece recorrentemente em contextos específicos e não-arbitrários"
  - A noção de pattern está voltada para a solução de problemas de design que se repetem em diferentes contextos
  - um pattern corresponde ao encapsulamento de uma informação instrutiva que captura a estrutura essencial e a efetividade de uma família de soluções de sucesso em problemas recorrentes que surgem em determinados contextos e sistemas de forças
  - patterns capturam soluções completas, não simplesmente princípios gerais ou estratégias
  - patterns são conceitos testados e aprovados, não teorias ou especulações - são descobertos, e não inventados !

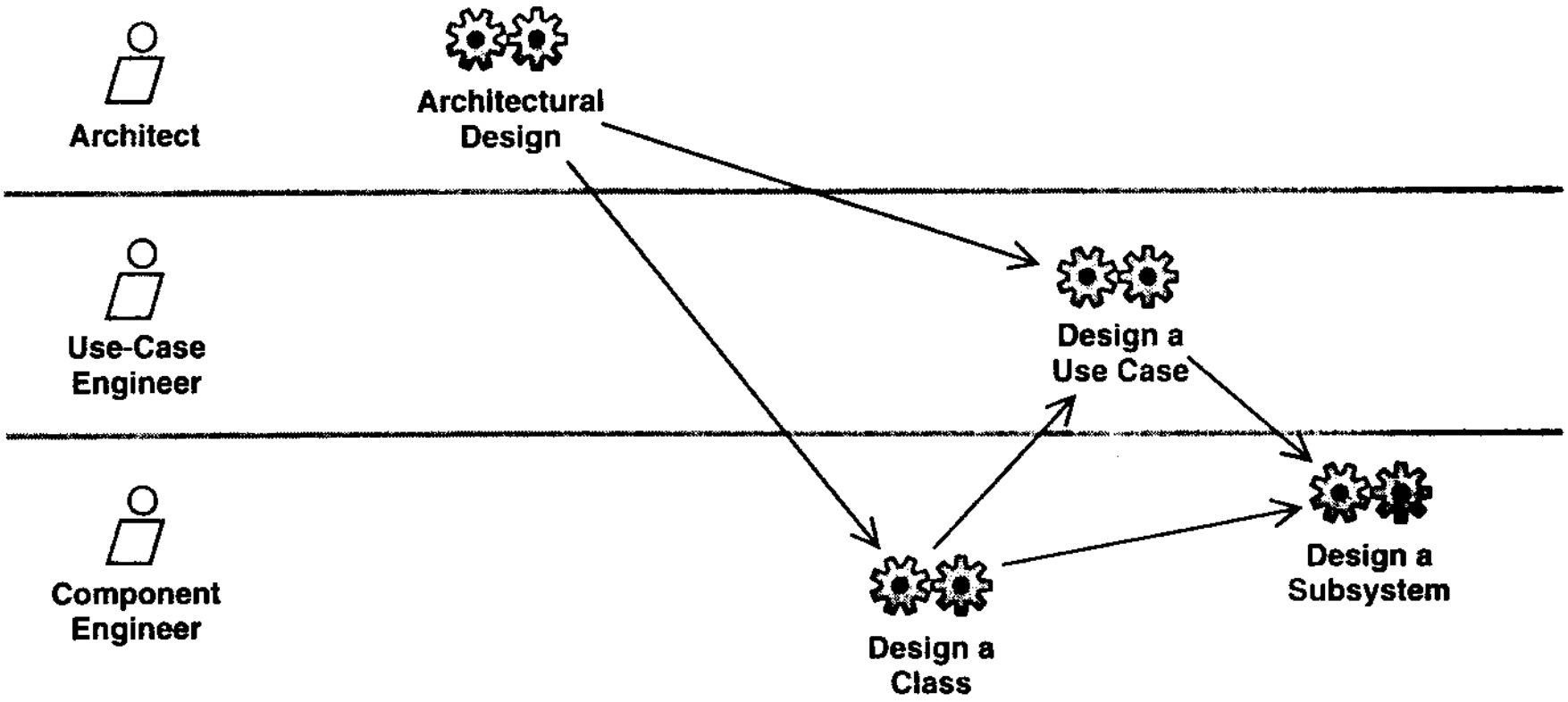
Frameworks

- Frameworks de Software
  - são mini-arquiteturas reutilizáveis que provêm a estrutura genérica e comportamento para famílias de abstrações de software, junto com um contexto de metáforas que especificam sua aplicação e uso dentro de um dado domínio
  - correspondem a um conjunto de classes cooperantes que permitem uma reutilização de design para uma classe de software específica. Um framework provê uma orientação arquitetural, definindo quais as responsabilidade e colaborações entre objetos. Um designer customiza um framework para uma aplicação em particular, normalmente por meio do sub-classeamento e geração de instâncias de classes derivadas das classes do framework
  - reutilização do design por meio da reutilização do código
  - implementação de um sistema de *design patterns*

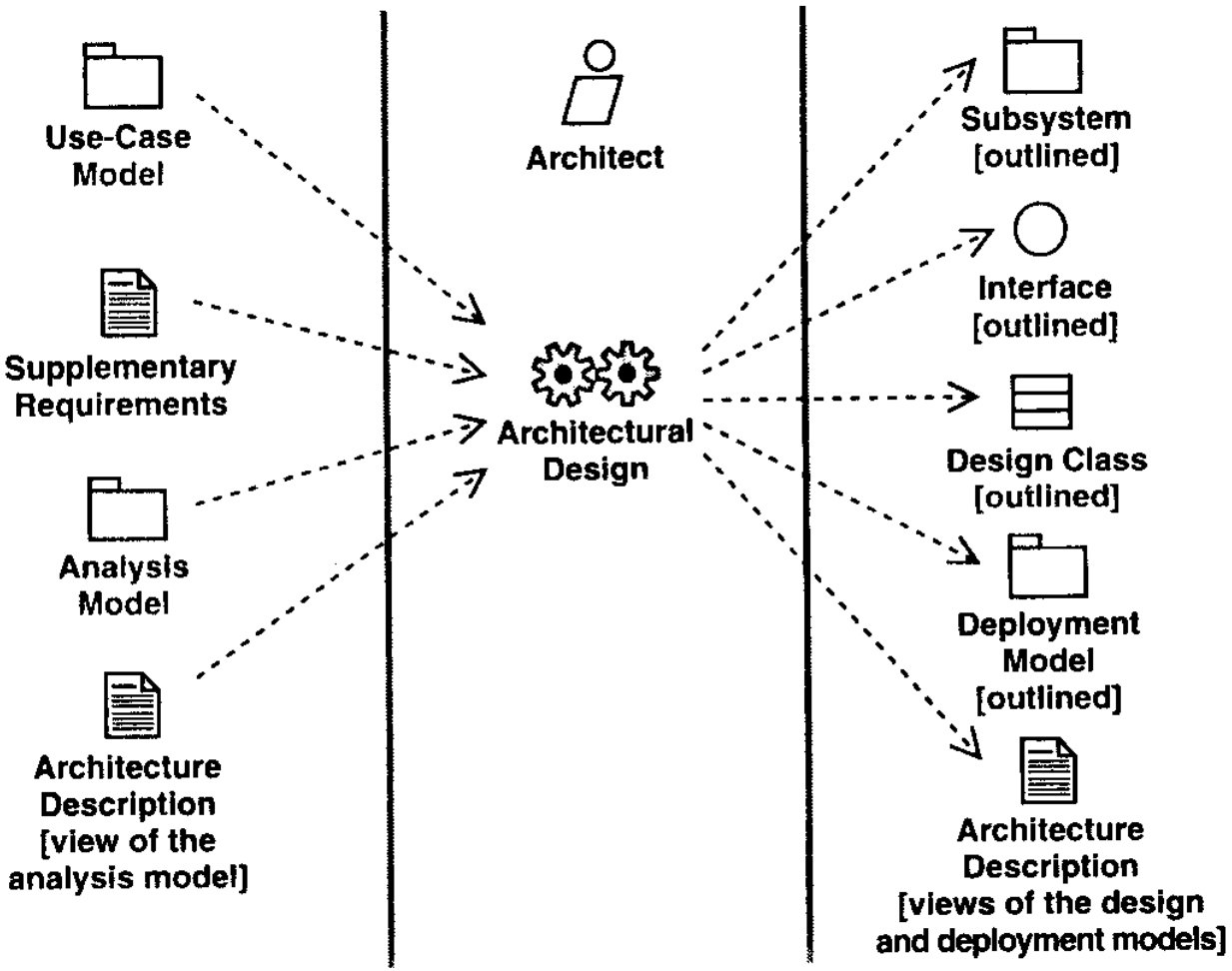
Pattern e Frameworks

- Patterns x Frameworks
  - Design Patterns são entidades mais abstratas que Frameworks: frameworks estão de certa forma embutidos no código, ao passo que somente exemplos (ou instâncias) de patterns estarão embutidas no código
  - Uma vantagem dos Frameworks é que estes podem ser formalizados diretamente em uma linguagem de programação e não somente estudados, sendo executados e reutilizados diretamente. Design patterns precisam ser implementados a cada vez que são utilizados
  - Um framework pode conter diversos design patterns, mas o contrário nunca é verdadeiro
  - Design patterns são menos especializados que frameworks: frameworks sempre têm um domínio particular de aplicações em vista. Design Patterns podem ser utilizados em qualquer tipo de aplicação

Design


Architect
Architectural Design
Use-Case Engineer
Design a Use Case
Component Engineer
Design a Class
Design a Subsystem

Design Arquitetural

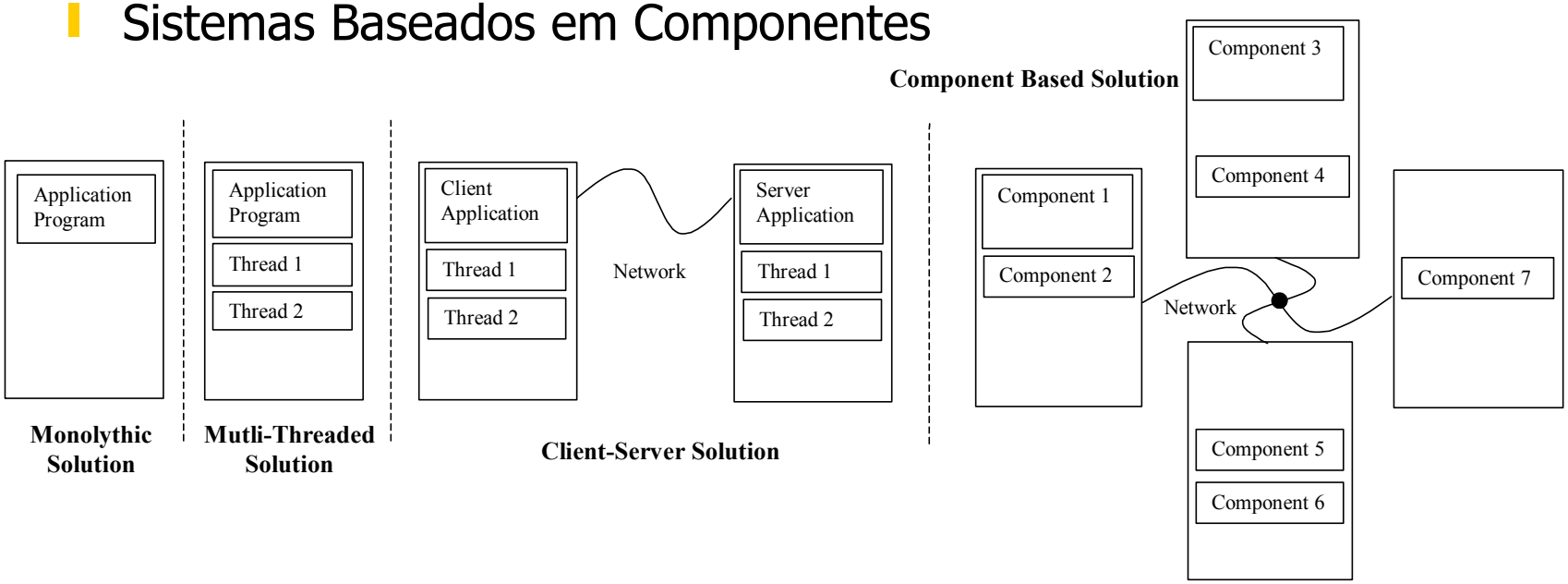


Design Arquitetural

- Objetivo
  - desenvolver um outline dos modelos de design e de distribuição, bem como sua arquitetura, identificando-se os seguintes:
    - nós computacionais envolvidos e arquitetura de rede
    - subsistemas e suas interfaces
    - classes de design arquiteturalmente significativas, tais como classes ativas
    - mecanismos genéricos de design que garantam a implementação dos requisitos especiais sobre persistência, distribuição, desempenho, etc.
- Reuso
  - nesta atividade, considera-se as diferentes possibilidade de reuso, tais como o reuso de partes, componentes, subsistemas, etc...

Design Arquitetural

- Sistemas de Software
  - Aplicações Monolíticas
  - Aplicações Multi-Thread
  - Sistemas Cliente-Servidor
  - Sistemas Baseados em Componentes



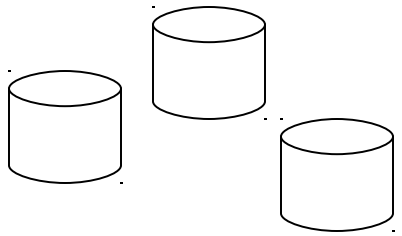
Design Arquitetural

- Arquitetura em Três Camadas
  - pattern que divide as funcionalidades de um sistema em três camadas
    - uma para interações com o usuário
    - uma para a lógica de aplicação
    - uma para acesso a banco de dados
  - arquiteturas do tipo cliente-servidor são uma possível instância desse pattern

Collect Data

Process Data

Generate Statistics

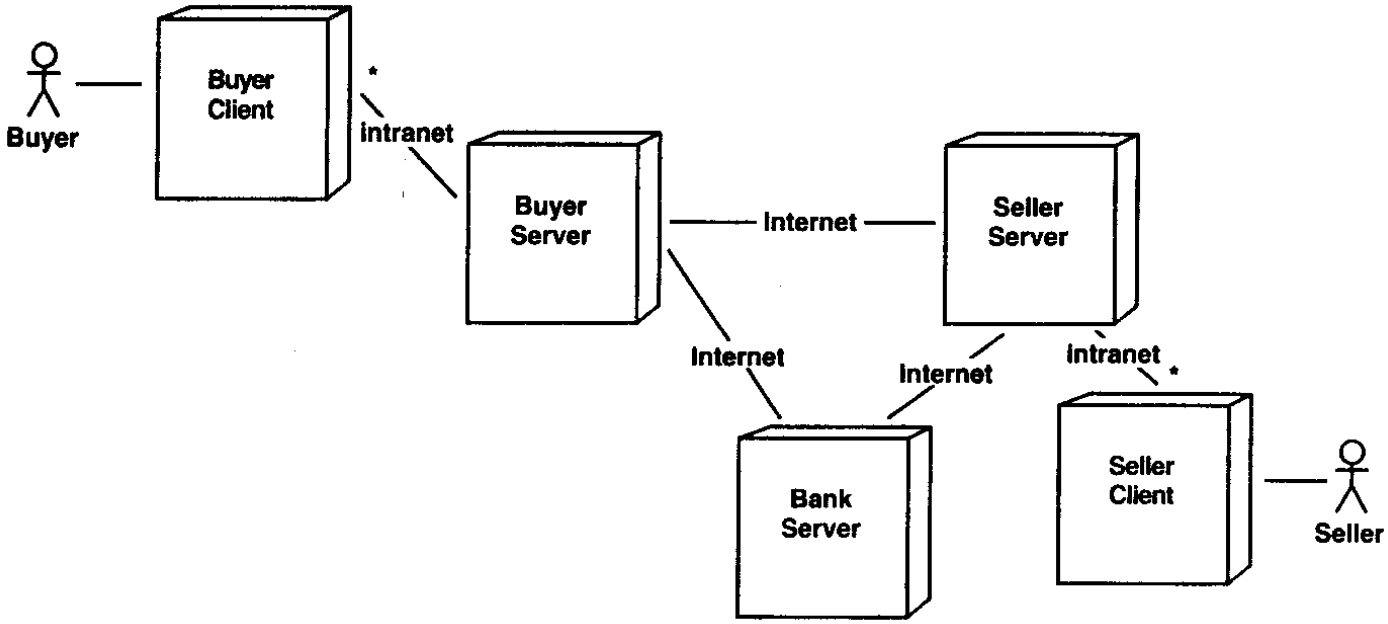


Design Arquitetural

- Identificação de Nós e Configuração de Rede
  - configuração de rede pode ser fundamental para a aplicação em desenvolvimento
  - configurações de rede normalmente utilizam uma arquitetura em três camadas
  - aspectos da configuração de rede incluem:
    - quais nós estão envolvidos e quais suas capacidades
    - que tipo de conexões há entre os nós e quais protocolos utilizam
    - quais as características das conexões (capacidade, qualidade, etc)
    - existe necessidade de processamento redundante, etc..
  - Conhecendo os limites e possibilidades dos nós e suas conexões, o arquiteto pode incorporar tecnologias tais como ORBs (Object Request Brokers), serviços de replicação de dados, etc ...

Design Arquitetural

- Exemplo de Identificação de Nós e Configuração de Rede
  - cada configuração de rede, incluindo configurações especiais para teste e simulações, deve ser descrita em um diagrama de distribuição em separado

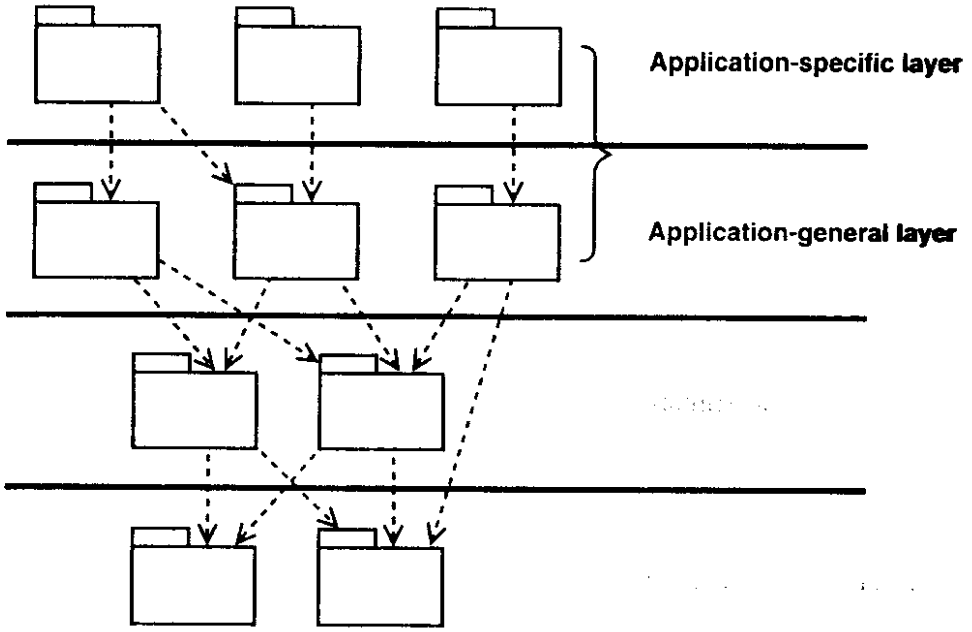


Design Arquitetural

- Identificando Subsistemas e suas Interfaces
  - subsistemas provêm um meio de organizar o design do sistema em partes gerenciáveis
    - podem ser criados ao início do design
    - podem ser desmembrados, a medida que pacotes tornam-se muito grandes e demandem uma decomposição
  - etapas de identificação de subsistemas
    - indentificação de subsistemas de aplicação
    - identificação de middleware e subsistemas de software de sistema
    - definição das dependências entre subsistemas
    - identificação de interfaces de subsistemas
  - nem todos os subsistemas são desenvolvidos no projeto
    - alguns subsistemas podem ser produtos reutilizados
    - a reutilização de subsistemas é decidida nesta fase

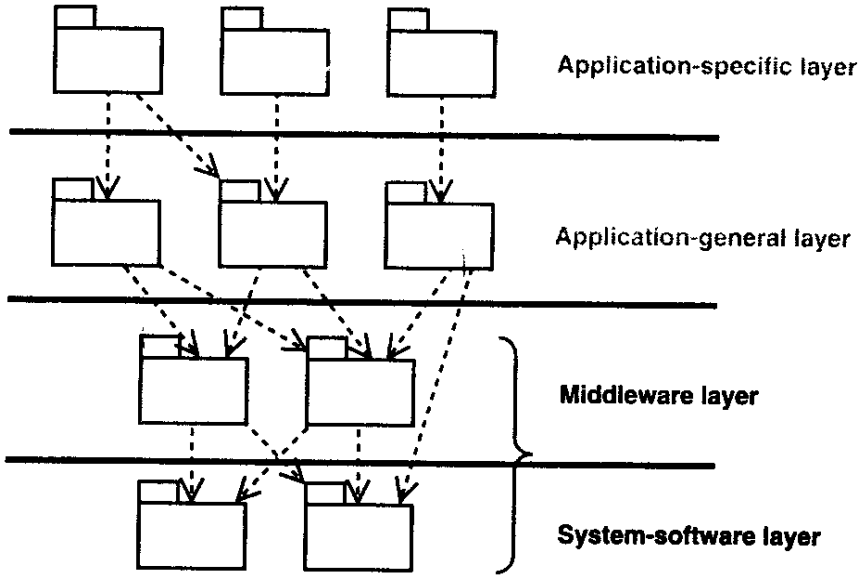
Design Arquitetural

- Identificando Subsistemas de Aplicação
  - neste passo se identificam os subsistemas específicos da aplicação e as camadas de aplicação geral
  - subsistemas levantados na fase de análise podem ser utilizados como base para a definição destes subsistemas
  - entretanto, esses subsistemas sofrerão um refino agora no design



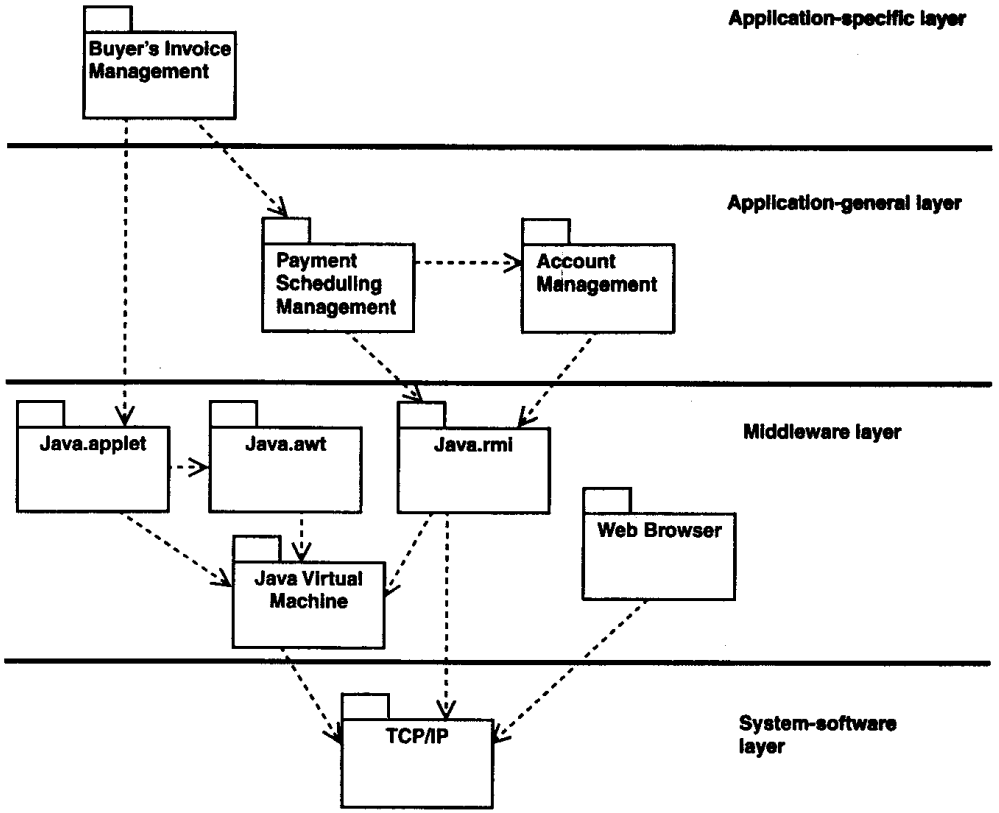
Design Arquitetural

- Identificando Middleware e Subsistemas de Software de Sistema
  - middleware e software de sistemas são a base das funcionalidades de um sistema
  - elementos a serem definidos
    - sistemas operacionais
    - sistemas de bancos de dados
    - software de comunicação
    - tecnologias de objetos distribuídos
    - kits de interface gráfica
    - tecnologias de gerenciamento de transações



Design Arquitetural

- Identificando Dependências entre Subsistemas
  - Devem ser definidas, caso haja relacionamentos entre subsistemas
  - navegabilidade da dependência deve ser equivalente a do relacionamento
  - dependências devem ser análogas àquelas encontradas na fase de análise
  - dependências entre interfaces (preferência)



Design Arquitetural

- Identificando Interfaces de Subsistemas
  - interfaces de subsistemas definem operações que são acessíveis "de fora" do subsistema
  - essas interfaces são providas por classes ou outros subsistemas (recursivamente) dentro do subsistema
  - inicialmente, antes que o conteúdo de um subsistema seja conhecido
    - começa-se considerando as dependências entre subsistemas
    - em seguida, analisa-se as classes dentro dos pacotes de análise
  - interfaces para os subsistemas de middleware e software de sistema
    - interfaces pré-definidas pelo produto utilizado
  - não basta identificar somente quais são as interfaces
    - identificar as operações disponibilizadas por cada interface

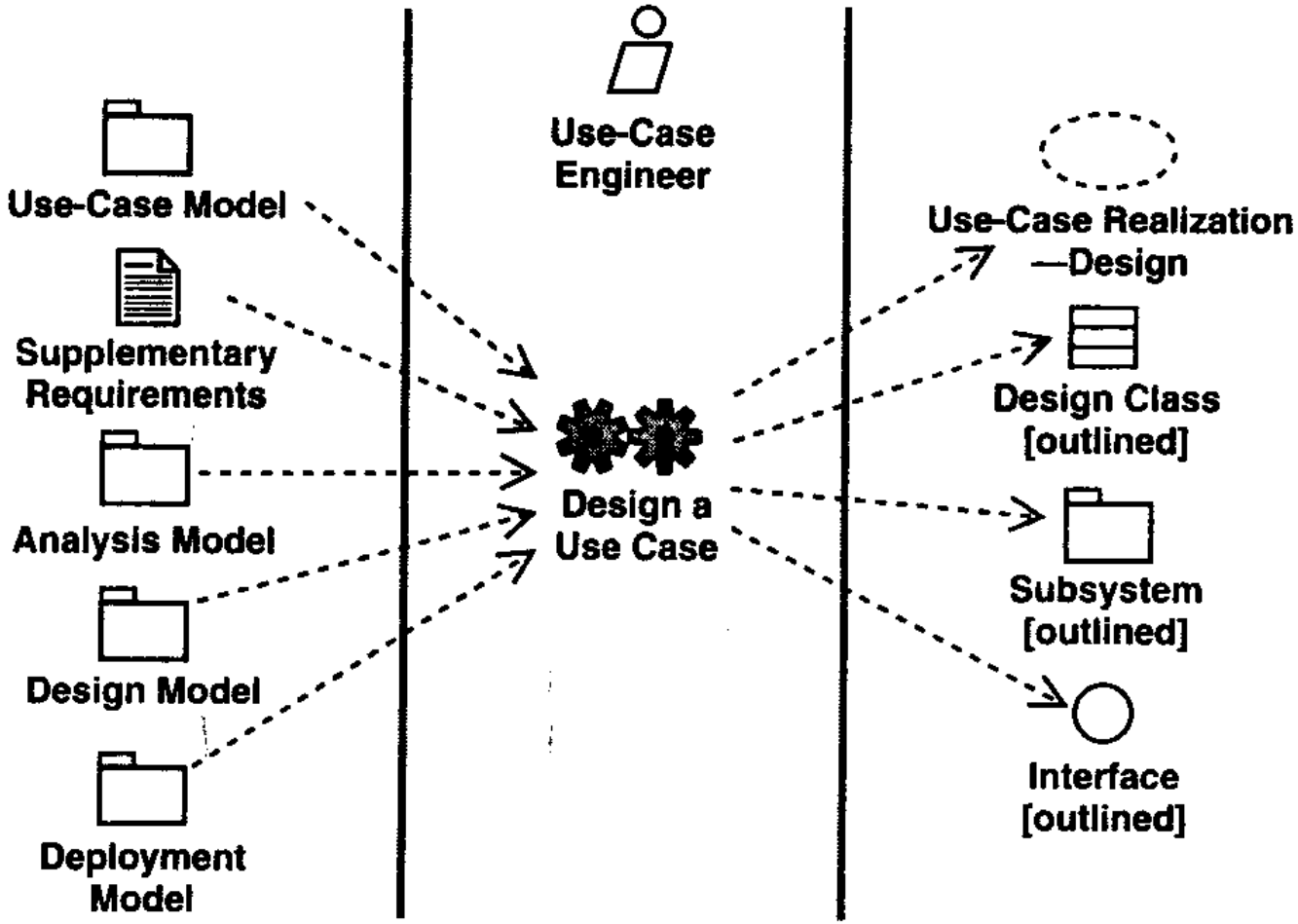
Design Arquitetural

- Identificando Classes de Design Arquiteturalmente Significativas
  - classes de design derivadas de classes de análise
  - identificação de classes ativas: considerações sobre os requisitos de concorrência do sistema
    - requisitos sobre desempenho, throughput, disponibilidade e tempo de resposta necessários pelos diferentes atores interagindo com o sistema (e.g. caso o ator demande certo tempo de resposta, deve-se disponibilizar um objeto ativo somente para a interação)
    - distribuição do sistema pelos nós (e.g. caso seja necessário suportar distribuição por diversos nós, cada nó demandará pelo menos um objeto ativo para gerenciar a comunicação)
    - outros requisitos, tais como requisitos de inicialização e shutdown, sobrevivência, evitação de deadlocks, capacidade de reconfiguração de nós, etc ...

Design Arquitetural

- Identificação de Mecanismos Genéricos de Design
  - neste passo, estudam-se os requisitos comuns, tais como os requisitos especiais identificados durante a análise, decidindo-se como implementá-los, dadas as tecnologias de implementação disponíveis
  - resultado é um conjunto de mecanismos genéricos de design, instanciados em classes de design
  - tipos de requisitos instanciados
    - persistência
    - distribuição de objetos transparente (objetos distribuídos)
    - requisitos de segurança
    - detecção e recuperação de erros
    - gerenciamento de transações

Design de Casos de Uso

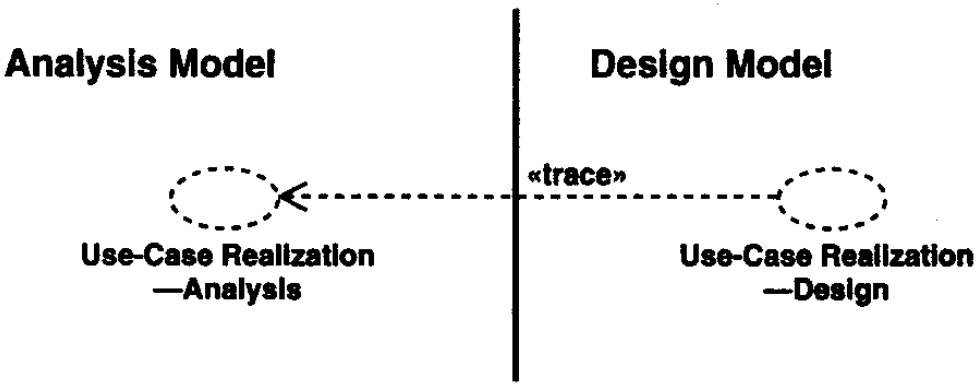


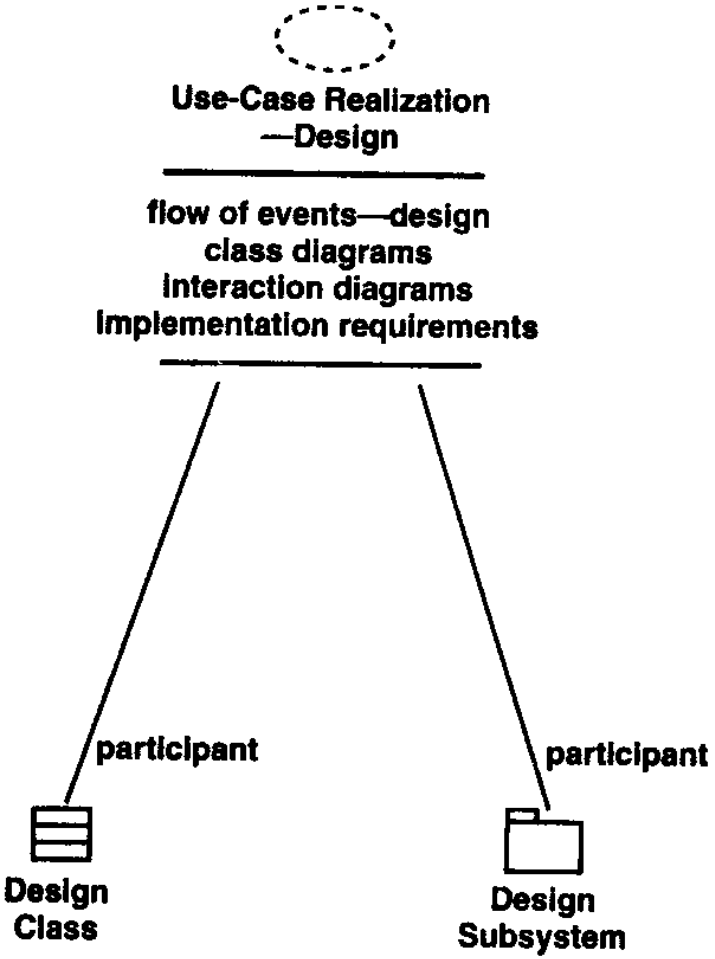
Design de Casos de Uso

- Objetivos
  - identificação das classes de design e subsistemas cujas instâncias são necessárias para realizar o fluxo de eventos de um caso de uso
  - distribuição do comportamento do caso de uso entre objetos de design interagindo entre si ou com outros subsistemas
  - definição dos requisitos sobre as operações disponíveis nas classes de design e/ou subsistemas com interfaces
- Sub-Atividades
  - identificação das classes de design participantes
  - descrição das interações entre objetos de design
  - identificação de subsistemas participantes e suas interfaces
  - descrição das interações entre subsistemas
  - captura dos requisitos de implementação para o caso de uso

Design de Casos de Uso

- Identificação das Classes de Design Participantes
  - estudo das classes de análise
  - estudo dos requisitos especiais
  - diagrama de classes de design
- Descrição das Interações
  - diagramas de sequência ou de colaboração

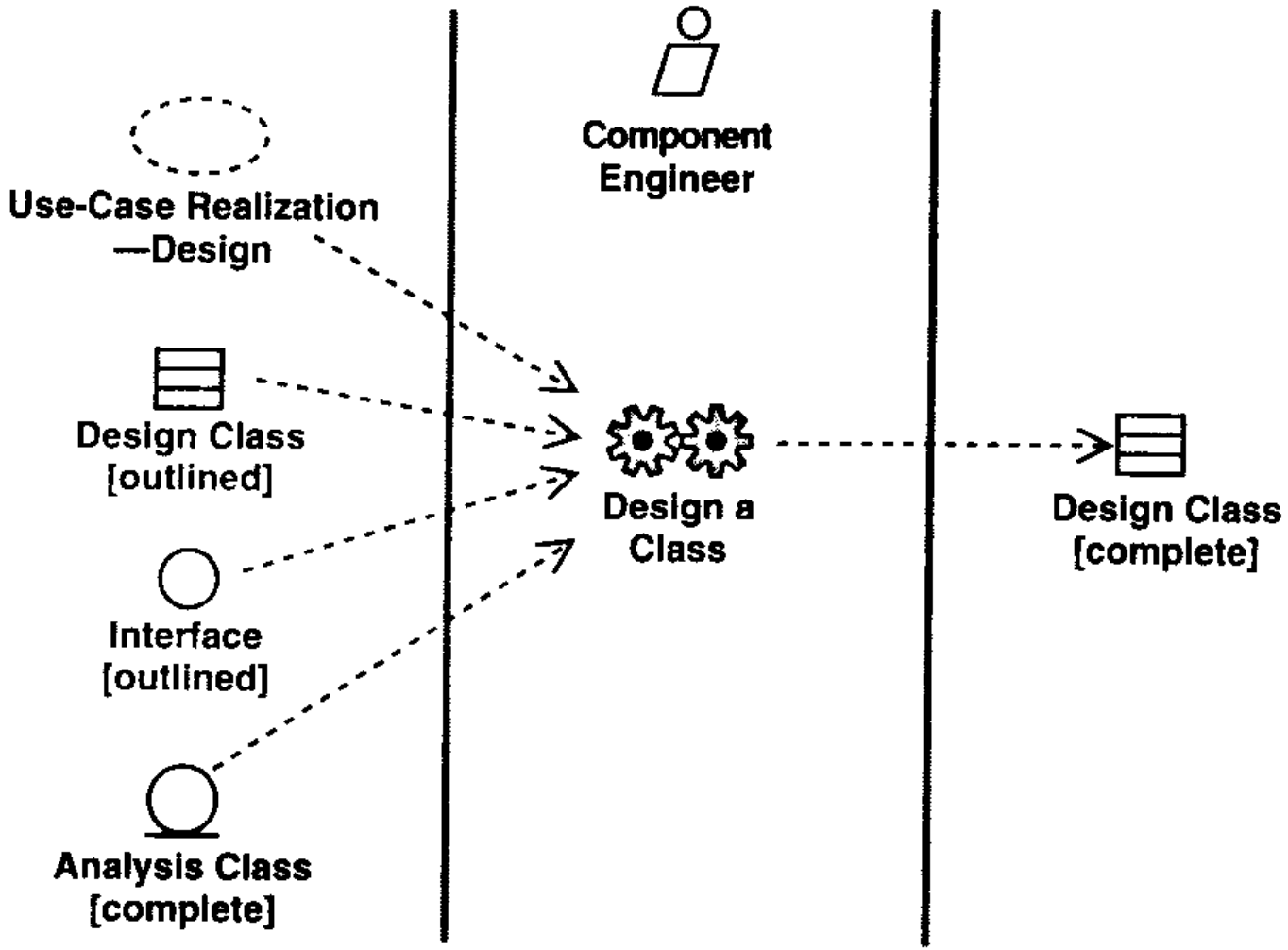




Design de Casos de Uso

- Identificação de Subsistemas e Interfaces
  - estudo das classes de análise
  - estudo dos requisitos especiais
  - diagrama de classes
- Descrição das Interações entre Subsistemas
  - diagramas de sequência
    - lifelines denotam subsistemas e não objetos
    - cada subsistema deve ter sua própria lifeline
    - mensagens dizem respeito a operações da interface do subsistema
- Captura de Requisitos de Implementação
  - captura de requisitos (não funcionais) que afetam diretamente a implementação

Design de Classes

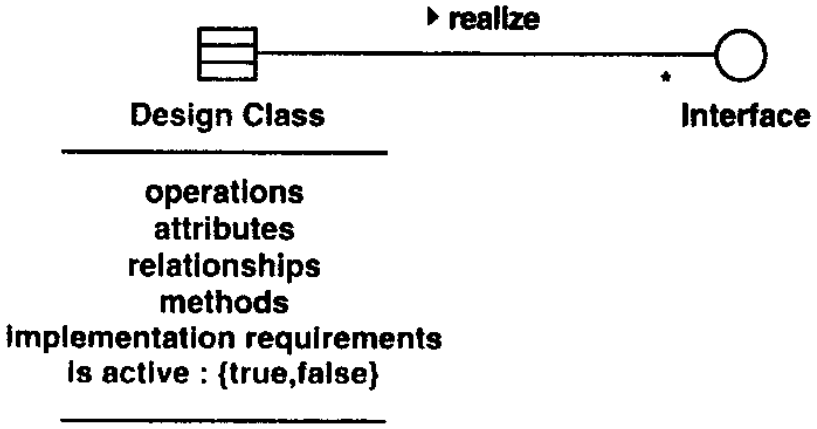


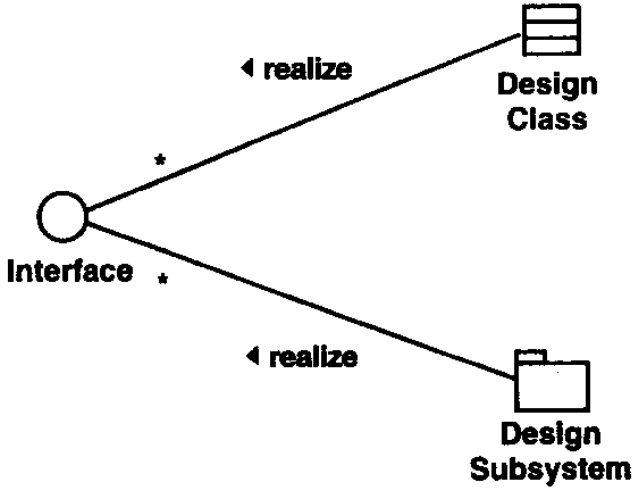
Design de Classes

- Objetivos
  - criação de classes de design que exercem seu papel na realização do caso de uso, obedecendo os requisitos não-funcionais que se aplicam
  - aspectos sendo detalhados:
    - operações
    - atributos
    - relacionamentos dos quais participa
    - métodos (como realizar as operações)
    - estados válidos
    - dependências para com mecanismos genéricos de design
    - requisitos importantes para a implementação
    - realização correta das interfaces com as quais está envolvida

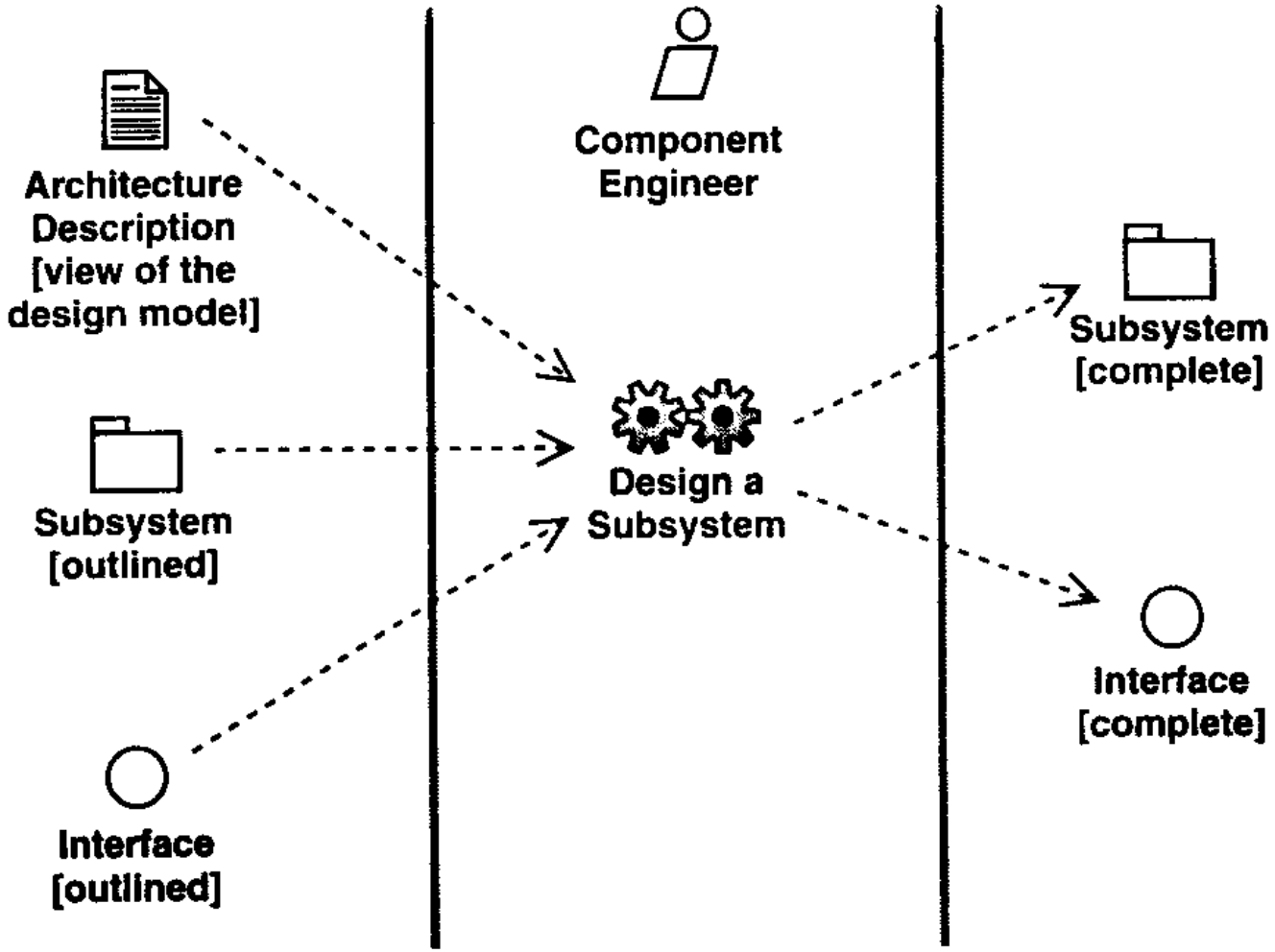
Design de Classes

- Sub-Etapas
  - criando um outline da classe de design
  - identificando operações e atributos
  - identificando associações, agregações e generalizações
  - descrevendo métodos
  - descrevendo estados
  - gerenciando requisitos especiais





Design de Subsistemas



Design de Subsistemas

- Objetivos
  - garantir que os subsistemas sejam tão independentes quanto possível de outros subsistemas e suas interfaces
  - garantir que os subsistemas provêm uma interface adequada
  - garantir que os subsistemas realizam seu propósito, ou seja, oferecem uma realização adequada das operações definidas por suas interfaces
- Sub-Etapas
  - Gerenciamento das Dependências entre Subsistemas
  - Gerenciamento das Interfaces providas pelo susbsistema
  - Gerenciamento do Conteúdo dos subsistemas
    - para cada interface, associa com as classes de design do subsistema que provê a funcionalidade