A Fase de Design no Processo Unificado

Ricardo R. Gudwin
DCA-FEEC-UNICAMP
09/11/2010

Introdução

Para compreendermos quais as finalidades da fase de design, é necessário fazermos uma comparação com os objetivos da fase de análise. A fase de análise enfatiza a compreensão dos requisitos em seu caráter detalhado. Ou seja, busca-se detalhar os conceitos e operações relacionadas ao sistema. Em outras palavras, na **fase de análise** o que se busca é saber **QUAIS** os processos, conceitos, … , etc, … relacionados ao software em desenvolvimento. Na **fase de design**, iniciamos o desenvolvimento de uma solução lógica baseada no paradigma da orientação a objetos. Ou seja, não mais nos satisfaz saber quais os processos, conceitos, etc. … relacionados com o software, mas saber **COMO** os processos, conceitos, etc… são implementados. Assim, diversos objetivos são esperados na fase de design. Dentre eles, temos:

- A aquisição de uma compreensão profunda dos fatores relacionados a **requisitos não-funcionais** e **restrições** relacionadas a diversas características do sistema, tais como: linguagens de programação, reutilização de componentes de software, sistemas operacionais, tecnologias de distribuição e concorrência, tecnologias de bancos de dados, tecnologias de interface com o usuário, tecnologias de gerenciamento de transações, etc…
- A criação dos elementos lógicos necessários para as atividades de implementação subsequentes, por meio da definição de subsistemas, interfaces e classes
- O 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 do design, permitindo e.g. a geração automática de código.

Vemos portanto que a etapa de design corresponde a uma mudança de postura em relação à fase de análise. Enquanto na fase de análise a nossa preocupação era compreender bem o que o sistema deveria fazer, na fase de design já se começa um planejamento estratégico para a implementação do sistema que será desenvolvido.

Analisando agora a etapa de design dentro do ciclo de vida de um software, ou seja, tendo em perspectiva a maturidade do desenvolvimento do mesmo em termos de iterações, veremos que o design é enfatizado nas iterações finais da fase de elaboração e o começo da fase de construção. Lembrando que as fases de especificação, análise, design, implementação e testes se repetem em cada iteração, com diferentes ênfases, veremos que somente a partir de um certo número de iterações a ênfase no design será maior. Entretanto, veremos que essa ênfase maior dura somente até o meio da fase de construção.

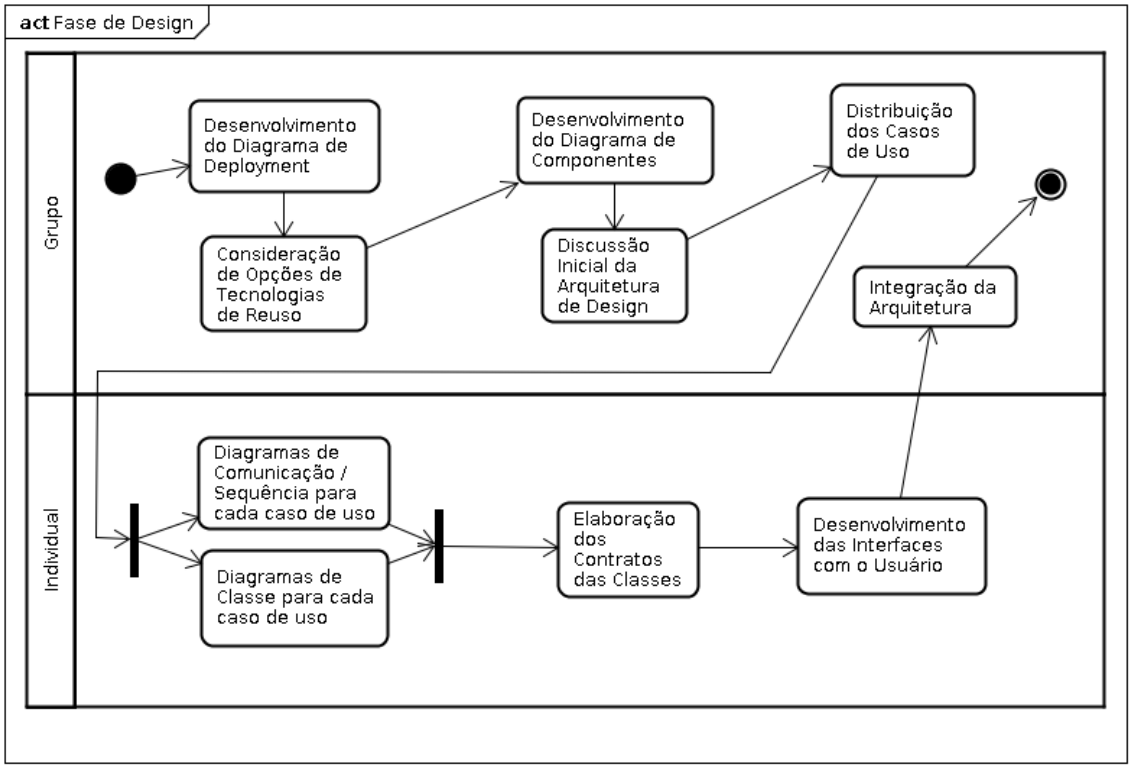
Ao final da fase de construção, a ênfase maior passa a ser sobre a etapa de implementação. De qualquer forma, a etapa de design é uma etapa crítica, pois as principais decisões quanto à arquitetura do sistema são tomadas nessa etapa. Portanto, veremos que a documentação do design será uma documentação mais detalhada, com diversos tipos de artefatos de software:

- Diagrama de Deployment
- Diagrama de Componentes da Arquitetura
- Arquitetura de Design, constituída por um conjunto de diagramas de classes estruturado
- Diagramas de Comunicação/Sequência para cada caso de uso
- Contratos (texto) para as operações de cada classe
- Interfaces com o Usuário

Em termos de atividades, a etapa de design foi customizada a partir do meta-modelo do Processo Unificado, em dez sub-atividades básicas:

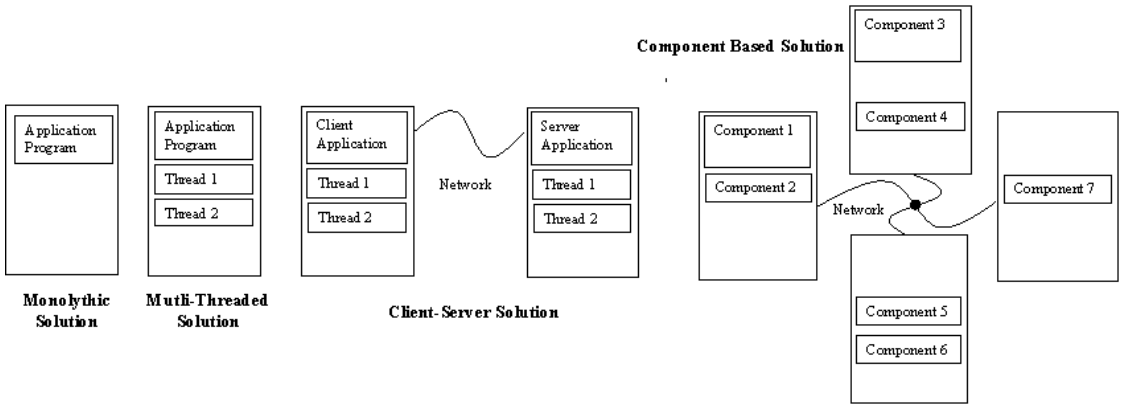
- O Desenvolvimento do Diagrama de Deployment
- A Consideração de Opções de Tecnologias de Reuso
- O Desenvolvimento do Diagrama de Componentes
- A Discussão Inicial da Arquitetura de Design
- A Distribuição dos Casos de Uso
- A Elaboração dos Diagramas de Comunicação/Sequência para cada caso de uso
- A Elaboração dos Diagramas de Classe para cada caso de uso
- A Elaboração dos Contratos das Classes
- O Desenvolvimento das Interfaces com o Usuário
- A Integração da Arquitetura

A sequência dessas atividades, algumas realizadas em grupo, outras realizadas individualmente é apresentada na figura a seguir.



Desenvolvimento do Diagrama de Deployment

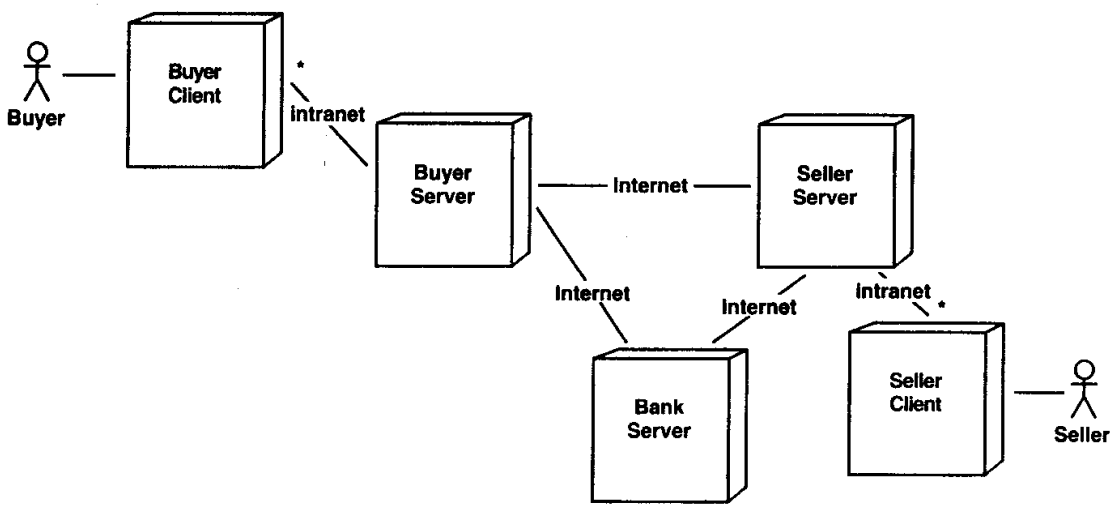
O primeiro passo de uma etapa que no meta-modelo do Processo Unificado é chamada de design arquitetural é a identificação do modelo de distribuição que se deseja implementar. Sistemas de software podem possuir diferentes modelos de distribuição, dependendo-se de como o sistema é distribuído (vide figura a seguir).



O tipo mais simples de solução é o das aplicações monolíticas. Aplicações monolíticas são aquelas que rodam de maneira completamente centralizada, dentro de uma máquina única, e em um único thread de controle. Esse tipo de solução é o mais antigo encontrado em sistemas de software e também o mais simples, pois ele desconsidera uma possível distribuição do sistema. Uma solução um pouco mais sofisticada é a que considera aplicações multi-threaded, ou seja, que se distribui por múltiplas threads de controle. Esse tipo de solução pode ser necessário, quando diversas tarefas devem ser executadas de maneira concorrente (por exemplo, uma animação tal como o ícone animado de um browser de internet, ao mesmo tempo que o sistema continua funcional). Para tornarmos essa solução mais sofisticada, teremos que promover a distribuição da aplicação em múltiplas máquinas. Assim, obtém-se a chamada solução cliente-servidor, que divide a aplicação em duas partes. Uma das partes roda em uma máquina de maior porte (o servidor), podendo se comunicar com diversos clientes, que constituem a outra parte da aplicação. Soluções do tipo cliente servidor são as mais utilizadas hoje em dia em aplicações comerciais. Diversos tipos diferentes de aplicações, tais como aplicações de web, bancos de dados, etc. utilizam esse tipo de arquitetura. Entretanto, não existe nada que nos impeça de distribuirmos nossa aplicação em mais de duas partes. Assim nasceram as primeiras aplicações baseadas em componentes. Ao contrário das aplicações do tipo cliente servidor, cada elemento participante da aplicação final pode ser um cliente, um servidor ou ambos. Assim, obtém-se uma total distribuição do sistema. Para a efetivação de soluções desse tipo, chamada de solução baseada em componentes, utilizam-se plataformas tecnológicas de suporte, tais como o CORBA, ou o DCOM.

Assim, para implementar esse primeiro passo do design arquitetural, devemos **identificar os nós e a configuração da rede** para nossa aplicação. Essa configuração pode ser fundamental para a aplicação em desenvolvimento. Aspectos dessa configuração de rede incluem quais os nós envolvidos e quais as suas capacidades, que tipos de conexões há entre os nós e quais protocolos utilizam, quais as características das conexões (em termos de capacidade, qualidade, etc) e se existe a necessidade de algum tipo de processamento redundante (para aumentar a confiabilidade do sistema, por exemplo). 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.

A configuração da rede é representada em nosso modelo por meio de um ou mais diagramas de deployment. Um exemplo de diagrama de deployment é mostrado a seguir:



Havendo diferentes possíveis configurações de rede para o produto (por exemplo, incluindo configurações para testes e simulações), estas devem ser descritas em diagramas de deployment separados.

Consideração de Opções de Tecnologias de Reuso

O reuso constitui-se hoje de uma das principais técnicas para tornar o desenvolvimento de software mais rápido, ágil e eficiente. Esse reuso, entretanto, pode ser um **reuso oportunista** ou um **reuso sistemático**. Em um reuso oportunista, durante a fase de design identifica-se que certas funcionalidades do sistema em desenvolvimento são semelhantes às funcionalidades de sistemas anteriormente desenvolvidos. Com isso, ao invés de realizar novamente todo o trabalho de design destas funcionalidades, opta-se por reaproveitar o trabalho previamente efetuado. Esse reaproveitamento pode ser tanto de um código (implementação) previamente desenvolvido como de um design previamente elaborado. Apesar desse reuso oportunista muitas vezes acelerar o desenvolvimento de novos sistemas, é hoje quase um consenso entre especialistas em desenvolvimento de software que um reuso *sistemático* deve ser fomentado nas empresas de desenvolvimento de software. Em um reuso sistemático, partes potencialmente reutilizáveis de um sistema já são concebidas originalmente com essa orientação. Dessa forma, durante o design os desenvolvedores podem considerar as diversas opções disponíveis e optar pelo reaproveitamento ou não de partes do sistema que estão projetando. Nesta linha de raciocínio, pode-se ir mais além, e poderemos perceber que estas partes reutilizáveis do sistema podem vir a constituir, por si só, um produto que, tanto pode ser desenvolvido pela própria empresa, como por outras empresas, eventualmente empresas especializadas em desenvolver partes reutilizáveis de sistemas. Atualmente, já existe um grande mercado de partes reutilizáveis, sendo que algumas companhias se especializaram em desenvolver essas partes. Assim, em alguns casos, pode-se considerar a aquisição dessas partes reutilizáveis de outras empresas quando se está efetuando o design de um sistema.

Existem basicamente três maneiras de se implementar o reuso:

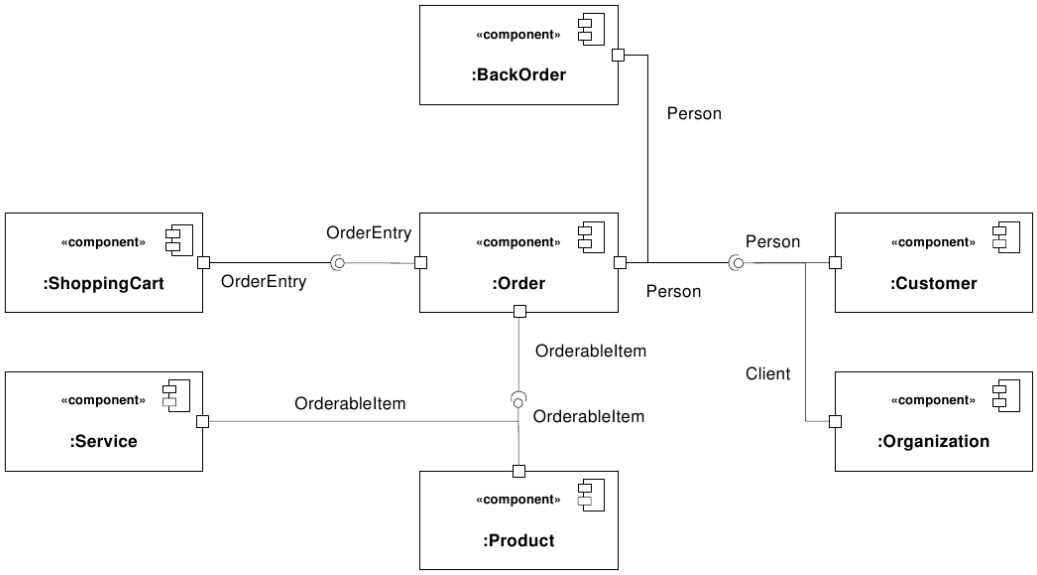
- O Reuso de Componentes
- O Reuso de Frameworks
- O Reuso de Designs

Componentes e **Frameworks** são duas maneiras diferentes de providenciar o reuso de *implementações*. Os assim chamados **Design Patterns** são uma maneira de providenciar o reuso de *designs*.

Particularmente, nesta etapa do design, cabe a nós decidir as partes do sistema que deverão ser reutilizadas, e a forma em que essa reutilização se dará. Nossas opções de reuso afetarão, consequentemente, as etapas posteriores de nosso *workflow* de design.

Desenvolvimento do Diagrama de Componentes

Esta etapa é uma etapa opcional, preparatória para a definição da arquitetura de design (que é iniciada na fase seguinte a esta), quando se opta por realizar o design do sistema orientado a componentes. Caso a opção seja realizar o design do sistema de forma orientada a frameworks, essa etapa pode ser suprimida e se passar diretamente à próxima etapa. A idéia desta fase é tentar fazer uma divisão inicial do sistema em **componentes** de alto nível, muitas vezes também chamados de **subsistemas**. Nesta etapa, que também faz parte do chamado design arquitetural no meta-modelo o objetivo é **a identificação de componentes e suas interfaces**. Subsistemas provêm um meio de organizar o design do sistema em partes gerenciáveis. A descoberta (e definição) de subsistemas pode ocorrer em diferentes etapas do desenvolvimento. Eles podem ser criados desde o início do design, ou podem ser desmembrados de outros subsistemas, a medida que um determinado pacote ou subsistema se torna muito grande e demande uma decomposição. Nesta etapa, desenvolvemos um diagrama de componentes UML, tentando modelar cada subsistema que podemos identificar de imediato como um componente, definindo-se as possíveis interfaces entre os componentes. Um exemplo de diagrama de componentes é mostrado na figura a seguir:



Observe na figura a representação de interfaces providas e interfaces demandadas, e o acoplamento entre elas para constituir o sistema. Esse diagrama visa constituir um primeiro ponto de partida para a definição da arquitetura de design, que deve ser iniciada na atividade a seguir.

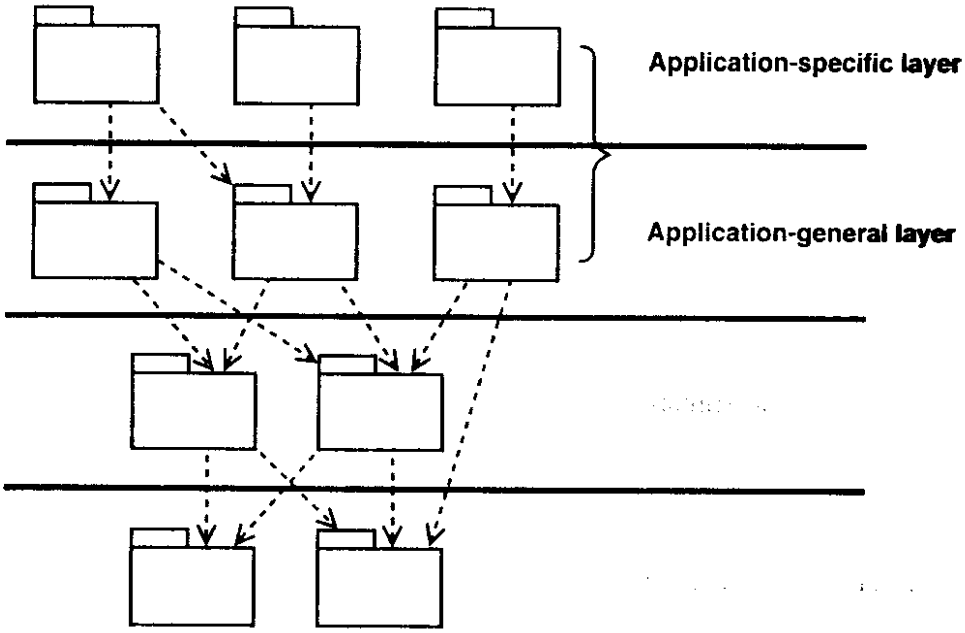
Discussão Inicial da Arquitetura de Design

Nesta etapa, iremos desenvolver um esboço da arquitetura do sistema delineado no diagrama de componentes da atividade anterior, de forma a refiná-la e detalhá-la. Esse refinamento se dará por meio de 4 sub-etapas, onde os componentes do diagrama de componentes são transformados agora em pacotes de um diagrama de classes. As interfaces poderão continuar a ser representadas em sua forma

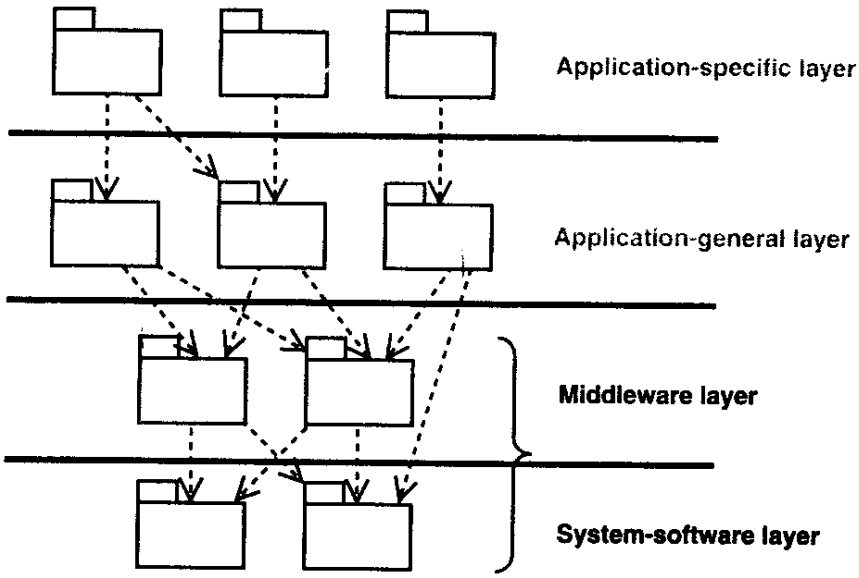
estereotipada, pelo menos nessa fase inicial. Nas etapas de refinamento posteriores, ainda durante o design, elas devem ser substituídas pela representação não icônica, com os detalhes da interface devidamente expostos. As 4 sub-etapas são as seguintes:

- identificação de subsistemas de aplicação
- identificação de middleware e subsistemas de software de sistema
- definição das dependências entre subsistemas
- representação das interfaces de subsistemas

A primeira delas é a identificação de subsistemas de aplicação. Nesse passo se identificam os subsistemas específicos da aplicação e as camadas de aplicação geral. Os subsistemas levantados na fase de análise podem ser utilizados como base para a definição destes subsistemas. Entretanto, esses subsistemas sofrerão agora um refino aqui no design.

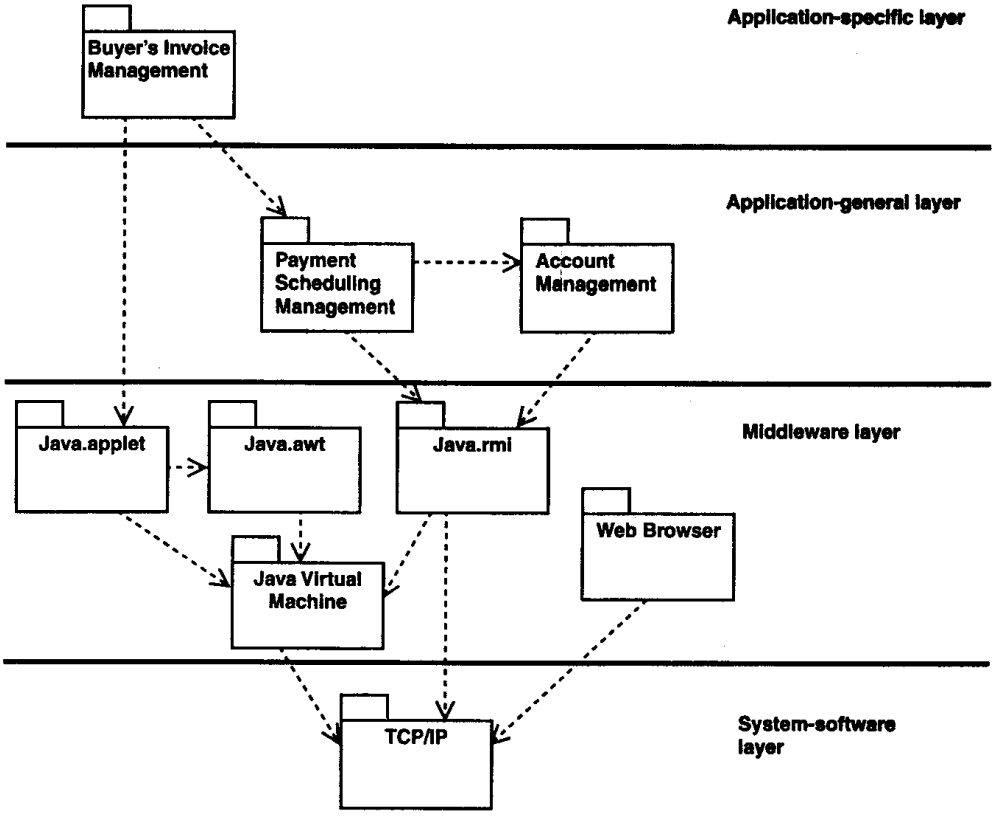


A etapa seguinte envolve a identificação do middleware e dos subsistemas de software do sistema.



O Middleware e o software de sistema são a base das funcionalidades de um sistema. Os elementos que devem ser definidos nessa sub-etapa são exatamente a determinação do sistema operacional, do banco de dados que será utilizado, do software de comunicação que se deseja utilizar, as tecnologias de objetos distribuídos, quais serão os kits de interface gráfica que iremos utilizar e as tecnologias de gerenciamento de transações que forem necessárias.

Uma vez que os subsistemas estejam devidamente identificados, procede-se então à sub-etapa de identificação das dependências entre subsistemas. Para tanto, analisam-se os subsistemas (principalmente aqueles que serão reutilizados), procurando-se identificar as possíveis dependências que hajam neles. Assim, caso haja algum tipo de relacionamento entre subsistemas, é possível haver alguma dependência entre eles. A navegabilidade da dependência deve ser equivalente a do relacionamento em questão. As dependências encontradas aqui devem ser análogas àquelas encontradas na fase de análise. Devemos nos lembrar de estar também identificando as dependências entre interfaces. Um exemplo de diagrama que identifica as dependências entre subsistemas é dado a seguir:



O passo seguinte é a **representação das interfaces dos subsistemas**. Nesse ponto é importante lembrarmos o significado de "interfaces". As interfaces dos subsistemas definem as 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. As interfaces para os subsistemas de middleware e de software de sistema normalmente são interfaces pré-definidas pelo produto utilizado. Um ponto importante a se colocar aqui é que não basta identificarmos quais são as interfaces, mas também identificarmos as operações disponibilizadas por cada interface.

O próximo passo é **a identificação das classes de design que são arquiteturalmente significativas**. Essas classes são normalmente derivadas das classes obtidas durante a análise. Outro tipo de classe arquiteturalmente significativa são as chamadas classes ativas, ou seja, aquelas que envolvem

considerações sobre os requisitos de concorrência do sistema.

Outro tipo de classe significativa é aquela que envolve 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).

Outro tipo ainda envolve classes que consideram a 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).

Ainda outro tipo são classes envolvendo outros requisitos, tais como requisitos de inicialização e shutdown, sobrevivência, precaução quanto a deadlocks, capacidade de reconfiguração de nós, etc ...

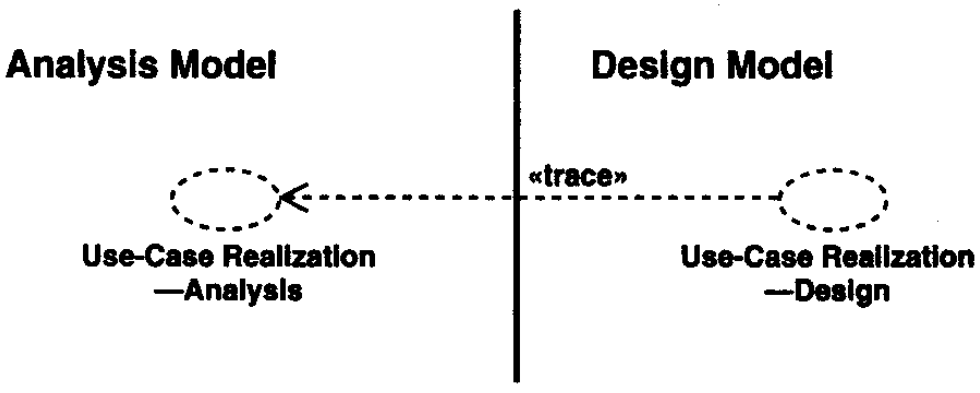
Por fim, para concluir esta etapa, desenvolve-se a **identificação de mecanismos genéricos de design**. Estudam-se os requisitos comuns, tais como os requisitos especiais identificados durante a análise, decidindo-se como implementá-los, dadas as tecnologias de implementação disponíveis. O resultado é um conjunto de mecanismos genéricos de design, instanciados em classes de design. Dentre os tipos de requisitos instanciados estão: a persistência, a distribuição de objetos (uso de objetos distribuídos), os requisitos de segurança, a deteção e recuperação de erros e o gerenciamento de transações.

Ao final desta etapa tem-se, de forma análoga ao que ocorria no design, um diagrama de classes estruturado que será o ponto de partida para a arquitetura de design do sistema.

Distribuição dos Casos de Uso

Uma vez que se disponha de uma primeira versão da arquitetura de design, desenvolvida na etapa anterior, pode-se passar às etapas subsequentes, que envolvem a chamada "realização dos casos de uso". Para tanto, deve-se distribuir os casos de uso entre os participantes da equipe, de tal forma que cada um possa desenvolver a realização dos casos de uso de maneira individual. Para cada caso de uso selecionado para desenvolvimento, iremos desenvolver um conjunto de artefatos que irão implementar a realização do caso de uso em questão. Essa realização se dará na forma das duas atividades consecutivas, a elaboração dos diagramas de sequência/comunicação e os diagramas de classe para cada caso de uso.

Assim, os casos de uso desenvolvidos na análise sofrerão agora um detalhamento que o consolidarão agora em termos de classes e subsistemas, preparando o terreno para uma implementação em uma linguagem orientada a objetos. Essa transformação pode ser representada na figura abaixo:



Elaboração dos Diagramas de Sequência/Comunicação para cada caso de uso

O principal objetivo dessa etapa é descrever as interações necessárias para a realização do caso de uso em questão. Dessa forma, para cada caso de uso, devemos desenvolver um ou mais diagramas de interação, onde a interação entre os objetos do sistema são descritas (veja o material sobre diagramas de interação para rever os detalhes, antes de prosseguir). Nesta etapa, criamos então os diagramas de interação (sequência ou de comunicação) que descrevem o caso de uso em questão. Esses diagramas de interação podem ser um refinamento dos diagramas desenvolvidos na análise, sem as classes estereotipadas, ou mesmo toda uma redefinição destas, caso não se mostrem convenientes em virtude das opções de reuso selecionadas.

Elaboração dos Diagramas de Classes para cada Caso de Uso

De modo simultâneo à etapa anterior, desenvolvemos os diagramas de classe para os objetos utilizados nos diagramas de interação sendo desenvolvidos.

Além disso, nesta atividade promove-se o detalhamento das estruturas internas das classes, considerando-se:

- 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

Elaboração dos Contratos das Classes

Uma vez que as classes tenham seus atributos, operações e relacionamentos detalhados, deve-se agora detalhar as operações das classes desenvolvendo-se os métodos para cada operação. Esses métodos devem ser formalizados por meio de contratos, desenvolvidos de modo análogo ao que foi feito na fase de análise.

Desenvolvimento das Interfaces com o Usuário

Uma vez que a interação entre os objetos que implementam a realização dos casos de uso já tenha sido concluída, pode-se agora focar nos requisitos não funcionais que afetam diretamente a implementação. Agora sim, questões de ordem estética-funcional devem ser abordadas, tais como o lay-out definitivo das interfaces gráficas e outras questões do design final do programa.

Integração da Arquitetura

Uma vez que os casos de uso foram realizados individualmente, por cada desenvolvedor, surge agora a necessidade de que os diagramas de classe sejam integrados na arquitetura do sistema, constituindo um único diagrama de classes estruturado, onde redundâncias entre classes sejam mitigadas, e as classes todas integradas constituindo uma única arquitetura de design do sistema.

Para tanto, os membros da equipe individual devem agora realizar uma grande reunião, onde cada desenvolvedor apresenta seu diagrama de interação realizando o caso de uso pelo qual ficou responsável, e integra as classes que utilizou na arquitetura do sistema.

Após a apresentação de cada membro, deve-se promover uma discussão sobre a estrutura geral da arquitetura, de forma a implementar melhorias e tornar a arquitetura mais consistente

Uma possível melhoria é garantir que os subsistemas sejam tão independentes quanto possível de outros subsistemas e suas interfaces, ao mesmo tempo que provêm uma interface adequada a suas operações, bem como realizem seu propósito, ou seja, ofereçam uma realização adequada das operações definidas por suas interfaces.

Inicialmente, devemos analisar os diagramas de classes e verificar se as dependências entre os diferentes subsistemas estão adequadas. Em seguida devemos analisar as interfaces de cada subsistema e verificar se podem ser aperfeiçoadas. Por fim, verificamos se todas as funcionalidades providas por cada subsistema estão devidamente representadas pelas interfaces dos subsistemas, ou seja, se para cada interface, existe uma associação com as classes de design do subsistema que provê a funcionalidade em questão.

Ao final desta etapa, a arquitetura de design fica definitivamente constituída e pode-se dar por encerrada a fase de design.