A Fase de Análise no Processo Unificado

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

Introdução

A fase de análise, também chamada muitas vezes de *análise dos requisitos* (não confundir com *especificação dos requisitos* nem com a *análise de domínio*), corresponde a uma fase onde faremos o aprofundamento das investigações acerca das especificações dos requisitos. Assim, procederemos ao detalhamento e refinamento dessas especificações.

O grande objetivo desta fase é a obtenção de uma melhor compreensão dos requisitos, mantendo uma descricão dos requisitos que seja compreensível e auxilie no posterior design do sistema. Podemos dizer que o que se faz na fase de análise é uma espécie de tradução das especificações dos requisitos, que se encontram na *linguagem do cliente*, para uma representação que use uma *linguagem do desenvolvedor*. Assim, passamos de um **modelo de casos de uso** para um **modelo de análise**. O modelo de casos de uso corresponde a uma visão do sistema sob a óptica do cliente - ou seja - viesado para o que o cliente deseja que seja implementado. O modelo de análise corresponde a uma visão do mesmo sistema já sob um ângulo diferente, ou seja, a descoberta e detalhamento do que o usuário necessita e sua descrição, preparando o terreno para a fase de design que virá no futuro, onde serão assumidos compromissos com relação às tecnologias que serão utilizadas para sua implementação.

Veremos que um dos principais desafios a serem vencidos na fase de análise é se manter um nível abstrato de investigação, sem entrar em questões de design do sistema.

O modelo de análise permite o refinamento do modelo de casos de uso por meio da especificação dos detalhes internos do sistema a ser construído. Dessa forma, provê um melhor poder de expressão e um formalismo mais refinado, principalmente para a descrição da dinâmica do sistema. Entretanto, apesar desse maior detalhamento, o modelo de análise deve usar somente de abstrações, evitando a solução definitiva de alguns tipos de problemas, que devem ser postergadas para a fase de design.

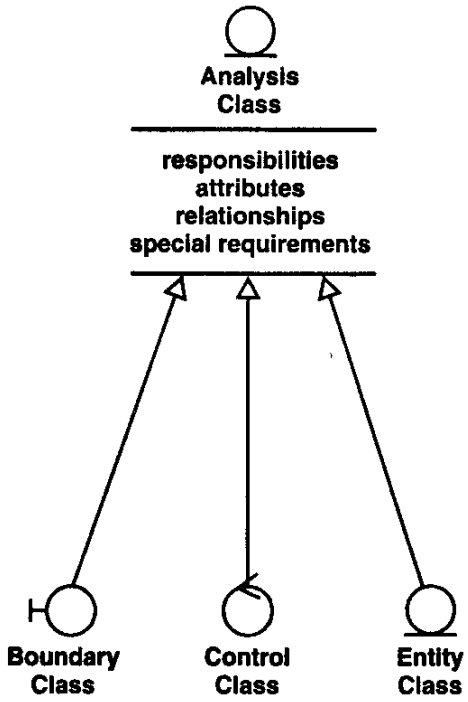
Veremos que algumas estruturas desenvolvidas na fase de análise nem sempre serão preservadas durante o design. Essa é uma das razões para que se mantenha estas estruturas de forma abstrata durante a análise. Estruturas muito amarradas podem se tornar um empecilho durante o design tendo que ser descartadas. Estruturas mais abstratas podem ter seu escopo negociado e compromissado quanto a flexibilidade quando se passar da análise para o design.

De um modo geral, podemos comparar as fases de análise e design no seguinte sentido. A fase de análise busca responder a seguinte pergunta: **o quê ?** O que o sistema deve fazer … o que o sistema deve ser … o que o sistema realiza. A fase de design, ao contrário, buscará responder a seguinte pergunta: **como ?** Como o sistema fará … como o sistema será … como o sistema realiza. Assim, durante a fase de análise, a compreensão de quais são as responsabilidades do sistema é mais importante do que como essas responsabilidades serão futuramente implementadas.

Durante a fase de análise, os principais artefatos de software desenvolvidos serão a **Arquitetura de Análise**, decomposta em diversos diagramas de classes que devem conter as **classes e pacotes de análise**. Além da arquitetura, teremos as **realizações de casos de uso** que serão modeladas por meio de **diagramas de interação** e dos **contratos** regulando o significado das mensagens nesses diagramas. Todos esses artefatos se integram, compondo a chamada **visão do modelo de análise**. Essa visão representa um ângulo por meio do qual podemos descrever o sistema, em termos de suas responsabilidades, sem nos comprometermos com sua implementação. Durante a fase de design, uma possível implementação será escolhida e planejada.

Classes de análise representam abstrações de uma ou mais classes ou subsistemas que irão aparecer no design do sistema. Podemos imaginá-las como grânulos grossos ou de mais alto nível por meio do qual estaremos a descrever a funcionalidade do sistema que estamos desenvolvendo. Dentre outras características, veremos que as classes de análise focalizam nos requisitos funcionais do sistema, postergando os requisitos não-funcionais (tais como os detalhes da interface, etc …) para a fase de design. Assim, torna-se mais importante na fase de análise a definição de interfaces em termos de responsabilidades, que poderão futuramente sofrer diferentes implementações. Assim, deve-se preocupar com a descoberta de atributos de alto nível e relacionamentos conceituais.

Uma das maneiras de se manter esse nível abstrato durante a análise é por meio da partição de nossas classes em três estereótipos: *boundary*, *control* e *entity*, conforme a figura abaixo:



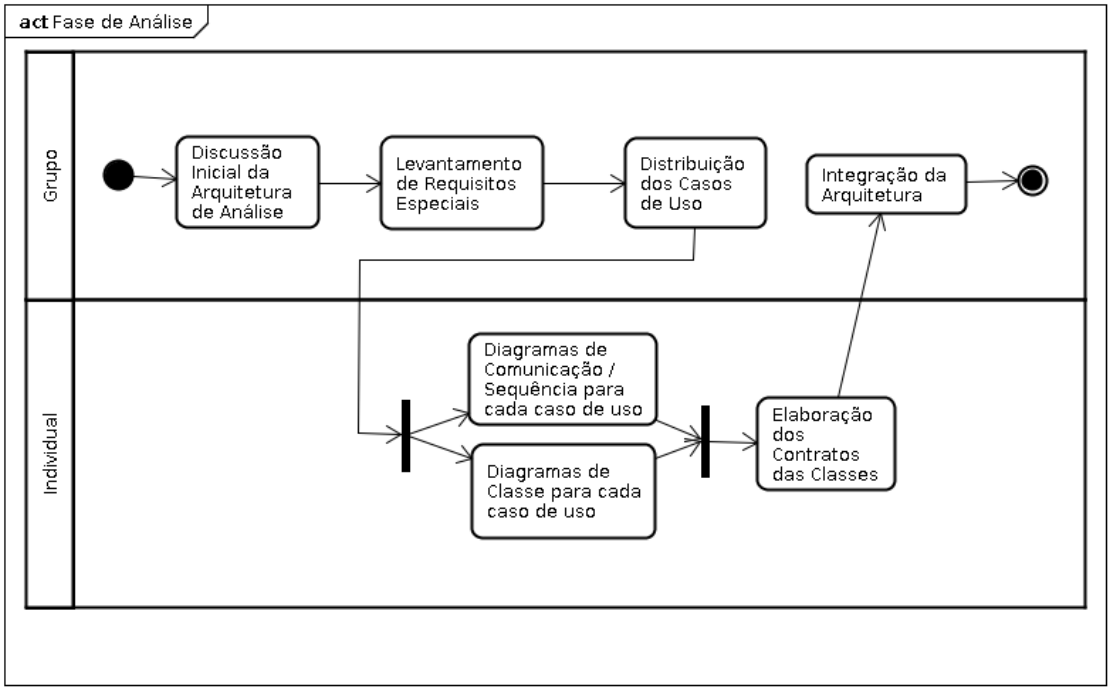
Classes do tipo *boundary* são classes utilizadas para modelar a interação entre o sistema e os atores. Essa interação envolve o recebimento e a apresentação de informações. Podemos então utilizar classes do tipo *boundary* para modelar janelas gráficas ou outros tipos de interface em que os atores enviem suas mensagens na forma de eventos, e que o sistema possa mostrar seu desempenho na forma de representações visuais ou sonoras.

Classes do tipo *control* são classes que representam a coordenação, sequenciamento, transações e controle de outros objetos. Podem ainda realizar derivações e cálculos com informações próprias ou oriundas de outros objetos. Assim, as classes do tipo *control* são classes que fazem alguma coisa, que realizam alguma atividade. Essa atividade pode envolver outros objetos de outras classes.

Por fim, classes do tipo *entity* são classes que modelam informações (dados) de longa duração, frequentemente dados que se desejam armazenar de maneira persistente (embora isso não seja necessário). Assim, classes do tipo *entity* são classes que comporão a estrutura de dados interna do sistema, que deverão ser manipulados por objetos de classes do tipo *control*, interagindo com o usuário por meio de classes do tipo *boundary*.

Por meio da criação de classes do tipo *boundary*, *control* e *entity*, somos capazes de descrever o sistema em um nível bem abstrato, como o necessário para os objetivos da fase de análise.

A fase de análise, em si, corresponde à sequência de atividades conforme pode ser visto na figura a seguir:



Definição Inicial da Arquitetura de Análise

O objetivo dessa fase é gerar um outline da arquitetura de análise, por meio da identificação de pacotes e classes que possam servir de pináculo para a posterior análise dos casos de uso. É certo que a arquitetura final de análise somente poderá ser concluída após a análise dos casos de uso. Entretanto, nessa etapa visa-se criar um ponto de partida unificado por meio do qual todos os diferentes desenvolvedores trabalhando individualmente sobre os casos de uso possam adotar um conjunto inicial de classes padronizado, diminuindo o esforço posterior na etapa de integração da arquitetura.

O resultado final que se espera dessa fase é um esboço inicial do diagrama de classes estruturado que constitui a **Arquitetura de Análise**. Um diagrama de classes estruturado é um conjunto de diagramas de classes associados mutuamente por uma relação de hierarquia, formando uma árvore. Em sistemas mais simples, esse diagrama será simplesmente um único diagrama de classes. Em sistemas mais complexos, a partir do diagrama raiz da árvore, muitas vezes chamado de diagrama de pacotes, associam-se diagramas subsequentes, um para cada pacote presente no diagrama raiz. Assim, sucessivamente, para cada pacote presente em um diagrama, associa-se um outro diagrama de classes, que poderá conter classes e/ou outros pacotes.

Inicia-se essa fase consultando-se os diagramas de atividade gerados na especificação, e para cada interface com o usuário identificada na especificação, gera-se uma classe do tipo *boundary*, que é inserida no diagrama de classes.

Após o levantamento de todas as classes do tipo *boundary*, deve-se acrescentar classes do tipo *control* controlando essas interfaces. Uma única classe do tipo *control* pode controlar mais de uma classe do tipo *boundary*. A escolha do número de classes do tipo *control* a serem inseridas é uma decisão dos desenvolvedores. Por fim, deve-se consultar o modelo conceitual de domínio, e dele extrair classes do tipo *entity* que se mostrem apropriadas aos tipos de dados necessários para a análise dos casos de uso. Não é necessário ser exaustivo nessa fase. Qualquer classe que não tenha uma utilidade óbvia não deve ser incluída.

À medida que o número de classes vai aumentando, pode ser conveniente re-organizá-las em pacotes. Pacotes permitem uma melhor organização do modelo de análise, dividindo-o em partes menores mais fáceis de serem gerenciadas. Essa divisão pode ser feita visando-se dividir o trabalho entre diferentes integrantes da equipe de desenvolvimento. Entretanto, deve-se fazer essa divisão alocando-se casos de usos com funcionalidades semelhantes em pacotes comuns, como por exemplo:

- casos de uso que suportam processos organizacionais comuns
- casos de uso que suportam um ator específico
- casos de uso relacionados por generalizações, extensões ou inclusões

Ao final dessa etapa, obtém-se uma versão inicial da arquitetura de análise, que servirá de ponto de partida para que a análise dos casos de uso possa ser iniciada.

Levantamento de Requisitos Especiais

Antes de proceder à análise de casos de uso, entretanto, é necessária ainda uma outra etapa preliminar. Nenhum projeto de software transcorre construindo-se o sistema todo da estaca zero. É bastante normal a reutilização de partes do sistema oriundas de sistemas anteriores, ou de componentes especialmente produzidos com a finalidade de serem reutilizados. Para uma série de requisitos do sistema, existem já soluções de prateleira prontas, que evitam o retrabalho em questões vitais da confiabilidade de um sistema. Requisitos especiais são aqueles que visam identificar a existência de partes do sistema que possam ser reutilizadas posteriormente no design. Dentre outros, requisitos especiais podem envolver requisitos de:

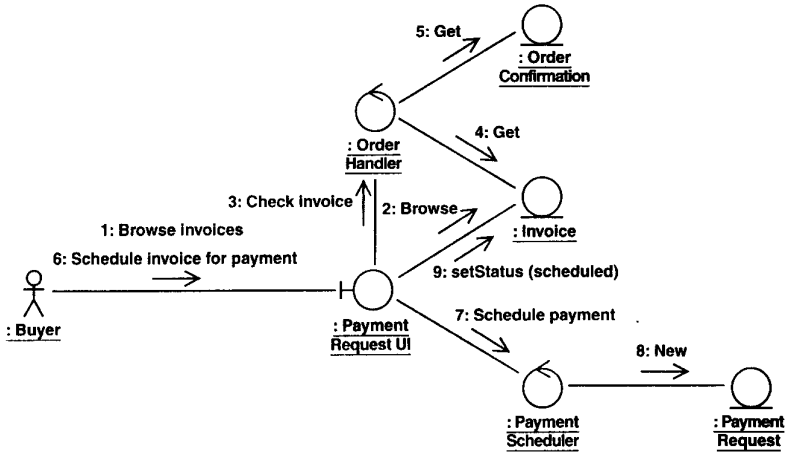
- persistência
- distribuição ou concorrência
- restrições de segurança
- tolerância a falhas
- gerenciamento de transações
- etc

Nessa fase, o grupo deve se reunir e deliberar sobre a existência de requisitos dessa natureza no projeto em desenvolvimento. Nesse momento, é suficiente a identificação de sua relevância para o sistema. Posteriormente, na fase de design, a equipe deve decidir se esses requisitos serão desenvolvidos pelo grupo ou se serão reutilizados de alguma maneira. Em alguns casos, os requisitos especiais só poderão ser encontrados durante a realização dos casos de uso. Ao final dessa fase, o resultado esperado é uma lista dos requisitos especiais que podem existir para o projeto.

Desenvolvimento dos Diagramas de Interação para cada Caso de Uso

O objetivo dessa atividade é implementar a assim chamada **Realização dos Casos de Uso**. Na fase de especificação, o sistema é visto sempre como uma unidade monolítica - o sistema. A grande mudança que se processa na fase da análise é exatamente a decomposição desse sistema em partes menores. Dessa forma, cada caso de uso detalhado por um diagrama de atividades deve agora ser realizado, desenvolvendo-se um diagrama de interação. No UML, existem basicamente dois tipos de diagramas de interação, os diagramas de comunicação (chamados também de diagramas de colaboração, no UML 1) e os diagramas de sequência. Esses dois diagramas, em princípio, são diagramas duais - um diagrama de comunicação pode ser re-escrito na forma de um diagrama de sequência e vice-versa. A escolha por um diagrama de comunicação ou um diagrama de sequência diz respeito somente ao número de objetos que devem interagir e à quantidade de mensagens que estes objetos devem trocar entre si. Caso haja um grande número de objetos, trocando entre si um número reduzido de mensagens, os diagramas de comunicação são mais adequados. Caso os mesmos objetos troquem entre si um grande número de mensagens, os diagramas de sequência mostram-se mais adequados.

Um exemplo de diagrama de comunicação é mostrado a seguir:



Observe algumas peculiaridades desse diagrama. Por exemplo, todos os nomes estão grifados, significando que se tratam de instâncias das classes e não as classes em si como no diagrama de classes. Como temos somente um objeto de cada classe não é necessário nomeá-los. Portanto, os nomes dos objetos são omitidos, e temos somente nomes do tipo **:NomeClasse.** Caso houvesse dois ou mais objetos de uma mesma classe, nomes poderiam ser acrescentados, resultando em algo como **Nome1:NomeClasse**, **Nome2:NomeClasse**.

Observe também que entre os objetos fluem mensagens que são numeradas e podem conter parâmetros. Neste ponto, somente indicamos as mensagens. Entretanto, devemos estar conscientes que em uma atividade subsequente, iremos especificar mais profundamente o significado destas mensagens por meio de contratos individuais para cada mensagem.

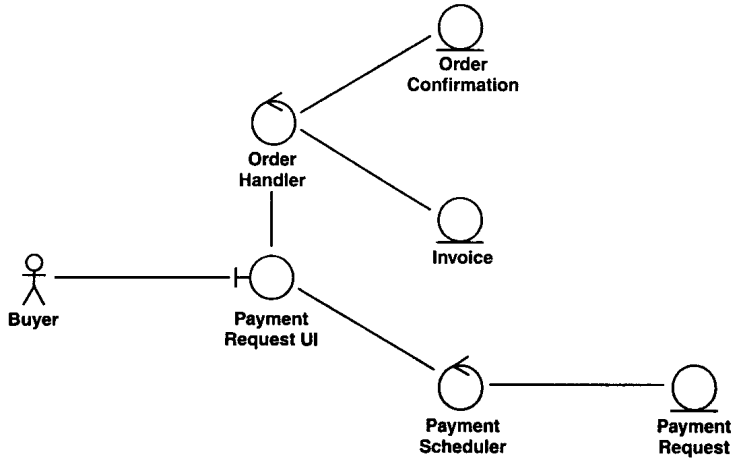
Essa atividade deve ser executada de maneira concomitante com a atividade a seguir, o desenvolvimento dos diagramas de classes para cada caso de uso. Muitas ferramentas CASE de edição de diagramas UML não permitem que um objeto de uma classe seja inserido em um diagrama de interação, se ele não estiver já definido em um diagrama de classes. Na prática, isso significa que o diagrama de interação precisa ser feito necessariamente em concomitância com o diagrama de classes.

Desenvolvimento de Diagramas de Classes para cada Caso de Uso

A realização de um caso de uso, além do diagrama de interação que o descreve, demanda também um diagrama de classes representando as classes que são utilizadas em um diagrama de interação. Esses diagramas de classes serão posteriormente integrados na arquitetura de análise do sistema.

Dessa forma, da mesma maneira que fizemos inicialmente no levantamento inicial da arquitetura, utilizaremos as idéias dos estereótipos *control*, *entity* e *boundary* para tentar delinear um conjunto de classes que dêem conta de realizar o caso de uso em questão. Deve-se partir das classes já existentes na arquitetura inicial de análise, e ir se inserindo novas classes à medida em que as mesmas mostrem-se necessárias para a realização do caso de uso.

Um exemplo de um diagrama de classes de análise pode ser visto na figura a seguir:



Há entretanto um desafio escondido aqui. Como os casos de uso são desenvolvidos individualmente, pode haver redundância na geração de classes entre diferentes desenvolvedores. Essa redundância precisará ser solucionada posteriormente, durante a fase de integração da arquitetura, quando os diagramas de classes desenvolvidos nessa fase devem ser consolidados em uma única arquitetura que integre todos os casos de uso do sistema.

Elaboração dos Contratos para as Classes

O objetivo desta atividade compreende a identificação e formalização das responsabilidades de uma classe de análise, baseadas em seu papel na realização de um caso de uso.

A identificação de responsabilidades pode ser perpetrada por meio da descoberta dos papéis que as diferentes classes assumem em diferentes realizações de casos de uso. Assim, as responsabilidades estão associadas às mensagens que uma classe pode receber.

A descrição das responsabilidades será formalizada por meio da elaboração dos chamados **contratos**. Contratos descrevem as responsabilidades de uma determinada classe em termos de quais mudanças no estado do objeto são realizadas quando este recebe mensagens (ou invocações de métodos). Esses contratos devem descrever **O QUÊ** o objeto deve fazer, sem explicar **COMO** ele o faz. Para cada mensagem aparecendo em um diagrama de interação, deve haver um contrato correspondente.

Os contratos devem estar organizados por classe. Dessa forma, mesmo que uma mesma mensagem apareça em mais de uma classe, deve haver um contrato diferente para cada classe onde a mensagem aparece.

Um contrato está dividido em diversas seções. Cada seção provê informações sobre uma parte específica do contrato. Nem todas as seções são obrigatórias, devendo aparecer somente quando necessário. Um contrato deve ter a seguinte estrutura:

Nome: nome da operação e eventualmente seus parâmetros
Responsabilidades: uma descrição informal das responsabilidades que a operação deve implementar
Tipo:conceitual, classe de software ou interface
Referências Cruzadas: outras classes que utilizam o mesmo contrato, etc.
Notas: informações adicionais, algoritmos, etc.
Exceções: casos excepcionais
Saídas Secundárias: saídas não relacionadas à interface com o usuário, e.g. mensagens de erros, logs, etc.
Pré-Condições: condições referentes ao estado do sistema antes da execução da operação
Pós-Condições: estado do sistema após a conclusão da operação

---

Para fazer um contrato, devemos inicialmente identificar as operações a serem reguladas por contratos, a partir dos diagramas de colaboração. Para cada operação, devemos implementar um contrato. Começamos pela seção "Responsabilidades", descrevendo informalmente o propósito da operação. Em seguida, passa-se à seção de "Pós-Condições", descrevendo declarativamente as mudanças no estado do objeto que devem ocorrer em virtude da operação. Para descrever as pós-condições, devemos atentar para os seguintes pontos:

- Criação e destruição de instâncias de outros objetos
- Modificação de Atributos
- Formação ou destruicão de associações

As pós-condições são a parte mais importante de um contrato. Por isso, devemos ter um cuidado todo especial em sua elaboração. Particularmente, devemos ter muita atenção para não confundirmos as pós-condições com as saídas secundárias. Outro ponto importante é lembrarmos que as pós-condições não são ações a serem executadas durante a operação, mas declarações sobre o estado do objeto após a conclusão da operação contratada. Algo que não podemos também esquecer é que as pós-condições dizem respeito ao modelo de análise. Portanto, as classes dos objetos criados devem existir no diagrama de classes, do mesmo modo que as associações.

Uma pergunta que muitas vezes aparece é a seguinte: quão completas devem ser as pós-condições.

Devemos ter consciência de que um conjunto de pós-condições completo e acurado é muito improvável - e nem mesmo necessário - durante a fase de análise. Os detalhes maiores poderão ser descobertos durante a fase de design.

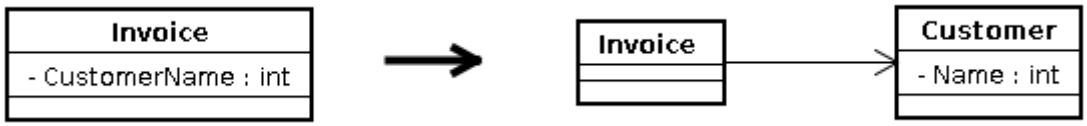
Agora voltemo-nos para as pré-condições. As pré-condições definem condições sobre o estado do sistema que são necessárias para que a operação possa ser realizada. Por exemplo: coisas que se deve testar a certo ponto da operação, ou coisas que não serão testadas, mas que são determinantes no sucesso da operação. Tais pré-condições devem ser documentadas para auxiliar a futura implementação da operação em fases futuras.

Após a determinação dos contratos, podemos nos dedicar à identificação de atributos. Os atributos especificam propriedades de uma classe de análise. Em princípio, devemos descrever, em uma classe de análise, somente os atributos que estão especificamente referenciados nos contratos. Por exemplo, caso um contrato especifique que o valor de um determinado atributo deve ser modificado, este atributo deve ser representado.

Podemos utilizar alguns guidelines para o levantamento de atributos. Por exemplo, sabemos que o nome de um atributo deve ser um substantivo. Do mesmo modo o tipo do atributo deve ser abstrato (pois estamos na análise). Assim, devemos utilizar por exemplo um tipo como "quantidade" ao invés de coisas como "inteiro". Devemos tentar reutilizar ao máximo os tipos utilizados. Devemos nos lembrar que os atributos são atrelados a cada instância do objeto e não à classe. Se houver muitos atributos, alguns deles devem ser transformados em classes. Por fim, devemos lembrar que nem sempre é necessário ter atributos.

Que atributos devemos incluir ? Somente aqueles que forem realmente necessários para satisfazer os requisitos de informação relevantes para os casos de uso sendo analisados e/ou aqueles que devem ser "lembrados" para serem utilizados em algum ponto de um caso de uso.

Algumas regras devem ser seguidas … por exemplo, … quando estivermos modelando quantidades, devemos usar um conceito separado "unidade" para mensurar atributos. Isso facilitará o entendimento do modelo e permitirá sua modificação mais facilmente. Da mesma maneira, devemos evitar fazer referência a atributos de classes externas como atributos (uma prática condenada batizada de uso de *foreign-keys*, ou chaves externas). Veja o exemplo da figura abaixo:



Na primeira especificação da classe, colocamos o nome do cliente (*CustomerName*) como um atributo de *Invoice*. Entretanto, esse tipo de representação pode nos causar problemas, pois existe uma entidade que é freguês (*Customer*) e o que queremos indicar é o nome desse freguês. Assim, a segunda representação é mais conveniente, pois permite que mudemos a estrutura da classe *Customer*, sem termos que alterar também *Invoice*.

Em seguida à identificação dos atributos, podemos nos dedicar à procura de associações e agregações. Sabemos que os objetos de análise interagem entre si por meio de conexões em diagramas de interação. Estas conexões são normalmente instâncias de associações entre as classes correspondentes. Deve-se pois estudar as conexões necessárias em um diagrama de interação de modo a determinar as associações necessárias no diagrama de classes, levando-se em conta que o número de relacionamentos entre classes deve ser minizado o quanto possível. Assim, devemos modelar somente os relacionamentos realmente necessários para se modelar a realização dos casos de uso.

Por fim, podemos identificar generalizações. Generalizações devem ser utilizadas para extrair comportamentos comuns ou compartilhados entre diferentes classes de análise. Assim, devemos olhar novamente as classes geradas anteriormente, tentando identificar comportamentos comuns que possam ser generalizados. Havendo a possibilidade, devemos então criar classes abstratas e apontar as generalizações possíveis.

Os resultados dessa atividade, portanto, serão a determinação da estrutura interna de cada classe de análise, em termos de seus atributos e seus métodos. Cada método deve estar associado a um contrato que descreve suas responsabilidades.

Integração da Arquitetura

A assim chamada Análise dos Casos de Uso (a soma das duas últimas atividades) é realizada de maneira individual por cada desenvolvedor, que gera os diagramas de interação e os diagramas de classes correpondentes para a realização dos casos de uso. Entretanto, durante essa atividade, há um grande potencial para redundância, onde classes com finalidades semelhantes e nomes distintos são geradas. Da mesma forma, pode ocorrer de classes com um mesmo nome, mas com propósitos distintos serem geradas, o que poderia gerar grande confusão, caso essas classes fossem simplesmente acrescentadas à arquitetura de maneira cumulativa.

O objetivo desta atividade é fazer uma integração criteriosa dos diagramas de classes desenvolvidos na etapa anterior à arquitetura inicial de análise desenvolvida no início da fase de análise, efetuando as correções necessárias para que se possa definir de maneira definitiva a arquitetura de análise.

Caso as classes estejam divididas em diversos pacotes, além disso deve-se garantir a independência entre os diferentes pacotes, diminuindo o acoplamento entre eles, e aumentando a coesão entre as classes dentro de um pacote.

Alguns guidelines podem ser utilizados para essa análise de pacotes. Devemos inicialmente definir e formalizar as dependências entre os pacotes, dependendo das classes que estes contiverem e seu uso de objetos de classes de outros pacotes. Essa dependência deve ser indicada explicitamente por meio de arcos do tipo dependência entre os pacotes aparecendo em algum diagrama da arquitetura. Devemos garantir que os pacotes contenham as classes corretas. Isso se faz mantendo-se os pacotes coesos, ou seja, incluindo somente objetos que estejam funcionalmente correlacionados. Devemos ainda tentar limitar as dependências entre pacotes. Assim, devemos considerar a relocação de classes que criem muitas dependências entre pacotes. Tais classes devem ser por exemplo transferidas para outros pacotes de tal forma que essas dependências diminuam.

Ao final desta atividade, os diagramas de classes levantados individualmente durante a análise dos casos de uso são descartados, e as classes correspondentes são então integradas à arquitetura de análise do sistema.

Ao final da fase de análise, os documentos gerados serão portanto os seguintes:

- Arquitetura do Sistema - composta por um diagrama de classes estruturado em diversos sub-diagramas
- Diagramas de Interação - Realização dos Casos de Uso
- Contratos - agrupados para cada classe da arquitetura
- Lista de Requisitos Especiais