ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Aplicação de padrões ao processo de desenvolvimento de software RUP

Trabalho de Conclusão de Curso

Engenharia da Computação

Tiago Moraes de Miranda Farias
Orientador: Prof. Márcio Lopes Cornélio

Recife, 20 novembro de 2006

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Aplicação de padrões ao processo de desenvolvimento de software RUP

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Tiago Moraes de Miranda Farias
Orientador: Prof. Márcio Lopes Cornélio

Recife, 20 de novembro de 2006

UNIVERSIDADE
DE PERNAMBUCO

Tiago Moraes de Miranda Farias

Aplicação de padrões ao processo de desenvolvimento de software RUP

Resumo

Processos de desenvolvimento de software e padrões são dois recursos utilizados para melhorar a qualidade dos softwares. Os processos de desenvolvimento de software organizam e controlam a produção de software, enquanto os padrões fornecem boas soluções para problemas encontrados em diversas atividades do desenvolvimento de software. A utilização combinada desses dois recursos pode ser uma importante ferramenta para fornecer qualidade na criação de softwares.

Este trabalho apresenta um estudo do uso de padrões no processo de desenvolvimento de software do RUP. Através do estudo de padrões de projeto, padrões de análise e *archetype patterns* identificamos em que fases do RUP esses padrões podem ser utilizados e como podem contribuir com a qualidade dos produtos desenvolvidos.

Utilizando alguns padrões como exemplos de modelos, este trabalho apresenta como é possível realizar a transição entre os modelos descritos utilizando os padrões estudados. Um modelo que descreve a solução de um problema utilizando *archetype patterns*, por exemplo, é descrito utilizando padrões de análise e padrões de projeto.

Através deste trabalho espera-se facilitar a utilização de padrões em um processo de desenvolvimento de software. Saber que tipos de padrões podem ser úteis em determinadas fases do processo pode ajudar a antecipar a solução de alguns problemas através da utilização dos padrões.

Abstract

Software development process and patterns are two features used to improve softwares quality. The software development processes organize and control software production whereas patterns provide good solutions to problems encountered in many activities of software development. The combined use of these two features can be an important instrument to provide quality in software creation.

This work presents a study on the use of patterns with the RUP software development process. Through the study of design patterns, analysis patterns and archetype patterns we identified the phases in which these patterns can be used, and how they can contribute in the quality of developed products.

Using some patterns as models examples, this work presents how it is possible to make a transition between models described using the studied patterns. A model that describes a problem solution using archetype patterns, for example, is described using analysis patterns and design patterns.

This work it aims to facilitate the use of patterns in a software development process. By knowing which types of patterns can be useful in certain process phases we can help to anticipate the solution of some problems by using these patterns.

Sumário

Índice de Figuras

Tabela de Símbolos e Siglas

RUP – *Rational Unified Process*
XP – *Extreme Programming*
MSF – *Microsof Solutions Framework*
UML – *Unified Modeling Language*
SI – Sistema Internacional de Unidades

Agradecimentos

Gostaria de agradecer primeiramente a Deus que sempre me orientou e me mostrou os caminhos a seguir, que sempre me confortou nos momentos difíceis e ilumina a minha vida com muita paz e alegria.

Agradeço a meu pai, José Carlos de Miranda Farias, meu melhor amigo, por continuar sempre me passando ensinamentos e compartilhando sua sabedoria.

Agradeço a minha mãe, Nádia Luna Moraes Farias, meu porto seguro, sempre presente, cuidadosa, atenciosa e carinhosa. Agradeço a ela por ter insistido que eu fizesse o vestibular de Engenharia da Computação, pois senão não estaria agora completando mais essa jornada.

Agradeço a meu irmão, Daniel Moraes de Miranda Farias, por ser mais que simplesmente irmão, ser sempre um companheiro, um amigo.

Agradeço a Deus por ter me dado uma segunda mãe, Ivonete "Vone" Fernandes da Silva (*in memorian*), a quem sempre amarei e que estará comigo para o resto da vida.

Agradeço a toda minha família que, sempre unida, me proporcionou uma base educacional e moral muito boa e que sempre está presente nos mais diversos momentos.

Agradeço a todos os meus amigos e amigas, companheiros para a vida toda, irmãos que conheci na vida e que sempre me acompanham. Aos amigos de infância, que cresceram comigo e acompanham todas as etapas da minha vida. Aos amigos de faculdade, muitos novos amigos que estiveram a meu lado nessa jornada e que passaram a fazer parte da minha vida, em especial ao grupo AMIGOSPOLI e a diretoria do grupo, Diogo Pacheco, Filipe Regueira e Herbert Menezes, amigos sempre presentes entre estudos e festas.

Agradeço Deus pela minha namorada, Mariana Pinheiro de Amorim, a quem conheci durante a faculdade e que desde então está sempre ao meu lado, que sempre me apóia e me dá amor e carinho.

Agradeço à empresa WPD Tecnologia por ter acreditado em mim, por ter ajudado bastante na minha formação profissional e por sempre fornecer as condições necessárias para minha formação acadêmica, ajudando sempre que precisei. Agradeço a todos que fazem dessa empresa uma grande família, um ambiente agradável de se trabalhar mesmo nos momentos difíceis. A todos os amigos que tenho na empresa, que nunca se negaram a me ensinar o que sabiam e a me ajudar quando precisei.

Agradeço a todos os professores do DSC que participaram da minha formação acadêmica, por fazerem com que eu tenha orgulho do curso que escolhi. Em especial ao professor Márcio Lopes Cornélio, meu orientador, por toda a sua dedicação e ajuda na concretização deste trabalho.

Agradeço a Deus por todas essas pessoas e por todas aquelas que por ventura não foram citadas, mas que também contribuíram para que mais um passo da minha vida fosse dado. Apenas um passo, mas importante passo, na longa jornada que é a vida.

Capítulo 1

Introdução

O desenvolvimento de softwares tem se tornado mais complexo ao longo dos anos. As exigências por parte dos clientes são cada vez maiores em termos de produtividade, qualidade de software e prazos cada vez menores [1]. O surgimento de novas tecnologias e a necessidade de realização de mudanças nos software desenvolvidos para atender às exigências dos clientes também dificulta a tarefa de desenvolver software com qualidade. Acompanhar as mudanças tecnológicas e atender às necessidades de mudança pode ser uma tarefa bastante complicada se o software não estiver preparado para suportar essas mudanças.

Alguns estudos sugerem que pouco mais da metade do esforço utilizado para o desenvolvimento de software esteja voltado para atividades voltadas para assegurar a qualidade de software [2]. Dois dos recursos utilizados como forma de buscar o desenvolvimento de software com qualidade são a utilização de processos de desenvolvimento de software e a utilização de padrões.

Entre os processo de desenvolvimento de software duas questões essenciais devem ser consideradas para assegurar a qualidade de software: o fornecimento de técnicas que auxiliem no desenvolvimento de software de qualidade e técnicas que assegurem os atributos de qualidade exigidos nos artefatos existentes [3]. A utilização de padrões durante o processo de desenvolvimento pode ser uma das técnicas utilizadas para assegurar qualidade nos documentos gerados.

O *Rational Unified Process* (RUP) [4] é o processo de desenvolvimento de software mais utilizado na indústria de software atualmente [3]. Uma das características principais desse processo é a busca contínua pelo desenvolvimento de software com qualidade. Na descrição do seu processo, o RUP define um guia a ser seguido para o desenvolvimento de software. O RUP define, por exemplo, diversas atividades que devem ser realizadas e diversos documentos que devem ser gerados a fim de garantir a qualidade do software que está sendo desenvolvido. No entanto, um processo de desenvolvimento de software é apenas um guia e, como tal, pode ser adaptado para ser utilizado em diversas situações.

Muitos dos documentos que são criados durante um processo de desenvolvimento de software são comuns a vários sistemas. Muitas vezes as mesmas técnicas são utilizadas para a criação desses documentos e, portanto, os mesmos problemas são encontrados. Os padrões apresentam uma forma de descrever soluções para esses problemas comuns baseado na experiência de outras pessoas.

Existem diversos tipos de padrões e esses padrões podem ser utilizados para auxiliar na criação dos documentos gerados em um processo de desenvolvimento de software como o RUP.

Padrões de projeto [5], padrões de análise [6] e *archetype patterns* [7] são apenas alguns dos diversos tipos de padrões que existem e poderiam ser utilizados juntamente com o RUP para buscar o desenvolvimento de software com qualidade.

1.1 Objetivos e Metas

O objetivo geral deste trabalho é apresentar o uso de padrões no processo de desenvolvimento de software do RUP. Diversos tipos de padrões podem ser utilizados em um processo de desenvolvimento de software para ajudar na criação de documentos e realização de tarefas. No entanto, muitas vezes, esses padrões não são utilizados por falta de conhecimento. Saber quando e como os diversos tipos de padrões podem ser utilizados, em um processo de desenvolvimento de software, pode ajudar a antecipar a consulta a determinados tipos de padrões durante o processo, auxiliando na tarefa de desenvolver software de qualidade. Este trabalho tem como objetivo utilizar três tipos de padrões e busca apresentar quando e como eles poderiam auxiliar no processo de desenvolvimento do RUP, além de apresentar em que fases do processo cada padrão mais se adequaria e poderia ser utilizado.

O objetivo específico deste trabalho é apresentar como esses tipos de padrões podem ser utilizados em conjunto para ajudar na descrição de problemas. Como podemos realizar uma transição de um problema descrito utilizando um determinado tipo de padrão para uma descrição utilizando outro tipo de padrão.

Um estudo de caso que utiliza o processo de desenvolvimento do RUP foi selecionado para apresentar como os padrões poderiam ajudar a resolver alguns problemas presentes.

1.2 Visão Geral do Trabalho

Este trabalho está organizado em seis capítulos. No Capítulo 2 apresentamos o processo de desenvolvimento de software RUP. É nele apresentada uma introdução a respeito do processo, apresentamos as seis boas práticas de desenvolvimento de software nas quais o processo se baseia e a estrutura como o processo é organizado.

No Capítulo 3 apresentamos padrões, o que são e quais os elementos essenciais de um padrão. Estudamos um pouco a respeito de três tipos de padrões: padrões de projeto, padrões de análise e *archetype patterns*. Apresentamos as definições, as características e as diferenças entre esses três tipos de padrões.

No Capítulo 4 mostramos como os três tipos de padrões podem ser utilizados em conjunto para ajudar na resolução de problemas. Através de dois exemplos estudamos como cada tipo de padrão pode ser utilizado para descrever o mesmo problema e como podemos representar esse problema passando de um padrão para outro.

No Capítulo 5 realizamos o mapeamento dos três tipos de padrões estudados para as fases do RUP. Apresentamos algumas disciplinas do RUP que se concentram na criação de artefatos em que os padrões poderiam ser utilizados para ajudar na criação de boas soluções. Verificamos as características desses artefatos criados e verificamos quais padrões mais se identificam com tais características.

No Capítulo 6 apresentamos as conclusões obtidas através do desenvolvimento deste trabalho, as contribuições realizadas e são apresentadas propostas para trabalhos futuros.

Capítulo 2

O *Rational Unified Process*

Desenvolver software com qualidade não é uma tarefa fácil desde o surgimento da indústria de software. A qualidade do software desenvolvido normalmente é avaliada em função de questões como a satisfação do cliente (principalmente ligadas a custo e prazo), a adequação aos requisitos funcionais (funcionalidades que espera-se que o sistema atenda), usabilidade, segurança, estabilidade, etc [3].

No começo os sistemas eram desenvolvidos como uma "forma de arte", pois havia poucos métodos sistemáticos e poucas pessoas utilizavam os métodos existentes. O desenvolvimento era realizado de maneira *ad-hoc*[1] e o sucesso de um projeto estava vinculado à qualidade individual dos profissionais envolvidos. Com o tempo os sistemas desenvolvidos se tornaram cada vez mais complexos e tornou-se evidente a necessidade de utilização de técnicas e métodos sistemáticos para o desenvolvimento de software, o que levou ao surgimento da engenharia de software [8].

Os processos de desenvolvimento de software surgiram como parte da engenharia de software e ajudam a assegurar a qualidade no desenvolvimento de softwares. São funções de um processo de desenvolvimento de software servir como guia para a execução de atividades, definir que produtos serão desenvolvidos e quando, assegurar a qualidade e coordenar as mudanças necessárias, e auxiliar o acompanhamento do progresso do desenvolvimento do software.

Atualmente existem diversos processos de desenvolvimento de software, entre os mais conhecidos estão: o *Rational Unified Process* (RUP) [4], o *Microsof Solutions Framework* (MSF) [9] e o *Extreme Programming* (XP) [10]. No entanto, o processo de desenvolvimento de software mais utilizado na indústria é o RUP [3], razão pela qual foi selecionado para o desenvolvimento deste projeto.

Este capítulo apresenta uma introdução ao *Rational Unified Process*. Apresenta o que é o RUP, as principais questões referentes a este processo e a estrutura em que está organizado.

2.1 Introdução

O *Rational Unified Process* (RUP) é um processo de desenvolvimento de software baseado no trabalho de Booch, Jacobson e Rumbaugh na definição do *Unified Process* (Processo Unificado)

[1] A palavra *ad-hoc* tem origem latina e significa: destinado para esta finalidade. O termo *ad-hoc* geralmente significa algo desenvolvido para um problema específico e que não pode ser adaptado para outros propósitos.

[11]. O foco principal do RUP é garantir a produção de sistemas com qualidade de maneira padronizada e, até certo ponto, previsível [4]. Para atingir esse objetivo o RUP é construído sobre seis boas práticas de desenvolvimento de software, como será apresentado no decorrer deste capítulo. O RUP pode ser definido também como um produto ou um *framework*[2].

O RUP possui diversas características comuns a produtos de software. Como os produtos de software o RUP foi projetado, desenvolvido, distribuído e recebe manutenção. Regularmente são disponibilizadas atualizações do processo. A descrição do processo, assim como exemplos de documentos utilizados ao longo do processo e outras informações são disponibilizadas através de um sistema Web (veja a Figura 1). Este sistema pode ser obtido através da internet ou CD-ROM e pode ser integrado com outras ferramentas de desenvolvimento de software.

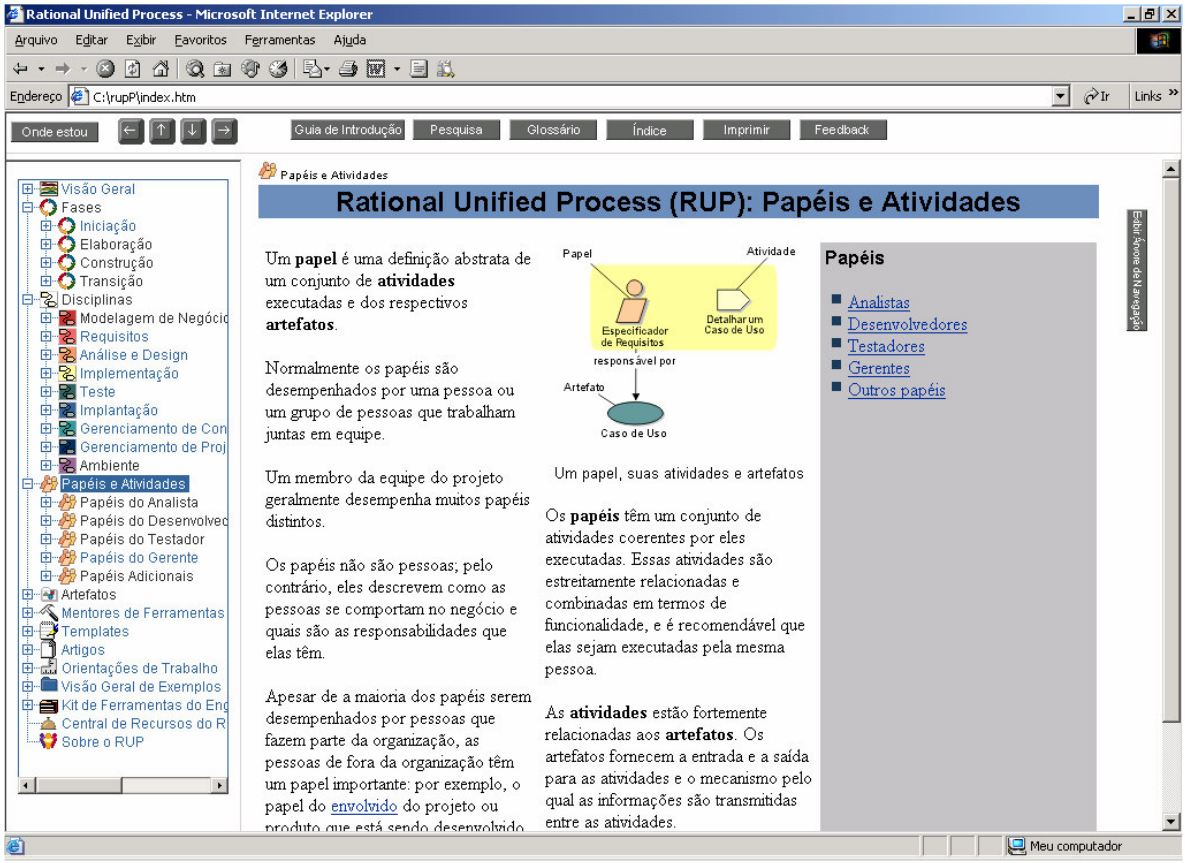

Figura 1. Página Web do RUP disponibilizada como ferramenta.

O RUP é também um *framework* de processos de desenvolvimento de software. Empresas que não necessitem de toda a metodologia ou documentação utilizada no processo de desenvolvimento proposto pelo RUP podem adaptá-lo. É possível utilizar apenas um subconjunto do que é definido pelo RUP, um exemplo disso pode ser visto em [12], como também é possível estender o processo adicionando novos métodos, técnicas, documentos, etc.

[2] *Framework* pode ser definido como uma estrutura definida para servir como uma base em que projetos de software podem utilizar para se organizar e facilitar o seu desenvolvimento.

2.2 Boas práticas do desenvolvimento de software

É grande o número de projetos de desenvolvimento de software que falham. Sistemas que não atendem às necessidades do usuário, mal documentados, softwares de baixa qualidade, difíceis de manter e integrar com outros sistemas, são alguns dos diversos problemas que ocasionam o fracasso de um projeto de desenvolvimento de software [1].

O RUP utiliza seis boas práticas de desenvolvimento de software para garantir o sucesso dos projetos que utilizem o processo. Essas boas práticas são abordagens comprovadas comercialmente que, quando combinadas, atacam a origem dos principais problemas que ocasionam o fracasso de um projeto de software, evitando que tais problemas ocorram ou antecipando-os de forma que o projeto possa ser reavaliado. As seis boas práticas adotadas pelo RUP são:

- desenvolver software iterativamente
- gerenciar requisitos
- utilizar arquiteturas baseadas em componentes
- modelar o software visualmente
- verificar a qualidade do software de forma contínua
- controlar as mudanças do software

2.2.1 Desenvolver software iterativamente

O modelo de desenvolvimento clássico da engenharia de software é o modelo cascata, como apresentado na Figura 2. Neste modelo é realizada uma abordagem seqüencial do desenvolvimento de software iniciando na engenharia de sistemas e passando por todas as etapas até a entrega do produto [8]. Um grande problema deste modelo é o impacto de erros causados nas etapas iniciais e descobertos apenas próximo ao fim do projeto.

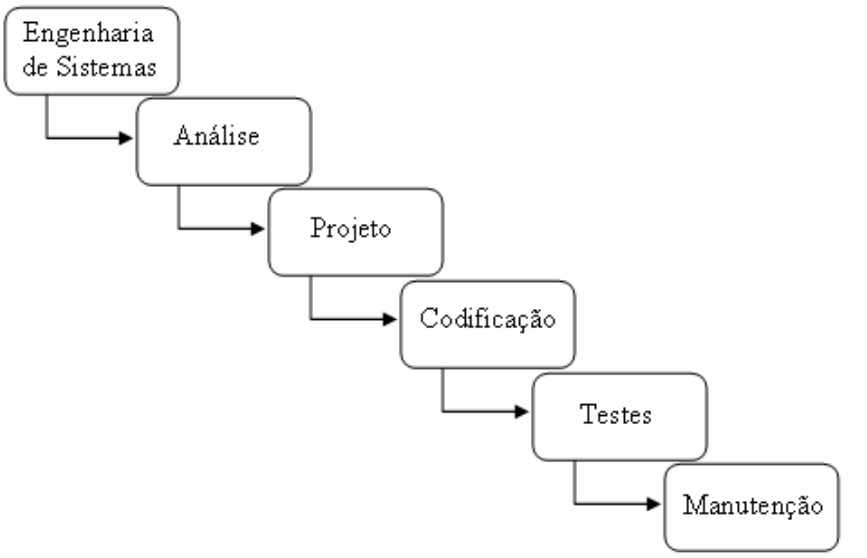

Figura 2. Ciclo de vida do modelo cascata de desenvolvimento de software.

O RUP utiliza um ciclo de vida iterativo e incremental como modelo de desenvolvimento. Nessa abordagem um conjunto de atividades em modelagem de negócios, requisitos, análise e projeto, implementação, testes e implantação são desenvolvidos de forma seqüencial a cada iteração, mas em diferentes proporções dependendo da fase projeto (veja Figura 3).

Utilizando essa abordagem os riscos do projeto são identificados e tratados logo no início do ciclo de vida de desenvolvimento, pois a cada iteração uma pequena parte do produto será produzida e validada. Esse modelo possui diversas vantagens sobre o modelo clássico, pois os riscos são reduzidos mais cedo, os requisitos são validados à medida que são desenvolvidos, a melhoria e o refinamento do produto são facilitados e a capacidade de reutilização aumenta.

Figura 3. Modelo de desenvolvimento iterativo e incremental.

2.2.2 Gerenciar requisitos

O gerenciamento de requisitos é uma abordagem sistemática para localizar, documentar, organizar e controlar os requisitos variáveis do sistema. Requisitos são funções ou capacidades que o sistema deve atender [4].

A tarefa de gerenciar requisitos possui diversos aspectos que exigem atenção. Os requisitos do sistema devem ser definidos de maneira clara e fácil de entender, precisam ser o mais precisos possível e precisam ser validados pelos interessados no projeto. Alguns fatores complicadores para o gerenciamento de requisitos é o fato de diferentes interessados possuírem diferentes visões a respeito dos requisitos, os requisitos também se relacionam entre si e, inevitavelmente, alguns requisitos serão alterados no decorrer do projeto.

O RUP recomenda a utilização de casos de uso como método de organização de requisitos funcionais. Casos de uso é uma técnica para capturar requisitos funcionais de um sistema. Os casos de uso descrevem as interações típicas entre os usuários do sistema e o sistema [13].

2.2.3 Arquitetura baseada em componentes

A arquitetura do software, possui um papel fundamental durante o seu desenvolvimento. É a partir da arquitetura que todos os envolvidos no desenvolvimento do projeto irão buscar o entendimento do que o software irá fazer, como irá funcionar e como estará organizado, por exemplo.

A arquitetura de um software define a organização de um sistema de software ou estrutura de elementos que o compõem. Determina como esses elementos interagem entre si e como se

combinam em subsistemas para compor um sistema maior. A arquitetura não se limita a questões estruturais e de comportamento, mas também atua em questões como usabilidade, funcionalidade, desempenho, recuperabilidade, reusabilidade, constantes econômicas e tecnológicas e estética [4].

No RUP, o início do processo de desenvolvimento é focado na definição da arquitetura do sistema, como será visto mais adiante neste capítulo. O RUP aconselha o uso de arquitetura baseada em componentes. Componentes podem ser definidos como um módulo, um pacote ou um subsistema que desempenha uma função clara e que pode ser integrado em uma arquitetura bem definida [4]. Os componentes utilizados para definir a arquitetura podem ser desenvolvidos e testados separadamente, ou mesmo adquiridos prontos de terceiros e, depois, integrados. Alguns componentes são desenvolvidos para serem reutilizáveis, pois resolvem problemas comuns em determinadas situações.

O RUP fornece diversos benefícios para o desenvolvimento de arquitetura baseado em componentes. Um desses benefícios é o fato de através do ciclo de desenvolvimento iterativo e incremental os componentes poderem ser desenvolvidos, testados e integrados de maneira incremental ao longo do projeto.

2.2.4 Modelagem visual de software

Modelos são utilizados para descrever sistemas de uma forma particular e simplificada da realidade. Através de modelos podemos entender e visualizar de uma maneira mais simples e clara o que deve ser desenvolvido.

A utilização de modelos em um processo de desenvolvimento de software é importante, pois permite que a equipe de desenvolvimento visualize, especifique, construa e documente a estrutura e o comportamento da arquitetura do sistema de software [4]. O RUP utiliza a UML (*Unified Modeling Language*) [13] como linguagem padrão para modelagem de forma a facilitar a comunicação entre a equipe de desenvolvimento.

Através da utilização de modelos e de uma linguagem padrão como a UML para descrever esses modelos, a compreensão de sistemas complexos é facilitada, alternativas de projeto podem ser melhor exploradas e comparadas, é possível a formação de uma base para implementação, requisitos são capturados com maior precisão e a comunicação de decisões entre a equipe é realizada com menos ambigüidade [4].

2.2.5 Verificação contínua de qualidade

No RUP a responsabilidade pela qualidade é atribuída a todos os envolvidos no desenvolvimento do software. O foco em relação à qualidade é dividido em duas áreas: qualidade do produto e qualidade do processo. A qualidade do produto está ligada à qualidade do software desenvolvido, assim como seus componentes, arquitetura, etc. A qualidade do processo verifica o grau de efetividade do processo definido, incluindo métricas e critérios de qualidade adotados para o processo, e qualidade dos documentos criados para dar suporte ao desenvolvimento do software.

O custo para identificação e correção de problemas no software é maior após a entrega do produto [4]. Para que possíveis problemas sejam identificados o mais cedo possível o RUP determina a verificação contínua de qualidade. A cada iteração o processo indica a realização de testes e avaliações de qualidade em diversos aspectos do software e do processo. Desta maneira é possível identificar os problemas cedo, reduzindo os custos de manutenção.

2.2.6 Controle de mudanças

Projetos de desenvolvimento de software geralmente envolvem diversas pessoas trabalhando juntas ou separadas, organizadas ou não em equipes, podendo estar até fisicamente separadas. Diversas atividades são realizadas e produtos são gerados e modificados com freqüência de uma forma que torna-se necessária a realização de algum tipo de controle.

O RUP fornece uma maneira de coordenar as atividades desempenhadas e produtos gerados pelos desenvolvedores e equipes participantes do processo de desenvolvimento. O controle é realizado através do estabelecimento de fluxos de trabalho para gerenciamento de mudanças do software e dos documentos gerados durante o desenvolvimento do software.

Em conjunto com o desenvolvimento iterativo, o gerenciamento de mudanças permite um maior monitoramento sobre as alterações realizadas. O controle de mudanças permite que mudanças de requisitos possuam um melhor monitoramento, facilita a organização e comunicação entre equipes trabalhando paralelamente e permite um maior controle sobre os produtos gerados no processo [4].

2.3 Estrutura do Processo

A estrutura do RUP é organizada em duas dimensões, como mostrado na Figura 4. O eixo vertical é a parte estática do processo, apresentando as principais disciplinas que compõem o processo. O eixo horizontal representa o tempo, o aspecto dinâmico do processo, e apresenta o aspecto do ciclo de vida do processo em termos de ciclos, fases, iterações e marcos.

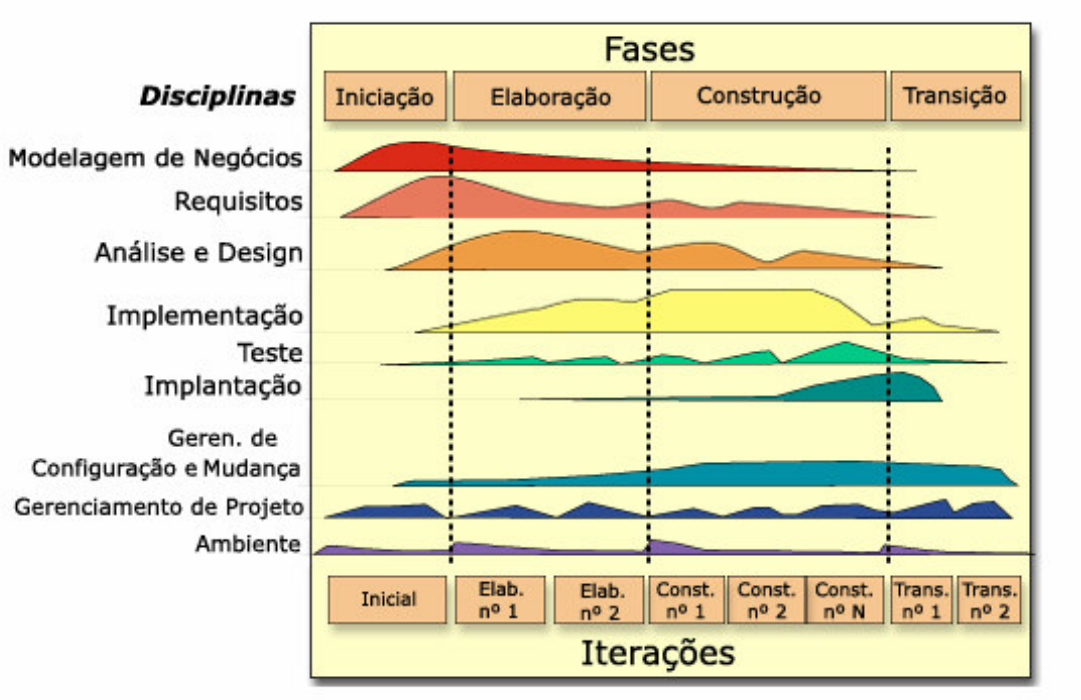

Figura 4. Gráfico do RUP – estrutura organizada em duas dimensões.

2.3.1 Estrutura estática

A parte estática do RUP, representada pelo eixo vertical da Figura 4, é descrita através dos conceitos de papéis, atividades, artefatos e fluxos de trabalho apresentados a seguir.

Papéis

Um papel define o comportamento e as responsabilidades assumidas por uma pessoa ou um conjunto de pessoas trabalhando em equipe. Cada papel é responsável pela criação, modificação e controle de determinados artefatos que são gerados a partir da realização de certas atividades ligadas ao papel em questão [4].

Uma papel não representa uma pessoa. Uma pessoa pode assumir diferentes papéis em momentos diferentes, devendo realizar as atividades pertinentes ao papel e gerar os artefatos devidos.

Atividades

Uma atividade é um trabalho desempenhado por um indivíduo. O indivíduo que realiza a atividade deve assumir o papel responsável pela realização de tal atividade. As atividades estão diretamente ligadas a criação ou manutenção de artefatos e são executadas repetidas vezes em diferentes iterações [4].

Artefatos

Um artefato é um pedaço de informação que é produzido, modificado e utilizado por um processo. Artefatos são utilizados como entrada para a realização de uma atividade e são resultados das saídas das atividades. Um artefato pode ser um documento, um modelo, um arquivo executável ou código-fonte, etc [4].

Fluxos de Trabalho

Um fluxo de trabalho, ou disciplina, descreve uma seqüência de atividades e interações entre papéis de forma a produzir um conjunto de artefatos. Esses fluxos são descritos de forma a apresentar todos os papéis, atividades e artefatos envolvidos, mostrando como esses papéis interagem para produzir os artefatos em um nível mais detalhado [4].

O RUP possui nove fluxos de trabalho principais, apresentados no eixo vertical da Figura 4, representando grupos de papéis e atividades separados por áreas de interesse. Através da definição de um plano de iteração é possível escolher quais atividades serão realizadas a cada iteração de acordo com a necessidade. A descrição dos fluxos de trabalho pode ser realizada através da UML utilizando diagramas de seqüência, diagramas de colaboração ou diagramas de atividades [13].

2.3.2 Estrutura dinâmica

A estrutura dinâmica do RUP representa o desenvolvimento do processo no decorrer do tempo. O processo é caracterizado pelo desenvolvimento iterativo e por uma evolução incremental e com foco na redução dos riscos. O ciclo de vida de um projeto utilizando o RUP é dividido em uma sucessão de pequenos ciclos do modelo clássico, em cascata, chamado de modelo iterativo [4].

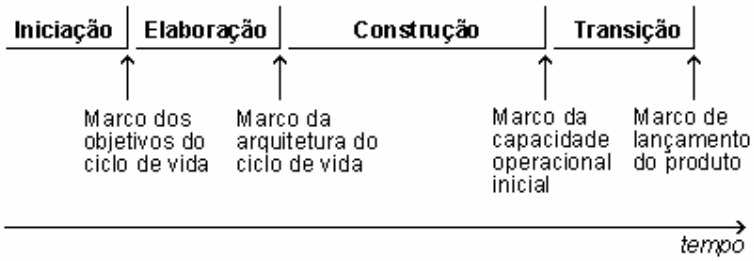

Figura 5. Fases do processo iterativo e seus marcos.

Em cada iteração o processo passa por todas as etapas do modelo clássico realizando as atividades para uma pequena parte do projeto. A cada nova iteração uma nova parte do projeto é desenvolvida e integrada ao que já foi realizado em iterações anteriores, evoluindo o projeto de maneira incremental.

Como forma de acompanhar e monitorar o progresso do projeto no desenvolvimento iterativo e incremental são definidos pontos no tempo em que o andamento do projeto é avaliado. Esses pontos, chamados de marcos, definem se o desenvolvimento irá prosseguir ou não, ou se será necessário alguma mudança. Existem quatro marcos principais no RUP que dividem as iterações em quatro fases: iniciação, elaboração, construção e transição. Cada fase é finalizada quando um dos marcos principais é realizado, como mostrado na Figura 5 [4]. A cada iteração do processo iterativo incremental atividades como levantamento de requisitos, análise, projeto, implementação e testes são realizadas em um ciclo de vida em cascata. No entanto, em cada fase do processo a ênfase em cada atividade muda, como podemos observar no gráfico da Figura 4. Abaixo detalhamos um pouco mais as quatro fases do RUP.

Iniciação

A principal meta da fase de iniciação é a definição dos objetivos do ciclo de vida do projeto. Nesta fase é estabelecido o escopo do projeto a ser desenvolvido, critérios de aceitação e o que deve ou não deve estar no produto; são identificados os casos de uso críticos do sistema; uma sugestão de arquitetura básica é apresentada; estimativas de custos geral e programação para o projeto; estimativas de riscos; e o ambiente do projeto é preparado.

O foco da fase de iniciação é concentrado nas atividades voltadas para modelagem de negócios, requisitos, gerência de projeto e ambiente. O marco para o final da fase é o marco dos objetivos do ciclo de vida, que avalia a viabilidade do projeto.

Elaboração

A principal meta da fase de elaboração é a definição de uma arquitetura estável para o sistema. A arquitetura definida deve ser estável o suficiente para diminuir os riscos do projeto e dar suporte à determinação dos custos e programação para a conclusão do desenvolvimento. Normalmente é desenvolvido um protótipo como forma de validar a arquitetura desenvolvida.

A fase de elaboração é focada nos requisitos, análise e projeto, além das atividades de gerência de projetos e ambiente, mas já é realizada a implementação do protótipo da arquitetura. O marco para o final da fase é o marco da arquitetura do ciclo de vida que estabelece uma base para a arquitetura do sistema.

Construção

Na fase de construção os últimos requisitos são detalhados e é concluído o desenvolvimento do sistema. Nesta fase os objetivos são para minimizar os custos de desenvolvimento, otimizando recursos e evitando retrabalho desnecessários; atingir a qualidade adequada; concluir a análise, implementação e testes de todas as funcionalidade; disponibilizar um produto completo e decidir se está pronto para ser implantado.

Na fase de construção as atividades de análise e implementação são o foco principal. O marco para o final da fase é o marco da capacidade operacional inicial, que determina se o produto está pronto para ser implantado.

Transição

A fase de transição deve assegurar que o software estará disponível para os usuários finais. Nesta fase os objetivos estão voltados para a realização de testes dos usuários, treinamentos, distribuição e vendas, engenharia voltada para implantação, atividades de ajustes, etc.

Na fase de transição o foco é voltado para testes e gerência de configuração. O marco final desta fase é o marco de lançamento do produto.

2.4 Resumo

Neste capítulo apresentamos a importância que um processo de desenvolvimento de software tem na busca de produção de softwares de qualidade. Apresentamos o RUP, processo de desenvolvimento de software mais utilizado pela indústria de software, e os benefícios que a adoção de um processo como o RUP proporciona.

Explicamos o que é o RUP, quais os seus conceitos e como pode ser utilizado. Mostramos que o RUP pode ser visto não apenas como um processo, mas como um produto ou um *framework* de processos de desenvolvimento.

Apresentamos seis boas práticas de desenvolvimento de software que ajudam a diminuir os riscos inerentes a projetos de software. Vimos como o RUP utiliza essas boas práticas em seu processo de desenvolvimento e os benefícios incorporados pela utilização delas.

Por fim, estudamos como o RUP é estruturado e que características possui. Apresentamos as características estáticas e dinâmicas ligadas ao processo e como funciona essa estrutura no processo. Conhecemos diversos conceitos utilizados pelo RUP, as fases em que se divide o ciclo de desenvolvimento do software e a importância dos marcos.

Capítulo 3

Padrões

As primeiras referências sobre o termo “padrões” surgiram em trabalhos desenvolvidos na área de arquitetura e engenharia civil por Christopher Alexander e alguns colegas. Através da publicação do livro *”A Patterns Language – Towns, Buildings, Construction”* (Uma Linguagem de Padrões – Cidades, Edificações, Construções) [14] Alexander e seus colegas descreveram técnicas de arquitetura e construções que resolviam problemas recorrentes em determinadas situações, o que chamaram de padrões.

Segundo Alexander, os padrões descrevem problemas que ocorrem com determinada freqüência em determinadas situações e apresentam elementos principais para a solução do problema de forma que a solução possa ser aplicada diversas vezes para resolvê-lo, mesmo que em diferentes maneiras [14].

Após a publicação do trabalho de Alexander surgiram diversos outros trabalhos a respeito de padrões. No entanto, o trabalho que mais se destacou na área de sistemas de software foi o trabalho desenvolvido por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides [5] em que são descritos Padrões de Projeto.

Este capítulo apresenta uma breve introdução a respeito de padrões e apresenta três tipos de padrões utilizados no desenvolvimento de softwares: Padrões de Projeto [5], Padrões de Análise [6] e *Enterprise Patterns*, conhecidos como *Archetype Patterns* [7].

3.1 Introdução

No desenvolver de uma atividade, muitas vezes nos deparamos com problemas que outras pessoas já resolveram, problemas que ocorrem com freqüência e que são, de certa forma, comuns. O tempo perdido para resolver esses problemas novamente poderia ser economizado através de alguma forma de consultar as soluções disponíveis para o problema. Além disso, o uso freqüente dessas soluções comuns para resolver tais problemas levaria a um amadurecimento das soluções aplicadas.

Com a utilização de padrões, as soluções para esses problemas comuns são descritas de uma maneira que possam ser utilizadas de uma forma mais rápida por uma pessoa que utilize um padrão como um guia. Um padrão é apresentado como um guia, pois não descreve a solução final para os problemas, mas descreve um ponto de partida que pode ser ajustado para resolver um problema específico.

3.1.1 O que são Padrões

Desde o surgimento do termo através do trabalho de Alexander diversos autores publicaram seus trabalhos a respeito de padrões e em diferentes áreas. Apenas na área de software, por exemplo, foram propostos padrões para diversas fases do processo de desenvolvimento: padrões de requisitos, padrões de análise, padrões de arquitetura, *archetype patterns*, padrões de projeto, entre outros [15].

No trabalho desenvolvido por Gamma et al [5] foi apresentado o seguinte conceito sobre o que seriam padrões:

"Um padrão consiste de uma descrição de um problema, da contextualização do problema e de uma possível solução para o problema dentro deste contexto."

Em seu trabalho os padrões são vistos como uma forma de distribuição de conhecimento, uma maneira de aumentar a reutilização de boas práticas e como uma forma de facilitar a comunicação sobre as soluções existentes. Um outro conceito apresentado para o que seriam padrões é o de Martin Fowler, no seu trabalho sobre padrões de análise:

"Um padrão é uma idéia que foi útil em um contexto prático e que provavelmente será útil em outros."

Independentemente da área, de uma maneira geral, os conceitos de padrões se assemelham. De modo geral os padrões descrevem problemas e apresentam soluções já utilizadas e que funcionaram em determinadas situações. Descrevem de uma maneira que outras pessoas possam reutilizar as soluções ao se depararem com problemas iguais ou semelhantes, podendo adaptar a solução proposta para o problema específico que deseja resolver.

3.1.2 Elementos Essenciais

Cada padrão definido no trabalho de Gamma et al [5] possui quatro elementos essenciais para identificá-los:

- O **nome do padrão** pode ser visto como um identificador para o padrão. É uma forma de descrever o problema, a solução e a conseqüência em poucas palavras. A utilização do nome do padrão facilita a comunicação entre pessoas que o utilizam, aumenta o nível de abstração que pode ser utilizado na hora de descrever a solução e discutir sobre o assunto, por isso é importante a escolha de um bom nome para o padrão proposto.

- O **problema** descreve quando o padrão deve ser aplicado. Descreve o contexto do problema que o padrão resolve propriamente. Um problema pode descrever também algumas condições que devem ser observadas antes da utilização do padrão.

- A **solução** descreve os elementos que compõem a solução assim como os seus relacionamentos e comportamentos. A solução não descreve uma solução concreta para o problema, pois ela representa apenas um modelo que pode ser aplicado em diferentes situações.

- A **conseqüência** representa os efeitos e resultados da aplicação do padrão. Descreve vantagens e desvantagens e pode ser utilizada para avaliar o impacto causado pela utilização do padrão.

3.2 Padrões de Projeto

Padrões de projeto são padrões que descrevem possíveis soluções a problemas comuns encontrados em projetos orientados a objetos [16]. A definição dada por Gamma et al [5] é de que padrões de projeto são descrições de objetos e classes que se relacionam e que são adaptados para resolver problemas de projeto em um dado contexto. Nos padrões são identificados as classes e objetos, como se relacionam, quais suas funções e quais as responsabilidades. Cada padrão de projeto busca resolver um problema específico em projetos orientados a objetos.

3.2.1 Descrição de um Padrão de Projeto

Descrever um padrão de projeto é uma tarefa muito importante para facilitar o entendimento e a utilização dos padrões. Através da descrição de um padrão é possível identificar particularidades que facilitem a comparação entre os padrões, auxiliando na decisão de escolha por um padrão específico. Para descrever um padrão de projeto existe um modelo, definido por Gamma et al [5], em que o cada padrão é dividido em seções como apresentado a seguir:

- **Nome do padrão e Classificação:** um bom nome é muito importante para um padrão, pois deve indicar a essência do padrão em poucas palavras. A classificação do padrão é apresentada como definida na Seção 3.2.2 a seguir.

- **Intenção do padrão:** a intenção do padrão descreve o que o padrão faz, qual a sua intenção e qual problema o padrão se propõe a resolver.

- **Também conhecido como:** um outro nome pelo qual o padrão é conhecido, caso exista.

- **Motivação:** um exemplo de problema de projeto e como a utilização do padrão resolve este problema.

- **Aplicabilidade:** condições em que o padrão pode ser aplicado, exemplo de situação em que o padrão pode ser aplicado e como identificar essas situações.

- **Estrutura:** uma representação gráfica das classes no padrão utilizando uma linguagem de modelagem, UML [13] por exemplo, em que possam ser representados os relacionamentos entre os objetos e seqüências de requisições.

- **Participantes:** apresenta as classes e objetos que fazem parte do padrão e quais as suas responsabilidades.

- **Colaborações**: como os participantes colaboram para atender as suas responsabilidades.

- **Conseqüências:** descreve como um padrão alcança seus objetivos, quais as decisões e resultados de usar o padrão e que aspectos da estrutura do sistema podem ser variados independentemente.

- **Implementação:** que detalhes de implementação devem ser observados o padrão for implementado. Deve ser observado se existem detalhes específicos a uma determinada linguagem.

- **Exemplo de código:** um exemplo de código que demonstre como o padrão deve ser implementado em alguma linguagem.

- **Usos conhecidos:** exemplos de padrões encontrados em sistemas reais.

- **Padrões relacionados:** apresenta padrões que se assemelham a este padrão e quais as diferenças. Apresenta também outros padrões que podem ser utilizados em conjunto.

3.2.2 Classificação dos Padrões de Projetos

Devido ao grande número de padrões de projeto existentes, assim como para facilitar a busca por padrões e o seu entendimento, os padrões foram agrupados e organizados. Os padrões foram classificados, segundo Gamma et al [5], de forma que é possível separá-los em famílias de padrões. A classificação dos padrões é realizada segundo dois critérios: o propósito e o escopo.

A classificação segundo o propósito agrupa os padrões de acordo com o que cada padrão faz. Nesta classificação os padrões são classificados como: padrões de criação, estruturais ou comportamentais. Ao classificar padrões segundo o escopo é especificado se o padrão se refere primariamente a classes ou objetos.

Os padrões de criação se concentram no processo de criação de objetos e, de acordo com o escopo, se referem a criação de objetos para subclasses (escopo de classe) ou para outros objetos (escopo de objeto).

Os padrões estruturais se preocupam com a composição das classes e dos objetos. Dependendo do escopo se utilizam de herança entre classes para realizar a composição (escopo de classe) ou apresentam formas de como agregar objetos (escopo de objeto).

Os padrões comportamentais tratam a forma como classes e objetos interagem e distribuem responsabilidade. No nível de escopo de classe, os padrões comportamentais utilizam herança para descrever algoritmos e controle de fluxo, enquanto que no nível de escopo de objeto demonstram como um grupo de objetos pode ser utilizado para solucionar tarefas mais complexas.

3.3 Padrões de Análise

Padrões de análise [6] são modelos de negócios de diferentes domínios. Estes modelos são definidos a partir da constatação de que a estrutura do negócio é uma estrutura comum, observada em determinados domínios.

3.3.1 O que são modelos

No desenvolvimento orientados a objetos o software é projetado de uma maneira que reflita a estrutura do problema. Ao realizar a análise de um problema projetamos uma estrutura que represente o que acontece no problema. Quando o problema e seus conceitos não são claros e simples de entender uma alternativa que podemos usar é a criação de modelos para representar o problema.

Um modelo é criado a fim de facilitar o entendimento do problema. Podem ser criados diversos modelos diferentes para representar um mesmo problema, o que significa que não existe apenas um modelo correto, mas existe um modelo mais adequado a cada situação [6]. Na realização da análise de um sistema a escolha do modelo pode afetar flexibilidade e reusabilidade. A escolha de um modelo errado pode torná-lo difícil de ser alterado, por ser pouco flexível, ou pode impactar a sua reusabilidade por ser muito complexo. O ideal é que o modelo escolhido seja simples e efetivo.

As técnicas de análise utilizadas para a criação dos modelos devem ser o mais independentes possível de tecnologias de software. Desta forma os modelos desenvolvidos podem ser úteis e aplicáveis a diferentes tecnologias.

3.3.2 O que são os padrões de análise

Alguns problemas de análise são difíceis de observar fora do contexto de um grande projeto. Esses problemas exigem do desenvolvedor alguma experiência em modelagem para que o problema possa ser melhor compreendido. Os padrões fornecem uma boa maneira de analisar estes problemas. Segundo Martin Fowler [6], os padrões não são criados, mas descobertos, pois eles apenas se tornam padrões quando observa-se que eles possuem um uso comum.

Baseado em suas experiências pessoais realizando modelagem de grandes sistemas de informação corporativos, Martin Fowler reuniu em seu trabalho uma série de padrões que descobriu analisando estes sistemas. Para esses padrões ele deu o nome de Padrões de Análise e fez a seguinte definição:

"Padrões de Análise são grupos de conceitos que representam uma construção comum na modelagem de negócios. Estes conceitos podem ser relevantes para apenas um domínio ou podem se estender a vários domínios."

Portanto, os padrões de análise definidos por Fowler são modelos que representam aspectos de negócios observados por ele. Os padrões de análise podem ser utilizados como um ponto de partida para a criação de modelos de negócios. Podemos utilizá-los para validar decisões de negócio de uma forma mais clara, ou mesmo para revisar modelos que utilizamos. Segundo Fowler os padrões não têm que ser utilizados na íntegra, mas podem ser adaptados para atender às reais necessidades de quem os utiliza.

3.4 Archetype Patterns

Archetype Patterns [7] são padrões que descrevem estruturas de negócios a partir da sua essência. Esses padrões de negócios são descritos em um nível de abstração maior que os padrões de análise, pois procuram representar os conceitos de uma forma mais geral. Utilizando *archetype patterns* é possível adaptar um conceito de negócio mantendo a essência que o identifica.

3.4.1 O que é Archetype

A origem da palavra *archetype* (no português: arquétipo[3], modelo ou original)[4] vem da palavra grega *archetypo*, que significa "padrão original" e pode ser definida como segue: [7]

[3] s.m. *Filos.* 1. Na filosofia platônica, modelo dos seres criados. 2. Modelo, padrão. Dicionário Michaelis

"Um *archetype* é uma coisa ou circunstância primordial que ocorre periodicamente de forma consistente e é considerado como uma situação ou conceito universal."

O psicologista Carl Gustav Jung [17] apresenta o conceito de *archetypes* partindo do conceito que ele chama de "inconsciente coletivo". Segundo ele os *archetypes* surgem de um ponto comum das experiências humanas. Um exemplo que pode esclarecer melhor o conceito defendido por Jung é o exemplo do herói. O modelo de um herói americano se comparado a um modelo de herói aborígene australiano seriam bem diferentes. No entanto, o herói seria identificado nos dois casos apesar da variação existente entre os dois modelos, o que definiria o arquétipo do herói [7].

Podemos dizer então que um *archetype* define a essência de um modelo de maneira tal que mesmo ocorrendo variações podemos identificar o conceito principal representado. Assim sendo, podemos imaginar como pode ser útil este conceito se aplicado ao desenvolvimento software. Através da utilização de *archetypes* podemos modelar conceitos de negócios de maneira que possamos reutilizá-los em diversas situações, pois podemos adaptá-lo e manter o conceito principal definido.

Na implementação de sistemas orientados a objetos buscamos modelar através de estruturas de classes e objetos os conceitos de negócios que desejamos representar. No entanto, em alguns desses conceitos podemos encontrar alguns *archetypes*. Por exemplo, se buscarmos modelar um sistema de compra de produtos, independente de como se realizará esta compra, podemos identificar conceitos principais que sempre farão parte do modelo como: produto, preço, comprador, vendedor, etc. Nesse contexto poderíamos identificar *archetypes* de negócios, assim como podemos identificar que entre esses conceitos também podem existir relacionamentos que sempre existirão. Esses *archetypes* e seus relacionamentos definem o que seriam os *archetype patterns*, que é definido da seguinte forma por Arlow e Neustadt [7] :

"Um *archetype pattern* de negócio é uma colaboração entre *archetypes* de negócios que ocorrem consistente e universalmente em ambientes de negócios e sistemas de softwares."

Algumas características essenciais são definidas para *archetypes* e *archetype patterns* como apresentado abaixo: [7]

- **Universal:** para ser considerado um *archetype*, deve ocorrer de maneira consistente em domínios de negócios e sistemas.

- **Difundido:** deve ocorrer tanto no domínio de negócios quanto no domínio de sistemas de software. Na construção de sistemas orientados a objetos os *archetype patterns* devem se comportar da mesma forma que no domínio dos negócios.

- **Possuir um Histórico:** o *archetype* deve possuir uma história, de maneira que seja um conceito bem formado e conhecido.

- **Evidente para especialistas:** um bom indicador para determinar um *archetype* é o fato de o conceito ser evidente para os especialistas no seu domínio. Um *archetype* que não é evidente pode não representar um *archetype*.

---

[4] Neste trabalho utilizaremos o termo *archetype* (em inglês) para facilitar o entendimento, pois não há uma tradução definida para o termo *archetype patterns*.

3.4.2 Archetype Patterns e Padrões de Análise

Archetype Patterns e Padrões de análise, apesar de modelarem aspectos no nível de negócios, não são a mesma coisa. *Archetypes* buscam reconhecer e identificar conceitos universais, enquanto que classes de análise não se preocupam com o conceito de universalidade. *Archetypes* estão sempre em um nível de abstração mais alto que as classes de análise [7].

Classes de análise mapeiam conceitos de negócios reais, representam uma abstração no domínio do problema. O objetivo das classes de análise é de facilitar a compreensão do negócio. Outro tipo de classes, as classes de projeto, são utilizadas para representar a solução em um nível mais técnico junto com a compreensão do negócio, como nos padrões de projeto. Os *archetypes* apresentam um nível de abstração maior que esses dois tipos de classes.

3.4.3 Características dos *Archetype Patterns*

Uma das principais características de um *archetype pattern* é a sua capacidade de variação. Como os *archetypes* representam a essência dos conceitos que modelam é possível adaptá-los para modelar o mesmo conceito de maneiras diferentes. Esta característica atende ao que é chamado de princípio da variação, que diz que domínios de negócios diferentes geralmente necessitam de diferentes modelos para representar o mesmo conceito [7].

Existem três tipos diferentes de variação suportadas pelos *archetype patterns* que possibilitam a sua capacidade de adaptação, são elas:

- Variação de *archetype*: um *archetype* pode possuir diferentes recursos (atributos, operações, constantes) para ser eficiente em diferentes contextos.

- Variação de *archetype pattern*: alguns recursos são opcionais no padrão, de maneira que podem ser omitidos se forem desnecessários para o domínio em questão.

- *Pleomorphism*: é um tipo especial de variação de *archetype patterns*, neste caso o padrão pode assumir diferentes estruturas (diferentes *archetypes*, recursos dos *archetypes* e relacionamentos) para se adaptar às necessidades de um contexto de negócio.

3.5 Resumo

Neste capítulo estudamos o conceito de padrões. O uso de padrões surgiu na área de arquitetura e engenharia civil, mas que logo despertou interesse de diversas outras áreas. Vimos que a utilização de padrões na área de desenvolvimento de software vem crescendo bastante e em diversas fases do processo de desenvolvimento.

Apresentamos os conceito de padrões e conhecemos seus quatro elementos essenciais: nome, problema, solução e conseqüência. Aprendemos como um padrão é importante para a distribuição de conhecimento.

Estudamos o conceito de padrões de projeto, aprendemos como é possível descrever soluções para problemas de projetos orientados a objetos utilizando padrões de projeto. Aprendemos também como classificar os padrões de projeto.

Estudamos também os padrões de análise. Aprendemos o que são modelos conceituais e como são importantes na realização de análise de negócios. Conhecemos a origem dos padrões de análise e sua definição.

Por fim, estudamos os *archetype patterns*. Aprendemos o que é um *archetype*, como os *archetype patterns* descrevem os problemas a partir da sua essência. Conhecemos as características principais dos *archetype patterns* e verificamos as formas que podem ser utilizadas para adaptar os *archetype patterns* sem perder a essência do conceito que ele define, um recurso importante para a reusabilidade deste tipo de padrão.

Capítulo 4

Transitando entre Padrões

Como vimos no capítulo anterior os padrões buscam apresentar boas soluções para problemas que normalmente ocorrem em determinadas situações. Cada padrão descreve uma solução para um problema em uma situação e cada tipo de padrão apresenta as soluções para os problemas em um nível de abstração. No entanto, nada impede que tipos diferentes de padrões apresentem soluções para um mesmo problema.

Um certo tipo de padrão pode apresentar um problema e descrever uma possível solução para este problema a partir de um determinado nível de abstração. Para esse mesmo problema podemos identificar que um outro tipo de padrão apresenta uma solução semelhante, mas utilizando um outro nível de abstração, apresentando uma nova visão para o problema em questão. Podemos ainda encontrar uma forma de implementar a solução para este problema utilizando-se um terceiro tipo de padrões.

No capítulo anterior discutimos três tipos de padrões: *Archetype Patterns*, Padrões de Análise e Padrões de Projeto. Neste capítulo buscaremos mostrar, através de um exemplo, como diferentes tipos de padrões podem descrever o mesmo problema, mas com visões e níveis de abstração diferentes, e como podemos utilizá-los.

4.1 Introdução

Os diversos tipos de padrões existentes normalmente são organizados em catálogos como uma forma de organizar os problemas já documentados que cada tipo de padrão se propões a resolver. No trabalho desenvolvido por Gamma et al [5], por exemplo, foi criado um catálogo com os 23 padrões de projeto descritos. *Archetype Patterns* [7] e Padrões de Análise [6] também foram organizados em catálogos de forma semelhante.

Os padrões de projeto descritos no catálogo definido por Gamma et al [5] buscam descrever soluções para problemas encontrados em projetos orientados a objetos, como já foi apresentado no capítulo anterior. Padrões de análise e *archetype patterns*, no entanto, agrupam padrões que descrevem soluções em um nível mais elevado, soluções no nível de regras de negócio.

Por tratarem de problemas em um nível mais próximo, ao analisarmos os catálogos de padrões de análise e *archetype patterns* podemos identificar padrões que descrevem os mesmos problemas, ou semelhantes. Alguns deles compartilham inclusive do mesmo nome para referenciar o padrão como: *Party*, *Inventory*, *Product*, *Quantity*, etc. Apesar de tratar de um nível

de abstração diferente dos demais, a solução apresentada para esses padrões pode ser implementada com o auxílio de padrões de projeto.

Um bom exemplo para descrever como esses tipos de padrões podem descrever um mesmo problema é através dos padrões *Quantity* e *Party*, presentes nos catálogos de padrões de análise e *archetype patterns*. A seguir apresentaremos como esses padrões são, ou podem ser, descritos em cada um dos três tipos de padrões apresentados. Dessa forma estaremos apresentando que, apesar de abordarem diferentes níveis de abstração, diferentes tipos de padrões podem ser combinados para ajudar a resolver um problema em comum.

4.2 Problema de representar quantidades

Em diversas situações e em diversos tipos de sistemas nos deparamos com a manipulação de valores. Em um sistema médico informações dos pacientes, como altura e peso, informações monetárias em um sistema financeiro, informações de estoque em sistemas de gerenciamento de estoque e diversos outros tipos de informações são manipuladas freqüentemente.

Esses valores geralmente representam uma determinada quantidade e estão associados a alguma unidade de medida. A altura de um paciente pode ser medida em centímetros ou metros, o dinheiro pode ser contado em reais ou dólares, um item de um estoque pode ser guardado em unidades ou caixas. Dependendo do sistema desenvolvido a maneira como essas informações são representadas e armazenadas pode ser bastante significativa.

Em sistemas orientados a objetos as quantidades podem ser representadas de diversas formas. Uma quantidade pode ser representada como um atributo de um objeto ou como um objeto associado por exemplo. No entanto, a forma como as quantidades serão representadas podem impactar diretamente no sistema à medida que é preciso considerar questões como converter valores de uma unidade de medida para outra por exemplo.

A seguir apresentaremos como podemos utilizar padrões de análise, *archetype patterns* e padrões de projeto para descrever uma possível forma para representar quantidades. Nos padrões de análise e *archetype patterns* existem padrões específicos que descrevem uma possível solução do problema, entretanto, também podemos aplicar padrões de projeto na descrição dessa solução.

4.2.1 Solução: Padrões de Análise

No catálogo de padrões de análise definido por Fowler [6] existe o padrão *quantity*, em que é definido um novo tipo associando um valor a uma unidade para representar uma quantidade de algo, como mostrado na Figura 6. A idéia é substituir a utilização de um atributo por um novo tipo (objeto) que represente a quantidade. Esse objeto pode conter operações que podem ser aplicadas ao valor armazenado.

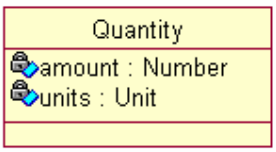

Figura 6. Representação do padrão de análise *Quantity*.

No entanto, apenas através da substituição de um atributo que represente a quantidade por um novo objeto que associa um valor a uma unidade não resolvemos os problemas envolvidos na representação de quantidades. Problemas como conversão entre unidades não estariam bem tratados através desta representação. No catálogo de padrões de análise existem outros dois padrões definidos que podem ser associados para resolver o problema de conversão de unidades, são eles: *Conversion Ratio* (Figura 7) e *Compound Units* (Figura 8). O padrão *Conversion Ratio* representa como podem ser realizadas conversões dos valores entre unidades. Para isto existe uma relação entre uma unidade e uma taxa de conversão, associada a um número, que define a qual será a taxa utilizada para realizar a conversão entre duas unidades. Entretanto, não se deve esquecer que existem unidades compostas, como m^2 (metro quadrado) por exemplo. Através do padrão *Compound Units* pode-se representar tais unidades realizando uma distinção entre unidades atômicas e unidades compostas, que utilizam unidades de referência, associadas a um valor inteiro, para compor unidades.

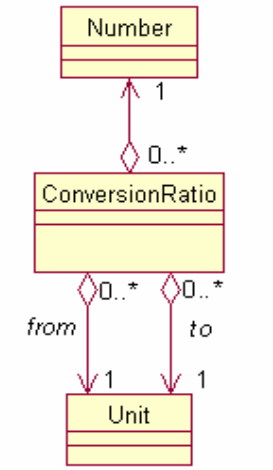

Figura 7. Representação do padrão de análise *Conversion Ratio*.

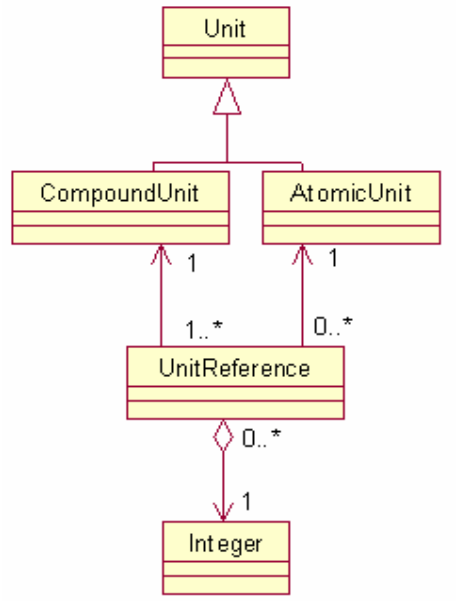

Figura 8. Representação do padrão de análise *CompoundUnit*.

Ao combinar esses três padrões definidos no catálogo de padrões de análise, colocando a classe de unidades como ponto central entre os três padrões, apresentamos o que seria uma boa solução para representar quantidades em um sistema orientado a objetos no nível dos padrões de análise, como apresentado na Figura 9.

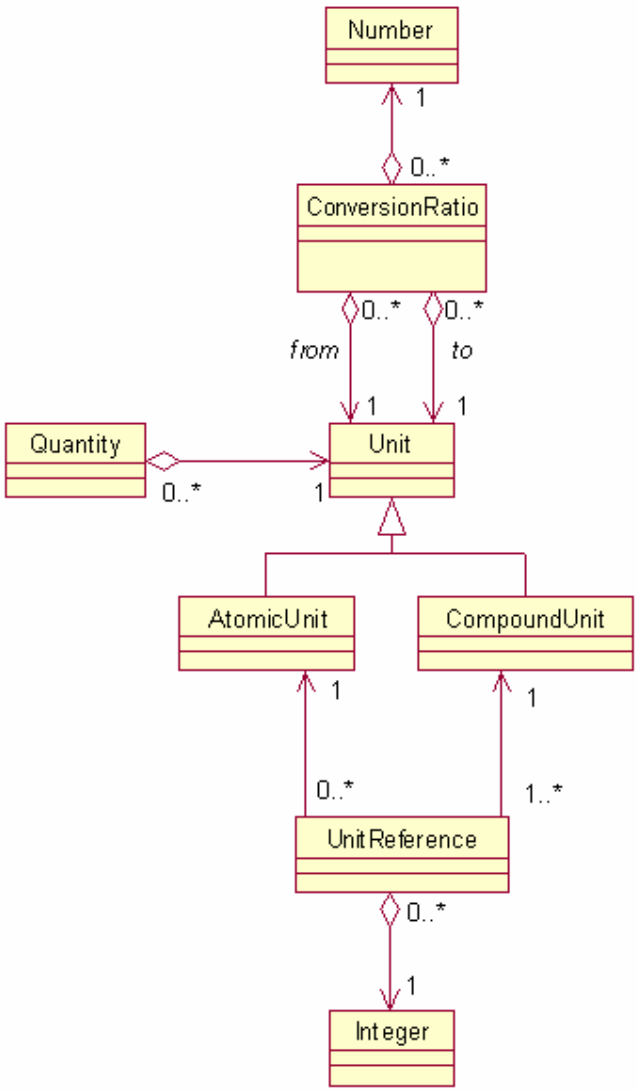

Figura 9. Solução de representação de quantidades utilizando padrões de análise.

4.2.2 Solução: Archetype Patterns

No catálogo de *archetype patterns*, apresentado por Arlow e Neustadt [7], o *quantity archetype pattern* é apresentado como uma extensão do padrão de análise *quantity* descrito por Martin Fowler em [6]. O modelo apresentado na Figura 10 apresenta a descrição da solução indicada pelo catálogo para a solução do problema de representar quantidades.

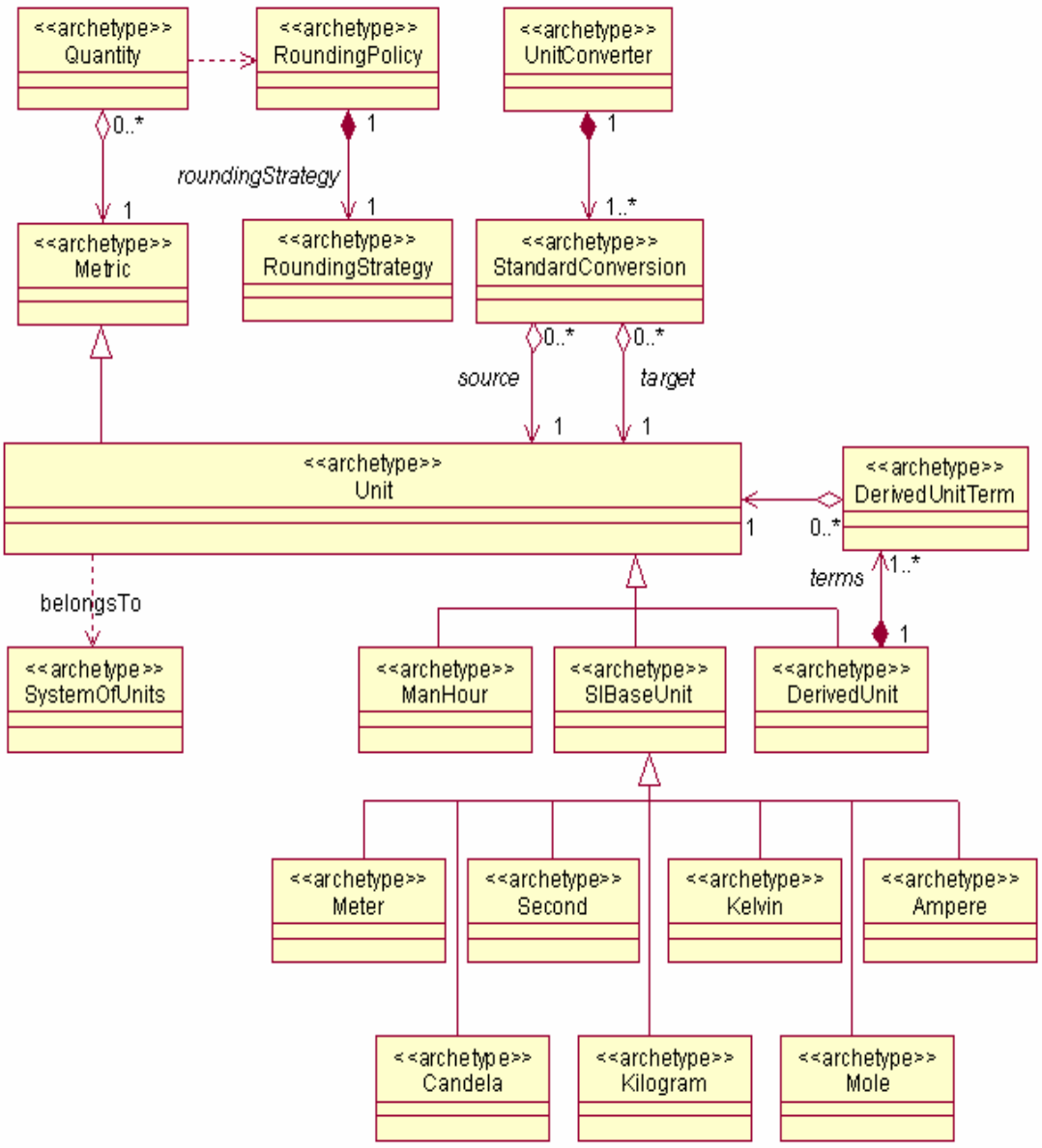

Figura 10. Modelo que representa o *quantity archetype pattern*.

No modelo podemos identificar alguns *archetypes* novos que não aparecem de maneira semelhante no modelo de padrões de análise apresentado anteriormente. O *archetype Metric*, por exemplo, representa padrões de medida, sendo uma unidade um tipo de métrica, como podemos ver através do *archetype Unit*. O *archetype SystemOfUnits* representa um conjunto de unidades definidos, como o conjunto definido pelo Sistema Internacional de Unidades (SI) [18]. Os *archetypes RoundingPolicy* e *RoundingStrategy* também merecem atenção. Esses *archetypes* definem políticas e estratégias de arredondamento de valores utilizados durante a manipulação desses valores. Os *archetypes* que herdam de *Unit* representam alguns subtipos, entre eles as unidades básicas definidas no conjunto padrão de unidades do SI.

No entanto, como vimos anteriormente, uma das principais características dos *archetype patterns* é a sua capacidade de adaptação através de variação. Através dessa característica podemos utilizar apenas o que desejarmos do *archetype pattern* definido para aplicarmos ao problema em que estamos tentando resolver. Através da variação podemos selecionar os *archetypes* que achamos relevantes para o nosso problema e eliminar o que acharmos desnecessário. É sempre bom lembrar que um padrão deve ser utilizado apenas como um guia, ou um ponto de partida, para ajudar a resolver determinados problemas.

Para visualizarmos melhor como o *quantity archetype pattern* pode apresentar uma descrição para um mesmo problema que um padrão de análise vamos adaptar o modelo proposto. Utilizando a característica de variação eliminamos alguns *archetypes* presentes no modelo de maneira a tornar mais próximo o modelo do *quantity archetype pattern* ao modelo do padrão de análise *quantity*, como podemos visualizar na Figura 11. Comparando o modelo da Figura 11 com o modelo que foi apresentado na Figura 9 podemos identificar similaridades. O relacionamento entre os *archetypes Unit*, *StandardConversion* e *UnitConverter*, que representa o como seria tratada a questão de conversão de unidades, é bastante similar ao relacionamento entre as classes de análise *Unit*, *ConversionRatio* e *Number* no modelo de padrões de análise.

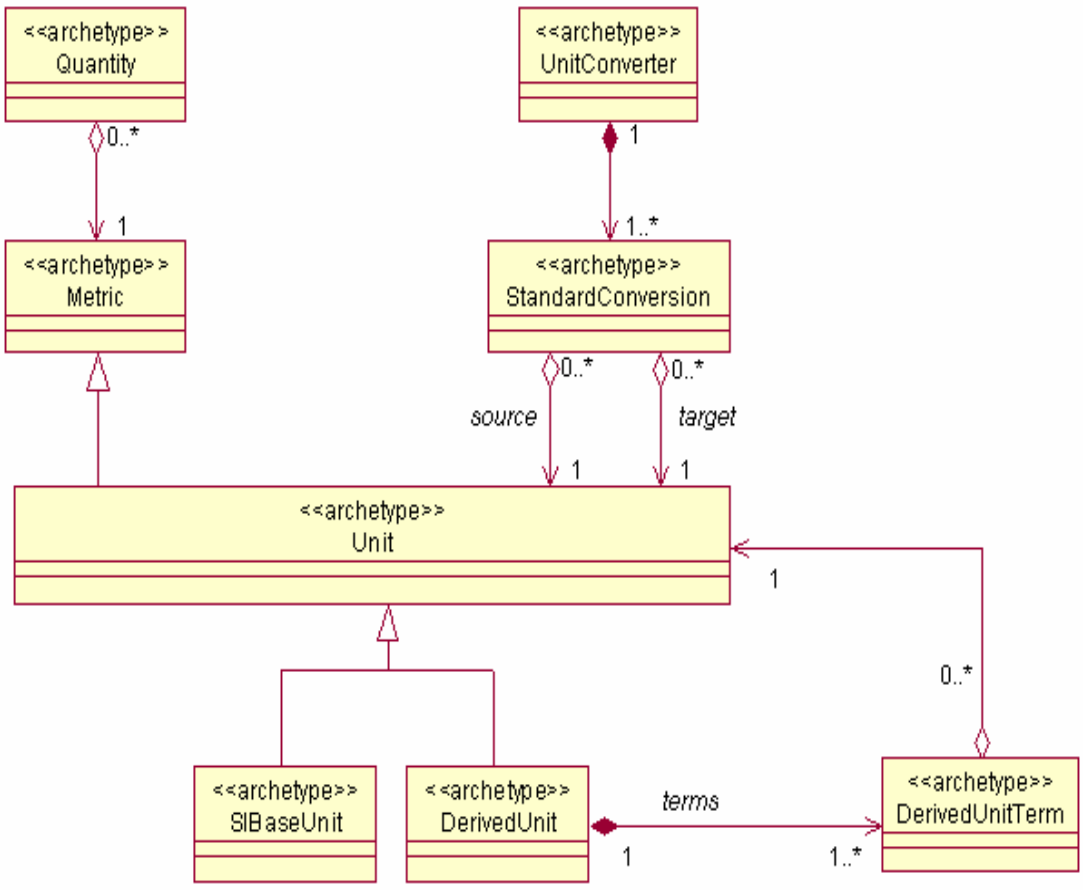

Figura 11. Modelo adaptado do *quantity archetype pattern* para se assemelhar ao modelo do padrão de análise *quantity*.

4.2.3 Solução: Padrões de Projeto

Apesar de descrever padrões em um outro nível de abstração em relação aos padrões de análise e *archetype patterns* os padrões de projeto podem, e se possível devem, também ser utilizados para ajudar a resolver os problemas. Como já apresentamos no Capítulo 3, os padrões de projeto descrevem soluções tanto a nível de classes quanto a nível de objetos, descrevem soluções em relação à criação de objetos, estruturação e comportamento dessas classes e objetos.

O modelo do *quantity archetype pattern*, em sua estrutura, já utiliza alguns padrões de projeto. Se considerarmos os *archetypes* definidos no modelo da Figura 10 como classes, como apresentado na Figura 12, podemos identificar que alguns padrões de projeto foram, ou poderiam ter sido, utilizados para a criação do modelo.

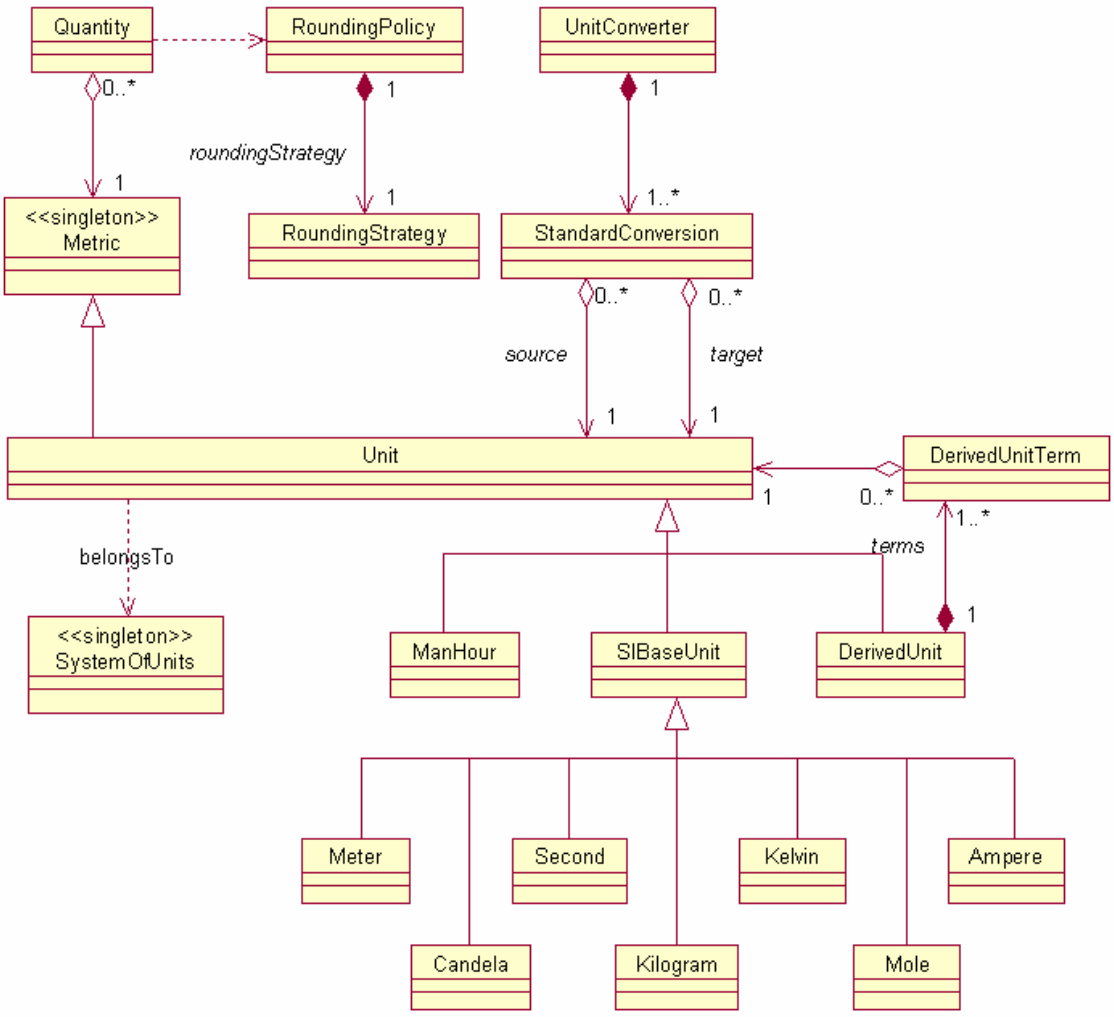

Figura 12. Modelo do *quantity archetype pattern* representado por classes no lugar dos *archetypes*.

O primeiro padrão de projeto que poderia ser identificado seria o padrão *Singleton*. Esse padrão de projeto faz com que para a classe que o implementa exista apenas uma única instância, de maneira que todos os objetos acessem esta mesma instância da classe de forma global. A

classe *SystemOfUnits* representa um conjunto de unidades definidas por um determinado padrão, como o SI. Portanto, esse conjunto de unidades é único e deve ser assim representado no sistema, fazendo com que a classe *SystemOfUnits* seja implementada com o padrão de projeto *Singleton*. A classe *Metric* também pode ser implementada com esse padrão de projeto, pois representa um padrão de medida único.

Um outro padrão de projeto que podemos identificar no modelo é o padrão *Strategy*. Uma das situações em que esse padrão de projeto pode ser utilizado é quando uma classe define muitos comportamentos e esses comportamentos apresentam muitas condições. Através desse padrão de projeto esses comportamentos podem ser agrupados em classes que serão responsáveis pela implementação desses comportamentos. A estrutura de como esse padrão é implementado, conforme definido em [5], pode ser visto na Figura 13. A classe *RoundingPolicy* determina como deve se comportar a operação de arredondamento de valores da classe quantidade. A classe *RoundingStrategy* representa um aspecto da classe *RoundingPolicy* que determina o tipo de arredondamento que deve ser utilizado. Utilizando o padrão de projeto *Strategy* poderíamos representar as estratégias de arredondamento como na Figura 14.

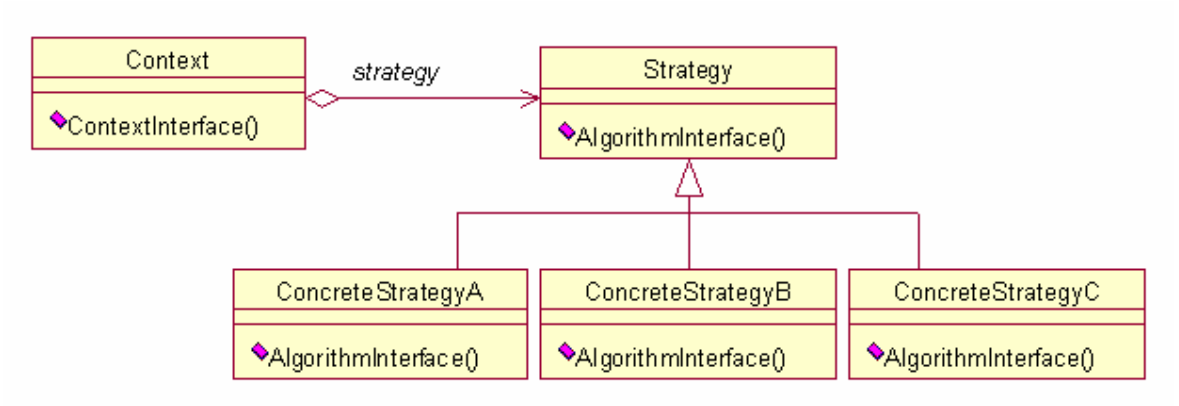

Figura 13. Estrutura que representa o padrão de projeto *Strategy*.

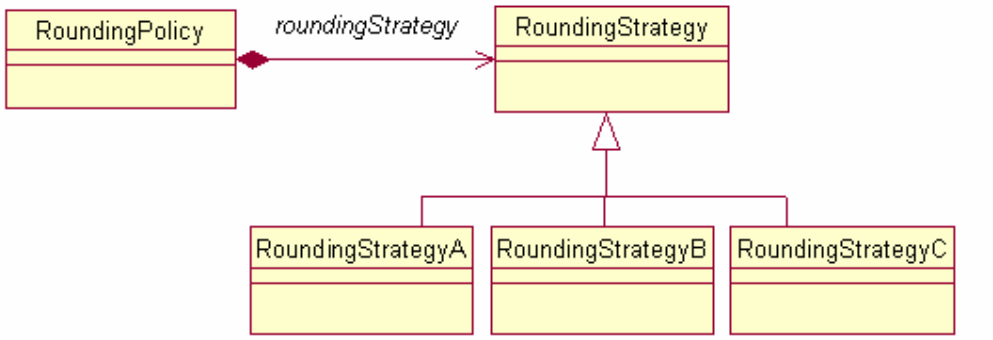

Figura 14. Representação de uma possível aplicação do padrão de projeto *Strategy* no modelo que descreve a solução do problema.

Podemos também destacar o padrão de projeto *Abstract Factory*. Esse padrão permite que sejam criadas famílias de objetos relacionados revelando apenas a interface desses objetos e sem revelar suas implementações. Esse padrão pode ser utilizado para fornecer um conjunto de unidades revelando apenas uma interface comum e deixando a implementação concreta de cada

unidade como responsabilidade da classe concreta de cada unidade. A estrutura desse padrão é definida em [5] como apresentado na Figura 15 e poderia ser aplicado no modelo para implementar a questão das unidades como na Figura 16.

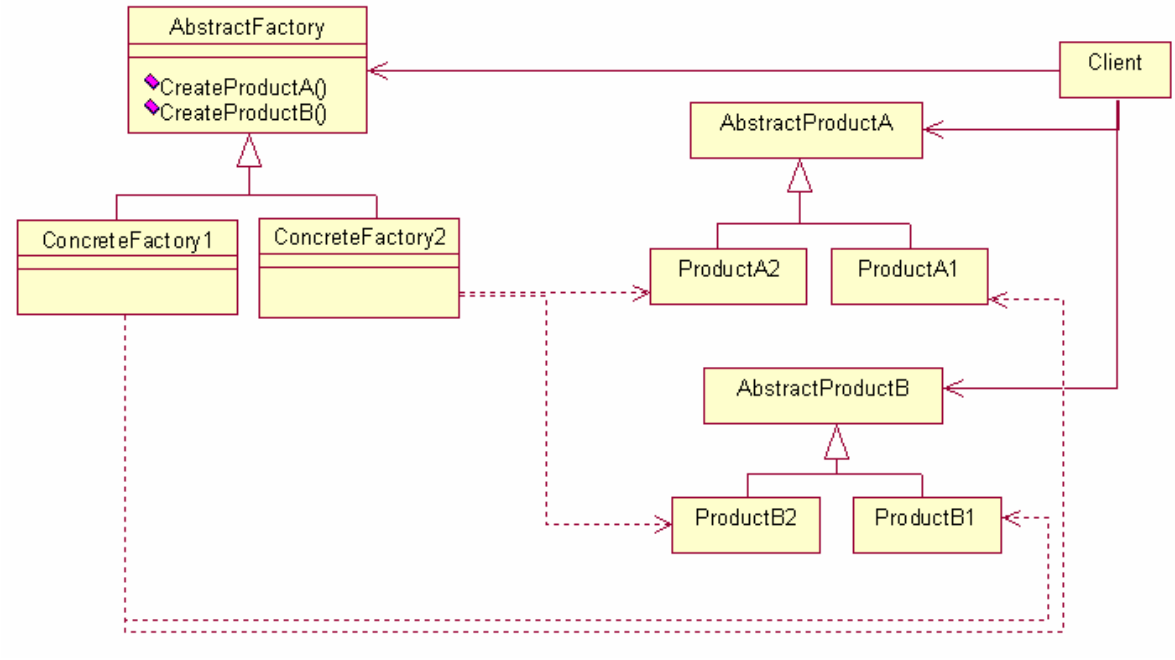

Figura 15. Estrutura do padrão de projeto *Abstract Factory*.

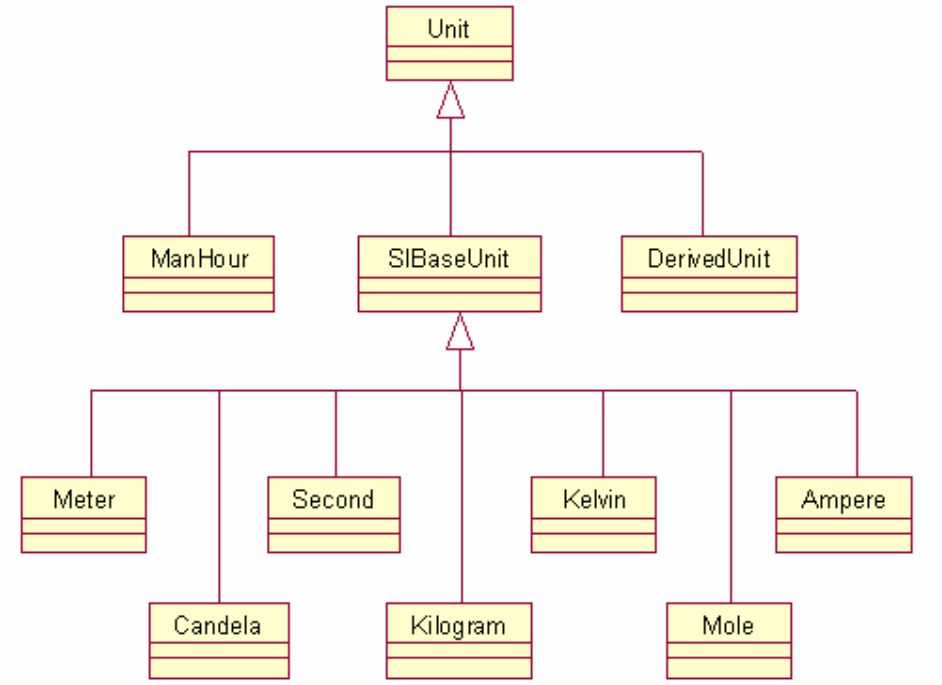

Figura 16. Representação do conjunto de unidades, que poderia utilizar o padrão *Abstract Factory*.

Por fim, um outro padrão de projeto que podemos identificar no modelo seria o padrão *Composite*. O padrão de projeto *Composite* permite que objetos individuais e composições de objetos sejam tratados de uma forma uniforme. A forma como esse padrão de projeto pode ser implementado é apresentado na Figura 17 (retirada de [5]). Em nosso modelo, a classe *DerivedUnit* representa uma combinação entre uma ou mais unidades e a classe *DerivedUnitTerm* representa uma parte de *DerivedUnit* composta de uma unidade e uma potência. Para que o sistema manipule unidades de maneira uniforme ele deve ser capaz de manipular tanto unidades simples como unidades compostas, considerando-as apenas como unidades em si. A aplicação do padrão *composite* no relacionamento dessas classes, apresentado na Figura 18, permite que as classes sejam manipuladas dessa forma.

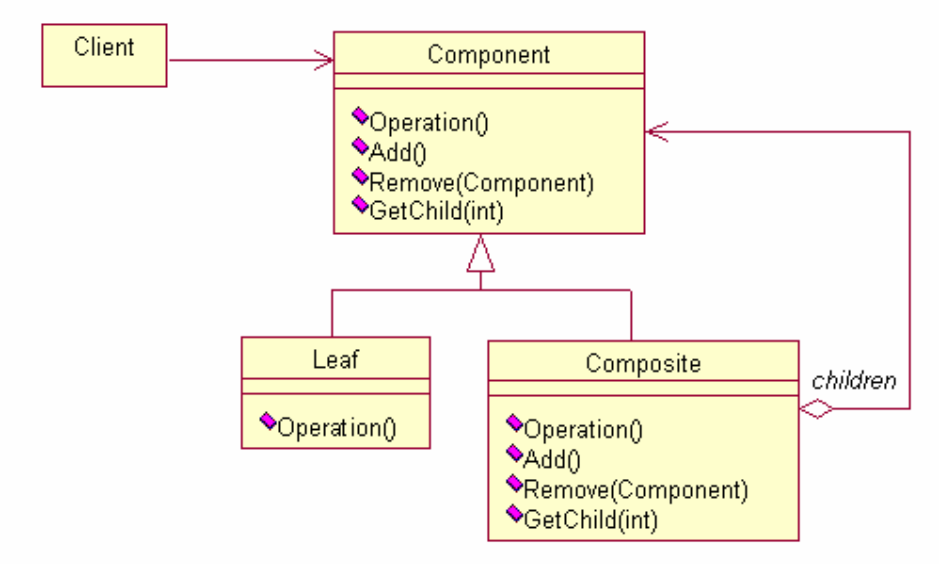

Figura 17. Estrutura do padrão de projeto *Composite*.

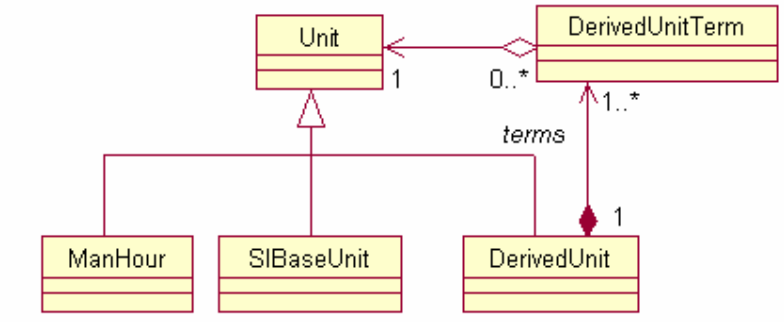

Figura 18. Estrutura que define o relacionamento para descrever unidades compostas no modelo.

4.3 Problema de representar pessoas e organizações

Um outro problema que os padrões de análise, *archetype patterns* e padrões de projeto podem ajudar a resolver em conjunto é o problema de representar e armazenar informações sobre pessoas e organizações em sistemas.

Em diversos tipos de sistemas, em diversos tipos de negócios, armazenar informações sobre clientes, fornecedores ou parceiros é essencial. Mesmo em uma simples agenda pessoal temos armazenadas informações como telefones, endereços ou e-mails de diversas pessoas, ou mesmo de empresas. A forma como essas informações são armazenadas é importante para um sistema.

A maneira como essas informações são armazenadas podem impactar em problemas de redundância de informações, qualidade de informações e até questões estratégicas como qualidade de serviço ou questões legais por não obter acesso a informações necessárias. O padrão *Party*, tanto nos padrões de análise quando nos *archetype patterns*, busca descrever uma boa maneira de representar informações essenciais de pessoas e organizações de uma maneira generalizada, tratando pessoas e organizações como 'partes'.

Os modelos apresentados, tanto em padrões de análise quanto em *archetype patterns*, abordam a questão na representação dos seus modelos questões no nível de negócios, como contabilidade envolvendo organizações e pessoas e responsabilidades entre partes. No entanto, para simplificar o exemplo apresentado, iremos focar apenas na questão da representação de organizações e pessoas de uma maneira geral.

4.3.1 Solução: Padrões de Análise

Para representar organizações e pessoas de uma forma geral e associar as informações de uma maneira centralizada o catálogo de padrões de análise fornece o padrão *party*, como mostrado na Figura 19. Neste padrão, pessoas e organizações são considerados como partes, de maneira que as informações estariam associadas às partes e não diretamente a uma pessoa ou organização.

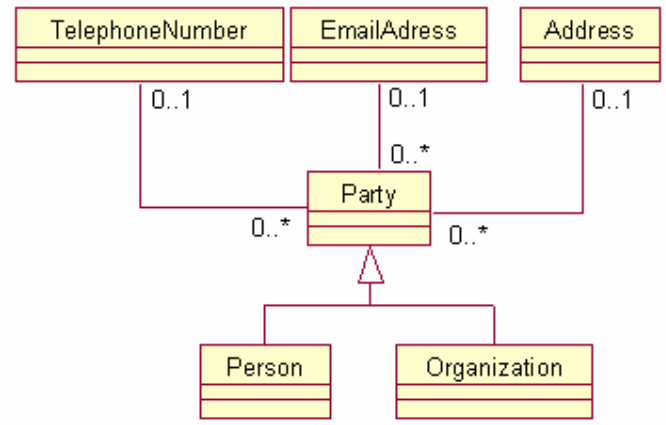

Figura 19. Representação do padrão de análise *party*.

No entanto, em uma grande organização podemos ter diversos setores ou divisões independentes e que possuem informações específicas associadas que precisam ser armazenadas também. Para esta situação o catálogo dispõe do padrão *Organization Hierarchies* que trata da criação de hierarquias dentro das organizações, como mostrado na Figura 20.

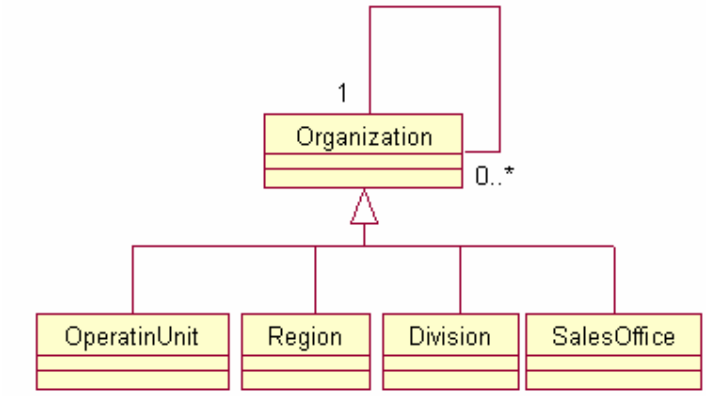

Figura 20. Representação do padrão de análise *organization hierarchies*.

Assim como fizemos com os padrões no caso da representação de quantidades, podemos combinar esses dois padrões e obter uma forma de representação geral de pessoas e organizações. O modelo apresentado na Figura 21 seria uma boa forma de representar e armazenar informações sobre pessoas e organizações considerando inclusive as subdivisões das organizações.

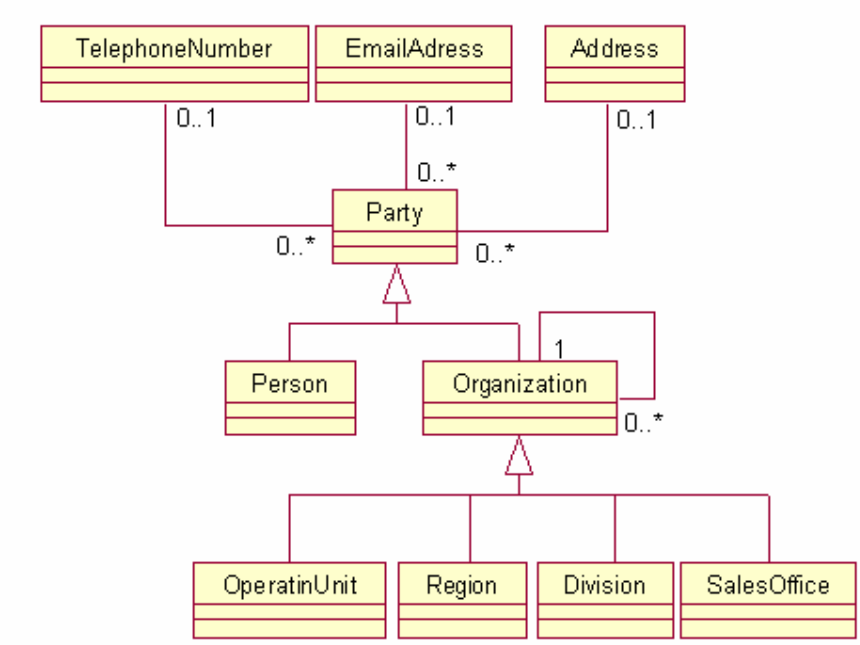

Figura 21. Solução de representação de pessoas e organizações utilizando padrões de análise.

4.3.2 Solução: Archetype Patterns

O catálogo de *archetype patterns* também possui entre os seus padrões o padrão *party* para descrever o problema de representar pessoas e organizações de uma forma geral e armazenar as informações das partes. O padrão apresentado no catálogo apresenta uma descrição bastante abrangente para problemas envolvendo partes, inclusive representando um relacionamento com um outro *archetype pattern* para simplificar o modelo, como podemos ver na Figura 22.

No entanto, utilizaremos da mesma técnica que utilizamos para o padrão *quantity* para representar o nosso problema. Através da aplicação da característica de variação dos *archetypes patterns* podemos obter um modelo semelhante ao apresentado na Figura 23.

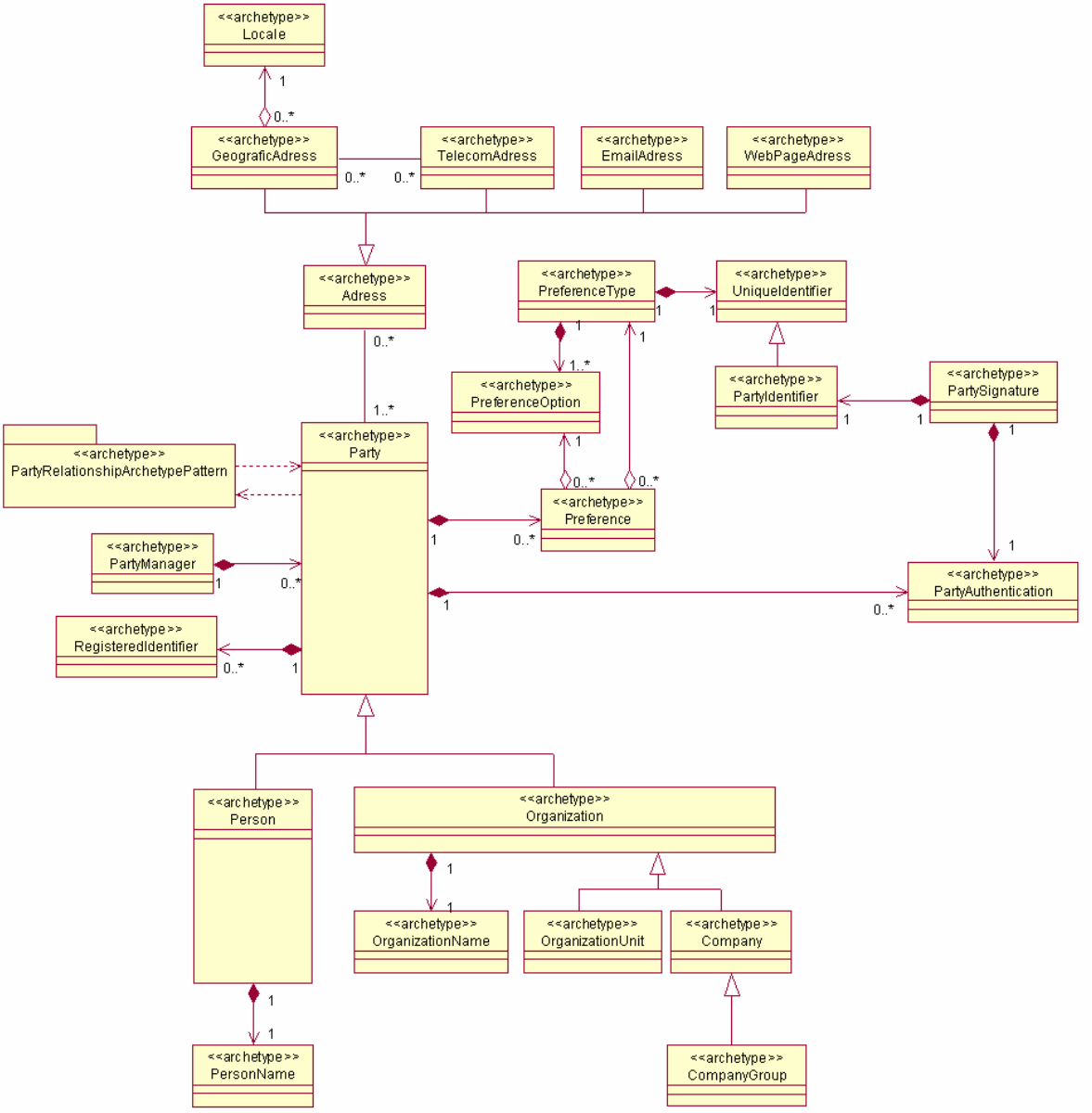

Figura 22. Representação do *party archetype pattern*.

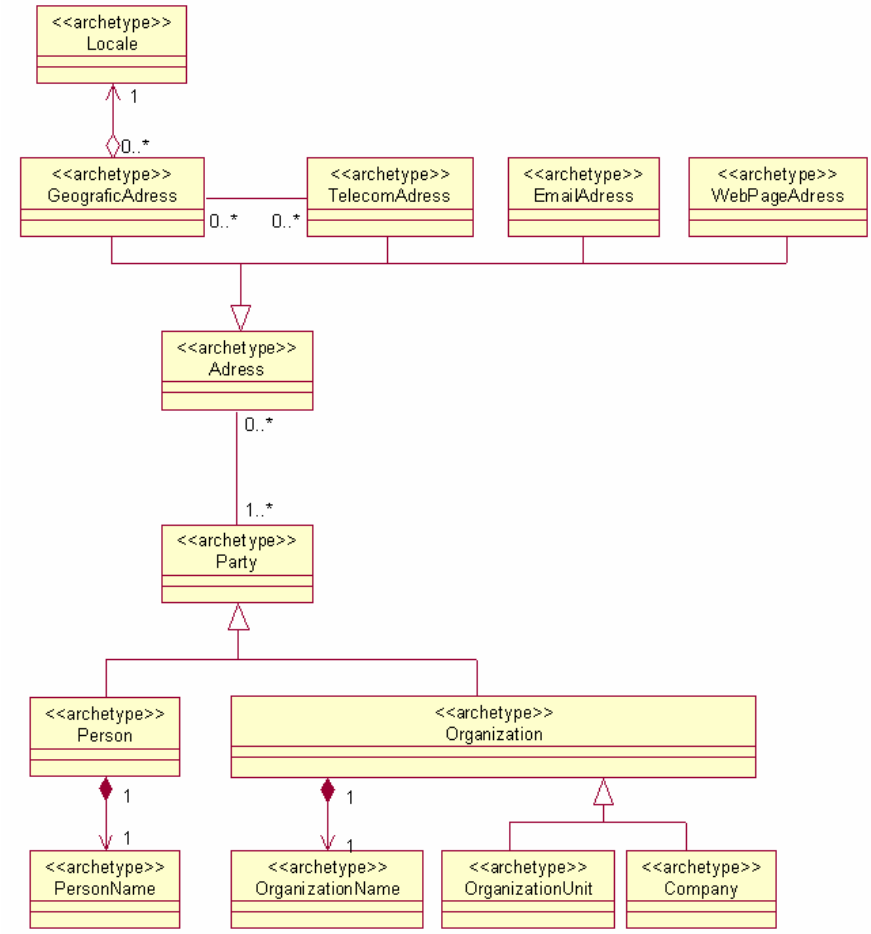

Figura 23. Adaptação do modelo do *party achetype pattern* para se assemelhar ao modelo definido através de padrões de análise.

4.3.3 Solução: Padrões de Projeto

Para identificar alguns padrões de projeto que poderiam ser utilizados na solução do padrão *party* vamos utilizar da mesma estratégia que utilizamos com o padrão *quantity*. Para a criar o relacionamento entre pessoas, organizações, e as subdivisões de uma organização revelando apenas a interface comum desses objetos poderíamos utilizar o padrão *Abstract Factory* (veja Figura 15), como utilizado para o problema da representação das unidades no padrão *quantity* .

Poderíamos também utilizar o padrão de projeto *Composite*, mostrado na Figura 17, para manipular as organizações simples e organizações compostas por diversos setores (veja modelo da Figura 20) de uma maneira uniforme. Para o padrão *quantity* utilizamos a mesma estratégia para solucionar o problema de manipular unidades simples e unidades derivadas.

Para implementar uma agenda com as informações armazenadas, por exemplo, seria necessário uma forma de percorrer as informações, como em uma lista. No entanto, a lista seria composta tanto de informações de pessoas quanto de organizações. O padrão de projeto *Iterator* poderia ajudar a resolver esse problema. Esse padrão possibilita uma maneira de percorrer uma lista composta de objetos de estruturas diferentes de uma maneira uniforme. A estrutura desse padrão é apresentada como na Figura 24 em [5].

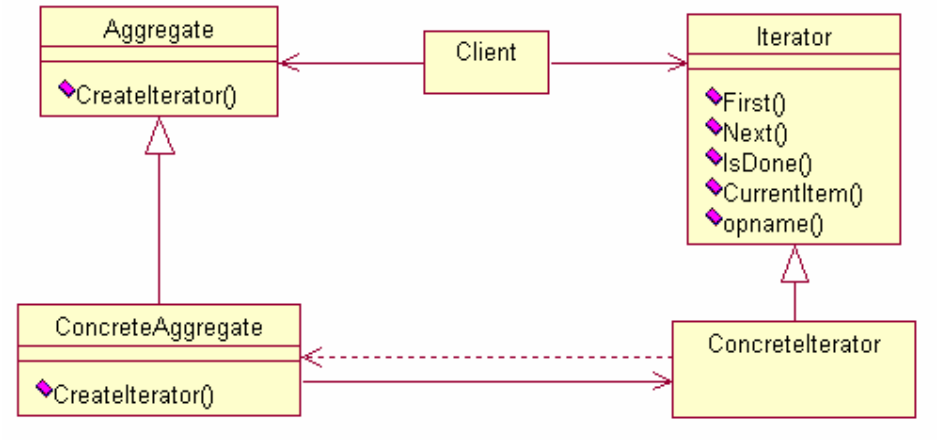

Figura 24. Estrutura do padrão de projeto *Iterator*.

4.4 Resumo

Neste capítulo observamos que nos catálogos dos diversos tipos de padrões podemos encontrar padrões que descrevem o mesmo problema e que essas descrições podem apresentar diferentes visões para o mesmo problema. Verificamos inclusive que no catálogo dos padrões de análise e de *archetype patterns* encontramos padrões que compartilham do mesmo nome, pois apresentam soluções para o mesmo problema.

No entanto, existem padrões que apesar de possuírem o mesmo nome não descrevem uma solução da mesma maneira, como é o caso do padrão *Product* (veja Anexo A). Na descrição do *Product Archetype Pattern* o produto é apresentado como elemento envolvido em compras e vendas, enquanto que no catálogo de padrões de análise o *Product Pattern* o produto é apresentado em termos de contratos e itens dos contratos em um aspecto mais financeiro.

Estudamos o problema de representação de quantidades em sistemas, pois o padrão *quantity* que descreve esse problema é um dos padrões que existem tanto no catálogo de padrões de análise quanto no de *archetype patterns*. Utilizamos o padrão *quantity* para mostrar que os três diferentes tipos de padrões estudados no Capítulo 3 podem ser utilizados para ajudar a encontrar uma solução para esse problema.

Apresentamos primeiramente três padrões de análise que foram descritos para ajudar a representar o problema da quantidade. Mostramos que esses três padrões podem ser combinados para descrever uma solução mais completa para o problema de representação de quantidades.

Mostramos como o mesmo problema foi tratado através do *quantity archetype pattern*. Verificamos que o modelo descrito para esse padrão foi desenvolvido a partir de uma extensão do padrão de análise *quantity* e que é possível obter um modelo bastante semelhante a esse modelo utilizando-se da capacidade de adaptação dos *archetype patterns* através da característica de variação.

Mostramos também que mesmo não possuindo um nível de abstração de negócio os padrões de projeto podem ser utilizados na composição da solução para o problema. Mostramos que podem ser identificados e utilizados alguns padrões de projeto no próprio modelo para o problema da representação de quantidades.

Por fim apresentamos um outro exemplo em que os três tipos de padrões podem auxiliar a resolver o problema apresentado. Através do problema da representação de pessoas e organizações mostramos mais uma vez que os três tipos de padrões apresentados podem ser utilizados em conjunto para resolver problemas. Os *archetype patterns* e padrões de análise

podem descrever o problema a nível de negócio, enquanto que os padrões de projeto podem auxiliar na criação de uma boa estrutura a nível de projeto para o modelo.

Através desses exemplos podemos destacar algumas estratégias e diretrizes utilizadas para realizar a transição entre os padrões. Por descreverem soluções no nível de negócio a transição entre padrões de análise e *archetype patterns* ocorre de uma maneira mais natural em relação à transição envolvendo padrões de projeto. Na descrição de um mesmo problema no nível de negócio os padrões de análise e *archetype patterns* naturalmente apresentam alguns elementos em comum, pois são elementos específicos do problema que estão modelando. Esses elementos acabam tornando-se elementos essenciais também na transição de um modelo descrito utilizando um tipo de padrão para o outro. Neste capítulo a estratégia adotada nos exemplos foi de utilizar a característica de variação dos *archetype patterns* para destacar os elementos principais presentes nos modelos de análise.

No entanto, a transição envolvendo padrões de projeto envolve a observação de algumas diretrizes além da identificação de certos elementos. No catálogo de padrões de projeto em Gamma et al. [5] existem algumas diretrizes que indicam situações específicas em que cada padrão poderia ser utilizado para solucionar um problema.

Para realizar a transição para os padrões de projeto a estratégia utilizada foi de identificar nos modelos de negócios elementos que satisfizessem as diretrizes dos padrões do catálogo. Elementos que devem ser únicos em todo o sistema (*Singleton*), representação de classes que podem apresentar muitos comportamentos e com muitas condições (*Strategy*), criação de família de objetos relacionados (*Abstract Factory*), representação uniforme de objetos independentes ou compostos (*Composite*) e manipulação de listas compostas por objetos de estruturas diferentes (*Iterator*) foram algumas das diretrizes observadas para a determinação dos padrões de projeto a serem utilizados nos modelos.

O objetivo deste capítulo era de apresentar como esses três tipos de padrões podem ser utilizados em conjunto para resolver problemas. Desta forma a seqüência em que os padrões foram utilizados não deve ser considerada, pois o que realmente determina a seqüência em que os padrões poderão ser utilizados são os modelos que serão gerados para resolver o problema e a seqüência que esses modelos serão necessários. No entanto, baseando-se no nível de abstração que cada tipo de padrão utiliza, uma boa seqüência para utilizar esses padrões seria como apresentado na Figura 25.

A medida que detalhamos o que definimos durante o processo de desenvolvimento de um sistema novos elementos são acrescentados. Uma classe definida em um modelo no nível de negócio poderá ser representada por uma ou mais classes no nível de análise, podendo ocorrer o mesmo em relação às classes de análise representadas no nível de projeto.

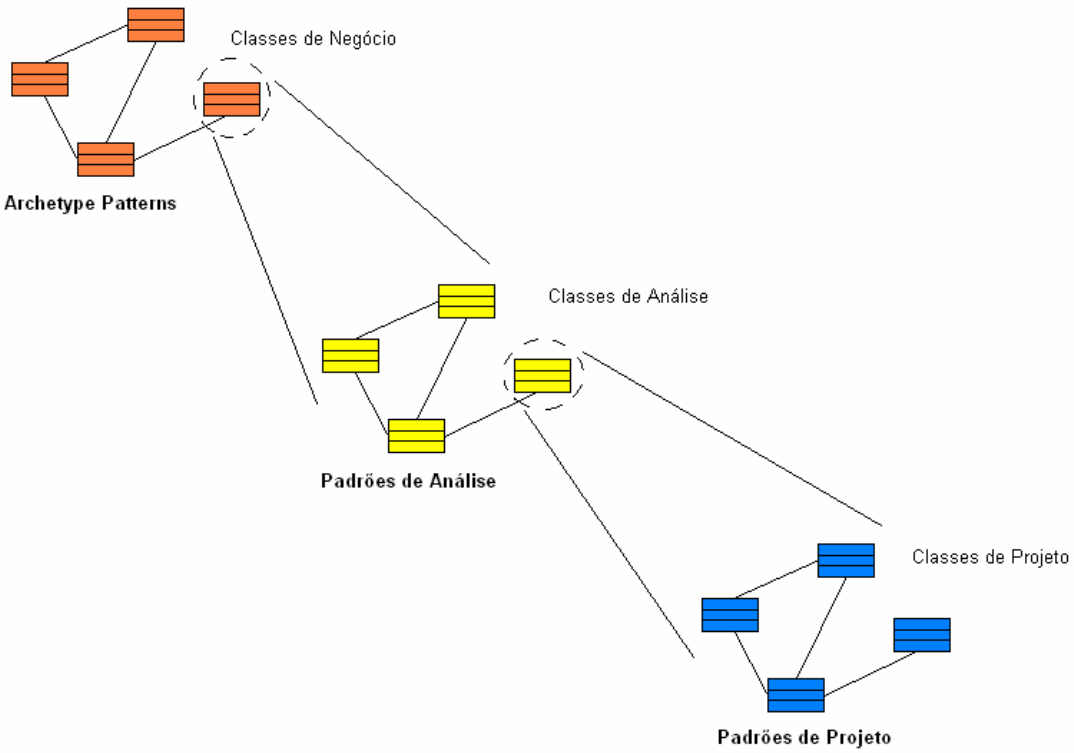

Figura 25. Seqüência de utilização dos três tipos de padrões de acordo com o nível de abstração. Classes de Negócio → Classes de Análise → Classes de Projeto

Capítulo 5

Estudo de Caso

Nos capítulos anteriores abordamos o RUP e três tipos de padrões: padrões de projeto, padrões de análise e *archetype patterns*. Aprendemos que um dos objetivos do RUP é a busca contínua por qualidade de software e que a utilização de padrões ajuda no desenvolvimento de software com qualidade. No entanto, não vimos ainda como o RUP e os padrões podem ser utilizados em conjunto.

Neste capítulo vamos apresentar como o RUP e os padrões podem ser utilizados juntos. Baseado em um estudo de caso desenvolvido por Craig Larman [19] vamos apresentar onde os padrões poderiam ser utilizados em um processo de desenvolvimento de software como o RUP.

No desenvolvimento do seu estudo de caso Larman utilizou o processo de desenvolvimento de software do RUP para apresentar uma introdução à análise e projeto de software orientado a objetos e utilizou alguns padrões de projeto. Nos basearemos neste estudo de caso, pois desta forma podemos manter o foco do nosso trabalho de apenas apresentar onde os padrões poderiam ser utilizados dentro do processo, abstraindo o processo completo definido pelo RUP.

Ao apresentar onde os tipos de padrões que apresentamos podem ser utilizados no RUP iremos realizar o mapeamento da utilização desses padrões às fases do processo. Desta forma podemos identificar quando cada tipo de padrão pode ser utilizado com mais intensidade.

5.1 Introdução

Como já vimos no Capítulo 2, a forma que o RUP é organizado divide o seu processo de desenvolvimento de software em quatro fases: iniciação, elaboração, construção e transição. No decorrer destas quatro fases são desempenhadas diversas disciplinas definidas no processo variando a ênfase dada a cada disciplina ao longo do tempo (ver Figura 4). No fluxo desenvolvido dessas disciplinas são realizadas diversas atividades e diversos artefatos são gerados, entre eles diversos modelos.

Em projetos de software orientados a objetos o desenvolvimento de alguns desses modelos poderiam ser auxiliados pela utilização de padrões, portanto é importante saber quando os padrões poderiam ser úteis no processo de desenvolvimento. Saber antecipadamente em que momento um determinado tipo de padrão pode ajudar a buscar possíveis soluções nos catálogos de padrões desde o começo do desenvolvimento dos modelos.

Para mostrar situações em que um determinado tipo de padrão poderia ser utilizado vamos utilizar como base o estudo de caso do ponto de venda desenvolvido em [19]. Durante esse estudo de caso Larman mostrou a evolução do desenvolvimento do sistema utilizando o RUP e dando ênfase às atividades de análise e projeto orientados a objetos presentes no RUP.

Durante o processo do RUP diversas atividades devem ser realizadas e artefatos gerados além da criação de modelos de análise e de projeto orientados a objetos. Como a realização dessas atividades não é objetivo deste trabalho aproveitamos um caso já realizado para nos concentrar apenas na aplicação dos padrões às fases do RUP.

5.1.1 O caso do Ponto de Venda

Sistemas de ponto de venda são relativamente fáceis de encontrar, sendo utilizado em diversos tipos de loja para controlar a compra e venda de produtos. São sistemas utilizados em caixas registradores mais modernos, como na Figura 26, e que auxiliam o controle de pagamentos, etc. O problema do ponto de venda foi descrito da seguinte forma em [19]:

"Um sistema de ponto de venda é uma aplicação de computador utilizada, em parte, para registrar vendas e controlar pagamentos; é tipicamente utilizado em lojas de varejo. O sistema normalmente inclui componentes como um computador, um leitor de código de barras e um software para executar o sistema. O sistema se comunica com diversas outras aplicações como sistemas de cálculo de impostos e de controle de estoque. Estes sistemas devem ser relativamente tolerante a falhas, isto é, mesmo que alguns serviços remotos, como o controle de estoque, fiquem indisponíveis temporariamente o sistema deve ser capaz de registrar vendas e controlar os pagamentos."

Figura 26. Ilustração de um sistema de ponto de venda.

5.2 Adaptação do RUP

Na utilização do RUP para o desenvolvimento de um sistema real uma das atividades iniciais do processo faz a definição do plano de projeto e do plano de iteração. Através da definição desses

dois documentos é definida a estratégia que será utilizada para o desenvolvimento do sistema. É definido como serão organizadas as fases, quantas iterações serão realizadas em cada fase e os marcos que definem o início e fim de cada fase, entre outras coisas. No entanto, para efeito deste trabalho iremos abstrair essas definições.

Para atender o foco principal deste capítulo, que é o mapeamento do uso de padrões ao RUP, nos concentraremos apenas nas fases e em algumas disciplinas e atividades dessas disciplinas. As disciplinas serão focadas em razão da relação entre os objetivos que a disciplina busca atender, os artefatos gerados e a possibilidade de utilização de algum dos três tipos de padrão para auxiliar na criação desses artefatos.

5.3 Aplicando os padrões às fases do RUP

As disciplinas que iremos focar na nossa análise serão as disciplinas de modelagem de negócios e análise e projeto. Como podemos observar no gráfico da Figura 4, a disciplina de modelagem de negócios tem um maior esforço concentrado logo no começo da fase de iniciação e um pouco da fase de elaboração. Já as disciplinas de análise e projeto têm maior concentração de esforço na fase de elaboração, mas também é bem presente nas fases de iniciação e construção.

A disciplina de modelagem de negócios tem como alguns de seus objetivos o entendimento do problema e do negócio alvo do sistema, assim como assegurar um entendimento comum do problema pelos interessados do projeto. O diagrama de estados apresentado na Figura 27 representa o fluxo de trabalho dessa disciplina. O fluxo principal para o nosso trabalho é o fluxo relacionado ao desenvolvimento de um modelo de domínio. Um dos artefatos gerados nas atividades deste fluxo é um modelo de objetos de negócio ou modelo de domínio [4].

Um modelo de domínio ilustra conceitos importantes em um domínio de negócio [19]. O modelo de domínio pode ser usado para ajudar a projetar os objetos do software e é utilizado como entrada em outras atividades do processo de desenvolvimento. Os padrões de análise e *archetype patterns* descrevem soluções para problemas no nível de negócio, como vimos no Capítulo 3. Uma das características desses dois tipos de padrões é que seus modelos podem ser utilizados como base para facilitar o entendimento do negócio em questão. Por essas razões, padrões de análise e *archetype patterns* poderiam ser utilizados para ajudar na descrição dos modelos de domínios.

Os *archetype patterns* poderiam ser utilizados para representar uma descrição mais geral, abordando questões essenciais do problema, independente do domínio de aplicação. No entanto, apesar da vantagem de apresentar uma solução mais genérica do problema, a utilização dos *archetype patterns* poderia ser substituída, ou complementada, pela utilização dos padrões de análise quando o objetivo fosse a descrição em um domínio de negócio específico.

O catálogo de padrões de análise atualmente está restrito a algumas áreas de domínio, como financeira e de contabilidade, de maneira que apresenta boas soluções específicas para problemas nesses domínios de negócio, situações em que a utilização de *archetype patterns* poderia se tornar redundante, e os padrões de análise poderiam substituir a utilização dos *archetype patterns*. Em outras situações os modelos de *archetype patterns* se apresentam mais completos e atualizados de maneira que poderiam ser complementados através do uso de padrões de análise para focar um determinado domínio de negócio descrito por esse tipo de padrões.

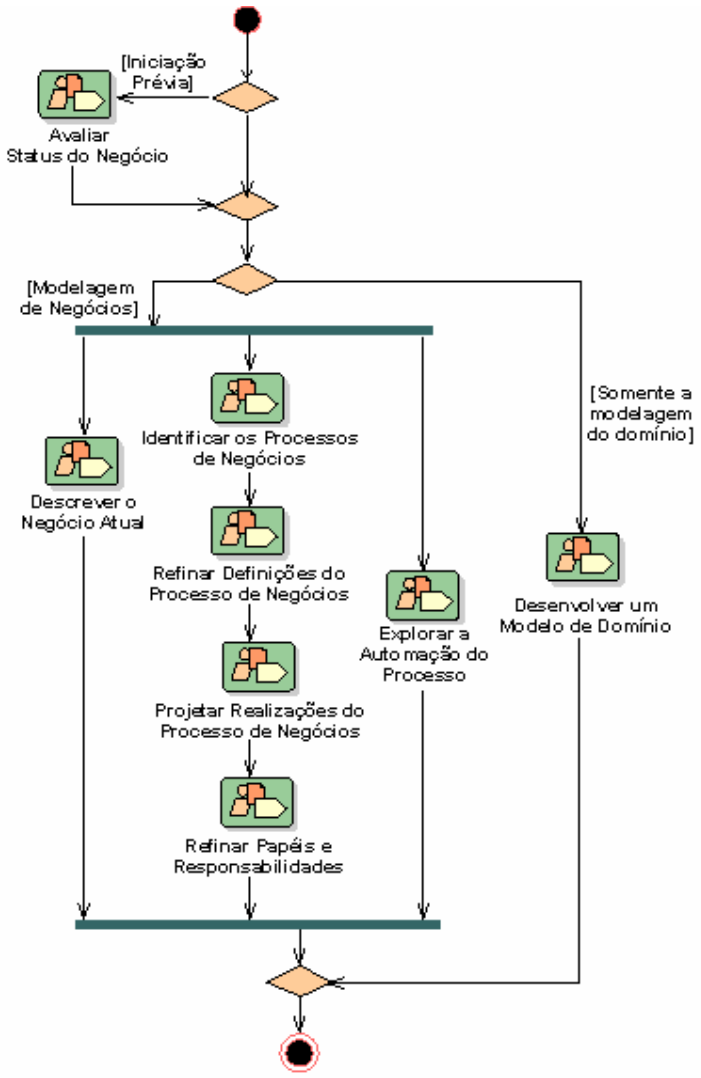

Figura 27. Fluxo de trabalho da disciplina de modelagem de negócios.

Os principais objetivos da disciplina de análise e projeto estão voltados para o desenvolvimento de uma arquitetura sofisticada para o sistema, transformação dos requisitos em um projeto do sistema a ser criado e adaptação desse projeto para atender requisitos não funcionais, como desempenho por exemplo. A disciplina de análise e projeto utiliza o modelos de negócios (modelo de domínio), gerado pela disciplina de modelagem de negócios, para gerar modelos de análise e projeto no desenvolvimento de suas atividades. [4] Os fluxos envolvidos nessa disciplina, que podem ser vistos no diagrama da Figura 28, utilizam com freqüência estes dois modelos na realização dos objetivos.

Um modelo de projeto é um modelo de objeto que descreve a realização de casos de uso. É um artefato composto e abrangente que envolve todas as classes de projeto, subsistemas, pacotes, colaborações e os relacionamentos entre eles. Um bom modelo deve satisfazer os requisitos do sistema, resistir a mudanças efetuadas no ambiente de implementação, ser de fácil manutenção, ter um procedimento de implementação claro e adaptar-se facilmente a mudanças de requisitos. [4]

Um modelo de análise funciona como uma abstração de um modelo de projeto. O modelo de análise pode ser um artefato temporário, como no caso em que evolui para um modelo de projeto, ou pode servir durante todo o projeto como uma visão geral conceitual do sistema. Na transformação para um modelo de projeto uma classe de análise pode vir a ser representada por mais de uma classe no modelo de projeto, pode fazer parte de uma classe, pode ser uma classe agregada, etc. Uma classe de análise trata principalmente de requisitos funcionais e questões relacionadas ao domínio do problema. [4]

A partir das definições de modelos de análise e projeto, e das definições dos três tipos de padrões que estudamos até agora, concluímos que a utilização dos três tipos de padrões em conjunto ajudariam na criação desses modelos. Os padrões de análise e *archetype patterns* poderiam servir de guia para o desenvolvimento dos modelos de análise, utilizados em conjunto ou separadamente de acordo com a situação, pois tratam de questões de negócio. Por outro lado os padrões de projeto poderiam ser bastante utilizados na criação dos modelos de projeto, pois poderiam ser aplicados para resolver problemas identificados nos modelos de negócios e de análise e que podem ser resolvidos através da utilização desses padrões.

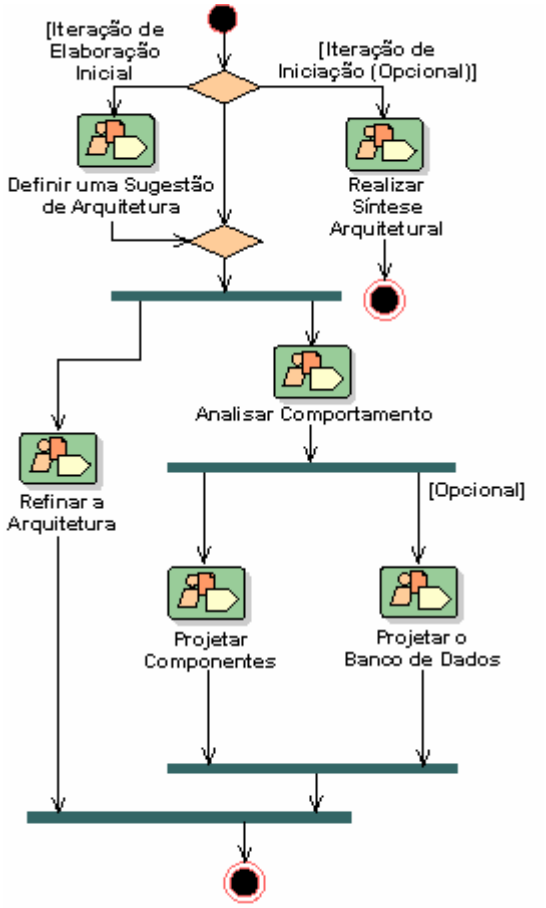

Figura 28. Fluxo de trabalho da disciplina de análise e projeto.

Mapeando a utilização dos três tipos de padrões que estudamos nessas duas disciplinas do RUP podemos identificar as fases do processo em que os padrões seriam mais utilizados. Analisando mais uma vez o gráfico da Figura 4 podemos dizer que os padrões seriam utilizados

com maior intensidade na fase de elaboração do processo. Nessa fase os três tipos de padrões poderiam ser utilizados para trabalhar nos modelos de domínio, análise e projeto. Na fase de iniciação, por estar o foco do processo voltado para o estabelecimento do consenso entre todos os envolvidos a respeito do projeto e definição de escopo, os padrões voltados para o domínio do negócio, como é o caso dos padrões de análise e *archetype patterns*, estariam mais evidentes. Em contrapartida, na fase de implementação, em que os requisitos já estariam praticamente analisados e sofrendo refinamentos, a utilização de padrões de projeto seria mais freqüente.

5.4 Utilização de padrões no ponto de venda

Como vimos na seção anterior as principais disciplinas em que os três tipos de padrões poderiam ser utilizados seriam as disciplinas de modelagem e negócios e análise e projeto. Nessas disciplinas três artefatos essenciais são: o modelo de negócios (modelo de domínio), o modelo de análise e o modelo de projeto.

No estudo de caso do ponto de venda, Larman definiu um exemplo de modelo de negócios parcial para o problema do ponto de venda (Figura 29) no início da fase de elaboração. No modelo definido uma das classes destacadas por Larman como principal foi a relativa ao pagamento (*CashPayment*). Utilizaremos essa classe para apresentar um ponto em que poderíamos utilizar padrões para nos ajudar a desenvolver uma boa solução.

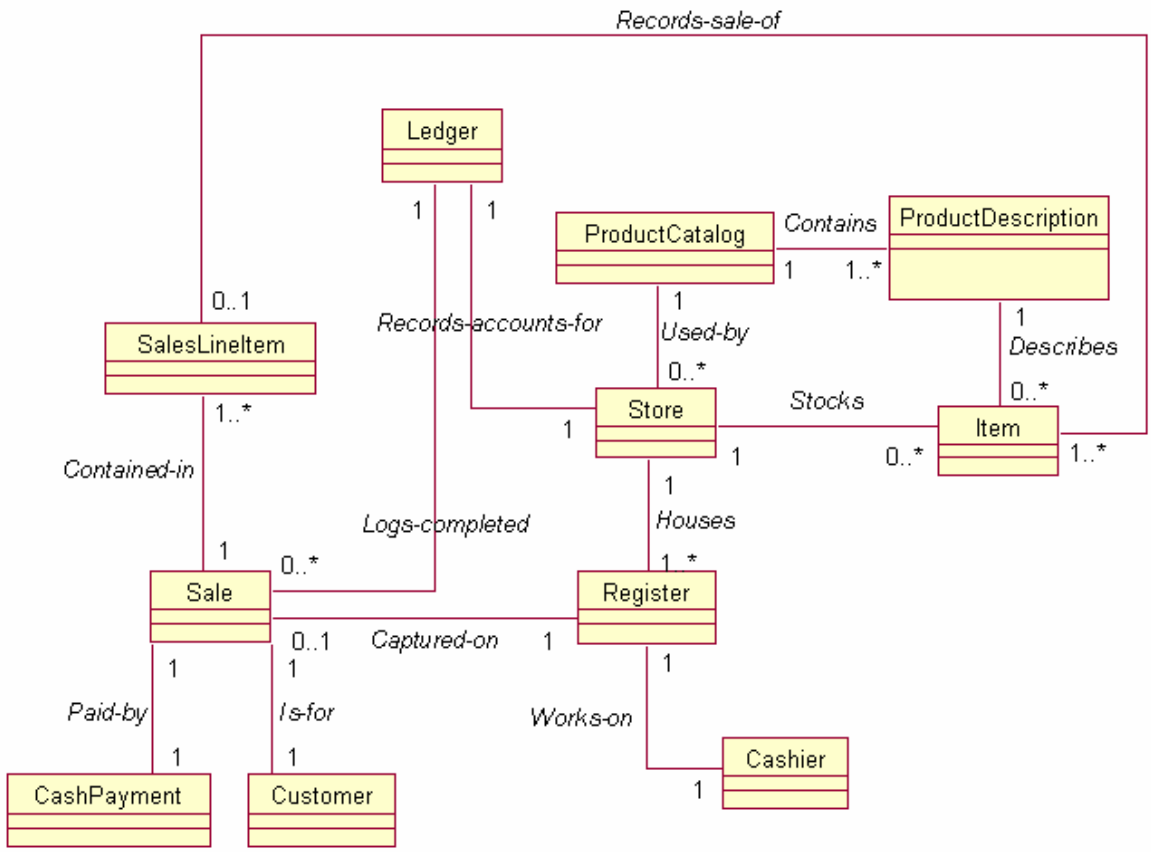

Figura 29. Modelo de negócios parcial do ponto de venda definido por Larman [19].

No seu modelo de negócios do ponto de venda, Larman definiu que os pagamentos das compras seriam realizados em dinheiro. No entanto, dependendo de como fosse definida a estrutura do sistema, um grande impacto poderia ser causado pela necessidade de dispor de novas formas de pagamento. Caso a loja que utilizasse o sistema de ponto de venda passasse a aceitar pagamentos em cheque ou cartões de crédito, o sistema deveria estar preparado para suportar essas variações. Do catálogo de *archetype patterns* [7] poderíamos utilizar o padrão *Money*, apresentado na Figura 30, para servir como ponto de partida para uma solução do problema. Uma das características que podemos observar no padrão *money* é o fato de que ele é também tratado através do padrão *quantity*. Dessa forma o modelo poderia utilizar as soluções que apresentamos no Capítulo 4, quando mostramos os padrões de análise e projeto que poderiam ser utilizados no problema de representação de quantidades.

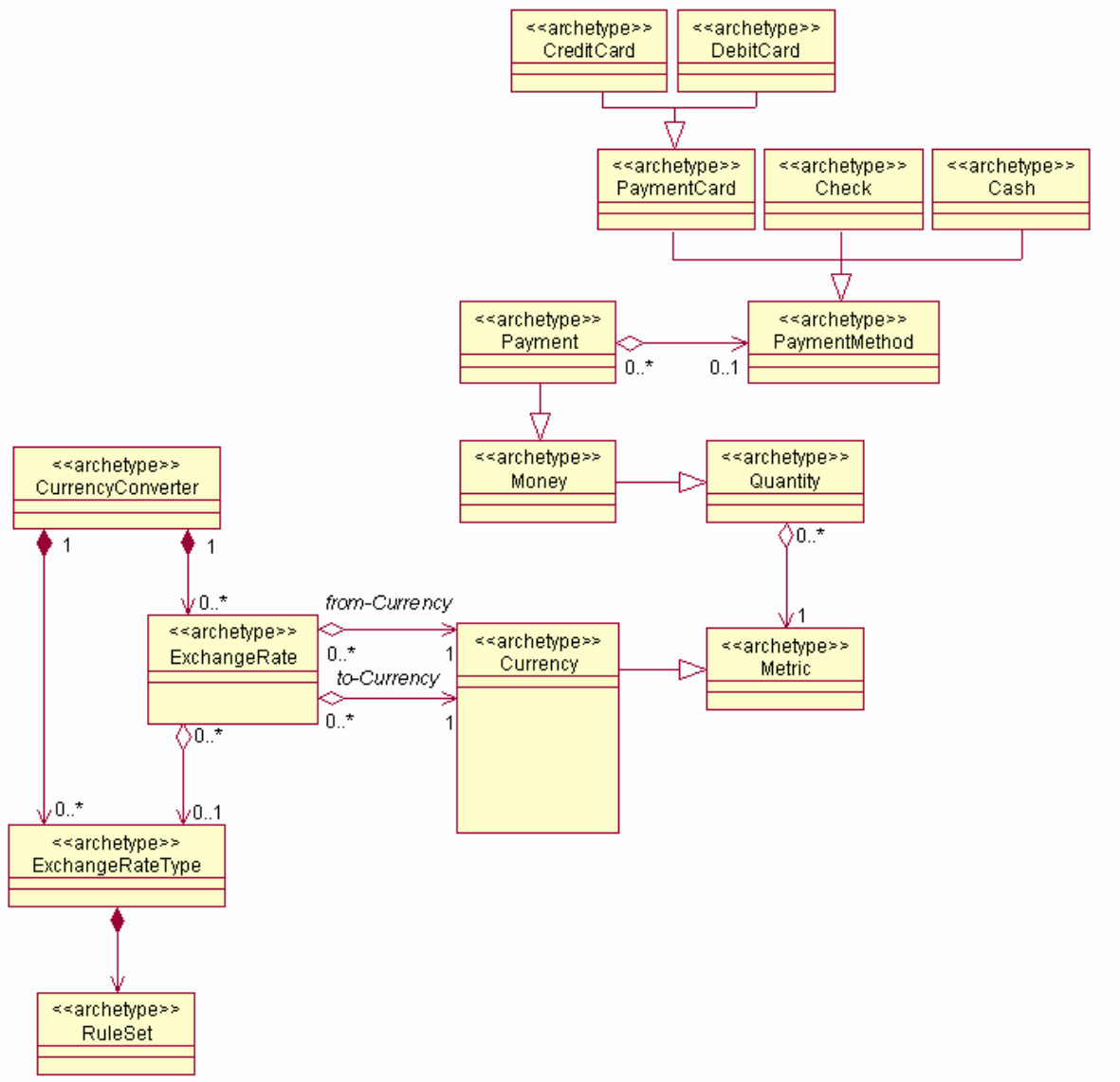

Figura 30. Modelo do *Money Archetype Pattern*.

Ainda partindo desse modelo podemos observar que padrões de projeto como *Strategy* e *Abstract Factory* poderiam ser utilizados para implementar as relações existentes nos tipos de pagamentos (Figura 31). O padrão *Strategy* poderia ajudar de maneira a abstrair os comportamentos diferentes existentes nos diferentes tipos de pagamentos, enquanto que o padrão *Abstract Factory* forneceria um meio uniforme de tratar os tipos de pagamentos.

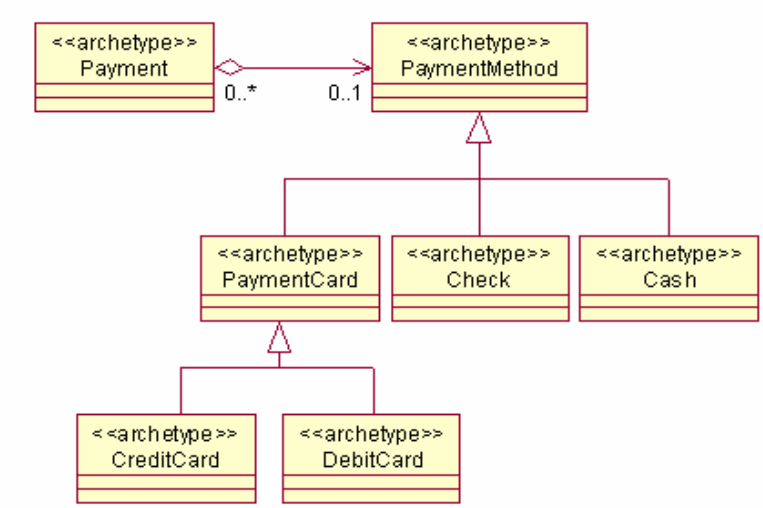

Figura 31. Relações envolvendo pagamentos no modelo do ponto de venda.

No desenvolvimento do modelo completo do ponto de venda diversos outros padrões poderiam ser utilizados. Padrões como: o *product archetype pattern*, para modelar os problemas envolvendo produtos, *inventory archetype pattern*, para implementar um estoque, padrões de análise como *inventory* e *accountability*, também para estoque e questões contábeis e diversos padrões de projeto, dependendo das situações, poderiam ser utilizados em conjunto para o desenvolvimento de uma boa solução.

5.5 Resumo

Neste capítulo estudamos como e quando os três tipos de padrões estudados no capítulo anterior podem ajudar durante um processo de desenvolvimento de software baseado no RUP. Vimos que durante o RUP diversos artefatos são gerados, inclusive modelos, e que os padrões podem ajudar na criação desses modelos.

Iniciamos apresentando o estudo de caso do ponto de venda desenvolvido por Craig Larman em [19]. Vimos que sistemas de ponto de ponto de venda são sistemas comuns de se encontrar e que envolvem diversas questões de negócio.

Apresentamos uma forma de visualizar o RUP, focando apenas nas fases e algumas disciplinas, para estudar apenas o mapeamento dos padrões às fases do RUP. Concentramos nosso estudo apenas nas disciplinas de modelagem de negócios e análise e projeto, pois são as fases que envolvem criação de modelos de negócios, modelos de classe e de projeto.

Apresentamos o que são os modelos de negócios, modelos de classe e de projeto, bem como quais são as disciplinas principais que envolvem a criação e manutenção desses modelos. Verificamos as características desses modelos e identificamos que tipo de padrões poderiam ser utilizados para ajudar na criação de boas soluções para os problemas encontrados.

Após a identificação e mapeamento dos padrões às disciplinas do RUP identificamos em que fases do processo os padrões são utilizados com maior freqüência. Identificamos que na fase de elaboração os padrões são utilizados em maior escala.

Por fim, apresentamos uma parte do modelo de negócios criado por Craig Larman em seu estudo de caso sobre o sistema do ponto de venda e identificamos alguns *archetype patterns*, padrões de análise e projeto que poderiam ser utilizados para ajudar a criar bons modelos durante o processo de desenvolvimento de software.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho estudamos o RUP e três tipos de padrões existentes: padrões de projeto, padrões de análise e *archetype patterns*. Mostramos como é possível transitar entre os três tipos de padrões apresentados e como esse padrões podem ser aplicados ao RUP.

Este capítulo apresenta as conclusões a que chegamos através deste trabalho e trabalhos futuros que podem ser desenvolvidos tomando como base as idéias apresentadas e defendidas neste trabalho.

6.1 Contribuições

Motivado pela necessidade de técnicas que auxiliem no desenvolvimento de software com qualidade, este trabalho procurou mostrar como diferentes tipos de padrões podem ser utilizados em conjunto e como esses padrões podem ser utilizados em um processo de desenvolvimento de software. Este trabalho apresentou três tipos de padrões, dentre diversos existentes, e confirmou que independentemente de descreverem soluções em níveis de abstração diferentes esses padrões podem auxiliar na descrição de uma boa solução para um determinado problema.

Em um processo de desenvolvimento de software como o RUP [4], em que diversos modelos são criados para auxiliar na produção de softwares de qualidade, a utilização de padrões em suas fases de desenvolvimento pode ajudar na elaboração de soluções de qualidade. No RUP as atividades a serem realizadas e artefatos a serem gerados durante o decorrer do processo são definidos no início do projeto. Saber com antecedência quais tipos de padrões poderiam ser úteis no desempenho das atividades e criação dos artefatos pode ajudar os responsáveis a se preparar para as tarefas, a medida que poderiam consultar os catálogos de padrões mais apropriados para encontrar soluções para determinados problemas. Uma maneira que isso poderia ser feito seria através da criação de uma relação mais íntima entre fases, disciplinas, atividades e artefatos com os diferentes tipos de padrões existentes. Esta realação poderia ser realizada através de documentação do uso de padrões em processos e indicações, na documentação dos próprios processos, de padrões que poderiam ser utilizados como auxílio no desenvolvimento das atividades e criação de artefatos.

Este trabalho mostrou que os padrões de projeto [5], padrões de análise [6] e *archetype patterns* [7] seriam utilizados com maior freqüência na fase de elaboração do RUP. Nesta fase as disciplinas de modelagem de negócios e análise e projeto utilizam bastante modelos de negócios,

modelos de análise e modelos de projeto, nos quais os três tipos de padrões estudados poderiam ajudar na criação de boas soluções.

Este trabalho mostrou também que apesar de descreverem soluções em níveis de abstrações diferentes, os diferentes tipos de padrões podem ser utilizados em conjunto. Mostramos que é possível realizar a transição de uma solução apresentada utilizando um determinado tipo de padrão para outro. Através de dois exemplos este trabalho mostrou que alguns padrões descrevem soluções para o mesmo problema e que podemos partir da solução de um deles para o outro. Vimos que os padrões de análise e *archetype patterns* possuem padrões que compartilham inclusive do mesmo nome e que os padrões de projeto podem ser utilizados na descrição das suas soluções.

Através da realização desses dois exemplos chegamos a conclusão que os *archetype patterns* e os padrões de análise são padrões que descrevem soluções de uma forma parecida, exceto pelo fato dos *archetype patterns* buscarem abstrair o domínio de aplicação específico do problema procurando uma solução mais geral para o problema. Concluímos também que através da criação de modelos de negócio bem estruturados podemos observar nestes modelos a presença de situações em que a utilização de padrões de projeto pode ser identificada observando as diretrizes que determinam a utilização dos padrões.

Um fator que limitou um maior relacionamento entre os padrões de análise e os *archetype patterns* foi o fato de os padrões de análise descritos por Fowler em [6] estarem bastante relacionados a aspectos financeiros e de contabilidade. Como apresentamos no Capítulo 3, Martin Fowler criou o seu catálogo de padrões de análise baseado na sua própria experiência pessoal. Esse fato fez com que tanto os *archetype patterns*, descritos por Arlow e Neudstadt, e os padrões de projeto, descritos por Gamma et. al., fossem mais abrangentes que os padrões descritos por Fowler, pois descreviam soluções mais gerais.

No entanto, as idéias lançadas por Fowler foram importantes para a definição de uma classe de padrões que devem ser observados em diferentes domínios de negócios. A observação de novos padrões de análise e a conseqüente evolução do catálogo desses padrões deve aumentar a potencialidade do relacionamento entre *archetype patterns* e padrões de análise a medida que os padrões de análise devem ser utilizados para realizar um maior detalhamento dos *achetype patterns* ou especialização desses padrões para um domínio de negócio específico.

Para exemplificar uma situação real em que os padrões poderiam ser utilizados no RUP este trabalho apresentou uma parte de um estudo de caso desenvolvido por Craig Larman [19]. A partir deste estudo de caso mostramos que os três tipos de padrões poderiam ter sido utilizados para solucionar alguns pontos principais do sistema identificados por Larman. Em seu trabalho Larman já havia realizado uma relação entre o processo de desenvolvimento de software com alguns padrões definidos por ele, mas de uma forma menos abrangente que a proposta por este trabalho. O relacionamento entre o processo de desenvolvimento de software e a utilização de padrões deveria ser mais acentuada de maneira a divulgar com mais afinco os benefícios trazidos pelo uso de padrões, benefícios esses que apresentamos neste trabalho.

6.2 Trabalhos Futuros

Existem diversos outros tipos de padrões e diversos catálogos desses padrões que podem ser utilizados para o desenvolvimento de software. O *Hillside Group* [20], um grupo dedicado a melhorar a qualidade do desenvolvimento de software, é uma boa fonte para encontrar diversos tipos de padrões.

Este trabalho apresentou a aplicação de padrões de projeto, padrões de análise e *archetype patterns* ao RUP. A aplicação de outros tipos de padrões ao processo de desenvolvimento de

software poderia ser apresentada em trabalhos futuros. Padrões de requisitos, padrões de casos de uso, padrões de arquitetura, entre outros diversos tipos de padrões poderiam ser aplicados ao processo do RUP para ajudar na realização de diversas atividades e geração de diversos artefatos utilizando esses padrões.

Acreditamos que o ideal seria associar ao máximo a utilização de padrões à realização de atividades do processo de desenvolvimento em todas as suas fases e disciplinas. Os padrões são uma forma de compartilhar boas soluções já aplicadas. Vincular o uso de padrões de forma integrada ao desenvolvimento de atividades em um processo de desenvolvimento poderia ajudar bastante na obtenção de softwares de qualidade. Explicitar nas documentações existentes dos processos de desenvolvimento ou na documentação dos padrões existentes esse vínculo seria uma boa forma de fortalecer a ligação existente entre a utilização desses dois recursos.

A fim de possibilitar uma maior integração entre os padrões de análise e os *archetype patterns* o catálogo de padrões de análise deveria ser atualizado e enriquecido, pois quando comparados aos *archetype patterns*, os padrões de análise aparentam ser irrelevantes, o que de fato não é verdade. A continuação do estudo de padrões de análise é importante para aumentar a documentação de problemas específicos a determinados domínios de negócio de forma que esses padrões possam ser utilizados de uma maneira mais abrangente.

Apesar de o RUP ser o processo de desenvolvimento de software mais utilizado atualmente existem também diversos outros processos de desenvolvimento. A aplicação de padrões a outros tipos de processo poderia também ser tema de trabalhos a serem realizados.

A partir da aplicação de padrões a mais de um tipo de processo de desenvolvimento de software poderiam ser realizados estudos comparativos sobre a utilização de padrões em processos de desenvolvimento de software. Estudos comparativos poderiam determinar quais processos de desenvolvimento realizam um maior incentivo à utilização de padrões na geração de seus artefatos. Poderia ser avaliado o impacto da utilização de padrões em um processo de desenvolvimento em relação a qualidade, à produtividade e outros fatores.

Bibliografia

[1] MASSONI, Tiago; SAMPAIO, Augusto; BORBA, Paulo. *A RUP-Based Software Process Supporting Progressive Implementation*. Idea Group Publishing, 2003.
[2] OSTERWEIL, Leon, et. al. *Strategic Direction in Software Quality*. ACM Press, 1996.
[3] ZUZER, Wolfgang; HEIL, Stefan; GRECHENIG, Thomas. *Software Quality Development and Assurance in RUP, MSF and XP – A Comparative Study*. ACM Press, 2005.
[4] KRUCHTEN, Philippe. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2003.
[5] GAMMA, Erich, et. al. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
[6] FOWLER, Martin. *Analysis Patters – Reusable Object Models*. Addison-Wesley, 1997.
[7] ARLOW, Jim; NEUSTADT, Ila. *Enterprise Patterns and MDA – Building Better Software with Archetype Patterns and UML*. Addison-Wesley, 2003.
[8] PRESSMAN, Roger S. Engenharia de Software. Pearson Education, 1995.
[9] MICROSOFT CORPORATION. *Microsoft Solutions Framework White Paper*. Microsoft Press, 2003
[10] BECK, Kent. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
[11] JACOBSON, Ivar, et. al. *The Unified Software Development Process*. Object Technology. Addison-Wesley, 1999.
[12] HIRSCH, Michael. *Making RUP Agile*. ACM Press, 2002.
[13] FOWLER, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2005.
[14] ALEXANDER, Cristopher; ISHIKAWA, Sara; SILVERSTEIN, Murray. *A Pattern Language – Towns, Buildings, Construction*. Oxford University Press, 1977.
[15] COPLIEN, James; SCHMIDT, Douglas. *Pattern Languages of Programming Design*. Addison-Wesley, 1995.
[16] MEYER, Bertrand. *Object-Oriented Software Construction*. Addison-Wesley, 1997.
[17] JUNG, Carl Gustav. *The Archetypes and the Collective Unconscious*. Princeton University Press, 1981.
[18] BUREAU INTERNATIONAL DES POIDS ET MESURES. *International System of Units*. Disponível em: http://www.bipm.fr. Acesso em: 1 de novembro de 2006.
[19] LARMAN, Craig. *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, 2005.
[20] HILLSIDE GROUP. *Hillside Group Patterns Library*. Disponível em: http://hillside.net/patterns. Acesso em: 08 de novembro de 2006.

Anexo A

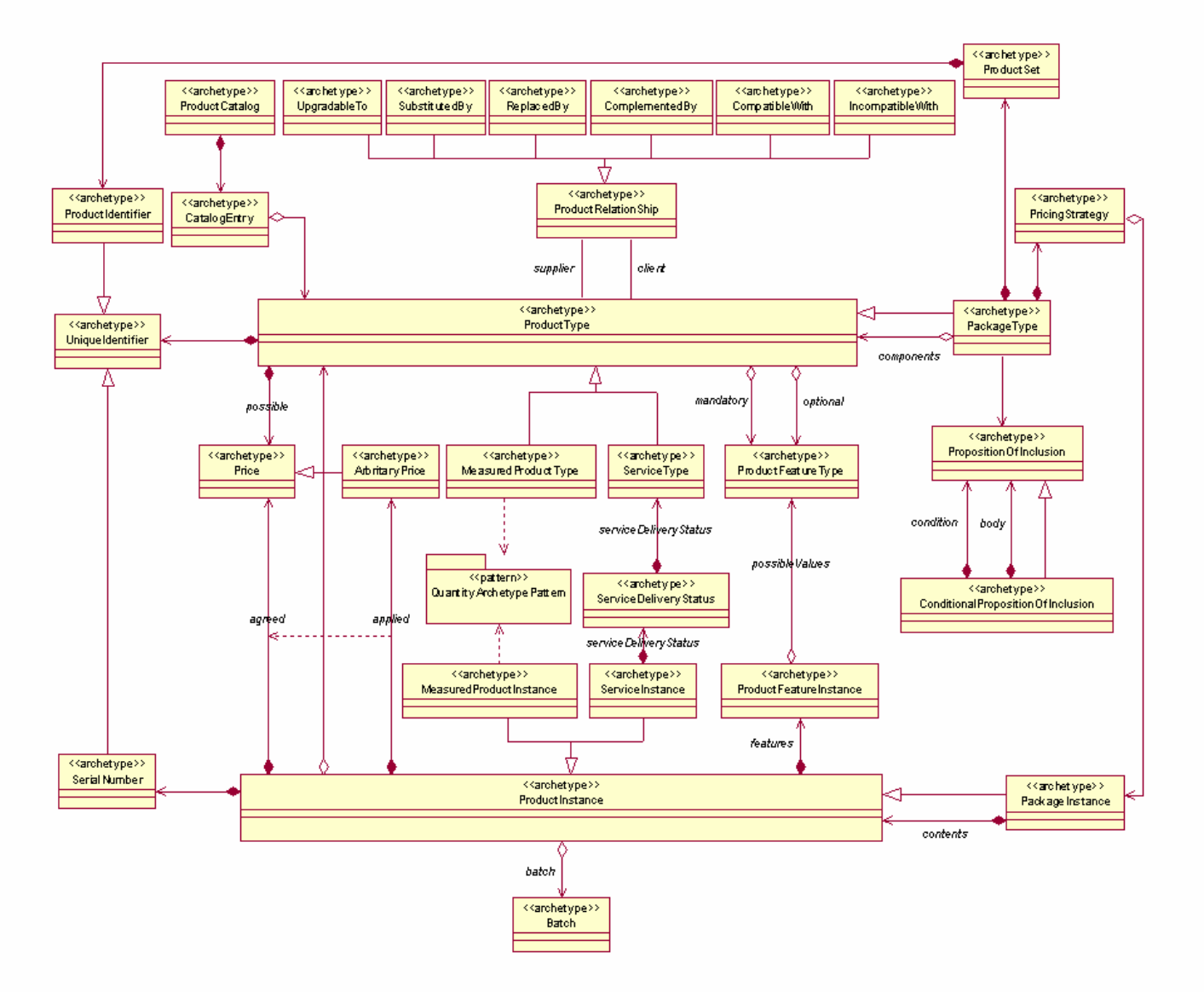

Figura 32. Modelo que representa o *product archetype pattern*.

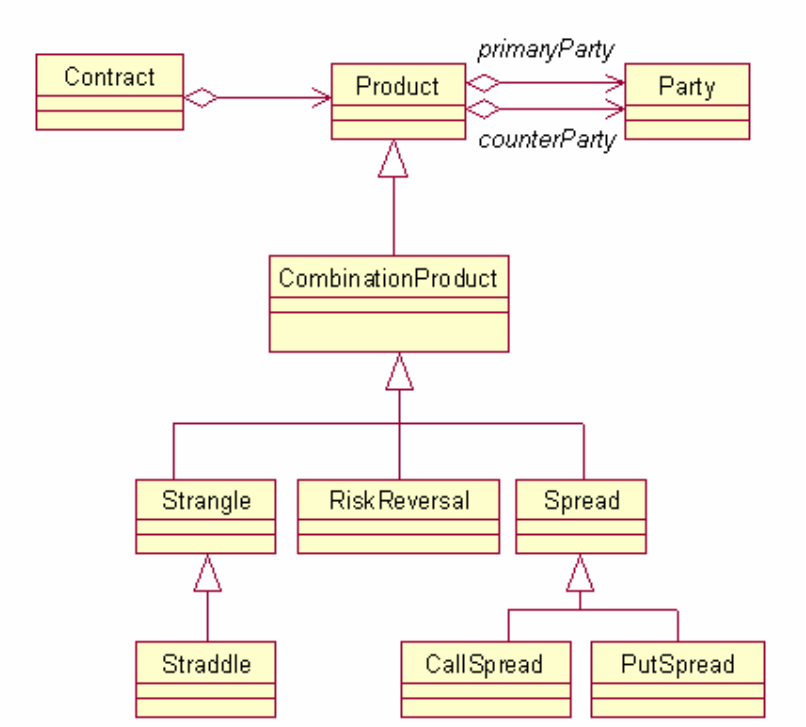

Figura 33. Modelo que representa o *product analysis pattern*.