Fundação Edson Queiroz
Universidade de Fortaleza - UNIFOR

Gabriela Telles de Souza

Um Catálogo de Metapadrões e Padrões para Sistemas de Informação e um Modelo para a sua Aplicação no Desenvolvimento de *Software*

Fortaleza-Ceará

2005

Fundação Edson Queiroz
Universidade de Fortaleza - UNIFOR

Gabriela Telles de Souza

Um Catálogo de Metapadrões e Padrões para Sistemas de Informação e um Modelo para a sua Aplicação no Desenvolvimento de *Software*

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza, como requisito parcial para obtenção do Título de Mestre em Informática Aplicada.

Orientador: Arnaldo Dias Belchior

Fortaleza-Ceará
Dezembro, 2005

Gabriela Telles de Souza

Um Catálogo de Metapadrões e Padrões para Sistemas de Informação e um Modelo para a sua Aplicação no Desenvolvimento de *Software*

Data de Aprovação: _____

Banca Examinadora:

Prof. Dr. Arnaldo Dias Belchior

(Orientador – UNIFOR)

Prof. Dr. Nabor das Chagas Mendonça

(Membro – UNIFOR)

Prof. Dra. Rossana Maria de Castro Andrade

(Membro – UFC)

SOUZA, Gabriela Telles. Um Catálogo de Metapadrões e Padrões para Sistemas de Informação e um Modelo para a sua Aplicação no Desenvolvimento de Software .Fortaleza: UNIFOR, Dissertação (Mestrado em Informática Aplicada), 2005.

Perfil do Autor: concluiu Bacharelado em Ciência da Computação na Universidade Federal do Ceará em 1997. Desempenha o cargo de coordenadora de projetos no Instituto Atlântico desde junho de 2002.

RESUMO: Desenvolver *software* reusando padrões pode reduzir o custo e condensar o ciclo de vida do desenvolvimento, e simultaneamente manter a qualidade dos sistemas desenvolvidos. Entretanto, o potencial de aplicar padrões em sistemas de *software* não é aproveitado inteiramente. Embora diversos padrões tenham sido descritos para analisar, projetar, e implementar *software*, não há ainda orientação ou metodologia madura e largamente aceita, que forneça uma abordagem sistemática para integrar esses diferentes tipos de padrões, durante o ciclo de desenvolvimento. Neste trabalho, apresentamos um catálogo de metapadrões e padrões para sistemas de informação e propomos um modelo para a aplicação de metapadrões e padrões integrado ao ciclo de vida de desenvolvimento de *software*, para apoiar a construção de sistemas. Os metapadrões apresentados são implementados por padrões de requisitos, padrões de projeto, e padrões de teste. Esses padrões foram utilizados com resultados satisfatórios em doze de projetos de *software* em uma organização de pesquisa e desenvolvimento de *software*.

Palavras chave: Desenvolvimento de *software* – Metapadrões - Padrões de *software* - Casos de uso - Casos de teste

Abstract

Software pattern oriented development may reduce the costs of software development lifecycles, and at the same time, keep the system quality. However, the potential of applying patterns in software development is not completely explored. Even though several patterns have been presented to analysis, design and implement software, there is a lack of guidelines or mature and broadly accepted methodologies to provide a systematic approach to integrate the different kinds of patterns through the development life cycle. In this work, we describe a metapattern and pattern catalog for information system and propose a model to apply metapattern and patterns within software development life cycles, supporting system construction. The metapatterns presented are implemented by requirement, design and test patterns. These patterns were used with good results in twelve software projects in a software research and development organization.

Key words: software development – metapatterns – software patterns – use cases.

Dedicatória

Ao meu marido, Giovano, que é minha fonte de inspiração e meu companheiro em todos os momentos.

Agradecimentos

A Deus por me proporcionar a vida, a sabedoria e a coragem para realizar esta pesquisa.

Aos meus pais, Tarcisio e Bruhilda, e meus irmãos, Valéria e Rodrigo, por todo o amor, carinho e incentivo ao longo de todos esses anos.

Ao meu marido e co-orientador, Giovano, pela paciência e incentivo durante os momentos difícéis; e por sua contribuição e inúmeras revisões do trabalho.

Ao meu filho, Luca, por iluminar meus dias com seu amor e alegria, mostrando a cada dia um motivo para continuar a caminhada. E por entender os momentos de ausência.

Ao meu filho, Levi, que vai chegar em janeiro de 2006, motivando a conclusão do trabalho antes da sua chegada.

Ao professor, Belchior, pela competência demonstrada na orientação deste trabalho e pela confiança depositada no nosso trabalho.

As amigas da Turma 6 do Mestrado em Informática Aplicada, Sergiana e Aretusa, que me acompanharam nas inúmeras noites de estudo em minha casa e tiveram paciência e sabedoria para estudarmos juntas.

À minha amiga, Patrícia, por ser minha confidente, me ouvir com paciência e amor.

Ao Instituto Atlântico, pelo apoio oferecido na publicação do nosso trabalho em eventos, liberação de horas de trabalho para estudo e apoio financeiro.

Sumário

Lista de figuras

Lista de Tabelas

Capítulo 1 Introdução

Neste capítulo, são apresentadas as principais questões que motivaram a realização deste trabalho, como também os objetivos a alcançar e sua organização.

1.1 Motivação

A última década testemunhou um interesse crescente em padrões de *software* tanto na academia quanto na indústria. Padrões surgiram como uma técnica efetiva para documentar e comunicar a experiência através de diferentes fases do desenvolvimento de *software* (GAMMA *et al.*, 1995). Contudo, o potencial dos padrões ainda não foi explorado completamente.

Existe uma série de questões que limitam a usabilidade dos padrões. Essas questões podem ser classificadas em dois tipos: questões de especificação e questões de desenvolvimento (HAMZA e FAYAD, 2005).

As questões de especificação estão relacionadas à forma pela qual os padrões são estruturados, especificados e formalizados. Por exemplo, o vocabulário empregado na descrição dos padrões, os *templates* adotados, as diversas formas de classificação, etc.

As questões de desenvolvimento, por outro lado, estão relacionadas com as dificuldades que surgem, quando temos que usar e integrar padrões ao desenvolvimento de *software*. Entre essas questões temos: (*i*) a redundância, em que diversos padrões visam solucionar o mesmo problema; (*ii*) a dificuldade na seleção do padrão a ser aplicado; (*iii*) a composição, que é o processo de relacionar diferentes padrões da mesma classificação para construir componentes; e (*iv*) a integração de padrões de diferentes classificações. Por exemplo: *Como um padrão de requisito, que foi utilizado na etapa de requisitos, poderia estar integrado a um padrão de projeto, a ser utilizado na etapa de análise e projeto*?

Segundo Hamza e Fayad (2005), as melhorias geradas pela aplicação de padrões em cada etapa do ciclo de desenvolvimento de *software* irão resultar na melhoria geral do produto final. Neste contexto, o problema de integração de padrões torna-se relevante.

Neste trabalho, apresentamos um catálogo de metapadrões e padrões para sistemas de informação e propomos um modelo para aplicação de padrões ao longo do ciclo de vida de desenvolvimento de *software*, abordando o problema de integração levantado por Hamza e

Fayad (2005), através do uso de metapadrões integrados a processos e metodologias de desenvolvimento de *software*.

1.2 Objetivos

O objetivo geral deste trabalho é apresentar um catálogo de metapadrões e padrões para sistemas de informação e propor um modelo para a aplicação de metapadrões e padrões, alinhado ao processo de desenvolvimento de *software*, buscando o uso integrado de vários tipos de padrões com o ciclo de vida de desenvolvimento de *software*.

Os objetivos específicos deste trabalho são:

- Estender e aplicar o conceito de evoluções de padrões com o uso de metapadrões e catálogos;
- Estender e aplicar o conceito de metapadrões para padrões de requisitos, padrões de projeto, e padrões de teste;
- Propor um catálogo de padrões e metapadrões para as etapas de requisitos, projeto, e teste de sistemas de informação; e
- Apresentar um estudo de caso do modelo de metapadrões e padrões proposto como um dos requisitos para sua validação, e poder auxiliar na aplicação desse modelo em projetos de *software*.

1.3 Metodologia

Para a elaboração deste trabalho, foi realizado um estudo bibliográfico na literatura, visando o embasamento teórico necessário sobre padrões de *software*, envolvendo mormente padrões de requisitos, padrões de projeto, e padrões de teste.

Analisamos as principais dificuldades encontradas na aplicação de padrões durante o desenvolvimento de *software*. Uma das conclusões deste estudo foi a constatação de que existem muitos padrões propostos. Porém, a aplicação desses padrões ao longo do ciclo de vida de desenvolvimento de *software* ainda necessita ser melhor explorada, havendo carência de métodos largamente aceitos ou de técnicas que auxiliem neste contexto.

A partir da pesquisa bibliográfica realizada, foi elaborado um catálogo de metapadrões e padrões, visando atender as principais etapas do ciclo de vida de desenvolvimento de *software*. A partir desse catálogo foi proposto um modelo para a aplicação desses

metapadrões e padrões. A seguir, foi realizado um estudo de caso em uma instituição de *software*, envolvendo vários projetos de *software* para a validação do modelo proposto.

Neste trabalho são utilizadas as seguintes notações, que visam facilitar seu entendimento:

- Termos em *itálico* são utilizados para:
  - Representar palavras ou expressões em língua estrangeira, com exceção de nomes de ferramentas ou produtos;
  - Identificar nome de classes e métodos em padrões de projeto; e
  - Destacar palavras ou expressões, que estão sendo definidas ao longo do texto.
- Termos em **negrito** são utilizados para:
  - Representar o nome dos padrões e metapadrões descritos e citados; e
  - Identificar componentes dos padrões.

1.4 Organização do trabalho

Este trabalho está organizado como se segue.

No Capítulo 2, descrevemos os principais modelos de ciclos de vida, processos e modelos de desenvolvimento e produtos de trabalhos gerados nas principais etapas do processo de desenvolvimento de *software*.

No Capítulo 3, apresentamos conceitos de padrões de *software*, envolvendo linguagem de padrões e coletânea de padrões e metapadrões. São identificados formatos de padrões e seus componentes, como também diversos critérios de classificação de padrões.

No Capítulo 4, propomos um catálogo de metapadrões e padrões. São descritos os metapadrões **Meta-CRUD**, **Relatório**, **Assistente**, e **Transação**, e os padrões a eles relacionados.

No Capítulo 5, propomos o modelo para a aplicação de metapadrões e padrões no desenvolvimento de *software*, e apresentamos os trabalhos relacionados a esse modelo.

No Capítulo 6, delineamos as principais conclusões deste trabalho e suas perspectivas futuras.

No Apêndice I, II e III, apresentamos a Especificação de caso de uso, o Projeto, e a Especificação de teste do Caso de uso *Manter Usuário*.

No Apêndice IV, V e VI, mostramos a Especificação de caso de uso, o Projeto, e a Especificação de teste do Caso de uso *Gerar Relatório de Obras de Arte*.

No Apêndice VII, VIII e IX, exibimos Especificação de caso de uso, o Projeto, e a Especificação de teste do Caso de uso *Exportar Obras para Catálogo*.

No Apêndice X, XI e XII, apresentamos Especificação de caso de uso, o Projeto, e a Especificação de teste do Caso de uso *Configurar Site*.

Capítulo 2 Desenvolvimento de *software*

Neste capítulo, são apresentados os conceitos de desenvolvimento de *software* utilizados para a estruturação deste trabalho.

2.1 Introdução

A engenharia de *software* é trata dos aspectos do desenvolvimento do *software*, desde sua especificação até sua manutenção. Ela não trata apenas de aspectos técnicos do desenvolvimento do *software*, mas também do gerenciamento do projeto de *software* e do desenvolvimento de ferramentas, métodos e técnicas que virão a dar suporte à produção de *software*.

Na engenharia de *software*, um processo é um conjunto de passos parcialmente ordenados com a intenção de construir um produto de *software* de qualidade, capaz de atender às necessidades e exigências do usuário final, de acordo com planejamento e orçamento previstos (SOMMERVILLE, 2001).

Um processo de desenvolvimento de *software* define:

- Um ciclo de vida para o *software*;
- Métodos que serão utilizados durante o desenvolvimento;
- Ferramentas que suportam esses métodos; e
- Papéis e responsabilidades da equipe de desenvolvimento.

Um modelo de processo é uma respresentação ou uma idealização de aspectos selecionados da estrutura, do comportamento, da operação, ou de outras características de um processo de *software* (IEEE, 1990). Também é uma representação simplificada de um processo de *software*, apresentado a partir de uma perspectiva específica.

Atualmente, existem vários processos de desenvolvimento de *software*, entre ele, podemos citar: RUP (*Rational Unified Process*) (RUP, 2003; KRUCHTEN, 2001), XP (*Extreme Programming*) (BECK, 1999), *Unified Process* (UP, 2005), MSF (MSF, 2005), entre outros. .Neste trabalho, serão abordados conceitos do RUP e do modelo de processo CMMI (*Capability Maturity Model Integration*) (CHRISSIS *et al.*, 2004), pois estes foram utilizados no estudo de caso apresentado no Capítulo 5.

Apesar dos diversos processos de desenvolvimento de *software* existentes possuírem abordagens próprias, as disciplinas (etapas) desses processos, descritas a seguir, são comuns a eles. Denominaremos essas disciplinas de "*Disciplinas Fundamentais*".

- *Requisitos*: definem as funcionalidades e as restrições do *software*.
- *Análise e Projeto*: produz a arquitetura utilizada como base para o desenvolvimento do *software*.
- *Implementação*: produz e libera o código para o cliente final.
- *Testes*: validam o *software* para garantir que funcionalidades e restrições sejam atendidas.

As disciplinas de um processo de *software* são organizadas de acordo com modelos de ciclo de vida. Um ciclo de vida é uma descrição de um conjunto de disciplinas, que devem ser executadas para o desenvolvimento de um produto de *software*. O ciclo de vida determina a interação entre essas disciplinas e o momento em que elas devem ser executadas. Os ciclos de vida a seguir têm sido bastante citados na literatura de engenharia de *software*.

2.2 Ciclos de vida

O ciclo de vida *Cascata* (clássico), mostrado na Figura 2.1, é o mais tradicional. Neste ciclo de vida, o desenvolvimento de *software* é organizado de forma a percorrer cada disciplina, em seqüência, apenas uma vez.

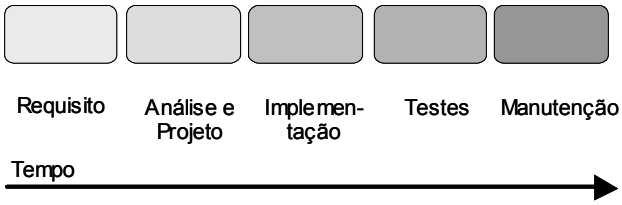


Figura 2.1: Ciclo de vida *Cascata* (adaptado de RUP (2003))

O ciclo de vida *Espiral*, observado na Figura 2.2, é um modelo cíclico e não linear que assume que o processo de desenvolvimento ocorre em ciclos. Cada ciclo representa uma fase do processo de desenvolvimento. Assim, o ciclo mais interno diz respeito à viabilidade do sistema, o próximo ciclo refere-se à definição dos requisitos, o ciclo seguinte contempla a análise e projeto e assim por diante.

A diferença fundamental entre o ciclo de vida *Espiral* e os demais ciclos de vida é que o *Espiral* se preocupa explicitamente com os riscos do projeto.

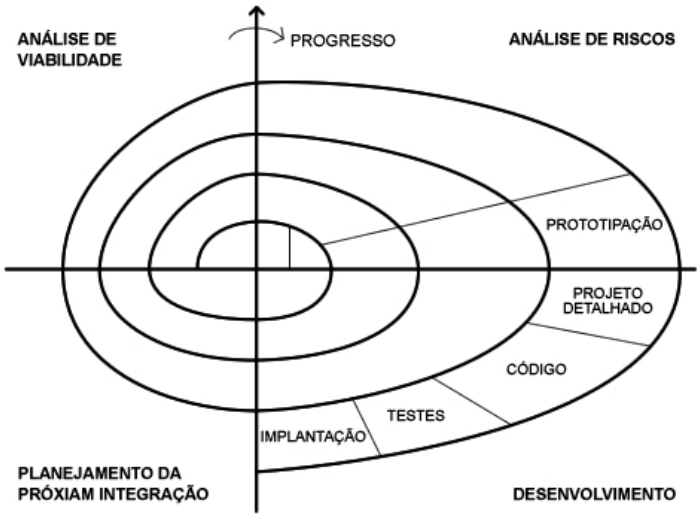


Figura 2.2: Ciclo de vida *Espiral* (SOMMERVILLE, 2001)

No ciclo de vida *Iterativo* as diversas disciplinas são percorridas várias vezes, gerando um melhor entendimento dos requisitos, planejando uma arquitetura robusta, elevando a organização do desenvolvimento, e, por fim, liberando uma série de implementações que são, gradualmente, mais completas. O ciclo de vida *Iterativo* é ilustrado na Figura 2.3.

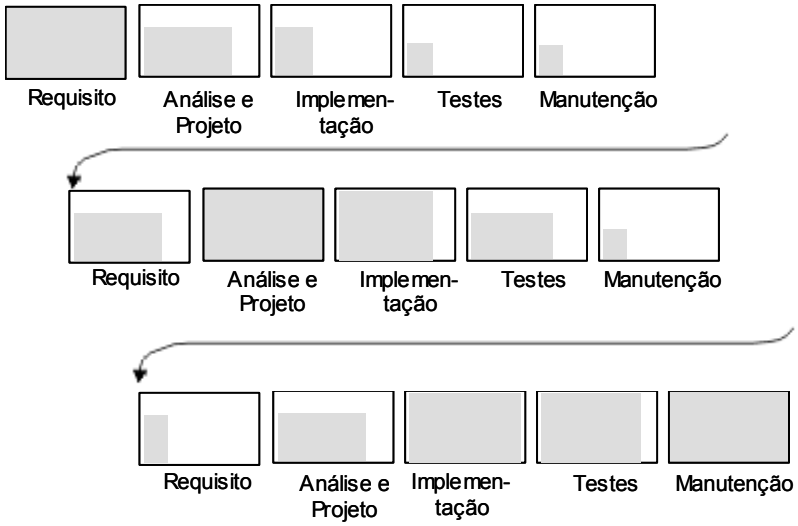


Figura 2.3: Ciclo de vida *Iterativo* (adaptado de RUP (2003))

Existem duas abordagens para o ciclo de vida *Iterativo*: a abordagem *Incremental* e a abordagem *Evolutiva*.

Na abordagem *Incremental*, Figura 2.4, as necessidades do usuário são determinadas e os requisitos do sistema são definidos. Em seguida, o restante do desenvolvimento é realizado por uma seqüência de iterações. A primeira iteração incorpora parte das capacidades

planejadas, a próxima iteração adiciona mais capacidades, e assim por diante, até o sistema estar completo.

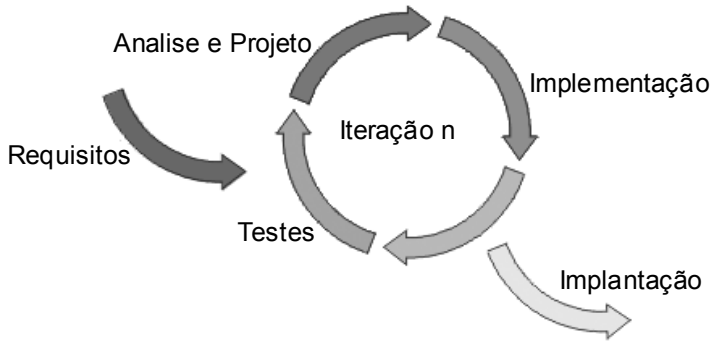


Figura 2.4: Abordagem *Incremental* (adaptado de RUP (2003))

Na abordagem *Evolutiva,* as necessidades do usuário não são totalmente entendidas, portanto os requisitos não podem ser definidos antecipadamente, sendo refinados em cada iteração, sucessivamente.

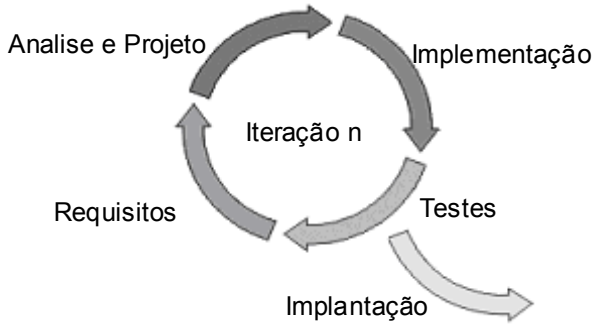


Figura 2.5: Abordagem *Evolutiva* (adaptado de RUP (2003))

2.3 Processos e Modelos de Desenvolvimento

Vários modelos e processos foram propostos com o objetivo de auxiliar o desenvolvimento de produtos de *software* de qualidade, como por exemplo, o RUP (2003) e o CMMI (CHRISSIS *et al.*, 2004). O CMMI é um modelo desenvolvido pelo SEI (*Software Engineering Institute*) que visa orientar as organizações de *software* na implementação de melhorias contínuas em seu processo de desenvolvimento. O RUP é um é um *framework* de processo customizável que abrange boas práticas do desenvolvimento de *software*.

2.3.1 RUP

O RUP é um *framework* de processo iterativo e incremental que provê uma abordagem disciplinada para o desenvolvimento de *software* (POLLICE, 2002).

Conforme apresentado na Figura 2.6, o RUP possui duas dimensões. O eixo horizontal representa o aspecto dinâmico do processo e mostra aspectos do ciclo de vida à medida que este se desenvolve. O eixo vertical representa o aspecto estático do processo: como ele é descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis do processo. Os processos são organizados em um conjunto de disciplinas para, posteriormente, definirem fluxos de trabalho e outros elementos do processo (RUP, 2003).

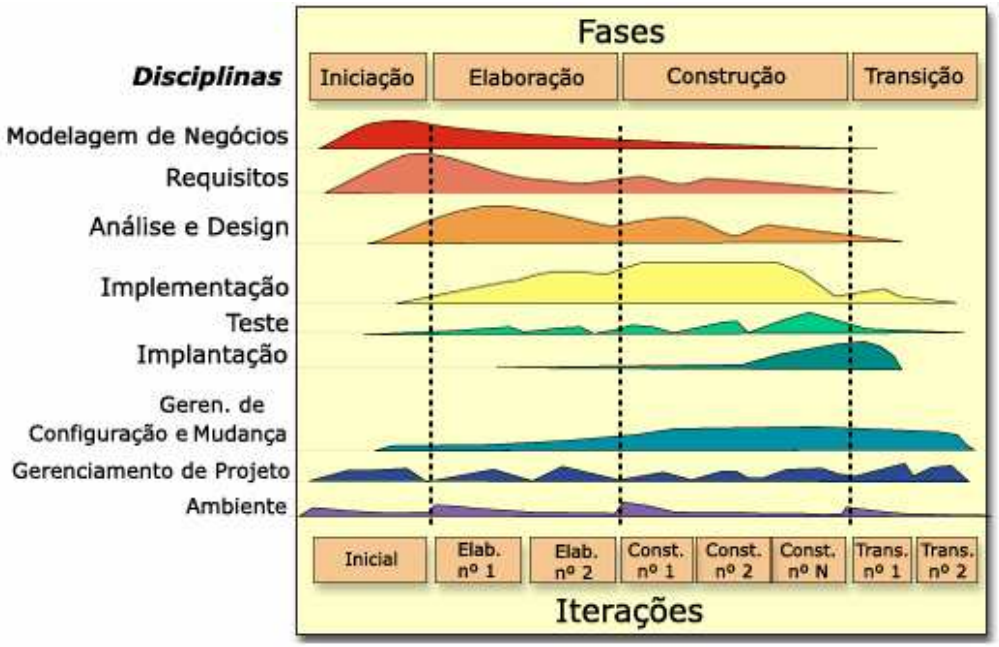


Figura 2.6: Ciclo de vida de desenvolvimento do RUP

Uma disciplina é uma coleção de atividades relacionadas que estão ligadas a uma área de interesse dentro do processo (KRUCHTEN, 2003). As disciplinas do RUP são descritas por meio de fluxos de trabalho.

Os fluxos mostram uma seqüência significativa de grupos de atividades, que produzem um determinado resultado. Cada grupo de atividades é descrito por um diagrama de detalhamento de fluxo de trabalho. Esses diagramas mostram todas as atividades do grupo, os papéis envolvidos e os artefatos de entrada e saída. Um artefato é um produto de trabalho do processo. Os papéis usam os artefatos na execução das atividades.

Outros elementos auxiliares ao processo também são descritos nos formatos de *templates,* relatórios, orientações de trabalho, pontos de verificação, diretriz de artefato,

mentor de ferramentas e conceitos. Entre esses elementos, os *templates* e as orientações de trabalho desempenham funções de destaque em uma arquitetura de processos. O primeiro tipo representa modelos pré-formatados que auxiliam na produção de artefatos. O segundo apresenta técnicas importantes na execução de atividades e produção de artefatos. Os elementos descritos e seus relacionamentos podem ser vistos na Figura 2.7.

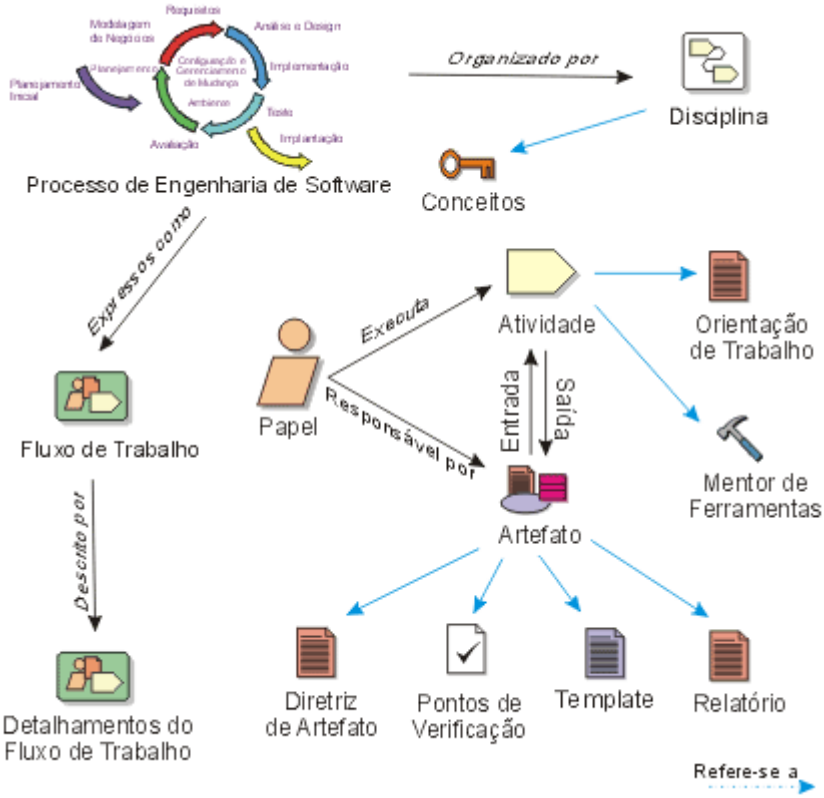


Figura 2.7: Elementos do RUP

O ciclo de vida de *software* do RUP é dividido em quatro fases seqüenciais: *Iniciação*, *Elaboração*, *Construção*, e *Transição*. Como mostrado na Figura 2.6, em cada fase podem ocorrer várias iterações. Uma iteração representa um ciclo completo de desenvolvimento. O gráfico mostra a variação da ênfase através do tempo. Por exemplo, nas iterações iniciais, é dedicado mais tempo aos requisitos. Já nas iterações posteriores, é gasto mais tempo com implementação.

O objetivo da fase de *Iniciação* é a delimitação do escopo do projeto, discriminando os principais cenários de operação. Além disso, custos, planos e riscos são estimados.

A fase de *Elaboração* tem por objetivo analisar o domínio do problema, de modo a propor uma arquitetura para os cenários da aplicação.

O objetivo da fase de *Construção* é o esclarecimento dos requisitos restantes e a conclusão do desenvolvimento do sistema com base na arquitetura definida.

Finalmente, a fase de *Transição* tem por objetivo assegurar que o *software* esteja disponível para seus usuários finais. A fase de *Transição* inclui testar o produto e realizar pequenos ajustes com base no *feedback* do usuário.

No fim do ciclo de vida da fase de *Transição*, os objetivos devem ter sido atendidos e o projeto deve estar em uma posição para fechamento. Em alguns casos, o fim do ciclo de vida atual pode coincidir com o início de outro ciclo de vida no mesmo produto, conduzindo à nova geração ou versão do produto. Para outros projetos, o fim da fase de transição pode coincidir com uma liberação total do produto a terceiros, que poderão ser responsáveis pela operação, manutenção e pelas melhorias no sistema liberado.

O RUP possui uma abordagem dirigida a casos de uso. Isso significa que os casos de uso definidos para o sistema são a base para todo o processo de desenvolvimento (KRUCHTEN, 2001) (Figura 2.8).

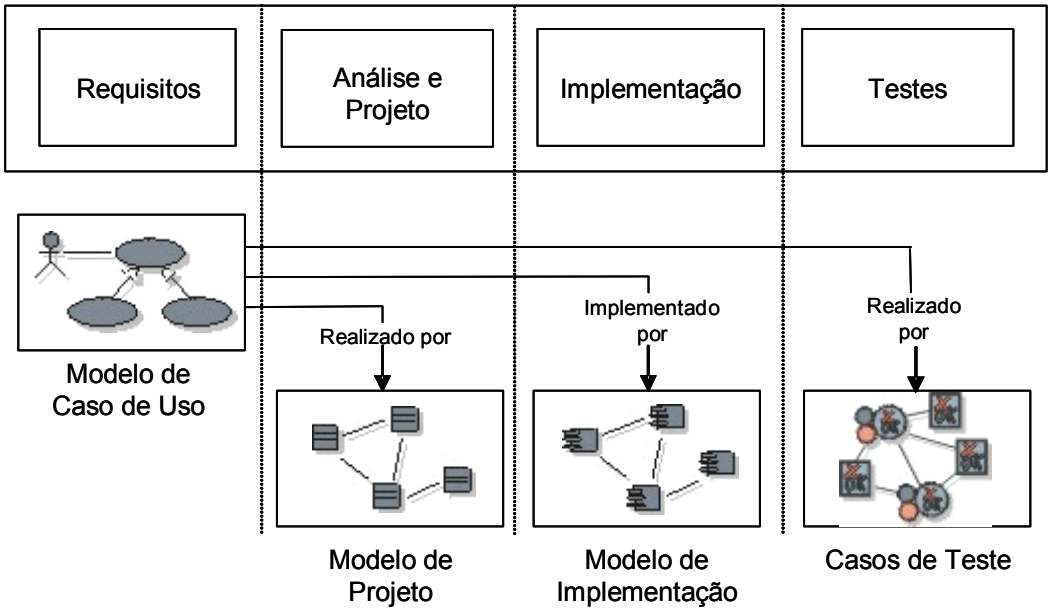


Figura 2.8: Fluxo de caso de uso durante o ciclo de desenvolvimento

Também é possível identificar as *disciplinas fundamentais* na estrutura do RUP. A disciplina de *Requisitos* é responsável por estabelecer e manter concordância com clientes e outros envolvidos sobre o que o sistema deve fazer, oferecer aos desenvolvedores do sistema uma compreensão melhor dos requisitos e definir as fronteiras do sistema. Os principais produtos de trabalho gerados nesta disciplina são o *Modelo de Caso de Uso* e os *Casos de Uso*.

A disciplina de *Análise e Projeto*, por sua vez, visa transformar os requisitos em um projeto do sistema e desenvolver a arquitetura. Os principais produtos de trabalho gerados nesta disciplina são o *Modelo de Projeto* e a *Realização de Caso de Uso*.

A finalidade da disciplina *Implementação* é implementar classes e objetos, testar e integrar os resultados produzidos. O principal produto de trabalho gerado nesta disciplina é o *Modelo de Implementação*.

A disciplina de *Testes* atua em vários aspectos como uma provedora de serviços para as outras disciplinas, enfatizando principalmente a avaliação da qualidade do produto. Nessa disciplina, são gerados os *Casos de Teste* que servirão de insumo para a execução dos testes.

2.3.2 CMMI

O CMMI é uma proposta de integração dos modelos CMM (*Capability Maturity Model*) (PAULK, 1993) em um único modelo, visando acomodar múltiplas disciplinas e ser flexível o suficiente para suportar dois tipos diferentes de representação (contínua e estágio).

Na *representação contínua*, a empresa possui a flexibilidade de escolher quais processos serão enfatizados ou priorizados para melhoria, conforme sua disponibilidade financeira e seus objetivos estratégicos. Já a *representação por estágios* proporciona uma melhoria organizacional, com a proposição de um conjunto de processos e metas a serem atingidos de acordo com o nível de maturidade desejado.

O CMMI é organizado em cinco níveis de maturidade. Cada nível de maturidade é composto de áreas de processo (*Processes Areas* - PAs) que indicam o foco a ser dado pela organização para melhorar seu processo de *software*.

A Figura 2.9 apresenta os componentes do modelo CMMI. Os objetivos específicos organizam as práticas específicas e os objetivos genéricos organizam as práticas genéricas. Cada prática, específica e genérica, corresponde a um nível de maturidade. Objetivos e práticas específicas aplicam-se a áreas de processo individuais. Objetivos e práticas genéricas se aplicam a múltiplas áreas de processo (CHRISSIS *et. al*, 2004).

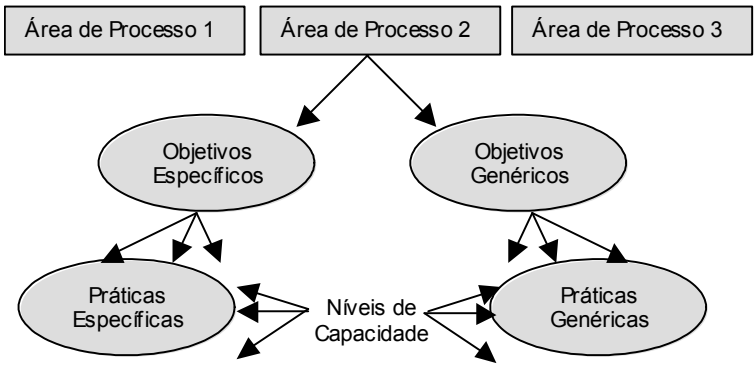


Figura 2.9: Componentes do modelo CMMI

As áreas de processo *Desenvolvimento de Requisitos*, *Solução Técnica* e *Verificação*, e *Validação* tratam as disciplinas de Requisitos, Análise e Projeto, Implementação e Testes, respectivamente, no modelo CMMI.

O objetivo da área de processo de *Desenvolvimento de Requisitos* é desenvolver, analisar e validar os requisitos do cliente e do produto. As práticas *Elicitar as Necessidades*, *Estabelecer Cenários Operacionais* e *Estabelecer a Definição dos Requisitos Funcionais* indicam como exemplo de produtos de trabalho o artefato *Caso de Uso*.

O objetivo da área de processo *Solução Técnica* é projetar, desenvolver e implementar soluções para os requisitos. Soluções, projetos e implementações abrangem produtos, componentes de produtos e processos relacionados ao ciclo de vida dos produtos. As duas práticas abaixo dessa área de processo corroboram com o uso de padrões.

A prática *Projetar o Produto ou Componentes do Produto,* cujo objetivo é desenvolver um projeto detalhado do produto e de seus componentes, indica o uso de padrões como uma técnica que facilita e apóia o desenvolvimento do projeto do produto. A prática *Implementar o Projeto,* cujo objetivo é implementar o projeto detalhado do produto e de seus componentes, também indica o uso de padrões como um método eficiente para implementar código.

O objetivo da área de processo *Verificação* é garantir que os produtos de trabalho estejam de acordo com os seus requisitos, enquanto que o da área de processo *Validação* é garantir que o produto desenvolvido está apto a ser usado. O artefato *Caso de Teste* é indicado como exemplo de produto de trabalho para algumas práticas desta PA.

2.4 Produtos de trabalho

Produto de trabalho é qualquer artefato produzido por um processo. Os artefatos que estão no contexto deste trabalho são: casos de uso, modelo de projeto, modelo de implementação, e casos de teste.

2.4.1 Caso de Uso

O conceito de caso de uso foi inicialmente publicado na conferência internacional OOPSLA 87 (*Object-Oriented Programming, Systems, Languages, and Applications*), no artigo "*Object-Oriented Development in an Industrial Environment*" (COCKBURN e FOWLER, 1998). Em 1992, uma descrição completa de casos de uso e de desenvolvimento de *software* orientado a casos de uso foi publicada por Jacobson *et al* (1992). A partir desta publicação, os casos de uso têm sido amplamente utilizados por diversos processos e modelos de desenvolvimento de *software*.

Caso de uso é um conceito bastante difundido e utilizado para a documentação e o desenvolvimento de requisitos (COCKBURN, 2001; SCHNEIDER; WINTERS, 2001; KRUCHTEN, 2001; BITTNER; SPENCE, 2002; JACOBSEN *et al.*, 1992). Segundo o RUP (2003), caso de uso é uma descrição de comportamento do sistema em termos de seqüências de ações. Um caso de uso deve produzir um resultado de valor observável para um ator. Ele contém todos os fluxos de eventos referentes à produção do "resultado de valor observável". Mais formalmente, um caso de uso define um conjunto de instâncias de casos de uso ou cenários.

Segundo Jacobson *appud* (COCKBURN e FOWLER, 1998), todo caso de uso identificado na etapa de requisitos é mapeado para a realização deste na análise ou projeto e, finalmente, mapeado para um conjunto de casos de teste, na etapa de testes. A habilidade de direcionar o desenvolvimento e não somente de ajudar na captura dos requisitos é o grande valor dos casos de uso.

A seguir, serão apresentados os principais conceitos que envolvem um caso de uso, mais especificamente uma Especificação de Casos de Uso:

- *Ator*: representa o papel que uma pessoa, um dispositivo de hardware ou um outro sistema pode desempenhar em relação a um sistema.
- *Fluxo de eventos*: contém as informações mais importantes derivadas da modelagem de casos de uso. Deve apresentar o que o sistema faz, e não como é o *design* do sistema para realizar o comportamento exigido.
- *Precondições*: descrevem o estado do sistema antes do início da execução de um caso de uso.

- *Pós-condições*: descrevem o estado do sistema após o término da execução de um caso de uso.
- *Pontos de extensão*: descrevem o que pode acontecer de diferente durante o cenário do caso de uso. Está ligado a um caso de uso de extensão.
- *Requisitos especiais*: em geral, descrevem requisitos não funcionais pertencentes ao caso de uso, que não é coberto no texto do fluxo de eventos.

Embora os casos de uso sejam reconhecidos como parte relevante na UML (Linguagem de Modelagem Unificada) (RUMBAUGH *et al.*, 2000), esta não descreve como se capturar o conteúdo de um caso de uso, uma vez que, praticamente, todo o valor dos casos de uso reside em seu conteúdo textual (Especificação de caso de uso). O diagrama de casos de uso por si só é de valor bastante limitado. Ainda não há uma maneira padronizada para se descrever o conteúdo de um caso de uso. Tem sido recomendado que um caso de uso seja descrito de forma breve e tenha fácil leitura. Um caso de uso rico em detalhes poderia ser justificado em função de seu risco ser elevado (FOWLER, 2005).

A finalidade mais importante de uma Especificação de caso de uso é comunicar o comportamento do sistema ao cliente ou ao usuário final. Conseqüentemente, essa especificação deve ser de fácil entendimento e de simples leitura (RUP, 2003).

Um dos pontos críticos de maior dificuldade em uma especificação de caso de uso é o aprendizado de como determinar qual o nível de detalhamento que o caso de uso deva ter: como "começar e terminar"; onde as características do sistema terminam e os casos de uso começam; onde os casos de uso terminam e o projeto começa?

De acordo com o RUP (2003), há três motivos relevantes para se elaborar uma Especificação de casos de uso:

- Facilitar o entendimento do caso de uso;
- Particionar o comportamento comum descrito em vários casos de uso; e
- Propiciar sua manutenção.

O *Modelo de Casos de Uso* é um modelo que descreve os requisitos de um sistema em termos de casos de uso. Este modelo contém representações dos casos de uso e de seus relacionamentos. O modelo de casos de uso é usado como fonte de informação essencial para atividades de análise, projeto, implementação, e teste.

2.4.2 Modelo de Projeto

O *Modelo de Projeto* é um modelo de objeto que descreve a realização dos casos de uso e serve como uma abstração do *Modelo de Implementação*, além de ser usado como base para atividades de implementação e teste.

Uma *Realização de Casos de Uso* descreve o modo como determinado caso de uso é realizado no modelo de projeto em termos de objetos de colaboração. A finalidade da *Realização de Casos de Uso* é separar as questões dos especificadores do sistema (representadas pelo modelo de casos de uso e pelos requisitos do sistema) das questões dos projetistas do sistema.

2.4.3 Modelo de Implementação

O *Modelo de Implementação* é formado pelo conjunto de componentes e de subsistemas de implementação. Os componentes de implementação incluem componentes de produtos liberados (como executáveis) e artefatos a partir dos quais esses produtos são criados (como arquivos de código-fonte).

No ambiente de programação, os componentes adquirem a forma de arquivos de código-fonte, arquivos binários, e arquivos de dados, organizados em diretórios. Os subsistemas são materializados na forma de diretórios, com informações adicionais sobre estruturas ou gerenciamento.

2.4.4 Casos de Teste

Os casos de teste encapsulam um conjunto específico de idéias de teste, condições de execução e resultados esperados identificados, com a finalidade de avaliar os casos de uso do sistema.

O objetivo do *Caso de Teste* é identificar e comunicar formalmente quais aspectos serão avaliados no caso de uso, delimitando um número adequado de testes específicos para garantir a abrangência do teste.

2.5 Conclusão

Este capítulo enfocou elementos do processo de desenvolvimento do *software*, relevante no contexto deste trabalho, através de conceitos do RUP (2003) e do CMMI (CHRISSIS *et al.*, 2004). No próximo capítulo, serão abordados os padrões de *software*.

Capítulo 3 Padrões de *software*

Neste capítulo, são apresentados os conceitos de padrões de *software* mais relevantes para o desenvolvimento deste trabalho.

3.1 Introdução

Os conceitos de padrões surgiram na área de arquitetura, com os trabalhos de Christopher Alexander, nas décadas de 60 e 70 (ALEXANDER *et al.*, 1977; ALEXANDER, 1979), criando as primeiras definições para os termos padrão e linguagens de padrões.

Alexander *et al.* (1977) definem um padrão como uma entidade que descreve um problema que ocorre repetidamente em um ambiente. Em seguida, descreve a essência de uma solução para este problema, de tal forma que se possa usar essa solução incontáveis vezes, sem, no entanto, utilizá-la do mesmo modo. Esse conceito tem sido utilizado no domínio da engenharia de *software* como uma forma de descrever boas soluções para problemas específicos em todo o ciclo de vida do projeto.

A engenharia de *software* tem adotado boas soluções para a resolução de problemas que ocorrem ao longo das etapas do ciclo de vida, em que se aplicam os conceitos de padrões. Por exemplo, Coplien (1996) descreveu um padrão de projeto como uma parte da literatura que descreve um problema de projeto e uma solução geral para o problema num contexto particular.

Considerando-se as definições apresentadas anteriormente, temos que um padrão é uma entidade que documenta uma solução, um problema recorrente e o contexto em que se deve aplicar tal solução. Para este trabalho, consideraremos este conceito de padrões.

É importante salientar que o objetivo da comunidade de padrões de *software* é documentar e compartilhar soluções comprovadas de engenheiros de *software* experientes, para problemas recorrentes em um determinado contexto, criando, assim, uma literatura que auxilie na adoção de boas práticas para o desenvolvimento de sistemas. Neste contexto, Vlissides (1997) cita quatro benefícios importantes no reuso de padrões de *software*:

- Capturar experiências tornando-as acessíveis aos não experientes;
- Formar um vocabulário comum a fim de ajudar desenvolvedores a se comunicarem melhor;

- Ajudar engenheiros de *software* a entenderem um sistema mais rapidamente, quando ele estiver documentado com padrões reutilizados; e
- Facilitar a reestruturação de um sistema, tendo ele sido ou não projetado com padrões.

Além dos benefícios citados, uma pesquisa realizada por Andrade e Marinho (2003) e publicada em Andrade *et al.* (2003) apresenta outras vantagens no reuso de padrões de projeto para o desenvolvimento de sistemas, como está listado a seguir:

- Evolução de código;
- Modularidade;
- Desacoplamento entre áreas de responsabilidades, de forma que as mudanças em uma área não ocasionem mudanças nas outras;
- Diminuição da complexidade do projeto e do código final;
- Facilidade na criação de *frameworks* contendo padrões de projeto já implementados, facilitando posteriormente o reuso dos padrões;
- Estabilidade do código;
- Confiabilidade no reuso de padrões cujas soluções são comprovadas;
- Ganho de produtividade;
- Facilidade de repassar conhecimento entre os desenvolvedores experientes; e
- Facilidade de aprendizado de novas áreas de conhecimento para uma equipe sem experiência na aplicação a ser desenvolvida.

Entretanto, algumas desvantagens no uso de padrões de *software* também podem ser constatadas. Por exemplo, Andrade *et al.* (2003) cita as seguintes desvantagens em casos particulares do reuso de padrões de projeto:

- *Perda de Eficiência*: o uso de alguns padrões pode tornar um programa mais lento, visto que, às vezes, é preciso adicionar novas classes ou novas camadas de aplicação ao modelo original do sistema, o que pode causar um retardo na sua execução;
- *Diminuição da Legibilidade/Manutenibilidade*: o uso de padrões pode diminuir a compreensão de um projeto ou de sua implementação. A aplicação de um padrão

pode ocasionar o aumento do número de classes, mensagens, linhas de código e níveis hierárquicos de classes, tornando o código do programa menos legível, ocasionando um custo maior de manutenção, e

- *Falta de Motivação da Equipe*: quando uma equipe de desenvolvimento não tem um entendimento prévio da solução de um padrão, das conseqüências da sua aplicação e do relacionamento com outros padrões, os membros desta equipe podem-se tornar desmotivados quanto à aplicação desse padrão.

3.2 Metapadrões

Os metapadrões foram apresentados em uma proposta de Pree (1995a) e constituem uma abordagem elegante e poderosa que pode ser aplicada para classificar e descrever padrões em um metanível. Portanto, metapadrões não substituem as abordagens de padrões, mas complementam-nas (PREE, 1995b; PREE, 1995c).

Segundo Kling e Flodström (2005) os metapadrões são padrões que descrevem padrões e são usados como uma ferramenta para apoiar o processo de desenvolvimento de *software*.

Este trabalho aplica e amplia o conceito de metapadrões proposto por Pree (1995a) no contexto de padrões de projeto, para metapadrões no contexto de padrões de requisitos e padrões de teste.

3.3 Coletânea de padrões

Conforme já mencionado anteriormente, existem centenas de padrões descritos nas diversas áreas e domínios de aplicação. Surge, então, a necessidade de se pensar em formas de agrupar os padrões existentes segundo algum critério, de forma a facilitar sua recuperação e reuso. Isto pode ser feito por meio de coleções de padrões, catálogos de padrões, sistemas de padrões ou linguagem de padrões. Estes conceitos serão descritos a seguir.

3.3.1 Coleção de padrões

Uma coleção de padrões é uma coletânea qualquer de padrões que não possuem nenhum vínculo entre si e, em geral, nenhuma padronização no formato de apresentação. Os padrões podem estar reunidos por terem sido apresentados em um mesmo congresso, por terem sido propostos pelo mesmo autor, ou por se referirem a um mesmo domínio, mas não possuem um relacionamento semântico significativo (BRAGA *et al.*, 2001).

3.3.2 Catálogo de padrões

Um catálogo de padrões é uma coletânea de padrões relacionados (talvez relacionados apenas fracamente ou informalmente). Em geral, subdivide os padrões em um pequeno número de categorias abrangentes e pode incluir algumas referências cruzadas entre os padrões (APPLETON, 2000).

Um catálogo de padrões pode oferecer um esquema de classificação e recuperação de seus padrões, já que eles estão subdivididos em categorias. Um catálogo de padrões não precisa ser completo morfológica e funcionalmente (GAMMA *et al.*, 1995; ALUR *et al.*, 2002; BUSCHMANN *et al.*, 1996).

3.3.3 Sistemas de padrões

Um sistema de padrões é um conjunto coeso de padrões co-relacionados que trabalham juntos para apoiar a construção e evolução de arquiteturas completas. Além de ser organizado em grupos e subgrupos relacionados em múltiplos níveis de granularidade, um sistema de padrões descreve além das diversas inter-relações entre os padrões e seus agrupamentos, a forma como eles podem ser combinados e compostos para resolver problemas mais complexos. Os padrões, num sistema de padrões, devem ser descritos em um estilo consistente e uniforme, precisando cobrir uma base de problemas e soluções suficientemente abrangente para permitir a construção de parcelas significativas de arquiteturas completas (APPLETON, 2000).

Além da estrutura e organização oferecidas por um catálogo de padrões, num sistema de padrões é adicionada uma maior profundidade à estrutura, maior riqueza à interação dos padrões e maior uniformidade ao catálogo de padrões (APPLETON, 2000).

A principal diferença entre sistemas de padrões e linguagem de padrões é que, linguegens de padrões são computacionalmente completas, mostrando as combinações possíveis dos padrões e de suas variações para produzir arquiteturas completas. Na prática, entretanto, a diferença entre sistemas de padrões e linguagens de padrões pode ser extremamente difícil de se verificar.

3.3.4 Linguagem de padrões

O conceito de linguagens de padrões foi introduzido por Alexander (ALEXANDER *et al.*, 1977). Em uma adaptação para a engenharia de *software,* Coplien (1996) descreveu uma

linguagem de padrões como uma coleção de padrões que trabalham em conjunto para construir um sistema.

As linguagens de padrões, além dos padrões que as compõem, possuem um título e, geralmente, também uma descrição ou resumo e um mapa, o qual consiste em um grafo ilustrador do modo como seus padrões estão relacionados (SANTOS, 2004).

Os padrões, em uma linguagem de padrões, trabalham em conjunto para solucionar um problema em comum (MARINHO, 2001). Cada padrão cria o contexto para o próximo padrão a ser aplicado e, dessa forma, os padrões de uma linguagem se completam. Neste sentido, segundo Hanmer (2003), uma linguagem de padrões deve ser completa morfológica e funcionalmente.

3.4 Formatos e componentes de padrões

Padrões são documentados textualmente e, geralmente, organizados em seções ou componentes, que definem o que chamamos de *template* ou formato do padrão. De acordo com Gamma *et al.* (1995), um template traz uma estrutura uniforme para a informação, tornando o padrão fácil de ser entendido, comparado e usado. Existem componentes que são essenciais para um padrão de *software* (MESZAROS e DOBLE, 1997; APPLETON, 2000):

- **Nome**: descreve um nome que referencia a solução ou o problema. Pode ser uma única palavra ou frase curta que se refira ao padrão e ao conhecimento ou estrutura descritos por ele.
- **Contexto**: descreve as pré-condições dentro das quais o problema e sua solução costumam ocorrer e para as quais a solução é desejável. Pode também ser considerado como a configuração inicial do sistema antes da aplicação do padrão.
- **Problema**: estabelece o problema a ser resolvido e descreve a intenção e objetivos do padrão perante o contexto e forças específicas.
- **Forças**: descrevem as considerações positivas e negativas a serem avaliadas a fim de definir o por quê da solução e se a solução proposta deve ser empregada.
- **Solução**: descreve os relacionamentos estáticos e regras dinâmicas usadas para obter o resultado desejado. Equivale a dar instruções que descrevem como o problema é resolvido, podendo, para isso, utilizar texto, diagramas e figuras. A solução pode conter os seguintes componentes:

Estrutura: descreve a estrutura básica da solução, geralmente através de diagramas de classes e de seqüência.

Participantes: descrevem as classes, objetos ou componentes participantes do padrão e suas responsabilidades.

Dinâmica: descreve o comportamento dinâmico do padrão.

Implementação: descreve detalhes de implementação do padrão.

Variantes: apresentam possíveis variações e especializações da solução.

- **Contexto resultante:** descreve o estado ou configuração do sistema, pós-condições e efeitos colaterais, após a aplicação do padrão. Em alguns formatos de padrões vem descrito como **Conseqüências**.
- **Padrões relacionados:** descrevem os relacionamentos estáticos e dinâmicos do padrão com outros padrões dentro da mesma linguagem ou sistema de padrões.
- **Usos conhecidos:** descrevem ocorrências conhecidas do padrão e suas aplicações em sistemas existentes. Isto ajuda a validar o padrão, verificando se ele é realmente uma solução provada para um problema recorrente.
- **Intenção**: descreve uma resposta rápida para as seguintes questões: o que o padrão faz? Qual a sua intenção? Que problema em particular o padrão se propõe a solucionar?

A seguir são descritos alguns componentes opcionais. Os componentes opcionais podem ser usados para comunicar informações importantes que não foram cobertas pelos componentes essenciais:

- **Também conhecido como**: descreve outro nome pelo qual padrão é conhecido, caso exista.
- **Motivação**: descreve um cenário que ilustra o problema e a forma com que as classes e os objetos contidos na estrutura do padrão resolvem o problema.
- **Aplicabilidade**: descreve em que situação o padrão pode ser aplicado.
- **Raciocínio** *(Rationale)*: descreve como o padrão funciona, porque funciona e qual a sua utilidade. Mostra a importância dos princípios atrás do padrão.
- ***Sketch*:** descreve, através de desenhos ou diagramas, como o padrão funciona ou como seus componentes se relacionam.

- **Exemplo**: fragmentos de código, diagramas, figuras, etc., que ilustram como o padrão deve ser implementado.

Existem diversos formatos de padrões. Entre eles podemos citar os formatos do Alexander (ALEXANDER *et al.*, 1977), Gang of Four (GoF) (GAMMA *et al.*, 1995), POSA (BUSCHMANN *et al.*, 1996), Coplien (COPLIEN, 1996), dentre outros.

As seções utilizadas no formato de Alexander (ALEXANDER *et al.*, 1977) não são fortemente delimitadas sintaticamente. A única estrutura sintática utilizada é o termo "*Therefore"* que precede a **Solução** (COPLIEN, 1996). Cada padrão inicia com uma figura que mostra um exemplo do padrão. Em seguida, após a figura, tem-se um parágrafo introdutório que define o contexto do padrão. Dando seqüência, existem diamantes que demarcam o inicio e o fim do problema e da solução. Padrões com dois diamantes são aqueles em que Alexander tem maior confiança, pois possuem fundamentos empíricos. Os padrões com poucos diamantes podem ter um significado social forte, mas são mais especulativos. E, finalmente, um parágrafo que relaciona o padrão aos demais padrões da linguagem.

Nos formatos do GoF (GAMMA *et al.*, 1995), Coplien (COPLIEN, 1996) e POSA (BUSCHMANN *et al.*, 1996), as seções do padrão são bem delimitadas. No entanto, cada formato possui um conjunto de componentes distinto. O formato do GoF inclui os seguintes componentes: **Nome**, **Classificação**, **Intenção**, **Também Conhecido Como**, **Motivação**, **Aplicabilidade**, **Estrutura**, **Participantes**, **Colaborações**, **Conseqüências**, **Implementação**, **Exemplo**, **Usos Conhecidos e Padrões Relacionados**. O formato de Coplien possui: **Nome**, **Problema**, **Contexto**, **Forças**, **Solução**, **Rationale** e **Contexto Resultante**. E, por último, o formato de POSA possui: **Nome**, **Também Conhecido Como**, **Exemplo**, **Contexto**, **Problema**, **Solução**, **Estrutura**, **Dinâmica**, **Implementação**, **Exemplo Resolvido**, **Variantes**, **Usos Conhecidos**, **Conseqüências** e **Veja Também**.

O formato utilizado neste trabalho possui os seguintes componentes: **Nome**, **Contexto**, **Problema**, **Forças**, **Solução**, **Estrutura**, **Exemplo**, **Participantes**, **Dinâmica**, **Conseqüências**, **Variantes (quando aplicável)**, **Padrões Relacionados** e **Usos Conhecidos**. Nos componentes **Estrutura** e **Dinâmica**, os diagramas são representados em UML (RUMBAUGH *et al.*, 2000). Os padrões de requisitos e testes não possuem os componentes **Participantes** e **Dinâmica**, pois estes não são relevantes para os padrões de requisitos descritos.

3.5 Classificação de padrões

Nesta seção, iremos apresentar uma visão geral sobre classificação de padrões de *software*. Porém, antes de falarmos de classificação, é importante citar aqui o uso dos termos *classificação* e *categoria.* Para esta dissertação, o termo *classificação* corresponde ao "ato ou processo de classificar" ou "a distribuição por classes", conforme definido por Santos (2004). O termo *categoria* é empregado quando falamos de uma classe particular, cujo conjunto define uma *classificação*. Em resumo, o conjunto de *categorias* às quais um padrão pertence define sua *classificação*. Desta forma, para estudarmos uma determinada *classificação*, precisamos estudar as *categorias* que a definem. A mesma terminologia é utilizada em POSA (BUSCHMANN *et al.*, 1996), onde as *categorias de padrões* e *categorias de problemas* são os critérios que definem o seu esquema de classificação.

Claramente, existem muitas maneiras de organizar os padrões. Inicialmente, os padrões foram classificados pela comunidade de padrões, de acordo com a camada de abstração do padrão, como idiomas, padrões de projeto e *frameworks* (COPLIEN, 1996). Entretanto, esse esquema de classificação mostrou-se limitado e ambíguo. Assim, surgiram outras formas de classificação de padrões que serão descritas a seguir.

3.5.1 Classificação segundo a fase do ciclo de vida de desenvolvimento do *software*

Segundo Andrade (2001), uma classificação bastante usada para padrões de *software* é a que classifica os padrões de acordo com a fase do ciclo de vida de desenvolvimento de *software* em que o padrão é aplicado. Desta forma, os padrões podem pertencer às seguintes categorias: padrões de requisitos (ROBERTSON, 1996; KONRAD; CHENG, 2002), padrões de análise (FOWLER, 1997a; FOWLER, 1997b), padrões de projeto (BUSCHMANN *et al.*, 1996; GAMMA *et al.*, 1995), idiomas (COPLIEN, 2000; BECK, 1997) e padrões de testes (RISING, 2000).

Os padrões de requisitos documentam as necessidades dos usuários e o comportamento dos sistemas em um alto nível de abstração. O objetivo dos padrões de análise é construir um modelo de análise que represente as estruturas conceituais dos processos do negócio. Os padrões de projeto são utilizados para refinar os componentes ou relacionamentos entre eles, podendo ser usados durante toda a fase de projeto do *software*. Os idiomas, por sua vez, são orientações para codificar padrões de projeto em uma linguagem de programação específica. Já os padrões de testes são orientações para as atividades de testes, incluindo documentação, execução e divulgação dos resultados.

A Figura 3.1 apresenta a classificação dos padrões de *software* segundo a fase do ciclo de vida de desenvolvimento de *software*.

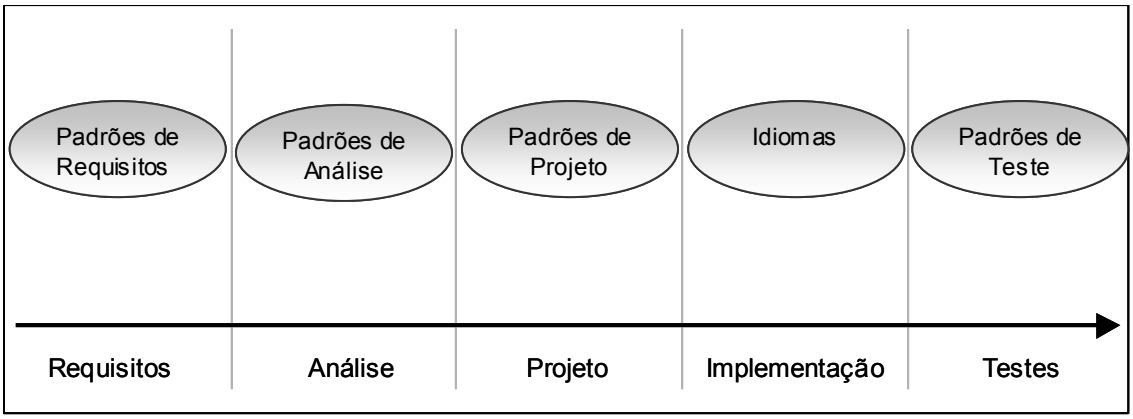


Figura 3.1: Classificação segundo a fase do ciclo de vida de desenvolvimento do *software* (adaptado de Santos (2004))

3.5.2 Classificação da GoF

GoF classifica os padrões de projeto de acordo com dois critérios. O primeiro critério, chamado Propósito, reflete o que o padrão faz. Quanto ao propósito, um padrão pode ser classificado como: Criação (*Creational*), Estrutural (*Strutural*), ou Comportamental (*Behavioral*). Os padrões de criação abrangem o processo de criação dos objetos. Os padrões estruturais lidam com a composição de classes ou objetos. Padrões Comportamentais definem como as classes ou os objetos interagem e distribuem responsabilidade.

O segundo critério, chamado Escopo, especifica se o padrão se aplica a classes ou a objetos. Padrões de Classes lidam com relacionamentos entre classes e suas subclasses. Padrões de Objetos lidam com os relacionamentos entre objetos, que podem ser alterados em tempo de execução e são mais dinâmicos.

3.5.3 Classificação POSA

Buschmann *et al.* (1996) classificam os padrões de acordo com dois critérios: categoria de padrões e categoria de problemas. Na categoria de padrões, estes podem ser classificados em:

- *Padrões arquiteturais*: expressam um esquema de organização estrutural fundamental para sistemas de *software*. Fornecem um conjunto de subsistemas predefinidos, especificam suas responsabilidades e incluem regras e guias para organizar o relacionamento entre os subsistemas.

- *Padrões de projeto*: padrões que fornecem um esquema para refinar subsistemas, componentes de um sistema de *software* ou os relacionamentos entre eles. Descrevem uma estrutura comumente recorrente de componentes comunicantes que resolvem um problema geral de projeto em um contexto particular.
- *Idiomas*: são padrões de baixo nível, específicos para uma linguagem de programação. Descrevem como implementar um aspecto particular de um componente.

Na segunda classificação, que toma por base o critério problema, os padrões podem pertencer às seguintes categorias:

- *Da "lama" à estrutura*: padrões que suportam uma decomposição adequada de um sistema de tarefas em subtarefas cooperativas.
- *Sistemas distribuídos*: padrões que fornecem infra-estrutura para sistemas que têm componentes localizadas em diferentes processos.
- *Sistemas interativos*: padrões que ajudam a estruturar sistemas com interação homem-computador.
- *Sistemas adaptáveis*: padrões que fornecem infra-estrutura para aplicações que possuem mudanças nos requisitos funcionais.
- *Decomposição estrutural*: padrões que suportam uma decomposição adequada de subsistemas e componentes complexos em parte cooperativas.
- *Organização de trabalho*: padrões que definem como os componentes colaboram entre si para fornecer um serviço complexo.
- *Controle de acesso*: padrões que guardam e controlam o acesso a serviços ou componentes.
- *Gerenciamento*: padrões para o tratamento de coleções de objetos, serviços e componentes homogêneos em sua plenitude.
- *Comunicação*: padrões que ajudam a organizar a comunicação entre componentes.
- *Tratamento de recursos*: padrões que ajudam a gerenciar componentes e objetos compartilhados.

Alguns padrões não podem ser atribuídos a uma única categoria de problemas por endereçarem vários problemas. São, portanto, atribuídos em mais de uma categoria de problemas.

3.5.4 Outras classificações

Outros critérios utilizados para a classificação de padrões são a classificação dos padrões, segundo o domínio ou a camada de aplicação. Na classificação segundo o domínio da aplicação, podemos ter padrões para Web, XML, Interface, Segurança de Sistemas, Redes, Banco de Dados, e Sistemas Distribuídos, entre outros.

O catálogo de padrões Core J2EE (ALUR *et al.*, 2002) classifica os padrões segundo a camada da aplicação e apresenta três categorias: Apresentação, Negócios e Integração. A camada de Apresentação trata toda a lógica relacionada com a apresentação dos dados. A camada de Integração trata a lógica relacionada com a integração do sistema com a camada de informação. A camada de Negócios trata a lógica central da aplicação.

Existem ainda algumas categorias de padrões que estão relacionadas a áreas que dão suporte ao desenvolvimento de sistemas. São exemplos dessas categorias: Gerenciamento de Configuração, Organizacional, Processo, Modelagem de Sistemas e Treinamento (AMBLER, 2005; COPLIEN, 1995; RISING, 2000).

3.6 Conclusão

Este capítulo abordou conceitos de padrões de *software* considerados mais relevantes para o desenvolvimento deste trabalho. No capítulo seguinte, serão apresentados o Catálogo de metapadrões e os padrões propostos neste trabalho.

Capítulo 4 Catálogo de metapadrões e padrões

Neste capítulo, serão apresentados os metapadrões e padrões que compõem o catálogo de metapadrões.

4.1 Introdução

Casos de uso que documentam as operações de manutenção (criação, consulta, atualização e remoção) são comumente conhecidos como casos de uso CRUD (*create, read, update e delete*). A documentação desses casos de uso tem sido uma questão polêmica: devemos documentar essas operações como um único caso de uso ou devemos documentá-las em casos de uso separados?

Segundo Cockburn (2001), documentar os casos de uso CRUD como casos de usos separados pode aumentar significativamente o número de casos de uso do sistema e, com isso, dificultar a gerência dos mesmos. Entretanto, Schneider e Winters (2001) definem que as operações CRUD devem ser documentadas em casos de uso separados, pois estas podem possuir atores distintos e diferentes níveis de segurança.

Övergaard e Palmkvist (2004) descrevem que documentar casos de uso CRUD em um único caso de uso tem as seguintes vantagens:

- O tamanho do modelo de casos de uso é reduzido, o que torna a gerência e a manutenção mais fáceis, devido ao pequeno número de casos;
- Agrupando as funcionalidades em um único caso de uso tem-se garantia de que todas elas estejam descritas, documentadas e que nenhuma foi esquecida; e
- O valor de cada funcionalidade separada é muito pequeno (ou inexistente) para os envolvidos. É o conjunto das funcionalidades de manutenção que tem valor para os usuários e desenvolvedores. Juntas, as funcionalidades formam uma unidade conceitual.

Adolph *et al.* (2002) descrevem uma linguagem de padrões para casos de uso. A linguagem contém mais de trinta padrões, que apresentam soluções simples e elegantes para os problemas mais comuns no desenvolvimento de casos de uso em geral. Esses padrões abordam o processo de desenvolvimento de casos de uso, a estrutura interna dos casos de uso e o relacionamento entre um conjunto de casos de uso de um sistema.

Os padrões estão organizados em duas categorias: padrões de desenvolvimento e padrões estruturais. Os padrões de desenvolvimento descrevem as melhores práticas para a escrita de casos de uso e os critérios para medir a qualidade do processo de escrita. Os padrões estruturais descrevem o comportamento básico dos casos de uso, explicando como estes devem ser organizados. Essas suas categorias são divididas em sub-categorias.

Os padrões de desenvolvimento são classificados em três subcategorias:

- *Organização da equipe*: padrões que definem aspectos para melhorar a forma de organizar a equipe que irá desenvolver os casos de uso;
- *Processo*: padrões que definem aspectos para melhorar a qualidade da metodologia que a equipe deve seguir para criar casos de uso; e
- *Edição*: padrões para melhorar a qualidade de casos de uso individuais de acordo com as mudanças de requisitos e o detalhamento do caso de uso.

Os padrões estruturais são classificados em quatro subcategorias:

- *Conjunto de casos de uso*: padrões para melhorar a qualidade de um conjunto de casos de uso;
- *Casos de uso*: padrões para melhorar a qualidade de um caso de uso individual;
- *Cenários e passos*: padrões para melhorar a qualidade dos cenários de casos de uso, e dos passos desses cenários; e
- *Relacionamento de casos de uso*: padrões para melhorar a qualidade da estrutura dos relacionamentos entre um conjunto de casos de uso.

Övergaard e Palmkvist (2004) descrevem um catálogo de padrões para criar casos de uso simples e fáceis de serem mantidos. Esses padrões focalizam-se nos projetos e nas técnicas usadas na criação de casos de uso de qualidade, e não em como modelar os casos de uso.

O catálogo de Övergaard e Palmkvist (2004) possui onze padrões, dentre os quais podemos destacar o padrão **CRUD**. A intenção deste padrão é juntar as funcionalidades de Criação, Consulta, Atualização e Exclusão de dados em um único caso de uso. Este padrão deve ser usado quando todos os fluxos contribuem para o mesmo valor de negócio e são todos pequenos e simples.

O Catálogo de Metapadrões e Padrões descrito neste trabalho é composto por:

- Quatro Metapadrões: **Meta-CRUD**, **Relatório**, **Transação** e **Assistente** (SOUZA *et al.*, 2005b);
- Quatro Padrões de Projeto: **CRUD-MVC** e **Registro com Busca** (SOUZA e PIRES, 2003); **Manutenção em Grade** e **Seleção com Secundária** (SOUZA e PIRES, 2004);
- Cinco Padrões de Requisitos: **Caso de Uso CRUD**, **Documentação de Atributos**, **Caso de Uso Relatório**, **Caso de Uso Transação** e **Caso de Uso Assistente** (SOUZA *et al.*, 2005a); e
- Quatro Padrões de Teste: **Teste CRUD**, **Teste Relatório**, **Teste Transação** e **Teste Assistente** (SOUZA *et al.*, 2005b).

Os padrões de requisitos são fundamentados no conceito de casos de uso. Esses padrões abordam soluções para problemas de especificação de requisitos funcionais considerando operações de manutenção, consulta, relatório e operações de transação. O padrão **Caso de Uso CRUD** adota a solução de que as operações de manutenção devam ser documentadas em um único caso de uso.

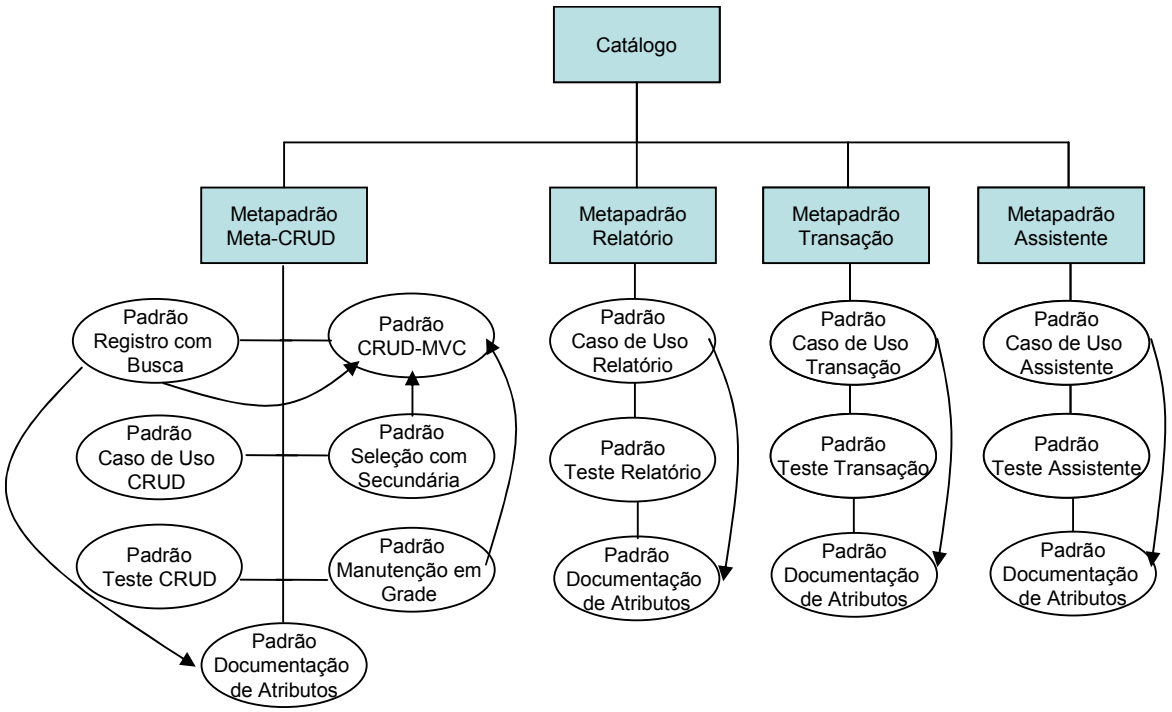


Figura 4.1: Relacionamento entre os metapadrões e os padrões proposto

A Figura 4.1 apresenta os relacionamento entre os metapadrões e os padrões propostos. Nessa figura, os retângulos representam os metapadrões e as elipses representam os padrões.

As setas (→) indicam que os padrões que se encontram na origem utilizam o padrão que se encontra no destino.

4.2 Metapadrão Meta-CRUD

Contexto

Projetos de desenvolvimento de sistemas de informação que requerem a criação dos produtos de trabalho gerados nas disciplinas de requisitos, análise e projeto, implementação e testes para operações de manutenção de entidades.

Problema

Como tratar a manutenção de entidades, nas diversas fases de um ciclo de vida de construção de *software*, de forma integrada e consistente?

Solução

Este metapadrão descreve diretrizes para manutenção de entidades. As entidades devem ser mantidas através de operações CRUD (criação, consulta, alteração e remoção). A estrutura dessas operações deve considerar o tipo de entidade, sua complexidade e o volume de dados tratados.

Padrões que implementam o metapadrão

- Caso de Uso CRUD;
- CRUD-MVC;
- Registro com Busca;
- Manutenção em Grade;
- Seleção com Secundária; e
- Teste CRUD.

4.2.1 Padrão Caso de Uso CRUD

Contexto

Este padrão é utilizado para a documentação dos requisitos de operações de manutenção em sistemas da informação, por meio do uso de modelos e especificações de casos de uso. Os requisitos de operações de manutenção são caracterizados por operações de Inclusão, Consulta, Alteração e Exclusão.

Problema

Como documentar os requisitos funcionais de inserção, atualização, exclusão e consulta de dados por meio de especificações de casos de uso?

Forças

- Todo caso de uso deve demonstrar um valor observável (RUP, 2003). Em alguns casos, o usuário identifica o valor observável como a manutenção da entidade. Em outros casos, o valor observável está nas operações individuais de Inclusão, Consulta, Alteração e Exclusão.
- As operações de manutenção podem ocorrer tanto sobre entidades simples com poucos atributos, como em entidades complexas com vários atributos e relacionamentos.
- As operações de inclusão, alteração, remoção e consulta devem ser tratadas e seus requisitos documentados. Esses requisitos incluem validação de atributos e regras de negócio.
- Os atributos mantidos de cada entidade devem ser documentados.
- Os requisitos documentados devem ser de fácil entendimento para os usuários e para a equipe de desenvolvimento.
- Uma grande quantidade de casos de uso dificulta a gestão dos requisitos e pode indicar a existência de decomposição funcional.

Solução

Organizar o fluxo de eventos do caso de uso no **Fluxo básico** e quatro subfluxos (**Incluir, Alterar, Remover e Consultar**), como se segue:

- O **Fluxo básico** descreve a condição de início e desvia o fluxo para um dos subfluxos, de acordo com as operações disponíveis: Incluir, Alterar, Excluir e Consultar. Condições de início indicam os eventos que provocam a execução do caso de uso. Por exemplo, a situação em que a entidade deve ser mantida, se existe alguma periodicidade requerida ou alguma questão de permissão de acesso.
- Cada subfluxo descreve o cenário operacional de uma das funcionalidades: Incluir, Alterar, Remover e Consultar.

- O Subfluxo **Incluir** apresenta os atributos para a inclusão e descreve o comportamento dessa inclusão.
- O Subfluxo **Alterar** apresenta os atributos atualizáveis, exibe seus valores e descreve o comportamento da atualização. Se os atributos atualizáveis forem os mesmos apresentados no Subfluxo **Incluir**, pode-se referenciar este subfluxo.
- O Subfluxo **Remover** descreve o comportamento da remoção e documenta as restrições da exclusão da entidade. Por exemplo, se há alguma regra de negócio que deve ser acionada ou se uma confirmação para a exclusão é exigida.
- O Subfluxo **Consultar** documenta requisitos para localização da entidade, que atributos devem ser filtros para a consulta, que atributos são obrigatórios e quais são exibidos no resultado.
- As validações de atributos e regras de negócio são documentadas em uma seção independente dos fluxos e subfluxos. Ver o padrão **Documentação de Atributo**. A decisão sobre o momento no qual as validações e regras são executadas fará parte do projeto do caso de uso. No entanto, se esse momento já for identificado como um requisito claro da aplicação, a regra ou a validação deve ser referenciada pelo subfluxo. As regras de negócio são o que, tipicamente, representam requisitos de cálculos e tratamento de relacionamentos com outras entidades. As validações, no entanto, têm a função de documentar o tratamento para a obrigatoriedade e formato de atributos (datas, limites numéricos, entre outros).

Estrutura

Fluxo básico

1. O caso de uso inicia quando o <*nome do ator*> necessita fazer a manutenção (inclusão, alteração, exclusão ou consulta) de uma <*nome da entidade*>. <*descrever a condição de início do caso de uso*>.
2. De acordo com o tipo de operação manutenção desejado pelo <*nome do ator*>, um destes subfluxos é executado:

   2.1. Se o <*nome do ator*> deseja incluir uma nova <*nome da entidade*> , o subfluxo Incluir <*nome da entidade*> é executado.

2.2. Se o <*nome da entidade*> deseja alterar informações de uma <*nome da entidade*> já cadastrada, o subfluxo Alterar <*nome da entidade*> é executado.

2.3. Se o <*nome do ator*> deseja excluir uma <*nome da entidade*> já cadastrada, o subfluxo Remover <*nome da entidade*> é executado.

2.4. Se o <*nome do ator*> deseja consultar informações sobre uma ou mais <*nome da entidade*> cadastradas, o subfluxo Consultar <*nome da entidade*> é executado.

Subfluxo Incluir <*nome da entidade*>

1. Este subfluxo inicia quando o <nome do ator> solicita incluir uma <nome da entidade>.
2. O sistema solicita ao <*nome do ator*> o preenchimento dos seguintes atributos:
   - <*lista de atributos*>.
3. O <*nome do ator*> preenche os atributos acima e confirma a inclusão.
4. O sistema realiza a inclusão dos dados informados pelo <*nome do ator*> no passo 3.
5. O sistema exibe uma mensagem informando que a inclusão da <*nome da entidade*> foi efetivada com sucesso.

Subfluxo Alterar <*nome da entidade*>

1. Este fluxo inicia quando o <nome do ator> solicita alterar uma <nome da entidade>.
2. O <*nome do ator*> seleciona uma única <*nome da entidade*>.
3. O sistema solicita a alteração dos seguintes atributos:
   - <*lista de atributos que podem ser alterados*>.
4. O <*nome do ator*> altera os dados desejados e confirma a alteração.
5. O sistema realiza a alteração dos dados informados no passo 4.
6. O sistema exibe uma mensagem de confirmação informando que a alteração do <*nome da entidade*> foi efetivada com sucesso.

Subfluxo Remover <*nome da entidade*>

1. Este subfluxo inicia quando o <nome do ator> solicita remover uma ou mais <nome da entidade>.

2. O <*nome do ator*> seleciona quais <*nome da entidade*> deseja remover e solicita sua remoção.
3. O sistema solicita a confirmação para remoção.
4. O <*nome do ator*> confirma remoção.
5. O sistema remove os <*nome da entidade*> confirmados.
6. O sistema exibe uma mensagem informando que a remoção dos <*nome da entidade*> foi efetivada com sucesso.

Subfluxo Consultar <nome da entidade>

1. Este fluxo inicia quando o <nome do ator> solicita consultar <nome da entidade>.
2. O sistema solicita o preenchimento dos seguintes filtros:
   - <*lista de filtros*>.
3. O <*nome do ator*> preenche os filtros e solicita a consulta.
4. O sistema apresenta as seguintes informações dos <*nome da entidade*> obtidos na consulta:
   - <*lista de atributos*>.

Validações e regras de negócio

- *Atributos obrigatórios*: se algum atributo obrigatório não tiver sido preenchido, <*descrever que ações o sistema deve tomar*>. Por exemplo: "o sistema não completará a operação e notificará ao <*nome do ator*>, solicitando o preenchimento". Esta regra aplica-se a todos os subfluxos.
- *Atributos com valores não permitidos*: se algum atributo for preenchido com valor não permitido, <*descrever que ações o sistema deve tomar*>. Por exemplo, "o sistema não completará a operação e notificará ao <*nome do ator*>, solicitando o preenchimento". Esta regra aplica-se a todos os subfluxos.
- No subfluxo **Remover**, o sistema valida os <*nome da entidade*> selecionados de acordo com as seguintes regras:
  - <*regras de remoção*>.

Exemplo do Padrão

Este exemplo apresenta o Caso de uso *Manter Cliente* de uma aplicação de *Telemarketing*.

Subfluxo Incluir Cliente

1. Este subfluxo inicia quando o *Operador de Telemarketing* solicita incluir um *cliente*.
2. O sistema solicita ao *Operador de Telemarketing* o preenchimento dos seguintes atributos:
   - * Nome.
   - * Logradouro (descreve a rua ou a avenida em que o cliente reside).
   - * Número.
   - * Bairro.
   - * Cidade.
   - * Estado (campo de escolha fechada; valores possíveis: todas os estados cadastrados no sistema).
   - * CPF.
   - Sexo (campo de escolha fechada. Valores possíveis: feminino e masculino).
   - Data de ativação.
   - Data de desativação.
3. O *Operador de Telemarketing* preenche os atributos acima e confirma a inclusão.
4. O sistema realiza a inclusão dos dados informados pelo *Operador de Telemarketing* no passo 3.
5. O sistema exibe uma mensagem informando que a inclusão do *cliente* foi efetivada com sucesso.

Subfluxo Alterar Cliente

1. Este fluxo inicia quando o *Operador de Telemarketing* solicita alterar um *cliente*.
2. O *Operador de Telemarketing* seleciona um único *cliente*.
3. O sistema solicita a alteração dos atributos listados no passo 2 do Subfluxo Incluir.
4. O *Operador de Telemarketing* altera os dados desejados e confirma a alteração.
5. O sistema realiza a alteração dos dados informados no passo 4.

6. O sistema exibe uma mensagem de confirmação informando que a alteração do *cliente* foi efetivada com sucesso.

Subfluxo Remover Cliente

1. Este subfluxo inicia quando o *Operador de Telemarketing* solicita remover um ou mais *clientes*.
2. O *Operador de Telemarketing* seleciona quais *clientes* deseja remover e solicita a remoção.
3. O sistema solicita a confirmação para a remoção.
4. O *Operador de Telemarketing* confirma a remoção.
5. O sistema remove os *clientes* confirmados.
6. O sistema exibe uma mensagem informando que a remoção dos *clientes* foi efetivada com sucesso.

Subfluxo Consultar Cliente

1. Este fluxo inicia quando o *Operador de Telemarketing* solicita consultar *clientes*.
2. O sistema solicita o preenchimento dos seguintes filtros:
   - Nome.
   - CPF.
3. O *Operador de Telemarketing* preenche os filtros e solicita a consulta.
4. O sistema apresenta as seguintes informações dos *clientes* obtidos na consulta:
   - Nome.
   - Logradouro.
   - Número.
   - Bairro.
   - Cidade.
   - Estado.
   - CPF.
   - Sexo.
   - Data de ativação.
   - Data de desativação.

Validações e regras de negócio

- *Atributos obrigatórios*: se algum atributo obrigatório não tiver sido preenchido, o sistema não completará a operação e notificará ao *Operador de Telemarketing*, informando quais campos obrigatórios não foram preenchidos e solicitando o preenchimento dos mesmos. Esta regra aplica-se a todos os subfluxos.
- *Atributos com valores não permitidos*: se algum atributo for preenchido com valor não permitido, o sistema não completará a operação e notificará ao *Operador de Telemarketing*, informando quais campos foram preenchidos com valores inválidos e solicitando o preenchimento correto. Esta regra aplica-se a todos os subfluxos.
- No subfluxo **Remover**, o sistema valida os clientes selecionados de acordo com as seguintes regras:

  Cliente que tiver algum chamado em aberto não poderá ser removido.

Conseqüências

- As operações de manutenção e seus requisitos são documentados de forma padronizada e estruturada para os diversos tipos de entidade, melhorando o entendimento do comportamento e dos requisitos e facilitando o desenvolvimento de produtos de trabalho das fases seguintes. Como por exemplo, análise, projeto e casos de teste;
- As validações e regras de negócio são documentadas de maneira estruturada, evitando omissões e destacando sua importância;
- Os atributos e as informações requeridos em cada operação são documentados, facilitando o entendimento da estrutura do sistema, como também a modelagem de dados e prototipação de telas;
- Fornece suporte ao conceito de caso de uso definido (RUP, 2003): “todo caso de uso deve demonstrar um valor observável”. A solução utiliza o conceito de subfluxos para agrupar em um único caso de uso as operações de Inclusão, Consulta, Alteração e Exclusão; e
- Reduz o número de casos de uso do sistema por meio do agrupamento da especificação das operações de manutenção em um único caso de uso, facilitando a gestão dos requisitos.

Padrões relacionados

- Padrão **Documentação de Atributos**:

  Utilizado no subfluxo **Inserir** para listar os atributos da entidade; no subfluxo **Alterar**, para descrever os atributos que podem ser alterados; e no subfluxo **Consultar**, para descrever os filtros e atributos que serão exibidos no resultado da consulta.

Usos conhecidos

Por motivos de confidencialidade, mais detalhes dos usos conhecidos abaixo não podem ser fornecidos.

- Sistema imobiliário;
- Sistema de portal Web para administração e publicação de informações de acervos culturais;
- Sistema de Telemarketing; e
- No RUP (2003), os exemplos de casos de uso CRUD seguem estrutura similar à proposta no padrão **Caso de Uso CRUD**.

4.2.2 Documentação de Atributos

Contexto

Em sistemas de informação, os atributos das entidades possuem diversas características como: nome, descrição, obrigatoriedade, validações, semântica, entre outras. Portanto, a documentação desses atributos deve ser elaborada de forma que essas características sejam bem explicitadas.

Problema

Como definir e documentar de forma padronizada os diversos atributos das entidades, informações tão necessárias durante operações CRUD?

Forças

- Atributos podem ser de tipos primitivos, enumerados, multivalorados ou de relacionamentos. Os atributos enumerados podem assumir um valor dentro de um domínio fixo de valores. Os atributos de relacionamentos podem assumir como valor uma referência para outras entidades cadastradas no sistema. Os atributos

multivalorados podem assumir um ou mais valores referentes a outras entidades cadastradas no sistema.

- Os atributos de entidades podem fazer parte de um conjunto de parâmetros ou filtros de consulta.
- Alguns atributos podem ser opcionais e outros obrigatórios. Atributos obrigatórios devem ter tratamento adequado em caso de não preenchimento na inclusão, alteração ou consulta.
- Se os atributos não forem documentados com as informações necessárias, os seguintes problemas poderão ocorrer: (i) dificuldade na validação dos requisitos com o usuário final por falta de informações sobre os atributos; e (ii) inconsistência nos produtos de trabalho gerado ao longo do ciclo de vida.

Solução

- Documentar os atributos como uma lista itemizada associada a uma operação de consulta, inclusão ou alteração. No caso da alteração, se os atributos que podem ser alterados forem os mesmos da inclusão, pode-se apenas fazer uma referência aos atributos listados na inclusão.
- Fornecer, quando necessário, uma descrição breve do atributo.
- Marcar com um caractere especial os atributos obrigatórios (“*”, por exemplo).
- Para atributos que indicam relacionamento, indicar que é um campo de escolha fechada e indicar a fonte origem dos dados de escolha. Por exemplo: Unidade federativa (campo de escolha fechada. Valores possíveis: todas as unidades federativas cadastradas no sistema).
- Para atributos enumerados, indicar que é um campo de escolha fechada e indique os valores possíveis. Por exemplo: Sexo (campo de escolha fechada. Valores possíveis: feminino e masculino).
- Para atributos multivalorados, indicar que é um campo de escolha múltipla e indicar a fonte de origem dos dados de escolha.
- Alguns atributos possuem restrição quanto aos valores aceitos. Neste caso, deve-se documentar esta restrição juntamente com o atributo.

Estrutura

- <atributo>. <descrição do atributo>
- <caractere>. <atributo obrigatório>
- <atributo> (Campo de escolha fechada. Valores possíveis: <entidade origem dos dados>). <descrição do atributo>
- <atributo> (Campo de escolha fechada. Valores possíveis: <valor 1>, <valor 2>, ... <valor n>). <descrição do atributo>
- <atributo> (Campo de escolha múltipla. Valores possíveis: <entidade origem dos dados>). <descrição do atributo>
- <atributo>. <descrição da validação de valores aceitos>

Exemplo do Padrão

Exemplo de atributo com descrição:

- Logradouro. Descreve a rua ou a avenida em que o cliente reside.

Exemplo de atributo obrigatório:

- * Nome.

Exemplo de atributo de relacionamento:

- Estado (campo de escolha fechada. Valores possíveis: todos os estados cadastrados no sistema).

Exemplo de atributo enumerado:

- Sexo (campo de escolha fechada. Valores possíveis: feminino e masculino).

Exemplo de atributo multivalorado:

- Autor do livro (campo de escolha múltipla. Valores possíveis: todos os autores cadastrados no sistema).

Exemplo de atributo com restrição de valores:

- Temperatura corpórea do paciente. Só poderá assumir valores entre 35 e 42 graus.

Conseqüências

- Os diversos tipos de atributos são documentados de forma simples e padronizada.

- Os atributos obrigatórios são declarados claramente, facilitando sua identificação e tratamento da implementação e testes.

Padrões relacionados

- Não se aplica.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.2.3 Padrão CRUD-MVC

Contexto

Sistemas de informação requerem funcionalidades de negócio implementadas através de IHC (interface humano-computador), componentes para tratamento de regras de negócio e acesso a dados para operações CRUD e operações de negócio.

Problema

Como tratar funcionalidades recorrentes de criação, consulta, atualização e exclusão de dados em sistemas de informação considerando aspectos de apresentação e tratamento de eventos, regras de negócio e persistência, isto é, no modelo, visão, e controle (MVC)?

Forças

- Deve fornecer baixo acoplamento, facilitando a troca de mecanismos de interface e acesso a dados; e
- Deve permitir reuso de estrutura e comportamentos de componentes de IHC, e tratamento de eventos CRUD.

Solução

A idéia básica da solução é criar uma camada de classes abstratas que capturam a estrutura, o comportamento e as colaborações genéricas entre modelo, visão e controle; e criar várias implementações para cada cenário de uso CRUD, herdando as características da camada abstrata e definindo as características concretas da aplicação. O projetista deve utilizar o padrão definindo as características gerais de interface, negócio e persistência na camada abstrata.

As classes abstratas de visão determinam operações para tratamento de eventos CRUD com enfoque em questões de interface com usuário. Por exemplo, a operação *tratarExclusao* pode sempre exibir uma interface de diálogo para confirmar a operações antes de acionar o controlador.

Outros eventos de interface com usuário e outras características que possam ser reutilizadas também são de responsabilidade das classes abstratas de visão. As classes abstratas de controle permitem a fabricação de entidades, tratamento de eventos CRUD e também suportam outras operações genéricas que o projetista possa definir. O tratamento de eventos CRUD nas classes de controle é voltado para o fluxo de negócio, ao contrário do tratamento de eventos CRUD nas classes de visão. A classe abstrata de entidade permite definir as operações CRUD propriamente ditas ou outras operações genéricas de entidades. As classes concretas implementam as operações definidas na camada abstrata e complementam comportamentos e características específicas de negócio.

Estrutura

O padrão apresenta uma camada de classes abstratas para visão, modelo e controle. As classes abstratas utilizam métodos de fabricação definidos de acordo com o padrão ***Factory Method*** (GAMMA *et al*, 1995), permitindo, com isso, que as subclasses concretas determinem suas implementações. A Figura 4.2 apresenta o diagrama de classes do padrão e seus componentes.

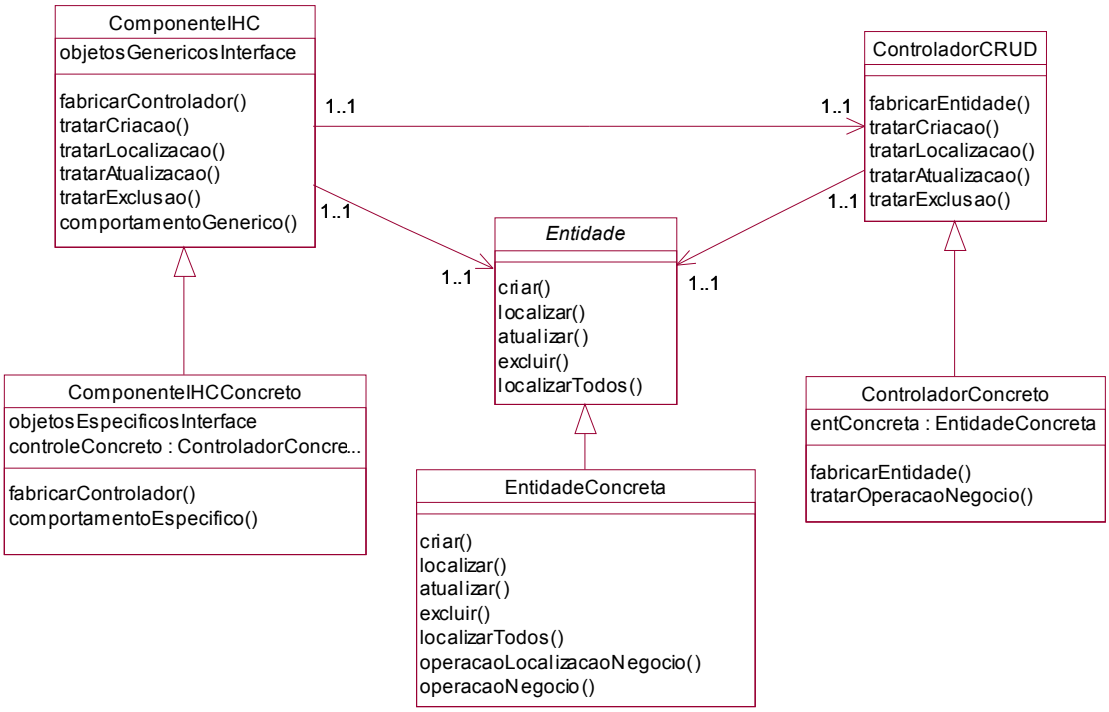


Figura 4.2: Diagrama de classes do Padrão CRUD-MVC

A estrutura do padrão permite que classes do nível abstrato possam implementar as interações entre visão, modelo e controlador, como também as operações CRUD que são

reutilizadas por todas as classes concretas da aplicação. Os relacionamentos do padrão **MVC** (modelo, visão e controle) (BUSCHMANN *et al.*, 1996) existem em nível das classes abstratas (para tratamento de eventos CRUD) e no nível das classes concretas (para tratamento de eventos de negócio específicos).

A classe de visão notifica *ControladorCRUD* sobre eventos de criação, alteração, exclusão e consulta de entidades de negócio. Por sua vez, *ControladorCRUD* é uma classe de controle que define os mecanismos para a criação da entidade de negócio e as operações reutilizáveis para o tratamento dos eventos CRUD, sobre uma interface genérica, para uma entidade de negócio. *Entidade* é uma classe de modelo que define a interface genérica de entidade de negócio, com contratos para as operações CRUD.

Para cada uma das classes genéricas podem existir várias implementações concretas de acordo com as diversas entidades de negócio do sistema. As implementações concretas especializam as classes genéricas, definindo características e comportamentos concretos de IHC (*ComponenteIHCConcreto*), fabricando controladores concretos (*ControladorConcreto*) que criam entidades concretas (*EntidadeConcreta*) e implementam tratamentos de outros eventos de negócio (que não operações CRUD). As entidades concretas definem os atributos (informação) e implementam as operações CRUD, além de outras operações de negócio.

Participantes

- *ComponenteIHC*
    - Pode ser formado por vários componentes de interface com usuário;
    - Define as características de interação com usuário (*objetosGenericosInterface*) presentes em todo componente de IHC de um sistema. Pelo menos, comandos para criação, localização, atualização e exclusão. Pode definir outras características a serem reutilizadas (por exemplo, ajuda e comando para fechar/ocultar interface);
    - Define o comportamento existente em toda interface de usuário (*comportamentoGenerico*). Por exemplo, controle de acesso a operações e alertas;
    - Fornece método *Factory* (GAMMA *et al.*, 1995) *fabricarControlador* para criação do objeto de controle e armazena referência através de interface *ControladorCRUD*; e
    - Fornece métodos genéricos para tratamento dos eventos de interface com usuário (*tratarCriacao, tratarLocalizacao, tratarExclusao*). Os métodos de evento notificam

seu observador (*ControladorCRUD*) das ações tomadas pelo usuário. Os métodos genéricos permitem definir ações de pré-processamento e pós-processamento das operações de criação, localização, atualização e exclusão. Por exemplo, antes de solicitar exclusão, a interface pode exibir um diálogo de confirmação de operação (pré-processamento de exclusão).

- *ComponenteIHCConcreto*
    - Define os campos de interface com o usuário para visualização e preenchimento das informações da entidade de negócio. Define também como os campos serão sincronizados com os atributos de *EntidadeConcreta*;
    - Define as características específicas de interação com usuário para a entidade de negócio: comandos para ações de negócio, pastas para manter relacionamentos com outras entidades e notificação de eventos de negócio para o *ControladorConcreto*; e
    - Implementa o método *fabricarControlador*, criando *ControladorConcreto* para implementar *ControladorCRUD*.
- *ControladorCRUD*
    - Fornece método *Factory fabricarEntidade* para criação da entidade de negócio; e
    - Define as operações para tratamento de eventos: *tratarCriacao, tratarLocalizacao, tratarAtualizacao, tratarExclusao* que operam sobre a interface genérica (*Entidade*) implementada pelo objeto criado pelo método *fabricarEntidade*.
- *ControladorConcreto*
    - Implementa método *Factory fabricarEntidade* para criação da entidade de negócio (*EntidadeConcreta*); e
    - Define as operações para tratamento de eventos de negócio.
- *Entidade*
    - Define interface genérica para entidade de negócio com operações para criação, localização, atualização e exclusão; e
    - Pode ser composta por objetos que concentrem a informação e objetos para acesso ao repositório de dados. No entanto, fornece interface única para o controlador. Três padrões podem ser combinados para gerar a entidade de negócio: **DAO (*Data Access Object)*** (ALUR *et al.*, 2002), **VO (*ValueObject*)** (ALUR *et al.*, 2002) e ***Facade***

(GAMMA *et al.*, 1995). Assim, utiliza-se um ***VO*** representando a informação, um **DAO** responsável por realizar a persistência do ***VO*** e um objeto ***Facade*** para coordenar o **VO** e o **DAO**, expondo para os clientes uma interface de negócio unificada: atributos, métodos do ciclo de vida – criar, atualizar, excluir, consultarPorId, consultarTodos e métodos de negócio. A fachada seria a *Entidade* propriamente dita.

- *EntidadeConcreta*
  - Implementa a interface de *Entidade* para os métodos de criação, localização, atualização e exclusão;
  - Define os atributos que são sincronizados com os campos de *ComponenteIHCConcreto*;
  - Define e implementa outros métodos de pesquisa; e
  - Define e implementa métodos de negócio utilizados por *ControladorConcreto*.

Dinâmica

A dinâmica do padrão é exemplificada na Figura 4.3, que apresenta o tratamento para o evento *criar*. Os outros eventos ocorrem de maneira análoga.

A seqüência de eventos abaixo descreve a dinâmica do padrão:

1. O usuário aciona algum comando CRUD em *ComponenteIHCConcreto*;
2. Se evento de atualização ou inclusão, a informação é sincronizada de *ComponenteIHCConcreto* para *EntidadeConcreta*;
3. O comportamento (métodos de tratamento) herdado de *ComponenteIHC* é executado, realizando algum pré-processamento e enviando uma notificação sobre o evento para o *ControladorConcreto*;
4. Em *ControladorConcreto*, o método adequado para tratamento do evento (*tratarCriacao*, *tratarLocalizacao*, *tratarAtualizacao*, ou *tratarExclusao*) é executado pelo código herdado de *ControladorCRUD*, executando o método CRUD adequado em *EntidadeConcreta* (através da interface de Entidade);
5. O método CRUD de *EntidadeConcreta* é executado e o controle retorna para o código herdado de *ComponenteIHC*, permitindo pós-processamento; e

6. Se evento de localização, a informação é sincronizada de *EntidadeConcreta* para *ComponenteIHCConcreto*.

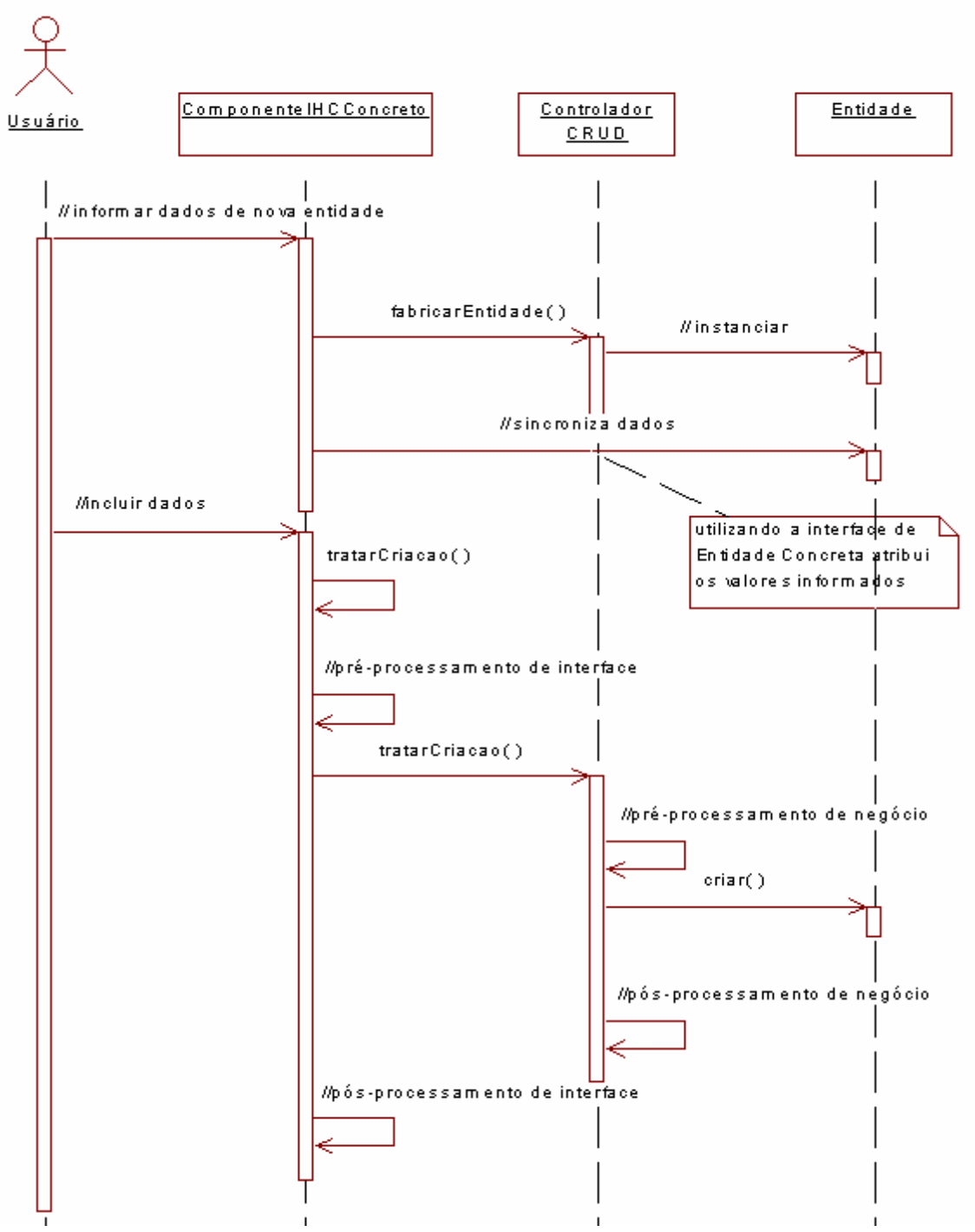


Figura 4.3: Diagrama de seqüência para o subfluxo de evento criar

Os eventos de negócio específicos das classes concretas são tratados de maneira similar, utilizando, no entanto, código implementado nas classes concretas.

Conseqüências

O padrão **CRUD-MVC** oferece os seguintes benefícios:

- A classe *ComponenteIHC* permite padronização e reuso de interface de usuário e de tratamento de eventos CRUD através de herança;
- Separação de código entre eventos CRUD e outros eventos de negócio;
- Permite a rápida extensão/manutenção de recursos de interface de usuário e tratamento de eventos através da centralização em classes abstratas que servem de base para o restante das classes da aplicação;
- Permite independência da interface de usuário com relação ao mecanismo de persistência através da interface de persistência de entidade; e
- Permite a troca de componentes de IHC sem afetar o restante do código.

As seguintes desvantagens merecem ser mencionadas:

- Apesar de separar aspectos de apresentação, controle, regras de negócio e persistência, o número de classes da aplicação pode aumentar bastante com o uso do padrão. Se essa separação não for muito crítica na aplicação, as variantes V1 e V2 podem ser consideradas; e
- Com a utilização deste padrão, a complexidade da solução tende a aumentar, porque envolve o relacionamento e a comunicação entre vários objetos.

Variantes

De acordo com as características da linguagem utilizada na implementação do sistema ou requisitos não-funcionais, o padrão pode ser utilizado com as seguintes variantes:

V1. Componente IHC e Controlador unificados

A unificação dessas duas classes aumenta o acoplamento, mas simplifica o projeto, reduz o número de classes e ainda continua fornecendo características importantes, como reutilização do tratamento de eventos e padronização de comportamento e componentes de interface. Nesse caso, os tratamentos dos eventos com relação à interface e regras de negócio estariam juntos na classe *ComponenteIHC* e as respectivas classes concretas também seriam unificadas. Algumas regras de negócio do controle poderiam eventualmente passar para a entidade.

V2. Simplificação da Entidade

A separação da entidade em ***Facade***, **VO** e **DAO** oferece uma boa separação entre operações de negócio, persistência, acesso e leitura de dados, mas aumenta o número de classes do sistema. Se essa separação não for importante para a aplicação, pode-se utilizar apenas uma classe de entidade, que tenha os atributos com seus métodos para leitura e escrita, os métodos de negócio relacionados aos dados, e o código para a persistência e a comunicação com o repositório de dados, todos juntos.

V3. Utilização do padrão *Observer* (GAMMA *et al.*, 1995)

Algumas linguagens e *frameworks* (como Java *Swing*) trazem suporte natural para o padrão ***Observer***. Em outras linguagens, a implementação do padrão pode ser mais complexa. No entanto, o uso do padrão ***Observer*** pode sofisticar a implementação do padrão **MVC**, diminuindo o acoplamento entre visão, controlador e modelo.

Há dois tipos de estratégias **MVC** que podem ser adotadas com o padrão ***Observer***: Modelo Ativo e Modelo Passivo. No Modelo Ativo, a classe *Entidade* seria observada por *ComponenteIHC* através da interface *Observable*, e qualquer mudança na *Entidade* geraria uma notificação de *ComponenteIHC* através da interface *Observer*. Com o Modelo Passivo, o *ControladorCRUD* é que seria observado por *ComponenteIHC*, e quando fizesse alterações na *Entidade* notificaria *ComponenteIHC*.

Padrões relacionados

- ***Factory Method*** (GAMMA *et al.*, 1995):

  Utilizado na criação de objetos de Entidade e de Controle.

- **MVC**:

  Utilizado para fornecer baixo acoplamento entre os componentes da aplicação.

- **DAO**, **VO** e **Facade**:

  Estes três padrões são utilizados para a construção de Entidade. Os padrões **DAO** e **VO** são bastante utilizados em aplicações J2EE, mas podem ser utilizados em qualquer tipo de linguagem orientada a objetos.

Usos conhecidos

Por motivos de confidencialidade, na organização onde este padrão foi implementado, mais detalhes dos usos conhecidos abaixo não podem ser fornecidos.

- Sistema imobiliário;
- Sistema de controle de processos jurídicos;
- Sistema de administração de recursos humanos;
- Sistema de *Telemarketing*; e
- Sistemas bancários.

4.2.4 Padrão Registro com Busca

Contexto

Utilizado para as principais entidades de negócio com muitos campos e/ou relacionamentos. Em geral, essas entidades são objetos complexos com muitos atributos e relacionamentos. O enfoque do cenário é de uma busca eficiente, seguida de visualização e edição de uma entidade.

Problema

Como criar componentes de cadastro e manutenção de entidades de negócio complexas, atendendo a requisitos de interface humano-computador, permitindo a reutilização de interação com usuário e estrutura, comuns aos vários tipos de entidades complexas existentes em uma aplicação de sistema de informação?

Forças

- Deve suportar reuso de características de interação com usuário, recorrentes na manutenção de entidades complexas;
- Deve suportar manipulação individual do objeto, com busca eficiente de entidades de negócio;
- Deve permitir fácil visualização de dados e relacionamentos de uma entidade complexa; e
- Deve fornecer baixo acoplamento entre interfaces de usuário para entrada e consulta de dados, tratamento de eventos CRUD e entidade de negócio.

Solução

Utilizar classe *Registro* para criação, atualização e exclusão de entidades, permitindo que, por vez, uma única entidade seja visualizada e editada. A classe *Registro* apresenta a

informação e os relacionamentos da entidade. Ela também fornece um mecanismo de localização eficiente, colaborando com a classe *Busca*. Por sua vez, a classe *Busca* fornece critérios de pesquisa e lista de resultados. Além de buscar a entidade complexa, ela pode ser utilizada para buscar entidades para preenchimento de relacionamentos.

O padrão **Registro com Busca** utiliza o padrão **CRUD-MVC** (ver seção 4.2.2) para definir como as classes de modelo, visão e controle podem-se relacionar. Desta forma, o padrão mantém seu foco na interação com usuário na manutenção de entidades complexas e utiliza a estrutura de **CRUD-MVC** para fornecer características como extensibilidade e baixo acoplamento. As classes *Registro* e *Busca* funcionam de acordo com a classe *ComponenteIHC*. As classes *RegistroConcreta* e *BuscaConcreta* funcionam de acordo com a classe *ComponenteIHCConcreto*. As classes de controle e modelo são utilizadas também de acordo com o padrão **CRUD-MVC**. Assim, o padrão **Registro com Busca** apresenta uma solução de interação com usuário para cenários de manutenção de entidades complexas, utilizando o padrão **CRUD-MVC** para estruturar suas classes.

Estrutura

A Figura 4.4 apresenta o diagrama de classes deste padrão e de seus componentes. As classes de controle e modelo não são apresentadas e seguem o padrão **CRUD-MVC**.

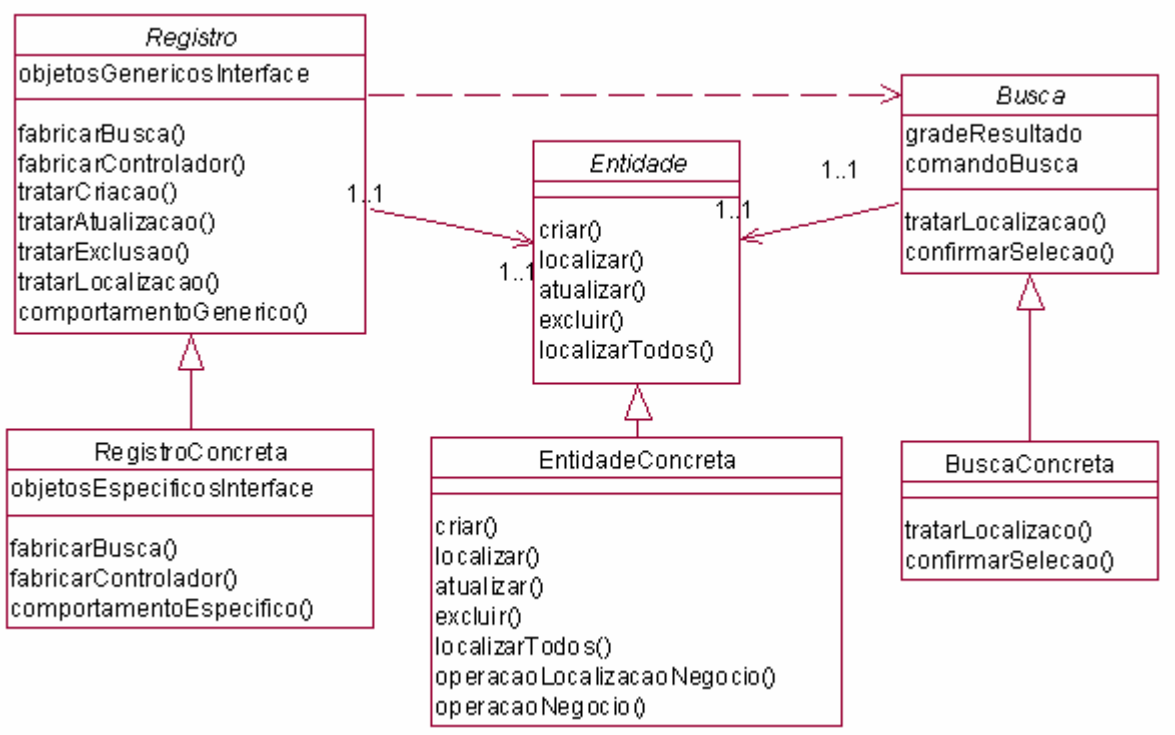


Figura 4.4: Diagrama de Classes do Padrão Registro com Busca

Participantes

- *Registro*
    - Define as características de interação com usuário presente em toda classe para manutenção de entidades complexas. Fornece um "botão" que pode acionar uma interface para a realização de pesquisas elaboradas sobre a entidade complexa;
    - Permite a edição, a criação e a exclusão de apenas uma entidade complexa por vez; e
    - Fornece método *Factory* (GAMMA *et al.*, 1995) *fabricarBusca* para criar uma instância de classe de *BuscaConcreta*. Após a criação, o objeto registro exibe a tela de busca.
- *Busca*
    - Fornece uma interface gráfica genérica para preenchimento de critérios de pesquisa e de apresentação de resultados;
    - Fornece botão para ação de pesquisa e grade para exibição de resultados; e
    - Executa comunicação genérica com o objeto Registro para seleção e apresentação da entidade de negócio.
- *RegistroConcreta*
    - Define os campos para visualização e preenchimento das informações da entidade de negócio complexa, e a forma como os campos serão sincronizados com os atributos de *EntidadeConcreta*;
    - Define as características específicas de interação com o usuário para a entidade de negócio: comandos para ações de negócio, pastas para a manutenção de relacionamentos com outras entidades e notificação de eventos de negócio para *ControladorConcreto.* As pastas de entidades relacionadas indicam relacionamentos *um-para-muitos* da entidade de negócio para entidades dependentes; e
    - Implementa o método *fabricarControlador*, criando *ControladorConcreto* para implementar *ControladorCRUD*.
- *BuscaConcreta*
    - Define os critérios concretos para a pesquisa;
    - Define os campos que são apresentados na grade de resultados;

- Faz a sincronização da coleção de entidades com a grade; e
- Pode ser utilizada para buscar entidades para preenchimento de campos de relacionamentos da entidade complexa.

- As classes *Entidade* e *EntidadeConcreta* foram apresentadas em participantes (seção 4.2.3).

Dinâmica

Os eventos de CRUD podem-se comportar de acordo com o padrão **CRUD-MVC**, sendo isto evidenciado no evento de localização. Após localizar a entidade, o usuário pode editá-la ou excluí-la. Já a criação pode ser realizada a qualquer momento. A dinâmica deste padrão é exemplificada na Figura 4.5.

Para localizar e visualizar uma entidade, o seguinte fluxo é executado:

1. Usuário dispara comando de localização;
2. O código herdado de Registro é executado, assim como o método *fabricarBusca*, que utiliza a implementação fornecida por *RegistroConcreta*. Após à criação do objeto de busca, o mesmo é disponibilizado para o usuário;
3. O usuário informa os parâmetros da pesquisa e solicita a localização;
4. O objeto *BuscaConcreta* notifica o *ControladorConcreto* para tratamento do evento de pesquisa;
5. O *ControladorConcreto* utiliza um método de negócio de pesquisa *localizar(params)*, da *EntidadeConcreta*, disponibilizando uma coleção de entidades para *BuscaConcreta*;
6. *BuscaConcreta* exibe alguns dados das entidades encontradas e o usuário seleciona a entidade que deseja visualizar ou editar; e
7. *BuscaConcreta* disponibiliza a entidade selecionada para *RegistroConcreta* que exibe seus dados.

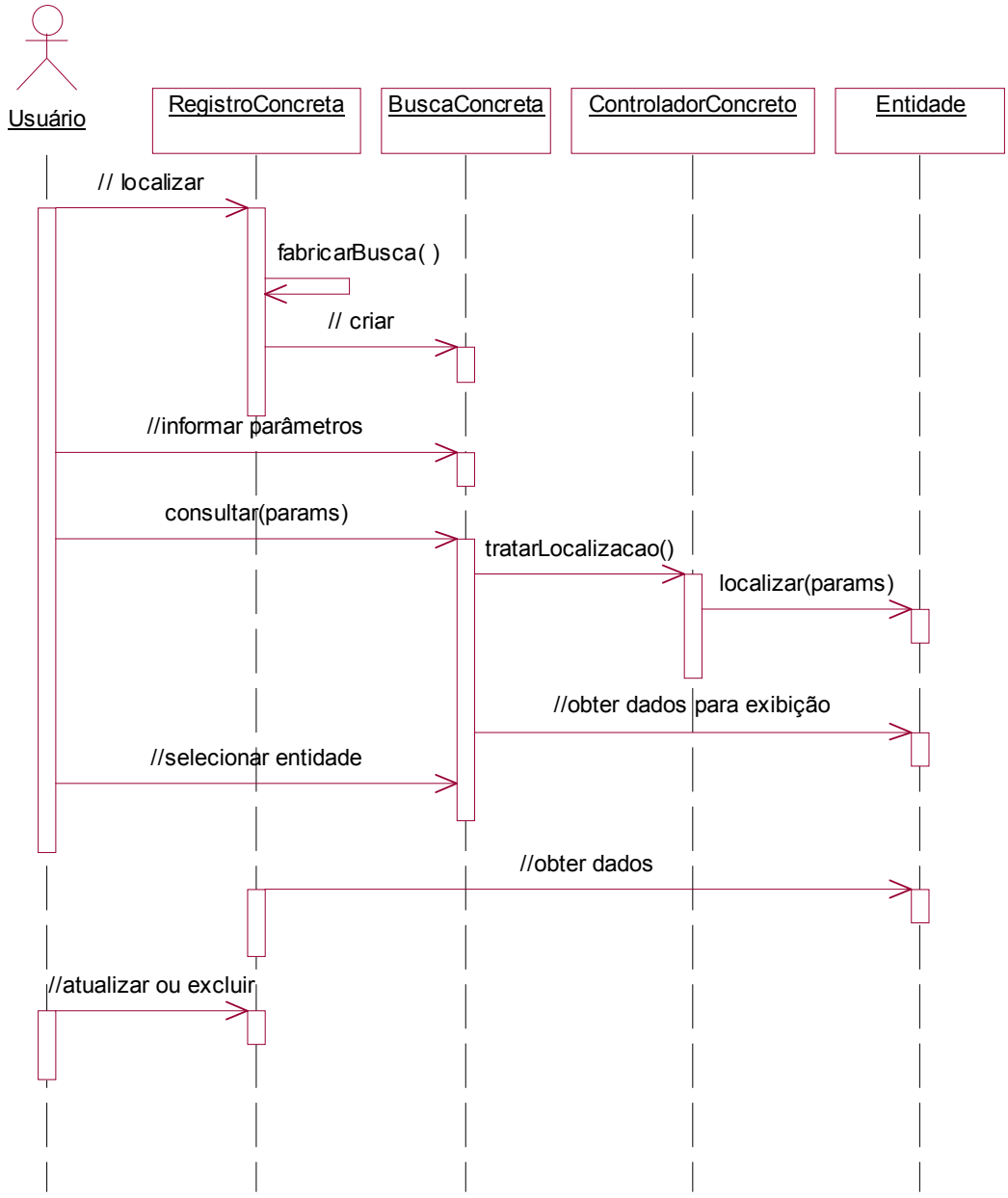


Figura 4.5: Diagrama de seqüência do padrão Registro com Busca

Conseqüências

O padrão **Registro com Busca** oferece os seguintes benefícios:

- Fornece um mecanismo eficiente de busca e visualização/edição de uma única entidade, aumentando, assim, a eficiência do sistema. Economiza recursos do sistema evitando alocação em memória de coleções de entidades complexas, pois estas consomem muitos recursos;

- Permite maior usabilidade do sistema, fornecendo para o usuário uma interface adequada para visualização e edição de entidade complexa;
- O reuso de *Registro* através de herança fornece interface de usuário padronizada para registro de entidades de negócio complexas; e
- Permite reuso de tratamento de eventos CRUD e interação com objeto de busca.

As seguintes desvantagens merecem ser mencionadas:

- Apesar de separar aspectos de apresentação, controle, regras de negócio e persistência, o número de classes da aplicação pode aumentar bastante com o uso do padrão; e
- Com a utilização do padrão, a complexidade da solução aumenta, envolvendo relacionamento e comunicação entre vários objetos.

Variantes

V1. Registro com Busca. O padrão **Registro com Busca** também pode ser utilizado sem a estrutura do padrão **CRUD-MVC**, considerando apenas seu principal foco, que é a interação com usuário para manutenção de entidades complexas. Esta situação pode ser implementada quando a aplicação não possuir fortes requisitos de extensibilidade e baixo acoplamento.

Padrões relacionados

- **Factory Method**:

  Utilizado na criação de objetos de Entidade e de Busca.

- **CRUD-MVC** (vide seção 4.2.3):

  Utilizado para definir interação entre as classes modelo, visão e controle.

Usos conhecidos

- Vide padrão **CRUD-MVC** (seção 4.2.3).

4.2.5 Padrão Manutenção em Grade

Contexto

Sistemas de informação utilizam-se de várias entidades básicas e simples para configuração do sistema. Essas entidades possuem poucos atributos sem objetos dependentes

e, também, um pequeno número de instâncias. Por exemplo: cadastro de unidades federativas, tipos de endereço, tipo de cliente, etc.

Entidades básicas são usadas em relacionamentos com entidades de negócio. Por exemplo, no momento de cadastrar um cliente é necessário informar a unidade federativa de seu endereço, o tipo de endereço e o tipo de cliente. A implementação de entidades básicas em sistemas de informação em geral não tem alcançado, de forma trivial, um bom grau de reuso e eficiência.

Problema

Como implementar funcionalidades de manutenção de entidades básicas de forma eficiente e reutilizável?

Forças

- A visualização geral das entidades existentes é essencial;
- A manipulação de entidades deve ser realizada sem níveis auxiliares de navegação;
- As operações de inclusão, alteração e exclusão devem ser realizadas em um conjunto de entidades; e
- As entidades devem suportar filtros.

Solução

Criar objetos genéricos de interface com usuário para manipulação de uma coleção de entidades de negócio em grade.

A grade é semelhante a uma tabela onde os atributos das entidades são mostrados em colunas. Ela deve permitir a visualização de todas as entidades ou das entidades selecionadas através de filtro. O filtro pode ser usado para restringir a quantidade de entidades apresentadas na grade. O mesmo objeto que exibe os itens na grade fornece opção para inserir, atualizar e remover entidades de negócio. Essas operações são executadas na própria grade.

Estrutura

A Figura 4.6 apresenta o diagrama de classes deste padrão e de seus componentes. As classes de controle e modelo não são apresentadas e seguem o padrão **CRUD-MVC**.

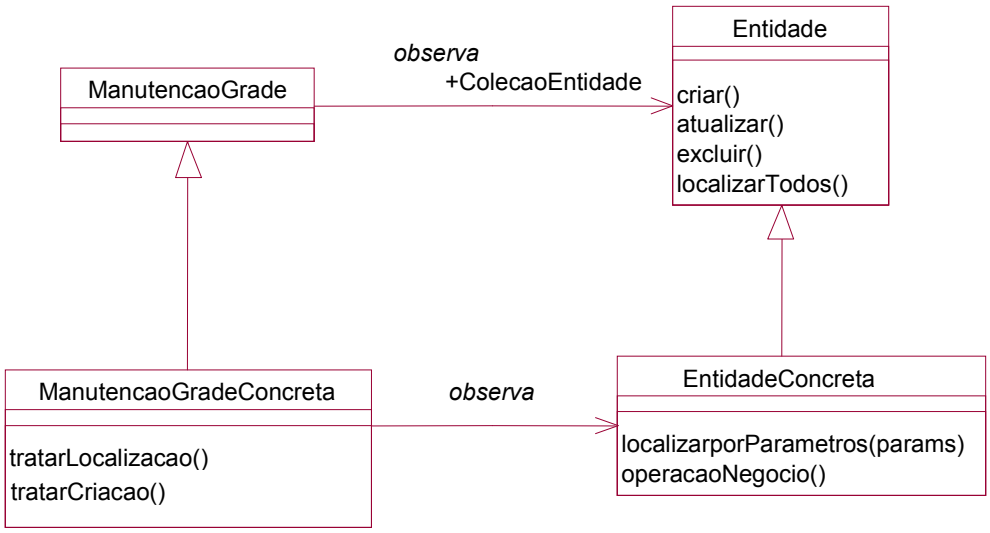


Figura 4.6: Diagrama de classes do padrão Manutenção em Grade

Participantes

- *ManutencaoGrade*
  - Define as características de interação com usuário presentes em toda classe tipo grade: grade de exibição e edição, comandos para criação, atualização e exclusão;
  - Fornece métodos genéricos para tratamento dos eventos de inclusão, alteração e exclusão de itens da coleção. Estes métodos permitem definir operações de pré-processamento e pós-processamento das operações de criação, consulta, atualização e exclusão; e
  - Realiza confirmação da atualização da coleção no repositório de dados.
- *ManutencaoGradeConcreta*
  - Define, quando necessário, as características de interação com usuário para filtro e as operações de negócio específicas; e
  - Responsável pela sincronização das informações das entidades de negócio com a grade.

Dinâmica

Os eventos de CRUD se comportam de acordo com o padrão **CRUD-MVC**. A dinâmica deste padrão está descrita abaixo. A Figura 4.7 ilustra a dinâmica deste padrão.

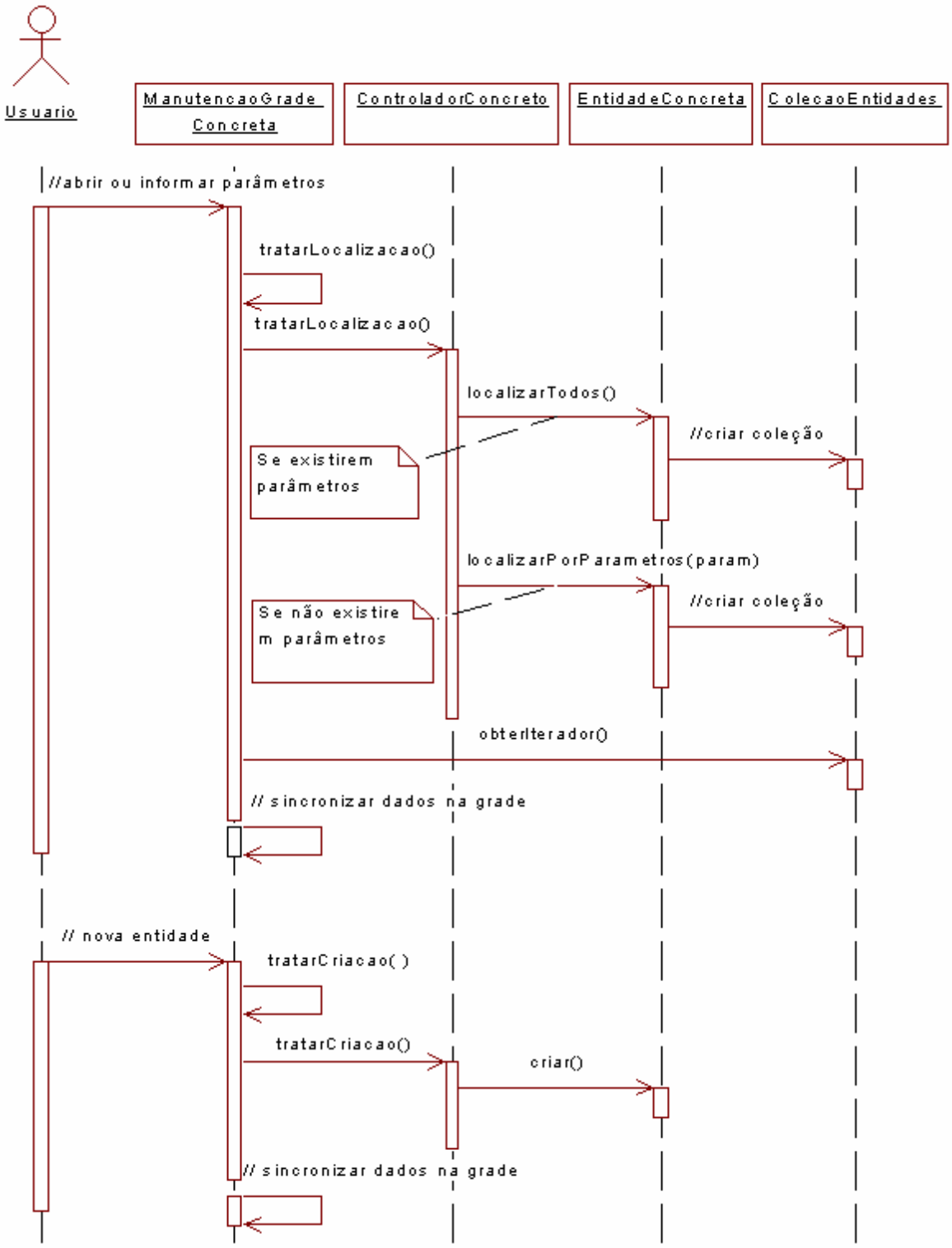


Figura 4.7*:* Diagrama de seqüência do padrão Manutenção em Grade

1. Usuário abre ManutencaoGradeConcreta:
    1.1. Se não houver parâmetro de consulta, *ManutencaoGradeConcreta* notifica *ControladorConcreto* sobre evento de localização de todos; e
    1.2. Se houver parâmetro de consulta, o usuário informa dados, solicita consulta e *ManutencaoGradeConcreta* e notifica *ControladorConcreto* sobre evento de localização de parâmetros.
2. O *ControladorConcreto* utiliza o método localizarTodos ou o método LocalizarPorParâmetros, da *EntidadeConcreta*. Cria *ColecaoEntidades* com base no resultado da operação e disponibiliza *ColecaoEntidades*;
3. *ManutencaoGradeConcreta* obtém iterador de *ColecaoEntidades* e apresenta dados na grade;
4. Usuário cria ou edita entidades interagindo com *ManutencaoGradeConcreta*, que comunica evento ao *ControladorConcreto* através de *tratarCriacao* ou *tratarAtualizacao*;
5. O *ControladorConcreto* envia mensagem de criar ou atualizar para *EntidadeConcreta*; e
6. *ManutencaoGradeConcreta* sincroniza dados de *EntidadeConcreta* utilizando a operação *localizarTodos* ou *LocalizarPorParametros*, exibindo as modificações em *ColecaoEntidades* na grade.

Conseqüências

- Fornece interface simples e com rápida manutenção de entidades permitindo a visualização geral das entidades básicas e sua manutenção em uma única grade;
- Suporta filtros para apresentação de entidades na grade;
- Permite inclusão, alteração e exclusão de entidades, sincronizando a grade com a coleção de entidades;
- Em situações de grande número de entidades, a carga e a sincronização de dados pode ficar lenta. Nesta situação, são necessários mecanismos de paginação; e
- Caso a entidade possua atributos multi-valorados, a edição da entidade pode ficar complexa em uma grade. Neste caso, seria interessante utilizar o padrão **Seleção com Secundária** (seção 4.2.6) mesmo para entidades básicas.

Padrões relacionados

- ***Iterator*** (GAMMA *et al*, 1995):

  Utilizado para navegação de ColecaoEntidade.

- **CRUD-MVC**:

  Utilizado para definir interação entre as classes do MVC e o reuso de estrutura e comportamento.

Usos conhecidos

- Vide padrão **CRUD-MVC** (seção 4.2.3).

4.2.6 Padrão Seleção com Secundária

Contexto

Utilizado para entidades principais ou secundárias, de média ou alta complexidade, com vários atributos e relacionamentos, em situações que requerem a seleção de entidades sob determinado critério, para somente depois realizar edição ou inclusão em objeto de interface auxiliar, e não diretamente sobre a grade. A edição em objeto auxiliar (interface secundária) facilita a edição de um número maior de atributos e relacionamentos.

Este padrão funciona de forma inversa ao padrão **Registro com Busca**. Seu objetivo principal é a consulta de um conjunto de entidades, tendo maior freqüência de cenários para atualização. O padrão **Seleção com Secundária** pode ser utilizado para a atualização de entidades complexas criadas com o uso do padrão **Registro com Busca**.

Neste caso, a atualização pode ser realizada em um subconjunto dos atributos da entidade selecionada com um critério de pesquisa, de acordo com determinado processo de negócio. Por exemplo, em um cenário de aprovação de pedidos realizados em um dado período, os pedidos poderiam ser criados com o uso do padrão **Registro com Busca,** e sua aprovação (ou cancelamento) realizado com o padrão **Seleção com Secundária**: o usuário informa um período e o sistema seleciona os pedidos realizados nesse período e que ainda não foram aprovados. Na interface secundária, o usuário informa seu parecer (cancelado ou aprovado), a data do parecer e uma observação, no caso de cancelamento.

Problema

Como implementar funcionalidades de seleção e manutenção de entidades de negócio de média e alta complexidade de forma eficiente e reutilizável?

Forças

- O reuso de características de interação com usuário deve ser suportado;
- A visualização conjunta de entidades deve ser fornecida;
- Devem ser fornecidas a visualização e a edição do detalhe de uma entidade; e
- Devem ser fornecidos filtros com critérios de seleção de entidades.

Solução

Criar classe *Selecao* para localização e exclusão de entidades, permitindo que grupos de entidades sejam visualizados de acordo com um determinado critério. A classe *Selecao* colabora com a classe *Secundaria* para atualização e criação de entidades. A classe *Secundaria* permite apenas confirmar ou desistir da operação.

A solução utiliza o padrão **CRUD-MVC** com as classes *ComponenteIHC* e *ComponenteIHCConcreto*, sendo implementados por quatro classes que dividem entre si as operações CRUD: *Selecao*, *SelecaoConcreta*, *Secundaria* e *SecundariaConcreta*. As classes de controle e modelo funcionam de acordo com o padrão **CRUD-MVC**.

Estrutura

A Figura 4.8 apresenta o diagrama de classes do padrão e seus componentes.

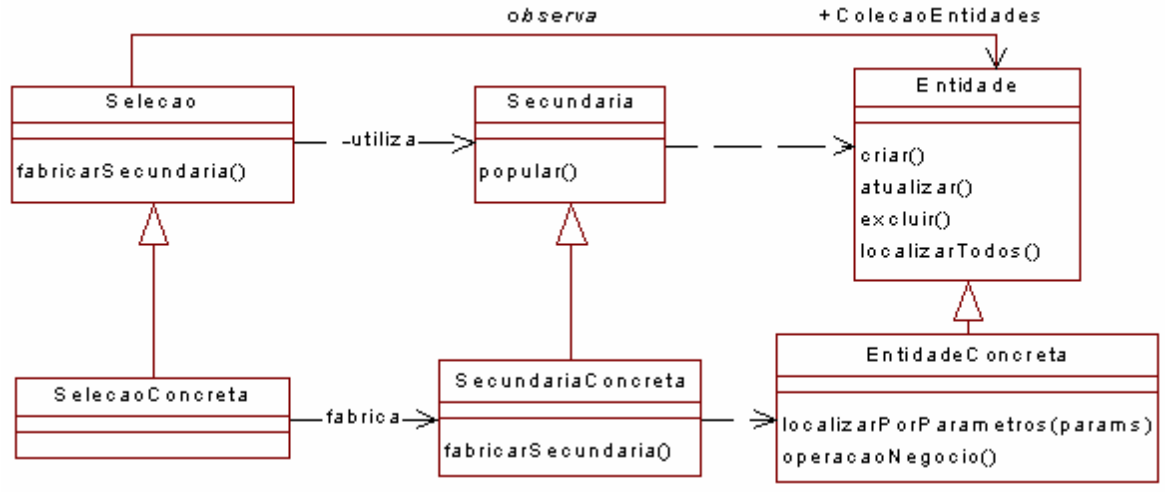


Figura 4.8: Diagrama de Classes do Padrão Seleção com Secundária

As classes de controle e modelo não são apresentadas no diagrama de classes e seguem o padrão **CRUD-MVC**.

Participantes

- *Seleção*
    - Define as características de interação com usuário presente em toda classe de seleção: comandos para iniciar criação e atualização, excluir e filtrar entidades com grade para exibir resultados do filtro;
    - Fornece método *fabricarControlador* para criação do objeto de controle;
    - Fornece métodos genéricos para o tratamento dos eventos de interface com usuário, para iniciar criação e atualização, excluir e filtrar entidades. Os métodos de evento notificam seu observador, Controlador, das ações tomadas pelo usuário; e
    - Fornece método *fabricarSecundaria* para criar uma instância de classe *SecundariaConcreta*. Após a criação, a interface secundária é exibida.
- *Secundaria*
    - Fornece uma interface genérica para entrada de dados da entidade selecionada; e
    - Fornece comando para confirmação da entrada de dados na criação ou atualização.
- *SelecaoConcreta*
    - Define critério de localização (filtro) e campos para visualização da informação na grade;
    - Implementa o método *fabricarControlador*, criando *ControladorConcreto* para implementar *Controlador*; e
    - Implementa o método *fabricarSecundaria*, criando *SecundariaConcreta* para implementar *Secundaria*.
- *SecundariaConcreta*
    - Define como a sincronização de dados deve ser feita e os campos concretos para a entidade; e
    - Define pastas para relacionamentos um-para-muitos da entidade principal de acordo com o padrão **Manutenção em Grade** ou o padrão **Seleção com Secundária**.

Dinâmica

Os eventos de CRUD se comportam de acordo com o padrão **CRUD-MVC**. A seguir, descrevemos um fluxo típico de execução. A Figura 4.9 ilustra a dinâmica deste padrão.

1. Usuário informa, em *SelecaoConcreta*, os parâmetros da pesquisa e solicita a localização;

2. *SelecaoConcreta* notifica o *ControladorConcreto* para a localização por parâmetros;

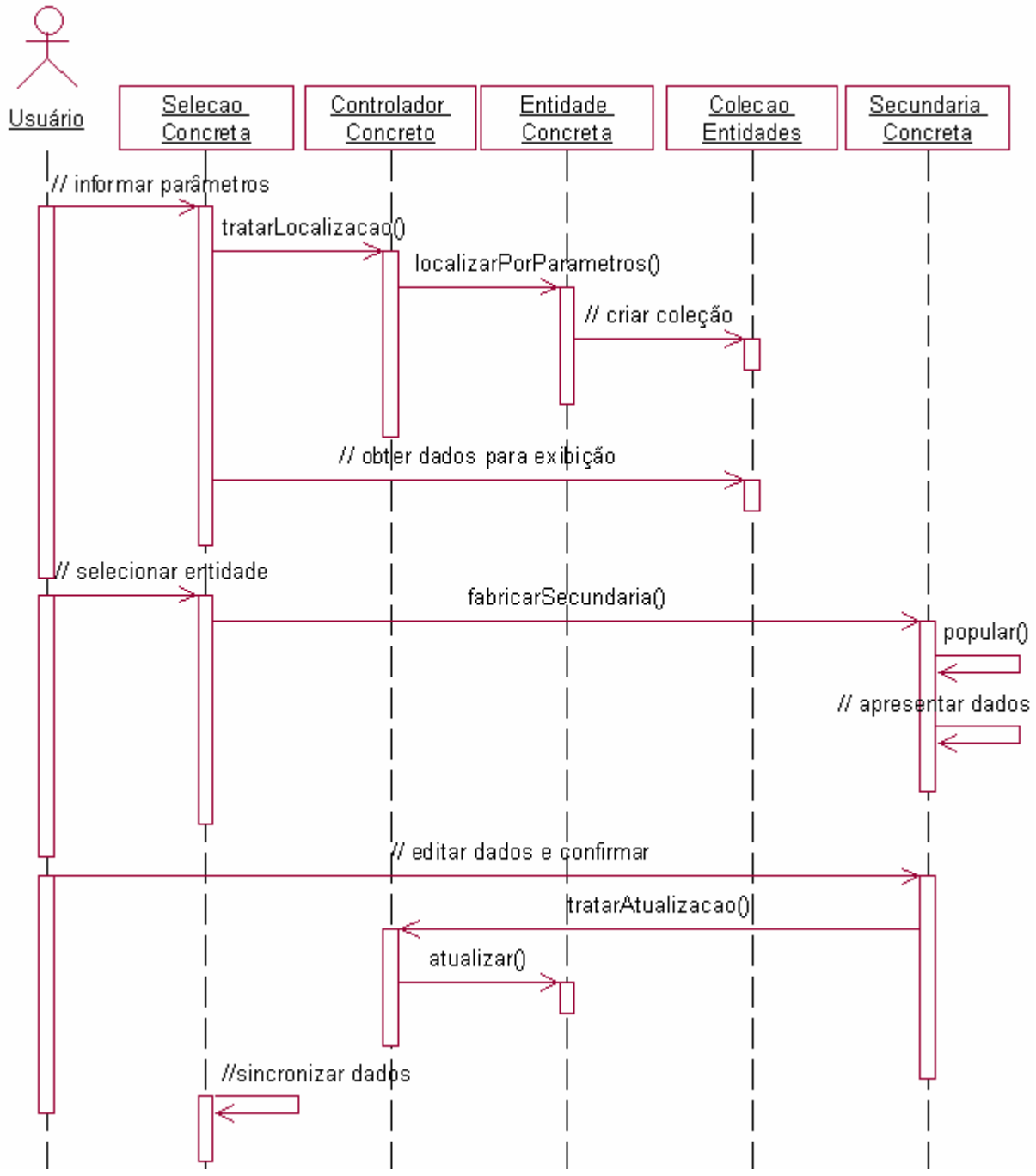


Figura 4.9: Diagrama de seqüência do padrão Seleção com Secundária

3. O *ControladorConcreto* utiliza o método *localizarPorParametros* da *EntidadeConcreta*, que cria *ColecaoEntidades* com base no resultado da operação e disponibiliza *ColecaoEntidades*;
4. Os dados são exibidos na grade da *SelecaoConcreta*;
5. O usuário solicita criação ou seleciona uma entidade e, depois, solicita atualização;
6. *SecundariaConcreta* é fabricada e exibida para usuário através do *Factory Method fabricarSecundaria*;
7. *SecundariaConcreta* executa o método popular que deve carregar entidades auxiliares para edição de atributos multivalorados de *EntidadeConcreta*. O método popular se comporta como um ***Template Method*** (GAMMA *et al.*, 1995), sendo chamado de Secundaria, mas definido apenas em *SecundariaConcreta*. A *SecundariaConcreta* é exibida sem dados (no caso de criação) ou seus dados são sincronizados e exibidos para o usuário (no caso de atualização);
8. Usuário informa os dados e confirma operação;
9. *SecundariaConcreta* solicita criação ou atualização dos dados ao *ControladorConcreto*;
10. *ControladorConcreto* envia mensagem de criar ou atualizar para *EntidadeConcreta*;
11. *SecundariaConcreta* é destruída; e
12. *SelecaoConcreta* sincroniza dados de *EntidadeConcreta* utilizando a operação *localizarTodos* ou *LocalizarPorParametros*, exibindo as modificações em *ColecaoEntidades* na grade (mensagem //sincronizar dados da Figura 4.9).

Conseqüências

- Fornece mecanismo eficiente de busca e visualização do resumo das entidades antes de recuperar e visualizar o detalhamento da entidade (demais campos e relacionamentos);
- Permite maior usabilidade do sistema, possibilitando que o usuário selecione apenas a entidade que deseja visualizar/editar, e ainda, fornece interface mais apropriada que uma grade para edição de entidade mais complexa;
- Fornece filtro para seleção de entidades que podem ser editadas em interface secundária;

- A utilização do elemento Secundária requer um nível a mais de navegação, aumentando o número de interações realizadas pelo usuário; e
- Em entidades muito complexas e com muitos relacionamentos, a composição de vários níveis de Secundárias pode dificultar o uso do padrão.

Padrões relacionados

- **Factory Method**:

  Utilizado na criação de objetos de *Entidade* e de *BuscaSecundariaConcreta*.

- **Template Method**:

  Utilizado para carregar entidades auxiliares para exibição em *SecundariaConcreta*.

- **CRUD-MVC**:

  Utilizado para definir a interação entre as classes de modelo, visão e controle e o reuso de estrutura e comportamento.

- **Manutenção em Grade**:

  Utilizado para definir relacionamentos de um-para-muitos de entidades simples com a entidade principal.

- **Seleção com Secundária** (recursivo):

  Utilizado para definir relacionamentos de um-para-muitos de entidades complexas com a entidade principal.

Usos conhecidos

- Vide padrão **CRUD-MVC** (seção 4.2.3).

4.2.7 Padrão Teste CRUD

Contexto

Especificação dos casos de teste das operações de inclusão, consulta, alteração e exclusão de um caso de uso CRUD.

Problema

Como especificar os casos de teste dos requisitos funcionais de inserção, atualização, exclusão e consulta de dados de um caso de uso CRUD por meio de especificações de testes?

Forças

- Os casos de teste devem cobrir vários cenários dos fluxos das funcionalidades de Incluir, Alterar, Remover e Consultar, incluindo exceções e fluxos alternativos; e
- Dados e valores específicos devem ser gerados para suportar a execução dos cenários.

Solução

Organizar os casos de teste em quatro grupos, um para cada operação de manutenção do caso de uso (**Incluir, Alterar, Remover** e **Consultar**). Em cada grupo, definir os casos de teste e idéias de teste para situações de sucesso e falha, como se segue:

- O grupo **Incluir** apresenta casos de teste para verificar os seguintes itens: (i) a correção do comportamento do sistema ao incluir uma entidade já existente; (ii) a correção do comportamento do sistema ao incluir uma nova entidade; (iii) o tratamento de problema de acesso à base de dados; e (iv) o tratamento de preenchimento de dados obrigatórios e valores inválidos.
- O grupo **Alterar** apresenta casos de teste para verificar os seguintes itens: (i) a correção do comportamento do sistema ao alterar uma entidade; (ii) o tratamento de problema de acesso à base de dados; e (iii) o tratamento de preenchimento de dados obrigatórios e valores inválidos.
- O grupo **Remover** apresenta casos de teste para verificar os seguintes itens: (i) a correção do comportamento do sistema ao remover uma entidade; e (ii) o tratamento de problema de acesso à base de dados.
- O grupo **Consultar** apresenta casos de teste para verificar os seguintes itens: (i) a correção do comportamento do sistema ao consultar entidades de acordo com os filtros ao realizar uma consulta que não retorna nenhuma entidade; (ii) o tratamento de problema de acesso à base de dados; e (iii) o tratamento de preenchimento de filtros obrigatórios e valores inválidos.

- Todos os grupos apresentam casos de teste para validar as regras de negócio específicas de cada grupo e indicar a ação que o sistema deve tomar caso a regra não seja satisfeita. Por exemplo: ao incluir um Cliente em um sistema de Telemarketing, a data de desativação deverá ser maior que a data de ativação.
- Os grupos **Incluir**, **Alterar** e **Consultar** apresentam idéias de teste para verificar o tamanho limite para preenchimento, os valores não permitidos e as restrições específicas dos tipos de dados, para cada atributo da entidade.

Estrutura

A estrutura a seguir envolve *casos de teste* e *idéias de teste*.

Casos de teste – Inserir <*nome da entidade*>

Caso de teste	Procedimento de teste	Resultado
Entidade existente	*<definir critérios para entidade existente>*	*<definir critérios para verificar o resultado>*
Entidade não existente	*<definir critérios para entidade não existente>*	*<descrever o procedimento para verificar se os dados foram inseridos corretamente>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*
Dados obrigatórios	*<listar os atributos que são obrigatórios e informar o procedimento para o teste>*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos atributos>*
Valores inválidos	*<listar os atributos no qual os valores têm que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*

Casos de teste – Alterar <*nome da entidade*>

Caso de teste	Procedimento de teste	Resultado
Alteração de entidade	*<definir procedimento para alterar uma entidade>*	*<definir critérios para verificar o resultado após a alteração de uma entidade>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*

Dados obrigatórios	*<listar os atributos que são obrigatórios e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos atributos listados>*
Valores inválidos	*<listar os atributos no qual os valores têm que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*

Casos de teste – Remover *<nome da entidade>*

Caso de teste	Procedimento de teste	Resultado
Remoção de entidade	*<definir procedimento para remover uma entidade>*	*<definir critérios para verificar o resultado após a remoção de uma entidade>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*

Casos de teste – Consultar *<nome da entidade>*

Caso de teste	Procedimento de teste	Resultado
Consulta de entidades de acordo com os filtros	*<definir procedimento para consultar>*	*<definir critérios para verificar o resultado após a consulta de uma ou mais entidades>*
Consulta que não retorna nenhuma entidade	*<definir procedimento para consultar>*	*<descrever qual o comportamento esperado do sistema>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*
Filtros obrigatórios	*<listar os filtros que são obrigatórios e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos filtros listados>*
Valores inválidos	*<listar os filtros nos quais os valores tenham que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*

Idéias de teste – Incluir, Alterar e Consultar <*nome da entidade*>

Nome do atributo	Validação	Valor
<*nome do atributo*>	Tamanho limite para preenchimento	<*descrever o tamanho limite para preenchimento*>
<*nome do atributo*>	Valores não permitidos	<*listar todos os valores não permitidos ou uma faixa de valores não permitidos*>
<*nome do atributo*>	Restrições específicas	<*se existir, identificar restrições específicas do tipo de dados do atributo*>

Exemplo do Padrão

Este exemplo apresenta os casos de teste do caso de uso Manter Cliente de uma aplicação de *Telemarketing*.

Casos de teste – Inserir Cliente

Caso de teste	Procedimento de teste	Resultado
Cliente existente	Para verificar se o cliente já existe no sistema, deve-se verificar se existe um cliente cadastrado com o mesmo CPF.	O sistema deve exibir a mensagem: "Cliente já cadastrado" e retornar para a tela de inserção.
Cliente não existente	Para verificar se o cliente não existe no sistema, deve-se verificar que não existe um cliente cadastrado com o mesmo CPF.	Ao consultar o cliente cadastrado o sistema deve exibir as informações do cliente de acordo com os dados informados na inserção.
Acesso à base de dados	Simular a queda do acesso a base de dados antes de solicitar a inserção do cliente.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Atributos obrigatórios: Nome, Logradouro, Número, Bairro, Cidade, Estado e CPF. Procedimento: deixar em branco os atributos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Nome, Logradouro, Número, Bairro, Cidade, Estado e CPF" e voltar para a tela de inserção com os dados previamente informados.
Valores inválidos	Atributos com restrição de valores: Número, CPF, Data de ativação e Data de desativação. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os campos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Número, CPF, Data de ativação e Data de desativação" e voltar para a tela de inserção com os dados previamente informados.

Casos de teste – Alterar Cliente

Caso de teste	Procedimento de teste	Resultado
Alteração de cliente	Inicialmente consultar o cliente desejado informando o CPF e posteriormente solicitar a alteração do cliente	Ao consultar o cliente alterado o sistema deve exibir as informações do cliente de acordo com os dados informados na alteração.
Acesso à base de dados	Simular a queda do acesso a base de dados antes de solicitar a alteração do cliente.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Atributos obrigatórios: Nome, Logradouro, Número, Bairro, Cidade, Estado e CPF. Procedimento: deixar em branco os atributos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Nome, Logradouro, Número, Bairro, Cidade, Estado e CPF" e voltar para a tela de inserção com os dados previamente informados.
Valores inválidos	Atributos com restrição de valores: Número, CPF, Data de ativação e Data de desativação. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os campos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Número, CPF, Data de ativação e Data de desativação" e voltar para a tela de inserção com os dados previamente informados.

Casos de teste – Remover Cliente

Caso de teste	Procedimento de teste	Resultado
Remoção de cliente	Inicialmente consultar o cliente desejado informando o CPF e posteriormente solicitar a remoção do cliente	Ao remover o cliente o sistema deve exibir a mensagem: "Cliente removido com sucesso"
Acesso à base de dados	Simular a queda do acesso a base de dados antes de solicitar a remoção do cliente.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".

Casos de teste – Consultar Cliente

Caso de teste	Procedimento de teste	Resultado
Consulta de clientes	Inicialmente, consultar os clientes informando o Nome e ou CPF	O sistema deve apresentar uma lista contendo os clientes que correspondem aos filtros especificados.
Consulta que não retorna nenhum cliente	Informar valores para o nome e CPF inexistentes na aplicação.	O sistema deve exibir a mensagem: "Nenhum registro encontrado para os filtros especificados".

Acesso à base de dados	Simular a queda do acesso a base de dados antes de solicitar a consulta dos clientes.	O sistema deve exibir a seguinte mensagem: “Erro no acesso a base de dados. Por favor, tente mais tarde”.
Filtros obrigatórios	Esta consulta não possui filtros obrigatórios	-
Valores inválidos	Filtro com restrição de valor: CPF Procedimento: preencher o CPF com valores inválidos de acordo com as idéias de teste, o campo acima.	O sistema deve exibir a mensagem: “Os seguintes atributos foram informados com valores inválidos: CPF” e retornam para a tela de consulta com o filtro previamente informado.

Idéias de teste – Incluir, Alterar e Consultar Cliente

Nome do atributo	Validação	Valor
Número	Tamanho limite para preenchimento	6 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, como por exemplo, # e $.
CPF	Tamanho limite para preenchimento	14 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, com exceção dos caracteres “–“ e “.”
	Restrições específicas	Deve ser informado no seguinte formato: *xxx.xxx.xxx-xx*
Data de ativação	Tamanho limite para preenchimento	10 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, com exceção do caractere “/“
	Restrições específicas	Deve ser informado no seguinte formato: *dd*/*mm*/*aaaa* *dd* deve possuir valores entre 1 e 31 *mm* deve possuir valores entre 1 e 12 Para *mm* = 4, 6, 9, e 11, *dd* = 31 Para *mm* = 2, dd ,30 Para *aaaa* múltiplo de 4 e *mm* = 2, dd < 29.

Conseqüências

- Diminuição de esforço para elaborar a especificação do testes;
- Garante a cobertura dos testes desejada; e
- Facilita e agiliza a execução dos testes.

Padrões relacionados

- Não se aplica.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.3 Metapadrão Relatório

Contexto

Projetos de desenvolvimento de sistemas de informação que requerem a criação dos produtos de trabalho gerados nas disciplinas de requisitos, análise e projeto, implementação e testes para funcionalidades de geração de relatórios.

Problema

Como tratar a geração de relatórios nas diversas fases do ciclo de vida de desenvolvimento de *software* de forma integrada e consistente?

Solução

Este metapadrão descreve diretrizes para geração de relatórios. O relatório deve permitir a parametrização (filtros), a visualização, e a exportação de dados. Estruturas complementares, como agrupamento e totalizações, são fornecidas.

Padrões que implementam o metapadrão

- Caso de Uso Relatório; e
- Teste Relatório.

4.3.1 Padrão Caso de Uso Relatório

Contexto

Em sistemas de informação, uma grande quantidade de dados é armazenada freqüentemente. Neste contexto, surge a necessidade de visualizar, exportar ou imprimir dados armazenados, com o objetivo de conferir, analisar e tomar decisões com base nesses dados.

Problema

Como documentar os requisitos de relatórios que podem incluir a necessidade de visualizar, exportar ou imprimir dados de entidades, de acordo com filtros especificados, agrupamentos, totalizações e informações a serem apresentadas?

Forças

- O sistema deve permitir a extração de dados em diversos formatos (tela, arquivo e impressão);
- O sistema deve tratar a estrutura do relatório, como por exemplo, disposição de campos, cabeçalho e rodapé, tamanho da fonte e orientação do papel; e
- O sistema deve tratar as necessidades para a exibição dos dados, como por exemplo, se os dados devem ser agrupados, se devem ser apresentadas totalizações e se existe a necessidade de algum filtro para restringir os dados que serão apresentados.

Solução

- O **Fluxo básico** do caso de uso descreve a condição de início. Condições de início indicam os eventos que provocam a execução do caso de uso. Por exemplo: em que situação o relatório deve ser visualizado ou impresso; ou se existe alguma periodicidade requerida;
- O **Fluxo básico** descreve quais atributos devem ser filtros, quais são de preenchimento obrigatório e quais atributos devem ser exibidos no cabeçalho, corpo ou rodapé; e
- Os requisitos especiais são documentados em uma seção independente dos fluxos e subfluxos. Tipicamente, devem ser documentados requisitos de exportação para diversos formatos, regras das seções (regras de agrupamento e cálculo para

totalização) e opções de ordenação. Um desenho esquemático do relatório e suas seções pode também ser apresentado. Descrever também o critério para filtro ou extração de dados.

Estrutura

Fluxo básico

1. Este fluxo inicia quando o <*nome do ator*> solicita gerar o relatório <nome do relatório>. <descrever a condição de início do caso de uso>.
2. O sistema solicita o preenchimento dos seguintes filtros:
   - <*lista de filtros*>.
3. Uma vez que o <*nome do ator*> forneça a informação solicitada, uma das seguintes ações é executada:
   - Se o <*nome do ator*> selecionar Imprimir, <*descrever ação que deve ser executada*>.
   - Se o <*nome do ator*> selecionar Visualizar, <*descrever ação que deve ser executada*>.
   - Se o <*nome do ator*> selecionar Exportar, <*descrever ação que deve ser executada*>.
4. O sistema apresenta o resultado na seguinte forma:
   - Cabeçalho. <*descrever as informações que devem estar contidas no cabeçalho*>.
   - Corpo. <*descrever as informações que devem estar contidas no corpo, informando lista de atributos, seções de agrupamento, e quebra de seção*>.
   - Rodapé. <*descrever as informações que devem estar contidas no rodapé*>.
   - Totalização. <*descrever que totalizações devem ser exibidas*>.

Requisitos especiais

- Exportar para diversos formatos. <*descrever para que formatos o resultado do relatório deve ser exportado, informando os requisitos necessários para a exportação de cada formato*>;
- Regras das seções. <*descrever quais são as regras de agrupamento de seções e as regras para o cálculo das totalizações*>;

- Opções de ordenação. <*listar as opções de ordenação disponíveis e descrever os requisitos para essas ordenações*>;
- Regra de extração. <*expressão lógica descrevendo como os atributos de filtro e outros critérios devem ser combinados para extrair os dados corretamente*>; e
- Modelo de desenho esquemático:

<Logo><Sistema>	**<Título>**	
<Grupo1>		
	<Campo 1>	<Campo 2>
<Total grupo 1>		**<Soma campo 2>**
		<Página x de y>

Exemplo do Padrão

Este exemplo apresenta um relatório de clientes de uma aplicação de *Telemarketing*. O relatório possui totalizações por bairro.

Fluxo básico

1. Este fluxo inicia quando o *Operador de Telemarketing* solicita gerar o relatório de *clientes por bairro*. Este relatório deve ser executado antes da avaliação da carteira de clientes.
2. O sistema solicita o preenchimento dos seguintes filtros:
   - * Código da filial.
3. Uma vez que o *Operador de Telemarketing* forneça a informação solicitada, uma das seguintes ações é executada:
   - Se o *Operador de Telemarketing* selecionar Imprimir, o sistema deve apresentar a janela de configuração de impressão.
   - Se o *Operador de Telemarketing* selecionar Visualizar, o sistema deve apresentar uma janela com a visualização do relatório.
   - Se o *Operador de Telemarketing* selecionar Exportar, o sistema deve solicitar o tipo de arquivo a ser exportado e gerar o arquivo solicitado conforme padrão definido nos requisitos especiais.
4. O sistema apresenta o resultado na seguinte forma:
   - Cabeçalho. Deve conter o nome do relatório, nome da empresa, nome da filial e a data em que o relatório foi executado.

- Corpo. Os clientes devem ser agrupados por bairro e as seções devem conter quebras de página a cada bairro. Os seguintes atributos devem ser apresentados: nome do bairro, nome, telefone e data de cadastro do cliente.
- Rodapé. Deve conter o número da página.
- Totalização. As totalizações devem ser efetuadas por bairro, apresentando quantos clientes existem em cada bairro.

Requisitos especiais

- Exportar para diversos formatos. Os dados deste relatório devem ser exportados para o Excel, apresentado as informações em colunas;
- Opções de ordenação. O relatório deve ser ordenado por nome do bairro e posteriormente por nome do cliente;
- Regra de extração. Devem ser apresentados no relatório todos os clientes cadastrados no sistema e que são relacionados à filial selecionada no filtro. A identificação da filial encontra-se no cadastro do cliente; e
- Modelo de desenho esquemático:

Empresa de Telemarketing - Filial Norte América **Relatório de clientes por bairro** **01/01/2005**		
Bairro: Varjota		
Nome	Telefone	Data de Cadastro
Gabriela Souza	42578958	01/01/2004
Carlo Pires	29487658	23/04/2004
Total de clientes da Varjota:		**2**
		Página 1

Conseqüências

- As opções e formatos para extração são descritos;
- A estrutura do relatório e das seções é documentada de forma clara e estruturada; e
- Os requisitos para extração da informação são documentados.

Padrões relacionados

- Padrão **Documentação de Atributos**:

  Utilizado no **Fluxo básico** para listar os filtros.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.3.2 Padrão Teste Relatório

Contexto

Especificação dos casos de teste de relatórios.

Problema

Como especificar os casos de teste de relatórios por meio de especificações de testes?

Forças

- Os casos de teste devem cobrir cenários para geração do relatório: formatação, agrupamentos e totalizações; e
- Dados e valores específicos devem ser gerados para suportar a execução dos cenários.

Solução

Definir os *casos de teste* e *idéias de teste* para situações de sucesso e falha, como se segue:

- Apresentar casos de teste para verificar se o comportamento do sistema está correto na geração do relatório de acordo com os filtros especificados, o tratamento de problema de acesso à base de dados na geração do relatório, o tratamento de preenchimento de filtros obrigatórios e valores inválidos;
- Apresentar casos de teste para verificar a impressão e o formato do relatório (cabeçalho, corpo e rodapé);
- Apresentar casos de teste para verificar totalizações, cálculos e agrupamentos;
- Apresentar casos de teste para verificar formato e dados em arquivos exportados; e
- Apresentar idéias de teste para verificar o tamanho limite para preenchimento, valores não permitidos e restrições específicas dos tipos de dados, para cada filtro.

Estrutura

Casos de teste – Relatório <*nome do relatório*>

Caso de teste	Procedimento de teste	Resultado
Gerar relatório de acordo com os filtros especificados	*<definir procedimento para a geração do relatório>*	*<definir critérios para verificar o resultado>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*
Dados obrigatórios	*<listar os filtros que são obrigatórios e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos filtros listados>*
Valores inválidos	*<listar os filtros nos quais os valores tenham que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*
Impressão	*<definir procedimento para a impressão do relatório>*	*<definir critérios para verificar o resultado>*
Totalizações, cálculos e agrupamentos	*<definir procedimento para as totalizações, cálculos e agrupamentos>*	*<definir critérios para verificar o resultado>*
Formato do relatório (cabeçalho, corpo e rodapé)	*<definir procedimento para formato do relatório>*	*<definir critérios para verificar o resultado>*
Verificar formato e dados em arquivos exportados	*<definir procedimento para formato e dados em arquivos exportados>*	*<definir critérios para verificar o resultado>*

Idéias de teste – Relatório <nome do relatório>

Nome do filtro	Validação	Valor
<nome do filtro>	Tamanho limite para preenchimento	*<descrever o tamanho limite para preenchimento>*
<nome do filtro>	Valores não permitidos	*<listar todos os valores não permitidos ou apenas uma faixa destes valores >*
<nome do filtro>	Restrições específicas	*<se existir, identificar restrições específicas do tipo de dados do atributo>*

Exemplo do Padrão

Este exemplo apresenta os casos de teste de um relatório de clientes de uma aplicação de *Telemarketing*. O relatório possui totalizações por bairro.

Casos de teste – Relatório de Clientes por Bairro

Caso de Teste	Procedimento de teste	Resultado
Gerar relatório de acordo com os filtros especificados	Informar o código da filial e solicitar a visualização do relatório	O sistema deve apresentar o relatório contendo somente os clientes que pertencem a filial informada no filtro
Acesso à base de dados	Simular a queda do acesso a base de dados antes de solicitar a geração do relatório.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Filtro obrigatório: Código da filial Procedimento: deixar em branco o filtro acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Código da filial" e voltar para a tela de filtros.
Valores inválidos	Filtro com restrição de valor: Código da filial Procedimento: preencher com valores inválidos de acordo com as idéias de teste, o filtro acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Código da filial" e voltar para a tela de filtros com os dados previamente informados.
Impressão	Informar o código da filial e solicitar a impressão do relatório	O sistema deve imprimir o relatório em folha A4 respeitando as margens e tamanho do papel. O formato impresso deve ser semelhante ao formato da visualização
Totalizações, cálculos e agrupamentos	Informar o código da filial e solicitar a visualização do relatório.	Os clientes devem estar agrupados por bairro e ao final de cada grupo deve existir o total de clientes do bairro.
Formato do relatório (cabeçalho, corpo e rodapé)	Informar o código da filial e solicitar a visualização do relatório.	O relatório deve conter no cabeçalho o nome do relatório, nome da empresa, nome da filial e a data em que o relatório foi executado. O corpo do relatório deve conter nome, telefone e data de cadastro de cada cliente. No inicio de cada grupo deve aparecer o nome do bairro e ao final de cada grupo o total de clientes do bairro. O rodapé deve conter o número da página.
Verificar formato e dados em arquivos exportados	Informar o código da filial e solicitar a exportação do relatório para o formato de planilha Excel.	O sistema deve gerar um arquivo Excel contendo os clientes da filial com as seguintes colunas: nome, bairro, telefone e data de cadastro do cliente.

Idéias de teste – Relatório de Clientes por Bairro

Nome do filtro	Validação	Valor
Código da filial	Tamanho limite para preenchimento	4 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, com exceção do caractere "–"

Conseqüências

- Diminuição de esforço para a elaboração da especificação do testes;
- Garante a cobertura dos testes desejada; e
- Facilita e agiliza a execução dos testes.

Padrões relacionados

- Não se aplica.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.4 Metapadrão Assistente

Contexto

Projetos de desenvolvimento de sistemas de informação que requerem a criação dos produtos de trabalho gerados nas disciplinas de requisitos, análise e projeto, implementação e testes para operações que são executadas em diversas etapas com a interação do usuário.

Problema

Como tratar o processamento de operações complexas que são executadas em diversos passos, onde decisões ou dados necessitam ser informados em cada passo, através da interação com o usuário, nas diversas fases de um ciclo de vida de desenvolvimento de *software*, de forma integrada e consistente?

Solução

Este metapadrão descreve diretrizes para a organização de operações baseadas em assistentes. O assistente deve organizar a operação em passos, de forma que cada passo tenha início em um ponto onde necessite de configurações ou decisões do usuário.

Padrões que implementam o metapadrão

- Caso de Uso Assistente; e
- Teste Assistente.

4.4.1 Padrão Caso de Uso Assistente

Contexto

Documentação dos requisitos de operações complexas que são executadas em diversos passos, onde decisões ou dados necessitam ser informados em cada passo, através da interação com o usuário.

Problema

Como documentar os requisitos de uma operação, na qual diversas decisões devem ser tomadas antes que a operação possa ser concluída completamente?

Forças

- Para concluir a operação, diversos passos precisam ser realizados; e
- Um determinado passo pode necessitar ser terminado antes que o passo seguinte possa ser feito.

Solução

Organizar o fluxo de eventos do caso de uso em um fluxo geral e um subfluxo para cada passo da operação, como se segue:

- O **Fluxo básico** do caso de uso descreve a condição de início. Condições de início indicam os eventos que provocam a execução do caso de uso. Por exemplo: em que situação o caso de uso deverá ser executado ou se existe alguma periodicidade requerida;
- O **Fluxo básico** descreve o objetivo da operação e quantos passos precisam ser executados;
- O **Fluxo básico** deve indicar que existe uma opção de cancelamento que pode ser solicitada a qualquer momento; e
- A cada **Subfluxo Passo <*n*>** deve determinar se o usuário pode começar o passo seguinte antes de terminar o atual.

Estrutura

Fluxo básico

1. O caso de uso inicia quando o <*nome do ator*> necessita <*nome do caso de uso*>. <*descrever a condição de inicio do caso de uso*>.
2. O sistema informa tipicamente o objetivo da operação e quantos passos precisam ser executados.
3. O sistema solicita que o <*nome do ator*> execute o Passo 1.
4. Uma vez que o <*nome do ator*> decida executar o Passo 1, subfluxo **Passo 1** é executado.
5. O caso de uso se encerra.

Subfluxo Passo 1

1. Este subfluxo se inicia quando o <*nome do ator*> solicita <*descrever as ações que serão executadas neste passo*>.
2. O sistema solicita ao <*nome do ator*> o preenchimento dos seguintes atributos:
   - <*lista de atributos*>.
3. O <*nome do ator*> preenche os atributos.
4. O sistema solicita que o <*nome do ator*> execute o Passo *n*.
5. Uma vez que o <*nome do ator*> decida executar o Passo <*n*>, o subfluxo Passo <*n*> é executado.

Subfluxo Passo <*n*>

1. Este subfluxo se inicia quando o <*nome do ator*> solicita <*descrever as ações que serão executadas neste passo*>.
2. Para este subfluxo ser executado, os subfluxos <*Passo 1, Passo 2, ... Passo n*> devem ter sido executados (se não existirem requisitos de precedência para a execução dos passos anteriores, esse item poderá ser omitido).
3. O sistema solicita ao <*nome do ator*> o preenchimento dos seguintes atributos:
   - <*lista de atributos*>.
4. O <*nome do ator*> preenche os atributos.

5. O sistema solicita que o <*nome do ator*> execute o Passo <*n*+1> ou conclua a operação.
6. Uma vez que o <*nome do ator*> decida executar o Passo <*n*+1>, subfluxo **Passo <*n*+1>** é executado.

Subfluxo Passo <*final*>

1. Este subfluxo se inicia quando o <*nome do ator*> solicita <*descrever as ações que serão executadas neste passo*>.
2. Para este subfluxo ser executado, os subfluxos <*Passo 1, Passo 2, ... Passo n*> devem ter sido executados (se não existirem requisitos de precedência para a execução dos passos, esse item poderá ser omitido).
3. O sistema solicita ao <*nome do ator*> o preenchimento dos seguintes atributos:
   - <*lista de atributos*>.
4. O <*nome do ator*> preenche os atributos.
5. O sistema solicita que o <*nome do ator*> conclua a operação.
6. O caso de uso retorna para o passo 5 do fluxo básico.

Exemplo do Padrão

Este exemplo apresenta o caso de uso Submeter Proposta de Seguro de um sistema de administração de seguros para automóveis, que deve ser realizado em três passos. No passo inicial, o proponente informa a cidade e o estado onde o veículo irá circular e os dados do veículo. No segundo passo, o sistema apresenta uma lista de coberturas e preços existentes de acordo com os dados informados no passo 1. O proponente seleciona as coberturas desejadas e avança para o passo seguinte. No terceiro e último passo, o sistema apresenta o preço total do seguro e solicita a conclusão da operação.

Fluxo básico

1. O caso de uso inicia quando o *Proponente* necessita submeter uma proposta .
2. O sistema informa que esta operação será executada em 3 passos.
3. O sistema solicita que o *Proponente* execute o Passo 1.
4. Uma vez que o *Proponente* decida executar o Passo 1, subfluxo **Passo 1** é executado.
5. O caso de uso se encerra.

Passo 1

1. Este subfluxo se inicia quando o *Proponente* solicita informar a cidade e o estado onde o veículo irá circular e os dados do veículo.
2. O sistema solicita ao *Proponente* o preenchimento dos seguintes atributos:
   - * Cidade. Indica a cidade onde o veículo irá circular.
   - * Estado. Indica o onde o veículo irá circular.
   - * Ano de fabricação do veículo.
   - * Ano do modelo do veículo.
   - * Modelo do veículo.
   - * Marca do veículo.
3. O *Proponente* preenche os atributos.
4. O sistema solicita que o *Proponente* execute o Passo 2.
5. Uma vez que o *Proponente* decida executar o Passo 2, subfluxo **Passo 2** é executado.

Passo 2

1. Este subfluxo se inicia quando o sistema apresenta uma lista de coberturas e preços existentes.
2. Para este subfluxo ser executado o subfluxo **Passo 1** deve ter sido executado.
3. O sistema apresenta a lista de coberturas e preços existente e solicita ao *Proponente* a seleção das coberturas desejadas.
4. O *Proponente* seleciona as coberturas.
5. O sistema solicita que o *Proponente* execute o Passo 3.
6. Uma vez que o *Proponente* decida executar o Passo 3, subfluxo **Passo 3** é executado.

Passo 3

1. Este subfluxo se inicia quando o *Proponente* solicita a conclusão da operação.
2. Para este subfluxo ser executado os subfluxos **Passo 1** e **Passo 2** devem ter sido executados.
3. O sistema apresenta o preço total do seguro e solicita a conclusão da operação.
4. O *Proponente* conclui a operação.
5. O caso de uso retorna para o passo 5 do fluxo básico.

Conseqüências

- Organiza e documenta todos os passos que devem ser realizados para concluir uma operação complexa; e
- Permite que o usuário possa realizar intervenções, decisões e configurações em estágios intermediários de uma operação complexa.

Padrões relacionados

- Padrão **Documentação de Atributos**:

  Utilizado no **Fluxo básico** e nos **subfluxos** para listar os atributos.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.4.2 Padrão Teste Assistente

Contexto

Especificação dos casos de teste de operações complexas que são executadas em diversos passos, onde são necessárias tomadas de decisão ou complementação de dados em cada passo através da iteração com o usuário, antes que a operação possa ser concluída completamente.

Problema

Como especificar os casos de teste de uma operação, onde são necessárias tomadas de decisão ou complementação de dados em cada passo, através da iteração com o usuário, antes que a operação possa ser concluída?

Forças

- Cada passo apresenta pontos potenciais de falha;
- O contexto deve ser mantido de um passo a outro, para o sucesso da operação;
- A operação só poderá ser concluída ao percorrer os passos necessários, se houver interrupção na operação os dados podem ficar inconsistentes; e
- A seqüência correta dos passos deve ser garantida para várias combinações de decisões;

Solução

Organizar os casos de teste em grupos, um para o fluxo geral e um para cada passo da operação. Em cada grupo, definir casos de teste e idéias de teste para situações de sucesso e falha:

- O grupo **Fluxo Geral** apresenta casos de teste para verificar se o comportamento do sistema está correto na execução da operação observando a seqüência correta dos passos e a conclusão da operação;
- Cada grupo **Passo n** apresenta casos de teste para verificar o tratamento de problema de acesso à base de dados, de regras de negócio, de preenchimento de parâmetros obrigatórios e valores inválidos e se as informações dos passos anteriores são passadas de forma correta entre os contextos; e
- Cada grupo **Passo n** apresenta idéias de teste para verificar o tamanho limite para preenchimento, valores não permitidos e restrições específicas dos tipos de dados para cada parâmetro.

Estrutura

Casos de teste – Fluxo Geral

Caso de teste	Procedimento de teste	Resultado
Executar a operação de acordo com os parâmetros especificados e passos requeridos	*<definir procedimento para a execução da operação>*	*<definir critérios para verificar o resultado>*
Seqüência dos passos	*<definir procedimento para a testar a seqüência dos passos>*	*<definir critérios para verificar o resultado>*

Casos de teste – Passo *n*

Caso de teste	Procedimento de teste	Resultado
<regra de negócio>	*<definir procedimento para a testar a regra de negócio>*	*<definir critérios para verificar o resultado>*
Passagem de informações de um passo para outro	*<definir procedimento para testar a passagem de informações de um passo para outro>*	*<definir critérios para verificar o resultado>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*

Dados obrigatórios	*<listar os parâmetros que são obrigatórios e informar o procedimento para o teste>*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos parâmetros listados>*
Valores inválidos	*<listar os parâmetros nos quais os valores tenham que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*

Idéias de teste – Passo *n*

Nome do parâmetro	Validação	Valor
<nome do parâmetro >	Tamanho limite para preenchimento	*<descrever o tamanho limite para preenchimento>*
<nome do parâmetro >	Valores não permitidos	*<listar todos os valores não permitidos ou uma faixa de valores não permitidos>*
<nome do parâmetro >	Restrições específicas	*<se existir, identificar restrições específicas do tipo de dados do parâmetro>*

Exemplo do Padrão

Este exemplo deste padrão apresenta o caso de uso Submeter Proposta de Seguro de um Sistema de administração de seguros para automóveis, que deve ser realizado em três passos.

Casos de teste – Fluxo Geral

Caso de teste	Procedimento de teste	Resultado
Executar a operação de acordo com os parâmetros especificados e passos requeridos	Executar os Passos 1, 2 e 3 informando todos os parâmetros obrigatórios e concluir a operação.	Após a conclusão da operação o sistema deverá ter gerado uma proposta de seguro cujo status seja "Submetida".
Seqüência dos passos	O Passo 2 só pode ser executado após o Passo 1 e o Passo 3 só pode ser executado após os Passos 1 e 2.	Se a seqüência de passos não for obedecida, o sistema deve mostrar a mensagem: "O Passo *n* deve ser executado antes do Passo *n+1*"

Casos de teste – Passo 1

Caso de teste	Procedimento de teste	Resultado
Passagem de informações de um passo para outro	O Passo 1 deverá passar os seguintes parâmetros para o Passo 2: Cidade, Estado, Ano de fabricação do veículo, Ano do modelo do veículo, Modelo do veículo e Marca do veículo.	Se os parâmetros não forem passados corretamente, as coberturas com os respectivos valores não serão calculadas corretamente.
Acesso à base de dados	Forçar a queda do acesso à base de dados antes da execução do passo.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Parâmetros obrigatórios: Cidade, Estado, Ano de fabricação do veículo, Ano do modelo do veículo, Modelo do veículo e Marca do veículo. Procedimento: deixar em branco os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes parâmetros não foram informados: Cidade, Estado, Ano de fabricação do veículo, Ano do modelo do veículo, Modelo do veículo e Marca do veículo" e voltar para a tela do Passo 1.
Valores inválidos	Parâmetros com restrições de valores: Ano de fabricação do veículo e Ano do modelo do veículo. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes parâmetros foram informados com valores inválidos: Cidade, Estado, Ano de fabricação do veículo, Ano do modelo do veículo, Modelo do veículo e Marca do veículo" e voltar para a tela do Passo 1 com os dados previamente informados.

Idéias de teste – Passo 1

Nome do parâmetro	Validação	Valor
Ano de fabricação do veículo	Tamanho limite para preenchimento	4 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, como "$" e "%"
Ano do modelo do veículo	Tamanho limite para preenchimento	4 caracteres
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, como "$" e "%"

Casos de teste – Passo 2

Caso de teste	Procedimento de teste	Resultado
Calcular coberturas e seus respectivos preços	Informar todos os parâmetros obrigatórios do Passo 1 e solicitar a execução do Passo 2.	O sistema deverá apresentar uma lista de coberturas e preços de acordo com os dados informados no Passo 1
Passagem de informações de um passo para outro	O Passo 2 deverá passar os seguintes parâmetros para o Passo 3: Lista de coberturas selecionadas.	Se os parâmetros não forem passados corretamente, o valor total do seguro não será calculado corretamente.
Acesso à base de dados	Forçar a queda do acesso a base de dados antes da execução do passo.	O sistema deve exibir a mensagem: “Erro no acesso a base de dados. Por favor, tente mais tarde”.
Dados obrigatórios	Parâmetros obrigatórios: pelo menos uma cobertura deve ser selecionada. Procedimento: não selecionar nenhuma cobertura.	O sistema deve exibir a mensagem: “Os seguintes parâmetros não foram informados: Lista de coberturas” e voltar para a tela do Passo 2.
Valores inválidos	Nenhum parâmetro com restrição de valores	

Casos de teste – Passo 3

Caso de teste	Procedimento de teste	Resultado
Calcular preço total do seguro	No Passo 2 selecionar as coberturas e solicitar a execução do Passo 3.	O sistema deverá apresentar o preço total do seguro de acordo com as coberturas selecionadas no Passo 2.
Passagem de informações de um passo para outro	Como este é o último passo não existe passagem de parâmetros para passos seguintes.	
Acesso à base de dados	Forçar a queda do acesso a base de dados antes da execução do passo.	O sistema deve exibir a mensagem: “Erro no acesso a base de dados. Por favor, tente mais tarde”.
Dados obrigatórios	Nenhum parâmetro obrigatório	
Valores inválidos	Nenhum parâmetro com restrição de valores	

Conseqüências

- Diminuição de esforço para a elaboração da especificação do testes;
- Garantia desejada de cobertura dos testes; e
- Facilidade e agilidade na execução dos testes.

Padrões relacionados

- Não se aplica.

Usos Conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.5 Metapadrão Transação

Contexto

Projetos de desenvolvimento de sistemas de informação que requerem a criação dos produtos de trabalho gerados nas disciplinas de requisitos, análise e projeto, implementação e testes para operações que são executadas como um comando atômico, que processa várias transações.

Problema

Como tratar operações que são executadas como um comando atômico, que processa várias transações nas diversas fases de um ciclo de vida de construção de *software*, de forma integrada e consistente?

Solução

Este metapadrão descreve diretrizes para organizar operações de comando. A operação deve ser disparada por uma parametrização. Manter o usuário informado da evolução das transações e tratar a consistência entre elas.

Padrões que implementam o metapadrão

- Caso de Uso Transação; e
- Teste Transação.

4.5.1 Padrão Caso de Uso Transação

Contexto

Documentação dos requisitos de operações que são tratadas como um comando atômico que processa várias transações. Tipicamente operações *batch* e operações que requerem apenas um comando de inicio do caso de uso pelo usuário, tendo pouca entrada de dados e iteração com o sistema.

Problema

Como documentar os requisitos de operações que possuam execução de longa duração ou que são executados em formato de comando atômico, dando ênfase aos requisitos especiais dessas operações?

Forças

- Transações que ocorrem freqüentemente em sistemas de informação possuem várias características em comum e é importante que sejam documentadas de forma uniforme, facilitando o entendimento dos casos de uso;
- O usuário necessita de informação sobre o progresso e o tempo estimado para a conclusão da operação;
- O usuário pode não ter familiaridade com a complexidade da tarefa;
- Transações complexas podem envolver algoritmos e cálculos; e
- Durante a operação, o usuário pode decidir interromper a transação.

Solução

- Os requisitos devem documentar a duração média do tempo de execução da operação;
- O **Fluxo básico** descreve que atributos devem ser fornecidos para a execução da operação, indicando quais são obrigatórios. Deve indicar que existe uma opção de cancelamento que pode ser solicitada a qualquer momento;
- O **Fluxo básico** descreve a condição de início. Condições de início indicam os eventos que provocam a execução do caso de uso; e
- Os requisitos especiais descrevem como o progresso da operação será apresentado. O progresso é o momento restante para o término, número das unidades

processadas ou porcentagem do trabalho feita. Tipicamente, deve ser fornecido para o usuário o status da execução da operação, informando se a operação ainda está sendo executada e quanto tempo o usuário necessitará esperar.

Estrutura

Fluxo básico

1. Este fluxo inicia quando o <*nome do ator*> solicita executar a <*nome da transação*>. <*descrever a condição de início do caso de uso*>.
2. O sistema solicita o preenchimento dos seguintes dados:
   - <*lista de atributos de parâmetro para a transação*>.
3. O <*nome do ator*> preenche os dados solicitados no passo 2 e confirma a execução da operação.
4. O sistema executa a operação:
   - <*Operações, indicações de algoritmos e de cálculos executados na operação*>.

Requisitos especiais

- O progresso da operação deverá ser apresentado em <*descrever a unidade ou formato em que será apresentado o progresso da operação*>.

Regras de negócio

- Descrição de algoritmos e cálculos eventualmente utilizados na operação.

Exemplo do Padrão

Este exemplo apresenta o Caso de uso *Transferir Chamado* de um sistema de *Telemarketing*. O objetivo deste caso de uso é transferir um ou mais chamados de um *Operador de Telemarketing* para outro.

Fluxo básico

1. Este fluxo inicia quando o *Operador de Telemarketing* solicita transferir um chamado.
2. O sistema solicita o preenchimento dos seguintes dados:
   - * Número dos chamados. (Campo de escolha múltipla).

- * Novo *Operador de Telemarketing* responsável pelo chamado. (Campo de escolha fechada. Valores possíveis: todos os *Operadores de Telemarketing* ativos). Esse campo deve aparecer em ordem alfabética pelo nome do *Operador*.
- * Descrição. Este campo deve conter a descrição do histórico da transferência.

3. O *Operador de Telemarketing* preenche os dados solicitados no passo 2 e confirma a execução da operação.

4. O sistema executa as seguintes operações:
   - Obtém o login do usuário corrente e atribui ao campo responsável pela transferência do chamado.
   - Obtém a data e hora corrente e atribui ao campo data de criação do histórico.
   - Atribui ao identificador do tipo do histórico o valor "transferência".
   - O sistema realiza a transferência do chamado salvando os dados informados pelo *Operador de Telemarketing* no passo 2 e obtidos pela aplicação no passo 4.

Requisitos especiais

- O progresso da operação deverá ser apresentado em % (percentual), que deverá ser calculado considerando quantos chamados já foram transferidos em relação ao total de chamados selecionado. Por exemplo: o *Operador de Telemarketing* selecionou 10 (dez) chamados para serem transferidos. Quando o sistema estiver efetuado a transferência de 2 (dois) chamados o progresso da operação será 20% (vinte por cento).

Regras de negócio

- Não se aplica.

Conseqüências

- O retorno sobre o status da execução da transação é fornecido; e
- Os passos da transação, algoritmos e cálculos são documentados de forma clara.

Variantes

- Em transações curtas, o tratamento do progresso da operação pode ser suprimido.

Padrões relacionados

- Padrão **Documentação de Atributos**:

  Utilizado no **Fluxo básico** para listar os filtros.

Usos conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.5.2 Padrão Teste Transação

Contexto

Especificação dos casos de teste de operações que são tratadas como um comando atômico que processa várias transações. Tipicamente, operações *batch* e operações que requerem apenas um comando de inicio do caso de uso pelo usuário, tendo pouca entrada de dados e iteração com o sistema.

Problema

Como especificar os casos de teste de operações que são executadas em formato de comando atômico?

Forças

- A transação pode falhar durante o processamento;
- Em caso de falha, a aplicação deve manter a consistência dos dados; e
- Os casos de teste devem cobrir casos de sucesso e falha da transação.

Solução

Defina os casos de teste e idéias de teste para situações de sucesso e falha, como se segue:

- Apresentar casos de teste para verificar se o comportamento do sistema está correto na execução da transação de acordo com os parâmetros especificados, no tratamento de problema de acesso à base de dados e no tratamento de preenchimento de parâmetros obrigatórios e valores inválidos;
- Apresentar a casos de teste para verificar o comportamento do sistema no caso de falha ou cancelamento durante a execução da transação e procedimento para verificar a consistência dos dados; e

- Apresentar idéias de teste para verificar o tamanho limite para preenchimento, valores não permitidos e restrições específicas dos tipos de dados, para cada parâmetro.

Estrutura

Casos de teste

Caso de teste	Procedimento de teste	Resultado
Executar transação de acordo com os parâmetros especificados	*<definir procedimento para a execução da transação>*	*<definir critérios para verificar o resultado>*
Falha ou cancelamento durante a execução da transação	*<definir procedimento para simular falha ou cancelamento durante a execução da transação>*	*<definir critérios para verificar o resultado e a consistência dos dados e descrever qual o comportamento esperado do sistema em caso de falha ou cancelamento da transação>*
Acesso à base de dados	*<definir procedimento para simular problema no acesso ao banco de dados>*	*<descrever qual o comportamento esperado do sistema em situações de falha de acesso à base de dados>*
Dados obrigatórios	*<listar os parâmetros que são obrigatórios e informar o procedimento para o teste>*	*<descrever qual o comportamento esperado do sistema no caso de não preenchimento dos parâmetros listados>*
Valores inválidos	*<listar os parâmetros no qual os valores têm que ser validados e informar o procedimento para o teste >*	*<descrever qual o comportamento esperado do sistema no caso de preenchimento de valores inválidos>*

Idéias de teste

Nome do parâmetro	Validação	Valor
<nome do parâmetro >	Tamanho limite para preenchimento	*<descrever o tamanho limite para preenchimento>*
<nome do parâmetro >	Valores não permitidos	*<listar todos os valores não permitidos ou uma faixa de valores não permitidos>*
<nome do parâmetro >	Restrições específicas	*<se existir, identificar restrições específicas do tipo de dados do parâmetro>*

Exemplo do Padrão

Este exemplo apresenta os casos de teste do Caso de uso Transferir Chamado de um sistema de *Telemarketing*.

Casos de teste

Caso de teste	Procedimento de teste	Resultado
Executar transação de acordo com os parâmetros especificados	Informar os Números dos chamados, Nome do novo *Operador de Telemarketing* responsável pelo chamado e Descrição e solicitar a execução da transação.	O sistema deve apresentar o progresso da transação e ao término deve ter executado as seguintes operações: - Atribuir ao campo responsável pela transferência dos chamados, o *login* do usuário corrente; - Atribuir ao campo data de criação do histórico a data e hora corrente; - Atribuir ao identificador do tipo do histórico o valor "transferência", e - Salvar os dados referentes à transferência dos chamados informados pelo *Operador de Telemarketing*.
Falha do durante a execução da transação	Iniciar a transação e em seguida provocar a queda do servidor de aplicação ou desligar o computador.	Verificar se o sistema não processou nenhuma das operações descritas no caso de teste *Executar transação de acordo com os parâmetros especificados*.
Acesso à base de dados	Forçar a queda do acesso à base de dados antes da execução da transação.	O sistema deve exibir a mensagem: "Erro no acesso a base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Parâmetros obrigatórios: Número dos chamados, Nome do novo *Operador de Telemarketing* responsável pelo chamado e Descrição. Procedimento: deixar em branco os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Número dos chamados, Nome do novo *Operador de Telemarketing* responsável pelo chamado e Descrição" e voltar para a tela de execução da transação.
Valores inválidos	Parâmetros com restrições de valores: Número dos chamados e Descrição. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Número dos chamados e Descrição" e voltar para a tela de execução da transação.

Idéias de teste

Nome do parâmetro	Validação	Valor
Número do chamado	Tamanho limite para preenchimento	10 caracteres.
	Valores não permitidos	Caracteres alfa numéricos e caracteres especiais, com exceção do caractere "–".
	Restrições específicas	O formato do número do chamado deve seguir a seguinte regra: *xxxx-xxxxx*.
Descrição	Tamanho limite para preenchimento	200 caracteres.

Conseqüências

- Diminuição de esforço para a elaboração da especificação dos testes;
- Garantia desejada da cobertura dos testes; e
- Facilidade e agilidade na execução dos testes.

Padrões relacionados

- Não se aplica.

Usos Conhecidos

- Vide padrão **Caso de Uso CRUD** (seção 4.2.1).

4.6 Conclusão

Neste capítulo, apresentamos um catálogo de metapadrões e padrões, que é composto por quatro metapadrões, cinco padrões de requisitos, quatro padrões de projeto e quatro padrões de teste.

Esses metapadrões e padrões serão utilizados pelo modelo proposto no próximo capítulo, em que é apresentado um modelo para aplicação de padrões ao longo do ciclo de vida de desenvolvimento de *software*. São apresentadas também aplicações reais deste modelo em uma empresa de tecnologia da informação.

Capítulo 5 Um modelo para aplicação de metapadrões e padrões no desenvolvimento de *software*

Neste capítulo, é apresentada a proposta do modelo para aplicação de metapadrões e padrões para o desenvolvimento de *softwares* e trabalhos relacionados a essa proposta.

5.1 Introdução

A última década testemunhou um interesse crescente em padrões de *software* tanto na academia quanto na indústria. Padrões surgiram como uma técnica efetiva para documentar e comunicar a experiência através das diferentes fases do desenvolvimento de *software* (GAMMA *et al.*, 1995). Contudo, o potencial dos padrões ainda não foi explorado completamente.

Existe uma série de questões que limitam a usabilidade dos padrões. Essas questões podem ser classificadas em dois tipos: questões de especificação, e questões de desenvolvimento (HAMZA e FAYAD, 2005).

As questões de especificação estão relacionadas à forma pela qual os padrões são estruturados, especificados e formalizados. Por exemplo, o vocabulário empregado na descrição dos padrões, os *templates* adotados, as diversas formas de classificação, etc.

As questões de desenvolvimento, por outro lado, estão relacionadas com as dificuldades que surgem, quando temos que usar e integrar padrões ao desenvolvimento de *software*. Entre essas questões temos a redundância (diversos padrões que visam solucionar o mesmo problema). Entre as dificuldades encontradas temos: dificuldade na seleção do padrão a ser aplicado, na composição (processo de relacionar diferentes padrões de mesma classificação para construírem componentes), e na integração de padrões (processo de integrar padrões de diferentes classificações). Por exemplo, como um padrão de requisito que foi usado na fase de requisitos pode estar integrado com um padrão de projeto, que deverá ser usado na fase de análise e projeto.

Segundo Hamza e Fayad (2005), as melhorias geradas pela aplicação de padrões em cada fase do ciclo de desenvolvimento de *software* irão resultar na melhoria geral do produto final. Neste contexto, o problema de integração de padrões torna-se relevante.

Neste capítulo, propomos um modelo para aplicação de padrões ao longo do ciclo de vida de desenvolvimento de *software*, abordando o problema de integração levantado em

Hamza e Fayad (2005), através do uso de metapadrões integrados a processos e metodologias de desenvolvimento de *software*.

5.2 Trabalhos relacionados

A seguir serão abordados alguns trabalhos relacionados à proposta apresenta neste capítulo.

5.2.1 Promovendo padrões com padrões

Guéhéneuc (2005) discute várias questões relacionadas a padrões e aplicação de padrões. A primeira questão abordada é sobre as várias reivindicações relacionadas à construção de sistemas usando padrões. As reivindicações a respeito do uso de padrões no desenvolvimento de *software* dizem respeito à flexibilidade, à reusabilidade, e à capacidade de entendimento dos programas implementados do ponto de vista dos desenvolvedores.

Guéhéneuc (2005) afirma que usar padrões em diversos níveis torna a implementação de um programa mais flexível e reusável. Entretanto, existe incerteza sobre o aumento ou a diminuição da capacidade de entender a implementação do programa. Por um lado, se o desenvolvedor já conhece os padrões usados, ele pode entender rapidamente as colaborações entre as várias partes do programa. Por outro lado, os desenvolvedores necessitam ser realmente capazes de entender muitos padrões, e a documentação geralmente não descreve os padrões usados e sua identificação automatizada é um problema difícil.

O segundo questionamento é sobre o conceito de sistemas de padrões. Guéhéneuc (2005) descreve um sistema de padrões como um conjunto padrões que pode promover a colaboração entre padrões sem haver conflitos. Assim, o subconjunto de padrões de projeto da Gang of Four (GAMMA *et al.*, 1995) forma um sistema de padrões porque podem ser usados sem conflitos, como por exemplo, **Abstract Factory**, **Builder**, e **Singleton**. Entretanto, todos os padrões de projeto não formam um sistema de padrões por causa de seus interesses conflitantes.

A dificuldade para definir sistema de padrão é principalmente a falta do formalismo dos padrões, em particular, nos componentes *Intenção*, *Motivação* e *Conseqüências*. Portanto, a comunidade de padrões deve formalizar melhor os padrões para fornecer meios não ambíguos de relacionar padrões uns aos outros.

Outra questão abordada diz respeito à composição de padrões. Certamente, dentro de um sistema de padrões, os padrões podem ser compostos garantindo suas qualidades, enquanto que entre diferentes sistemas, alguns padrões podem ser conflitantes e assim reduzir a flexibilidade, a reusabilidade, e a capacidade de compreensão.

Guéhéneuc (2005) realizou uma experiência com estudantes de bacharelado para identificar padrões de projeto em diversos programas. Os melhores resultados foram obtidos, quando os estudantes procuraram um padrão de projeto específico e usaram o "Mapa do padrão" do livro do GoF para procurar outros padrões relacionados, o que confronta com a idéia de composição relacionada aos sistemas de padrões.

A busca e a seleção de padrões é outra questão importante. Guéhéneuc afirma que os padrões devem ser selecionados baseado no conhecimento que temos sobre os padrões e apoiado por uma ferramenta para avaliar a adequação do padrão em um contexto específico.

Muitos trabalhos foram desenvolvidos com o objetivo se auxiliar a busca e seleção de padrões. O Grupo Hillside (HILLSIDE, 2005), que é uma organização sem fins lucrativos que busca disseminar a área de padrões de *software* e mantém o site oficial que disponibiliza um sistema de metabusca para padrões. Existem também diversos catálogos e repositórios de padrões que armazenam e auxiliam na busca de padrões (SANTOS, 2004; PORTLAND, 2005; DIEMEN, 2005; WELIE, 2005).

Segundo Santos (2004), com todas essas fontes de pesquisa fica a critério do desenvolvedor o modo como irá executar a busca e a seleção de padrões específicos que atendam às suas necessidades. Além disso, é difícil encontrar um modelo de processo que auxilie o desenvolvedor na busca e aplicação de padrões durante o desenvolvimento de sistemas.

A última questão abordada no trabalho de Guéhéneuc (2005) está relacionada às técnicas existentes para integrar padrões em um ciclo de vida de desenvolvimento de *software*. Este autor usa e promove duas técnicas para integrar padrões no ciclo de desenvolvimento: Engenharia reversa e *Refactoring*. Guéhéneuc concorda com Kerievsky (2004) que *Refactoring* é um meio eficiente para integrar padrões no ciclo de desenvolvimento de *software*.

O autor conclui que a comunidade de padrões deve promover o uso de padrões no desenvolvimento de *software* e no meio acadêmico. E ainda, deve trabalhar no formalismo dos padrões e desenvolver ferramentas para identificar e aplicar padrões.

5.2.2 Usando padrões para Particionamento em Aplicações de Objetos Distribuídos

Sardjono e Simons (2005) afirma que o uso de padrões em sistemas de *software* tem-se tornado uma norma hoje em dia. Entretanto, a aplicação de padrões não é assim tão trivial. Como Alexander (2002), Sardjono e Simons (2005) também afirma que é necessário algo além dos padrões para que as vantagens do uso dos padrões sejam aproveitadas completamente no desenvolvimento *software*. Com isso, Sardjono propõe uma técnica dirigida a padrões para ajudar desenvolvedores de aplicações de objetos distribuídos a realizar particionamento.

A técnica consiste em uma linguagem de padrões para particionamento, com doze padrões arquiteturais e de projeto, e cinco padrões de processo (SARDJONO e SIMONS, 2004). A técnica é usada conjuntamente com um método orientado a objetos existente, que serve como um *framework* para modelagem orientada a objetos.

A linguagem de padrões para particionamento permite que os projetistas apliquem os padrões selecionados no estágio de desenvolvimento correto baseado no contexto encontrado. Porém, a linguagem de padrões sozinha não é bastante para permitir que os projetistas façam decisões de projeto. Assim, os padrões de processo permitem aos projetistas selecionar padrões arquiteturais e de projeto, e ainda, obterem uma arquitetura da aplicação mais estruturada e baseada nos requisitos do sistema.

A técnica proposta por Sardjono e Simons (2004) utiliza padrões de processo para direcionar a aplicação dos padrões arquiteturais e de projeto. A iniciativa, apesar de primissora, é restrita a aplicações de objetos distribuídos e não cobre todo o ciclo de vida de desenvolvimento de *software*.

5.2.3 Um catálogo de padrões para sistemas de informação

O catálogo de padrões, **Design of Business Information Systems,** do projeto ARGUS, possui seis linguagens de padrões que tratam problemas de sistemas de informação (RENZEL, 1997). As linguagens do catálogo são:

- *Relational Database Access*;
- *Mapping Objects to Tables*;
- *Decoupling*;
- *Error Handling*;

- *Form-Based User Interface*; e
- *Client / Server Distribution*.

A linguagem de padrões ***Relational Database Access*** trata de problemas encontrado em aplicações que acessam base de dados relacional. Os padrões não são restritos a aplicações orientadas a objetos. Essa linguagem de padrões apresenta uma estrutura e um conjunto padrões de projeto que ajudam modelar a camada de acesso à base de dados para bases de dados relacionais (KELLER, 1997a).

A linguagem de padrões ***Mapping Objects to Tables*** trata de problemas que ocorrem quando uma aplicação é desenvolvida usando orientação a objetos e acessa uma base de dados relacional. Os problemas mais comuns encontrados neste mapeamento são como representar Agregação, Herança, Polimorfismo, associação entre classes e tipos de dados complexos em bases de dados relacionais (KELLER, 1997c).

A coleção de padrões ***Decoupling*** contém um conjunto de padrões que ajudam o projetista a gerenciar a dependência entre as classes de um projeto, para com isso garantir um grau de acoplamento adequado (COLDEWEY, 1997).

A linguagem de padrões ***Error Handling*** descreve padrões de projeto para aplicações orientadas a objeto que tratam dos problemas de tratamento de exceções (KELLER, 1997b).

Apesar de todos os benefícios das interfaces de usuários orientadas a objetos, há os domínios que ainda exigem o uso de interface baseada em formulários. Os sistemas de informação que devem ser capazes de suportar processamentos rápidos é um exemplo típico. A linguagem de padrões ***Form-Based User Interface*** ajuda desenvolver a arquitetura do *software* para tais sistemas (COLDEWEY e KRÜGER, 1997).

A linguagem de padrões ***Client/Server Architectures*** apresenta diversos padrões para sistemas de informação distribuídos. Os padrões são baseados na arquitetura cliente/servidor. Todos os padrões apresentados dão uma resposta à mesma pergunta: Como é distribuído um sistema de informação? Entretanto, as conseqüências de aplicar os padrões são muito diferentes em relação às forças que influenciam o projeto distribuído dos sistemas (KELLER, 1997d).

As linguagens de padrões são uma forma eficiente de relacionar e agrupar padrões, facilitando a busca e seleção dos mesmos para aplicação no desenvolvimento de sistemas.

Porém carecem de uma abordagem mais integrada ao ciclo de vida e geralmente abordam somente problemas de uma fase do ciclo de vida.

5.2.4 Melhorando a Produtividade e a Qualidade de Sistemas GIS

Sodré *et al.* (2005) descrevem como um catálogo de padrões de análise pode ser aplicado para melhorar a produtividade durante o desenvolvimento de uma aplicação GIS (*Geographical Information Systems*) e, conseqüentemente, da qualidade de suas bases de dados.

O catálogo é unido ao *ArgoCASEGEO* (LISBOA e IOCHPE, 2000), que é uma ferramenta CASE de código aberto que suporta o modelo conceitual *UMLGeoFrame* (LISBOA e IOCHPE, 2000), específico para o projeto de bases de dados geográficas.

O módulo de gerenciamento da ferramenta ArgoCASEGEO organiza todos os padrões gravados em uma arquitetura de diretório, que aumenta a eficiência ao procurar por um novo padrão de análise a ser usado. Deste modo, o tempo empregado durante a fase conceptual será diminuído e, conseqüentemente, a redução de custo será verificada. Também, a qualidade das bases de dados geográficas aumenta.

O esquema conceitual elaborado por esta ferramenta é armazenado no formato XML (*Extensible Markup Language*). Assim, pode facilmente ser acessado e usado por outros sistemas. Essa ferramenta fornece também um módulo automático de geração, capaz de gerar esquema de dados nos formatos geralmente encontrados em GIS comercial. Uma atenção especial será dada ao módulo dos padrões da análise que implementa um gerenciador para o catálogo de padrões de análise.

O uso de um catálogo de padrões de análise proposto Sodré é uma forma eficiente de agrupar padrões, facilitando a busca e a seleção dos mesmos. Porém, o catálogo limita-se a catalogar padrões de análise abordando problemas somente de uma fase do ciclo de vida.

5.2.5 Desenvolvimento de *Software* baseado na Evolução de Padrões de *Software*

Segundo Kobayashi e Saeki (1999a), o desenvolvimento de *software* pode ser considerado como a evolução dos artefatos a serem produzidos. Assim, um artefato pode ser desenvolvido a partir de outro artefato que foi produzido em uma fase anterior. Por exemplo, a atividade de codificar consiste em uma transformação da especificação do projeto em código fonte. Portanto, o processo que envolve transformar os requisitos do cliente em um

produto final pode ser considerado como um processo de desenvolvimento de *software* (KATAYAMA, 1998).

Kobayashi e Saeki (1999b) descreve uma técnica para modelar padrões de *software* para suportar desenvolvimento de *software* baseado em padrões. A seguir descrevemos, em resumo, esta técnica.

Para desenvolver um artefato, inicialmente, deve-se selecionar, a partir de um catálogo de padrões, o padrão que será utilizado e instanciar e adaptar esse padrão em um artefato concreto. Depois deste passo, baseado no artefato desenvolvido através do uso de padrões, pode-se desenvolver um novo artefato. O passo de instanciar e adaptar o padrão selecionado é expresso como sendo a função *ins* e o passo de produzir o novo artefato como a função *develop*. O processo de desenvolver o artefato#2 a partir do artefato#1, sem o uso de padrões, pode ser expresso como:

$$artefato\#2 = develop(artefato\#1)$$

Por outro lado, o processo de desenvolver o artefato#1 usando o padrão#1 pode ser definido como:

$$artefato\#1 = ins\#1(padrão\#1)$$

No caso do artefato#2, tem-se uma expressão similar:

$$artefato\#2 = ins\#2(padrão\#2)$$

Onde a função *ins*#2 é o passo de instanciar e adaptar o padrão#2 para o artefato#2. Existem algumas regras e guias para realizar a evolução. Essas regras são formalmente definidas como a função *evol*. O relacionamento entre os passos de instanciação e os padrões usando evolução pode ser especificado como a seguir:

$$ins\#2 = evol(ins\#1)$$

$$padrão\#2 = evol(padrão\#1)$$

O desenvolvimento do artefato#2 pode ser descrito da seguinte forma:

$$artefato\#2 = evol(ins\#1)(evol(padrão\#1))$$

A expressão acima expressa um estilo de desenvolvimento de *software* baseado em padrões. Nesse processo de desenvolvimento, o desenvolvedor seleciona o padrão adequado (padrão#1) e instancia esse padrão. Então esse padrão é adaptado a seu problema e assim o

artefato#1 é gerado. De acordo com o progresso do desenvolvimento do *software*, o próximo artefato pode ser obtido semi-automaticamente, através do uso da função de evolução *evol*. A Figura 5.1 ilustra a evolução de padrões no desenvolvimento de *software*.

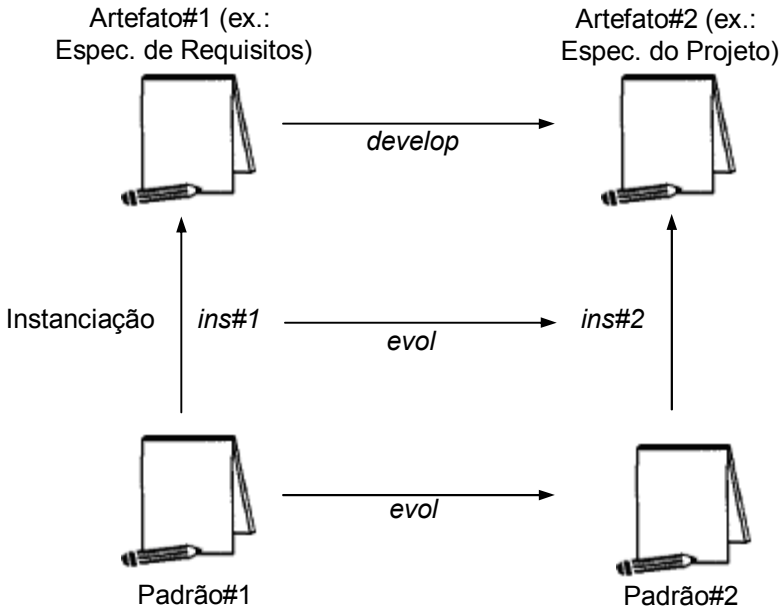


Figura 5.1: Evolução de padrões no desenvolvimento de *software* (KOBAYASHI; SAEKI, 1999b)

De forma geral, Kobayashi e Saeki (1999a) apresenta o conceito de evolução com padrões ao longo do ciclo de vida. No entanto, ele apresenta o assunto sem explorar em maior profundidade as funções de evolução nas etapas de ciclo e vida e se concentra mais em definir um modelo para suportar o conceito de evolução.

A Tabela 5.1 apresenta um resumo da comparação feita entre os trabalhos relacionados analisados e o modelo proposto por este trabalho.

Trabalhos	Integração ao ciclo de vida de desenvolvimento de *software*	Integração de padrões	Padrões que atendem as diversas fases do ciclo de vida
Usando padrões para particionamento em aplicações de objetos distribuídos	-	X	-
Um catálogo de padrões para sistemas de informação	-	X	-
Melhorando a produtividade e a qualidade de sistemas GIS	-	X	-
Catálogo descrito e modelo proposto	X	X	X

Tabela 5.1: Comparação entre os trabalhos existentes e o proposto

5.3 Um Modelo para Aplicação de Metapadrões e Padrões no Desenvolvimento de *Software*

Com o objetivo de fornecer um modelo que permita o uso e a integração de padrões com o ciclo de vida de desenvolvimento de *software*, propomos um modelo baseado no conceito de metapadrões e evolução de padrões. Esse modelo utiliza a notação de Kobayashi e Saeki (1999b) e a estende para a descrição do uso de metapadrões.

O modelo proposto pretende abordar de forma detalhada a aplicação de padrões ao longo do ciclo de vida de desenvolvimento de *software* e a evolução de produtos de trabalho[1] gerados nas etapas (disciplinas) de *Requisitos*, *Análise e Projeto*, *Implementação* e *Teste*. Partimos do princípio de que os principais produtos de trabalho no desenvolvimento de um sistema de informação são gerados nessas etapas. Para direcionar a solução para o contexto de produto de trabalho, propomos a utilização de metapadrões que, aplicados a cada uma dessas etapas, definem um conjunto de padrões que podem ser aplicados. A aplicação de padrões poderá auxiliar na construção de produtos de trabalho de acordo com o ciclo de desenvolvimento de forma mais consistente, dado que os padrões são delineados com base nos metapadrões.

Nas fases iniciais do ciclo de vida, os metapadrões associados aos diversos requisitos do sistema são identificados. À medida que o ciclo de vida evolui, os produtos de trabalho derivados são desenvolvidos, tomando-se como base o produto gerado na disciplina anterior. Em paralelo, os padrões são selecionados de acordo com o metapadrão associado ao requisito e a disciplina em execução.

A seguir, descreveremos o modelo proposto utilizando a notação de Kobayashi e Saeki (1999b). A função *evolmeta* (Figura 5.2) define como selecionar um metapadrão adequado e, a partir desse metapadrão, selecionar os padrões relacionados ao longo do ciclo de vida de desenvolvimento do *software*. Essa função é definida da seguinte forma:

$$padrão\#2 = evolmeta(padrão\#1) = ins(metapadrão\#1)evol(padrão\#1)$$

Assim, o artefato#2 pode ser obtido da seguinte forma:

$$artefato\#2 = evol(ins\#1)evolmeta(padrão\#1)$$

---

[1] Produto de trabalho envolve qualquer artefato produzido no processo de desenvolvimento: arquivos, documentos, partes de um produto, serviços, etc.

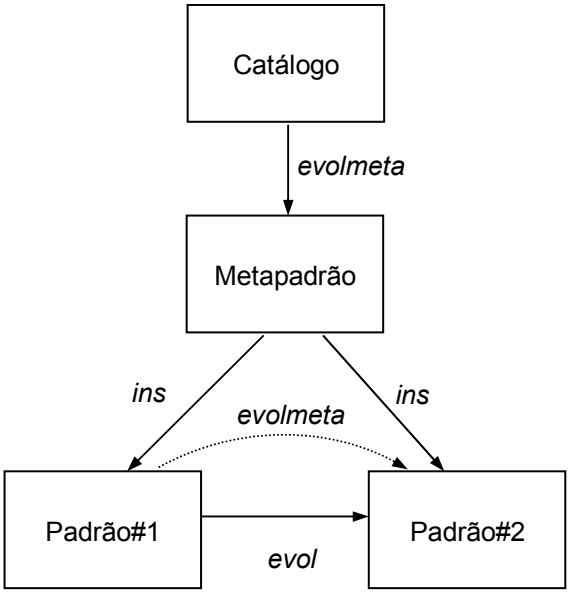


Figura 5.2: Função *evolmeta*

Inicialmente, o metapadrão deve ser selecionado com base na descrição, no tipo do caso de uso e em outras informações disponíveis. Para isso, o contexto do metapadrão deve ser analisado contra as informações disponíveis no caso de uso. Uma vez selecionado o metapadrão e associado ao caso de uso, a seleção dos padrões nas diversas fases do ciclo de desenvolvimento será baseada no metapadrão selecionado. A estrutura abaixo apresenta essa seleção.

$$metapadrão\#1 = evolmeta(catálogo)$$

Os requisitos são especificados com base nos padrões de requisitos e os artefatos gerados na elicitação dos requisitos.

$$artefatoRequisitos\#1 = evol(elicitação\#1)ins(metapadrão\#1)$$

Os requisitos servem de base para análise e projeto. Além dos próprios requisitos, o projetista terá como insumo os padrões de projeto.

$$artefatoAnálise\ \&\ Projeto\#1 = evol(artefato\ Requisitos\#1)evolmeta(padrão\ Requisitos\#1)$$

A análise e projeto serve de base para a implementação, juntamente com os padrões de implementação.

$$artefato\ Implementação\#1 = evol(artefatoAnálise\ \&\ Projeto\#1)$$
$$evolmeta(padrãoAnálise\ \&\ Projeto\#1)$$

Da mesma forma, os artefatos de testes são construídos com base nos requisitos, na análise e no projeto e em padrões de testes de forma consistente e facilitada.

$$artefatoTeste\#1 = evol(artefato\,Requisito\#1)evol(artefatoAnálise\,\&\,Projeto\#1)$$
$$evolmeta(padrão\,Requisitos\#1)evolmeta(padrãoAnálise\,\&\,Projeto\#1)$$

A Figura 5.3 mostra a geração dos produtos de trabalho no ciclo de desenvolvimento de *software* baseado na evolução de padrões.

Aplicando-se o modelo em um sistema desenvolvido no processo de desenvolvimento orientado a objetos (RUP, por exemplo), inicialmente iremos identificar o escopo do sistema através do modelo de casos de uso. Para isso, os casos de uso devem ser identificados com uma descrição inicial. A partir dessa descrição, o caso de uso é associado a um metapadrão, durante a execução da atividade de especificação de caso de uso na disciplina de *Requisitos*.

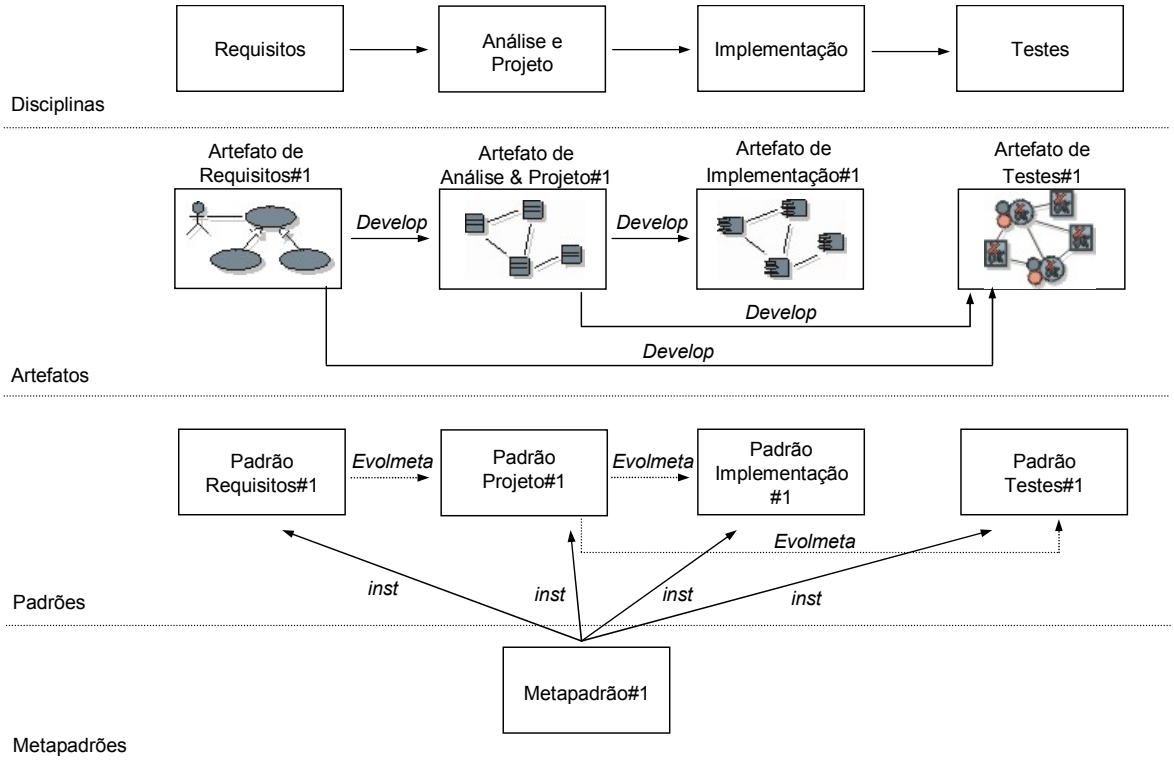


Figura 5.3: Geração dos produtos de trabalho no ciclo de desenvolvimento

Assim, um padrão de requisitos associado ao metapadrão selecionado é identificado com base na adequação do padrão ao contexto e problema, e usado na elaboração da especificação do caso de uso. Em seguida, durante a disciplina de *Análise e Projeto*, os padrões de projeto associados ao metapadrão são verificados para utilização na criação do

projeto do caso de uso. Este processo deve ser repetido até se chegar à seleção do padrão de teste para criação da especificação de testes (Figura 5.4).

Por exemplo, quando um caso de uso é associado ao metapadrão **Meta-CRUD**, durante a disciplina de *Requisitos*, o padrão **Caso de Uso CRUD** é selecionado e utilizado na elaboração da especificação do caso de uso. Em seguida, durante a **Análise e Projeto** do caso de uso, os padrões de projeto associados ao metapadrão **Meta-CRUD** são verificados para uso na criação do projeto do caso de uso. O padrão é então selecionado com base na adequação do padrão ao contexto e problema. Esse processo é repetido até se chegar à seleção do padrão **Teste CRUD** para criação da especificação de testes.

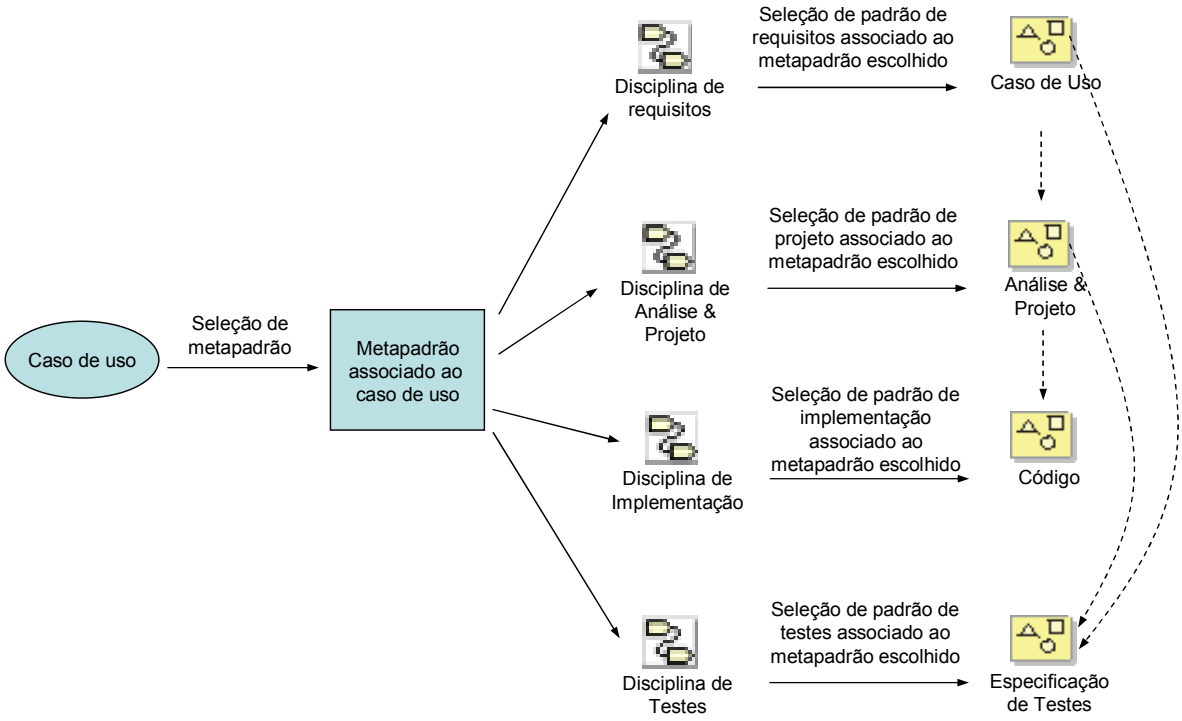


Figura 5.4: Aplicação de metapadrões e padrões no processo de desenvolvimento

A Figura 5.5 apresenta os metapadrões e padrões propostos e seus relacionamentos de acordo com as disciplinas fundamentais do ciclo de vida de desenvolvimento. Cada um dos padrões detalha como aplicar uma solução geral indicada nos metapadrões.

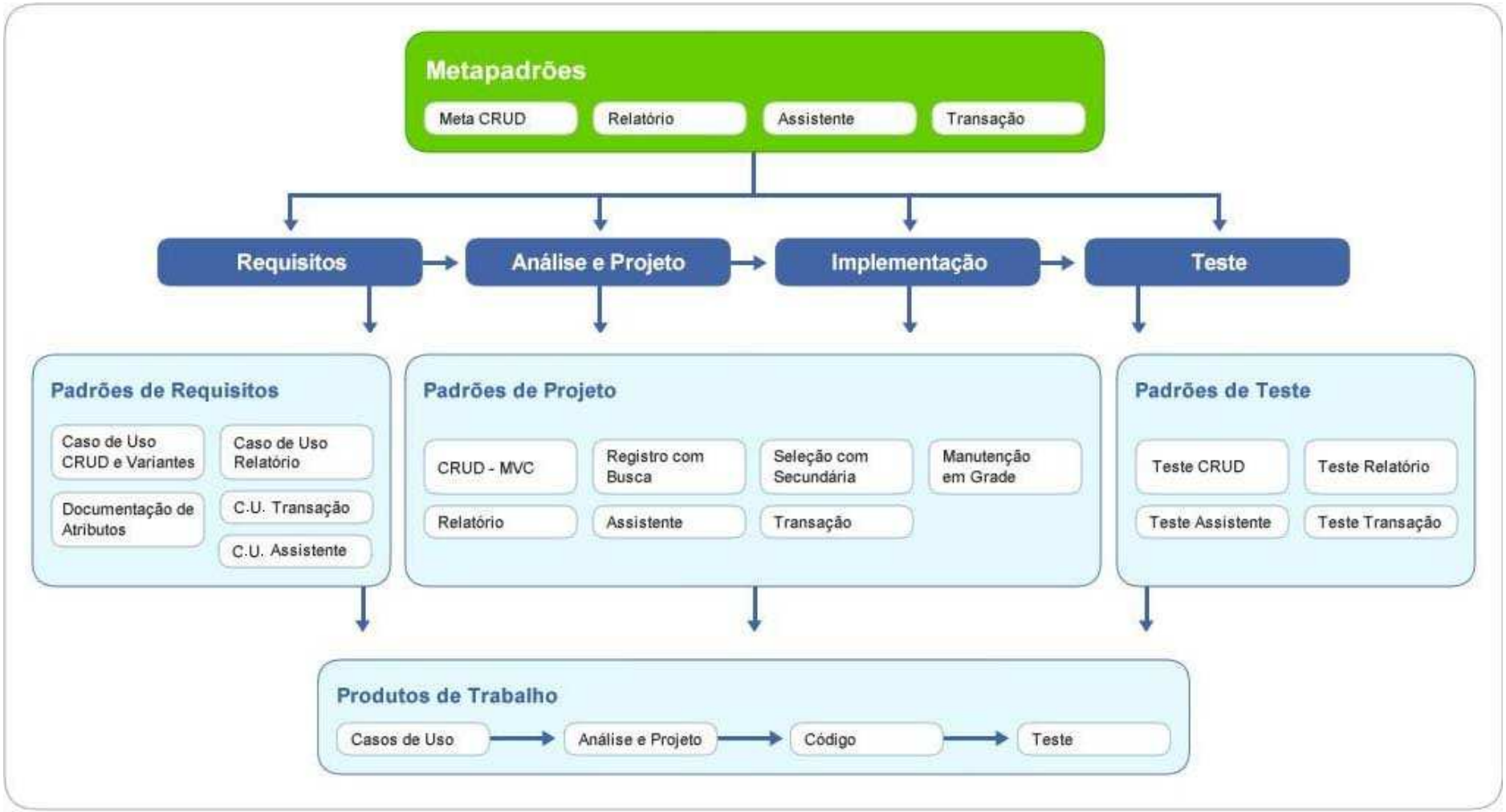


Figura 5.5: Relacionamento entre os metapadrões e os padrões no ciclo de desenvolvimento

5.4 Aplicação do modelo

Nesta seção, iremos demonstrar a aplicação do modelo proposto e dos padrões descritos no Capítulo 4. Esses padrões foram utilizados em mais de 12 projetos reais de *software*, na empresa de soluções de tecnologia denominada Instituto Atlântico.

5.4.1 Caracterização da Empresa e dos Projetos

O Instituto Atlântico é uma instituição de pesquisa e desenvolvimento (P&D), fundada em novembro de 2001. É uma organização certificada como SW-CMM nível 2, desde outubro de 2003 e ISO 9001:2000 em junho de 2005. Este empresa foi certificada CMMI nível 3 em fevereiro de 2006.

O escopo de atuação do Instituto abrange pesquisa e desenvolvimento de soluções de *software*, hardware, e serviços de consultoria na área de tecnologia da informação e telecomunicações. Essas soluções são desenvolvidas mediante projetos próprios ou específicos para clientes, e podem ter como objetivo uma nova tecnologia, um novo produto ou a atualização de algum produto já existente. Os projetos desenvolvidos pertencem a diversas áreas, possuem tamanhos variados e são desenvolvidos em diferentes plataformas (J2EE, J2ME, .NET, C, C++, etc.).

Selecionamos quatro casos de uso, desenvolvidos nos Instituto Atlântico, que pertencem ao projeto que aqui iremos denominar de Projeto Levi, para demonstrar a aplicação dos metapadrões **Meta_CRUD, Relatório, Assistente e Transação**. Por questão de sigilo de informações alguns detalhes sobre os casos de uso e informações do projeto serão omitidos.

O Projeto Levi foi um projeto de porte médio, com duração de 12 meses, tendo sido desenvolvido em uma interface Web, com as seguintes características:

- *Tecnologia*: J2EE;
- *Sistema operacional*: Linux;
- *Ferramenta de desenvolvimento*: Eclipse;
- *Ferramenta de banco de dados*: PostgreSQL;
- *Ferramenta de controle de versão*: CVS;
- *Ferramentas de análise e projeto*: Entreprise Architect; e
- *Ferramentas de requisitos*: Requisite Pro, e Documentos Word.

Foi prevista a alocação de oito membros para a equipe do projeto, incluindo os papéis de coordenador de projeto e dos líderes de requisitos e configuração. O coordenador do projeto executava as atividades de planejamento e acompanhamento do projeto.

O líder de configuração gastou em torno de 50% de seu tempo com as atividades de gerência de configuração. O líder de requisitos foi alocado totalmente às atividades de gestão de requisitos.

A seguir, o modelo proposto é aplicado através dos padrões apresentados anteriormente.

5.4.2 Metapadrão Meta-CRUD

Para ilustrar a aplicação do modelo iremos usar o Caso de uso **Manter Usuário**. A entidade Usuário possui as seguintes características:

- É uma entidade simples;
- Possui poucos atributos e somente um relacionamento;
- Possui pequeno volume de dados; e
- Possui os seguintes atributos: Nome, Login, Senha e Perfil.

Com base nas características do caso de uso em questão, selecionamos o Metapadrão **Meta-CRUD**, para tratar a manutenção da entidade Usuário. Esta seleção é formalizada como se segue:

$$metapadrãoMeta - CRUD = evolmeta(catálogo)$$

A seguir, iremos apresentar o processo de geração dos artefatos das disciplinas (etapas) de *Requisitos*, *Análise e Projeto*, e *Teste*, e a instanciação de cada artefato de acordo com o padrão selecionado.

Disciplina de Requisitos: <u>Padrão Caso de Uso CRUD</u>

Inicialmente, devemos selecionar o padrão de requisitos que será usado para realizarmos a especificação do caso de uso. O padrão de requisitos que está associado ao metapadrão **Meta-CRUD** é o padrão **Caso de Uso CRUD**. A seguir, temos a formalização da evolução do artefato de caso de uso e do padrão de requisitos.

Evolução do Artefato
$CasodeUsoManterUsuário = evol(elicitaçãoUsuário)ins(metapadrãoMeta - CRUD)$
Evolução do Padrão
$PadrãoCasodeUsoCRUD = ins(metapadrãoMeta - CRUD)$

A especificação do Caso de uso **Manter Usuário** encontra-se no Apêndice I.

Disciplina de Análise e Projeto: Padrão Manutenção em Grade

Nesta disciplina devemos selecionar o padrão de projeto que será usado para realizarmos o detalhamento do projeto do caso de uso. Os padrões de projeto que estão associados ao metapadrão **Meta-CRUD** são os padrões **CRUD-MVC**, **Registro com Busca**, **Manutenção em Grade**, e **Seleção com Secundária**. De acordo com o contexto de cada padrão e as características do caso de uso, devemos selecionar o padrão adequado. O padrão **Manutenção em Grade** trata da manutenção de entidade simples que possuam poucos atributos e/ou relacionamentos. Assim, após análise realizada, este padrão foi considerado o mais adequado para realizar o projeto deste caso de uso. A seguir, temos a formalização da evolução do artefato de projeto e do padrão de projeto:

Evolução do Artefato
$Pr ojetoManterUsuário = evol(casodeUsoManterUsuário)$ $evolmeta(padrãoCasodeUsoCRUD)$
Evolução do Padrão
$PadrãoManutençãoemGrade = evolmeta(padrãoCasodeUsoCRUD)$

O diagrama de classes e o diagrama de seqüência referentes à operação de Inclusão do Caso de uso **Manter Usuário** encontram-se no Apêndice II.

Disciplina de Teste: Padrão Teste CRUD

Nesta disciplina, devemos selecionar o padrão de teste que será usado para realizarmos a especificação de teste do caso de uso. O padrão de teste que está associado ao metapadrão **Meta-CRUD** é o padrão **Teste CRUD**. A seguir, temos a formalização da evolução do artefato de teste e do padrão de teste:

Evolução do Artefato
$EspecificaçãodeTesteManterUsuário = evol(casodeUsoManterUsuário)$ $evol(projetoManterUsuário)evolmeta(padrãoCasodeUsoCRUD)$ $evolmeta(padrãoManutençaoemGrade)$
Evolução do Padrão
$PadrãoTesteCRUD = evolmeta(padrãoCasodeUsoCRUD)$ $evolmeta(padrãoManutençãoemGrade)$

A especificação de teste do Caso de uso **Manter Usuário** encontra-se no Apêndice III.

5.4.3 Metapadrão Relatório

Para ilustrar a aplicação do Metapadrão **Relatório** iremos utilizar o Caso de uso **Gerar Relatório de Obras de Arte**. Esse caso de uso tem como objetivo listar todas as obras de artes cadastradas no sistema de um museu com base nos seguintes filtros: categoria da obra (pintura, escultura, fotografia etc), técnica da obra (óleo sobre tela, aquarela, etc.) e artista.

O metapadrão **Relatório** trata da geração de relatórios nas diversas fases do ciclo de vida de desenvolvimento de *software*. A seleção do metapadrão é formalizada como se segue:

$$metapadrão\ Relatório = evolmeta(catálogo)$$

A seguir, iremos apresentar o processo de geração dos artefatos das disciplinas (etapas) de *Requisitos*, *Análise e Projeto*, e *Teste*, e a instanciação de cada artefato de acordo com o padrão selecionado.

Disciplina de Requisitos: <u>Padrão Caso de Uso Relatório</u>

Inicialmente, devemos selecionar o padrão de requisitos que será utilizado para realizarmos a especificação do caso de uso. O padrão de requisitos que está associado ao metapadrão **Relatório** é o padrão **Caso de Uso Relatório**. A seguir temos a formalização da evolução do artefato de caso de uso e do padrão de requisitos:

Evolução do Artefato
$CasodeUsoGerar\ RelatórioObrasArte = evol(elicitação\ RelatórioObrasArte)ins(metapadrão\ Relatório)$
Evolução do Padrão
$Padrão\ Relatório = ins(metapadrão\ Relatório)$

A especificação do Caso de uso **Gerar Relatório de Obras de Arte** encontra-se no Apêndice IV.

Disciplina de Análise e Projeto: <u>Padrão MVC</u>

Nesta disciplina, devemos selecionar o padrão de projeto que será usado para realizarmos o detalhamento do projeto do caso de uso. O catalogo proposto não contém padrões de projeto para casos de uso Relatório. Contudo, pelas características deste artefato, existem diversos padrões de projeto que podem ser selecionados para realizá-lo. Assim, selecionamos o padrão **MVC**, que se adeque às características do caso de uso em questão. A seguir, temos a formalização da evolução do artefato de projeto e do padrão de projeto:

Evolução do Artefato
$Pr ojetoGerar\ RelatórioObraArte = evol(casodeUsoGerar\ RelatórioObraArte)$ $evolmeta(padrãoCasodeUso\ Relótorio)$
Evolução do Padrão
$PadrãoMVC = evolmeta(padrãoCasodeUso\ Relatório)$

O diagrama de classes do Caso de uso **Gerar Relatório de Obras de Arte** encontra-se no Apêndice V.

Disciplina de Teste: <u>Padrão Teste Relatório</u>

Na disciplina de teste, devemos selecionar o padrão de teste que será usado para realizarmos a especificação de teste do caso de uso. O padrão de teste que está associado ao metapadrão **Relatório** é o padrão **Teste Relatório**. A seguir, temos a formalização da evolução do artefato de teste e do padrão de teste:

Evolução do Artefato
$EspecificaçãodeTesteGerar\,\mathrm{Re}latórioObrasArte =$ $evol(casodeUsoGerar\,\mathrm{Re}latórioObrasArte)evol(projetoGerar\,\mathrm{Re}latórioObrasArte)$ $evolmeta(padrãoCasodeUso\,\mathrm{Re}latório)evolmeta(padrãoMVC)$
Evolução do Padrão
$PadrãoTeste\,Re latório =$ $evolmeta(padrãoCasodeUso\,Re latório)evolmeta(padrãoMVC)$

A especificação de teste do Caso de uso **Gerar Relatório de Obras de Arte** encontra-se no Apêndice VI.

5.4.4 Metapadrão Transação

Para ilustrar a aplicação do Metapadrão **Transação**, iremos usar o caso de uso **Exportar Obras para Catálogo**. Este caso tem uso tem como objetivo exportar os dados das obras de arte selecionadas, no formato padrão do catálogo, para que as mesmas sejam incluídas na publicação do catálogo.

O metapadrão **Transação** trata de operações que são executadas como um comando atômico que processa várias transações. Tipicamente operações *batch* e operações que requerem apenas um comando de início, tendo pouca entrada de dados e iteração com o sistema. A seleção do metapadrão é formalizada como se segue:

$$metapadrãoTransação = evolmeta(catálogo)$$

A seguir iremos apresentar o processo de geração dos artefatos das três disciplinas consideradas, e a instanciação de cada artefato de acordo com o padrão selecionado.

Disciplina de Requisitos: <u>Padrão Caso de Uso Transação</u>

Inicialmente, devemos selecionar o padrão de requisitos que será usado para realizarmos a especificação do caso de uso. O padrão de requisitos que está associado ao metapadrão **Transação** é o padrão **Caso de Uso Transação**. A seguir, temos a formalização da evolução do artefato de caso de uso e do padrão de requisitos:

Evolução do Artefato
$CasodeUsoExportarObrasCatálogo = evol(elicitaçãoExportarObrasCatálogo)ins(metapadrãoTransação)$
Evolução do Padrão
$PadrãoTransação = ins(metapadrãoTransação)$

A especificação do Caso de uso **Exportar Obras para Catálogo** encontra-se no Apêndice VII.

Disciplina de Análise e Projeto: Padrão MVC

Nesta disciplina devemos selecionar o padrão de projeto que será usado para realizarmos o detalhamento do projeto do caso de uso. O catalogo proposto não contém padrões de projeto para casos de uso Transação. Contudo, pelas características deste artefato, existem diversos padrões de projeto que podem ser selecionados para realizarmos este artefato. Assim, selecionamos o padrão **MVC**. A seguir temos a formalização da evolução do artefato de projeto e do padrão de projeto:

Evolução do Artefato
$Pr ojetoExportarObraCatálogo = evol(casodeUsoExportarObraCatálogo)$ $evolmeta(padrãoCasodeUsoTransação)$
Evolução do Padrão
$PadrãoMVC = evolmeta(padrãoCasodeUsoTransação)$

O diagrama de classes do Caso de uso **Exportar Obras para Catálogo** encontra-se no Apêndice VIII.

Disciplina de Teste: Padrão Teste Transação

Na disciplina de teste, devemos selecionar o padrão de teste que será usado para realizarmos a especificação de teste do caso de uso. O padrão de teste que está associado ao metapadrão **Transação** é o padrão **Teste Transação**. A seguir, temos a formalização da evolução do artefato de teste e do padrão de teste:

Evolução do Artefato
$EspecificaçãodeTesteExportarObrasCatálogo = evol(casodeUsoExportarObrasCatálogo)evol(projetoExportarObrasCatálogo) evolmeta(padrãoCasodeUsoTransação)evolmeta(parãoMVC)$
Evolução do Padrão
$PadrãoTesteTransação = evolmeta(padrãoCasodeUsoTransação)evolmeta(padrãoMVC)$

A especificação de teste do Caso de uso **Exportar Obras para Catálogo** encontra-se no Apêndice IX.

5.4.5 Metapadrão Assistente

Para ilustrar a aplicação do Metapadrão **Assistente**, iremos usar o Caso de uso **Configurar Site**. Esse caso tem uso tem como objetivo configurar o site que publicará as informações de um museu na Web. O caso de uso permite escolher um modelo, configurar um esquema de cores, alterar características de fontes, fundo, posicionamento de texto, bordas, alinhamento de objetos e títulos.

O metapadrão **Assistente** trata o processamento de operações complexas que são executadas em diversos passos, onde decisões ou dados necessitam serem informados em cada passo, através da interação com o usuário. A seleção do metapadrão é formalizada como se segue:

$$metapadrãoAssistente = evolmeta(catálogo)$$

A seguir, iremos apresentar o processo de geração dos artefatos das disciplinas (etapas) de *Requisitos*, *Análise e Projeto*, e *Teste*, e a instanciação de cada artefato de acordo com o padrão selecionado.

Disciplina de Requisitos: <u>Padrão Caso de Uso Assistente</u>

Inicialmente, devemos selecionar o padrão de requisitos que será usado para realizarmos a especificação do caso de uso. O padrão de requisitos que está associado ao metapadrão **Assistente** é o padrão **Caso de Uso Assistente**. A seguir, temos a formalização da evolução do artefato de caso de uso e do padrão de requisitos.

Evolução do Artefato
$CasodeUsoConfigurarSite = evol(elicitaçãoConfigurarSite)ins(metapadrãoAssistente)$
Evolução do Padrão
$PadrãoAssistente = ins(metapadrãoAssistente)$

A especificação do Caso de uso **Configurar Site** encontra-se no Apêndice X.

Disciplina de Análise e Projeto: Padrão MVC

Nesta disciplina, devemos selecionar o padrão de projeto que será usado para realizarmos o detalhamento do projeto do caso de uso. O catalogo proposto não contém padrões de projeto para casos de uso Assistente. Contudo, existem diversos padrões de projeto que podem ser selecionados para realizarmos este artefato. Assim, selecionamos o padrão **MVC**. A seguir, temos a formalização da evolução do artefato de projeto e do padrão de projeto.

Evolução do Artefato
$Pr ojetoConfigurarSite = evol(casodeUsoConfigurarSite)$ $evolmeta(padrãoCasodeUsoAssistente)$
Evolução do Padrão
$PadrãoMVC = evolmeta(padrãoCasodeUsoAssistente)$

O diagrama de classes do Caso de uso **Configurar Site** encontra-se no Apêndice XI.

Disciplina de Teste: Padrão Teste Assistente

Na disciplina de teste, devemos selecionar o padrão de teste que será usado para realizarmos a especificação de teste do caso de uso. O padrão de teste que está associado ao metapadrão **Assistente** é o padrão **Teste Assistente**. A seguir, temos a formalização da evolução do artefato de teste e do padrão de teste:

Evolução do Artefato
$EspecificaçãodeTesteConfigurarSite = evol(casodeUsoConfigurarSite)evol(projetoConfigurarSite)\ evolmeta(padrãoCasodeUsoAssistente)evolmeta(parãoMVC)$
Evolução do Padrão
$PadrãoTesteAssistente = evolmeta(padrãoCasodeUsoAssistente)evolmeta(padrãoMVC)$

A especificação de teste do Caso de uso **Configurar Site** encontra-se no Apêndice XII.

5.5 Conclusão

Neste capítulo, propomos um modelo para aplicação de padrões integrado ao ciclo de vida de desenvolvimento de *software*. Essa proposta é uma tentativa de minimizar um dos problemas existentes devido a grande diversidade de padrões de *software* existentes, que é a ausência de um mecanismo ou metodologia que auxilie na aplicação desses padrões. No capítulo seguinte apresentamos as considerações finais deste trabalho e as possíveis contribuições em trabalhos futuros.

Capítulo 6 Conclusão

Este capítulo apresenta as principais conclusões deste trabalho, cujo objetivo foi apresentar um catálogo de metapadrões e padrões para sistemas de informação e propor um modelo para a aplicação de metapadrões e padrões ao longo do ciclo de vida de desenvolvimento de *software*.

6.1 Considerações

Desenvolver *software* usando padrões pode reduzir o custo e condensar o ciclo de vida do desenvolvimento, e simultaneamente manter a qualidade dos sistemas desenvolvidos. Entretanto, o potencial de usar padrões em sistemas de *software* não tem sido aproveitado inteiramente. Embora diversos padrões tenham sido desenvolvidos para analisar, projetar, e implementar o *software*, ainda há carência de orientações ou metodologias suficientes maduras, que forneçam uma abordagem sistemática para integrar estes diferentes tipos de padrões durante o ciclo de desenvolvimento.

Este trabalho apresentou um catálogo de metapadrões e padrões, juntamente com um modelo para a aplicação desses padrões ao longo do ciclo de vida de desenvolvimento de *software*. O catálogo contém quatro metapadrões e diversos padrões de requisitos, padrões de projeto, e padrões de teste. Os metapadrões relacionam os padrões de acordo com o problema a ser solucionado, facilitando a seleção de padrões para a elaboração de artefatos gerados nas principais etapas do ciclo de vida de desenvolvimento de *software*.

Foi também apresentado um estudo de caso do uso do modelo de metapadrões e padrões proposto para sua experimentação e validação, e para auxiliar na aplicação desse modelo em projetos de *software*.

Como considerações relevantes, apreendidas ao longo dos experimentos realizados neste trabalho, podemos destacar que:

- Existe uma vasta gama de padrões disponíveis, no entanto o uso dos padrões não é potencializado por falta de abordagens de uso integradas aos processos e metodologias de desenvolvimento de *software*;
- Abordagens que proporcionam o uso de padrões de forma consistente para os diversos produtos de trabalho do ciclo de vida, incentivam o uso de padrões e facilitam o uso de padrões;

- Metapadrões podem ser utilizados como um meio efetivo para organizar e associar os diversos tipos de padrões de *software*.
- A seleção de padrões ao longo de ciclo de vida é reduzida a um único ponto com a seleção do metapadrão no início do ciclo de vida de desenvolvimento durante a fase de requisitos.
- A formalização da transformação entre modelos pode ser útil como meio de documentar o racional das decisões de projeto ao longo do ciclo de vida de desenvolvimento.
- A rastreabilidade dos requisitos é enriquecida com o raciocínio por trás de cada evolução/geração de artefato.
- Reuso e consistência dos artefatos de documentação.

6.2 Contribuições deste Trabalho

Podem-se destacar como principais contribuições deste trabalho:

- *Catálogo de metapadrões para sistemas de informação*: fornece diretrizes para a seleção, aplicação e integração de diversos tipos de padrões de *software*, servindo de base para o modelo proposto de aplicação de padrões. Além disso, o conceito dos metapadrões proposto por Pree (1995a) foi estendido para outros tipos de padrões além dos padrões de projeto.
- *Catálogo de padrões de software para sistema de informação*: provê soluções para vários tipos de produto de trabalho ao longo do processo de desenvolvimento do *software*. Esse catálogo compreende padrões de requisitos, padrões de projeto, e padrões de teste.
- *Modelo para a aplicação de padrões*: apresenta uma abordagem estruturada e integrada ao ciclo de vida de desenvolvimento, permitindo a aplicação de padrões a vários tipos de produto de trabalho (artefatos) de forma consistente e integrada. A abordagem facilita a seleção de padrões e a construção de produtos de trabalho. O modelo pode ser aplicado para diversos ciclos de vida.
- *Aplicação do modelo utilizando o RUP*: o modelo foi aplicado ao *framework* de processos RUP (2003), que utiliza o ciclo de vida iterativo e incremental, dada a sua crescente utilização em projetos de *software* atualmente.

- *Aumento de reuso em projetos*: padrões de *software* são reconhecidos como uma técnica efetiva de reuso de soluções. Ficou evidenciado, no estudo de caso, que o modelo proposto incentiva e facilita o uso de padrões ao longo do ciclo de vida do projeto, proporcionando o aumento do grau de reuso.

6.3 Trabalhos Futuros

Dentre os trabalhos futuros que podem dar continuidade à pesquisa aqui apresentada, sugerem-se:

- Uso de MDA (*Model Driven Architecture*) para automação das transformações e evoluções de padrões e produtos de trabalho propostos. MDA (MDA, 2005) vem despontando como uma técnica efetiva para aumentar a produtividade em todo o ciclo de desenvolvimento de *software*. Esta abordagem também possibilita a especificação de *software* em alto nível e permite a criação dos diversos produtos de trabalho, incluindo o código, através de transformações que podem ser intra-modelo e inter-modelo. Na transformação intra-modelo, padrões de *software* são utilizados para refinar a solução dentro do mesmo tipo de modelo. É o caso de utilizar o padrão de projeto ***Singleton*** sobre uma classe, para fornecer a essa classe todo o arcabouço do padrão com sua aplicação automática. Nas transformações inter-modelo, os produtos de trabalho são refinados de um modelo para o outro. É o caso de gerar o código automaticamente através de um modelo de classes UML. O modelo proposto pode ser utilizado em transformações intra-modelo com o uso do catálogo de padrões e inter-modelo através de evoluções de produtos.

- Desenvolvimento de ferramentas que integrem o modelo proposto a ferramentas de desenvolvimento de *software* disponíveis. O uso dos metapadrões e padrões podem ser incorporados a ferramentas *case* de análise e projeto, bem como a ferramentas de desenvolvimento de código ou ferramentas de testes.

- Documentação e incorporação de padrões de codificação e IHC (Interface Humano Computador) ao catálogo de padrões proposto. O catálogo de padrões pode ser enriquecido com a identificação de novos padrões. Esses padrões também devem ser associados aos metapadrões, aumentando o potencial de reuso da proposta.

- Aplicação do modelo em outros projetos de *software*.

- Aplicação do modelo com outros padrões. O modelo pode ser aplicado com padrões que não pertencem ao catálogo de padrões descrito neste trabalho.

Referências bibliográficas

ADOLPH, S.; BRAMBLE, P.; COCKBURN, A.; POLS, A. **Patterns for Effective Use Cases.** Addison Wesley, Agosto, 2002. ISBN: 0-201-72184-8

ALEXANDER, C. et al. **A Pattern Language: Towns, Buildings, Construction**. Oxford University Press, New York, NY, 1977.

ALEXANDER, C. **The Nature of Order: An Essay of the Art of Building and the Nature of the Universe**. Book Two: The Process of Creating Life. The Center for Environmental Structure, 2002.

ALEXANDER, C. **The Timeless Way of Building. Oxford University Press**. New York, NY, 1979.

ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns As melhores práticas e estratégias de design**. Editora Campus, 2002.

AMBLER, S. W. **The Process Patterns Resource Page**. Disponível em: http://www.ambysoft.com/processPatternsPage.html. Acessado em: 29/09/2005.

ANDRADE, R. **Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems**. Ph.D. Thesis, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, May 2001.

ANDRADE, R.; MARINHO, F. **Diagnóstico do Reuso de Padrões de Software no Instituto Atlântico – Projeto: Métodos e Técnicas associados a Padrões de Software**. Relatório Técnico do Projeto de Pesquisa e Desenvolvimento em Colaboração Departamento de Computação / Universidade Federal do Ceará e Instituto Atlântico (Convênio CVE/P&D/005/02), Fortaleza, CE, Fev. 2003.

ANDRADE, R.; MARINHO, F.; SANTOS, M.; NOGUEIRA, R. **Uma Proposta de um Repositório de Padrões Integrado ao RUP**. Session Pattern Application (SPA), SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, Ago. 2003.

APPLETON, Brad. **Patterns and Software: Essential Concepts and Terminology**. Fevereiro de 2000. Disponível em: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html. Acessado em: 01/09/2005.

BECK, K. **Extreme Programming Explained – Embrace Change**. Addison-Wesley, 1999.

BECK, K. **Smalltalk Best Practice Patterns**. Prentice Hall, Upper Saddle River, NJ, 1997.

BITTNER, K.; SPENCE, I. **Use Case Modeling**. Addison Wesley, 2002.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO P. C.; MALDONADO, J. C. **Introdução aos Padrões de Software**. São Carlos, 2001.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern-Oriented Software Architecture**. John Wiley and Sons, New York, NY, 1996.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. **CMMI Guidelines for Process Integration and Product Improvement**. Addison-Wesley, 2004.

COCKBURN, A. **Writing Effective: Use Cases**. Addison-Wesley Boston, 2001.

COCKBURN, A.; FOWLER, M. **Question Time! About Use Cases**. In Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '98). Vancouver, British Columbia, Canada, 1998.

COLDEWEY, J. **Decoupling of Object-Oriented Systems - A Collection of Patterns**. 1997. Disponível em: http://www.objectarchitects.de/arcus/cookbook/decoupling/index.htm. Acessado em: 08/10/2005.

COLDEWEY, J.; KRÜGER, I. **Form-Based User Interface - A Pattern Language**. Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany, Siemens Technical Report 120/SW1/FB, 1997.

COPLIEN, J. O. **A Development Process Generative Pattern Language**. In Pattern Languages of Program Design. Ed. Reading, MA: Addison-Wesley, 1995, pp. 183-237.

COPLIEN, J. O. **C++ Idioms Patterns**. In Brian Foote, Neil Harrison, and Hans Rohnert, editors, Pattern Languages of Program Design 4, chapter 10, 167-197. Addison Wesley, Reading, MA, 2000.

COPLIEN, J. O. **Software Patterns**. SIGS books and Multimedia, June 1996.

Enterprise Unified Process (EUP) Home Page. Disponível em: http://www.enterpriseunifiedprocess.com/. Acessado em: 20/12/2005.

FOWLER, M. **Analysis Patterns: Reusable Object Models**. Addison-Wesley, Reading, MA, 1997.

FOWLER, M. **Implementing Analysis Patterns**. Presentation at OOP'97, 1997.

FOWLER, M; Scott, K. **UML Essencial - Um Breve Guia Para a Linguagem-PAD**. Bookman, 2005.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object Oriented Software**. Addison Wesley, Reading, MA, 1995.

GUÉHÉNEUC, Y. **Ptidej: Promoting Patterns with Patterns**. The 19th European Conference on Object-Oriented Programming. Glasgow, Scotland, 2005.

HAMZA, H.S.; FAYAD, M. E. **On the Use of Patterns for Software Development: Challenges and Opportunities**. The 19th European Conference on Object Oriented Programming. Glasgow, Scotland, 2005.

HANMER, R. **Introduction to Pattern Languages**. SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, 2003.
Institute of Electrical and Electronics Engineers. **IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries**. New York, NY: 1990.

JACOBSEN; CHRISTERSON; OVERGAARD. **Object-Oriented Software Engineering: A Use Case-Driven Approach**. Addison-Wesley, 1992.

KATAYAMA, T. **A Theoretical Framework of Software Evolution**. In Proceeding of IWPSE98, pages 1-5,1998.

KELLER, W. **Error Handling for Business Information Systems – A Pattern Language**. 1997. Disponível em: http://www.objectarchitects.de/arcus/cookbook/exhandling/index.htm. Acessado em: 08/10/2005.

KELLER, W. **Mapping Objects to Tables - A Pattern Language**. Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany, Siemens Technical Report 120/SW1/FB, 1997.

KELLER, W.; COLDEWEY, J. **Relational Database Access Layers – A Pattern Language**. Collected Papers from the PLoP'96 and EuroPLoP'96 Conferences, Washington University, Department of Computer Science, Technical Report WUCS 97-07, February 1997.

KELLER, W.; RENZEL, K. **Client/Server Architectures for Business Information Systems**. Pattern Languages of Programming Conference (PLoP), Monticello, Illinois, USA, 1997.

KERIEVSKY, J. **Refactoring to Patterns**. Addison Wesley, 1st edition, August 2004.

KLING, D.; FLODSTRÖM, S. **Metapatterns – An Overview**. Disponível em: www.idt.mdh.se/kurser/cd5130/msg/2002lp3/download/CD5130%20VT02%20Metapatterns.pdf. Acessado em: 14/10/2005.

KOBAYASHI, T.; SAEKI, M. 1999. **Pattern-Based Software Evolution**. In Proceedings of the Workshop on Object-Oriented Technology (June 14 - 18, 1999). A. M. Moreira and S. Demeyer, Eds. Lecture Notes In Computer Science, vol. 1743. Springer-Verlag, London, 77.

KOBAYASHI, T.; SAEKI, M. 1999. **Software Development Based on Software Pattern Evolution**. In Proceedings of the Sixth Asia Pacific Software Engineering Conference (December 07 - 10, 1999). Washington, DC, 18.

KONRAD, S.; CHENG, B. H. C. **Requirements Patterns for Embedded Systems**. In Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE02), Essen, Germany, September 2002.

KRUCHTEN, P. **Rational Unified Process Made Easy: A Practioner´s Guide to the RUP**. Addison-Wesley Publishers Ltd. 2003.

KRUCHTEN, P. **The Rational Unified Process: An Introduction**. Addison-Wesley, 2001.

LISBOA F., J.; SODRÉ, V. F.; DALTIO, J.; RODRIGUES Jr., M. F.; VILELA, V. M. **A CASE tool for geographic database design supporting analysis patterns**. In Proc. of Conceptual Modeling for Advanced Application Domains. 1st Int. Workshop on Conceptual Modelling for GIS (CoMoGIS – ER2004), LNCS 3289, Springer Shanghai, China, 2004.
LISBOA, J.; IOCHPE, C. **Specifying Analysis Patterns for Geographic Databases on the Basis of a Conceptual Framework**. In Procs 7th ACM GIS. 2000.

MARINHO, F. G. **Padrões para a Integração de Visões Modeladas com UML**. Dissertação de Mestrado. Fortaleza, dezembro de 2001.

MESZAROS, G.; DOBLE, J. **A Pattern Language for Pattern Writing**. Pattern Language of Program Design 3, edited by Robert C. Martin, Dirk Riehle, and Frank Buschmann, Addison-Wesley (Software Patterns Series), 1997.

Microsoft Solutions Framework. Disponível em: http://msdn.microsoft.com/vstudio/teamsystem/msf/. Aessado em: 20/12/2005.

MOTELET, O. **An Intelligent Tutoring System to Help OO System Designers Using Design Patterns**. Master's thesis, Vrije Universitët, 1999.

OMG Model Driven Architecture. Disponível em: http://www.omg.org/mda/. Acessado em: 20/11/2005.

ÖVERGAARD, G.; PALMKVIST, K. **Use Cases Patterns and Blueprints.** Addison Wesley Professional, 2004. ISBN: 0-13-145134-0.

PAULK, M. C. **Capability Maturity Model for Software**. Version 1.1, Technical Report CMU/SEI-93-TR-024, 1993.

POLLICE, G. **Using the Rational Unified Process for Small Projects: expanding Upon Extreme Programming**. Rational Software White Paper, 2002.

Portland Patterns Repository. Disponível em: http://c2.com/ppr/index.html. Acessado em: 08/10/2005.

PREE, W. **Design Patterns for Object-Oriented Software Development**. Addison-Wesley, 1995.

PREE, W. **Meta Patterns - A Means for Capturing the Essentials of Reusable Object-Oriented Design**. Proc. ECOOP ‘94, LNCS 821, Springer, Berlin 1995 pp. 150-162.
PREE, W. **State-of-the-art Design Pattern Approaches - An Overview**. Technology of Object-Oriented Languages and Systems (TOOLS 95), Paris, 6-9 March 1995.

RATIONAL UNIFIED PROCESS Tutorial. Versão 2003 06 00.

RENZEL, K. **Design of Business Information Systems**. Setembro de 1997. Disponível em: http://www.objectarchitects.de/arcus/cookbook/. Acessado em: 07/10/2005.

RISING, L. **The Pattern Almanac 2000**. Software Pattern Series, Addison-Wesley, 2000. ISBN 0-201-61567-3.

ROBERTSON, S. **Requirements Patterns Via Events/Use Cases**. The Atlantic Systems Guild, 1996. Disponível em: http://www.systemsguild.com/GuildSite/SQR/Requirements_Patterns.html. Acessado em: 29/09/2005.

RUMBAUGH, J.; BOOCH, G.; JACOBSON, I. **UML Guia do Usuário**. Editora Campus. 2000.

SANTOS, M. S. **Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas**. Dissertação de Mestrado. Fortaleza, 2004.

SARDJONO, W. P.; SIMONS, A. J. H. **Using Patterns for Partitioning Distributed Object Applications**. The 19th European Conference on Object-Oriented Programming. Glasgow, Scotland, 2005.

SARDJONO, W. P.; SIMONS. A. J. H. **Pattern-Drive Partitioning in Designing Distributed Object Applications**. In Dietmar Sch¸tz and Klaus Marquardt, editors, EuroPLoP 2004 Proceedings, 2004.

SCHNEIDER, G.; WINTERS, J. **Applying Use Case: A Practical Guide**. 2nd ed. Addison Wesley, 2001.

SIMONS. A. J. H. **Object Discovery - A Process For Developing Medium-sized Applications**. Tutorial 14, ECOOP 1998 Tutorials, Brussels, 1998. AITO/ACM.

SODRÉ, V. F.; LISBOA, J. F.; VILELA, V. M.; ANDRADE, M. V. A. **Improving Productivity and Quality of GIS Databases Design using an Analysis Pattern Catalog**. Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), University of Newcastle, Newcastle, Australia. 2005.

SOMMERVILLE, I. **Software Engineering**. 6th Edition, Addison-Wesley Publishers Ltd., 2001. ISBN 0-201-39815-X.

SOUZA, G. T.; PIRES, C. G. **PATI-MVC: Uma Família de Padrões para Sistemas de Informação Baseada no Padrão MVC**. The Fourth Latin American Conference on Pattern Languages of Programming (SugarloafPloP). Porto das Dunas, Ceará, 2004.

SOUZA, G. T.; PIRES, C. G.; BARROS, M. **Padrões MVC para Sistemas de Informação**. The Third Latin American Conference on Pattern Languages of Programming (SugarloafPloP). Porto de Galinhas, Pernambuco, 2003.

SOUZA, G. T.; PIRES, C. G.; BELCHIOR, A. D. **Padrões de Requisitos para Especificação de Casos de Uso em Sistemas de Informação**. The Fifth Latin American Conference on Pattern Languages of Programming (SugarloafPloP). Campos do Jordão, São Paulo, 2005.

SOUZA, G. T.; PIRES, C. G.; MARINHO, F. G.; BELCHIOR, A. D. **Aplicação de metapadrões e padrões em desenvolvimento de software para sistemas de informação**. The Fifth Latin American Conference on Pattern Languages of Programming (SugarloafPloP). Campos do Jordão, São Paulo, 2005.

The Diemen Repository of Interaction Design Patterns. Disponível em: http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/. Acessado em: 08/10/2005.

The Hillside Group. Disponível em: http://www.hillside.net. Acessado em: 08/10/2005

VLISSIDES, J. **Patterns: The Top Ten Misconceptions**. Object Magazine, Mar, 1997. Disponível em: http://hillside.net/patterns/papersbibliographys.htm. Acesso em: 29/09/2005.

WELIE, M. V. **Amsterdam Patterns Collection**. Disponível em: http://www.welie.com/patterns. Acessado em: 08/10/2005.

Apêndice I – Especificação do caso de uso Manter Usuário

PROJETO LEVI

ESPECIFICAÇÃO DE CASO DE USO

MANTER USUÁRIO

VERSÃO A.02

1 Introdução

1.1. Objetivos

Este documento tem como objetivo detalhar o caso de uso Manter Usuário.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Manter Usuário

2.1. Descrição do caso de uso

Este caso de uso permite ao Administrador incluir, alterar, excluir, e consultar usuários. Na criação do usuário, será atribuído um perfil para o mesmo. O perfil atribuído indicará quais funções do sistema o usuário tem acesso. Existirá um usuário com perfil de *Administrador* com acesso irrestrito ao sistema.

2.2. Atores envolvidos

- Administrador.

2.3. Fluxos de Eventos

Fluxo Básico (*Happy Day*)

1. O caso de uso inicia quando o *Administrador* necessita fazer a manutenção (inclusão, alteração, exclusão, ou consulta) de um *Usuário*;
2. De acordo com o tipo de manutenção desejado pelo *Administrador*, um dos subfluxos é executado:
   - 2.1. Se o *Administrador* deseja incluir um novo *Usuário*, o Subfluxo **Incluir Usuário** é executado.
   - 2.2. Se o *Administrador* deseja alterar informações de um *Usuário* já cadastrado, o Subfluxo **Alterar Usuário** é executado.
   - 2.3. Se o *Administrador* deseja excluir um *Usuário* já cadastrado, o Subfluxo **Remover Usuário** é executado.
   - 2.4. Se o *Administrador* deseja consultar informações sobre um ou mais *Usuários* cadastrados, o Subfluxo **Consultar Usuário** é executado.

Subfluxo Incluir Usuário

1. Este subfluxo inicia, quando o *Administrador* solicita incluir um *Usuário*.
2. O sistema solicita ao *Administrador* o preenchimento dos seguintes atributos:
   - * Nome;
   - * Login;
   - * Senha;
   - * Perfil (campo de escolha fechada. Valores possíveis: todos os perfis cadastrados no sistema);
   - * Data de Ativação. Data em que o usuário foi cadastrado no sistema; e
   - Data de Desativação. Data em que o usuário foi desativado do sistema. Se esse campo não estiver preenchido significa que o usuário está ativo. Caso contrário, se estiver preenchido significa que o usuário está inativo.
3. O *Administrador* preenche os atributos acima e confirma a inclusão.
4. O sistema realiza a inclusão dos dados informados pelo *Administrador* no passo 3.
5. O sistema exibe uma mensagem informando que a inclusão do *Usuário* foi efetivada com sucesso.

Subfluxo Alterar Usuário

1. Este fluxo inicia quando o *Administrador* solicita alterar um *Usuário*.
2. O *Administrador* seleciona um único *Usuário*.
3. O sistema solicita a alteração dos atributos listados no passo 2 do subfluxo **Incluir Usuário**.
4. O *Administrador* altera os dados desejados e confirma a alteração.
5. O sistema realiza a alteração dos dados informados no passo 3.
6. O sistema exibe uma mensagem de confirmação informando que a alteração do *Usuário* foi efetivada com sucesso.

Subfluxo Remover Usuário

1. Este subfluxo inicia quando o *Administrador* solicita remover um ou mais *Usuários*.
2. O *Administrador* seleciona quais *Usuários* deseja remover e solicita a remoção.

3. O sistema solicita a confirmação para a remoção.
4. O *Administrador* confirma a remoção.
5. O sistema remove os *Usuários* confirmados.
6. O sistema exibe uma mensagem informando que a remoção dos *Usuários* foi efetivada com sucesso.

Subfluxo Consultar Usuário

1. Este fluxo inicia quando o *Administrador* solicita consultar *Usuários.*
2. O sistema solicita o preenchimento dos seguintes filtros:
   - Nome; e
   - Perfil (campo de escolha fechada. Valores possíveis: todos os perfis cadastrados no sistema).
3. O *Administrador* preenche os filtros e solicita a consulta.
4. O sistema apresenta as seguintes informações dos *Usuários* obtidos na consulta:
   - Nome;
   - Login;
   - Senha;
   - Perfil;
   - Data de Ativação; e
   - Data de Desativação.

2.4. Precondições

- O *Administrador* deverá ter efetuado login no sistema para executar esta funcionalidade.

2.5. Pós-condições

- O usuário inserido ou alterado pelo *Administrador* deverá ser salvo na base de dados;
- O usuário excluído pelo *Administrador* deverá ser removido da base de dados.

2.6. Relacionamento com outros casos de uso

- Não se aplica.

2.7. Pontos de Extensão

- Não se aplica.

2.8. Requisitos especiais

- Não se aplica.

2.9. Regras de Negócio

- *Atributos obrigatórios*: se algum atributo obrigatório não tiver sido preenchido, o sistema não completará a operação e notificará ao *Administrador*, informando quais campos obrigatórios não foram preenchidos, solicitando o preenchimento dos mesmos. Esta regra aplica-se a todos os subfluxos.

- *Atributos com valores não permitidos*: se algum atributo for preenchido com valor não permitido, o sistema não completará a operação e notificará ao *Administrador*, informando quais campos foram preenchidos com valores inválidos e solicitando o preenchimento correto. Esta regra aplica-se a todos os subfluxos.

- No subfluxo **Remover Usuário**, o sistema valida os usuários selecionados de acordo com as seguintes regras:

  Usuário que tiver alguma visita agendada em aberto não poderá ser removido.

- No subfluxo **Incluir** e **Alterar Usuário**, o sistema valida os atributos preenchidos de acordo com as seguintes regras:

  Não é permitido mais de um usuário com um mesmo *login*;

  O campo senha deverá conter no mínimo 5 e no máximo 16 caracteres;

  A data de ativação não poderá ser menor que a data atual do servidor;

  A data de desativação deverá ser maior que a data de ativação.

2.10. Diagrama do caso de uso

Apêndice II – Projeto do caso de uso Manter Usuário

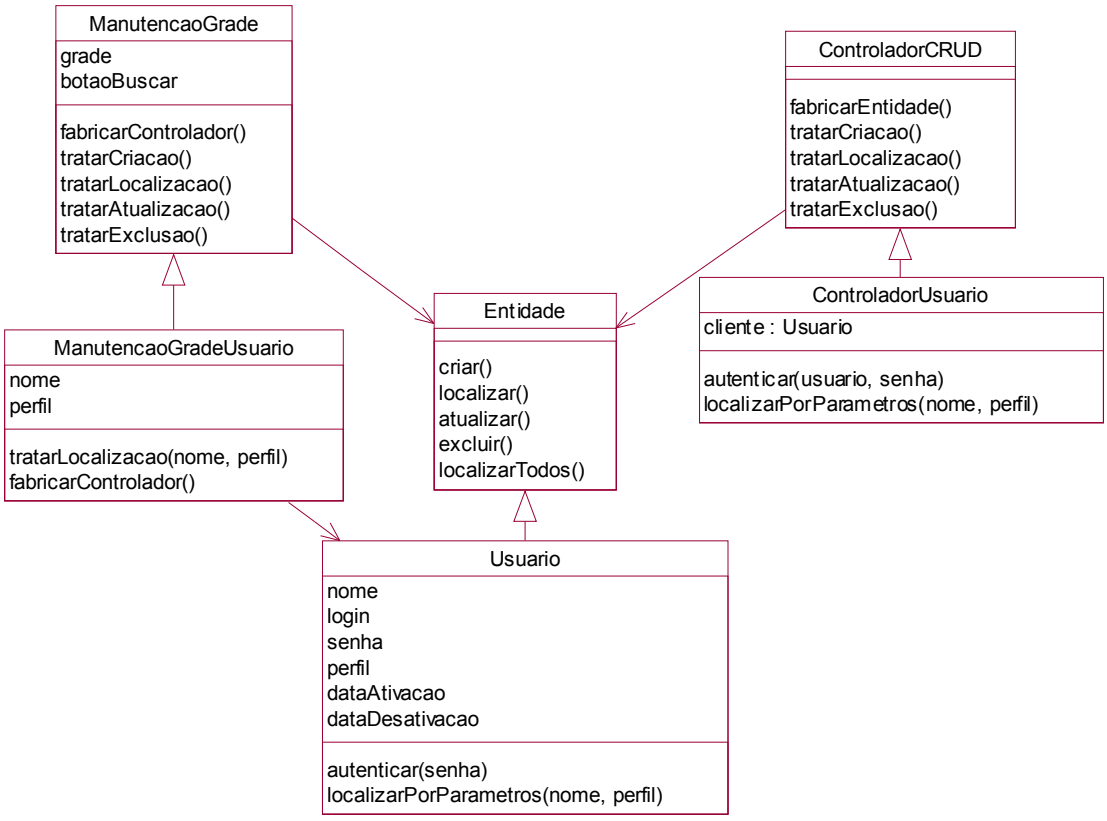


Figura II.1: Diagrama de classe do caso de uso Manter Usuário

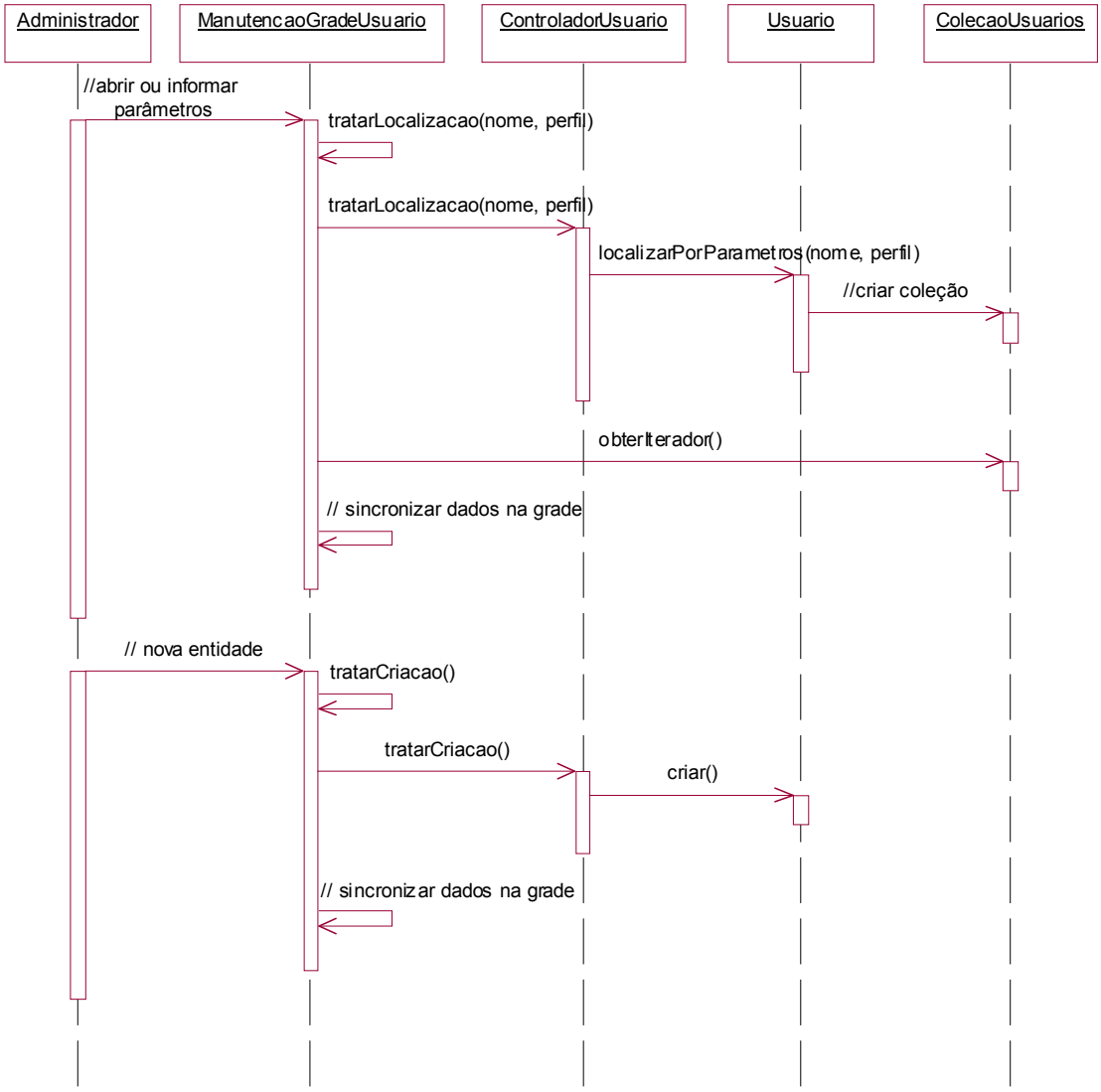


Figura II.2: Diagrama de seqüência da operação de Inclusão do caso de uso Manter Usuário

Apêndice III – Especificação de teste do caso de uso Manter Usuário

PROJETO LEVI

ESPECIFICAÇÃO DE TESTE

MANTER USUÁRIO

VERSÃO A.01

1 Introdução

1.1. Objetivos

Este documento tem como objetivo especificar os casos de teste e idéias de teste do caso de uso Manter Usuário.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Manter Usuário

2.1. Casos de teste – Inserir Usuário

Caso de teste	Procedimento de teste	Resultado
Usuário existente	Para verificar se o usuário já existe no sistema, deve-se verificar se existe um usuário cadastrado com o mesmo *login*.	O sistema deve exibir a mensagem: "Usuário já cadastrado" e retornar para a tela de inserção.
Usuário não existente	Para verificar se o usuário não existe no sistema, deve-se verificar se não existe um usuário cadastrado com o mesmo *login*.	Ao consultar o usuário cadastrado o sistema deve exibir as informações do usuário de acordo com os dados informados na inserção.
Acesso à base de dados	Simular a queda do acesso à base de dados antes de solicitar a inserção do usuário.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Atributos obrigatórios: Nome, *Login*, Senha, Perfil e Data de Ativação. Procedimento: deixar em branco os atributos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Nome, *Login*, Senha, Perfil e Data de Ativação" e voltar para a tela de inserção com os dados previamente informados.
Valores inválidos	Atributos com restrição de valores: Senha, Data de Ativação e Data de Desativação. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os campos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Senha, Data ed Ativação e Data de Desativação" e voltar para a tela de inserção com os dados previamente informados.

2.2. Casos de teste – Alterar Usuário

Caso de teste	Procedimento de teste	Resultado
Alterar usuário	Inicialmente, consultar o usuário desejado informando o Nome e, posteriormente, solicitar a alteração do usuário.	Ao consultar o usuário alterado o sistema deve exibir as informações do usuário de acordo com os dados informados na alteração.
Acesso à base de dados	Simular a queda do acesso à base de dados antes de solicitar a alteração do usuário.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Atributos obrigatórios: Nome, *Login*, Senha, Perfil e Data de Ativação. Procedimento: deixar em branco os atributos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Nome, *Login*, Senha, Perfil e Data de Ativação" e voltar para a tela de inserção com os dados previamente informados.
Valores inválidos	Atributos com restrição de valores: Senha, Data de Ativação, e Data de Desativação. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os campos acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Senha, Data ed Ativação e Data de Desativação" e voltar para a tela de inserção com os dados previamente informados.

2.3. Casos de teste – Remover Usuário

Caso de teste	Procedimento de teste	Resultado
Remover usuário sem chamado em aberto	Inicialmente, consultar o usuário desejado informando o Nome e, posteriormente, solicitar a remoção do usuário.	Ao remover o usuário o sistema deve exibir a mensagem: "Usuário removido com sucesso".
Remover usuário com visita agendada em aberto	Inicialmente, consultar o usuário desejado informando o Nome e, posteriormente, solicitar a remoção do usuário.	Ao tentar remover o usuário o sistema deve exibir a mensagem: "Usuário não pode ser removido, pois possui visita agendada em aberto".
Acesso à base de dados	Simular a queda do acesso à base de dados antes de solicitar a remoção do usuário.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".

2.4. Casos de teste – Consultar Usuário

Caso de teste	Procedimento de teste	Resultado
Consultar entidades	Inicialmente consultar os usuários informando o Nome e o Perfil.	O sistema deve apresentar uma lista com os usuários que correspondem aos filtros especificados.
Consulta que não retorna nenhum usuário	Informar valores para o nome e Perfil inexistentes na aplicação.	O sistema deve exibir a mensagem: "Nenhum registro encontrado para os filtros especificados".
Acesso à base de dados	Simular a queda do acesso à base de dados antes de solicitar a consulta dos usuários.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Filtros obrigatórios	Esta consulta não possui filtros obrigatórios.	-
Valores inválidos	Não existem filtros com restrição de valor.	-

2.5. Idéias de teste – Incluir, Alterar e Consultar Usuário

Nome do atributo	Validação	Valor
Senha	Tamanho limite para preenchimento.	Entre 5 e 16 caracteres.
Data de Ativação	Tamanho limite para preenchimento.	10 caracteres.
	Valores não permitidos.	Caracteres alfa numéricos e caracteres especiais, com exceção do caractere "/".
	Restrições específicas.	Deve ser informado no seguinte formato: • *dd*/*mm*/*aaaa*; • *dd* deve possuir valores entre 1 e 31; • *mm* deve possuir valores entre 1 e 12; • Deve ser maior ou igual à data atual.
Data de Desativação	Tamanho limite para preenchimento.	10 caracteres.
	Valores não permitidos.	Caracteres alfa numéricos e caracteres especiais, com exceção do caractere "/".
	Restrições específicas.	Deve ser informado no seguinte formato: • *dd*/*mm*/*aaaa*; • *dd* deve possuir valores entre 1 e 31; • *mm* deve possuir valores entre 1 e 12; • Deve ser maior que a Data de Ativação.

Apêndice IV – Especificação do caso de uso Gerar Relatório de Obras de Arte

PROJETO LEVI

ESPECIFICAÇÃO DE CASO DE USO

GERAR RELATÓRIO DE OBRAS DE ARTE

VERSÃO A.02

1 Introdução

1.1. Objetivos

Este documento tem como objetivo detalhar o caso de uso Gerar Relatório de Obras de Arte.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Gerar Relatório de Obras de Arte

2.1. Descrição do caso de uso

Este caso de uso permite ao *Catalogador* listar todas as obras de artes cadastradas no sistema de um museu com base nos seguintes filtros: categoria da obra (pintura, escultura, fotografia etc), técnica da obra (óleo sobre tela, aquarela etc) e artista.

2.2. Atores envolvidos

- Catalogador.

2.3. Fluxos de Eventos

Fluxo básico

1. Este fluxo inicia quando o *Catalogador* solicita gerar o relatório de *obras de arte*.
2. O sistema solicita o preenchimento dos seguintes filtros:
   - Categoria. (campo de escolha fechada. Valores possíveis: todas as categorias cadastradas no sistema);
   - Técnica. (campo de escolha fechada. Valores possíveis: todas as técnicas cadastradas no sistema); e
   - Artista (campo de escolha fechada. Valores possíveis: todos os artistas cadastrados no sistema).
3. Uma vez que o *Catalogador* forneça a informação solicitada, uma das seguintes ações é executada:
   - Se o *Catalogador* selecionar Imprimir, o sistema deve apresentar a janela de configuração de impressão;

- Se o *Catalogador* selecionar Visualizar, o sistema deve apresentar uma janela com a visualização do relatório.

4. O sistema apresenta o resultado na seguinte forma:

- Cabeçalho. Deve conter o nome do relatório, nome do museu e a data em que o relatório foi gerado;
- Corpo. As obras devem ser agrupadas por categoria e técnica e as seções devem conter quebras de página a cada categoria. Os seguintes atributos devem ser apresentados: número do patrimônio, título da obra, e nome do artista;
- Rodapé. Deve conter o número da página;
- Totalização. As totalizações devem ser efetuadas por categoria, apresentando quantas obras existem na categoria selecionada ou em todas as categorias, se nenhuma categoria tiver sido selecionada.

2.4. Precondições

- O *Catalogador* deverá ter efetuado login no sistema para executar esta funcionalidade.

2.5. Pós-condições

- Relatório de obras de arte gerado de acordo com os filtros preenchidos pelo *Catalogador*.

2.6. Relacionamento com outros casos de uso

- Não se aplica.

2.7. Pontos de Extensão

- Não se aplica.

2.8. Requisitos especiais

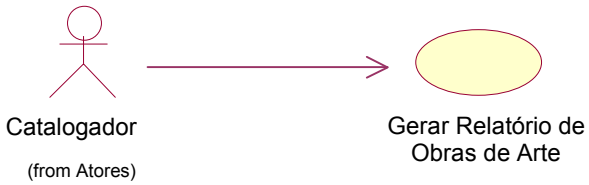
- Opções de ordenação. O relatório deve ser ordenado por nome da categoria, nome da técnica, título da obra, e nome do artista (nesta ordem).
- Regra de extração. Devem ser apresentadas no relatório todas as obras que estão cadastradas no sistema e que possuem número de patrimônio.
- Modelo de desenho esquemático:

Museu de Arte **Relatório de obras de arte** **03/01/2005**		
Categoria: Pintura		**Técnica: Óleo sobre tela**
Patrimônio	**Título**	**Artista**
<número do patrimônio>	*<título da obra>*	*<nome do artista>*
Total de Obras: <total de obras>		
		Página: *<número da página>*

2.9. Regras de Negócio

- Não se aplica.

2.10. Diagrama do caso de uso



Apêndice V – Projeto do caso de uso Gerar Relatório de Obras de Arte

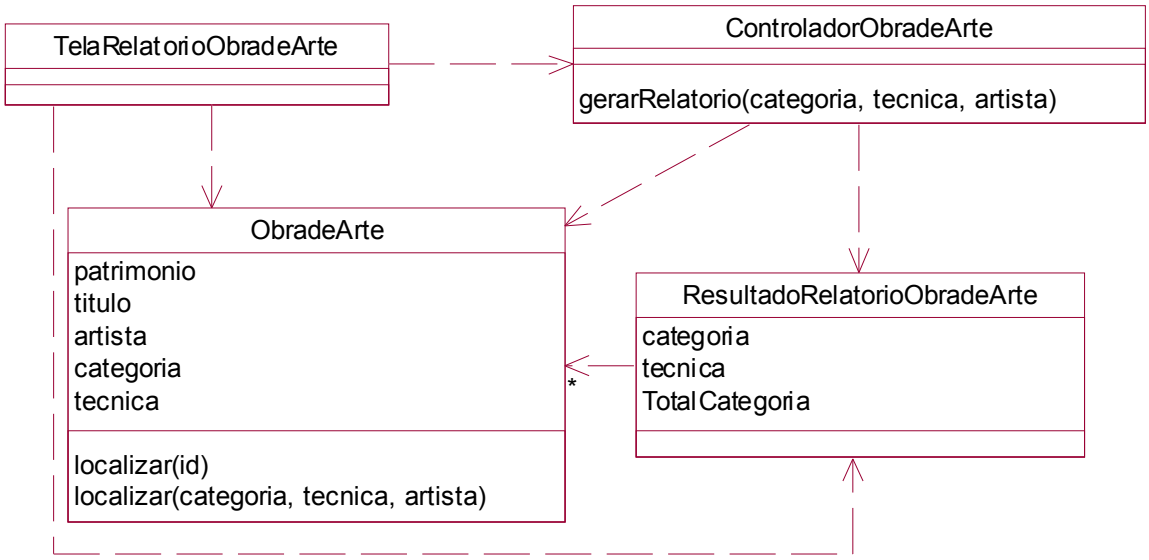


Figura V.1: Diagrama de classe do caso de uso Gerar Relatório de Obras de Arte

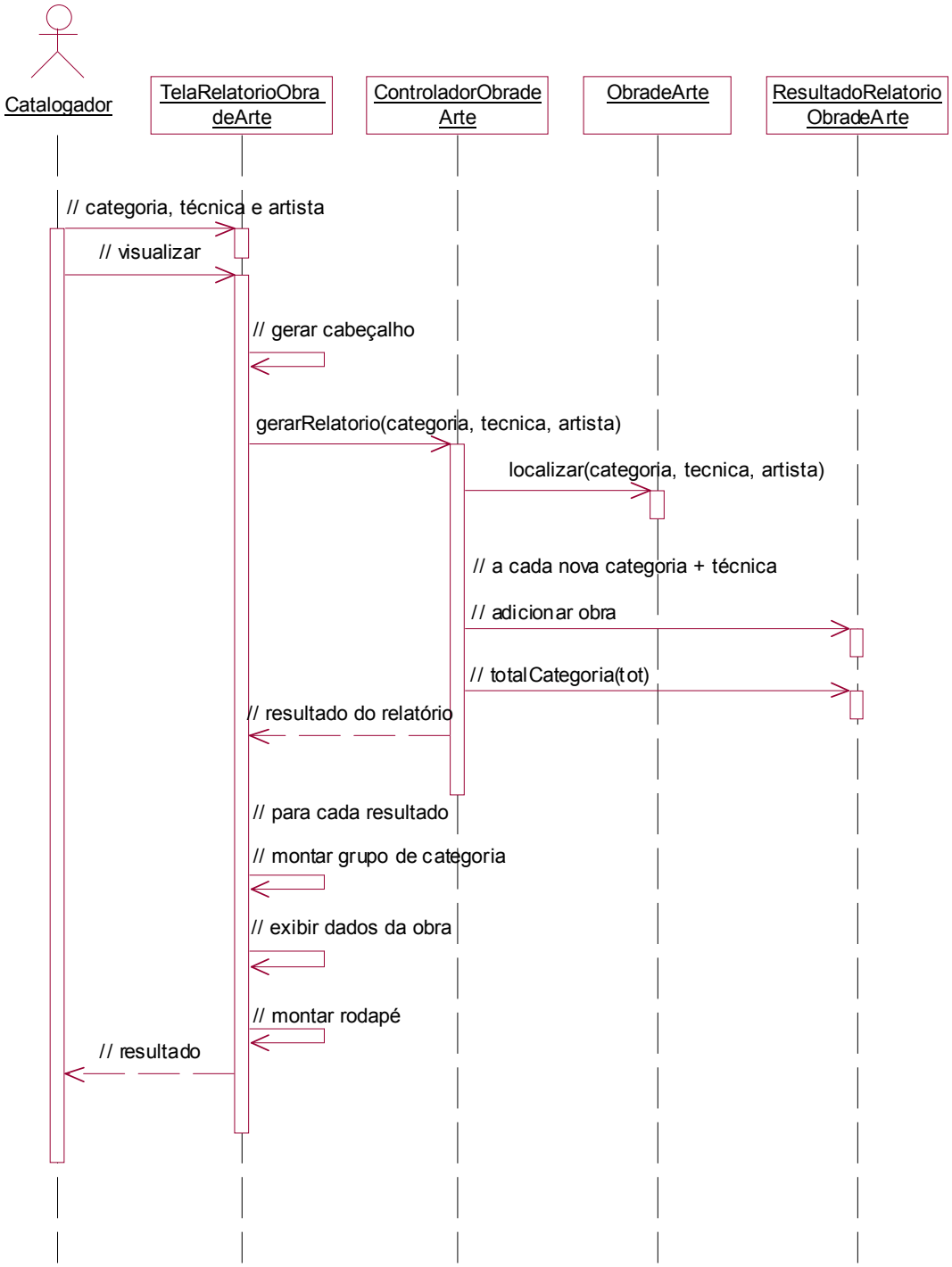


Figura V.2: Diagrama de seqüência do caso de uso Gerar Relatório de Obras de Arte

Apêndice VI – Especificação de teste do caso de uso Gerar Relatório de Obras de Arte

PROJETO LEVI

ESPECIFICAÇÃO DE TESTE

GERAR RELATÓRIO DE OBRAS DE ARTE

VERSÃO A.01

1 Introdução

1.1. Objetivos

Este documento tem como objetivo especificar os casos de teste e idéias de teste do caso de uso Gerar Relatório de Obras de Arte.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Gerar Relatório de Obras de Arte

2.1. Casos de teste – Relatório de Obras de Arte

Caso de Teste	Procedimento de teste	Resultado
Gerar relatório segundo os filtros especificados	Selecionar os filtros segundo as idéias de teste e solicitar a visualização do relatório.	O sistema deve apresentar o relatório contendo somente as obras de arte que correspondem aos filtros selecionados.
Acesso à base de dados	Simular a queda do acesso à base de dados antes de pedir a geração do relatório.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Não há filtros obrigatórios.	
Valores inválidos	Não há filtros com restrição de valores.	
Impressão	Selecionar uma categoria e uma técnica e solicitar a impressão do relatório.	O sistema deve imprimir o relatório em folha A4 respeitando as margens e tamanho do papel. O formato impresso deve ser semelhante ao formato da visualização.
Totalizações, cálculos e agrupamentos	Selecionar uma categoria e uma técnica e solicitar a visualização do relatório.	As obras devem ser agrupadas por categoria e técnica e ao final de cada categoria deve existir o total de obras da categoria.
Formato do relatório (cabeçalho, corpo e rodapé)	Selecionar uma categoria e uma técnica e solicitar a visualização do relatório.	O relatório deve ter no cabeçalho o nome do relatório, nome do museu e sua data de geração. O corpo do relatório deve ter o número do patrimônio, título da obra e nome do artista. No inicio de cada grupo, devem aparecer categoria e técnica selecionadas. Ao final de cada categoria, deve aparecer o total de obras da categoria. O rodapé deve conter o número da página.
Checar formato e dados em arquivos exportados	Não existe requisito de exportação de dados.	

2.2. Idéias de teste – Relatório de Clientes por Bairro

Nome do filtro	Validação
Categoria, Técnica e Artista	Não selecionar nenhum filtro.
	Selecionar somente uma categoria.
	Selecionar uma categoria e uma técnica.
	Selecionar uma categoria, uma técnica e um artista.

Apêndice VII – Especificação do caso de uso Exportar Obras para Catálogo

PROJETO LEVI

ESPECIFICAÇÃO DE CASO DE USO

EXPORTAR OBRAS PARA CATÁLOGO

VERSÃO A.03

1 Introdução

1.1. Objetivos

Este documento tem como objetivo detalhar o caso de uso Exportar Obras para Catálogo.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Exportar Obras para Catálogo

2.1. Descrição do caso de uso

Este caso de uso permite ao *Catalogador* exportar os dados das obras de arte selecionadas, no formato padrão do catálogo, para que as mesmas sejam incluídas na publicação do catálogo.

2.2. Atores envolvidos

- Catalogador.

2.3. Fluxos de Eventos

Fluxo básico

1. Este fluxo inicia quando o *Catalogador* solicita exportar obras de arte para um catálogo.
2. O sistema solicita o preenchimento dos seguintes dados:
   - * Obra de arte. (Campo de escolha múltipla. Valores possíveis: todas as obras de arte que possuem número de patrimônio). Esse campo deve aparecer em ordem alfabética;
   - * Formato para exportação. (Campo de escolha fechada. Valores possíveis: Excel, Word ou Texto simples);
   - Local e nome do arquivo. Este campo indica o diretório onde o arquivo gerado será gravado e o seu nome.
3. O *Catalogador* preenche os dados solicitados no passo 2 e confirma a execução da operação.
4. O sistema executa as seguintes operações:

- Obtém os atributos imagem, título, artista, categoria, técnica e dimensões das obras selecionadas;
- O sistema gera um arquivo contendo as informações das obras selecionadas, no formato, local e nome especificados pelo *Catalogador*.

2.4. Precondições

- O *Catalogador* deverá ter efetuado login no sistema para executar esta funcionalidade.

2.5. Pós-condições

- Arquivo gerado de acordo com os parâmetros especificados pelo *Catalogador*.

2.6. Relacionamento com outros casos de uso

- Não se aplica.

2.7. Pontos de Extensão

- Não se aplica.

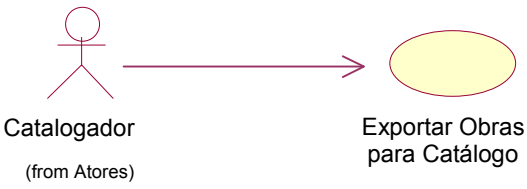
2.8. Requisitos especiais

- O progresso da operação deverá ser apresentado em % (percentual), que deverá ser calculado considerando quantas obras já foram exportadas, em relação ao total de obras selecionadas. Por exemplo: o Catalogador selecionou 10 (dez) obras para serem exportadas. Quando o sistema estiver efetuado a exportação de 2 (duas) obras o progresso da operação será 20% (vinte por cento).

2.9. Regras de Negócio

- Somente deverão ser exportadas as obras que possuírem número de patrimônio. As obras que não possuem número de patrimônio não estão efetivadas no acervo do museu e não devem ser consideradas.

2.10. Diagrama do caso de uso



Apêndice VIII – Projeto do caso de uso Exportar Obras para Catálogo

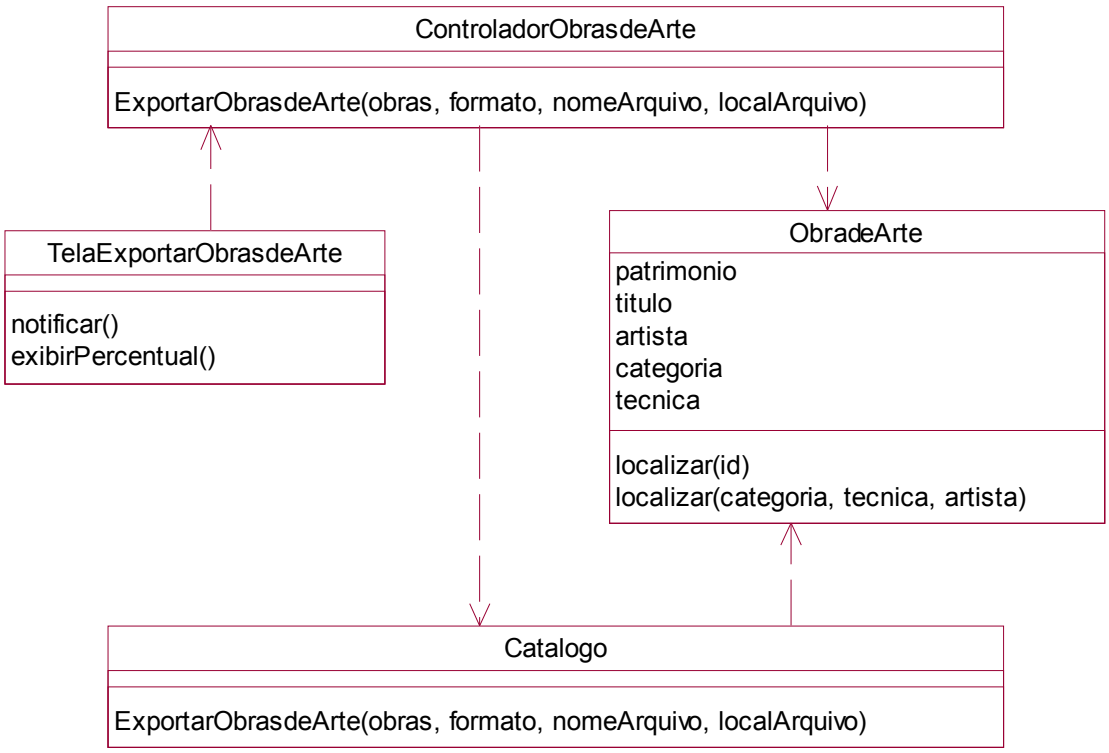


Figura VIII.1: Diagrama de classe do caso de uso Exportar Obras de Arte

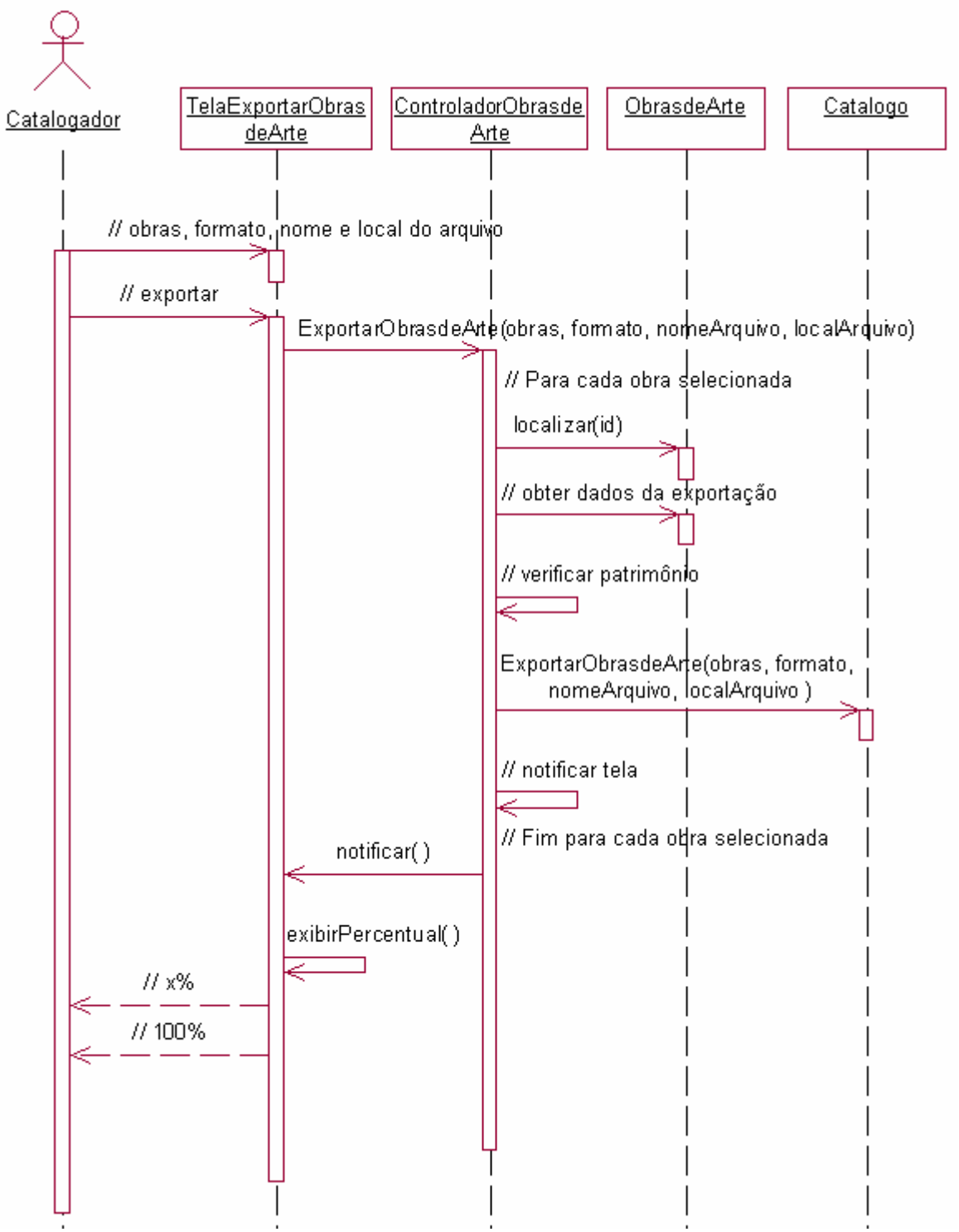


Figura VIII.2: Diagrama de seqüência do caso de uso Exportar Obras de Arte

Apêndice IX – Especificação de teste do caso de uso Exportar Obras para Catálogo

PROJETO LEVI

ESPECIFICAÇÃO DE TESTE

EXPORTAR OBRAS PARA CATÁLOGO

VERSÃO A.01

1 Introdução

1.1. Objetivos

Este documento tem como objetivo especificar os casos de teste e idéias de teste do caso de uso Exportar Obras para Catálogo.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Exportar Obras para Catálogo

2.1. Casos de teste

Caso de teste	Procedimento de teste	Resultado
Executar transação de acordo com os parâmetros especificados	Informar as obras, formato, local e nome do arquivo a ser gerado e solicitar a execução da transação.	O sistema deve apresentar o progresso da transação e ao término deve ter executado as seguintes operações: • Obter os atributos imagem, título, artista, categoria, técnica e dimensões das obras selecionadas; • Gerar um arquivo contendo as informações das obras selecionadas, no formato, local e nome especificados.
Falha do durante a execução da transação	Iniciar a transação e em seguida provocar a queda do servidor de aplicação ou desligar o computador.	Verificar se o sistema não processou nenhuma das operações descritas no caso de teste *Executar transação de acordo com os parâmetros especificados*.
Acesso à base de dados	Simular a queda do acesso à base de dados antes da execução da transação.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Parâmetros obrigatórios: Obras, Formato, Local e Nome do arquivo. Procedimento: deixar em branco os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes atributos não foram informados: Obras, Formato, Local e Nome do arquivo" e voltar para a tela de execução da transação.
Valores inválidos	Parâmetros com restrições de valores: Local e nome do arquivo. Procedimento: preencher com valores inválidos de acordo com as idéias de teste, os parâmetros acima.	O sistema deve exibir a mensagem: "Os seguintes atributos foram informados com valores inválidos: Local e nome do arquivo" e voltar para a tela de execução da transação.

2.2. Idéias de teste

Nome do parâmetro	Validação	Valor
Local	Restrições específicas.	O local do arquivo deve corresponder a um diretório válido.
Nome do arquivo	Tamanho limite para preenchimento.	8 caracteres.
	Valores não permitidos.	Caracteres numéricos, alfanuméricos e caracteres especiais, com exceção dos seguintes caracteres: / \ : * ? ” < > |.
	Restrições específicas.	Deve seguir o formato: *<nome>.<extensão>*. As extensões válidas são: *doc*, *xls* e *txt*.

Apêndice X – Especificação do caso de uso Configurar *Site*

PROJETO LEVI

ESPECIFICAÇÃO DE CASO DE USO

CONFIGURAR *SITE*

VERSÃO A.01

1 Introdução

1.1. Objetivos

Este documento tem como objetivo detalhar o caso de uso Configurar *Site*.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Configurar *Site*

2.1. Descrição do caso de uso

Este caso de uso permite ao *Administrador* configurar o *site* do museu, onde serão publicadas as informações das obras, visitas, eventos, etc. Será possível escolher um modelo e configurar seu esquema de cores. Em seguida, será possível alterar características de fontes, fundo, posicionamento de texto, bordas, alinhamento de objetos e títulos.

2.2. Atores envolvidos

- Administrador.

2.3. Fluxos de Eventos

Fluxo básico

1. O caso de uso inicia quando o *Administrador* necessita configurar o *site* do museu.
2. O sistema informa que esta operação será executada em 3 passos.
3. O sistema solicita que o *Administrador* execute o Passo 1.
4. Uma vez que o *Administrador* decida executar o Passo 1, subfluxo **Passo 1** é executado.
5. O caso de uso se encerra.

Subfluxo Passo 1

1. Este subfluxo se inicia quando o *Administrador* solicita selecionar o modelo do *site* do museu.

2. O sistema apresenta os modelos disponíveis e solicita que o Administrador selecione o modelo desejado. O modelo do *site* corresponde à forma de como as informações estarão disponíveis (cabeçalho, logo do museu, rodapé e informações de detalhe).
3. O *Administrador* seleciona o modelo desejado.
4. O sistema solicita que o *Administrador* execute o Passo 2.
5. Uma vez que o *Administrador* decida executar o Passo 2, o subfluxo **Passo 2** é executado.

Subfluxo Passo 2

1. Este subfluxo se inicia quando o *Administrador* solicita selecionar o esquema de cores do *site*.
2. Para este subfluxo ser executado, o subfluxo Passo 1 deve ter sido executado.
3. O sistema apresenta os esquemas de cores disponíveis e solicita que o Administrador selecione o esquema desejado de cor.
4. O *Administrador* seleciona o esquema desejado de cor.
5. O sistema solicita que o *Administrador* execute o Passo 3 ou conclua a operação.
6. Uma vez que o *Administrador* decida executar o Passo 3, subfluxo **Passo 3** é executado.

Subfluxo Passo 3

1. Este subfluxo se inicia quando o *Administrador* solicita alterar características de fontes, fundo, posicionamento de texto, bordas, alinhamento de objetos e títulos.
2. Para este subfluxo ser executado, os subfluxos **Passo 1** e **Passo 2** devem ter sido executados.
3. O sistema solicita ao *Administrador* o preenchimento dos seguintes atributos:
    - Fontes do texto. Neste campo é possível configurar o tipo, tamanho, cor, alinhamento e estilo da fonte;
    - Fundo. Neste campo é possível configurar a cor e imagem de fundo;
    - Bordas. Neste campo é possível configurar estilo, largura e cor das bordas;
    - Fonte dos títulos. Neste campo é possível configurar o tipo, tamanho, cor, alinhamento e estilo da fonte.

4. O *Administrador* preenche os atributos acima.

5. O sistema solicita que o *Administrador* conclua a operação.

6. O caso de uso se encerra.

2.4. Precondições

- O *Administrador* deverá ter efetuado login no sistema para executar esta funcionalidade.

2.5. Pós-condições

- O site do museu configurado de acordo com os parâmetros informados pelo *Administrador*.

2.6. Relacionamento com outros casos de uso

- Não se aplica.

2.7. Pontos de Extensão

- Não se aplica.

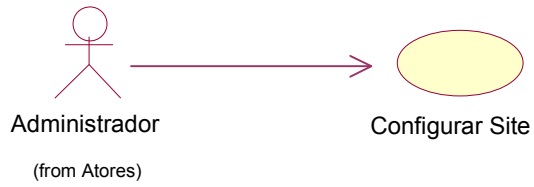
2.8. Requisitos especiais

- As informações configuradas neste caso de uso devem ser gravadas no banco de dados e em arquivos css.

2.9. Regras de Negócio

- Não se aplica.

2.10. Diagrama do caso de uso



Apêndice XI – Projeto do caso de uso Configurar *Site*

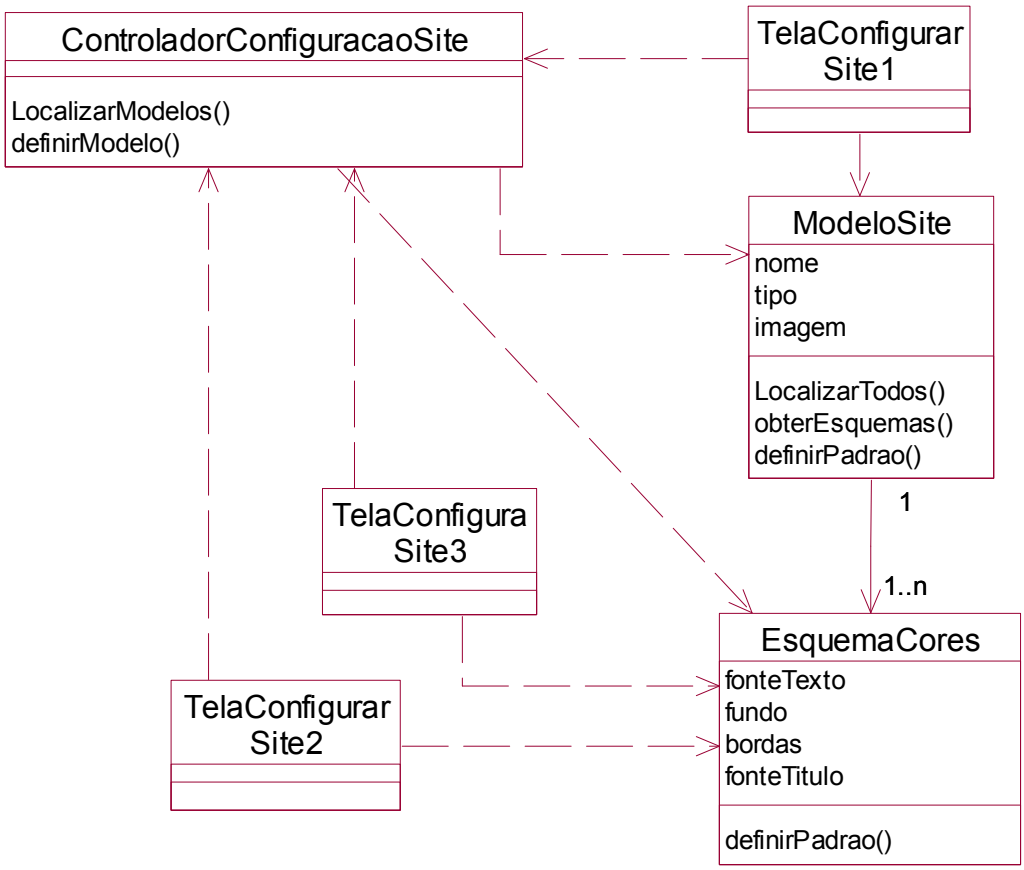


Figura XI.1: Diagrama de classe do caso de uso Configurar *Site*

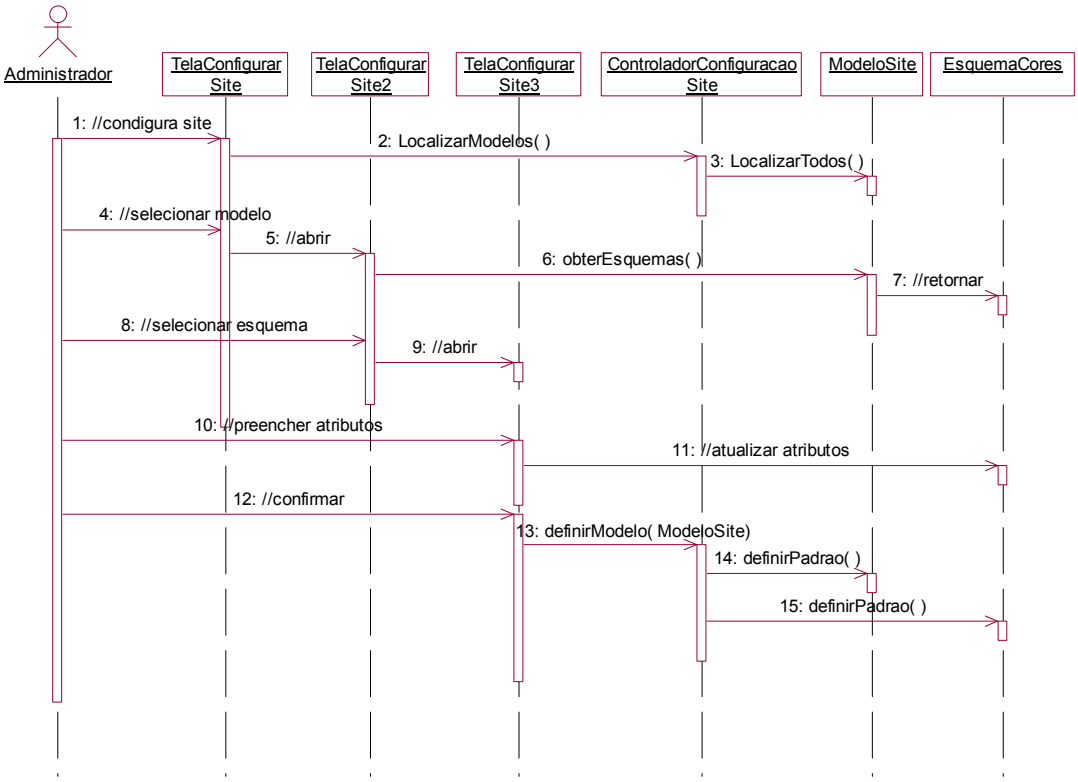


Figura XI.2: Diagrama de seqüência do caso de uso Configurar *Site*

Apêndice XII – Especificação de teste do caso de uso Configurar *Site*

PROJETO LEVI

ESPECIFICAÇÃO DE TESTE

CONFIGURAR *SITE*

VERSÃO A.01

1 Introdução

1.1. Objetivos

Este documento tem como objetivo especificar os casos de teste e idéias de teste do caso de uso Configurar *Site*.

1.2. Público alvo deste documento

- Coordenador do projeto;
- Equipe do projeto;
- Cliente (fornecedor de requisitos); e
- Grupo de teste.

2 Configurar *Site*

2.1. Casos de teste

Fluxo Geral

Caso de teste	Procedimento de teste	Resultado
Executar a operação de acordo com os parâmetros especificados e passos requeridos	Executar os Passos 1, 2 e 3 informando todos os parâmetros obrigatórios e concluir a operação.	Após a conclusão da operação, o sistema deverá ter gravado no banco de dados as informações de configuração do *site* e gerado os arquivos os de acordo com os parâmetros informados.
Seqüência dos passos	O Passo 2 só pode ser executado após o Passo 1 e o Passo 3 só poderá ser executado após os Passos 1 e 2. O Passo 3 não é obrigatório.	Se a seqüência de passos não for obedecida, o sistema deve mostrar a mensagem: "O Passo *n* deve ser executado antes do Passo *n+1*"

Passo 1

Caso de teste	Procedimento de teste	Resultado
Passagem de informações de um passo para outro	O Passo 1 deverá passar o seguinte parâmetro para o Passo 2: modelo do *site*.	Se o parâmetro não for passado corretamente, os esquemas de cores não serão apresentados no próximo passo.
Acesso à base de dados	Simular a queda do acesso à base de dados antes da execução do passo.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".

Dados obrigatórios	Parâmetro obrigatório: modelo do *site*. Procedimento: não selecionar um modelo.	O sistema deve exibir a mensagem: "O seguinte parâmetro não foi informado: modelo do *site*" e voltar para a tela do Passo 1.
Valores inválidos	Nenhum parâmetro com restrição de valores.	

Passo 2

Caso de teste	Procedimento de teste	Resultado
Exibir os esquemas de cores referentes ao modelo selecionado	Selecionar o modelo do *site* no Passo 1 e solicitar a execução do Passo 2.	O sistema deverá apresentar uma lista de esquemas de cores disponíveis de acordo com modelo do *site* selecionado no Passo 1.
Passagem de informações de um passo para outro	O Passo 2 deverá passar os seguintes parâmetros para o Passo 3: modelo do *site* e esquema de cores.	Se os parâmetros não forem passados corretamente, o sistema deverá exibir a mensagem: "Erro na passagem de parâmetros, iniciar a operação".
Acesso à base de dados	Simular a queda do acesso à base de dados antes da execução do passo.	O sistema deve exibir a mensagem: "Erro no acesso à base de dados. Por favor, tente mais tarde".
Dados obrigatórios	Parâmetro obrigatório: esquema de cores. Procedimento: não selecionar um esquema de cores.	O sistema deve exibir a mensagem: "O seguinte parâmetro não foi informado: esquema de cores" e voltar para a tela do Passo 2.
Valores inválidos	Nenhum parâmetro com restrição de valores.	

Passo 3

Caso de teste	Procedimento de teste	Resultado
Apresentar os itens que podem ser configurados	No Passo 2, selecionar o esquema de cores e solicitar a execução do Passo 3.	O sistema deverá apresentar os itens que podem ser configurados de acordo com o modelo e o esquema de cores selecionados.
Passagem de informações de um passo para outro	Como este é o último passo não existe passagem de parâmetros para passos seguintes.	
Acesso à base de dados	Simular a queda do acesso à base de dados antes da execução do passo.	O sistema deve exibir a mensagem: “Erro no acesso à base de dados. Por favor, tente mais tarde”.
Dados obrigatórios	Nenhum parâmetro obrigatório.	
Valores inválidos	Nenhum parâmetro com restrição de valores.	

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo