UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gabriel Silva Bornia

Estruturação de Descrições de Casos de Uso através de Mecanismos de Extensibilidade da UML

Dissertação de Mestrado

Prof. Dr. Roberto Tom Price
Orientador

Porto Alegre, janeiro de 2005

Sumário

Índice de Figuras

Índice de Tabelas

Resumo

Este trabalho apresenta uma forma de representação estruturada de descrições de casos de uso através do uso de diagramas de atividade estereotipados. O uso da extensibilidade da UML permite configurar elementos da linguagem de tal forma que esta possa também ser utilizada para a descrição do comportamento do sistema. É apresentada uma forma de representação de descrições de casos de uso em vários níveis de abstração, bem como a associação entre elementos da descrição e o modelo estático do sistema. Uma ferramenta CASE é apresentada como prova de conceito para o método de descrição proposto.

Palavras-chave: Caso de uso, descrição de casos de uso, casos de colaboração, UML, diagramas de atividade, mecanismos de extensibilidade, ferramenta CASE.

Abstract

This paper presents a structured representation of use case descriptions using stereotyped activity diagrams. The use of the extensibility mechanism of UML is used to configure language elements to be used for the description of system behavior. A way of representing use case descriptions in different levels of abstraction is shown, and ways of associating between descriptive elements and the static model of the system. A CASE tool is presented to demonstrate the proposed use case description method.

Key-words: Use case, use case description, collaboration case, UML, activity diagram, extensibility mechanism, CASE tool.

1 Introdução

Na análise de sistemas orientada a objetos a modelagem de casos de uso possui um papel fundamental. Os casos de uso são o principal mecanismo de levantamento de requisitos [Leffingwell 2000], caracterizando-se por uma representação narrativa do comportamento do sistema [Jacobson 1992][Jacobson 1998].

As descrições de casos de uso caracterizam-se por serem descrições do funcionamento do sistema sem, no entanto, entrar em detalhes específicos de como o mesmo será implementado. Diversos autores [Cockburn 2002][Firesmith 1995][Fowler 1999] defendem que os casos de uso devem ser o mais simples possível, uma vez que devem ser entendidos pelos usuários do sistema – que os utilizam como forma de validação dos requisitos levantados pelo analista – e servir como base aos demais envolvidos na construção do sistema.

A necessidade de se criar descrições simples que possam ser entendidas por usuários comuns representa um obstáculo ao uso de mecanismo mais formais de descrição [Fowler 1998] que possam ser utilizados de forma estruturada por projetistas, desenvolvedores e gerentes de projeto.

Existem diversas formas de se representar descrições de casos de uso. A mais comum delas é a forma textual, que embora possua diversos estilos propostos [Ambler 2000][Armour 2000][Larman 1998] continua praticamente livre de uma estrutura que permita a modelagem da descrição. Existem diversas propostas de formalizações de casos de uso [Hurlbut 1997][Hurlbut 1997b] de difícil disseminação no mercado.

A UML é uma linguagem de modelagem [Booch 1999] amplamente difundida com a qual geralmente os casos de uso são modelados, mas a interação entre o sistema e o ator (descrição do caso de uso) é realizada de forma narrativa [Scheneider 1998]. As descrições dessas interações podem ser realizadas através de diversos elementos da linguagem UML como, por exemplo, diagramas de atividade [Armour 2000], diagramas de seqüência, diagramas de estado, entre outros. UML possui a facilidade de estender os seus elementos através do conceito de estereótipos, o que poderia ser utilizado para ajudar a enriquecer uma descrição feita com os elementos desta linguagem.

O objetivo deste trabalho é a representação de descrições de casos de uso através de mecanismos de extensibilidade da UML, mais especificamente permitindo a modelagem das atuais representações narrativas por meio de diagramas de atividade estereotipados, garantindo uma representação mais estruturada que possa ser utilizada em outros passos do processo de desenvolvimento de sistemas sem, no entanto, perder a capacidade de se

comunicar com o usuário através de mecanismos que permitam gerar uma linguagem que ele compreenda.

Inicialmente, é apresentado nesta introdução o estado da arte da análise de sistemas orientada a objetos, sua linguagem principal de descrição, bem como o processo de desenvolvimento orientado a objetos, mais especificamente o Rational Unified Process (RUP).

Ao longo do segundo capítulo, será apresentada em maiores detalhes a forma de captura e representação de requisitos através da modelagem de casos de uso, entrando em detalhes sobre os elementos que compõe esta modelagem na UML.

No terceiro capítulo, será realizado um aprofundamento sobre a forma de descrições de casos de uso, através da análise dos tipos de casos de uso, da representação dos fluxos de eventos, entre outros elementos. São apresentados também modelos ou *templates* de descrição de casos de uso.

No capítulo seguinte é apresentada uma forma estruturada de representação de casos de uso através de diagramas de atividade. São explorados os meios de representação do fluxo principal de eventos e suas seqüências alternativas, de cenários e relacionamentos entre os casos de uso.

No quinto capítulo é proposta uma forma de se relacionar as descrições dos casos de uso com o modelo estático do sistema, identificando o processo de refinamento das descrições até o momento de vinculá-las com subsistemas, camadas ou classes.

O sexto capítulo apresenta o modelo da solução proposta de descrição de casos de uso. O modelo apresentado visa suportar tanto as descrições como o relacionamento dos casos de uso como o modelo de classes.

A partir da estruturação das descrições é mostrado, no capítulo seguinte, como é possível gerar automaticamente diversos artefatos e documentações importantes para o processo de desenvolvimento.

O capítulo 8 apresenta uma ferramenta CASE que implementa a solução proposta como prova de conceito. A ferramenta permite a descrição de casos de uso através de diagramas de atividade permitindo a geração de artefatos de análise como, por exemplo, descrições textuais.

Para a realização deste trabalho foi utilizada uma vasta bibliografia. Entretanto cabe identificar os pilares que estruturaram todo o trabalho.

A principal bibliografia utilizada para apresentar os conceitos da linguagem Unified Modeling Language foi a seguinte:

- *The Unified Modeling Language Reference Manual*, de Ivar Jacobson, Grady Booch e James Rumbaugh;

- *UML Distilled: A brief guide to the standard object modeling language,* de Martin Fowler e Kendall Scott.

A bibliografia utilizada para apresentar em maiores detalhes a modelagem de casos de uso, a descrição dos mesmos, e a sua importância dentro do processo unificado de desenvolvimento (RUP) foi a seguinte:

- *Advanced Use Case Modeling: Software systems*, de Frank Armour e Granville Miller;

- *Applying UML and Patterns: An introduction to object-oriented analysis and design,* de Craig Larman;

- *The Unified Software Development Process*, de Ivar Jacobson, Grady Booch e James Rumbaugh.

Os conceitos que definem o modelo de análise apresentado neste trabalho, foram extraídos da seguinte fonte:

- *Object-Oriented Software Engineering: A use case driven approach*, de Ivar Jacobson, Magnus Christerson, Patrik Jonsson e Gunnar Övergaard.

Algumas boas práticas de descrição de casos de uso também são apresentadas, muitas extraídas de artigos como os de Alistair Cockburn, Martin Fowler e Scott Ambler, ou de livros como o de Doug Rosenberg e Kendall Scott, e o de Geri Schneider e Jason Winters.

1.1 Unified Modeling Language (UML)

A Unified Modeling Language (UML) é uma linguagem visual de modelagem de propósito genérico, utilizada para especificar, visualizar, construir e documentar artefatos[1] de um sistema [Rumbaugh 1999]. A linguagem tem como objetivo unificar as melhores práticas de modelagem em uma abordagem padrão.

A UML captura informações sobre a estrutura estática e o comportamento dinâmico de um sistema. A estrutura estática define os conceitos de uma aplicação (modelados como classes), suas propriedades internas e os relacionamentos entre estes. O comportamento dinâmico pode ser descrito

[1] *Artefato* é um termo genérico para designar qualquer tipo de informação criada, produzida, modificada ou utilizada pelas pessoas envolvidas no processo de desenvolvimento de um sistema. Um artefato pode ser um modelo, uma descrição, um esboço de uma interface gráfica, um software, etc.

através da análise da vida de um objeto à medida que interage com o resto do mundo, ou através de padrões de comunicação entre objetos conectados à medida que interagem para implementar um comportamento.

Os modelos são utilizados para capturar e documentar requisitos e conhecimentos do domínio, de forma que as pessoas envolvidas no processo de desenvolvimento de um sistema possam entender a descrição criada e se comunicar através dela.

A UML possui uma série de diagramas que permitem modelar o sistema sob vários ângulos, criando uma série de visões que facilitam o entendimento e mapeamento do sistema para um modelo. A utilização de um modelo de representação com múltiplas facetas permite a criação de um vocabulário e de um meio de comunicação efetivo entre as pessoas envolvidas no desenvolvimento de um sistema.

As várias visões que UML oferece podem ser separadas em três grupos principais: estrutura, comportamento dinâmico e gerenciamento do modelo. A visão estrutural descreve elementos no sistema e sua relação com outros elementos: possui uma visão estática, visão de caso de uso, visão de implementação e visão de *deployment*.

O comportamento dinâmico descreve o comportamento do sistema ao longo do tempo, através de visões de máquina de estados, de atividade e de interação. O gerenciamento do modelo descreve a organização dos modelos propriamente ditos em unidades hierárquicas que permitem o agrupamento dos modelos em pacotes. Um resumo das visões, dos diagramas utilizados e dos principais conceitos utilizados pode ser visto na tabela 1.

A UML possui mecanismos de extensibilidade, como, por exemplo, os estereótipos que permitem criar novos elementos a partir de outros já existentes. A vantagem desse tipo de mecanismo é a possibilidade de criar extensões da UML sem alterar o seu meta-modelo.

Tabela 1 - Visões e diagramas da UML

Grupos principais	Visão	Diagramas	Conceitos principais
Estrutural	visão estática	diagrama de classe	classe, associação, generalização, dependência, realização, interface
	visão de caso de uso	diagrama de caso de uso	caso de uso, ator, associação, “extend”, “include”, generalização de caso de uso
	visão de implementação	diagrama de componente	componente, interface, dependência, localização

	visão de deployment	diagrama de deployment	nodo, componente, dependência, localização
Dinâmico	visão de máquina de estados	diagrama de estado	estado, evento, transição, ação
	visão de atividade	diagrama de atividade	estado, atividade, transição, *fork*, *join*
	visão de interação	diagrama de seqüência	interação, objeto, mensagem, ativação
		diagrama de colaboração	colaboração, interação, papel de colaboração, mensagem
gerenciamento de modelo	visão de gerenciamento de modelo	diagrama de classes	pacote, subsistema, modelo

1.2 Casos de Uso

Os casos de uso são os veículos de captura de requisitos e servem de base para a definição de requisitos funcionais. Além de facilitarem a visualização da aplicação, ajudam na delimitação do sistema. Servem como meio de comunicação com usuários finais e clientes por oferecerem uma visão dinâmica e fechada do sistema.

Outra característica dos casos de uso é a de servirem como base para a descoberta de objetos e servirem como ferramenta para a rastreabilidade de requisitos. Através deles pode-se definir a interface com o usuário, e ajudam na visualização de elementos desde requisitos funcionais até objetos e estruturas de componentes.

Também são utilizados como mecanismo para definir a interação entre objetos e interfaces de objetos, bem como podem ser usados como base para a alocação de funcionalidades a componentes e objetos, e inclusive para definir padrões de acesso à base de dados.

Os casos de uso ajudam a dimensionar a capacidade de processamento necessária. São utilizados também para a definição dos testes (case test), como base para os testes de integração, e como fonte para a documentação de usuários e manuais.

Servem como base para o desenvolvimento incremental e ajudam a estimar o tamanho de projetos e recursos necessários; são utilizados como ferramenta para controlar um projeto. As atividades de desenvolvimento são conduzidas através deles. Os casos de uso se tornaram a forma padrão de representação de processo de negócio.

1.3 Processo de análise

A modelagem de casos de uso pode ser vista como sendo uma atividade de um framework de processo, que pode ser customizado para uma organização em particular [Armour 2000].

O desenvolvimento de software pode ser visto de forma genérica em fases, como mostrado na figura 1. A fase de análise de requisitos e análise propriamente dita, consiste na realização de uma análise de domínio, especificação de interfaces, definição da arquitetura e modelagem de casos de uso.

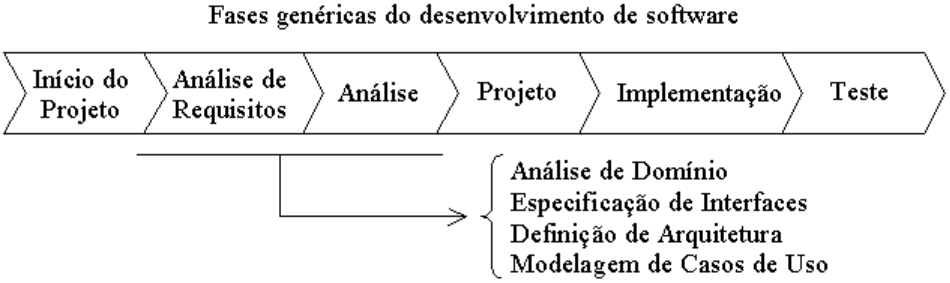


Figura 1 - Fases genéricas do desenvolvimento de software

O modelo de caso de uso descreve o comportamento composto do sistema, através da combinação de casos de uso e atores. Os casos de uso oferecem uma forma incremental e modular de descrever um sistema, evidenciando a forma como um sistema complexo é utilizado pelos seus usuários. Adições ou mudanças na funcionalidade do sistema são facilmente feitas no modelo, que descreve um sistema completo e cuja descrição geralmente envolve muitos casos de uso.

Os casos de uso provêm representações de requisitos do sistema que podem ser facilmente entendidos por diferentes participantes[2] do projeto. Cada caso de uso é descrito utilizando textos narrativos e um pequeno conjunto de símbolos de fácil compreensão, definidos na linguagem UML.

O processo tradicional de análise utilizando casos de uso envolve os seguintes passos:

1. Encontrar os atores
2. Encontrar os casos de uso
3. Descrever cada caso de uso

Entretanto outros passos estão implícitos ou são opcionais:

[2] Tradução de *stakeholders*, que representa os participantes do processo de desenvolvimento, desde usuários e clientes até analistas, projetistas e desenvolvedores.

1. Definir os limites do sistema
2. Encontrar os atores
3. Encontrar os casos de uso
4. Descrever cada caso de uso
5. Refinar o modelo de casos de uso
6. Priorizar casos de uso
7. Adicionar requisitos futuros
8. Organizar o modelo de casos de uso

Em sistemas pequenos, apenas os três passos essências são suficientes para criar um modelo de casos de uso. Já para sistemas mais complexos, um processo mais elaborado torna-se necessário. O Rational Unified Process é um exemplo de processo comercial mais elaborado.

1.4 Rational Unified Process (RUP)

O RUP é um processo de desenvolvimento de software. Um processo de desenvolvimento de software é um conjunto de atividades necessárias para transformar os requisitos de um usuário em um sistema [Jacobson 1998]. Entretanto o RUP é mais do que um simples processo: ele é um *framework* genérico que pode ser especializado para uma grande gama de sistemas, de diferentes áreas de atuação, de diferentes tipos de organizações e diferentes tamanhos de projeto.

Os casos de uso não são apenas uma ferramenta para especificar os requisitos do sistema. Eles direcionam o projeto, a implementação e os testes. Baseados no modelo de casos de uso, os desenvolvedores criam uma série de modelos de projeto e implementação que realizam os casos de uso. Os desenvolvedores revisam sucessivamente cada modelo para que sejam adequados ao modelo de casos de uso. Os testadores do sistema testam a implementação para garantir que os componentes por esta gerados implementem corretamente a funcionalidade descrita nos casos de uso.

Dentro do RUP, os casos de uso orientam o processo, mas também são desenvolvidos de acordo com a arquitetura do sistema. Tanto a arquitetura quanto os casos de uso amadurecem durante o ciclo de vida do sistema.

O RUP é um processo iterativo, onde a cada iteração, são identificados e especificados os casos de uso relevantes, é criado um projeto baseado na arquitetura escolhida como guia, é implementado o projeto em componentes, e é verificado se esses componentes satisfazem os casos de uso. Se a iteração atinge o seu objetivo, o desenvolvimento procede com a próxima iteração.

Os conceitos do RUP são os seguintes:

- Orientado por casos de uso;
- Centrado na arquitetura;
- Iterativo e incremental;

Esses conceitos são igualmente importantes no RUP. A arquitetura provê a estrutura que guia o trabalho nas iterações, enquanto os casos de uso definem os objetivos e orientam o trabalho de cada iteração. Estes são os três pilares do RUP; sem algum deles, o processo perde severamente o seu valor.

1.4.1 Processo conduzido por casos de uso

Dentro do processo, os casos de uso são utilizados não apenas como ferramenta de captura de requisitos, mas como elemento que conduz o projeto, a implementação e o teste: ou seja, os casos de uso conduzem o processo de desenvolvimento de software.

Os casos de uso não apenas iniciam o processo de desenvolvimento, mas também servem como base para uma série de workflows que são seguidos durante o desenvolvimento.

O modelo de casos de uso tem papel fundamental no Processo Unificado, é a partir dele que surge uma série de modelos, mostrada a seguir:

- O modelo de análise, que especifica o modelo de casos de uso, refinando os casos de uso em maior detalhe e realizando uma alocação inicial do comportamento do sistema a um conjunto de objetos;

- O modelo de projeto define a estrutura estática do sistema em subsistemas, classes, interfaces e realizam os casos de uso através de colaborações[3] entre esses elementos;

- Os modelos de implementação incluem componentes (que representam o código fonte) e o mapeamento de classes em componentes. O modelo de deployment define os nodos físicos de computadores e os mapeamentos de componentes nesses nodos;

- O modelo de testes, que descrevem os casos de testes que validam se a implementação atende os requisitos definidos nos casos de uso.

[3] Colaborações são o conjunto de classes, interfaces e outros elementos que juntos promovem um comportamento cooperativo maior que a soma de todos os elementos. Um diagrama de colaboração enfatiza a organização estrutural dos objetos que enviam e recebem mensagens, mostrando as interações entre eles.

Na figura 2 pode-se ver a relação do modelo de casos de uso com outros modelos de sistema. Todos esses modelos estão relacionados. Juntos eles representam o sistema.

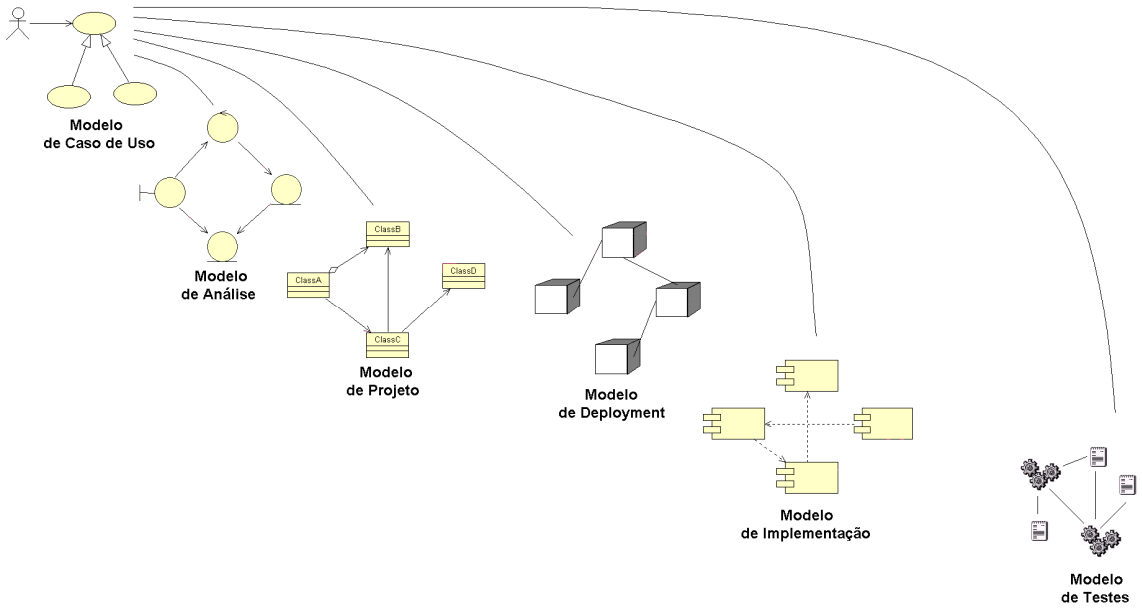


Figura 2 - Modelos do Processo Unificado

Os casos de uso não apenas iniciam o processo de desenvolvimento de software, mas também o mantém coeso. Eles também ajudam os gerentes de projeto a planejar, atribuir e monitorar muitas das tarefas atribuídas aos envolvidos na construção do sistema.

1.4.2 Processo iterativo

O RUP divide o processo de desenvolvimento em Incepção, Elaboração, Construção e Transição. Cada uma dessas fases pode ser dividida em iterações. Os objetivos de cada fase são os seguintes:

- O objetivo da fase de incepção é a definição do escopo do que o produto deve fazer, reduzir os piores riscos, e estabelecer os objetivos do ciclo de vida para o projeto;

- A fase de elaboração tem como objetivo a definição de uma linha-base para a arquitetura do sistema e a captura da maior parte dos requisitos;

- A fase de construção tem como objetivo o desenvolvimento completo do sistema e garantir a transição do sistema para os clientes;

- O objetivo da fase de transição é garantir que o produto está pronto para ser liberado à comunidade de usuários.

Essas divisões ajudam o gerenciamento e os clientes a perceberem o que foi feito durante as fases de incepção e elaboração – tipicamente fases de menor custo -, antes de decidir pela realização da construção do sistema – geralmente a fase mais cara do ciclo de desenvolvimento.

Uma iteração é um mini-projeto – um ciclo quase completo em todos os fluxos de trabalho envolvidos – resultando em uma liberação interna. Uma iteração genérica passa pelos seguintes fluxos: levantamento de requisitos, análise, projeto, implementação e testes. A figura 3 retrata os elementos que compõe uma iteração.

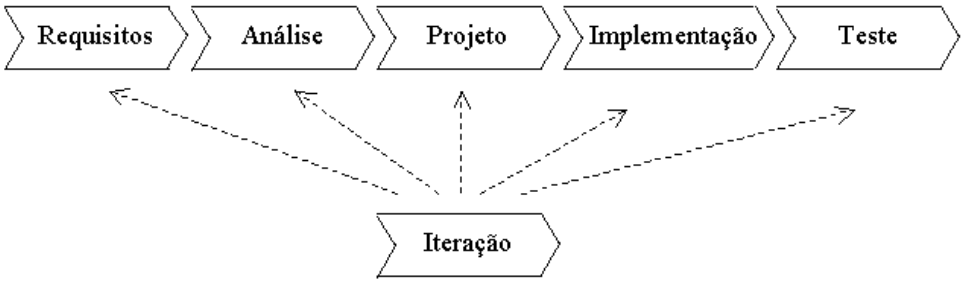


Figura 3 - Uma iteração do RUP

Os casos de uso ajudam na manutenção do processo iterativo do RUP. A cada iteração os casos de uso guiam todos os fluxos de trabalho envolvidos, desde o levantamento de requisitos até a implementação e testes. Em cada iteração do processo de desenvolvimento do sistema um conjunto de casos de uso é identificado e implementado.

1.4.3 Arquitetura de software

A arquitetura do sistema pode ser definida como o conjunto de decisões significativas sobre a organização do sistema, a escolha dos elementos estruturais e interfaces que compõe o sistema. A arquitetura de software não se preocupa apenas com a estrutura e o comportamento do sistema, mas também com a usabilidade, funcionalidade, desempenho, reuso, compreensibilidade, bem como restrições econômicas e tecnológicas.

Os casos de uso ajudam na definição da arquitetura. Ao escolher um conjunto adequado de casos de uso – aqueles significantes à definição da arquitetura – que serão realizados nas primeiras iterações, definindo e

implementando um sistema com uma arquitetura estável, que pode ser utilizada nos ciclos de vida seguintes.

A seleção de casos de uso inicial inclui aqueles casos de uso que os clientes mais precisam. Quando a arquitetura se torna estável, é possível complementar a funcionalidade do sistema através da implementação do resto dos casos de uso.

Figura 4 - Relação entre casos de uso e a arquitetura

O desenvolvimento da arquitetura do sistema é orientado pelos casos de uso, e a arquitetura guia quais os casos de uso que podem ser implementados, como mostrado na figura 4.

O modelo 4+1 [Krutchen 1995] é uma proposta de organização da descrição de uma arquitetura de software utilizando uma série de visões, cada uma descrevendo um determinado conjunto de características.

A representação é realizada através de quatro visões (lógica, de processo, física e de desenvolvimento) interligadas por uma quinta visão, a de casos de uso ou cenários, como mostrado na figura 5.

A visão lógica representa o modelo de objetos de projeto; a visão de processo captura características de concorrência e distribuição do projeto; a visão física descreve o mapeamento do software para o hardware e reflete suas características de distribuição; e a visão de desenvolvimento que descreve a organização estática em seu ambiente de desenvolvimento.

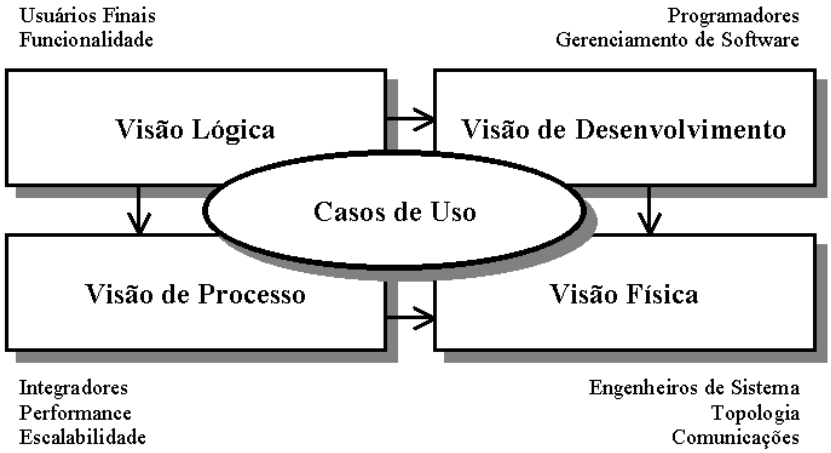


Figura 5 - O modelo de visão "4+1"

A descrição de uma arquitetura pode ser organizada ao redor dessas quatro visões e ilustrada por alguns cenários ou casos de uso. A quinta visão serve como mecanismo de descoberta de elementos da arquitetura durante o projeto da mesma e como meio de validar a arquitetura depois dela estar pronta.

2 Modelagem de Casos de Uso

Neste capítulo são apresentados em maior profundidade os elementos que compõe a modelagem de casos de uso dentro dos processos convencionais de análise de sistemas orientada a objetos.

2.1 Diagramas de casos de uso

Parte da modelagem de casos de uso envolve a criação de um diagrama de casos de uso, que é uma forma de mostrar o modelo sem entrar nos detalhes dos casos de uso propriamente ditos.

Uma das etapas iniciais da modelagem é a definição dos limites do sistema, que geralmente é realizada e não completada. No início da modelagem geralmente não existe consenso sobre o quê deve ser construído.

O sistema geralmente é delimitado em um diagrama de casos de uso por um retângulo com o nome do sistema identificado. O retângulo é chamado de limite do sistema[4]. Os elementos que fazem parte do sistema estão dispostos dentro do limite. Entidades externas que não são parte do sistema e que interagem com este são mostradas do lado de fora, como mostrado na figura 6.

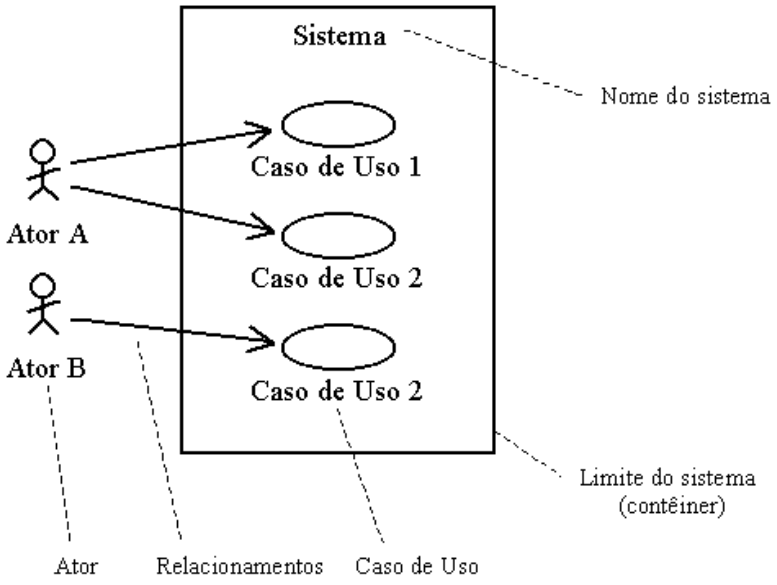


Figura 6 - Elementos de um diagrama de caso de uso

O nome do sistema é necessário para completar o primeiro passo. O nome geralmente é demonstrativo do seu objetivo, uma vez que os participantes devem

[4] Em inglês, *system boundary*.

poder facilmente identificar o que eles esperam do sistema e onde ele começa e termina.

Os outros elementos que compõe o diagrama de caso de uso são os seguintes:

1. **Casos de uso**: podem ser descritos elementos que representam uma seqüência de ações requeridas pelo sistema;

2. **Atores**: Podem ser qualquer coisa fora do sistema que troque informações com o mesmo, incluindo usuários e outros sistemas;

3. **Contêineres**: um contêiner contém casos de uso de um sistema (limite do sistema). Pacotes[5] podem ser utilizados para conter casos de uso e/ou atores e inclusive outros contêineres;

4. **Relacionamentos**: existem vários tipos de relacionamentos como, por exemplo, o de associação, que liga casos de uso e atores, indicando a forma como interagem; relacionamentos que representem dependência, por exemplo, entre dois pacotes; relacionamentos entre casos de uso, como o de "extend" e "include"; assim como a generalização, que pode, por exemplo, indicar a herança de comportamento entre dois atores.

2.2 Elementos

2.2.1 Atores

"Um ator é uma entidade que interage com o sistema com a finalidade de completar um evento".

Ivar Jacobson

Atores não são necessariamente usuários humanos do sistema, e sim entidades do ambiente que interagem com o sistema (humanos, outros sistemas, dispositivos, sensores, etc.), iniciando eventos ou como resultado de um evento.

Os atores ajudam no entendimento de quais interações ocorrem com o sistema. Quando o ator é humano, ele representa um papel executado por um usuário interagindo com o sistema, e não a personificação de um indivíduo (um ator seria, por exemplo, o conceito "aluno", e não "João Silva", o aluno).

Da mesma forma que um ator pode modelar mais de uma pessoa, uma pessoa pode desempenhar o papel de mais de um ator. Excepcionalmente quando um ator representa uma entidade física, como, por exemplo, um sensor ou um outro sistema, pode ser explicitamente identificado como tal.

[5] Em inglês, *package*.

A definição dos atores oferece perspectivas do porquê da necessidade de um caso de uso e as características correspondentes do mesmo. Focar a atenção nos atores significa concentrar-se em como o sistema será utilizado e não em como ele será construído.

Os atores podem ser categorizados [Jacobson 1992] em dois tipos:

1. **Atores primários**: são usuários que obtêm informações do sistema e que diretamente o utilizam. Cada um desses atores realiza uma ou mais tarefas principais do sistema. O sistema é construído para esse tipo de ator (por exemplo, o ator “cliente”);

2. **Atores secundários**: os atores secundários supervisionam e mantêm o sistema. Eles existem apenas para que os atores primários possam utilizar o sistema. Este tipo de ator pode ser um humano (por exemplo, um “operador”) ou então outro sistema (por exemplo, um ”sistema de armazenamento de banco de dados”).

Os atores também podem assumir diversas personalidades [Armour 2000]. As personalidades podem ser utilizadas no processo de descoberta e identificação de atores. Um ator pode ter múltiplas personalidades em um ou mais casos de uso.

Alguns tipos de personalidades poderiam ser as seguintes:

1. **Iniciador**: aquele que inicia, ou dispara, o caso de uso;

2. **Servidor externo**: é uma entidade externa (pessoa, organização, ou sistema) que responde a uma requisição do sistema;

3. **Receptor**: é uma entidade externa que recebe informação do sistema (por exemplo, um repositório de dados);

4. **Facilitador (proxy)**: serve como intermediário entre o ator primário e o sistema.

Os atores também podem ser vistos como atores de negócio ou de sistema. Um ator de negócio, modelado em casos de uso de negócio, é uma entidade que interage com o ambiente de negócio. Um ator de sistema, modelado em casos de uso de sistema, geralmente interage diretamente no sistema. É o caso de uma agência de turismo onde temos o “cliente” como ator de negócio e o “agente de viagens” como ator de sistema. Em alguns casos essas características se confundem, por exemplo, em um sistema de agência de turismos com interface Web, onde o “cliente” é ao mesmo tempo um ator de negócio e um ator de sistema.

Uma possível solução seria modelar ambos os atores, criando um facilitador entre o ator de negócio e o sistema. A interface Web poderia ser identificada como tal, assim como o agente de viagens, como mostrado na figura 7

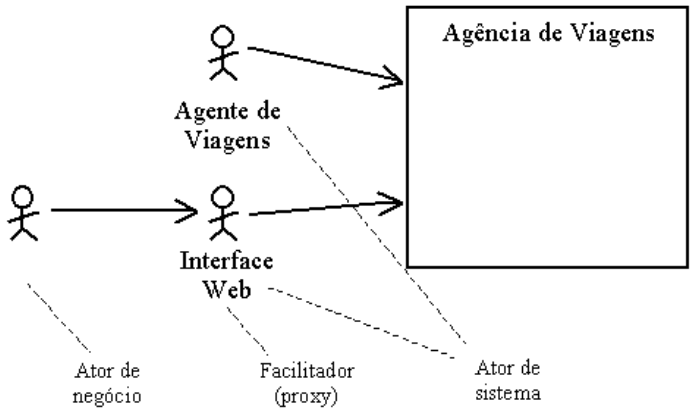


Figura 7 - Exemplo de "facilitador" para um sistema de agência de viagens

É necessário, entretanto, saber quais os verdadeiros usuários do sistema, para poder identificar os limites do mesmo: condição fundamental para iniciar as próximas fases do desenvolvimento do sistema. O mais aconselhável é realizar duas modelagens, uma de negócio e outra de sistema, já que ambas se destinam a públicos diferentes e mistura-las pode tornar a modelagem e identificação dos limites do sistema confuso. Por exemplo, no sistema de agência de viagens mostrado na figura 7 fica confuso identificar até onde vai o limite do sistema: a interface Web é ou não parte do sistema?

A identificação de tipos de atores não tem como objetivo classificar os atores rigidamente e sim ajudar a identificá-los e definir suas participações nos casos de uso.

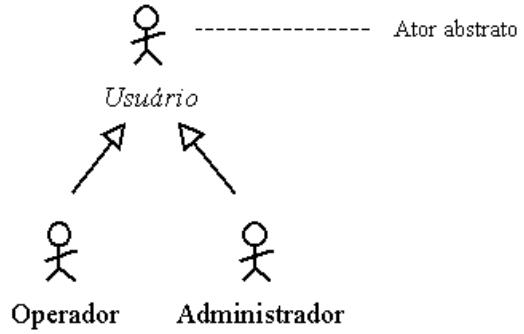


Figura 8 - Especialização/generalização entre atores

Alguns atores podem desempenhar um papel conceitual, outros um papel mais concreto. É possível então definir um “superpapel” de forma a definir um comportamento comum de interação com o sistema (ator abstrato), que possa ser

especializado para outros atores, como mostrado na figura 8. Atores abstratos [Jacobson 1992] podem também ser utilizados para especificar diferentes privilégios no sistema.

Esse relacionamento entre os atores é chamado de especialização ou generalização entre os atores. O propósito dessa representação é reduzir a redundância na comunicação dos atores com o sistema.

2.2.2 Casos de Uso

"Os casos de uso são uma forma elegante de comunicar as necessidades de um negócio ou sistema".

Ivar Jacobson

Um caso de uso mostra como os atores interagem com o sistema para atingir um objetivo. Formalmente, um caso de uso é "uma descrição de um conjunto de seqüências e ações, incluindo variantes, que um sistema realiza que levam a um resultado observável de valor a um ator" [Booch 1999].

Um caso de uso descreve um único objetivo e todas as possíveis coisas que podem ocorrer à medida que o usuário tenta alcançar esse objetivo. Para encontrar os casos de uso de um determinado sistema, deve-se examinar os objetivos do mesmo.

Os casos de uso sempre descrevem uma interação entre um sistema e ao menos um ator. Esse relacionamento entre um ator e o caso de uso é chamado de associação. Quando um ator se associa com um caso de uso diz-se que ele se comunica com o caso de uso.

As associações entre os atores e os casos de uso podem ser unidirecionais ou bidirecionais. Uma associação unidirecional é representada por uma linha em forma de seta, e uma associação bidirecional é representada apenas por uma linha, como mostrado na figura 9.

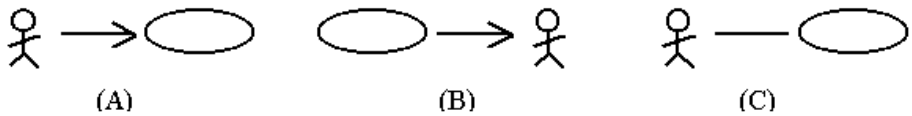


Figura 9 - Associações entre atores e casos de uso

Um ator pode iniciar a comunicação com um caso de uso (A), assim como um caso de uso pode iniciar a comunicação com um ator (B). Existem casos em que ambos podem iniciar a comunicação, sendo então uma associação bidirecional (C).

O conjunto de casos de uso compõe o **modelo de casos de uso** que descreve a funcionalidade do sistema, substituindo a especificação tradicional de requisitos funcionais de um sistema [Jacobson 1998].

Um modelo de caso de uso bem-formado deve mostrar algum tipo de associação entre cada caso de uso e algum ator. Casos de uso que não se comunicam com algum ator são suspeitos de não estarem corretos, já que todo caso de uso, por definição, oferece um valor a um ator.

As interfaces representam o protocolo ou meio pelo qual os atores interagem com sistemas complexos. Cada associação representa uma interface entre o sistema e seus atores. Uma interface é a composição de elementos necessários para a realização da interação entre o sistema e o ator.

Frank Armour e Granville Miller [Armour 2000] estendem a modelagem de casos de uso, definidos na UML, adicionando duas formas de casos de uso: casos de uso priorizados e casos de mudança[6]. O primeiro identifica o caso de uso com um nível de prioridade com o objetivo de diferenciar os casos de uso mais urgentes ou importantes. O segundo representa potenciais casos de uso futuros, que não fazem parte do modelo de desenvolvimento ou negócio atual.

Os casos de uso priorizados permitem a definição do processo de engenharia de uma forma ordenada, permitindo o reconhecimento de aspectos do sistema que devem ser desenvolvidos antes e aqueles que podem ser deixados para mais tarde.

Os casos de mudança foram apresentados por Earl Ecklund [Ecklund 1996] na OOPSLA de 1996 como forma de poder antecipar requisitos futuros e construir uma melhor arquitetura para o sistema.

Um modelo de caso de uso geralmente é construído através de uma abordagem top-down, onde se assume a existência de um conhecimento prévio sobre as funcionalidades do sistema. Inicialmente se encontram os atores, depois os casos de uso e posteriormente se realiza o detalhamento destes últimos.

Regnell [Regnell 1996] apresenta uma abordagem bottom-up para a construção de modelos de caso de uso. Nessa abordagem inicialmente se definem os cenários. A partir da generalização dos cenários é possível encontrar os casos de uso, que posteriormente são organizados dentro do modelo.

2.2.3 Cenários

Um cenário é uma seqüência de interações acontecendo em determinadas condições com o objetivo de alcançar o objetivo principal do ator, e obtendo um determinado resultado com relação a esse objetivo. As interações começam a

[6] Em inglês, *change case*.

partir da ação inicial e continuam até que o objetivo seja alcançado ou abandonado [Cockburn 2000].

Os cenários são utilizados para estruturar os casos de uso. O cenário principal representa o caminho mais comum e de sucesso para a realização do objetivo do caso de uso, como mostra a figura 10.

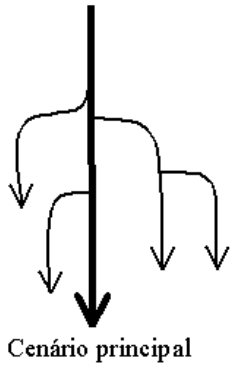


Figura 10 - Cenários de um caso de uso

Percorrer os cenários permite identificar se o caso de uso está completo ou não, uma vez que a falta de um cenário torna o caso de uso incompleto.

Um risco potencial durante a modelagem de casos de uso para definição de um desenvolvimento orientado a objetos é o fato dele poder terminar em um modelo funcional, ao invés de um modelo de objetos [Firesmith 1995].

Um problema existente em projetos grandes é o fato de que um número grande de desenvolvedores de casos de uso acaba trazendo desvios naturais do vocabulário comum e do entendimento do sistema. Casos de uso escritos por diferentes desenvolvedores podem descrever a mesma coisa de forma diferente. O resultado leva a um modelo de casos de uso disjunto e à frustração nos passos seguintes da realização do sistema.

A análise de domínio trás como resultado um glossário, utilizado para a identificação de objetos durante a análise. Uma forma de rastreabilidade é identificar para cada item do glossário o caso de uso relacionado. Essa rastreabilidade ajudará nas fases seguintes do desenvolvimento do sistema como, por exemplo, a definição da arquitetura ou o agrupamento de objetos em subsistemas.

A especificação de interfaces é outra forma de delimitar o sistema. Ela especifica as responsabilidades do sistema com as entidades externas que interagem com o sistema: os atores. Um tipo de especificação de sistema é a especificação da interface com o usuário, que deveria ser concebida a partir dos

casos de uso. Outro tipo de especificação é a especificação de interface de sistema, usualmente utilizada para comunicar-se com outros sistemas externos.

2.3 Relacionamentos entre casos de uso

Dois ou mais casos de uso podem precisar descrever a mesma funcionalidade. Os relacionamentos do tipo “extend” e “include” são uma forma de estruturar os casos de uso de modo a eliminar a redundância.

2.3.1 Relacionamento “extend”

O relacionamento de “extend” permite que um caso de uso seja estendido com um comportamento adicional ou variações. Um caso de uso pode ter vários relacionamentos de extensão. O relacionamento é representado no diagrama como uma seta com o estereótipo “<<extend>>”, como mostrado na figura 11.

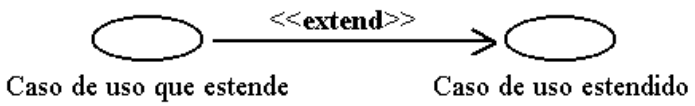


Figura 11 - Relacionamento "extend"

O caso de uso que está sendo estendido define pontos onde a funcionalidade pode ser estendia. Esses pontos são chamados de *pontos de extensão*[7].

Um caso de uso pode definir mais de um ponto de extensão. Os casos de uso que estendem um determinado caso de uso devem conhecer os pontos de extensão e indicar quais deles estão estendendo.

Este tipo de relacionamento é muito mal-interpretado e pode levar a problemas de entendimento e dificuldade de clareza na especificação de requisitos [Fowler 1999].

2.3.2 Relacionamento “include”

O relacionamento de “include” permite que um caso de uso tenha acesso a comportamentos definidos em outro caso de uso. É um bom mecanismo para capturar comportamentos comuns usados por vários casos de uso. Essa funcionalidade comum pode ser representada em casos de uso abstratos. Um caso de uso abstrato é simplesmente um pedaço de funcionalidade reutilizável. Os casos de uso abstratos têm como característica o fato de poderem estar isolados,

[7] Do inglês, *extension points*.

sem associação com algum ator. O relacionamento é representado no diagrama como uma seta com o estereótipo “include”, como mostrado na figura 12.

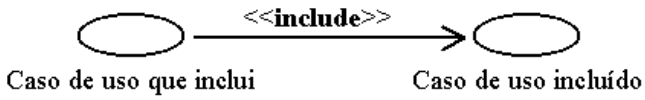


Figura 12 - Relacionamento "include"

2.3.3 Generalização e especialização

As informações comuns entre casos de usos podem ser generalizadas em um único super caso de uso, através da relação de generalização. Os casos de uso especializados podem então se focar na descrição de comportamentos específicos que devem ser documentados.

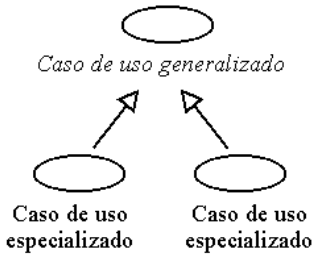


Figura 13 - Generalização / especialização de casos de uso

A figura 13 mostra como a generalização / especialização é representada no modelo de casos de uso.

Os casos de uso especializados herdam o comportamento definido no super caso de uso. Entretanto, um caso de uso especializado pode redefinir algum tipo de comportamento sobrescrevendo funcionalidades.

2.4 Processo de Modelagem

Embora a modelagem de casos de uso possa parecer simples, ela deve ser realizada de uma forma disciplinada. Armour e Miller [Armour 2000] propõe um framework para o processo de modelagem de casos de uso.

Inicialmente os casos de uso são modelados conceitualmente, focando no comportamento do sistema e são posteriormente refinados aumentando gradativamente o nível de detalhamento dos requisitos.

À medida que a análise de requisitos evolui, as descrições dos casos de uso passam por vários níveis de abstração, refinando o modelo de casos de uso gradativamente. Durante esse processo de refinamento, os casos de uso podem ter suas descrições melhor detalhadas; novos casos de uso podem ser identificados; detalhes podem ser estendidos do modelo através dos relacionamentos de “extend”, “include” e generalização.

As descrições dos casos de uso podem ser vistas em três estágios diferentes:

1. **Descrições iniciais**: as descrições são criadas no início da análise de requisitos, dando uma breve idéia dos objetivos do sistema;

2. **Descrições básicas**: expandem as descrições iniciais em um nível maior de detalhamento, concentrando-se no comportamento “ideal” do sistema;

3. **Descrições elaboradas**: detalhes do comportamento como condições lógicas ou caminhos alternativos são adicionados à descrição básica.

Adicionalmente às descrições dos casos de uso, o modelo de casos de uso com seus diagramas, casos de uso, relacionamentos, cenários instanciados[8], entre outros elementos, servem para mapear os casos de uso para o projeto, através de diagramas de seqüência ou colaboração, fluxos de dependência entre os casos de uso, e pacotes de funcionalidades.

A utilização de várias representações torna-se necessária uma vez que diversos participantes possuem diferentes visões do sistema. As múltiplas representações ajudam a entender e validar o modelo. Os casos de uso devem ser entendidos tanto pelos clientes, usuários e também projetistas que precisam mapear os casos de uso para o modelo de objetos.

Uma vez que o projeto é concebido, os desenvolvedores devem escolher, refinar ou criar um framework de processo para o projeto. O Rational Unified Process (RUP) é um exemplo de framework comercial.

O framework precisa ser ajustado levando-se em consideração alguns fatores específicos como, por exemplo, o tamanho do projeto, a natureza do mesmo, o uso de um desenvolvimento incremental e iterativo, e a experiência do time.

Cada participante deve ajudar no esforço de modelagem de casos de uso, uma vez que cada tipo de participante possui necessidades e perspectivas

[8] Os cenários instanciados descrevem exemplos de como os casos de uso são executados; são úteis para a validação e geração de planos de teste.

diferentes. A modelagem de casos de uso envolve uma série de participantes, como mostrado na tabela 2.

Tabela 2 - Benefícios trazidos pelo modelo de casos de uso

Participante	Benefícios trazidos pelo modelo
Cliente	- O modelo oferece requisitos do cliente capturados para validação; - Ajuda a determinar o escopo geral do sistema; - Ajuda a estimar prazos e custos; - Serve como base para testes de homologação.
Usuário	- O modelo oferece requisitos de usuário capturados para validação; - Modela a interação do usuário com o sistema.
Gerente de Projeto	- Ajuda a estimar prazos e custos; - Ajuda a estimar o risco do projeto; - Ajuda a mapear requisitos; - Ajuda a rastrear o progresso do desenvolvimento do sistema.
Arquiteto do Sistema	- Ajuda a identificar requisitos arquiteturais; - Conduz a arquitetura do sistema; - Facilita a análise de prós e contras de escolhas tecnológicas; - Ajuda a atingir a completeza, consistência e coerência da arquitetura.
Desenvolvedor do Sistema	- Oferece os modelos de requisitos para o projeto do sistema; - Ajuda na documentação do sistema.
Mantenedor do Sistema	- Provê um guia para modificação do sistema; - Provê um guia para a evolução da arquitetura.

O processo deve ser customizado pela organização de tal forma que o papel da modelagem de casos de uso fique claro, assim como sua relação com outros modelos de análise e projeto. A padronização deve ser estabelecida de forma a estabelecer diretivas de como, por exemplo, representar apropriadamente descrições textuais ou o grau de detalhamento ao qual uma descrição de caso de uso deve chegar.

Durante a modelagem inicial de casos de uso, os comportamentos chave do sistema são definidos, assim como os principais atores e um mapeamento inicial para objetos de negócio é realizado.

Inicialmente a descrição dos casos de uso básica não leva em consideração todas as alternativas, exceções e variações que um único caso de uso pode ter. O tempo gasto em analisar todas as alternativas e a necessidade de mais análise com usuários e clientes pode levar a discussões que possam fazer

perder o foco do problema. Além disso, quanto maior o detalhamento, menor a clareza para usuários finais.

Os casos de uso servem como meio de validação do modelo de objetos de domínio, garantindo que todas as abstrações necessárias foram encontradas.

A expansão dos casos de uso de uma descrição inicial para uma descrição básica (com maiores detalhes) consiste em elaborar o fluxo de eventos e outras informações relacionadas, a inclusão de relacionamentos como “extend”, “include” e generalizações, mapear os casos de uso para modelos de objetos e de análise, e o desenvolvimento de cenários instanciados.

3 Descrição de Casos de Uso

Neste capítulo serão estudadas com maior profundidade as descrições dos casos de uso. Estas descrições não fazem parte do modelo definido para o sistema através da linguagem UML e são construídas na grande maioria dos casos através de uma linguagem narrativa própria.

Um caso de uso é uma representação narrativa do comportamento do sistema. Ele é composto por um nome e uma descrição. A estrutura de descrição de um caso de uso divide o corpo de descrição em partes lógicas. As variações das estruturas de um caso de uso variam para cada organização, mas um conjunto mínimo de elementos está presente na maioria dos padrões[9] de descrições de casos de uso. Os principais elementos são:

1. **Atores**: descrição dos atores envolvidos no caso de uso;

2. **Pré-condições**: regras de estado que definem como o sistema deve se encontrar antes de disparar o caso de uso;

3. **Fluxo de eventos**: atividades que ocorrem entre os atores e o sistema à medida que interagem para atingir um objetivo;

4. **Pós-condições**: regras de estado que definem como o sistema deve se encontrar depois de executado o caso de uso.

O fluxo de eventos descreve todas as formas como um objetivo pode ser alcançado. Pode ser descrito na forma narrativa ou sob a forma de passos. Durante a descrição diversos pontos de decisão normalmente aparecem. Um ponto de decisão é uma situação onde o sistema ou ator deve decidir por que caminho seguir para atingir um objetivo.

Durante a descrição eventualmente exceções podem ocorrer, levando a seqüências alternativas que podem retornar a um fluxo que leva ao objetivo desejado ou então terminar sem sucesso.

3.1 Casos de uso essenciais e reais

Os casos de uso podem ser classificados quanto ao seu nível de abstração em relação à tecnologia utilizada [Larman 1998].

Casos de uso **essenciais** são casos de uso cuja descrição permanece relativamente livre de detalhes sobre tecnologia e formas de implementação.

[9] Padrões ou *templates* são documentos com a estrutura de descrição de um caso de uso já definida.

Decisões de projeto são ignoradas e abstraídas, especialmente aquelas que se referem à interface com o usuário.

Já os casos de uso **reais** descrevem concretamente o fluxo de eventos utilizando termos relacionados com o projeto do sistema, como por exemplo, tecnologias para representar a entrada e saída de informação do sistema (interfaces gráficas com o usuário).

Tabela 3 – Exemplos de casos de uso essenciais e reais

Caso de uso essencial

Ator	Sistema
1. O operador identifica todos os items.	
	2. O sistema mostra o preço e espera pela quantidade de items comprados.
3. O operador informa a quantidade dos items.	
	4. A informação do preço total é informada.
5. O operador confirma a compra.	
	5. O sistema realiza a compra e imprime o comprovante da compra.

Caso de uso real

Ator	Sistema
1. Para cada item o operador informa o código do produto no campo "Código" da janela de Compras.	
	2. O sistema busca o item de mesmo código e apresenta as informações de preço no campo "Valor" da janela de Compras. Habilita o campo "Quantidade" para ser preenchido e foca o cursor nesse campo.
3. O operador preenche o campo "Quantidade' e pressiona <Enter> ou clica no botão "OK" da janela.	
	4. O sistema mostra no campo "Total" a multiplicação da quantidade pelo preço do item corrente.
...	...

Entretanto, a descrição de elementos de interface e outros detalhes de implementação e projeto na descrição dos casos de uso é visto por outros autores como um erro. É consenso que o caso de uso deve apenas representar os requisitos do sistema e não se ater a detalhes de implementação. Evitar colocar muitos detalhes e informações sobre a interface com o usuário torna a descrição do caso de uso mais clara e simples de se entender [Cockburn 2002].

3.2 Fluxo de eventos

O fluxo de eventos representa atividades realizadas pelo ator ou pelo sistema, ordenadas e que resultam em um "fim" ou pós-condição. Os objetivos de fazer uma descrição mais detalhada são: entender as interações entre o sistema e os usuários, bem como o comportamento do sistema; documentação do escopo do caso de uso; iniciar o levantamento de requisitos não comportamentais; e ajudar a determinar as prioridades dos casos de uso dentro do esforço do desenvolvimento.

Em uma descrição básica de caso de uso, o fluxo de eventos descreve as atividades básicas que ocorrem durante o diálogo entre o ator e o sistema. O fluxo indica a ordem em que as atividades ocorrem como uma seqüência de passos ou texto-livre. O uso de passos é mais conveniente por ser facilmente mapeável e referenciável para outros artefatos de análise ou projeto.

A descrição do caso de uso pode ser realizada de diferentes formas e níveis de formalidade. A escolha da forma de representação depende basicamente dos participantes do projeto.

Martin Fowler e Alistair Cockburn destacam a importância de se manter uma descrição de caso de uso o mais simples possível [Fowler 1998]. O objetivo do caso de uso de validar os requisitos com o usuário deve ser sempre mantido. Alguns analistas caem no erro de documentar excessivamente os casos de uso, alguns inclusive com uma pseudo-programação, difícil de ser entendida pelos usuários.

Não há consenso sobre qual deve ser a melhor forma de se descrever um caso de uso, mas é unânime o objetivo de mantê-lo claro o suficiente para poder ser validado com o usuário e ser substancial a ponto de servir em outras etapas do ciclo de desenvolvimento de software.

Descrever o caso de uso em forma de prosa pode servir para a modelagem de negócio, mas não deixa clara uma idéia de seqüência. Já a descrição por passos introduz o conceito de lista numerada e é uma forma mais estruturada que se aproxima conceitualmente mais aos desenvolvedores. Entretanto, quando a análise não determina precisamente uma ordem, ou esta não é importante, o formato em passos pode forçar um ordenamento que não é necessário ou até errado. Uma outra alternativa de descrição é o uso de estados[10], onde cada parágrafo é identificado por um estado e seu conteúdo define as formas de passagem de um estado para outro. O uso de estados é uma boa alternativa para domínios que possuam vários caminhos não lineares no caso de uso como, por exemplo, para um sistema de telecomunicações.

Na forma como os casos de uso são descritos, alguns advogam o uso de uma visão externa (ou caixa-preta) e outros o uso de uma visão interna (ou caixa-branca). Na visão externa, somente as atividades que são visíveis ao ator externo são descritas e nada é indicado de como o sistema faz para realizar ou suportar as interações. Na visão interna, é descrito o comportamento interno do sistema. Essas descrições são requisitos (e não como é realizado ou implementado) que acaba "abrindo" o sistema para a descrição de requisitos internos do sistema não visíveis do lado de fora.

[10] Diagramas de estado, assim como outros artefatos, podem ser utilizados para expandir a descrição do caso de uso como, por exemplo, descrevendo o estado de um objeto. Neste tipo de descrição apresentada, os estados são utilizados como descrição propriamente dita, e não como um complemento.

Embora a descrição interna do sistema não signifique descrever a implementação ou elementos de projeto, ela apresenta detalhes difíceis de validar com o usuário, e pode levar à decomposição funcional do sistema. A visão externa permite se concentrar na interação com o usuário, mas pode acabar perdendo requisitos críticos para a concepção do modelo de objetos.

Durante uma modelagem intensa dos casos de uso é preferível realizar em paralelo o modelo de objetos (e não esperar pelo modelo de casos de uso final) pelo simples fato deste último ajudar na compreensão dos requisitos que estão sendo levantados.

Da mesma forma que uma visão externa pode levar à perda de alguns requisitos, a visão interna pode levar a um detalhamento complexo, parecido com um pseudocódigo, diferente do objetivo principal da análise envolvendo os casos de uso: o levantamento de requisitos.

Se o caso de uso é complexo, o uso de uma visão externa facilita a validação com o usuário. Uma visão interna é necessária para agregar valor a projetistas e testadores.

Embora a visão interna possa ser representada através de outros artefatos, como por exemplo, através de diagramas de seqüência ou de colaboração, nem sempre essas alternativas são utilizadas, e geralmente acabam tornando mais confuso a descrição da funcionalidade pretendida.

O fluxo de eventos pode ser contínuo ou separado por colunas, como inicialmente proposto por Wirfs-Brock [Wirfs-Brock 1993] e utilizado por Larman [Larman 1998], conforme a figura 14. Essa forma de representação serve tanto para a utilização de uma descrição caixa-preta (visão externa) como para caixa-branca (visão interna).

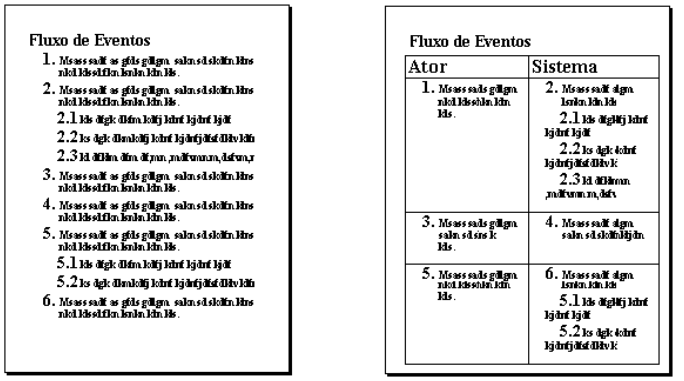


Figura 14 - Descrição do fluxo de eventos

Le Roy Mattingly e Harsha Rao [Mattingly 1998] propuseram o uso de casos de colaboração, que seriam casos de uso estereotipados, e que resolveriam o problema da visão interna. Consiste em descrever o caso de uso conforme uma caixa-preta e dentro dos casos de colaboração, cuja estrutura é semelhante ao de um caso de uso, detalhar o comportamento complexo do sistema. A vantagem é a manutenção do caso de uso, uma vez que a interface com o ator já está definida pelo caso de uso (caixa-preta) e as descrições internas do sistema estão nos casos de colaboração (caixa-branca). A figura 15 exemplifica como a descrição do caso de uso ficaria simplificada através do uso de casos de colaboração.

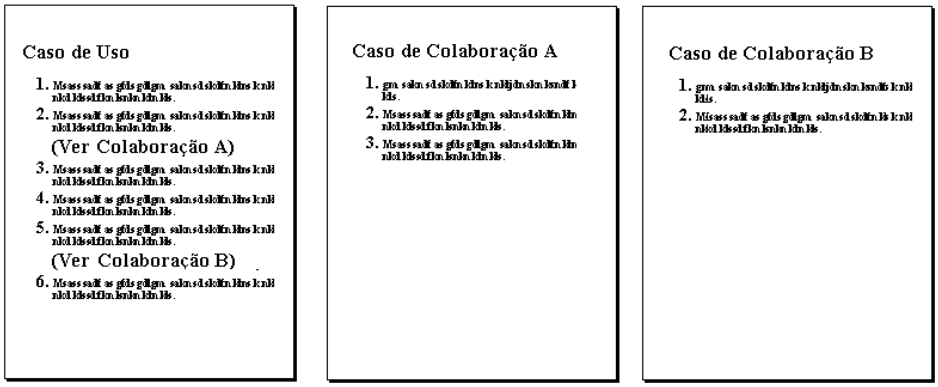


Figura 15 - Casos de colaboração

Armour e Miller [Armour 2000] propõem um guia para a descrição do fluxo de eventos:

1. Usar números ou pontos para delinear as atividades;
2. Manter o fluxo de eventos o mais entendível possível;
3. Evitar entrar em muitos detalhes;
4. Procurar por outros atores;
5. Cuidar do tamanho do fluxo de eventos;
6. Possuir padrões para a criação de fluxos de eventos.

A descrição do caso de uso deveria ser entendida por todos e poderia servir, inclusive, de contrato para o cliente.

Durante o refinamento das descrições iniciais dos casos de uso para uma descrição básica, um maior número de detalhes e requisitos tornam-se evidentes. Entretanto múltiplos casos de uso podem ser derivados a partir de um único caso de uso.

Dentro de um processo iterativo deve-se priorizar o refinamento dos casos de uso das primeiras iterações.

Os novos casos de uso podem aparecer da descoberta de novas funcionalidades do sistema ou pela divisão de funcionalidades de um caso de uso

existente. Um exemplo da divisão de um caso de uso em dois é quando durante o refinamento do caso de uso e descrição do fluxo de eventos surgem dois fluxos ou alternativas que parecem ser as principais (divisão vertical).

Outra forma de divisão é a divisão horizontal, quando se descreve dois casos de usos seqüenciais em um único fluxo, tornando a descrição muito extensa. A figura 16 ilustra a divisão vertical e a divisão horizontal de casos de uso.

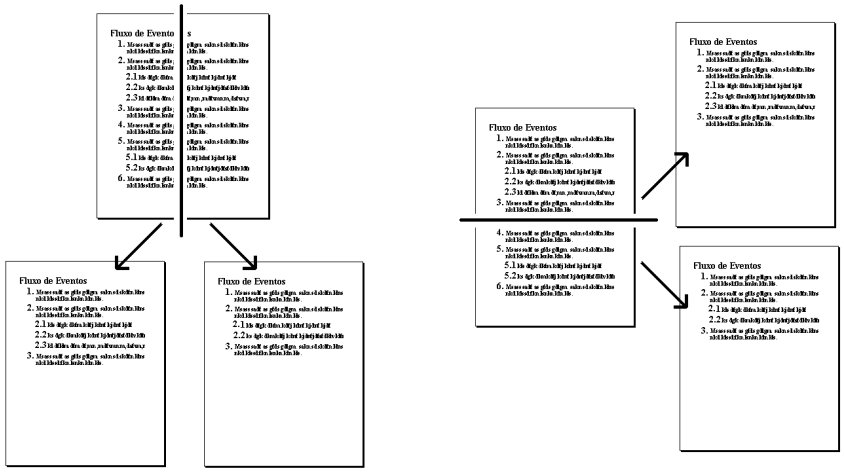


Figura 16 - Divisão vertical e horizontal de descrições de casos de uso

Segundo Armour e Miller [Armour 2000], uma descrição elaborada de casos de uso é expandida a partir da descrição básica de casos de uso, através do detalhamento adicional de fluxos alternativos e da introdução de elementos de lógica condicional ao fluxo de eventos.

3.3 Seqüências alternativas e lógica condicional

Uma seqüência alternativa descreve o comportamento que ocorre quando uma alternativa ou variação ocorre dentro do fluxo principal de eventos. Uma alternativa pode incluir um processamento diferenciado baseado nas entradas do usuário, uma decisão tomada no fluxo de eventos ou uma condição excepcional ocorrida como, por exemplo, a insuficiência de saldo em uma transação bancária.

As alternativas podem ser documentadas dentro do fluxo principal – por exemplo, em uma descrição básica de casos de uso – ou, se os detalhes dessa alternativa são considerados requisitos importantes, podem ser separados em seções [Larman 1996].

A descrição das alternativas descritas dentro do fluxo de eventos tem como vantagem o fato dos participantes terem apenas um lugar para validar o caso de uso. Pode ser utilizado quando existem diversas alternativas pequenas

relativamente iguais em importância, ou quando a alternativa é tão pequena que não possui um fluxo que possa ser descrito.

À descrição textual do caso de uso pode ser adicionados elementos de lógica condicional diretamente ao fluxo de eventos. O refinamento da descrição do caso de uso identificando lógicas condicionais pode ser um indicativo de que o caso de uso deve ser dividido em mais casos de uso ou que relacionamentos de generalização tornam-se necessários.

O uso de seqüências alternativas e de lógica condicional depende do contexto: quando utilizar uma ou outra? Seqüências alternativas são mais apropriadas quando a alternativa é longa ou complexa; já o uso de lógica condicional é mais apropriado quando a variação é importante para o usuário. Um caso de uso com muitas alternativas pode indicar a necessidade de divisão do caso de uso.

Em determinados casos é necessário representar iterações dentro do fluxo de eventos. A descrição da iteração pode ser informal ou formal, sempre observando que a descrição do caso de uso não deve ser um pseudocódigo.

O grau de descrição de um caso de uso é sem dúvida um desafio. Uma descrição muito detalhada torna-se complexa e pouco compreensível, o que torna o esforço de validar os requisitos com o usuário de certa forma inútil. Da mesma forma, descrições superficiais não identificam corretamente os requisitos e alguns destes podem ser “assumidos”, o que acaba levando a surpresas no resultado final.

Um dos problemas de se utilizar fluxos alternativos é o fato que estes podem alterar a pós-condição do caso de uso, o que neste caso deve ser documentado com várias pós-condições.

O problema de se utilizar lógica condicional na descrição do caso de uso é que pode parecer como projeto. Certas decisões são de projeto e não podem ser realizadas na análise.

A lógica condicional e as iterações devem ser utilizadas para validação de requisitos com o usuário, e não para influenciar decisões de projeto. É comum encontrar lógica condicional ou iterativa formalmente, o que deve ser evitado pois o objetivo não é projetar ou codificar o sistema.

Uma forma de diminuir a complexidade das descrições de casos de uso que possuem muitas alternativas é a utilização de diagramas de atividade. Esse tipo de artefato oferece um meio visual de representação das alternativas, dependências, relacionamentos e paralelismo das atividades que compõe o fluxo de descrição de um caso de uso.

3.4 Descrição com diagramas de atividade

As descrições do caso de uso podem ser mapeadas para um diagrama de atividades, considerando um passo ou atividade do caso de uso como uma atividade do diagrama. A cada atividade é dado um nome para indicar o objetivo da mesma e sua correspondência com o caso de uso. Um exemplo é mostrado na figura 17.

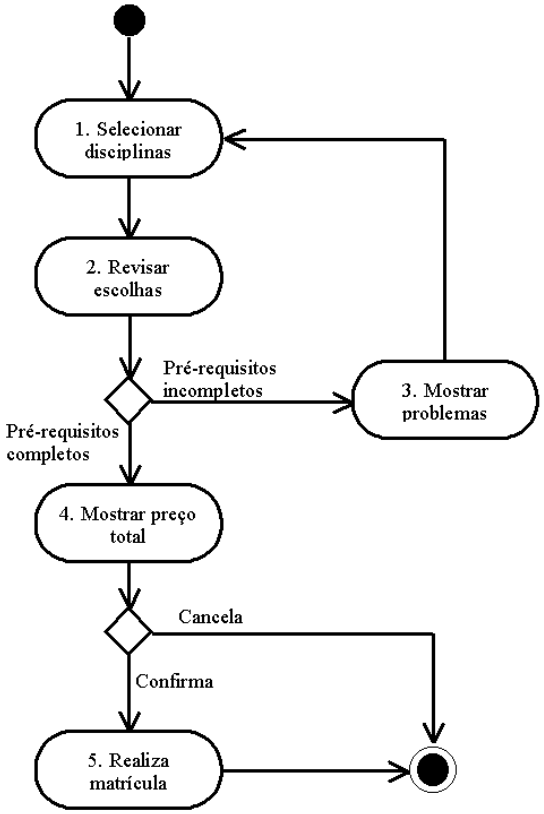


Figura 17 - Exemplo de descrição com diagrama de atividade

A descrição textual fornece um nível de detalhamento que os diagramas não podem fornecer.

Entretanto, os diagramas apresentam o fluxo de eventos de forma simples e clara. Podem ser utilizadas em conjunto, entretanto manter os dois artefatos sempre atualizados com os últimos detalhes pode se tornar um desafio.

Um diagrama de atividade é composto basicamente por estados de atividade, ou simplesmente atividade [Fowler 1999]. O diagrama de atividade descreve uma seqüência de atividades, com suporte à representação de elementos condicionais e de paralelismo.

As transições entre as atividades são representadas através de setas entre os elementos de representação das atividades – retângulos de borda arredondada.

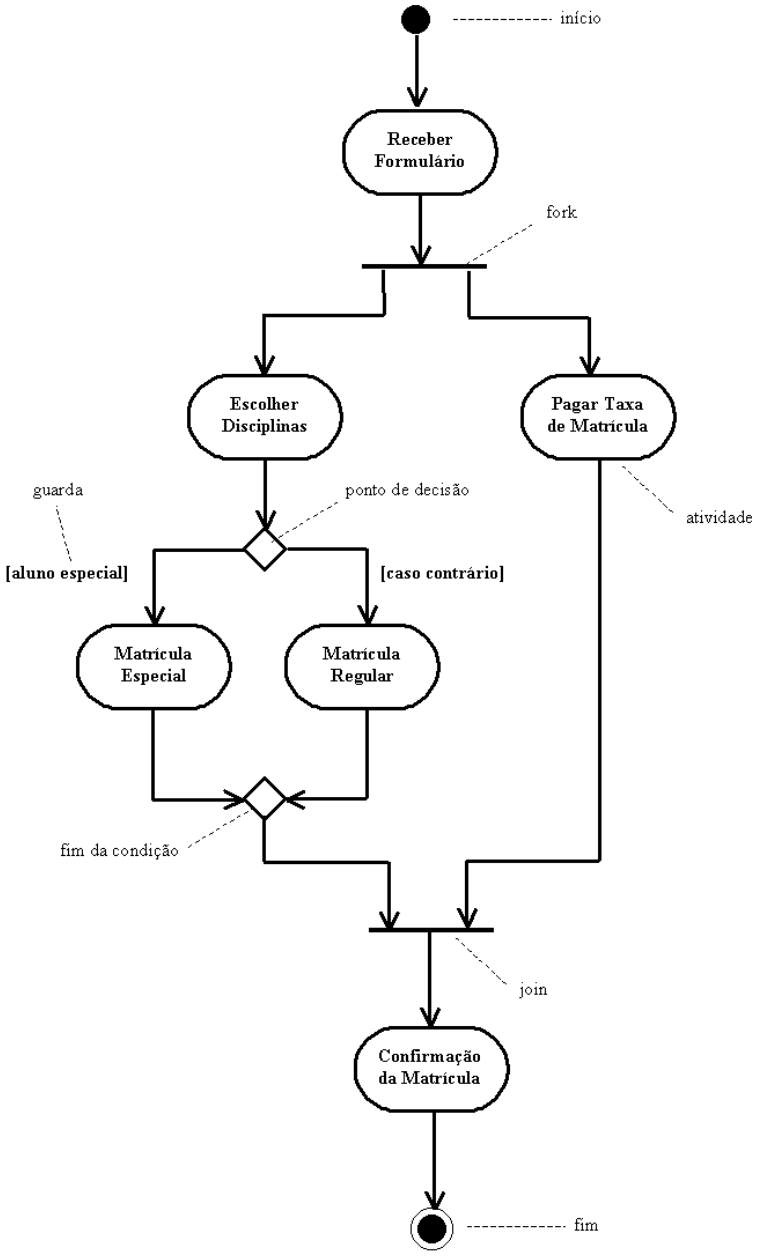


Figura 18 - Elementos do diagrama de atividade

O início de um diagrama de atividades é representado por um círculo preto, como mostrado na figura 18. Somente pode existir um ponto de início, enquanto que podem existir vários pontos de fim, representados por um círculo preto dentro de um outro círculo maior. Os pontos de decisão – condicionais – são representados por um losango em forma de diamante. Esse ponto de decisão

recebe apenas uma transição e pode se ramificar em tantas transições quanto necessárias. Cada uma das transições, saídas de um ponto de decisão, é decorada com uma guarda ou condição, que geralmente se representa entre colchetes. Um outro losango representa o fim do comportamento condicional, ou um ponto de finalização. A representação dos pontos de decisão e finalização não precisa explicitamente ser representada por losangos e pode estar diretamente associada a outras atividades, o seu uso é então opcional para tornar o diagrama mais claro.

O paralelismo de um diagrama de atividades é representado por uma barra, chamada de *fork* (ponto de paralelismo). Todas as transições saídas da barra representam transições que podem ocorrer em paralelo. O final do paralelismo é representado por outra barra chamada *join* (ponto de sincronismo).

Uma atividade pode ser decomposta em outras sub-atividades. Isso permite quebrar a complexidade de uma atividade em um outro diagrama, como mostrado na figura 19.

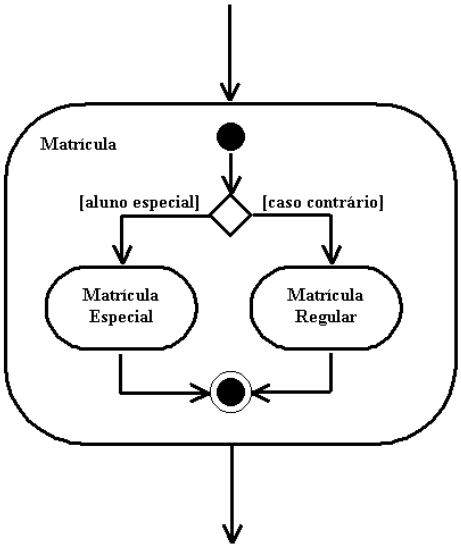


Figura 19 - Exemplo de sub-atividades

A utilização de diagramas de atividade é um exemplo de como a descrição de um caso de uso pode ser descrita de uma maneira mais formal. Outros diagramas podem ser utilizados em conjunto com os diagramas de atividade para representar melhor o caso de uso em nível de detalhe e precisão como, por exemplo, diagramas de colaboração e diagramas de estado [Hurlbut 1997].

3.5 Relacionamento extend, include e generalização

A descrição do caso de uso é influenciada pelo modelo de casos de uso e o relacionamento destes com outros casos de uso. Os relacionamentos do tipo

extend, *include* e generalização/especialização precisam ser representados de alguma forma nas descrições dos casos de uso.

Entender a forma como o fluxo de eventos ocorre em casos de uso que se relacionam entre si através destes relacionamentos é importante para a compreensão dos requisitos e de como o sistema será implementado.

3.5.1 Relacionamento "extend"

Um relacionamento de *extend* define que instâncias de um caso de uso podem ser complementadas com algumas funcionalidades adicionais definidas em um caso de uso estendido.

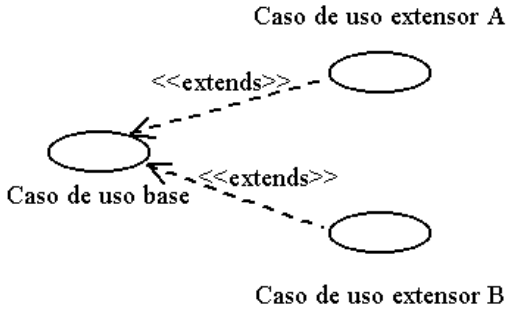


Figura 20 - Relacionamentos "extend"

No relacionamento de *extend* o caso de uso "extensor"[11] adiciona comportamento ao caso de uso base inserindo um ou mais segmentos de seqüência de comportamento dentro do fluxo de eventos do caso de uso base. Como mostrado na figura 20, os casos de uso que estendem funcionalidades apontam para o caso de uso base, que será estendido.

Basicamente, os relacionamentos de *extend* permitem que determinados comportamentos sejam expostos e destacados. Esses comportamentos podem ser adicionados a um caso de uso base. O relacionamento permite representar e estruturar o comportamento estendido de tal forma que essas extensões não se confundam ou tornem confuso o fluxo normal de eventos do caso de uso base. À medida que novas extensões são descobertas no sistema, elas podem ir sendo adicionadas sem causar distúrbios ao fluxo básico de eventos.

Em uma relação de extensão, um caso de uso pode ser estendido em determinados pontos do seu fluxo de eventos. Esses pontos são chamados de pontos de extensão ou *extension points*. Nesses pontos, sob certas condições os comportamentos estendidos são executados. O controle retorna ao fluxo de

[11] Do inglês, *extending use case*.

controle do caso de uso base no mesmo local onde o ponto de extensão ocorreu. Cada ponto de extensão deve ter um nome único dentro do caso de uso.

Opcionalmente, pode existir uma guarda condicional associada ao ponto de extensão. Se essa condição for satisfeita, o caso de uso extensor é executado; caso contrário o fluxo de eventos do caso de uso base segue normalmente e o comportamento do caso de uso extensor não é executado, como pode ser visto na figura 21.

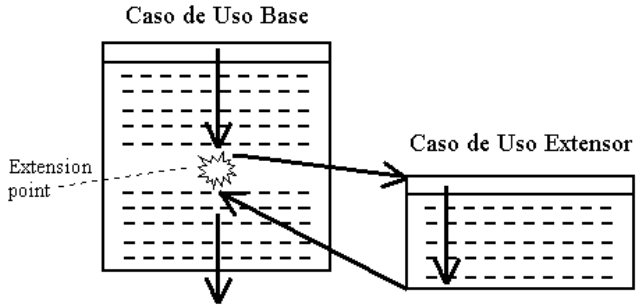


Figura 21 - Fluxo de eventos com extensão

Dentro de um caso de uso podem existir mais de um ponto de extensão, como mostra a figura 22. Entretanto o uso dessa técnica deve ser usado com cuidado, já que o uso de muitos pontos de extensão pode tornar o caso de uso difícil de entender.

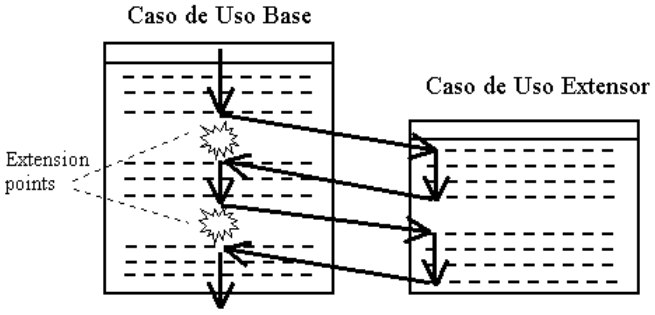


Figura 22 - Múltiplos pontos de extensão

Um exemplo seria um cliente realizando uma reserva de uma passagem de avião. O caso de uso “Fazer reserva de vôo” seria o caso de uso base. Um caso de uso extensor poderia ser um chamado “Fazer reserva de assento”. Nesse exemplo, poderiam existir dois pontos de extensão no caso de uso base, uma para efetuar a reserva do assento e outro para imprimir sua descrição.

3.5.2 Relacionamento “include”

À medida que o modelo de casos de uso cresce, é muito provável que comportamentos que aparecem em um caso de uso possam aparecer em outros. O relacionamento *include* permite a um caso de uso incluir dentro do seu fluxo de eventos comportamentos definidos em outro(s) caso(s) de uso. Esse tipo de relacionamento permite:

- Capturar comportamentos comuns entre os casos de uso;
- Diminuir a redundância e facilitar mudanças no modelo de casos de uso;
- Servir como entrada para a definição da arquitetura funcional através da identificação de aspectos comuns do sistema;
- Determinar e representar prioridades de desenvolvimento.

O relacionamento de *include* envolve dois casos de uso: o caso de uso incluído; e o caso de uso que inclui ou o utiliza. Casos de uso incluídos são casos de uso simples que são utilizados por outros.

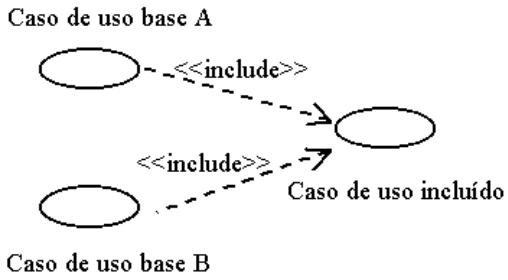


Figura 23 - Relacionamentos "include"

O relacionamento de *include* envolve dois casos de uso: o caso de uso incluído. O caso de uso incluído é um caso de uso simples que é referenciado por outros casos de uso. O relacionamento de “include” aponta para o caso de uso que está sendo incluído, o contrário do que ocorre com o relacionamento de “extend”, conforme a figura 23. O caso de uso incluído não sabe sobre os casos de uso base que o utilizam.

Os comportamentos definidos no caso de uso incluído são executados quando o ponto de inclusão é alcançado dentro do fluxo de eventos do caso de uso base, conforme mostrado na figura 24.

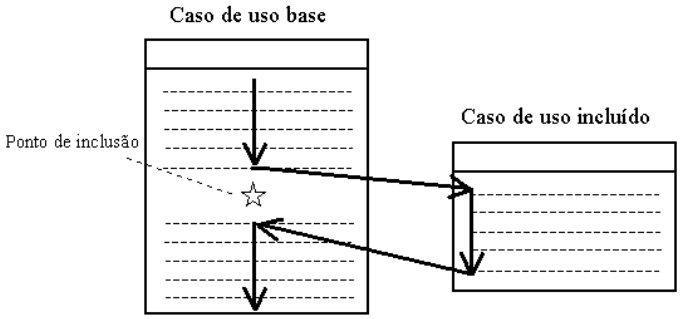


Figura 24 - Fluxo de eventos com relacionamento "include"

As principais diferenças entre o relacionamento de “include” e o relacionamento de “extend” são as seguintes:

- A dependência no relacionamento de “extend” é do caso de uso extensor para o caso de uso base. Isto significa que o caso de uso extensor depende do caso de uso base; e o caso de uso base não possui conhecimento da existência do caso de uso extensor;

- Um relacionamento de “include” possui dependência do caso de uso base para o caso de uso incluído. Isto é, o caso de uso que inclui precisa conhecer o caso de uso que será incluído;

- No relacionamento de “include” não existe nenhuma guarda condicional associada ao ponto de inclusão, diferentemente do que ocorre com os pontos de extensão;

- Um relacionamento de inclusão representa um único segmento encapsulado de comportamento que é executado até o fim quando chamado. Não existem múltiplos pontos de inclusão como ocorre com os pontos de extensão.

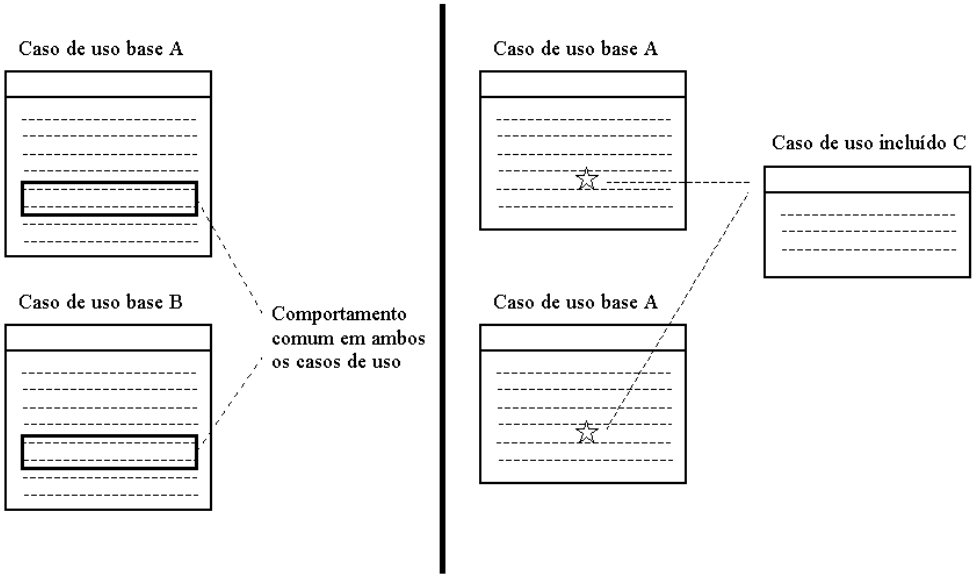


Figura 25 - Identificação de comportamento comum em casos de uso

Durante o processo de desenvolvimento do modelo de casos de uso, funcionalidades comuns são identificadas dentro de vários casos de uso. Essas funcionalidades podem ser extraídas para um caso de uso comum, que seria incluído dentro dos casos de uso base, conforme mostrado na figura 25.

Como os casos de uso que podem ser incluídos oferecem funcionalidades que podem ser utilizados por vários outros casos de uso, é interessante se manter uma relação desses casos de uso. Os responsáveis pela realização da análise devem saber as dependências entre os diversos casos de uso e seus relacionamentos. Armour e Miller [Armour 2000] propõem a criação ou geração de uma *tabela de relacionamentos de casos de uso incluídos*. Um exemplo de como essa tabela poderia ser construída é mostrada na tabela 4.

Tabela 4 - Exemplo de tabela de casos de uso incluídos

Número	Nome do caso de uso incluído	Descrição	Casos de uso que o utilizam
IUC27	Verificação de crédito	Realiza uma pesquisa sobre a condição de crédito do cliente	UC03 - Confirmação de pedido UC08 - Oferta de linha de crédito
IUC32	Calcular pontos de crédito	Calcula uma pontuação para avaliar a situação financeira de um indivíduo	UC03 - Confirmação de pedido UC08 - Oferta linha de crédito UC23 - Encerramento de conta
IUC41	Imprime comprovante	Impressão de comprovante fiscal com os itens da compra	UC03 - Confirmação de pedido UC12 - Geração de relatório
IUC43	Envia notificação	Envia confirmação do pedido por e-mail	UC34 – Confirmação on-line

3.5.3 Relacionamento de generalização

A generalização de casos de uso é uma relação entre um caso de uso pai[12] e o seu caso de uso filho[13]. Esse tipo de relacionamento implica que o caso de uso filho possua todos os atributos, seqüências de comportamento, pontos de extensão, além de participar de todos os relacionamentos, do caso de uso pai.

O caso de uso filho é uma especialização do caso de uso pai, podendo adicionar novas funcionalidades e sobrescrever funcionalidades existentes. A generalização representa elegantemente os relacionamentos de herança entre os casos de uso, mas a documentação de detalhes da generalização da descrição dos fluxos de eventos é um pouco complicada.

Em um fluxo de eventos de um caso de uso, um caso de uso filho pode inserir no meio do fluxo de eventos que herdou novos comportamentos, de forma que a ordem em que os eventos ocorrem é importante. Sem uma boa ferramenta CASE que suporte o gerenciamento desses aspectos, mantendo e documentando relacionamentos de generalização [Armour 2000]. Algumas questões devem ser levantadas: Como a lista de eventos do caso de uso filho é mostrada? Os eventos herdados são mostrados? Se não forem, como indicar onde as funcionalidades novas entram no fluxo de eventos? Se todo o fluxo de eventos é herdado, como identificar as novas funcionalidades?

3.6 Modelo de descrição

Um bom modelo de descrição ou *template* deve possuir um conjunto de seções que permitam documentar os principais aspectos da descrição de um caso de uso.

As seções de um modelo de descrição de caso de uso podem ser a seguintes [Ambler 2000]:

- **Nome:** O nome deve implicitamente expressar o propósito do caso de uso;
- **Identificador [opcional]:** Um identificador único como, por exemplo, UC 1024, que pode ser utilizado por outros artefatos do sistema;
- **Descrição:** Breve resumo do caso de uso;
- **Atores [opcional]:** Uma lista com os atores envolvidos no caso de uso. Embora essa informação esteja contida no próprio modelo de

[12] Do inglês, *parent use case.*
[13] Do inglês, *child use case.*

caso de uso, ajuda a aumentar a clareza da descrição, quando o modelo não estiver presente;

- **Status [opcional]:** Uma indicação do status do caso de uso, cujas opções poderiam ser, por exemplo, “em progresso”, “pronto para revisão”, “revisado” e “rejeitado”;

- **Freqüência [opcional]:** Quantas vezes o caso de uso é invocado pelo ator. É uma informação livre, que pode ser, por exemplo, “uma vez por mês”, “todo o dia” ou qualquer outra freqüência;

- **Pré-condições:** Uma lista de condições que deve ser atendida antes que o caso de uso seja invocado;

- **Pós-condições:** Uma lista de condições que deve ser mantida após a execução do caso de uso;

- **Caso de uso estendido [opcional]:** Referência ao caso de uso, se existir, que está sendo estendido ou especializado;

- **Casos de uso incluídos [opcional]:** Uma lista de casos de uso que são incluídos;

- **Considerações [opcional]:** Qualquer consideração importante sobre o domínio que foi levantada durante a concepção do caso de uso;

- **Fluxo de eventos:** O caminho principal e lógico do ator através do caso de uso. Também chamado de *fluxo principal* ou *fluxo feliz* já que é a descrição do comportamento do caso de uso quando tudo ocorre como planejado;

- **Fluxos alternativos:** Os caminhos não usuais que podem ser tomados no decorrer do caso de uso, que podem ser alternativas para se alcançar objetivo, ou tratamento de erros e exceções;

- **Histórico de alterações [opcional]:** Detalhes de quando o caso de uso foi modificado, o porquê e por quem;

- **Pontos [opcional]:** uma lista de pontos que são relevantes ao desenvolvimento do caso de uso;

- **Decisões:** Uma lista de decisões críticas relacionadas ao conteúdo do caso de uso. É importante armazenar essas decisões para manter uma memória do grupo.

Outras informações suplementares também podem ser encontradas em diversos modelos, como por exemplo, a prioridade do caso de uso, requisitos não funcionais e o detalhamento de regras de negócio e procedimentos.

Os requisitos não funcionais de um sistema não são representados dentro de um fluxo de eventos. Eles especificam as características globais do sistema, como por exemplo, portabilidade, disponibilidade, confiabilidade, segurança, capacidade, tempo de resposta, etc.

4 Descrição através de Diagramas de Atividade

Neste capítulo é proposto o método de descrição de casos de uso através de diagramas de atividades. Tal proposta constitui o foco principal deste trabalho e serve como meio de estruturar as descrições de casos de uso.

Como visto anteriormente, descrições de casos de uso complexas podem ser representadas através de diagramas de atividade. Esse tipo de técnica somente é utilizado quando é preciso descrever mecanismos de paralelismo, sincronização e elementos de lógica condicional. A descrição através de diagramas de atividade torna a descrição em um artefato mais formal e estruturado, porém é de difícil entendimento para os usuários e sua validação junto aos mesmos fica prejudicada.

Entretanto, uma ferramenta CASE pode extrair uma representação textual fácil de ser compreendia por usuários, e ainda manter a descrição na sua forma estruturada. A mesma ferramenta poderia utilizar a descrição formal para extrair outros artefatos úteis durante o processo de desenvolvimento, como por exemplo, casos de teste, casos de colaboração, entre outros. Esse mecanismo de transformação de um artefato em outro é mostrado na figura 26.

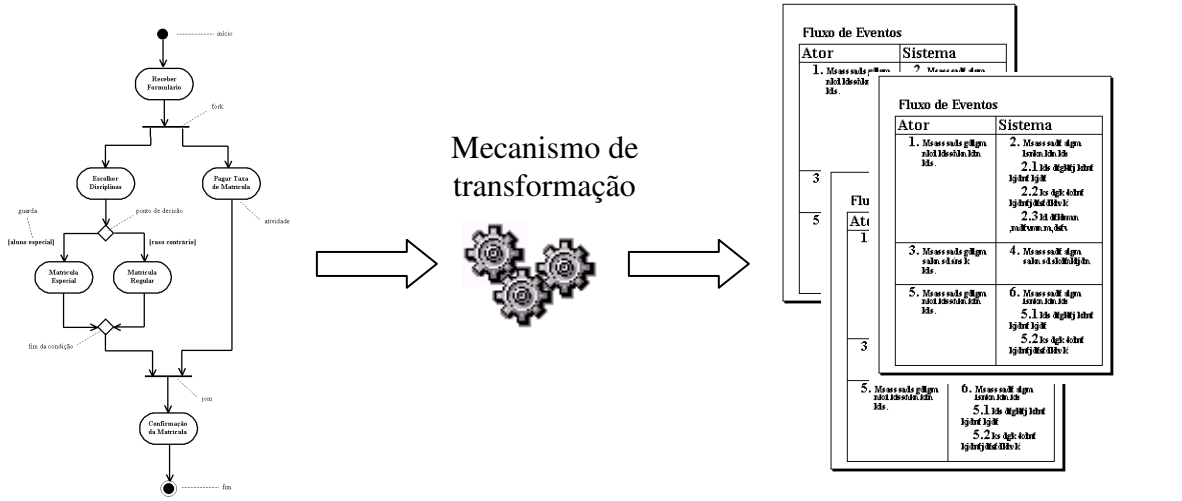


Figura 26 - Transformação de diagramas de atividade em outros artefatos

Uma ferramenta CASE interessante poderia ter formas que permitissem customizar os mecanismos de transformação para gerar artefatos específicos para cada organização. Tal ferramenta será apresentada no capítulo 8 como prova de conceito do método de descrição proposto neste trabalho.

A utilização de diagramas de atividade para a representação da descrição dos casos de uso precisa ser adaptada para suportar algumas informações que devem ser armazenadas. Através do mecanismo de extensibilidade da UML é possível representar diferentes tipos de atividade, como será mostrado a seguir.

Para poder descrever casos de uso através desta técnica, torna-se necessário detalhar a representação dos seguintes elementos:

- Fluxo de eventos principal
- Seqüências alternativas
- Cenários
- Casos de colaboração
- Relacionamento “include”
- Relacionamento “extend”
- Relacionamento de generalização

4.1 Porque diagrama de atividades?

Diversas notações podem ser utilizadas para a representação de descrições de casos de uso como, por exemplo, redes de Petri, linguagens formais, pseudo-código, entre outros [25].

O trabalho de doutorado de Russel Hurlbut [Hurlbut 1997b] descreve diversos tipos de formalismos utilizados na descrição e formalização de casos de uso. Hurlbut faz uma comparação entre os formalismos levando em consideração a representação textual, gráfica e dinâmica, além de identificar o foco de cada método de descrição. Os formalismos e formatos de descrição estudados por Hurlbut são os seguintes:

- Abstract Business Processes
- Adaptive Use Cases
- Agent Based Use Cases
- ALBERT
- Analysis Patterns
- Animating Use Cases
- Archetypes, Deltas, and Frames
- Baseline Requirements Document
- Behavioral Lifecycle Modeling
- Business Rule Atomic Elements
- Business Rules Model (GUIDE)
- Business Rules Model Based (ECA)
- Change Cases
- Collaboration Based Design

- Collaboration Frameworks
- Collective Behavior/Business Rules
- Commitment-based software
- Component Contracts
- Design Patterns
- Directed Use Case Graphs
- Distributing Requirements/Episodes
- Elicitation of State Machines
- Enhanced Scenarios
- Exception Handling
- Formal Verification of Use Cases
- Formalized Use Cases/State Charts
- Hierarchical Policy Model
- Hypertext Use Cases
- Incremental Evolution of Specs
- Information Mapping
- Interactions/Hypermedia/Visuals
- Iterative Use Case Prototyping
- Key Event Dictionary
- Message Sequence Charts
- Modeling Large Business System
- OO Design from Functional Specs
- Organization Knowledge
- Path Expression Groups
- Problem Frames
- Requirements Definition/Verification
- Requirements Scripts
- Scenario Contexts
- Scenario Strategies
- Scenario Trees
- Service Usage Models
- Speech Acts
- Structuring Use Cases With Goals
- Synthesized Use Case Modeling
- Task Scripts
- Transaction Based Analysis
- Transaction, Event and Rule Models
- Use Case Classes
- Use Case Components
- Use Case Dialog Maps
- Use Case Formats
- Use Case Maps
- Use Case Reuse
- Validate Component Use Cases

- Workflow Management System

Diversos formalismos podem ser utilizados para descrever o comportamento de caso de usos, entre eles diagramas da própria UML. A representação da própria descrição do caso de uso através da UML pode ser realizada através dos diagramas de atividades, diagramas de seqüência e diagramas de estado.

Os diagramas de atividades se apresentam como uma notação mais clara para a representação de paralelismos, elementos de lógica condicional e sincronização do que os diagramas de seqüência e diagramas de estado [Armour 2000], além de possuírem a característica de decomposição hierárquica, provida através do uso de sub-diagramas de atividade.

Os diagramas de atividade possuem propriedades que permitem a descrição de Workflows [Dumas 2001], o que representa uma característica importante dentro do conjunto de diagramas da UML para a descrição de casos de uso.

Além de permitirem a representação de forma apropriada dos fluxos necessarios que compõe a descrição de um caso de uso, os diagramas de atividade possuem características que permitem a formalização desses fluxos através de semânticas formais [Eshuis 2002].

Outros diagramas da UML também possuem características que permitem com que se derivem modelos formais, como por exemplo diagramas de seqüência e diagramas de estado [Bernardi 2002], porém os diagramas de atividade representam os diferentes fluxos presentes nas descrições de casos de uso de forma mais clara e direta.

Os diagramas de atividades podem ser verificados através de ferramentas de verificação formal [Eshuis 2004]. Através destas validações, descrições de casos de uso estruturadas utilizando diagramas de atividades podem ser também validadas formalmente.

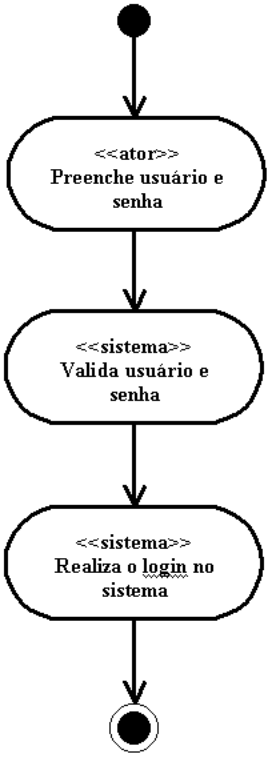
A geração de casos de teste a partir de fluxos definidos através de diagramas de atividades [Linzhang 2004] pode ser realizada através de processos automáticos que percorram a árvore de caminhos possíveis definidos através do fluxo do diagrama. Neste aspecto, a utilização de diagramas de atividades para a descrição de comportamentos de casos de uso provê a estruturação adequada para fins de geração de casos de teste.

4.2 Representação de um fluxo de eventos principal

O fluxo de eventos de uma descrição de caso de uso representa a interação entro o ator e o sistema. Essa interação se dá através de eventos numerados representados textualmente através de um fluxo contínuo ou separado por colunas[14].

Cada evento é realizado ou pelo ator ou pelo sistema, e representam uma interação do tipo ação-reação. A representação através de atividades dentro de um diagrama de atividades pode ser realizada através da utilização dos estereótipos <<ator>> e <<sistema>>. Um exemplo de representação de uma descrição de um caso de uso simples de Login é mostrado na tabela 6.

Tabela 5 - Caso de uso simples representado por diagrama de atividade

<table>
<tr><th>Descrição textual</th><th>Descrição com diagrama de atividade</th></tr>
<tr>
<td>
<table>
<tr><th>Ator</th><th>Sistema</th></tr>
<tr><td>1. Preenche o usuário e a senha.</td><td></td></tr>
<tr><td></td><td>2. Valida o usuário e a senha.</td></tr>
<tr><td></td><td>3. Realiza o login no sistema.</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

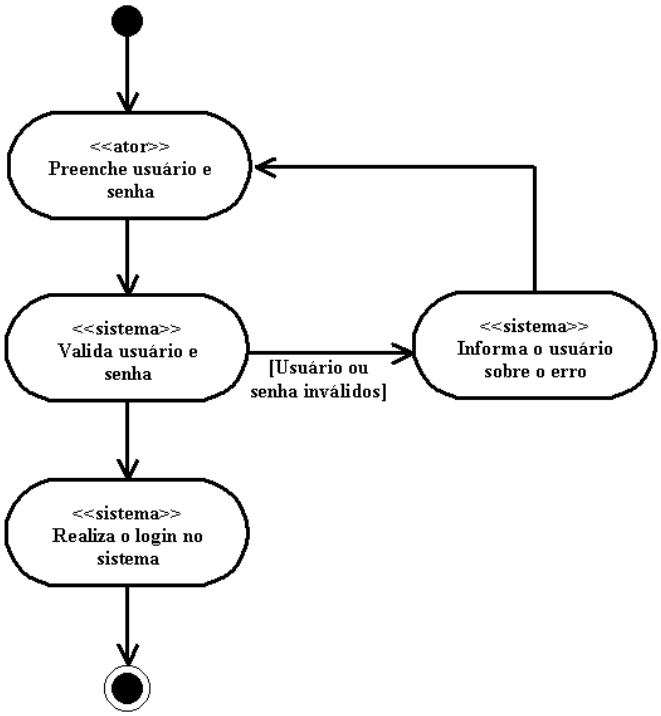
4.3 Representação de seqüências alternativas

A descrição dos casos de uso permite a introdução de seqüências alternativas, que representam desvios no fluxo principal. Essas seqüências alternativas podem ser utilizadas para realizar o tratamento de exceções ou indicar um outro caminho possível. No exemplo anteriormente apresentado, o usuário pode ter informado o usuário ou senha inválida, nesse caso o sistema deve

[14] Ver página 28.

mostrar uma mensagem alertando sobre o ocorrido e voltar a requisitar os dados de login, como mostrado na tabela 7.

Tabela 6 - Exemplo de caso de uso com seqüência alternativa

<table>
<tr><th>Descrição textual</th><th>Descrição com diagrama de atividade</th></tr>
<tr>
<td>
<table>
<tr><th>Ator</th><th>Sistema</th></tr>
<tr><td>1. Preenche o usuário e a senha.</td><td></td></tr>
<tr><td></td><td>2. Valida o usuário e a senha.</td></tr>
<tr><td></td><td>3. Realiza o login do usuário.</td></tr>
<tr><td colspan="2">Alternativa 1: O usuário e senha estão inválidos. Informa o usuário sobre o erro e requisita novamente as informações de login.</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

Uma seqüência alternativa possui uma condição lógica para acontecer. No exemplo, essa condição é “Usuário ou senha inválidos”. Essa condição lógica pode ser indicada através de uma guarda[15] na transição entre as atividades envolvidas.

Entretanto, mesmo com poucas alternativas – e ainda mais com muitas – uma descrição de casos de uso utilizando um diagrama de atividade pode ser tornar confuso. Como identificar o caminho principal? Se existir mais de um caminho levando ao fim, qualquer caminho que leve ao fim do caso de uso pode ser o caminho principal.

Esse problema pode ser resolvido através do uso de **swim lanes**[16] ou partições. Pode-se estabelecer uma swim lane principal onde o fluxo normal das atividades pode acontecer. As atividades dispostas em outras regiões podem ser

[15] Uma guarda é uma expressão condicional colocada entre colchetes e indicam que a mensagem ou transição somente ocorre se a guarda for verdadeira [Fowler 1999].
[16] *Swim lane* ou “raia de natação” é uma divisão uni-dimensional, que permite particionar um diagrama de atividades [Fowler 1999].

consideradas seqüências que não fazem parte do fluxo principal e, portanto representando seqüências alternativas, como mostrado na figura 27.

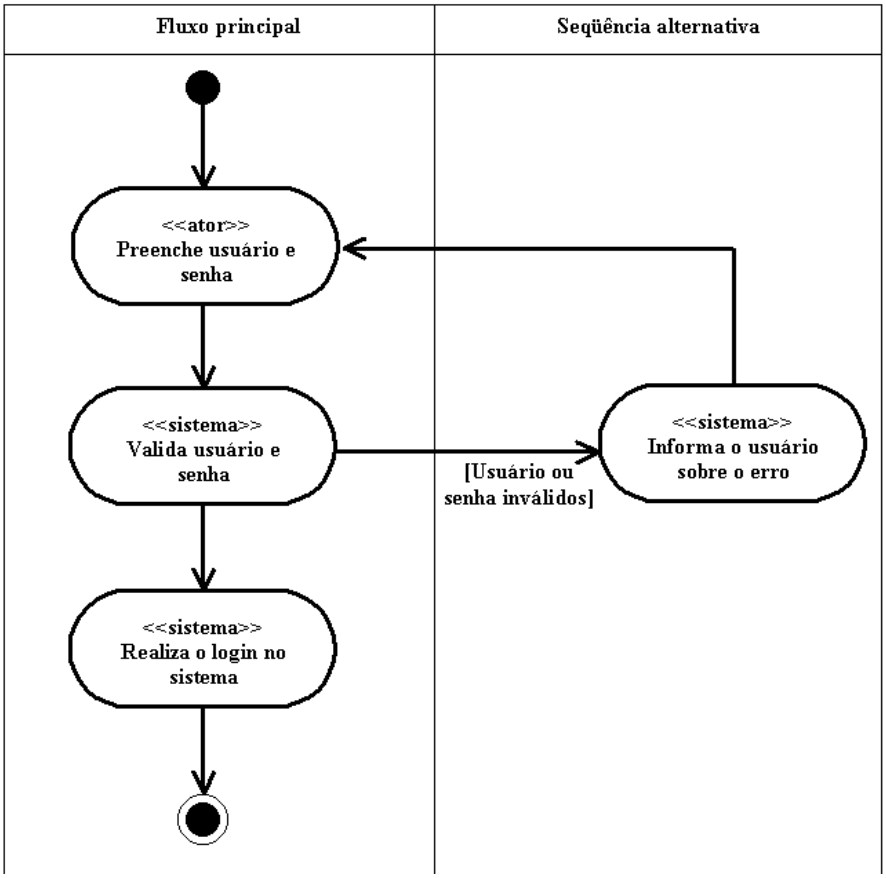


Figura 27 - Uso de swim lanes para identificar o fluxo principal

4.4 Representação de cenários

"Um cenário é um conjunto de passos descrevendo a ação entre o ator e o sistema. Um caso de uso é um conjunto de cenários unidos por um objetivo comum" [Fowler 1999].

De certa forma, até agora o que foi visto foram descrições sobre um único cenário, ou seja, o cenário principal. Um exemplo do uso de vários cenários poderia ser um caso de uso de compra, onde o cliente poderia realizar o pagamento com dinheiro, cartão de débito ou cartão de crédito. Caso o pagamento seja feito com cartão de débito, o sistema deve entrar em contato com a instituição financeira responsável, e assim por diante. Todos os cenários levam a um único objetivo, que é a realização da compra.

A representação de cenários pode ser representada em um único diagrama de atividades, e ser especificamente divididos dentro do diagrama através de swim

lanes. A figura 28 mostra o exemplo de um caso de uso com três cenários distintos.

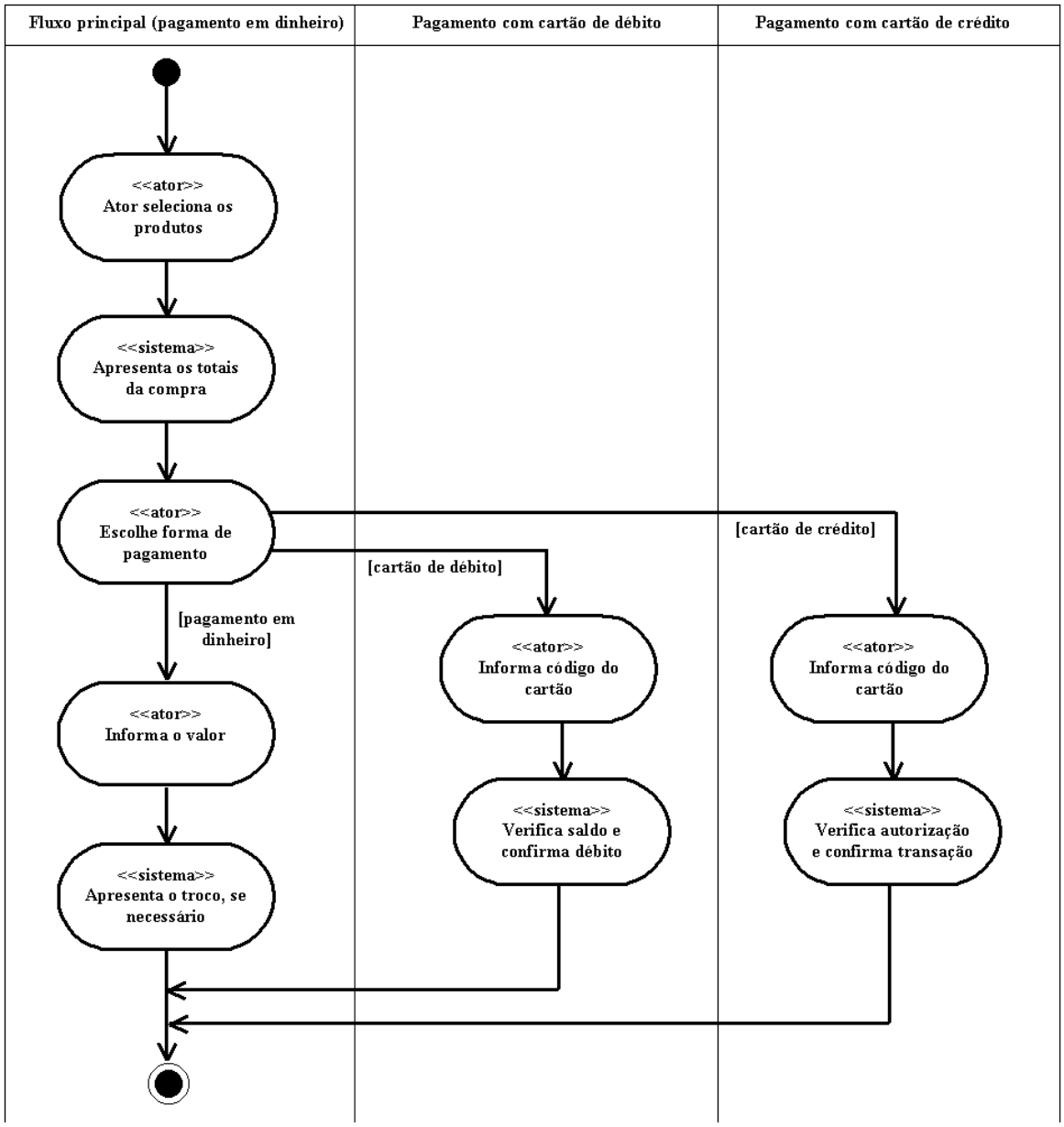


Figura 28 - Representação de cenários através de swim lanes

Embora seja uma forma simples e fácil de representar cenários, o uso de swim lanes para tal representação tem os seus problemas. O diagrama pode ficar muito poluído à medida que a descrição é incrementada. A representação de seqüências alternativas fica prejudicada e o seu uso combinado com a separação de cenários torna o diagrama difícil de ser mantido e compreendido.

A especificação de UML 1.5 possui o conceito de sub-diagramas de atividade. Essa característica permite que uma atividade possua um diagrama de

atividade associado dentro dele. A notação é um desenho de uma atividade comum decorada pelo símbolo de um mini sub diagrama. Utilizando o conceito de sub-diagramas, é possível representar os cenários conforme mostrado na figura 29. Nesse caso, cada cenário é uma atividade que pode ser expandida em outro diagrama.

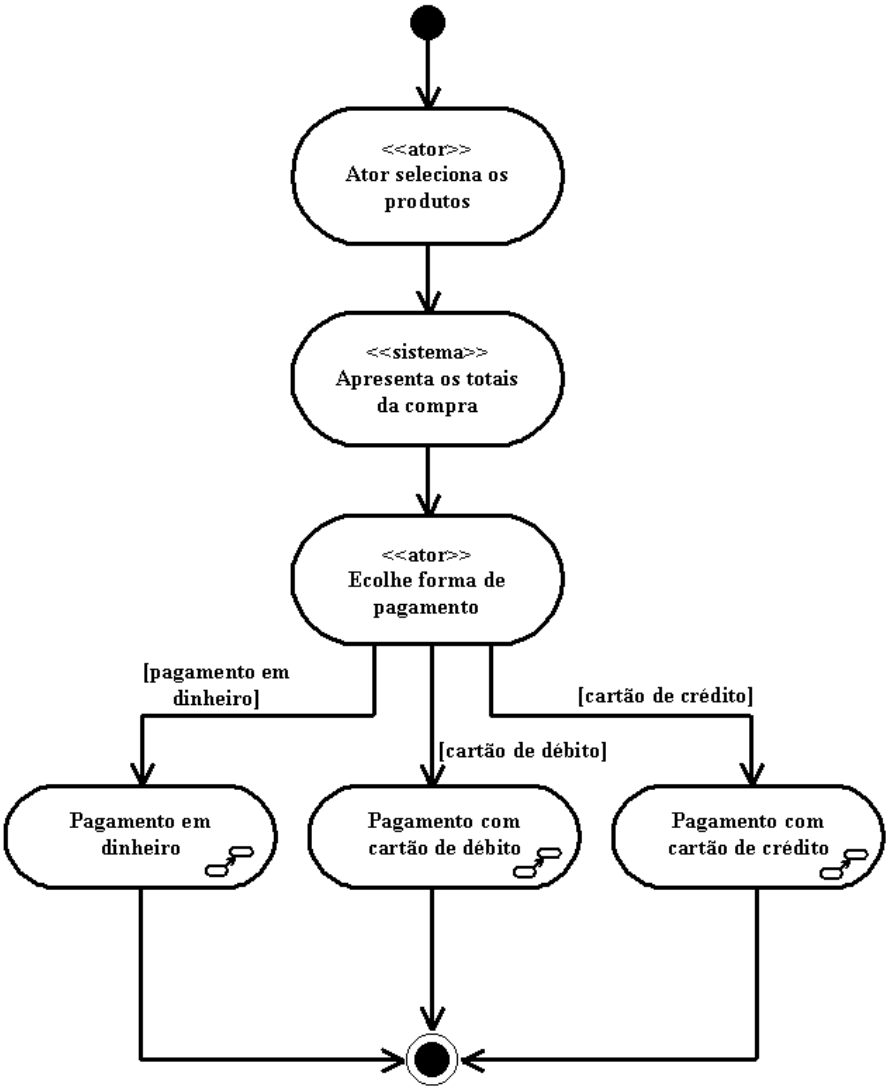


Figura 29 - Representação de cenários através de sub-atividades

Dentro de uma ferramenta CASE que suporte esse tipo de elemento de sub-atividade pode existir um mecanismo que permite facilmente navegar dentro do caso de uso, explodindo os sub-diagramas aninhados dentro desse tipo de elemento.

Um exemplo de como essa navegação é realizada é mostrada na figura 30. Clicando no elemento que representa uma atividade aninhada é possível abrir o

diagrama de atividade que representa o fluxo de eventos correspondente àquela ação.

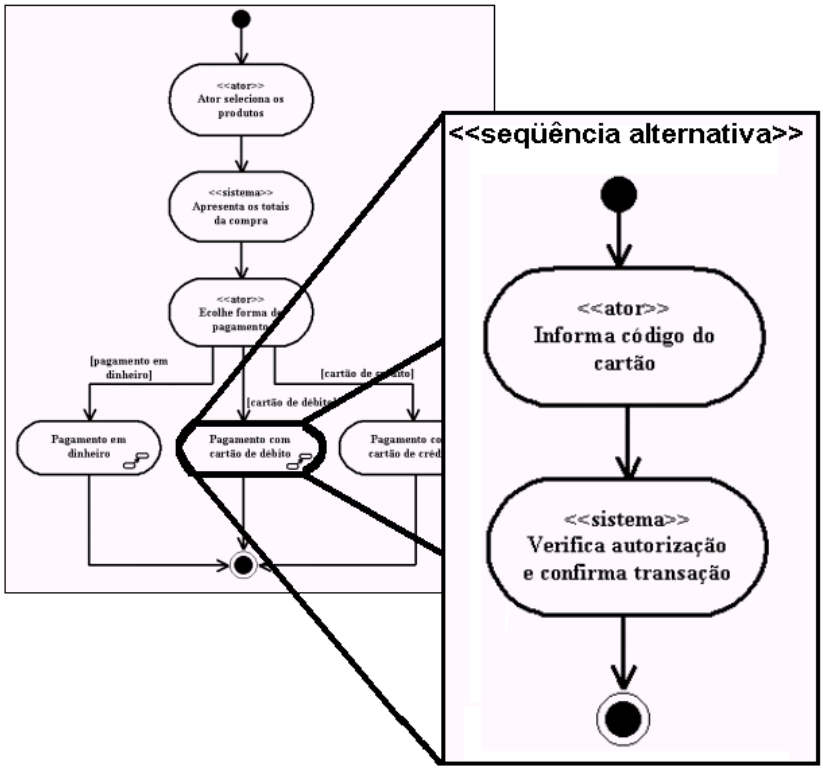


Figura 30 - Exemplo de navegação entre os diagramas de atividade

O diagrama de atividade é estereotipado como <<seqüência alternativa>>, de forma a identificar a sua intenção. Isto torna-se necessário pois a característica de diagrama de atividade aninhado pode ser utilizado para a representação de outras informações como, por exemplo, a representação de casos de colaboração, mostrada a seguir.

4.5 Representação de casos de colaboração

Os casos de colaboração são casos de uso estereotipados que descrevem o funcionamento interno do sistema [Mattingly 1998]. Eles servem para detalhar as ações do sistema uma vez que a interface com o usuário está bem definida. Os casos de colaboração podem ser representados através dos elementos de sub-diagramas, de forma semelhante à representação de seqüências alternativas, como mostrado anteriormente.

Os casos de colaboração tornam-se interessantes com esta forma de representação, pois permitem ir aprofundando as características do sistema. O caso de uso mais abstrato ou de maior nível pode definir apenas o comportamento geral do caso de uso e ser utilizado como forma de validação com o usuário final.

Durante o processo de desenvolvimento do sistema, durante a especificação mais real dos requisitos, o caso de uso pode ir sendo refinado através de um maior detalhamento das atividades do sistema através do uso de sub-diagramas.

O nível de detalhamento pode ir aumentando à medida que novos níveis de descrição podem ir sendo incorporados. Um exemplo de como os casos de colaboração podem ser utilizados como diagramas de atividades estereotipados é mostrado na figura 31.

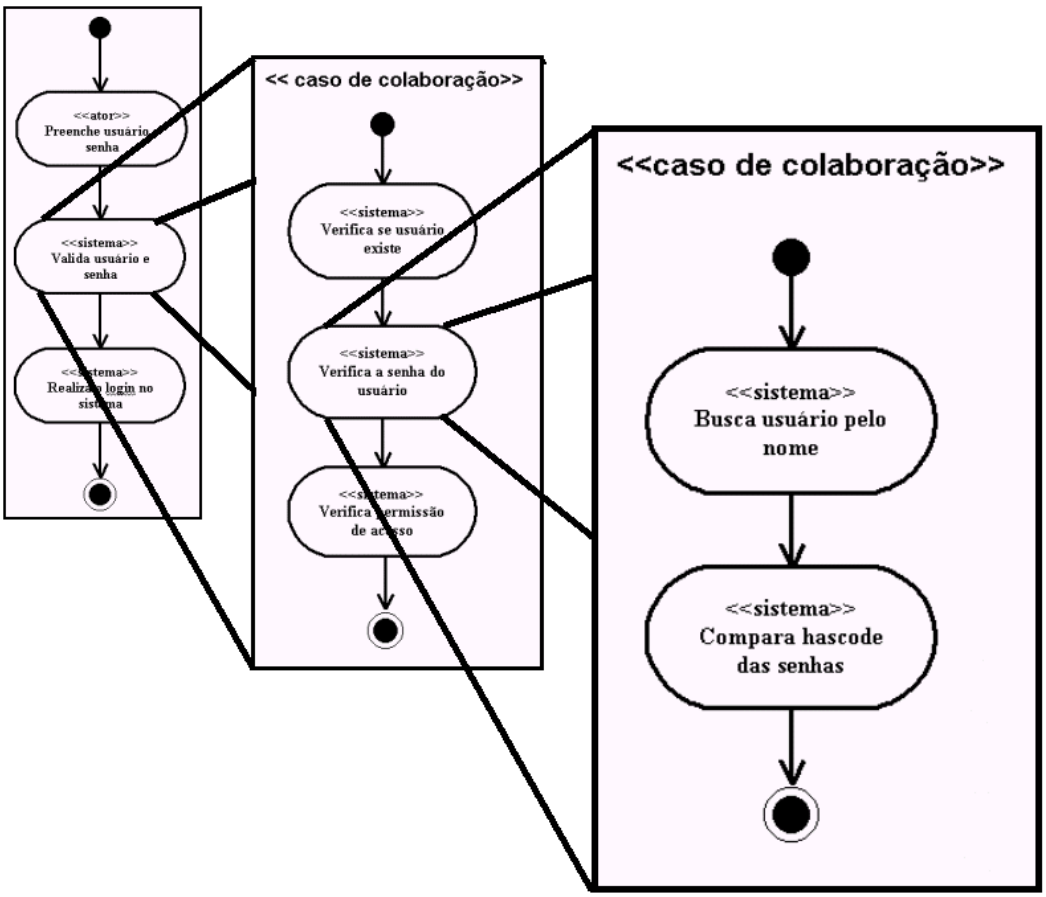


Figura 31 - Exemplo de representação de casos de colaboração

A navegação de forma hierárquica permite que novos casos de colaboração sejam incorporados à descrição inicial do caso de uso. A validação dos requisitos com o usuário pode ser feita através da descrição mais alto-nível (primeiro diagrama na hierarquia), enquanto que os outros níveis serviriam de apoio às outras etapas envolvidas no ciclo de desenvolvimento do sistema.

4.6 Representação do relacionamento "include"

O relacionamento de include é realizado entre dois casos de uso, sendo o caso de uso base aquele que referencia o caso de uso incluído. Na descrição do caso de uso base que deve existir uma referência do ponto onde o outro caso de uso é incluído. Esse ponto é chamado de ponto de inclusão.

Dentro da representação da descrição com diagramas de atividade, é possível definir um estereotipo chamado <<ponto de inclusão>>. Atividades com tal estereótipo devem ter uma referência ao caso de uso ao qual apontam, como mostra a figura 32.

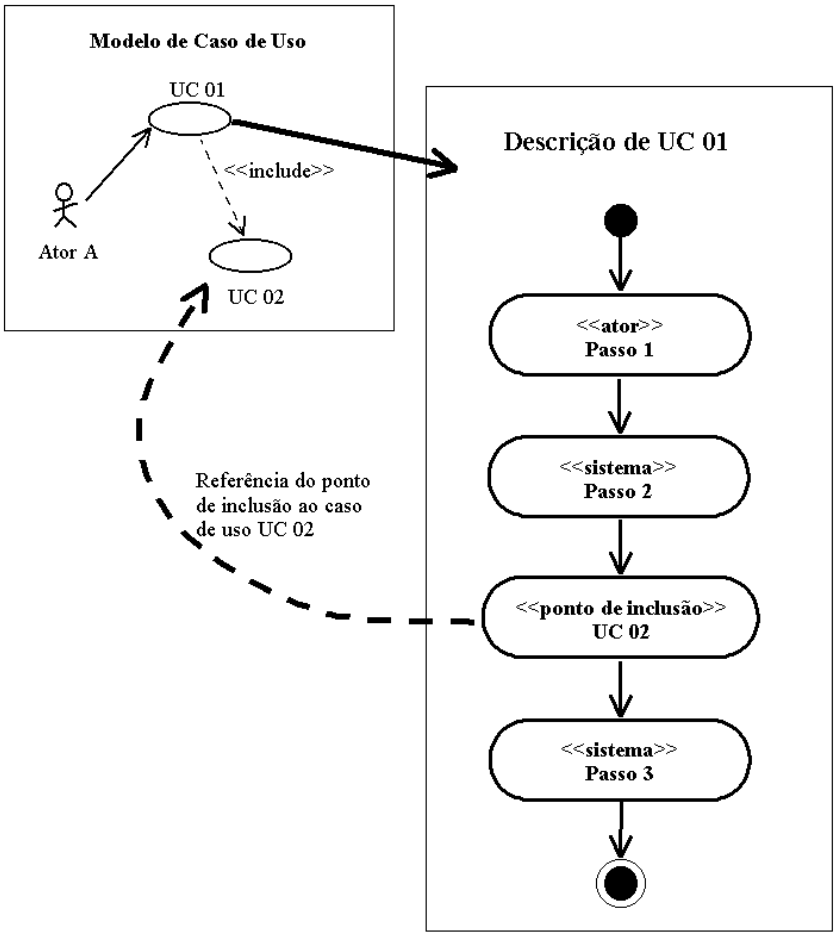


Figura 32 - Representação do relacionamento "include"

Uma ferramenta CASE poderia abrir uma janela de dialogo para que o analista que está definindo a descrição do caso de uso pudesse indicar qual o caso de uso que ele deseja incluir. É possível através desse tipo de representação garantir que o modelo de casos de uso esteja de acordo com as descrições dentro dos casos de uso. A ferramenta poderia indicar os relacionamentos do tipo “include” mal formados, ou seja, aqueles que não possuem pontos de inclusão definidos na origem do relacionamento.

4.7 Representação do relacionamento "extend"

O relacionamento do tipo extend também envolve dois casos de uso. Porém o relacionamento possui um sentido contrário. O caso de uso que é extendido não possui referência a nenhum caso de uso. Ele apenas define pontos de extensão, que serão estendidos pelo caso de uso extensor. O caso de uso extensor deve referenciar o caso de uso que ele quer estender e indicar quais os pontos de extensão que ele estende.

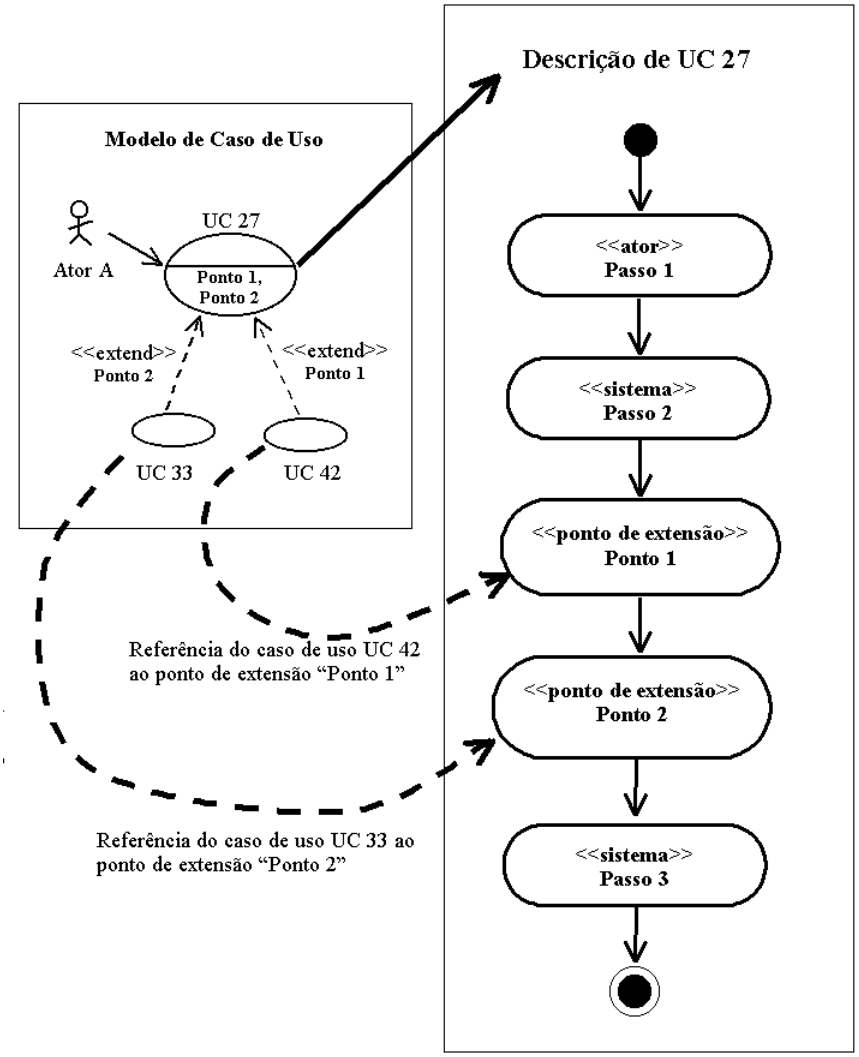


Figura 33 - Representação do relacionamento de "extend"

Dentro da representação com diagramas de atividade, o ponto de extensão é definido através de uma atividade com o estereótipo <<ponto de extensão>>, como pode ser visto na figura 33.

4.8 Representação da generalização

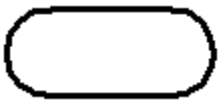
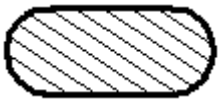
A generalização dos casos de uso é uma característica pouco usada por sua complexidade e dificuldade de manutenção. Isso ocorre atualmente porque as descrições dos casos de uso são em linguagem natural e o controle do fluxo de eventos herdado é complicado. Entretanto uma ferramenta CASE que auxiliasse os analistas a compreender o que ocorre com a descrição dos casos de uso quando a herança está presente seria de grande valor.

Armour e Miller [Armour 2000] sugerem marcar os passos do fluxo de eventos com três tipos de marcações:

- Passo herdado e não alterado;
- Passo herdado e alterado (especializado);
- Passo novo.

É possível representar essas marcações para os diagramas de atividade que descrevem os casos de uso através de uma simbologia gráfica[17], como mostrado na tabela 8.

Tabela 7 - Simbologia necessária para a representação de herança

Símbolo	Significado
	É o símbolo normal de atividade, entretanto, dentro da descrição de um caso de uso que herdou uma descrição, ele significa um novo passo, ou seja, um passo do caso de uso corrente.
	O símbolo de atividade preenchido com a cor cinza indica que o passo é herdado e não foi alterado. Se a descrição não for alterada, todos os passos estão neste estado.
	O símbolo de atividade hachurado significa que o passo foi herdado e de alguma forma modificado (por exemplo, a sua descrição ou os seus sub-diagramas).

Os passos herdados podem ser modificados e outros passos novos podem ser acrescentados. Os passos herdados não podem ser excluídos, para evitar o risco da descrição perder o sentido. Se existe a necessidade de se excluir o caso de uso ou realizar alterações muito significativas, talvez seja o caso de não utilizar a generalização, e sim escolher alguma outra técnica como, por exemplo, o uso de relacionamentos de include ou extend.

[17] Os estereótipos não podem ser utilizados para realizar a marcação dos passos do fluxo de eventos pois eles são utilizados para determinar outro tipo de informação.

Na figura 34 pode ser visto um exemplo de como ficaria o fluxo de eventos representado por diagramas de atividade no caso da generalização.

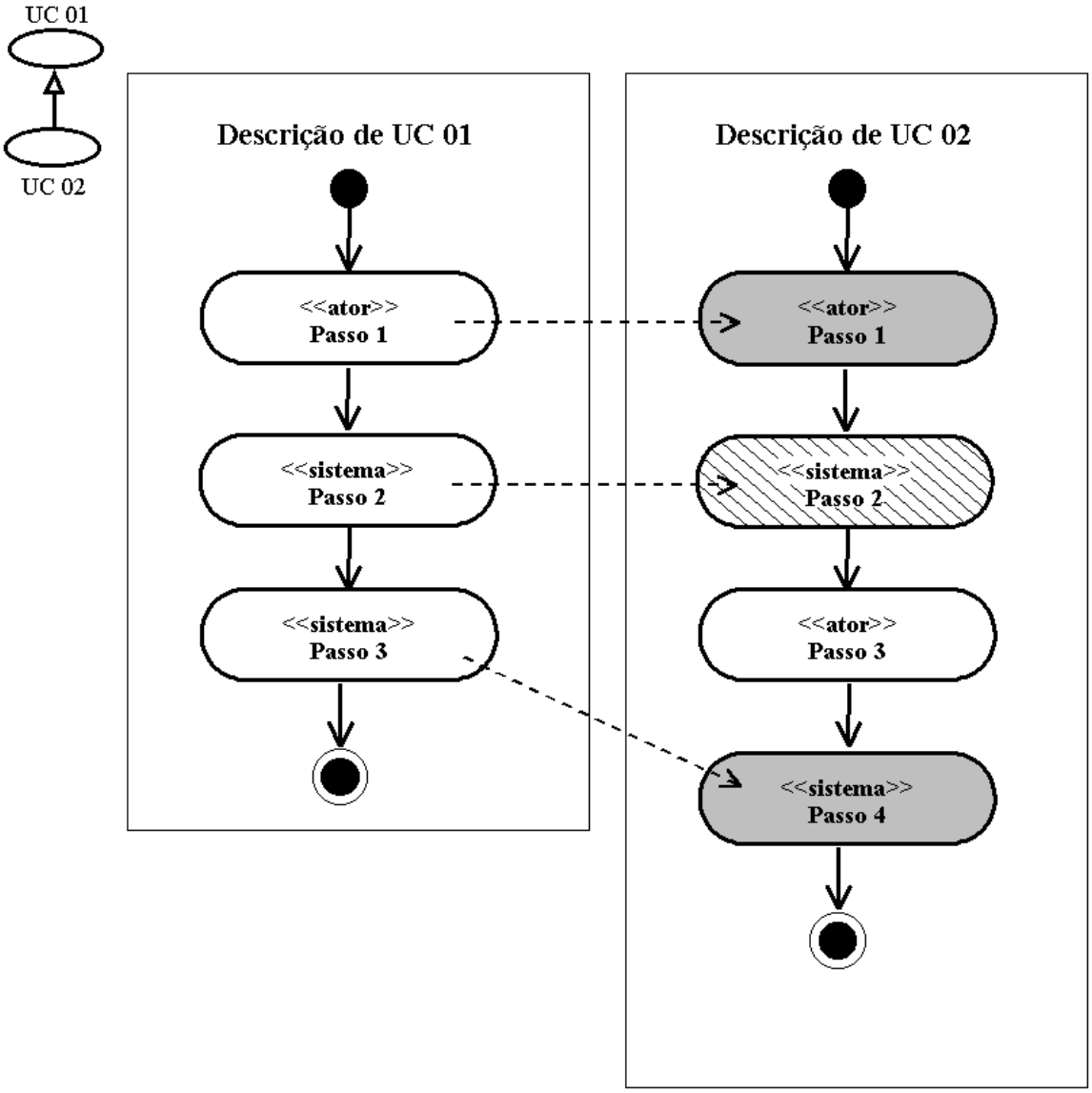


Figura 34 - Representação de herança com diagramas de atividade

No exemplo apresentado acima o segundo caso de uso (UC 02) herda o fluxo de eventos do caso de uso UC 01. O passo 2 é alterado no novo caso de uso e um novo passo é acrescentado em seguida. Os demais passos não são alterados. Note-se que o passo 3 do UC 01 foi renumerado para ser o passo 4 no caso de uso especializado. Essa alteração foi necessária, pois outro passo foi inserido. Uma ferramenta CASE de descrição com diagramas de atividade deveria suportar esse tipo de característica automaticamente.

5 Relação com a arquitetura do sistema

Este capítulo tem como objetivo mostrar como a descrição de casos de uso proposta através de diagramas de atividade pode ser relacionado com o modelo estático do sistema, proporcionando uma amarração entre a descrição e a realização - o que proporciona uma desejada rastreabilidade entre os requisitos descritos e os elementos que implementam o sistema (código, sub-sistemas, etc).

A definição da arquitetura ocorre já nas primeiras fases do ciclo de desenvolvimento. Os casos de uso iniciais ajudam a definir o que seria uma arquitetura adequada para o sistema, o suficiente para poder colocar um protótipo de arquitetura a suportar os casos de uso das primeiras iterações, dentro de um processo de desenvolvimento iterativo.

A arquitetura, uma vez definida, começa a influenciar os casos de uso, que também por sua vez são capazes de fazer amadurecer a base arquitetônica do sistema.

Entretanto, muitas vezes a arquitetura é definida sem levar em consideração alguns aspectos importantes que são identificados pelos casos de uso. Isso porque as descrições dos casos de uso são pouco utilizadas nas etapas posteriores do desenvolvimento do sistema que se seguem à analise de requisitos. Basicamente, as descrições dos casos de uso são utilizadas no inicio do ciclo e posteriormente no fim, durante a validação daquilo que foi construído.

Apesar do papel da análise não ser a de se intrometer em aspectos de projeto do sistema, é através dela que todo o resto é disparado. Os elementos que disparam o resto do processo são as descrições dos casos de uso.

Através da notação proposta para a descrição de casos de uso é possível criar diferentes níveis de abstração, de forma que no nível mais abstrato, a descrição do caso de uso seja aquela definida pelo analista.

5.1 Refinamento das descrições dos casos de uso

É possível que a mesma técnica de descrição possa ser utilizada para refinar cada vez mais os casos de uso. Os casos de colaboração[18] são um exemplo de como uma atividade de sistema pode ser explorada em maior detalhe. Embora os objetivos dos casos de colaboração estejam dentro do escopo da análise de sistemas, é possível entendê-los como refinamentos que, à medida que

[18] Ver páginas 29 e 52.

vão se especializando cada vez mais, entram no universo do projeto de sistemas[19].

À medida que as descrições dos casos de uso vão sendo especializadas, através da descrição de cada ação de sistema, a necessidade de se representar aspectos arquitetônicos do sistema se torna interessante.

O fato de cada ação (ou atividade) poder ser descrita em sub-diagramas de atividade permite que arquitetura possa ser referenciada tanto nos primeiros níveis hierárquicos da descrição, como nos últimos.

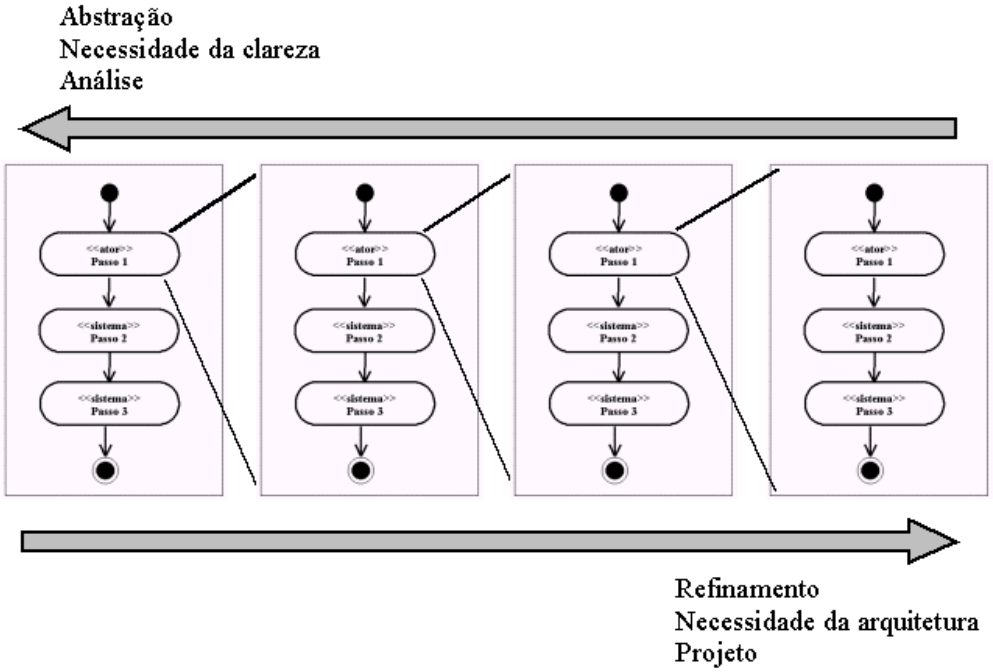


Figura 35 - Fluxo de refinamento das descrições dos casos de uso

Um exemplo de como é o fluxo de refinamento da descrição é mostrado na figura 35. Os níveis mais abstratos são os mais próximos da análise e, portanto dos usuários, o que exige clareza. À medida que a descrição vai sendo refinada ela cada vez mais se aproxima do projeto e necessita saber sobre a arquitetura; muitas vezes é necessário identificar elementos da arquitetura – como, por exemplo, subsistemas ou camadas que compõe o sistema.

[19] Existe uma linha tênue entre a análise e projeto de sistemas. Essa divisão é um tema polêmico e não faz parte do escopo deste trabalho.

5.2 Identificação de subsistemas ou camadas

Dentro da UML existem vários diagramas que podem ser utilizados para a descrição da arquitetura. Não faz parte do escopo deste trabalho identificar a melhor forma de descrição da arquitetura. Entretanto, na descrição dos casos de uso, especificamente nos níveis hierárquicos mais baixos da descrição – ou seja, aqueles mais refinados – torna-se necessária a identificação de elementos da arquitetura.

Para a descrição da interação entre objetos existem diversos outros diagramas que são mais adequados do que os diagramas de atividade como, por exemplo, os diagramas de seqüência ou de colaboração. Entretanto, a técnica de descrição do fluxo de eventos através de diagramas de atividade permite descrever de forma muito interessante as ações que o sistema de realizar.

Sobre esse aspecto, dentro dos diagramas de atividade que descrevem as ações do sistema, os elementos arquitetônicos do sistema - tal como subsistemas ou camadas - podem ser também representados através de swim lanes[20].

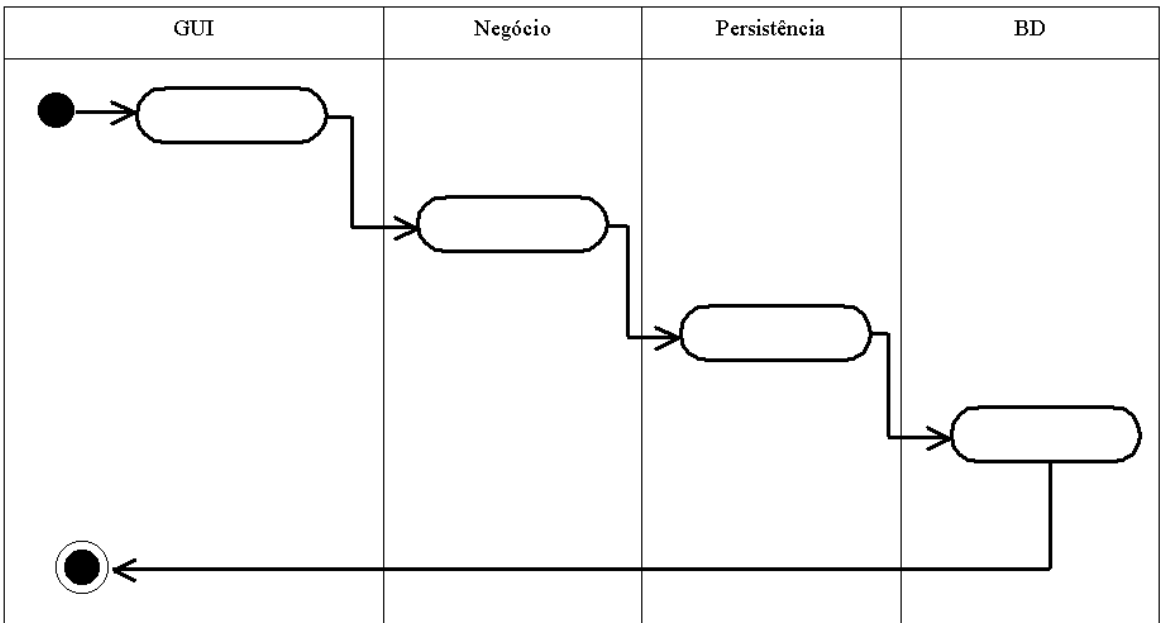


Figura 36 - Exemplo de representação de camadas através de swim lanes

[20] As swim lanes foram apresentadas como meio de se identificar seqüências alternativas, neste documento. Entretanto, a melhor forma de representação das seqüências alternativas é através de sub-diagramas, o que permite que as swim lanes possam ser utilizadas para a descrição de elementos da arquitetura sem maiores transtornos.

Na figura 36 pode-se ver como camadas da arquitetura podem ser representadas através de swim lanes. Esse tipo de descrição é interessante já que permite identificar a que camada cada atividade do sistema pertence.

Por não existir um limite quanto ao número de níveis hierárquicos que uma descrição pode utilizar, é possível ir refinando as descrições conforme a necessidade e o bom senso.

As atividades contidas nas descrições podem também orientar o fluxo de trabalho dos profissionais e servir de mecanismo para controlar o andamento do desenvolvimento. No exemplo mostrado anteriormente, pode-se verificar que a própria descrição do comportamento do sistema indica os tipos de atividades que precisam ser trabalhadas (camada com o usuário, regras de negócios, persistência dos objetos e acesso ao banco de dados).

Uma ferramenta CASE adequada, além de permitir a navegação pelos sub-diagramas das atividades, pode permitir a associação de outros artefatos a cada atividade. Com isso, é possível que as descrições continuem sendo refinadas, inclusive com outros artefatos e outros diagramas da UML.

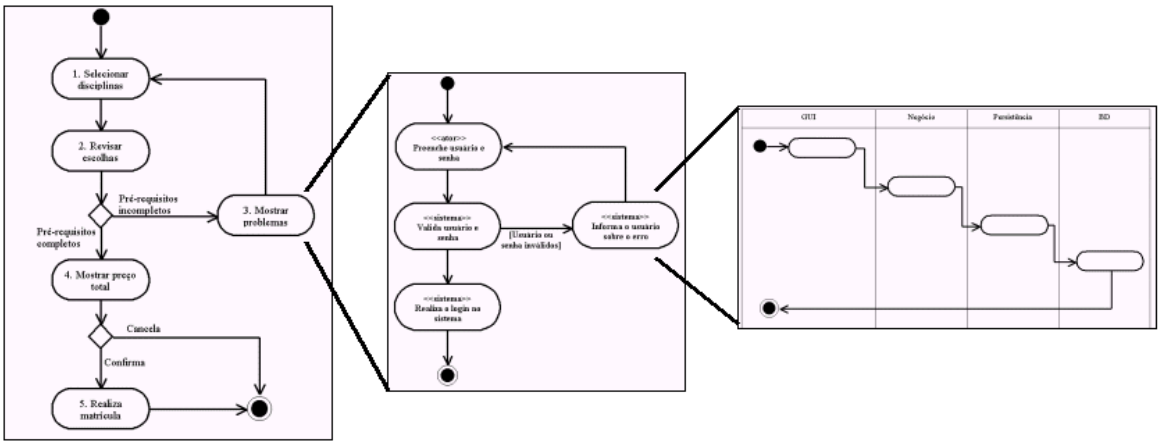


Figura 37 - Exemplo de detalhamento de uma descrição até a arquietura

A figura 37 mostra como o nível de detalhamento das descrições pode ir aumentando até se chegar a descrições onde seja interessante representar a arquitetura.

5.3 Relação com o modelo de classes

Durante o processo de análise, à medida que os requisitos do sistema são levantados e as descrições dos casos de uso são construídas, os conceitos que mais tarde se transformarão em classes, começam a ser definidos.

Esses conceitos ou classes são construídos dentro do modelo estático do sistema, por exemplo, através de diagramas de classe.

O mapeamento entre a descrição do caso de uso e o modelo de classes pode ocorrer em qualquer nível da descrição. Um exemplo de vinculação de descrição com os conceitos do modelo estático é mostrado na figura 38.

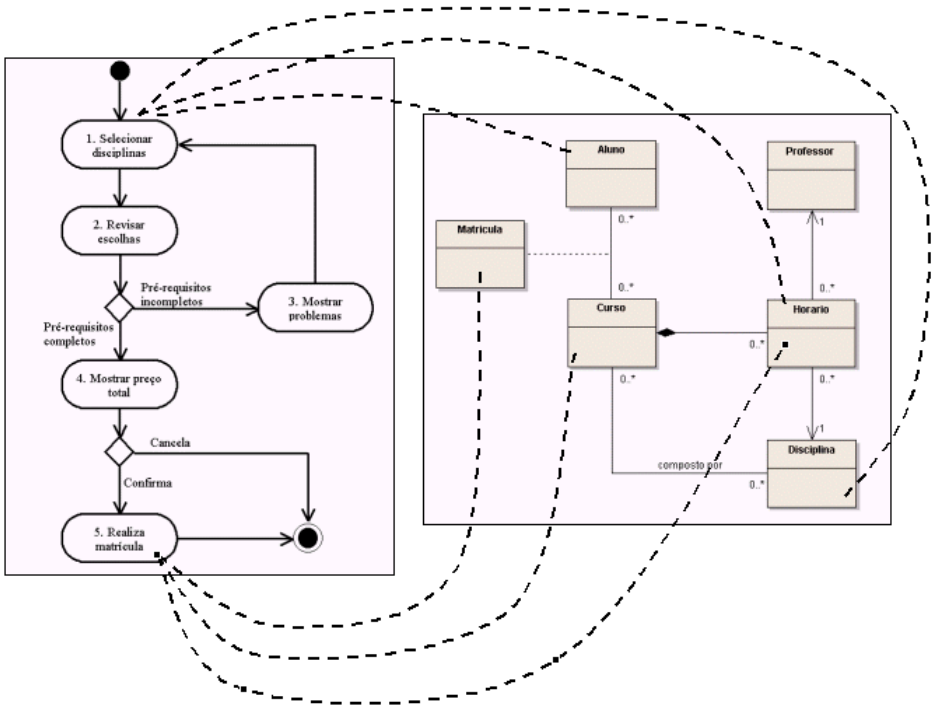


Figura 38 - Mapeamento de atividades de sistema para conceitos

Esse mapeamento tem como objetivo manter uma rastreabilidade dos requisitos e dos conceitos envolvidos. É possível, por exemplo, verificar o impacto que alterações nos requisitos podem ter sobre o modelo estático do sistema. Da mesma forma, alterações no modelo estático do sistema – por exemplo, para suportar alguma funcionalidade nova especificada por um novo requisito - podem afetar significativamente requisitos já existentes. Isso é facilmente verificável através da existência desse tipo de mapeamento.

Dentro de um contexto de descrição em diferentes níveis hierárquivos, ao serem mapeados os conceitos na última camada de descrição, todos os elementos em camadas hierárquicas mais acima automaticamente encontram-se vinculados também. Esse mecanismo permite uma rastreabilidade em todos os níveis de detalhamento da descrição, como mostrado na figura 39.

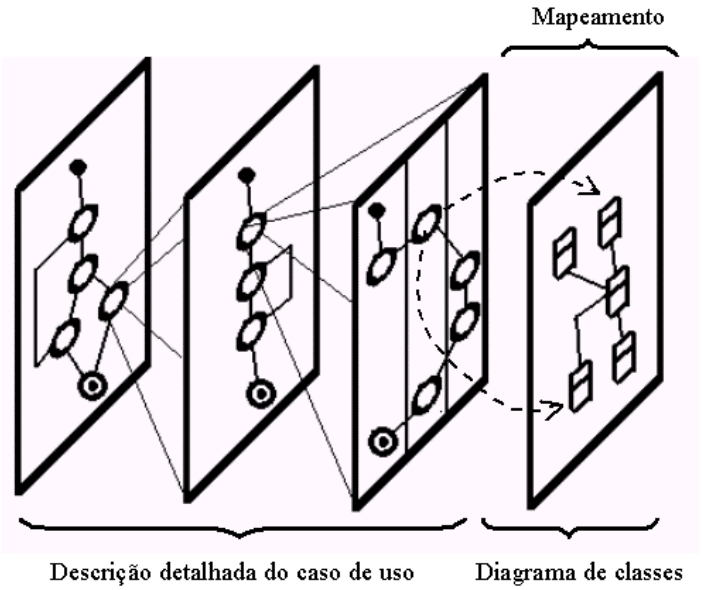


Figura 39 - Detalhamento hierárquico e mapeamento

O objetivo deste tipo de mapeamento é apenas o da rastreabilidade. Desta forma torna-se interessante poder mapear não apenas elementos da descrição (atividades) em classes, mas também em métodos ou até mesmo atributos, identificando na relação de mapeamento a intenção do mesmo (por exemplo, “referencia”, “utiliza”, “executa”, “implementado por”, etc), como mostrado na figura 40.

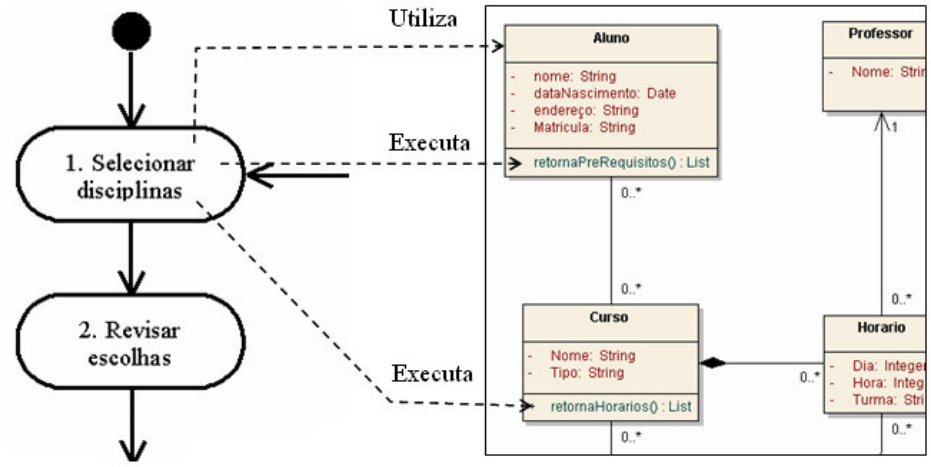


Figura 40 - Relação da descrição com classes, métodos ou atributos

6 Modelo da solução proposta

Este capítulo apresenta o modelo de implementação para a solução proposta de descrição dos casos de uso através de diagramas de atividade estereotipados. A intenção não é a construção de um modelo que represente as estruturas definidas pela UML. Ao contrário, a OMG apresenta um modelo de mapeamento muito mais genérico para a sintaxe da UML do que o apresentado neste trabalho.

Os modelos que se seguem são definidos para provar o conceito de descrição de casos de uso através de uma estrutura mais formal. Outra modelagem seria possível, inclusive uma que reaproveitasse a modelagem definida pela OMG.

A seguir são apresentados modelos para representação dos elementos da UML envolvidos na solução, e modelos que interligam o modelo de descrição ao modelo estático do sistema.

6.1 Elementos básicos

Um diagrama genérico pode ser definido como uma coleção de elementos também genéricos, eventualmente interligados por conexões. Essas conexões representam uma ligação entre dois pontos - a origem e o fim. O modelo que representa essas características pode ser verificado na figura 41.

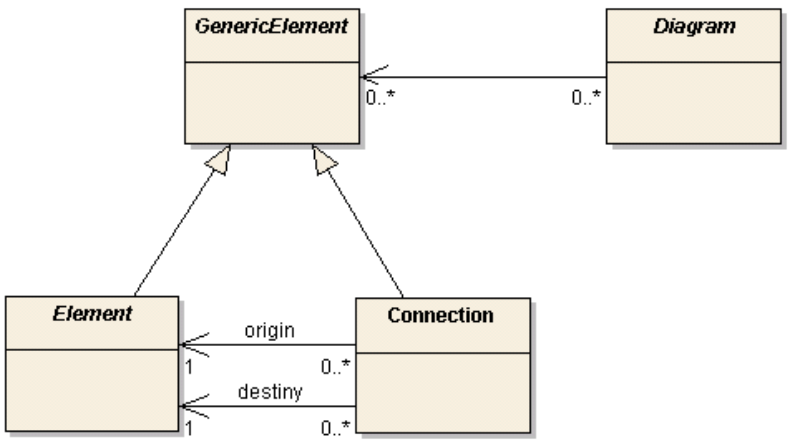


Figura 41 - Modelo genérico de um diagrama

6.2 Diagrama de Atividade

O modelo de representação do diagrama de atividade pode ser visto na figura 42.

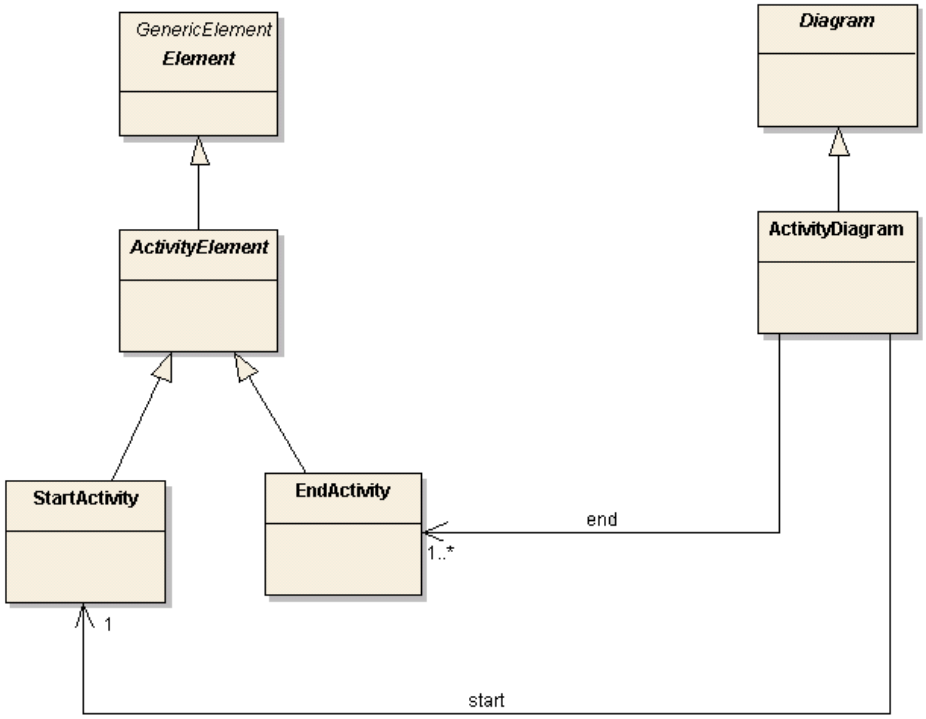


Figura 42 - Modelo do Diagrama de Atividade

Um diagrama de atividade bem formado é composto por apenas um estado de ínicio e por um ou mais estados de término.

A representação de diagramas de atividade requer alguns outros tipos de elementos. Os possíveis[21] elementos que o diagrama de atividade recebe são os seguintes:

- **Atividade inicial:** representa o ínicio do fluxo;
- **Atividade final:** representa o fim do fluxo;
- **Atividade de ator:** representa uma atividade realizada pelo ator, correspondendo a uma atividade com o estereotipo <<actor>>;

- **Atividade de sistema:** representa uma atividade realizada pelo sistema, correspondendo a uma atividade com o estereotipo <<system>>;

[21] Note-se que os elementos modelados referem-se àqueles necessários para a descrição de casos de uso, tal como proposto neste trabalho.

- **Atividade de ponto-de-inclusão:** a atividade que representa um ponto de inclusão dentro de uma descrição com diagramas de atividade. Corresponde a uma atividade com estereotipo <<inclusion point>>;
- **Atividade de ponto-de-extensão:** a atividade que representa um ponto de extensão. Corresponde a uma atividade com o estereotipo <<extension point>>;
- **Atividade de sincronização:** o mecanismo de sincronização é aqui representado como uma atividade. Na UML ele é representado como uma barra horizontal;
- **Outra atividade:** representa qualquer outra atividade que queira ser representada. Pode possuir um estereotipo associado.

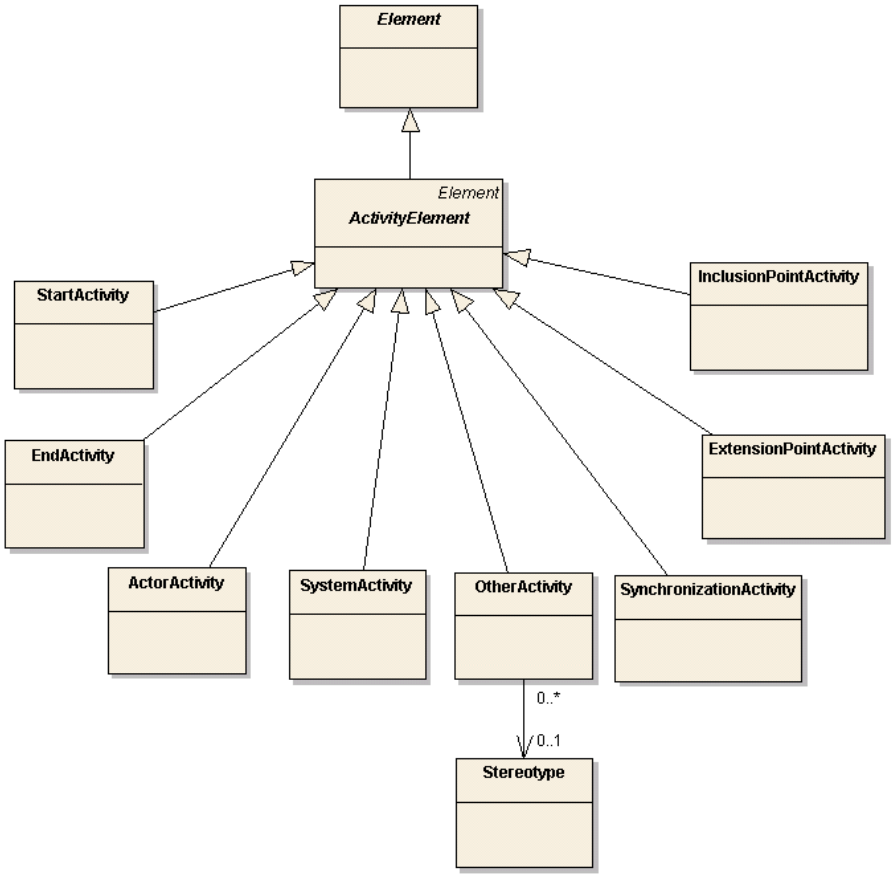


Figura 43 - Elementos do tipo "atividade"

O elemento de sistema pode possuir outro diagrama de atividade associado. Essa associação permite a navegabilidade através das atividades de sistema, e serve para representar diferentes níveis hierárquicos de descrição. Essa relação é mostrada na figura 44.

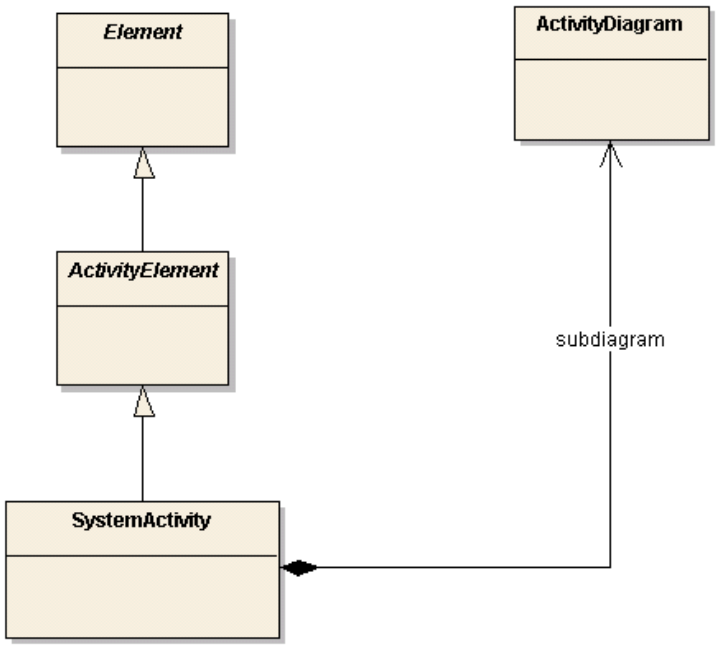


Figura 44 - Modelo para a atividade de sub-diagrama

O diagrama de atividade por sua vez, também pode ser estereotipado. O estereotipo do diagrama de atividade pode ser utilizado para diferenciar um diagrama cujo objetivo seja a representação de uma descrição de caso de uso, de outros como, por exemplo, diagramas que representem casos de colaboração, conforme mostrado na figura 45.

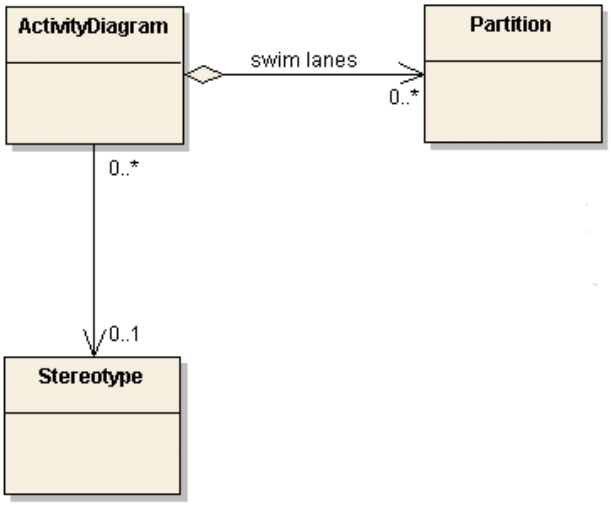


Figura 45 - Swim lanes e estereotipos dos diagramas de atividade

Os diagramas de atividade podem possuir divisões chamadas swim lanes. Essas divisões podem ser utilizadas para representar o contexto de um determinado fluxo de atividades ou então dividir o diagrama em camadas que representem uma visão da arquitetura do sistema.

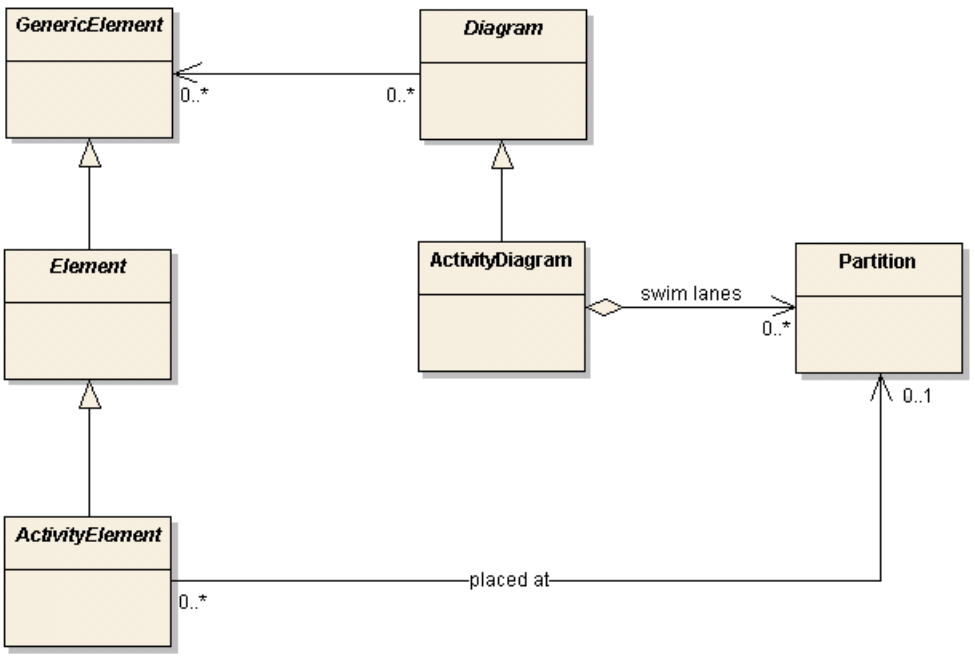


Figura 46 - Representação da localização das atividades

As atividades que compõe um diagrama de atividade podem ser associadas a uma determinada partição ou swim lane do diagrama. Quando o diagrama é dividido, obrigatoriamente os seus elementos devem estar localizados em alguma divisão. A modelagem que permite a representação da localização dos elementos de atividade dentro do diagrama pode ser vista na figura 46.

6.2 Diagrama de Caso de Uso

Os diagramas de casos de uso podem possuir dois tipos de elementos, atores e casos de uso. A importância de um bom diagrama de caso de uso reside não apenas nesses dois elementos, mas também nos diferentes tipos de associações que podem ocorrer entre os mesmos.

A conexão entre um ator e um caso de uso tipicamente é uma conexão simples. Já as conexões ou relacionamentos entre dois casos de uso possuem diferenças significativas de semântica.

Entre dois casos de uso pode existir um relacionamento de inclusão (representado por uma conexão com o estereotipo <<include>>), um relacionamento de extensão (representado por uma conexão com estereotipo <<extends>>) e o relacionamento do tipo generalização (representado por uma conexão em forma de flecha com a ponta larga).

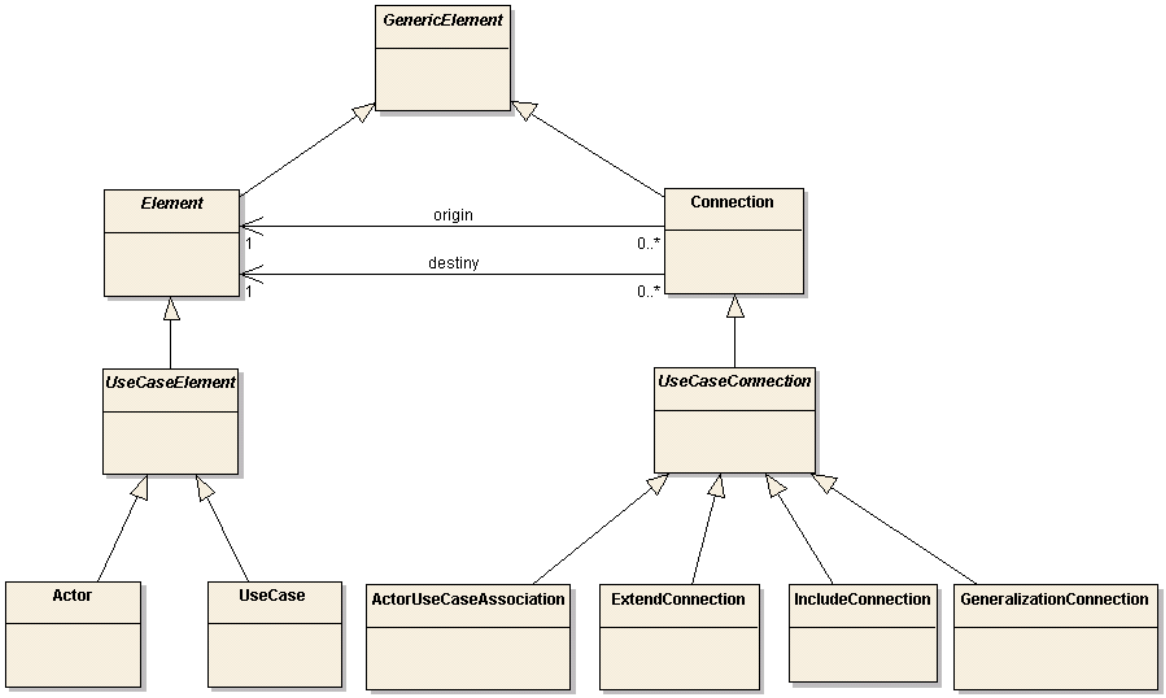


Figura 47 - Elementos e conexões de um diagrama de caso de uso

Os dois tipos de elementos (ator e caso de uso) e os quatro tipos de associação (ator-caso de uso, extensão, inclusão e generalização) estão representados no modelo apresentado na figura 47.

Um caso de uso pode possuir um ou mais pontos de extensão, ou inclusive nenhum.

Quando um relacionamento de extensão ocorre entre dois casos de uso, é indicado o ponto de extensão que está sendo estendido no próprio relacionamento. Um relacionamento de extensão obrigatoriamente deve indicar ao menos um ponto de extensão. O relacionamento de extensão pode estender mais de um ponto de extensão, como mostrado na figura 48.

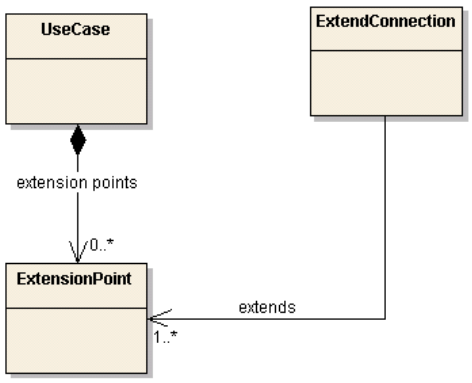


Figura 48 - Extension points

Os pontos de inclusão não precisam ser representados nos diagramas de caso de uso, mas são representados na descrição interna do fluxo de eventos.

6.3 Modelo para descrição dos casos de uso

A representação da descrição dos casos de uso requer algumas estruturas específicas capazes de representar a descrição de um caso de uso externamente aos modelos representam os diagramas de caso de uso e diagrama de atividades. Essa estrutura é mostrada na figura 49.

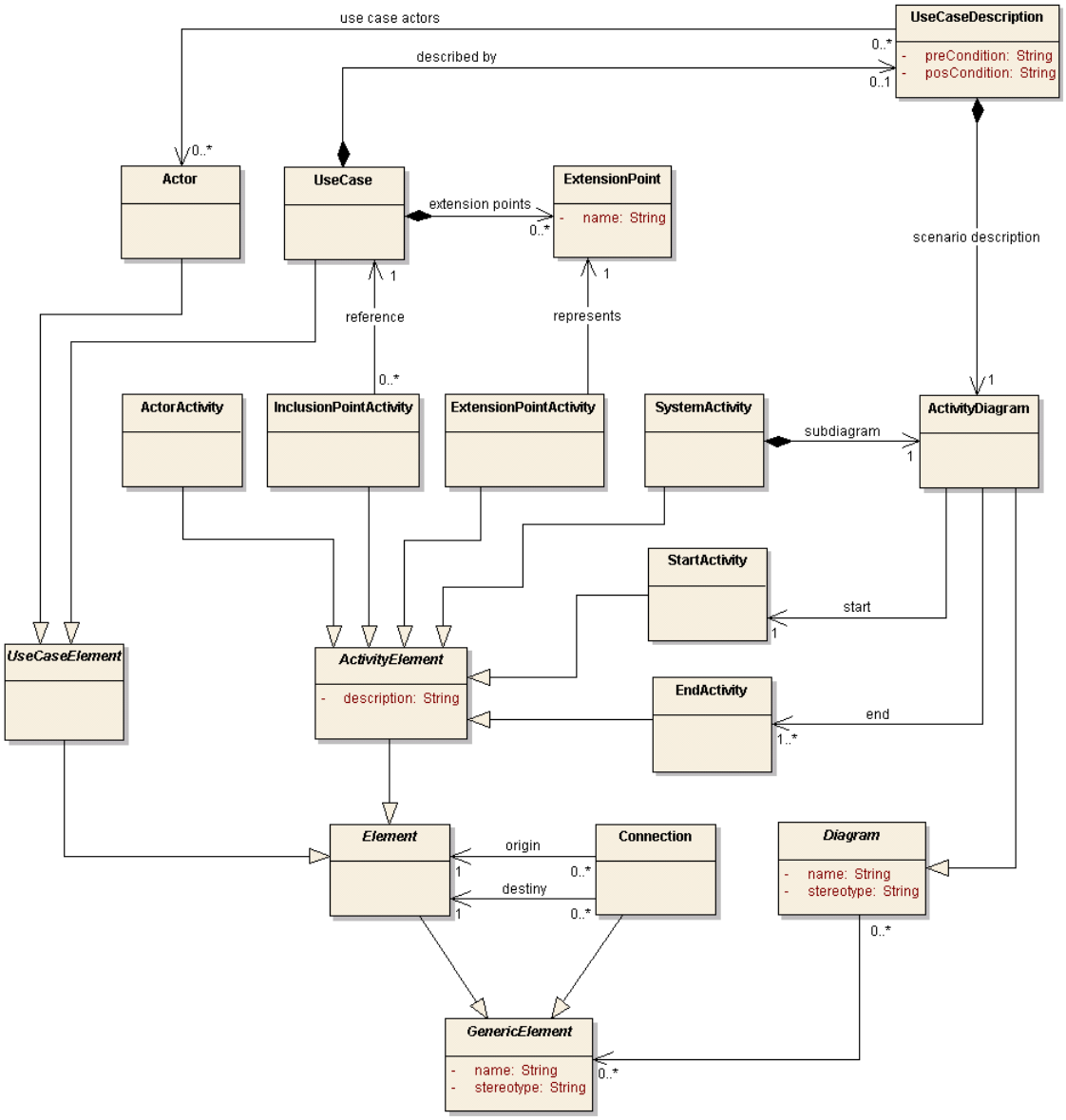


Figura 49 - Modelo para a descrição de casos de uso

6.4 Relação com o modelo estático do sistema

O modelo apresentado na figura 50 mostra a associação dos elementos que compõe a descrição (elementos de atividade) com classes e métodos, além de permitir definir as swinlanes onde os elementos se localizam como descrições da arquitetura.

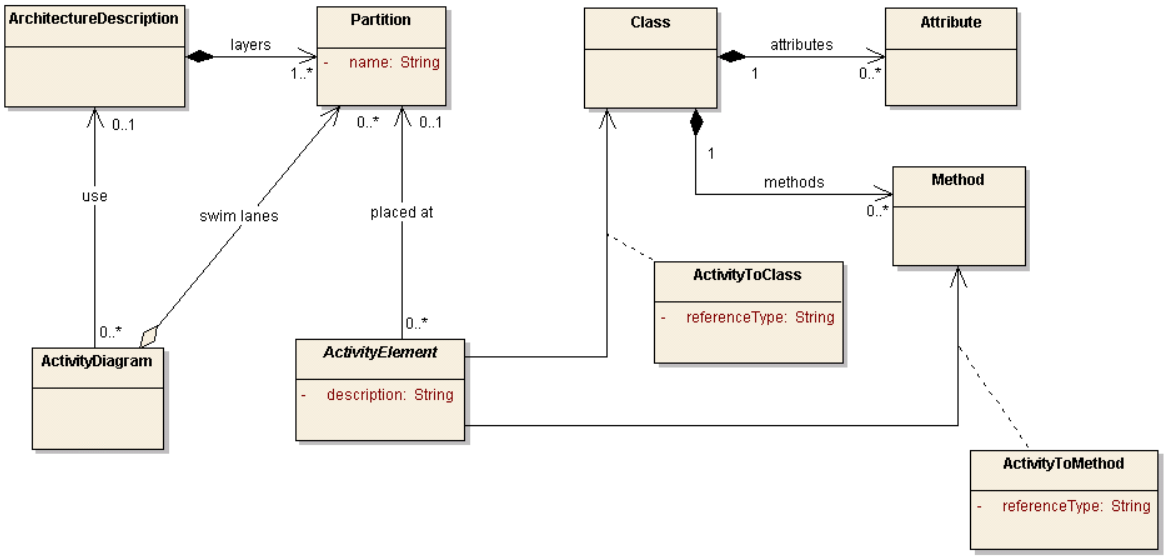


Figura 50 - Referência ao modelo estático

7 Ferramenta CASE de descrição

A maioria das ferramentas CASE de modelagem UML encontradas no mercado possuem funcionalidades que permitem associações simples entre casos de uso e artefatos de documentação externos (arquivos), geralmente utilizados para descrever os primeiros de forma narrativa sem nenhum comprometimento com uma estrutura de representação. Algumas ferramentas como, por exemplo, Visual Paradigm [Visual 2004], possuem meios um pouco mais elaborados que permitem a descrição dos casos de uso dentro da própria ferramenta. Outros projetos como, por exemplo, o desenvolvido pela PUC-Rio chamado C&L (Cenário e Léxico) [Cristoph 2004], auxiliam a descrição colaborativa de cenários introduzindo uma organização da informação de forma estruturada. Entretanto, essas ferramentas não permitem a associação de elementos internos das descrições dos casos de uso (passos / atividades) com outros elementos que compõe o modelo do sistema, nem possuem mecanismos para a geração de outros artefatos.

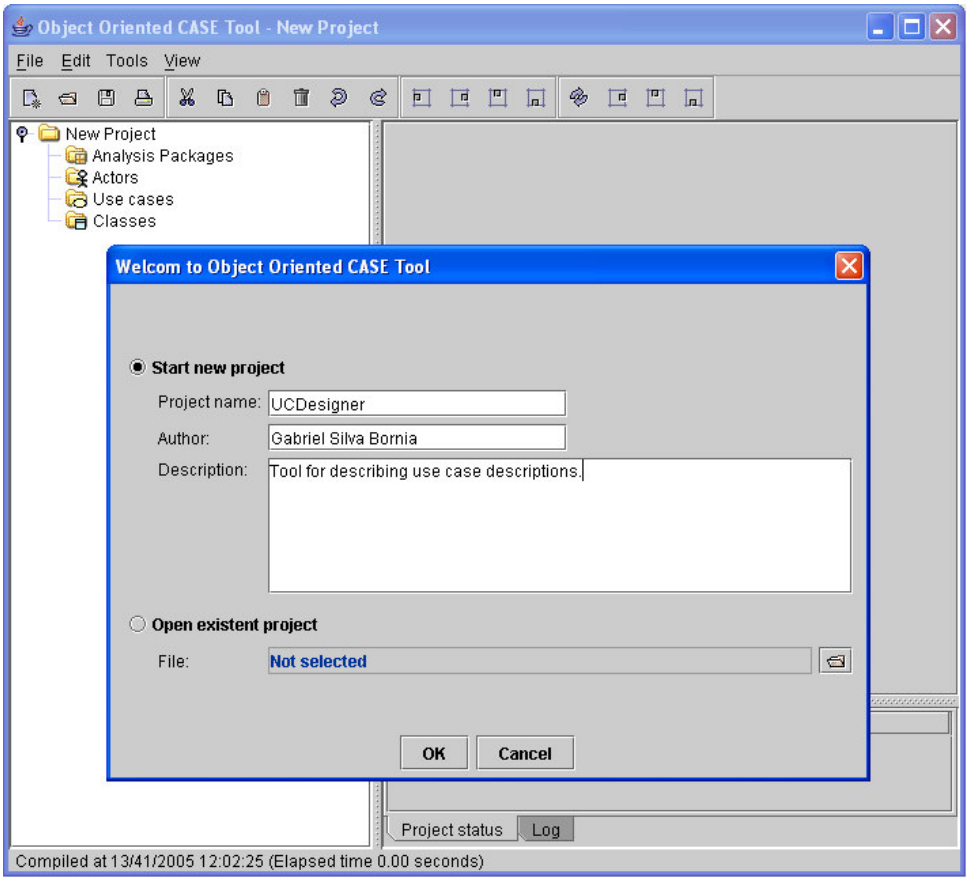


Figura 51 - A ferramenta UC Designer

Neste trabalho é apresentada uma ferramenta CASE desenvolvida com o intuito de oferecer um mecanismo de descrição baseado no método apresentado, utilizando diagramas de atividade estereotipados para estruturar as descrições de casos de uso, permitindo o processamento para a geração de artefatos e a associação entre outros elementos do modelo do sistema. A ferramenta foi chamada de UC Designer.

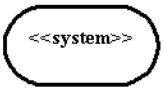
A ferramenta tem valor acadêmico e possui licença GPL (GNU Public License). Ela está disponível em http://sourceforge.net/projects/ucddesigner a toda a comunidade do software livre. O objetivo da construção da ferramenta é provar que descrições de caso de uso podem ser formalizadas em estruturas simples, já existentes em ferramentas comerciais que implementam os mecanismos de extensibilidades definidas pela OMG para a Unified Modeling Language.

A ferramenta é um editor diagramático de descrições de caso de uso baseadas em diagramas de atividade. As atividades estereotipadas foram transformadas em ícones para facilitar a leitura do diagrama.

7.1 Nomenclatura da ferramenta

Os elementos de um diagrama de atividades são representados pelos ícones mostrados na tabela 8:

Tabela 8 - Notação da ferramenta UC Designer

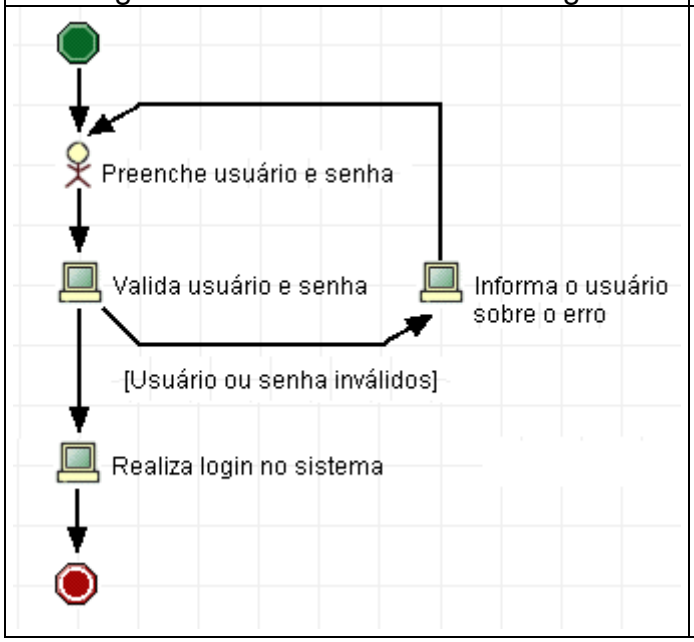
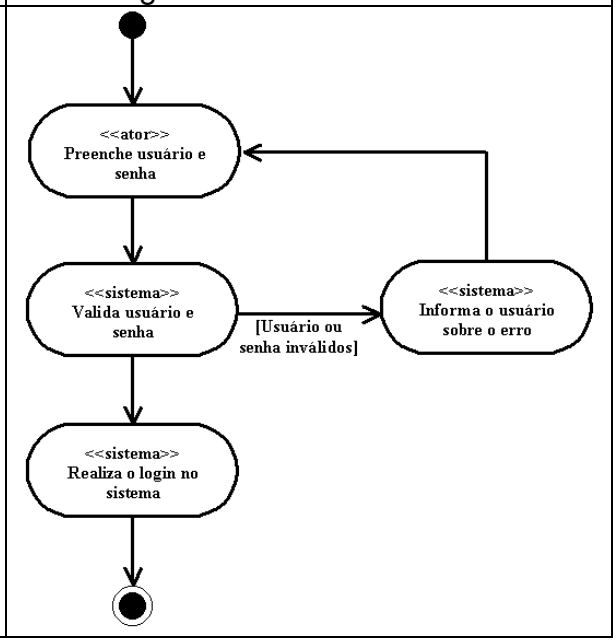
UC Designer (ícone)	Elemento UML	Descrição
		Estado de inicio do diagrama.
		Atividade que representa o ator.
		Atividade que representa o sistema.
		Atividade que representa um ponto de inclusão.
		Atividade que representa um ponto de extensão.
		Elemento que representa o sincronismo de fluxos de atividade.
		Estado final do diagrama.

Diversas ferramentas de editoração permitem a associação de ícones a elementos estereotipados da UML. É o caso da ferramenta Rational Rose®.

A ferramenta não utiliza o elemento de decisão definida pela UML, por entender que os caminhos alternativos podem ser representados como transições condicionadas[22] saídas diretamente dos elementos de atividade.

A representação de inicio de paralelismo também foi abstraídas, uma vez que duas transições saídas de uma mesma atividade podem ser consideradas atividades paralelas. Como o objetivo da ferramenta não é a geração de artefatos que requisitem esse tipo de precisão, determinados tipos de controle foram retirados do que normalmente se vê em implementações de diagramas de atividade. Para fins de geração de artefatos de documentação o conjunto de elementos de diagramas de atividade é suficiente.

Tabela 9 - Diagrama do UC Designer comparado com um diagrama padrão

Diagrama de Atividade do UC Designer	Diagrama de Atividade Padrão
	

7.2 Funcionalidades da ferramenta

A ferramenta permite a criação de casos de uso, a descrição e a associação dessas descrições estruturadas com o modelo lógico, como mostra a figura abaixo.

[22] Transições com guardas (condições lógicas).

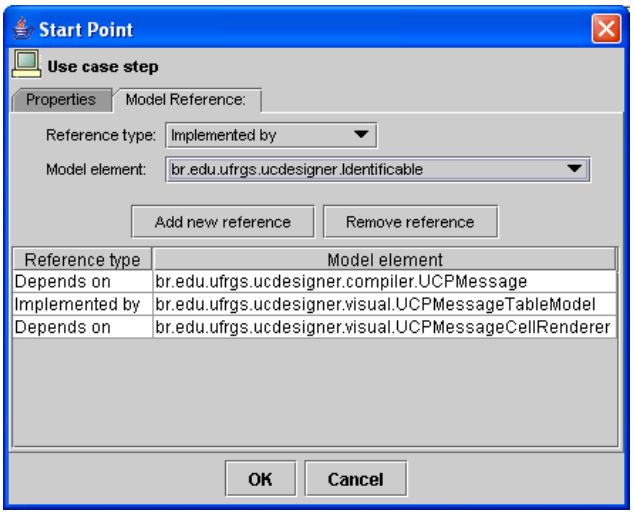


Figura 52 - Associação de item da descrição com o modelo lógico

A ferramenta possui um editor de diagrama de atividade voltado para a construção de descrições de caso de uso. À medida que o diagrama de atividade é construído, a descrição textual do caso de uso vai sendo gerada. Na figura abaixo pode se ver um exemplo da interface na construção de uma descrição de caso de uso.

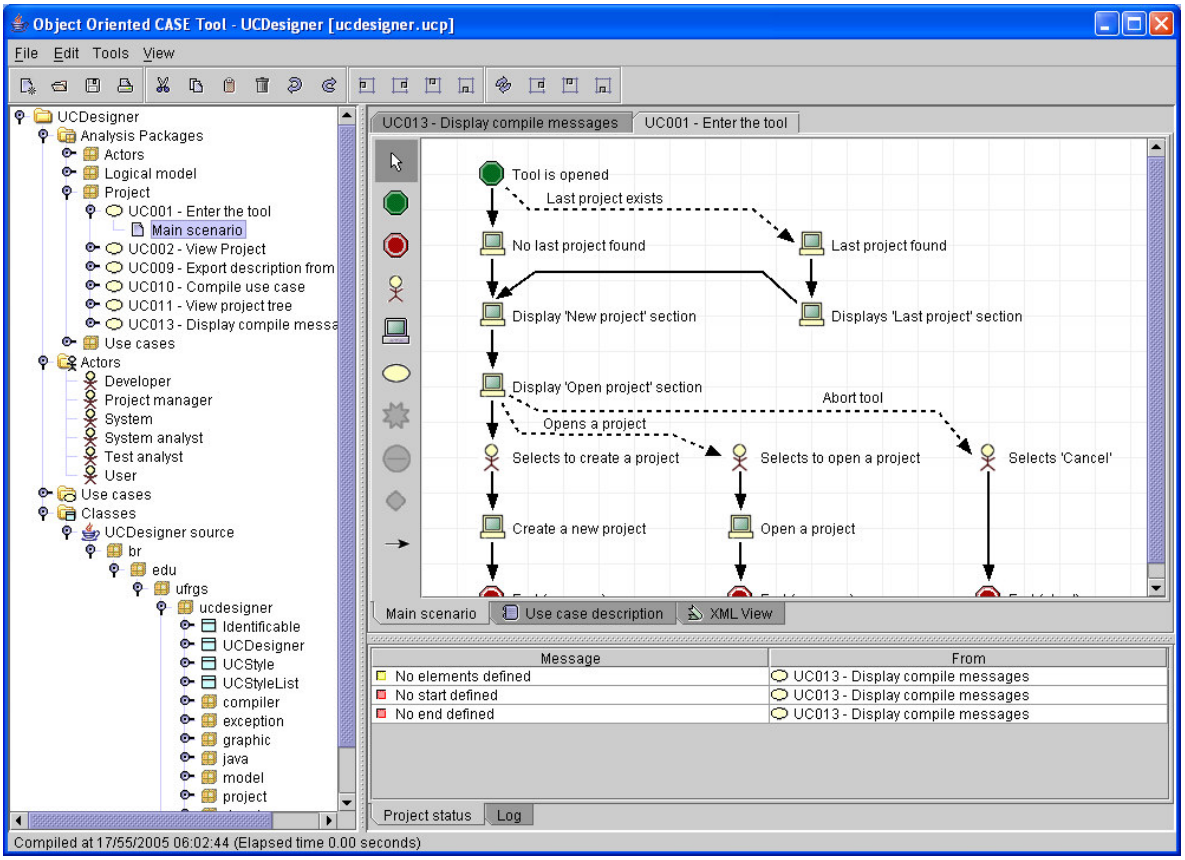


Figura 53 - Exemplo de descrição de um caso de uso

7.3 Mecanismo de transformação

A representação da descrição é armazenada através de XML. É essa representação que serve de entrada ao mecanismo de transformação que processa o XML e o transforma em uma saída diferente.

A saída pode ser uma descrição textual como a verificada na imagem anterior, ou então qualquer outro artefato que possa servir aos demais processos do desenvolvimento de software. Um exemplo seria a saída de uma descrição em LOTOS que pudesse ser executada para poder encontrar todos os caminhos possíveis, e ser utilizada em mecanismos automatizados de testes.

A representação da descrição do exemplo anterior é dada pelo seguinte trecho de XML:

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
<project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use case descriptions.</description>
  <actors>
    <actor id="9" name="User"/>
    <actor id="5" name="Test analyst" parent="9"/>
    <actor id="4" name="Project manager" parent="9"/>
    <actor id="3" name="System analyst" parent="9"/>
  </actors>
  <usecases>
    <usecase id="10" ucid="UC001" name="Enter the project">
      <description>The user opens the tool for creating
                   or updating a project.</description>
      <preconditions>The tool is opened.</preconditions>
      <actor id="9"/>
      <scenario name="Main scenario">
        <path type="main" name="Main flow">
          <step id="11" name="Start" index="1" type="start">
            <description>The tool is opened. The main window is opened
                         and the 'Start project' window is opened in front
                         of it.</description>
          </step>
          <step id="12" name="No project was created" index="2" type="system">
            <description>Verifies that it's no pre-existing project was created
                          with this tool.</description>
          </step>
          <step id="13" name="Show 'Create new project'" index="3" type="system">
            <description>Allows the user to enter a new project, showing the
                         'Create new project' section.</description>
          </step>
          <step id="14" name="Show 'Open existing project'" index="4" type="system">
            <description>Allows the user to search and open other project,
                         showing the 'Open existing project' section.</description>
          </step>
          <step id="15" name="Selects an option" index="5" type="actor">
            <description>Selects to create an option and inform all the
                         required data if necessary.</description>
          </step>
          <step id="16" name="Selects 'OK'" index="6" type="actor">
            <description>Selectes the 'OK' option.</description>
          </step>
          <step id="22" name="Creates a new project" index="7" type="system">
            <description>Creates a new project based on the data provided by
                         the user.</description>
          </step>
          <step id="24" name="End (success)" index="8" type="end">
            <description>Close the 'Start project' window and return to the
                         main screen.</description>
```

```
          </step>
        </path>
        <path type="alternative" name="Not the first project" begin="12"
              return="13">
          <step id="26" name="Last project exists" index="1" type="system">
            <description>Verifies that a last project exists.</description>
          </step>
          <step id="28" name="Show 'Open last project'" index="2" type="system">
            <description>Allows user to select the last opened project, showing
                         the 'Last project' section with the name and file of
                         the last project.</description>
          </step>
        </path>
        <path type="alternative" name="Cancel action" begin="14">
          <step id="32" name="Selects 'Cancel'" index="1" type="actor">
            <description>User selects to cancel the action.</description>
          </step>
          <step id="34" name="End (abort)" index="2" type="end">
            <description>The action is aborted and the tool is closed.</description>
          </step>
        </path>
      </scenario>
    </usecase>
  </usecases>
</project>
```

O mecanismo de transformação poderia ser qualquer parser que interpretasse um XML como o representado acima. Entretanto, a ferramenta foi contruída com um parser que utiliza um arquivo de regras XSL para realizar a transformação, conforme a figura abaixo.

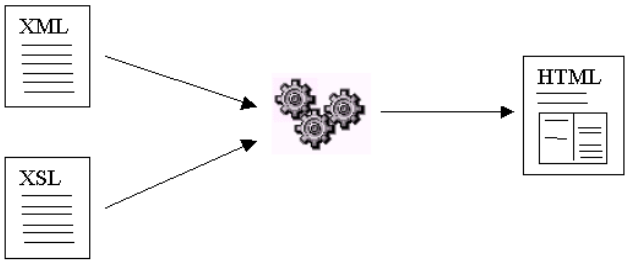


Figura 54 - Transformação através de XSL

A ferramenta vem com um arquivo de regras padrão que pode ser alterado para gerar a descrição textual ao lado do diagrama, como mostrado anteriormente.

O arquivo de regras criado é o seguinte:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >

<!-- UCDesigner name="simple-desctiption" description="Simple description (default)"
extension="html" target="word" -->

<xsl:output method="html" indent="yes"/>


<xsl:template match="project/usecases/usecase">
<HTML><BODY>
```

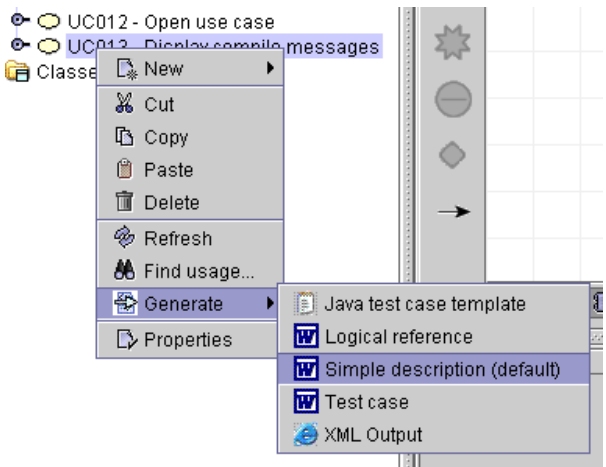
```
  <table border="1"  width="100%">
  <tr><td colspan="2" bgcolor="#cacaca"><b>Use case summary</b></td></tr>
  <tr><td valign="top">Use case index:</td><td>
            <xsl:value-of select="@ucid"/></td></tr>
  <tr><td valign="top">Use case name:</td><td>
            <xsl:value-of select="@name"/></td></tr>
  <tr><td valign="top">Description:</td><td>
            <xsl:value-of select="description"/></td></tr>
  <tr><td valign="top">Actors:</td>
        <td>
              <xsl:for-each select="actor">
                   <xsl:value-of select="@name"/> <br/>
              </xsl:for-each>
        </td>
     </tr>
    <tr valign="top"><td>Pre Conditions:</td><td>
                  <xsl:value-of select="preconditions"/></td></tr>
    <tr valign="top"><td>Post Conditions:</td><td>
                  <xsl:value-of select="postconditions"/></td></tr>
  <xsl:apply-templates/>
  </table>
  <br/>
  UCDesigner 1.0
</BODY></HTML>
</xsl:template>

<xsl:template match="project/usecases/usecase/scenario">
  <tr><td colspan="2" bgcolor="#cacaca"><b>Scenario: </b>
         <xsl:value-of select="@name"/></td></tr>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="project/usecases/usecase/scenario/path">
  <tr>
  <td valign="top">
      <xsl:if test="@type = 'alternative'"> Alternative flow</xsl:if>
      <xsl:if test="@type = 'main'">Main flow</xsl:if>
  </td>
  <td valign="top">
      <xsl:if test="@type = 'alternative'">
        <b><xsl:value-of select="@name"/></b><br/>
        <i>Starts in step <xsl:value-of select="id(@begin)/@index"/>
           (<xsl:value-of select="id(@begin)/@name"/>) of
            <xsl:value-of select="id(@begin)/../@name"/></i><br/>
      </xsl:if>

  <xsl:for-each select="step">
            <b>
            <xsl:value-of select="format-number(@index,'00')"/> 
            <xsl:value-of select="@name"/>
            </b><br/>
            <xsl:if test="@type = 'system'">System:</xsl:if>
            <xsl:if test="@type = 'include'">System:</xsl:if>
            <xsl:if test="@type = 'actor'">Actor:</xsl:if>
            <xsl:value-of select="description"/><br/>
            <xsl:if test="@type = 'include'"> &lt;&lt;include
                  <u><xsl:value-of select="@ucid"/> -
                     <xsl:value-of select="@ucname"/></u><br/></xsl:if>
            <br/>
  </xsl:for-each>
  </td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```



Os mecanismos de transformação tornam-se disponíveis através de um

menu que pode ser acessado para cada caso de uso, conforme a figura ao lado.

O mecanismo de extensibilidade da ferramenta permite que facilmente novos mecanismos sejam incorporados à ferramenta e associados a ferramentas externas, conforme figura 55.

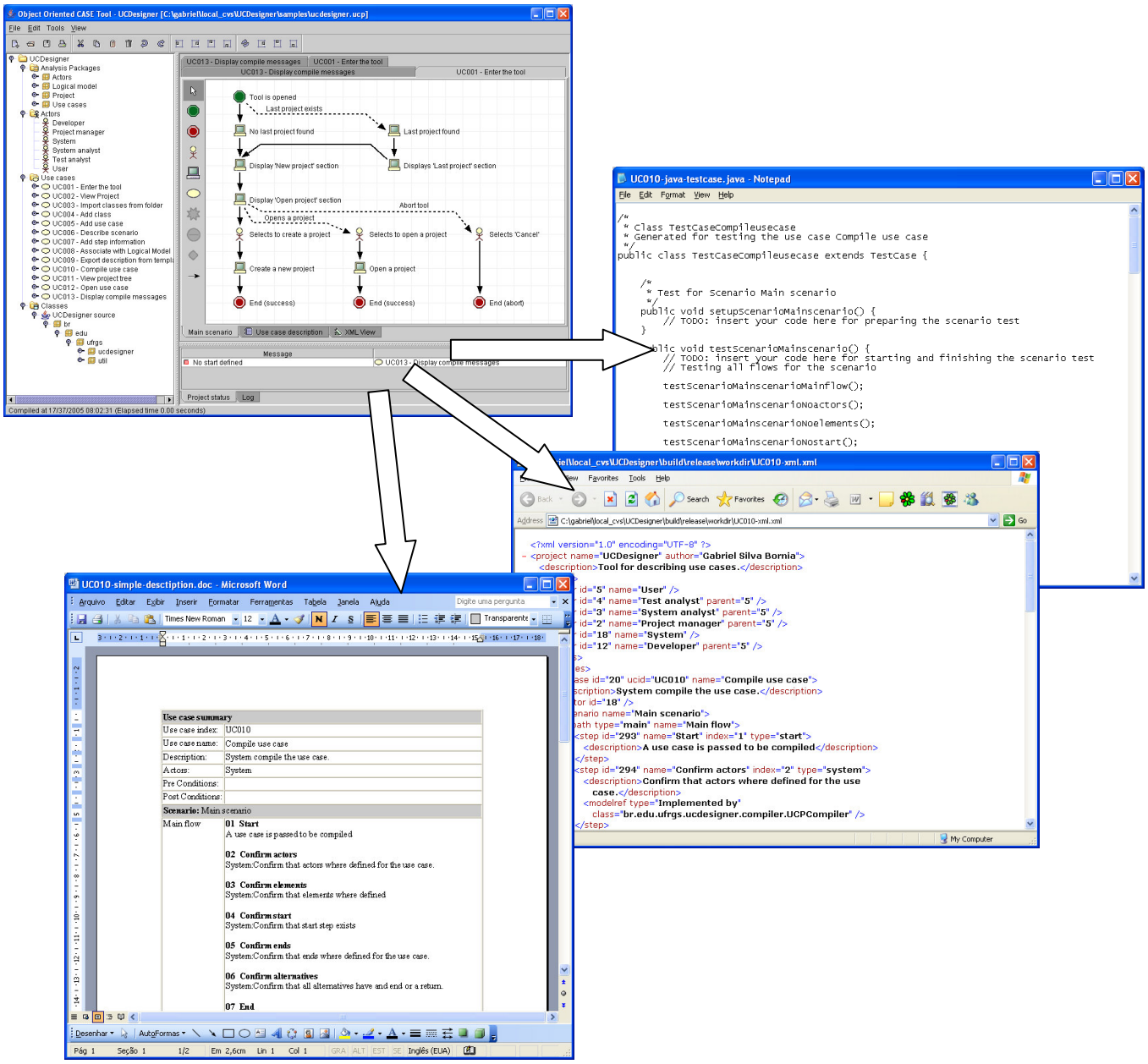


Figura 55 - Geração de artefatos para aplicativos externos

8 Considerações finais

A importância dos casos de uso vai muito além do escopo da representação de requisitos. Os casos de uso guiam todo o processo de desenvolvimento, a concepção da arquitetura, o projeto do sistema, a validação dos componentes implementados, a documentação do sistema, entre outras tarefas que compõe o ciclo de vida do sistema.

O fato de se utilizar uma linguagem natural para a descrição interna dos casos de uso pode ser importante para a validação com os usuários, mas é uma informação que pouco pode ser sistematizada em processos automatizados que possam reaproveitar essas informações em outras etapas do desenvolvimento de software.

Este trabalho propõe um mecanismo que estruture as descrições dos casos de uso, de forma que a própria representação dos requisitos sirva como mecanismo de validação junto aos usuários e ao mesmo tempo sirva de entrada para mecanismos computacionais capazes de utilizar essas informações para a geração de outros artefatos úteis nas demais etapas no desenvolvimento do sistema.

O uso da extensibilidade da UML é indicado como meio de utilizar a própria linguagem UML como forma de representação estruturada para descrições de casos de uso. O objetivo é mostrar que as descrições de casos de uso podem ser modeladas através da própria UML e – se devidamente assistidas por uma ferramenta CASE – servir de base para a geração de outros artefatos que auxiliem o processo de desenvolvimento de software.

Pelo fato da descrição dos casos de uso não possuir nenhum mecanismo formal de representação, não existe tampouco uma forma de validar o modelo de casos de uso com as descrições contidas dentro de cada caso de uso. Por exemplo, o modelo de casos de uso pode indicar a existência de um relacionamento de extensão entre dois casos de uso enquanto que as descrições desses casos de uso podem não possuir nenhuma indicação dessas relações. Como isso é validado? Atualmente cabe ao analista manter uma coerência com aquilo que ele representa no modelo de caso de uso e o que escreve nas descrições dos casos de uso. Mas e se o modelo e a descrição forem realizadas por pessoas diferentes?

Existem muitos passos no desenvolvimento de software que poderiam ser facilitados por uma ferramenta CASE que estruturasse as descrições dos casos de uso. Por exemplo, se o fluxo de eventos fosse devidamente representado, poder-se-ia gerar diversos artefatos úteis ao processo de desenvolvimento.

A proposta deste trabalho é então apresentar o uso de diagramas de atividade da UML como veículo para um método de representação estruturado de

descrições de casos de uso que se baseia na flexibilidade oferecida pelos mecanismos de extensibilidade da UML.

A geração de diversos artefatos pressupõe a existência de uma ferramenta que transforme a descrição estruturada sob a forma de um diagrama de atividade estereotipado em artefatos úteis durante o processo de desenvolvimento do sistema.

8.1 Descrições textuais

As descrições de casos de uso representadas através de diagramas de atividade podem ser processadas para gerar descrições textuais, conforme os padrões ou *templates* que forem definidos.

Esta funcionalidade permite que documentações narrativas possam ser geradas para validação com usuários e comunicação entre os participantes do projeto. Os diferentes níveis hierárquicos de descrição permitidos pelo método proposto garantem a possibilidade de se gerar descrições com diferentes graus de detalhamento. Por exemplo, descrições textuais de casos de uso essenciais e reais podem ser extraídas a partir de uma descrição hierárquica de dois níveis.

Descrições que venham a ser utilizadas por projetistas ou desenvolvedores poderiam possuir, além de um nível de detalhamento alto, referências ao modelo estático do sistema, de forma a tornar a descrição gerada um elemento útil e importante dentro do processo de desenvolvimento.

8.2 Cenários de teste

O formato estruturado de representação permite que todos os fluxos (principal e alternativos) possam ser percorridos, possibilitando então a geração de cenários de teste.

Embora os cenários de teste sejam artefatos que precisam de diversas informações como, por exemplo, uma massa de dados de entrada, a possibilidade de se gerar um esqueleto com todos os cenários de teste possíveis para que estes possam ser preenchidos é animador. Uma ferramenta CASE poderia ainda auxiliar o usuário a preencher os cenários de teste.

8.3 Índices e matrizes de dependências

O armazenamento de descrições de casos de uso através de um mecanismo estruturado como o proposto neste trabalho permite a geração de

índices de casos de uso – úteis para a manutenção da documentação do projeto – além de permitir gerar matrizes de dependência.

As matrizes de dependência são as informações que correlacionam os casos de uso que são incluídos ou estendidos por outros. É uma artefato extremamente relevante para o gerenciamento do projeto uma vez que permite organizar e estabelecer ordens para implementação.

8.4 Métricas

As descrições estruturadas permitem a geração de diversas métricas, uma vez que permitem a contabilização de interações entre o ator e o sistema. Além disso, é possível facilmente extrair a complexidade analisando o número de relações com outros casos de uso (presentes nas descrições) além de poderem se basear também em diferentes níveis hierárquicos de descrição. Esses dados podem ser utilizados como entrada para cálculos de estimativa de esforço para estimar tempos de implementação [2].

8.5 Simulação

A estruturação das descrições de casos de uso permite a geração de scripts para realizar a simulação de passos que possam ser executados dentro de um caso de uso como, por exemplo escripts em LOTOS, de forma que esse script possa servir como base para possíveis estudos para testes automatizados.

8.6 Padrões de casos de uso

A existência de uma descrição diagramática para os casos de uso pode permitir a identificação de padrões de descrições de casos de uso, que eventualmente podem vir a ser catalogados em padrões.

A catalogação em padrões de casos de uso permite que ferramentas sejam estendidas e que um novo vocabulário possa ser criado para identificar diversos tipos de comportamentos de casos de uso.

8.7 Interação Homem-Máquina

As descrições podem ser enriquecidas com informações que permitam auxiliar a construção de interfaces homem-máquina analisando e até mesmo simulando a interação do ator com o sistema.

8.8 Gerenciamento de projetos

O uso de descrições de casos de uso estruturado traz impactos ao gerenciamento de projetos. As principais vantagens é a análise de impacto de alterações aos casos de uso e a rastreabilidade de requisitos.

Alterações em determinados casos de uso podem gerar diversos impactos no desenvolvimento. Através de um mecanismo de descrição estruturado como o proposto, que permite o mapeamento entre as descrições e o modelo estático, é possível medir o impacto de alterações de análise.

Bibliografia

[Ambler 2000] Ambler, Scott W. *Documenting a use case: what to include, and why.* Ronin International. EUA, 2000.

[Anda 2001] Anda, Bente; Dreiem, Hege; Sjøberg, Drag I.K.; Jørgensen, Magne. *Estimating software developmen effort based on use cases: experiences from industry.* 4th International Conference on the Unified Modeling Language: UML 2001. Canada, 2001.

[Armour 2000] Armour, Frank; Miller, Granville. *Advanced use case modeling: software systems.* Addison-Wesley. EUA, 2000.

[Bernandi 2002] Bernardi, Simona; Donatelli, Susanna; Merseguer, José. *From UML sequence diagrams and statecharts to analysable petri net models.* Workshop on Software and Performance; Proceedings of the 3rd international workshop on Software and performance. Itália, 2002.

[Boerger 2000] Börger, Egon; Cavarra, Alessandra; Riccobene, Elvinia. *An ASM Semantics for UML Activity Diagrams.* Algebraic Methodology and Software Technology: 8th International Conference, AMAST 2000, Iowa City, Iowa. EUA, 2000.

[Booch 1999] Booch, Grady; Rumbaugh, James; Jacobson, Ivar. *The Unified modeling language user guide.* Addison-Wesley. EUA, 1999.

[Cockburn 2000] Cockburn, Alistair. *Structuring use cases with goal.* Humans and Technology. EUA, 2000.

[Cockburn 2000b] Cockburn, Alistair. *Writing Effective Use Cases.* Addison-Wesley. EUA, 2000.

[Cockburn 2002] Cockburn, Alistair. *Use cases, ten years later – from their evolution, learn what to expect and how to better work with them.* STQE Magazine. EUA, 2002.

[Cristoph 2004] Cristoph, Roberto; Felicíssimo, Carolina; Leite, Julio. *C&L: Uma ferramente de edição e visualização de cenários léxicos.* PUC-RS. http://sl.les.inf.puc-rio.br/cel/. Último acesso em 10/02/2004.

[Dumas 2001] Dumas, Marlon; Hofstede, Arthur H.M. ter. *UML Activity Diagrams as a Workflow Specification Language.* In *UML*

2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings. Springer Berlin / Heidelberg. Canada, 2001.

[Ecklund 1996] Ecklund, Earl F.; Delcambre, Lois; Freiling, Michael. *Change cases: use cases that identify future requirements.* 11th ACM Conference on Object-Oriented Programming Systems, Languages and Applications: OOPSLA'96. EUA, 1996.

[Eshuis 2002] Eshuis, Rik; Wieringa, Roel. *A Formal Semantics for UML Activity Diagrams – Formalising Workflow Models*. University of Twente, Department of Computer Science. Holanda, 2002.

[Eshuis 2004] Eshuis, Rik; Wieringa, Roel. *Tool Support for Verifying UML Activity Diagrams*. IEEE Transactions on Software Engineering, vol. 30, no. 7, pp. 437-447. Holanda, 2004.

[Firesmith 1995] Firesmith, Donald G. *Use cases: the pros and cons. Report on Object Analysis and Design 2.* EUA, 1995.

[Fowler 1998] Fowler, Martin; Cockburn, Alistair; Jacobson, Ivar; Anderson, Bruce; Graham, Anderson. *"Question time! About use cases"*. 13th ACM Conference on Object-Oriented Programming Systems, Languages and Applications: OOPSLA'98. Canada, 1998.

[Fowler 1999] Fowler, Martin; Scott, Kendall. *UML distilled: a brief guide to the standard object modeling language.* Addison-Wesley. EUA, 1999.

[Génova 2002] Génova, Gonzalo; Llorens, Juan; Quintana, Victor. *Digging into use case relationships.* 5th International Conference on the Unified Modeling Language: UML 2002. Alemanha, 2002.

[Hurlbut 1997] Hurlbut, Russel R. *The Three R's of Use Case Formalisms: Realization, Refinement, and Reification.* Expertech, Ltda. EUA, 1997.

[Hurlbut 1997b] Hurlbut, Russel R. *A Survey of Approaches for Describing and Formalizing Use Cases.* Expertech, Ltda. EUA, 1997.

[Jacobson 1992] Jacobson, Ivar; Christerson, Magnus; Patrik Jonsson; Övergaard, Gunnar. *Object-oriented software engineering: a use case driven approach.* Addison-Wesley. EUA, 1992.

[Jacobson 1998] Jacobson, Ivar; Booch, Grady; Rumbaugh, James. *The*

unified software development process. Addison-Wesley. EUA, 1998.

[Kruchten 1995] Kruchten, Philippe. *Architectural blueprints – the "4+1" view model of software architecture.* IEEE Software. EUA, 1995.

[Larman 1998] Larman, Craig. *Applying UML and patterns: an introduction to object-oriented analysis and design.* Prentice Hall. EUA, 1998.

[Leffingwell 2000] Leffingwell, Dean; Widrig, Don. *Managing Software Requirements: A Unified Approach.* Addison-Wesley. EUA, 2000.

[Linzhang 2004] Linzhang, Wang; Jiesong, Yuan; Xiaofeng, Yu; Jun, Hu; Xuandong, Li; Guoliang, Zheng. *Generating Test Cases from UML Activity Diagram based on Gray-Box Method*. pp. 284-291, 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004.

[Mattingly 1998] Mattingly, Le Roy; Rao, Harsha. *Writing effective use cases and introducing collaboration cases. Journal of object oriented programming 11.* EUA, 1998.

[Rosenberg 1999] Rosenberg, Doug; Scott, Kendall. *Use Case Driven Object Modeling with UML: A practical approach. Addison-Wesley. EUA, 1999.*

[Rumbaugh 1999] Rumbaugh, James; Jacobson, Ivar; Booch, Grady. *The unified modeling language reference manual.* Addison-Wesley. EUA, 1999.

[Scheneider 1998] Schneider, Geri; Winters, Jason. *Applying Use Cases: A practical guide.* Addison-Wesley. EUA, 1998.

[Sendall 2000] Sendall, Shane; Strohmeier, Alfred. *From use caso to system operation specifications.* 3th International Conference on the Unified Modeling Language: UML 2000. Reino Unido, 2000.

[Shaw 1996] Shaw, Mary; Garlan, David. *Software Architecture: perspectives on an emerging discipline.* Prentice Hall. EUA, 1996.

[Visual 2004] Visual Paradigm for UML. http://www.visual-paradigm.com. Último acesso em 10/02/2004.

[Wirfs-Brock 1993] Wirfs-Brock, Rebecca. *Designing scenarios: making the case*

for the use case framework. Smalltalk report. EUA, 1993.

Anexo 1 – Templates XSL da ferramenta UCDesigner

A seguir, os arquivos XSL utilizados na ferramenta UCDesigner para o processamento das descrições estruturadas de casos de uso.

Arquivo default.xsl
Objetivo Prover uma descrição textual do caso de uso.
Conteúdo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >

<!--  UCDesigner  name="simple-desctiption"  description="Simple  description
(default)" extension="doc" target="word" -->

<xsl:output method="html" indent="yes"/>


<xsl:template match="project">
<HTML><BODY>
<xsl:for-each select="usecases/usecase">
  <table border="1"  width="100%">
  <tr><td colspan="2" bgcolor="#cacaca"><b>Use case summary</b></td></tr>
  <tr><td valign="top">Use case index:</td><td><xsl:value-of
 select="@ucid"/></td></tr>
  <tr><td valign="top">Use case name:</td><td><xsl:value-of
 select="@name"/></td></tr>
  <tr><td valign="top">Description:</td><td><xsl:value-of
 select="description"/></td></tr>
  <tr><td valign="top">Actors:</td>
        <td>
              <xsl:for-each select="actor">
                   <xsl:value-of select="id(@id)/@name"/> <br/>
              </xsl:for-each>
        </td>
     </tr>
    <tr valign="top"><td>Pre Conditions:</td><td><xsl:value-of
 select="preconditions"/></td></tr>
    <tr valign="top"><td>Post Conditions:</td><td><xsl:value-of
 select="postconditions"/></td></tr>

  <xsl:for-each select="scenario">
  <tr><td colspan="2" bgcolor="#cacaca"><b>Scenario: </b><xsl:value-of
 select="@name"/></td></tr>
      <xsl:for-each select="path">
          <tr>
          <td valign="top">
              <xsl:if test="@type = 'alternative'"> Alternative flow</xsl:if>
              <xsl:if test="@type = 'main'">Main flow</xsl:if>
          </td>
          <td valign="top">
              <xsl:if test="@type = 'alternative'">
                <b><xsl:value-of select="@name"/></b><br/>
                <i>Starts  in  step  <xsl:value-of  select="id(@begin)/@index"/>
(<xsl:value-of select="id(@begin)/@name"/>) of <xsl:value-of
 select="id(@begin)/../@name"/></i><br/>
              </xsl:if>

            <xsl:for-each select="step">
                    <b>
                    <xsl:value-of select="format-number(@index,'00')"/> 
                    <xsl:value-of select="@name"/>
                    </b><br/>
```

```
                    <xsl:if test="@type = 'system'">System:</xsl:if>
                    <xsl:if test="@type = 'include'">System:</xsl:if>
                    <xsl:if test="@type = 'actor'">Actor:</xsl:if>
                    <xsl:value-of select="description"/><br/>
                    <xsl:if test="@type = 'include'"> &lt;&lt;include
 <u><xsl:value-of select="@ucid"/> - <xsl:value-of
 select="@ucname"/></u><br/></xsl:if>
                    <br/>
              <xsl:if test="position() = last()">
                <xsl:if test="../@type = 'alternative'">
                    <xsl:if test="@type != 'end'">
                        <i>Return to <xsl:value-of
 select="id(../@return)/@index"/> (<xsl:value-of
 select="id(../@return)/@name"/>) of <xsl:value-of
 select="id(../@return)/../@name"/></i><br/>
                    </xsl:if>
                </xsl:if>
              </xsl:if>

            </xsl:for-each>
          </td>
          </tr>
      </xsl:for-each>
  </xsl:for-each>

  </table>
  <br/>
</xsl:for-each>
  UCDesigner 1.0
</BODY></HTML>
</xsl:template>



</xsl:stylesheet>
```

Arquivo xmlOutput.xsl

Objetivo Gerar uma descrição XML do caso de uso, semelhante à utilizada internamente pela ferramenta.

Conteúdo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >

<!--   UCDesigner   name="xml"   description="XML   Output"   extension="xml"
target="browser" -->

<xsl:output method="txt" indent="yes"/>


<xsl:template match="project">
    <xsl:copy-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Arquivo testCases.xsl

Objetivo Gerar um esqueleto para um plano de testes. O processo gera os possíveis passos de cada cenário de teste.

Conteúdo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
```

```
<!-- UCDesigner name="test-case" description="Test case" extension="html"
target="word" -->

<xsl:output method="html" indent="yes"/>

<xsl:template match="project">
<HTML><BODY>
  <xsl:for-each select="usecases/usecase">
      <table border="1"  width="100%">
      <tr><td colspan="2" bgcolor="#cacaca"><b>Test case
 summary</b></td></tr>
      <tr><td valign="top">Use case index:</td><td><xsl:value-of
 select="@ucid"/></td></tr>
      <tr><td valign="top">Use case name:</td><td><xsl:value-of
 select="@name"/></td></tr>
      <xsl:for-each select="scenario">
          <xsl:for-each select="path">
          <tr><td colspan="2" bgcolor="#cacaca"><b>
                  <xsl:if test="@type = 'alternative'"><xsl:value-of
 select="@name"/></xsl:if>
                  <xsl:if test="@type = 'main'">Main flow</xsl:if>
          </b></td></tr>
          <tr><td colspan="2">
                  <table border="0"  width="100%">
                  <tr><td width="50%"><b>Action</b></td>
<td width="50%"><b>Expected result</b></td></tr>

                  <xsl:if test="@type = 'alternative'">
                    <xsl:call-template name="findBegin">
                        <xsl:with-param name="begin" select="@begin"/>
                    </xsl:call-template>
                  </xsl:if>

                  <xsl:for-each select="step">
                  <tr><td>
                        <xsl:if test="position() = 1">
                            <i>Go to <xsl:value-of
 select="../@name"/></i><br/>
                        </xsl:if>
                       <xsl:value-of select="format-number(@index,'00')"/>
  <xsl:value-of select="@name"/><br/>
                        <xsl:if test="@type = 'end'">
                           <i>Use case ends</i><br/>
                        </xsl:if>
                  </td><td> </td></tr>
                  </xsl:for-each>

                  <xsl:if test="@type = 'alternative'">
                    <xsl:call-template name="findReturn">
                        <xsl:with-param name="return" select="@return"/>
                    </xsl:call-template>
                  </xsl:if>

                  </table>
              </td>
              </tr>
          </xsl:for-each>
      </xsl:for-each>
      </table>
  </xsl:for-each>
  <br/>
  UCDesigner 1.0
</BODY></HTML>
</xsl:template>


<xsl:template name="findBegin">
        <xsl:param name="begin"/>
        <xsl:variable name="targetIndex" select="id($begin)/@index"/>

        <xsl:if test="id($begin)/../@type = 'alternative'">
```

```
            <xsl:call-template name="findBegin">
                <xsl:with-param name="begin" select="id($begin)/../@begin"/>
            </xsl:call-template>
        </xsl:if>

        <xsl:for-each select="id($begin)/../step[@index &lt;= $targetIndex]">
        <tr><td>
            <xsl:if test="position() = 1">
                <i>Go to <xsl:value-of select="../@name"/></i><br/>
            </xsl:if>
            <xsl:value-of select="format-number(@index,'00')"/>
  <xsl:value-of select="@name"/><br/>
        </td><td> </td></tr>
        </xsl:for-each>

</xsl:template>

<xsl:template name="findReturn">
        <xsl:param name="return"/>
        <xsl:variable name="targetIndex" select="id($return)/@index"/>

       <xsl:for-each select="id($return)/../step[@index &gt;= $targetIndex]">
        <tr><td>
            <xsl:if test="position() = 1">
                <i>Return to <xsl:value-of select="../@name"/></i><br/>
            </xsl:if>
            <xsl:value-of select="format-number(@index,'00')"/>
  <xsl:value-of select="@name"/><br/>
            <xsl:if test="@type = 'end'">
                <i>Use case ends</i><br/>
            </xsl:if>
        </td><td> </td></tr>
        </xsl:for-each>

        <xsl:if test="id($return)/../@type = 'alternative'">
            <xsl:call-template name="findReturn">
              <xsl:with-param name="return" select="id($return)/../@return"/>
            </xsl:call-template>
        </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Arquivo javaTestCase.xsl

Objetivo Gerar o esqueleto de uma classe de testes unitários em JAVA.

Conteúdo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >

<!-- UCDesigner name="java-testcase" description="Java test case template"
extension="java" target="text"-->

<xsl:output method="text" indent="yes"/>

<xsl:template match="project">

<xsl:for-each select="usecases/usecase">
/*
 * Class TestCase<xsl:value-of select="translate(normalize-space(@name),
    ' ','')"/>
 * Generated for testing the use case <xsl:value-of select="@name"/>
 */
public class TestCase<xsl:value-of select="translate(normalize-space(@name),
     ' ','')"/> extends TestCase {

    <xsl:for-each select="scenario">
```

```
/*
 * Test for Scenario <xsl:value-of select="@name"/>
 */
public void setupScenario<xsl:value-of
       select="translate(normalize-space(@name),' ','')"/> {
    // TODO: insert your code here for preparing the scenario test
}

public void testScenario<xsl:value-of
         select="translate(normalize-space(@name),' ','')"/> {
    &#32;
    // TODO: insert your code here for starting and finishing
    // the scenario test
    &#32;
    // Testing all flows for the scenario
    <xsl:for-each select="path">
    &#32;
    testScenario<xsl:value-of
               select="translate(normalize-space(../@name),' ','')"/>
     <xsl:value-of select="translate(normalize-space(@name),' ','')"/>();
    </xsl:for-each>
}

public void cleanupScenario<xsl:value-of
             select="translate(normalize-space(@name),' ','')"/> {
    // TODO: insert your code here
}

<xsl:for-each select="path">
/*
 * Test for <xsl:value-of select="@name"/>
 *
 */
public void testScenario<xsl:value-of
          select="translate(normalize-space(../@name),' ','')"/>
    <xsl:value-of select="translate(normalize-space(@name),' ','')"/> {
    // TODO: insert your code here for preparing and finishing the
    // flow test
    &#32;
    <xsl:if test="@type = 'alternative'">
        <xsl:call-template name="findBegin">
            <xsl:with-param name="begin" select="@begin"/>
        </xsl:call-template>
    </xsl:if>

    <xsl:for-each select="step">
        <xsl:if test="position() = 1">
    // Go to <xsl:value-of select="../@name"/>
        </xsl:if>
    runStep<xsl:value-of select="translate(normalize-space(../@name),
       ' ','')"/>
     <xsl:value-of select="translate(normalize-space(@name),' ','')"/>();
    <xsl:if test="@type = 'end'">
    // Use case ends
    </xsl:if>
    </xsl:for-each>

    <xsl:if test="@type = 'alternative'">
        <xsl:call-template name="findReturn">
            <xsl:with-param name="return" select="@return"/>
        </xsl:call-template>
    </xsl:if>
}
&#32;
&#32;
</xsl:for-each>
</xsl:for-each>

/*
 * Steps for use case <xsl:value-of select="@name"/>
 */
```

```
    <xsl:for-each select="scenario/path/step">
    public void runStep<xsl:value-of
            select="translate(normalize-space(../@name),' ','')"/>
       <xsl:value-of select="translate(normalize-space(@name),' ','')"/>() {
        //TODO: insert your code here for implementing this step
    }
    &#32;
    </xsl:for-each>
}
</xsl:for-each>
// Generated by UCDesigner 1.0
</xsl:template>

<xsl:template name="findBegin">
        <xsl:param name="begin"/>
        <xsl:variable name="targetIndex" select="id($begin)/@index"/>

        <xsl:if test="id($begin)/../@type = 'alternative'">
            <xsl:call-template name="findBegin">
                <xsl:with-param name="begin" select="id($begin)/../@begin"/>
            </xsl:call-template>
        </xsl:if>

        <xsl:for-each select="id($begin)/../step[@index &lt;= $targetIndex]">
            <xsl:if test="position() = 1">
        // Go to <xsl:value-of select="../@name"/>
            </xsl:if>
        runStep<xsl:value-of select="translate(normalize-space(../@name),
                 ' ','')"/>
         <xsl:value-of select="translate(normalize-space(@name),' ','')"/>();
        </xsl:for-each>

</xsl:template>

<xsl:template name="findReturn">
        <xsl:param name="return"/>
        <xsl:variable name="targetIndex" select="id($return)/@index"/>

        <xsl:for-each
              select="id($return)/../step[@index &gt;= $targetIndex]">
            <xsl:if test="position() = 1">
        // Return to <xsl:value-of select="../@name"/>
            </xsl:if>
        runStep<xsl:value-of select="translate(normalize-space(../@name),
           ' ','')"/>
        <xsl:value-of select="translate(normalize-space(@name), ' ','')"/>();
            <xsl:if test="@type = 'end'">
        // Use case ends
            </xsl:if>
        </xsl:for-each>

        <xsl:if test="id($return)/../@type = 'alternative'">
            <xsl:call-template name="findReturn">
                <xsl:with-param name="return"
                      select="id($return)/../@return"/>
            </xsl:call-template>
        </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Arquivo logicalReferences.xsl

Objetivo Gerar um documento com os mapeamentos das dependências entre o modelo de descrição de uso e o modelo de classes do sistema.

Conteúdo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
```

```
  >

<!--  UCDesigner  name="logical-reference"  description="Logical  reference"
extension="doc" target="word" -->

<xsl:output method="html" indent="yes"/>

<xsl:key name="modelref-by-class" match="modelref" use="@class" />


<xsl:template match="project">
<HTML><BODY>
<xsl:for-each select="usecases/usecase">
  <table border="1"  width="100%">
  <tr><td colspan="2" bgcolor="#cacaca"><b>Use case summary</b></td></tr>
  <tr><td valign="top">Use case index:</td><td>
            <xsl:value-of select="@ucid"/></td></tr>
  <tr><td valign="top">Use case name:</td><td>
            <xsl:value-of select="@name"/></td></tr>
  <tr><td valign="top">Description:</td><td>
            <xsl:value-of select="description"/></td></tr>

        <xsl:for-each select="scenario/path/step/modelref[
                  count(. | key('modelref-by-class', @class)[1]) = 1]">
              <xsl:sort select="@class" />

               <tr><td colspan="2" bgcolor="#cacaca"><b>References to class:
                  </b><xsl:value-of select="@class" /></td></tr>
              <xsl:for-each select="key('modelref-by-class', @class)">
              <xsl:sort select="@type" />
                   <tr><td colspan="2">
                       <b><xsl:value-of select="../@name"/>,
                       step <xsl:value-of select="../@index"/>
                       of <xsl:value-of select="../../@name"/></b><br/>
                     <xsl:value-of select="../description"/><br/>
                       <i><xsl:value-of select="@type"/> 
                        <xsl:value-of select="@class"/></i>
                   </td></tr>
              </xsl:for-each>
        </xsl:for-each>
  </table>
  <br/>
</xsl:for-each>
  UCDesigner 1.0
</BODY></HTML>
</xsl:template>



</xsl:stylesheet>
```

Anexo 2 – Exemplos de descrições com UCDesigner

Nesta seção serão apresentadas as descrições de casos de uso da ferramenta UCDesigner utilizando a própria ferramenta para a construção das descrições. Entretanto, as figuras 56 e 57 foram construídas com outra ferramenta CASE de UML chamada Enterprise Architect. Na primeira pode-se verificar o modelo de caso de uso que apresenta os atores envolvidos no sistema.

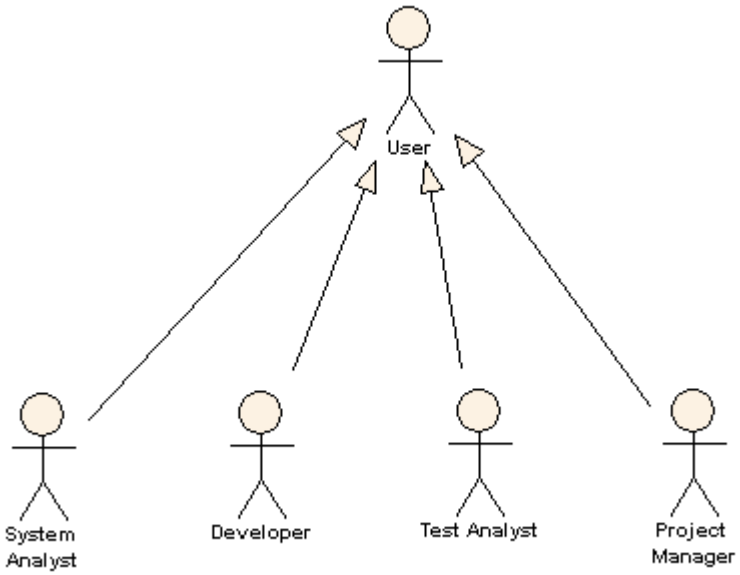


Figura 56 - Atores do sistema

Note-se que o objetivo da ferramenta UCDesigner é ser utilizada por diferentes participantes do processo de desenvolvimento, tais como o Analista de Sistemas, o Desenvolvedor, o Analista de Testes e o Gerente do Projeto. Os casos de uso com os quais estes atores interagem são os seguintes:

- UC001 – Enter the tool
- UC002 – View Project
- UC003 – Import Classes from Folder
- UC004 – Add Class
- UC005 – Add Use Case
- UC006 – Describe Scenario
- UC007 – Add Step Information
- UC008 – Associate with Logical Model
- UC009 – Export Description from Template
- UC010 – Compile Use Case
- UC011 – View Project Tree
- UC012 – Open Use Case
- UC013 – Display Compile Message

Na figura 57, pode-se observar o modelo de caso de uso principal.

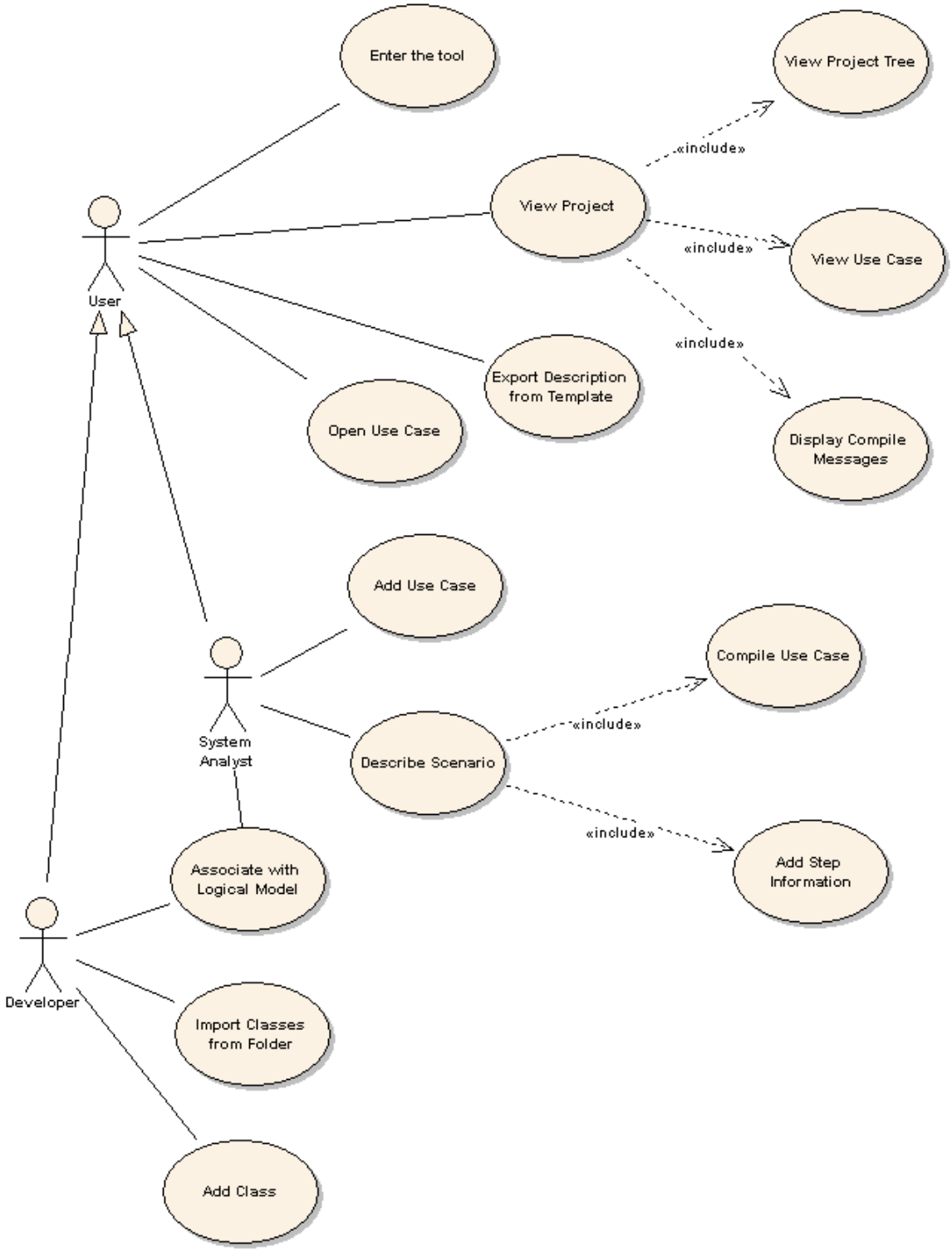
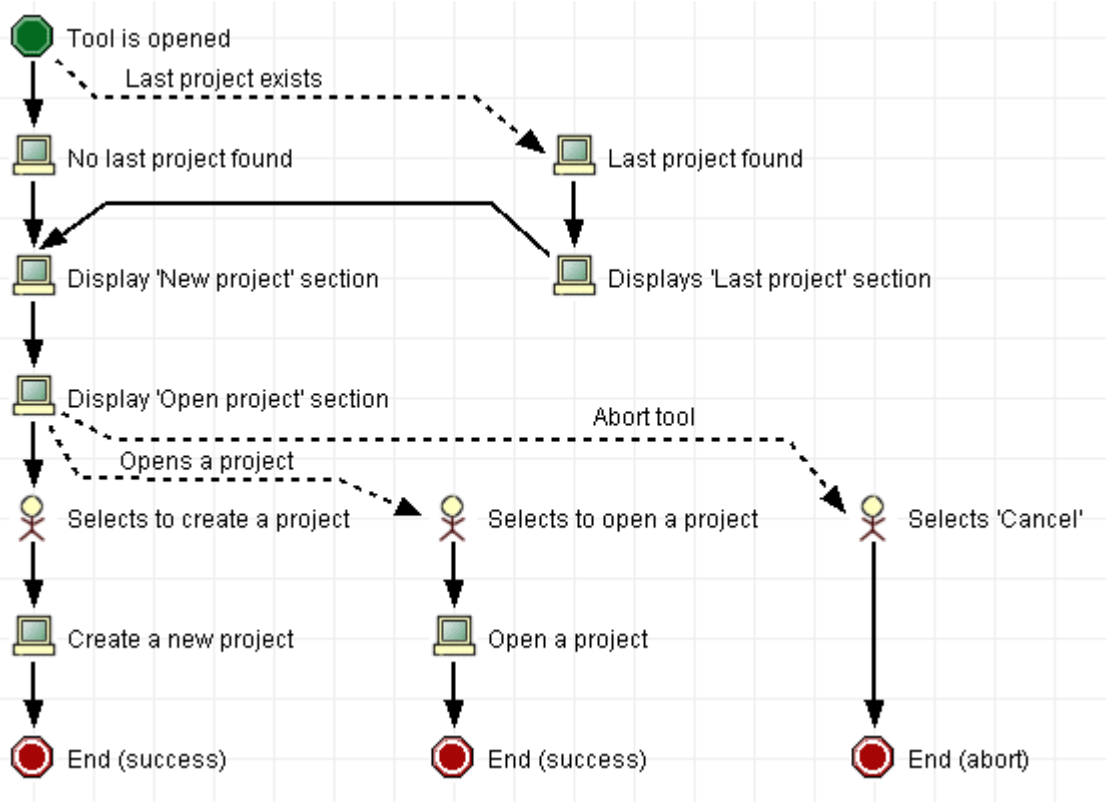


Figura 57 - Modelo de casos de uso do sistema

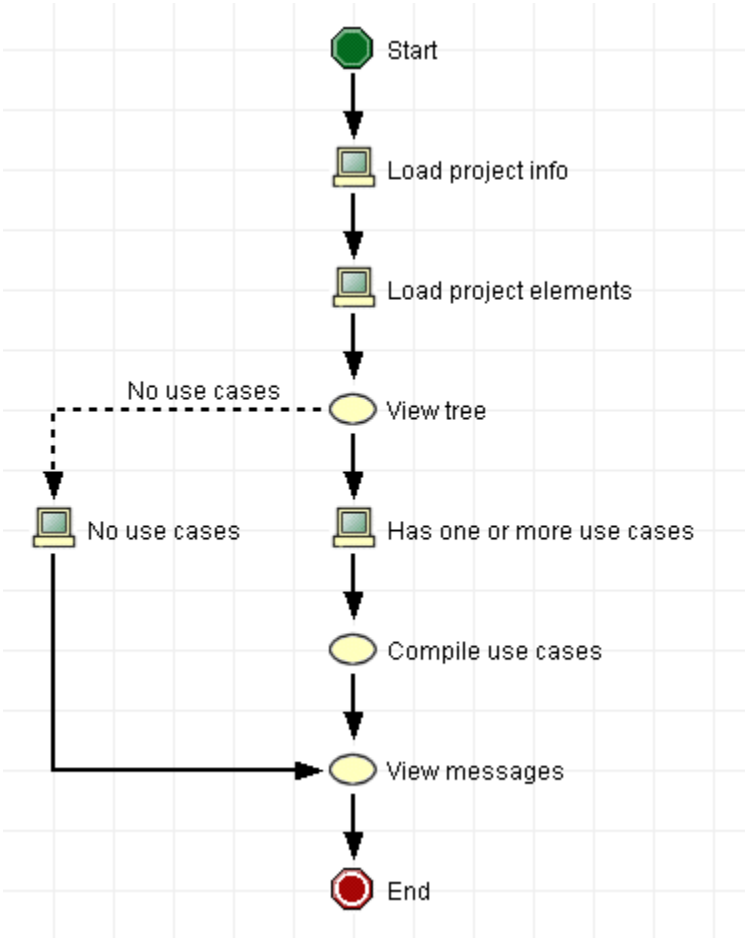
Descrições construídas com a ferramenta

Use case: UC001 – Enter the tool



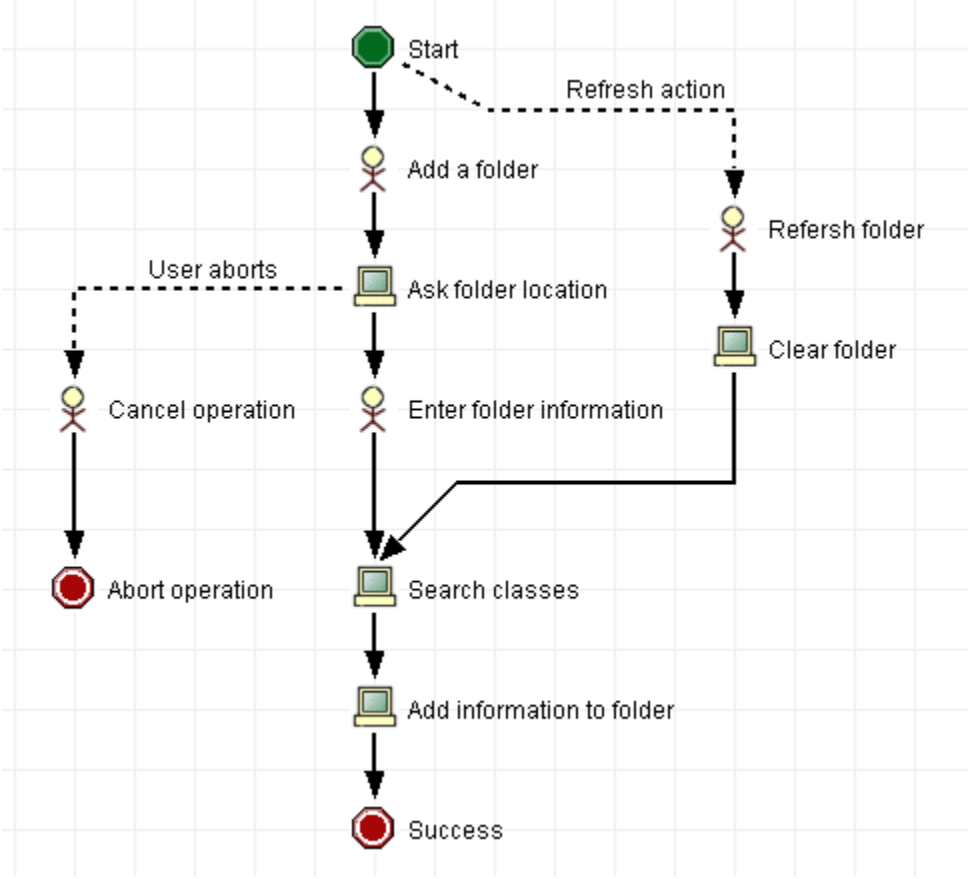
Descrições contruídas com a ferramenta

Use case: UC002 – View Project



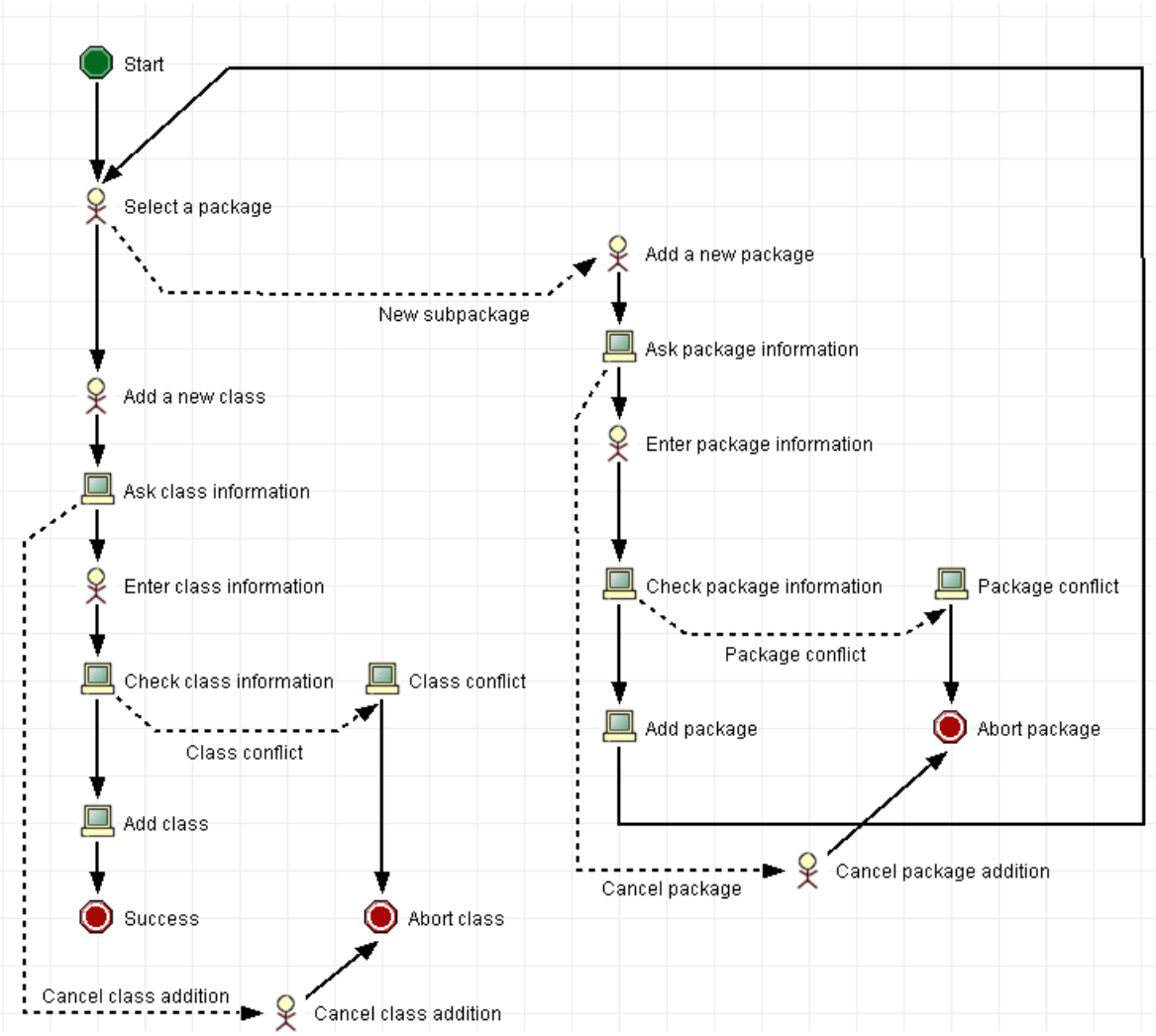
Descrições contruídas com a ferramenta

Use case: UC003 – Import Classes From Folder



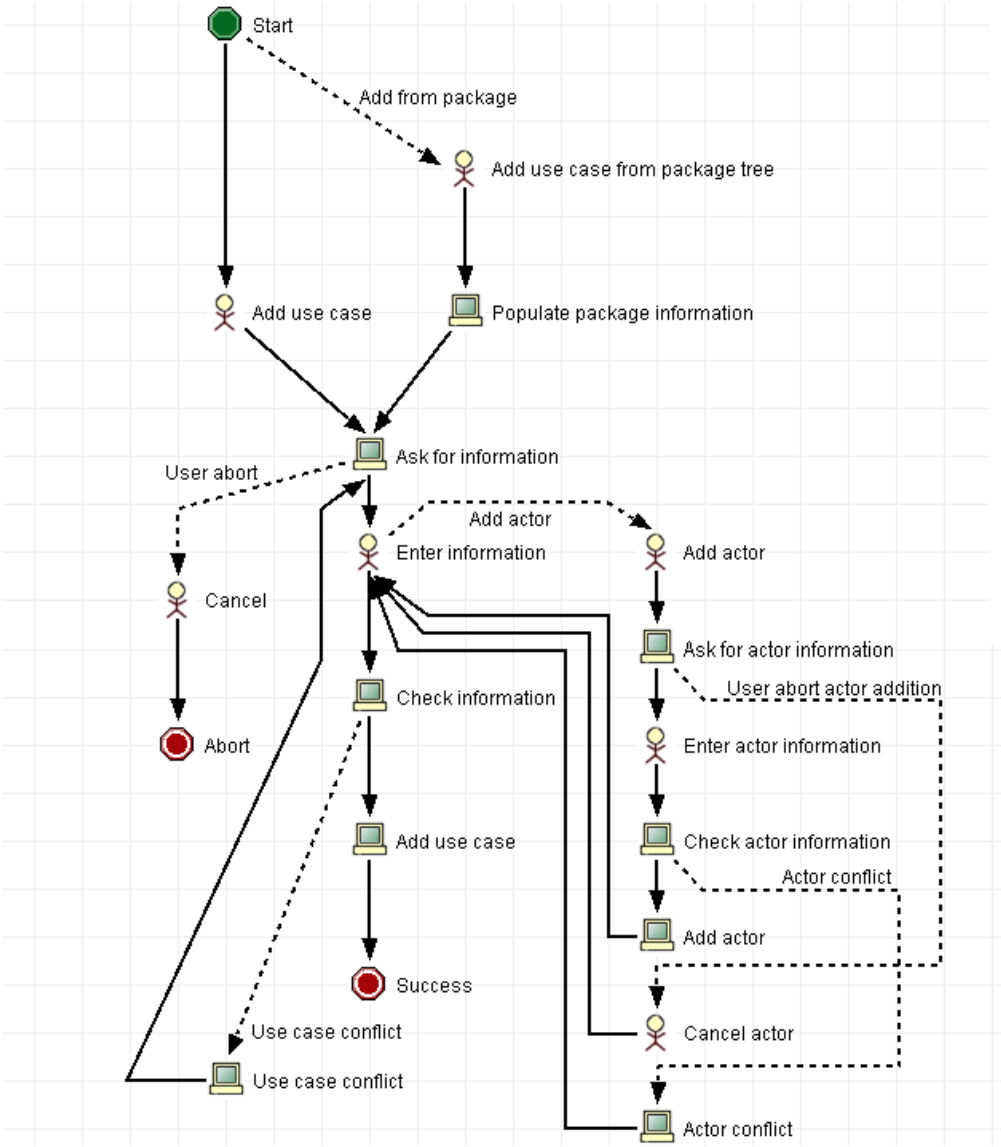
Descrições contruídas com a ferramenta

Use case: UC004 – Add Class



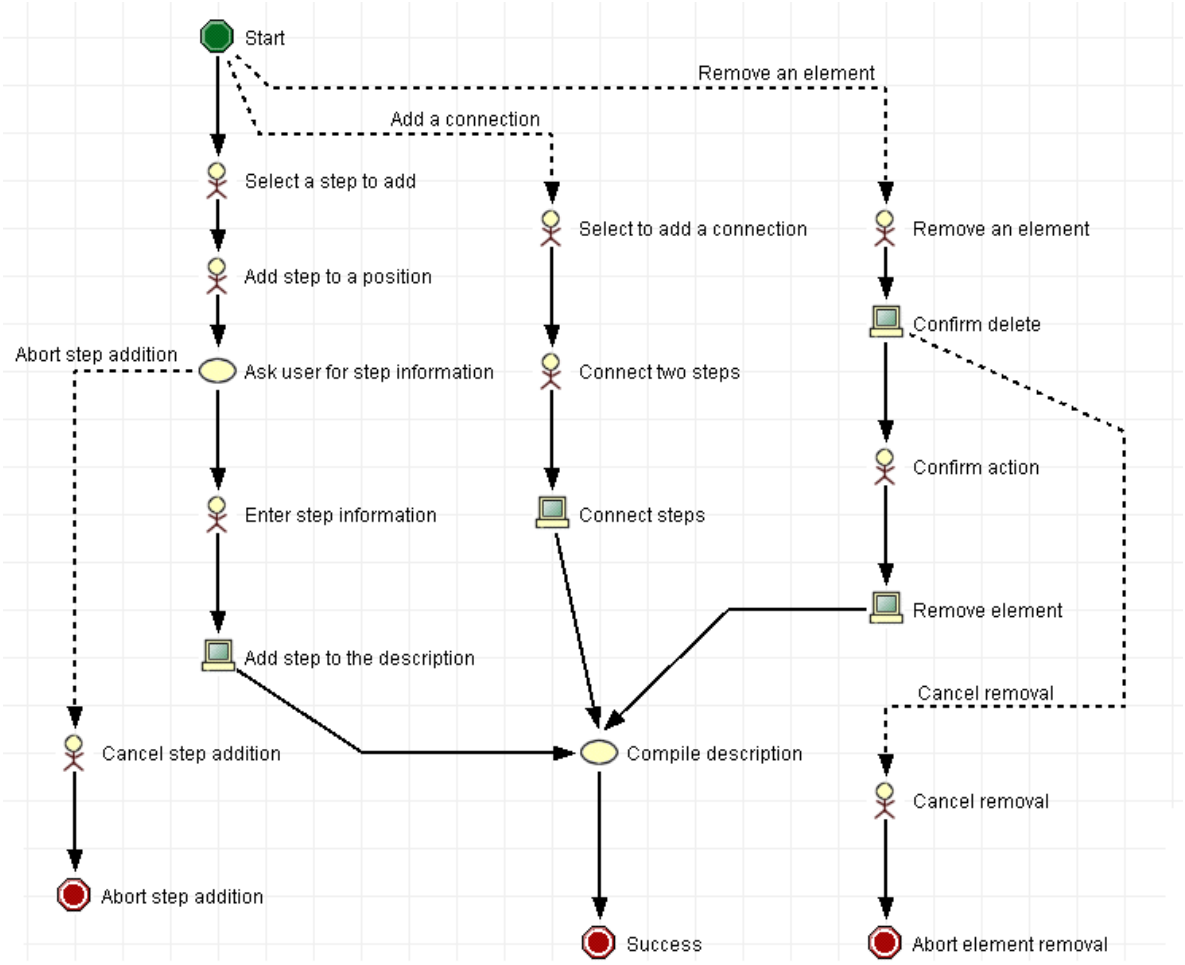
Descrições contruídas com a ferramenta

Use case: UC005 – Add Use Case



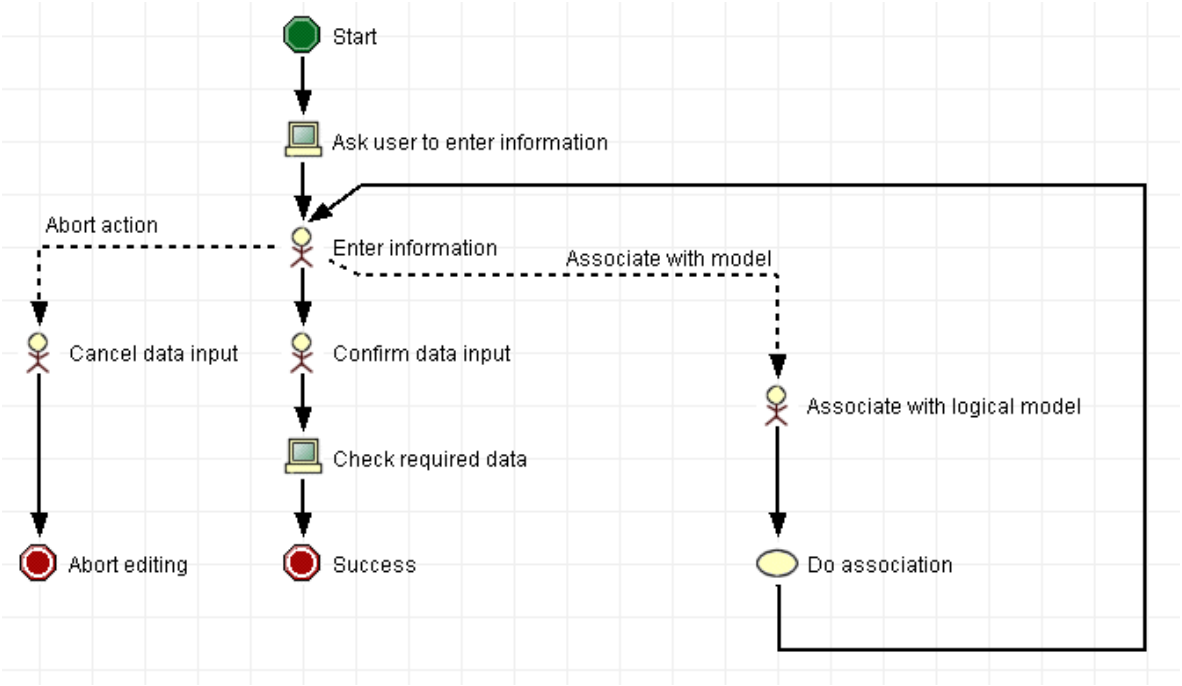
Descrições contruídas com a ferramenta

Use case: UC006 – Describe Scenario

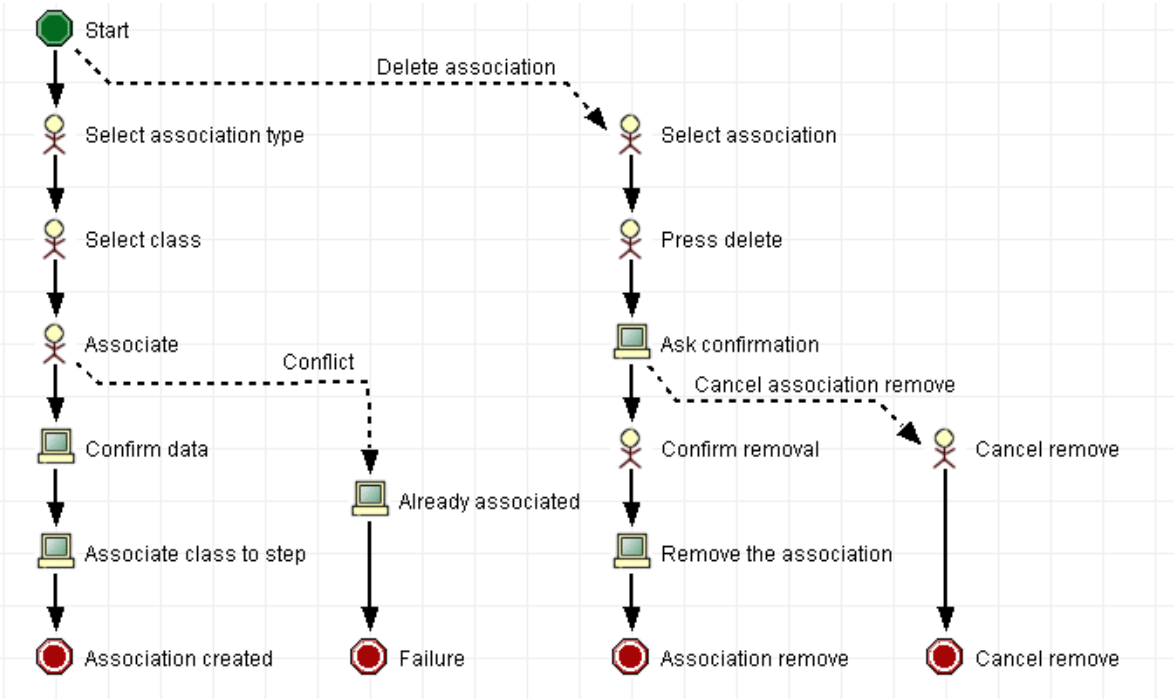


Descrições contruídas com a ferramenta

Use case: UC007– Add Sep Information

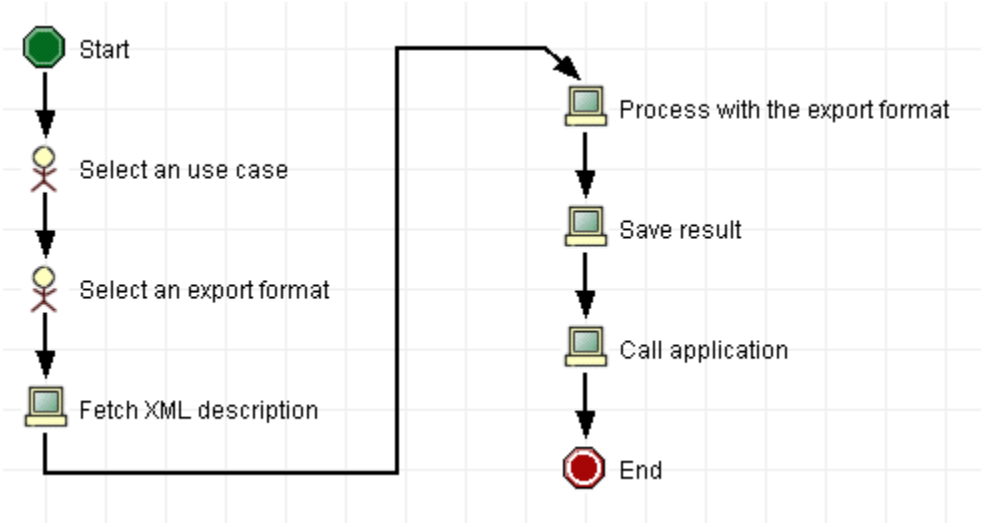


Use case: UC008 – Associate with Logical Model

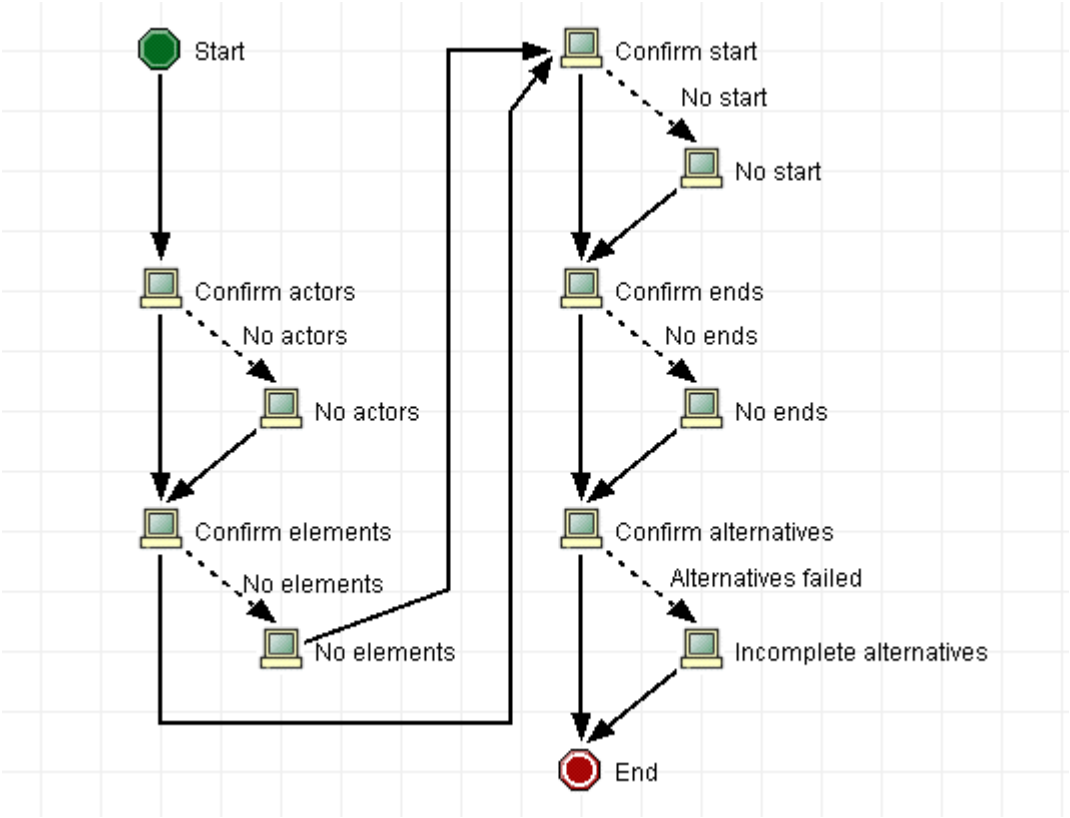


Descrições contruídas com a ferramenta

Use case: UC009 – Export Description from Template

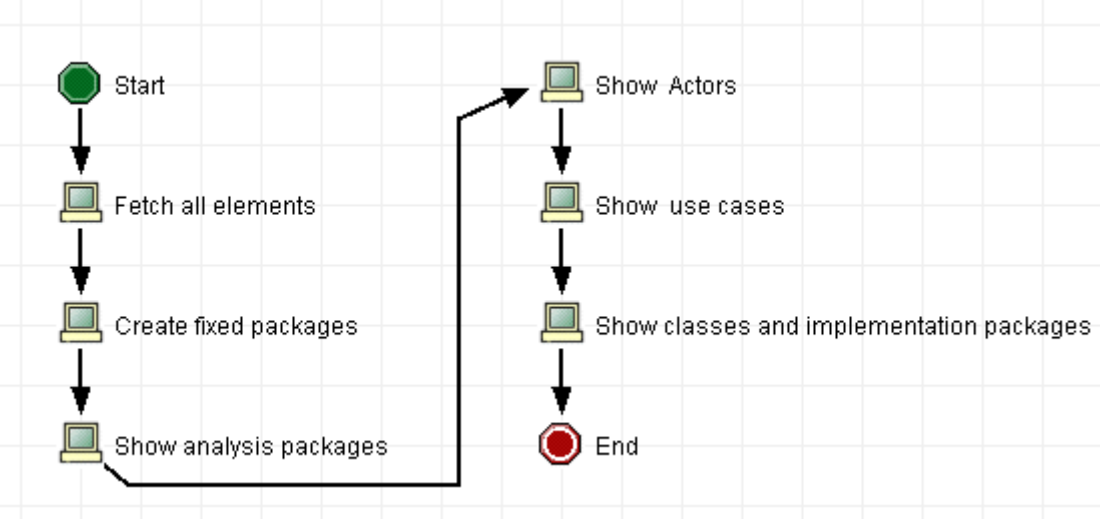


Use case: UC010 – Compile Use Case

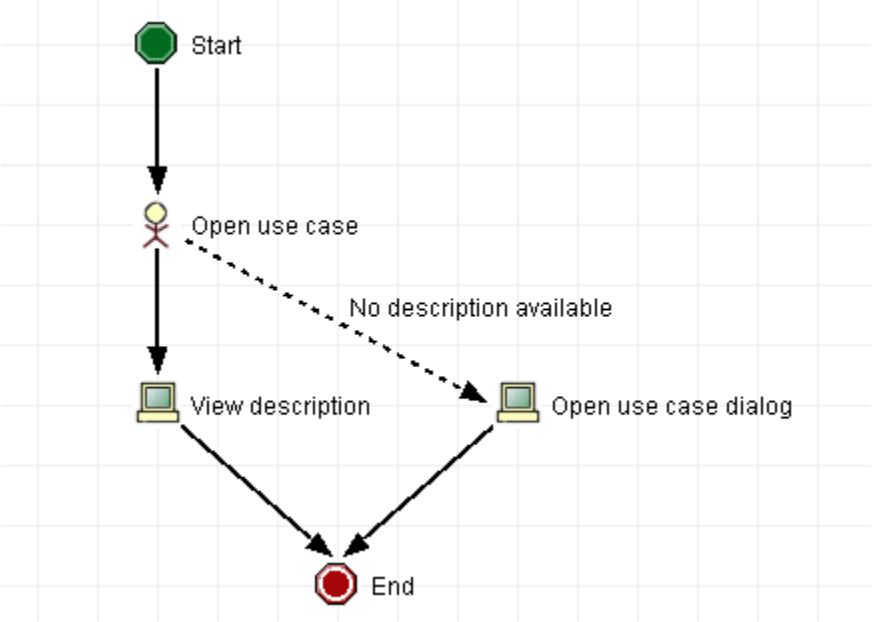


Descrições contruídas com a ferramenta

Use case: UC011 – View Project Tree

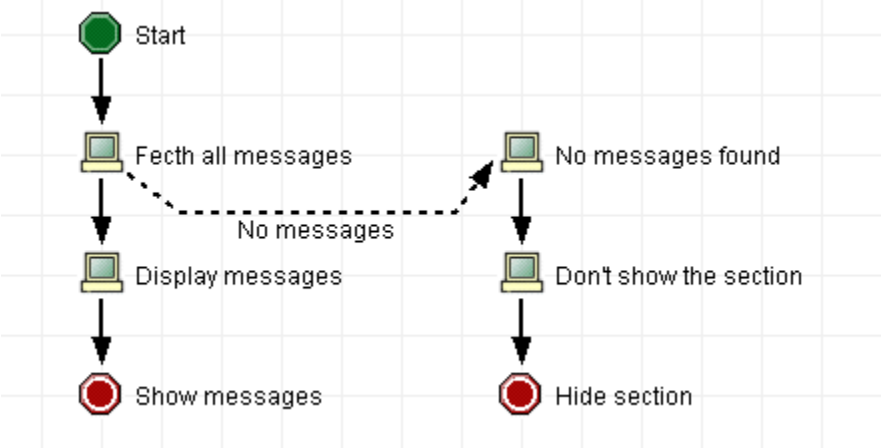


Use case: UC012 – Open Use Case



Descrições contruídas com a ferramenta

Use case: UC013 – Display Compile Message



Descrições XML geradas (xmlOutput.xsl)

Use case: UC001 – Enter the tool

```
<?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
- <actors>
  <actor id="18" name="System" />
  <actor id="5" name="User" />
  <actor id="4" name="Test analyst" parent="5" />
  <actor id="3" name="System analyst" parent="5" />
  <actor id="2" name="Project manager" parent="5" />
  <actor id="12" name="Developer" parent="5" />
    </actors>
- <usecases>
- <usecase id="9" ucid="UC001" name="Enter the tool">
  <description>The user enter for the first time in the tool.</description>
  <preconditions>No other instance of the tool is opened.</preconditions>
  <actor id="5" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="25" name="Tool is opened" index="1" type="start">
  <description>The tool is opened</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
- <step id="26" name="No last project found" index="2" type="system">
  <description>System checks that no last project is found.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
- <step id="28" name="Display 'New project' section" index="3" type="system">
  <description>Allow the user to enter a new project, showing the 'New project' section.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
- <step id="29" name="Display 'Open project' section" index="4" type="system">
  <description>Allows the user to search and open another project, showing the 'Open project'
      section.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
- <step id="32" name="Selects to create a project" index="5" type="actor">
  <description>Selects the option to create a new project and selects 'OK'.</description>
    </step>
- <step id="33" name="Create a new project" index="6" type="system">
  <description>Creates a new empty project with the provided information.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
    </step>
- <step id="36" name="End (success)" index="7" type="end">
  <description>Close the start window and go to the main window with the new created project
      opened.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
    </path>
- <path type="alternative" name="Last project exists" begin="25" return="28">
- <step id="38" name="Last project found" index="1" type="system">
  <description>Checks that there is a last project and that the file exists.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
- <step id="40" name="Displays 'Last project' section" index="2" type="system">
  <description>Allows user to select the last project option. Provides information about the last project (name
      and file).</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
    </path>
- <path type="alternative" name="Opens a project" begin="29">
```

Descrições XML geradas (xmlOutput.xsl)

```
- <step id="43" name="Selects to open a project" index="1" type="actor">
  <description>Selects to open an existent project (last project or open project) and selects 'OK'.</description>
    </step>
- <step id="44" name="Open a project" index="2" type="system">
  <description>Opens an existent project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
    </step>
- <step id="47" name="End (success)" index="3" type="end">
  <description>The start screen is closed and the main screen is loaded with the opened project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JStartProject" />
    </step>
    </path>
- <path type="alternative" name="Abort tool" begin="29">
- <step id="49" name="Selects 'Cancel'" index="1" type="actor">
  <description>Selects to abort the operation and close the tool.</description>
    </step>
- <step id="51" name="End (abort)" index="2" type="end">
  <description>The tool is closed.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  </step>
  </path>
  </scenario>
  </usecase>
  </usecases>
  </project>
```

Use case: UC002 – View Project

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
- <actors>
  <actor id="18" name="System" />
  <actor id="5" name="User" />
  <actor id="4" name="Test analyst" parent="5" />
  <actor id="3" name="System analyst" parent="5" />
  <actor id="2" name="Project manager" parent="5" />
  <actor id="12" name="Developer" parent="5" />
    </actors>
- <usecases>
- <usecase id="10" ucid="UC002" name="View Project">
  <description>View a project after entering the tool or opening a new project.</description>
  <preconditions>The user already selected an existent project or created a new one.</preconditions>
  <postconditions>The project is displayed.</postconditions>
  <actor id="5" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="53" name="Start" index="1" type="start">
  <description>The user entered the tool and now need to view the project (new or already existent).</description>
    </step>
- <step id="54" name="Load project info" index="2" type="system">
  <description>Load all project information (author, last update date, description, etc).</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.Project" />
    </step>
- <step id="55" name="Load project elements" index="3" type="system">
  <description>Load use cases, use case descriptions, actors, packages and classes.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.xml.XMLProject" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.GraphicRepresentation" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.ExtensionPoint" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCaseDescription" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCaseElement" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
```

Descrições XML geradas (xmlOutput.xsl)

```
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
    </step>
- <step id="58" name="View tree" index="4" type="include" ucid="UC011" ucname="View project tree">
  <description>View tree for the elements loaded.</description>
    </step>
- <step id="60" name="Has one or more use cases" index="5" type="system">
  <description>Confirms that one or more use cases exist in the project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
    </step>
- <step id="62" name="Compile use cases" index="6" type="include" ucid="UC010" ucname="Compile use case">
  <description>Compile use cases of the project.</description>
    </step>
- <step id="64" name="View messages" index="7" type="include" ucid="UC013" ucname="Display compile
      messages">
  <description>Display all existent messages for the project.</description>
    </step>
- <step id="69" name="End" index="8" type="end">
  <description>The project is displayed.</description>
    </step>
    </path>
- <path type="alternative" name="No use cases" begin="58" return="64">
- <step id="66" name="No use cases" index="1" type="system">
  <description>Confirm that no use cases were entered for the project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```

Use case: UC003 – Import Classes From Folder

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="11" ucid="UC003" name="Import classes from folder">
  <description>Allow to import existent JAVA classes form a folder.</description>
  <actor id="12" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="71" name="Start" index="1" type="start">
  <description>The user will import classes from a folder for the project</description>
    </step>
- <step id="72" name="Add a folder" index="2" type="actor">
  <description>Selects to add a new folder for JAVA classes.</description>
    </step>
- <step id="73" name="Ask folder location" index="3" type="system">
  <description>Ask the user to select the folder from the file system</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
    </step>
- <step id="76" name="Enter folder information" index="4" type="actor">
  <description>Select folder from the file system and confirms</description>
    </step>
- <step id="82" name="Search classes" index="5" type="system">
  <description>Searches classes inside the folder. The operation also includes all subfolders. All java files are
      examined and all information for classes are fetched (class name, methods and attributes).</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JJavaFolder" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.java.RefreshJavaFolder" />
  <modelref type="Used by" class="br.edu.ufrgs.ucdesigner.model.logical.Attribute" />
  <modelref type="Used by" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
  <modelref type="Used by" class="br.edu.ufrgs.ucdesigner.model.logical.Method" />
    </step>
- <step id="89" name="Add information to folder" index="6" type="system">
```

Descrições XML geradas (xmlOutput.xsl)

```
  <description>All information fetched are added to the project as new elements.</description>
      </step>
- <step id="91" name="Success" index="7" type="end">
  <description>The folder in the project contains all classes found in the file system.</description>
      </step>
      </path>
- <path type="alternative" name="User aborts" begin="73">
- <step id="79" name="Cancel operation" index="1" type="actor">
  <description>Select to cancel the operation of adding a folder to the project</description>
      </step>
- <step id="78" name="Abort operation" index="2" type="end">
  <description>No folder will be added</description>
      </step>
      </path>
- <path type="alternative" name="Refresh action" begin="71" return="82">
- <step id="84" name="Refersh folder" index="1" type="actor">
  <description>Refresh existent JAVA folder</description>
      </step>
- <step id="86" name="Clear folder" index="2" type="system">
  <description>Clear all folder contents</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JJavaFolder" />
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```

Descrições XML geradas (xmlOutput.xsl)

Use case: UC004 – Add Class

```
<?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="13" ucid="UC004" name="Add class">
  <description>Allows user to add a class manually.</description>
  <actor id="12" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="93" name="Start" index="1" type="start">
  <description>The user needs to add a class manually.</description>
    </step>
- <step id="94" name="Select a package" index="2" type="actor">
  <description>Select an existent package for adding the class.</description>
    </step>
- <step id="99" name="Add a new class" index="3" type="actor">
  <description>Selects to add a new class to the selected package</description>
    </step>
- <step id="118" name="Ask class information" index="4" type="system">
  <description>Ask the user for class information</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JClass" />
    </step>
- <step id="120" name="Enter class information" index="5" type="actor">
  <description>Enter class information (class name, package, description, stereotype, methods and
      attributes).</description>
    </step>
- <step id="122" name="Check class information" index="6" type="system">
  <description>Confirms that no other class exists with the same name.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JClass" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
    </step>
- <step id="124" name="Add class" index="7" type="system">
  <description>Add the class to the project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JClass" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Attribute" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Method" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Stereotype" />
    </step>
- <step id="126" name="Success" index="8" type="end">
  <description>The user added a new class manually.</description>
    </step>
    </path>
- <path type="alternative" name="New subpackage" begin="94" return="94">
- <step id="96" name="Add a new package" index="1" type="actor">
  <description>Selects to add another package</description>
    </step>
- <step id="104" name="Ask package information" index="2" type="system">
  <description>Ask the user for package information</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JPackage" />
    </step>
- <step id="106" name="Enter package information" index="3" type="actor">
  <description>Enter package name and parent.</description>
    </step>
- <step id="108" name="Check package information" index="4" type="system">
  <description>Check if no other package exists with the same name.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JPackage" />
    </step>
- <step id="114" name="Add package" index="5" type="system">
  <description>Add package to project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
    </step>
    </path>
```

Descrições XML geradas (xmlOutput.xsl)

```
– <path type="alternative" name="Package conflict" begin="108">
– <step id="110" name="Package conflict" index="1" type="system">
  <description>Anotger package with the same name exists for the package parent. Show a message.</description>
      </step>
– <step id="112" name="Abort package" index="2" type="end">
  <description>No package is added</description>
      </step>
      </path>
– <path type="alternative" name="Class conflict" begin="122">
– <step id="128" name="Class conflict" index="1" type="system">
  <description>Confirms that other class already exist with the same name. Display a message.</description>
      </step>
– <step id="130" name="Abort class" index="2" type="end">
  <description>The class will not be added.</description>
      </step>
      </path>
– <path type="alternative" name="Cancel package" begin="104" return="112">
– <step id="134" name="Cancel package addition" index="1" type="actor">
  <description>The user cancel the addition of the package</description>
      </step>
      </path>
– <path type="alternative" name="Cancel class addition" begin="118" return="130">
– <step id="137" name="Cancel class addition" index="1" type="actor">
  <description>Selects to cancel the class addition.</description>
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```

Use case: UC005 – Add Use Case

```
    <?xml version="1.0" encoding="UTF-8" ?>
– <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
– <usecases>
– <usecase id="14" ucid="UC005" name="Add use case">
  <description>User enters a new use case.</description>
  <actor id="3" />
– <scenario name="Main scenario">
– <path type="main" name="Main flow">
– <step id="140" name="Start" index="1" type="start">
  <description>User want to add a new use case</description>
      </step>
– <step id="141" name="Add use case" index="2" type="actor">
  <description>Select to add use case from menu</description>
      </step>
– <step id="143" name="Ask for information" index="3" type="system">
  <description>Ask the user for use case information</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JUseCase" />
      </step>
– <step id="150" name="Enter information" index="4" type="actor">
  <description>Enter information for the use case</description>
      </step>
– <step id="156" name="Check information" index="5" type="system">
  <description>Confirms that no other use case exists with the same use case ID or use case name.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JUseCase" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
      </step>
– <step id="158" name="Add use case" index="6" type="system">
  <description>The use case is added to the project.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
      </step>
```

Descrições XML geradas (xmlOutput.xsl)


```
- <step id="160" name="Success" index="7" type="end">
  <description>The use case is successfuly created.</description>
      </step>
      </path>
- <path type="alternative" name="Add from package" begin="140" return="143">
- <step id="145" name="Add use case from package tree" index="1" type="actor">
  <description>User select a particular package and selects to add an use case.</description>
      </step>
- <step id="147" name="Populate package information" index="2" type="system">
  <description>Pre-populate the pacjage information selecting the package that user choosed.</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JUseCase" />
      </step>
      </path>
- <path type="alternative" name="User abort" begin="143">
- <step id="152" name="Cancel" index="1" type="actor">
  <description>Selects to cancel the use case addition</description>
      </step>
- <step id="154" name="Abort" index="2" type="end">
  <description>No use case is added.</description>
      </step>
      </path>
- <path type="alternative" name="Add actor" begin="150" return="150">
- <step id="162" name="Add actor" index="1" type="actor">
  <description>Selects to add an actor and associate it with the use case information.</description>
      </step>
- <step id="164" name="Ask for actor information" index="2" type="system">
  <description>Ask the user to enter more information to create the actor.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JActor" />
      </step>
- <step id="166" name="Enter actor information" index="3" type="actor">
  <description>Enter actor information to create the actor.</description>
      </step>
- <step id="168" name="Check actor information" index="4" type="system">
  <description>Confirms that no conflict exists with the actor name.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JActor" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
      </step>
- <step id="170" name="Add actor" index="5" type="system">
  <description>The actor is added to the project and associated to the use case.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.project.Project" />
      </step>
      </path>
- <path type="alternative" name="User abort actor addition" begin="164" return="150">
- <step id="173" name="Cancel actor" index="1" type="actor">
  <description>Cancel the actor addition</description>
      </step>
      </path>
- <path type="alternative" name="Actor conflict" begin="168" return="150">
- <step id="176" name="Actor conflict" index="1" type="system">
  <description>Confirms that other actor exists with the same name. Display a message.</description>
      </step>
      </path>
- <path type="alternative" name="Use case conflict" begin="156" return="143">
- <step id="179" name="Use case conflict" index="1" type="system">
  <description>Confirms that other use case exists with the same use case ID or use case name. Display a
      message.</description>
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```


Use case: UC006 – Describe Scenario


```
<xml version="1.0" encoding="UTF-8" ?>
```

Descrições XML geradas (xmlOutput.xsl)


```
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="15" ucid="UC006" name="Describe scenario">
  <description>Use can describe the scenario using activity diagram.</description>
  <actor id="3" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="182" name="Start" index="1" type="start">
  <description>User will enter the use case description for a particular use case.</description>
    </step>
- <step id="183" name="Select a step to add" index="2" type="actor">
  <description>Select a step to add to the description.</description>
    </step>
- <step id="184" name="Add step to a position" index="3" type="actor">
  <description>Click on the place the step will be added.</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel" />
    </step>
- <step id="185" name="Ask user for step information" index="4" type="include" ucid="UC007" ucname="Add step
      information">
  <description>Ask the user for more step information, considering the type of the step.</description>
    </step>
- <step id="189" name="Enter step information" index="5" type="actor">
  <description>User enter all information necessary for the step and press "OK".</description>
    </step>
- <step id="191" name="Add step to the description" index="6" type="system">
  <description>The step is added to the use case description.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.GraphicRepresentation" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
    </step>
- <step id="206" name="Compile description" index="7" type="include" ucid="UC010" ucname="Compile use case">
  <description>Compile the use case description based on the updated data.</description>
    </step>
- <step id="193" name="Success" index="8" type="end">
  <description>The new step is added to the use case description.</description>
    </step>
    </path>
- <path type="alternative" name="Abort step addition" begin="185">
- <step id="196" name="Cancel step addition" index="1" type="actor">
  <description>Selects to cancel step addition</description>
    </step>
- <step id="198" name="Abort step addition" index="2" type="end">
  <description>The step will not be added.</description>
    </step>
    </path>
- <path type="alternative" name="Add a connection" begin="182" return="206">
- <step id="200" name="Select to add a connection" index="1" type="actor">
  <description>Select to add a connection between two steps.</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel" />
    </step>
- <step id="202" name="Connect two steps" index="2" type="actor">
  <description>Connect two steps by clicking on the origin step and clicking on the destination step.</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
    </step>
- <step id="204" name="Connect steps" index="3" type="system">
  <description>Connect the steps.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.graphic.GraphicConnection" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.graphic.GraphicElement" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.graphic.GraphicComponent" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.GraphicRepresentation" />
    </step>
    </path>
- <path type="alternative" name="Remove an element" begin="182" return="206">
- <step id="210" name="Remove an element" index="1" type="actor">
  <description>Remove an element from the description (could be a step or a connection).</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel" />
    </step>
- <step id="212" name="Confirm delete" index="2" type="system">
```

Descrições XML geradas (xmlOutput.xsl)

```
  <description>Ask the user to confirm the element removal.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog" />
     </step>
- <step id="214" name="Confirm action" index="3" type="actor">
  <description>Confirm to delete the element.</description>
     </step>
- <step id="216" name="Remove element" index="4" type="system">
  <description>Remove the element from the use case description.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.GraphicRepresentation" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.ObjectRepository" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.graphic.GraphicElement" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.graphic.GraphicConnection" />
     </step>
     </path>
- <path type="alternative" name="Cancel removal" begin="212">
- <step id="219" name="Cancel removal" index="1" type="actor">
  <description>Cancel the element removal.</description>
     </step>
- <step id="220" name="Abort element removal" index="2" type="end">
  <description>No element is removed from the description.</description>
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```

Use case: UC007– Add Sep Information

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="16" ucid="UC007" name="Add step information">
  <description>User enters specific step information</description>
  <actor id="3" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="223" name="Start" index="1" type="start">
  <description>User selected to add a new step</description>
     </step>
- <step id="224" name="Ask user to enter information" index="2" type="system">
  <description>Ask the user to enter information accordingly to the type of the step.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog" />
     </step>
- <step id="225" name="Enter information" index="3" type="actor">
  <description>Enter the information required.</description>
     </step>
- <step id="230" name="Confirm data input" index="4" type="actor">
  <description>Confirm data input by pressing the "OK" button.</description>
     </step>
- <step id="226" name="Check required data" index="5" type="system">
  <description>Confirm that all the required data was entered correctly.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog" />
     </step>
- <step id="233" name="Success" index="6" type="end">
  <description>Return the step with the new information provided.</description>
     </step>
     </path>
- <path type="alternative" name="Abort action" begin="225">
- <step id="235" name="Cancel data input" index="1" type="actor">
  <description>Cancel data input by clicking on the "Cancel" button.</description>
     </step>
- <step id="237" name="Abort editing" index="2" type="end">
  <description>Abort step information editing.</description>
     </step>
     </path>
- <path type="alternative" name="Associate with model" begin="225" return="225">
```

Descrições XML geradas (xmlOutput.xsl)


```
– <step id="239" name="Associate with logical model" index="1" type="actor">
  <description>Selects to associate the step with the logical model.</description>
    </step>
– <step id="241" name="Do association" index="2" type="include" ucid="UC008" ucname="Associate with Logical
    Model">
  <description>Do the association between the step and the logical model.</description>
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```


Use case: UC008 – Associate with Logical Model


```
   <?xml version="1.0" encoding="UTF-8" ?>
– <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
– <usecases>
– <usecase id="17" ucid="UC008" name="Associate with Logical Model">
  <description>Associate use case step with the logical model.</description>
  <actor id="12" />
  <actor id="3" />
– <scenario name="Main scenario">
– <path type="main" name="Main flow">
– <step id="245" name="Start" index="1" type="start">
  <description>Within the step information is possible to associate the step with the logical model.</description>
    </step>
– <step id="246" name="Select association type" index="2" type="actor">
  <description>Select association type within the options "Implemented by", "Depends on" and "Used
    by".</description>
    </step>
– <step id="248" name="Select class" index="3" type="actor">
  <description>Select class from the model</description>
    </step>
– <step id="250" name="Associate" index="4" type="actor">
  <description>Select option to associate the selected class and association type to the step.</description>
    </step>
– <step id="252" name="Confirm data" index="5" type="system">
  <description>Confirms that no other association exists to the step for the same class and association
    type.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog" />
    </step>
– <step id="254" name="Associate class to step" index="6" type="system">
  <description>Associate the selected class to the step using the association type selected.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityToModel" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
    </step>
– <step id="256" name="Association created" index="7" type="end">
  <description>The step association is successfuly created.</description>
    </step>
    </path>
– <path type="alternative" name="Conflict" begin="250">
– <step id="258" name="Already associated" index="1" type="system">
  <description>Confirms that the class is already associated with the step using the same association type. Show a
    message.</description>
    </step>
– <step id="260" name="Failure" index="2" type="end">
  <description>No association is created.</description>
    </step>
    </path>
– <path type="alternative" name="Delete association" begin="245">
– <step id="262" name="Select association" index="1" type="actor">
  <description>Select a pre-existing association to delete.</description>
    </step>
– <step id="263" name="Press delete" index="2" type="actor">
  <description>Selectes delete option</description>
    </step>
```

Descrições XML geradas (xmlOutput.xsl)


```
- <step id="264" name="Ask confirmation" index="3" type="system">
  <description>Ask the user to confirm the association removal.</description>
    </step>
- <step id="265" name="Confirm removal" index="4" type="actor">
  <description>Confirm to remove the association.</description>
    </step>
- <step id="266" name="Remove the association" index="5" type="system">
  <description>Remove the association between the step and the class.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityElement" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.model.activity.ActivityToModel" />
    </step>
- <step id="267" name="Association remove" index="6" type="end">
  <description>The association is removed.</description>
    </step>
    </path>
- <path type="alternative" name="Cancel association remove" begin="264">
- <step id="274" name="Cancel remove" index="1" type="actor">
  <description>Selects the "Cancel" button of the confirmation dialog box.</description>
    </step>
- <step id="275" name="Cancel remove" index="2" type="end">
  <description>The association is not removed.</description>
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```


Use case: UC009 – Export Description from Template


```
   <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="19" ucid="UC009" name="Export description from template">
  <description>Export the use case description based on a template.</description>
  <actor id="18" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="278" name="Start" index="1" type="start">
  <description>User is on the main screen.</description>
    </step>
- <step id="279" name="Select an use case" index="2" type="actor">
  <description>Select an use case from the tree view.</description>
    </step>
- <step id="280" name="Select an export format" index="3" type="actor">
  <description>Select an export format to process the use case description.</description>
    </step>
- <step id="281" name="Fetch XML description" index="4" type="system">
  <description>Fetch the XML description of the selected use case</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.xml.XMLProject" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.xml.XMLContent" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.xml.XMLUseCase" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.xml.XMLActor" />
    </step>
- <step id="282" name="Process with the export format" index="5" type="system">
  <description>Process the fetched XML description with the XSL obtained from the export format.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.xml.ProjectParser" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.xml.XMLProject" />
    </step>
- <step id="283" name="Save result" index="6" type="system">
  <description>Save result export into an appropriate file in the workdir.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
    </step>
- <step id="284" name="Call application" index="7" type="system">
  <description>Call the appropriate application to open the exported documentation.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
    </step>
```

Descrições XML geradas (xmlOutput.xsl)

```
- <step id="285" name="End" index="8" type="end">
  <description>The use case description was exported.</description>
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```

Use case: UC010 – Compile Use Case

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="20" ucid="UC010" name="Compile use case">
  <description>System compile the use case.</description>
  <actor id="18" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="293" name="Start" index="1" type="start">
  <description>A use case is passed to be compiled</description>
      </step>
- <step id="294" name="Confirm actors" index="2" type="system">
  <description>Confirm that actors where defined for the use case.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
      </step>
- <step id="295" name="Confirm elements" index="3" type="system">
  <description>Confirm that elements where defined</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
      </step>
- <step id="298" name="Confirm start" index="4" type="system">
  <description>Confirm that start step exists</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
      </step>
- <step id="300" name="Confirm ends" index="5" type="system">
  <description>Confirm that ends where defined for the use case.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
      </step>
- <step id="302" name="Confirm alternatives" index="6" type="system">
  <description>Confirm that all alternatives have and end or a return.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
      </step>
- <step id="310" name="End" index="7" type="end">
  <description>End of compilation. Return the compile messages (if any)</description>
      </step>
      </path>
- <path type="alternative" name="No actors" begin="294" return="295">
- <step id="303" name="No actors" index="1" type="system">
  <description>No actors where defined. Add a compile message.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
      </step>
      </path>
- <path type="alternative" name="No elements" begin="295" return="298">
- <step id="306" name="No elements" index="1" type="system">
  <description>No elements where defined for the description. Add a message.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
      </step>
      </path>
- <path type="alternative" name="No start" begin="298" return="300">
- <step id="312" name="No start" index="1" type="system">
  <description>No start was defined. Add a message.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
      </step>
      </path>
- <path type="alternative" name="No ends" begin="300" return="302">
```

Descrições XML geradas (xmlOutput.xsl)

```
- <step id="315" name="No ends" index="1" type="system">
  <description>No ends were defined. Add a message.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
     </step>
     </path>
- <path type="alternative" name="Alternatives failed" begin="302" return="310">
- <step id="318" name="Incomplete alternatives" index="1" type="system">
  <description>Some alternative is not complete. Add message.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
   </step>
   </path>
   </scenario>
   </usecase>
   </usecases>
   </project>
```

Use case: UC011 – View Project Tree

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="21" ucid="UC011" name="View project tree">
  <description>Displays the project tree.</description>
  <actor id="18" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="321" name="Start" index="1" type="start">
  <description>The open project is loaded.</description>
     </step>
- <step id="323" name="Fetch all elements" index="2" type="system">
  <description>Fetch all elements to be shown in the project tree.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.project.Project" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
     </step>
- <step id="326" name="Create fixed packages" index="3" type="system">
  <description>Create fixed packages for 'Analysis Packages', 'Actors','Use Cases' and 'Classes'.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.model.Package" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
     </step>
- <step id="322" name="Show analysis packages" index="4" type="system">
  <description>Show all analysis packages whitin the fixed 'Analysis package' package.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
     </step>
- <step id="324" name="Show Actors" index="5" type="system">
  <description>Show all actors found within their packages and within the 'Actors' fixed package..</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.Actor" />
     </step>
- <step id="325" name="Show use cases" index="6" type="system">
  <description>Show all use cases found within their packages and within the fixed 'Use case'
       package.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
     </step>
- <step id="327" name="Show classes and implementation packages" index="7" type="system">
  <description>Show all classes and implementation packages within the fixed 'Classes' package.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.UCDesigner" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.logical.Class" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.Package" />
```

Descrições XML geradas (xmlOutput.xsl)

```
    </step>
- <step id="328" name="End" index="8" type="end">
  <description>The tree is shown.</description>
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```

Use case: UC012 – Open Use Case

```
    <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="22" ucid="UC012" name="Open use case">
  <description>Open use case and display it in the main window.</description>
  <actor id="5" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="336" name="Start" index="1" type="start">
  <description>User selects to open an use case</description>
    </step>
- <step id="337" name="Open use case" index="2" type="actor">
  <description>Selects to open the use case.</description>
    </step>
- <step id="339" name="View description" index="3" type="system">
  <description>Confirms that description exists for at least one scenario and shows the description.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.UCDescription" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JScenarioDescription" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
    </step>
- <step id="343" name="End" index="4" type="end">
  <description>The use case is displayed.</description>
    </step>
    </path>
- <path type="alternative" name="No description available" begin="337" return="343">
- <step id="341" name="Open use case dialog" index="1" type="system">
  <description>Confirms that no description is available. Open the use case dialog.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.JUseCase" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.model.usecase.UseCase" />
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```

Use case: UC013 – Display Compile Message

```
    <?xml version="1.0" encoding="UTF-8" ?>
- <project name="UCDesigner" author="Gabriel Silva Bornia">
  <description>Tool for describing use cases.</description>
+ <actors>
- <usecases>
- <usecase id="23" ucid="UC013" name="Display compile messages">
  <description>System displays the compile messages.</description>
  <actor id="18" />
- <scenario name="Main scenario">
- <path type="main" name="Main flow">
- <step id="346" name="Start" index="1" type="start">
  <description>The project is compiled.</description>
    </step>
- <step id="347" name="Fecth all messages" index="2" type="system">
  <description>Fetch all messages from the compiler.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
```

Descrições XML geradas (xmlOutput.xsl)

```
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
     </step>
- <step id="348" name="Display messages" index="3" type="system">
  <description>Displays all messages with the message description, the severity and the element that originated
       the alert.</description>
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.compiler.UCPMessage" />
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.UCPMessageTableModel" />
  <modelref type="Depends on" class="br.edu.ufrgs.ucdesigner.visual.UCPMessageCellRenderer" />
     </step>
- <step id="349" name="Show messages" index="4" type="end">
  <description>All messages were displayed.</description>
     </step>
     </path>
- <path type="alternative" name="No messages" begin="347">
- <step id="350" name="No messages found" index="1" type="system">
  <description>No messages were found.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.compiler.UCPCompiler" />
     </step>
- <step id="351" name="Don't show the section" index="2" type="system">
  <description>The message section should not be shown.</description>
  <modelref type="Implemented by" class="br.edu.ufrgs.ucdesigner.visual.UCPMessageTableModel" />
     </step>
- <step id="352" name="Hide section" index="3" type="end">
  <description>The section is not shown.</description>
    </step>
    </path>
    </scenario>
    </usecase>
    </usecases>
    </project>
```

Descrições textuais geradas (default.xsl)

Use case: UC001 – Enter the tool

Use case summary	
Use case index:	UC001
Use case name:	Enter the tool
Description:	The user enter for the first time in the tool.
Actors:	User
Pre Conditions:	No other instance of the tool is opened.
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Tool is opened** The tool is opened **02 No last project found** System:System checks that no last project is found. **03 Display 'New project' section** System:Allow the user to enter a new project, showing the 'New project' section. **04 Display 'Open project' section** System:Allows the user to search and open another project, showing the 'Open project' section. **05 Selects to create a project** Actor:Selects the option to create a new project and selects 'OK'. **06 Create a new project** System:Creates a new empty project with the provided information. **07 End (success)** Close the start window and go to the main window with the new created project opened.
Alternative flow	**Last project exists** *Starts in step 1 (Tool is opened) of Main flow* **01 Last project found** System:Checks that there is a last project and that the file exists. **02 Displays 'Last project' section** System:Allows user to select the last project option. Provides information about the last project (name and file). *Return to 3 (Display 'New project' section) of Main flow*
Alternative flow	**Opens a project** *Starts in step 4 (Display 'Open project' section) of Main flow* **01 Selects to open a project** Actor:Selects to open an existent project (last project or open project) and selects 'OK'. **02 Open a project** System:Opens an existent project. **03 End (success)** The start screen is closed and the main screen is loaded with the opened project.

Alternative flow	**Abort tool** *Starts in step 4 (Display 'Open project' section) of Main flow* **01 Selects 'Cancel'** Actor:Selects to abort the operation and close the tool. **02 End (abort)** The tool is closed.

Use case: UC002 – View Project

Use case summary	
Use case index:	UC002
Use case name:	View Project
Description:	View a project after entering the tool or opening a new project.
Actors:	User
Pre Conditions:	The user already selected an existent project or created a new one.
Post Conditions:	The project is displayed.
Scenario: Main scenario	
Main flow	**01 Start** The user entered the tool and now need to view the project (new or already existent). **02 Load project info** System:Load all project information (author, last update date, description, etc). **03 Load project elements** System:Load use cases, use case descriptions, actors, packages and classes. **04 View tree** System:View tree for the elements loaded. <<include UC011 - View project tree>> **05 Has one or more use cases** System:Confirms that one or more use cases exist in the project. **06 Compile use cases** System:Compile use cases of the project. <<include UC010 - Compile use case>> **07 View messages** System:Display all existent messages for the project. <<include UC013 - Display compile messages>> **08 End** The project is displayed.
Alternative flow	**No use cases** *Starts in step 4 (View tree) of Main flow* **01 No use cases** System:Confirm that no use cases were entered for the project. *Return to 7 (View messages) of Main flow*

Use case: UC003 – Import Classes From Folder

Descrições textuais geradas (default.xsl)

Use case summary	
Use case index:	UC003
Use case name:	Import classes from folder
Description:	Allow to import existent JAVA classes form a folder.
Actors:	Developer
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** The user will import classes from a folder for the project **02 Add a folder** Actor:Selects to add a new folder for JAVA classes. **03 Ask folder location** System:Ask the user to select the folder from the file system **04 Enter folder information** Actor:Select folder from the file system and confirms **05 Search classes** System:Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all information for classes are fetched (class name, methods and attributes). **06 Add information to folder** System:All information fetched are added to the project as new elements. **07 Success** The folder in the project contains all classes found in the file system.
Alternative flow	**User aborts** *Starts in step 3 (Ask folder location) of Main flow* **01 Cancel operation** Actor:Select to cancel the operation of adding a folder to the project **02 Abort operation** No folder will be added
Alternative flow	**Refresh action** *Starts in step 1 (Start) of Main flow* **01 Refersh folder** Actor:Refresh existent JAVA folder **02 Clear folder** System:Clear all folder contents *Return to 5 (Search classes) of Main flow*

Use case: UC004 – Add Class

Use case summary	
Use case index:	UC004
Use case name:	Add class

Descrições textuais geradas (default.xsl)

Description:	Allows user to add a class manually.
Actors:	Developer
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** The user needs to add a class manually. **02 Select a package** Actor:Select an existent package for adding the class. **03 Add a new class** Actor:Selects to add a new class to the selected package **04 Ask class information** System:Ask the user for class information **05 Enter class information** Actor:Enter class information (class name, package, description, stereotype, methods and attributes). **06 Check class information** System:Confirms that no other class exists with the same name. **07 Add class** System:Add the class to the project. **08 Success** The user added a new class manually.
Alternative flow	**New subpackage** *Starts in step 2 (Select a package) of Main flow* **01 Add a new package** Actor:Selects to add another package **02 Ask package information** System:Ask the user for package information **03 Enter package information** Actor:Enter package name and parent. **04 Check package information** System:Check if no other package exists with the same name. **05 Add package** System:Add package to project. *Return to 2 (Select a package) of Main flow*
Alternative flow	**Package conflict** *Starts in step 4 (Check package information) of New subpackage* **01 Package conflict** System:Anotger package with the same name exists for the package parent. Show a message. **02 Abort package** No package is added
Alternative flow	**Class conflict** *Starts in step 6 (Check class information) of Main flow* **01 Class conflict**

	System:Confirms that other class already exist with the same name. Display a message. **02 Abort class** The class will not be added.
Alternative flow	**Cancel package** *Starts in step 2 (Ask package information) of New subpackage* **01 Cancel package addition** Actor:The user cancel the addition of the package *Return to 2 (Abort package) of Package conflict*
Alternative flow	**Cancel class addition** *Starts in step 4 (Ask class information) of Main flow* **01 Cancel class addition** Actor:Selects to cancel the class addition. *Return to 2 (Abort class) of Class conflict*

Use case: UC005 – Add Use Case

Use case summary	
Use case index:	UC005
Use case name:	Add use case
Description:	User enters a new use case.
Actors:	System analyst
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** User want to add a new use case **02 Add use case** Actor:Select to add use case from menu **03 Ask for information** System:Ask the user for use case information **04 Enter information** Actor:Enter information for the use case **05 Check information** System:Confirms that no other use case exists with the same use case ID or use case name. **06 Add use case** System:The use case is added to the project. **07 Success** The use case is successfuly created.
Alternative flow	**Add from package** *Starts in step 1 (Start) of Main flow* **01 Add use case from package tree** Actor:User select a particular package and selects to add an use case. **02 Populate package information**

	System:Pre-populate the pacjage information selecting the package that user choosed. *Return to 3 (Ask for information) of Main flow*
Alternative flow	**User abort** *Starts in step 3 (Ask for information) of Main flow* **01 Cancel** Actor:Selects to cancel the use case addition **02 Abort** No use case is added.
Alternative flow	**Add actor** *Starts in step 4 (Enter information) of Main flow* **01 Add actor** Actor:Selects to add an actor and associate it with the use case information. **02 Ask for actor information** System:Ask the user to enter more information to create the actor. **03 Enter actor information** Actor:Enter actor information to create the actor. **04 Check actor information** System:Confirms that no conflict exists with the actor name. **05 Add actor** System:The actor is added to the project and associated to the use case. *Return to 4 (Enter information) of Main flow*
Alternative flow	**User abort actor addition** *Starts in step 2 (Ask for actor information) of Add actor* **01 Cancel actor** Actor:Cancel the actor addition *Return to 4 (Enter information) of Main flow*
Alternative flow	**Actor conflict** *Starts in step 4 (Check actor information) of Add actor* **01 Actor conflict** System:Confirms that other actor exists with the same name. Display a message. *Return to 4 (Enter information) of Main flow*
Alternative flow	**Use case conflict** *Starts in step 5 (Check information) of Main flow* **01 Use case conflict** System:Confirms that other use case exists with the same use case ID or use case name. Display a message. *Return to 3 (Ask for information) of Main flow*

Use case: UC006 – Describe Scenario

Use case summary	
Use case index:	UC006
Use case name:	Describe scenario
Description:	Use can describe the scenario using activity diagram.
Actors:	System analyst

Descrições textuais geradas (default.xsl)

Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** User will enter the use case description for a particular use case. **02 Select a step to add** Actor:Select a step to add to the description. **03 Add step to a position** Actor:Click on the place the step will be added. **04 Ask user for step information** System:Ask the user for more step information, considering the type of the step. <<include UC007 - Add step information>> **05 Enter step information** Actor:User enter all information necessary for the step and press "OK". **06 Add step to the description** System:The step is added to the use case description. **07 Compile description** System:Compile the use case description based on the updated data. <<include UC010 - Compile use case>> **08 Success** The new step is added to the use case description.
Alternative flow	**Abort step addition** *Starts in step 4 (Ask user for step information) of Main flow* **01 Cancel step addition** Actor:Selects to cancel step addition **02 Abort step addition** The step will not be added.
Alternative flow	**Add a connection** *Starts in step 1 (Start) of Main flow* **01 Select to add a connection** Actor:Select to add a connection between two steps. **02 Connect two steps** Actor:Connect two steps by clicking on the origin step and clicking on the destination step. **03 Connect steps** System:Connect the steps. *Return to 7 (Compile description) of Main flow*
Alternative flow	**Remove an element** *Starts in step 1 (Start) of Main flow* **01 Remove an element** Actor:Remove an element from the description (could be a step or a connection). **02 Confirm delete** System:Ask the user to confirm the element removal. **03 Confirm action** Actor:Confirm to delete the element.

	04 Remove element System:Remove the element from the use case description. *Return to 7 (Compile description) of Main flow*
Alternative flow	**Cancel removal** *Starts in step 2 (Confirm delete) of Remove an element* **01 Cancel removal** Actor:Cancel the element removal. **02 Abort element removal** No element is removed from the description.

Use case: UC007– Add Sep Information

Use case summary	
Use case index:	UC007
Use case name:	Add step information
Description:	User enters specific step information
Actors:	System analyst
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** User selected to add a new step **02 Ask user to enter information** System:Ask the user to enter information accordingly to the type of the step. **03 Enter information** Actor:Enter the information required. **04 Confirm data input** Actor:Confirm data input by pressing the "OK" button. **05 Check required data** System:Confirm that all the required data was entered correctly. **06 Success** Return the step with the new information provided.
Alternative flow	**Abort action** *Starts in step 3 (Enter information) of Main flow* **01 Cancel data input** Actor:Cancel data input by clicking on the "Cancel" button. **02 Abort editing** Abort step information editing.
Alternative flow	**Associate with model** *Starts in step 3 (Enter information) of Main flow* **01 Associate with logical model** Actor:Selects to associate the step with the logical model.

	02 Do association System:Do the association between the step and the logical model. <<include UC008 - Associate with Logical Model>> *Return to 3 (Enter information) of Main flow*

Use case: UC008 – Associate with Logical Model

Use case summary	
Use case index:	UC008
Use case name:	Associate with Logical Model
Description:	Associate use case step with the logical model.
Actors:	Developer System analyst
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** Within the step information is possible to associate the step with the logical model. **02 Select association type** Actor:Select association type within the options "Implemented by", "Depends on" and "Used by". **03 Select class** Actor:Select class from the model **04 Associate** Actor:Select option to associate the selected class and association type to the step. **05 Confirm data** System:Confirms that no other association exists to the step for the same class and association type. **06 Associate class to step** System:Associate the selected class to the step using the association type selected. **07 Association created** The step association is successfuly created.
Alternative flow	**Conflict** *Starts in step 4 (Associate) of Main flow* **01 Already associated** System:Confirms that the class is already associated with the step using the same association type. Show a message. **02 Failure** No association is created.
Alternative flow	**Delete association** *Starts in step 1 (Start) of Main flow* **01 Select association** Actor:Select a pre-existing association to delete. **02 Press delete** Actor:Selectes delete option

	03 Ask confirmation System:Ask the user to confirm the association removal. **04 Confirm removal** Actor:Confirm to remove the association. **05 Remove the association** System:Remove the association between the step and the class. **06 Association remove** The association is removed.
Alternative flow	**Cancel association remove** *Starts in step 3 (Ask confirmation) of Delete association* **01 Cancel remove** Actor:Selects the "Cancel" button of the confirmation dialog box. **02 Cancel remove** The association is not removed.

Use case: UC009 – Export Description from Template

Use case summary	
Use case index:	UC009
Use case name:	Export description from template
Description:	Export the use case description based on a template.
Actors:	System
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** User is on the main screen. **02 Select an use case** Actor:Select an use case from the tree view. **03 Select an export format** Actor:Select an export format to process the use case description. **04 Fetch XML description** System:Fetch the XML description of the selected use case **05 Process with the export format** System:Process the fetched XML description with the XSL obtained from the export format. **06 Save result** System:Save result export into an appropriate file in the workdir. **07 Call application** System:Call the appropriate application to open the exported documentation. **08 End** The use case description was exported.

Descrições textuais geradas (default.xsl)

Use case: UC010 – Compile Use Case

Use case summary	
Use case index:	UC010
Use case name:	Compile use case
Description:	System compile the use case.
Actors:	System
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** A use case is passed to be compiled **02 Confirm actors** System:Confirm that actors where defined for the use case. **03 Confirm elements** System:Confirm that elements where defined **04 Confirm start** System:Confirm that start step exists **05 Confirm ends** System:Confirm that ends where defined for the use case. **06 Confirm alternatives** System:Confirm that all alternatives have and end or a return. **07 End** End of compilation. Return the compile messages (if any)
Alternative flow	**No actors** *Starts in step 2 (Confirm actors) of Main flow* **01 No actors** System:No actors where defined. Add a compile message. *Return to 3 (Confirm elements) of Main flow*
Alternative flow	**No elements** *Starts in step 3 (Confirm elements) of Main flow* **01 No elements** System:No elements where defined for the description. Add a message. *Return to 4 (Confirm start) of Main flow*
Alternative flow	**No start** *Starts in step 4 (Confirm start) of Main flow* **01 No start** System:No start was defined. Add a message. *Return to 5 (Confirm ends) of Main flow*
Alternative flow	**No ends** *Starts in step 5 (Confirm ends) of Main flow* **01 No ends** System:No ends were defined. Add a message.

	Return to 6 (Confirm alternatives) of Main flow
Alternative flow	**Alternatives failed** *Starts in step 6 (Confirm alternatives) of Main flow* **01 Incomplete alternatives** System:Some alternative is not complete. Add message. *Return to 7 (End) of Main flow*

Use case: UC011 – View Project Tree

Use case summary	
Use case index:	UC011
Use case name:	View project tree
Description:	Displays the project tree.
Actors:	System
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** The open project is loaded. **02 Fetch all elements** System:Fetch all elements to be shown in the project tree. **03 Create fixed packages** System:Create fixed packages for 'Analysis Packages', 'Actors','Use Cases' and 'Classes'. **04 Show analysis packages** System:Show all analysis packages whitin the fixed 'Analysis package' package. **05 Show Actors** System:Show all actors found within their packages and within the 'Actors' fixed package.. **06 Show use cases** System:Show all use cases found within their packages and within the fixed 'Use case' package. **07 Show classes and implementation packages** System:Show all classes and implementation packages within the fixed 'Classes' package. **08 End** The tree is shown.

Use case: UC012 – Open Use Case

Use case summary	
Use case index:	UC012
Use case name:	Open use case
Description:	Open use case and display it in the main window.
Actors:	User
Pre Conditions:	

Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** User selects to open an use case **02 Open use case** Actor:Selects to open the use case. **03 View description** System:Confirms that description exists for at least one scenario and shows the description. **04 End** The use case is displayed.
Alternative flow	**No description available** *Starts in step 2 (Open use case) of Main flow* **01 Open use case dialog** System:Confirms that no description is available. Open the use case dialog. *Return to 4 (End) of Main flow*

Use case: UC013 – Display Compile Message

Use case summary	
Use case index:	UC013
Use case name:	Display compile messages
Description:	System displays the compile messages.
Actors:	System
Pre Conditions:	
Post Conditions:	
Scenario: Main scenario	
Main flow	**01 Start** The project is compiled. **02 Fecth all messages** System:Fetch all messages from the compiler. **03 Display messages** System:Displays all messages with the message description, the severity and the element that originated the alert. **04 Show messages** All messages were displayed.
Alternative flow	**No messages** *Starts in step 2 (Fecth all messages) of Main flow* **01 No messages found** System:No messages were found. **02 Don't show the section** System:The message section should not be shown. **03 Hide section** The section is not shown.

Casos de teste gerados (testCases.xsl)

Use case: UC001 – Enter the tool

Test case summary	
Use case index:	UC001
Use case name:	Enter the tool

Main flow

Action	**Expected result**
Go to Main flow 01 Tool is opened	
02 No last project found	
03 Display 'New project' section	
04 Display 'Open project' section	
05 Selects to create a project	
06 Create a new project	
07 End (success) *Use case ends*	

Last project exists

Action	**Expected result**
Go to Main flow 01 Tool is opened	
Go to Last project exists 01 Last project found	
02 Displays 'Last project' section	
Return to Main flow 03 Display 'New project' section	
04 Display 'Open project' section	
05 Selects to create a project	
06 Create a new project	
07 End (success) *Use case ends*	

Opens a project

Action	**Expected result**
Go to Main flow 01 Tool is opened	
02 No last project found	
03 Display 'New project' section	
04 Display 'Open project' section	
Go to Opens a project 01 Selects to open a project	
02 Open a project	
03 End (success) *Use case ends*	

Abort tool

Action	**Expected result**

Go to Main flow
01 Tool is opened

02 No last project found

03 Display 'New project' section

04 Display 'Open project' section

Go to Abort tool
01 Selects 'Cancel'

02 End (abort)
Use case ends

Use case: UC002 – View Project

Test case summary	
Use case index:	UC002
Use case name:	View Project

Main flow

Action	**Expected result**
Go to Main flow 01 Start	
02 Load project info	
03 Load project elements	
04 View tree	
05 Has one or more use cases	
06 Compile use cases	
07 View messages	
08 End *Use case ends*	

No use cases

Action	**Expected result**
Go to Main flow 01 Start	
02 Load project info	
03 Load project elements	
04 View tree	
Go to No use cases 01 No use cases	
Return to Main flow 07 View messages	
08 End *Use case ends*	

Casos de teste (testCases.xsl)

Use case: UC003 – Import Classes From Folder

Test case summary	
Use case index:	UC003
Use case name:	Import classes from folder

Main flow

Action	**Expected result**
Go to Main flow 01 Start	
02 Add a folder	
03 Ask folder location	
04 Enter folder information	
05 Search classes	
06 Add information to folder	
07 Success *Use case ends*	

User aborts

Action	**Expected result**
Go to Main flow 01 Start	
02 Add a folder	
03 Ask folder location	
Go to User aborts 01 Cancel operation	
02 Abort operation *Use case ends*	

Refresh action

Action	**Expected result**
Go to Main flow 01 Start	
Go to Refresh action 01 Refersh folder	
02 Clear folder	
Return to Main flow 05 Search classes	
06 Add information to folder	
07 Success *Use case ends*	

Use case: UC004 – Add Class

Test case summary	
Use case index:	UC004
Use case name:	Add class

Main flow

Action	**Expected result**
Go to Main flow	

01 Start

02 Select a package

03 Add a new class

04 Ask class information

05 Enter class information

06 Check class information

07 Add class

08 Success

Use case ends

New subpackage

Action	Expected result
Go to Main flow	
01 Start	
02 Select a package	
Go to New subpackage	
01 Add a new package	
02 Ask package information	
03 Enter package information	
04 Check package information	
05 Add package	
Return to Main flow	
02 Select a package	
03 Add a new class	
04 Ask class information	
05 Enter class information	
06 Check class information	
07 Add class	
08 Success	
Use case ends	

Package conflict

Action	Expected result
Go to Main flow	
01 Start	
02 Select a package	
Go to New subpackage	
01 Add a new package	
02 Ask package information	
03 Enter package information	
04 Check package information	
Go to Package conflict	
01 Package conflict	
02 Abort package	
Use case ends	

Class conflict

Action	Expected result
Go to Main flow	
01 Start	
02 Select a package	

03 Add a new class 04 Ask class information 05 Enter class information 06 Check class information *Go to Class conflict* 01 Class conflict 02 Abort class *Use case ends*

Cancel package	
Action	**Expected result**
Go to Main flow 01 Start 02 Select a package *Go to New subpackage* 01 Add a new package 02 Ask package information *Go to Cancel package* 01 Cancel package addition *Return to Package conflict* 02 Abort package *Use case ends*	

Cancel class addition	
Action	**Expected result**
Go to Main flow 01 Start 02 Select a package 03 Add a new class 04 Ask class information *Go to Cancel class addition* 01 Cancel class addition *Return to Class conflict* 02 Abort class *Use case ends*	

Use case: UC005 – Add Use Case

Test case summary	
Use case index:	UC005
Use case name:	Add use case

Main flow	
Action	**Expected result**
Go to Main flow 01 Start 02 Add use case 03 Ask for information 04 Enter information 05 Check information 06 Add use case	

07 Success
Use case ends

Add from package

Action	Expected result
Go to Main flow 01 Start	
Go to Add from package 01 Add use case from package tree	
02 Populate package information	
Return to Main flow 03 Ask for information	
04 Enter information	
05 Check information	
06 Add use case	
07 Success *Use case ends*	

User abort

Action	Expected result
Go to Main flow 01 Start	
02 Add use case	
03 Ask for information	
Go to User abort 01 Cancel	
02 Abort *Use case ends*	

Add actor

Action	Expected result
Go to Main flow 01 Start	
02 Add use case	
03 Ask for information	
04 Enter information	
Go to Add actor 01 Add actor	
02 Ask for actor information	
03 Enter actor information	
04 Check actor information	
05 Add actor	
Return to Main flow 04 Enter information	
05 Check information	
06 Add use case	
07 Success *Use case ends*	

User abort actor addition

Action	Expected result
Go to Main flow	

01 Start

02 Add use case

03 Ask for information

04 Enter information

Go to Add actor
01 Add actor

02 Ask for actor information

Go to User abort actor addition
01 Cancel actor

Return to Main flow
04 Enter information

05 Check information

06 Add use case

07 Success
Use case ends

Actor conflict

Action	Expected result
Go to Main flow 01 Start	
02 Add use case	
03 Ask for information	
04 Enter information	
Go to Add actor 01 Add actor	
02 Ask for actor information	
03 Enter actor information	
04 Check actor information	
Go to Actor conflict 01 Actor conflict	
Return to Main flow 04 Enter information	
05 Check information	
06 Add use case	
07 Success *Use case ends*	

Use case conflict

Action	Expected result
Go to Main flow 01 Start	
02 Add use case	
03 Ask for information	
04 Enter information	
05 Check information	
Go to Use case conflict 01 Use case conflict	
Return to Main flow 03 Ask for information	
04 Enter information	
05 Check information	

06 Add use case 07 Success *Use case ends*

Use case: UC006 – Describe Scenario

Test case summary	
Use case index:	UC006
Use case name:	Describe scenario

Main flow

Action	**Expected result**
Go to Main flow 01 Start 02 Select a step to add 03 Add step to a position 04 Ask user for step information 05 Enter step information 06 Add step to the description 07 Compile description 08 Success *Use case ends*	

Abort step addition

Action	**Expected result**
Go to Main flow 01 Start 02 Select a step to add 03 Add step to a position 04 Ask user for step information *Go to Abort step addition* 01 Cancel step addition 02 Abort step addition *Use case ends*	

Add a connection

Action	**Expected result**
Go to Main flow 01 Start *Go to Add a connection* 01 Select to add a connection 02 Connect two steps 03 Connect steps *Return to Main flow* 07 Compile description 08 Success *Use case ends*	

Remove an element

Action	**Expected result**
Go to Main flow	

01 Start *Go to Remove an element* 01 Remove an element 02 Confirm delete 03 Confirm action 04 Remove element *Return to Main flow* 07 Compile description 08 Success *Use case ends*
Cancel removal

Action	**Expected result**
Go to Main flow 01 Start *Go to Remove an element* 01 Remove an element 02 Confirm delete *Go to Cancel removal* 01 Cancel removal 02 Abort element removal *Use case ends*	

Use case: UC007– Add Sep Information

Test case summary	
Use case index:	UC007
Use case name:	Add step information
Main flow	

Action	**Expected result**
Go to Main flow 01 Start 02 Ask user to enter information 03 Enter information 04 Confirm data input 05 Check required data 06 Success *Use case ends*	
Abort action	

Action	**Expected result**
Go to Main flow 01 Start 02 Ask user to enter information 03 Enter information *Go to Abort action* 01 Cancel data input 02 Abort editing *Use case ends*	
Associate with model	

Action	Expected result
Go to Main flow 01 Start	
02 Ask user to enter information	
03 Enter information	
Go to Associate with model 01 Associate with logical model	
02 Do association	
Return to Main flow 03 Enter information	
04 Confirm data input	
05 Check required data	
06 Success *Use case ends*	

Use case: UC008 – Associate with Logical Model

Test case summary	
Use case index:	UC008
Use case name:	Associate with Logical Model

Main flow

Action	Expected result
Go to Main flow 01 Start	
02 Select association type	
03 Select class	
04 Associate	
05 Confirm data	
06 Associate class to step	
07 Association created *Use case ends*	

Conflict

Action	Expected result
Go to Main flow 01 Start	
02 Select association type	
03 Select class	
04 Associate	
Go to Conflict 01 Already associated	
02 Failure *Use case ends*	

Delete association

Action	Expected result
Go to Main flow 01 Start	
Go to Delete association 01 Select association	

02 Press delete 03 Ask confirmation 04 Confirm removal 05 Remove the association 06 Association remove *Use case ends*

Cancel association remove	
Action	**Expected result**
Go to Main flow 01 Start *Go to Delete association* 01 Select association 02 Press delete 03 Ask confirmation *Go to Cancel association remove* 01 Cancel remove 02 Cancel remove *Use case ends*	

Use case: UC009 – Export Description from Template

Test case summary	
Use case index:	UC009
Use case name:	Export description from template

Main flow	
Action	**Expected result**
Go to Main flow 01 Start 02 Select an use case 03 Select an export format 04 Fetch XML description 05 Process with the export format 06 Save result 07 Call application 08 End *Use case ends*	

Use case: UC010 – Compile Use Case

Test case summary	
Use case index:	UC010
Use case name:	Compile use case

Main flow	
Action	**Expected result**
Go to Main flow 01 Start	

02 Confirm actors

03 Confirm elements

04 Confirm start

05 Confirm ends

06 Confirm alternatives

07 End
Use case ends

No actors

Action	**Expected result**
Go to Main flow 01 Start	
02 Confirm actors	
Go to No actors 01 No actors	
Return to Main flow 03 Confirm elements	
04 Confirm start	
05 Confirm ends	
06 Confirm alternatives	
07 End *Use case ends*	

No elements

Action	**Expected result**
Go to Main flow 01 Start	
02 Confirm actors	
03 Confirm elements	
Go to No elements 01 No elements	
Return to Main flow 04 Confirm start	
05 Confirm ends	
06 Confirm alternatives	
07 End *Use case ends*	

No start

Action	**Expected result**
Go to Main flow 01 Start	
02 Confirm actors	
03 Confirm elements	
04 Confirm start	
Go to No start 01 No start	
Return to Main flow 05 Confirm ends	
06 Confirm alternatives	
07 End *Use case ends*	

No ends	
Action	**Expected result**
Go to Main flow 01 Start 02 Confirm actors 03 Confirm elements 04 Confirm start 05 Confirm ends *Go to No ends* 01 No ends *Return to Main flow* 06 Confirm alternatives 07 End *Use case ends*	

Alternatives failed	
Action	**Expected result**
Go to Main flow 01 Start 02 Confirm actors 03 Confirm elements 04 Confirm start 05 Confirm ends 06 Confirm alternatives *Go to Alternatives failed* 01 Incomplete alternatives *Return to Main flow* 07 End *Use case ends*	

Use case: UC011 – View Project Tree

Test case summary	
Use case index:	UC011
Use case name:	View project tree

Main flow	
Action	**Expected result**
Go to Main flow 01 Start 02 Fetch all elements 03 Create fixed packages 04 Show analysis packages 05 Show Actors 06 Show use cases 07 Show classes and implementation packages 08 End *Use case ends*	

Use case: UC012 – Open Use Case

Test case summary	
Use case index:	UC012
Use case name:	Open use case

Main flow

Action	Expected result
Go to Main flow 01 Start 02 Open use case 03 View description 04 End *Use case ends*	

No description available

Action	Expected result
Go to Main flow 01 Start 02 Open use case *Go to No description available* 01 Open use case dialog *Return to Main flow* 04 End *Use case ends*	

Use case: UC013 – Display Compile Message

Test case summary	
Use case index:	UC013
Use case name:	Display compile messages

Main flow

Action	Expected result
Go to Main flow 01 Start 02 Fecth all messages 03 Display messages 04 Show messages *Use case ends*	

No messages

Action	Expected result
Go to Main flow 01 Start 02 Fecth all messages *Go to No messages* 01 No messages found 02 Don't show the section 03 Hide section *Use case ends*	

Referências ao modelo lógico (logicalReferences.xsl)

Use case: UC001 – Enter the tool

Use case summary	
Use case index:	UC001
Use case name:	Enter the tool
Description:	The user enter for the first time in the tool.
References to class: br.edu.ufrgs.ucdesigner.project.Project	
Create a new project, step 6 of Main flow Creates a new empty project with the provided information. *Depends on br.edu.ufrgs.ucdesigner.project.Project*	
Open a project, step 2 of Opens a project Opens an existent project. *Depends on br.edu.ufrgs.ucdesigner.project.Project*	
References to class: br.edu.ufrgs.ucdesigner.UCDesigner	
Tool is opened, step 1 of Main flow The tool is opened *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
Create a new project, step 6 of Main flow Creates a new empty project with the provided information. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
End (success), step 7 of Main flow Close the start window and go to the main window with the new created project opened. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
Open a project, step 2 of Opens a project Opens an existent project. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
End (success), step 3 of Opens a project The start screen is closed and the main screen is loaded with the opened project. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
End (abort), step 2 of Abort tool The tool is closed. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
References to class: br.edu.ufrgs.ucdesigner.visual.JStartProject	
Tool is opened, step 1 of Main flow The tool is opened *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*	
No last project found, step 2 of Main flow System checks that no last project is found. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*	
Display 'New project' section, step 3 of Main flow Allow the user to enter a new project, showing the 'New project' section. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*	
Display 'Open project' section, step 4 of Main flow Allows the user to search and open another project, showing the 'Open project' section. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*	
End (success), step 7 of Main flow Close the start window and go to the main window with the new created project opened. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*	

Referências ao modelo lógico (logicalReferences.xsl)

Last project found, step 1 of Last project exists Checks that there is a last project and that the file exists. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*
Displays 'Last project' section, step 2 of Last project exists Allows user to select the last project option. Provides information about the last project (name and file). *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*
Open a project, step 2 of Opens a project Opens an existent project. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*
End (success), step 3 of Opens a project The start screen is closed and the main screen is loaded with the opened project. *Implemented by br.edu.ufrgs.ucdesigner.visual.JStartProject*

Use case: UC002 – View Project

Use case summary	
Use case index:	UC002
Use case name:	View Project
Description:	View a project after entering the tool or opening a new project.
References to class: br.edu.ufrgs.ucdesigner.compiler.UCPCompiler	
Has one or more use cases, step 5 of Main flow Confirms that one or more use cases exist in the project. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No use cases, step 1 of No use cases Confirm that no use cases were entered for the project. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
References to class: br.edu.ufrgs.ucdesigner.model.activity.ActivityElement	
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
References to class: br.edu.ufrgs.ucdesigner.model.logical.Class	
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Class*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.Actor	
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.ExtensionPoint	
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.ExtensionPoint*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCase	
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
Has one or more use cases, step 5 of Main flow Confirms that one or more use cases exist in the project. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
No use cases, step 1 of No use cases Confirm that no use cases were entered for the project.	

Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCaseDescription
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCaseDescription*
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCaseElement
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCaseElement*
References to class: br.edu.ufrgs.ucdesigner.project.GraphicRepresentation
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Implemented by br.edu.ufrgs.ucdesigner.project.GraphicRepresentation*
References to class: br.edu.ufrgs.ucdesigner.project.ObjectRepository
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
References to class: br.edu.ufrgs.ucdesigner.project.Project
Has one or more use cases, step 5 of Main flow Confirms that one or more use cases exist in the project. *Depends on br.edu.ufrgs.ucdesigner.project.Project*
No use cases, step 1 of No use cases Confirm that no use cases were entered for the project. *Depends on br.edu.ufrgs.ucdesigner.project.Project*
Load project info, step 2 of Main flow Load all project information (author, last update date, description, etc). *Implemented by br.edu.ufrgs.ucdesigner.project.Project*
References to class: br.edu.ufrgs.ucdesigner.UCDesigner
Load project info, step 2 of Main flow Load all project information (author, last update date, description, etc). *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*
References to class: br.edu.ufrgs.ucdesigner.xml.XMLProject
Load project elements, step 3 of Main flow Load use cases, use case descriptions, actors, packages and classes. *Implemented by br.edu.ufrgs.ucdesigner.xml.XMLProject*

Use case: UC003 – Import Classes From Folder

Use case summary	
Use case index:	UC003
Use case name:	Import classes from folder
Description:	Allow to import existent JAVA classes form a folder.
References to class: br.edu.ufrgs.ucdesigner.java.RefreshJavaFolder	
Search classes, step 5 of Main flow Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all information for classes are fetched (class name, methods and attributes). *Implemented by br.edu.ufrgs.ucdesigner.java.RefreshJavaFolder*	
References to class: br.edu.ufrgs.ucdesigner.model.logical.Attribute	
Search classes, step 5 of Main flow Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all	

information for classes are fetched (class name, methods and attributes). *Used by br.edu.ufrgs.ucdesigner.model.logical.Attribute*
References to class: br.edu.ufrgs.ucdesigner.model.logical.Class
Search classes, step 5 of Main flow Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all information for classes are fetched (class name, methods and attributes). *Used by br.edu.ufrgs.ucdesigner.model.logical.Class*
References to class: br.edu.ufrgs.ucdesigner.model.logical.Method
Search classes, step 5 of Main flow Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all information for classes are fetched (class name, methods and attributes). *Used by br.edu.ufrgs.ucdesigner.model.logical.Method*
References to class: br.edu.ufrgs.ucdesigner.UCDesigner
Ask folder location, step 3 of Main flow Ask the user to select the folder from the file system *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*
References to class: br.edu.ufrgs.ucdesigner.visual.JJavaFolder
Search classes, step 5 of Main flow Searches classes inside the folder. The operation also includes all subfolders. All java files are examined and all information for classes are fetched (class name, methods and attributes). *Implemented by br.edu.ufrgs.ucdesigner.visual.JJavaFolder*
Clear folder, step 2 of Refresh action Clear all folder contents *Implemented by br.edu.ufrgs.ucdesigner.visual.JJavaFolder*

Use case: UC004 – Add Class

Use case summary	
Use case index:	UC004
Use case name:	Add class
Description:	Allows user to add a class manually.
References to class: br.edu.ufrgs.ucdesigner.model.logical.Attribute	
Add class, step 7 of Main flow Add the class to the project. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Attribute*	
References to class: br.edu.ufrgs.ucdesigner.model.logical.Class	
Add class, step 7 of Main flow Add the class to the project. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Class*	
References to class: br.edu.ufrgs.ucdesigner.model.logical.Method	
Add class, step 7 of Main flow Add the class to the project. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Method*	
References to class: br.edu.ufrgs.ucdesigner.model.Package	
Add class, step 7 of Main flow Add the class to the project. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	
Add package, step 5 of New subpackage Add package to project. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	

Referências ao modelo lógico (logicalReferences.xsl)

References to class: br.edu.ufrgs.ucdesigner.model.Stereotype
Add class, step 7 of Main flow Add the class to the project. *Depends on br.edu.ufrgs.ucdesigner.model.Stereotype*
References to class: br.edu.ufrgs.ucdesigner.project.ObjectRepository
Check class information, step 6 of Main flow Confirms that no other class exists with the same name. *Depends on br.edu.ufrgs.ucdesigner.project.ObjectRepository*
Add class, step 7 of Main flow Add the class to the project. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
Add package, step 5 of New subpackage Add package to project. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
References to class: br.edu.ufrgs.ucdesigner.project.Project
Add package, step 5 of New subpackage Add package to project. *Depends on br.edu.ufrgs.ucdesigner.project.Project*
References to class: br.edu.ufrgs.ucdesigner.visual.JClass
Ask class information, step 4 of Main flow Ask the user for class information *Implemented by br.edu.ufrgs.ucdesigner.visual.JClass*
Check class information, step 6 of Main flow Confirms that no other class exists with the same name. *Implemented by br.edu.ufrgs.ucdesigner.visual.JClass*
Add class, step 7 of Main flow Add the class to the project. *Implemented by br.edu.ufrgs.ucdesigner.visual.JClass*
References to class: br.edu.ufrgs.ucdesigner.visual.JPackage
Ask package information, step 2 of New subpackage Ask the user for package information *Implemented by br.edu.ufrgs.ucdesigner.visual.JPackage*
Check package information, step 4 of New subpackage Check if no other package exists with the same name. *Implemented by br.edu.ufrgs.ucdesigner.visual.JPackage*

Use case: UC005 – Add Use Case

Use case summary	
Use case index:	UC005
Use case name:	Add use case
Description:	User enters a new use case.
References to class: br.edu.ufrgs.ucdesigner.model.Package	
Populate package information, step 2 of Add from package Pre-populate the pacjage information selecting the package that user choosed. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.Actor	
Add use case, step 6 of Main flow The use case is added to the project. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*	

Referências ao modelo lógico (logicalReferences.xsl)

Check actor information, step 4 of Add actor Confirms that no conflict exists with the actor name. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*
Add actor, step 5 of Add actor The actor is added to the project and associated to the use case. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCase
Check information, step 5 of Main flow Confirms that no other use case exists with the same use case ID or use case name. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*
Add use case, step 6 of Main flow The use case is added to the project. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*
References to class: br.edu.ufrgs.ucdesigner.project.ObjectRepository
Check information, step 5 of Main flow Confirms that no other use case exists with the same use case ID or use case name. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
Add use case, step 6 of Main flow The use case is added to the project. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
Check actor information, step 4 of Add actor Confirms that no conflict exists with the actor name. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
Add actor, step 5 of Add actor The actor is added to the project and associated to the use case. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*
References to class: br.edu.ufrgs.ucdesigner.project.Project
Add use case, step 6 of Main flow The use case is added to the project. *Depends on br.edu.ufrgs.ucdesigner.project.Project*
Add actor, step 5 of Add actor The actor is added to the project and associated to the use case. *Depends on br.edu.ufrgs.ucdesigner.project.Project*
References to class: br.edu.ufrgs.ucdesigner.visual.JActor
Ask for actor information, step 2 of Add actor Ask the user to enter more information to create the actor. *Implemented by br.edu.ufrgs.ucdesigner.visual.JActor*
Check actor information, step 4 of Add actor Confirms that no conflict exists with the actor name. *Implemented by br.edu.ufrgs.ucdesigner.visual.JActor*
References to class: br.edu.ufrgs.ucdesigner.visual.JUseCase
Ask for information, step 3 of Main flow Ask the user for use case information *Implemented by br.edu.ufrgs.ucdesigner.visual.JUseCase*
Check information, step 5 of Main flow Confirms that no other use case exists with the same use case ID or use case name. *Implemented by br.edu.ufrgs.ucdesigner.visual.JUseCase*
Populate package information, step 2 of Add from package Pre-populate the pacjage information selecting the package that user choosed. *Implemented by br.edu.ufrgs.ucdesigner.visual.JUseCase*

Use case: UC006 – Describe Scenario

Referências ao modelo lógico (logicalReferences.xsl)

Use case summary	
Use case index:	UC006
Use case name:	Describe scenario
Description:	Use can describe the scenario using activity diagram.
References to class: br.edu.ufrgs.ucdesigner.graphic.GraphicComponent	
Connect steps, step 3 of Add a connection Connect the steps. *Depends on br.edu.ufrgs.ucdesigner.graphic.GraphicComponent*	
References to class: br.edu.ufrgs.ucdesigner.graphic.GraphicConnection	
Remove element, step 4 of Remove an element Remove the element from the use case description. *Depends on br.edu.ufrgs.ucdesigner.graphic.GraphicConnection*	
Connect steps, step 3 of Add a connection Connect the steps. *Implemented by br.edu.ufrgs.ucdesigner.graphic.GraphicConnection*	
References to class: br.edu.ufrgs.ucdesigner.graphic.GraphicElement	
Connect steps, step 3 of Add a connection Connect the steps. *Depends on br.edu.ufrgs.ucdesigner.graphic.GraphicElement*	
Remove element, step 4 of Remove an element Remove the element from the use case description. *Depends on br.edu.ufrgs.ucdesigner.graphic.GraphicElement*	
References to class: br.edu.ufrgs.ucdesigner.model.activity.ActivityElement	
Add step to the description, step 6 of Main flow The step is added to the use case description. *Depends on br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
Connect two steps, step 2 of Add a connection Connect two steps by clicking on the origin step and clicking on the destination step. *Depends on br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
Connect steps, step 3 of Add a connection Connect the steps. *Depends on br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
References to class: br.edu.ufrgs.ucdesigner.project.GraphicRepresentation	
Add step to the description, step 6 of Main flow The step is added to the use case description. *Implemented by br.edu.ufrgs.ucdesigner.project.GraphicRepresentation*	
Connect steps, step 3 of Add a connection Connect the steps. *Implemented by br.edu.ufrgs.ucdesigner.project.GraphicRepresentation*	
Remove element, step 4 of Remove an element Remove the element from the use case description. *Implemented by br.edu.ufrgs.ucdesigner.project.GraphicRepresentation*	
References to class: br.edu.ufrgs.ucdesigner.project.ObjectRepository	
Remove element, step 4 of Remove an element Remove the element from the use case description. *Implemented by br.edu.ufrgs.ucdesigner.project.ObjectRepository*	
References to class: br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog	
Confirm delete, step 2 of Remove an element Ask the user to confirm the element removal. *Implemented by br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog*	

References to class: br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel
Add step to a position, step 3 of Main flow Click on the place the step will be added. *Depends on br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel*
Select to add a connection, step 1 of Add a connection Select to add a connection between two steps. *Depends on br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel*
Connect two steps, step 2 of Add a connection Connect two steps by clicking on the origin step and clicking on the destination step. *Depends on br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel*
Remove an element, step 1 of Remove an element Remove an element from the description (could be a step or a connection). *Depends on br.edu.ufrgs.ucdesigner.visual.JDescriptionPanel*

Use case: UC007– Add Sep Information

Use case summary	
Use case index:	UC007
Use case name:	Add step information
Description:	User enters specific step information
References to class: br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog	
Ask user to enter information, step 2 of Main flow Ask the user to enter information accordingly to the type of the step. *Implemented by br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog*	
Check required data, step 5 of Main flow Confirm that all the required data was entered correctly. *Implemented by br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog*	

Use case: UC008 – Associate with Logical Model

Use case summary	
Use case index:	UC008
Use case name:	Associate with Logical Model
Description:	Associate use case step with the logical model.
References to class: br.edu.ufrgs.ucdesigner.model.activity.ActivityElement	
Associate class to step, step 6 of Main flow Associate the selected class to the step using the association type selected. *Implemented by br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
Remove the association, step 5 of Delete association Remove the association between the step and the class. *Implemented by br.edu.ufrgs.ucdesigner.model.activity.ActivityElement*	
References to class: br.edu.ufrgs.ucdesigner.model.activity.ActivityToModel	
Associate class to step, step 6 of Main flow Associate the selected class to the step using the association type selected. *Implemented by br.edu.ufrgs.ucdesigner.model.activity.ActivityToModel*	
Remove the association, step 5 of Delete association Remove the association between the step and the class. *Implemented by br.edu.ufrgs.ucdesigner.model.activity.ActivityToModel*	
References to class: br.edu.ufrgs.ucdesigner.model.logical.Class	

Associate class to step, step 6 of Main flow Associate the selected class to the step using the association type selected. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Class*
References to class: br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog
Confirm data, step 5 of Main flow Confirms that no other association exists to the step for the same class and association type. *Implemented by br.edu.ufrgs.ucdesigner.visual.elements.JActivityElementDialog*

Use case: UC009 – Export Description from Template

Use case summary	
Use case index:	UC009
Use case name:	Export description from template
Description:	Export the use case description based on a template.
References to class: br.edu.ufrgs.ucdesigner.UCDesigner	
Save result, step 6 of Main flow Save result export into an appropriate file in the workdir. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
Call application, step 7 of Main flow Call the appropriate application to open the exported documentation. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
References to class: br.edu.ufrgs.ucdesigner.xml.ProjectParser	
Process with the export format, step 5 of Main flow Process the fetched XML description with the XSL obtained from the export format. *Implemented by br.edu.ufrgs.ucdesigner.xml.ProjectParser*	
References to class: br.edu.ufrgs.ucdesigner.xml.XMLActor	
Fetch XML description, step 4 of Main flow Fetch the XML description of the selected use case *Implemented by br.edu.ufrgs.ucdesigner.xml.XMLActor*	
References to class: br.edu.ufrgs.ucdesigner.xml.XMLContent	
Fetch XML description, step 4 of Main flow Fetch the XML description of the selected use case *Depends on br.edu.ufrgs.ucdesigner.xml.XMLContent*	
References to class: br.edu.ufrgs.ucdesigner.xml.XMLProject	
Process with the export format, step 5 of Main flow Process the fetched XML description with the XSL obtained from the export format. *Depends on br.edu.ufrgs.ucdesigner.xml.XMLProject*	
Fetch XML description, step 4 of Main flow Fetch the XML description of the selected use case *Implemented by br.edu.ufrgs.ucdesigner.xml.XMLProject*	
References to class: br.edu.ufrgs.ucdesigner.xml.XMLUseCase	
Fetch XML description, step 4 of Main flow Fetch the XML description of the selected use case *Implemented by br.edu.ufrgs.ucdesigner.xml.XMLUseCase*	

Use case: UC010 – Compile Use Case

Use case summary	
Use case index:	UC010

Referências ao modelo lógico (logicalReferences.xsl)

Use case name:	Compile use case
Description:	System compile the use case.
References to class: br.edu.ufrgs.ucdesigner.compiler.UCPCompiler	
Confirm actors, step 2 of Main flow Confirm that actors where defined for the use case. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
Confirm elements, step 3 of Main flow Confirm that elements where defined *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
Confirm start, step 4 of Main flow Confirm that start step exists *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
Confirm ends, step 5 of Main flow Confirm that ends where defined for the use case. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
Confirm alternatives, step 6 of Main flow Confirm that all alternatives have and end or a return. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No actors, step 1 of No actors No actors where defined. Add a compile message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No elements, step 1 of No elements No elements where defined for the description. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No start, step 1 of No start No start was defined. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No ends, step 1 of No ends No ends were defined. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
Incomplete alternatives, step 1 of Alternatives failed Some alternative is not complete. Add message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
References to class: br.edu.ufrgs.ucdesigner.compiler.UCPMessage	
No actors, step 1 of No actors No actors where defined. Add a compile message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPMessage*	
No elements, step 1 of No elements No elements where defined for the description. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPMessage*	
No start, step 1 of No start No start was defined. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPMessage*	
No ends, step 1 of No ends No ends were defined. Add a message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPMessage*	
Incomplete alternatives, step 1 of Alternatives failed Some alternative is not complete. Add message. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPMessage*	

Use case: UC011 – View Project Tree

Referências ao modelo lógico (logicalReferences.xsl)

Use case summary	
Use case index:	UC011
Use case name:	View project tree
Description:	Displays the project tree.
References to class: br.edu.ufrgs.ucdesigner.model.logical.Class	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Class*	
Show classes and implementation packages, step 7 of Main flow Show all classes and implementation packages within the fixed 'Classes' package. *Depends on br.edu.ufrgs.ucdesigner.model.logical.Class*	
References to class: br.edu.ufrgs.ucdesigner.model.Package	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	
Show analysis packages, step 4 of Main flow Show all analysis packages whitin the fixed 'Analysis package' package. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	
Show classes and implementation packages, step 7 of Main flow Show all classes and implementation packages within the fixed 'Classes' package. *Depends on br.edu.ufrgs.ucdesigner.model.Package*	
Create fixed packages, step 3 of Main flow Create fixed packages for 'Analysis Packages', 'Actors','Use Cases' and 'Classes'. *Implemented by br.edu.ufrgs.ucdesigner.model.Package*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.Actor	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*	
Show Actors, step 5 of Main flow Show all actors found within their packages and within the 'Actors' fixed package.. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.Actor*	
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCase	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
Show use cases, step 6 of Main flow Show all use cases found within their packages and within the fixed 'Use case' package. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
References to class: br.edu.ufrgs.ucdesigner.project.Project	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Implemented by br.edu.ufrgs.ucdesigner.project.Project*	
References to class: br.edu.ufrgs.ucdesigner.UCDesigner	
Fetch all elements, step 2 of Main flow Fetch all elements to be shown in the project tree. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
Create fixed packages, step 3 of Main flow Create fixed packages for 'Analysis Packages', 'Actors','Use Cases' and 'Classes'. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	
Show analysis packages, step 4 of Main flow Show all analysis packages whitin the fixed 'Analysis package' package. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*	

Show Actors, step 5 of Main flow Show all actors found within their packages and within the 'Actors' fixed package.. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*
Show use cases, step 6 of Main flow Show all use cases found within their packages and within the fixed 'Use case' package. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*
Show classes and implementation packages, step 7 of Main flow Show all classes and implementation packages within the fixed 'Classes' package. *Implemented by br.edu.ufrgs.ucdesigner.UCDesigner*

Use case: UC012 – Open Use Case

Use case summary	
Use case index:	UC012
Use case name:	Open use case
Description:	Open use case and display it in the main window.
References to class: br.edu.ufrgs.ucdesigner.model.usecase.UseCase	
View description, step 3 of Main flow Confirms that description exists for at least one scenario and shows the description. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
Open use case dialog, step 1 of No description available Confirms that no description is available. Open the use case dialog. *Depends on br.edu.ufrgs.ucdesigner.model.usecase.UseCase*	
References to class: br.edu.ufrgs.ucdesigner.visual.JScenarioDescription	
View description, step 3 of Main flow Confirms that description exists for at least one scenario and shows the description. *Implemented by br.edu.ufrgs.ucdesigner.visual.JScenarioDescription*	
References to class: br.edu.ufrgs.ucdesigner.visual.JUseCase	
Open use case dialog, step 1 of No description available Confirms that no description is available. Open the use case dialog. *Implemented by br.edu.ufrgs.ucdesigner.visual.JUseCase*	
References to class: br.edu.ufrgs.ucdesigner.visual.UCDescription	
View description, step 3 of Main flow Confirms that description exists for at least one scenario and shows the description. *Implemented by br.edu.ufrgs.ucdesigner.visual.UCDescription*	

Use case: UC013 – Display Compile Message

Use case summary	
Use case index:	UC013
Use case name:	Display compile messages
Description:	System displays the compile messages.
References to class: br.edu.ufrgs.ucdesigner.compiler.UCPCompiler	
Fecth all messages, step 2 of Main flow Fetch all messages from the compiler. *Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler*	
No messages found, step 1 of No messages No messages were found.	

Referências ao modelo lógico (logicalReferences.xsl)

Implemented by br.edu.ufrgs.ucdesigner.compiler.UCPCompiler
References to class: br.edu.ufrgs.ucdesigner.compiler.UCPMessage
Fecth all messages, step 2 of Main flow Fetch all messages from the compiler. *Depends on br.edu.ufrgs.ucdesigner.compiler.UCPMessage*
Display messages, step 3 of Main flow Displays all messages with the message description, the severity and the element that originated the alert. *Depends on br.edu.ufrgs.ucdesigner.compiler.UCPMessage*
References to class: br.edu.ufrgs.ucdesigner.visual.UCPMessageCellRenderer
Display messages, step 3 of Main flow Displays all messages with the message description, the severity and the element that originated the alert. *Depends on br.edu.ufrgs.ucdesigner.visual.UCPMessageCellRenderer*
References to class: br.edu.ufrgs.ucdesigner.visual.UCPMessageTableModel
Display messages, step 3 of Main flow Displays all messages with the message description, the severity and the element that originated the alert. *Implemented by br.edu.ufrgs.ucdesigner.visual.UCPMessageTableModel*
Don't show the section, step 2 of No messages The message section should not be shown. *Implemented by br.edu.ufrgs.ucdesigner.visual.UCPMessageTableModel*

Java test cases gerados (javaTestCases.xsl)

Use case: UC001 – Enter the tool

```
1
2  /*
3   * Class TestCaseEnterthetool
4   * Generated for testing the use case Enter the tool
5   */
6  public class TestCaseEnterthetool extends TestCase {
7
8
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioLastprojectexists();
23
24         testScenarioMainscenarioOpensaproject();
25
26         testScenarioMainscenarioAborttool();
27
28     }
29
30     public void cleanupScenarioMainscenario() {
31         // TODO: insert your code here
32     }
33
34    /*
35     * Test for Main flow
36     *
37     */
38     public void testScenarioMainscenarioMainflow() {
39         // TODO: insert your code here for preparing and finishing the flow test
40
41         // Go to Main flow
42         runStepMainflowToolisopened();
43
44         runStepMainflowNolastprojectfound();
45
46         runStepMainflowDisplay_Newproject_section();
47
48         runStepMainflowDisplay_Openproject_section();
49
50         runStepMainflowSelectstocreateaproject();
51
52         runStepMainflowCreateanewproject();
53
54         runStepMainflowEnd_success_();
55         // Use case ends
56     }
57
58    /*
59     * Test for Last project exists
60     *
61     */
62     public void testScenarioMainscenarioLastprojectexists() {
63         // TODO: insert your code here for preparing and finishing the flow test
64
65         // Go to Main flow
66         runStepMainflowToolisopened();
67         // Go to Last project exists
68         runStepLastprojectexistsLastprojectfound();
69
70         runStepLastprojectexistsDisplays_Lastproject_section();
71         // Return to Main flow
72         runStepMainflowDisplay_Newproject_section();
```

```
73
74          runStepMainflowDisplay_Openproject_section();
75
76          runStepMainflowSelectstocreateaproject();
77
78          runStepMainflowCreateanewproject();
79
80          runStepMainflowEnd_success_();
81          // Use case ends
82      }
83
84      /*
85       * Test for Opens a project
86       *
87       */
88      public void testScenarioMainscenarioOpensaproject() {
89          // TODO: insert your code here for preparing and finishing the flow test
90
91          // Go to Main flow
92          runStepMainflowToolisopened();
93
94          runStepMainflowNolastprojectfound();
95
96          runStepMainflowDisplay_Newproject_section();
97
98          runStepMainflowDisplay_Openproject_section();
99          // Go to Opens a project
100         runStepOpensaprojectSelectstoopenaproject();
101
102         runStepOpensaprojectOpenaproject();
103
104         runStepOpensaprojectEnd_success_();
105         // Use case ends
106     }
107
108     /*
109      * Test for Abort tool
110      *
111      */
112     public void testScenarioMainscenarioAborttool() {
113         // TODO: insert your code here for preparing and finishing the flow test
114
115         // Go to Main flow
116         runStepMainflowToolisopened();
117
118         runStepMainflowNolastprojectfound();
119
120         runStepMainflowDisplay_Newproject_section();
121
122         runStepMainflowDisplay_Openproject_section();
123         // Go to Abort tool
124         runStepAborttoolSelects_Cancel_();
125
126         runStepAborttoolEnd_abort_();
127         // Use case ends
128     }
129
130     /*
131      * Steps for use case Enter the tool
132      */
133
134     public void runStepMainflowToolisopened() {
135         //TODO: insert your code here for implementing this step
136     }
137
138     public void runStepMainflowNolastprojectfound() {
139         //TODO: insert your code here for implementing this step
140     }
141
142     public void runStepMainflowDisplay_Newproject_section() {
143         //TODO: insert your code here for implementing this step
144     }
145
146     public void runStepMainflowDisplay_Openproject_section() {
147         //TODO: insert your code here for implementing this step
148     }
149
150     public void runStepMainflowSelectstocreateaproject() {
151         //TODO: insert your code here for implementing this step
152     }
153
154     public void runStepMainflowCreateanewproject() {
```

Java test cases gerados (javaTestCases.xsl)

```
155         //TODO: insert your code here for implementing this step
156     }
157
158     public void runStepMainflowEnd_success_() {
159         //TODO: insert your code here for implementing this step
160     }
161
162     public void runStepLastprojectexistsLastprojectfound() {
163         //TODO: insert your code here for implementing this step
164     }
165
166     public void runStepLastprojectexistsDisplays_Lastproject_section() {
167         //TODO: insert your code here for implementing this step
168     }
169
170     public void runStepOpensaprojectSelectstoopenaproject() {
171         //TODO: insert your code here for implementing this step
172     }
173
174     public void runStepOpensaprojectOpenaproject() {
175         //TODO: insert your code here for implementing this step
176     }
177
178     public void runStepOpensaprojectEnd_success_() {
179         //TODO: insert your code here for implementing this step
180     }
181
182     public void runStepAborttoolSelects_Cancel_() {
183         //TODO: insert your code here for implementing this step
184     }
185
186     public void runStepAborttoolEnd_abort_() {
187         //TODO: insert your code here for implementing this step
188     }
189
190 }
191
192 // Generated by UCDesigner 1.0
```

Use case: UC002 – View Project

```
 1
 2 /*
 3  * Class TestCaseViewProject
 4  * Generated for testing the use case View Project
 5  */
 6 public class TestCaseViewProject extends TestCase {
 7
 8
 9     /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioNousecases();
23
24     }
25
26     public void cleanupScenarioMainscenario() {
27         // TODO: insert your code here
28     }
29
30    /*
31     * Test for Main flow
32     *
33     */
34     public void testScenarioMainscenarioMainflow() {
35         // TODO: insert your code here for preparing and finishing the flow test
36
37         // Go to Main flow
38         runStepMainflowStart();
```

Java test cases gerados (javaTestCases.xsl)

```
39
40          runStepMainflowLoadprojectinfo();
41
42          runStepMainflowLoadprojectelements();
43
44          runStepMainflowViewtree();
45
46          runStepMainflowHasoneormoreusecases();
47
48          runStepMainflowCompileusecases();
49
50          runStepMainflowViewmessages();
51
52          runStepMainflowEnd();
53          // Use case ends
54      }
55
56      /*
57       * Test for No use cases
58       *
59       */
60      public void testScenarioMainscenarioNousecases() {
61          // TODO: insert your code here for preparing and finishing the flow test
62
63          // Go to Main flow
64          runStepMainflowStart();
65
66          runStepMainflowLoadprojectinfo();
67
68          runStepMainflowLoadprojectelements();
69
70          runStepMainflowViewtree();
71          // Go to No use cases
72          runStepNousecasesNousecases();
73          // Return to Main flow
74          runStepMainflowViewmessages();
75
76          runStepMainflowEnd();
77          // Use case ends
78      }
79
80      /*
81       * Steps for use case View Project
82       */
83
84      public void runStepMainflowStart() {
85          //TODO: insert your code here for implementing this step
86      }
87
88      public void runStepMainflowLoadprojectinfo() {
89          //TODO: insert your code here for implementing this step
90      }
91
92      public void runStepMainflowLoadprojectelements() {
93          //TODO: insert your code here for implementing this step
94      }
95
96      public void runStepMainflowViewtree() {
97          //TODO: insert your code here for implementing this step
98      }
99
100     public void runStepMainflowHasoneormoreusecases() {
101         //TODO: insert your code here for implementing this step
102     }
103
104     public void runStepMainflowCompileusecases() {
105         //TODO: insert your code here for implementing this step
106     }
107
108     public void runStepMainflowViewmessages() {
109         //TODO: insert your code here for implementing this step
110     }
111
112     public void runStepMainflowEnd() {
113         //TODO: insert your code here for implementing this step
114     }
115
116     public void runStepNousecasesNousecases() {
117         //TODO: insert your code here for implementing this step
118     }
119
120 }
```

Java test cases gerados (javaTestCases.xsl)

```

// Generated by UCDesigner 1.0
```

Use case: UC003 – Import Classes From Folder

```

/*
 * Class TestCaseImportclassesfromfolder
 * Generated for testing the use case Import classes from folder
 */
public class TestCaseImportclassesfromfolder extends TestCase {


    /*
     * Test for Scenario Main scenario
     */
    public void setupScenarioMainscenario() {
        // TODO: insert your code here for preparing the scenario test
    }

    public void testScenarioMainscenario() {
        // TODO: insert your code here for starting and finishing the scenario test
        // Testing all flows for the scenario

        testScenarioMainscenarioMainflow();

        testScenarioMainscenarioUseraborts();

        testScenarioMainscenarioRefreshaction();

    }

    public void cleanupScenarioMainscenario() {
        // TODO: insert your code here
    }

   /*
    * Test for Main flow
    *
    */
    public void testScenarioMainscenarioMainflow() {
        // TODO: insert your code here for preparing and finishing the flow test

        // Go to Main flow
        runStepMainflowStart();

        runStepMainflowAddafolder();

        runStepMainflowAskfolderlocation();

        runStepMainflowEnterfolderinformation();

        runStepMainflowSearchclasses();

        runStepMainflowAddinformationtofolder();

        runStepMainflowSuccess();
        // Use case ends
    }

   /*
    * Test for User aborts
    *
    */
    public void testScenarioMainscenarioUseraborts() {
        // TODO: insert your code here for preparing and finishing the flow test

        // Go to Main flow
        runStepMainflowStart();

        runStepMainflowAddafolder();

        runStepMainflowAskfolderlocation();
        // Go to User aborts
        runStepUserabortsCanceloperation();

        runStepUserabortsAbortoperation();
        // Use case ends
```

Java test cases gerados (javaTestCases.xsl)

```
74      }
75
76     /*
77      * Test for Refresh action
78      *
79      */
80      public void testScenarioMainscenarioRefreshaction() {
81          // TODO: insert your code here for preparing and finishing the flow test
82
83          // Go to Main flow
84          runStepMainflowStart();
85          // Go to Refresh action
86          runStepRefreshactionRefershfolder();
87
88          runStepRefreshactionClearfolder();
89          // Return to Main flow
90          runStepMainflowSearchclasses();
91
92          runStepMainflowAddinformationtofolder();
93
94          runStepMainflowSuccess();
95          // Use case ends
96      }
97
98      /*
99       * Steps for use case Import classes from folder
100      */
101
102     public void runStepMainflowStart() {
103         //TODO: insert your code here for implementing this step
104     }
105
106     public void runStepMainflowAddafolder() {
107         //TODO: insert your code here for implementing this step
108     }
109
110     public void runStepMainflowAskfolderlocation() {
111         //TODO: insert your code here for implementing this step
112     }
113
114     public void runStepMainflowEnterfolderinformation() {
115         //TODO: insert your code here for implementing this step
116     }
117
118     public void runStepMainflowSearchclasses() {
119         //TODO: insert your code here for implementing this step
120     }
121
122     public void runStepMainflowAddinformationtofolder() {
123         //TODO: insert your code here for implementing this step
124     }
125
126     public void runStepMainflowSuccess() {
127         //TODO: insert your code here for implementing this step
128     }
129
130     public void runStepUserabortsCanceloperation() {
131         //TODO: insert your code here for implementing this step
132     }
133
134     public void runStepUserabortsAbortoperation() {
135         //TODO: insert your code here for implementing this step
136     }
137
138     public void runStepRefreshactionRefershfolder() {
139         //TODO: insert your code here for implementing this step
140     }
141
142     public void runStepRefreshactionClearfolder() {
143         //TODO: insert your code here for implementing this step
144     }
145
146 }
147
148 // Generated by UCDesigner 1.0
```

Use case: UC004 – Add Class

Java test cases gerados (javaTestCases.xsl)

```
1
2  /*
3   * Class TestCaseAddclass
4   * Generated for testing the use case Add class
5   */
6  public class TestCaseAddclass extends TestCase {
7
8
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioNewsubpackage();
23
24         testScenarioMainscenarioPackageconflict();
25
26         testScenarioMainscenarioClassconflict();
27
28         testScenarioMainscenarioCancelpackage();
29
30         testScenarioMainscenarioCancelclassaddition();
31
32     }
33
34     public void cleanupScenarioMainscenario() {
35         // TODO: insert your code here
36     }
37
38    /*
39     * Test for Main flow
40     *
41     */
42     public void testScenarioMainscenarioMainflow() {
43         // TODO: insert your code here for preparing and finishing the flow test
44
45         // Go to Main flow
46         runStepMainflowStart();
47
48         runStepMainflowSelectapackage();
49
50         runStepMainflowAddanewclass();
51
52         runStepMainflowAskclassinformation();
53
54         runStepMainflowEnterclassinformation();
55
56         runStepMainflowCheckclassinformation();
57
58         runStepMainflowAddclass();
59
60         runStepMainflowSuccess();
61         // Use case ends
62     }
63
64    /*
65     * Test for New subpackage
66     *
67     */
68     public void testScenarioMainscenarioNewsubpackage() {
69         // TODO: insert your code here for preparing and finishing the flow test
70
71         // Go to Main flow
72         runStepMainflowStart();
73
74         runStepMainflowSelectapackage();
75         // Go to New subpackage
76         runStepNewsubpackageAddanewpackage();
77
78         runStepNewsubpackageAskpackageinformation();
79
80         runStepNewsubpackageEnterpackageinformation();
81
82         runStepNewsubpackageCheckpackageinformation();
```

Java test cases gerados (javaTestCases.xsl)

```
83
84          runStepNewsubpackageAddpackage();
85          // Return to Main flow
86          runStepMainflowSelectapackage();
87
88          runStepMainflowAddanewclass();
89
90          runStepMainflowAskclassinformation();
91
92          runStepMainflowEnterclassinformation();
93
94          runStepMainflowCheckclassinformation();
95
96          runStepMainflowAddclass();
97
98          runStepMainflowSuccess();
99          // Use case ends
100     }
101
102    /*
103     * Test for Package conflict
104     *
105     */
106     public void testScenarioMainscenarioPackageconflict() {
107         // TODO: insert your code here for preparing and finishing the flow test
108
109         // Go to Main flow
110         runStepMainflowStart();
111
112         runStepMainflowSelectapackage();
113         // Go to New subpackage
114         runStepNewsubpackageAddanewpackage();
115
116         runStepNewsubpackageAskpackageinformation();
117
118         runStepNewsubpackageEnterpackageinformation();
119
120         runStepNewsubpackageCheckpackageinformation();
121         // Go to Package conflict
122         runStepPackageconflictPackageconflict();
123
124         runStepPackageconflictAbortpackage();
125         // Use case ends
126     }
127
128    /*
129     * Test for Class conflict
130     *
131     */
132     public void testScenarioMainscenarioClassconflict() {
133         // TODO: insert your code here for preparing and finishing the flow test
134
135         // Go to Main flow
136         runStepMainflowStart();
137
138         runStepMainflowSelectapackage();
139
140         runStepMainflowAddanewclass();
141
142         runStepMainflowAskclassinformation();
143
144         runStepMainflowEnterclassinformation();
145
146         runStepMainflowCheckclassinformation();
147         // Go to Class conflict
148         runStepClassconflictClassconflict();
149
150         runStepClassconflictAbortclass();
151         // Use case ends
152     }
153
154    /*
155     * Test for Cancel package
156     *
157     */
158     public void testScenarioMainscenarioCancelpackage() {
159         // TODO: insert your code here for preparing and finishing the flow test
160
161         // Go to Main flow
162         runStepMainflowStart();
163
164         runStepMainflowSelectapackage();
```

Java test cases gerados (javaTestCases.xsl)

```
165         // Go to New subpackage
166         runStepNewsubpackageAddanewpackage();
167 
168         runStepNewsubpackageAskpackageinformation();
169         // Go to Cancel package
170         runStepCancelpackageCancelpackageaddition();
171         // Return to Package conflict
172         runStepPackageconflictAbortpackage();
173         // Use case ends
174     }
175 
176    /*
177     * Test for Cancel class addition
178     *
179     */
180     public void testScenarioMainscenarioCancelclassaddition() {
181         // TODO: insert your code here for preparing and finishing the flow test
182 
183         // Go to Main flow
184         runStepMainflowStart();
185 
186         runStepMainflowSelectapackage();
187 
188         runStepMainflowAddanewclass();
189 
190         runStepMainflowAskclassinformation();
191         // Go to Cancel class addition
192         runStepCancelclassadditionCancelclassaddition();
193         // Return to Class conflict
194         runStepClassconflictAbortclass();
195         // Use case ends
196     }
197 
198     /*
199      * Steps for use case Add class
200      */
201 
202     public void runStepMainflowStart() {
203         //TODO: insert your code here for implementing this step
204     }
205 
206     public void runStepMainflowSelectapackage() {
207         //TODO: insert your code here for implementing this step
208     }
209 
210     public void runStepMainflowAddanewclass() {
211         //TODO: insert your code here for implementing this step
212     }
213 
214     public void runStepMainflowAskclassinformation() {
215         //TODO: insert your code here for implementing this step
216     }
217 
218     public void runStepMainflowEnterclassinformation() {
219         //TODO: insert your code here for implementing this step
220     }
221 
222     public void runStepMainflowCheckclassinformation() {
223         //TODO: insert your code here for implementing this step
224     }
225 
226     public void runStepMainflowAddclass() {
227         //TODO: insert your code here for implementing this step
228     }
229 
230     public void runStepMainflowSuccess() {
231         //TODO: insert your code here for implementing this step
232     }
233 
234     public void runStepNewsubpackageAddanewpackage() {
235         //TODO: insert your code here for implementing this step
236     }
237 
238     public void runStepNewsubpackageAskpackageinformation() {
239         //TODO: insert your code here for implementing this step
240     }
241 
242     public void runStepNewsubpackageEnterpackageinformation() {
243         //TODO: insert your code here for implementing this step
244     }
245 
246     public void runStepNewsubpackageCheckpackageinformation() {
```

Java test cases gerados (javaTestCases.xsl)

```
247         //TODO: insert your code here for implementing this step
248     }
249
250     public void runStepNewsubpackageAddpackage() {
251         //TODO: insert your code here for implementing this step
252     }
253
254     public void runStepPackageconflictPackageconflict() {
255         //TODO: insert your code here for implementing this step
256     }
257
258     public void runStepPackageconflictAbortpackage() {
259         //TODO: insert your code here for implementing this step
260     }
261
262     public void runStepClassconflictClassconflict() {
263         //TODO: insert your code here for implementing this step
264     }
265
266     public void runStepClassconflictAbortclass() {
267         //TODO: insert your code here for implementing this step
268     }
269
270     public void runStepCancelpackageCancelpackageaddition() {
271         //TODO: insert your code here for implementing this step
272     }
273
274     public void runStepCancelclassadditionCancelclassaddition() {
275         //TODO: insert your code here for implementing this step
276     }
277
278 }
279
280 // Generated by UCDesigner 1.0
```

Use case: UC005 – Add Use Case

```
 1
 2 /*
 3  * Class TestCaseAddusecase
 4  * Generated for testing the use case Add use case
 5  */
 6 public class TestCaseAddusecase extends TestCase {
 7
 8
 9     /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioAddfrompackage();
23
24         testScenarioMainscenarioUserabort();
25
26         testScenarioMainscenarioAddactor();
27
28         testScenarioMainscenarioUserabortactoraddition();
29
30         testScenarioMainscenarioActorconflict();
31
32         testScenarioMainscenarioUsecaseconflict();
33
34     }
35
36     public void cleanupScenarioMainscenario() {
37         // TODO: insert your code here
38     }
39
40    /*
41     * Test for Main flow
42     *
```

Java test cases gerados (javaTestCases.xsl)

```
43      */
44      public void testScenarioMainscenarioMainflow() {
45          // TODO: insert your code here for preparing and finishing the flow test
46
47          // Go to Main flow
48          runStepMainflowStart();
49
50          runStepMainflowAddusecase();
51
52          runStepMainflowAskforinformation();
53
54          runStepMainflowEnterinformation();
55
56          runStepMainflowCheckinformation();
57
58          runStepMainflowAddusecase();
59
60          runStepMainflowSuccess();
61          // Use case ends
62      }
63
64     /*
65      * Test for Add from package
66      *
67      */
68      public void testScenarioMainscenarioAddfrompackage() {
69          // TODO: insert your code here for preparing and finishing the flow test
70
71          // Go to Main flow
72          runStepMainflowStart();
73          // Go to Add from package
74          runStepAddfrompackageAddusecasefrompackagetree();
75
76          runStepAddfrompackagePopulatepackageinformation();
77          // Return to Main flow
78          runStepMainflowAskforinformation();
79
80          runStepMainflowEnterinformation();
81
82          runStepMainflowCheckinformation();
83
84          runStepMainflowAddusecase();
85
86          runStepMainflowSuccess();
87          // Use case ends
88      }
89
90     /*
91      * Test for User abort
92      *
93      */
94      public void testScenarioMainscenarioUserabort() {
95          // TODO: insert your code here for preparing and finishing the flow test
96
97          // Go to Main flow
98          runStepMainflowStart();
99
100         runStepMainflowAddusecase();
101
102         runStepMainflowAskforinformation();
103         // Go to User abort
104         runStepUserabortCancel();
105
106         runStepUserabortAbort();
107         // Use case ends
108     }
109
110    /*
111     * Test for Add actor
112     *
113     */
114     public void testScenarioMainscenarioAddactor() {
115         // TODO: insert your code here for preparing and finishing the flow test
116
117         // Go to Main flow
118         runStepMainflowStart();
119
120         runStepMainflowAddusecase();
121
122         runStepMainflowAskforinformation();
123
124         runStepMainflowEnterinformation();
```

```
125         // Go to Add actor
126         runStepAddactorAddactor();
127 
128         runStepAddactorAskforactorinformation();
129 
130         runStepAddactorEnteractorinformation();
131 
132         runStepAddactorCheckactorinformation();
133 
134         runStepAddactorAddactor();
135         // Return to Main flow
136         runStepMainflowEnterinformation();
137 
138         runStepMainflowCheckinformation();
139 
140         runStepMainflowAddusecase();
141 
142         runStepMainflowSuccess();
143         // Use case ends
144     }
145 
146     /*
147      * Test for User abort actor addition
148      *
149      */
150     public void testScenarioMainscenarioUserabortactoraddition() {
151         // TODO: insert your code here for preparing and finishing the flow test
152 
153         // Go to Main flow
154         runStepMainflowStart();
155 
156         runStepMainflowAddusecase();
157 
158         runStepMainflowAskforinformation();
159 
160         runStepMainflowEnterinformation();
161         // Go to Add actor
162         runStepAddactorAddactor();
163 
164         runStepAddactorAskforactorinformation();
165         // Go to User abort actor addition
166         runStepUserabortactoradditionCancelactor();
167         // Return to Main flow
168         runStepMainflowEnterinformation();
169 
170         runStepMainflowCheckinformation();
171 
172         runStepMainflowAddusecase();
173 
174         runStepMainflowSuccess();
175         // Use case ends
176     }
177 
178     /*
179      * Test for Actor conflict
180      *
181      */
182     public void testScenarioMainscenarioActorconflict() {
183         // TODO: insert your code here for preparing and finishing the flow test
184 
185         // Go to Main flow
186         runStepMainflowStart();
187 
188         runStepMainflowAddusecase();
189 
190         runStepMainflowAskforinformation();
191 
192         runStepMainflowEnterinformation();
193         // Go to Add actor
194         runStepAddactorAddactor();
195 
196         runStepAddactorAskforactorinformation();
197 
198         runStepAddactorEnteractorinformation();
199 
200         runStepAddactorCheckactorinformation();
201         // Go to Actor conflict
202         runStepActorconflictActorconflict();
203         // Return to Main flow
204         runStepMainflowEnterinformation();
205 
206         runStepMainflowCheckinformation();
```

Java test cases gerados (javaTestCases.xsl)

```
207
208         runStepMainflowAddusecase();
209
210         runStepMainflowSuccess();
211         // Use case ends
212     }
213
214     /*
215      * Test for Use case conflict
216      *
217      */
218     public void testScenarioMainscenarioUsecaseconflict() {
219         // TODO: insert your code here for preparing and finishing the flow test
220
221         // Go to Main flow
222         runStepMainflowStart();
223
224         runStepMainflowAddusecase();
225
226         runStepMainflowAskforinformation();
227
228         runStepMainflowEnterinformation();
229
230         runStepMainflowCheckinformation();
231         // Go to Use case conflict
232         runStepUsecaseconflictUsecaseconflict();
233         // Return to Main flow
234         runStepMainflowAskforinformation();
235
236         runStepMainflowEnterinformation();
237
238         runStepMainflowCheckinformation();
239
240         runStepMainflowAddusecase();
241
242         runStepMainflowSuccess();
243         // Use case ends
244     }
245
246     /*
247      * Steps for use case Add use case
248      */
249
250     public void runStepMainflowStart() {
251         //TODO: insert your code here for implementing this step
252     }
253
254     public void runStepMainflowAddusecase() {
255         //TODO: insert your code here for implementing this step
256     }
257
258     public void runStepMainflowAskforinformation() {
259         //TODO: insert your code here for implementing this step
260     }
261
262     public void runStepMainflowEnterinformation() {
263         //TODO: insert your code here for implementing this step
264     }
265
266     public void runStepMainflowCheckinformation() {
267         //TODO: insert your code here for implementing this step
268     }
269
270     public void runStepMainflowAddusecase() {
271         //TODO: insert your code here for implementing this step
272     }
273
274     public void runStepMainflowSuccess() {
275         //TODO: insert your code here for implementing this step
276     }
277
278     public void runStepAddfrompackageAddusecasefrompackagetree() {
279         //TODO: insert your code here for implementing this step
280     }
281
282     public void runStepAddfrompackagePopulatepackageinformation() {
283         //TODO: insert your code here for implementing this step
284     }
285
286     public void runStepUserabortCancel() {
287         //TODO: insert your code here for implementing this step
288     }
```

Java test cases gerados (javaTestCases.xsl)

```
289
290     public void runStepUserabortAbort() {
291         //TODO: insert your code here for implementing this step
292     }
293
294     public void runStepAddactorAddactor() {
295         //TODO: insert your code here for implementing this step
296     }
297
298     public void runStepAddactorAskforactorinformation() {
299         //TODO: insert your code here for implementing this step
300     }
301
302     public void runStepAddactorEnteractorinformation() {
303         //TODO: insert your code here for implementing this step
304     }
305
306     public void runStepAddactorCheckactorinformation() {
307         //TODO: insert your code here for implementing this step
308     }
309
310     public void runStepAddactorAddactor() {
311         //TODO: insert your code here for implementing this step
312     }
313
314     public void runStepUserabortactoradditionCancelactor() {
315         //TODO: insert your code here for implementing this step
316     }
317
318     public void runStepActorconflictActorconflict() {
319         //TODO: insert your code here for implementing this step
320     }
321
322     public void runStepUsecaseconflictUsecaseconflict() {
323         //TODO: insert your code here for implementing this step
324     }
325
326 }
327
328 // Generated by UCDesigner 1.0
```

Use case: UC006 – Describe Scenario

```
1
2  /*
3   * Class TestCaseDescribescenario
4   * Generated for testing the use case Describe scenario
5   */
6  public class TestCaseDescribescenario extends TestCase {
7
8
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioAbortstepaddition();
23
24         testScenarioMainscenarioAddaconnection();
25
26         testScenarioMainscenarioRemoveanelement();
27
28         testScenarioMainscenarioCancelremoval();
29
30     }
31
32     public void cleanupScenarioMainscenario() {
33         // TODO: insert your code here
34     }
35
36    /*
```

Java test cases gerados (javaTestCases.xsl)

```
37      * Test for Main flow
38      *
39      */
40      public void testScenarioMainscenarioMainflow() {
41          // TODO: insert your code here for preparing and finishing the flow test
42
43          // Go to Main flow
44          runStepMainflowStart();
45
46          runStepMainflowSelectasteptoadd();
47
48          runStepMainflowAddsteptoaposition();
49
50          runStepMainflowAskuserforstepinformation();
51
52          runStepMainflowEnterstepinformation();
53
54          runStepMainflowAddsteptothedescription();
55
56          runStepMainflowCompiledescription();
57
58          runStepMainflowSuccess();
59          // Use case ends
60      }
61
62     /*
63      * Test for Abort step addition
64      *
65      */
66      public void testScenarioMainscenarioAbortstepaddition() {
67          // TODO: insert your code here for preparing and finishing the flow test
68
69          // Go to Main flow
70          runStepMainflowStart();
71
72          runStepMainflowSelectasteptoadd();
73
74          runStepMainflowAddsteptoaposition();
75
76          runStepMainflowAskuserforstepinformation();
77          // Go to Abort step addition
78          runStepAbortstepadditionCancelstepaddition();
79
80          runStepAbortstepadditionAbortstepaddition();
81          // Use case ends
82      }
83
84     /*
85      * Test for Add a connection
86      *
87      */
88      public void testScenarioMainscenarioAddaconnection() {
89          // TODO: insert your code here for preparing and finishing the flow test
90
91          // Go to Main flow
92          runStepMainflowStart();
93          // Go to Add a connection
94          runStepAddaconnectionSelecttoaddaconnection();
95
96          runStepAddaconnectionConnecttwosteps();
97
98          runStepAddaconnectionConnectsteps();
99          // Return to Main flow
100         runStepMainflowCompiledescription();
101
102         runStepMainflowSuccess();
103         // Use case ends
104     }
105
106    /*
107     * Test for Remove an element
108     *
109     */
110     public void testScenarioMainscenarioRemoveanelement() {
111         // TODO: insert your code here for preparing and finishing the flow test
112
113         // Go to Main flow
114         runStepMainflowStart();
115         // Go to Remove an element
116         runStepRemoveanelementRemoveanelement();
117
118         runStepRemoveanelementConfirmdelete();
```

Java test cases gerados (javaTestCases.xsl)

```
119
120         runStepRemoveanelementConfirmaction();
121
122         runStepRemoveanelementRemoveelement();
123         // Return to Main flow
124         runStepMainflowCompiledescription();
125
126         runStepMainflowSuccess();
127         // Use case ends
128     }
129
130    /*
131     * Test for Cancel removal
132     *
133     */
134     public void testScenarioMainscenarioCancelremoval() {
135         // TODO: insert your code here for preparing and finishing the flow test
136
137         // Go to Main flow
138         runStepMainflowStart();
139         // Go to Remove an element
140         runStepRemoveanelementRemoveanelement();
141
142         runStepRemoveanelementConfirmdelete();
143         // Go to Cancel removal
144         runStepCancelremovalCancelremoval();
145
146         runStepCancelremovalAbortelementremoval();
147         // Use case ends
148     }
149
150     /*
151      * Steps for use case Describe scenario
152      */
153
154     public void runStepMainflowStart() {
155         //TODO: insert your code here for implementing this step
156     }
157
158     public void runStepMainflowSelectasteptoadd() {
159         //TODO: insert your code here for implementing this step
160     }
161
162     public void runStepMainflowAddsteptoaposition() {
163         //TODO: insert your code here for implementing this step
164     }
165
166     public void runStepMainflowAskuserforstepinformation() {
167         //TODO: insert your code here for implementing this step
168     }
169
170     public void runStepMainflowEnterstepinformation() {
171         //TODO: insert your code here for implementing this step
172     }
173
174     public void runStepMainflowAddsteptothedescription() {
175         //TODO: insert your code here for implementing this step
176     }
177
178     public void runStepMainflowCompiledescription() {
179         //TODO: insert your code here for implementing this step
180     }
181
182     public void runStepMainflowSuccess() {
183         //TODO: insert your code here for implementing this step
184     }
185
186     public void runStepAbortstepadditionCancelstepaddition() {
187         //TODO: insert your code here for implementing this step
188     }
189
190     public void runStepAbortstepadditionAbortstepaddition() {
191         //TODO: insert your code here for implementing this step
192     }
193
194     public void runStepAddaconnectionSelecttoaddaconnection() {
195         //TODO: insert your code here for implementing this step
196     }
197
198     public void runStepAddaconnectionConnecttwosteps() {
199         //TODO: insert your code here for implementing this step
200     }
```

Java test cases gerados (javaTestCases.xsl)

```
201
202     public void runStepAddaconnectionConnectsteps() {
203         //TODO: insert your code here for implementing this step
204     }
205
206     public void runStepRemoveanelementRemoveanelement() {
207         //TODO: insert your code here for implementing this step
208     }
209
210     public void runStepRemoveanelementConfirmdelete() {
211         //TODO: insert your code here for implementing this step
212     }
213
214     public void runStepRemoveanelementConfirmaction() {
215         //TODO: insert your code here for implementing this step
216     }
217
218     public void runStepRemoveanelementRemoveelement() {
219         //TODO: insert your code here for implementing this step
220     }
221
222     public void runStepCancelremovalCancelremoval() {
223         //TODO: insert your code here for implementing this step
224     }
225
226     public void runStepCancelremovalAbortelementremoval() {
227         //TODO: insert your code here for implementing this step
228     }
229
230 }
231
232 // Generated by UCDesigner 1.0
```

Use case: UC007– Add Sep Information

```
 1
 2 /*
 3  * Class TestCaseAddstepinformation
 4  * Generated for testing the use case Add step information
 5  */
 6 public class TestCaseAddstepinformation extends TestCase {
 7
 8
 9     /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioAbortaction();
23
24         testScenarioMainscenarioAssociatewithmodel();
25
26     }
27
28     public void cleanupScenarioMainscenario() {
29         // TODO: insert your code here
30     }
31
32    /*
33     * Test for Main flow
34     *
35     */
36     public void testScenarioMainscenarioMainflow() {
37         // TODO: insert your code here for preparing and finishing the flow test
38
39         // Go to Main flow
40         runStepMainflowStart();
41
42         runStepMainflowAskusertoenterinformation();
43
```

```
44          runStepMainflowEnterinformation();
45
46          runStepMainflowConfirmdatainput();
47
48          runStepMainflowCheckrequireddata();
49
50          runStepMainflowSuccess();
51          // Use case ends
52      }
53
54      /*
55       * Test for Abort action
56       *
57       */
58      public void testScenarioMainscenarioAbortaction() {
59          // TODO: insert your code here for preparing and finishing the flow test
60
61          // Go to Main flow
62          runStepMainflowStart();
63
64          runStepMainflowAskusertoenterinformation();
65
66          runStepMainflowEnterinformation();
67          // Go to Abort action
68          runStepAbortactionCanceldatainput();
69
70          runStepAbortactionAbortediting();
71          // Use case ends
72      }
73
74      /*
75       * Test for Associate with model
76       *
77       */
78      public void testScenarioMainscenarioAssociatewithmodel() {
79          // TODO: insert your code here for preparing and finishing the flow test
80
81          // Go to Main flow
82          runStepMainflowStart();
83
84          runStepMainflowAskusertoenterinformation();
85
86          runStepMainflowEnterinformation();
87          // Go to Associate with model
88          runStepAssociatewithmodelAssociatewithlogicalmodel();
89
90          runStepAssociatewithmodelDoassociation();
91          // Return to Main flow
92          runStepMainflowEnterinformation();
93
94          runStepMainflowConfirmdatainput();
95
96          runStepMainflowCheckrequireddata();
97
98          runStepMainflowSuccess();
99          // Use case ends
100     }
101
102     /*
103      * Steps for use case Add step information
104      */
105
106     public void runStepMainflowStart() {
107         //TODO: insert your code here for implementing this step
108     }
109
110     public void runStepMainflowAskusertoenterinformation() {
111         //TODO: insert your code here for implementing this step
112     }
113
114     public void runStepMainflowEnterinformation() {
115         //TODO: insert your code here for implementing this step
116     }
117
118     public void runStepMainflowConfirmdatainput() {
119         //TODO: insert your code here for implementing this step
120     }
121
122     public void runStepMainflowCheckrequireddata() {
123         //TODO: insert your code here for implementing this step
124     }
125
```

Java test cases gerados (javaTestCases.xsl)

```
126     public void runStepMainflowSuccess() {
127         //TODO: insert your code here for implementing this step
128     }
129
130     public void runStepAbortactionCanceldatainput() {
131         //TODO: insert your code here for implementing this step
132     }
133
134     public void runStepAbortactionAbortediting() {
135         //TODO: insert your code here for implementing this step
136     }
137
138     public void runStepAssociatewithmodelAssociatewithlogicalmodel() {
139         //TODO: insert your code here for implementing this step
140     }
141
142     public void runStepAssociatewithmodelDoassociation() {
143         //TODO: insert your code here for implementing this step
144     }
145
146 }
147
148 // Generated by UCDesigner 1.0
```

Use case: UC008 – Associate with Logical Model

```
1
2  /*
3   * Class TestCaseAssociatewithLogicalModel
4   * Generated for testing the use case Associate with Logical Model
5   */
6  public class TestCaseAssociatewithLogicalModel extends TestCase {
7
8
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioConflict();
23
24         testScenarioMainscenarioDeleteassociation();
25
26         testScenarioMainscenarioCancelassociationremove();
27
28     }
29
30     public void cleanupScenarioMainscenario() {
31         // TODO: insert your code here
32     }
33
34    /*
35     * Test for Main flow
36     *
37     */
38     public void testScenarioMainscenarioMainflow() {
39         // TODO: insert your code here for preparing and finishing the flow test
40
41         // Go to Main flow
42         runStepMainflowStart();
43
44         runStepMainflowSelectassociationtype();
45
46         runStepMainflowSelectclass();
47
48         runStepMainflowAssociate();
49
50         runStepMainflowConfirmdata();
51
52         runStepMainflowAssociateclasstostep();
```

Java test cases gerados (javaTestCases.xsl)

```
53
54          runStepMainflowAssociationcreated();
55          // Use case ends
56      }
57
58      /*
59       * Test for Conflict
60       *
61       */
62      public void testScenarioMainscenarioConflict() {
63          // TODO: insert your code here for preparing and finishing the flow test
64
65          // Go to Main flow
66          runStepMainflowStart();
67
68          runStepMainflowSelectassociationtype();
69
70          runStepMainflowSelectclass();
71
72          runStepMainflowAssociate();
73          // Go to Conflict
74          runStepConflictAlreadyassociated();
75
76          runStepConflictFailure();
77          // Use case ends
78      }
79
80      /*
81       * Test for Delete association
82       *
83       */
84      public void testScenarioMainscenarioDeleteassociation() {
85          // TODO: insert your code here for preparing and finishing the flow test
86
87          // Go to Main flow
88          runStepMainflowStart();
89          // Go to Delete association
90          runStepDeleteassociationSelectassociation();
91
92          runStepDeleteassociationPressdelete();
93
94          runStepDeleteassociationAskconfirmation();
95
96          runStepDeleteassociationConfirmremoval();
97
98          runStepDeleteassociationRemovetheassociation();
99
100         runStepDeleteassociationAssociationremove();
101         // Use case ends
102     }
103
104     /*
105      * Test for Cancel association remove
106      *
107      */
108     public void testScenarioMainscenarioCancelassociationremove() {
109         // TODO: insert your code here for preparing and finishing the flow test
110
111         // Go to Main flow
112         runStepMainflowStart();
113         // Go to Delete association
114         runStepDeleteassociationSelectassociation();
115
116         runStepDeleteassociationPressdelete();
117
118         runStepDeleteassociationAskconfirmation();
119         // Go to Cancel association remove
120         runStepCancelassociationremoveCancelremove();
121
122         runStepCancelassociationremoveCancelremove();
123         // Use case ends
124     }
125
126     /*
127      * Steps for use case Associate with Logical Model
128      */
129
130     public void runStepMainflowStart() {
131         //TODO: insert your code here for implementing this step
132     }
133
134     public void runStepMainflowSelectassociationtype() {
```

Java test cases gerados (javaTestCases.xsl)

```
135         //TODO: insert your code here for implementing this step
136     }
137
138     public void runStepMainflowSelectclass() {
139         //TODO: insert your code here for implementing this step
140     }
141
142     public void runStepMainflowAssociate() {
143         //TODO: insert your code here for implementing this step
144     }
145
146     public void runStepMainflowConfirmdata() {
147         //TODO: insert your code here for implementing this step
148     }
149
150     public void runStepMainflowAssociateclasstostep() {
151         //TODO: insert your code here for implementing this step
152     }
153
154     public void runStepMainflowAssociationcreated() {
155         //TODO: insert your code here for implementing this step
156     }
157
158     public void runStepConflictAlreadyassociated() {
159         //TODO: insert your code here for implementing this step
160     }
161
162     public void runStepConflictFailure() {
163         //TODO: insert your code here for implementing this step
164     }
165
166     public void runStepDeleteassociationSelectassociation() {
167         //TODO: insert your code here for implementing this step
168     }
169
170     public void runStepDeleteassociationPressdelete() {
171         //TODO: insert your code here for implementing this step
172     }
173
174     public void runStepDeleteassociationAskconfirmation() {
175         //TODO: insert your code here for implementing this step
176     }
177
178     public void runStepDeleteassociationConfirmremoval() {
179         //TODO: insert your code here for implementing this step
180     }
181
182     public void runStepDeleteassociationRemovetheassociation() {
183         //TODO: insert your code here for implementing this step
184     }
185
186     public void runStepDeleteassociationAssociationremove() {
187         //TODO: insert your code here for implementing this step
188     }
189
190     public void runStepCancelassociationremoveCancelremove() {
191         //TODO: insert your code here for implementing this step
192     }
193
194     public void runStepCancelassociationremoveCancelremove() {
195         //TODO: insert your code here for implementing this step
196     }
197
198 }
199
200 // Generated by UCDesigner 1.0
```

Use case: UC009 – Export Description from Template

```
1
2 /*
3  * Class TestCaseExportdescriptionfromtemplate
4  * Generated for testing the use case Export description from template
5  */
6 public class TestCaseExportdescriptionfromtemplate extends TestCase {
7
8
9     /*
10     * Test for Scenario Main scenario
11     */
```

Java test cases gerados (javaTestCases.xsl)

```
12      public void setupScenarioMainscenario() {
13          // TODO: insert your code here for preparing the scenario test
14      }
15
16      public void testScenarioMainscenario() {
17          // TODO: insert your code here for starting and finishing the scenario test
18          // Testing all flows for the scenario
19
20          testScenarioMainscenarioMainflow();
21
22      }
23
24      public void cleanupScenarioMainscenario() {
25          // TODO: insert your code here
26      }
27
28     /*
29      * Test for Main flow
30      *
31      */
32      public void testScenarioMainscenarioMainflow() {
33          // TODO: insert your code here for preparing and finishing the flow test
34
35          // Go to Main flow
36          runStepMainflowStart();
37
38          runStepMainflowSelectanusecase();
39
40          runStepMainflowSelectanexportformat();
41
42          runStepMainflowFetchXMLdescription();
43
44          runStepMainflowProcesswiththeexportformat();
45
46          runStepMainflowSaveresult();
47
48          runStepMainflowCallapplication();
49
50          runStepMainflowEnd();
51          // Use case ends
52      }
53
54      /*
55       * Steps for use case Export description from template
56       */
57
58      public void runStepMainflowStart() {
59          //TODO: insert your code here for implementing this step
60      }
61
62      public void runStepMainflowSelectanusecase() {
63          //TODO: insert your code here for implementing this step
64      }
65
66      public void runStepMainflowSelectanexportformat() {
67          //TODO: insert your code here for implementing this step
68      }
69
70      public void runStepMainflowFetchXMLdescription() {
71          //TODO: insert your code here for implementing this step
72      }
73
74      public void runStepMainflowProcesswiththeexportformat() {
75          //TODO: insert your code here for implementing this step
76      }
77
78      public void runStepMainflowSaveresult() {
79          //TODO: insert your code here for implementing this step
80      }
81
82      public void runStepMainflowCallapplication() {
83          //TODO: insert your code here for implementing this step
84      }
85
86      public void runStepMainflowEnd() {
87          //TODO: insert your code here for implementing this step
88      }
89
90  }
91
92  // Generated by UCDesigner 1.0
```

Java test cases gerados (javaTestCases.xsl)

Use case: UC010 – Compile Use Case

```
1
2  /*
3   * Class TestCaseCompileusecase
4   * Generated for testing the use case Compile use case
5   */
6  public class TestCaseCompileusecase extends TestCase {
7
8
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19
20         testScenarioMainscenarioMainflow();
21
22         testScenarioMainscenarioNoactors();
23
24         testScenarioMainscenarioNoelements();
25
26         testScenarioMainscenarioNostart();
27
28         testScenarioMainscenarioNoends();
29
30         testScenarioMainscenarioAlternativesfailed();
31
32     }
33
34     public void cleanupScenarioMainscenario() {
35         // TODO: insert your code here
36     }
37
38    /*
39     * Test for Main flow
40     *
41     */
42     public void testScenarioMainscenarioMainflow() {
43         // TODO: insert your code here for preparing and finishing the flow test
44
45         // Go to Main flow
46         runStepMainflowStart();
47
48         runStepMainflowConfirmactors();
49
50         runStepMainflowConfirmelements();
51
52         runStepMainflowConfirmstart();
53
54         runStepMainflowConfirmends();
55
56         runStepMainflowConfirmalternatives();
57
58         runStepMainflowEnd();
59         // Use case ends
60     }
61
62    /*
63     * Test for No actors
64     *
65     */
66     public void testScenarioMainscenarioNoactors() {
67         // TODO: insert your code here for preparing and finishing the flow test
68
69         // Go to Main flow
70         runStepMainflowStart();
71
72         runStepMainflowConfirmactors();
73         // Go to No actors
74         runStepNoactorsNoactors();
75         // Return to Main flow
76         runStepMainflowConfirmelements();
```

Java test cases gerados (javaTestCases.xsl)

```
77
78          runStepMainflowConfirmstart();
79
80          runStepMainflowConfirmends();
81
82          runStepMainflowConfirmalternatives();
83
84          runStepMainflowEnd();
85          // Use case ends
86      }
87
88      /*
89       * Test for No elements
90       *
91       */
92      public void testScenarioMainscenarioNoelements() {
93          // TODO: insert your code here for preparing and finishing the flow test
94
95          // Go to Main flow
96          runStepMainflowStart();
97
98          runStepMainflowConfirmactors();
99
100         runStepMainflowConfirmelements();
101         // Go to No elements
102         runStepNoelementsNoelements();
103         // Return to Main flow
104         runStepMainflowConfirmstart();
105
106         runStepMainflowConfirmends();
107
108         runStepMainflowConfirmalternatives();
109
110         runStepMainflowEnd();
111         // Use case ends
112     }
113
114     /*
115      * Test for No start
116      *
117      */
118     public void testScenarioMainscenarioNostart() {
119         // TODO: insert your code here for preparing and finishing the flow test
120
121         // Go to Main flow
122         runStepMainflowStart();
123
124         runStepMainflowConfirmactors();
125
126         runStepMainflowConfirmelements();
127
128         runStepMainflowConfirmstart();
129         // Go to No start
130         runStepNostartNostart();
131         // Return to Main flow
132         runStepMainflowConfirmends();
133
134         runStepMainflowConfirmalternatives();
135
136         runStepMainflowEnd();
137         // Use case ends
138     }
139
140     /*
141      * Test for No ends
142      *
143      */
144     public void testScenarioMainscenarioNoends() {
145         // TODO: insert your code here for preparing and finishing the flow test
146
147         // Go to Main flow
148         runStepMainflowStart();
149
150         runStepMainflowConfirmactors();
151
152         runStepMainflowConfirmelements();
153
154         runStepMainflowConfirmstart();
155
156         runStepMainflowConfirmends();
157         // Go to No ends
158         runStepNoendsNoends();
```

```
159         // Return to Main flow
160         runStepMainflowConfirmalternatives();
161 
162         runStepMainflowEnd();
163         // Use case ends
164     }
165 
166    /*
167     * Test for Alternatives failed
168     *
169     */
170     public void testScenarioMainscenarioAlternativesfailed() {
171         // TODO: insert your code here for preparing and finishing the flow test
172 
173         // Go to Main flow
174         runStepMainflowStart();
175 
176         runStepMainflowConfirmactors();
177 
178         runStepMainflowConfirmelements();
179 
180         runStepMainflowConfirmstart();
181 
182         runStepMainflowConfirmends();
183 
184         runStepMainflowConfirmalternatives();
185         // Go to Alternatives failed
186         runStepAlternativesfailedIncompletealternatives();
187         // Return to Main flow
188         runStepMainflowEnd();
189         // Use case ends
190     }
191 
192     /*
193      * Steps for use case Compile use case
194      */
195 
196     public void runStepMainflowStart() {
197         //TODO: insert your code here for implementing this step
198     }
199 
200     public void runStepMainflowConfirmactors() {
201         //TODO: insert your code here for implementing this step
202     }
203 
204     public void runStepMainflowConfirmelements() {
205         //TODO: insert your code here for implementing this step
206     }
207 
208     public void runStepMainflowConfirmstart() {
209         //TODO: insert your code here for implementing this step
210     }
211 
212     public void runStepMainflowConfirmends() {
213         //TODO: insert your code here for implementing this step
214     }
215 
216     public void runStepMainflowConfirmalternatives() {
217         //TODO: insert your code here for implementing this step
218     }
219 
220     public void runStepMainflowEnd() {
221         //TODO: insert your code here for implementing this step
222     }
223 
224     public void runStepNoactorsNoactors() {
225         //TODO: insert your code here for implementing this step
226     }
227 
228     public void runStepNoelementsNoelements() {
229         //TODO: insert your code here for implementing this step
230     }
231 
232     public void runStepNostartNostart() {
233         //TODO: insert your code here for implementing this step
234     }
235 
236     public void runStepNoendsNoends() {
237         //TODO: insert your code here for implementing this step
238     }
239 
240     public void runStepAlternativesfailedIncompletealternatives() {
```

```
241         //TODO: insert your code here for implementing this step
242     }
243 
244 }
245 
246 // Generated by UCDesigner 1.0
```

Use case: UC011 – View Project Tree

```
1  
2  /*
3   * Class TestCaseViewprojecttree
4   * Generated for testing the use case View project tree
5   */
6  public class TestCaseViewprojecttree extends TestCase {
7  
8  
9      /*
10      * Test for Scenario Main scenario
11      */
12     public void setupScenarioMainscenario() {
13         // TODO: insert your code here for preparing the scenario test
14     }
15 
16     public void testScenarioMainscenario() {
17         // TODO: insert your code here for starting and finishing the scenario test
18         // Testing all flows for the scenario
19 
20         testScenarioMainscenarioMainflow();
21 
22     }
23 
24     public void cleanupScenarioMainscenario() {
25         // TODO: insert your code here
26     }
27 
28    /*
29     * Test for Main flow
30     *
31     */
32     public void testScenarioMainscenarioMainflow() {
33         // TODO: insert your code here for preparing and finishing the flow test
34 
35         // Go to Main flow
36         runStepMainflowStart();
37 
38         runStepMainflowFetchallelements();
39 
40         runStepMainflowCreatefixedpackages();
41 
42         runStepMainflowShowanalysispackages();
43 
44         runStepMainflowShowActors();
45 
46         runStepMainflowShowusecases();
47 
48         runStepMainflowShowclassesandimplementationpackages();
49 
50         runStepMainflowEnd();
51         // Use case ends
52     }
53 
54     /*
55      * Steps for use case View project tree
56      */
57 
58     public void runStepMainflowStart() {
59         //TODO: insert your code here for implementing this step
60     }
61 
62     public void runStepMainflowFetchallelements() {
63         //TODO: insert your code here for implementing this step
64     }
65 
66     public void runStepMainflowCreatefixedpackages() {
67         //TODO: insert your code here for implementing this step
68     }
```

Java test cases gerados (javaTestCases.xsl)

```
69
70      public void runStepMainflowShowanalysispackages() {
71          //TODO: insert your code here for implementing this step
72      }
73
74      public void runStepMainflowShowActors() {
75          //TODO: insert your code here for implementing this step
76      }
77
78      public void runStepMainflowShowusecases() {
79          //TODO: insert your code here for implementing this step
80      }
81
82      public void runStepMainflowShowclassesandimplementationpackages() {
83          //TODO: insert your code here for implementing this step
84      }
85
86      public void runStepMainflowEnd() {
87          //TODO: insert your code here for implementing this step
88      }
89
90  }
91
92  // Generated by UCDesigner 1.0
```

Use case: UC012 – Open Use Case

```
1
2   /*
3    * Class TestCaseOpenusecase
4    * Generated for testing the use case Open use case
5    */
6   public class TestCaseOpenusecase extends TestCase {
7
8
9       /*
10       * Test for Scenario Main scenario
11       */
12      public void setupScenarioMainscenario() {
13          // TODO: insert your code here for preparing the scenario test
14      }
15
16      public void testScenarioMainscenario() {
17          // TODO: insert your code here for starting and finishing the scenario test
18          // Testing all flows for the scenario
19
20          testScenarioMainscenarioMainflow();
21
22          testScenarioMainscenarioNodescriptionavailable();
23
24      }
25
26      public void cleanupScenarioMainscenario() {
27          // TODO: insert your code here
28      }
29
30     /*
31      * Test for Main flow
32      *
33      */
34      public void testScenarioMainscenarioMainflow() {
35          // TODO: insert your code here for preparing and finishing the flow test
36
37          // Go to Main flow
38          runStepMainflowStart();
39
40          runStepMainflowOpenusecase();
41
42          runStepMainflowViewdescription();
43
44          runStepMainflowEnd();
45          // Use case ends
46      }
47
48     /*
49      * Test for No description available
50      *
51      */
52      public void testScenarioMainscenarioNodescriptionavailable() {
```

Java test cases gerados (javaTestCases.xsl)

```
53          // TODO: insert your code here for preparing and finishing the flow test
54
55          // Go to Main flow
56          runStepMainflowStart();
57
58          runStepMainflowOpenusecase();
59          // Go to No description available
60          runStepNodescriptionavailableOpenusecasedialog();
61          // Return to Main flow
62          runStepMainflowEnd();
63          // Use case ends
64      }
65
66      /*
67       * Steps for use case Open use case
68       */
69
70      public void runStepMainflowStart() {
71          //TODO: insert your code here for implementing this step
72      }
73
74      public void runStepMainflowOpenusecase() {
75          //TODO: insert your code here for implementing this step
76      }
77
78      public void runStepMainflowViewdescription() {
79          //TODO: insert your code here for implementing this step
80      }
81
82      public void runStepMainflowEnd() {
83          //TODO: insert your code here for implementing this step
84      }
85
86      public void runStepNodescriptionavailableOpenusecasedialog() {
87          //TODO: insert your code here for implementing this step
88      }
89
90  }
91
92  // Generated by UCDesigner 1.0
```

Use case: UC014 – Display Compile Message

```
 1
 2  /*
 3   * Class TestCaseDisplaycompilemessages
 4   * Generated for testing the use case Display compile messages
 5   */
 6  public class TestCaseDisplaycompilemessages extends TestCase {
 7
 8
 9      /*
10       * Test for Scenario Main scenario
11       */
12      public void setupScenarioMainscenario() {
13          // TODO: insert your code here for preparing the scenario test
14      }
15
16      public void testScenarioMainscenario() {
17          // TODO: insert your code here for starting and finishing the scenario test
18          // Testing all flows for the scenario
19
20          testScenarioMainscenarioMainflow();
21
22          testScenarioMainscenarioNomessages();
23
24      }
25
26      public void cleanupScenarioMainscenario() {
27          // TODO: insert your code here
28      }
29
30      /*
31       * Test for Main flow
32       *
33       */
34      public void testScenarioMainscenarioMainflow() {
```

Java test cases gerados (javaTestCases.xsl)

```
35          // TODO: insert your code here for preparing and finishing the flow test
36
37          // Go to Main flow
38          runStepMainflowStart();
39
40          runStepMainflowFecthallmessages();
41
42          runStepMainflowDisplaymessages();
43
44          runStepMainflowShowmessages();
45          // Use case ends
46      }
47
48     /*
49      * Test for No messages
50      *
51      */
52      public void testScenarioMainscenarioNomessages() {
53          // TODO: insert your code here for preparing and finishing the flow test
54
55          // Go to Main flow
56          runStepMainflowStart();
57
58          runStepMainflowFecthallmessages();
59          // Go to No messages
60          runStepNomessagesNomessagesfound();
61
62          runStepNomessagesDon"tshowthesection();
63
64          runStepNomessagesHidesection();
65          // Use case ends
66      }
67
68      /*
69       * Steps for use case Display compile messages
70       */
71
72      public void runStepMainflowStart() {
73          //TODO: insert your code here for implementing this step
74      }
75
76      public void runStepMainflowFecthallmessages() {
77          //TODO: insert your code here for implementing this step
78      }
79
80      public void runStepMainflowDisplaymessages() {
81          //TODO: insert your code here for implementing this step
82      }
83
84      public void runStepMainflowShowmessages() {
85          //TODO: insert your code here for implementing this step
86      }
87
88      public void runStepNomessagesNomessagesfound() {
89          //TODO: insert your code here for implementing this step
90      }
91
92      public void runStepNomessagesDon"tshowthesection() {
93          //TODO: insert your code here for implementing this step
94      }
95
96      public void runStepNomessagesHidesection() {
97          //TODO: insert your code here for implementing this step
98      }
99
100 }
101
102 // Generated by UCDesigner 1.0
```

Anexo 3 – Artigo apresentado no CLEI 2004

Evento:	XXX Conferencia Latino-Americana de Estudios de Informática
Local:	Arequipa, Peru
Período:	27/09/2004 a 02/10/2004
URL:	http://www.spc.org.pe/clei2004
Artigo:	Estruturação de Descrições de Casos de Uso através de Mecanismos de Extensibilidade da UML
Data:	27/09/2004
URL:	http://clei2004.spc.org.pe/es/html/pdfs/151.pdf

Artigo apresentado na XXX Conferencia Latinoamericana de Estudios de Informática - CLEI 2004
27/09/2004 em Arequipa - Peru

Estruturação de Descrições de Casos de Uso através de Mecanismos de Extensibilidade da UML

Gabriel Silva Bornia, BSc.
Roberto Tom Price, Eng., MSc., D. Phil.

Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática
Av. Bento Gonçalves, 9500
Porto Alegre – RS – Brasil
e-mail: {bornia, tomprice}@inf.ufrgs.br

Abstract

This paper presents a structured representation of use case descriptions using stereotyped activity diagrams. The use of the extensibility mechanism of UML is used to configure language elements to be used for the description of system behavior. A way of representing use case descriptions in different levels of abstraction is shown, and ways of associating between descriptive elements and the static model of the system. A CASE tool is presented to demostrate the proposed use case description method.

Key-words: Use case, use case description, collaboration case, UML, activity diagram, extensibility mechanism, CASE tool.

Resumo

Este trabalho apresenta uma forma de representação estruturada de descrições de casos de uso através do uso de diagramas de atividade estereotipados. O uso da extensibilidade da UML permite configurar elementos da linguagem de tal forma que esta possa também ser utilizada para a descrição do comportamento do sistema. É apresentada uma forma de representação de descrições de casos de uso em vários níveis de abstração, bem como a associação entre elementos da descrição e o modelo estático do sistema. Uma ferramenta CASE é apresentada como prova de conceito para o método de descrição proposto.

Palavras-chave: Caso de uso, descrição de casos de uso, casos de colaboração, UML, diagramas de atividade, mecanismos de extensibilidade, ferramenta CASE.

1 Introdução

Na análise de sistemas orientada a objetos a modelagem de casos de uso possui um papel fundamental. Os casos de uso são o principal mecanismo de levantamento de requisitos [24], caracterizando-se por uma representação narrativa do comportamento do sistema [14][15].

As descrições de casos de uso caracterizam-se por serem descrições do funcionamento do sistema sem, no entanto, entrar em detalhes específicos de como o mesmo será implementado. Diversos autores [6][8][10] defendem que os casos de uso devem ser o mais simples possível, uma vez que devem ser entendidos pelos usuários do sistema – que os utilizam como forma de validação dos requisitos levantados pelo analista – e servir como base aos demais dos envolvidos na construção do sistema.

A necessidade de se criar descrições simples que possam ser entendidas por usuários comuns representa um obstáculo ao uso de mecanismo mais formais de descrição [9] que possam ser utilizados de forma estruturada por projetistas, desenvolvedores e gerentes de projeto.

Existem diversas formas de se representar descrições de casos de uso. A mais comum delas é a forma textual, que embora possua diversos estilos propostos [1][3][17] continua praticamente livre de uma estrutura que permita a modelagem da descrição. Existem diversas propostas de formalizações de casos de uso [12][13] de difícil disseminação no mercado. UML é uma linguagem de modelagem [4] amplamente difundida. Geralmente, os casos de uso são modelados com esta linguagem, mas a interação entre o sistema e o ator (descrição do caso de uso) é realizada de forma narrativa [21]. As descrições dessas interações podem ser realizadas através de diversos elementos da linguagem UML como, por exemplo, diagramas de atividade [3], diagramas de seqüência, diagramas de estado, entre outros. UML possui a facilidade de estender os seus elementos através do conceito de estereótipos, o que poderia ser utilizado para ajudar a enriquecer uma descrição feita com os elementos desta linguagem.

O objetivo deste trabalho é a representação de descrições de casos de uso através de mecanismos de extensibilidade da UML, mais especificamente permitindo a modelagem das atuais representações narrativas por meio de diagramas de atividade estereotipados, garantindo uma representação mais estruturada que possa ser utilizada em outros passos do processo de desenvolvimento de sistemas sem, no entanto, perder a capacidade de se comunicar com o usuário através de mecanismos que permitam gerar uma linguagem que ele compreenda.

2 Descrição de Casos de Uso

Usualmente a descrição dos casos de uso é realizada de forma textual sem nenhum compromisso com uma estrutura de representação mais formal ou estruturada [25]. Este tipo de descrição não pode ser utilizado de forma adequada por mecanismos automatizados que auxiliem o desenvolvimento de sistemas em outras etapas do processo de construção, por não possuir uma estrutura ou modelo de representação adequado.

O formato de uma descrição de casos de uso é dividido em diversas seções, que variam para cada organização, mas as seções mais comuns de serem encontrados em templates de descrições de casos de uso são [5]:

1. Atores: seção com a descrição do atores envolvidos no caso de uso;
2. Pré-condições: regras de estado que definem como o sistema deve se encontrar antes de ocorrer o caso de uso;
3. Fluxo de eventos: interações que ocorrem entre os atores e o sistema à medida que interagem para atingir um determinado objetivo;
4. Pós-condições: regras de estado que definem como o sistema deve se encontrar depois de executado o caso de uso.

A seção mais importante de uma descrição de caso de uso é o fluxo de eventos entre o ator e o sistema. O fluxo de eventos é um conjunto seqüência de eventos do tipo ação-reação que descrevem os caminhos percorridos pelo ator do caso de uso para atingir o seu objetivo.

Um exemplo simples de descrição é mostrado na Tabela 1, na qual o caso de uso "User Login" é descrito através de uma tabela que separa o fluxo entre ator e sistema [23].

Ator	**Sistema**
1. Preenche o usuário e a senha.	
	2. Valida o usuário e a senha.
	3. Realiza o login do usuário.
Alternativa 1: O usuário e senha estão inválidos. Informa o usuário sobre o erro e requisita novamente as informações de login.	

Tabela 1 – Descrição do caso de uso "User Login"

As descrições de casos de uso podem possuir uma visão externa ou interna do sistema. Em uma visão externa (ou caixa-preta) somente as atividades que são visíveis aos atores externos são descritas sem indicação de como o sistema faz para obter o resultado de uma ação. Na visão interna (ou caixa-branca) o comportamento interno do sistema é descrito. A visão externa permite a validação dos requisitos com o

usuário mas pode acabar perdendo requisitos importantes que poderiam ser obtidos através de uma descrição mais detalhada. A visão interna é de grande utilidade nas próximas fases do desenvolvimento do sistema, mas difícil de ser validada com o usuário [3] devido ao detalhamento utilizado nesta visão.

Le Roy Mattingly e Harsha Rao [18] propuseram os casos de colaboração, que são casos de uso estereotipados para a descrição interna do sistema. A vantagem é a manutenção e refinamento interno dos casos de uso (casos de colaboração/visão interna) uma vez que a interface com o usuário esteja bem definida (casos de uso/visão externa).

3 Descrição através de Diagramas de Atividade

Diversas notações podem ser utilizadas para a representação de casos de uso como, por exemplo, redes de Petri, linguagens formais, pseudo-código, entre outros [25]. Em UML, diagramas de atividades, diagramas de seqüência e diagramas de estado podem ser utilizados para descrever casos de uso.

Os diagramas de atividades se apresentam como uma notação mais clara para a representação de paralelismos, elementos de lógica condicional e sincronização do que os diagramas de seqüência e diagramas de estado [3]. A descrição através de diagramas de atividade torna a descrição um artefato mais formal e estruturado, porém pode ser de difícil entendimento para os usuários e sua validação junto aos mesmos pode ficar prejudicada.

Entretanto, uma ferramenta CASE a partir de uma representação estruturada como essa poderia extrair uma descrição textual fácil de ser compreendia por usuários. A mesma ferramenta pode utilizar essa descrição estruturada para extrair outros artefatos úteis durante o processo de desenvolvimento, como por exemplo, casos de teste, casos de colaboração, métricas de complexidade, diagramas de navegação, entre outros.

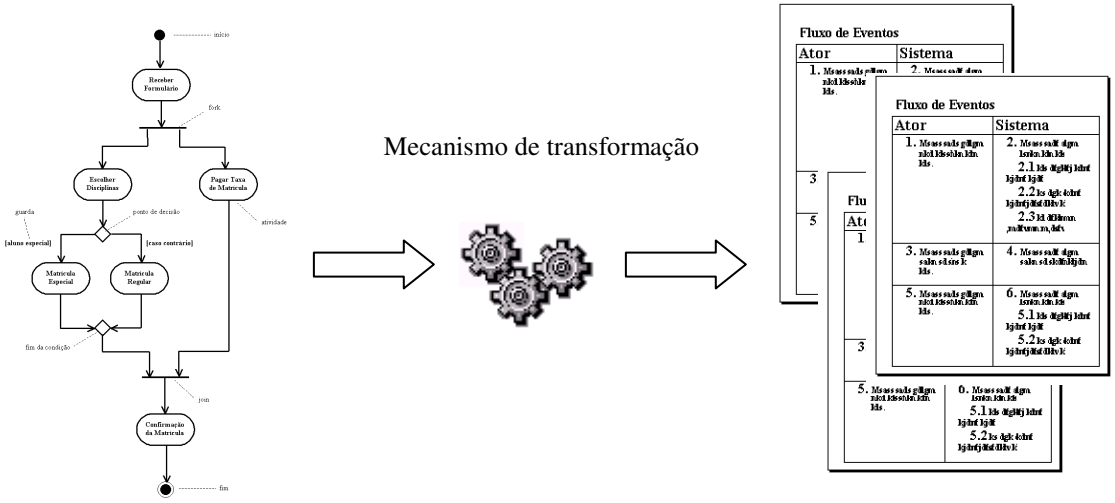


Figura 58 - Mecanismo de transformação

A utilização de diagramas de atividade para a representação de descrições de casos de uso precisa ser adaptada para suportar algumas características próprias das descrições dos casos de uso. As adaptações necessárias – como, por exemplo, a caracterização de atividades que representam passos realizados pelos atores ou pelo sistema dentro de um fluxo de eventos – podem ser realizadas através do mecanismo de extensibilidade da UML [20].

3.1 Representação do fluxo de eventos

O fluxo de eventos de uma descrição de caso de uso representa a interação entro o ator e o sistema. Essa interação se dá através de eventos numerados representados textualmente através de um fluxo contínuo ou de um fluxo separado por colunas [3][23].

Cada evento é realizado pelo ator ou pelo sistema e representa uma interação do tipo ação-reação [20]. A representação através de atividades dentro de um diagrama de atividades pode ser realizada através da utilização dos estereótipos <<ator>> e <<sistema>> para identificar o responsável pela atividade. A descrição dos casos de uso permite a introdução de seqüências alternativas, que representam desvios no fluxo principal. Essas seqüências alternativas podem ser utilizadas para realizar o tratamento de exceções ou indicar um outro caminho possível. A Figura 2 mostra como o exemplo mostrado na Tabela 1 pode ser representado através de um diagrama de atividade que utilize estereótipos.

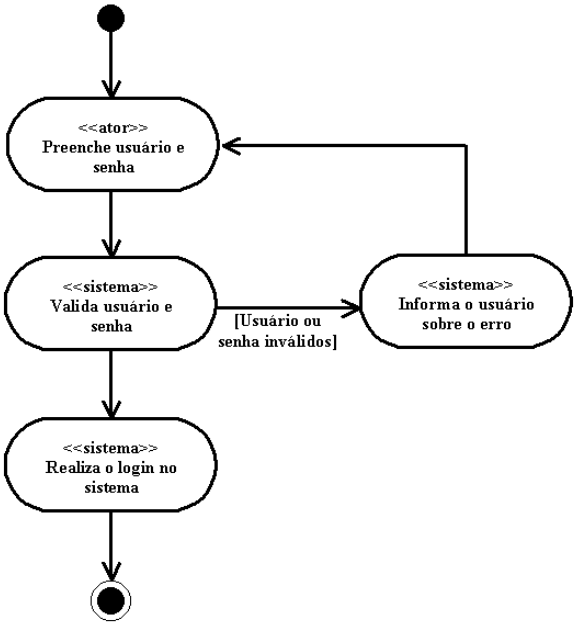


Figura 2 – Descrição através de diagramas de atividades do caso de uso "Login User"

A representação de seqüências alternativas ou fluxos condicionais é representada através de transições com guardas [4].

Os diagramas de atividades possuem apenas um estado inicial e podem possuir um ou mais estados finais. Os estados finais são utilizados para indicar os possíveis términos a que a execução do caso de uso pode levar. As pré-condições e pós-condições do sistema são representadas como atributos dos elementos que representam respectivamente os estados iniciais e finais do diagrama de atividade.

Os diagramas de atividade suportam outros elementos como barra de paralelismo e de sincronização, utilizadas para representar fluxos que podem ocorrer em paralelo. As descrições textuais não possuem uma forma simples de representar comportamentos paralelos.

3.2 Representação de casos de colaboração

Os casos de colaboração são casos de uso estereotipados que descrevem o funcionamento interno do sistema [18]. A especificação de UML 1.5 possui o conceito de sub-diagramas de atividade. Essa característica permite que uma atividade possua um outro diagrama de atividade associado dentro dela. Utilizando esse conceito é possível representar casos de colaboração como sub-diagramas das atividades estereotipadas como <<sistema>>.

O caso de uso mais abstrato ou de maior nível pode definir apenas o comportamento geral do caso de uso. Durante o processo de desenvolvimento do sistema, durante a especificação mais especifica dos requisitos, o caso de uso pode ir sendo refinado através de um maior detalhamento das atividades do sistema com o uso de sub-diagramas que representem casos de colaboração.

A navegação de forma hierárquica permite que novos casos de colaboração sejam incorporados à descrição inicial do caso de uso. A validação dos requisitos com o usuário pode ser feita através da descrição mais alto-nível (primeiro diagrama na hierarquia), enquanto que os outros níveis serviriam de apoio às outras etapas envolvidas no ciclo de desenvolvimento do sistema.

O nível de detalhamento por ir aumentando à medida que novos níveis de descrição são incorporados. Um exemplo de como os casos de colaboração podem ser utilizados como diagramas de atividades estereotipados é mostrado na Figura 3.

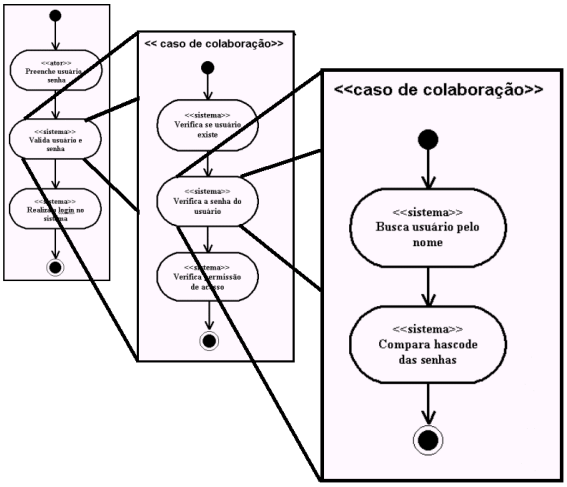


Figura 3 - Representação de casos de colaboração

Uma ferramenta CASE facilitaria o processo de descrição através de mecanismos de navegabilidade drill-down e drill-up. A descrição mais abstrata (alto-nível) poderia ser utilizada para a validação da interação do usuário com o sistema enquanto que a descrição mais específica poderia detalhar o funcionamento interno do projeto.

3.3 Representação de relacionamentos entre os casos de uso

Durante a análise são identificados os casos de uso que compõe o sistema e estes são organizados dentro de um modelo de casos de uso, onde se representam as relações dos atores com os casos de uso, e as relações entre os casos de uso. Os relacionamentos que podem ocorrer entre os casos de uso são os de inclusão, extensão e generalização/especialização [10][11][20].

O relacionamento de inclusão representa uma relação na qual o fluxo de eventos do caso de uso incluído é executado durante o fluxo de eventos de um caso de uso base. Essa chamada deve ser indicada dentro da descrição [3]. Essa indicação é representada através de uma atividade estereotipada como <<ponto de inclusão>>, e existe uma referência entre ela e o caso de uso que está sendo incluído.

Já o relacionamento de extensão apresenta dentro do fluxo de eventos indicações dos lugares onde o comportamento pode ser estendido por outros casos de uso. Essas indicações são chamadas de pontos de extensão [3] e são representadas por atividades estereotipadas como <<ponto de extensão>>. Cada ponto de extensão pode ser referenciado externamente por outro caso de uso.

O relacionamento de generalização/especialização representa a herança de comportamento entre os casos de uso. Nesse caso o fluxo de eventos é herdado e o seu comportamento pode ser alterado ou incrementado [3][11]. Este tipo de relacionamento é pouco utilizado e seu controle é mais complexo, embora possa simplificar as descrições de casos de uso que possuam pequenas alterações de comportamento. Uma ferramenta CASE poderia auxiliar na manutenção de descrições para casos de uso especializados[23].

[23] Este trabalho não contempla a descrição de casos de uso especializados, pelo fato de seu uso ser pouco utilizado [3][6][25].

A descrição de forma estruturada permite validar o modelo de casos de uso garantindo a consistência entre o que é modelado (relacionamentos de inclusão e extensão entre os casos de uso) e as descrições dos mesmos, uma vez que esses relacionamentos são referenciados na descrição. A relação entre a descrição estruturada e o modelo de casos de uso pode ser vista na Figura 4.

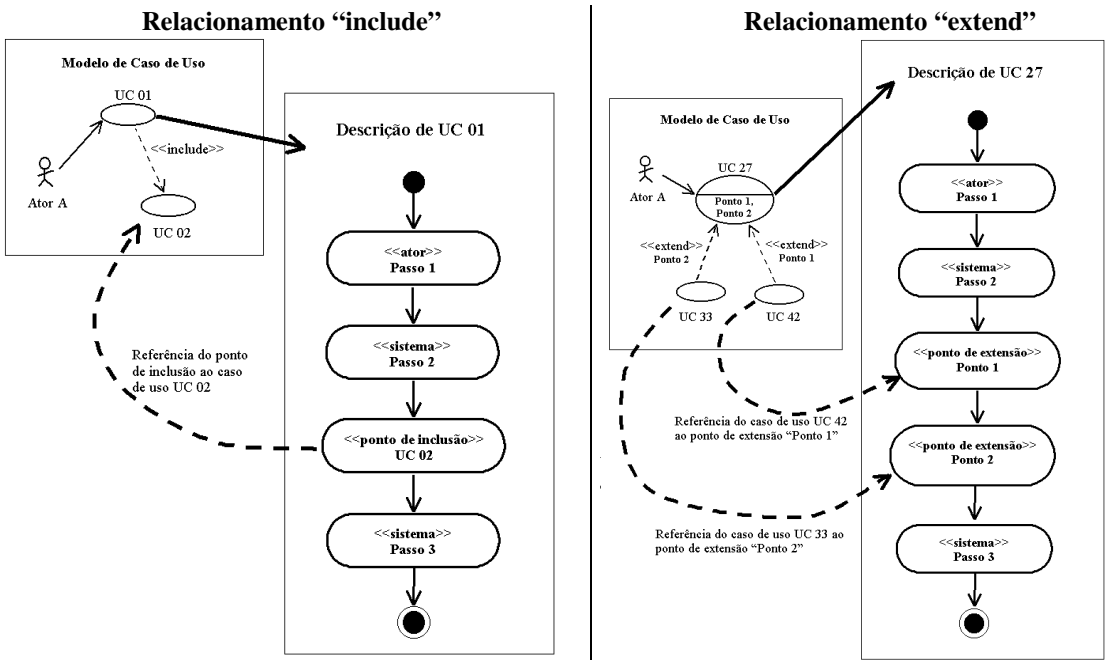


Figura 4 - Representação do relacionamento "include" e "extend"

4 Relação com a arquitetura e o modelo estático do sistema

A arquitetura do sistema pode ser definida como o conjunto de decisões significativas tomadas sobre a organização do sistema, a escolha dos elementos estruturais e interfaces que compõe o sistema [26]. O modelo de visão 4+1 proposto por Kruchten [16] identifica os casos de uso como a interligação entre as diversas visões concorrentes que compõe a arquitetura.

A definição da arquitetura ocorre já nas primeiras fases do ciclo de desenvolvimento. Os casos de uso iniciais ajudam a definir o que seria uma arquitetura adequada para o sistema, o suficiente para poder colocar um protótipo de arquitetura a suportar os casos de uso das primeiras iterações, dentro de um processo de desenvolvimento iterativo [15][19].

À medida que as descrições dos casos de uso e de colaboração vão se detalhando cada vez mais, as descrições entram no universo do projeto de sistemas e acabam se relacionando com a arquitetura definida para a construção do sistema.

A descrição de casos de uso através de diagramas de atividade pode utilizar o conceito de *swim lanes* [10] para identificar estilos arquiteturais (como camadas, sub-sistemas ou componentes do sistema) e identificar os eventos ou atividades associados a cada elemento da arquitetura.

Por exemplo, uma descrição de um caso de colaboração através dos diagramas de atividade poderia possuir as seguintes swim lanes: camada de apresentação, camada de negócio, camada de persistência e camada de banco de dados. Dentro de um mesmo diagrama é possível representar um fluxo de eventos e indicar em que sub-sistemas ou camadas cada evento é realizado.

Na Figura 5 pode ser visto um exemplo no qual o aprofundamento das descrições leva à necessidade de se identificar em que camada cada evento está sendo realizado. Esse tipo de associação pode ser realizado nas etapas de desenvolvimento, através da qual os próprios desenvolvedores ou projetistas podem dar continuidade à descrição criada na análise, refinando-a cada vez mais.

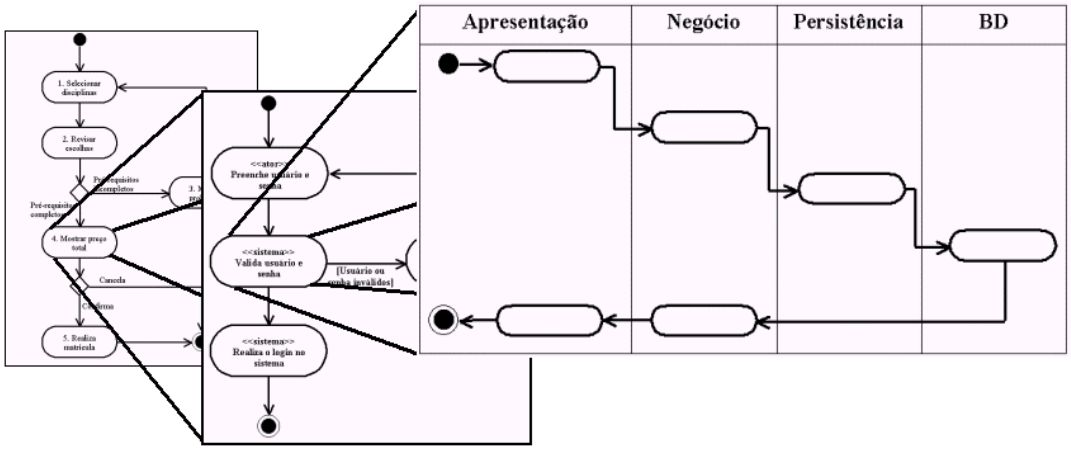


Figura 5 - Exemplo de detalhamento de uma descrição até a arquitetura

Durante o processo de análise de sistemas o modelo de classes vai sendo construído, inicialmente com classes que representam os conceitos de negócio, e posteriormente nas etapas de projeto e desenvolvimento o modelo é enriquecido com classes de projeto mais voltadas à forma como o sistema será implementado, e fortemente ligado à arquitetura definida.

Uma ferramenta CASE pode permitir a associação de elementos da descrição (eventos do fluxo de eventos do caso de uso) a classes do modelo estático ou a métodos de classes. Essa associação permite o rastreamento de conceitos afetados por alterações de requisitos. Se, por exemplo, o usuário quiser alterar alguma funcionalidade é possível identificar os pontos afetados por tal alteração de forma automática.

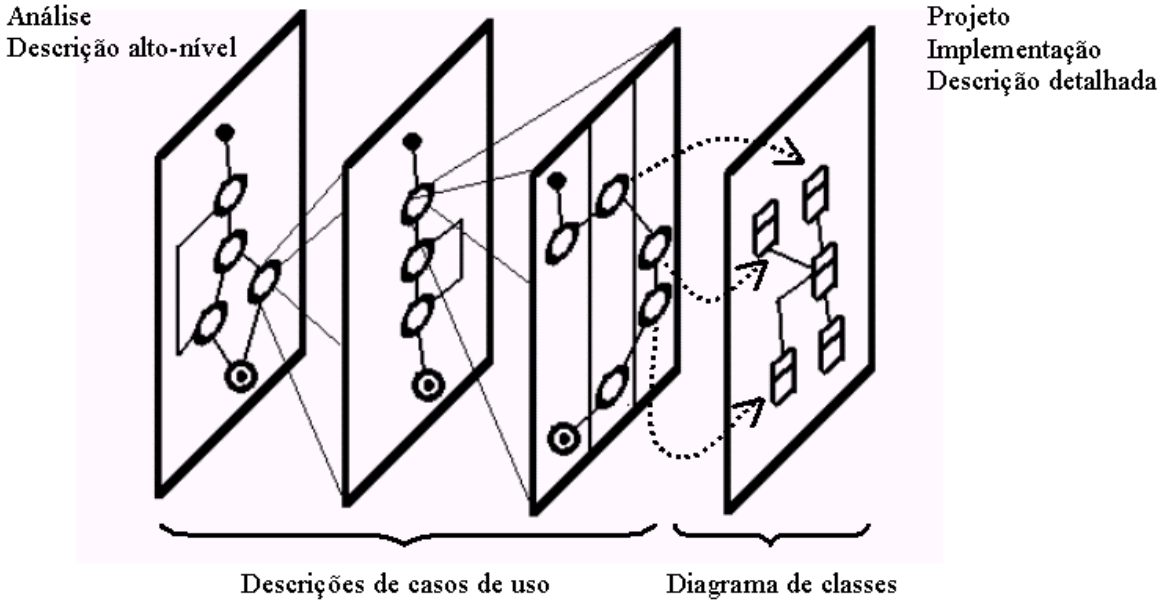


Figura 6 – Refinamento das descrições, relação com a arquitetura e modelo de classes

A Figura 6 mostra como o refinamento das descrições pode evoluir até o ponto de se ter elementos de descrição associados a classes ou métodos.

5 Modelo para a descrição dos casos de uso

A solução proposta de descrição dos casos de uso através de diagramas de atividade estereotipados é mapeada para as classes a seguir. A intenção não é a construção de um modelo que represente as estruturas definidas pela UML. A OMG já apresenta um modelo de mapeamento muito mais genérico para a sintaxe da UML do que o apresentado neste trabalho. O modelo aqui proposto é mostrado na Figura 7, definido para provar o conceito de descrição de casos de uso através de uma estrutura mais formal, baseada em diagramas de atividade. Outra modelagem seria possível, inclusive uma que reaproveitasse a modelagem definida pela OMG.

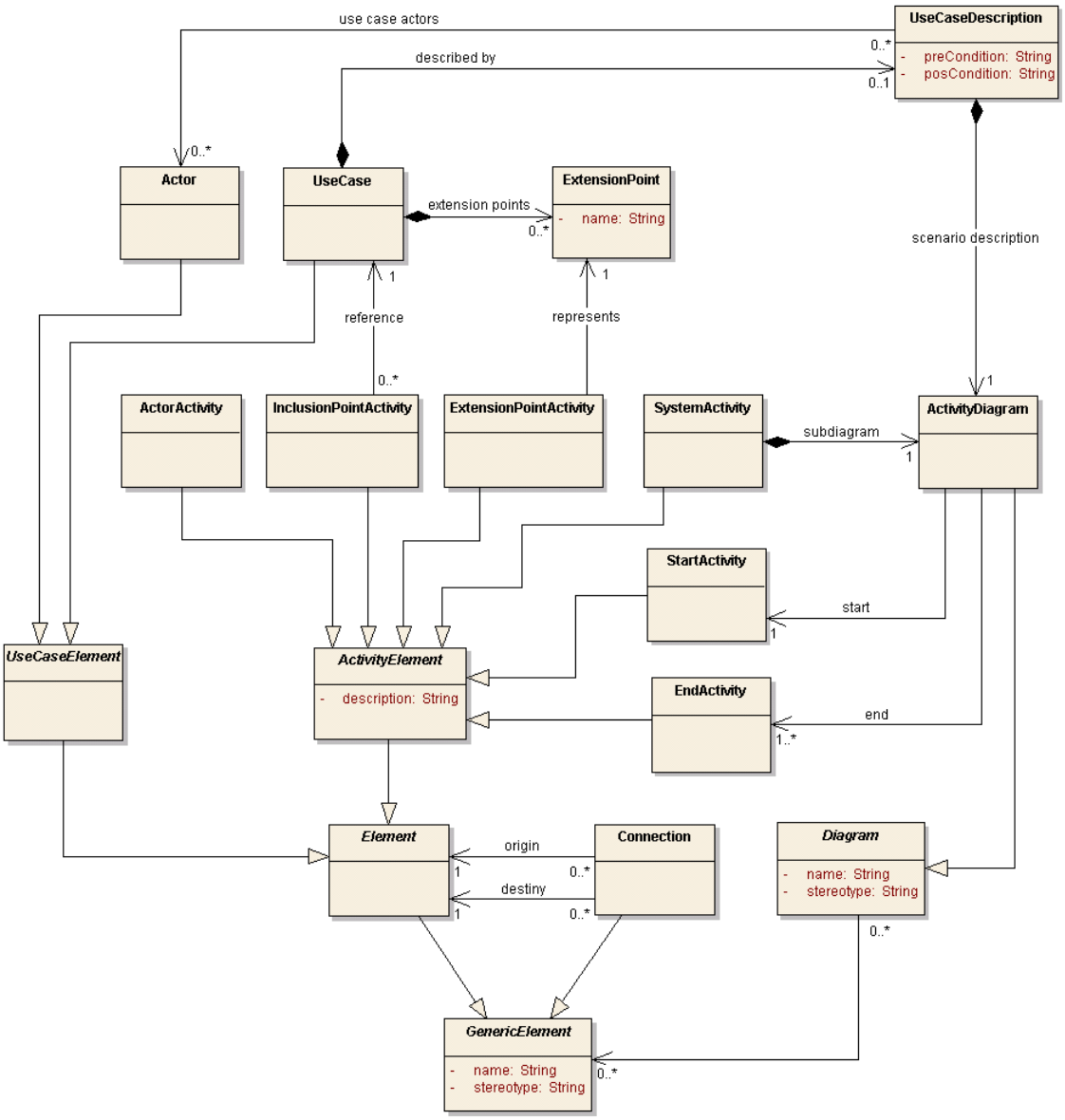


Figura 7 – Modelo geral para representação de descrições de casos de uso através de diagramas de atividade

O relacionamento da descrição com a arquitetura ocorre quando o diagrama de atividade que descreve o caso de uso possui swim lanes associadas às partições ou camadas de uma descrição de arquitetura definida. São identificadas as camadas onde cada atividade da descrição está localizada bem como as classes ou métodos aos quais a atividade está associada. A associação entre a atividade da descrição e classes ou métodos possui uma indicação sobre o tipo de referência realizado (o tipo de referência pode ser "utiliza", "executa", entre outros). A Figura 8 mostra o modelo para essas características.

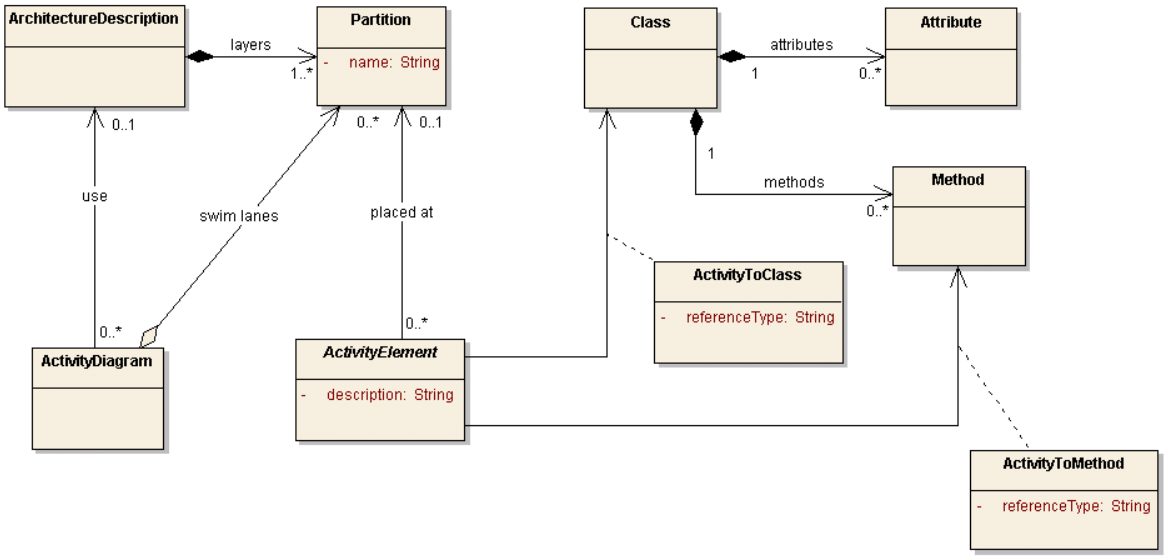


Figura 8 – Relação com a arquitetura e o modelo de classes

A Figura 9 mostra um exemplo de como uma atividade pode estar associada a uma classe e a métodos de classes. A relação com o modelo estático do sistema ocorre nos níveis mais detalhados das descrições de casos de uso.

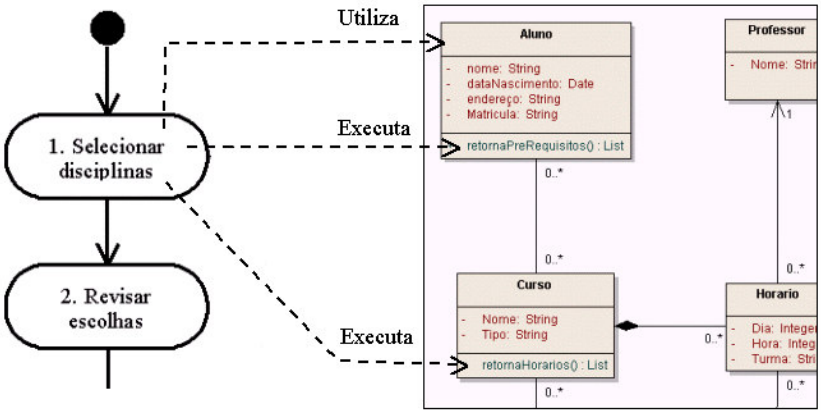


Figura 9 – Associação entre os elementos de descrição e o modelo estático

No exemplo, as referências entre as atividades e o modelo de classes do sistema (classes e métodos) possuem uma classificação cuja semântica poderia ser definida livremente pelos usuários de uma ferramenta CASE com suporte a esse tipo de descrição.

A maioria das ferramentas CASE de modelagem UML encontradas no mercado possuem funcionalidades que permitem associações simples entre casos de uso e artefatos de documentação externos (arquivos), geralmente utilizados para descrever os primeiros de forma narrativa sem nenhum

comprometimento com uma estrutura de representação. Algumas ferramentas como, por exemplo, Visual Paradigm [27], possuem meios um pouco mais elaborados que permitem a descrição dos casos de uso dentro da própria ferramenta. Outros projetos como, por exemplo, o desenvolvido pela PUC-Rio chamado C&L (Cenário e Léxico) [28], auxiliam a descrição colaborativa de cenários introduzindo uma organização da informação de forma estruturada. Entretanto, essas ferramentas não permitem a associação de elementos internos das descrições dos casos de uso (passos / atividades) com outros elementos que compõe o modelo do sistema, nem possuem mecanismos para a geração de outros artefatos.

Neste trabalho é apresentada uma ferramenta CASE desenvolvida com o intuito de oferecer um mecanismo de descrição baseado no método apresentado, utilizando diagramas de atividade estereotipados para estruturar as descrições de casos de uso, permitindo o processamento para a geração de artefatos e a associação entre outros elementos do modelo do sistema.

6 Ferramenta CASE de descrição

Para a prova de conceito do método de descrição de casos de uso através de diagramas de atividade com o mecanismo de extensibilidade da UML, foi desenvolvida uma ferramenta para a construção de descrições desse tipo baseado no modelo proposto. A ferramenta se chama UC Papyrus, tem valor apenas acadêmico e possui licença GPL (GNU Public License). A ferramenta e os códigos fontes estão disponíveis em http://sourceforge.net/projects/ucpapyrus.

A ferramenta é um editor diagramático de descrições de caso de uso baseadas em diagramas de atividade. As atividades estereotipadas foram transformadas em ícones para facilitar a leitura do diagrama. A Figura 10 mostra a aparência da ferramenta.

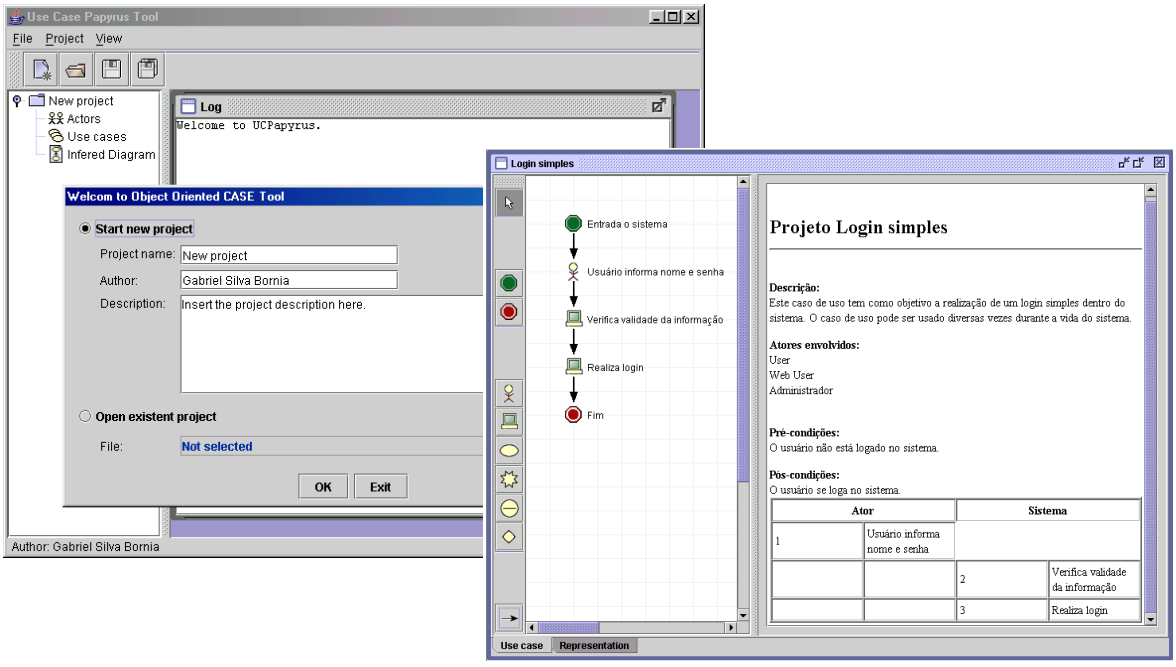


Figura 10 – Ferramenta CASE implementada

Todas as descrições de casos de uso são internamente representadas através de documentos semi-estruturados (XML). Através de regras de transformação XSL é possível processar o documento de representação e gerar diversos artefatos como, por exemplo, uma descrição amigável ao usuário semelhante às descrições textuais normalmente utilizadas no processo de análise. A ferramenta permite a adição de arquivos das regras XSL para processar as descrições.

7 Considerações finais e trabalhos futuros

A importância dos casos de uso vai muito além do escopo da representação de requisitos. Os casos de uso guiam todo o processo de desenvolvimento, a concepção da arquitetura, o projeto do sistema, a validação dos componentes implementados, a documentação do sistema, entre outras tarefas que compõe o ciclo de vida do sistema.

O fato de se utilizar uma linguagem natural para a descrição interna dos casos de uso pode ser importante para a validação com os usuários, mas é uma informação que pouco pode ser sistematizada em processos automatizados que possam reaproveitar essas informações em outras etapas do desenvolvimento de software.

Este trabalho propõe um mecanismo que estruture as descrições dos casos de uso, de forma que a própria representação dos requisitos sirva como mecanismo de validação junto aos usuários e ao mesmo tempo sirva de entrada para mecanismos computacionais capazes de utilizar essas informações para a geração de outros artefatos úteis nas demais etapas no desenvolvimento do sistema.

O uso da extensibilidade da UML é indicado como meio de utilizar a própria linguagem UML como forma de representação estruturada para descrições de casos de uso. O objetivo é mostrar que as descrições de casos de uso podem ser modeladas através da própria UML e – se devidamente assistidas por uma ferramenta CASE – servir de base para a geração de outros artefatos que auxiliem o processo de desenvolvimento de software.

Atualmente a ferramenta CASE está sendo aprimorada para suportar a geração de informações que possam auxiliar o gerenciamento de projetos. Essas informações são métricas extraídas das próprias descrições como, por exemplo, número de transações, número de pontos de inclusão, número de pontos de extensão. Esses dados serão utilizados como entrada para cálculos de estimativa de esforço para estimar tempos de implementação [2].

A estruturação de casos de uso pode permitir a realização de trabalhos como, por exemplo:

- Geração de scripts em LOTOS para realizar a simulação de passos que possam ser executados dentro de um caso de uso, de forma que esse script possa servir como base para possíveis estudos para testes automatizados;
- A existência de uma descrição diagramática para os casos de uso pode permitir a identificação de padrões de descrições de casos de uso, que eventualmente podem vir a ser catalogados;
- As descrições podem ser enriquecidas com informações que permitam auxiliar a construção de interfaces homem-máquina;
- Implementação da rastreabilidade de alterações de requisitos, indicando os artefatos e classes que devem ser re-visitados [7].

Referências

[1] Ambler, Scott W. *Documenting a use case: what to include, and why.* Ronin International. EUA, 2000.

[2] Anda, Bente; Dreiem, Hege; Sjøberg, Drag I.K.; Jørgensen, Magne. *Estimating software development effort based on use cases: experiences from industry.* 4th International Conference on the Unified Modeling Language: UML 2001. Canada, 2001.

[3] Armour, Frank; Miller, Granville. *Advanced use case modeling: software systems.* Addison-Wesley. EUA, 2000.

[4] Booch, Grady; Rumbaugh, James; Jacobson, Ivar. *The Unified modeling language user guide.* Addison-Wesley. EUA, 1999.

[5] Cockburn, Alistair. *Structuring use cases with goal.* Humans and Technology. EUA, 2000.

[6] Cockburn, Alistair. *Use cases, ten years later – from their evolution, learn what to expect and how to better work with them.* STQE Magazine. EUA, 2002.

[7] Ecklund, Earl F.; Delcambre, Lois; Freiling, Michael. *Change cases: use cases that identify future requirements.* 11th ACM Conference on Object-Oriented Programming Systems, Languages and Applications: OOPSLA'96. EUA, 1996.

[8] Firesmith, Donald G. *Use cases: the pros and cons. Report on Object Analysis and Design 2.* EUA, 1995.

[9] Fowler, Martin; Cockburn, Alistair; Jacobson, Ivar; Anderson, Bruce; Graham, Anderson. *"Question time! About use cases"*. 13th ACM Conference on Object-Oriented Programming Systems, Languages and Applications: OOPSLA'98. Canada, 1998.

[10] Fowler, Martin; Scott, Kendall. *UML distilled: a brief guide to the standard object modeling language.* Addison-Wesley. EUA, 1999.

[11] Génova, Gonzalo; Llorens, Juan; Quintana, Victor. *Digging into use case relationships.* 5th International Conference on the Unified Modeling Language: UML 2002. Alemanha, 2002.

[12] Hurlbut, Russel R. *The Three R's of Use Case Formalisms: Realization, Refinement, and Reification.* Expertech, Ltda. EUA, 1997.

[13] Hurlbut, Russel R. *A Survey of Approaches for Describing and Formalizing Use Cases.* Expertech, Ltda. EUA, 1997.

[14] Jacobson, Ivar; Christerson, Magnus; Patrik Jonsson; Övergaard, Gunnar. *Object-oriented software engineering: a use case driven approach.* Addison-Wesley. EUA, 1992.

[15] Jacobson, Ivar; Booch, Grady; Rumbaugh, James. *The unified software development process.* Addison-Wesley. EUA, 1998.

[16] Kruchten, Philippe. *Architectural blueprints – the "4+1" view model of software architecture.* IEEE Software. EUA, 1995.

[17] Larman, Craig. *Applying UML and patterns: an introduction to object-oriented analysis and design.* Prentice Hall. EUA, 1998.

[18] Mattingly, Le Roy; Rao, Harsha. *Writing effective use cases and introducing collaboration cases. Journal of object oriented programming 11.* EUA, 1998.

[19] Rosenberg, Doug; Scott, Kendall. *Use Case Driven Object Modeling with UML: A practical approach. Addison-Wesley. EUA, 1999.*

[20] Rumbaugh, James; Jacobson, Ivar; Booch, Grady. *The unified modeling language reference manual.* Addison-Wesley. EUA, 1999.

[21] Schneider, Geri; Winters, Jason. *Applying Use Cases: A practical guide.* Addison-Wesley. EUA, 1998.

[22] Sendall, Shane; Strohmeier, Alfred. *From use caso to system operation specifications.* 3th International Conference on the Unified Modeling Language: UML 2000. Reino Unido, 2000.

[23] Wirfs-Brock, Rebecca. *Designing scenarios: making the case for the use case framework. Smalltalk report.* EUA, 1993.

[24] Leffingwell, Dean; Widrig, Don. *Managing Software Requirements: A Unified Approach.* Addison-Wesley. EUA, 2000.

[25] Cockburn, Alistair. *Writing Effective Use Cases*. Addison-Wesley. EUA, 2000.

[26] Shaw, Mary; Garlan, David. *Software Architecture: perspectives on an emerging discipline.* Prentice Hall. EUA, 1996.

[27] Visual Paradigm for UML. http://www.visual-paradigm.com. Acesso em 10/02/2004.

[28] Cristoph, Roberto; Felicíssimo, Carolina; Leite, Julio. *C&L: Uma ferramenta de edição e visualização de cenários e léxicos*.PUC-RJ. http://sl.les.inf.puc-rio.br/cel/. Acesso em 10/02/2004.