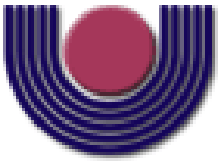

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Melhorando a Ferramenta JGOOSE

Mauro Brischke

CASCAVEL

2011

MAURO BRISCHKE

MELHORANDO A FERRAMENTA JGOOSE

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Victor Francisco Araya Santander

CASCAVEL

2011

MAURO BRISCHKE

MELHORANDO A FERRAMENTA JGOOSE

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Victor Francisco Araya Santander
(Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Adair Santa Catarina
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Carlos José Maria Olguin
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 03 de novembro de 2011.

DEDICATÓRIA

Dedico este trabalho aos meus pais **Glaci de Fatima Mendes Brischke** e **Elemar Brischke** por sempre estarem ao meu lado nessa jornada, dedicando-se para que tudo isso fosse possível.

AGRADECIMENTOS

Agradeço a meu irmão **Marcelo Brischke,** pelo apoio e incentivo, que foram de grande importância para me manter motivado e dedicado ao curso.

Aos meus grandes amigos, que são como irmãos, **Aymar Antonio Vilas Boas Pescador Junior, Bernardo Victor Engelke, Lucas Boschetto, Alex Picksius Gomes** e a todos os amigos próximos do qual tenho grande carinho e sentimento de irmandade para com todos.

A minha namorada **Cristiane Haubricht**, que mesmo tendo conhecido ao final desse curso, me proporcionou grande motivação para concluir o curso, além do carinho, amor e felicidades proporcionadas.

Lista de Figuras

2.1: Elementos do Modelo de Dependências Estratégicas 5
2.2: Exemplo de Modelo de Dependências Estratégicas 6
2.3: Principais Ligações Modelo de Razões Estratégicas 7
2.4: Exemplo do Modelo de Razões Estratégicas 8
3.1: Elementos do Diagrama de Casos de Uso 11
3.2: Diagrama de Casos de Uso 12
3.3: *Template* de Caso de Uso [11] 13
3.4: Exemplo de Caso de Uso Utilizando a *template* 14
4.1: Uma Visão Geral do Processo de Integração de i* e Casos de Uso [3] 16
5.1: Tela de Escolha de Diretriz da Ferramenta JGOOSE 21
5.2: Tela de Exibição do Caso de Uso Mapeado 22
5.3: Arquitetura da Ferramenta JGOOSE Demonstrando o seu Funcionamento 23
6.1: Selecionar o Ator Referente ao Sistema 26
6.2: Seleção das Diretrizes de Mapeamento 27
6.3: Casos de Uso Gerados pelo JGOOSE 28
6.4: Diagrama de Caso de Uso Gerado pelo JGOOSE 29

Lista de Abreviaturas e Siglas

GOOD	*Goals Into Object Oriented Development*
GOOSE	*Goal Into Object Oriented Standard Extension*
IDE	*Integrated Development Environment*
ITU	*International Telecommunication Union*
JGOOSE	*Java Goal Into Object Oriented Standard Extension*
ONU	Organização das Nações Unidas
OpenOME	*Organization Modelling Environment*
RUP	*Rational Unified Process*
SD	Modelo de Dependências Estratégicas
SR	Modelo de Razões Estratégicas
UML	*Unified Modeling Language*
XGOOD	*eXtended Goal Into Object Oriented Development*
XMI	*Xml Metadata Interchange*

Sumário

Lista de Figuras ..... **vi**
Lista de Abreviaturas e Siglas ..... **vii**
Sumário ..... **viii**
Resumo ..... **ix**
1. Introdução **1**
1.1 Introdução 1
1.2 Motivações 2
1.3 Proposta 2
1.4 Contribuições Esperadas 3
1.5 Estrutura do Trabalho 3
2. Framework i* **4**
2.1 Modelo de Dependências Estratégicas 4
2.2 Modelo de Razões Estratégicas 7
3. Casos de Uso UML **10**
3.1 Casos de Uso 10
4. Proposta de Derivação de Casos de Uso a partir de i* **15**
4.1 Diretrizes de Mapeamento 15
5. Melhorando a Ferramenta JGOOSE **20**
5.1 Funcionalidades Existentes 20
5.2 Melhorias Realizadas 22
6. Estudo de Caso **25**
7. Considerações Finais **30**
7.1 Trabalhos Futuros 30
Referências Bibliográficas **31**

Resumo

O desenvolvimento de *software* com qualidade é uma tarefa trabalhosa para os engenheiros de *software*, principalmente na fase inicial de concepção do sistema, na qual requisitos funcionais e não funcionais são gerados. Para que isso ocorra da melhor forma é necessário estudar e entender as questões não apenas que envolvem o sistema computacional, mas também os elementos organizacionais abordados pelo *framework*i*. Contudo é necessário termos como resultado o que realmente deve ser implementado no sistema computacional, com fidelidade aos requisitos exigidos. Uma das formas de representar requisitos funcionais é através de casos de uso UML (Unified Modeling Language). Contudo, se pudermos derivar estes casos de uso a partir de modelos organizacionais, podemos gerá-los de forma consistente com as intencionalidades expressas por atores do ambiente organizacional no qual o *software* irá funcionar. Também pode-se garantir que novos casos de uso do sistema gerados estão associados a novas metas estratégicas de atores da organização. Assim, observando estas vantagens é que foi proposta uma integração entre o *framework* i* e os casos de uso em UML, visando alcançar uma qualidade maior nessa fase de grande importância para a engenharia de *software*. Para apoiar essa integração é que foi iniciado o desenvolvimento de uma ferramenta denominada JGOOSE (Java Goal Into Object Oriented Stantard Extension). Contudo, esta ferramenta ainda precisa ser melhorada em vários aspectos tais como usabilidade, implementação de novas funcionalidades bem como possível integração com outras ferramentas *open source*. Este é o foco deste trabalho.

Palavras-chave: JGOOSE, Modelagem Organizacional, *framework* i*, Casos de Uso, Engenharia de S*oftware*.

Capítulo 1

Introdução

Para este capítulo inicial será feita uma abordagem geral sobre a proposta, motivações e principais contribuições desta monografia e, por último, uma descrição da estrutura deste trabalho.

1.1 Introdução

A engenharia de requisitos é uma fase de grande importância na elaboração de um sistema computacional que visa ajudar os desenvolvedores a melhor visualizar o problema da organização e obter um sistema viável, confiável e funcional, de forma a cumprir com as reais necessidades do cliente.

Nesse contexto destaca-se o uso de *frameworks* de modelagem organizacional como i*. Esse *framework* propõe uma abordagem orientada a atores com foco nas intencionalidades, relacionamentos e motivações entre os atores, o que permite compreender melhor a organização e as relações entre os participantes [1][2]. Dentre os trabalhos utilizando o *framework* i* destaca-se a proposta apresentada por Santander [3] a qual propõe derivar casos de uso a partir de modelos i*. Para dar suporte computacional a esta proposta foi desenvolvida uma ferramenta denominada JGOOSE (Java Goal Into Object Oriented Stantard Extension) [4], que visa à integração dos diagramas i* e Casos de Uso UML, gerando de forma automática os Casos de Uso a partir dos modelos criados no *framework* de modelagem i* usando as diretrizes propostas por Santander [3]. Contudo, esta ferramenta ainda possui diretrizes e algumas funcionalidades desejadas não implementadas. Além disso, esta ferramenta precisa ser melhorada para atender a alguns requisitos não funcionais tais como usabilidade, desempenho, portabilidade, manutenibilidade, entre outros.

Mais especificamente, pretende-se gerar modelos de Casos de Uso textuais e gráficos a partir de modelos i* considerados como base. Estes modelos serão gerados em um formato

que facilite sua comunicação e utilização por outras ferramentas *Open Source*, através do formato XMI (XML Metadata Interchange) [6].

Também pretende-se documentar a nova versão com os artefatos essenciais a um processo de desenvolvimento orientado a objetos.

1.2 Motivações

É de grande importância que a fase de levantamento de requisitos funcionais e não funcionais seja feita da melhor maneira possível, atendendo as necessidades e funcionalidades do *software* desejado. Porém esse levantamento é um tanto abstrato e, muitas vezes, de difícil concepção.

Pensando em auxiliar os engenheiros de *software* nessa fase tão importante para o bom desenvolvimento de um sistema computacional é que surgem diversos trabalhos, pesquisas e ferramentas com o intuito de ajudar e facilitar esse processo. Mais especificamente a utilização de modelos organizacionais pode auxiliar na elicitação e análise de requisitos. Observando as técnicas existentes de modelagem organizacional, destaca-se a técnica i* a partir da qual, inclusive, pode-se gerar representações de requisitos funcionais na forma de casos de uso em UML como proposto em [3].

As diretrizes propostas por Santander [3], para derivar casos de uso a partir de modelos gerados via i*, são implementadas pela ferramenta JGOOSE, a qual precisa de melhorias significativas para poder ser utilizada de forma eficiente. Assim, o foco deste trabalho está em estudar, analisar e melhorar a referida ferramenta.

1.3 Proposta

Neste trabalhoestudou-se as técnicas de engenharia de requisitos, modelagem organizacional i*, casos de uso bem como a integração entre as mesmas baseando-se nas diretrizes propostas na tese de Santander [3]. Contudo, o objetivo principal da proposta era concluir a implementação do conjunto de diretrizes na ferramenta JGOOSE, de forma a obtermos uma ferramenta completa para auxiliar, consideravelmente, os membros da comunidade acadêmica e engenheiros de *software*na geração de Casos de Uso durante a engenharia de requisitos.

Para que fosse possível essa pesquisa e implementaçãofoi feito um levantamento bibliográfico sobre as técnicas e ferramentas que englobam o tema. Também foi realizada uma minuciosa análise e estudo dos códigos fontes da ferramenta JGOOSE, para dar continuidade nas implementações propostas, além de melhorias na ferramenta.

Também foi feito um estudo de caso utilizando a ferramenta proposta para demonstrar sua utilidade em um problema do mundo real e assim mostrar que é uma ferramenta que pode vir a auxiliar os engenheiros de *software*, tanto no mundo acadêmico quanto em situações empresariais.

1.4 Contribuições Esperadas

Com o fim dessa monografia obtivemos uma ferramenta de engenharia de *software* que ajuda, de forma satisfatória, aos engenheiros e estudantes da área na elaboração de sistemas mais consistentes e que atenda de forma satisfatória as reais necessidades a qual o sistema foi proposto. Para isso foi concluído o melhoramento da ferramenta JGOOSE, foco das implementações dessa proposta, de forma que realize a geração automatizada dos Casos de Uso a partir de modelos i*.

Para que possíveis melhorias ou mudanças na ferramenta possam ser feitas em trabalhos futuros, a mesma possui uma documentação adequada para esse fim, seguindo os bons princípios do desenvolvimento orientado a objetos.

1.5 Estrutura do Trabalho

Esse trabalho está dividido em 5 capítulos, do qual o capítulo 2 aborda o *framework* i*, demonstrando sua importância para a engenharia de *software*, assim como uma descrição de seus elementos e exemplos. O capítulo 3 apresenta os casos de uso em UML, trazendo seus principais conceitos e que serão de grande importância para o desenvolvimento desse trabalho. O capítulo 4 aborda a proposta de derivação de casos de uso a partir de i*, bem como as diretrizes de mapeamento entre as duas abordagens, além de exemplos para um bom entendimento das diretrizes. O capítulo 5 apresenta a ferramenta JGOOSE cujo principal objetivo é apoiar a proposta de derivação de casos de uso a partir de i*, tornando esse processo automatizado. Também nesse capítulo são apresentadas as melhorias realizadas na ferramenta, que é objetivo desse trabalho.

Capítulo 2

Framework i*

Para o processo de desenvolvimento de um sistema a compreensão da organização, contexto e lógicas de negócio, ou seja, os “porquês” do sistema e de suas funcionalidades são de grande importância [2].

O *framework* i*, proposto por Eric Yu[5], traz uma visão estratégica e intencional dos processos que envolvem o sistema e a organização [7]. Esse *framework* propõe uma abordagem orientada a atores organizacionais com foco nas intencionalidades, relacionamentos e motivações entre os mesmos, o que permite compreender melhor a organização e as relações entre os participantes [1][4].

Os atores são ligados a outros atores por meio de ligações que representam suas intenções um para com o outro, como por exemplo, atingir um objetivo ou completar uma tarefa ou requisitar um recurso.

Recentemente o *framework* i* se tornou um padrão internacional de modelagem de *software* aprovado pela norma *ITU-T Recommendation Z.151* em Genebra, na Suíça. ITU (*International Telecommunication Union*) é a agência da ONU (Organização das Nações Unidas) para as tecnologias de informação e comunicação [1], o que demonstra que esse *framework* é internacionalmente aprovado e de grande importância para os engenheiros de *software* e bastante utilizado, pois auxilia na construção de sistemas que atendam aos requisitos impostos de forma mais consistente.

O *Framework* i* é divido em dois modelos, SD (Modelo de Dependências Estratégicas) e SR (Modelo de Razões Estratégicas) que serão descritos a seguir.

2.1 Modelo de Dependências Estratégicas

É composto por um conjunto de nós e ligações onde cada nó representa um ator e cada ligação demonstra uma dependência de um ator com outro, que pode ser um recurso, tarefa, objetivo ou objetivo-*soft*, como mostrado na Figura 2.1 a seguir.

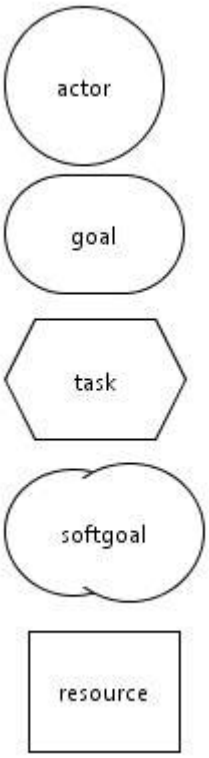

Figura 2.1: Elementos do Modelo de Dependências Estratégicas

Os atores nada mais são do que os participantes mais ativos do sistema, aqueles que dependem de outros atores para a satisfação de suas requisições. Na literatura é comum ver a utilização dos termos ***depender***, ***dependee*** e ***dependum***, onde *depender* é o ator que depende de outro para alguma atividade ou requisição de algo, *dependee* é o que satisfaz e atende a requisição do depender e *dependum* é o elemento ou relação de dependência entre um ator e outro [5]. A seguir descrevem-se detalhadamente os demais elementos que compõem o modelo SD mostrados na Figura 2.1.

Goal (Objetivo): São dependências de um ator para com o outro em relação a um objetivo especifico, ou seja, o *depender* depende do *dependee* para a satisfação de um objetivo. A forma como o objetivo será satisfeito passa a ser de responsabilidade do *dependee*.

Task (Tarefa): São soluções que proveem operações, respostas, representações de dados. Um ator requisita a execução de uma determinada tarefa para outro ator, ou seja, o *depender* depende do *dependee* para executar uma atividade ou tarefa, e a forma de realização desta tarefa é prescrita pelo *depender*. O *dependee* pode falhar na realização da tarefa.

Softgoal (Objetivos-soft): O objetivo-*soft* é mais como um desejo de um ator para com relação ao outro, pode vir a se tornar requisito não funcional do sistema nas etapas posteriores

da análise. O *depender* depende do *dependee* para a realização de algumas tarefas que satisfazem o objetivo-*soft*.

Resoure (Recurso): É a dependência de dados ou informações entre atores para a realização de tarefas e/ou objetivos.

O modelo SD expressa à rede de intencionalidades entre os atores, retratando as dependências estratégicas entre os mesmos [5][8]. Utilizaremos um exemplo simples para auxiliar no entendimento desse modelo e de seus elementos (Figura 2.2).

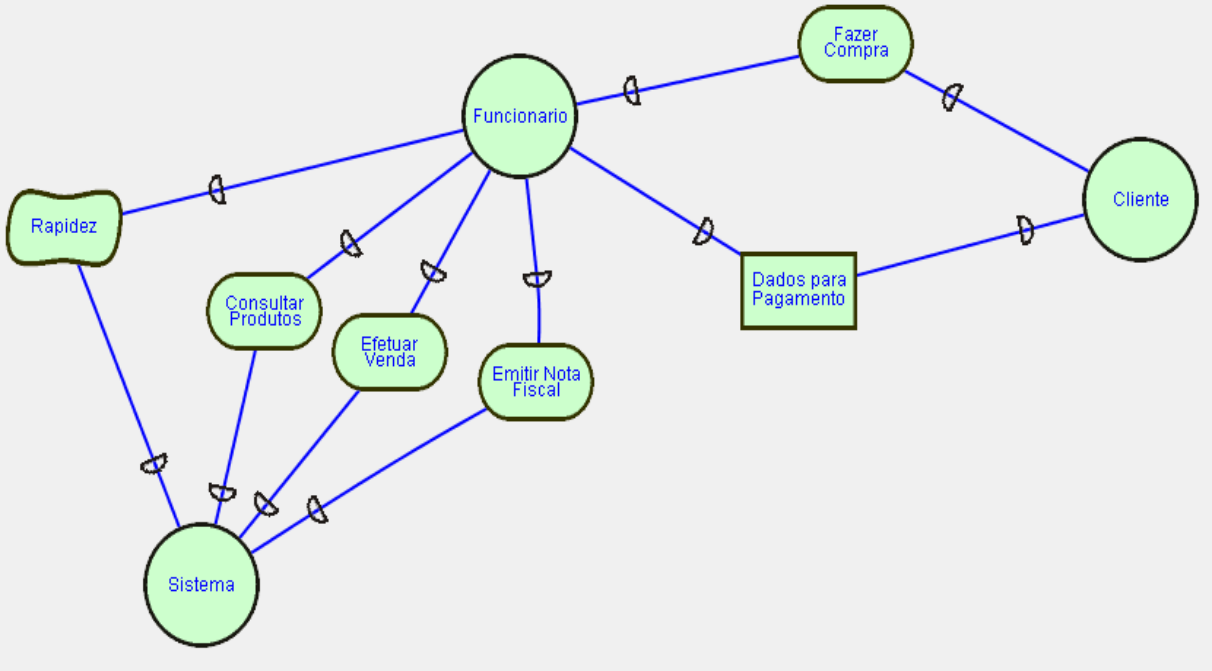

Figura 2.2: Exemplo de Modelo de Dependências Estratégicas

Explicando de forma mais detalhada a Figura 2.2, que trata de um sistema genérico de compra e venda, vemos que o ator "Cliente" depende do ator "Funcionário" para atingir o objetivo de "Fazer Compra" e o "Funcionário" por sua vez depende do ator "Sistema" para atingir o objetivo de "Efetuar Venda", ainda o "Funcionário" depende de uma solicitação do recurso "Dados para Pagamento" do ator "Cliente". O ator "Funcionário" tem uma dependência do "Sistema" para realizar o objetivo de "Consultar Produtos", também depende do "Sistema" para realizar o objetivo de "Emitir Nota Fiscal", que poderia estar pendente. Ainda o ator "Funcionário" depende do "Sistema"para atingir o objetivo-*soft* "Rapidez", o qual claramente representa um requisito não funcional que deve ser atendido pelo sistema.

2.2 Modelo de Razões Estratégicas

O modelo de razão estratégica explora as intenções específicas de cada ator e permite ainda a modelagem de decomposições de tarefas, fornecendo uma representação das razões atribuídas às dependências [8] e tornando o modelo ainda mais rico de informações importantes, satisfazendo uma gama ainda maior de requisitos funcionais e não funcionais do sistema demonstrado através da expansão dos atores do modelo SD. Além dos elementos já apresentados no modelo SD, o modelo SR possui alguns tipos de ligações conforme apresentado na figura 2.3.

Figura 2.3: Principais Ligações Modelo de Razões Estratégicas

Essas são as principais ligações para a construção do modelo SR e serão mais bem descritas a seguir.

Mean-End (Meio-fim): Essas ligações indicam os meios para se chegar a um determinado fim, os meios são expressos geralmente como tarefas, uma vez que as tarefas já indicam como fazer algo, enquanto o fim é uma meta a ser atingida, podendo ser um objetivo, objetivo-*soft* ou recurso.

Decomposition (Decomposição): As ligações de decomposição, como o próprio nome sugere, servem para decompor uma tarefa, apresentando alguns requisitos para o cumprimento da tarefa a ser decomposta. As decomposições podem ser sub-objetivos, sub-tarefas, recursos e objetivos-*soft*.

Para melhor visualizar e entender o modelo SR mostraremos a expansão do ator “Sistema” apresentado na Figura 2.2, bem como suas ligações e razões estratégicas, que serão vistas na Figura 2.4, logo a seguir.

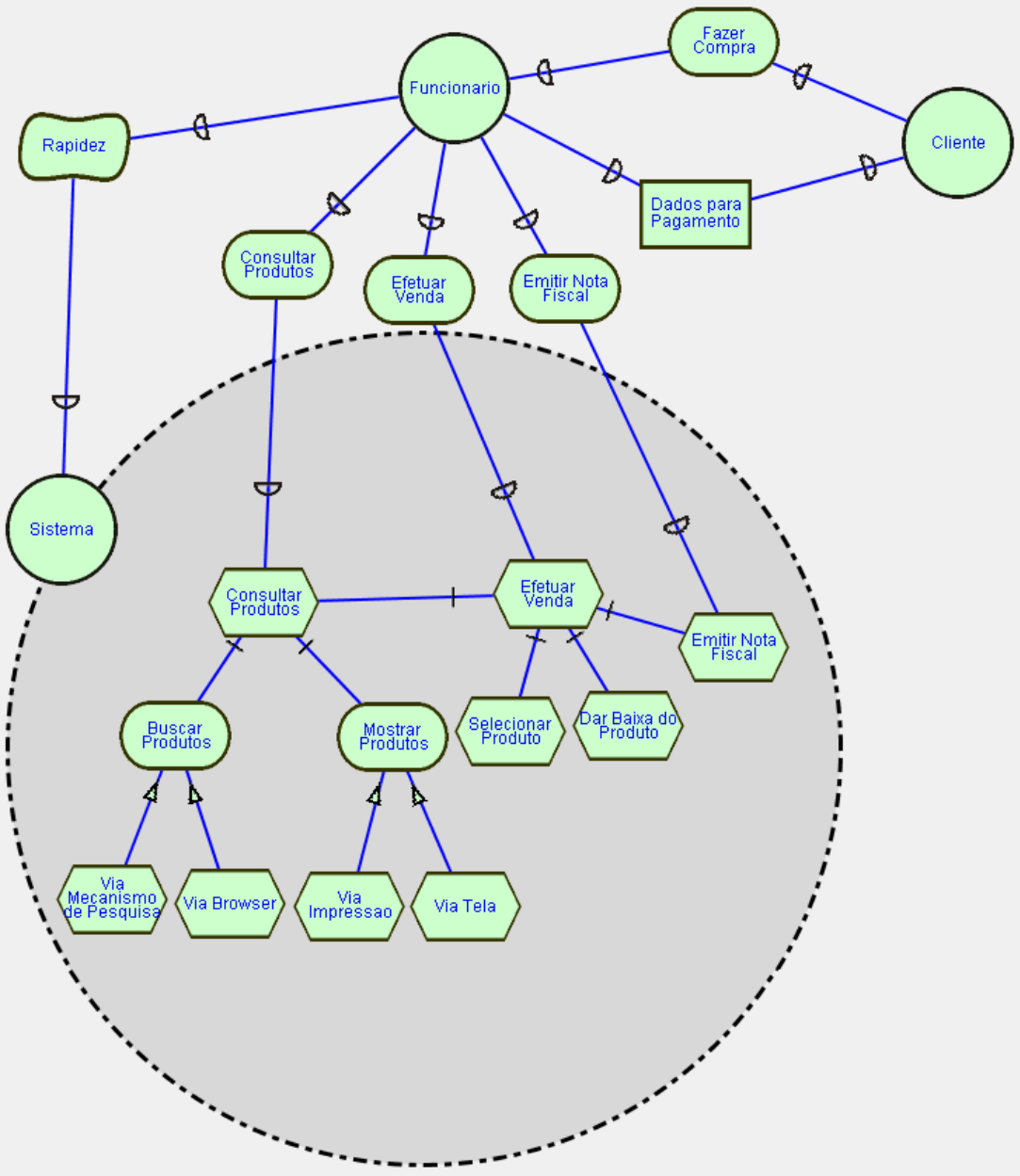

Figura 2.4: Exemplo do Modelo de Razões Estratégicas

Analisando o modelo SR, vemos que as razões estratégicas por trás de cada dependência se revelam de forma mais detalhada, isso pode ser exemplificado no objetivo de “Consultar Produtos” que quando detalhado é decomposto por duas ligações de decomposição, demonstrando que para atingir o objetivo principal é necessário que se realize o sub-objetivo “Buscar Produtos” e posteriormente “Mostrar Produtos” resultantes da busca. A busca ainda pode ser feita de duas formas “Via Mecanismo de Pesquisa” e “Via Browser” demonstrados pelas ligações meio-fim, onde, “Via Browser”, é um meio de se obter o fim de “Buscar

Produtos", assim como, "Via Mecanismo de Pesquisa". O sub-objetivo "Mostrar Produtos" possui dois meios para ser alcançada, "Via Impressão" e "Via Tela". O objetivo "Efetuar Venda" é detalhado por meio de ligações de decomposição, que como já mencionado acima, elas decompõem esse objetivo em "Consultar Produtos", "Selecionar Produto", "Dar Baixa do Produto" e "Emitir Nota Fiscal", onde essa ultima decomposição pode ser acessada de forma independente, demonstrada na tarefa "Emitir Nota Fiscal" solicitado pelo ator "Funcionário".

Capítulo 3

Casos de Uso UML

Para que um *software* atenda de forma satisfatória as reais necessidades para as quais ele foi proposto, faz-se necessário o bom entendimento de todos os aspectos organizacionais através de modelos vistos no *framework* i*. No entanto, ainda é de grande importância que haja uma ampla compreensão do sistema e como transformar essa representaçãoem implementação do sistema. Para apoiar melhor essa transformação é que surge a UML (Unified Modeling Language).

A UML pode ser bem descrita como "*uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software*" [9]. A linguagem UML facilita modificações futuras no sistema, mantendo assim a qualidade do *software* em longo prazo.

Existem diversos diagramas e descrições em UML que auxiliam no desenvolvimento de um *software*, entre eles temos os casos de uso. Os casos de uso propiciam o entendimento do funcionamento do sistema a todas as partes, programadores, especialistas e usuários finais, também contribuem na validação do sistema enquanto é desenvolvido [9].

Os casos de uso são usados em padrões de arquitetura e desenvolvimento de *software*, aprovados comercialmente, e bastante conhecidos na literatura, como o RUP (Rational Unified Process), que faz uso das técnicas de engenharia de *software* descritas na UML [10].

A seguir teremos uma descrição, exemplos de diagramas de casos de uso e a apresentação de um *template*, para descrição textual dos casos de uso de forma mais detalhada.

3.1 Casos de Uso

A linguagem UML utiliza-se de vários diagramas e especificações para auxiliar os engenheiros de *software* a melhor entender o sistema pretendido e assim possibilitar uma implementação mais segura e comprometida com os requisitos do *software*. Um dos principais diagramas é o de casos de uso, que visa à organização e os comportamentos do sistema tendo como base atores e suas ligações [9]. De forma geral os casos de uso

apresentam o que o sistema realmente irá fazer do ponto de vista do usuário, através de um conjunto de ações de ambas as partes.

Na Figura 3.1 são apresentados os elementos do diagrama de casos de uso UML.

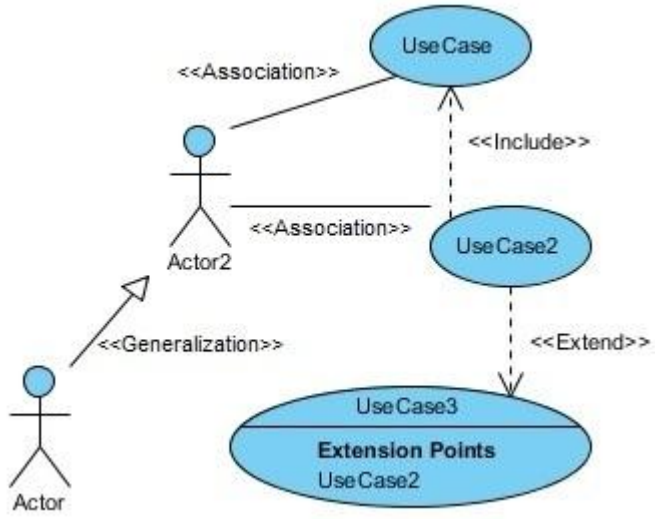

Figura 3.1: Elementos do Diagrama de Casos de Uso

A figura anterior apresenta os principais elementos que compõem um diagrama de casos de uso UML, que iremos descrevê-los a seguir.

Actor (Ator): Os atores são os usuários que interagem diretamente com o sistema, podendo ser pessoas, componentes de *hardware* e até outros sistemas que interagem com o sistema proposto.

Use Case (Caso de Uso): Os casos de uso expressam o que o sistema faz, porém não apresentam a forma como o sistema realiza essas funções [9].

Association (Associação): Essa é uma ligação utilizada entre os atores e os casos de uso para representar sua comunicação, ou seja, que podem enviar e receber mensagens entre si.

Include (Inclusão): Uma ligação de inclusão demonstra que um caso de uso utiliza informações do outro caso de uso, mas não necessariamente o contrario é verdadeiro.

Extend (Extensão): A ligação de extensão apresenta uma divisão para um caso de uso, a qual serve para separar o comportamento obrigatório, que é o caso de uso base, do comportamento opcional, que é o caso de uso estendido.

Generalization (Generalização): A generalização descreve uma ligação de herança, na qual um ator herda a estrutura e o comportamento de outro.

Um exemplo desse diagrama é visto na Figura 3.2, o qual é baseado nos diagramas construídos no capítulo 2 (Figura 2.2 e 2.4).

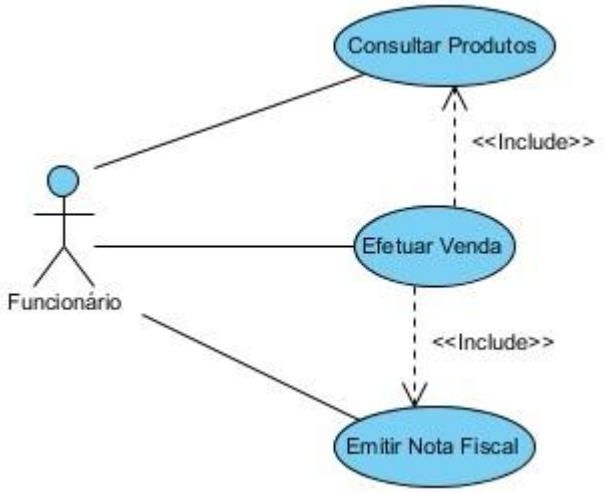

Figura 3.2: Diagrama de Casos de Uso

Vemos que no diagrama o ator "Funcionário" interage com o sistema através dos casos de uso, ou funcionalidades do sistema, descritos como "Consultar Produtos", "Efetuar Venda" e "Emitir Nota Fiscal". Observando mais atentamente a Figura 3.2, podemos verificar que para efetuar uma venda junto ao sistema é necessário consultar os produtos e emitir nota fiscal. Estas relações são expressas por meio do estereótipo <<include>>. Contudo, cabe ressaltar que também o ator funcionário pode realizar a consulta aos produtos e a emissão de notas fiscais de forma independente.

Os casos de uso podem ser expressos em forma de texto, permitindo representar as ações que devem ser realizadas entre o usuário e o sistema para levara cabo o objetivo associado ao caso de uso. Isto permite uma melhor compreensão do caso de uso por todos os envolvidos [11]. Dessa forma, para complementar e descrever o diagrama de casos de uso de forma mais consistente, neste trabalho adota-se a descrição textual dos casos de uso via *template* proposto por Cockburn [11]. Na Figura 3.3, modificada de [4], é apresentado o *template* criado por Cockburn.

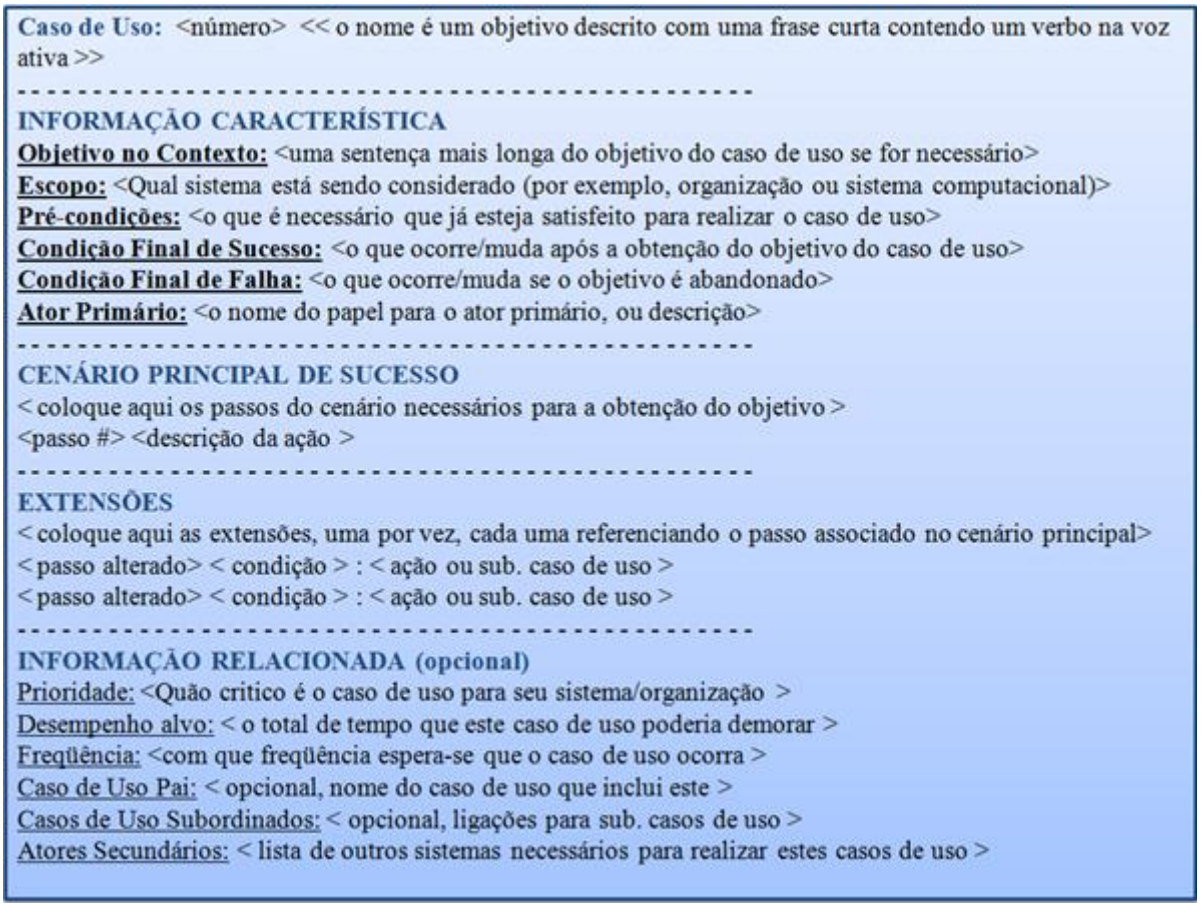

Figura 3.3: *Template*de Caso de Uso [11]

Como por exemplo, iremos utilizar esse *template* para descrever o caso de uso “Efetuar Venda”, apresentado na Figura 3.2.

Caso de Uso: Efetuar Venda

INFORMAÇÃO CARACTERÍSTICA

Objetivo no Contexto: O objetivo é efetuar a venda de um produto
Escopo: Sistema computacional
Pré-condições: É necessário que o funcionário esteja no sistema
Condição Final de Sucesso: Uma venda é alcançada e as informações referentes a venda são armazenadas no sistema.
Condição Final de Falha: O sistema volta para um estado anterior
Ator Primário: Funcionário

CENÁRIO PRINCIPAL DE SUCESSO

1. O Funcionário consulta um produto;
   O caso de uso Consultar Produtos é incluído << include >>;
2. O Funcionário seleciona o produto;
3. O Sistema da a baixa;
4. O Funcionário emite a nota fiscal;
   O caso de uso Emitir Nota Fiscal é incluído << include >>;

EXTENSÕES

INFORMAÇÃO RELACIONADA (opcional)

Prioridade: Possui prioridade alta
Desempenho alvo: Poucos minutos, para evitar um melhor atendimento ao cliente
Freqüência: Varias vezes ao dia
Caso de Uso Pai:
Casos de Uso Subordinados: Consultar Produtos e Emitir Nota Fiscal
Atores Secundários:

Figura 3.4: Exemplo de Caso de Uso Utilizando a *template*

Esse exemplo apresenta algumas inclusões para outros casos de uso, demonstrado pelo <<include>>, mostrando que o caso de uso exemplificado apresenta um maior detalhamento.

Cada caso de uso pode ser amplamente detalhado, contudo, esse exemplo apresenta um nível de detalhamento suficiente para o entendimento do caso de uso e foi descoberto através da aplicação de algumas diretrizes que utilizam o *framework* i* como base, o qual será visto no próximo capítulo. Esse *template* é o mesmo utilizado na ferramenta JGOOSE.

Capítulo 4

Proposta de Derivação de Casos de Uso a partir de i*

Na engenharia de requisitos é cada vez mais comum a idéia de que a fase de especificação de requisitos tenha informações relacionadas à organização, modelos de negócios e outras informações além das especificações do *software*, para que exista um entendimento do contexto em que o sistema irá funcionar [12].

Uma dificuldade dos engenheiros de *software* é encontrar o que realmente é importante para o usuário, considerando os objetivos organizacionais. Para alcançar esse objetivo existem técnicas que auxiliam nesse processo, mas que necessitam de complementos [3].

Abordagens baseadas em cenários tem se destacado pela facilidade de entendimento dos usuários e desenvolvedores do sistema [13].Essas abordagens têm ajudando para a elicitação de requisitos e até mesmo para a validação do sistema ao longo de seu desenvolvimento. Contudo, estas abordagens não expressam todos os desejos organizacionais envolvidos na criação do sistema [14][15].

Nesse contexto é consenso, na engenharia de requisitos, o fato que podemos melhorar a modelagem e elicitação de requisitos para os nossos sistemas se evoluirmos de modelos com aspectos organizacionais, como o *framework* i*, que surgiu com esse propósito, para uma elicitação e especificação de requisitos mais completa, utilizando-se da técnicade casos de uso, que se destaca por ser importante na Linguagem de Modelagem Unificada (UML) [9].

Para uma evolução de casos de uso a partir de modelos i*, de forma satisfatória, foram propostas diretrizes que apoiam o mapeamento dessas informações [3].

4.1 Diretrizes de Mapeamento

O mapeamento e a derivação de casos de uso a partir de modelos organizacionais i* possibilita uma melhor elicitação dos requisitos [3]. A Figura 4.1 mostra uma visão geral do mapeamento de i* para casos de uso.

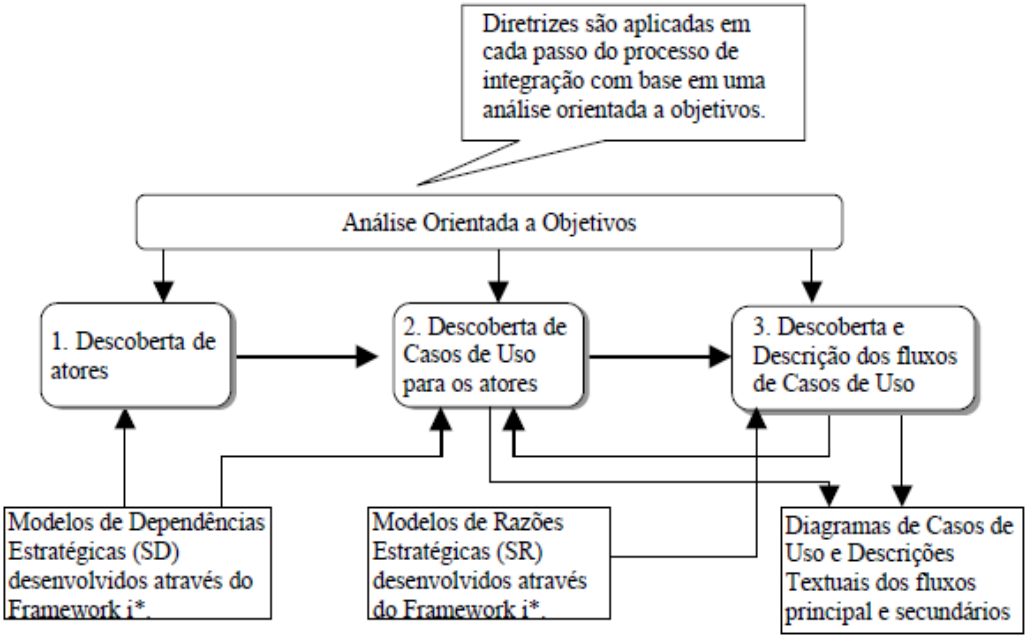

Figura 4.1: Uma Visão Geral do Processo de Integração de i* e Casos de Uso [3]

O mapeamento é feito através de diretrizes que utilizam como base os modelos SD e SR. Através desses dois modelos é feita a descoberta dos atores, casos de uso, cenários principais e secundários e todos os demais elementos dos casos de uso, do qual, cada diretriz possui um propósito especifico nesse processo.

A seguir apresentamos as diretrizes propostas em [3] com algumas modificações vistas em [16], para derivar casos de uso em UML a partir de i*. Cada diretriz é exemplificada com o estudo de caso brevemente descrito e utilizado no capítulo 2.

1ºPasso da Proposta: Descoberta de atores

Diretriz 1: todo ator em i* deve ser analisado para um possível mapeamento para ator em caso de uso. Por exemplo, o ator “Funcionário” na Figura 2.2 é um candidato;

Diretriz 2: inicialmente, deve-se analisar se o ator em i* é externo ao sistema computacional pretendido. Caso o ator seja externo ao sistema, o mesmo é considerado candidato a ator em Casos de Uso. Por exemplo, o ator “Funcionário” é externo ao Sistema;

Diretriz 3: o ator i* candidato deve ter pelo menos uma dependência com o sistema computacional pretendido. Por exemplo, o ator “Funcionário” possui varias ligações de dependência com o sistema;

Diretriz 4: atores em i*, relacionados através do mecanismo ISA, ou seja, com heranças de suas atividades nos modelos organizacionais e mapeados individualmente para atores em casos de uso (aplicando diretrizes 1, 2 e 3), serão relacionados no diagrama de casos de uso

através do relacionamento do tipo <<generalização>>. Por exemplo, se existisse na Figura 2.2 um ator "Gerente" que tivesse uma relação ISA com o ator "Funcionário", ambos seriam candidatos a atores em casos de uso e no diagrama teriam uma ligação de <<generalização>>.

2ºPasso da Proposta: Descoberta de Casos de Uso.

Diretriz 5: para cada ator descoberto para o sistema (1º passo da proposta), devemos observar todas as suas dependências (*dependum*) como *dependee* em relação ao ator que representa o sistema computacional pretendido (*depender*), visando descobrir casos de uso para o ator. Inicialmente, para efeitos de sistematização do mapeamento, bem como uma organização das informações relevantes, recomendamos criar uma tabela contendo os atores já descobertos para o modelo de casos de uso e adicionar a esta tabela as informações associadas com as dependências destes atores, observando os mesmos como *dependee* e o sistema computacional pretendido como *depender* (sistema computacional → *dependum* → ator em consideração).

SubDiretriz 5.1: deve-se avaliar as dependências do tipo objetivo associadas com o ator. Objetivos em i* podem ser mapeados para objetivos em Casos de Uso. Por exemplo, o objetivo "Efetuar Venda" da Figura 2.2 pode se tornar um objetivo em Caso de Uso;

SubDiretriz 5.2: deve-se avaliar as dependências do tipo tarefa, associadas com o ator. Se um ator depende de outro ator para realizar uma tarefa, deve-se investigar se esta tarefa necessita ser refinada em sub-tarefas;

SubDiretriz 5.3: deve-se avaliar as dependências do tipo recurso, associadas com o ator. Se um ator depende de outro ator para obter um recurso, por que o mesmo é requerido? Se para esta resposta existe um objetivo mais abstrato, o mesmo será candidato a ser um objetivo de um Caso de Uso para este ator. Por exemplo, o recurso "Dados para Pagamento" é um candidato a ser um objetivo em Caso de Uso;

SubDiretriz 5.4: deve-se avaliar as dependências do tipo objetivo-*soft*, associadas com o ator. Tipicamente uma dependência do tipo objetivo-*soft* em i* é um requisito não funcional associado ao sistema pretendido. Assim um objetivo-*soft* não representa um caso de uso do sistema, mas um requisito não funcional associado com o caso de uso específico do sistema ou com o sistema como um todo. Por exemplo, o objetivo-*soft*"Rapidez" representa um requisito não funcional do sistema como um todo.

Diretriz 6: analisar a situação especial na qual um ator de sistema (descoberto seguindo as diretrizes do passo 1) possui dependências (como *depender*) em relação ao ator em i* que

representa o sistema computacional pretendido ou parte dele. (ator → *dependum* → sistema computacional). Por exemplo, o ator "Funcionário" possui um objetivo de "Efetuar Venda" no sistema, logo, esse objetivo já mapeado pela diretriz 5.1, irá ser um caso de uso.

Diretriz 7: classificar cada caso de uso de acordo com seu tipo de objetivo associado (objetivo contextual, objetivo de usuário, objetivo de subfunção). Essa diretriz mapeia o nível de um caso de uso, que nessa proposta foi omitido por não apresentar grande relevância para o caso de uso, mas em outros contextos pode ser interessante ter isso presente nos casos de uso.

3ºPasso da Proposta: Descoberta e descrição do fluxo principal e alternativo dos Casos de Uso.

Diretriz 8: analisar cada ator e seus relacionamentos no Modelo de Razões Estratégicas para extrair informações que possam conduzir à descrição de fluxos principal e alternativos, bem como pré-condições e pós-condições dos casos de uso descobertos para o ator. Para isso precisamos analisar os sub-componentes em uma ligação de decomposição de tarefa em um possível mapeamento para passos na descrição do cenário primário (fluxo principal) de casos de uso. Também devemos analisar ligações do tipo meio-fim em um possível mapeamento para passos alternativos na descrição de casos de uso. Ainda se um fim for um objetivo ou tarefa que possua alguma dependência previamente mapeada para caso de uso, essas alternativas podem descrever um <<extend>> e, por fim, devemos analisar os relacionamentos de dependências de sub-componentes no modelo de Razões Estratégicas em relação a outros atores do sistema. Estas dependências podem originar pré-condições e pós-condições para os casos de uso descobertos. Por exemplo, a tarefa "Selecionar Produto" da Figura 2.4 seria um passo do cenário principal do caso de uso "Efetuar Venda", devido a sua ligação de decomposição de tarefa, da mesma forma as tarefas, "Consultar Produtos", "Dar Baixa do Produto" e "Emitir Nota Fiscal", também fazem parte do cenário principal do caso de uso.

Diretriz 9: Investigar a possibilidade de derivar novos objetivos de casos de uso a partir da observação dos passos nos cenários (fluxos de eventos) dos casos de uso descobertos. Cada passo de um caso de uso deve ser analisado para verificar a possibilidade do mesmo ser refinado em um novo caso de uso.

Diretriz 9.1: Inicialmente deve-se averiguar qual é o objetivo que se quer atingir com a ação que o passo representa na descrição do caso de uso;

Diretriz 9.2: Descoberto o objetivo, deve-se averiguar a necessidade de decompor/refinar o objetivo para que o mesmo possa ser alcançado;

Diretriz 9.3: Um novo caso de uso será gerado a partir do objetivo analisado, se pudermos definir os passos que representam a necessidade de interações adicionais entre o ator e o sistema para se atingir o sub-objetivo desejado;

Diretriz 9.4: Adicionalmente, pode-se encontrar novos objetivos e cenários com base na observação dos obstáculos de objetivos já descobertos.

A diretriz 9 é de responsabilidade do engenheiro de *software*, pois a análise proposta pela diretriz depende do engenheiro de *software* e não pode ser automatizada completamente.

Diretriz 10: Desenvolver o diagrama de casos de uso utilizando os casos de uso descobertos, bem como observando os relacionamentos do tipo <<include>>, <<extend>> e <<generalization>> usados para estruturar as especificações dos casos de uso.

Essas são as diretrizes que possibilitam a derivação de casos de uso a partir de modelos organizacionais i* e que são implementadas parcialmente pela ferramenta JGOOSE. A nossa proposta de melhoramento dessa ferramenta tem por objetivo concluir as implementações das diretrizes. O capítulo a seguir apresenta a ferramenta JGOOSE.

Capítulo 5

Melhorando a Ferramenta JGOOSE

A ferramenta JGOOSE (Java Goal Into Object Oriented Standard Extension) teve seu desenvolvimento motivado pela possibilidade de integração de técnicas da engenharia de *software* para modelagem de sistemas computacionais já apresentadas nos capítulos anteriores. O JGOOSE teve como base outros sistemas (GOOD, XGOOD, GOOSE) que tinham a mesma motivação [4], mas que devido à defasagem de tecnologia, entre outros fatores, optou-se pelo desenvolvimento de uma ferramenta que herdasse as funcionalidades de seus antecessores, com relativo avanço na implementação das diretrizes do capítulo anterior. Contudo ainda necessita da implementação de três diretrizes, sendo estas, as diretrizes 8, 9 e 10. Cabe ressaltar que as funcionalidades geradas na ferramenta, oriundas da implementação das referidas diretrizes, são essenciais para que o engenheiro de requisitos e o engenheiro de *software* façam uso de todos os benefícios advindos da proposta. Também detectamos a necessidade de melhorias no que tange a aspectos de usabilidade e suporte a outros formatos de arquivo.

5.1 Funcionalidades Existentes

A princípio a ferramenta JGOOSE utiliza-se de diagramas i*, SD e SR, criados no formato Telos [17], oferecido pelo ambiente OME3 (Organization Modelling Environment), que também possui integração com outras ferramentas que suportam a engenharia de *software*. O arquivo no formato Telos é à entrada da ferramenta JGOOSE, ele passará por um pré-tratamento e em seguida são escolhidas as diretrizes que se deseja utilizar no mapeamento para casos de uso, como visto na Figura 5.1.

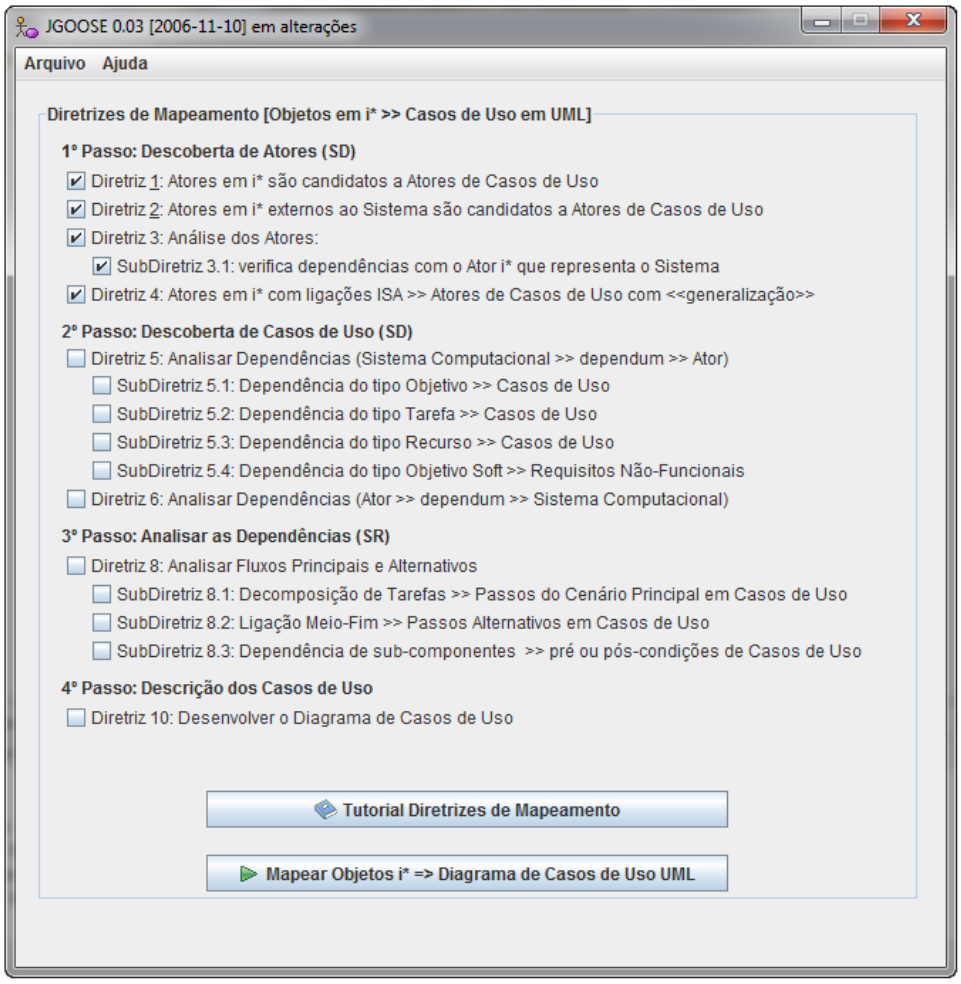

Figura 5.1: Tela de Escolha de Diretriz da Ferramenta JGOOSE

Após a escolha das diretrizes geram-se os casos de uso. Contudo, as diretrizes foram implementadas até a diretriz 6. Na figura 5.2 está um caso de uso gerado pela ferramenta no formato do *template* de Cockburn.

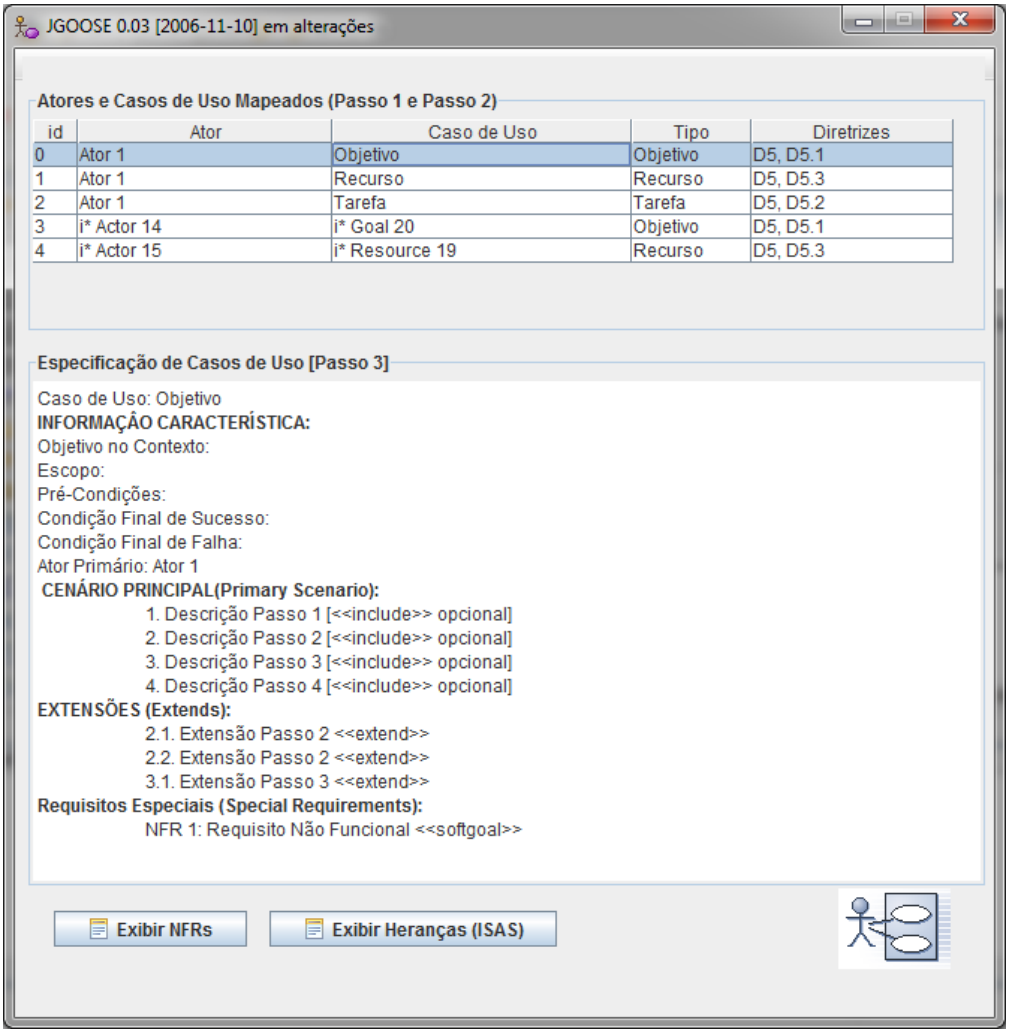

Figura 5.2: Tela de Exibição do Caso de Uso Mapeado

5.2 Melhorias Realizadas

A ferramenta mantém a sua arquitetura inicial, apresentando um foco maior na conclusão das diretrizes e posteriormente a implementação de funcionalidades de integração com outras ferramentas da engenharia de *software*, além de uma documentação mais adequada. A arquitetura do JGOOSE mostrando os passos tomados pela ferramenta para o seu funcionamento, retirada e modificada de [4], é apresentada na figura 5.3.

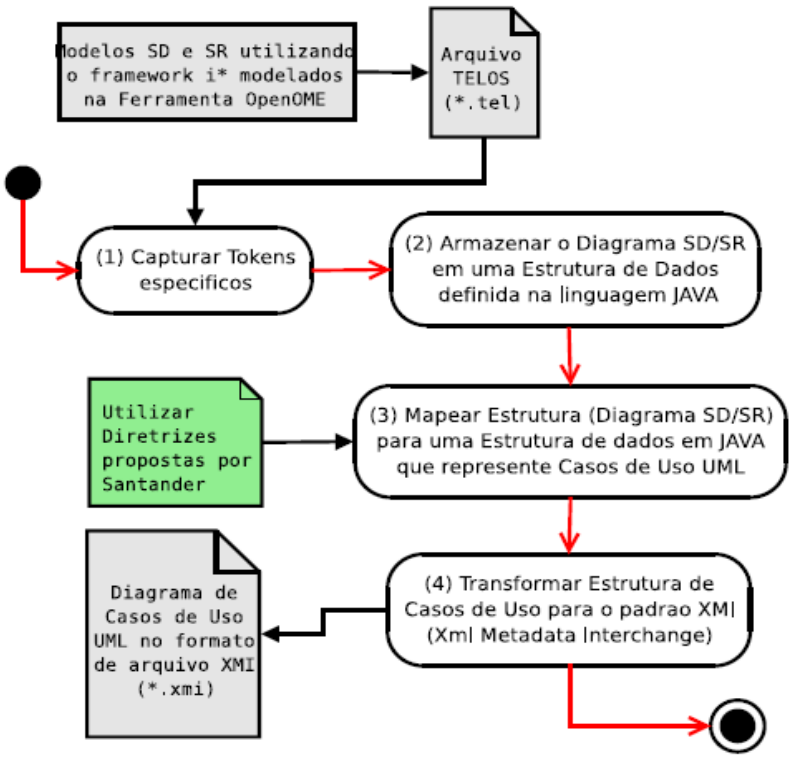

Figura 5.3: Arquitetura da Ferramenta JGOOSE Demonstrando o seu Funcionamento

A linguagem utilizada para o desenvolvimento do JGOOSE é a linguagem orientada a objetos Java. O projeto JGOOSE está sendo desenvolvido na IDE NetBeans, por haver maior conhecimentoe familiarização com a ferramenta. A linguagem de programação Java foi mantida para realização das melhorias na ferramenta, por questões como o conhecimentoda linguagem e a disponibilidade de material sobre a linguagem.

Como visto na Figura 5.3, foi visada a implementação de uma integração com outras ferramentas de apoio a engenharia de *software* através da extensão XMI (Xml Metadata Interchange)[6], para exportar os casos de uso.

A parte de usabilidade da ferramenta JGOOSE segue princípios de boa usabilidade e uma verificação da mesma, conforme proposto em [18].

Na verificação da usabilidade da ferramenta, verificou-se que nem todas as funcionalidades tem mensagens de sucesso e/ou erro implementadas e nem um sistema de ajuda mais elaborado para as mensagens de erro existentes, também na fase de visualização dos casos de uso não existe uma opção para voltar para a seleção das diretrizes, porém, existe a opção de abrir um novo arquivo .tel em qualquer momento. Essas verificações não foram implementadas de forma a serem satisfeitas, devido ao fator tempo disponível para as implementações.

Algumas pequenas melhorias foram realizadas, como uma pequena correção de *bug* na função de cancelar o processo de abrir o arquivo .tel, bem como a mudança na exibição do caso de uso, deixando de forma mais completa e de acordo com o *template* da Figura 3.3. Ainda foi incorporada a opção de visualizar a descrição das diretrizes de mapeamento em qualquer momento que o usuário necessitar.

Também foi permitido que o usuário da ferramenta altere os campos do *template* gerado, cumprindo a diretriz 9, ou seja, permitindo um refinamento manual do caso de uso.

A diretriz 8 foi implementada, gerando o cenário principal e secundário de cada caso de uso, e ainda, uma opção para salvar o *template* gerado pela ferramenta em formato de texto, para que pode ser editado pelas mais populares ferramentas de edição de texto e impresso.

Também foi implementado uma visualização do diagrama de cada caso de uso, referente a diretriz 10, e uma opção para que o usuário do JGOOSE possa exportá-los em formato XMI, de forma que possa ser importado por outras ferramentas da engenharia de *software*.

Capítulo 6

Estudo de Caso

Como estudo de caso para a ferramenta JGOOSE, foi utilizado o exemplo já descrito no capítulo 2. Foi criado o diagrama SR de um sistema de comércio genérico, na ferramenta OME3 no formato Telos, visto na Figura 2.4. Utilizando a ferramenta JGOOSE e todos os seus mapeamentos,foi possível gerar os casos de uso.A seguir mostraremos as telas da ferramenta desenvolvida seguindo o seu fluxo de execução.

A Figura 6.1 mostra como é feita a seleção do ator referente ao sistema computacional pretendido, essa seleção é realizada logo após a importação do arquivo .tel.

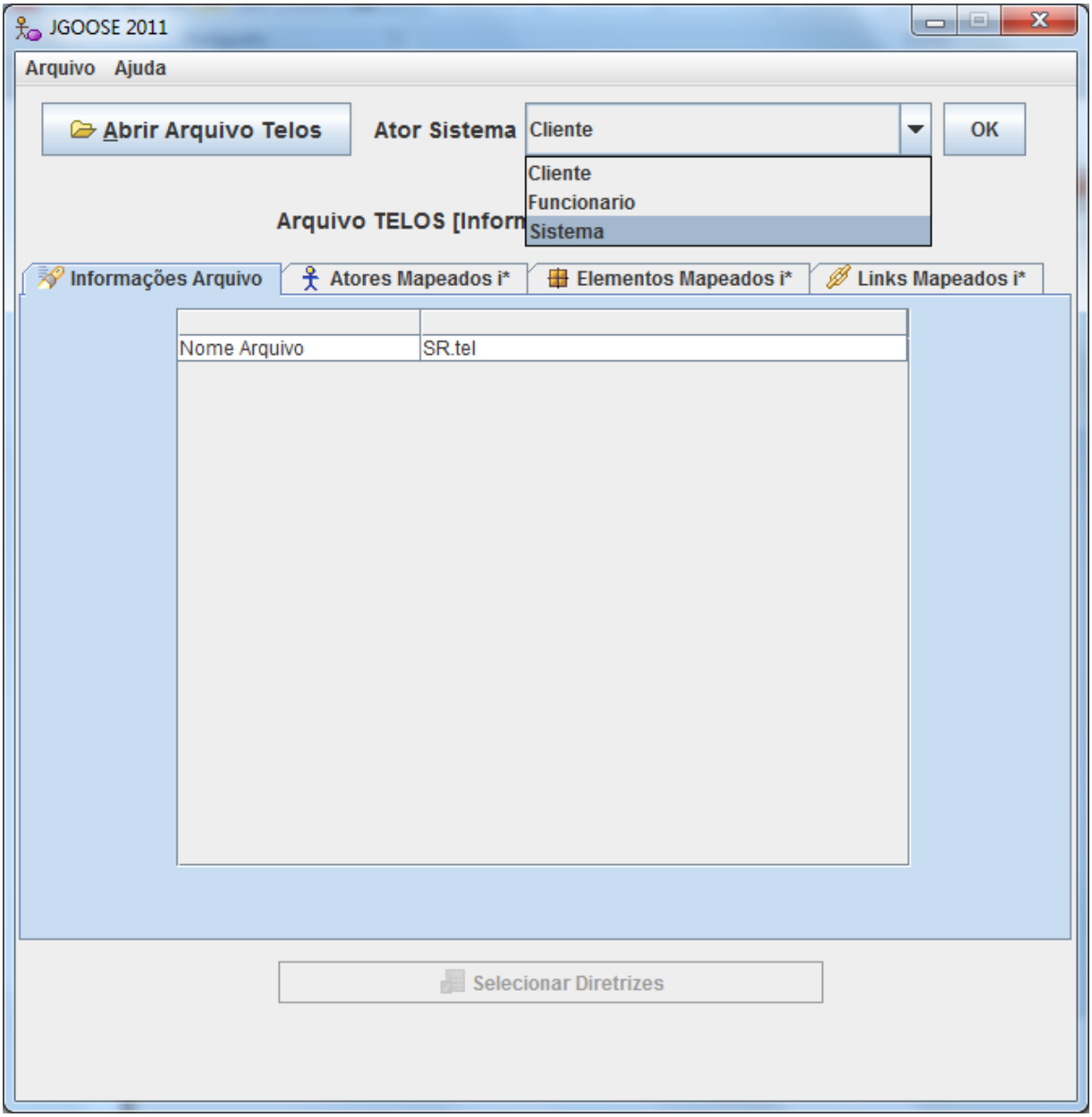

Figura 6.1:Selecionar o Ator Referente ao Sistema

Na Figura 6.2 é apresentada a tela de seleção das diretrizes que serão utilizadas no mapeamento dos casos de uso.

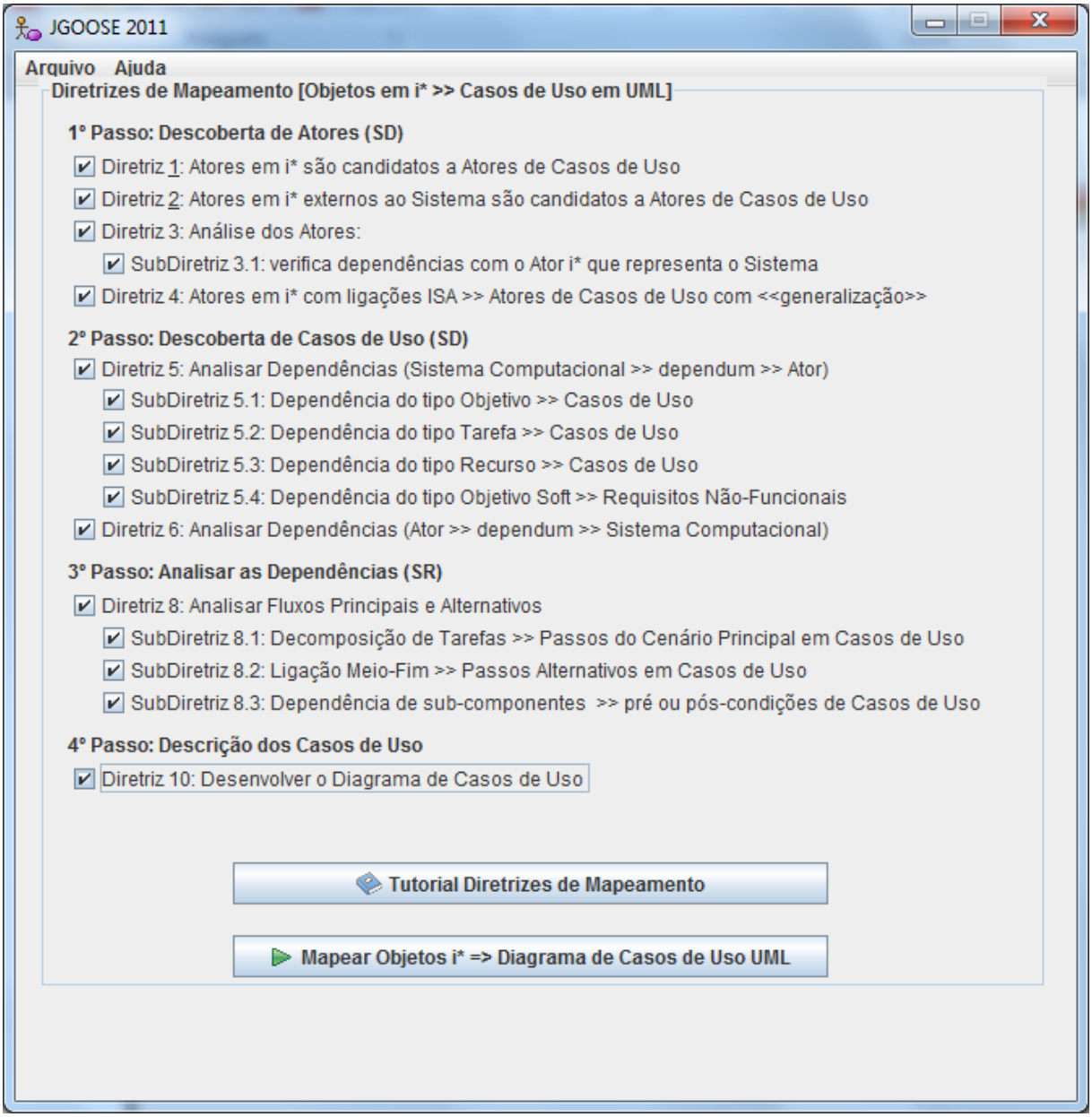

Figura 6.2: Seleção das Diretrizes de Mapeamento

Na Figura 6.3 vemos os casos de uso gerados, bem como sua descrição textual. É possível salvar a descrição textual dos casos de uso através do botão “Salvar Descrição”, que irá salvar o caso de uso, em formato que possa ser utilizado pelos principais editores de texto, e também através do botão “Diagrama”, representado por uma figura, é possível a visualização do diagrama para o caso de uso selecionado.

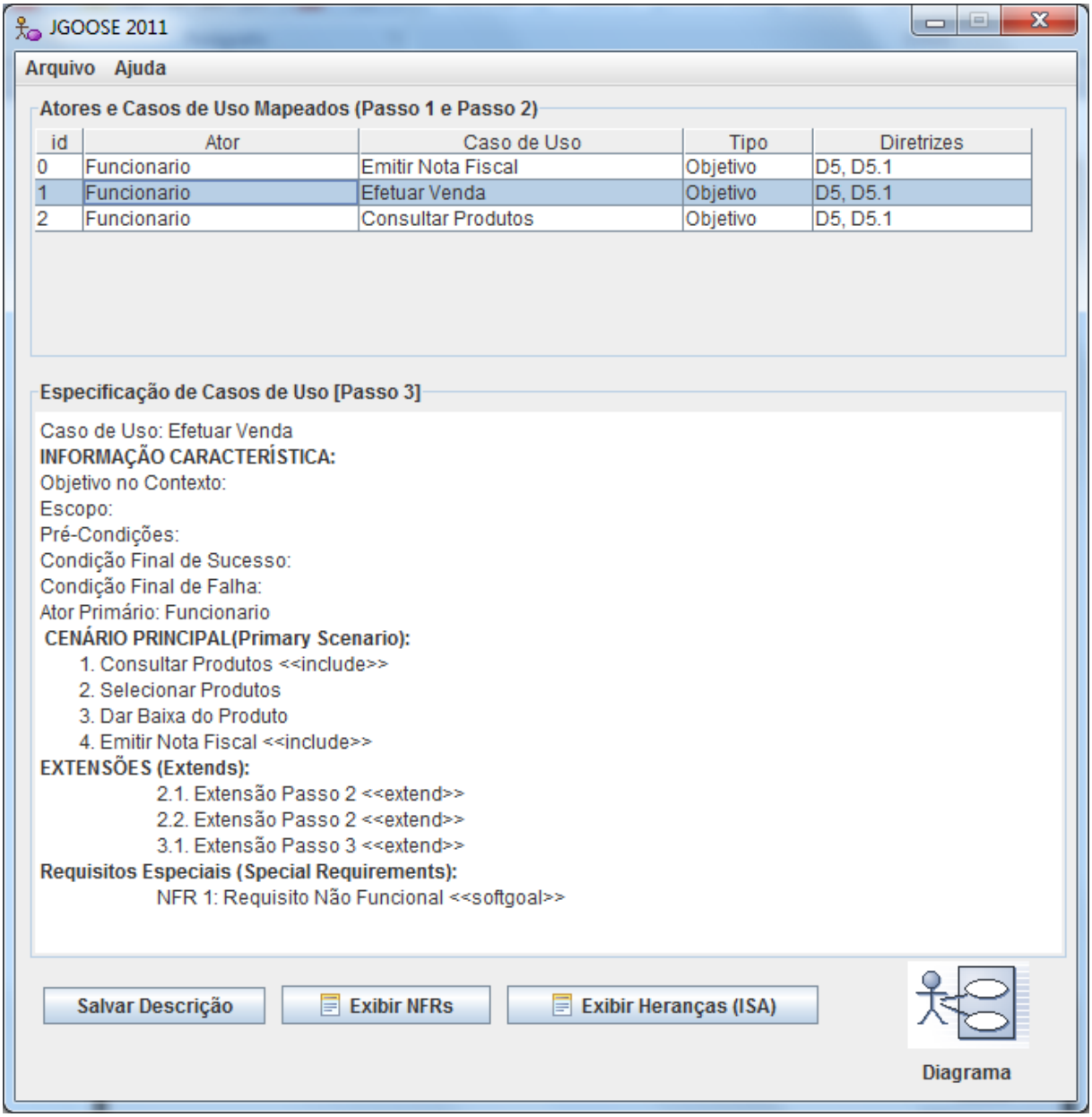

Figura 6.3:Casos de Uso Gerados pelo JGOOSE

Na Figura 6.4 é visto o diagrama de caso de uso para o caso de uso “Efetuar Venda”, gerado pela ferramenta e a opção de exportar no formato XMI.

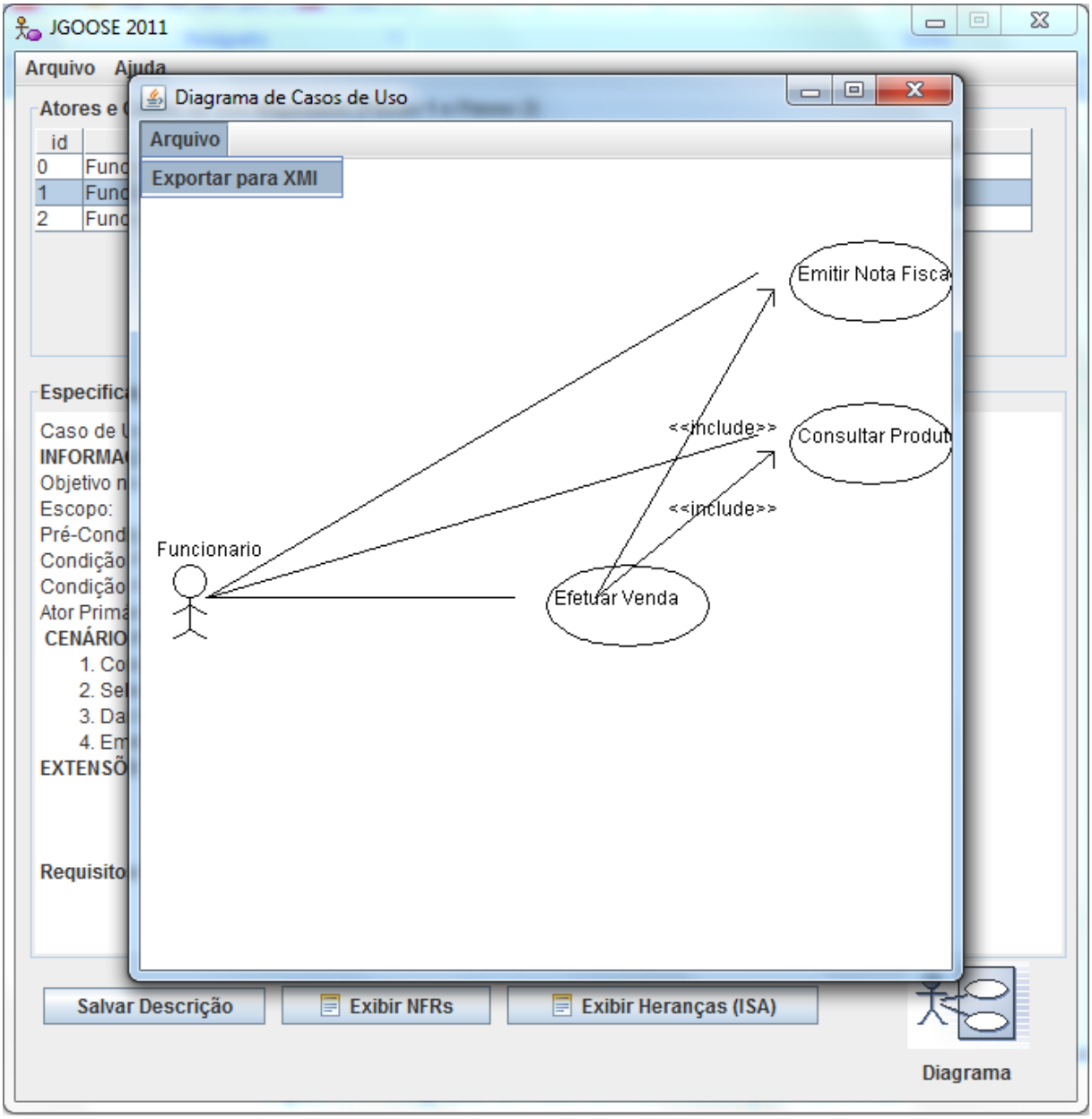

Figura 6.4: Diagrama de Caso de UsoGerado pelo JGOOSE

Após exportar no formato XMI o usuário da ferramenta pode utilizar seu diagrama em outras ferramentas que apóiam a engenharia de *software*através da opção disponibilizada pelas ferramentas de importação do XMI. Essa funcionalidade do JGOOSE foi testada utilizando a ferramenta StarUML [19], disponibilizada gratuitamente, e Enterprise Architect [20], ferramenta proprietária, contudo, após a exportação é necessário criar um documento de diagrama de caso de uso próprio da ferramenta e arrastar os elementos importados para a área de desenho, para que seja devidamente desenhado e manipulado.

Capítulo 7

Considerações Finais

Com a motivação e estudo dos processos da engenharia de requisitos, abordados nesse trabalho, foi possível implementar a proposta de melhoramento da ferramenta JGOOSE, que pode auxiliar os engenheiros de *software* nessa fase de grande importância da engenharia de *software*.

A conclusão da implementação das diretrizes de mapeamento visando à integração dos diagramas i* e Casos de Uso UML, gerando de forma automatizada os Casos de Uso a partir dos modelos criados no *framework* i*, permitiu a conclusão da ferramenta e o seu uso de forma completa, também permitindo a fácil obtenção dos casos de uso, como visto do estudo de caso e ainda integrando os resultados com outras ferramentas que apóiam a engenharia de *software*.

7.1 Trabalhos Futuros

Considera-se que existe a necessidade de aprimoramento da estrutura do código fonte, deixando a sua interface totalmente independente das implementações das funcionalidades. Ressalta-se que nesta versão da ferramenta, algumas mudanças neste sentido foram realizadas, mas é necessário evoluir para uma arquitetura mais coesa e que facilite futuras manutenções.

Na versão final deste documento será entregue a documentação da ferramenta desenvolvida, de acordo com o padrão UML. Para isso será feita uma descrição dos requisitos funcionais da ferramenta, através de descrições textuais via *template,* casos de uso em UML e requisitos não funcionais, por meio do *framework* NFR. Também será construído o diagrama de classe atualizado, diagrama de sequência, diagrama de subsistemas e diagrama de componentes. Dessa forma, teremos uma documentação da ferramenta que poderá ajudar em implementações futuras.

Referências Bibliográficas

[1] YU, E. ***i*: an agent- and goal-oriented modeling framework***. 2011. Consultado na Internet: http://www.cs.toronto.edu/km/istar/, em Março de 2011.

[2] YU, E. S. K. ***Towards modelling and reasoning support for early-phase requirements engineering***. In: 3RD IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING - RE97, 1997. **Proceedings...** Washington D.C., USA: [s.n.], 1997. p.226–235.

[3] SANTANDER, V. F. A. ***Integrando Modelagem Organizacional com Modelagem Funcional***. Pernambuco: Centro de Informática, Universidade Federal de Pernambuco, Dezembro, 2002. Tese de doutorado.

[4] VICENTE, A. ABE. ***JGOOSE: Uma ferramenta de Engenharia de Requisitos para a Integração da Modelagem Organizacional i* com a Modelagem Funcional de Casos de Uso UML***. Trabalho (Trabalho de conclusão de graduação) – Unioeste – Universidade Estadual do Oeste do Paraná, Cascavel – PR. Dezembro de 2006.

[5] YU, E. GIORGINI, P. MAIDEN, N.MYLOPOULOS, J. ***Social Modeling for Requirements Engineering***. 2011. The MIT Press (January 31, 2011).

[6] XMI. ***XML Metadata Interchange.*** 2011. Consultado na Internet: http://www.omg.org/spec/XMI/, em Outubro de 2011.

[7] SANTOS. B. S. ***IStar Tool - Uma proposta de ferramenta para modelagem de i****, Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, 2008.

[8] ***i* Wiki***. 2011. Consultadona Internet: http://istar.rwth-aachen.de/tiki-index.php?page=i*%20Wiki%20Home.

[9] BOOCH, G. RUMBAUGH, J. JACOBSON, I. ***UML: Guia do Usuário***, 2º Edição, Editora Camus, 2005.

[10] Rational Unified Process**.** ***Rational Unified Process,*** 2002.Consultado na Internet: http://www.wthreex.com/rup/portugues/index.htm

[11] COCKBURN, A. ***Writing Effective Use Cases***. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[12] ERIKSSON, H.-E.; PENKER, M. ***Business Modeling With UML: Business Patterns at Work***. New York, NY, USA: John Wiley Sons, Inc., 1998.

[13] BREITMAN, K.K., LEITE, J.C.S.P., ***A Framework for Scenario Evolution***, Third International Conference on Requirements Engineering – III, IEEE Computer Society Press. Los Alamitos, CA, U.S.A, (1998), pp 214 – 221.

[14] POTTS, C., ***ScenIC: A Strategy for Inquiry-Driven Requirements Determination***, In Proceedings of the Fourth IEEE International Symposium on Requirements Engineering - RE'99, June 7-11, Ireland, (1999).

[15] SUTCLIFFE, A., GREGORIADES, A., ***Validating Functional Requirements with Scenarios***, In: IEEE Joint International Requirements Engineering Conference, RE'02, University of Essen, Germany, September, 9-13, (2002), pp. 181-188.

[16] CASTRO, J., ALENCAR, F., SANTANDER, V., SILVA, C. ***Integration of i* and Object-Oriented Models***, 2011. The MIT Press (January 31, 2011).

[17] MYLOPOULOS, J., BORGIDA, A., JARKE, M., & KOUBARAKIS, M. ***Telos: Representing knowledge about information systems***. 1990. ACM Transactionon Information Systems, 8(4), 483-497.

[18] ***ErgoList***, 2011. Consultado na internet: http://www.labiutil.inf.ufsc.br/ergolist/, em Outubro de 2011.

[19] StarUML. ***StarUML - The Open Source UML/MDA Platform.*** Consultado na Internet: http://staruml.sourceforge.net/en/, em Outubro de 2011.

[20] Enterprise Architect. ***UML tools for software development and modelling – Enterprise Architect UML modeling tool.***Consultado na Internet: http://www.sparxsystems.com/, em Outubro de 2011.