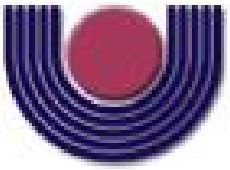

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Utilizando a Versão 2.0 de i* para Melhorar o Processo de Derivação de Casos de Uso

Bruno Luiz Casarotto

CASCAVEL
2019

Bruno Luiz Casarotto

Utilizando a Versão 2.0 de i* para Melhorar o Processo de Derivação de Casos de Uso

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 Arya Santander

CASCAVEL
2019

BRUNO LUIZ CASAROTTO

UTILIZANDO A VERSÃO 2.0 DE I* PARA MELHORAR O PROCESSO DE DERIVAÇÃO DE CASOS DE USO

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. Dr. Victor Francisco Arya Santander
(Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Ivonei Freitas da Silva
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Marco Antonio Toranzo Cespedes
Universidad Católica del Maule, Talca, Chile

Cascavel, 19 de dezembro de 2019

EPÍGRAFE

″Não é nossa função controlar todas as marés do mundo, mas sim fazer o que pudermos para socorrer os tempos em que estamos inseridos, erradicando o mal dos campos que conhecemos, para que aqueles que viverem depois tenham terra limpa para cultivar. Que tempo encontrarão não é nossa função determinar." *Gandalf, o Cinzento*

Lista de Figuras

Lista de Abreviaturas e Siglas

SD	Dependência Estratégica
SR	Razão Estratégica
GOOSE	Goal into Object Oriented Standard Extension
IDE	Integrated Development Environment
UML	Unified Modeling Language
UI	User Interface

Sumário

Resumo

O processo de derivação de casos de uso a partir de modelos organizacionais construídos via *framework* i* tem sido apresentado em trabalhos prévios. Este processo também é suportado pela ferramenta denominada JGOOSE. Contudo, as diretrizes utilizadas para derivar casos de uso utilizam como base os elementos da versão original de i*. Mais recentemente uma nova versão do framework i* tem sido proposta. Desta forma, é necessário revisar o processo de derivação considerando agora esta nova versão. Para alcançar este objetivo, primeiro foi necessário realizar um estudo sobre a versão original do i* para verificar quais elementos sofreram mudanças em relação a sua versão 2.0, para então realizar sua caracterização. Após isso, foi feito uma validação com um exemplo escolhido pelo autor, aplicando as diretrizes que foram modificadas de modo a incorporar os novos elementos do i* 2.0.

Palavras-chave: Engenharia de Requisitos, Istar, Diretrizes, Modificação

Capítulo 1

Introdução

1.1 Contexto

A Engenharia de Requisitos é a subárea da Engenharia de Software responsável por atividades que abrangem principalmente a etapa inicial do desenvolvimento de sistemas, a qual define os objetivos do software através da identificação de *stakeholders* (pessoas ou organizações afetadas pelo sistema e que, diretamente ou indiretamente, influenciam nos requisitos do software) (KOTONYA; SOMMERVILLE, 1998) (LAPOUCHNIAN, 2005). A Engenharia de Requisitos compreende as atividades de análise de domínio, elicitação, negociação, especificação, análise da especificação, documentação e evolução dos requisitos de um software (LAMSWEERDE, 2000). É uma das áreas mais críticas para o sucesso e qualidade de um projeto de sistema e sem ela o software resultante tem grandes possibilidades de não atender as necessidades dos *stakeholders* (KOTONYA; SOMMERVILLE, 1998) (PRESSMANN, 2006). Erros cometidos nessa etapa são mais difíceis de serem detectados e mais caros de serem corrigidos (BOEHM, 1981).

A Engenharia de Requisitos é geralmente vista como um processo contendo duas fases, a fase de requisitos iniciais (*early requirements phase*) e a fase de requisitos detalhados (*late requirements phase*). A fase de requisitos iniciais analisa e modela o ambiente a ser transformado em software, o contexto organizacional, os *stakeholders*, seus objetivos e seus relacionamentos. A fase de requisitos detalhados concentra-se na modelagem do sistema em conjunto com o ambiente, determinando e ajustando as fronteiras do sistema, sendo possível identificar requisitos do software (LAPOUCHNIAN, 2005). Os requisitos de software correspondem à descrições do que o sistema deve fazer, serviços que deve oferecer e restrições a seu funcionamento (SOM-

MERVILLE, 2011). Requisitos de software podem ser especificados de diversas maneiras, como através da abordagem orientada a objetivos (GORE - *Goal Oriented Requirements Engineering*). A GORE evidencia a motivação para o desenvolvimento do sistema, focando na expectativa do usuário em relação ao que o sistema deve fazer ou como ele deve se comportar (LAMSWEERDE, 2001) (LAPOUCHNIAN, 2005).

Uma das técnicas que representam essa abordagem, sendo uma das mais conhecidas e a qual será o foco desse trabalho, é a i* (lê-se istar), a qual propõe uma abordagem orientada a atores, focando nas intencionalidades, relacionamentos e motivações entre os membros da organização, possibilitando um melhor entendimento das relações organizacionais. Tipicamente, esta técnica é utilizada na fase de requisitos iniciais (*early requirements phase*) a qual tem como foco o entendimento inicial do contexto organizacional no qual o software pretendido está inserido.

Esta técnica também tem sido utilizada para apoiar o processo de obtenção de requisitos de software. Em (SANTANDER; CASTRO, 2002) propõe-se uma abordagem para derivar requisitos funcionais de sistemas computacionais via de casos de uso UML (YU et al., 2011) a partir de modelos i*. São propostas diretrizes as quais auxiliam engenheiros de requisitos a derivar Casos de Uso a partir da observação dos elementos presentes nos modelos i* os quais são categorizados em modelos de dependências estratégicas, denominados de SD (*Strategic Dependence*) e razões estratégicas, denominados de SR (*Strategic Rationale*).

O trabalho citado também possui suporte computacional via ferramenta denominada JGOOSE (*Java Goal Object into Oriented Standard Extension*), esta é uma ferramenta de auxílio no mapeamento de modelos organizacionais para modelos funcionais (VICENTE, 2006). Esta ferramenta implementa seus processos guiados pelas diretrizes propostas por (SANTANDER; CASTRO, 2002) e é com base nessas diretrizes que a ferramenta interpreta os modelos organizacionais do *framework* i* e gera os casos de uso UML, apresentando-os no *template* proposto por (COCKBURN, 2000). Desta maneira, a ferramenta permite derivar casos de uso com base nas intencionalidades associadas aos atores de um ambiente organizacional. Seu funcionamento, bem como uma apresentação geral da ferramenta serão detalhados no capítulo 2, seção 2.3.

Contudo, a proposta apresentada em (SANTANDER; CASTRO, 2002) adota a versão de construção de i* inicial proposta por (YU, 1995), tal versão do i* suportada pela ferramenta sofreu algumas mudanças que, ainda estão sendo propostas desde então. Estas mudanças rea-

lizadas pela comunidade científica vêm do uso desta técnica em vários domínios de aplicação. Assim, estudar estas mudanças na técnica i* pode beneficiar a derivação de casos de uso conforme proposto em (YU, 1995).

1.2 Motivação

Conforme mencionado na seção anterior, o i* tem incorporado várias mudanças sendo a última versão apresentada na versão i* 2.0 proposta por (DALPIAZ; FRANCH; HORKOFF, 2016). Assim, é necessário estudar e avaliar as mudanças que a versão 2.0 sofreu em relação a original, e depois decidir se a proposta de (SANTANDER; CASTRO, 2002) pode ser melhorada observando tais diferenças do i* e sua nova versão.

Desta forma, neste trabalho motiva-nos o fato de estudar a fundo a versão 2.0 da técnica i* com foco na melhoria das diretrizes propostas em (SANTANDER; CASTRO, 2002).

1.3 Objetivos

O objetivo geral do trabalho é descrever novos elementos da técnica i* que possam melhorar a elaboração de casos de uso, modificando ou não as diretrizes, proposto por (SANTANDER; CASTRO, 2002).

Como objetivos específicos podemos citar:

- Descrever os elementos propostos na versão 2.0 de i* (Capítulos 3 e 4);
- Descrever mudanças e/ou melhorias nas diretrizes apresentadas em (SANTANDER; CASTRO, 2002) (Capítulo 4);
- Aplicar as diretrizes a um exemplo escolhido pelo autor (Capítulo 5);
- Apontar os benefícios advindos da proposta de derivação de casos de uso considerando os novos elementos de i* incorporados (Capítulo 5);

1.4 Contribuições

Entre as contribuições esperadas podemos destacar:

- Apresentar um estudo de elementos da versão i* 2.0 considerando como estes elementos poderiam ser usados para derivar requisitos funcionais na forma de casos de uso;
- Melhorar a proposta apresentada em (SANTANDER; CASTRO, 2002) incorporando novos elementos de i* às diretrizes usadas para derivar casos de uso;
- Com um exemplo, demonstrar como as diretrizes modificadas de (SANTANDER; CASTRO, 2002) podem facilitar ou melhorar a descoberta de casos de uso a partir de modelos i*.
- Discutir as modificações necessárias da ferramenta JGOOSE de forma a incorporar as mudanças realizadas nas diretrizes da proposta de (SANTANDER; CASTRO, 2002).

1.5 Estrutura do Trabalho

Este trabalho será dividido em sete capítulos, sendo eles os seguintes:

- Capítulo 1 - Introdução
- Capítulo 2 - Referencial Teórico
- Capítulo 3 - Versão 2.0 do *Framework* i*
- Capítulo 4 - Avaliação da Versão 2.0 Focando na Melhoria da Proposta de Derivação de Casos de Uso
- Capítulo 5 - Validação da Proposta e Discussão sobre as Mudanças Necessárias na Ferramenta JGOOSE para Acomodar as Diretrizes Modificadas
- Capítulo 6 - Considerações Finais e Trabalhos Futuros.

Capítulo 2

Referencial Teórico

Neste capítulo será feito uma introdução do *framework* i* na seção 2.1, uma explicação do que são casos de uso na 2.2 e como os mesmos podem ser derivados a partir do i* na 2.3, e o que é a ferramenta JGOOSE na seção 2.4.

2.1 Framework i*

O *framework* i* originalmente foi proposto por (YU, 1995) para modelar e raciocinar sobre ambientes organizacionais e seus sistemas de formação. Ele consiste de dois modelos principais, o de dependência estratégica (SD), o qual descreve a relação de dependência dos atores em um contexto organizacional, e o de razão estratégica (SR), o qual descreve os interesses e preocupações dos *stakeholders* e como eles podem ser influenciados por diversas configurações do sistema e pelo ambiente. Nas subseções 2.1.1 e 2.1.2 são apresentados os conceitos de cada um dos modelos.

2.1.1 Modelo SD

O Modelo SD é composto por nós e ligações. Os nós representam os atores no ambiente e as ligações são as dependências entre os atores. Por ator entende-se uma entidade que realiza ações para obter objetivos no contexto do ambiente organizacional. Atores dependem uns dos outros para atingir objetivos, realizar tarefas e obter recursos no ambiente organizacional. O ator que depende de alguma forma de outro ator é chamado de *Depender* e o ator que atende e satisfaz o *Depender* é denominado de *Dependee*. O objeto ou elemento de dependência entre *Depender* e *Dependee* é denominado de *Dependum*. Portanto, haverá relacionamentos do tipo

Depender -> Dependum -> Dependee (YU et al., 2011). Um exemplo de um modelo SD pode ser visto na Figura 2.1. Esse exemplo nos mostra a relação do cliente com o funcionário e consequentemente com o sistema, para realizar uma compra. Primeiramente, o cliente deve ter o objetivo de realização da compra, após isso o funcionário deve consultar o produto, ou efetuar a venda, ou emitir uma nota fiscal, sendo esses três objetivos dependentes do sistema, e o mesmo possui como um *softgoal* a rapidez para sua utilização. Depois o funcionário deve receber os dados de pagamento do cliente, e a compra estará concluída.

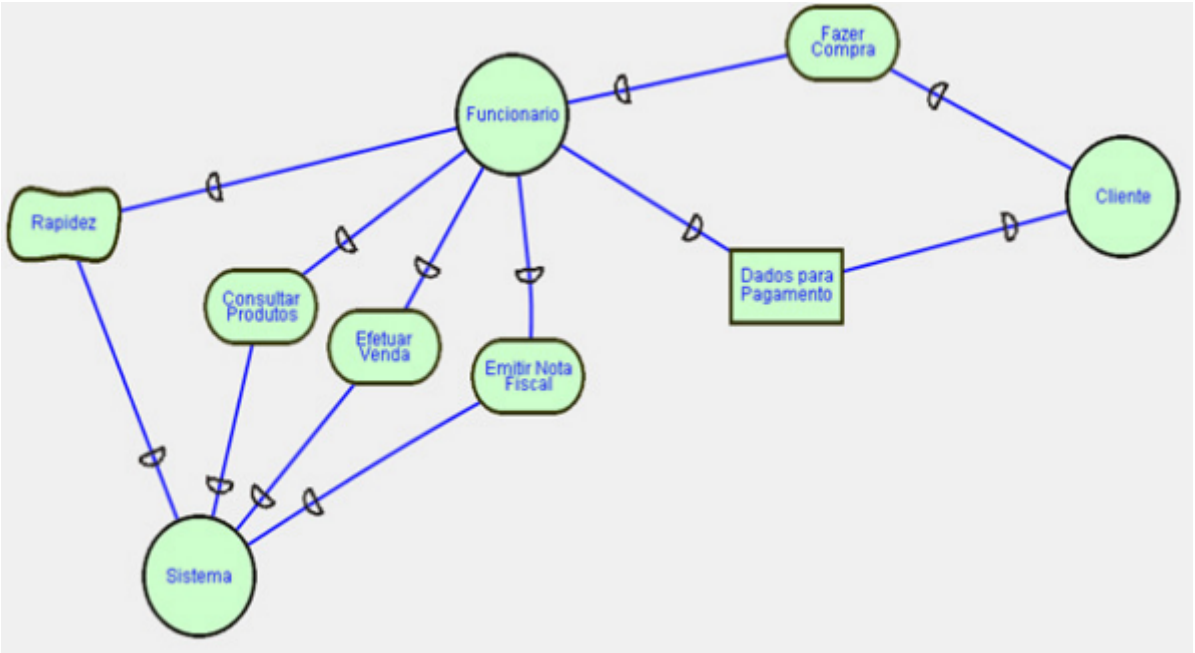

Figura 2.1: Exemplo de modelo SD, adaptado de (BRISCHKE; SANTANDER; SILVA, 2012)

2.1.2 Modelo SR

O modelo de SR é complementar ao modelo de SD. O SR permite compreender e modelar de forma mais detalhada as razões associadas com cada ator e suas dependências (YU et al., 2011). Enquanto o modelo de SD provê um nível de abstração, no qual modelasse somente os relacionamentos externos entre atores, o modelo de SR permite uma maior compreensão a respeito das razões estratégicas de atores em relação aos processos da organização e como os mesmos são expressos. O modelo de SR auxilia no processo de Engenharia de Requisitos permitindo que elementos de processos e as razões por detrás dos mesmos sejam expressos. Um exemplo de um modelo SR pode ser visto na Figura 2.2. Esse exemplo é basicamente o

mesmo que o do modelo da Figura 2.1, no entanto, há uma especificação maior para as *Tasks* de "Consultar Produtos", "Efetuar Venda"e "Emitir Nota Fiscal". Por exemplo, para efetuar a venda, ela precisa ser decomposta em 4 *subtasks* e todas elas precisam ser satisfeitas, a primeira é a consulta de produto, a qual deve ser satisfeita por dois *goals*, o de busca de produto e o de mostra de produtos, os mesmos são dependentes de apenas uma de duas *subtasks* associadas a eles. Então, após satisfazer a primeira *subtask*, todas as outras três poderão ser satisfeitas de forma análoga. Por fim, com todas as *subtasks* satisfeitas, o *goal* de efetuar venda é completo.

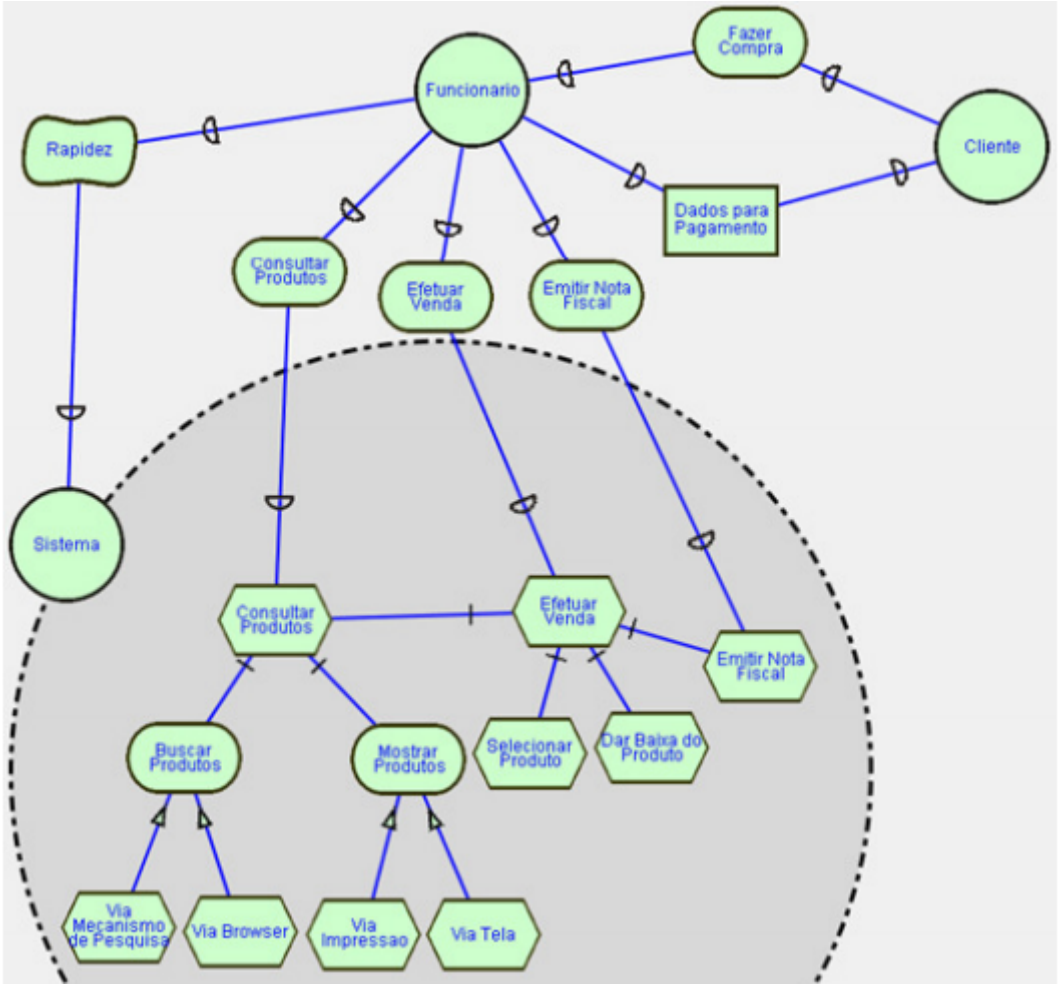

Figura 2.2: Exemplo de modelo SR, adaptado de (BRISCHKE; SANTANDER; SILVA, 2012)

Na Engenharia de Requisitos o modelo de SR pode ser utilizado para compreender como sistemas estão relacionados/envolvidos em rotinas de atores da organização para gerar alternativas e para modelar e suportar o raciocínio de atores organizacionais a respeito destas alternativas.

2.2 Casos de Uso

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 como os propostos pelo *framework* i*. No entanto, ainda é de grande importância que haja uma ampla compreensão do sistema e como transformar essa representação em implementação do sistema. A UML (Unified Modeling Language) surge no contexto de desenvolvimento orientado a objetos e auxilia nessa transformação dentre outras utilidades.

A UML pode ser descrita como "uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela pode ser empregada para a visualização, especificação, construção e documentação de artefatos que façam uso de sistemas complexos de software" (BOOCH; RUMBAUGH; JACOBSON, 2005). 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 os quais se destaca os casos de uso. Os casos de uso propiciam o entendimento do funcionamento do sistema a todas as partes, programadores, especialistas e usuários finais, sendo que também contribuem na validação do sistema enquanto é desenvolvido (SANTANDER; CASTRO, 2002).

Casos de Uso em UML (BOOCH; RUMBAUGH; JACOBSON, 2005) são utilizados para capturar o comportamento desejado do sistema a ser desenvolvido, sem ter de especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para os desenvolvedores chegar a um entendimento comum com os usuários finais do sistema e especialistas de domínio. Além disso, casos de uso servem para ajudar a validar a sua arquitetura e para verificar o sistema à medida que o mesmo evolui durante o desenvolvimento (BOOCH; RUMBAUGH; JACOBSON, 2005).

Um caso de uso envolve uma situação de utilização do sistema por um ator, o qual representa qualquer elemento externo que interage com o sistema. Um caso de uso pode gerar vários cenários. Cenários estão para casos de uso assim como instâncias estão para classes. Isso significa que um cenário é basicamente uma instância de um caso de uso. Nesta situação, vários caminhos podem ser seguidos dependendo do contexto na execução do sistema.

Estes caminhos são os possíveis cenários do caso de uso. Considera-se que o caminho

básico para realizar um caso de uso, sem problemas e sem erros em nenhum dos passos da sequência, é denominado de cenário primário. Neste tipo de cenário, a execução dos passos para realizar a funcionalidade básica do caso de uso é obtida com sucesso. Por outro lado, caminhos alternativos bem como situações de erro podem ser representados através de cenários secundários. Cenários secundários descrevem sequências alternativas e de erros que podem ocorrer em um cenário primário associado com um caso de uso. Cenários secundários podem ser descritos separadamente ou como extensão da descrição de um cenário primário. Se um cenário secundário é bastante complexo e inclui um conjunto considerável de passos, é conveniente descrevê-lo separadamente.

Outras técnicas também podem ser usadas na Linguagem de Modelagem Unificada para refinar fluxos de eventos em Casos de Uso. A ideia consiste basicamente em incluir relacionamentos que permitam descrever diversos aspectos de comportamento entre casos de uso. Os relacionamentos apontados na UML incluem:

- «include»: quando for detectado no sistema um conjunto de passos comuns aos vários casos de uso, pode-se criar um caso de uso com estes passos com potencial para ser reutilizado por outros casos de uso. A ideia consiste em abstrair em um caso de uso específico, um comportamento comum aos vários casos de uso, estabelecendo que os demais casos de uso do sistema podem fazer uso do mesmo (incluí-lo) quando necessário;

- «extend»: utiliza-se este tipo de relacionamento quando existe uma sequência opcional ou condicional de passos que queremos incluir em um caso de uso. Esta sequência de passos deve ser descrita em um caso de uso específico que poderá ser utilizado por outros casos de uso em certo ponto de sua execução;

- «generalization»: generalização entre casos de uso tem o mesmo significado de generalização entre classes na orientação a objetos. Isto significa que um caso de uso "filho" herda o comportamento e estrutura do caso de uso "pai". Considera-se que um caso de uso "filho" é uma especialização do caso de uso "pai", podendo adicionar nova estrutura e comportamento bem como modificar o comportamento do caso de uso "pai". Este relacionamento também pode ser utilizados entre atores de casos de uso.

Um resumo dos elementos gráficos utilizados na UML para representar Casos de Uso são

apresentados na Figura 2.3.

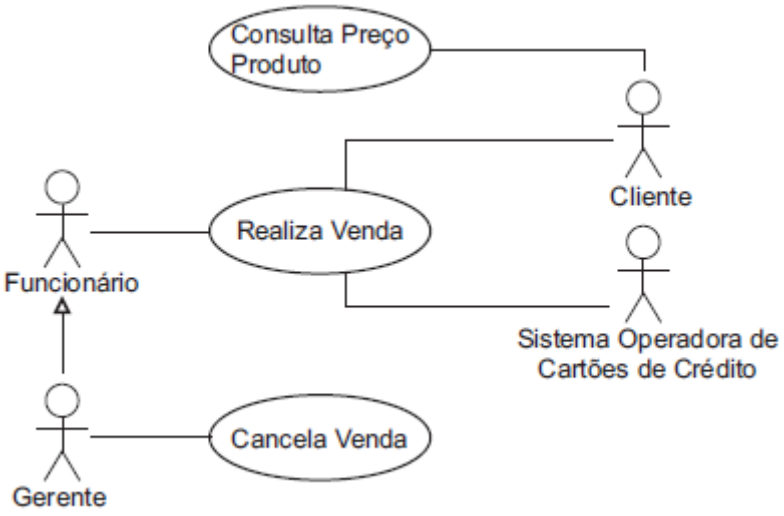

Figura 2.3: Exemplo de diagrama de caso de uso em UML

Adicionalmente, casos de uso também podem ser descritos de forma textual (COCKBURN, 2000). O *template* de especificação de caso de uso proposto por (COCKBURN, 2000) define explicitamente objetivos de casos de uso bem como os níveis associados com estes objetivos. As demais informações presentes no *template* também são importantes para tornar a descrição textual de casos de uso o mais claro possível. Salienta-se que foi adotado este *template* pela ferramenta JGOOSE como modelo para a especificação de casos de uso no processo derivação de casos de uso a partir de modelos organizacionais. Este *template* é descrito a seguir na Figura 2.4.

Casos de Uso podem ser uma parte do documento de requisitos que deve ser desenvolvido no processo de engenharia de requisitos e representam basicamente aspectos funcionais e comportamentais do sistema a ser desenvolvido. É consensual que casos de uso não são suficientes para detalhar todos os elementos que devem ser definidos no processo de engenharia de requisitos. No entanto, as vantagens do uso desta técnica, como também de outras técnicas baseadas em cenários, é que podemos juntamente com as descrições de interações entre um usuário e o sistema, relacionar outros tipos de requisitos tais como requisitos não funcionais e organizacionais e evoluir posteriormente para outros artefatos do processo de desenvolvimento. Neste contexto, também cabe destacar que os casos de uso fazem parte de diversas metodologias de desenvolvimento de software.

Number	*Unique use case number*	
Name	*Brief noun-verb phrase*	
Summary	*Brief summary of use case major actions*	
Priority	*1-5 (1 = lowest priority, 5 = highest priority)*	
Preconditions	*What needs to be true before use case "executes"*	
Postconditions	*What will be true after the use case successfully "executes"*	
Primary Actor(s)	*Primary actor name(s)*	
Secondary Actor(s)	*Secondary actor name(s)*	
Trigger	*The action that causes this use case to begin*	
Main Scenario	**Step**	**Action**
	Step #	*This is the "main success scenario" or "happy path."*
	…	*Description of steps in successful use case* "execution"
	…	*This should be in a "system-user-system, etc." format.*
Extensions	**Step**	**Branching Action**
	Step #	*Alternative paths that the use case may take*
Open Issues	*Issue #*	*Issues regarding the use case that need resolution*

Figura 2.4: Exemplo do *template* de caso de uso proposto por (COCKBURN, 2000)

2.3 Derivação de Casos de Uso a partir de i*

Atualmente na engenharia de requisitos está sendo mais comum a ideia de que a fase de especificação de requisitos tenha informações relacionadas à organização, modelos de negócios e outras informações além das especificações do *software* (BRISCHKE; SANTANDER; SILVA, 2012). 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 (SANTANDER; CASTRO, 2002).

Para tal, existe uma possível alternativa para aproximar modelos organizacionais de modelos funcionais, a qual é proposta em (SANTANDER; CASTRO, 2002). Usando esta proposta, podemos elicitar e documentar casos de uso em UML a partir de modelos organizacionais construídos utilizando o *framework* i* (YU et al., 2011). Para este fim, são propostas algumas diretrizes que são brevemente descritas a seguir e aplicadas ao exemplo apresentado nas Figuras 2.1 e 2.2.

A Figura 2.5 resume os passos desta proposta e apresenta uma visão geral do mapeamento para casos de uso a partir de i*.

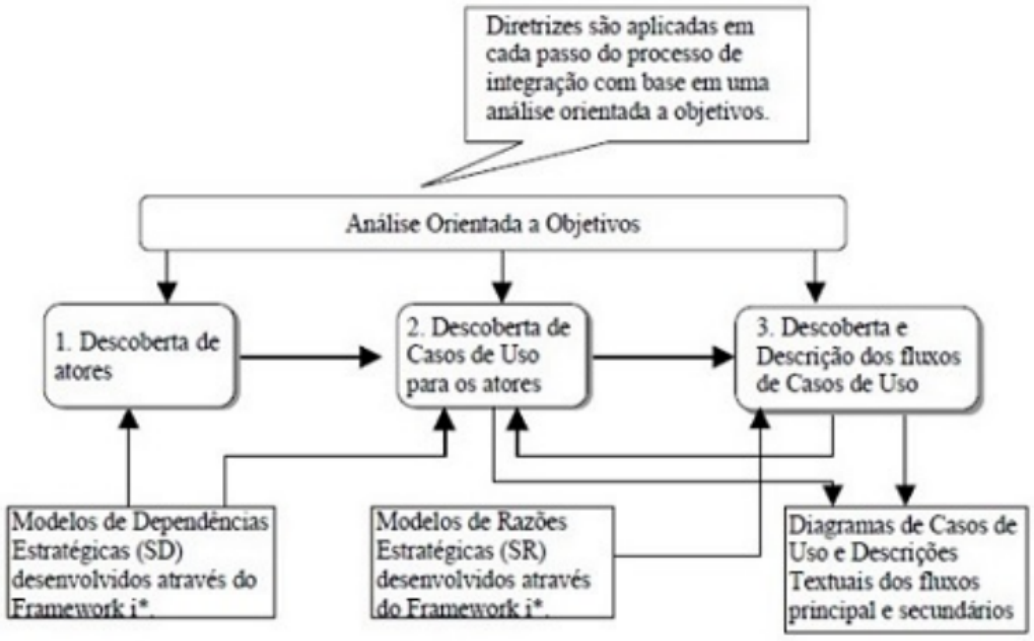

Figura 2.5: Visão geral do mapeamento para casos de uso a partir de i*, adaptado de (SANTANDER; CASTRO, 2002)

Primeiro Passo 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 2.1 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 várias ligações de dependência com o sistema;

Diretriz 4: atores em i* relacionados através do mecanismo *IS-A*, 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 2.1 um ator

"Gerente" que tivesse uma relação *IS-A* com o ator "Funcionário", ambos seriam candidatos a atores em casos de uso e no diagrama seriam relacionados via mecanismo de «generalização».

Segundo Passo 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.

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.

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 subtarefas. Este tipo de tarefa pode ser mapeada para caso de uso.

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, o mesmo será candidato a ser um objetivo de um Caso de Uso para este ator.

SubDiretriz 5.4: deve-se avaliar as dependências do tipo *softgoal* associadas com o ator. Tipicamente uma dependência do tipo *softgoal* em i* é um requisito não-funcional associado ao sistema pretendido. Por exemplo, o *softgoal* "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 as dependências do tipo objetivo "Consultar Produtos", "Efetuar Venda" e "Emitir Nota Fiscal" em relação ao sistema, e estas dependências geram casos de uso do sistema para o ator "Funcionário".

Diretriz 7: classificar cada caso de uso de acordo com seu tipo de objetivo associado (objetivo contextual, objetivo de usuário, objetivo de subfunção). A nova versão da ferramenta JGOOSE permite ao Engenheiro de Requisitos preencher e salvar esta informação.

Terceiro Passo 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 SR 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 subcomponentes em uma ligação de decomposição de tarefa mapeando-os para passos na descrição do cenário primário (fluxo principal) de casos de uso. Também devemos analisar ligações do tipo meio-fim mapeando os meios para passos alternativos na descrição de casos de uso. Por exemplo, as tarefas "Consultar Produtos", "Selecionar Produto", "Dar Baixa do Produto" e "Emitir Nota Fiscal" na Figura 2.2, que decompõem a tarefa "Efetuar Venda" já mapeada para caso de uso, fazem parte do cenário principal deste caso de uso (ver Figura 2.7). Também podemos observar que a tarefa "Consultar Produtos" é decomposta nos objetivos "Buscar Produtos" e "Mostrar Produtos", os quais geram os passos do cenário principal para este caso de uso. Observando mais atentamente a Figura 2.2, verifica-se que há dois meios "Via *Browser*" e "Via Mecanismo de Pesquisa" para satisfazer o objetivo "Buscar Produtos". Um destes meios fará parte do cenário principal e o outro será mapeado para um passo no cenário secundário representando uma alternativa a execução 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. Por exemplo, observando a Figura 2.2, o engenheiro de requisitos poderia considerar que a tarefa "Dar baixa do Produto", mesmo não sendo decomposta no modelo SR, necessita ser refinada em vários passos. Esta análise levaria a definição de um novo caso de uso. A nova versão da ferramenta permite realizar esta mudança bem como salvá-la.

Diretriz 10: Desenvolver o diagrama de casos de uso utilizando os casos de uso descobertos e os relacionamentos do tipo «include», «extend» e «generalization» usados para estruturar as especificações dos casos de uso. Por exemplo, observando as Figuras 2.1 e 2.2 das seções 2.1 e 2.2, e aplicando as diretrizes dos passos 1 e 2, é possível mapear os atores e casos de uso para o sistema em questão (ver Figura 2.6). Também, conforme exemplificado na diretriz 8, é possível perceber que os passos associados a "Consultar Produtos" e "Emitir Nota Fiscal são descritos textualmente (ver Figura 2.7) e representados graficamente (ver Figura 2.6) utilizando o estereotipo «include». O mesmo ocorre no cenário secundário onde um determinado passo estende «extend» um caso de uso ou quando atores relacionados via ligação *IS-A* no modelo SD

são ligados no diagrama de casos de uso via estereotipo «generalization».

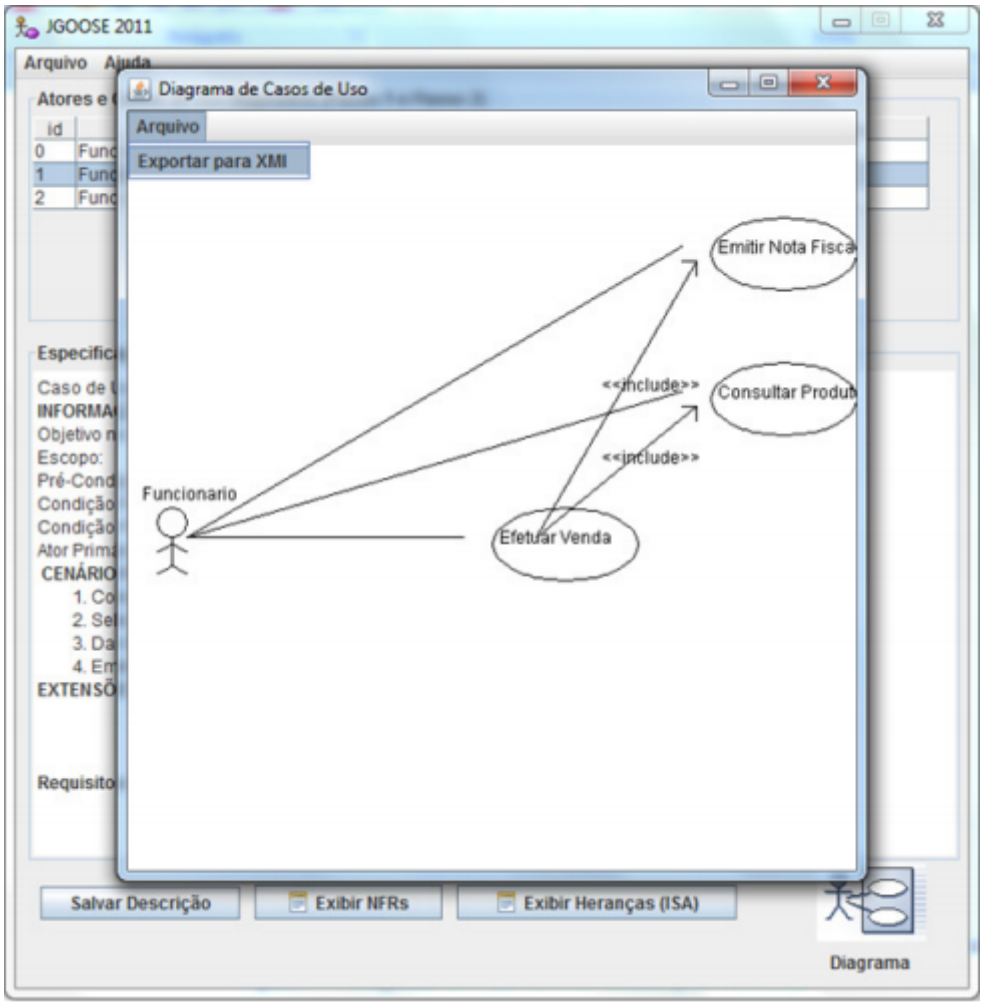

Figura 2.6: Representação gráfica de caso de uso UML

2.4 Ferramenta JGOOSE

A JGOOSE é uma ferramenta de auxílio para o mapeamento de modelos organizacionais para modelos funcionais (VICENTE, 2006). Atualmente a ferramenta permite a elaboração de modelos organizacionais do framework i*, através do E4J i*, a criação de Diagramas de Casos de Uso e a obtenção automática de Casos de Uso a partir de modelos i* (MERLIN, 2013). Os Casos de Uso obtidos são apresentados através de Diagrama de Casos de Uso UML e descrições textuais, utilizando uma versão baseada no template proposto por (COCKBURN, 2000). Com essa ferramenta, é possível derivar casos de uso com base nas intencionalidades associadas aos atores de um ambiente organizacional.

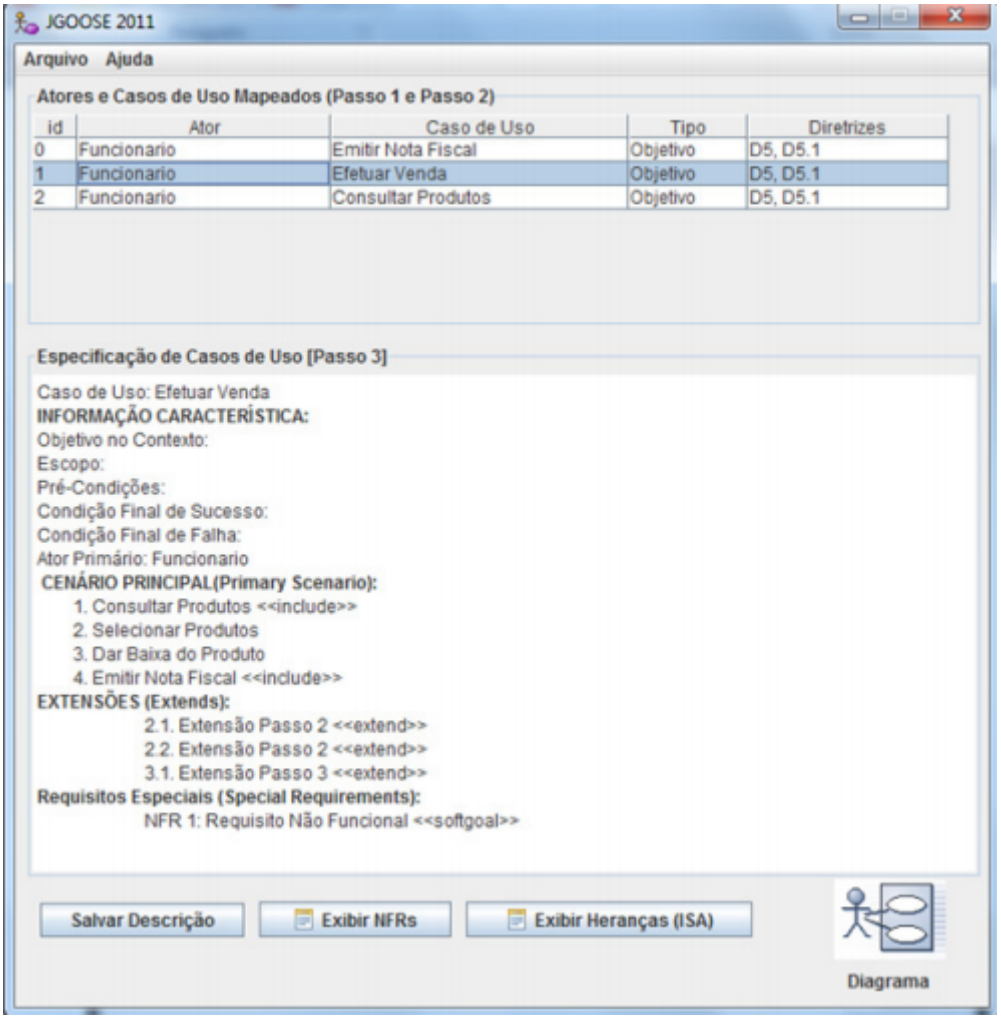

Figura 2.7: Representação textual de caso de uso de acordo com (COCKBURN, 2000)

A primeira versão da ferramenta foi desenvolvida por (VICENTE, 2006), com a reimplementação, utilizando Java, da ferramenta GOOSE (Goal into Object Oriented Standard Extension), a qual foi implementada na linguagem *Rational Rose Scripting* (BRISCHKE, 2005). Ao longo dos anos a ferramenta passou por várias melhorias e aprimoramentos, realizados por acadêmicos vinculados ao Laboratório de Engenharia de Software (LES) da Universidade Estadual do Oeste do Paraná (UNIOESTE), através de projetos de iniciação científica e de Trabalhos de Conclusão de Curso. Dentre as mudanças realizadas estão a refatoração do código-fonte, correção de *bugs*, implementação de novas funcionalidades e alterações na interface gráfica do usuário em (BRISCHKE, 2005) e em (BRISCHKE; SANTANDER; SILVA, 2012).

A partir da implementação do editor gráfico de Diagrama de Casos de Uso, denominado E4J Use Cases, possibilitou-se a criação e manipulação de Diagrama de Casos de Uso diretamente

no ambiente de trabalho proposto pela JGOOSE em (VICENTE, 2006) e em (BRISCHKE; SANTANDER; SILVA, 2012). Para a implementação do E4J Use Cases adotou-se a JGraphX, uma biblioteca Java para visualização de grafos, a qual consiste de um conjunto de estruturas e funcionalidades que possibilitam a criação de aplicações interativas voltadas para a manipulação de diagramas (ALDER, 2002).

Capítulo 3

Versão i* 2.0

Neste capítulo serão apresentadas as diferenças entre o i* original proposto por (YU, 1995) e a sua versão 2.0, na seção 3.1 e em todas as suas subseções.

3.1 Versão 2.0

A versão 2.0 foi proposta por (DALPIAZ; FRANCH; HORKOFF, 2016) e possui algumas diferenças em relação a sua versão original proposta por (YU, 1995), as quais serão detalhadas nas subseções a seguir.

3.1.1 Atores

Os atores possuem dois papéis na versão 2.0, o de *Agent*, um ator com manifestações físicas concretas, como um ser humano, uma organização ou um departamento, ou o de *Role*, uma caracterização abstrata do comportamento de um ator social dentro de algum contexto especializado ou domínio. Sempre que o tipo de ator é irrelevante, seja por causa do cenário em questão, a noção de ator genérico pode ser utilizado no modelo, essa noção é a mesma que em sua versão original.

A diferença notada da versão proposta por (YU, 1995) e por (DALPIAZ; FRANCH; HORKOFF, 2016) é que na anterior havia mais uma especialização de ator, ela era chamada de *Position*, especialização a qual era uma abstração intermediária entre *Agent* e *Role*, no i* 2.0 essa noção foi retirada e incorporada na relação de atores. Seguem as representações de atores na versão 2.0 mostradas na Figura 3.1. Nesta figura há três atores os quais estão presentes em um cenário de reembolso de viagens em uma universidade.

Figura 3.1: Ator genérico, *Role* e *Agent* em (DALPIAZ; FRANCH; HORKOFF, 2016)

3.1.2 Relações

Atores geralmente estão relacionados entre si, e na versão 2.0 isso não é diferente, no entanto, diferentemente da proposta de (YU, 1995), dois tipos diferentes de relacionamentos entre atores podem ser definidos:

- *is-a*: representa o conceito de generalização/especialização no iStar 2.0. Somente as funções podem ser especializadas em funções, bem como apenas atores genéricos podem ser especializados em atores genéricos. Por exemplo: um doutorando (*role*) pode ser uma especialidade de estudante (*role*). Os agentes não podem ser especializados via *is-a*, pois são instâncias concretas.

- *participates-in*: representa algum tipo de associação, além da generalização/especialização, entre dois atores, retirando assim a nomenclatura de *is-part-of*, *plays*, *occupies*, *covers* que existe no i* original de (YU, 1995). Não há restrição quanto ao tipo de atores ligados por esta associação.

Como pode ser observado na Figura 3.2, os *links* de associações de atores são representados usando setas no diagrama. A ponta da flecha identifica o alvo e deve possuir um rótulo identificando o tipo de *link* associado.

3.1.3 Intenções

Todos os atores possuem algumas intenções no sistema, como tal, eles modelam diferentes tipos de requisitos para o diagrama. Uma intenção que aparece dentro do limite de um ator (isto ocorre no modelo SR - ver seção 2.1.2) denota algo que é desejado por esse ator. Ela também pode aparecer fora dos limites do ator (isto ocorre no modelo SD - ver seção 2.1.1), como parte de um relacionamento de dependência entre dois atores.

Para o primeiro caso podemos citar os seguintes elementos:

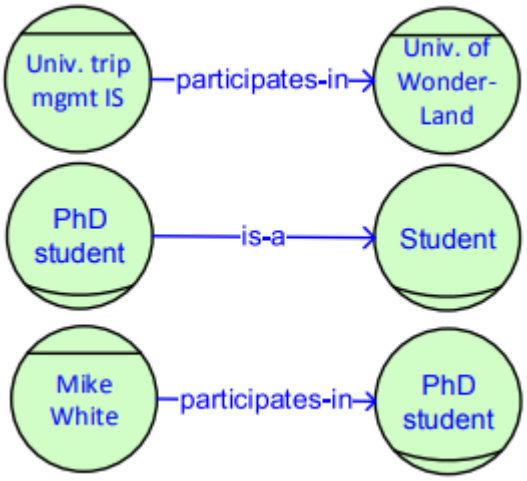

Figura 3.2: *Links* de associação entre atores em (DALPIAZ; FRANCH; HORKOFF, 2016)

- *Goal*: um estado do sistema que o ator deseja alcançar e que possui requisitos claros para completá-lo.
- *Quality*: um atributo para o qual um ator deseja algum nível de realização. Este elemento é basicamente o *Softgoal* da versão original, a justificativa proposta por (DALPIAZ; FRANCH; HORKOFF, 2016) é que a noção de métrica para atingir *Goals* deve ser substituída pela noção do quão bem tal *Goal* é alcançado, por isso a noção de qualidade.
- *Task*: representa ações que um ator deseja executar, geralmente com o objetivo de atingir algum *Goal*.
- *Resource*: uma entidade física ou informativa que o ator exige para executar uma tarefa.

Para o segundo caso, os elementos de *Dependum*, *Depender* e *Dependee*, os quais estão associados em uma relação *Depender* -> *Dependum* -> *Dependee*, são representados da mesma maneira que para a versão original, no entanto, a versão 2.0 adicionou mais dois elementos: o *DependerElmt* e o *DependeeElmt*. O primeiro é a intenção dentro do limite do ator que mostra onde a dependência começa, o que explica o porquê ela existe. O segundo é a intenção que explica como o *Dependee* pretende fornecer o *Dependum*. Uma apresentação gráfica dos elementos do primeiro caso pode ser observada na Figura 3.3, e do segundo caso pode ser observada na Figura 3.4.

Cabe ressaltar que os elementos *DependerElmt* e *DependeeElmt* são opcionais e ambos

podem ser omitidos. Esta opção é usada quando é criado um modelo inicial de SR, ou para expressar conhecimento parcial do problema, ou seja, quando o por quê (*DependerElmt*) ou como (*DependeeElmt*) são desconhecidos. Se ambos estão omitidos, há um *link* entre o ator *Depender* e o ator *Dependee* através do *Dependum*. Também é possível especificar apenas um dos dois.

Figura 3.3: Intenções de atores em (DALPIAZ; FRANCH; HORKOFF, 2016)

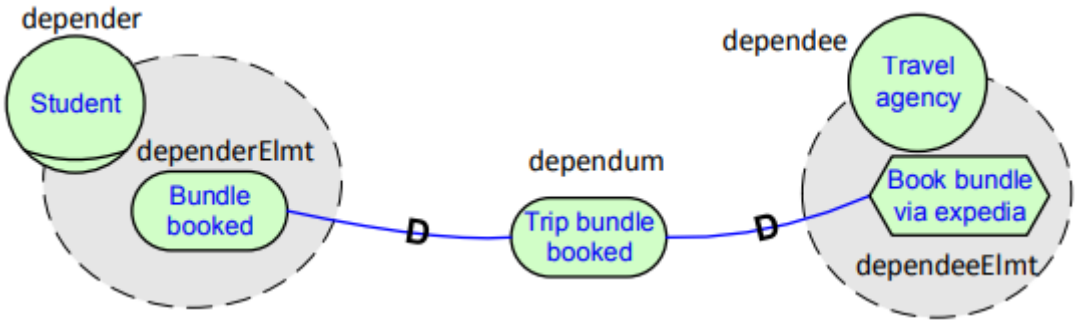

Figura 3.4: Dependência entre atores em (DALPIAZ; FRANCH; HORKOFF, 2016)

3.1.4 *Links* de intenções

Há quatro tipos de *links* de intenções: refinamento, necessário em, contribuição e qualificação. Eles serão descritos a seguir e estão resumidos na Figura 3.5.

		Apontado para	Apontado para	Apontado para	Apontado para
		Goal	*Quality*	*Task*	*Resource*
Começa de	*Goal*	Refinamento	Contribuição	Refinamento	-
Começa de	*Quality*	Qualificação	Contribuição	Qualificação	Qualificação
Começa de	*Task*	Refinamento	Contribuição	Refinamento	-
Começa de	*Resource*	-	Contribuição	Necessário em	-

Figura 3.5: *Links* entre intenções

a) Refinamento

Para promover a facilidade de adoção, o iStar 2.0 apresenta um relacionamento genérico chamado "refinamento" que vincula metas e tarefas hierarquicamente. O "refinamento"é uma relação n-ária que relaciona um dos pais a um ou mais filhos. Um elemento intencional pode ser o pai em no máximo um relacionamento de refinamento.

Existem dois tipos de refinamento, os quais se aplicam a qualquer tipo de pai (*goal* ou *task*), e os mesmos são definidos a partir do operador lógico que relaciona o pai com os filhos:

- *AND*: o cumprimento de todas os n filhos faz com que os pais sejam cumpridos;
- *OR* Inclusivo: o cumprimento de pelo menos um filho faz com que o pai seja cumprido.

Um pai pode apenas assumir um dos dois tipos de refinamento, nunca ambos, e dependendo dos elementos que estão conectados, o refinamento possui diferentes significados:

- Se o pai é um *Goal*:

- No caso de AND, um *Goal* filho é um subestado de que faz parte do *Goal* pai, enquanto uma *Task* filha é uma sub-tarefa que deve ser preenchida, um exemplo pode ser visto no item a) da Figura 3.6, onde o pai é um *Goal* e possui dois filhos *Goals* que devem ser satisfeitos;
- No caso de OR, uma *Task* filha é uma forma particular (um “meio”) de cumprir o *Goal* pai (o "fim"), enquanto um *Goal* filho é um sub-objetivo que pode ser alcançado para cumprir o *Goal* do pai, um exemplo pode ser visto no item b) da Figura 3.6, onde o pai é um *Goal* e ele possui uma *Task* filha chamada *book bundle*, que serve como um "meio"para fazer com que o *Goal* pai seja satisfeito;

- Se o pai é uma *Task*:

- No caso de AND, uma *Task* filha é uma sub-tarefa que é identificada como parte da *Task* pai, enquanto um *Goal* filho é um objetivo que é descoberto por meio da análise da *Task* pai, um exemplo pode ser visto no item d) da Figura 3.6;
- No caso de OR, um *Goal* filho é uma meta cuja existência é descoberta analisando a *Task* pai que pode substituir a *Task* original, enquanto uma *Task* filha é uma maneira de executar a *Task* pai.

As relações de refinamento não implicam um processo estritamente *top-down*, elas podem ser construídas *top-down*, *bottom-up* ou por meio de uma abordagem mista.

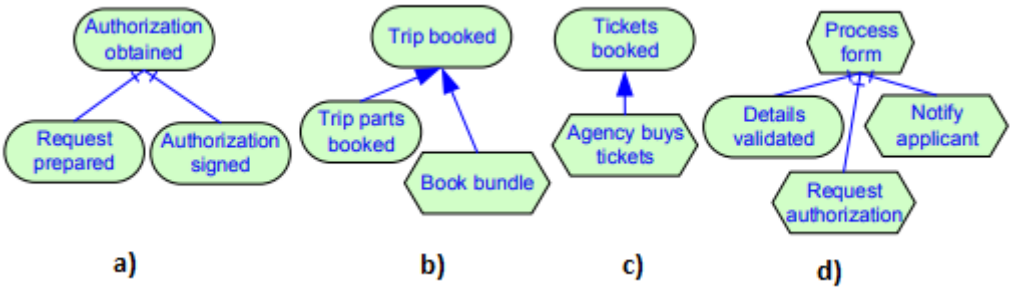

Figura 3.6: Exemplo de "refinamento" adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

b) Necessário em

O relacionamento "necessário em" vincula uma tarefa a um recurso e indica que o ator precisa do recurso para executar a tarefa. Este relacionamento não especifica qual é a razão para esta necessidade, e graficamente é representado como uma seta com uma ponta de seta circular direcionada para a tarefa, como mostrado na Figura 3.7. Neste exemplo podemos perceber que existe um relacionamento que indica que o recurso cartão de crédito é necessário para realizar a tarefa de realizar o pagamento de ingresso.

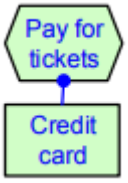

Figura 3.7: Exemplo de "necessário em" adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

c) Contribuição

Os *links* de "contribuição" representam os efeitos dos elementos intencionais nas *qualities*, ver seção 3.1.3, e são essenciais para auxiliar os analistas no processo de tomada de decisão entre objetivos ou tarefas alternativas. As contribuições são definidas como relacionamentos de um elemento intencional de origem para uma qualidade de destino e pode ser um dos seguintes tipos:

- *Make*: A fonte fornece contribuição positiva suficiente para a satisfação do alvo.
- *Help*: A fonte fornece um pouco de contribuição positiva para a satisfação do alvo.
- *Hurt*: A fonte fornece contribuição fraca contra a satisfação (ou negação) do alvo.
- *Break*: A fonte fornece contribuição suficiente contra a satisfação (ou negação) do alvo.

Esses 4 *links* de "contribuição"estão presentes na versão original proposta por (YU, 1995), portanto não é uma diferença. No entanto, a proposta original possui mais cinco *links*, sendo eles: *And*, *Or*, *Some+*, *Some-* e *Unknown*, os quais foram retirados na versão 2.0.

As contribuições são representadas graficamente como setas sólidas e com um *label* que indica o tipo de contribuição, conforme mostrado na Figura 3.8.

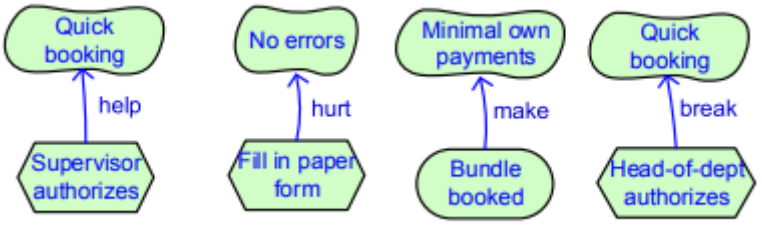

Figura 3.8: Exemplo de "contribuição" adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

d) Qualificação

O *link* de intenção de "qualificação" relaciona uma *quality* ao elemento ligado a ela: uma *task*, *goal* ou *resource*. Colocar uma relação de qualificação expressa uma *quality* desejada sobre a execução de uma *task*, a realização de um *goal* ou a provisão do *resource*. A relação de qualificação é representada graficamente por meio de um linha pontilhada conectando o elemento que está sendo qualificado. Um exemplo pode ser visto na Figura 3.9.

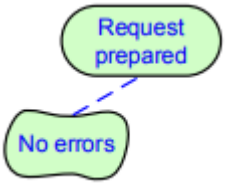

Figura 3.9: Exemplo de "qualificação" adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

3.1.5 Visões do modelo

Além das visões de SR e SD, demonstrados nas seções 2.1.2 e 2.1.1 respectivamente, é proposto também uma visão híbrida entre os dois modelos, é adotado quando alguns dos atores estão abertos, mas não todos, concentrando-se na lógica estratégica de um conjunto particular de atores, e os *links* dos atores estão escondidos. O exemplo demonstrado na Figura 3.10 nos mostra o cenário geral de um reembolso de viagem, onde existem três atores, cada um com uma especialização, o estudante sendo ele um *Actor*, a agência de viagem sendo ela um ator genérico, e o sistema de organização de viagem universitário sendo ele uma *Role*.

Outras visões úteis podem ser definidas conforme necessário, por exemplo, a visão do ator, mostrando apenas atores e *links* de atores, ou uma visão funcional, escondendo todas as *qualities*, *links* de contribuição e restrição.

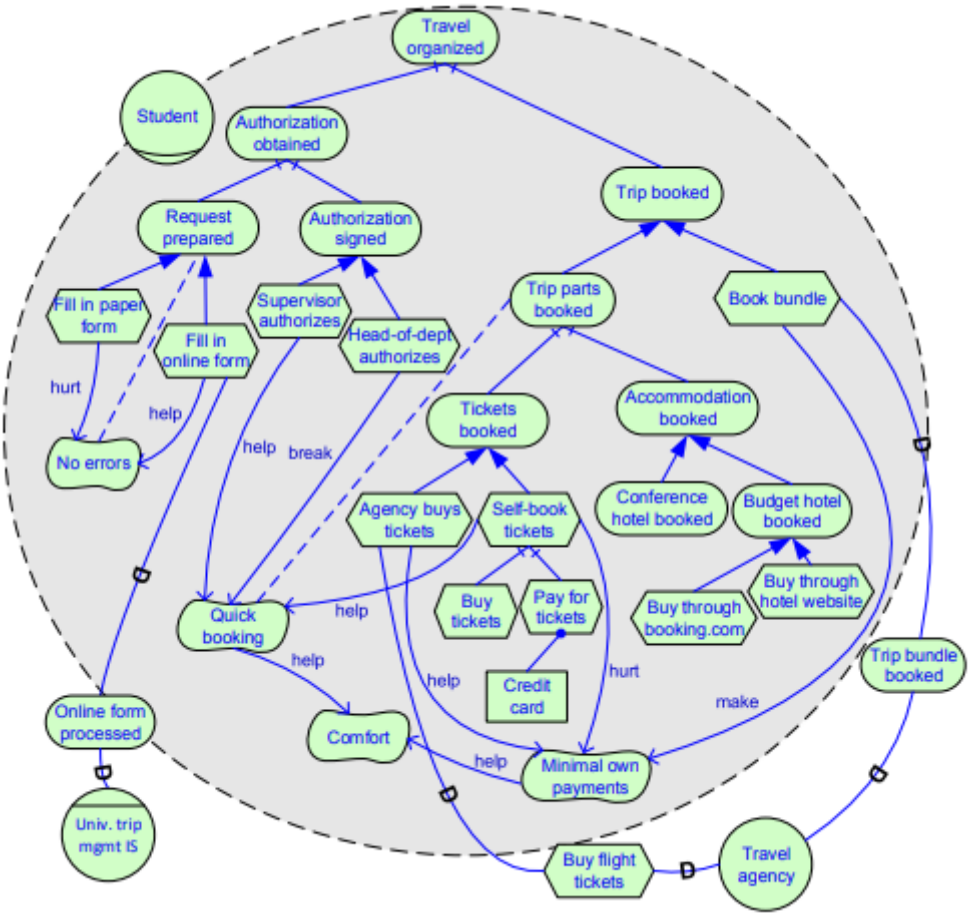

Figura 3.10: Exemplo da visão híbrida, usando o cenário de reembolso de viagem, adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

3.1.6 Resumo das diferenças

Assim, observando as seções anteriores, podemos destacar as diferenças da versão original de i* proposta por (YU, 1995) em relação à versão 2.0, conforme segue:

- Atores são agora representados de 3 formas diferentes, a genérica, a de *Agent* e a de *Role*, ou seja, a mesma representação que o original, no entanto foi retirado a representação de *Position*;

- Além da relação *IS-A*, também possui a relação de *participates-in*, a qual representa qualquer tipo de associação, além da generalização/especialização entre dois atores, sendo retirado a noção de *is-part-of*, *plays*, *occupies* e *covers* proposto por (YU, 1995) na versão original do i*;

- As intenções são quase iguais as do i* original, a única diferença é a mudança de nomenclatura do *softgoal* para *quality*;

- *Links* de intenções no i* original diferem bastante do 2.0, pois foram criados dois *links*, o de *needed by* e *qualification*, e os *links* antigos de *means-end* e *task decomposition* foram agrupados em apenas um chamado *refinement*, o único que ficou igual ao original foi o de *contribution*;

- Uma visão nova, além da visão SD e SR é apresentada, sendo ela a visão híbrida, que engloba elementos de ambas as visões anteriores e é dependente da função procurada pelo autor.

Capítulo 4

Avaliação da Versão 2.0 Focando na Melhoria da Proposta de Derivação de Casos de Uso

Na seção 3.1, e em suas subseções, foram descritas as características da versão 2.0 do i* com foco na diferenciação em relação à versão original proposta em (YU, 1995). Neste capítulo, as mudanças da versão 2.0 i* são analisadas visando averiguar o impacto das mesmas no processo de derivação de casos de de uso a partir de modelos i*, apresentado na seção 2.3.

4.1 Caracterizando Elementos da Versão 2.0 i* na Proposta de Derivação

4.1.1 Modificação 1

- Antes na versão original: Atores possuem, além da representação genérica, as representações *Agent*, *Role* e *Position*.

- Depois na versão 2.0: Retirado a noção de *Position*, pois a mesma foi avaliada por (DALPIAZ; FRANCH; HORKOFF, 2016) como uma abstração entre os atores *Agent* e *Role*, o que causava confusão em alguns casos.

- Impacto nas diretrizes: Observando as diretrizes específicas associadas à derivação de atores em casos de uso a partir de modelos i* (primeiro passo) é possível verificar que na proposta inicial somente considera-se o ator genérico (*Actor*) no processo de derivação. Na nova versão do i*, esta categoria de ator não sofreu modificação no que tange à sua

representação diagramática nem tampouco do ponto de vista conceitual. Desta forma, a mudança da representação desse elemento não afeta as diretrizes do passo 1 da referida proposta. No entanto, como foi considerado apenas o ator genérico, é válido realizar uma verificação para ver se a introdução dos conceitos de atores *Role* e *Agent* terá algum impacto em alguma diretriz do primeiro passado. O estudo inicial demonstrou que não há impacto, pois o conceito de ator genérico é apenas uma generalização de ambos os conceitos de atores, portanto, em determinadas situações o uso do *Role* será necessário, enquanto em outras o uso do *Agent* será. Desta forma, as diretrizes do primeiro passo da proposta serão alteradas para deixar explícito que os atores nos modelos i* analisados podem ser do tipo *Agent*, *Role* e *Genérico*. Cabe ressaltar que a versão atual da JGOOSE já considera essas categorias de atores no processo de mapeamento. Entretanto, a ferramenta será modificada para retirar a opção de escolha da categoria de ator *Position*, a qual foi retirada na versão i* 2.0.

4.1.2 Modificação 2

- Antes na versão original: Existem algumas relações entre dois atores, são elas: *is-a*, *is-part-of*, *plays*, *occupies*, *covers*.

- Depois na versão 2.0: Foram retiradas as noções de *is-part-of*, *plays*, *occupies*, *covers* e foram agregadas em *participates-in*, sendo que a mesma representa qualquer tipo de associação, que não seja representável pela relação *is-a*. Já a associação de especialização/generalização modelada através da relação *is-a* entre dois atores, não sofreu alteração.

- Impacto nas diretrizes: Observando a diretriz 4 do primeiro passo da proposta, a qual trata da associação que representa generalização/especialização, podemos verificar que a mesma não sofrerá alteração. Contudo, esta diretriz não considera as demais associações já mencionadas, as quais foram, na versão i* 2.0, agregadas na associação *participates-in*. Desta forma, cabe realizar uma análise sobre como a inclusão desta associação poderia melhorar o processo de derivação de casos de uso. Para avaliar este aspecto, podemos observar um exemplo na Figura 4.1. Neste exemplo podemos identificar que o *Agent* Mike White está relacionado com o ator PhD Student do tipo *Role*. Isto significa que Mike é um agente que assume o *role* de PhD Student assim como outros estudantes poderiam

assumir este papel. Em casos de uso (BOOCH; RUMBAUGH; JACOBSON, 2005), um ator representa tipicamente quem interage com o sistema computacional assumindo um *role* neste contexto. Desta forma, não é visto nenhuma vantagem e viabilidade em termos de representação diagramática de casos de uso de agentes que estão relacionados a atores representando *roles* via relação *participates-in*. Como percebido na explicação, tal inclusão não é válida ser implementada.

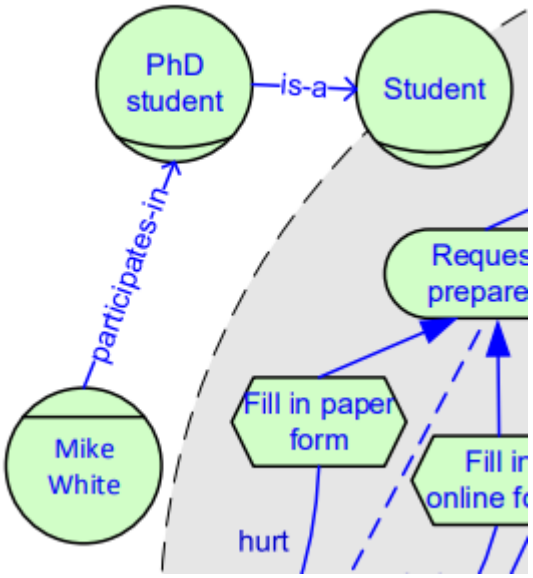

Figura 4.1: Exemplo para análise de *participates-in* adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

4.1.3 Modificação 3

- Antes na versão original: Um dos elementos intencionais que um ator pode possuir é chamado de *softgoal*, o qual representa um objetivo flexível que o ator deseja satisfazer. Este objetivo flexível denominado de *softgoal*, na engenharia de requisitos é mapeada para um requisito não funcional.

- Depois na versão 2.0: Para retirar a noção de objetivo do *softgoal*, o mesmo foi renomeado para *quality* para fazer com que sua interpretação se transformasse em uma qualidade que deve ser fornecida cuja satisfação ainda é subjetiva.

- Impacto nas diretrizes: A subdiretriz 5.4 da proposta indica que um *sofgoal* é mapeado para um requisito não funcional do sistema pretendido. Neste sentido, a mudança de

nomenclatura desta dependência para *quality* afeta esta diretriz apenas com a mudança do nome da dependência utilizado.

4.1.4 Modificação 4

- Antes na versão original: Existem três tipos de *links* intencionais entre dependências (*goal*, *resource*, *softgoal*, *task*) os quais são: *means-end*, *task-decomposition* e *contribution*.

- Depois na versão 2.0: Na nova versão, os *links* de *means-end* e *task-decomposition* foram englobados em um só chamado de *refinement*. O *link* intencional *contribution* não sofreu modificação, no entanto, há a adição de mais dois *links* intencionais, *qualification* e *neededBy*, sendo que o primeiro representa um *link* entre dependência do tipo objetivo ou tarefa e uma *quality*, e o segundo representa um *link* entre uma dependência do tipo objetivo ou tarefa e um *resource*.

- Impacto nas diretrizes: Observando as diretrizes específicas e esses aspectos, conseguimos identificar que haverá um impacto na diretriz 8, visto que a mesma analisa no modelo SR, ligações do tipo *means-end* para mapear os meios para passos alternativos de casos de uso. Também há uma análise no modelo SR dos subcomponentes de ligações de *task-decomposition* para mapear o cenário primário de casos de uso. Contudo, o impacto é apenas na nomenclatura utilizada na diretriz e não no processo utilizado já que semanticamente ambas ligações continuam com o mesmo significado. Por outro lado, é necessário analisar de forma mais específica as ligações do tipo *contribution*, *qualification* e *neededBy*.

  Em relação à ligação *qualification*, é possível notar no exemplo de modelo SR da Figura 4.4, que a ligação entre a *quality No Errors* e o *goal Request Prepared* permite associar uma qualidade desejada a um objetivo definido. No processo de derivação de casos de uso a partir de i*, se um objetivo interno ao ator que representa o sistema for mapeado para um caso de uso, a qualidade associada a esse objetivo será agora um requisito de qualidade na descrição textual desse caso de uso. O campo *open issues* na descrição textual proposto por (COCKBURN, 2000) será usado para esse fim. Vejamos o exemplo da Figura 4.2

para explicar o contexto dessa mudança.

Figura 4.2: Exemplo para explicação do *qualification*

Neste exemplo, podemos perceber que a *task Data Base Quering* possui uma influência negativa em relação a *quality Performance*, pois se uma *query* que será executada em um banco de dados não for ótima, ela vai ferir um pouco a performance do sistema. Para a descrição textual de caso de uso (ver Figura 2.4) seria escrito no campo *open issues*, do caso de uso pertinente, algo a respeito dessa influência negativa, na descrição textual de acordo com o *template* de (COCKBURN, 2000) ficaria desse jeito, para este exemplo, no campo de *Open Issues*: 1: *Data Base Quering hurt Performance*. No processo de análise

de caso de uso, isto significa dizer que: *Data Base Quering* possui um impacto negativo na performance do sistema, logo deve-se cuidar com as *queries* que serão executadas. Isto pode auxiliar na de descrever como a implementação deve ser feita.

Por outro lado, o *goal Seek Products* teria uma linha tracejada que conectaria ele e a *quality Performance*, caracterizando assim, uma *qualification* do *goal*, significando assim, uma qualidade desejada, mas não necessária, e a mesma seria apresentada na descrição textual de caso de uso, dentro do campo *open issues*.

Já para a ligação *neededBy* permite associar um recurso que é obrigatório para a realização de uma tarefa associada a um objetivo ou até mesmo outra tarefa, sua representação gráfica é uma linha que parte do recurso até sua tarefa ou objetivo associado, com uma circunferência completamente pintada na ponta, como pode ser visto no exemplo da Figura 4.3. Pode-se observar neste exemplo que o recurso associado a uma tarefa ou objetivo será acrescido à descrição textual do passo no cenário principal ou nas extensões, ver Figura 5.2 no Capitulo 5.

E finalmente, em relação à ligação *contribution*, podemos perceber que há várias diferenças entre o i* original e o 2.0, como por exemplo a retirada de algumas ligações, mas a conservação de outras, conforme visto na seção 3.1.4. No entanto, as nomenclaturas, bem como suas representações diagramáticas, não sofreram quaisquer mudanças, como pode ser visto no exemplo da Figura 4.4.

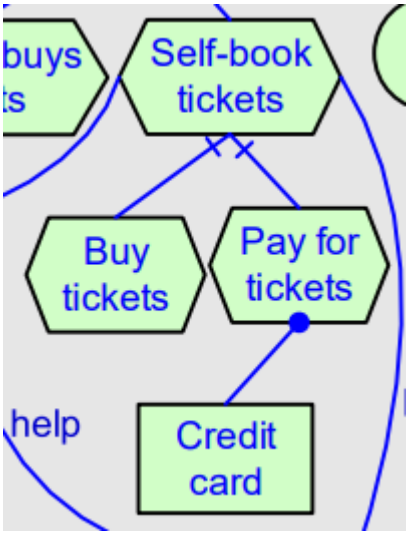

Figura 4.3: Exemplo de *neededBy* adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

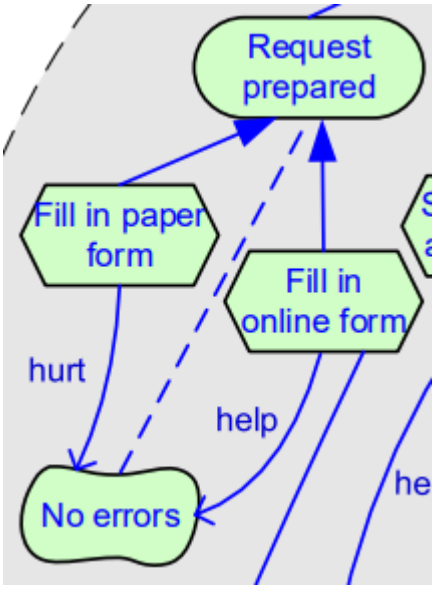

Figura 4.4: Exemplo de *contribution* e de *qualification* adaptado de (DALPIAZ; FRANCH; HORKOFF, 2016)

4.1.5 Modificação 5

- Antes na versão original: As visões suportadas pelo i* original proposto por (YU, 1995) eram apenas as visões SD e SR.

- Depois na versão 2.0: Há a proposta de uma nova visão, chamada de híbrida, a qual mistura os elementos de ambas as visões anteriores, e ela é dependente da necessidade percebida pelo autor.

- Impacto nas diretrizes: Para tal diferença, foi identificado a diretriz 8, no entanto, não há como modificá-la, pois ela utiliza especificamente o modelo SR para efetuar a extração de informações que possam conduzir à descrição de fluxos principais e alternativas de casos de uso que são representados no diagrama de caso de uso, e como a representação híbrida permite utilizar tanto elementos de SD quanto elementos de SR, tal diretriz não precisará ser modificada.

  Contudo, é importante salientar que a diretriz 8 considera que o modelo SR pode conter alguns atores refinados e outros não, sendo que isto impacta diretamente na qualidade da descrição textual dos casos de uso gerados. Desta forma, esta diretriz será modificada para esclarecer ao engenheiro de requisitos que um modelo híbrido no qual um dos atores

não seja refinado, impactará na geração da descrição textual dos casos de uso.

Também foi possível na análise desta modificação, detectar algumas falhas na implementação da ferramenta, em virtude da falta de clareza das diretrizes propostas. Mais especificamente, exemplificado no cenário representado nas Figuras 4.5 e 4.6. Nos exemplos das figuras anteriores podemos perceber que ao relacionar uma *task* interna de um ator com um *goal* (as dependências internas de um ator sempre aparecem dentro de sua *boundary*), essa relação não é apresentada como um caso de uso, no entanto, como demonstrado nas Figuras 4.7 e 4.8, se existir uma relação direta entre o *goal* e o ator, tal relação é apresentada como um caso de uso.

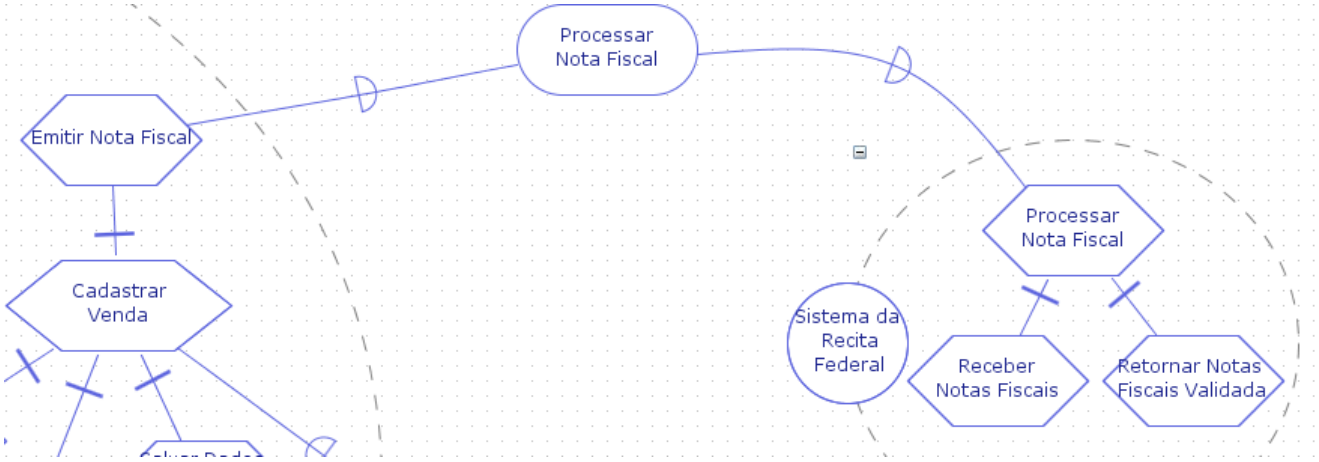

Figura 4.5: Exemplo diagramático do problema de mapeamento de caso de uso

4.2 Proposta de Diretrizes Modificadas

Com as modificações demonstradas, explicadas e exemplificadas na seção anterior, podemos agora propôr as mudanças necessárias para as diretrizes propostas por (SANTANDER; CASTRO, 2002).

Primeiro Passo da Proposta - Descoberta de Atores

Diretriz 1: todo ator em i* deve ser analisado para um possível mapeamento para ator em caso de uso, podendo ele ser do tipo *Role*, *Agent* ou Genérico.

Diretriz 2: inicialmente, deve-se analisar se o ator, o qual é do tipo *Role*, *Agent* ou Genérico, 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.

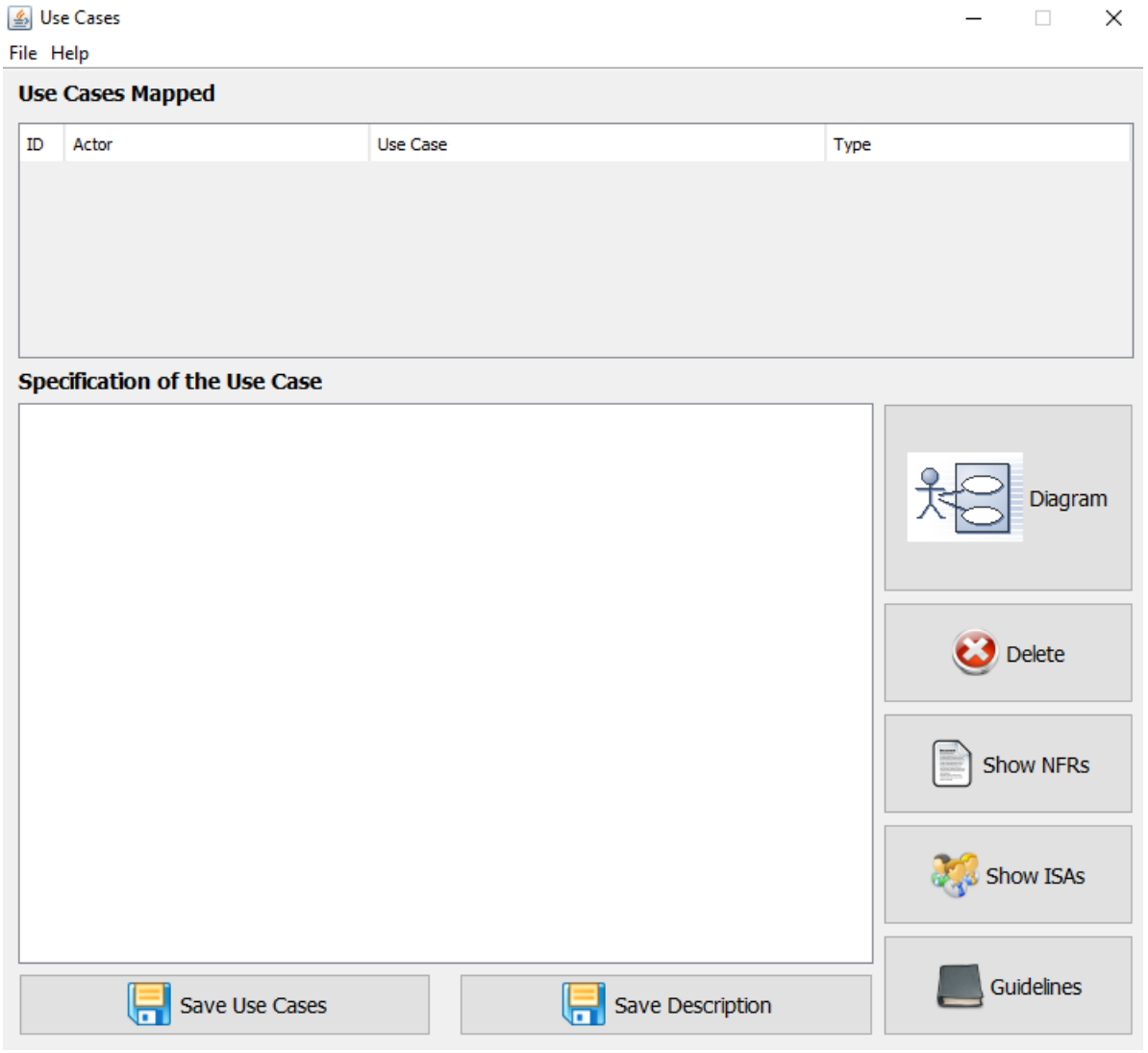

Figura 4.6: Caso de uso não mapeado

Diretriz 3: o ator i* candidato deve ter pelo menos uma dependência com o sistema computacional pretendido.

Diretriz 4: atores em i* relacionados através do mecanismo *IS-A*, 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».

Segundo Passo 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

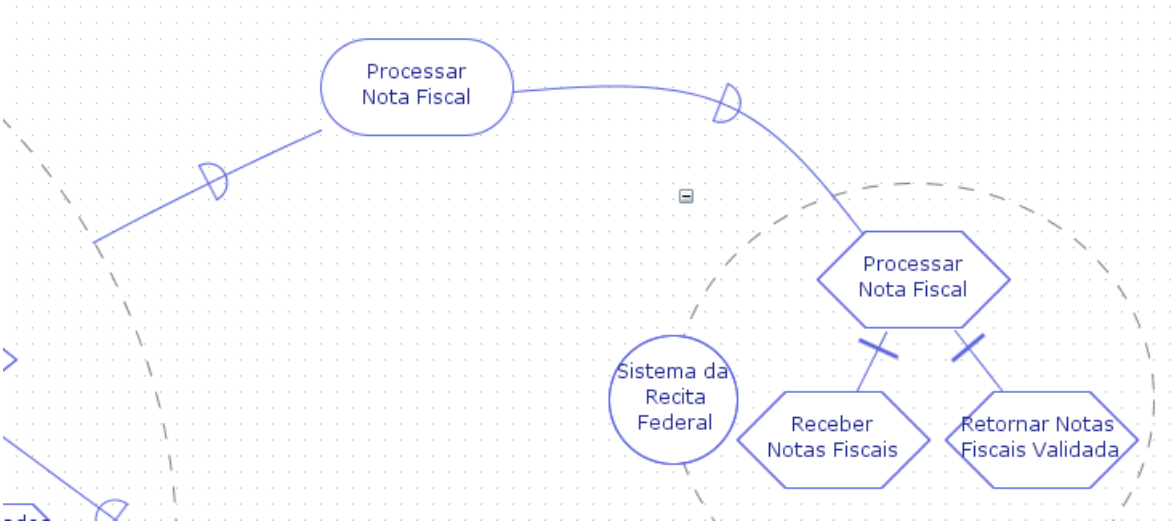

Figura 4.7: Exemplo diagramático de sucesso ao realizar o mapeamento de caso de uso

o sistema computacional pretendido (*depender*), visando descobrir casos de uso para o ator.

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.

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 subtarefas. Este tipo de tarefa pode ser mapeada para caso de uso.

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, o mesmo será candidato a ser um objetivo de um Caso de Uso para este ator.

SubDiretriz 5.4: deve-se avaliar as dependências do tipo *quality* associadas com o ator. Tipicamente uma dependência do tipo *quality* em i* é um requisito não-funcional associado ao sistema pretendido.

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).

Diretriz 7: classificar cada caso de uso de acordo com seu tipo de objetivo associado (objetivo contextual, objetivo de usuário, objetivo de subfunção).

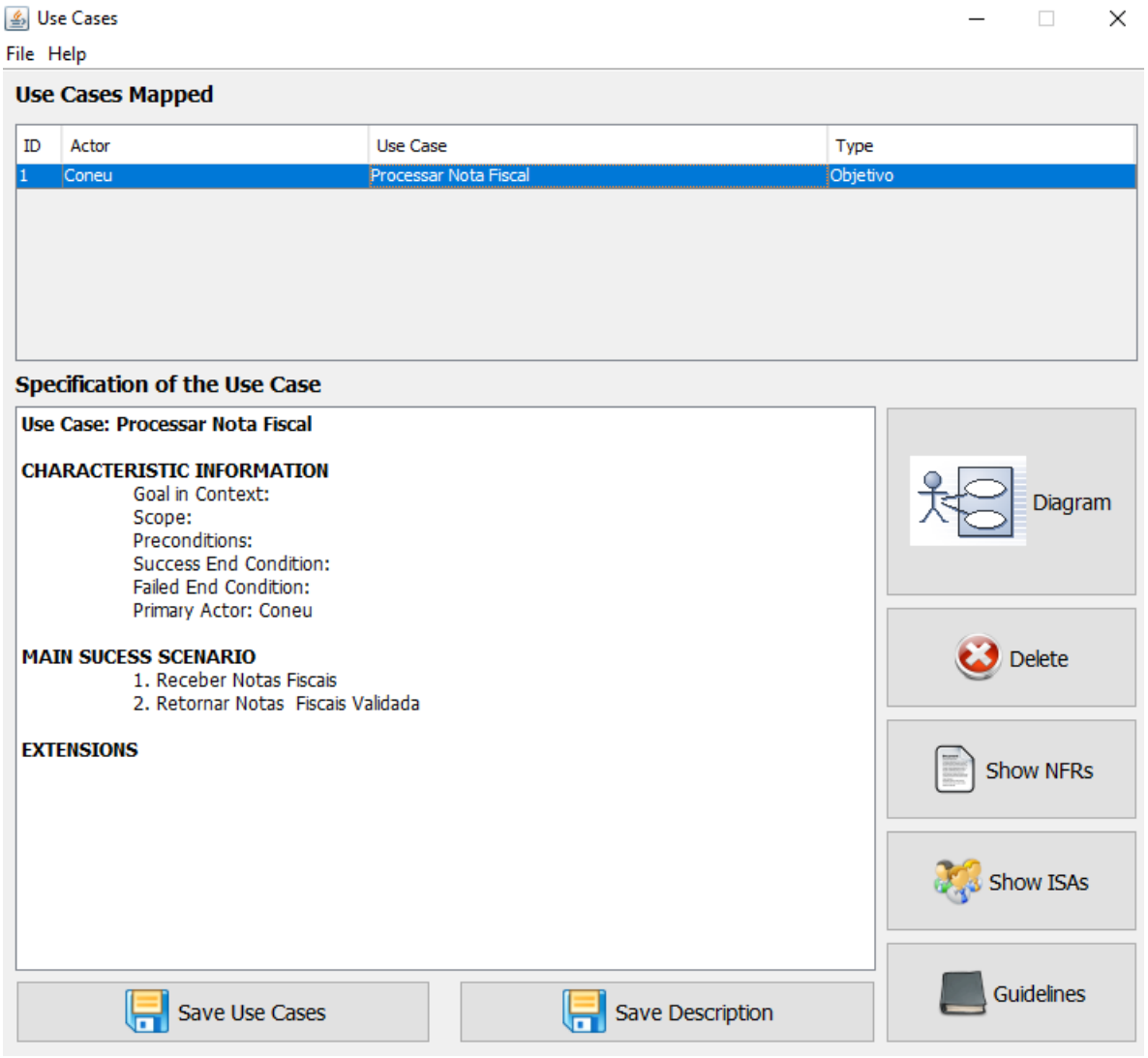

Figura 4.8: Caso de uso mapeado corretamente

Terceiro Passo 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 SR 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.

SubDiretriz 8.1: analisar os subcomponentes em uma ligação de *refinement* mapeando-os para passos na descrição do cenário primário (fluxo principal) de casos de uso, se a análise não descrever o fluxo principal, a mesma pode ser utilizada para mapear os passos alternativos na descrição. As ligações podem ser do tipo *AND* para o antigo *means-end* ou do tipo *OR* para o

antigo *task decomposition*.

SubDiretriz 8.2: analisar, se existir, *qualities* ligadas a outros elementos para assim, colocá-las como *open issues* que devem ser respeitados para o caso de uso em questão.

SubDiretriz 8.3: analisar, se existir, *resource* ligado a alguma *task* com a ligação de *neededby* para assim, determinar que tal recurso é obrigatório para o caso de uso em questão, colocando-o após o passo do caso de uso em questão, podendo ele ser do fluxo principal ou do fluxo secundário.

SubDiretriz 8.4: analisar, se existir, ligação do tipo *contribution* para verificar o quanto uma qualidade influencia no caso de uso em questão. Se a ligação for *hurt* ou *break*, sua influência é pouco negativa e muito negativa, respectivamente, enquanto se a ligação for *help* ou *make*, sua influência é pouco positiva ou muito positiva, respectivamente. Após ser feita essa análise, colocá-la na descrição textual no campo *open issues*.

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.

Diretriz 10: Desenvolver o diagrama de casos de uso utilizando os casos de uso descobertos e os relacionamentos do tipo «include», «extend» e «generalization» usados para estruturar as especificações dos casos de uso.

Capítulo 5

Validação da Proposta

Neste capítulo, será feito a validação das mudanças nas diretrizes escritas por (SANTANDER; CASTRO, 2002) propostas na seção 4.2 aplicando-as a um exemplo escolhido pelo autor. Tal exemplo foi escolhido considerado que o mesmo já foi publicado em (SANTANDER; SILVA, 2014) sendo que o mesmo foi adaptado para exemplificar as mudanças realizadas nas diretrizes.

5.1 Validação

Para realizar a validação, faremos o mapeamento usando as diretrizes modificadas da proposta original de (SANTANDER; CASTRO, 2002) para, a partir do exemplo da Figura 5.2 utilizando a versão 2.0 do i* proposto por (DALPIAZ; FRANCH; HORKOFF, 2016), os elementos novos estão destacados em vermelho na Figura 5.2, e também é apresentado um diagrama do i* original na Figura 5.1 para comparar os dois diagramas.

Segue a aplicação das diretrizes modificadas:

Primeiro Passo da Proposta - Descoberta de Atores

Diretriz 1 (diretriz modificada): Os atores identificados foram os seguintes: ator genérico Gerente, ator genérico Funcionário, ator genérico Cliente e ator genérico Sistema.

Diretriz (diretriz modificada): Os atores externos ao sistema computacional são Cliente, Funcionário e Gerente.

Diretriz 3: O único ator que possui uma dependência com o ator sistema é Funcionário.

Diretriz 4: Há uma relação do tipo *is-a* entre o ator Gerente e Funcionário que origina uma relação entre os mesmos do tipo «generalização».

Segundo Passo da Proposta - Descoberta de Casos de Uso

Diretriz 5: Não há dependências do sistema em relação ao ator Funcionário mapeado na Diretriz 3.

SubDiretriz 5.1: Não se aplica

SubDiretriz 5.2: Não se aplica.

SubDiretriz 5.3: Não se aplica.

SubDiretriz 5.4 (subdiretriz modificada): Essa diretriz teve uma mudança de nomenclatura, mas não diagramática, portanto não houve impacto, para o exemplo utilizado, não se aplica.

Diretriz 6: De acordo com esta diretriz, as dependências mapeadas para casos de uso são: Consultar Produtos, Efetuar Venda e Emitir Nota Fiscal. Também as dependência do tipo *quality* Segurança e Desempenho são mapeadas para requisitos não funcionais do sistema.

Diretriz 7: Todos os casos de uso gerados Consultar Produtos, Efetuar Venda e Emitir Nota Fiscal são classificados como objetivos de usuário.

Terceiro Passo da Proposta - Descoberta e descrição do fluxo principal e alternativo dos Casos de Uso

Diretriz 8 (diretriz modificada): não houve mudança diagramática nessa diretriz, apenas uma separação em 4 sub diretrizes que devem auxiliar na geração de descrição textual para o caso de uso.

SubDiretriz 8.1: A *task* Consultar Produtos possui ligações do tipo *AND*, antigo *means-end*, com as *tasks* Buscar Produtos e Mostrar Produtos, as quais são mapeadas para passos do cenário principal do caso de uso Consultar Produtos. Aplica-se o mesmo procedimento para os casos de uso Efetuar Venda e Emitir Nota Fiscal.

SubDiretriz 8.2: Existe uma *qualification* que liga a *quality* Rapidez com o *goal* Consultar Produtos, isso significa dizer que a Rapidez está diretamente ligada ao *goal* Consultar Produtos, ou seja, não é estritamente necessário que exista Rapidez no sistema no caso de uso de consulta de produtos, no entanto é algo desejável, tal situação deve ser colocada no campo *open issues*. Também, do mesmo modo, podemos citar a *quality* Integridade ligada via *qualification* com o caso de uso Consultar Produtos. Observando os exemplos das Figuras 5.6 e 5.2, percebemos que as *qualifications* que ligam as *qualities* Rapidez e Integridade a *task* Consultar Produtos foram incorporadas no campo *open issues*, demonstrando assim uma *quality* desejada para o

sistema.

SubDiretriz 8.3: Existem dois *resources* ligados a duas *tasks* que são *resources* necessários para a realização da *task* as quais eles estão associados, como por exemplo, o *resource* Mecanismo de Pesquisa é necessário para a realização da *task* Buscar Produtos, pois sem ele, não há como pesquisar qualquer produto no Sistema. Observando os exemplos das Figuras 5.6 e 5.2, vemos que os recursos que são necessários para efetuar um *goal* foram anexados a descrição textual do cenário principal, como por exemplo o recurso SGBD é necessário para a execução do objetivo de Buscar Produtos e o recurso Monitor é necessário para Mostrar Produtos.

SubDiretriz 8.4: Existe um *link* de *contribution* do tipo *hurt* entre a *task* Buscar Produtos e a *quality* Rapidez, a qual deve ser adicionada como um *open issue* na descrição textual. Do mesmo modo, a *quality* Integridade está ligada à *task* Buscar Produtos com uma ligação *contribution* do tipo *help*. Observando os exemplos das Figuras 5.6 e 5.2, podemos perceber que os *links* de contribuição foram incorporados no campo *open issues* como visto nos *issues*: 3. Buscar Produtos *hurt* Rapidez e 4. Buscar Produtos *help* Integridade.

Diretriz 9: Não se aplica.

Diretriz 10: O diagrama gerado é apresentado na figura 5.5.

Os casos de uso gerados para o ator Sistema do exemplo se encontram nas Figuras 5.6. Os casos de uso gerados para o mesmo ator no i* original se encontram na Figuras 5.3, 5.4 e 5.5, logo podemos perceber que são os mesmos casos de uso gerados tanto para uma versão quanto para outra, no entanto, o que muda é a descrição textual. Enquanto a Figura 5.7 mostra como ficou mapeado o caso de uso escolhido dentro de um diagrama *Unified Modeling Language* (UML).

Podemos perceber que com o uso das diretrizes modificadas, a descrição textual fica mais extensa e explicativa devido a adição do campo *open issues* no *template* de descrição textual de caso de uso, sem modificar tanto a descrição diagramática, nem modificando o diagrama do caso de uso em UML. Isso pode ser utilizado pelo engenheiro de requisitos para auxiliar a melhor descrição dos métodos e dos atributos que devem ser implementados pelos desenvolvedores do sistema.

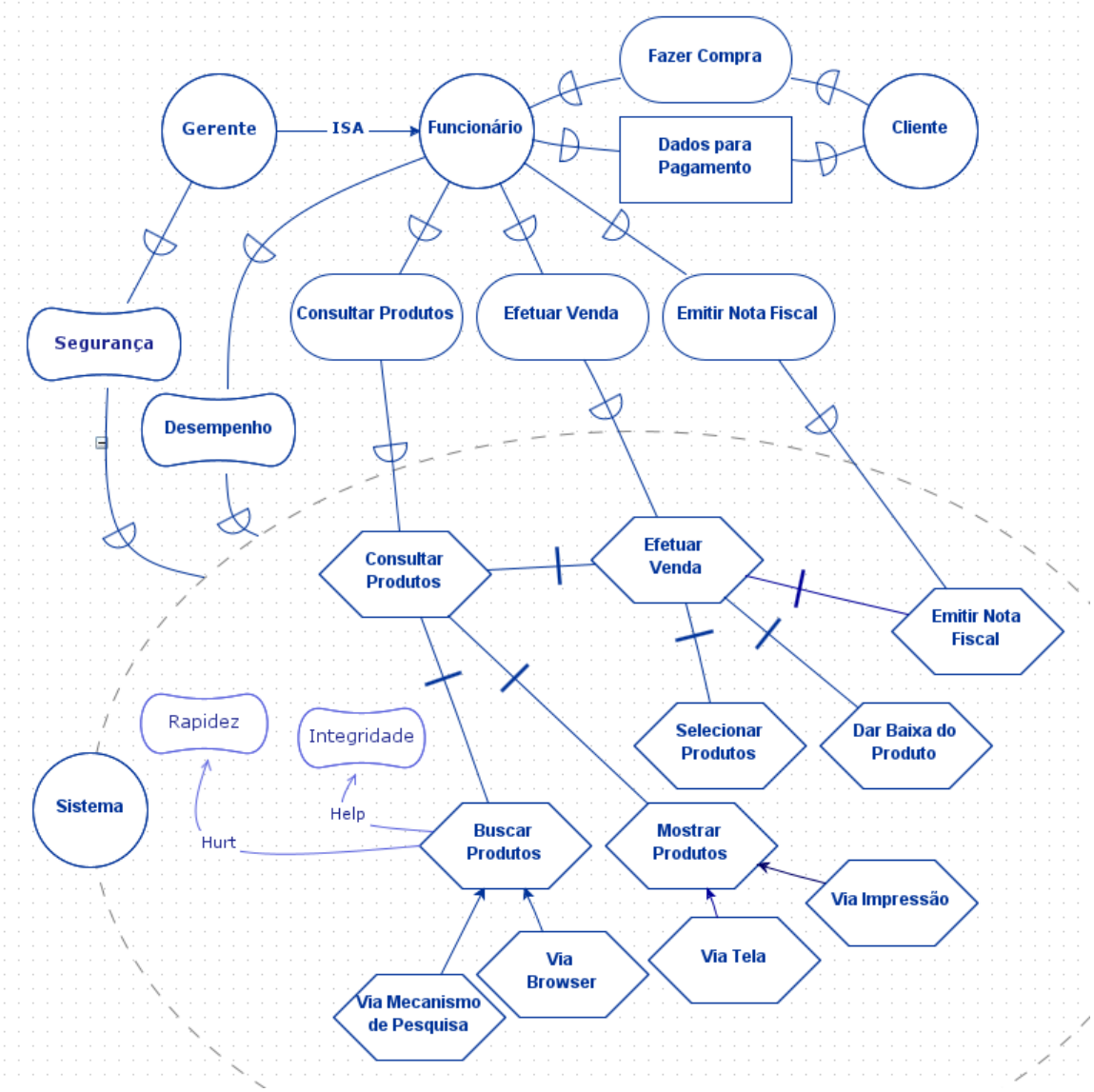

Figura 5.1: Exemplo de i* original para comparação

5.2 Discussão sobre as Mudanças Necessárias na Ferramenta JGOOSE para Acomodar as Diretrizes Modificadas

As mudanças necessárias para a ferramenta JGOOSE contemplam a inclusão dos seguintes elementos da versão i* 2.0: a ligação de relação entre dois atores chamada *participates-in*, trocado a nomenclatura de *softgoal* para *quality*, englobado as noções de *task decomposition* e *means-end* e criado uma nova chamada *refinement*, também houve a adição de dois *links* intencionais chamados *qualification* e *neededBy*, bem como a exclusão dos seguintes elementos que já não existem mais na versão 2.0: o ator do tipo *position*, as relações entre dois atores do

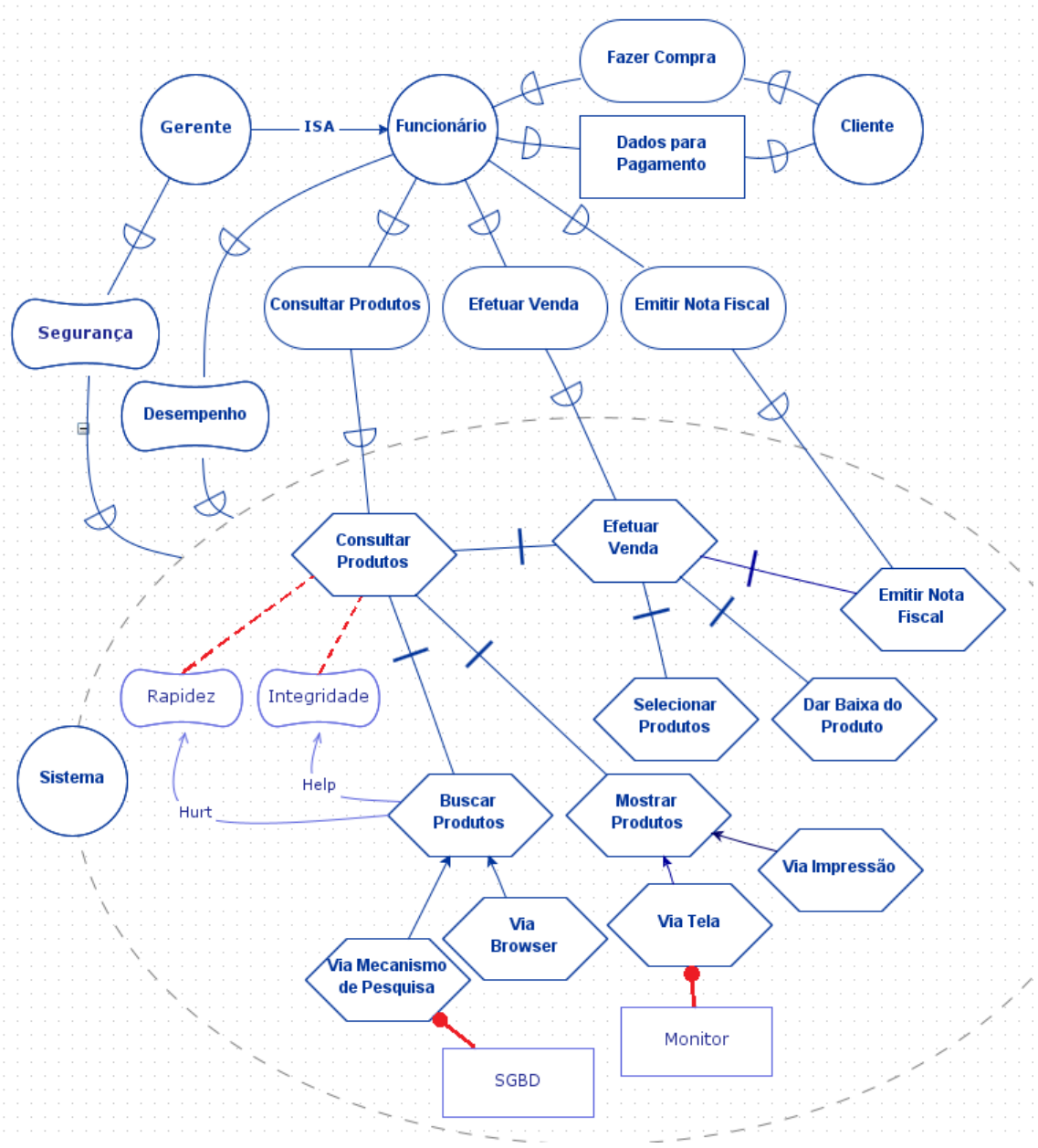

Figura 5.2: Exemplo para validação da proposta, já com os elementos do i* 2.0

tipo *is-part-of*, *plays*, *occupies* e *covers* e retirado as nomenclaturas de *task decomposition* e *means-end*.

Também vale lembrar que há a necessidade de modificar as diretrizes internas da ferramenta, bem como a adição de um campo novo, chamado *open issues*, na descrição textual quando a mesma for gerada e a adequação do cenário principal para comportar a obrigatoriedade de recursos necessários, como demonstrado na seção anterior.

A arquitetura atual da ferramenta JGOOSE é a MVC (*Model - View - Controller*), o que

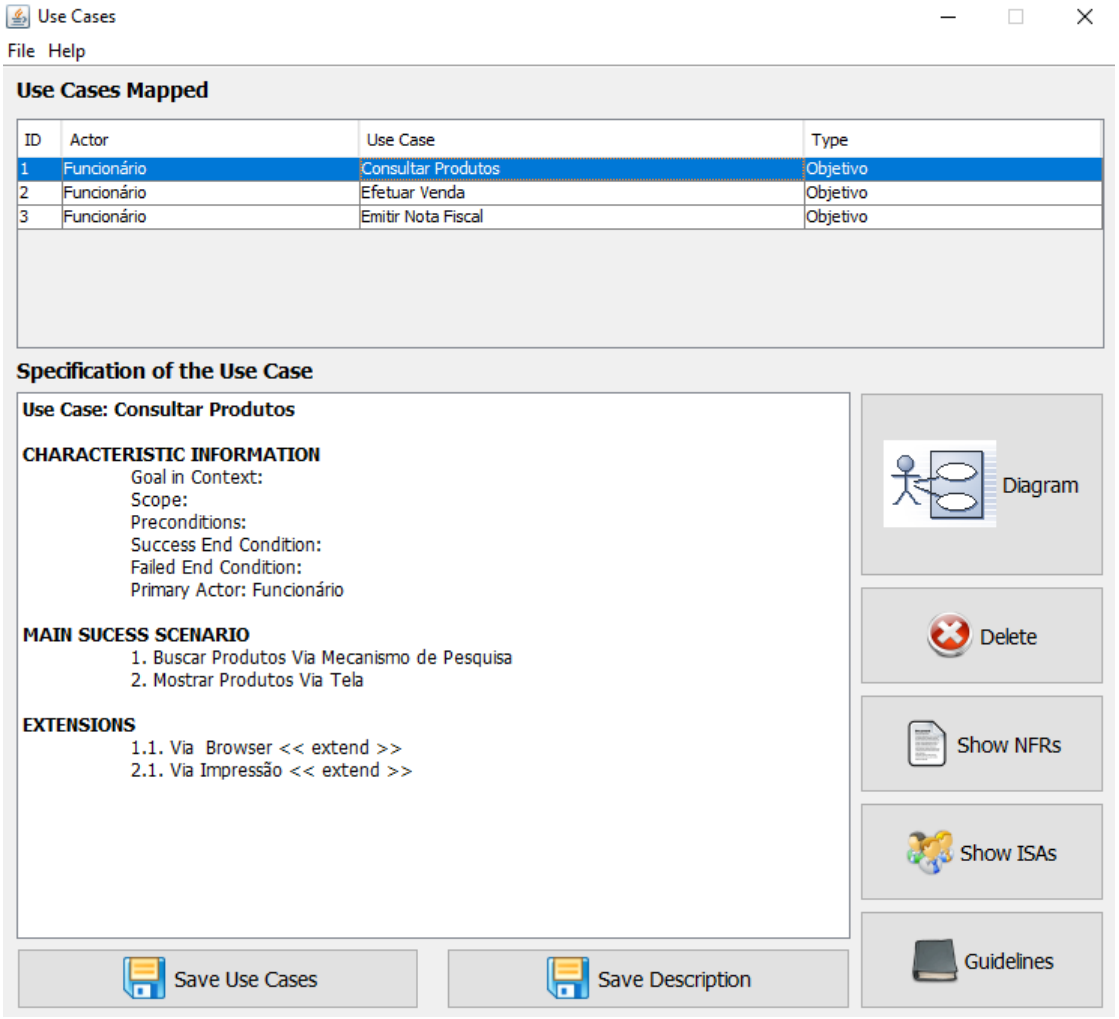

Figura 5.3: Casos de uso gerados para o i* original, primeiro caso de uso

significa dizer que existem três pacotes principais no projeto, o pacote de visão (*View*) onde está a *User Interface* (UI), o pacote de modelo (*Model*) onde está a camada de manipulação de dados do *software* e o pacote de controle (*Controller*) onde estão os métodos de controle e lógicos do sistema.

Também conforme descrito na subseção 4.1.5, há um problema de mapeamento de caso de uso quando há uma relação direta entre um *goal* e uma dependência interna de um ator, onde deveria ser feito o mapeamento do caso de uso, no entanto, ele é simplesmente ignorado. Para tal erro, é proposto um *bugfix* para a correção do mesmo.

O que foi implementado:

- Modificação das diretrizes do JGOOSE para serem usadas na nova tela que deve ser implementada para comportar a nova versão do i* 2.0, para isso foi criado um novo arquivo

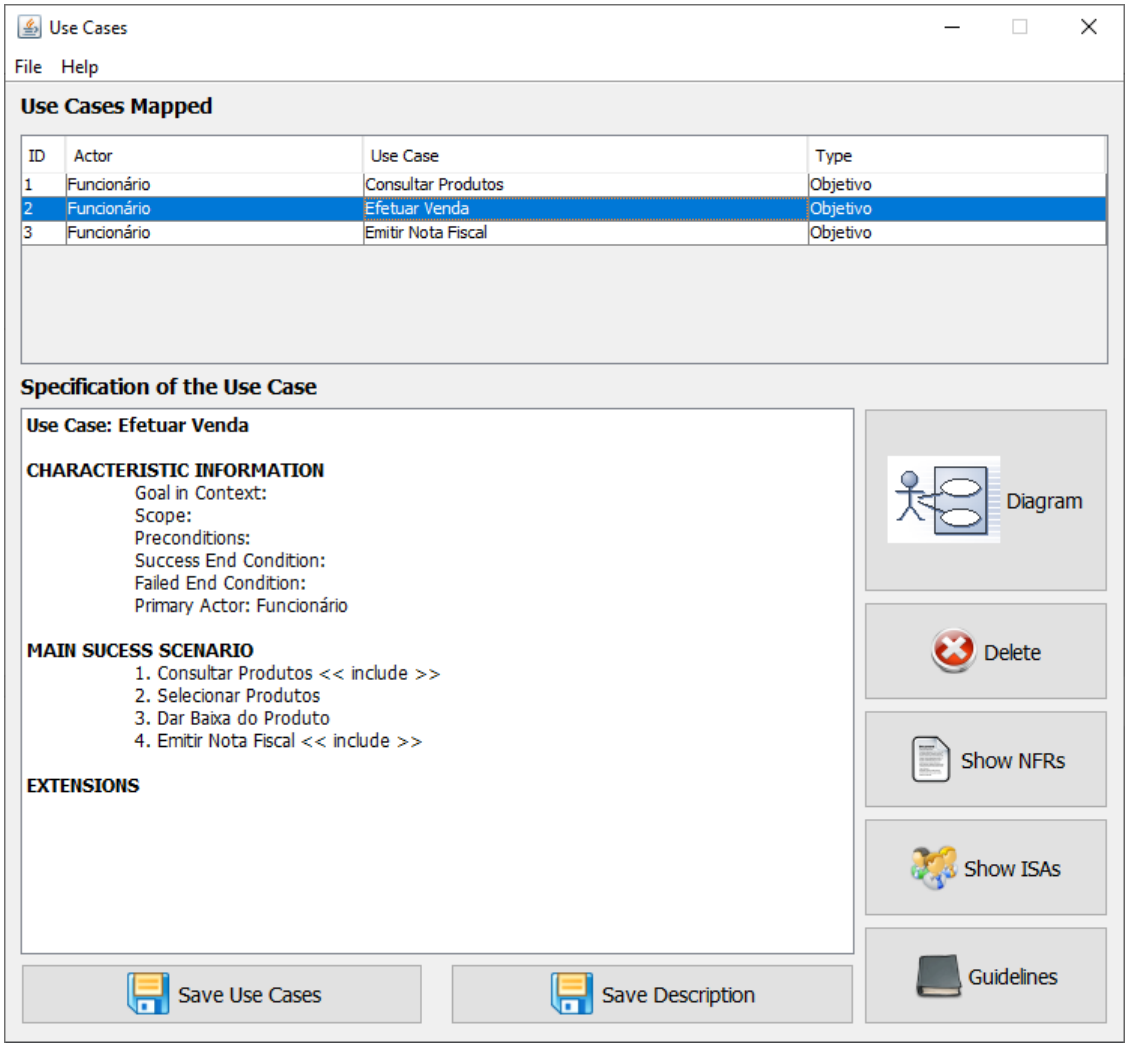

Figura 5.4: Casos de uso gerados para o i* original, primeiro caso de uso

no pacote de *Controller* chamado *Guidelines2.java* o qual é um *Enum* e descreve todas as diretrizes modificadas;

- O *bugfix* do problema de mapeamento errado de caso de uso, o qual foi comentado na seção 4.2, se encontra nas Figuras 5.8 e 5.9.

O que falta ser implementado:

- Uma nova tela para comportar a nova versão do i* 2.0, a razão disso é deixar a cargo do usuário escolher qual versão do i* ele deseja utilizar;
- Implementar os novos elementos do i* 2.0 nesta nova tela;
- Remover os antigos elementos que não estão mais presentes na versão 2.0 do i*.

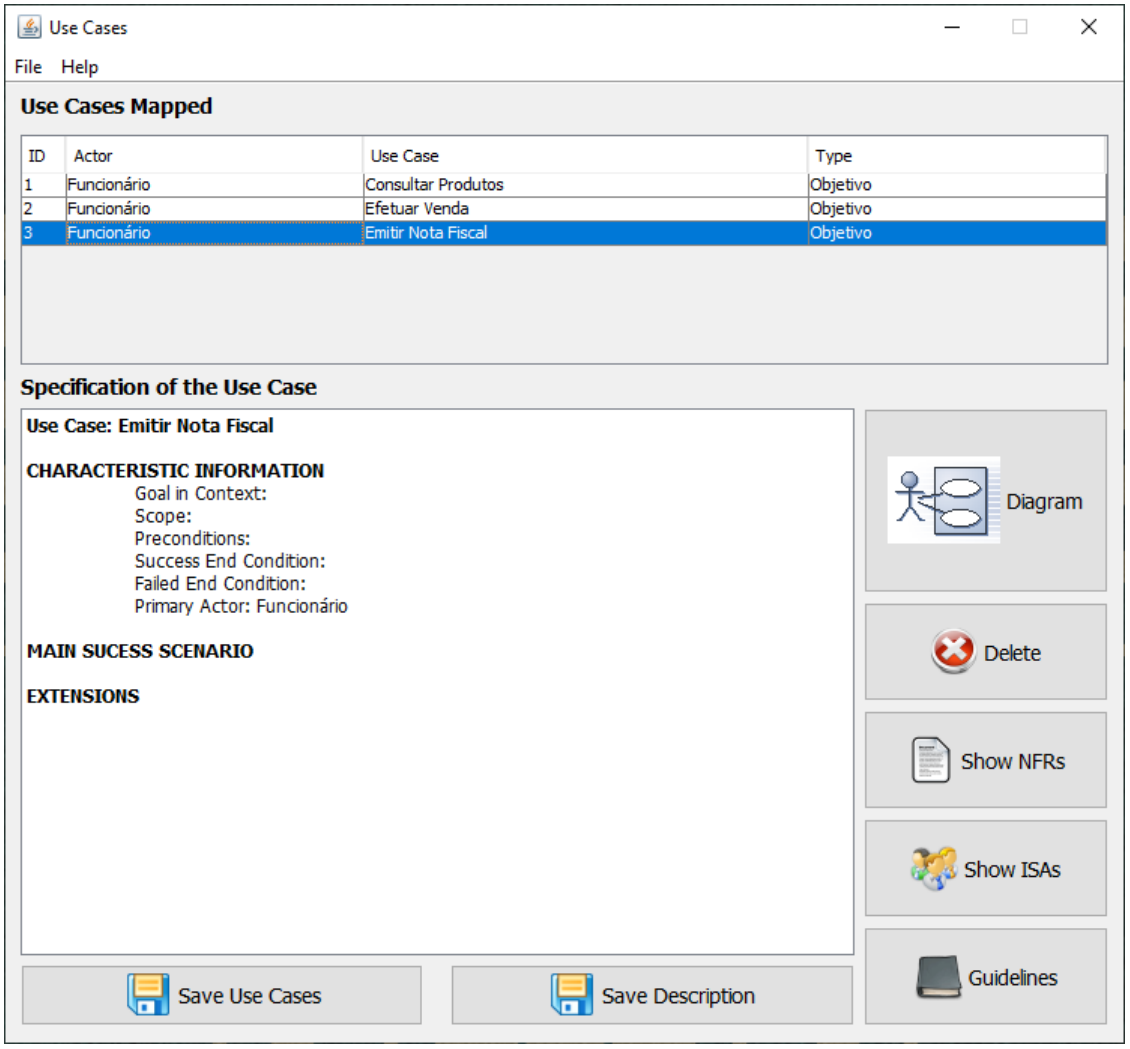

Figura 5.5: Casos de uso gerados para o i* original, primeiro caso de uso

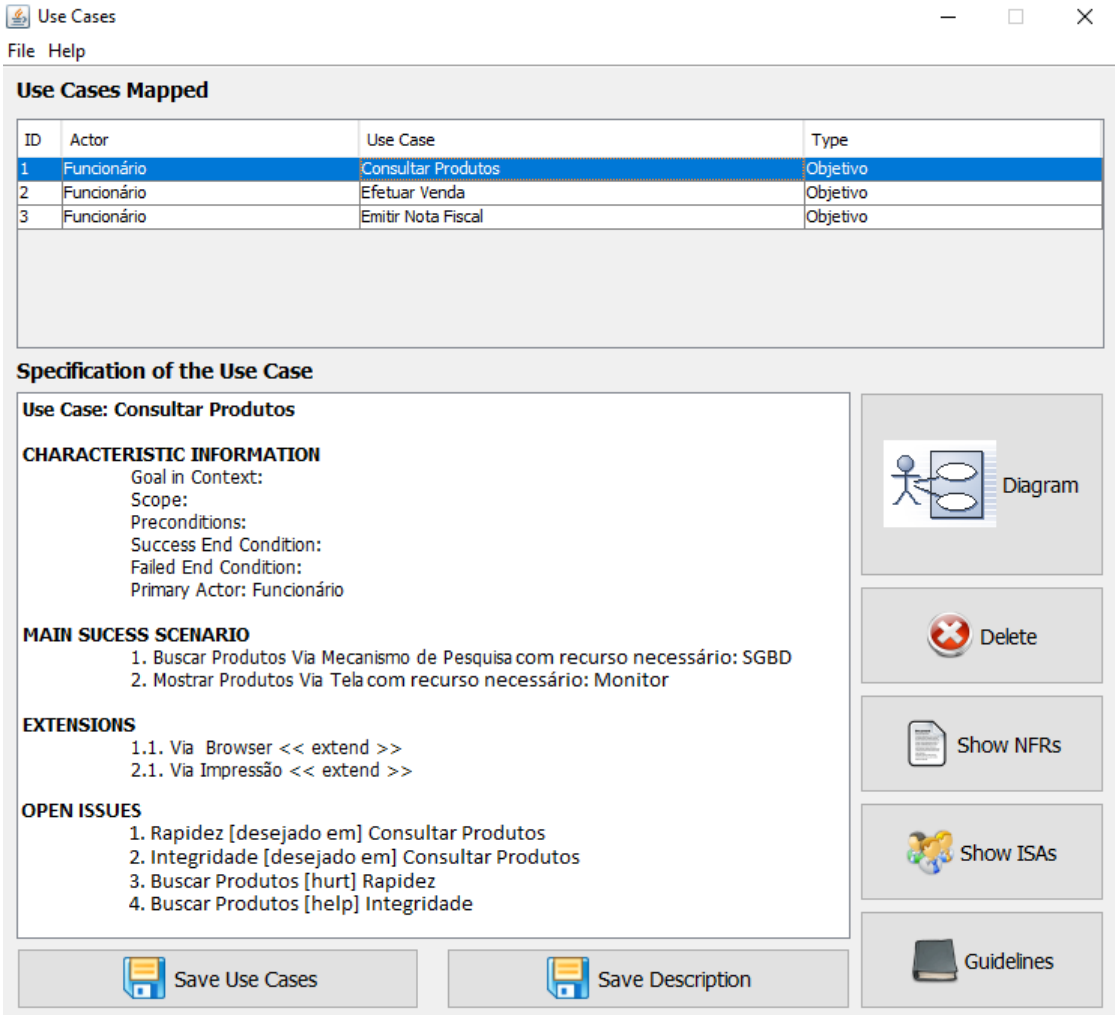

Figura 5.6: Casos de uso gerados para o i* 2.0

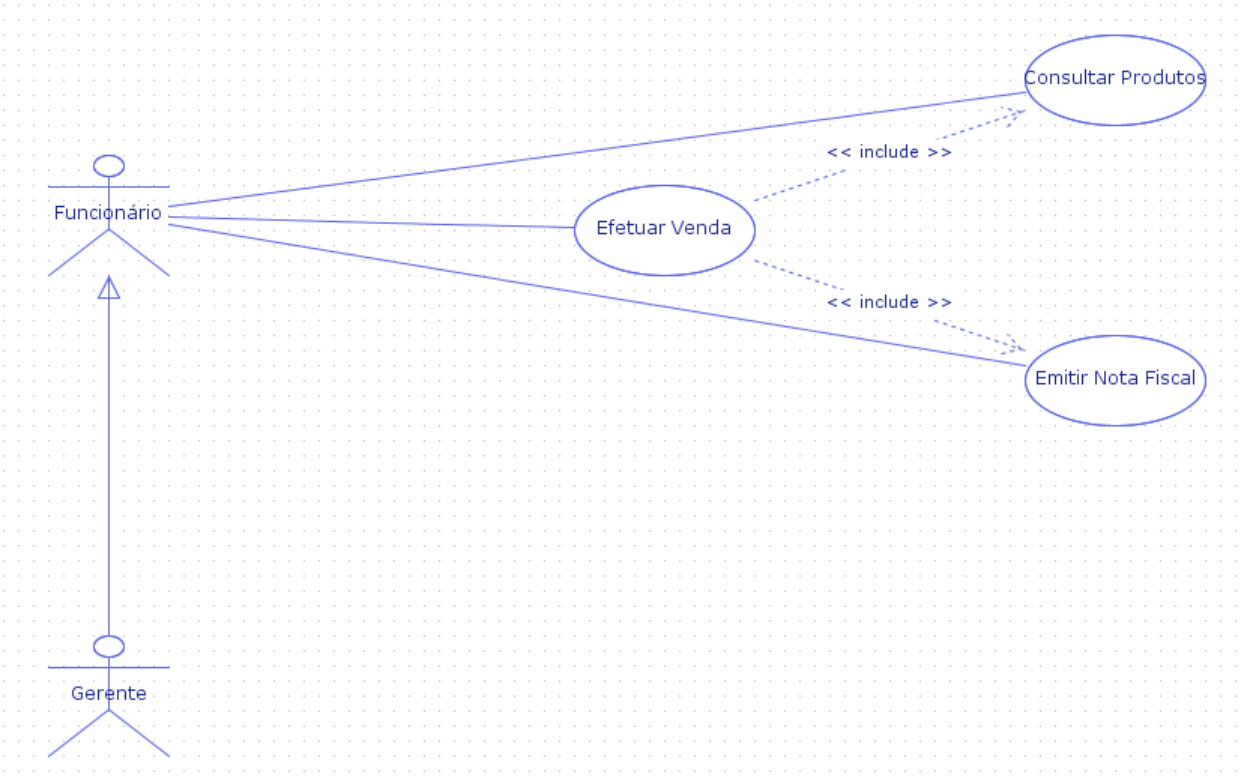

Figura 5.7: Caso de uso mapeado

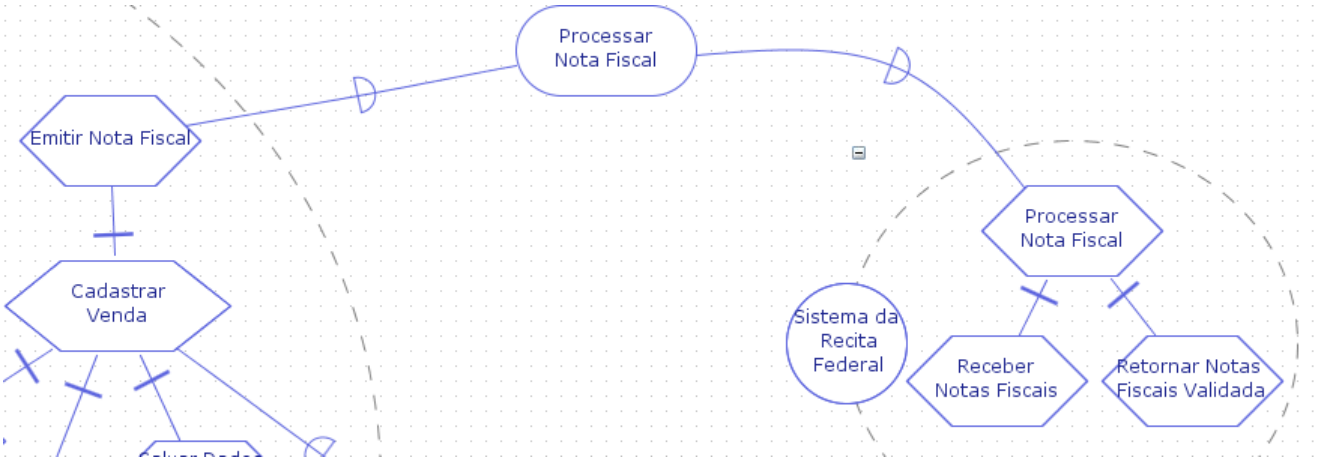

Figura 5.8: Diagrama do antigo problema de mapeamento de caso de uso

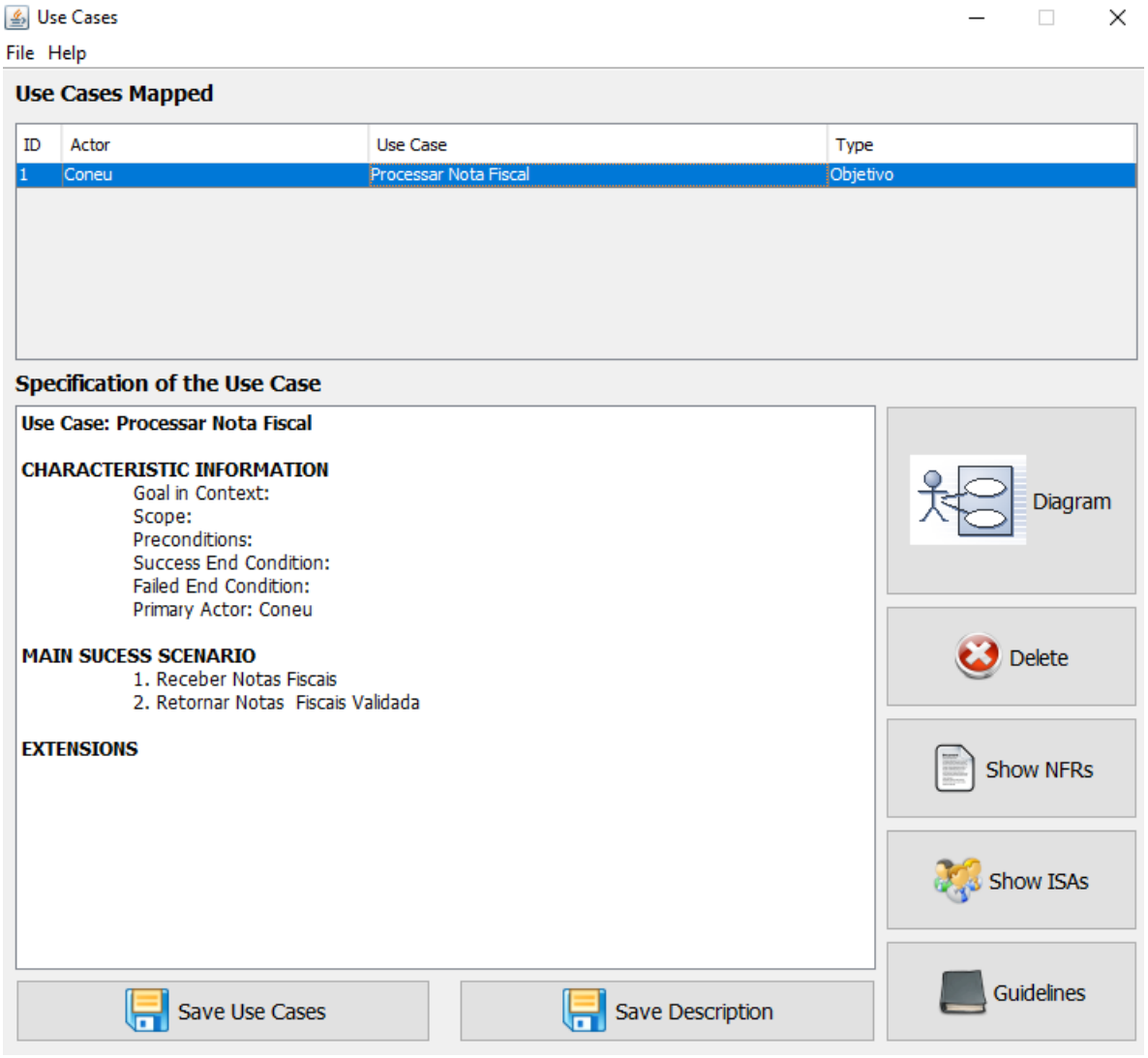

Figura 5.9: Caso de uso mapeado corretamente, após o *bugfix*

Capítulo 6

Considerações Finais e Trabalhos Futuros

Neste capítulo serão apresentadas as considerações finais do autor, bem como possíveis trabalhos futuros para melhorar a ferramenta JGOOSE ou mapeamento de casos de uso a partir do i* 2.0 (DALPIAZ; FRANCH; HORKOFF, 2016).

6.1 Considerações Finais

De acordo com o resultado apresentado nas Figuras 5.6 e 5.7 podemos perceber que não há uma mudança no mapeamento de caso de uso. No entanto, podemos perceber que a descrição textual foi modificada, de modo que a mesma fique mais completa e mais bem descrita devido a adição do campo *open issues*.

Quanto as diretrizes, elas foram modificadas para contemplar mais situações e mais elementos que já existiam no i* original (YU, 1995) e não estavam sendo levados em consideração nas diretrizes propostas por (SANTANDER; CASTRO, 2002), bem como comportar alguns elementos do i* 2.0 (DALPIAZ; FRANCH; HORKOFF, 2016).

Portanto, podemos afirmar que a descrição textual ficou mais completa e as diretrizes agora abrangem mais cenários possíveis, logo, considero válido este trabalho de conclusão de curso.

Também, como mencionado no capítulo anterior, foi feito o *bugfix* do problema de mapeamento de caso de uso, como foi demonstrado nas Figuras 5.8 e 5.9.

6.2 Trabalhos Futuros

Para possíveis trabalhos futuros podemos citar as seguintes linhas de pesquisa:

- Fazer experimentos com mais cenários de mapeamento para caso de uso de i*, para fazer uma comparação entre as descrições textuais e diagramáticas entre a versão original de i* (YU, 1995) e a versão 2.0 (DALPIAZ; FRANCH; HORKOFF, 2016), a luz da engenharia de *software* experimental;

- Implementar os novos elementos do i* 2.0, bem como uma nova tela para comportá-los, tendo como base a discussão sobre a mesma realizada na seção 5.2, na ferramenta de suporte JGOOSE.

- Testar a usabilidade da ferramenta, para verificar se o processo de mapeamento de caso de uso, bem como sua descrição textual, diagramática e geração de caso de uso, atendem às necessidades do usuário.

Referências Bibliográficas

ALDER, G. *Design and implementation of the JGraph swing component*. 2002. Acessado em: http://www.jgraph.com/doc/paper/.

BOEHM, B. *Software Engineering Economics*. 1981. Prentice Hall PTR Upper Saddle River, NJ, USA.

BOOCH, R.; RUMBAUGH, A.; JACOBSON, J. *Unified Modeling Language User Guide*. 2005.

BRISCHKE, A. *Desenvolvimento de uma ferramenta para integrar modelagem organizacional e modelagem funcional na engenharia de requisitos*. 2005.

BRISCHKE, M.; SANTANDER, V.; SILVA, I. *Melhorando a ferramenta JGOOSE*. 2012.

COCKBURN, A. *Writing Effective Use Cases*. 2000. Addison Wesley Longman Publishing Co., Boston, MA, USA.

DALPIAZ, F.; FRANCH, X.; HORKOFF, J. *iStar 2.0 Language Guide*. 2016. Acessado em: https://sites.google.com/site/istarlanguage/home.

KOTONYA, G.; SOMMERVILLE, I. *Requirements Engineering: Processes and Techniques*. 1998.

LAMSWEERDE, A. *Requirements Engineering in the Year 00: A Research Perspective*. 2000. 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland.

LAMSWEERDE, A. *Goal-Oriented Requirements Engineering: A Guided Tour*. 2001. 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada.

LAPOUCHNIAN, A. *Goal-Oriented Requirements Engineering: An Overview of the Current Research*. 2005.

MERLIN, L. *E4J: Editor i* para JGOOSE*. 2013. Requirements Engineering Workshop 2015.

PRESSMANN, R. *Software Engineering*. 2006. McGraw-Hill Science/Engineering/Math. 8th Edition.

SANTANDER, V.; CASTRO, J. *Deriving Use Cases from Organizational Modeling*. 2002. IEEE Joint International Requirements Engineering Conference - RE 02, 2002, Essen, Germany.

SANTANDER, V. F. A.; SILVA, I. F. *Avaliando a utilização da Ferramenta JGOOSE no Processo de Ensino e Aprendizagem na Engenharia de Requisitos Um Relato de Experiência*. 2014. XIX Conferência Internacional sobre Informática na Educação (TISE), 2014, Fortaleza. Anais da XIX Conferência Internacional sobre Informática na Educação, 2014.

SOMMERVILLE, I. *Software Engineering*. 2011. Pearson Education, Inc. 9th Edition.

VICENTE, A. *JGOOSE: Uma ferramenta de Engenharia de Requisitos para a Integração da Modelagem Organizacional i* com a Modelagem Funcional de Casos de Uso UML*. 2006.

YU, E. *Modelling Strategic Relationships for Process Reengineering*. 1995. Ph.D. Thesis. Dept. of Computer Science, University of Toronto.

YU, E. et al. *Social Modeling for Requirements Engineering*. [S.l.]: The MIT Press, 2011.