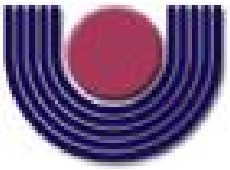


Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

E4J Use Cases: um editor de diagrama de casos de uso integrado à ferramenta JGOOSE

Diego Peliser

CASCAVEL
2014

Diego Peliser

E4J Use Cases: um editor de diagrama de casos de uso integrado à 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
2014

DIEGO PELISER

E4J Use Cases: um editor de diagrama de casos de uso integrado à 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. Dr. Victor Francisco Araya Santander
Colegiado de Ciência da Computação,
UNIOESTE

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

Prof. Me. Elder Schemberger
UTFPR

Prof. Me. Sidgley Camargo de Andrade
UTFPR

Cascavel, 20 de novembro de 2014

DEDICATÓRIA

Aos meus pais, por sempre acreditarem e confiarem em mim, dedicando-se para me dar todo o suporte necessário e tornar isso possível. Sem vocês jamais teria chegado até aqui. Muito obrigado!

AGRADECIMENTOS

Aos meus pais, Cleunir Peliser e Inês Blank Peliser, e ao meu irmão, Douglas Peliser, pelo amor, dedicação, incentivo e auxílio não somente durante o desenvolvimento desse trabalho, mas em toda minha vida. Vocês são meu maior orgulho.

A minha namorada, Taíza Gabriela Zanatta Crestani, por todo o amor, carinho, felicidade e ajuda proporcionada durante o desenvolvimento desse trabalho. Com certeza você foi a minha maior motivação durante essa etapa da minha vida.

Ao meu orientador Victor Francisco Araya Santander, pelos conselhos, apoio, disponibilidade e interesse ao longo dos quatro anos de pesquisa. Agradeço também pela confiança e estímulo para o desenvolvimento de todos os trabalhos realizados em conjunto bem como por todo o grande e valioso conhecimento que me proporcionou.

Também agradeço a todos os outros membros do grupo de estudo do Laboratório de Engenharia de Software (LES), os quais, durante as apresentações de trabalhos do grupo, fizeram contribuições de grande valor.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BSD	*Berkeley Software Distribution*
E4J	*Editor for* JGOOSE
GOOSE	*Goal into Object Oriented Standard Extension*
GQM	*Goal/Question/Metric*
GUI	*Graphical User Interface*
JGOOSE	*Java Goal into Object Oriented Standard Extension*
OME	*Organization Modelling Environment*
OO	Orientado à Objetos
UML	*Unified Modeling Language*
UNIOESTE	Universidade Estadual do Oeste do Paraná

Sumário

Resumo

O desenvolvimento de *software* 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 elicitados. A engenharia de requisitos, subárea da engenharia de *software*, tem por objetivo tratar esse processo inicial de geração de requisitos funcionais e não funcionais. Para que isso ocorra de forma adequada e eficiente é necessário estudar e entender as questões não apenas que envolvem o sistema computacional, mas também os elementos abordados por modelos organizacionais, como o *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*) juntamente com o diagrama de casos de uso. Neste contexto, foi proposta uma integração entre o *framework* i* e os casos de uso UML visando alcançar uma qualidade maior nessa fase de grande importância para a engenharia de *software*. Para apoiar essa integração foi desenvolvida uma ferramenta denominada JGOOSE. Entretanto, essa ferramenta apresentava como maior limitação a sua dependência com a ferramenta StarUML para manipular diagramas de casos de uso, pois o JGOOSE permitia gerar apenas diagramas de casos de uso estáticos, não sendo possível realizar quaisquer modificações. Assim, era necessário exportar o diagrama de casos de uso estático gerando um arquivo intermediário para ser importado pela StarUML. Desta forma, este trabalho apresenta o projeto e desenvolvimento de um editor gráfico de diagrama de casos de uso integrado à ferramenta JGOOSE, visando melhorar suas funcionalidades e eliminar a necessidade de outro *software* para este fim.

Palavras-chave: E4J, JGOOSE, casos de uso UML, *framework* i*, engenharia de requisitos.

Capítulo 1

Introdução

Este primeiro capítulo tem por objetivo a apresentação geral do trabalho. Inicialmente, na seção 1.1, é realizada a contextualização do trabalho no âmbito da engenharia de requisitos e, mais especificamente, das ferramentas de geração de casos de uso e de modelos organizacionais. Na seção 1.2 é apresentada a motivação para a realização desse trabalho. Seguindo para a seção 1.3, apresenta-se a proposta de desenvolvimento desse trabalho. Na seção 1.4 é descrito as contribuições esperadas com a nossa proposta. Por fim, na seção 1.5, é descrito a estrutura geral desse trabalho e a organização do restante da monografia.

1.1 Contexto

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, modelar a organização é o primeiro passo para garantir que os requisitos de um sistema estejam de acordo com as necessidades reais da organização. Um dos *frameworks* mais utilizados tanto na comunidade acadêmica como industrial é o i* (*i-star*) [YU 1995].

Também 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 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 surge a UML (*Unified Modeling Lan-*

guage) [BOOCH, RUMBAUGH e JACOBSON 2005], a qual é uma linguagem que define uma série de artefatos que nos ajuda na tarefa de modelar e documentar os sistemas orientados a objetos que são desenvolvidos. Ela possui, em sua segunda edição do guia do usuário [BOOCH, RUMBAUGH e JACOBSON 2005], onze tipos de diagramas que são usados para documentar e modelar diversos aspectos do sistema. Entre esses diagramas, destaca-se o Diagrama de Casos de Uso que visa documentar o sistema do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e suas interações com os usuários do sistema. Esse artefato é comumente derivado da especificação de requisitos e pode ser utilizado para criar o documento de requisitos.

Assim, destaca-se a proposta apresentada por [SANTANDER e CASTRO 2002], a qual propõe derivar casos de uso UML [BOOCH, RUMBAUGH e JACOBSON 2005] a partir de modelos construídos via *framework* i*; e para dar suporte computacional à essa proposta, foi desenvolvida uma ferramenta denominada JGOOSE [VICENTE 2006], que permite integrar modelos i* e casos de uso UML, gerando de forma automática os casos de uso a partir dos modelos criados no *framework* i* como também um diagrama de casos de uso **estático** a partir dos casos de uso gerados.

A ferramenta JGOOSE também possui integração com a ferramenta EJ4 (*Editor for JGOOSE*) [MERLIM 2013], a qual é um editor de modelos organizacionais i*. Os modelos SD e SR construídos através da ferramenta E4J bem como os construídos via ferramenta OME [YU e YU 2014] são utilizados como entrada para a geração de casos de uso na ferramenta JGOOSE.

A versão atual do JGOOSE não permite realizar quaisquer edições nos diagramas de casos de uso gerado, diminuindo dessa forma a produtividade de engenheiros de requisitos na utilização do ambiente de trabalho do JGOOSE.

1.2 Motivação

A principal motivação para o presente projeto está no fato de que a ferramenta JGOOSE gera, a partir dos casos de uso mapeados, um diagrama de casos de uso **estático**, ou seja, não é possível realizar quaisquer tipos de modificação nesse diagrama. Assim, o JGOOSE possui a dependência para com outras ferramentas para criação e manipulação desse tipo de diagrama.

Nesse contexto, percebe-se a necessidade de facilitar a fase inicial do processo de engenharia de *software* e melhorar o processo de criação de diagrama de casos de uso de forma integrada ao JGOOSE bem como o de manipular os diagramas de casos de uso gerados a partir dos casos de uso mapeados pelo JGOOSE. Isso pode melhorar a produtividade do ambiente de trabalho da ferramenta JGOOSE bem como torná-la uma aplicação *standalone*, permitindo a manipulação de diagramas de casos de uso no mesmo ambiente e facilitando futuras evoluções na geração automática de diagramas de casos de uso a partir dos casos de uso mapeados pelo JGOOSE.

1.3 Proposta

Neste trabalho, apresenta-se o desenvolvimento de um editor de diagrama de casos de uso integrado à ferramenta JGOOSE. Foram realizados inicialmente um estudo sobre modelagem organizacional i* e casos de uso UML, bem como a integração destes a partir das diretrizes propostas por [SANTANDER e CASTRO 2002]. Essas diretrizes são a base para o processo de mapeamentos dos casos de uso a partir dos modelos i* realizado pelo JGOOSE.

Para o desenvolvimento do editor de diagrama de casos de uso foi utilizada a base estrutural da ferramenta E4J [MERLIM 2013] para o editor, incluindo: interface gráfica (com modificações), estrutura de grafos JGraphX [JGRAPH 2013], funções para salvar/abrir os diagramas do editor, funções de "*Undo/Redo*", entre outros. Para apresentar a arquitetura do editor, foram adotadas as visões de diagrama de atividades, diagrama de pacotes e diagrama de classes. Cabe destacar que a partir desse projeto, a ferramenta E4J passa a ser denominada E4J i* e o editor de diagrama de casos de uso proposto denominado E4J Use Cases. Para validar a ferramenta E4J Use Cases foi realizado um quase-experimento.

Os diagramas de casos de uso gerados a partir dos casos de uso mapeados pelo JGOOSE refletem as descrições textuais dos casos de uso do JGOOSE. Isso significa que todos os casos de uso são mapeados para o diagrama de casos de uso juntamente com os estereótipos *« include »* e *« extend »* presentes nas descrições textuais associados aos seus devidos casos de uso. Esses diagramas também refletem as ligações do tipo *generalization* entre os atores.

1.4 Contribuições Esperadas

Espera-se com este trabalho contribuir com a área de Engenharia de Requisitos por meio da ferramenta desenvolvida E4J Use Cases, que fornece um ambiente de desenvolvimento de diagrama de casos de uso de forma integrada ao JGOOSE. Essa integração visa maior independência para a ferramenta JGOOSE e, consequentemente, para seus usuários, diminuindo a necessidade de outras ferramentas para realizar a criação e manipulação de diagrama de casos de uso. Espera-se também que este trabalho contribua e incentive o uso da ferramenta JGOOSE e do editor E4J Use Cases para experiências acadêmicas e industriais.

1.5 Estrutura do Trabalho

Este trabalho está dividido em 6 capítulos:

- **Capítulo 2**: aborda os conceitos básicos e elementos que envolvem o *framework* i*, os conceitos necessários para o entendimento de casos de uso UML e os elementos que compõe o diagrama de casos de uso e a integração entre *framework* i* e casos de uso UML proposta por [SANTANDER e CASTRO 2002];
- **Capítulo 3**: apresenta as ferramentas JGOOSE e E4J i*;
- **Capítulo 4**: apresenta o projeto e desenvolvimento do E4J Use Cases;
- **Capítulo 5**: apresenta um quase-experimento realizado com o E4J Use Cases e os resultados obtidos;
- **Capítulo 6**: por fim, apresentam-se as considerações finais deste trabalho.

Capítulo 2

Framework i* e Casos de Uso UML

Para o processo de desenvolvimento de um sistema, as compreensões da organização, do contexto e das lógicas de negócio são de grande importância [YU 1997]. O *framework* i* é um dos artefatos que propiciam essa compreensão, oferecendo uma visão estratégica e intencional dos processos que envolvem o sistema e a organização. Outro artefato comumente utilizado são casos de uso UML, os quais propiciam o entendimento do funcionamento do sistema a todas as partes envolvidas, incluindo programadores, especialistas e engenheiros de requisitos.

Desta maneira, na seção 2.1, são apresentados os conceitos fundamentais do *framework* i* através de seus dois modelos para modelagem. Na seção 2.2, são apresentados os conceitos fundamentais de casos de uso UML, bem como o template adotado para a descrição textual dos mesmos. Na seção 2.3 é apresentada uma proposta de integração do *framework* i* com casos de uso UML, a qual é a base do nosso trabalho conforme justificado no capítulo 1. Por fim, na seção 2.4, são feitas as considerações finais deste capítulo.

2.1 *Framework* i*

O *framework* i*[1] foi originalmente proposto por Eric Yu [YU 1995] e é um *framework* de modelagem organizacional conceitual utilizado no desenvolvimento de modelos que auxiliam a análise de sistemas sob uma visão estratégica e intencional de processos que envolvem vários participantes. Esse *framework* propõe uma abordagem orientada a objetivos com foco nas intencionalidades, relacionamentos e motivações entre os mesmos, o que permite compreender melhor a organização e as relações entre os participantes [YU 1997] [YU et al. 2011]. O *fra-*

[1] Pronuncia-se "i-star". O nome i* faz referência ao conceito sobre uma intencionalidade distribuída. No Brasil também é comum a pronúncia "i-estrela".

mework i* também é um padrão internacional de modelagem de *software* aprovado pela norma ITU-T *Recommendation* Z. 151 em Genebra, na Suíça [YU 2011].

O i* propõe dois modelos para descrever aspectos de intencionalidades e motivações envolvendo os atores no ambiente organizacional: o Modelo de Dependências Estratégicas (SD) e o Modelo de Razões Estratégicas (SR). Esses modelos auxiliam na representação, respectivamente, das dependências entre atores e dos detalhes para satisfazer essas dependências de cada ator. Nas subseções 2.1.1 e 2.1.2 são apresentados os conceitos de cada um dos modelos.

2.1.1 Modelo de Dependências Estratégicas (SD)

O modelo SD, composto por nós e ligações, representa um conjunto de relacionamentos estratégicos externos entre os atores organizacionais, formando uma rede de dependências [YU 1995]. Ele fornece uma visão mais abstrata e ampla da organização, sem se preocupar com os detalhes (razões internas) por trás dessas dependências. A seguir, são apresentadas as descrições dos elementos que compõem este modelo:

- **Ator**: pode ser definido com uma entidade (humana ou computacional) que age sobre o meio que está inserido para alcançar seus objetivos, aplicando seu *know-how* [YU 1995]. Quando existe uma necessidade de maiores detalhes sobre um modelo organizacional, atores podem ser diferenciados em três especializações: agentes, posições e papéis.

  - **Agente**: é a decomposição de um ator que possui manifestações físicas concretas. Refere-se tanto aos humanos quanto aos agentes de *software* ou hardware. Um agente possui independência do papel que está executando;
  - **Posição**: representa uma abstração intermediaria entre um agente e um papel. É o conjunto de papéis tipicamente executados por um agente, ou seja, representa uma posição dentro da organização onde o agente pode desempenhar várias funções (papéis). Diz-se que um agente ocupa uma posição e uma posição cobre um papel;
  - **Papel**: é a caracterização abstrata do comportamento de um ator dentro de determinados contextos ou domínio de informação. Essas características devem ser facilmente transferíveis a outro ator social. As dependências associadas a um papel são aplicáveis independentemente do agente que desempenha o papel.

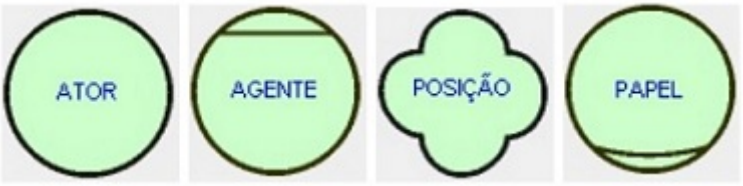


Figura 2.1: Atores e especializações

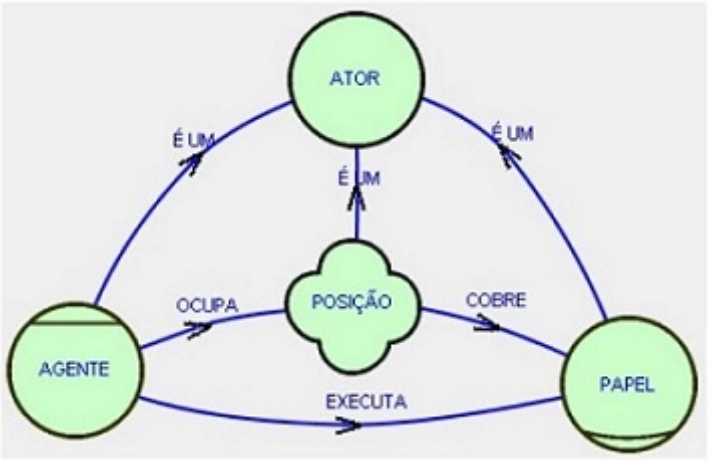


Figura 2.2: Exemplo de relações entre atores

As associações entre atores são descritas através de links de associação:

- **Is-part-of** (faz parte de): é uma associação onde cada papel, posição e agente pode ter sub-partes. Em is-part-of existem dependências intencionais entre o todo e sua parte;
- **Isa** (é um): é uma associação entre dois atores. Essa associação representa uma generalização, onde um ator é um caso especializado de outro ator;
- **Plays** (executa): é uma associação entre um agente e um papel, com um agente executando um papel. A identidade do agente que executa um papel não deverá ter efeito algum nas responsabilidades do papel ao qual está associado e, similarmente, os aspectos de um agente deverão permanecer inalterados mesmo associados a um papel que este desempenha;
- **Covers** (cobre): é uma associação usada pra descrever uma relação entre uma posição e os papéis que a mesma cobre;

- **Occupies** (ocupa): é uma associação usada para mostrar que um agente ocupa uma posição, ou seja, o ator executa todos os papéis que são cobertos pela posição que ele ocupa;
- **Ins**: é uma associação usada para representar uma instância específica de uma entidade mais geral.

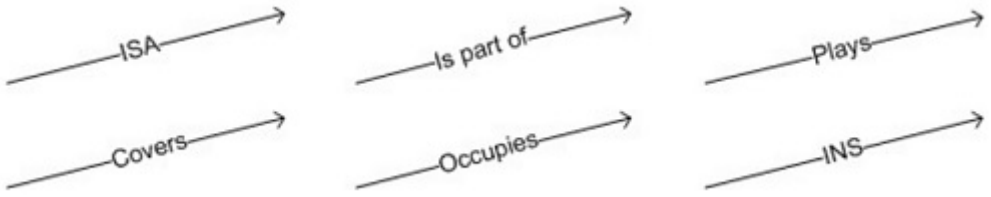


Figura 2.3: Tipos de associações entre atores

Existem nos modelos SD e SR as relações de dependência, as quais podem ser definidas como um acordo entre dois atores. Os elementos que as compõe são:

- ***Depender***: é um ator dependente, ou seja, o ator que precisa que um acordo (*Dependum*) seja realizado. Esse ator não se importa como o outro ator (*Dependee*) irá satisfazer a dependência;
- ***Dependum***: é o elemento intermediário de uma relação de dependência. É o objeto de questionamento e validação da relação de dependência;
- ***Dependee***: é o ator que tem a responsabilidade de satisfazer a relação de dependência.

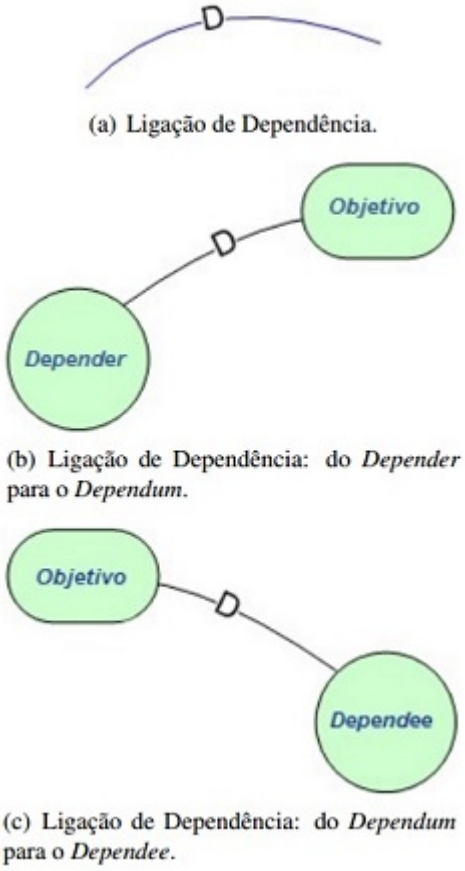
Assim, as relações de dependências podem ser classificadas com base nos seguintes tipos de *Dependum*:

- **Objetivo** (*goal*): é uma declaração de afirmação sobre certo estado do mundo. Deve ser de fácil verificação. O *Dependee* é livre para tomar qualquer decisão para satisfazer o objetivo e é esperado que ele o faça. Não importa para o *Depender* como o *Dependee* irá alcançar esse objetivo;
- **Tarefa** (*task*): é uma atividade a ser realizada pelo *Dependee*. Tarefas podem ser vistas como a realização de operações, processos, etc. Porém, não devem ser uma descrição passo-a-passo ou uma especificação completa de execução de uma rotina;

- **Recurso** (*resource*): é a entidade (física ou informativa) a ser entregue para o *Depender* pelo *Dependee*. Satisfazendo-se esta dependência, o *Depender* está habilitado a usar essa entidade como um recurso;

- **Objetivo-soft** (*softgoal*): é semelhante ao Objetivo, porém os critérios de avaliação e verificação são mais subjetivos. O *Depender* pode decidir sobre o que constitui a realização satisfatória do objetivo;

- **Ligacão de dependência**: é uma conexão direcionada entre dois elementos. No modelo SD, pode-se ter somente duas opções dessa conexão: ou do *Depender* para o *Dependum* ou do *Dependum* para o *Dependee*. Essa ligação é representada por um segmento contínuo e direcionado da origem para o destino, com a letra "D" sobrescrita, conforme exemplos apresentados na Figura 2.4.



(a) Ligação de Dependência.

(b) Ligação de Dependência: do *Depender* para o *Dependum*.

(c) Ligação de Dependência: do *Dependum* para o *Dependee*.

Figura 2.4: Exemplo de ligações de dependência.

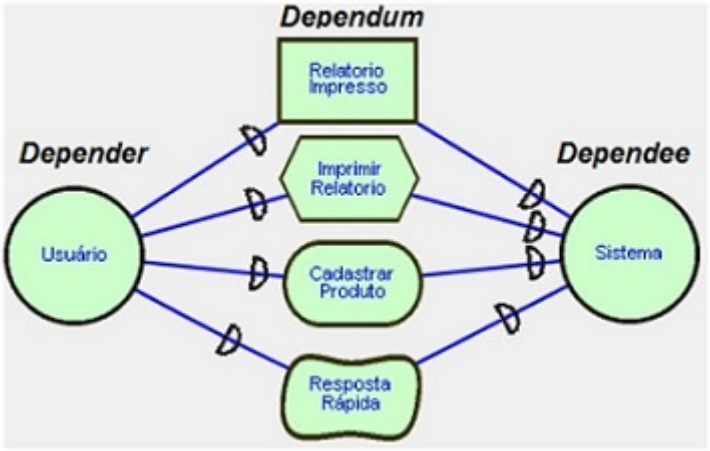


Figura 2.5: Exemplo de Relação de Dependência (*Depender -> Dependum -> Dependee*)

A Figura 2.5 apresenta alguns exemplos de relações de dependências. Nesta figura existem dois atores denominados “Usuário” e “Sistema” que possuem quatro relações de dependências:

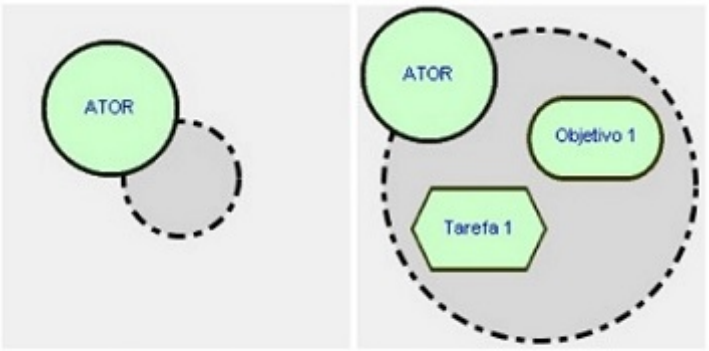
- “Relatório Impresso”: representa um **recurso** concreto de relatórios impressos;
- “Imprimir Relatório”: é uma **tarefa** que o ator “Usuário” depende do ator “Sistema” para realizá-la;
- “Cadastrar Produto”: representa um **objetivo** do ator “Usuário” sobre o ator “Sistema” para que o cadastro de produto seja realizado;
- “Resposta Rápida”: é um **objetivo-soft** cuja satisfação é relativa aos critérios do ator “Usuário”.

2.1.2 Modelos de Razões Estratégicas (SR)

O modelo SR é um complemento ao modelo SD. Ele representa os detalhes das razões internas que estão por trás das dependências entre atores [YU 1995]. Visa retratar os interesses, preocupações e motivações específicas de um ator. Os elementos desse refinamento de motivações internas são agrupados e envolvidos por um limite conhecido como **fronteira** do ator. A seguir são descritos os elementos do modelo SR:

- **Fronteira**: uma fronteira indica os limites intencionais de um determinado ator. Todos os

elementos dentro dos limites de um ator são explicitamente desejos ou pretensões desse ator;



(a) Fronteira Vazia (b) Fronteira com elementos internos

Figura 2.6: Exemplos de fronteira do ator.

- ***Means-end*** (meio-fim): é uma ligação que significa o meio (objetivo, recurso softgoal ou uma tarefa) para atingir um fim (objetivo). Um exemplo desta ligação é mostrado na Figura 2.7;

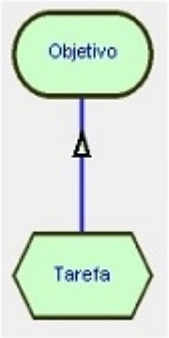


Figura 2.7: Exemplo de ligação meio-fim.

- ***Task-decomposition*** (decomposição de tarefa): é uma ligação responsável por detalhar uma determinada tarefa, através da decomposição em sub-elementos ligados a tarefa principal. Os sub-elementos podem ser: objetivos, tarefas, recursos e objetivos-soft. Um exemplo desta ligação é mostrado na Figura 2.8;

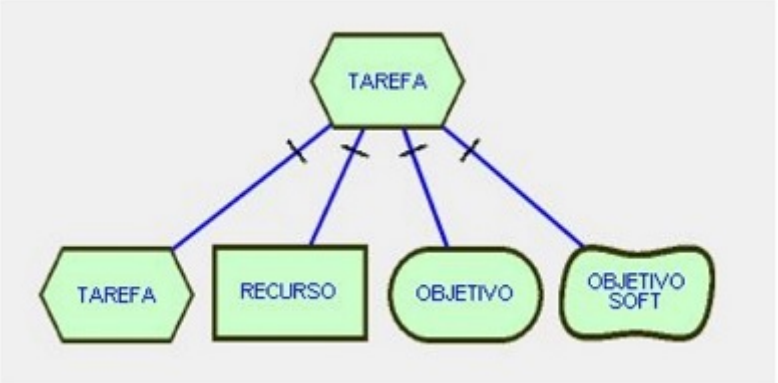


Figura 2.8: Exemplo de ligação de decomposição de tarefa.

- ***Contribution*** (contribuição): é responsável em ligar os elementos do tipo tarefa e objetivo-soft à exclusivamente um objetivo-soft (*softgoal*). Essa ligação ajuda a modelar a forma de como os elementos contribuem para a satisfação desse objetivo-soft. Essas ligações de contribuição, ilustradas na Figura 2.9, podem ser dos tipos:
  - ***Make***: é uma contribuição positiva, suficientemente forte para satisfazer o objetivo-soft;
  - ***Some+***: é uma contribuição positiva, mas sua força de influencia é desconhecida. Pode equivaler a um *make* ou a um *help*;
  - ***Help***: é uma contribuição positiva fraca, pois não é suficiente para que ela sozinha satisfaça o objetivo-soft;
  - ***Unknown***: é uma contribuição cuja influencia é desconhecida;
  - ***Hurt***: é uma contribuição negativa fraca, porem não é suficiente para que ela sozinha recuse a satisfação de um objetivo-soft;
  - ***Some-***: é uma contribuição negativa, mas a força de sua influencia é desconhecida. Pode equivaler a um *hurt* ou a um *break*;
  - ***Break***: é uma contribuição negativa, suficientemente forte para rejeitar a satisfação do objetivo-soft;
  - ***Or***: é uma contribuição onde o objetivo-soft é satisfeito se algum dos descendentes for satisfeitos;

– ***And***: é uma contribuição onde o objetivo-soft é satisfeito se todos os descendentes forem satisfeitos.

Figura 2.9: Ligações de contribuição.

2.2 Utilização do *Framework* i*

O *framework* i* é bastante flexível para representar situações envolvendo interações entre múltiplos participantes. Assim, o mesmo pode ser utilizado para representar muitos contextos de adoção. Por exemplo, apresenta-se a seguir alguns contextos e trabalhos de adoção da modelagem i*:

- **Engenharia de Requisitos**: é uma das áreas de aplicações mais comuns do i*. Esse *framework* é utilizado principalmente na fase inicial do processo de engenharia de requisitos (*Early Requirements*), na qual se argumenta que para obter bons requisitos é preciso compreender as motivações subjacentes do sistema proposto. Pois não é suficiente escrever o que os usuários e clientes dizem que querem, porque muitas vezes esses não são capazes de articular essas necessidades diretamente. O analista precisaria ajuda-los a descobrir suas reais necessidades [YU et al. 2011] [MAIDEN et al. 2004];

- **Modelagem de Negócio**: estudos na área apresentaram o uso do i* para visualização explícita da intencionalidade dos processos de negócios. Isso ajuda a obter um melhor entendimento sobre o trabalho, além de facilitar seu planejamento [YU, MYLOPOULOS e LESPERANCE 1996] [KOLP, GIORGINI e MYLOPOULOS 2003];

- **Desenvolvimento Orientado à Objeto**: alguns trabalhos [CASTRO, ALENCAR e CYSNEIROS 2000] [CASTRO et al. 2001] utilizaram-se da pUML (*precise UML*) [EVANS e KENT 1999] e da *Object Constraint Language* (OCL) [WARMER e KLEPPE 2003] para tratar dos requisitos finais (*Late Requirements*), além de usar o *framework* i* para os requisitos iniciais;

- **Desenvolvimento Orientado à Agentes**: [BRESCIANI et al. 2004] apresentou o uso de agentes com estrutura BDI (*Belive, Desire and Intention*) [RAO e GEORGEFF 1995] para realizar análises na fase inicial de requisitos. Em [BASTOS e CASTRO 2004] foi utilizado Sistemas Multi-Agentes (SMA) para especificar a estrutura organizacional;

- **Segurança, Confiabilidade e Privacidade**: a modelagem i* pode ajudar a lidar com elementos de segurança, confiabilidade e privacidade, através do estudo dos conflitos de intenções de diferentes entidades sociais [YU e LIU 2001].

Segundo [YU et al. 2011], pode-se dizer que o *framework* i* abrange técnicas de modelagem tanto orientadas à agentes quanto orientadas à objetivos, pois sua essência é realizada na combinação de conceitos agentes e objetivos. Ambos os paradigmas, Orientação à Agentes [MAO e YU 2005] e Orientação à Objetivos [LAMSWEERDE 2004], têm apresentado bons resultados em contextos de modelagem organizacional, principalmente em modelagens na fase inicial do processo de engenharia de requisitos.

2.3 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 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 e 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 e CASTRO 2002].

2.3.1 Conceitos e *template* para especificação

Casos de Uso em UML [BOOCH, RUMBAUGH e 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 e 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 (UML) 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".

Os casos de uso de um sistema permitem representar as ações que devem ser realizadas entre o usuário e o sistema para satisfazer o objetivo associado ao caso de uso. Isso permite uma melhor compreensão do caso de uso por todos os envolvidos [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:

Use Case: *<nome> « o nome é um objetivo descrito com uma frase curta contendo um verbo na voz ativa »*

CHARACTERISTIC INFORMATION

Goal in Context: *<uma sentença mais longa do objetivo do caso de uso se for necessário>*

Scope: *<Qual sistema está sendo considerado (por exemplo, organização ou sistema computacional)>*

Preconditions: *<o que é necessário que já esteja satisfeito para realizar o caso de uso>*

Success End Condition: *<o que ocorre/muda após a obtenção do objetivo do caso de uso>*

Failed End Condition: *<o que ocorre/muda se o objetivo é abandonado>*

Primary Actor: *<o nome do papel para o ator primário, ou descrição>*

MAIN SUCESS SCENARIO

<coloque aqui os passos do cenário necessários para a obtenção do objetivo>

<#:> <descrição da ação>

EXTENSIONS

<coloque aqui as extensões, uma por vez, cada uma referenciando o passo associado no cenário principal>

<#:> <ação ou sub.caso de uso>

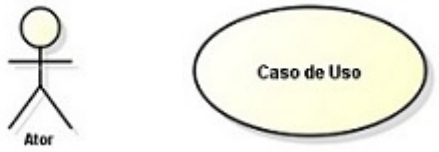
<#:> <ação ou sub.caso de uso>

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*. Podemos citar o Processo Unificado (*Unified Process*) [JACOBSON, BOOCH e RUMBAUGH 1999], o qual adota a descrição de casos de uso como fonte de informações para gerar outros artefatos, entre os quais podemos citar: diagramas de classes, diagramas de sequência bem como descrições arquiteturais do *software*.

2.3.2 Diagrama de Casos de Uso

Um Diagrama de Casos de Uso é composto pelos seguintes componentes: atores, casos de uso e relacionamentos (associação, generalização, dependência do tipo inclusão ou do tipo extensão), os quais já foram conceituados na subseção 2.3.1. As Figuras 2.10, 2.11 e 2.12 apresentam as notações básicas utilizadas para descrever os elementos do Diagrama de Casos de Uso.



(a) Ator (b) Caso de Uso

Figura 2.10: Elementos Ator e Caso de Uso.

(a) Associação (b) Generalização

Figura 2.11: Ligações de Associação e Generalização.

(a) Extensão (b) Inclusão

Figura 2.12: Ligações de Extensão e Inclusão.

A Figura 2.13 exemplifica um Diagrama de Casos Uso com todos os elementos possíveis.

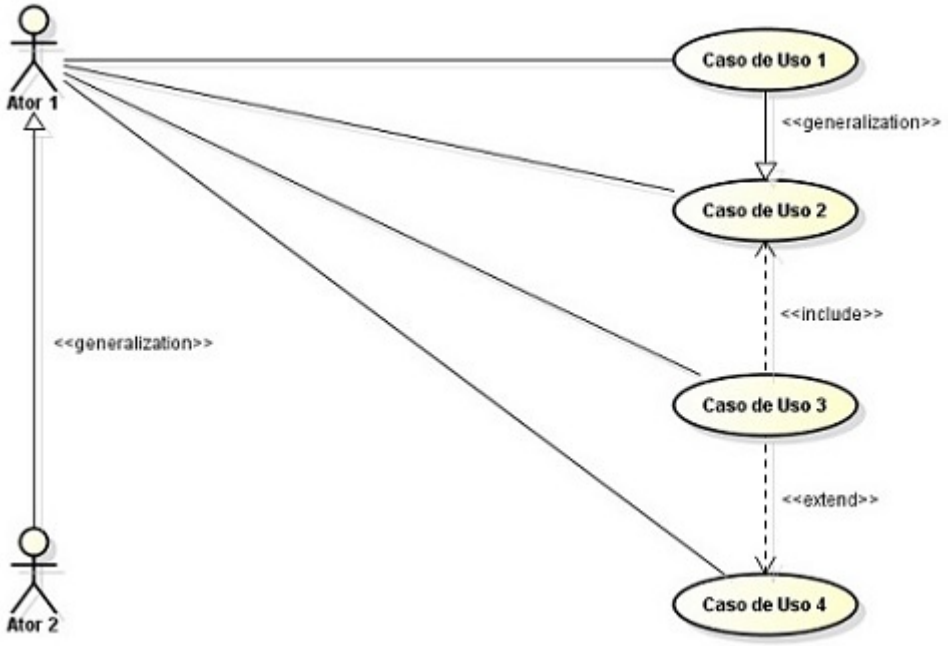


Figura 2.13: Exemplo de Diagrama de Casos de Uso

Na Figura 2.13, o Ator 2, é uma especialização do Ator 1. Neste caso, o Ator 2 herda toda a estrutura e comportamento do Ator 1 e pode evoluir a estrutura e comportamento em relação ao Ator 1. O Ator 1 possui os Casos de Uso 1, 2, 3 e 4 associados a ele, ou seja, ele interage com o sistema buscando atingir quatro diferentes objetivos. O Caso de Uso 3 possui o Caso de Uso 2 incluído, ou seja, o Caso de Uso 2 é uma etapa obrigatória para se realizar o Caso de Uso 3. O Caso de Uso 4 também possui o Caso de Uso 3 extendido, ou seja, o Caso de Uso 3 é uma etapa opcional para se realizar o Caso de Uso 4.

A construção de casos de uso normalmente é afetada pela experiência e subjetividade do engenheiro de requisitos. Assim, procura-se sempre a melhor abordagem para especificação de casos de uso de modo que essa experiência e subjetividade dos profissionais não interfiram. Na literatura da área, existem várias abordagens para esse fim, mas poucas apresentam diretrizes para ajudar nessa tarefa. Por exemplo, no trabalho apresentado em [KULAK e GUINEY 2000] é sugerido uma forma iterativa de construir casos de uso, a qual é composta de quatro iterações pré-definidas. No entanto, não são fornecidas diretrizes para identificar e especificar os casos de uso. Em [SCHNEIDER e WINTERS 2001] é fornecido algumas questões para ajudar a identificar atores e algumas sugestões para criar casos de uso do tipo CRUD (*create, read,*

update e delete). Mas assim como em outros trabalhos, não é apresentado diretrizes que ajudem a escrever a especificação dos casos de uso.

Visando preencher esta lacuna, na próxima seção é apresentada uma proposta de derivação de casos de uso guiados por diretrizes a partir dos modelos organizacionais construídos via i* [SANTANDER e CASTRO 2002]. A ideia não é gerar todos os casos de uso para um sistema computacional, mas sim definir aqueles considerados essenciais a partir dos quais pode-se evoluir na descoberta de outros casos de uso necessários.

2.4 Integração do *framework* i* com Casos de Uso UML

É cada vez 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*, para que exista um entendimento do contexto em que o sistema irá funcionar [ERIKSSON e PENKER 1998].

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 2002].

Abordagens baseadas em cenários tem se destacado pela facilidade de entendimento dos usuários e desenvolvedores do sistema [BREITMAN e LEITE 1998]. Essas abordagens têm ajudado 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 [POTTS 1999] [SUTCLIFFE e GREGORIADES 2002]. Assim, surge o *framework* i* juntamente da técnica de casos de uso, a qual se destaca por ser importante na Linguagem de Modelagem Unificada (UML), para elicitar e especificar requisitos de forma mais completa.

Neste sentido, a proposta apresentada em [SANTANDER e CASTRO 2002] propõe uma abordagem que considera a construção de modelos organizacionais via *framework* i* e posterior derivação de casos de uso em UML a partir destes modelos. Para este fim, é proposto um processo guiado por diretrizes que auxiliam engenheiros de requisitos na utilização da abordagem. Entre algumas vantagens deste processo de integração que podemos destacar é a possibilidade de derivar casos de uso com base nas intencionalidades associadas aos atores no ambiente

organizacional, bem como compreender o ambiente para elicitar requisitos funcionais e não funcionais do sistema computacional pretendido.

A Figura 2.14 apresenta uma visão geral do mapeamento de modelos i* para casos de uso UML.

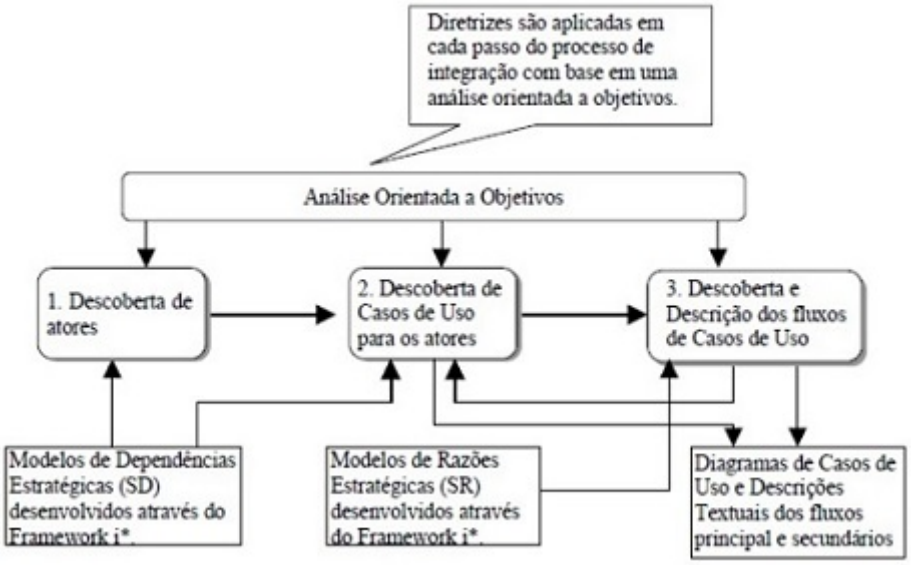


Figura 2.14: Visão geral do mapeamento de modelos i* para casos de uso [SANTANDER e CASTRO 2002].

Os passos 1, 2 e 3 representam a descoberta de atores do sistema e seus casos de uso associados juntamente com suas descrições. A entrada para o processo de integração são os modelos SD e SR desenvolvidos através do *framework* i*. Nas etapas 1 e 2, a entrada é o modelo SD. A descrição de cenários para casos de uso, que corresponde a etapa 3, é derivada de elementos representados no modelo SR. Os resultados do processo de mapeamento são diagramas de casos de uso para o sistema pretendido bem como as descrições textuais de cenários para cada caso de uso mapeado.

A seguir as diretrizes propostas por [SANTANDER e CASTRO 2002] para derivar casos de uso em UML a partir de i* são apresentadas. Cabe ressaltar que a última versão destas diretrizes está disponível em [YU et al. 2011].

- **Passo 1:** descobrir os atores do sistema.
  - **Diretriz 1:** todo ator em i* é um **candidato** a ser mapeado para um ator em caso de

uso.

- **Diretriz 2:** o ator **candidato** i* deve ser externo ao sistema computacional pretendido. Isso implica em que atores que representam o sistema computacional, ou partes do mesmo, não são candidatos a atores de casos de uso.
- **Diretriz 3:** o ator **candidato** i* deve ter pelo menos uma dependência com o sistema computacional pretendido. Caso contrário, esse ator não pode ser mapeado para um ator de caso de uso.
- **Diretriz 4:** atores em i*, relacionados através da associação ISA e mapeados individualmente para atores em casos de uso (após aplicação das diretrizes 1, 2 e 3), serão associados no diagrama de casos de uso através do relacionamento do tipo "generalização".

- **Passo 2:** descobrir casos de uso para os atores.

  - **Diretriz 5:** para cada ator do sistema descoberto no Passo 1, deve-se analisar todas as dependências entre o sistema pretendido e esse ator, na qual esse ator é o *dependee* da relação de dependência, buscando casos de uso para esse ator.
    - **Subdiretriz 5.1:** as dependências do tipo **objetivo** podem ser mapeadas diretamente para casos de uso.
    - **Subdiretriz 5.2:** as dependências do tipo **tarefa** podem ser mapeadas diretamente para casos de uso.
    - **Subdiretriz 5.3:** as dependências do tipo **recurso** devem ser analisadas com o seguinte questionamento: "porque este recurso é requerido?". Se para esta resposta existir um objetivo, esse objetivo será candidato a ser um caso de uso para este ator.
    - **Subdiretriz 5.4:** as dependências do tipo **objetivo-soft** normalmente são requisitos não-funcionais associadas ao sistema pretendido. Portanto, um **objetivo-soft** não representa um caso de uso do sistema, e sim um requisito não-funcional de um caso de uso especifico ou do sistema como um todo.

- **Diretriz 6:** analisar situações especiais na qual um ator (descoberto no Passo 1) possui dependências (como *depender*) em relação ao ator no modelo i* que representa o sistema pretendido ou parte dele (Ator -> *Dependum* -> Sistema Pretendido).
- **Diretriz 7:** classificar cada caso de uso de acordo com seu tipo de objetivo associado: de negócio, contextual, de usuário ou subfunção.

• **Passo 3:** descobrir e determinar cenários dos casos de uso.

- **Diretriz 8:** analisar cada ator e seus relacionamentos no modelo SR para extrair informações que possam conduzir à descrição do cenário do caso de uso para o ator. É importante ressaltar que os diagramas SR representam as razoes internas associadas aos objetivos do ator. Por isso, deve-se considerar os elementos internos que são usados para o ator conquistar os objetivos e objetivos-soft, realizar as tarefas e obter os recursos. O ator possui a responsabilidade de satisfazer esses elementos. A decomposição em um diagrama SR mostra como o ator irá fazer isso. Normalmente, as dependências associadas ao ator são satisfeitas internamente por meio de dois tipos de relacionamentos usados no SR: meio-fim e decomposição de tarefa. Deve-se observar esses relacionamentos a fim de obter os passos dos cenários dos casos de uso. Os subcomponentes em relações de composição de tarefa normalmente são mapeados para passos (atividades) do cenário de caso de uso associado à tarefa. Note que se a tarefa que esta sendo decomposta cumpre alguma dependência (com outros atores) previamente mapeada para um caso de uso, os subcomponentes são mapeados para atividades (passos) do cenário principal do caso de uso. Por outro lado, em uma relação meio-fim, os meios representam alternativas para atingir um fim. Este fim pode ser um **objetivo** a ser alcançado, uma **tarefa** a ser realizada, um **recurso** a ser produzido, ou um **objetivo-soft** a ser satisfeito. Se este fim é um objetivo ou tarefa que cumpre alguma dependência previamente mapeada para um caso de uso, essas alternativas (meios) são descritas como extensões do cenário de um caso de uso (« *extend*», mecanismo de estruturação UML). Além disso, também é permitido associar objetivos-soft representados no diagrama SR com casos de uso, se um subcomponente em uma relação de decomposição de tarefa é um objetivo-soft

e a decomposição de tarefa cumpre alguma dependência mapeada para um caso de uso, este objetivo-soft deve ser associado com o caso de uso como um requisito não funcional no cenário primário.

- **Diretriz 9:** cada caso de uso deve ser analisado a fim de refinar e derivar novos casos de uso a partir da observação dos cenários.
- **Diretriz 10:** construir o diagrama de casos de uso utilizando os casos de uso e atores descobertos, bem como os seguintes relacionamentos de casos de uso UML: *include*, *extend* e *generalization*.

Essas são as diretrizes que possibilitam a derivação de casos de uso a partir dos modelos construídos via *framework* i*. Maiores informações podem ser obtidas no capítulo IV do livro "*Social Modeling for Requirements Engineering*" [YU et al. 2011] ou no artigo "*Deriving Use Cases from Organizational Modeling*" [SANTANDER e CASTRO 2002].

2.5 Considerações Finais do Capítulo

Neste capítulo foram apresentados os conceitos gerais sobre *framework* i* e casos de uso UML bem como a proposta de integração entre essas duas abordagens, a qual é apoiada por diretrizes específicas. A importância do estudo desses conceitos é justificável para o desenvolvimento deste trabalho, pois os modelos SD e SR do *framework* i* são a base para gerar casos de uso UML segundo as diretrizes propostas em [SANTANDER e CASTRO 2002] e com suporte computacional provido pela ferramenta JGOOSE, a qual será apresentada no próximo capítulo juntamente com a ferramenta E4J i*. Essas duas ferramentas provêm suporte para geração de casos de uso a partir de modelos i* e para construção de modelos i*, respectivamente.

Capítulo 3

JGOOSE e E4J i*

Neste capítulo, é apresentada a ferramenta JGOOSE (*Java Goal Into Object Oriented Standard Extension*), a qual teve seu desenvolvimento motivado pela possibilidade de integração de técnicas da engenharia de *software* para modelagem de sistemas computacionais. Também é apresentada a ferramenta E4J (*Editor for JGOOSE*) i* (i-star), a qual está integrada ao JGOOSE e foi desenvolvida com o objetivo de eliminar a dependência do JGOOSE para com outras ferramentas para criar modelos i*.

Desta forma, a seção 3.1 apresenta a ferramenta JGOOSE e suas versões ao longo dos anos. Esta seção também demonstra seu uso e apresenta o projeto e arquitetura da mesma. Na seção 3.2 é apresentada a ferramenta E4J i* e a adaptação feita pelo JGOOSE para realizar a integração do E4J i* bem como é apresentado e comentado sobre a interface gráfica do E4J i*. Cabe ressaltar que essa interface será utilizada como base para a construção do E4J Use Cases, a ser apresentado no próximo capítulo. Por fim, na seção 3.3, são feitas as considerações finais deste capítulo.

3.1 JGOOSE

O JGOOSE é uma ferramenta de auxílio no mapeamento de modelos organizacionais para modelos funcionais [VICENTE 2006]. Ele implementa seus processos guiados pelas diretrizes propostas por [SANTANDER 2002] (ver capítulo 2, seção 2.4) 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] (ver capítulo 2, seção 2.2). Desta maneira, a ferramenta permite derivar casos de uso com base nas intencionalidades

associadas aos atores de um ambiente organizacional.

Na subseção seguinte (3.1.1) será apresentado um resumo sobre as principais mudanças já realizadas na ferramenta JGOOSE.

3.1.1 Versões da ferramenta

Desde a primeira versão [VICENTE 2006], a ferramenta JGOOSE passou por várias melhorias e aprimoramentos. Estas mudanças têm variado desde a refatoração do código fonte (classes e *packages* Java) até alterações na interface gráfica do usuário [BRISCHKE 2012]. A seguir, é apresentado um resumo sobre a origem do JGOOSE e as principais alterações já realizadas na ferramenta.

- **GOOSE (*Goal into Object Oriented Standard Extension*)**: foi a ferramenta que antecedeu à JGOOSE. Foi implementada na linguagem *Rational Rose Scripting* por [BRISCHKE 2005] como uma extensão da *Rational Rose*. Além da dependência da *Rational Rose*, existia também a dependência da ferramenta OME (*Organizational Modeling Environment*) para criar o modelo i* e o arquivo telos;

- **JGOOSE versão 2006**: desenvolvido por [VICENTE 2006], essa nova implementação passou a ser na linguagem Java e foi atribuído o nome de JGOOSE (*Java Goal into Object Oriented Standard Extension*). Por ser em uma nova linguagem de programação, todo o projeto teve que ser re-implementado em Java. Entre outros aspectos, a solução continuou dependente da ferramenta OME, contudo não utilizada mais a solução proprietária *Rational Rose*;

- **JGOOSE versão 2011**: melhorada por [BRISCHKE 2012], a nova versão contemplava a implementação de três diretrizes faltantes nas versões anteriores: 8, 9 e 10. Também foi fruto desse trabalho, a implementação da exportação dos casos de uso no formato XMI [JECKLE 2004], melhorando a comunicação com outras ferramentas como a StarUML [WONG 2007]. Nessa versão, foram implementadas soluções que permitiram o usuário da ferramenta realizar um refinamento manual dos casos de uso gerados, bem como visualizar graficamente os casos de uso na forma digramas de casos de uso estáticos. A interface principal dessa versão é apresentada na Figura 3.1;

- **JGOOSE versão 2013**: melhorada por [PELISER, SANTANDER e MERLIM 2013], essa nova versão corrigiu erros e bugs na aplicação das diretrizes bem como foi realizado uma otimização do código fonte. Também foi acrescentado a essa versão novos elementos possíveis de serem definidos nos modelos organizacionais SD e SR em i*, tais como os tipos de ligações: *INS*, *Plays*, *Occupies*, *Covers*, *Is-Part-Of* e *Contribution*. E também os novos tipos de atores: *Agent*, *Role* e *Position*. Essa versão também apresenta uma nova interface gráfica, exibida na Figura 3.2, que foi desenvolvida com base em conceitos de Interação Humano-Computador [BARBOSA e SILVA 2010]. Por fim, também foi adicionado duas novas funcionalidades que permitem excluir um caso de uso gerado e a opção para salvar todos os casos de uso gerados em formato *.doc*. Paralelamente às melhorias realizadas nessa versão, também foi desenvolvido um editor de modelos i* denominado E4J i* o qual está integrado ao JGOOSE. A ferramenta E4J i* será apresentada na seção 3.5.

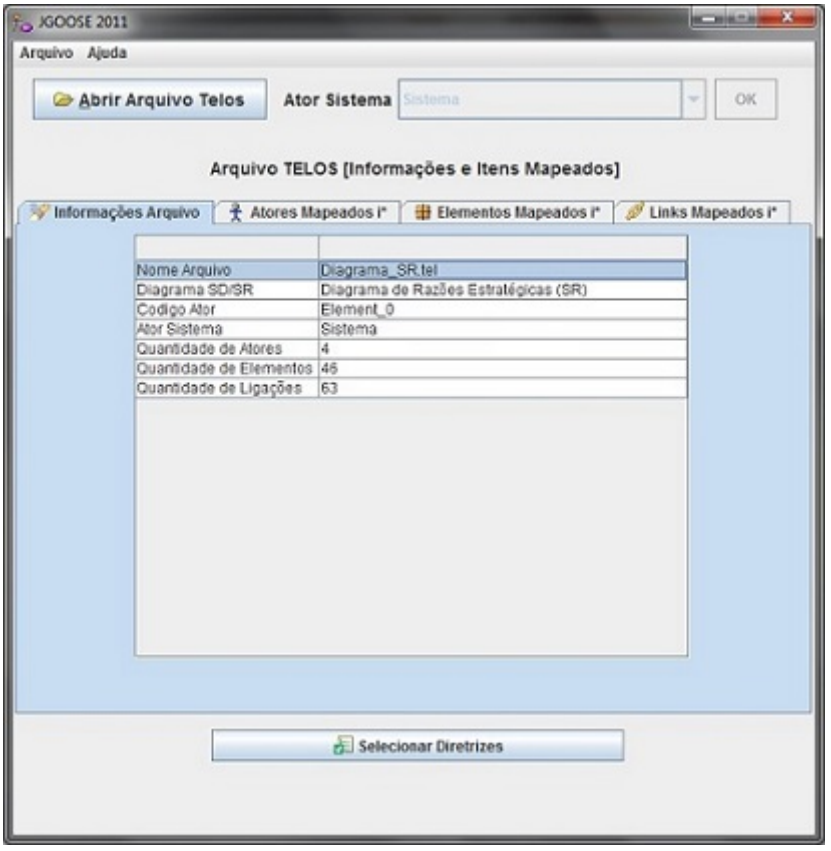


Figura 3.1: Interface gráfica da ferramenta JGOOSE versão 2011.

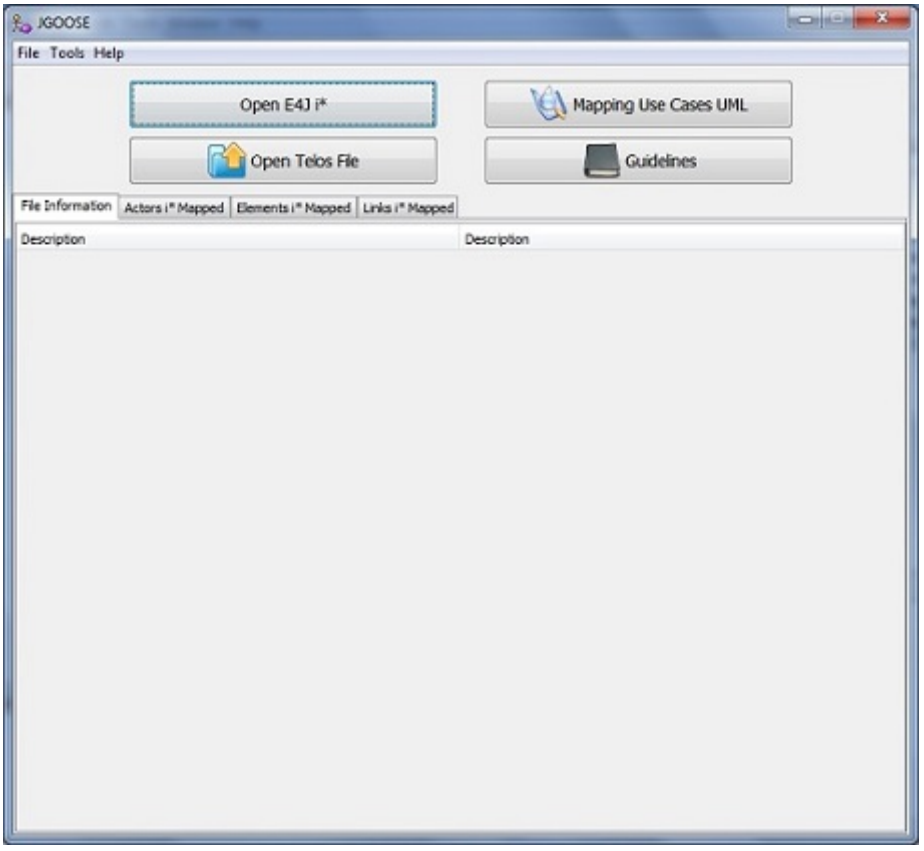


Figura 3.2: Interface gráfica da ferramenta JGOOSE versão 2013.

3.1.2 Demostrando o uso da ferramenta

Nesta subseção será demonstrado o uso da ferramenta JGOOSE no desenvolvimento deste projeto, utilizando a mesma para gerar os casos de uso essenciais que devem ser cobertos pelo editor de diagrama de casos de uso (E4J Use Cases) proposto neste trabalho. Assim, seguindo as diretrizes apresentadas no capítulo 2 (seção 2.3), inicialmente, foram criados os modelos organizacionais SD e SR para o E4J Use Cases, os quais são apresentados nas Figuras 3.3 e 3.4, respectivamente.

A partir desse modelo SD (Figura 3.3), podem-se obter as seguintes informações sobre os atores:

- *E4J Use Cases*: ator que representa o sistema. O desenvolvimento deste editor e integração do mesmo ao JGOOSE é o foco principal deste trabalho;
- *JGOOSE*: após a integração do editor E4J Use Cases, o JGOOSE dependerá do *E4J Use*

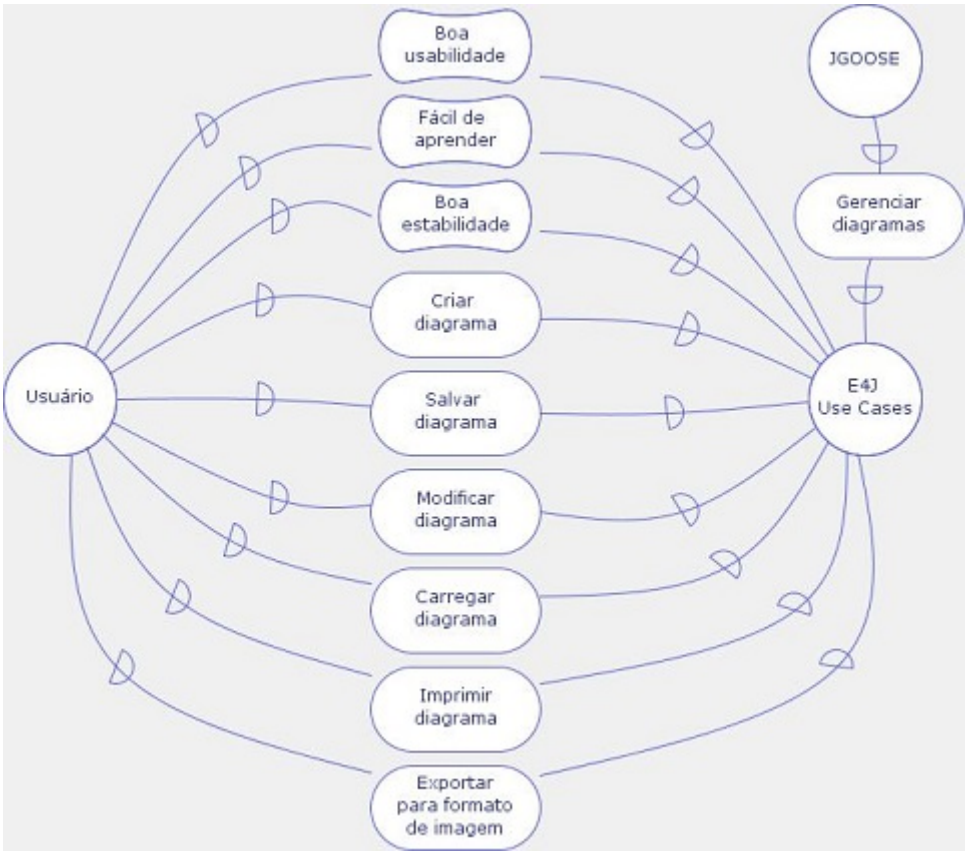


Figura 3.3: Modelo SD da ferramenta E4J Use Cases.

Cases para satisfazer o objetivo de *Gerenciar diagramas*;

- *Usuário*: é o ator que utilizará o E4J Use Cases. Segundo o modelo, este ator depende do *E4J Use Cases* para atingir os objetivos *Criar diagrama*, *Salvar diagrama*, *Modificar diagrama*, *Carregar diagrama*, *Imprimir diagrama* e *Exportar para formato de imagem*. Este ator também deseja que a ferramenta: apresente uma *Boa usabilidade*, seja *Fácil de aprender* e tenha uma *Boa usabilidade*;

- *Criar diagrama*: o ator *Usuário* deseja *Criar diagrama* (objetivo) utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Salvar diagrama*: o ator *Usuário* deseja *Salvar diagrama* (objetivo) utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Modificar diagrama*: o ator *Usuário* deseja *Modificar diagrama* (objetivo) utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Carregar diagrama*: o ator *Usuário* deseja *Carregar diagrama* (objetivo) utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Imprimir diagrama*: o ator *Usuário* deseja *Imprimir diagrama* (objetivo) utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Exportar para formato de imagem*: o ator *Usuário* deseja *Exportar para formato de imagem* utilizando o sistema (*E4J Use Cases*). Este objetivo será mapeado para um caso de uso conforme as diretrizes 5 e 5.1;

- *Boa usabilidade*: o ator *Usuário* espera que seja um editor que facilite algumas tarefas como: copiar e colar, arrastar e soltar, etc. Este objetivo-soft será mapeado para um requisito não-funcional conforme as diretrizes 5 e 5.4;

- *Fácil de aprender*: o ator *Usuário* espera que editor possua ícones e menus intuitivos. Este objetivo-soft será mapeado para um requisito não-funcional conforme as diretrizes 5 e 5.4;

- *Boa estabilidade*: o ator *Usuário* espera que o editor se mantenha estável (não "trave") quando uma modelagem é realizada. Este objetivo-soft será mapeado para um requisito não-funcional conforme a diretriz 5.4.

O modelo SR, apresentado na Figura 3.4, contém o detalhamento do editor (ator *E4J Use Cases*) e, deste ator, podemos extrair as seguintes informações:

- *Criar diagrama*: esta tarefa reflete o objetivo *Criar diagrama* especificado no SD. Ela foi decomposta em: *Inicializar estrutura do grafo* (tarefa) e *Salvar diagrama* (objetivo). Conforme as diretrizes 5, 5.1 e 8, os elementos da decomposição são mapeados para o cenário principal do caso de uso gerado;

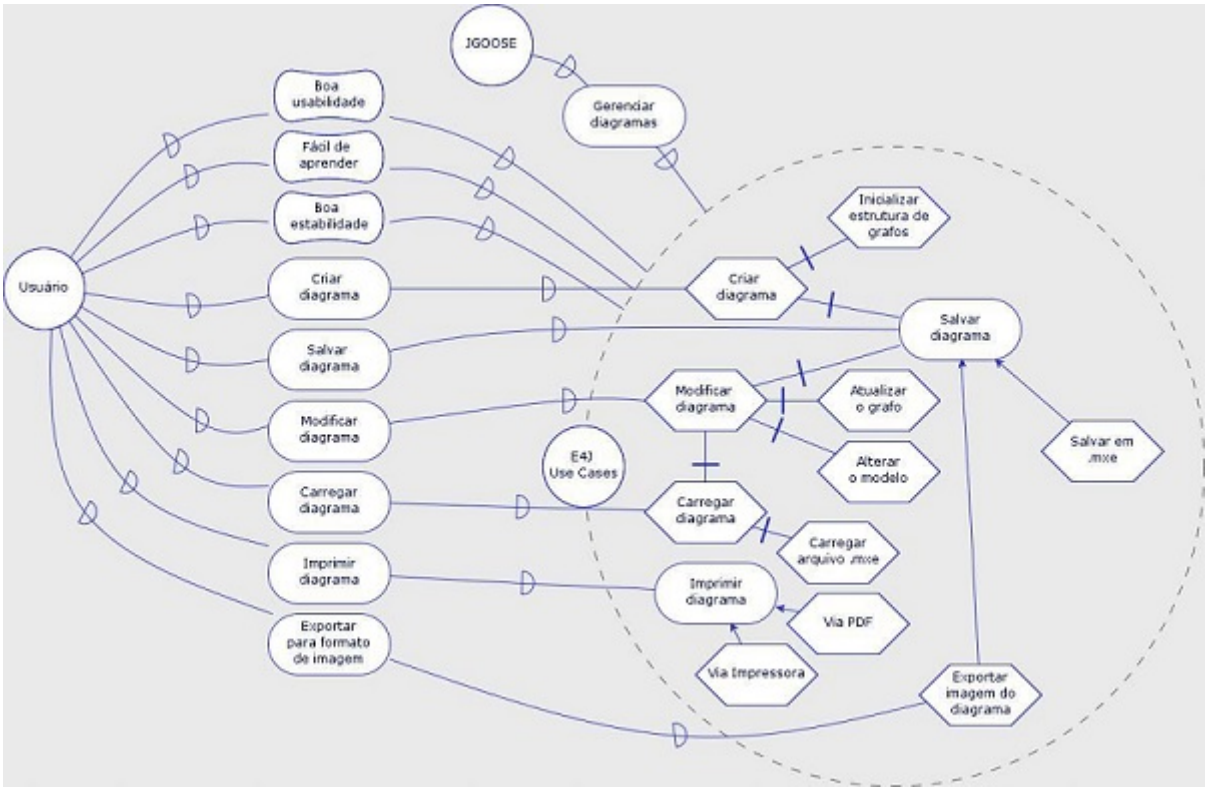


Figura 3.4: Modelo SR da ferramenta E4J Use Cases.

- *Salvar diagrama*: este objetivo reflete o objetivo *Salvar diagrama* especificado no SD e pode ser atingido realizando umas das duas tarefas *Salvar em .mxe* ou *Exportar imagem do diagrama*;

- *Modificar diagrama*: esta tarefa reflete o objetivo *Modificar diagrama* especificado no SD e foi decomposta em: *Atualizar o grafo* (tarefa), *Alterar o modelo* (tarefa) e *Salvar diagrama* (objetivo). Conforme as diretrizes 5, 5.1 e 8, os elementos da decomposição são mapeados para o cenário principal do caso de uso gerado;

- *Carregar diagrama*: esta tarefa reflete o objetivo *Carregar diagrama* especificado no SD e foi decomposta em: *Carregar arquivo .mxe* (tarefa) e *Modificar diagrama* (tarefa). Conforme as diretrizes 5, 5.1 e 8, os elementos da decomposição são mapeados para o cenário principal do caso de uso gerado;

- *Imprimir diagrama*: este objetivo reflete o objetivo *Imprimir diagrama* especificado no SD e pode ser atingido através de uma das duas tarefas *Via PDF* ou *Via Impressora*;

- *Exportar imagem do diagrama*: esta tarefa reflete o objetivo *Exportar para formato de imagem* especificado no SD.

Cabe ressaltar a importância do objetivo *Salvar diagrama* do ator *E4J Use Cases*, o qual é sub-elemento (via mecanismo de decomposição de tarefas) das tarefas *Criar diagrama* e *Modificar diagrama*. Isso impacta na geração do estereótipo *« include »* nos casos de uso *Criar diagrama* e *Modificar diagrama* incluindo o objetivo *Salvar diagrama*.

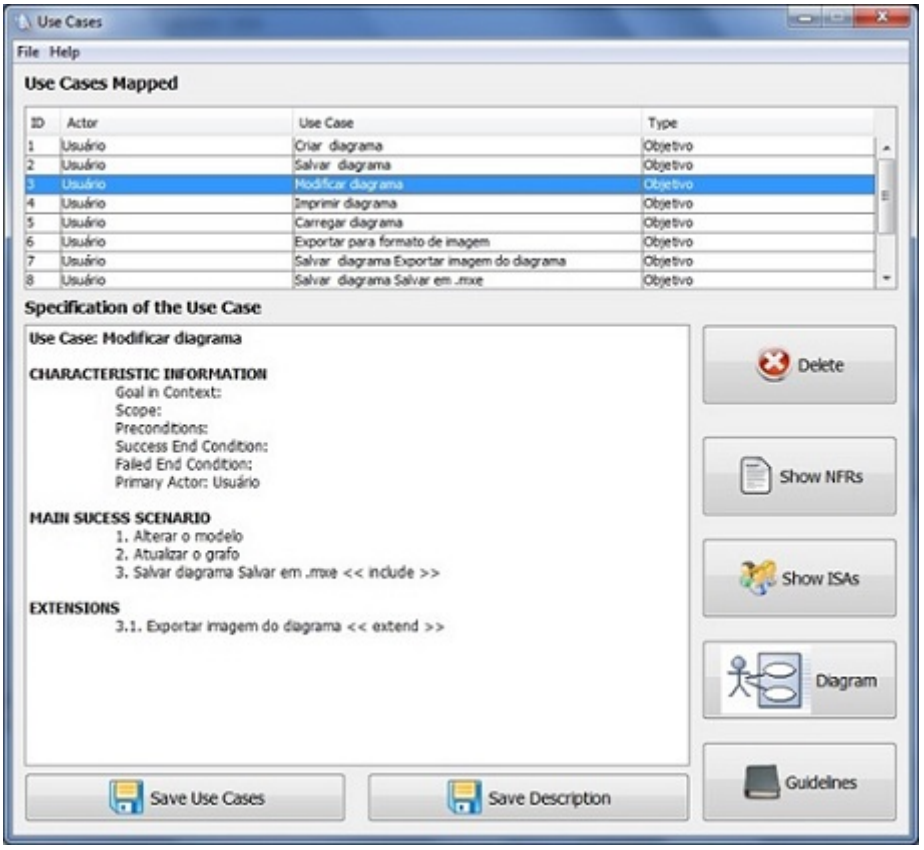


Figura 3.5: Caso de Uso Modificar diagrama.

O diagrama SR, que possui informações detalhadas da modelagem, foi utilizado como entrada na ferramenta JGOOSE para geração dos casos de uso e dos requisitos não-funcionais do editor (Figura 3.6). Explica-se a seguir o caso de uso *Modificar diagrama*, presente na Figura 3.5, o qual possui os estereótipos *« include »* e *« extend »*, a fim de melhor entender esses tipos de ligações.

Esse caso de uso possui três etapas para o cenário principal de sucesso (veja Figura 3.5), as quais são derivadas dos sub-elementos que decompõe a tarefa *Modificar diagrama* (veja Figura

3.4): *1. Alterar o modelo*, *2. Atualizar o grafo* e *3. Salvar diagrama Salvar em .mxe « include »*. Podemos observar que há um estereótipo *« include »* em *3. Salvar diagrama Salvar em .mxe « include »*, isso ocorre devido ao objetivo *Salvar diagrama* ser um caso de uso por si só. Esse *« include »* indica que para se realizar o caso de uso *Modificar diagrama* é necessário também realizar o caso de uso *Salvar diagrama*. Observa-se também que existe a extensão *Exportar imagem do diagrama « extend »* com o estereótipo *« extend »* para a etapa 3. Isso indica que a etapa *3. Salvar Diagrama Salvar em .mxe « include »* pode ser realizada através da tarefa *Salvar em .mxe* já incluída nessa etapa, ou através da extensão *Exportar imagem do diagrama « extend »* (o qual reflete o caso de uso *Exportar para formato de imagem*) para essa mesma etapa, mostrando assim um caminho alternativo para realizar essa atividade.

A partir dos casos de uso gerados, também foi criado o diagrama de casos de uso do editor E4J Use Cases conforme apresentado na Figura 3.7.

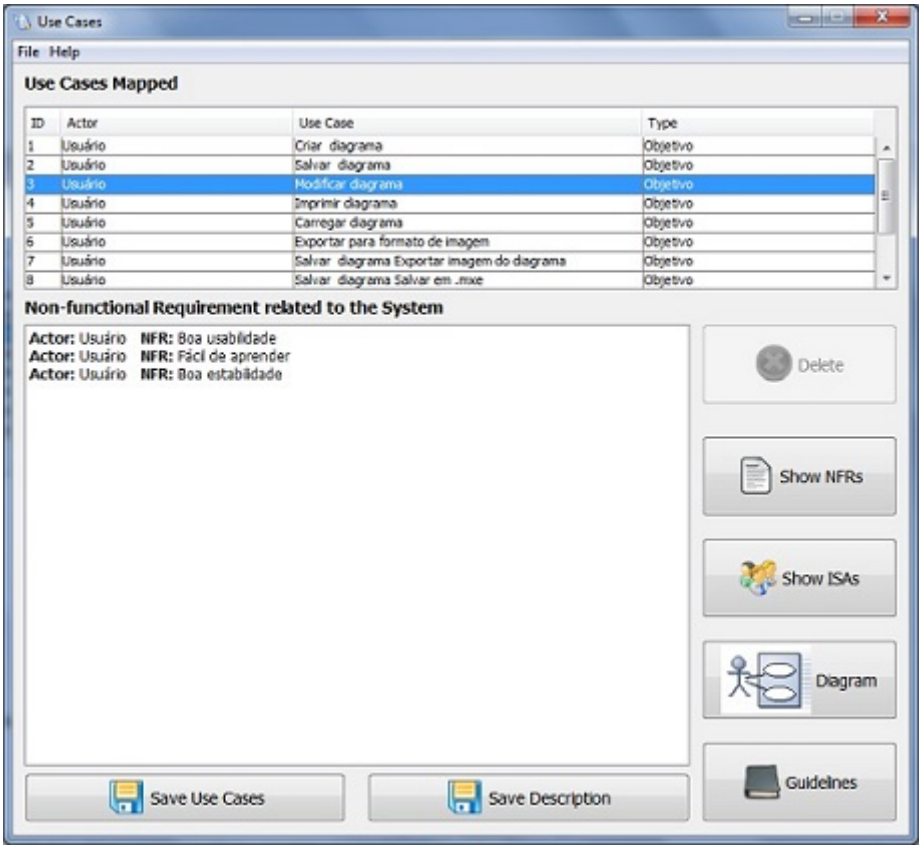


Figura 3.6: Requisitos Não-Funcionais do Editor E4J Use Cases.

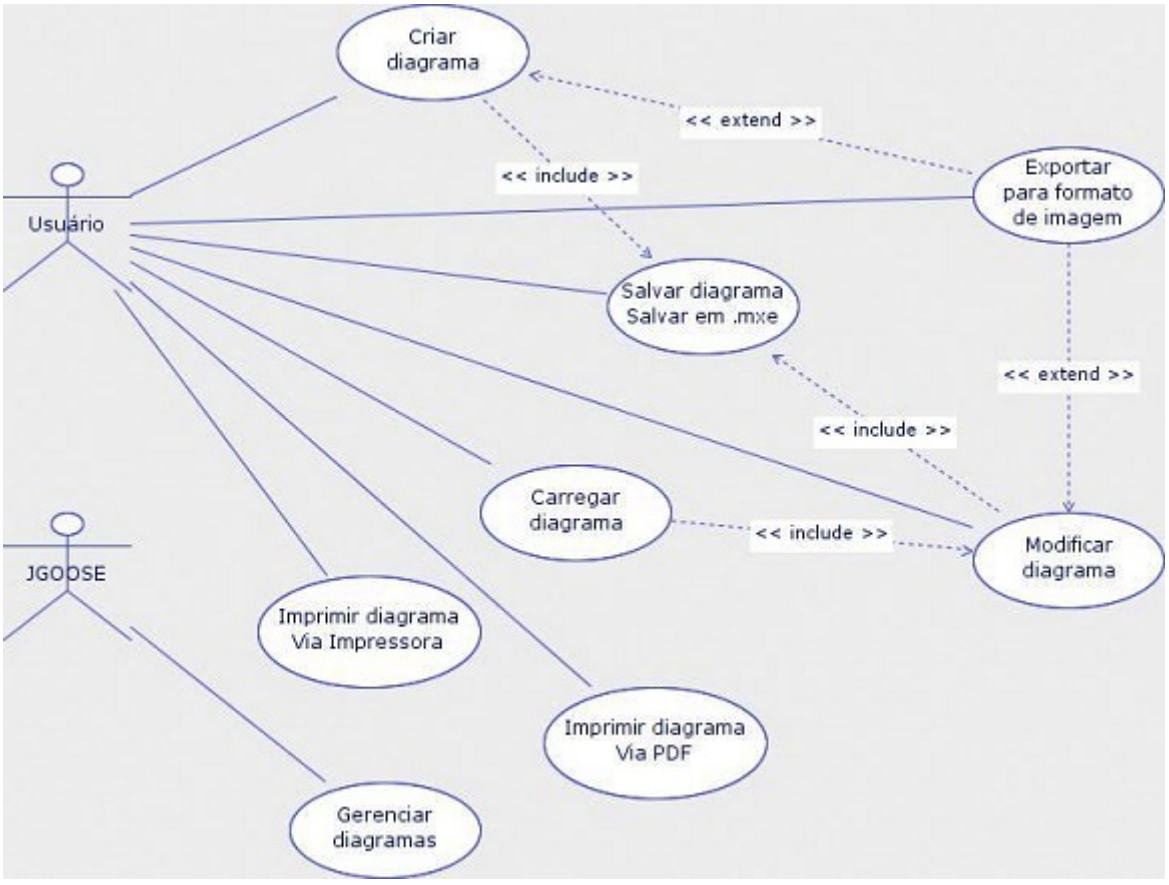


Figura 3.7: Diagrama de Casos de Uso do Editor E4J Use Cases.

3.1.3 Projeto e Arquitetura

Conforme visto anteriormente, o JGOOSE sofreu várias alterações ao longo dos anos. Entretanto, nenhumas dessas alterações influenciaram de forma significativa nos processos tradicionais da ferramenta, mantendo como base as diretrizes e passos propostos em [SANTANDER 2002]. Desta forma, apresenta-se na Figura 3.8 o diagrama de atividades da ferramenta JGOOSE com o E4J i* integrado, para melhor expressar alguns comportamentos do JGOOSE. O diagrama de atividades UML tem como foco principal a representação gráfica de modelos que coordenam as sequências e condições de comportamentos de um sistema [BOOCH, RUMBAUGH e JACOBSON 2005]. Também são chamados de fluxo de controle ou de modelo de fluxo de objetivos. É uma abstração em alto nível que permite diferentes fluxos de execuções e controles simultâneos, assim como é possível realizar o sincronismo desses fluxos e garantir que as atividades executarão em uma ordem específica [GROUP 2007].

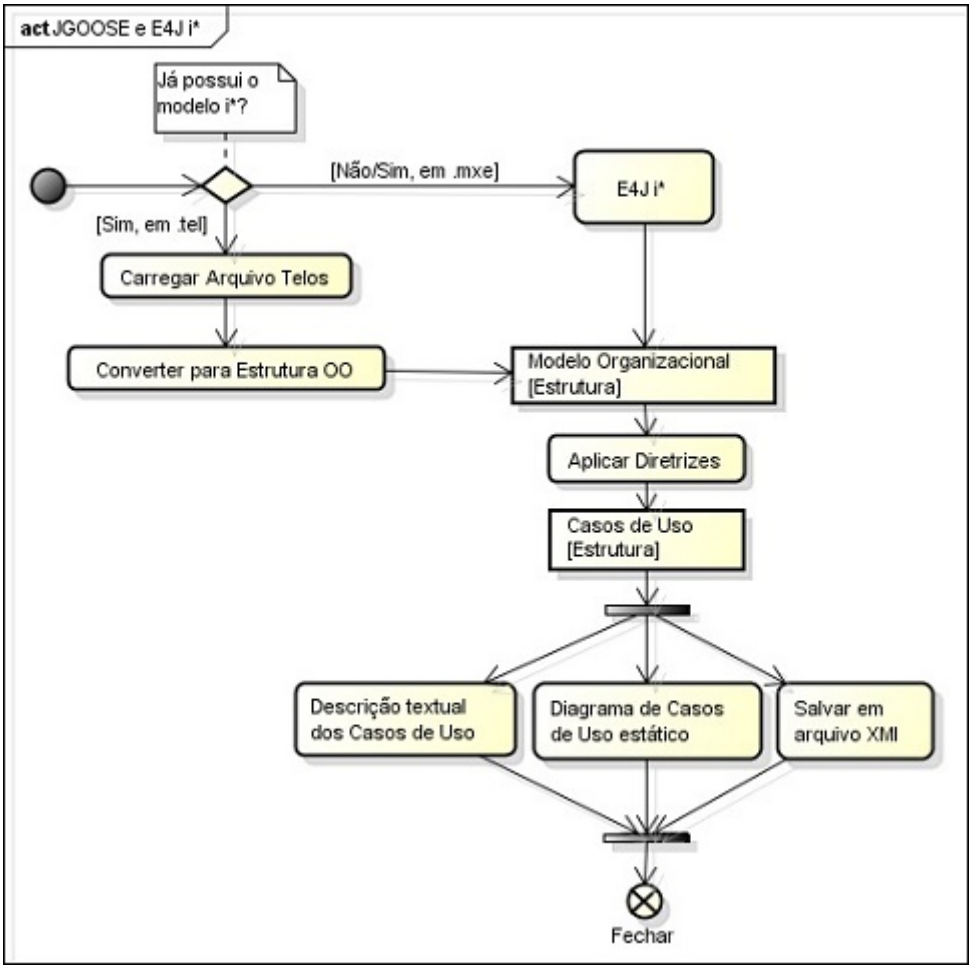


Figura 3.8: Diagrama de atividades da ferramenta JGOOSE com E4J i*.

Inicialmente, tem-se um elemento de decisão para verificar se o usuário já possui um modelo i* em um arquivo com extensão ".tel" ou ".mxe". Dependendo da resposta do usuário, ele segue para as etapas consequentes do diagrama descritas a seguir:

- **E4J i***: se o usuário possui um arquivo com extensão em ".mxe" ou não possui um modelo i*, ele pode abrir a ferramenta E4J i* e abrir ou criar seu modelo. Posteriormente, ele pode chamar a função para gerar casos de uso com a ferramenta JGOOSE;

- **Carregar Arquivo Telos**: o usuário será solicitado para escolher um único arquivo com extensão ".tel". O JGOOSE irá ler este arquivo e interpretar como uma única cadeia de caracteres;

- **Converter para Estrutura OO**: nesta etapa, o JGOOSE irá interpretar a cadeia de carac-

teres e criará os objetos correspondentes as informações contidas no arquivo;

- **Modelo Organizacional (Estrutura JGOOSE)**: representa a estrutura orientada à objetos do JGOOSE criada para manipulação do modelo organizacional. Mais especificamente, esta estrutura é composta por: uma lista contendo os atores, agentes, posições e papeis; uma lista para cada tipo de relacionamento, por exemplo, uma lista chamada "*decompositions*" armazena todas as relações de decomposição de tarefa de um modelo;

- **Aplicar Direitrizes**: nesta etapa, o JGOOSE aplica as diretrizes com base no modelo organizacional (estrutura anterior) para gerar os casos de uso (próxima estrutura);

- **Casos de Uso (Estrutura)**: representa a estrutura orientada a objetos do JGOOSE criada para manipulação de casos de uso. Basicamente, esta estrutura engloba uma lista de casos de uso, uma lista de atores e uma lista de ligações e extensões;

- **Apresentar Diagrama de Casos de Uso esático**: nesta atividade o JGOOSE mostra ao usuário uma imagem estática dos casos de uso mapeados;

- **Descrição textual dos Casos de Uso**: conforme o *template* apresentado no capítulo 2, seção 2.2, os casos de uso são apresentados nesta atividade;

- **Salvar em arquivo XMI**: ação realizada pelo usuário para gravar a estrutura do diagrama de casos de uso mapeados para um arquivo XMI.

Destaca-se a etapa "**Apresentar Diagrama de Casos de Uso estático**", pois nesse diagrama não é possível realizar qualquer modificação. Caso o usuário deseja fazer uma modificação, o mesmo deve exportar o arquivo para XMI. Esse arquivo é suportado somente pela ferramenta StarUML [WONG 2007], não sendo um padrão que possa ser importado por todas as ferramentas de manipulação de diagramas de casos de uso. Está é a principal dificuldade encontrada no JGOOSE, já que o usuário precisa usar de forma independente a ferramenta StarUML para modificar o diagrama de casos de uso. Desta maneira, no próximo capítulo será apresentado o editor E4J Use Cases bem como um novo diagrama de atividades mostrando como o E4J Use Cases impacta na estrutura do JGOOSE.

3.2 E4J i*

O E4J i* é um ambiente de desenvolvimento de diagramas de modelos organizacionais i* e foi desenvolvido em um trabalho de conclusão de curso por [MERLIM 2013]. O E4J i* provê recursos e funcionalidades para criação e manipulação de diagramas SD e SR e a conversão desses diagramas para a estrutura de modelo do JGOOSE. Como o E4J i* está integrado ao JGOOSE, a conversão de estruturas e modelos, do E4J i* para o JGOOSE, é realizada via rotinas internas, sem a necessidade de geração de arquivos intermediários.

Segundo [YU 1995], os diagramas SD e SR são estruturas de grafos com diversos tipos de vértices e ligações que, em conjunto, expressam as razões por de trás de processos. Neste contexto, surge a biblioteca Java para visualizações de grafos JGraphX [ALDER 2002]. Com o JGraphX é possível criar aplicações interativas voltadas principalmente para a manipulação de diagramas. Assim, o E4J i* utiliza os recursos do JGraphX para criar a estrutura que representa os modelos i*. Cabe ressaltar que, essa estrutura usada pelo E4J i* também foi adotada para o desenvolvimento do E4J Use Cases e apresentaremos a mesma em detalhes no próximo capítulo.

Para realizar a integração do E4J i* ao JGOOSE, foi preciso realizar uma adaptação no JGOOSE. Conforme é exibido na Figura 3.9, a interface gráfica do JGOOSE possui um botão e um item de menu para realizar a chamada do E4J i*. Essas opções realizam a função de carregar a interface gráfica do E4J i* e apresentá-lo ao usuário. Após a execução do evento gerado por essas opções, a linha de execução principal do programa passa a ser de responsabilidade das classes do E4J i*, retornando ao JGOOSE apenas em situações que o usuário deseja gerar os casos de uso do modelo construído no E4J i*.

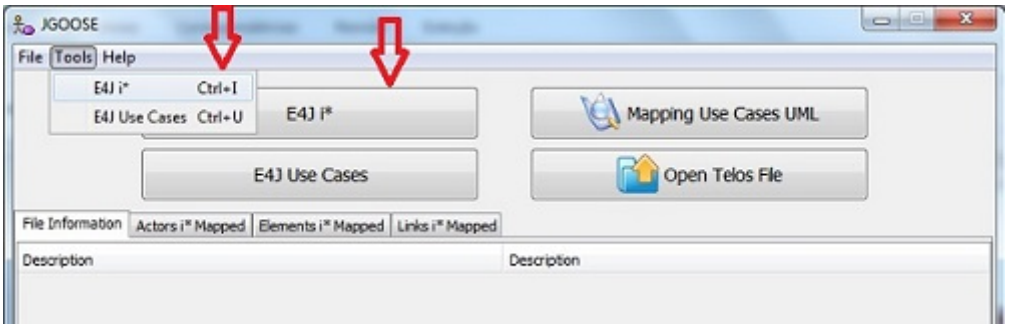


Figura 3.9: JGOOSE com opções de chamada ao editor E4J i*.

A interface gráfica do usuário é composta por uma tela principal e é nesta tela que o usuário terá a sua disposição as principais funcionalidades do editor E4J i*. A Figura 3.10 apresenta a tela principal do editor. Para melhor apresentá-la, a mesma foi dividida em seis áreas comentadas a seguir.

1. **Área de desenho**: é a área de trabalho do usuário na qual é construído o modelo. É uma área de visualização e manipulação do modelo, onde ficam visualmente os elementos do modelo i* (SD e SR) e suas ligações;

2. **Barra de menus**: dá o acesso a todas as ações gerais sobre a aplicação. Todos os menus (*File*, *Edit*, *View*, *Format*, *Shape*, *Model*, *Options* e *Help*) são apresentados para prover funcionalidades à ferramenta, tais como abrir/salvar arquivos, controles de zoom, exportar para iStarML e gerar casos de uso;

3. **Barra de ferramentas**: é um conjunto de atalhos para as funções mais comuns de uma aplicação ou funções usadas com frequência durante a criação ou edição de modelos;

4. **Paleta de elementos**: contém cinco abas que agrupam os elementos dos modelos SD e SR (vértices e arestas). Os elementos estão divididos nas seguintes abas: *Actor* (*Actor*, *Role*, *Agent* e *Position*); *Actor Associations* (*ISA*, *Is-part-of*, *Plays*, *Covers*, *Occupies* e *Instance of*); *Dependency Elements* (*Goal*, *Task*, *SoftGoal* e *Resource*); *Relationship Links* (*Dependency*, *Means-end* e *Task-decomposition*); e *Contribution Links* (*Make*, *Some+*, *Help*, *Break*, *Some-*, *Hurt*, *Unknown*, *And* e *Or*);

5. **Mini-mapa**: é uma miniatura do modelo inteiro. Sua função é ajudar o trabalho com modelos muito grandes. Contém um retângulo azul que representa uma porção do modelo total correspondente ao que está sendo exibido na área de desenho;

6. **Barra de informações**: basicamente apresenta ao usuário a posição do mouse ou fornece algumas informações sobre o estado da área de desenho.

Cabe ressaltar que neste projeto, o editor E4J Use Cases considera como base esta interface e modifica essencialmente a área 4, a qual será composta pelos elementos que compõe os diagramas de casos de uso e será apresentada no próximo capítulo.

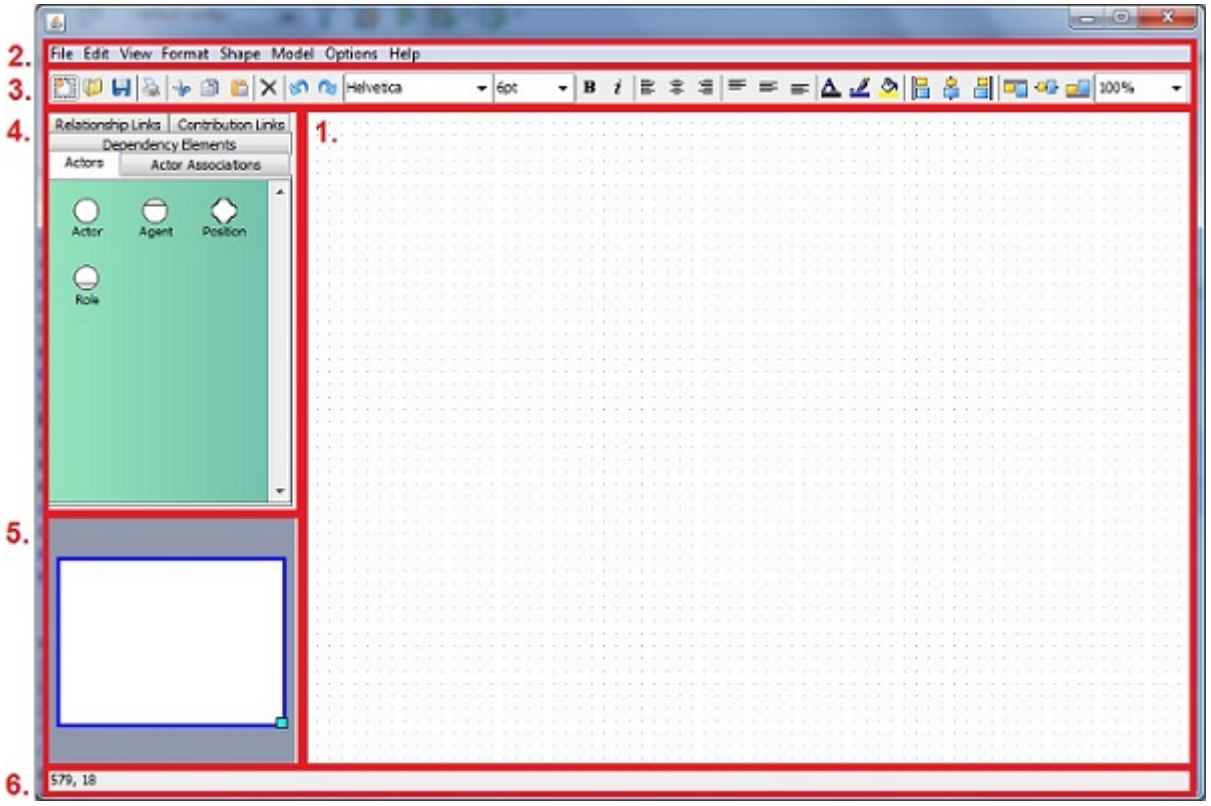


Figura 3.10: Tela principal do editor E4J i*.

3.3 Considerações Finais do Capítulo

Neste capítulo foi apresentada a ferramenta JGOOSE e mostrado o seu funcionamento para um cenário específico. O JGOOSE tem como foco principal a geração dos casos de uso a partir de modelos i*, os quais podem ser construídos utilizando à ferramenta E4J i* que está integrada ao JGOOSE e também foi brevemente apresentada neste capítulo. O estudo e entendimento dessas ferramentas foram de grande importância para o desenvolvimento desse projeto, pois é a estrutura base do E4J i* que será utilizada para o desenvolvimento do editor E4J Use Cases. O E4J Use Cases terá o foco no **gerenciamento dos diagramas de casos de uso**. Desta forma, o JGOOSE passará a cuidar especificamente da geração de casos de uso, deixando as funcionalidades que envolvem o diagrama de casos de uso a cargo do E4J Use Cases.

Capítulo 4

E4J Use Cases

Foram apresentados no capítulo 2 os fundamentos teóricos envolvendo o *framework* i* e casos de uso UML. Também foram apresentadas, no capítulo 3, as ferramentas JGOOSE, a qual suporta a geração de casos de uso a partir de modelos i*, e E4J i* com seu papel de criação e manipulação dos modelos i* que servem como entrada para a ferramenta JGOOSE. Neste capítulo é apresentado o projeto e o desenvolvimento do E4J Use Cases, uma ferramenta para criação e manipulação gráfica de diagramas de casos de uso para o ambiente de trabalho do JGOOSE.

Inicialmente, na seção 4.1, tem-se uma visão geral da ferramenta apresentando sua estrutura de grafos adotada para a representação dos diagramas de casos de uso. Na seção 4.2, é apresentado o projeto e arquitetura do editor, contendo o diagrama de atividades, diagrama de pacotes e diagrama de classes. O modelo organizacional (veja seção 3.1.2, Figuras 3.3 e 3.4) e o diagrama de casos de uso (veja seção 3.1.2, Figura 3.7) já foram apresentados no capítulo anterior. Em seguida, na seção 4.3, é detalhado o desenvolvimento do editor. Na seção 4.4, é apresentado o impacto do E4J Use Cases sobre o fluxo de atividades do JGOOSE através de um novo diagrama de atividades da ferramenta JGOOSE. Por fim, na seção 4.5, são realizadas as considerações finais do capítulo.

4.1 Visão Geral

O E4J Use Cases é um editor de desenvolvimento de diagramas de casos de uso. Ele provê recursos e funcionalidades para criação de diagramas de casos de uso bem como suporta a manipulação dos diagramas gerados através dos casos de uso mapeados pelo JGOOSE. Como o E4J

Use Cases está integrado ao JGOOSE, a conversão de estruturas e modelos, do JGOOSE para o E4J Use Cases, é realizada via rotina interna, sem a necessidade de arquivos intermediários. Essencialmente, essa rotina interna envolve encontrar uma classe na estrutura do E4J Use Cases que seja equivalente à estrutura do JGOOSE.

Cabe ressaltar que se optou por desenvolver o editor E4J Use Cases utilizando a base estrutural do E4J i* (ver capítulo 3, seção 3.2) considerando aos seguintes aspectos:

- Possui uma interface gráfica do usuário (do inglês GUI - *Graphical User Interface*) para manipulação gráfica;
- Possui as funcionalidades básicas da GUI, mais especificamente, funcionalidades de manipulação de entidades geométricas como, por exemplo, atores e ligações;
- Possui meios para salvar e abrir as entidades geométricas manipuladas;
- Melhorias estão sendo realizadas no E4J i* por um discente do curso de Ciência da Computação da Unioeste, campus de Cascavel, o que facilita a utilização da base estrutural e a integração ao JGOOSE;
- A ferramenta E4J i* foi desenvolvida especificamente para o JGOOSE e já está integrada ao mesmo;
- O uso da mesma base estrutural facilitará futuras evoluções tanto do E4J i* quanto do E4J Use Cases e do JGOOSE.

Além disso, essa estrutura base utiliza a biblioteca JGraphX para a manipulação de diagramas e a mesma está sob licença BSD [PROJECT 2005] e possui uma boa representatividade na comunidade, contanto com:

- Um repositório público no GitHub[1] com 38 *Forks*[2] (cópias de outros usuários para novos trabalhos ou melhorias);
- Um tópico específico no StackOverflow[3], um fórum de discussão reunindo especialistas em áreas específicas do conhecimento;

[1] https://github.com/jgraph/jgraphx.
[2] Forks são cópias de outros usuários para novos trabalhos ou melhorias.
[3] http://stackoverflow.com/questions/tagged/jgraphx.

- Um antigo fórum, com mais de 1600 perguntas e mais de 1800 respostas dos usuários e desenvolvedores.

Outra vantagem da utilização da biblioteca JGraphX, é que o núcleo da mesma está fundamentada na Teoria dos Grafos [ALDER 2002]. Um grafo é formado por um conjunto de vértices e um conjunto de arestas. Um vértice pode ser chamado também de nodo ou nó. As arestas são conexões entre os nós. Na estrutura JGraphX existe também o conceito de célula que representa um elemento do grafo: uma aresta, um vértice ou um grupo destes. Desta maneira, podemos representar os diagramas de casos de uso sobre as estruturas de grafos e isso também influenciou na escolha da estrutura base do E4J i* para ser usada como estrutura base também do E4J Use Cases. Na próxima subseção, apresentaremos em detalhes a biblioteca JGraphX.

4.1.1 Biblioteca JGraphX

A JGraphX é uma biblioteca Java para visualização de grafos [ALDER 2002]. Consiste em um conjunto de estruturas e funcionalidades que facilitam a produção de aplicações Java Swings. Com essa biblioteca é possível criar aplicações interativas voltadas principalmente para manipulação de diagramas.

A estrutura arquitetural do JGraphX, segundo [JGRAPH 2013], assemelha-se com um arquitetura MVC (*Model-View-Controller*) [PRESSMAN 2009]. Conforme a Figura 4.1, pode-se observar que a estrutura do grafo (*mxGraph*) possui um *model* (*mxGraphModel*), uma *view* (*mxGraphView*) e os elementos *mxStylesheet*, *mxCellRenderes* e *mxGraphSelection* atuando como o controller da arquitetura.

Um *mxGraph* é a essência da estrutura do grafo da JGraphX. Conforme a Figura 4.2, pode-se observar que o *mxCell* é a estrutura que representa um vértice (*vertex*), uma aresta (*edge*) ou um conjunto destes. E é na propriedade *value* do *mxCell* que podemos armazenar os dados desejados. No caso da presente proposta, ficam armazenadas as informações dos elementos do diagrama de casos de uso referentes as seguintes propriedades:

- *id*: identificação única do elemento do diagrama;
- *title*: título ou rótulo do elemento do diagrama, com o nome de um ator ou de um caso de uso;

- *type*: o tipo do elemento do diagrama. Como exemplo: actor, use case, include e etc.

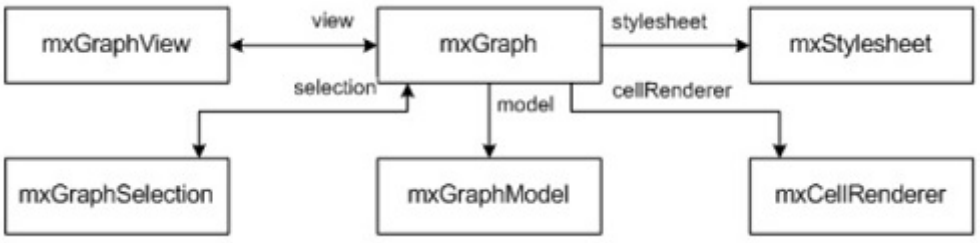


Figura 4.1: Estrutura central da JGraphX.

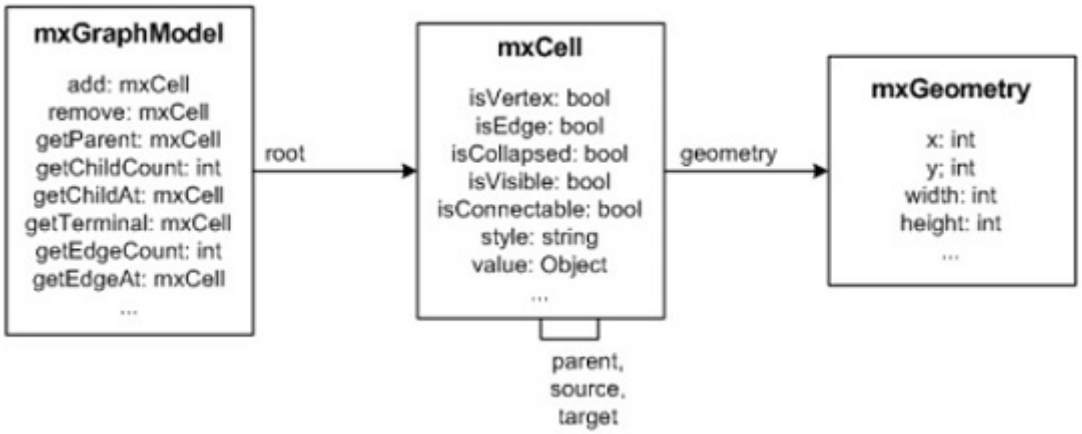


Figura 4.2: Estrutura do modelo da JGraphX.

4.2 Projeto e Arquitetura

A arquitetura do E4J Use cases está contemplada pelas seguintes visões: Visão Organizacional via *framework* i*; Diagrama de Casos de Uso, mostrando uma visão das funcionalidades da ferramenta; Diagrama de Atividades; mostrando os processos e fluxo de objetos; Diagrama de Pacotes e Classes, propiciando uma visão estrutural da ferramenta.

A visão organizacional, já apresentada na Figura 3.4 (seção 3.1.2 do capítulo 3), ajuda a identificar as principais relações entre os envolvidos (*stakeholders*) com o sistema. Já o diagrama de casos de uso, apresentado na Figura 3.7 (seção 3.1.2 do capítulo 3), permite visualizar os principais requisitos funcionais do sistema proposto. O Diagrama de atividades, diagrama de pacotes e diagrama de classes são apresentados nas subseções seguintes.

4.2.1 Diagrama de Atividades

Conforme visto no capítulo 3 (seção 3.1.3), o diagrama de atividades UML tem como principal objetivo a representação gráfica de modelos que coordenam as sequencias e condições de comportamentos de um sistema [BOOCH, RUMBAUGH e JACOBSON 2005]. Na Figura 4.3 são apresentadas as partições *E4J Use Cases* e *JGraphX* que, em conjunto, coordenam as principais atividades do editor E4J Use Cases. Iniciando pela partição *E4J Use Cases*, as configurações e elementos da interface gráfica do usuário são carregados e apresentados. A partir disso, o fluxo passa do conector "A" da partição E4J Use Cases para o conector "A" da partição *JGraphX*. Em paralelo, são realizadas as rotinas de envio e recebimento de sinais (*« signal sending »*, *« signal receipct »*) ou eventos.

Neste diagrama de atividades da ferramenta E4J Use Cases (Figura 4.3), são representados os seguintes elementos:

- *E4J Use Cases* (partição): esta partição concentra as principais funcionalidades de responsabilidade do E4J Use Cases;
  - Nodo inicial: é onde começa a aplicação E4J Use Cases. É dado início ao carregamento dos recursos necessários para utilização da ferramenta;
  - Carregar Arquivos de Configuração: nesta ação, os arquivos de idioma e de configuração do *Logger* são carregados. Os arquivos de idioma interferem diretamente nos nomes dos *shapes* e nos elementos da interface gráfica, e por isso deve ser a primeira ação do E4J Use Cases;
  - Carregar *Shapes*: os *shapes* são a definição da apresentação gráfica dos elementos. Esta ação efetua o carregamento dos *shapes* e a adição de cada um na paleta de elementos;
  - Carregar Interface Gráfica: é a ação que constrói toda a interface gráfica. Após esta ação, os elementos da interface gráfica são apresentados e já podem enviar e receber eventos do usuário;
  - Conectar (A): indica que o fluxo é transferido para o Conector (A) da partição *JGraphX*. Este, por sua vez, começa a tratar (enviar e receber) os eventos do usuário;

- Salvar (evento recebido): é o evento gerado para salvar o diagrama projetado em um formato nativo do JGraphX;
- Salvar em .mxe: ação para salvar, em arquivo, a estrutura do E4J Use Cases no formado padrão do JGraphX. Após está ação, o E4J Use Cases volta a tratar os eventos do usuário pelo Conector (A);
- Carregar arquivo.mxe: evento que, quando recebido, gera a ação de mesmo nome. Esta ação é a rotina de carregamento do arquivo e sua apresentação na interface gráfica. Após está ação, o editor volta a tratar os eventos do usuário;
- Sair: evento recebido quando o usuário clica no item de menu "Sair" ou quando pressiona as teclas de atalho "Alt + F4". Após este evento, o fluxo de execução da aplicação é finalizado.

- *JGraphX* (partição): esta partição se resume ao tratamento (envio e recebimento) de eventos do usuário realizado pelo JGraphX;
  - Conector (A): elemento de conexão com as outras partições. Pode ser encarada como nodo inicial desta partição;
  - *Split*: divide o fluxo entre as atividades de envio e recebimento de eventos da interação com o usuário;
  - Eventos do Usuário (*signal sending*): são os eventos gerados pelo usuário. Esses eventos são enviados para toda a aplicação. Por exemplo, caso o usuário clique no item de menu "Sair", via interface gráfica, este sinal de evento será enviado para toda a aplicação e o recebimento deste evento é tratado pelo elemento *Sair* (da partição *E4J Use Cases*);
  - Eventos do Usuário (*signal receipt*): são os eventos captados pelo JGraphX. O principal evento apresentado no diagrama é o de atualização do modelo. Por exemplo, a adição de um ator no diagrama é um evento que implica na atualização do modelo;
  - Condição: este elemento verifica se o evento que foi gerado é um evento de alteração do diagrama. Caso positivo, o fluxo de atividades passa para a execução da ação *Atualizar Modelo*. Caso contrário, o evento é enviado para o restando da aplicação;

– Atualizar Modelo: ação de manipulação do modelo. Todas as alterações são aplicadas na estrutura do modelo e o resultado é estrutura do *Grafo Atualizado*;

– Grafo Atualizado: objeto resultado da atualização do diagrama. Este objeto, já atualizado, é enviado como um sinal de evento para tratamento em outras partes da aplicação, como exemplo, a verificação da consistência do digrama. Essa verificação consiste em validar se uma ligação está sendo corretamente inserida. Por exemplo, se o usuário tentar colocar uma ligação do tipo *Include* entre um ator e um caso de uso, o editor não permitirá e exibirá uma mensagem informando que a ligação é inválida.

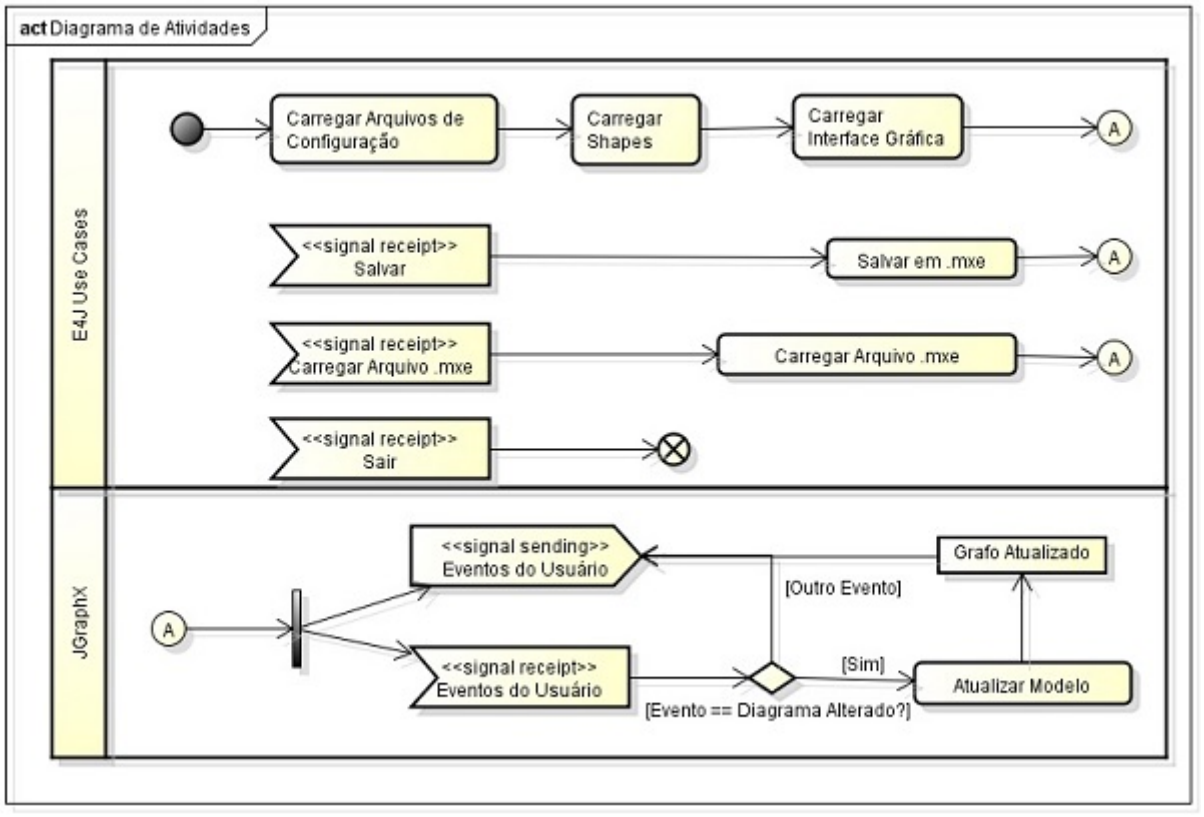


Figura 4.3: Diagrama de Atividades do E4J Use Cases.

4.2.2 Diagrama de Pacotes

Sob o contexto de diagramas arquiteturais, foi desenvolvido um diagrama de pacotes conforme apresentado na Figura 4.4. Um pacote é um mecanismo de propósito geral para auxi-

liar a organização do sistema desenvolvido [BOOCH, RUMBAUGH e JACOBSON 2005]. O "JGOOSE-MAVEN" é o pacote principal. Os pacotes internos ao "JGOOSE-MAVEN" formam as dependências do pacote principal. O pacote "JGOOSE-MAVEN" possui as seguintes dependências:

- **JGOOSE**: representa à versão projeto do JGOOSE refatorada para a estrutura de projetos Maven [MAVEN 2014]. A refatoração foi realizada no desenvolvimento do editor E4J i* [MERLIM 2013] e foi necessária para criar compatibilidade entre os projetos do JGOOSE e do E4J i*. Sobre as classes desse pacote, destaca-se a classe *MainView*, que é responsável pela interface gráfica do JGOOSE; a classe *Tokens*, que contem a estrutura de dados para representar o modelo organizacional ; e a classe *UseCases*, que contém a estrutura dos casos de uso gerados pelo JGOOSE após a aplicação das diretrizes de mapeamento;

- **E4JUseCases**: representa toda a estrutura da aplicação do editor E4J Use Cases. Contém vários elementos de gerenciamento da interface gráfica e de tratamento dos eventos do usuário. A maioria desses elementos são heranças de classes do pacote *jgraphx*. É neste pacote que ficam também as funções de mapeamento entre as estruturas do JGOOSE para o E4J *Use Cases*.

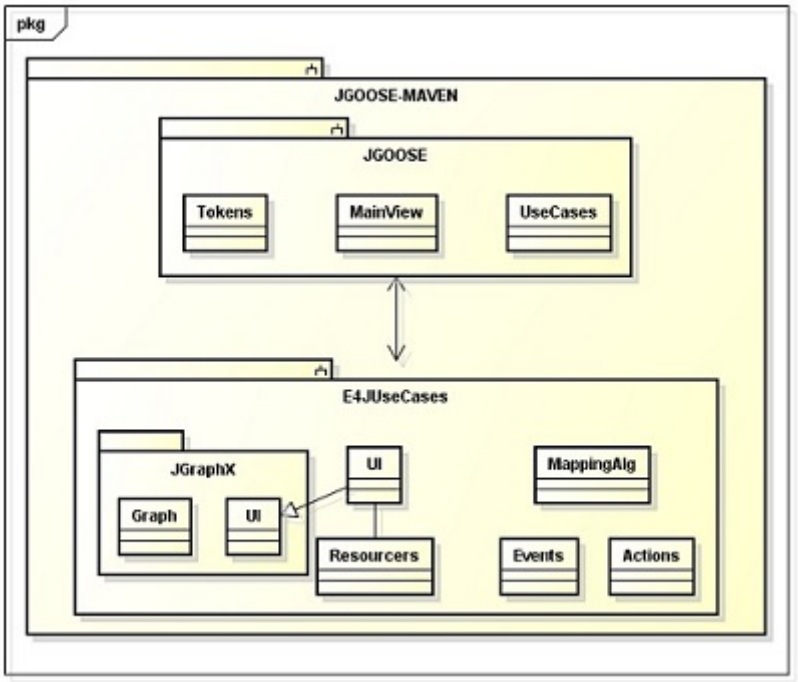


Figura 4.4: Diagrama de Pacotes do E4J Use Cases.

4.2.3 Diagrama de Classes

Diagramas de classe estão entre os considerados mais importantes na área de modelagem de sistemas orientados a objetos [GROUP 2007]. Esses diagramas mostram um conjunto de classes, interfaces e relacionamentos, representando uma visão estática do projeto do sistema. Os diagramas de classe não só ajudam na visualização, especificando e documentando os modelos estruturais, como também servem para a construção de sistemas por meio de técnicas de geração de código automática. Neste trabalho, foram desenvolvidos os diagramas de classe da estrutura principal presente no pacote *E4JUseCases* da Figura 4.4 e do conjunto de ações. Esses diagramas são exibidos, respectivamente, nas Figuras 4.5 e 4.6.

A seguir descrevemos o diagrama de classes da estrutura principal do E4J Use Cases:

- *EditorPalette*: classe para gerenciamento dos componentes da paleta de elementos e conectores. Esta classe trata a seleção dos tipos de conexões (via método "*setSelectionEntry*" e a funcionalidade de arrastar e soltar dos elementos (via "*listeners*"). O ato de arrastar e soltar os elementos reflete na alteração do grafo (*CustomGraph*);

- *EditorKeyboardHandler*: é a classe responsável por tratar os eventos do teclado. É nesta

classe que são especificados os atalhos (presentes no apêndice A). Um evento gerenciado por esta classe normalmente reflete na alteração do grafo (*CustomGraph*);

- *AbstractAction*: classe abstrata para prover funcionalidades comuns as outras classes de ações do usuário. As classes que estendem esta abstração são apresentadas na figura 5.6;

- *CustomGraph*: é a classe responsável por gerenciar a estrutura grafo, suas alterações e restrições. É nesta classe que são criados os tratamentos de validações das ligações entre elementos do modelo (método " *createEdge*");

- *CustomGraphComponente*: é uma classe intermediária entre a visão do usuário (*BasicGraphEditor*) e o modelo em grafo (*CustomGraph*). É responsável por mapear o conteúdo do valor de um elemento do grafo (retornando pelo método "*convertValueToString*" do modelo) para a visão;

- *BasicGraphEditor*: é a classe principal do projeto. Estende a classe *JFrame* para apresentar os elementos gráficos (visão), além de agregar os elementos acima descritos (modelo e eventos). Esta classe também agrega o gerenciador de histórico ("*UndoManger*"), tornando possível o uso de "*Undo/Redo*".

Na Figura 4.6, temos um diagrama de classes para tratamento e execução das principais ações do usuário. Essas ações estão normalmente acopladas aos itens de menus, menus de contexto ou atalhos do teclado. Observa-se que todas são extensões da classe *AbstractAction*, classe utilizada para implementar ações relativas aos componentes Java Swing [ECKSTEIN, LOY e WOOD 1998].

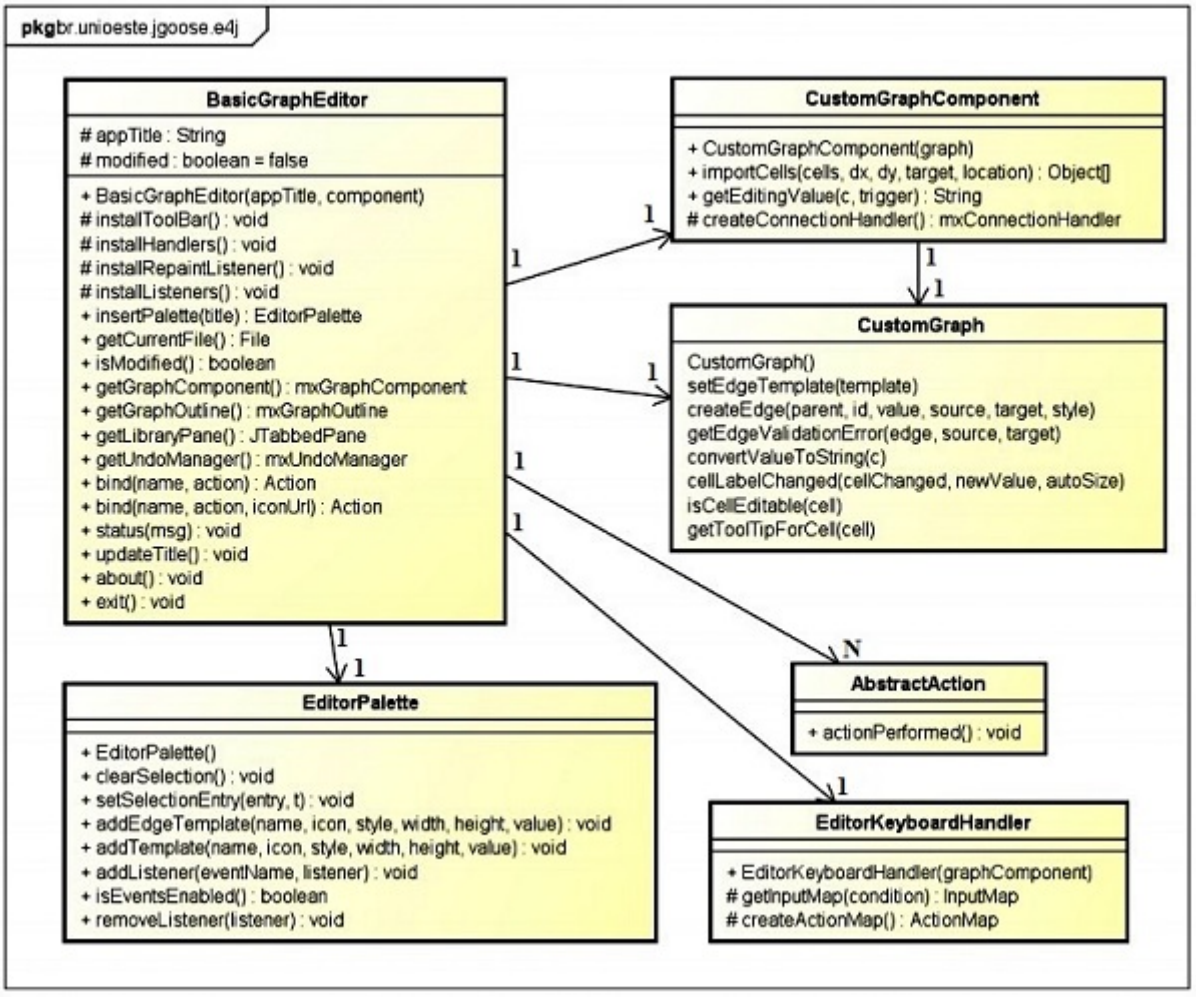


Figura 4.5: Diagrama de Classes parcial do E4J Use Cases. Estrutura principal.

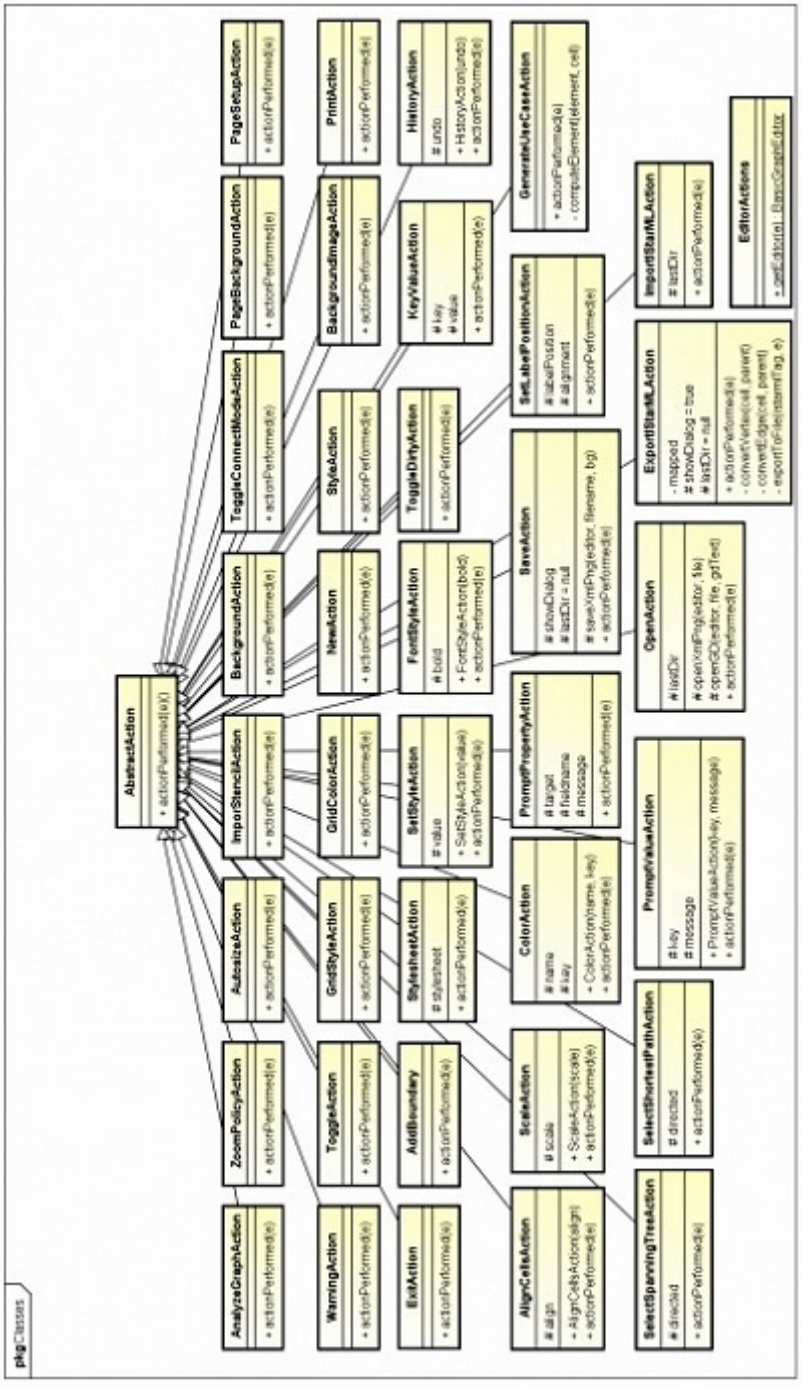


Figura 4.6: Diagrama de Classes parcial do E4J Use Cases. Principais ações do usuário.

4.3 Desenvolvimento

O processo de desenvolvimento adotado foi iterativo e incremental, com apresentações e discussões junto ao Grupo LES (Laboratório de Engenharia de *Software*). Foram realizados encontros com o professor orientador para avaliar as versões após o término de cada iteração. No total foram realizadas 14 iterações com 2 semanas cada.

O código fonte da estrutura base dos editores (E4J i* e E4J Use Cases) está dividido em vários pacotes, sendo que todo o código fonte necessário para o diagrama de casos de uso foi adicionado a classes existentes ou foram criadas novas classes, caso fosse necessário. Destaca-se que ambos os editores (E4J i* e E4J Use Cases) fazem parte de um projeto maior, mais especificamente, do ambiente de trabalho do JGOOSE. Esse ambiente permite trabalhar de forma integrada com modelagem organizacional via *framework* i*, casos de uso e diagramas de casos de uso.

A tela principal do E4J Use Cases é apresentada na Figura 4.7 e também foi dividida em seis áreas conforme a Figura 3.10 (seção 3.2.2 do capítulo anterior) que apresenta a tela principal do E4J i*. As áreas 1, 2, 3, 5 e 6 são equivalentes as do E4J i* e já foram comentadas (veja seção 3.2.2 do capítulo anterior). A área (4) sofreu as alterações necessárias para atender ao diagrama de casos. Mais especificamente, ela apresenta uma única paleta de elementos contendo os elementos do diagrama de casos de uso (vértices e arestas). Os elementos são: *Actor*, *Use Case*, *Association*, *Generalization*, *Include* e *Extend*.

Foi desenvolvida uma rotina de mapeamento no E4J Use Cases para realizar a geração do diagrama de casos de uso com base nos casos de uso mapeados pelo JGOOSE. O pseudocódigo denominado *Algorithm 1* apresentado a seguir é uma abstração da rotina de mapeamento implementada.

Basicamente, essa rotina de mapeamento gera um elemento (elipse contendo o nome do caso de uso) para cada caso de uso e um elemento (“boneco” contendo o nome do ator) para cada ator mapeado pelo JGOOSE e uma ligação (do tipo *association*) entre cada ator e seus casos de uso. Posteriormente é verificado se cada passo presente nas descrições textuais de cada caso de uso é um caso de uso por si só, e caso positivo, esse caso de uso está presente no diagrama e é gerada uma ligação (do tipo *« include »*) entre o caso de uso sendo verificado e o caso de uso que representa o passo. Além disso, junto com a verificação dos passos, é verificado também se

cada passo possui extensões e se essas também são casos de uso por si só e, se forem, também estão presentes no diagrama e assim também é gerada uma ligação (do tipo *« extend »*) entre o caso de uso sendo verificado e o caso de uso que representa a extensão.

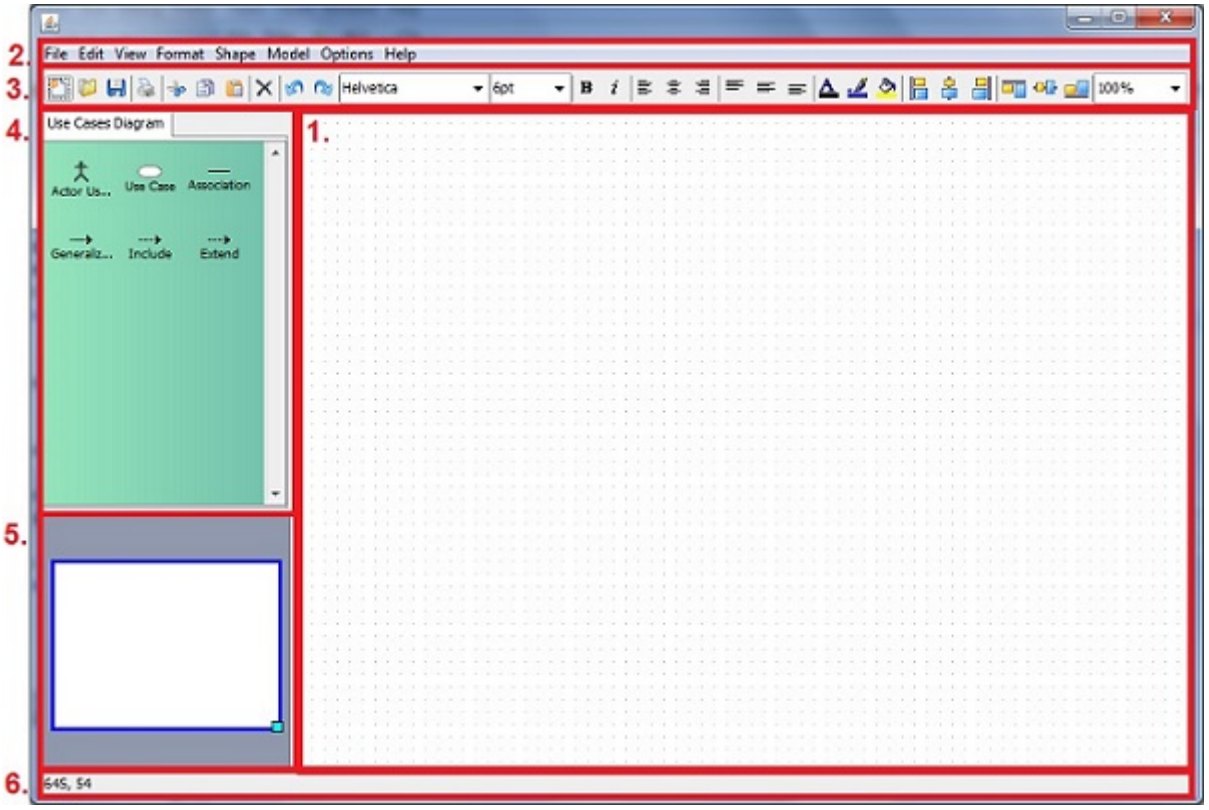


Figura 4.7: Tela principal do editor E4J Use Cases.

Cabe ressaltar que após a geração do diagrama de casos de uso com base nos casos de uso mapeados pelo JGOOSE, o usuário pode realizar todas as modificações e/ou adições que desejar como, por exemplo, adicionar novos casos de usos, atores, ligações, etc. Essa possibilidade de edição do diagrama de casos de uso não estava presente anteriormente (antes do E4J Use Cases ser desenvolvido) no JGOOSE, sendo que era gerado apenas um diagrama estático, ou seja, não era possível fazer quaisquer modificações.

Após o diagrama de casos de uso ser gerado e realizadas as modificações e/ou adições desejadas, o usuário pode ainda salvar o diagrama nos formatos de imagem e no formato nativo da biblioteca JGraphX (*.mxe*). Esse último formato é o padrão do E4J Use Cases e é utilizado para poder abrir novamente o diagrama.

Algorithm 1 Rotina de Mapeamento para geração do Diagrama de Casos de Uso

Parâmetros: casos de uso mapeados pelo JGOOSE
Saída: diagrama de casos de uso gerado no editor E4J Use Cases
para cada ator de caso de uso **faça**
  adicionar o ator no grafo
  para cada caso de uso do ator **faça**
    adicionar o caso de uso no grafo se esse caso de uso ainda não foi adicionado
    adicionar uma aresta do tipo association entre o ator e o caso de uso no grafo
  fim-para
fim-para
para cada ator de caso de uso **faça**
  para cada caso de uso do ator **faça**
    para cada passo do caso de uso **faça**
      se o passo é um include **faça**
        adicionar uma aresta do tipo « include » entre o caso de uso atual e o caso de uso que representa o passo no grafo
        para cada extensão do passo **faça**
          se a extensão é um caso de uso **faça**
            adicionar uma aresta do tipo « extend » entre o caso de uso atual e o caso de uso que representa a extensão no grafo
          fim-se
        fim-para
      fim-se
    fim-para
  fim-para
fim-para
para cada ator do tipo ISA **faça**
  para cada ator que é pai do ator do tipo ISA **faça**
    adicionar uma aresta do tipo generalization entre os atores no grafo
  fim-para
fim-para

4.4 Impacto do E4J Use Cases no JGOOSE

A Figura 4.8 apresenta um novo diagrama de atividades da ferramenta JGOOSE. Esse diagrama apresenta o fluxo de atividades do JGOOSE atual com a integração do editor E4J Use Cases.

Nota-se que o E4J Use Cases pode ser utilizado tanto no inicio do ambiente de trabalho do JGOOSE, para o usuário criar seu próprio diagrama de casos de uso, bem como pode ser utilizado após a geração dos casos de uso. Nesse caso, ao se chamar o E4J Use Cases, a rotina de mapeamento é executada e o E4J Use Cases é exibido contendo um diagrama de casos de uso construído com base nos casos de uso mapeados pelo JGOOSE. Para ilustrar esse processo, é descrito a seguir uma exemplificação desse fluxo de atividades no ambiente de trabalho do JGOOSE.

Inicialmente, se construirmos o modelo SR da Figura 3.4 (seção 3.1.2 do capítulo 3) no E4J i* e acionarmos o JGOOSE para o mapeamento de casos de uso, como é exibido na Figura 4.9, o fluxo de atividades do E4J i* se encerra e o JGOOSE é acionado exibindo uma tela para seleção do ator que representa o sistema computacional, como é exibido pela Figura 4.10. Após esse ator ser selecionado, o usuário deve acionar a rotina de mapeamento de casos de uso, a qual é responsável pela aplicação das diretrizes (ver capítulo 2, seção 2.3), como também é exibido pela Figura 4.10. Posteriormente, o JGOOSE exibe uma tela contendo todos os casos de uso mapeados juntamente com o botão "*Diagram*" responsável por acionar a rotina de mapeamento apresentada anteriormente e gerar o diagrama de casos de uso no E4J Use Cases, como é exibido pela Figura 4.11. Acionado o botão "*Diagram*", o fluxo de atividades do JGOOSE se encerra e o E4J Use Cases exibe uma tela contendo o diagrama de casos de uso gerado, como é exibida pela Figura 4.12. Cabe ressaltar que o diagrama de casos de uso é gerado com uma pré-organização, sendo que o usuário pode manipulá-lo para atingir uma melhor visualização, como foi feito com o diagrama exibido pela Figura 4.12.

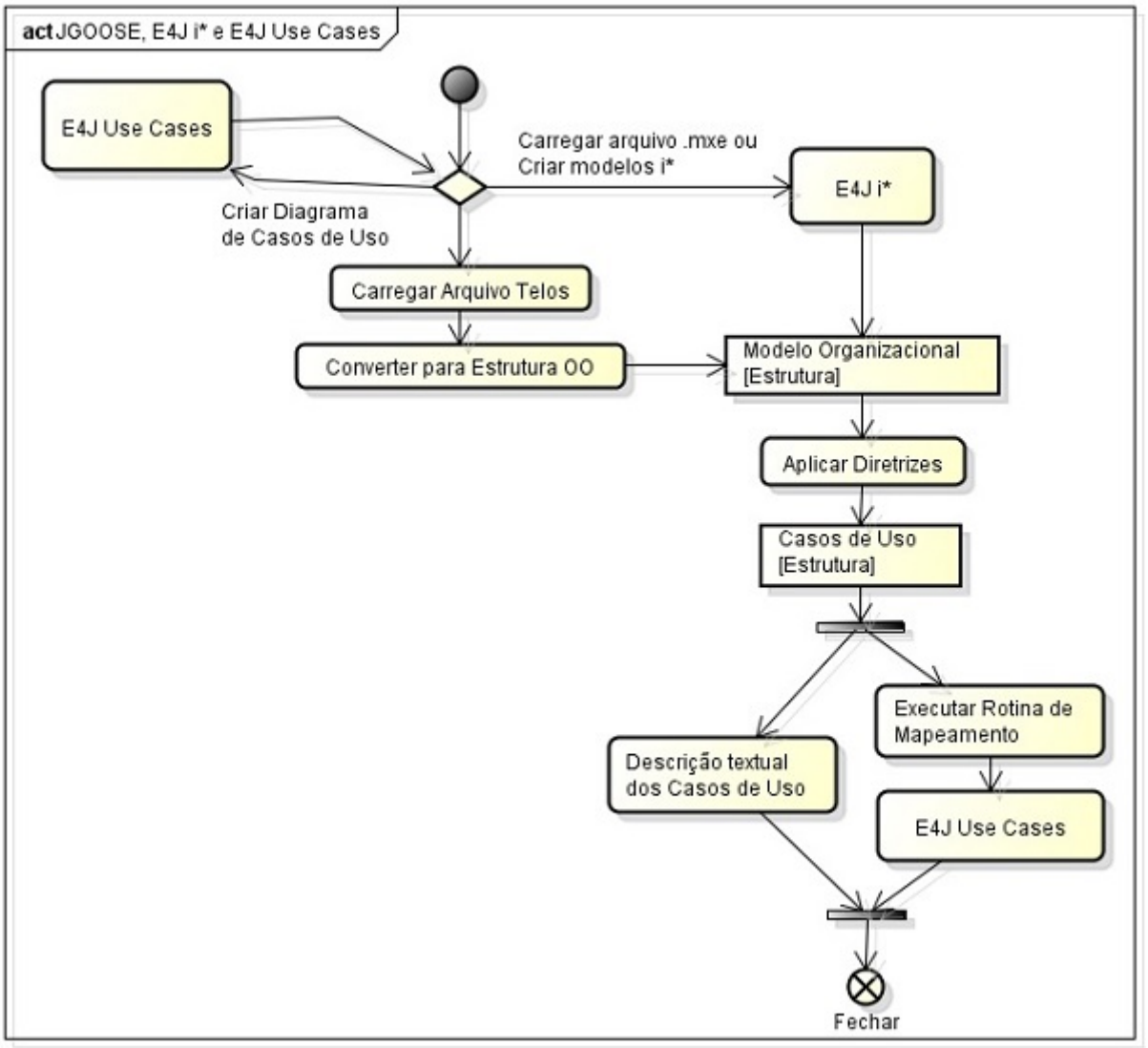


Figura 4.8: Diagrama de atividades da ferramenta JGOOSE com E4J i* e E4J Use Cases.

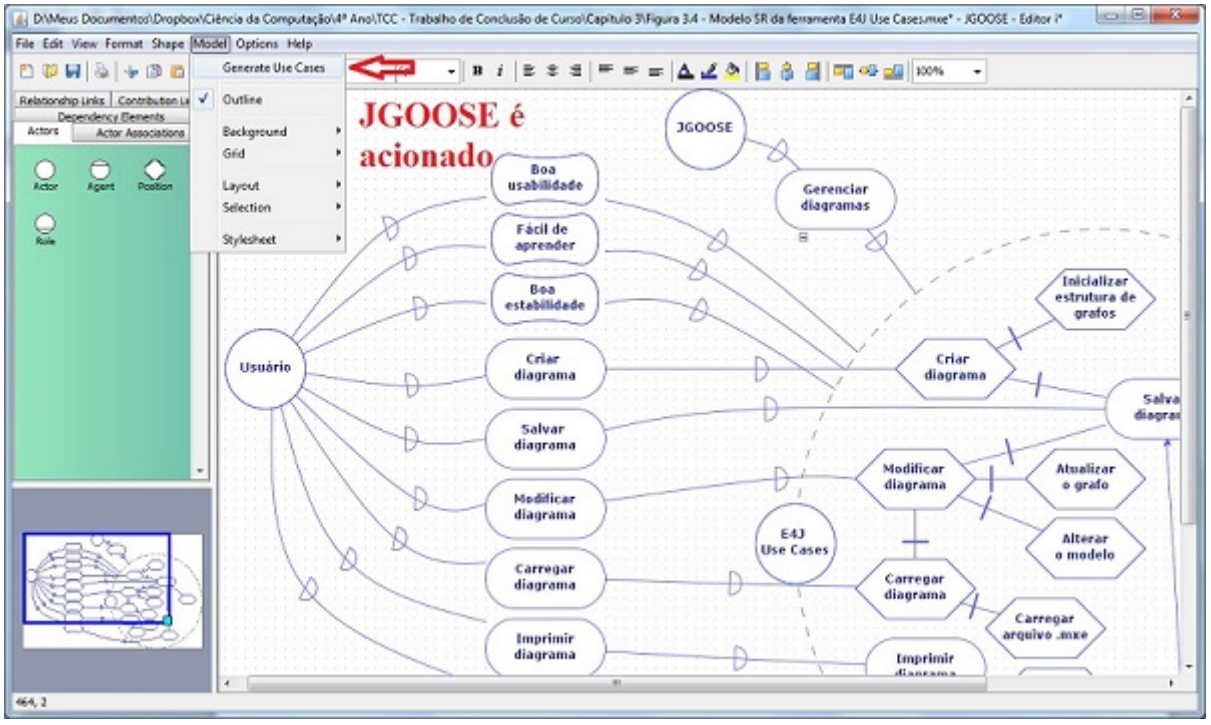


Figura 4.9: JGOOSE sendo acionado pela ferramenta E4J i*.

4.5 Considerações Finais do Capítulo

Neste capítulo foi apresentada a ferramenta E4J Use Cases, o seu projeto e o impacto que o E4J Use Cases causou no JGOOSE. No próximo capítulo é descrito um quase-experimento realizado com a ferramenta E4J Use Cases, o qual foi realizado com o objetivo de obter um *feedback* sobre a mesma para realizar possíveis melhorias e/ou correções.

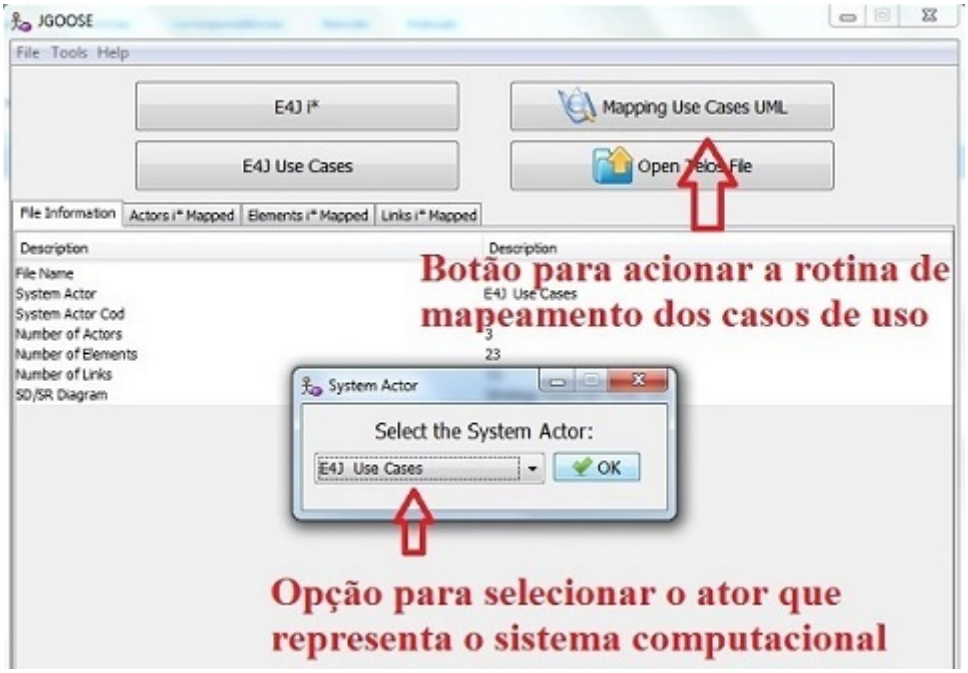


Figura 4.10: Opção para selecionar o ator que representa o sistema computacional.

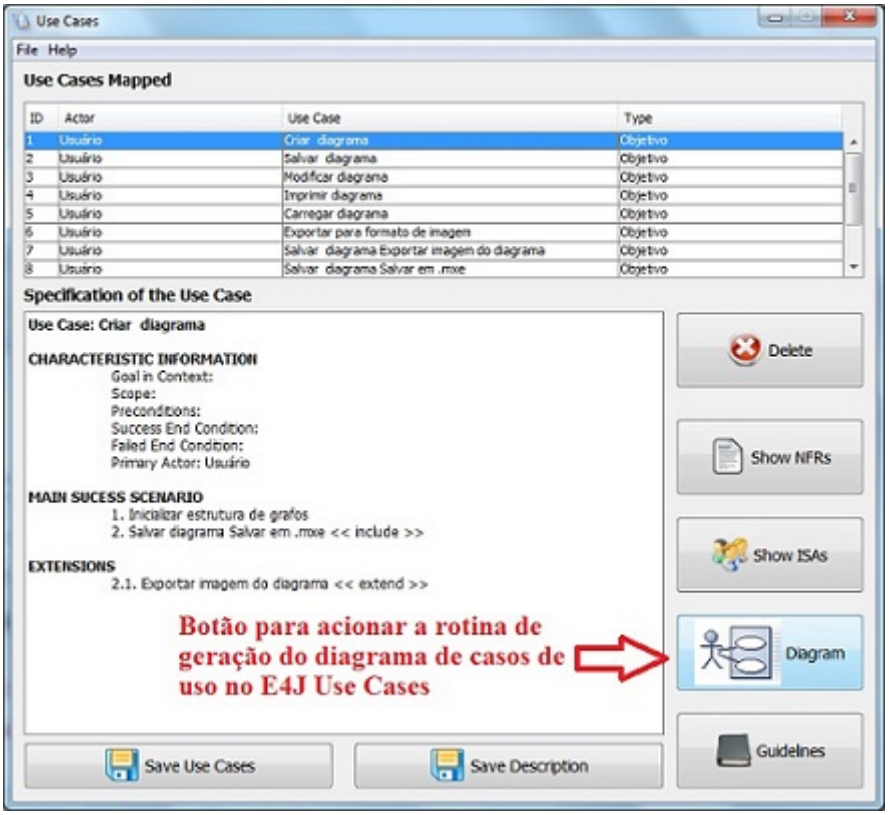


Figura 4.11: Opção para acionar a rotina de geração do diagrama de casos de uso.

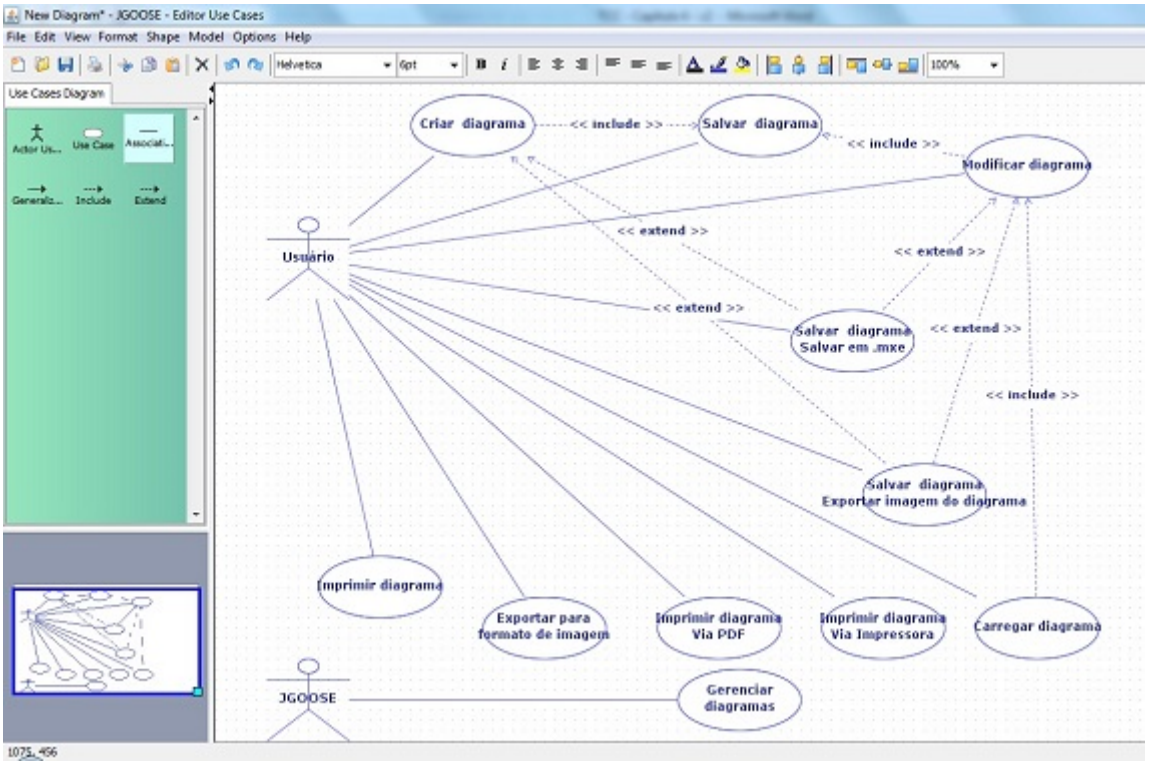


Figura 4.12: E4J Use Cases com diagrama de casos de uso gerado.

Capítulo 5

Quase-Experimento

Há uma compreensão cada vez maior na comunidade de Engenharia de *Software* que os estudos empíricos são necessários para avaliar, desenvolver ou melhorar processos, métodos e ferramentas para desenvolvimento de *software* e manutenção dos mesmos.

Os experimentos são o centro do processo científico, pois somente os experimentos podem validar as teorias ou explorar os fatores críticos para que as teorias possam ser formuladas e corrigidas. Novos métodos, técnicas, linguagens e ferramentas não deveriam ser apresentados para venda sem experimentação e validação. É necessário que as novas invenções sejam avaliadas em comparação com as existentes. É importante ressaltar que experimentos não provam nada, eles apenas verificam a precisão da teoria junto à realidade [TRAVASSOS, GUROV e AMARAL 2002].

Em [TRAVASSOS, GUROV e AMARAL 2002], os atores supõem que a abordagem mais aceita para a experimentação na engenharia de *software* seja o método experimental, que considera a proposição e avaliação do modelo com os estudos experimentais.

Dessa maneira, apresenta-se nesse capítulo a execução de um quase-experimento utilizando a ferramenta E4J Use Cases, com o objetivo de identificar erros e falhas da mesma para realizar possíveis correções e/ou melhorias. Inicialmente, na seção 5.1, são descritos os tipos de experimentos segundo [TRAVASSOS, GUROV e AMARAL 2002]. Na seção 5.2 é apresentado todo o quase-experimento, desde sua definição e planejamento até a apresentação dos resultados. Por fim, na seção 5.3, são feitas as considerações finais deste capítulo.

5.0.1 Tipos de Experimento

Existem atualmente inúmeros tipos de classificação de experimentos. Acredita-se que esse grande número é devido ao fato de que a experimentação ainda é uma abordagem nova na área de Engenharia de *Software*. Um tipo de experimento é mais apropriado para determinada situação de acordo com, por exemplo, os objetivos do estudo ou os resultados finais esperados.

A princípio destacam-se três estratégias experimentais, as quais podem ser diferenciadas de acordo com o controle de execução, o controle de medição, o custo de investigação e a facilidade de repetição [TRAVASSOS, GUROV e AMARAL 2002]. São elas: *survey*, estudo de caso e experimento. Uma comparação entre as estratégias se encontra na Tabela 5.1.

Tabela 5.1: Comparação das estratégias experimentais.

Fator	**Survey**	**Estudo de Caso**	**Experimento**
Controle da execução	Nenhum	Nenhum	Tem
Controle da medição	Nenhum	Tem	Tem
Controle da investigação	Baixo	Médio	Alto
Facilidade da repetição	Alta	Baixa	Alto
Custo	Baixo	Média	Alto

O *survey* é uma pesquisa conduzida quando algumas técnicas ou ferramentas já tenham sido utilizadas. O principal meio para coletar as informações, sejam elas qualitativas ou quantitativas, é o questionário. Essa estratégia experimental possui os seguintes objetivos: descritivo, explanatório e explorativo.

O estudo de caso é empregado para monitorar os projetos, atividades e atribuições. Essa estratégia tem como objetivo observar um atributo específico e estabelecer o relacionamento entre atributos diferentes. O nível de controle em estudo de caso é baixo, porém, de maneira contrária ao *survey*, o estudo de caso possui o controle sobre a medição das variáveis do experimento.

Já o experimento, normalmente é realizado em laboratório oferecendo o maior nível de controle. O principal objetivo dessa estratégia é manipular uma ou mais variáveis e manter as outras fixas, medindo o efeito do resultado. Geralmente os experimentos são utilizados para confirmar teorias ou validar medidas. Esses experimentos podem ser *in-vitro* (sob condições de laboratório) ou *in-vivo* (sob condições normais).

Uma variação do experimento controlado é a estratégia experimental do tipo quase-experimento. Essa estratégia se difere do experimento na seleção dos indivíduos. En-

quanto no experimento os indivíduos são selecionados aleatoriamente, no quase-experimento é utilizado algum critério ou até mesmo alguma razão ética para a seleção dos indivíduos. [Shull, Singer e Sjoberg 2008].

De acordo com as estratégias experimentais existem três principais métodos para coleta de dados [TRAVASSOS, GUROV e AMARAL 2002]

- **Histórico**: utilizado para coletar os dados experimentais dos projetos que já tenham sido terminados. Os dados já existem e é preciso analisa-los;
- **De observação**: coleta os dados relevantes enquanto o projeto está sendo executado. Esse método oferece o controla fraco sobre o processo de desenvolvimento;
- **Controlado**: provê as instâncias multiplicas de uma observação oferecendo a validade estatística dos resultados do estudo. Em [ZELKOWITZ e WALLACE 1998], o método controlado também é classificado como: passível de repetição, sintético, passível de análise dinâmica e de simulação.

5.1 Metodologia e Experimentação

Para validar esse trabalho foi utilizada a estratégia experimental do tipo quase-experimento. Um quase-experimento é executado com a realização de cinco etapas gerais que sempre estão presentes num processo de experimentação. A definição é a primeira fase onde o quase-experimento é expresso em termos dos problemas e objetivos. A fase de planejamento vem em seguida onde o projeto do quase-experimento é determinado. A execução do quase-experimento segue o planejamento. Nesse momento os dados experimentais são coletados para serem analisados e avaliados na fase de análise e interpretação. Finalmente, os resultados são apresentados e empacotados durante a fase da apresentação e empacotamento.

Nas subseções a seguir são apresentadas as cinco etapas executadas com o quase-experimento da ferramenta E4J Use Cases.

5.1.1 Primeira e segunda etapa: Definição e Planejamento

A fase de definição descreve os objetivos, o objeto de estudo, o foco da qualidade, o ponto de vista e o contexto. Na fase de planejamento acontecem a seleção do contexto, dos partici-

pantes, o projeto do quase-experimento e a preparação conceitual da instrumentação. Após o planejamento estar concluído, temos o quase-experimento totalmente elaborado e pronto para execução [TRAVASSOS, GUROV e AMARAL 2002].

O foco da experimentação foi encontrar erros e fraquezas da ferramenta E4J Use Cases com o intuito de realizar possíveis correções e/ou melhorias na mesma.

Dessa maneira, utilizou-se a abordagem *Goal/Question/Metric* (GQM) [SOLINGEN e BERGHOUT 1999] para apoiar essas duas fases iniciais do quase-experimento, a qual nos ajuda tanto a definir quanto planejar o quase-experimento. O GQM é uma abordagem orientada a metas e utilizada em Engenharia de *Software* para a medição de produtos e processos de *software*. O GQM é baseado no requisito de que toda a coleta de dados deve ser baseada num fundamento lógico, em um objetivo ou meta, que é documentado explicitamente.

Desta forma, a representação da estrutura da experimentação é da seguinte forma:

- **Objetivo global**: verificar a corretude da sintaxe dos elementos gráficos da ferramenta E4J Use Cases segundo os padrões UML bem como da rotina de mapeamento dos casos de uso do JGOOSE para a representação diagramática no E4J Use Cases e também o verificar o grau de coerência e utilidade do diagrama de casos de uso gerado pelo E4J Use Cases;

- **Objetivo do estudo**:

  Analisar os elementos gráficos e a rotina de mapeamento do E4J Use Cases

  Com a finalidade de verificá-los

  Com respeito à sintaxe dos elementos gráficos segundo os padrões UML e à equivalência do diagrama de casos de uso gerado com base nos casos de uso do JGOOSE bem como à coerência e utilidade desse diagrama

  Do ponto de vista dos stakeholders

  No contexto dos discentes da disciplina de Processo de Engenharia de *Software* I (PES I) do curso de Ciência da Computação da Unioeste.

- **Questões (Q)/Métricas (M)**:

Grupo 1 - Questões sobre a sintaxe segundo os padrões UML

Q01: Todos os elementos gráficos representando os atores no diagrama de casos de uso são bonecos contendo o nome do ator?

M01: Todos os elementos gráficos representando os atores no diagrama de casos de uso são bonecos contendo o nome do ator.

Q02: Todos os elementos gráficos representando os casos de uso no diagrama de casos de uso são elipses contendo o nome do caso de uso?

M02: Todos os elementos gráficos representando os casos de uso no diagrama de casos de uso são elipses contendo o nome do caso de uso.

Q03: Todas as ligações entre ator e casos de uso no diagrama de casos de uso são do tipo *association* (ligação contínua)?

M03: Todas as ligações entre ator e casos de uso no diagrama de casos de uso são do tipo *association*(ligação contínua).

Q04: Todas as ligações entre atores no diagrama de casos de uso são do tipo *generalization* (ligação contínua com seta em um dos extremos)?

M04: Todas as ligações entre atores no diagrama de casos de uso são do tipo *generalization* (ligação contínua com seta em um dos extremos).

Q05: Todas as ligações entre casos de uso no diagrama de casos de uso são do tipo *« include »* (ligação tracejada com seta em um dos extremos e contendo a label *« include »*) ou do tipo *« extend »* (ligação tracejada com seta em um dos extremos e contendo a label *« extend »*)?

M05: Todas as ligações entre casos de uso no diagrama de casos de uso são do tipo *« include »* (ligação tracejada com seta em um dos extremos e contendo a label *« include »*)

ou do tipo *« extend »* (ligação tracejada com seta em um dos extremos e contendo a label *« extend »*).

Grupo 2 - Questões sobre a rotina de mapeamento dos casos de uso do JGOOSE para o diagrama de casos de uso no E4J Use Cases

Q06: Todos os atores gerados no diagrama do E4J Use Cases são equivalentes aos atores presentes nas descrições textuais e/ou tabela dos casos de uso do JGOOSE?

M06: Todos os atores gerados no diagrama do E4J Use Cases são equivalentes aos atores presentes nas descrições textuais e/ou tabela dos casos de uso do JGOOSE.

Q07: Todos os casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos casos de uso presentes na tabela de casos de uso do JGOOSE?

M07: Todos os casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos casos de uso presentes na tabela de casos de uso do JGOOSE.

Q08: Todas as ligações com o estereótipo *« include »* entre os casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos estereótipos *« include »* presentes nas descrições textuais do cenário principal de sucesso (MAIN SUCESS CENARIO) dos casos de uso no JGOOSE?

M08: Todas as ligações com o estereótipo *« include »* entre casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos estereótipos *« include »* presentes nas descrições textuais do cenário principal de sucesso (MAIN SUCESS CENARIO) dos casos de uso no JGOOSE.

Q09: Todas as ligações com o estereótipo *« extend »* entre os casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos estereótipos *« extend »* presentes nas descrições textuais de extensões (*EXTENSIONS*) dos casos de uso no JGOOSE?

M09: Todas as ligações com o estereótipo *« extend »* entre os casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos estereótipos *« extend »* presentes nas descrições textuais de extensões (*EXTENSIONS*) dos casos de uso no JGOOSE.

Q10: Todas as ligações do tipo *generalization* entre atores gerados no diagrama do E4J Use Cases são equivalentes as ligações do tipo *generalization* presentes no JGOOSE (botão "*Show ISAs*")?

M10: Todas as ligações do tipo *generalization* entre atores gerados no diagrama do E4J Use Cases são equivalentes as ligações do tipo *generalization* presentes no JGOOSE (botão "*Show ISAs*").

Q11: Todas as ligações de associação entre atores e seus casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos atores e seus casos de uso presentes na tabela de casos de uso do JGOOSE?

M11: Todas as ligações de associação entre atores e seus casos de uso gerados no diagrama do E4J Use Cases são equivalentes aos atores e seus casos de uso presentes na tabela de casos de uso do JGOOSE.

Grupo 3 - Questões sobre a coerência e utilidade do diagrama de casos de uso gerado pelo E4J Use Cases

Q12: Todos os atores gerados no diagrama de casos de uso do E4J Use Cases realmente interagem através de alguma dependência com o sistema computacional nos modelos i*?

M12: Todos os atores gerados no diagrama de casos de uso do E4J Use Cases realmente interagem através de alguma dependência com o sistema computacional nos modelos i*.

Q13: Todos os casos de uso gerados no diagrama de casos de uso do E4J Use Cases realmente são dependências (requisitos funcionais) de atores com o sistema computacional nos modelos i*?

M13: Todos os casos de uso gerados no diagrama de casos de uso do E4J Use Cases realmente são dependências (requisitos funcionais) de atores com o sistema computacional nos modelos i*.

Q14: Todas as ligações com o estereótipo *« include »* entre casos de uso geradas no diagrama do E4J Use Cases realmente fazem sentido?

M14: Todas as ligações com o estereótipo *« include »* entre casos de uso geradas no diagrama do E4J Use Cases realmente fazem sentido.

Q15: Todas as ligações com o estereótipo *« extend »* entre casos de uso geradas no diagrama do E4J Use Cases realmente fazem sentido?

M15: Todas as ligações com o estereótipo *« extend »* entre casos de uso geradas no diagrama do E4J Use Cases realmente fazem sentido.

Q16: O processo automatizado de, a partir dos modelos i*, mapear casos de uso UML no JGOOSE e posterior geração do diagrama de casos de uso no E4J Use Cases foi útil e vantajoso? Por quê?

M16: O processo automatizado de, a partir dos modelos i*, mapear casos de uso UML no JGOOSE e posterior geração do diagrama de casos de uso no E4J Use Cases foi útil e vantajoso.

Q17: Ocorreu algum erro durante a utilização do E4J Use Cases? Quais?

M17: Não ocorreram erros durante a utilização do E4J Use Cases.

Q18: Há diferenças entre o diagrama de casos de uso gerado pelo E4J Use Cases e o diagrama de casos de uso que você desenvolveu?

M18: Houve diferenças entre o diagrama de casos de uso gerado pelo E4J Use Cases e o diagrama de casos de uso que você desenvolveu.

Cabe ressaltar que os discentes selecionados para a experimentação já possuíam grupos de trabalhos formados na disciplina de PES I, o que facilitou a fase inicial de execução, pois esses grupos já haviam realizado uma modelagem utilizando o *framework* i* para o trabalho da disciplina. Dessa forma, não foi necessário que os discentes realizassem uma nova modelagem para a execução do quase-experimento. A seleção desses grupos para a execução da experimentação caracteriza a mesma como um quase-experimento [Shull, Singer e Sjoberg 2008].

Para aplicar o questionário GQM, foi planejada a seguinte sistemática:

1. Inicialmente realiza-se um exemplo da utilização da ferramenta E4J Use Cases. Para isso, utiliza-se um modelo SR construído via E4J i* e mapeiam-se os casos de uso desse modelo na ferramenta JGOOSE. Planeja-se gastar 3 minutos para essa etapa;

2. Posteriormente, gera-se o diagrama de casos de uso na ferramenta E4J Use Cases utilizando a rotina de mapeamento apresentada na seção 4.3.2 do capítulo 4. Planeja-se gastar 2 minutos para essa etapa;

3. Com o diagrama de casos de uso gerado, explica-se o que cada questão do GQM deve responder. Planeja-se gastar 10 minutos para essa etapa;

4. Após essa explicação, os grupos realizam o experimento. São grupos de trabalho da disciplina de Processo de Engenharia de *Software* I (PES I) do curso de Ciência da Computação da Unioeste. Cada grupo deve utilizar o modelo SR já construído na disciplina de PES I para mapear os casos de uso no JGOOSE e gerar o diagrama de casos de uso no E4J Use Cases. Planeja-se gastar 5 minutos para essa etapa;

5. Os grupos então devem responder os questionários utilizando como base o diagrama de casos de uso gerado, os casos de uso mapeados pelo JGOOSE e o modelo SR utilizado. Planeja-se gastar 75 minutos para essa etapa;

6. Os grupos devem submeter no sistema Moodle os seguintes arquivos utilizados e/ou gerados durante o experimento: modelo SR no formato .mxe (formato do E4J i*), casos de uso mapeados pelo JGOOSE no formato .doc, diagrama de casos de uso nos formatos de imagem e .mxe (formato do E4J Use Cases). Planeja-se gastar 5 minutos para essa etapa.

5.1.2 Terceira etapa: Execução

O aspecto mais importante da fase de execução é que a parte humana deve ser considerada. Os participantes devem ser preparados para a experimentação do ponto de vista moral e metodológico para evitar os resultados errôneos devido ao mal-entendido ou falta de interesse [TRAVASSOS, GUROV e AMARAL 2002].

Dessa maneira, foi executado o quase-experimento seguindo as definições e planejamento definido na seção anterior. A explicação e exemplo inicial do quase-experimento foram dados justamente para preparar os discentes metodologicamente e evitar mal-entendidos sobre as questões. Os grupos 1 e 5 eram formados por dois discentes cada e os grupos 2, 3 e 4 eram formados por três discentes cada. O tempo gasto para a execução do quase-experimento foi equivalente ao tempo planejado. Cabe ressaltar que eventuais dúvidas sobre as questões durante a execução do experimente foram tiradas e repassadas a todos os grupos.

Executado o quase-experimento, realizou-se a coleta de dados via sistema Moodle e os mesmos são analisados e interpretados na próxima seção.

5.1.3 Quarta etapa: Análise e Interpretação

De posse dos dados obtidos via sistema Moodle, apresenta-se na Tabela 5.2 os resultados das questões de cada grupo conforme as métricas definidas para as mesmas.

Tabela 5.2: Resultados das questões conforme as métricas.

MÉTRICA	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
M01	Sim	Sim	Sim	Sim	Sim
M02	Sim	Sim	Sim	Sim	Sim
M03	Sim	Sim	Sim	Sim	Sim
M04	Sim	Não aplicável	Sim	Sim	Não aplicável
M05	Não aplicável	Não aplicável	Não aplicável	Não aplicável	Não aplicável
M06	Sim	Sim	Sim	Sim	Sim
M07	Sim	Sim	Sim	Sim	Sim
M08	Não aplicável	Não aplicável	Não aplicável	Não aplicável	Não aplicável
M09	Não aplicável	Não aplicável	Não aplicável	Não aplicável	Não aplicável
M10	Sim	Não aplicável	Sim	Sim	Não aplicável
M11	Sim	Sim	Sim	Sim	Sim
M12	Sim	Sim	Sim	Sim	Sim
M13	Sim	Sim	Sim	Sim	Sim
M14	Não aplicável	Não aplicável	Não aplicável	Não aplicável	Não aplicável
M15	Não aplicável	Não aplicável	Não aplicável	Não aplicável	Não aplicável
M16	Sim	Sim	Sim	Sim	Sim
M17	Sim	Sim	Sim	Sim	Sim
M18	Sim	Sim	Sim	Sim	Sim

Análise quantitativa

Com base no resultado do GQM apresentado na Tabela 5.2 realizou-se uma análise quantitativa sobre os dados referente a cada questão:

Grupo 1 - Questões sobre a sintaxe segundo os padrões UML

- **Q01**: todos os cinco grupos geraram diagramas de casos de uso onde todos os elementos gráficos representando os atores no diagrama de casos de uso são bonecos contendo o nome do ator. Assim, o E4J Use Cases apresentou 100% de corretude em relação à geração do elemento gráfico do tipo Ator;

- **Q02**: todos os cinco grupos geraram diagramas de casos de uso onde todos os elementos gráficos representando os casos de uso no diagrama de casos de uso são elipses contendo o nome do caso de uso. Assim, o E4J Use Cases apresentou 100% de corretude em relação à geração do elemento gráfico do tipo Caso de Uso;

- **Q03**: todos os cinco grupos geraram diagramas de casos de uso onde todas as ligações entre ator e casos de uso no diagrama de casos de uso são do tipo *association* (ligação contínua). Assim, o E4J Use Cases apresentou 100% de corretude em relação à geração do elemento gráfico do tipo ligação *association*;

- **Q04**: três grupos geraram diagramas de casos de uso onde todas as ligações entre atores no diagrama de casos de uso são do tipo *generalization* (ligação contínua com seta em um dos extremos). Os grupos 2 e 5 não obtiveram nenhuma ligação do tipo *generalization* em seus diagramas de casos de uso, portanto, a métrica M04 não foi aplicável para esses grupos. Assim, o E4J Use Cases apresentou 100% de corretude em relação à geração do elemento gráfico do tipo ligação *generalization*;

- **Q05**: nenhuns dos grupos geraram diagramas de casos de uso contendo ligações do tipo *« include »* e/ou *« extend »*. Assim, não foi possível verificar a corretude desse tipo de ligação gerada entre casos de uso.

Grupo 2 - Questões sobre a rotina de mapeamento dos casos de uso do JGOOSE para o diagrama de casos de uso no E4J Use Cases

- **Q06**: todos os cinco grupos geraram diagramas de casos de uso onde todos os atores gerados são equivalentes aos atores presentes nas descrições textuais e/ou tabela de casos de uso do JGOOSE. Assim, o E4J Use Cases apresentou 100% de equivalência em relação à geração dos atores presentes nas descrições dos casos de uso do JGOOSE;

- **Q07**: todos os cinco grupos geraram diagramas de casos de uso onde todos os casos de uso gerados são equivalentes aos casos de uso presentes na tabela dos casos de uso no JGOOSE. Assim, o E4J Use Cases apresentou 100% de equivalência em relação à geração dos casos de uso presentes na tabela de casos de uso no JGOOSE;

- **Q08**: como nenhum dos grupos gerou diagramas de casos de uso contendo ligações do tipo *« include »*, a métrica M08 não foi aplicável aos grupos. Assim, não foi possível verificar a equivalência em relação à geração das ligações do tipo *« include »* presentes nas descrições textuais dos casos de uso no JGOOSE;

- **Q09**: como nenhum dos grupos gerou diagramas de casos de uso contendo ligações do tipo *« extend »*, a métrica M09 não foi aplicável aos grupos. Assim, não foi possível verificar a equivalência em relação à geração das ligações do tipo *« extend »* presentes nas descrições textuais dos casos de uso no JGOOSE;

- **Q10**: três grupos geraram diagrama de casos de uso onde todas as ligações do tipo *generalization* entre atores geradas são equivalentes às ligações do tipo *generalization* presentes no JGOOSE (botão "*Show ISAs*"). Os grupos 2 e 5 não obtiveram nenhuma ligação do tipo *generalization* em seu diagrama de casos de uso, portanto, a métrica M10 não foi aplicável a esses grupos. Assim, o E4J Use Cases apresentou 100% de equivalência em relação à geração das ligações do tipo *generalization* presentes no JGOOSE;

- **Q11**: todos os cinco grupos geraram diagramas de casos de uso onde todas as ligações de associação entre atores e seus casos de uso gerados são equivalentes aos atores e seus casos de uso presentes na tabela de casos de uso do JGOOSE. Assim, o E4J Use Cases apresentou 100% de equivalência em relação à geração das ligações do tipo *association* entre atores e seus casos de uso presentes na tabela de casos de uso no JGOOSE;

Grupo 3 - Questões sobre a coerência e utilidade do diagrama de casos de uso gerado pelo E4J Use Cases

- **Q12**: todos os cinco grupos geraram diagramas de casos de uso onde todos os atores gerados realmente interagem através de alguma dependência com o sistema computacional nos modelos i*. Assim, o E4J Use Cases apresentou 100% de coerência em relação à geração dos atores que realmente interagem através de alguma dependência com o sistema computacional nos modelos i*;

- **Q13**: todos os cinco grupos geraram diagramas de casos de uso onde todos os casos de uso gerados realmente são dependências (requisitos funcionais) de atores com o sistema computacional nos modelos i*. Assim, o E4J Use Cases apresentou 100% de coerência em relação à geração dos casos de uso que realmente são dependências (requisitos funcionais) de atores com o sistema computacional nos modelos i*;

- **Q14**: como nenhum dos grupos gerou diagramas de casos de uso contendo ligações do tipo *« include »*, a métrica M14 não foi aplicável aos grupos. Assim, não foi possível verificar a coerência em relação à geração das ligações do tipo *« include »*;

- **Q15**: como nenhum dos grupos gerou diagramas de casos de uso contendo ligações do tipo *« extend »*, a métrica M15 não foi aplicável aos grupos. Assim, não foi possível

verificar a coerência em relação à geração das ligações do tipo *« extend »*;

- **Q16**: todos os grupos descreveram que o processo automatizado de, a partir dos modelos i*, mapear casos de uso UML no JGOOSE e posterior geração do diagrama de casos de uso no E4J Use Cases foi útil e vantajoso;

- **Q17**: todos os grupos descreveram que não ocorreram erros durante a utilização do E4J Use Cases;

- **Q18**: todos os grupos descreveram que houve diferenças entre o diagrama de casos de uso gerado pelo E4J Use Cases e o diagrama de casos de uso que desenvolveram sem a ferramenta.

Análise qualitativa

Com base na análise quantitativa e nos resultados presentes na Tabela 5.2 é possível fazer uma análise qualitativa sobre o comportamento da ferramenta E4J Use Cases.

Inicialmente, tendo em vista a corretude da sintaxe dos elementos gráficos e observando o resultado das questões **Q01**, **Q02**, **Q03** e **Q04**, verificou-se que o E4J Use Cases se encontra estável e gerando corretamente os elementos gráficos dos tipos Ator, Caso de Uso, Ligação *association* e Ligação *generalization*. Observando o resultado da questão **Q05**, não foi possível verificar a corretude da geração das ligações do tipo *« include »* e *« extend »* pois não houve nenhuma modelagem que resultava na geração desses tipos de ligações no diagrama de casos de uso. Entretanto, verificou-se que como o modelo não resultava na geração desses tipos de ligações, o E4J Use Cases não gerou incorretamente esse tipo de ligação nos diagramas, o que, caso fosse gerado, seria uma inconsistência.

Em relação à corretude da rotina de mapeamento dos casos de uso do JGOOSE para o diagrama de casos de uso no E4J Use Cases e observando as questões **Q06** e **Q07**, verificou-se que o E4J Use Cases também se encontra estável e que todos os atores e todos os casos de uso gerados são equivalentes aos atores e casos de uso presentes no JGOOSE.

Observando o resultado das questões **Q08** e **Q09**, novamente não foi possível verificar a equivalência da geração das ligações do tipo *« include »* e *« extend »* em relação aos presentes nos casos de uso do JGOOSE, pois não houve nenhum caso de uso utilizando esse estereó-

tipo incluindo ou extendendo outro caso de uso. Entretanto, verificou-se que o E4J Use Cases também não gerou incorretamente esse tipo de ligação nos diagramas, pois caso fosse gerado, novamente seria uma inconsistência.

Observando o resultado das questões **Q10** e **Q11**, verificou-se que o E4J Use Cases também se encontra estável e que todas as ligações do tipo *generalization* (para os grupos que tiverem esse tipo de ligação) são equivalentes em relação aos ISA's presentes no JGOOSE e que todas as ligações do tipo *association* são equivalentes aos atores e seus casos de uso presentes no JGOOSE.

Por fim, em relação à coerência e utilidade do diagrama de casos de uso gerado pelo E4J Use Cases e observando o resultado das questões **Q12** e **Q13**, verificou-se que o E4J Use Cases também se encontra estável e coerente, sendo que todos os atores gerados realmente interagem através de alguma dependência com o sistema computacional nos modelos i* bem como os casos de uso realmente são dependências de atores com o sistema.

Observando o resultado das questões **Q14** e **Q15**, novamente não foi possível verificar se as ligações do tipo *« include »* e *« extend »* realmente fazem sentido no contexto de suas modelagens, já que não houve esse tipo de ligação nos diagramas gerados.

Observando o resultado da questão **Q16**, todos os grupos descreveram que o processo automatizado de, a partir dos modelos i*, mapear casos de uso UML no JGOOSE e posterior geração do diagrama de casos de uso no E4J Use Cases foi útil e vantajoso. Nessa questão, o grupo 1 destacou que através da ferramenta E4J Use Cases conseguiram alterar e visualizar melhor os casos de uso. O grupo 2 destacou que esse processo foi muito útil e rápido, facilitando a análise do diagrama de casos de uso. O grupo 3 destacou que a ferramenta E4J Use Cases facilita a manipulação dos diagramas por ser compatível com os sistemas operacionais Linux e Windows. O grupo 4 destacou que o E4J Use Cases facilita a apresentação do diagrama para o cliente, descrevendo que o modelo i* as vezes pode ser um tanto quanto confuso e o grupo 5 destacou que a geração dos casos de uso em formato de texto foi muito útil para a elaboração da segunda parte do trabalho de PES I.

Observando o resultado da questão **Q17**, verificou-se que o E4J Use Cases novamente se encontrou estável não ocorrendo nenhum erro durante a execução do quase-experimento. Finalmente, observando a questão **Q18**, todos os grupos descreveram que houve diferenças entre

o diagrama de casos de uso gerado pelo E4J Use Cases e o diagrama de casos de uso que os grupos criaram. Como a ferramenta E4J Use Cases permite a manipulação dos diagramas de casos de uso gerados, os usuários podem manipular esses diagramas da forma que desejarem, sendo que o diagrama gerado pode ser uma base para a construção de um diagrama mais rico e completo.

Apesar do E4J Use Cases não ter apresentado nenhuma inconsistência e/ou incoerência bem como nenhum erro durante o quase-experimento, foi identificado um erro na rotina de mapeamento durante a reprodução da geração do diagrama de casos de uso do grupo 5. Caso houvesse um caso de uso que era comum para dois ou mais atores, o E4J Use Cases gerava um caso de uso para cada ator no diagrama de casos de uso, sendo que deve ser gerado apenas um caso de uso e todos os atores que possuem aquele caso de uso devem ser ligado ao mesmo. Esse erro foi corrigido na rotina de mapeamento do E4J Use Cases.

5.1.4 Quinta etapa: Apresentação e Empacotamento

Segundo [TRAVASSOS, GUROV e AMARAL 2002], essa é a fase menos elaborada da metodologia da experimentação na área de Engenharia de Software e o autor descreve que o empacotamento dos dados é necessário para ser possível realizar a repetição do quase-experimento, o qual é uma das características mais importantes da experimentação.

Dessa maneira, são apresentados nas Figuras 5.1, 5.2, 5.3, 5.4 e 5.5 os modelos SR e nas Figuras 5.6, 5.7, 5.8, 5.9 e 5.10 os diagramas de casos de uso gerados pelos grupos 1, 2, 3, 4 e 5, respectivamente.

Com esses modelos SR, qualquer usuário pode construí-los na ferramenta E4J i*, mapear os casos de uso no JGOOSE e gerar o diagrama de casos de uso na ferramenta E4J Use Cases, como é exemplificado na seção 4.4 do capítulo 4, reproduzindo assim o quase-experimento e podendo verificar se os resultados são equivalentes aos diagramas de casos de uso aqui apresentados.

5.2 Considerações Finais do Capítulo

Neste capítulo foi apresentado o quase-experimento usando a ferramenta E4J Use Cases e através desse quase-experimento, pode-se verificar a corretude dos elementos gráficos do E4J Use Cases, a equivalência do diagrama de casos de uso gerado pela rotina de mapeamento com

base nos casos de uso do JGOOSE bem como o qual útil esse processo automatizado é para os usuários do ambiente de trabalho do JGOOSE.

A ferramenta E4J Use Cases teve um ótimo desempenho e foi identificado apenas um erro na rotina de mapeamento, o qual já foi corrigido.

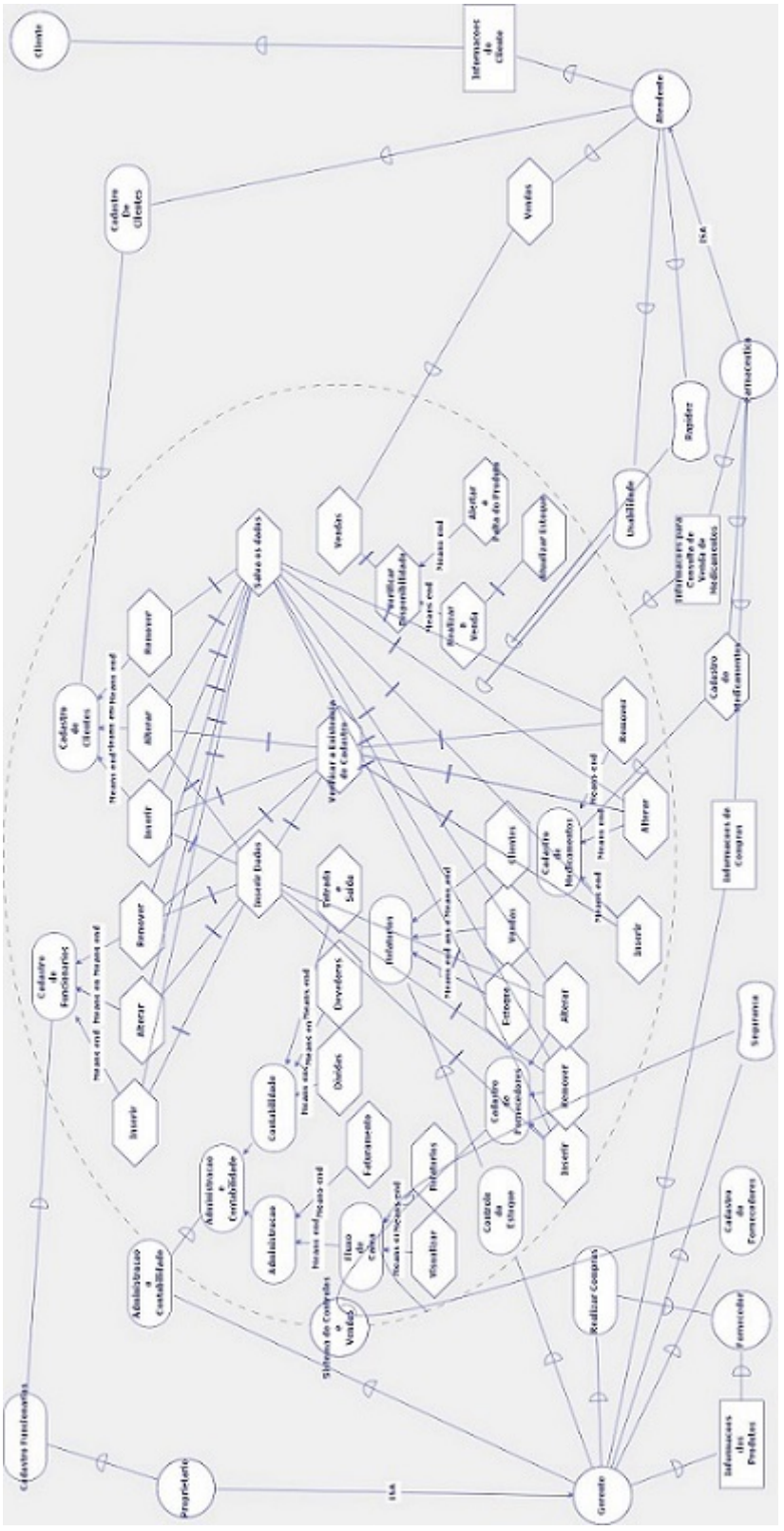


Figura 5.1: Modelo SR do Grupo 1

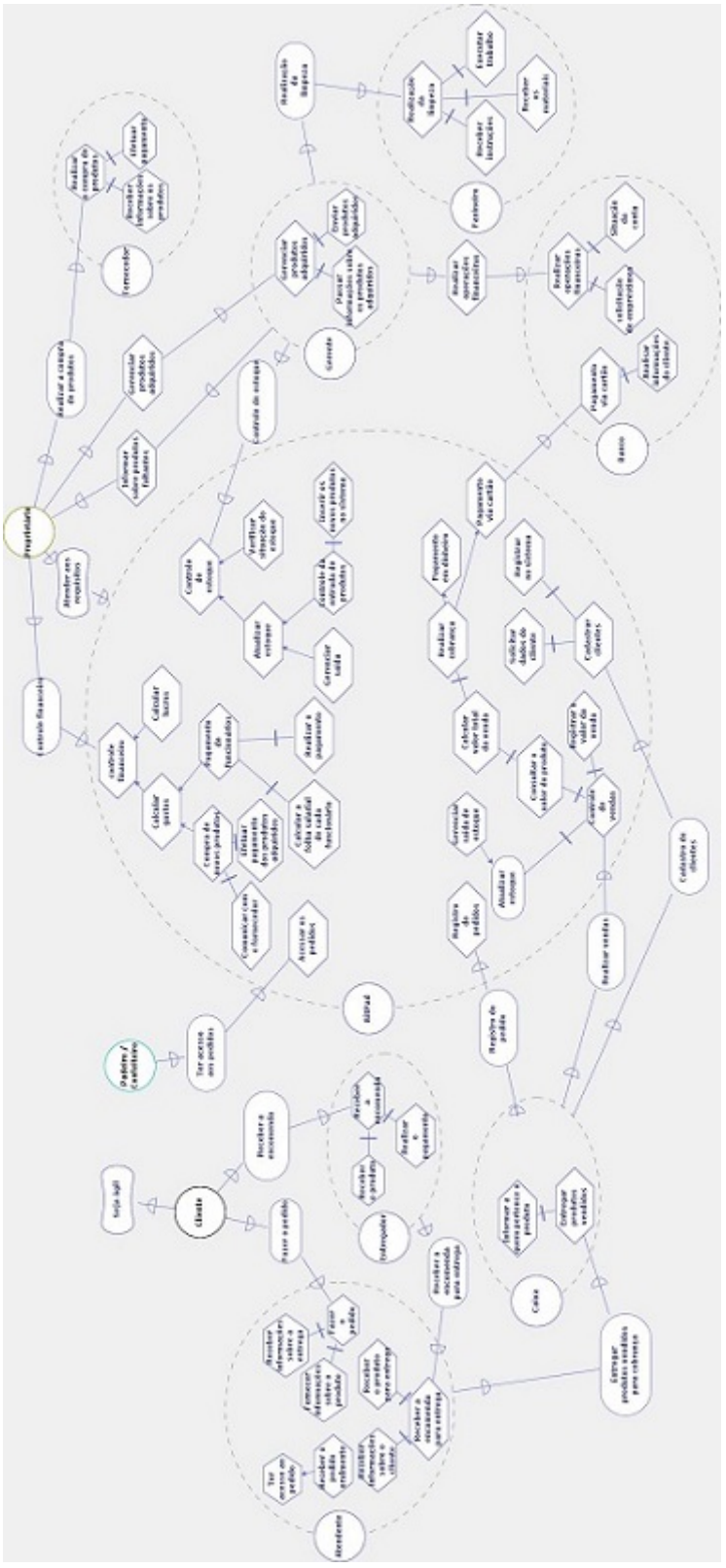


Figura 5.2: Modelo SR do Grupo 2

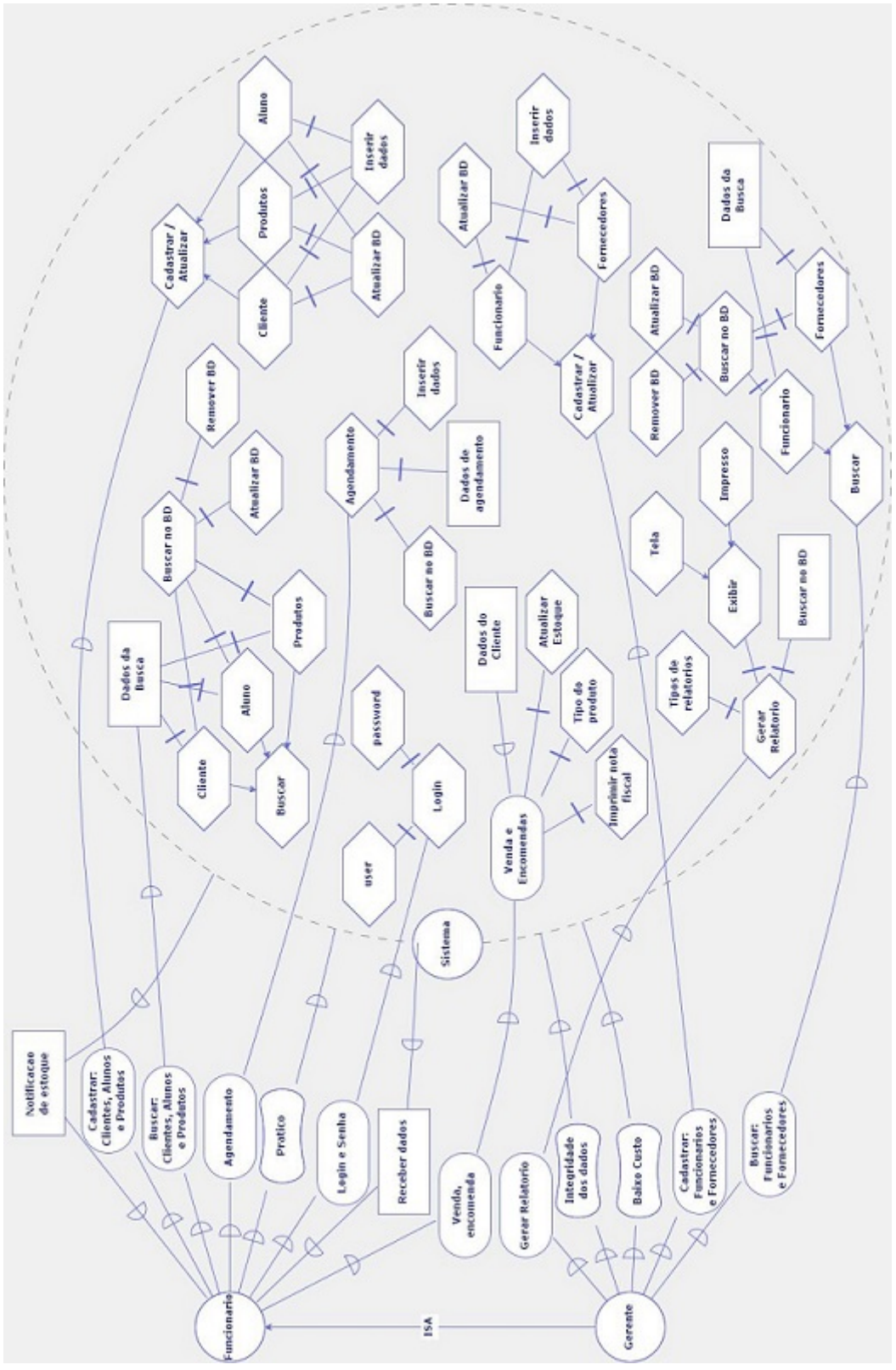


Figura 5.3: Modelo SR do Grupo 3

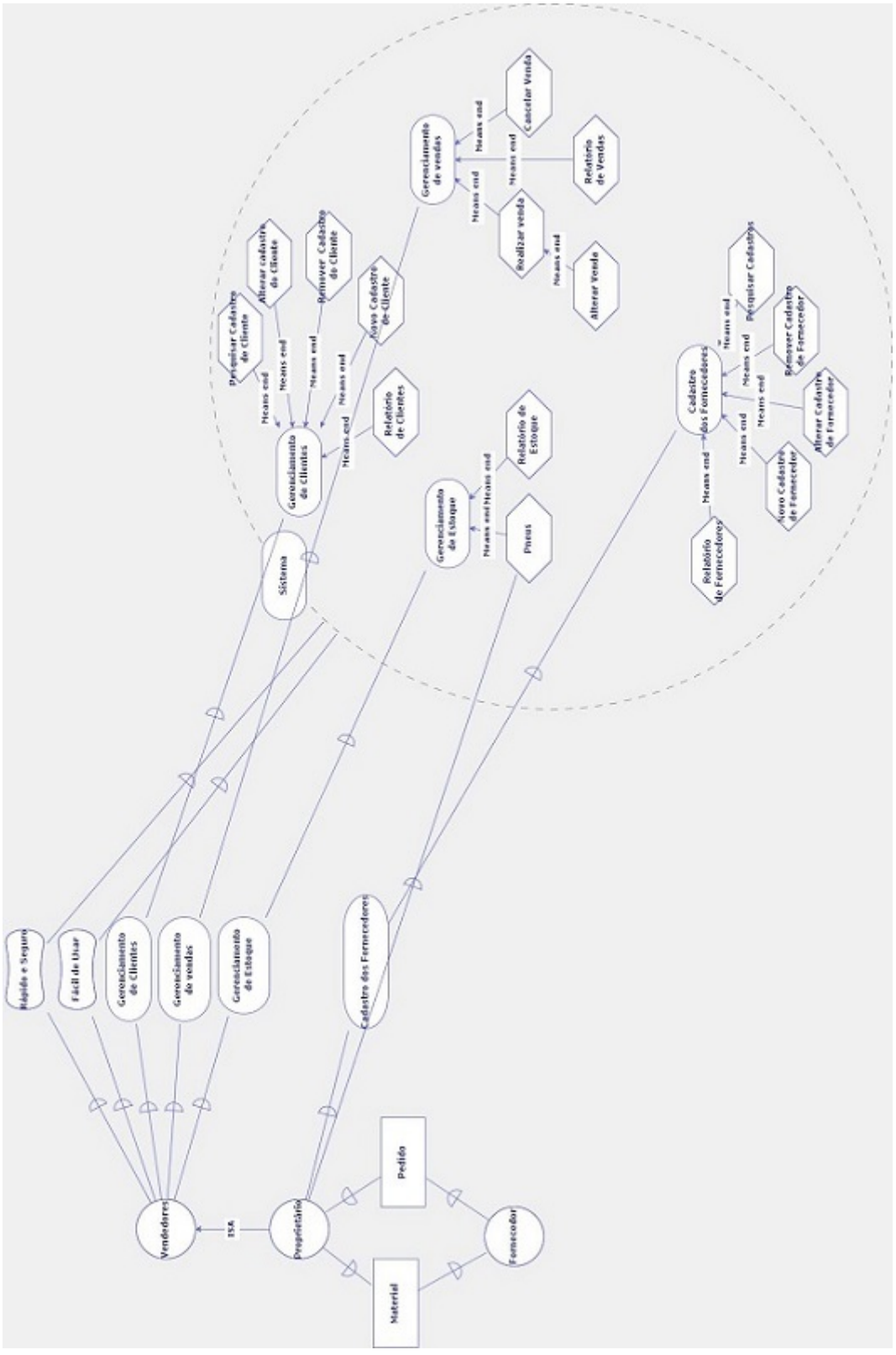


Figura 5.4: Modelo SR do Grupo 4

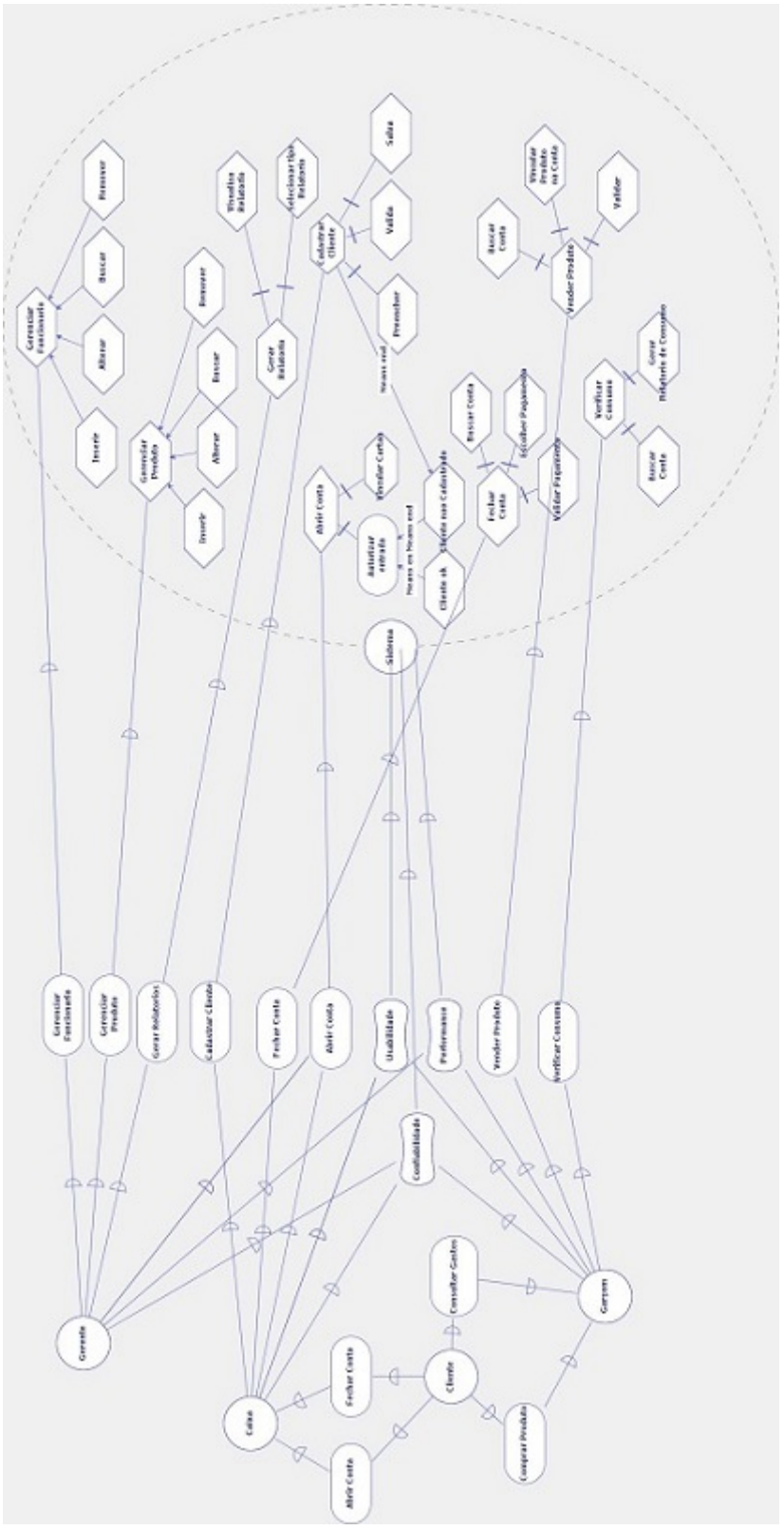


Figura 5.5: Modelo SR do Grupo 5

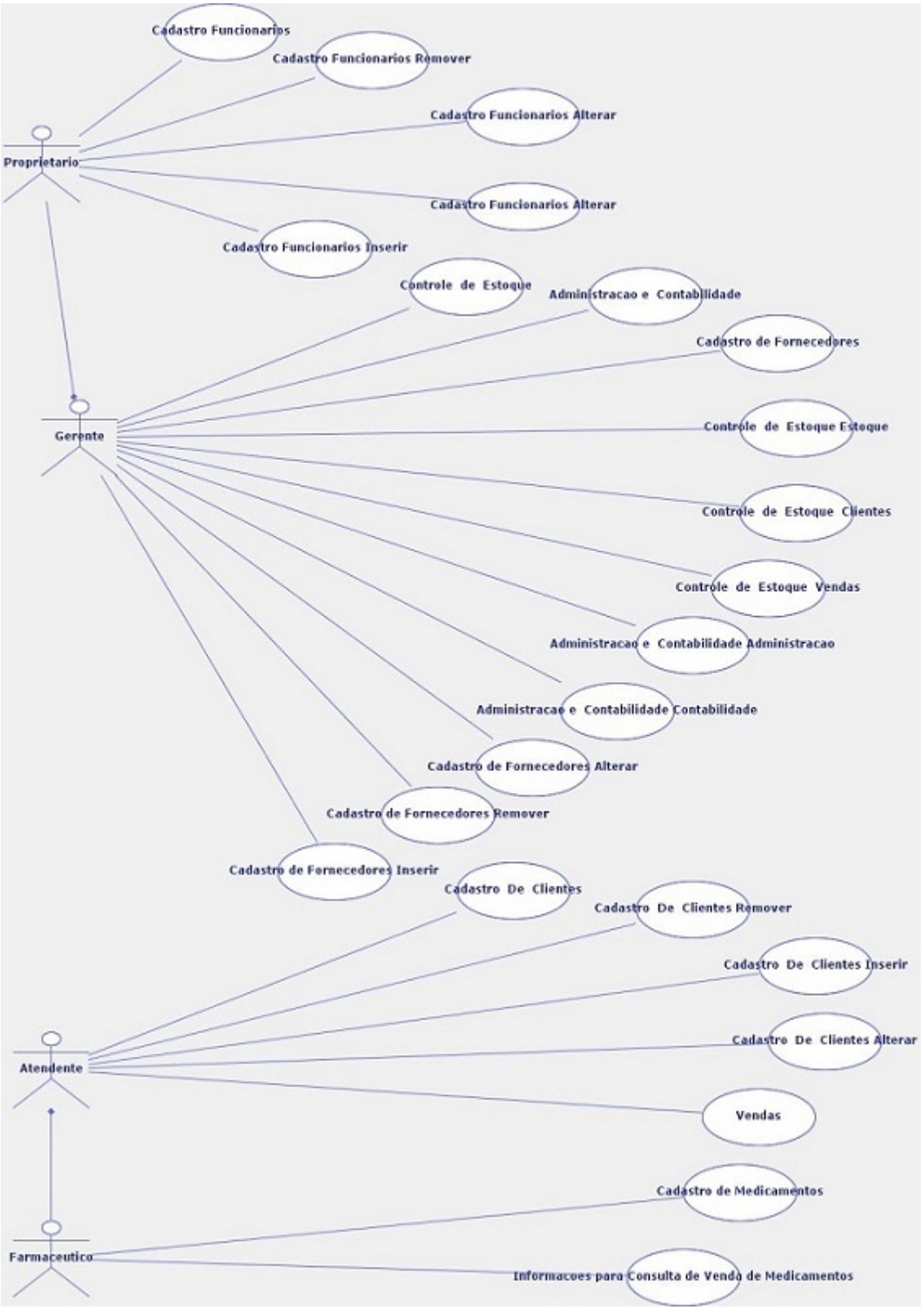


Figura 5.6: Diagrama de casos de uso gerado pelo Grupo 1

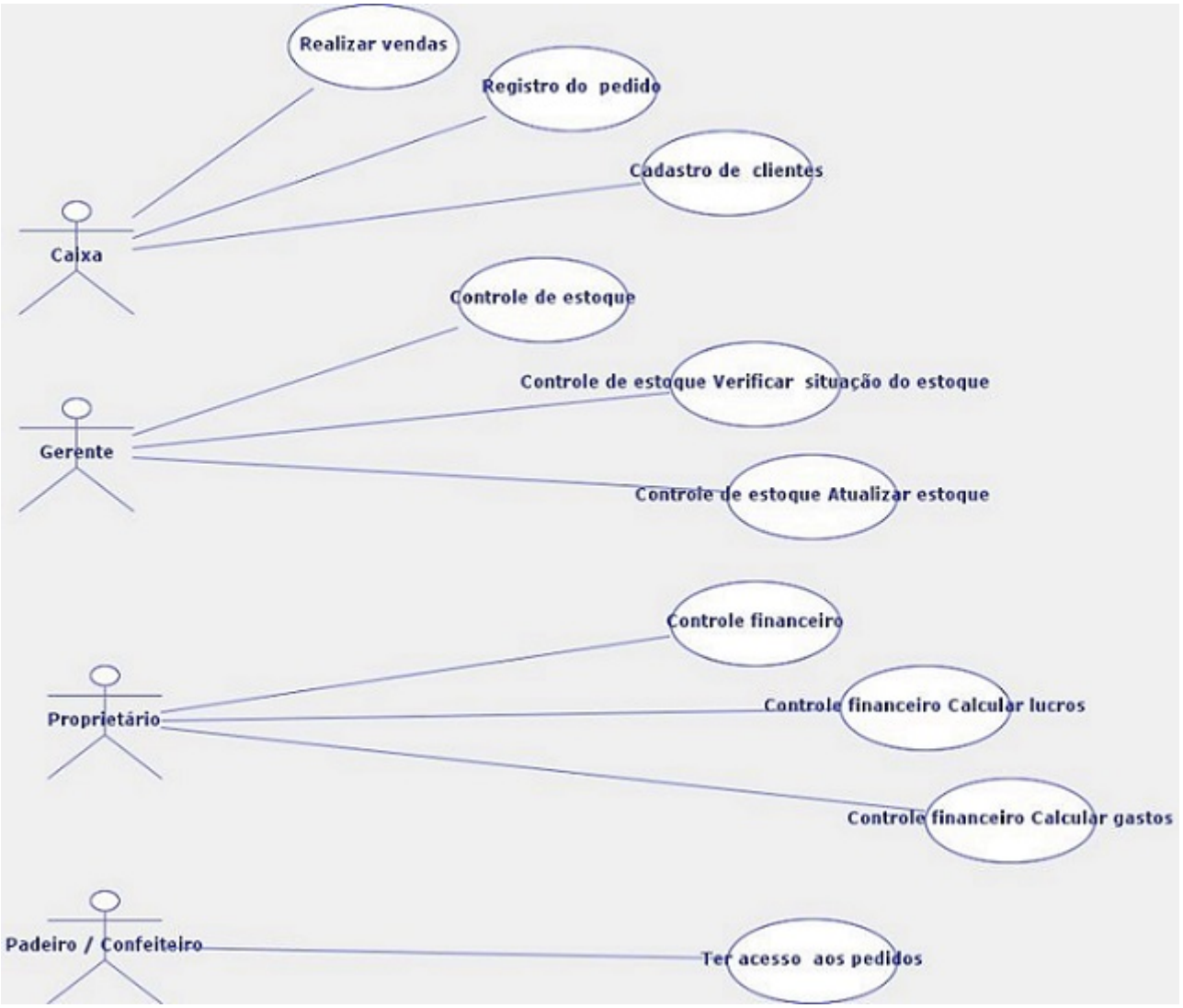


Figura 5.7: Diagrama de casos de uso gerado pelo Grupo 2

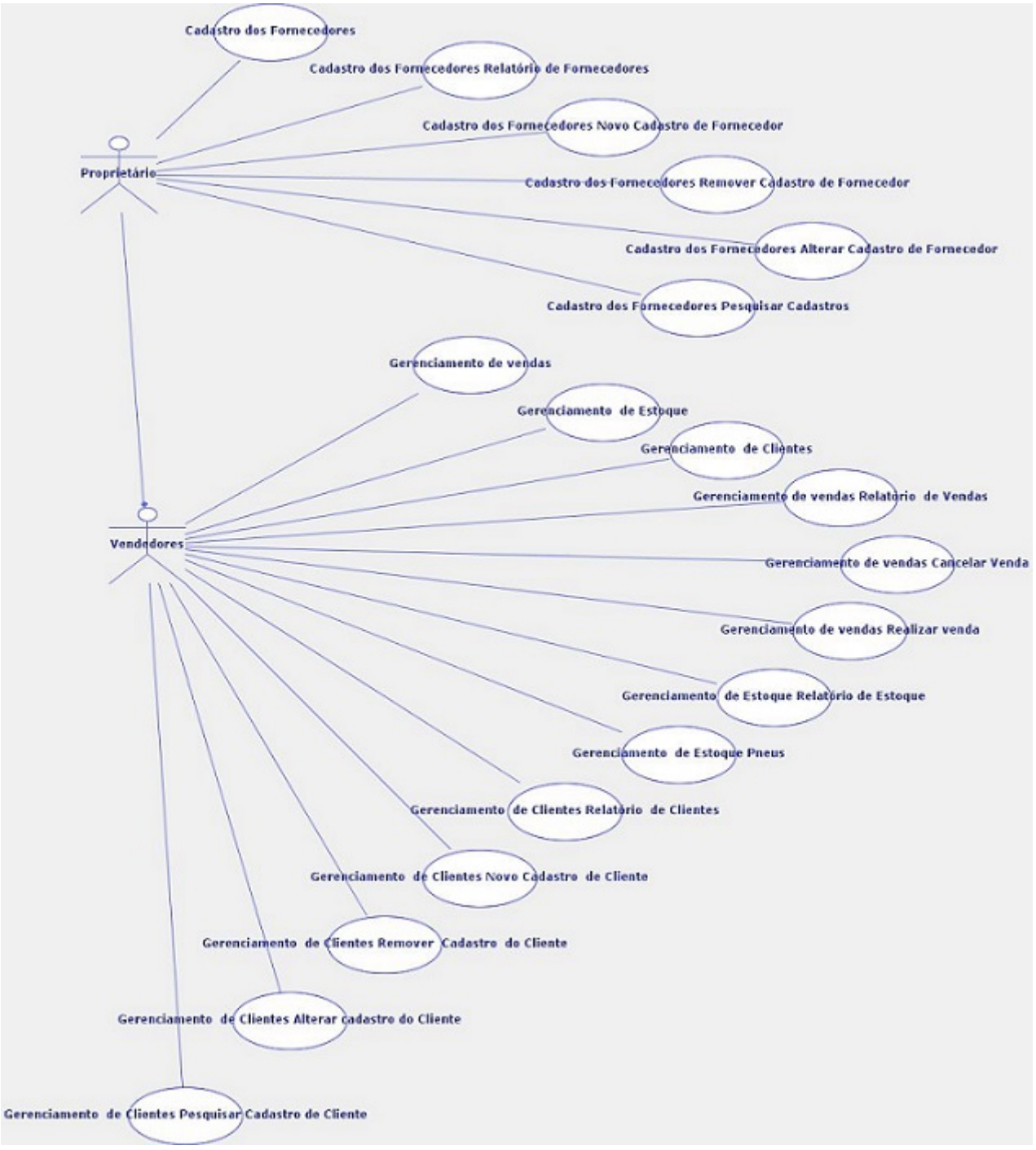


Figura 5.8: Diagrama de casos de uso gerado pelo Grupo 3

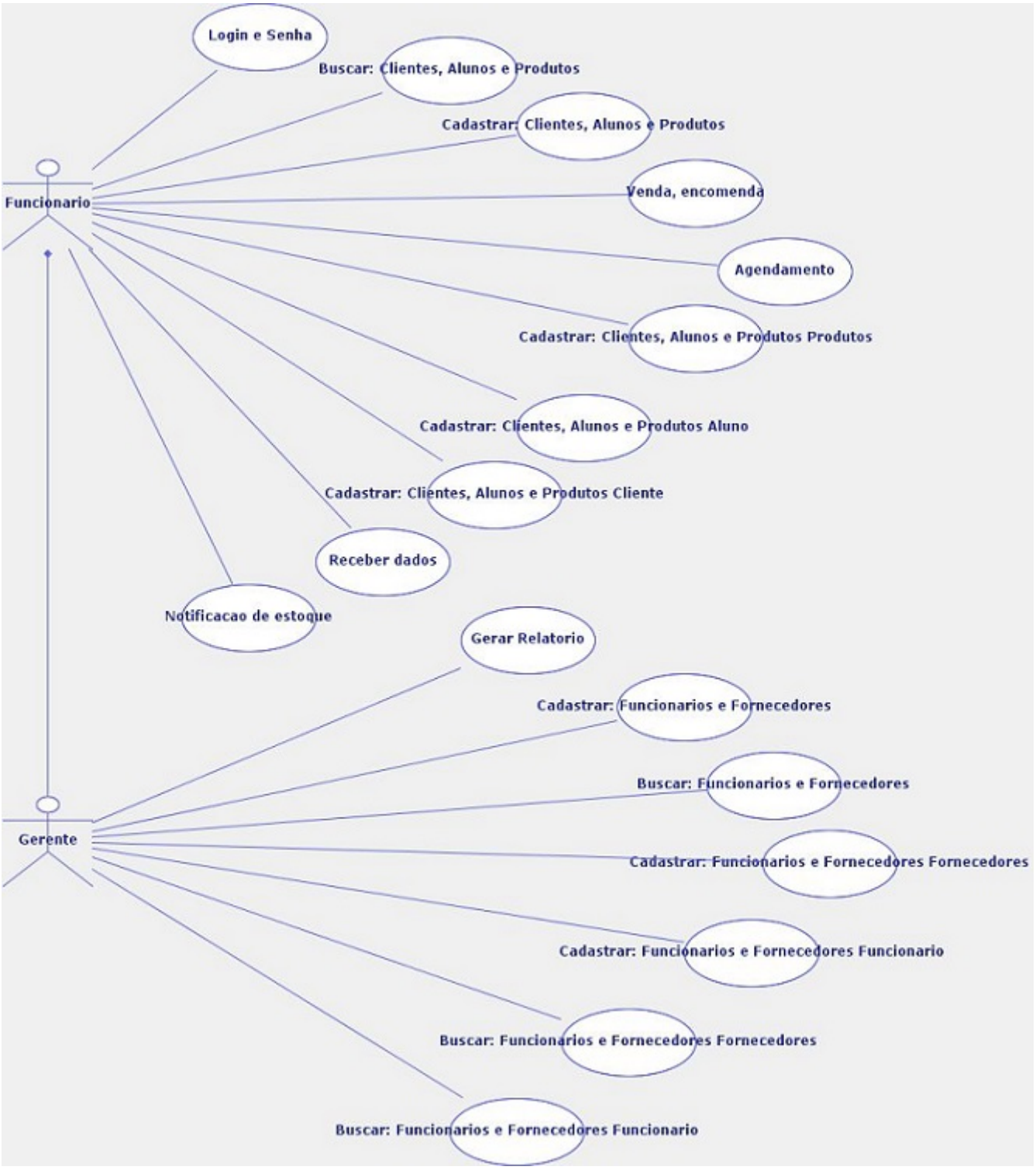


Figura 5.9: Diagrama de casos de uso gerado pelo Grupo 4

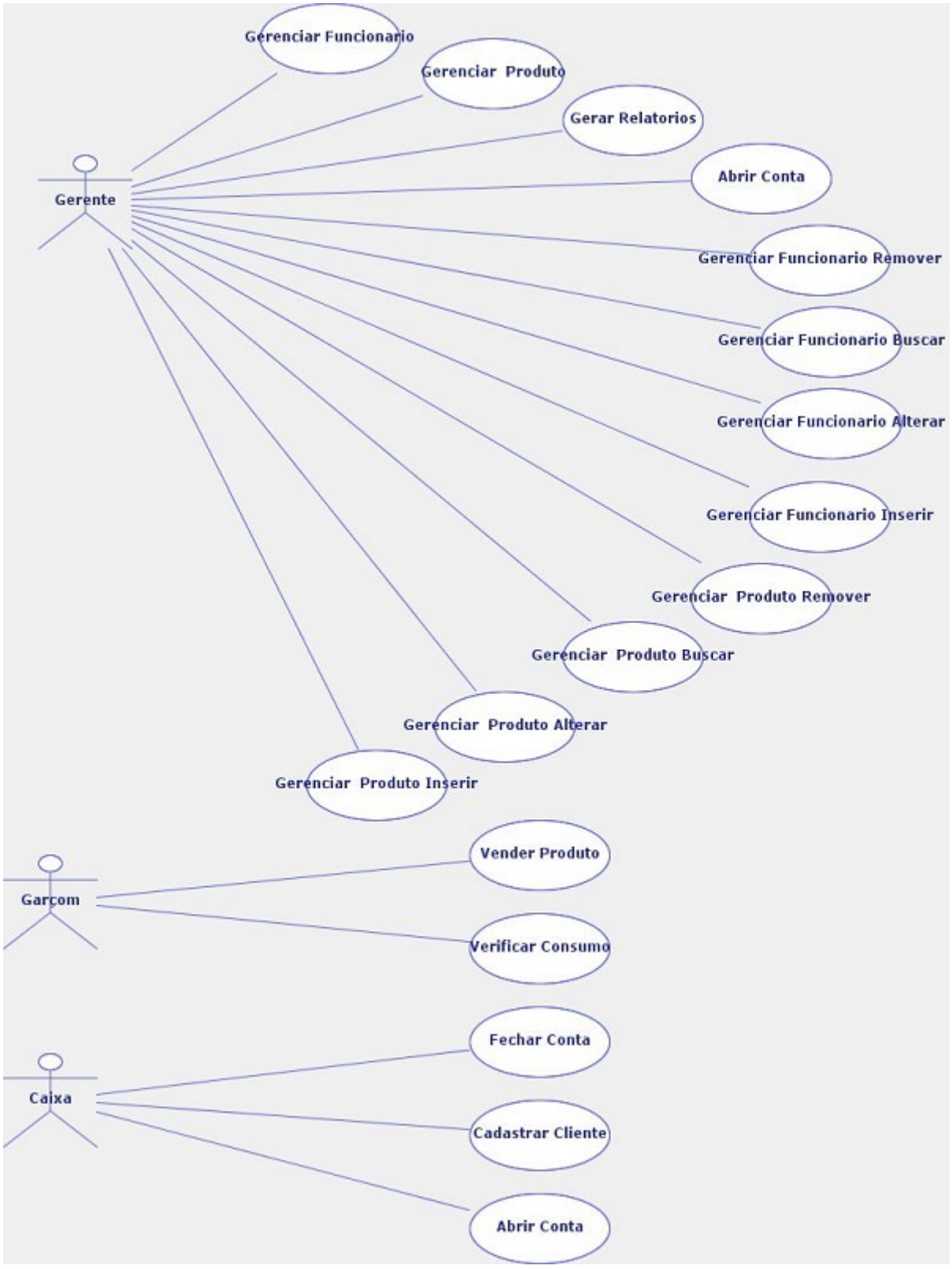


Figura 5.10: Diagrama de casos de uso gerado pelo Grupo 5

Capítulo 6

Considerações Finais

Neste trabalho de conclusão de curso foi desenvolvido o E4J Use Cases, uma ferramenta para manipulação de diagramas de casos de uso integrado à ferramenta JGOOSE. Apresenta-se a seguir os resultados desta pesquisa e as conclusões obtidas com base nesses resultados. Por fim, são propostos alguns trabalhos futuros relacionados ao E4J Use Cases.

6.1 Resultados

A ferramenta JGOOSE tornou-se uma aplicação auto-suficiente (*standalone application*) após a integração do E4J Use Cases, pois não é mais necessário utilizá-la a ferramenta StarUML (através da exportação de um arquivo .xmi pelo JGOOSE) para manipular o diagrama de casos de uso gerado pelo JGOOSE. Todo o processo de criação de diagrama de casos de uso e de manipulação do diagrama de casos de uso gerado pela rotina de mapeamento pode ser feito diretamente no ambiente de trabalho proposto pelo JGOOSE.

O quase-experimento realizado com a ferramenta E4J Use Cases (ver capítulo 5), mostrou que a ferramenta se encontra estável e contempla as funcionalidades necessárias para a criação e manipulação de diagrama de casos de uso.

6.2 Conclusões

Com os resultados obtidos, pode-se afirmar que um editor gráfico de diagramas de casos de uso integrado ao JGOOSE, como o E4J Use Cases, melhora o processo de desenvolvimento desses diagramas com a ferramenta. A criação desses modelos complementa as funcionalidades do JGOOSE, sendo que também é possível realizar a manipulação dos diagramas de casos de

uso gerados em conformidade com a rotina de mapeamento. Esse mapeamento é realizado internamente, sem a necessidade do uso de arquivos auxiliares para tal fim. Esse processo automático de geração do diagrama de casos de uso foi descrito como útil, vantajoso e ágil para a análise desse tipo de diagrama pelos grupos que realizaram o quase-experimento.

Outra vantagem do uso do ambiente de trabalho do JGOOSE, é que é possível executar mais uma instância da ferramenta ao mesmo tempo, além de ela ser compatível com os sistemas operacionais Linux e Microsoft Windows.

A ferramenta E4J Use Cases possui algumas limitações. Uma delas é que não existe uma função de sincronização entre o diagrama de casos de uso gerado pela rotina de mapeamento e os artefatos das ferramentas JGOOSE e E4J i*. Dessa maneira, caso o usuário modifique o diagrama de casos de uso gerado, essas modificações não se refletirão nos casos de uso do JGOOSE e/ou nos modelos i* do E4J Use Cases. Outra limitação inclui a inexistência do elemento que representa a fronteira do sistema computacional, o qual é utilizado para organizar os casos de uso.

6.3 Trabalhos Futuros

Realizada a análise e as conclusões dos resultados deste trabalho, foram identificados como trabalhos futuros:

- Elaborar o Manual do Usuário, contendo informações de cada elemento da interface gráfica que é possível de ser acessada pelo usuário;
- Elaborar uma melhor forma de pré-organizar o diagrama de casos de uso gerado pela rotina de mapeamento objetivando uma melhor visualização inicial do mesmo;
- Desenvolver uma função para realizar a sincronização entre o diagrama de casos de uso gerado pela rotina de mapeamento do E4J Use Cases e os artefatos das ferramentas E4J i* e JGOOSE;
- Incluir o elemento que representa a fronteira do sistema, o qual é um dos elementos que são utilizados em diagramas de casos de uso. Esse elemento define o escopo do sistema incluindo todos os casos de uso do mesmo e é representado como um retângulo que mede todos os casos de uso do sistema;

- Evoluir o quase-experimento para verificar a corretude dos elementos que não foram possíveis de serem verificados nao experimentação realizada.

Referências Bibliográficas

[ALDER 2002]ALDER, G. Design and implementation of the jgraph swing component. *Technical Report*, v. 1, n. 6, 2002.

[BARBOSA e SILVA 2010]BARBOSA, S. D. J.; SILVA, B. S. *Interação Humano-Computador*. Rio de Janeiro - RJ: Campus/Elsevier, 2010.

[BASTOS e CASTRO 2004]BASTOS, L. R.; CASTRO, J. F. Enhancing requirements to derive multi-agent architectures. *Proceedings of WER*, p. 127–139, 2004.

[BOOCH, RUMBAUGH e JACOBSON 2005]BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário*. 2. ed. [S.l.]: Editora Campus, 2005.

[BOOCH, RUMBAUGH e JACOBSON 2005]BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *Unified Modeling Language User Guide*. 2. ed. [S.l.]: Addison Wesley Professional, 2005.

[BREITMAN e LEITE 1998]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, USA, p. 214–221, 1998.

[BRESCIANI et al. 2004]BRESCIANI, P. et al. Tropos: An agent-oriented software development methodology. *Kluwer Academic Publishers*, v. 8, n. 3, p. 203–236, 2004.

[BRISCHKE 2005]BRISCHKE, M. Desenvolvimento de uma ferramenta para integrar modelagem organizacional e modelagem funcional na engenharia de requisitos. Cascavel - PR, Novembro 2005.

[BRISCHKE 2012]BRISCHKE, M. *Melhorando a Ferramenta JGOOSE*. Dissertação (Trabalho de conclusão de graduação) — Unioeste - Universidade Estadual do Oeste do Paraná, Cascavel - PR, Dezembro 2012.

[CASTRO, ALENCAR e CYSNEIROS 2000]CASTRO, J.; ALENCAR, F.; CYSNEIROS, G. Closing the gap between organizational requirements and object oriented modeling. *Journal of the Brazilian Computer Society, SciELO*, Brasil, v. 7, n. 1, p. 05–16, 2000.

[CASTRO et al. 2001]CASTRO, J. F. et al. Integrating organizational requirements and object oriented modeling. *RE '01 Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, Washington, DC, USA, p. 146–153, 2001.

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

[ECKSTEIN, LOY e WOOD 1998]ECKSTEIN, R.; LOY, M.; WOOD, D. Java swing. *O'Reilly & Associates*, 1998.

[ERIKSSON e PENKER 1998]ERIKSSON, H. E.; PENKER, M. Business modeling with uml: Business patterns at work. *John Wiley Sons*, New York, NY, USA, 1998.

[EVANS e KENT 1999]EVANS, A.; KENT, S. Core meta-modelling semantics of uml: the puml approach. *UML' 99 - The Unified Modeling Language, Springer*, p. 140–155, 1999.

[GROUP 2007]GROUP, O. M. *Unified Modeling Language: Superstructure*. [S.l.]: OMG, 2007.

[JACOBSON, BOOCH e RUMBAUGH 1999]JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Boston: Addison Wesley Longman Publishing Co., 1999.

[JECKLE 2004]JECKLE, M. . omg's xml metadata interchange format xmi. p. 25–42, 2004.

[JGRAPH 2013]JGRAPH. *JGraphX (JGraph 6) User Manual*. 2013. Consultado na Internet: http://jgraph.github.io/mxgraph/docs/manual_javavis.html, em 25 de Julho de 2014.

[KOLP, GIORGINI e MYLOPOULOS 2003]KOLP, M.; GIORGINI, P.; MYLOPOULOS, J. Organizational patterns for early requirements analysis. *Lecture Notes in Computer Science, Springer*, v. 2681, p. 617–632, 2003.

[KULAK e GUINEY 2000]KULAK, D.; GUINEY, E. Use cases: Requirements in context. *Addison Wesley*, 2000.

[LAMSWEERDE 2004]LAMSWEERDE, A. Goal-oriented requirements enginering: a round-trip from research to practice [enginering read engineering]. *IEEE. Requirements Engineering Conference*, p. 4–7, 2004.

[MAIDEN et al. 2004]MAIDEN, N. A. M. et al. Model-driven requirements engineering synchronising models in an air traffic management case study. *SPRINGER, Advanced Information Systems Engineering*, p. 368–383, 2004.

[MAO e YU 2005]MAO, X.; YU, E. Organizational and social concepts in agent oriented software engineering. *Agent-Oriented Software Engineering*, p. 1–15, 2005.

[MAVEN 2014]MAVEN, A. *Apache Maven Project*. 2014. Consultado na Internet: http://maven.apache.org/, em 25 de Julho de 2014.

[MERLIM 2013]MERLIM, L. *E4J: Editor i* para JGOOSE*. Dissertação (Trabalho de conclusão de graduação) — Unioeste - Universidade Estadual do Oeste do Paraná, Cascavel - PR, Junho 2013.

[PELISER, SANTANDER e MERLIM 2013]PELISER, D.; SANTANDER, V. F. A.; MERLIM, L. Jgoose: Melhorias realizadas. *V EPAC - Encontro Paranaense de Computação*, Cascavel - PR, Setembro 2013.

[POTTS 1999]POTTS, C. Scenic: A strategy for inquiry-driven requirements determination. *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering - RE'99*, Ireland, June 1999.

[PRESSMAN 2009]PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. [S.l.]: McGrawHill Science/Engineering/Math, 2009.

[PROJECT 2005]PROJECT, T. L. I. *BSD License Definition*. 2005. Consultado na Internet: http://www.linfo.org/bsdlicense.html, em 30 de Julho de 2014.

[RAO e GEORGEFF 1995]RAO, A. S.; GEORGEFF, M. P. Bdi agents: From theory to practice. *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*, SAN FRANCISCO, p. 312–319, 1995.

[SANTANDER 2002]SANTANDER, V. F. A. *Integrando Modelagem Organizacional com Modelagem Funcional*. Tese (Tese de Doutorado) — Universidade Federal do Rio Grande do Sul, Pernambuco: Centro de Informática, Universidade Federal de Pernambuco, Dezembro 2002.

[SANTANDER e CASTRO 2002]SANTANDER, V. F. A.; CASTRO, F. F. B. Deriving use cases from organizational modeling. *IEEE Joint International Requirements Engineering Conference - RE' 02*, Essen Germany, p. 32–39, 2002.

[SCHNEIDER e WINTERS 2001]SCHNEIDER, G.; WINTERS, J. P. *Applying Use Cases, A Practical Guide*. 2. ed. [S.l.]: Addison Wesley, 2001.

[Shull, Singer e Sjoberg 2008]SHULL, F.; SINGER, J.; SJOBERG, D. I. K. Guide to advanced empirical software engineering. *Springer*, 2008.

[SOLINGEN e BERGHOUT 1999]SOLINGEN, R.; BERGHOUT, E. The goal/question/metric method: a practical guide for quality improvement of software development. *UK: McGraw-Hill*, 1999.

[SUTCLIFFE e GREGORIADES 2002]SUTCLIFFE, A.; GREGORIADES, A. Validating functional requirements with scenarios. *IEEE Joint International Requirements Engineering Conference, RE'02*, University of Essen, Germany, p. 181–188, September 2002.

[TRAVASSOS, GUROV e AMARAL 2002]TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. *Introdução á Engenharia de Software Experimental*. Rio de Janeiro, 2002.

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

[WARMER e KLEPPE 2003]WARMER, J.; KLEPPE, A. *The object constraint language: getting your models ready for MDA*. [S.l.]: Addison Wesley Longman Publishing Co., 2003.

[WONG 2007]WONG, S. *StarUML Tutorial*. 2007. Consultado na Internet: http://cnx.org/content/m15092/1.1/, em 15 de Julho de 2014.

[YU 2011]YU, E. *i*: an agent and goal-oriented modeling framework*. 2011. Consultado na Internet: http://www.cs.toronto.edu/km/istar/, em Junho de 2014.

[YU et al. 2011]YU, E. et al. Social modeling for requirements engineering. *The MIT Press*, January 2011.

[YU e LIU 2001]YU, E.; LIU, L. Modelling trust for system design using the i* strategic actors framework. *Trust in Cyber-societies*, p. 175–194, 2001.

[YU e YU 2014]YU, E.; YU, Y. *Organization Modelling Environment*. 2014. Consultado na Internet: http://www.cs.toronto.edu/km/ome/, em 25 de Julho de 2014.

[YU, MYLOPOULOS e LESPERANCE 1996]YU, E. S.; MYLOPOULOS, J.; LESPERANCE, Y. Al models for business process reengineering. *IEEE expert*, v. 11, n. 4, p. 16–23, 1996.

[YU 1995]YU, E. S. K. *MODELLING STRATEGIC RELATIONSHIPS FOR PROCESS REENGINEERING*. Tese (Tese de Doutorado) — University of Toronto, 1995.

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

[ZELKOWITZ e WALLACE 1998]ZELKOWITZ, M.; WALLACE, D. Experimental models for validating technology. *IEEE Computer*, v. 31, n. 5, p. 23–31, 1998.