UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERRAMENTA PARA AUTOMATIZAR AS TRANSFORMAÇÕES BIDIRECIONAIS ENTRE I* E BPMN

Eduardo Bezerra de Melo

Trabalho de Graduação

Recife
MARÇO DE 2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Eduardo Bezerra de Melo

FERRAMENTA PARA AUTOMATIZAR AS TRANSFORMAÇÕES BIDIRECIONAIS ENTRE I* E BPMN

Trabalho apresentado ao Programa de GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO como requisito parcial para obtenção do grau de Bacharel em CIÊNCIA DA COMPUTAÇÃO.

Orientador(a): *Profª Drª Carla Taciana Lima Lourenço Silva Schuenemann*

Recife
MARÇO DE 2014

Agradecimentos

Agradeço

A Deus, pela oportunidade

À minha família, em especial aos meus pais, pelo apoio e confiança

À professora Carla Taciana, pelo tempo dedicado em me orientar nesta monografia

Aos colegas e amigos conquistados durante a graduação, pelo apoio e companheirismo.

Resumo

As fases de descoberta e especificação dos requisitos de um sistema de software são fundamentais para a satisfação das necessidades reais dos *stakelhoders*. Nesse contexto, a abordagem orientada a objetivos visa especificar os requisitos do sistema a partir dos objetivos que os *stakeholders* esperam atingir com sua utilização. Além da abordagem orientada a objetivos, outras abordagens de engenharia de requisitos têm sido pesquisadas por estudiosos da área, como a abordagem que se baseia na modelagem de processos de negócios. Os processos de negócios devem refletir os procedimentos que as entidades de uma organização devem executar para que os objetivos organizacionais sejam alcançados. Logo, existe uma forte relação entre os objetivos organizacionais e esses processos. O *framework* i* dá suporte à primeira abordagem, possibilitando a modelagem de objetivos centrada nos relacionamentos sociais entre os atores, nas dependências de recursos, objetivos ou tarefas existentes entre eles. A notação BPMN (*Business Process Management Notation)* se destaca pela capacidade de modelar a complexa semântica dos processos de negócios presentes em uma organização e de facilitar a compreensão desses processos pelos usuários e pelos participantes do negócio. O objetivo deste trabalho é a implementação de uma ferramenta capaz de transformar modelos i* em modelos BPMN, e vice-versa, a partir de um conjunto de heurísticas de mapeamento. As heurísticas de mapeamento e, consequentemente, as regras de transformação, devem garantir a consistência entre os modelos i* e BPMN. Dessa forma, os modelos BPMN gerados estarão alinhados com os objetivos estratégicos da organização.

Palavras-chave: Engenharia de Requisitos Orientada a Objetivos, Framework i*, BPMN, Processos de Negócio, Gestão de Processos de Negócio.

Sumário

Lista de Figuras

CAPÍTULO 1

Introdução

Este capítulo está dividido em quatro seções. A primeira seção apresenta o contexto em que está inserido este trabalho. Na segunda seção são expostas as motivações que suscitaram o estudo do tema e a implementação da ferramenta. Posteriormente, são descritos os objetivos de sua elaboração. Na última seção é apresentada a estrutura do documento.

1.1. Contexto

No processo de desenvolvimento de software, várias atividades são realizadas a fim de garantir que as funcionalidades ou serviços de um produto final estejam em conformidade com o que os usuários ou *stakeholders* especificaram. A Engenharia de Requisitos é o ramo da Engenharia de Software que abrange a fase inicial de desenvolvimento de um sistema de software. Constitui-se de um conjunto de atividades de elicitação, especificação e validação que resultam na produção de um documento de requisitos, que é uma descrição formal dos mesmos [1]. Tais requisitos devem estar alinhados com as necessidades reais dos *stakeholders* envolvidos na produção e utilização do software.

A Engenharia de Requisitos Orientada a Objetivos (*Goal Oriented Requirements Engineering* – GORE) é centrada nos objetivos que uma organização e no que seus atores pretendem alcançar com o desenvolvimento do software. Nesse contexto, ficam claros os objetivos que devem ser atingidos a partir do correto e completo funcionamento do sistema. Os requisitos devem ser modelados ou descritos em função desses objetivos.

O framework i* é uma abordagem GORE que permite a representação dos atores organizacionais, que dependem uns dos outros para alcançar metas, realizar tarefas ou fornecer recursos. É uma abordagem centrada no conceito do ator intencional, visto como um elemento organizacional possuidor de propriedades como objetivos, crenças, habilidades e compromissos [2]. Logo, nessa representação, são descritos os aspectos intencionais e motivacionais entre os atores de uma organização [3].

Por outro lado, a modelagem de processos de negócios permite que engenheiros de requisitos percebam o ambiente organizacional no aspecto procedural, isto é, como os atores ou entidades participantes do negócio executam suas tarefas para alcançar objetivos. Os processos podem ser observados a partir de diferentes perspectivas: funcional, comportamental, organizacional e informacional.

A notação gráfica BPMN (*Business Process Modeling Notation*) [10] é bastante utilizada na indústria para modelar processos de negócio. É um recurso que emprega vários elementos gráficos representativos que modelam conceitos como atividades, fluxos de sequência, fluxos de mensagem, eventos, etc.

Como os processos de negócio são elaborados para satisfazer os objetivos organizacionais, eles podem ser vistos como soluções para a satisfação de objetivos descritos em modelos de objetivos. Assim, modelos de objetivos podem ser transformados em modelos de processos de negócio que especificam como esses objetivos devem ser satisfeitos [4]. O uso conjunto da modelagem orientada a objetivos e da modelagem baseada em processos de negócio pode ser feita a partir do mapeamento entre esses modelos. O mapeamento garante que os processos de negócio estejam alinhados com os objetivos estratégicos da organização.

Na seção seguinte são expostas as motivações para a elaboração deste trabalho.

1.2. Motivações

Os processos de negócios devem refletir as atividades e as decisões tomadas por uma organização para satisfazer os seus objetivos estratégicos. A eficiência de uma organização está em função de uma gestão adequada desses processos, bem como na melhoria constante dos mesmos para o alcance eficiente de metas organizacionais.

Já existem métodos sistemáticos para a obtenção de modelos de processos de negócio a partir de modelos de objetivos, e vice-versa. Em particular, o trabalho de Alves [5] propõe heurísticas de mapeamento entre esses modelos, visando garantir a consistência entre eles e o alinhamento dos processos de negócio com os objetivos estratégicos da organização. As heurísticas definidas no trabalho de Alves [5] são uma evolução das heurísticas previamente definidas por Koliadis e outros [11] e constituem um método sistemático para a geração de um modelo BPMN a partir de um i*, diminuindo a interferência do analista no processo de

mapeamento e definição dos modelos. No entanto, o processo de mapeamento proposto por Alves [5] não é automatizado, o que prejudica a produtividade e a qualidade dos modelos gerados, visto que erros podem ser inseridos no processo manual.

Nesse contexto, este trabalho visa o desenvolvimento de uma ferramenta que implemente as regras de transformação propostas por Alves [5], o que possibilita a geração automática de um modelo a partir do outro.

1.3. Objetivos do Trabalho

O objetivo principal deste trabalho é o desenvolvimento de uma ferramenta capaz de transformar modelos descritos no framework i* em modelos descritos na notação BPMN e vice-versa. Como objetivos específicos, tem-se:

- Estudo das diretrizes de mapeamento entre i* e BPMN contidas no trabalho de Alves [5] e sua aplicação em um exemplo;
- Estudo da tecnologia *Epsilon* a ser utilizada para a construção da ferramenta;
- Criação de um editor gráfico para i*;
- Criação de um editor gráfico para BPMN;
- Implementação das regras de transformação i*-BPMN;
- Implementação das regras de transformação BPMN-i*;
- Ilustração de uso da ferramenta realizando a modelagem do primeiro exemplo.

A próxima seção apresenta a estrutura deste documento, apontando o conteúdo que será abordado ao longo deste trabalho.

1.4. Estrutura do Documento

Além desse capítulo introdutório, este trabalho possui mais quatro capítulos. O capítulo 2 apresenta a fundamentação teórica necessária para a compreensão dos capítulos subsequentes. Nele são apresentados os conceitos referentes à Engenharia de Requisitos, em especial a Engenharia de Requisitos Orientada a Objetivos, dando ênfase às características da modelagem de objetivos com o uso do framework i*. O capítulo 2 também apresenta uma

introdução à modelagem de processos de negócios e à notação gráfica BPMN. O trabalho de Alves [5], que propõe novas diretrizes para mapeamento entre modelos i* e modelos BPMN, também é apresentado, visto que é a base da ferramenta construída. Por fim, é apresentado um resumo acerca dos aspectos da tecnologia que foi utilizada para a construção da ferramenta.

No capítulo 3, são apresentadas as principais características da ferramenta e um exemplo que ilustra a sua execução, mostrando o passo-a-passo para a transformação do modelo i* no modelo BPMN correspondente e vice-versa, com base nas heurísticas definidas no trabalho de Alves [5].

No capítulo 4, é apresentado outro exemplo que ilustra a aplicação das regras de transformação e que visa identificar as limitações presentes nas heurísticas de mapeamento definidas por Alves [5].

No capítulo 5, serão apresentadas as conclusões deste trabalho, ressaltando as contribuições e limitações encontradas, bem como as perspectivas futuras. Por fim, o capítulo descreve algumas considerações finais advindas da elaboração desta monografia.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica necessária para a compreensão dos capítulos seguintes. Inicialmente, apresenta uma visão geral da engenharia de requisitos e a sua importância no processo de desenvolvimento de software. Apresenta também conceitos da engenharia de requisitos orientada a objetivos, com uma atenção especial ao framework i*. Posteriormente, são mostrados os conceitos da modelagem de processos de negócio com a notação gráfica BPMN. Logo após, apresenta o trabalho de Alves [5], fundamental para a elaboração desta monografia. Por fim, uma seção é dedicada a apresentar conceitos acerca da tecnologia utilizada para a construção da ferramenta.

2.1. Engenharia de Requisitos

A engenharia de requisitos abrange a fase inicial de desenvolvimento de um sistema de software. É nesta fase que o problema a ser solucionado com o desenvolvimento do software é entendido e onde são identificadas as necessidades reais dos clientes e/ou usuários finais. A engenharia de requisitos constrói uma ponte entre o projeto e a construção do software [6]. Segundo Brooks [7], em seu artigo:

> "A parte mais difícil da construção de um sistema é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil como estabelecer os requisitos técnicos detalhados, incluindo todas as interfaces para as pessoas, às máquinas, e outro sistema de software. Nenhuma outra parte do trabalho gera tanto prejuízo ao sistema resultante se feita de forma errada. Nenhuma outra parte é mais difícil de corrigir mais tarde."

Nesse artigo, Brooks [7] destaca a dificuldade na identificação dos requisitos de um sistema de software. Destaca ainda a importância da Engenharia de Requisitos no que concerne à garantia de um projeto de software completo e correto, enfatizando que um projeto realizado a partir de requisitos mal estabelecidos pode resultar em prejuízos para as partes envolvidas no processo de desenvolvimento do software.

2.1.1. Classificação dos requisitos

Requisitos são descrições das funções que um sistema deve realizar e das restrições associadas ao seu funcionamento. Tais descrições devem estar em conformidade com os objetivos reais dos usuários finais e dos *stakelhoders* envolvidos no processo de desenvolvimento do software. São classificados em dois níveis: requisitos de usuário e requisitos de sistema e em três tipos: requisitos funcionais, requisitos não funcionais e requisitos de domínio [8].

Requisitos de usuário são descrições, em linguagem natural e diagramas, dos serviços que o sistema deve oferecer para os usuários finais e as restrições nas quais o sistema deve operar. São destinados às pessoas envolvidas na aquisição e uso do sistema [8].

Requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições do sistema. É estruturado em um documento escrito como um contrato entre cliente e contratante. Nesse nível de requisitos, é especificado o que deverá ser implementado no sistema e não como o sistema será implementado [8].

Entre os tipos de requisitos, requisitos funcionais descrevem as funções ou serviços que o sistema de software deve prover, bem como seu comportamento em determinadas situações. Em alguns casos, também incluem descrições de funções ou serviços que o sistema não deve realizar. Dependem do tipo de software a ser desenvolvido, dos usuários que são esperados na utilização do software e da abordagem adotada pela organização na escrita dos mesmos. A especificação de requisitos funcionais deve ser ao mesmo tempo completa e consistente [8].

Requisitos não funcionais são restrições dos serviços ou funções do sistema, frequentemente aplicados ao sistema de software como um todo. Eles englobam, por exemplo, restrições de tempo, restrições no processo de desenvolvimento do software e restrições impostas por padrões da organização [8].

Requisitos de domínio estão relacionados com o domínio da aplicação do sistema. O principal problema dos requisitos de domínio é a dificuldade de entendimento do domínio pelos engenheiros de software.

A seguir, será apresentado o processo da engenharia de requisitos, cuja execução é fundamental para a produção de um documento de requisitos completo e consistente.

2.1.2. Processo da Engenharia de Requisitos

O processo de engenharia de requisitos envolve a elicitação, análise, especificação e validação dos requisitos de um sistema. Na fase de identificação de requisitos, são identificadas as partes interessadas no sistema (*stakeholders*), a captura das necessidades e objetivos dos usuários com o desenvolvimento do mesmo e o entendimento do problema a ser solucionado com a utilização do software. Segundo Sommerville [8], a divisão do processo de engenharia de requisitos é realizada em quatro atividades de alto nível de abstração:

- Estudo de viabilidade: é um breve estudo que analisa as possibilidades do sistema, suas limitações, contribuições para os objetivos da organização e a sua viabilidade no que se refere às tecnologias utilizadas para a construção do mesmo. Um documento de viabilidade é gerado como artefato na conclusão desta atividade;
- Elicitação e análise de requisitos: é nesta atividade que engenheiros de software trabalham com clientes e usuários finais a fim de descobrir detalhes do domínio de aplicação do sistema, quais serviços devem ser providos pelo software, as restrições de desempenho do sistema, restrições de hardware, etc.;
- Validação de requisitos: é a atividade de checagem da conformidade dos requisitos com o que o cliente realmente deseja. É uma etapa importante, pois problemas descobertos na fase de desenvolvimento podem ocasionar altos custos de retrabalho;
- Gerenciamento de requisitos: é nesta atividade que os requisitos são atualizados, refletindo as mudanças na visão do problema a ser solucionado.

Na seção a seguir, a Engenharia de Requisitos Orientada a Objetivos com o framework i* é apresentada.

2.2. Engenharia de Requisitos Orientada a Objetivos

2.2.1. Framework i*

O i* foi desenvolvido visando a modelagem do ambiente organizacional e seus sistemas de informação. Nesse framework, atores dependem uns dos outros para que objetivos sejam alcançados, tarefas sejam realizadas e recursos sejam fornecidos [9].

O framework i* é uma abordagem da engenharia de requisitos orientada a objetivos que é centrada no conceito do ator intencional e estratégico. O ator é intencional quando possui motivações, intenções e razões para executar ações. É estratégico quando não é meramente focado em alcançar seus objetivos imediatamente, mas é preocupado com as implicações em longo prazo de seus relacionamentos com outros atores. Um processo pode então ser modelado como uma configuração de relacionamentos entre atores intencionais [10].

A abordagem de modelagem com i* é uma tentativa de levar a concepção social para o processo de engenharia de sistemas, introduzindo conceitos sociais nas atividades diárias de analistas de sistemas e projetistas.

Os principais elementos da modelagem com i* foram definidos por Yu [10] e podem ser classificados em 3 categorias: atores, elementos intencionais e dependências. Um ator é uma entidade ativa que realiza ações para atingir objetivos. Elementos intencionais são elementos necessários para que uma tarefa executada por um ator seja realizada com sucesso. Na categoria dos elementos intencionais, estão os objetivos, *softgoals*, tarefas e recursos. Dependências descrevem relacionamentos intencionais entre atores em que um ator depende do outro para alcançar algo. A seguir, são apresentados os elementos do framework i*:

- Ator (*Actor*): é uma entidade ativa que executa ações para alcançar objetivos. É representado por um círculo;
- Objetivo (*Goal*): representa algum estado ou condição que os *stakeholders* desejam alcançar. É representado por uma forma oval;
- *Softgoal*: representa alguma condição ou estado do mundo que os *stakeholders* desejam experimentar. São os requisitos não funcionais do sistema. É representado por uma forma de nuvem;
- Tarefa (*Task*): forma específica de se fazer algo. São representadas por hexágonos;

- Recurso (*Resource*): representa uma entidade física ou não física, que possui como requisito não funcional a disponibilidade. Um retângulo é usado para representar um recurso;
- Dependência (*Dependency*): relacionamento entre dois atores, onde um precisa do outro para conseguir alguma coisa. O ator que necessita do outro é chamado *depender* e o outro *dependee.* O elemento do qual o *depender* necessita é chamado *dependum*.

A seguir, serão apresentados os dois modelos do framework i*: o modelo de dependência estratégica ou SD (do inglês, *Strategic Dependency Model*) e o modelo de razão estratégica ou SR (do inglês, *Strategic Rationale Model*). O primeiro procura mostrar os relacionamentos intencionais entre os atores, enquanto o segundo tenta mostrar as razões que estão por trás dos elementos do processo [10].

O Modelo de Dependência Estratégica

O modelo de dependência estratégica ou SD descreve processos em termos de relacionamentos intencionais entre atores [10]. Um ator é visto como uma entidade ativa que realiza ações para alcançar objetivos. Quando um ator (*depender*) depende de outro ator (*dependee*) para conseguir algum *dependum*, ele está apto a alcançar objetivos que não alcançaria sem a dependência, ou pelo menos não alcançaria tão facilmente [10].

No modelo SD, os objetivos internos dos atores não são explicitamente modelados [10]. Dessa forma, o modelo SD provê uma visão clara e sucinta dos relacionamentos e dependências existentes entre os atores de uma organização, mas não se preocupa no detalhamento das ações realizadas por esses atores para o alcance dos seus objetivos internos.

As dependências no modelo SD podem ser de quatro tipos, definidos em função do tipo de *dependum* associado à dependência:

- Dependência de tarefa (*Task Dependency*): nesse tipo de dependência, o *dependum* é uma tarefa (*task*). Dessa forma, o ator *depender* precisa do ator *dependee* para que uma tarefa (*dependum*) seja executada. Caso o *dependee* não consiga realizar a tarefa, o *depender* pode ser impactado no curso de suas atividades;
- Dependência de objetivo (*Goal Dependency*): nessa dependência, um ator necessita do outro para alcançar algum estado ou condição do mundo, ou seja, um *dependum* do tipo objetivo;

- Dependência de recurso (*Resource Dependency*): o ator *depender* necessita que o ator *dependee* disponibilize algum recurso (*resource*). O ator *depender* se torna vulnerável caso o recurso torne-se indisponível. A continuidade das atividades do *depender* é diretamente impactada pela indisponibilidade do recurso;
- Dependência de softgoal (*Softgoal Dependency*): ator *depender* depende do ator *dependee* para realizar alguma tarefa que alcance algum *softgoal*. Nesse caso, o elemento *dependum* é do tipo *softgoal*. Caso o *dependee* falhe em satisfazer essa dependência, o ator *depender* se tornará vulnerável na execução das suas atividades no processo.

A Figura 1 mostra um exemplo de modelo de dependência estratégica do domínio da assistência médica. Nele são mostrados os relacionamentos entre pacientes (*Patient*), médicos (*Physician*), laboratórios (*Lab*), companhias de seguro (*Insurance Company)* e gerentes (*Claim Manager*). Pacientes dependem de médicos para obterem tratamentos, enquanto médicos dependem de pacientes para que eles tomem a medicação e dos laboratórios para realizar exames. Médicos dependem dos gerentes para receberem o pagamento referente aos tratamentos e laboratórios dependem dos gerentes para o pagamento de uma taxa (*Lab Fee*). O paciente depende da companhia de seguro para receber cobertura caso esteja doente. A companhia de seguro, por sua vez, depende o paciente para receber um pagamento premium (*Premium Payment*). Gerentes dependem da companhia de seguros para obter informação dos pacientes. E a companhia de seguros depende do gerente para processar solicitações.

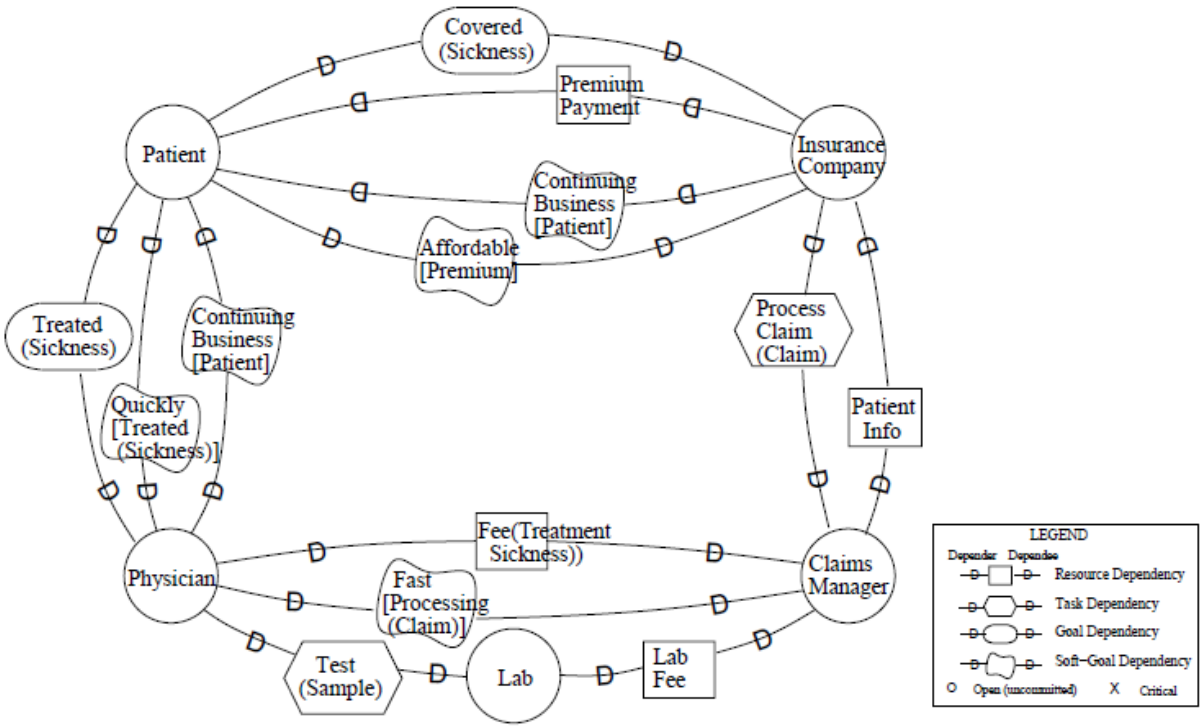


Figura 1 Modelo de dependência estratégica - domínio de assistência médica [10]

O Modelo de Razão Estratégica

Enquanto o modelo de dependência estratégica (SD) modela apenas os relacionamentos externos entre os atores, o modelo de razão estratégica (SR) permite o refinamento dos elementos internos dos mesmos, apresentando o raciocínio estratégico adotado por eles no processo do qual participam [10].

No modelo SR, os elementos intencionais aparecem no modelo não apenas nas dependências externas, mas também como elementos internos dos atores. Esses elementos podem estar ligados por meio de relações de meio-fim (*means-end links*), decomposições de tarefas (*task-decompositions*) e ligações de contribuição (*contribution links*).

Uma ligação de meio-fim (*means-end*) indica um relacionamento entre um "fim" - que pode ser um objetivo a ser alcançado, um recurso a ser fornecido, uma tarefa a ser executada ou um *softgoal* a ser satisfeito – e um meio para que esse fim seja alcançado. Um meio é expresso como uma tarefa, que pode ser decomposta em outras tarefas a partir de ligações de decomposição de tarefas, ou um *softgoal* (caso o fim também seja um *softgoal*). Logo, existem quatro tipos de ligações meio-fim: de tarefa para objetivo (*GTLink*), de tarefa para recurso (*RTLink*), de tarefa para *softgoal* (*STLink*) e de *softgoal* para *softgoal* (*SSLink*).

Uma ligação de decomposição de tarefa liga uma tarefa pai (tarefa a ser decomposta) a uma tarefa filho que faz parte da decomposição. Existem quatro tipos de ligações de decomposição de tarefa - sub-objetivo (*subgoal*), sub-tarefa (*subtask*), para um recurso (*resourceFor*) e para um *softgoal* (*softgoalFor*) – que correspondem aos quatro tipos de elementos intencionais (Figura 2).

Uma ligação de contribuição mede o grau de satisfação de um *softgoal*. Assim como nas ligações de dependência, existem vários tipos de ligações de contribuição, cada qual representando um grau diferente de satisfação. São eles:

- Make: contribuição positiva, capaz de satisfazer um *softgoal* por completo;
- Some +: contribuição positiva, de grau de satisfação desconhecido;
- Help: contribuição positiva, porém sozinha não consegue satisfazer um *softgoal* por completo.;
- Unknow: contribuição de influência não conhecida;
- Break: contribuição negativa que pode comprometer um *softgoal*;
- Some-: contribuição negativa, de grau de satisfação desconhecido;
- Hurt: contribuição negativa, mas não é capaz de comprometer um *softgoal*;
- Or: *softgoal* só é satisfeito se ao menos um elemento de origem for satisfeito;
- And: *softgoal* só é satisfeito se todos os elementos de origem forem satisfeitos.

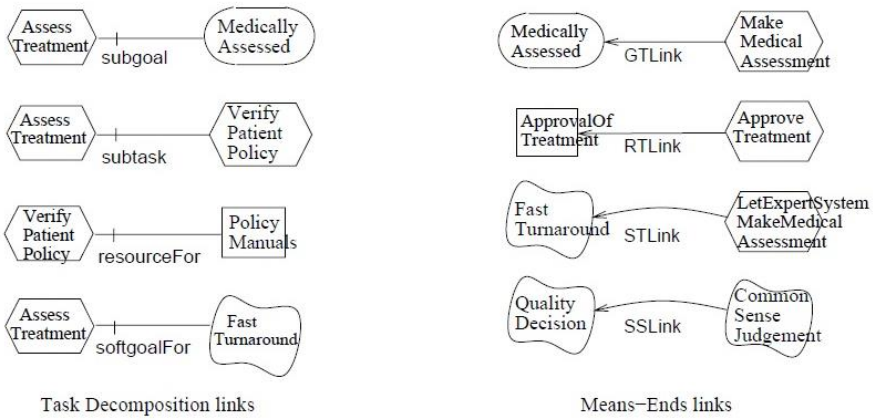


Figura 2 Tipos de ligações de decomposição de tarefa e ligações meio-fim [10]

O modelo SR é então uma extensão do modelo SD, uma vez que possui os mesmos elementos do segundo, com a peculiaridade de representar as razões estratégicas associadas a um ator e os relacionamentos entre essas razões e as dependências externas com outros atores.

A Figura 3 mostra um modelo SR que modela as razões estratégicas do ator *Claims Manager*, presente no modelo SD da figura anterior. O médico (*Physician*) depende do *Claims Manager* para obter a aprovação do tratamento (*ApprovalOfTreatment*). O gerente (*Claims Manager*), por sua vez, aprova o tratamento executando a tarefa *Approve Treatment*, elemento interno do ator que, portanto, expressa parte do raciocínio estratégico do mesmo. Para que o tratamento seja aprovado, é preciso que o tratamento seja avaliado (*TreatmentBeAssessed*) e o documento de aprovação seja assinado (*SignApprovalDocument*). O tratamento pode ser avaliado pelo próprio gerente (*AssessTreatment*) ou por outro funcionário (*LetClaimsClerkAssessTreatment*). Caso o gerente avalie o tratamento, ele deve avaliar a política da companhia de seguros do paciente (*VerifyPatientPolicy*) e avaliar o plano de tratamento (*MedicallyAssessed*). A última avaliação pode ser realizada por um *MedicalAssessor* ou pelo próprio gerente, a partir de uma revisão dos históricos médicos de outros pacientes, presentes em um repositório (*ClaimsCasesRepository*).

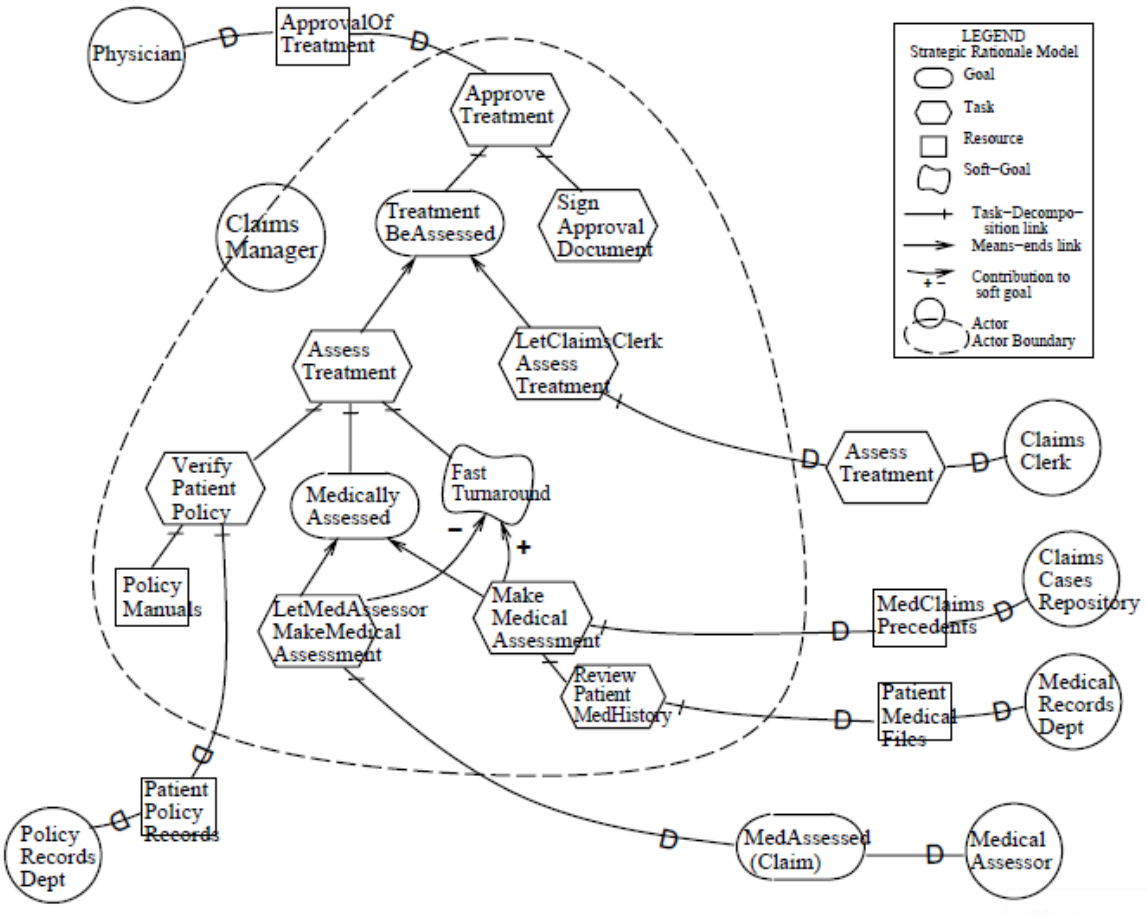


Figura 3 Modelo de Razão Estratégica – domínio de assistência médica [10]

A análise social do i* complementa a visão do fluxo de atividades, concentrando-se nos objetivos que uma organização pretende alcançar [10]. Assim, uma abordagem orientada a objetivos, como o framework i*, poderia complementar a modelagem de processos de negócio, com o uso de abordagens orientadas a fluxograma, como o BPMN e o diagrama de atividades do UML [10]

Na próxima seção, serão apresentados os conceitos relativos à modelagem de processos de negócio, com ênfase na modelagem de processos com a utilização da notação BPMN para a especificação de requisitos de software.

2.3. Modelagem de Processos de Negócio

2.3.1. A notação BPMN

A notação BPMN é composta por um conjunto de elementos gráficos que facilitam a modelagem de processos pelos analistas de negócio. O BPMN permite que analistas de negócio entendam e definam seus procedimentos internos e externos, o que permite que as organizações compartilhem e comuniquem esses processos de forma padronizada [11]. Uma das principais motivações do BPMN é a criação de um mecanismo simplificado para criar modelos de processos de negócio e ao mesmo tempo lidar com a complexidade inerente a esses processos [11].

O diagrama resultante da modelagem com BPMN é chamado de BPD (*Business Process Diagram*). O BPD é composto de um conjunto de elementos gráficos interconectados. As quatro categorias de elementos gráficos do BPMN são:

- Objetos de fluxo: constituem os principais elementos da modelagem gráfica em BPMN. Podem ser atividades (tarefa ou subprocesso) (Figura 4), que expressam tarefas realizadas pela organização e são representadas por um retângulo de bordas arredondadas; eventos (inicial, intermediário ou final), que representam algo que acontece durante o curso do processo de negócio e são representados por um círculo (Figura 5); e *gateways*, representados por um losango (Figura 6) e utilizados para controlar a convergência ou divergência do fluxo de sequência de um processo.

Figura 4 Notação em BPMN para atividades [11].

Figura 5 Notação em BPMN para evento inicial, intermediário e final, respectivamente [11].

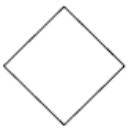


Figura 6 Notação em BPMN para *gateways* [11]

- Objetos de conexão: são responsáveis por criar a estrutura básica do processo de negócio. Podem ser fluxos de sequência, fluxos de mensagens ou associações. Fluxos de sequência são representados por uma seta de traço e ponta sólidos e são usados para expressar a ordem de execução das atividades do processo (Figura 7). Um fluxo de mensagem é representado por uma linha tracejada como uma seta de ponta não preenchida e é utilizado para expressar o sentido das mensagens trocadas entre participantes do processo (Figura 8). Uma associação é representada por uma linha pontilhada com uma ponta de seta e é usada para associar objetos de dados a objetos de fluxo (Figura 9).

Figura 7 Notação em BPMN para fluxo de sequência [11]

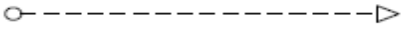


Figura 8 Notação em BPMN para fluxo de mensagem [11]

Figura 9 Notação em BPMN para associação [11]

- *Swimlanes*: são úteis para organizar atividades. Os dois tipos de *swimlanes* num BPD são *pools* (Figura 10) e *lanes* (Figura 11).

Figura 10 Notação gráfica em BPMN para *pools* [11]

Figura 11 Notação gráfica em BPMN para *lanes*.[11]

- Artefatos: fornecem informações adicionais sobre o processo. Podem ser objetos de dados, que representam os dados produzidos ou consumidos pelas atividades; grupos, que agrupam atividades e são utilizados para fins de documentação e análise (não afetam o fluxo de sequência); ou anotações, informação textual útil para quem estiver visualizando o diagrama do processo. A Figura 12 mostra a representação gráfica desses elementos.

Figura 12 Notação gráfica em BPMN para objetos de dados, grupos e anotações [11].

Na próxima seção, será apresentado o trabalho de Alves [5], que motivou o desenvolvimento deste trabalho.

2.4. Integração bidirecional entre os modelos i* e BPMN

O método de transformação entre modelos orientados a objetivos e modelos de processos de negócio proposto por Alves [5] é uma melhoria do método de transformação proposto por Koliadis e outros [12], que apoia a evolução conjunta desses modelos, visando garantir a consistência e conformidade dos processos de negócio com os objetivos organizacionais.

O método de Alves [5] é composto por um conjunto de heurísticas de mapeamento, em que cada heurística transforma elementos do modelo de origem em elementos do modelo destino. Foram definidas heurísticas de transformação tanto de i* para BPMN quanto de BPMN para i*. Três das dez heurísticas de mapeamento de i* para BPMN, definidas no trabalho de Alves [5], foram baseadas no método proposto por Koliadis e outros [12] . As outras heurísticas de mapeamento de i* para BPMN foram adicionadas para que o processo de transformação de um modelo no outro pudesse ser mais sistemático [5]. A seguir é apresentado o conjunto de heurísticas de mapeamento definido por Alves [5] em seu trabalho.

Heurísticas de mapeamento do modelo i* para o modelo BPMN

Como explicado anteriormente, o modelo i* modela processos como um conjunto de atores e seus relacionamentos intencionais. O modelo de razão estratégica (SR) é uma extensão do modelo de dependência estratégica (SD), pois além de permitir a modelagem das dependências externas entre os participantes do processo, modela também as razões estratégicas desses atores, configuradas como relacionamentos de elementos internos aos mesmos, tais como objetivos, tarefas, ligações meio-fim e ligações de decomposição de tarefas.

As heurísticas de mapeamento propostas por Alves [5] e Koliadis e outros [12] baseiam-se nos conceitos de rotina e escopo de um processo. Segundo Yu [10], uma rotina no modelo i* é um subdiagrama do modelo SR que representa um determinado curso entre as alternativas. O escopo é constituído pelo conjunto de subtarefas da rotina, pelas dependências ligadas a essas subtarefas e pelos atores que estão conectados a essas dependências. As regras de transformação estão transcritas a seguir:

I. Primeiramente identificamos a rotina e o escopo do processo. O conceito de rotina já foi explicado anteriormente. Porém, o escopo do processo deve ser

obtido, incluindo as sub-tarefas da rotina no primeiro nível da decomposição, as dependências ligadas a essas sub-tarefas, independente se o proprietário da rotina é o ator *depender* ou *dependee* e os atores que participam dessas dependências, como também as tarefas conectadas a elas. Cada rotina identificada no processo criará um modelo de BPMN diferente.

a. Se na decomposição da rotina existir um sub-objetivo ou sub-recurso que é alcançado por uma tarefa, essa tarefa deve está dentro do escopo e sua transformação no BPMN será de uma atividade atômica. Se a tarefa for decomposta, no BPMN, ela será um sub-processo.

II. Cada ator presente no escopo será transformado em um participante no modelo BPMN.

a. Atores que não pertencem à mesma organização ficarão em *pools* diferentes.
b. Atores que pertencem à mesma organização ficarão no mesmo *pool*, mas em *lanes* diferentes.

III. As tarefas internas dos atores presentes no escopo são incluídas como atividades atômicas nas *lanes*/*pools* dos participantes correspondentes no modelo BPMN.

IV. Se a tarefa dentro do escopo é decomposta, essas sub-tarefas devem ser analisadas:

a. Se as sub-tarefas devem ser realizadas em paralelo, elas tornam-se atividades paralelas dentro da *lane*/*pool* do participante correspondente.
b. Se as sub-tarefas devem ser realizadas em sequência, elas tornam-se atividades conectadas através do fluxo de sequência.
c. Se a tarefa em questão não for a rotina escolhida do processo e sua decomposição possui mais de um nível de decomposição, no modelo BPMN esta tarefa se tornará um sub-processo e suas sub-tarefas se tornarão o detalhamento deste sub-processo. O detalhamento do sub-processo deve ser feito de acordo com a regra (4.a) e (4.b).

V. Se existir algum recurso interno dos atores dentro do escopo, no BPMN esse recurso será um artefato gerado pela atividade atômica ou sub-processo correspondente a tarefa que alcança esse recurso.

VI. Uma dependência de tarefa é incluída como uma atividade na *lane* correspondente do ator *dependee* e a mensagem de fluxo ou a ligação de controle de fluxo é direcionada a atividade correspondente do ator *depender*.

VII. Uma dependência de recurso é transformada em um artefato produzido pela atividade presente no participante que representa o ator *dependee*. Duas mensagens de fluxo ou controle de sequência de fluxo são adicionadas entre as atividades presentes nos participantes mapeados. O ator *depender* requisita o artefato e o ator *dependee* fabrica e envia o artefato para o ator *depender*. Essas duas ligações são conectadas em sentido

opostos e foram adicionadas ao BPMN, porque, quando um ator necessita de um recurso, ele requisita a outro ator e esse ator, realiza a atividade que fabrica e recurso e responde a solicitação, enviando o artefato.

VIII. Um objetivo torna-se um evento final porque é o estado que os atores participantes do processo querem alcançar.

a. Se o objetivo é uma dependência, o evento final é incluído na *lane*/*pool* do ator *depender* correspondente.
b. Se o objetivo é um elemento interno de um ator, o evento final é incluído na *lane*/*pool* do ator correspondente.

IX. A tarefa raiz relacionada com a rotina escolhida torna-se o evento inicial que desencadeia o processo.

X. Qualquer tarefa que é decomposta em mais de um nível de decomposição, será um sub-processo no modelo BPMN.

A autora também propôs algumas regras de consistência entre os modelos, apresentadas a seguir:

1. Todo ator é necessariamente um participante no modelo BPMN.

2. Todas as tarefas internas dos atores, são atividades internas na *lane/pool* do participante correspondente.

3. Toda dependência deve ter uma ligação de mensagem ou controle de fluxo.
    a. Se for uma dependência de recurso, deve ter duas ligações entre as atividades e um artefato gerado.
4. Todo objetivo é um evento final não vazio.

5. O evento inicial do processo será a tarefa que realiza o objetivo geral.

Heurísticas de mapeamento do modelo BPMN para o modelo i*

Além de definir as heurísticas de mapeamento de i* para BPMN, a autora propôs heurísticas de mapeamento de BPMN para i*, assim como as regras de consistência associadas a esse tipo de transformação. As regras de transformação e de consistência estão transcritas a seguir:

(i) Cada participante que corresponde a *lane* ou *pool* no modelo BPMN é um ator no modelo i*.

(ii) Cada atividade atômica dentro da *lane* ou *pool* deve ser uma tarefa interna do ator.

(iii) Ligações de fluxo de mensagem ou ligações de controle de fluxo entre *pools*/*lanes* se tornarão dependências entre os atores.

a. Se a ligação entre as atividades gerará um artefato, essa dependência no modelo i* será uma dependência de recurso. O ator *dependee*, dessa dependência, é o ator que produz o recurso.

(iv) Um evento final não vazio pode tornar-se, dependendo do julgamento feito pelo analista, um objetivo interno relacionado com a rotina que está sendo modelada ou pode ser transformado em uma dependência de objetivo.

a. No primeiro caso, o objetivo é interno ao ator que possui o evento final.

b. No último caso, o ator *depender* da dependência de objetivo no modelo i* é o ator que possui o evento final. Como o evento final é do ator, isso é um objetivo que ele deseja alcançar, mas ele depende de outro ator (*dependee*) para atingir o seu objetivo.

(v) A sequência de atividades no modelo BPMN deve ser analisada, e dependendo do julgamento feito pelo analista, pode se tornar sub-tarefas de alguma decomposição de tarefa ou uma tarefa sem um nó pai e filhos.

(vi) Os sub-processos são transformados em decomposição de tarefas. E suas tarefas são transferidas para o modelo i* de acordo com as regras acima, dependendo da interpretação do analista.

(vii) O evento inicial, que desencadeia o processo, será transformado na tarefa que alcança o objetivo geral do processo.

(viii) Como os *softogoals* não são modelados no BPMN, eles podem ser inferidos pela pesquisa de atributos de qualidade associados às atividades desenvolvidas pelos participantes.

As regras de consistência de elementos do modelo BPMN para o i* são:

1. Todo participante é necessariamente um ator no modelo i*.

2. Todas as atividades internas dos participantes são tarefas internas dos atores correspondentes.

3. Toda ligação de mensagem ou controle de fluxo deve ter uma dependência correspondente no modelo i*.

a. Se forem duas ligações no sentido oposto entre atividade e uma atividade gera um artefato, a dependência será dependência de recurso.

4. Todo evento final não vazio é um objetivo.

2.5. Exemplo de aplicação: o processo *Managed Indemnity Insurance*

Esta seção apresenta a aplicação da heurísticas de mapeamento ao processo *Managed Indemnity Insurance*, utilizado por Alves [5] em seu trabalho a fim de ilustrar o uso de suas heurísticas. O processo é modelado com um modelo de razão estratégica (SR), em que os compartimentos dos atores são estendidos para que sejam mostrados os seus elementos internos, como objetivos, tarefas, decomposição de tarefas e ligações de contribuição. A Figura 13 apresenta o modelo SR desse processo.

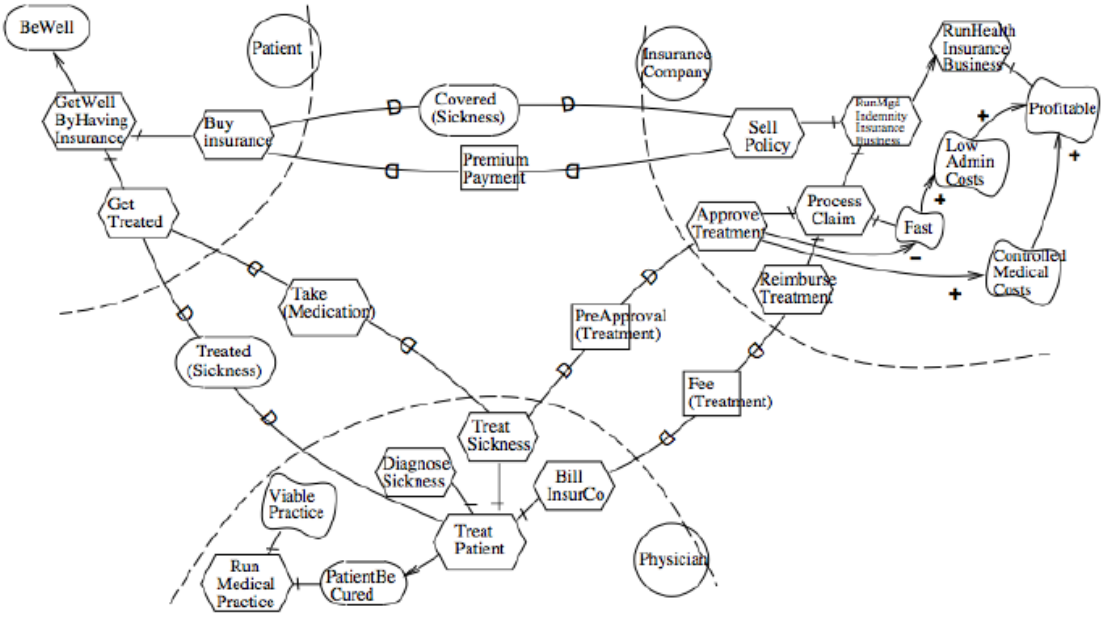


Figura 13 Modelo SR do processo de gerenciamento de seguros de saúde

O processo *Managed Indemnity Insurance* é uma modelagem para gerenciamento de seguros de saúde. Nele, pacientes (*Patient*), médicos (*Physician*) e companhias de seguros (*Insurance Company*) dependem uns dos outros para realizar tarefas e alcançar objetivos.

O objetivo principal do paciente é o de estar bem (*BeWell*). Para alcançar esse objetivo, o mesmo precisa comprar um seguro de saúde (*Buy Insurance*) e receber o tratamento (*Get Treated*). O paciente ainda depende da companhia de seguros para receber cobertura em caso de doença (*Covered*) e do médico para ser tratado (*Treated*).

O médico, por sua vez, tem como objetivo geral a cura do paciente (*PatientBeCured*). Para isso ele deve tratar o paciente (*Treat Pacient*), realizando o diagnóstico da doença (*Diagnose Sickness*), tratando a doença (*Treat Sickness*) e realizando o pagamento pela companhia de seguros. Para que a doença seja tratada, o médico precisa que o paciente tome a

medicação prescrita. Além disso, o médico tem o *softgoal ViablePractice*, que representa a prática viável da profissão.

Por outro lado, a companhia de seguros deve aplicar a sua política de vendas de seguros (*Sell Policy*) e processar solicitações (*Process Claims*). A companhia depende ainda dos pacientes para receber um pagamento *premium* na venda das apólices de seguro (*Premium Payment*). Para que as solicitações sejam processadas, é necessário que o pagamento seja pré-aprovado (*Approve Treatment*) e que o tratamento seja reembolsado (*Reimburse Treatment*). A companhia tem como *softgoal* a rentabilidade (*Profitable*). A tarefa de pré-aprovação do tratamento contribui negativamente para a agilidade (*Fast*) da companhia no processamento de solicitações. A agilidade da companhia contribui positivamente para a diminuição de custos da mesma, o que torna o negócio rentável (*Profitable*). Por outro lado, a pré-aprovação do tratamento contribui positivamente para o controle dos custos médicos (*Controlled Medical Costs*), que também contribui positivamente para a rentabilidade.

O modelo BPMN gerado pela aplicação manual das heurísticas de mapeamento de i* para BPMN ao exemplo do processo *Managed Indemnity Insurance* é apresentado na Figura 14.

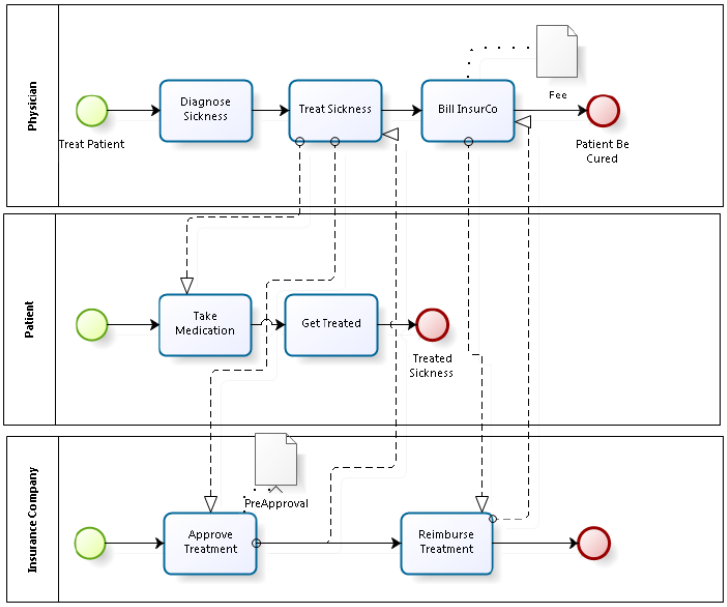


Figura 14 Modelo BPMN gerado do modelo i* da Figura 13

Com a aplicação manual do método inverso para obter um modelo i* a partir de um modelo BPMN, obtemos o modelo da Figura 15. O modelo utilizado para a aplicação das heurísticas foi o da Figura 14.

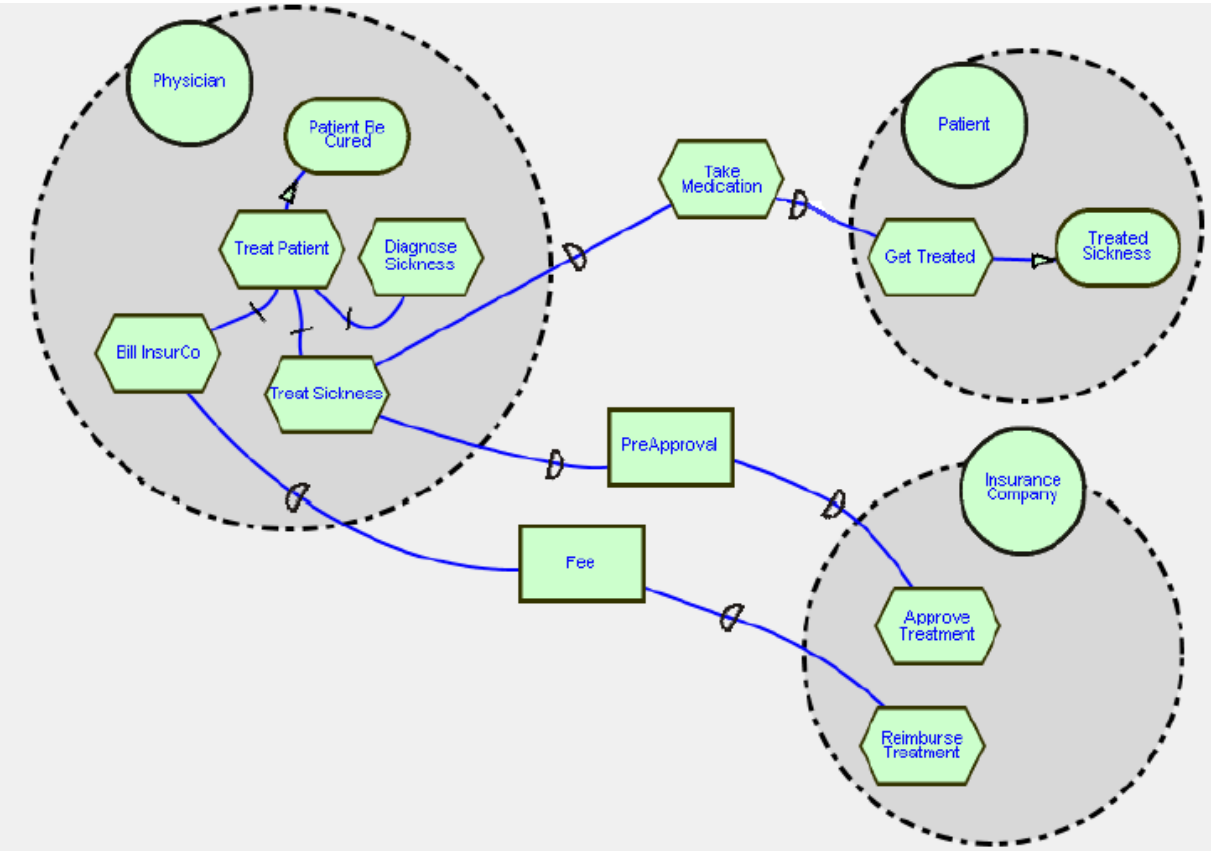


Figura 15 Modelo i* gerado do modelo BPMN da Figura 14

As heurísticas propostas por Alves [5] visam garantir a consistência entre os objetivos estratégicos e os processos de negócio de uma organização. A geração automática de um modelo a partir do outro aumentaria a produtividade na construção dos modelos e facilitaria a integração entre i* e BPMN na modelagem dos processos de negócio. A seção a seguir apresenta a tecnologia que foi utilizada para a construção da ferramenta proposta neste trabalho, que automatiza as heurísticas de transformação propostas por Alves [5].

2.6. A tecnologia EuGENia/Epsilon

Epsilon é um conjunto de linguagens e ferramentas para geração de código, transformações modelo para modelo, validação de modelos, comparação e migração de modelos que trabalham em conjunto com a tecnologia EMF (*Eclipse Modeling Framework*) [13].

A principal linguagem desse conjunto é a linguagem EOL (*Epsilon Object Language*). EOL combina as principais características de *Javascript* e OCL, e é uma linguagem imperativa para criar, consultar e modificar modelos EMF.

Epsilon também contem outras linguagens de propósito específico, que usam EOL como linguagem de expressão. São elas:

- ETL (*Epsilon Transformation Language*): é uma linguagem de transformação entre modelos (*model-to-model*) baseada em regras que permite consulta de modelos de origem e modelos destino, regras *lazy*, regras *greedy*, entre outras funções.
- EVL (*Epsilon Validation Language*): é uma linguagem que permite a validação de modelos, verificação de consistência e integração com EMF para exibição de mensagens de erro quando ocorrem falhas na validação.
- EGL (*Epsilon Generation Language*): é uma linguagem de transformação de modelos para texto (*model-to-text*) baseada em *templates* que permite geração de código, documentação e outros artefatos a partir de modelos.
- EWL (*Epsilon Wizard Language*): é uma linguagem que permite a atualização de modelos, onde as transformações são realizadas no próprio modelo dado como entrada.
- ECL (*Epsilon Comparison Language*): é uma linguagem utilizada para a descoberta de correspondências entre elementos de modelos de diversos metamodelos.
- EML (*Epsilon Merging Language*): é uma linguagem baseada em regras que visa unir modelos com elementos correspondentes, identificados a partir de ECL.
- *Epsilon Flock:* é uma linguagem baseada em regras que permite a atualização de modelos em resposta à mudança no metamodelo correspondente.

Além de prover várias linguagens para transformação, atualização e geração de modelos, a tecnologia *Epsilon* também dispõe da tecnologia *EuGENia* [14].

EuGENia funciona como um *front-end* para GMF (*Graphical Modelling Framework*). GMF é executado na plataforma Eclipse e funciona como um *plugin* que provê uma série de facilidades para a manipulação de modelos *Ecore*. O modelo *Ecore* define o metamodelo de uma linguagem, responsável por definir os principais elementos e restrições da mesma.

A Figura 16 apresenta, na notação gráfica BPMN, o fluxo de trabalho (*workflow*) do GMF.

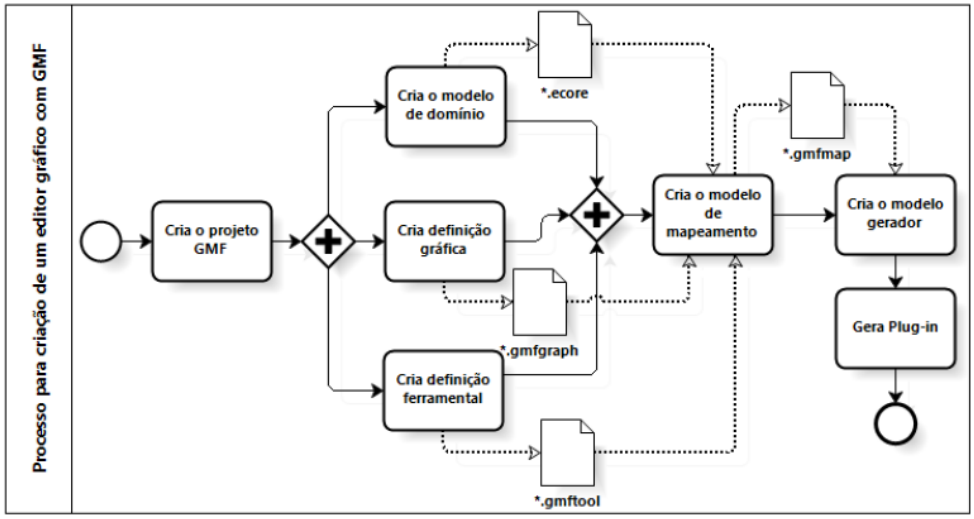


Figura 16 Processo de criação de um editor gráfico com GMF [16]

Após a criação do projeto em GMF, devem ser realizadas as atividades de criação do modelo de domínio (metamodelo), que gera como artefato um modelo em arquivo *.ecore*, criação da definição gráfica, que produz como artefato um *.gmfgraph* e definição ferramental que gera um arquivo *.gmftool*. O modelo de mapeamento pode então ser criado, gerando como artefato um arquivo *.gmfmap*.

A tecnologia *EuGENia* facilita esse processo, uma vez que gera automaticamente o *.gmfgraph*, o *.gmftool* e o *.gmfmap* necessários para a construção de uma ferramenta em GMF a partir da definição de um modelo *.ecore*. Os modelos *.ecore* podem ser construídos graficamente ou podem ser gerados a partir da definição do modelo em uma linguagem chamada *Emfatic* [15].

Para a definição das regras de transformação da ferramenta, foi utilizada a linguagem ETL, citada anteriormente. O elemento principal dessa linguagem são as *rules* (regras). A sintaxe abstrata da linguagem ETL e a sintaxe concreta de uma regra nessa linguagem são apresentadas nas figuras 17 e 18, respectivamente.

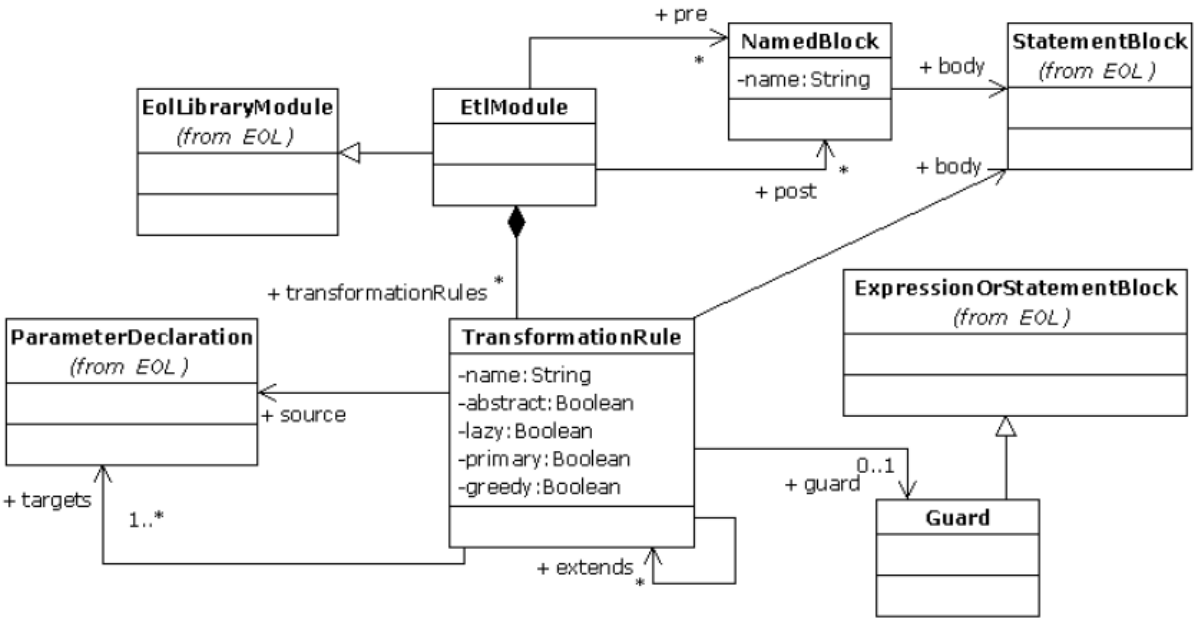


Figura 17 Sintaxe abstrata da linguagem ETL [17]

```
1  (@abstract)?
2  (@lazy)?
3  (@primary)?
4  rule <name>
5    transform <sourceParameterName>:<sourceParameterType>
6    to (<rightParameterName>:<rightParameterType>
7    (, <rightParameterName>:<rightParameterType>)*
8    (extends (<ruleName>,)*<ruleName>)? {
9
10   (guard (:expression)|({statement+}))?
11
12   statement+
13 }
```

Figura 18 Sintaxe concreta de uma regra na linguagem ETL [17]

CAPÍTULO 3

Ferramenta iStar2BPMN

Este capítulo apresenta as principais características da ferramenta de transformação entre modelos i* e BPMN, denominada iStar2BPMN. Apresenta também um exemplo ilustrativo que demonstra a execução da ferramenta.

3.1. Visão geral da ferramenta

A ferramenta *iStar2BPMN* está inserida na categoria de ferramentas CASE (*Computer-Aided Software Engineering*), que apoiam atividades de engenharia de software, e sua principal funcionalidade é a transformação entre modelos i* e BPMN.

Para a construção da ferramenta, foi utilizada a tecnologia *Epsilon*, já explicada no capítulo anterior. Além da implementação da ferramenta que automatiza as heurísticas de mapeamento, foram implementados dois editores gráficos - um para i* e outro para BPMN - que possibilitam a modelagem nas duas notações gráficas e a visualização dos modelos gerados pelas transformações. Os editores gráficos são plug-ins do *Eclipse,* já a ferramenta foi implementada de forma *standalone,* ou seja, não é associada à plataforma.

Como as heurísticas de mapeamento não são totalmente independentes da experiência do analista, conforme afirma Alves [5], a ferramenta transforma modelos de uma notação gráfica para outra de forma interativa, requisitando ao usuário informações necessárias para o processo de transformação.

O funcionamento da ferramenta será ilustrado nas seções seguintes, a partir da execução da mesma a um exemplo. As regras de transformação de i* para BPMN serão aplicadas ao exemplo, modelado com o editor gráfico do i*, e posteriormente, ao modelo BPMN gerado serão aplicadas as regras de transformação de BPMN para i*.

3.2. Aspectos de implementação

Como explicado no capítulo anterior, o primeiro passo para a definição de uma linguagem de modelagem gráfica em GMF é a criação do modelo de domínio (ou metamodelo). Para os editores gráficos do i* e do BPMN, foi necessário definir dois modelos de domínio. Para a definição desses modelos foi utilizada a linguagem *Emfatic*.

A linguagem *Emfatic* foi projetada para possibilitar a representação textual de modelos *Ecore* [15]. Os metamodelos do i* e do BPMN estão no apêndice A e B desta monografia. O metamodelo da linguagem i* foi baseado no metamodelo do i* do trabalho de Paes [16]. Já o metamodelo do BPMN foi baseado no metamodelo simplificado do BPMN definido pelo grupo de pesquisa orientado pela professora Carla Silva.

O próximo passo, após a definição dos modelos de domínio, foi a criação da definição gráfica, criação da definição ferramental, criação do modelo de mapeamento e criação do modelo gerador, conforme descrito na Figura 16. Essas atividades puderam ser realizadas automaticamente a partir da tecnologia *EuGENia/Epsilon*. No entanto, foi necessário customizar as imagens dos elementos gráficos dos editores, a partir da modificação do código gerado da definição gráfica. Foi necessário modificar as imagens de todos os elementos, inclusive as ligações de dependência, contribuição e meio-fim.

As regras de transformação foram implementadas com a linguagem ETL. Foram definidos dois conjuntos de regras de mapeamento, um para cada tipo de transformação. Cada conjunto constitui um arquivo de extensão *.etl*. As regras operam sobre arquivos no formato XMI (*XML Metadata Interchange)*, padrão para troca de informações baseado em XML. As regras de transformação de i* para BPMN recebem como entrada arquivos de extensão *.istar* e geram arquivos de extensão *.bpmn*. Analogamente, o processo inverso de transformação recebe arquivos *.bpmn* como entrada e gera arquivos *.istar* como saída.

Após a execução das regras, o usuário deve inicializar os arquivos dos diagramas (*.istardiagram* ou *.bpmndiagram*) a partir dos arquivos XMI, clicando com o botão direito no arquivo XMI e selecionando a opção correspondente a essa funcionalidade (Figura 19). Os arquivos com terminação *diagram* são os que possibilitam a visualização da representação gráfica do modelo i* ou BPMN nos editores gráficos.

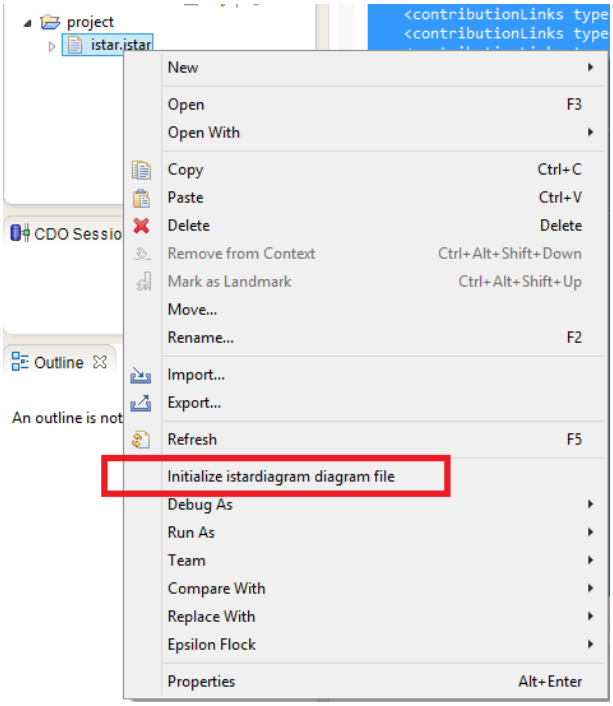


Figura 19 Funcionalidade para inicializar os arquivos com terminação *diagram*

A próxima seção ilustra a execução da ferramenta *iStar2Bpmn* ao exemplo do processo descrito no capítulo 2, o mesmo contido no trabalho de Alves [5] para ilustrar a aplicação das suas heurísticas.

3.3. Execução da ferramenta

O primeiro passo, antes do processo de transformação, é a modelagem do exemplo do processo *Managed Indemnity Insurance*, descrito no capítulo 2, no editor gráfico do i*. Os editores, tanto de i* quanto de BPMN, são constituídos por quatro painéis (comuns em ferramentas desenvolvidas com a tecnologia utilizada):

- Paleta de Ferramentas: nesse painel, o usuário pode escolher os elementos da modelagem. No caso do editor gráfico do i*, a paleta de ferramentas oferece as seguintes opções:
    - Modelagem de elementos (*IstarElement*), como tarefas, objetivos, recursos e *softgoals*;
    - Modelagem de compartimentos (*IstarComparment*), como atores, agentes, papéis e posições; e conexões, como ligações entre atores (*ISA, COVERS, ISPARTOF, OCCUPIES, PLAYS, INS*);

- Modelagem de ligações, como ligações de contribuição (*MAKE*, *BREAK*, *UNKNOWN*, *SOMEPLUS*, *SOMEMINUS*, *AND*, *OR*, *HELP*, *HURT*), ligações de dependência (*COMMITTED*, *OPEN*, *CRITICAL*) e ligações de decomposição de tarefa.

No caso do editor gráfico do BPMN, as seguintes opções são oferecidas na paleta de ferramentas:

- Modelagem de objetos de dados/artefatos (*BPMNDataObject*);
- Modelagem de eventos (*BPMNInitialEvent*, *BPMNIntermediateEvent*, *BPMNFinalEvent*);
- Modelagem de *Gateway* (*BPMNGateway*);
- Modelagem de *Pools* (Piscinas) e *Lanes* (Raias) (*BPMNPool*, *BPMNLane*);
- Modelagem de tarefas (*BPMNTask*);

- *Outline*: painel que possibilita a visualização do modelo em miniatura.
- *Stage*: onde é possível a criação e edição de modelos i*.
- Painel de Propriedades: nesse painel, é possível a visualização das propriedades dos elementos modelados, permitindo, por exemplo, a mudança dos rótulos dos mesmos no *stage*.

A figuras 20, 21 e 22 mostram a modelagem do processo *Managed Indemnity Insurance* no editor gráfico (cada figura focaliza em um ator que participa do processo). Após o salvamento da modelagem, o usuário terá um arquivo de extensão .istar, que será usado como entrada para as regras de transformação entre modelos.

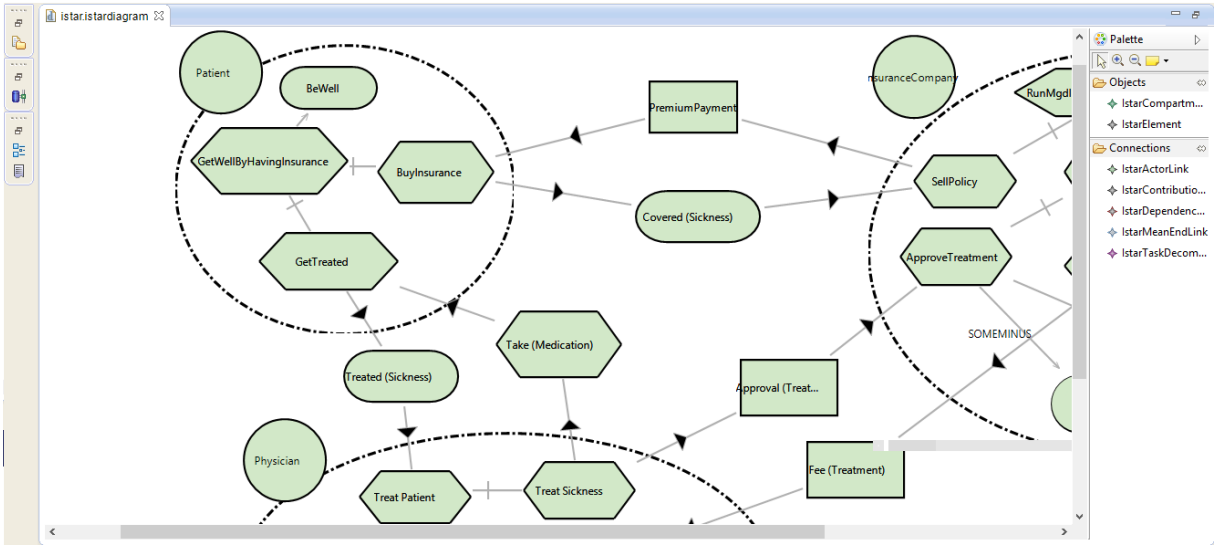


Figura 20 Ator *Patient* no processo *Managed Indemnity Insurance*

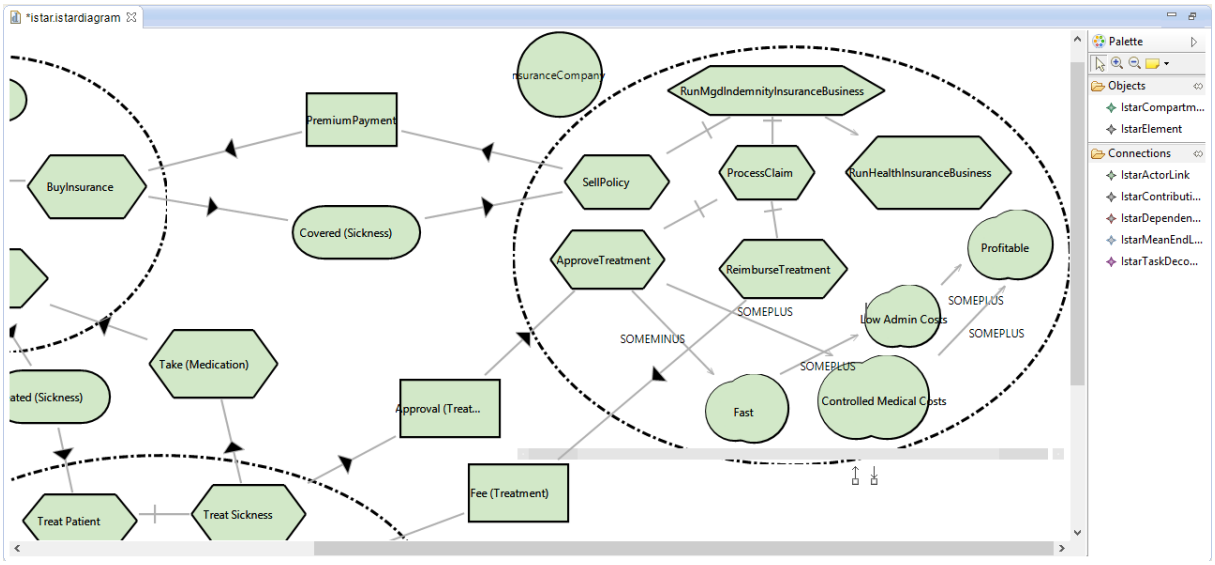


Figura 21 Ator *InsuranceCompany* no processo *Managed Indemnity Insurance*

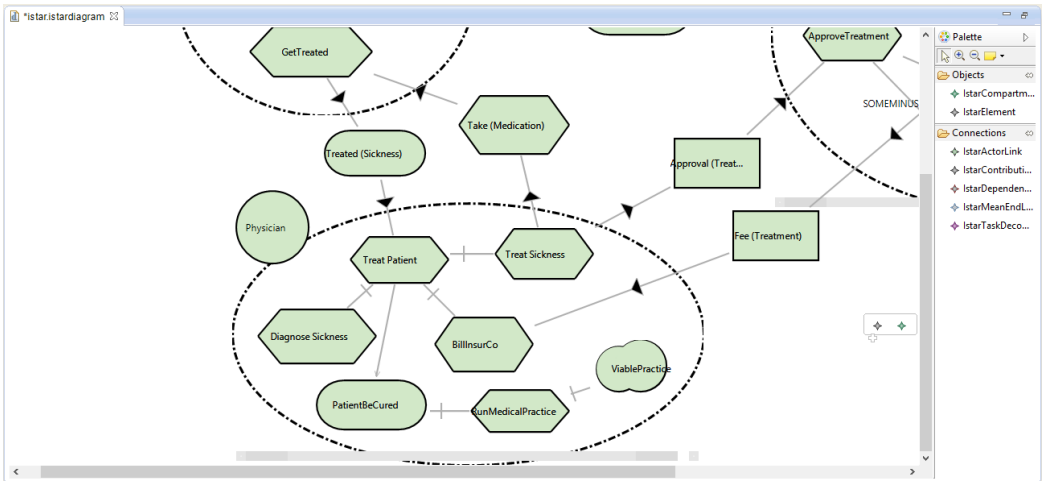


Figura 22 Ator *Physician* no processo *Managed Indemnity Insurance*

Transformação de i* para BPMN

Uma vez realizada a modelagem do processo como um modelo de razão estratégica (SR) no editor gráfico do i*, o próximo passo é a aplicação das regras de mapeamento a esse modelo. A funcionalidade de transformação entre modelos está associada aos arquivos de extensão *.istar* e de extensão *.bpmn*. A estrutura dos arquivos com essas extensões está no padrão XMI (Apêndice C).

Como explicado anteriormente, o processo de transformação entre modelos i* e BPMN é interativo, dado que as heurísticas de Alves [5] não são totalmente independentes da interferência do analista. Logo, no processo de transformação de um modelo i* em um modelo BPMN, várias perguntas são feitas ao usuário para que o modelo BPMN gerado esteja em conformidade com o que o analista deseja.

Primeiro, o usuário deve escolher o arquivo de entrada e o diretório em que o arquivo gerado deve ser salvo. Isso pode ser feito na tela principal da ferramenta, que é ilustrada na Figura 23. Em seguida, para que o processo de transformação seja iniciado, o usuário deve clicar no botão *Execute*.

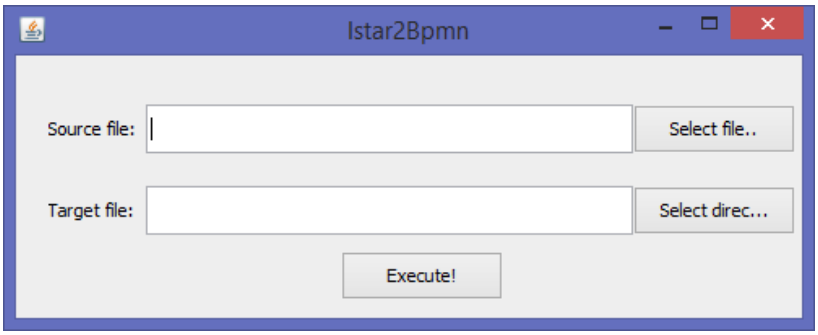


Figura 23 Tela principal da ferramenta

O primeiro passo no processo de transformação é a escolha da tarefa que será a rotina do processo (regra I no processo de transformação i* para BPMN). Após a escolha da rotina, a ferramenta calcula o escopo do processo, o que inclui o nó raiz da rotina, as suas subtarefas e as dependências ligadas a essas tarefas. A Figura 24 mostra a tela em que o usuário pode escolher a tarefa do modelo i* que será usada como rotina.

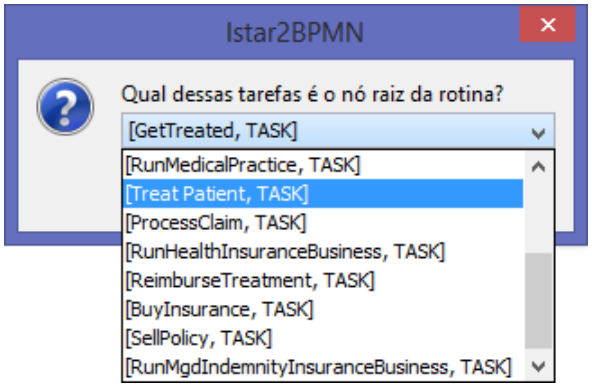


Figura 24 Tela de escolha da rotina na transformação i* para BPMN

Nessa tela, o usuário pode visualizar todas as tarefas do processo e escolher a tarefa que será o nó raiz da rotina. Após escolher a tarefa e clicar em OK, uma nova tela será apresentada ao usuário, em que se pode definir quantas organizações existem no modelo BPMN destino (Figura 25). Desta forma, pode-se escolher a organização em que cada ator se encontra, como ilustrado na Figura 27. O usuário deve escolher a rotina *Treat Patient* (como na aplicação das heurísticas ao mesmo exemplo contido no trabalho de [5]).

Na aplicação das heurísticas de mapeamento ao exemplo do processo *Managed Indemnity Insurance*, Alves [5] assume que todos os atores pertencem a organizações diferentes e, portanto, cada qual está em uma piscina diferente no modelo BPMN destino (regra II no processo de transformação de i* para BPMN). Analogamente, o usuário deve informar que existirão três organizações diferentes no modelo destino, já que o modelo i* de origem possui três atores: *Patient*, *Physician* e *InsuranceCompany*. A Figura 25 apresenta a tela em que o usuário pode informar a quantidade de organizações/piscinas presentes modelo BPMN destino.

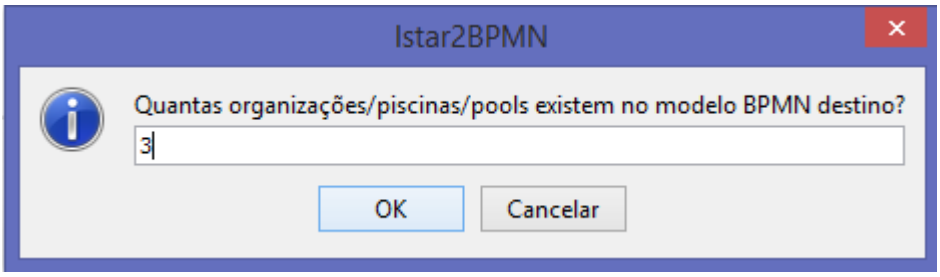


Figura 25 Tela para informar a quantidade de piscinas presentes no modelo BPMN destino

Escolhido o número de organizações do modelo BPMN destino, o usuário deve informar o nome de cada uma delas. Assumiremos que cada organização terá o mesmo nome do ator correspondente. A Figura 26 apresenta a tela em que o usuário pode definir o nome de cada organização presente no processo. Posteriormente, o usuário pode escolher em que organização/piscina está presente cada participante do modelo BPMN destino (Figura 27)

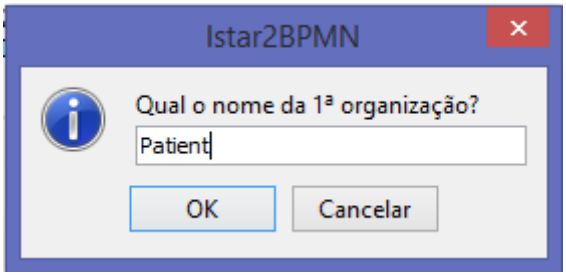


Figura 26 Tela para informar o nome de cada organização/piscina na transformação i* para BPMN

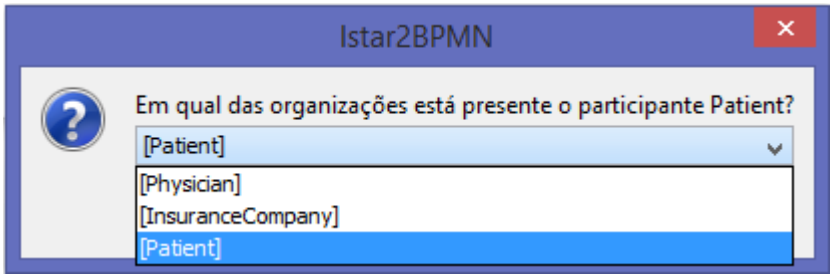


Figura 27 Tela de escolha da organização à qual um participante deve pertencer

Como a rotina do processo possui subtarefas (a tarefa *Treat Patient* possui, como subtarefas, as tarefas *Diagnose Sickness*, *Treat Sickness* e *BillInsurCo*), a ferramenta pergunta ao usuário se essas subtarefas serão atividades sequenciais no modelo BPMN destino (Figura 28. O usuário deve clicar em OK, uma vez que no trabalho de Alves [5] o modelo BPMN destino possui essas atividades realizadas em sequência.

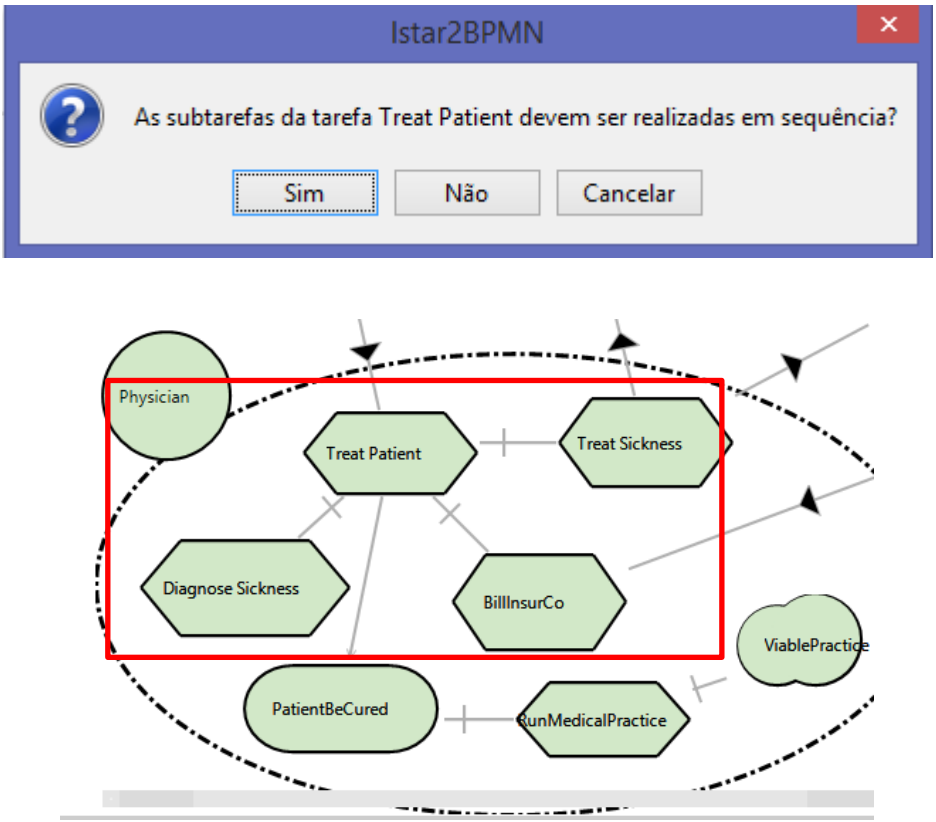


Figura 28 Tela para decidir se subtarefas serão executadas em sequência ou paralelo e fragmento do modelo relacionado à decisão

O próximo passo é a definição da ordem de execução dessas atividades (tarefas no modelo i* de origem). O usuário então deve informar qual será a primeira, a segunda e a terceira atividades a serem executadas (Figura 29).

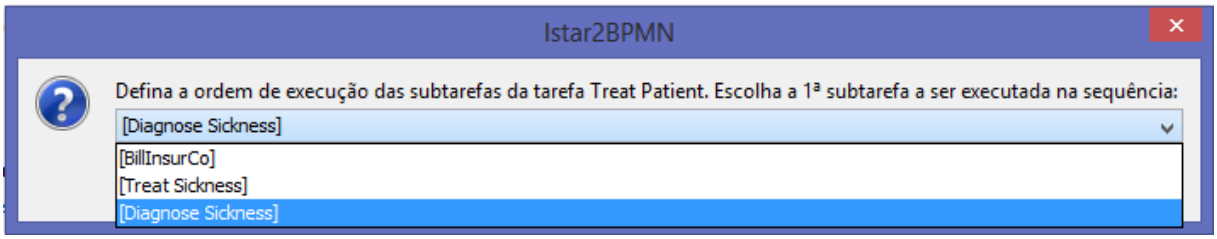


Figura 29 Tela para escolha da sequência das atividades no modelo BPMN destino

As atividades devem ser realizadas nesta ordem: *Diagnose Sickness*, *Treat Sickness*, *BillInsurCo*, conforme análise feita por *Alves* [5].

Este é o último passo que o analista precisa seguir no processo de transformação do modelo i* do processo *Managed Indemnity Insurance* para o modelo BPMN, apresentado na Figura 30. Os outros elementos são transformados de forma automática, sem a interferência do analista.

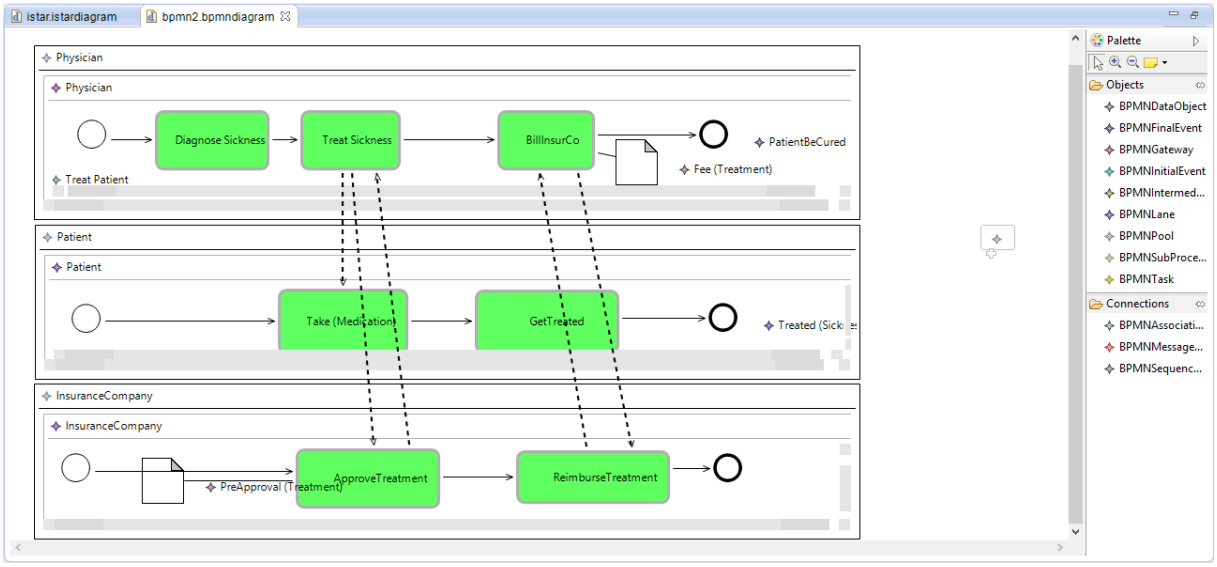


Figura 30 Modelo BPMN gerado pela transformação

O modelo da Figura 30 está em conformidade com o modelo BPMN gerado pela aplicação das heurísticas de mapeamento ao exemplo do processo *Managed Indemnity Insurance*, modelo este apresentado na seção 2.5 do capítulo 2 desta monografia. O nó raiz da rotina (*Treat Patient*) no modelo i* de origem foi transformado no evento inicial do participante *Physician*, que inicia o processo (regra IX). As sub-tarefas da tarefa *Treat Patient* foram transformadas em atividades sequenciais, com a ordem definida pelo próprio usuário no processo de transformação. As dependências de recurso existentes entre as tarefas *Treat Sickness* e *ApproveTreatment* (recurso *PreApproval*) e entre as tarefas *BillInsurCo* e *ReimburseTreatment* (recurso *Fee(Treatment)*) foram transformadas em duas ligações de mensagem de fluxo e um artefato gerado (duas ligações e um artefato para cada dependência). Os objetivos *PatientBeCured* e *Treated(Sickness)* foram transformados em eventos finais não-vazios.

Transformação de BPMN para i*

O modelo BPMN gerado na transformação i* para BPMN será utilizado como entrada para o processo de transformação de BPMN para i*. O principal objetivo desta etapa é conseguir gerar um modelo i* o mais fiel possível ao modelo que foi utilizado para a obtenção do modelo BPMN dado como entrada. A primeira pergunta que a ferramenta faz ao usuário (Figura 31) é se a tarefa *GetTreated* é subtarefa de alguma outra tarefa no modelo i* destino (regra V do processo de transformação de BPMN para i*). A tarefas são transformadas na ordem em que aparecem no arquivo XMI dado como entrada. Como a tarefa *GetTreated* é a primeira na hierarquia do arquivo, ela é transformada primeiro.

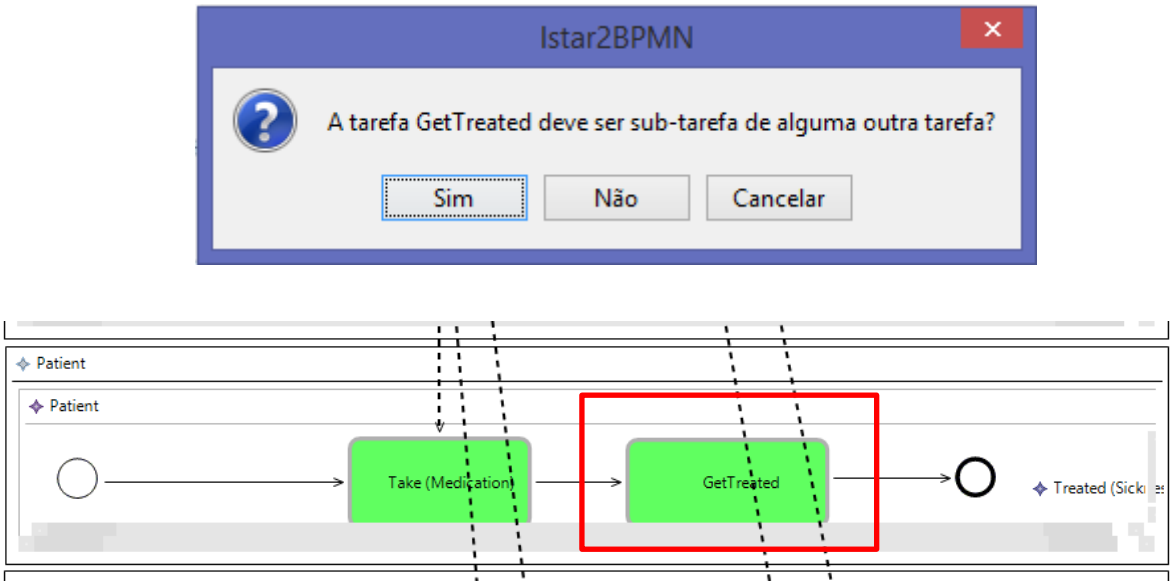


Figura 31 Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa Get Treated) e fragmento do modelo envolvido com a decisão

Como a atividade *GetTreated* no modelo BPMN de origem faz parte de uma sequência de atividades no participante *Patient*, o usuário precisa informar se a tarefa correspondente a essa atividade faz parte de alguma decomposição de tarefa (regra V da transformação BPMN para i*). Caso faça parte, o mesmo deve informar qual tarefa será a tarefa pai da tarefa correspondente a essa atividade. Como a tarefa *GetTreated* não deve fazer parte de uma decomposição de tarefa, o usuário deverá clicar no segundo botão (*Não*).

A ferramenta pergunta ainda sobre as tarefas *Diagnose Sickness*, *Treat Sickness* e *BillInsurCo*. Para cada uma delas, o usuário deve escolher a primeira opção (*OK*), uma vez que elas devem fazer parte de uma decomposição de tarefa (Figura 32). Posteriormente, a ferramenta pergunta ao usuário se a tarefa a ser decomposta corresponde a algum evento inicial no modelo BPMN de origem (Figura 33). O usuário deve clicar na primeira opção e, na próxima tela, escolher qual evento inicial corresponde à tarefa a ser decomposta no modelo i* destino (Figura 34).

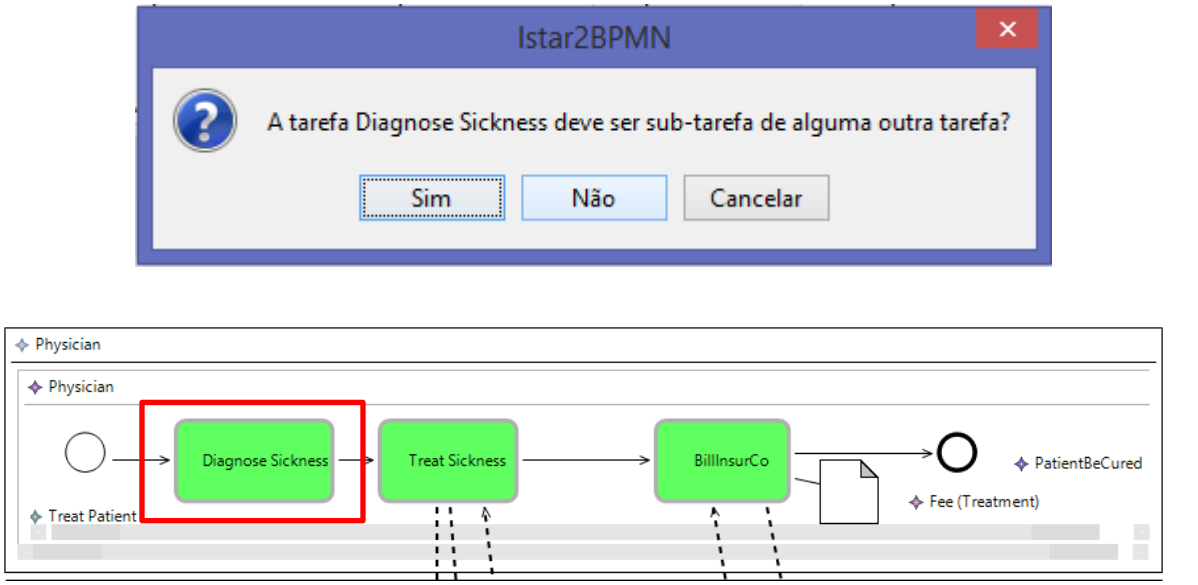


Figura 32 Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa Diagnose Sickness) e fragmento do modelo envolvido com a decisão

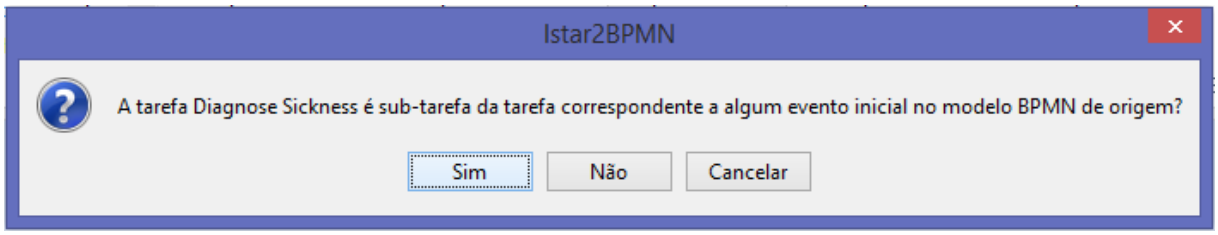


Figura 33 Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa Diagnose Sickness)

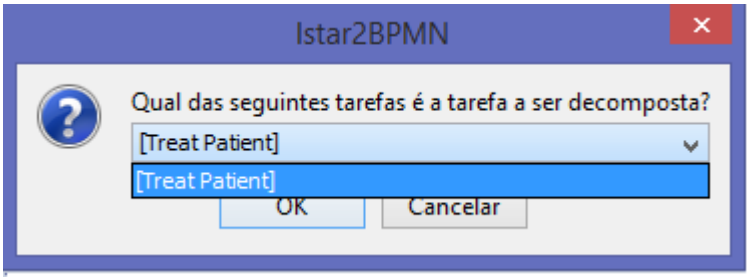


Figura 34 Tela para escolha da tarefa a ser decomposta

A mesma pergunta é feita para as atividades *ApproveTreatment* e *ReimburseTreatment*, do participante *InsuranceCompany*. Da mesma forma, como não pertencem a nenhuma decomposição de tarefa que está no escopo do processo, o usuário deve escolher a segunda opção (*Não*). Posteriormente, para cada evento final não-vazio do modelo BPMN de origem, a ferramenta indaga ao usuário se esse evento deverá ser transformado em uma dependência de objetivo ou não. Caso o usuário escolha a primeira opção, ele deverá escolher a tarefa a qual essa dependência de objetivo deve estar ligada. Caso contrário, um objetivo e uma ligação meio-fim entre esse objetivo e à tarefa que corresponde à atividade imediatamente anterior ao evento final são incluídos no ator correspondente. No entanto, caso essa tarefa faça parte de uma decomposição de tarefa, a ligação partirá da tarefa pai da decomposição. Nesse caso, o objetivo *Treated (Sickness)* e o objetivo *Patient Be Cured* devem ser incluídos como um objetivo interno ao ator, e, portanto, o usuário deve escolher a opção *Não* (Figura 35).

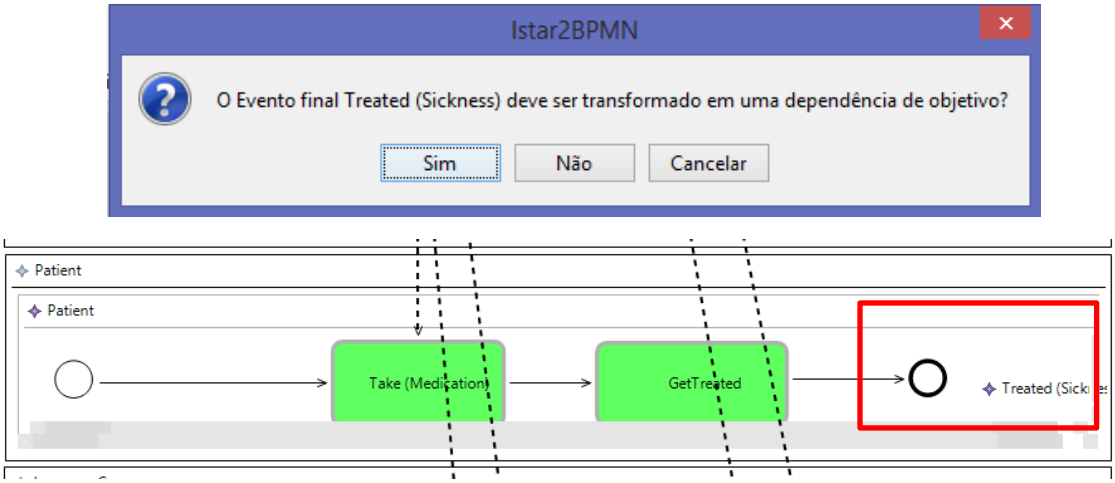


Figura 35 Tela para decidir se um evento final será transformado em uma dependência de objetivo

Assim termina o processo de transformação do modelo BPMN para o modelo i* do processo *Managed Indemnity Insurance*. A transformação dos elementos restantes é realizada sem a interferência do analista. O modelo i* gerado pela transformação é apresentado na Figura 36.

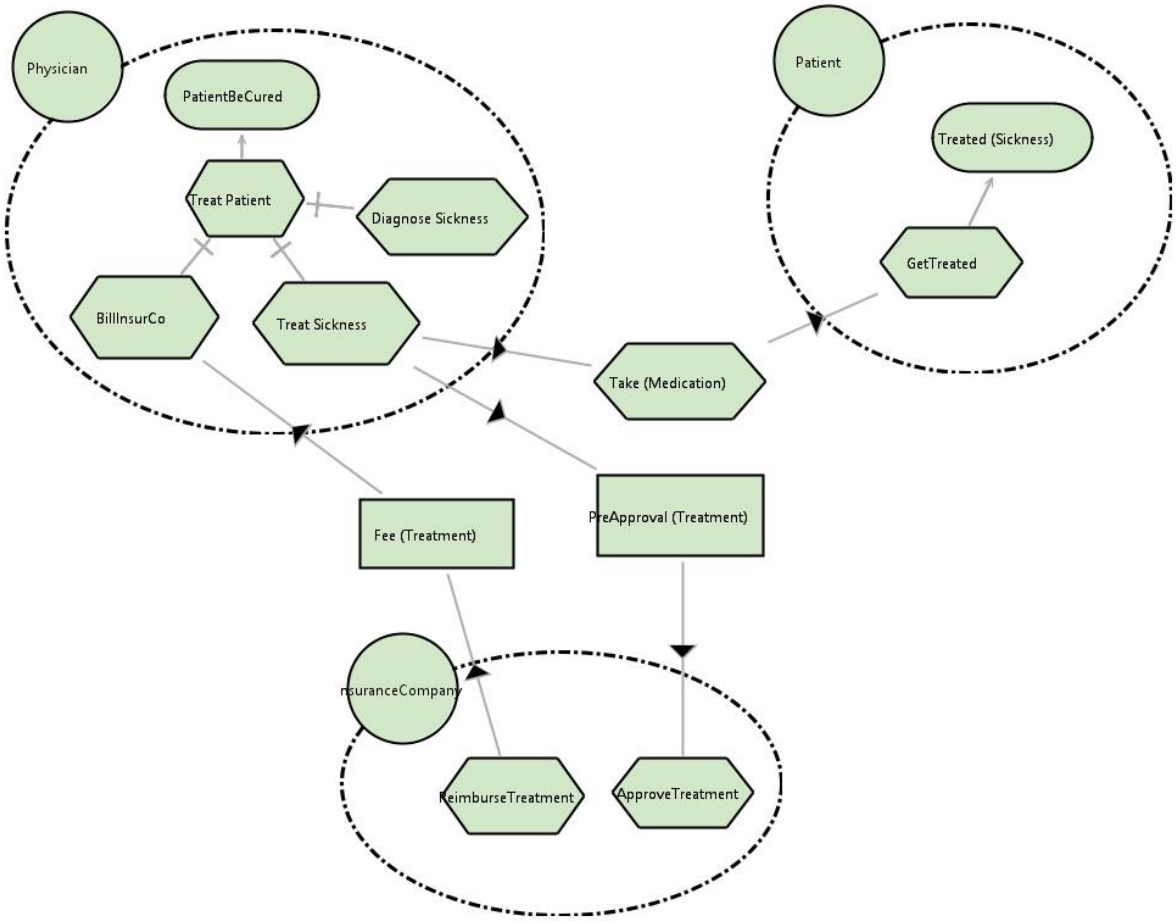


Figura 36 Modelo i* gerado pela transformação

Cada participante no modelo BPMN de origem foi transformado em um ator no modelo i* destino (regra I do processo de transformação de BPMN para i*). As ligações de fluxo de mensagem entre duas atividades que geraram artefatos foram transformadas em dependências de recurso (*Fee(Treatment)* e *PreApproval(Treatment)*) (regra III). Os eventos finais Treated(Sickness) e PatientBeCured foram transformados em objetivos internos do ator correspondente (regra IV). As atividades *DiagnoseSickness*, *TreatSickness* e *BillInsurCo* foram transformadas em sub-tarefas da tarefa *Treat Patient*.

Os *softgoals* devem ser inseridos pelo próprio usuário após a aplicação das regras de mapeamento. O analista deve inferir os *softgoals* a partir de uma análise de qualidade e atributos das tarefas, conforme recomenda Alves [5].

O modelo i* gerado pela ferramenta está em conformidade com o modelo obtido com a aplicação manual das heurísticas, apresentado na seção 2.5 desta monografia.

CAPÍTULO 4

Exemplo de Aplicação

Este capítulo apresenta a aplicação das regras de transformação ao exemplo do processo de agendamento de reuniões [2], transformando o modelo i* do mesmo em um modelo BPMN e, em seguida, fazendo o processo inverso. Além da descrição desse processo e a aplicação das regras de transformação, o capítulo apresenta algumas limitações que foram encontradas nas heurísticas de mapeamento tomadas como base para estas regras.

4.1. Descrição do processo

O processo de agendamento de reuniões já é bastante conhecido na literatura e é apresentado neste trabalho como um modelo de razão estratégica (SR). A Figura 37 mostra a modelagem desse processo no editor gráfico do i*. Nesse modelo, existem 3 atores: *Agendador*, *Participante* e *Iniciador*. O objetivo geral do *Agendador* é ter a reunião agendada. Para que esse objetivo seja atendido, o ator deve executar a tarefa de *Agendar Reunião Automaticament*e. Para que esta tarefa seja executada, é necessário que o iniciador da reunião tenha executado a tarefa de *Entrar Datas Disponíveis*. O agendador de reunião ainda deve obter as datas disponíveis para a reunião (*Obter Datas Disponíveis*), propor uma data (*Propor Data*) e obter a concordância dos participantes da reunião (*Obter Concordância*). Para obter as datas disponíveis, o agendador precisa que os participantes da reunião executem a tarefa de *Entrar Datas Disponíveis*.

Do lado do *Iniciador*, a principal tarefa a ser executada é a de *Organizar Reunião*. Para isso, é necessário que a tarefa seja rápida (*Rápido*) e de pouco esforço (*Pouco Esforço*) e que a reunião seja agendada (*Reunião Agendada*). As tarefas de *Agendar Reunião* e *Deixar Agendador Agendar Reuniões* contribuem positivamente (ligações de contribuição) para que a que o processo seja rápido e de pouco esforço. Por outro lado, essas tarefas possibilitam (ligações meio-fim) o agendamento da reunião (*Reunião Agendada*).

O *Participante*, por sua vez, tem como principal tarefa a ser executada a de *Participar de Reunião*. Para isso, ele deve comparecer à reunião (*Comparecer Reunião*), encontrar uma data conveniente para reunião (*Conveniente (Reunião, Data)*) e concordar com a reunião (*Concordar Reunião*).

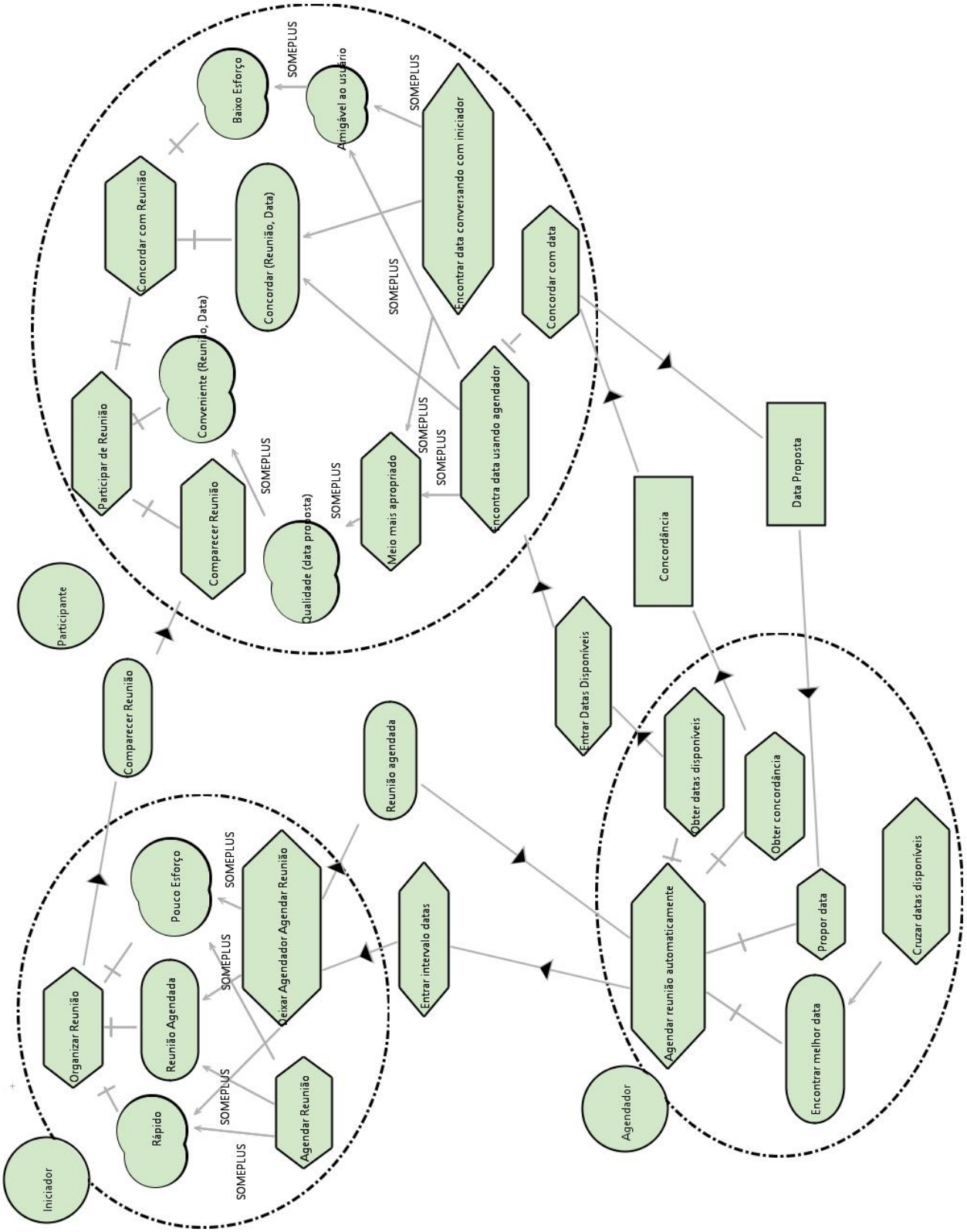


Figura 37 Modelo SR do exemplo do agendador de reuniões no editor gráfico do i*

4.2. Transformação de i* para BPMN

O modelo BPMN gerado pela ferramenta a partir da aplicação ao exemplo do modelo i* da Figura 37 é apresentado na Figura 38. Para o processo de transformação, foi escolhida como rotina a tarefa *Agendar reunião automaticamente*, do ator *Agendador*.

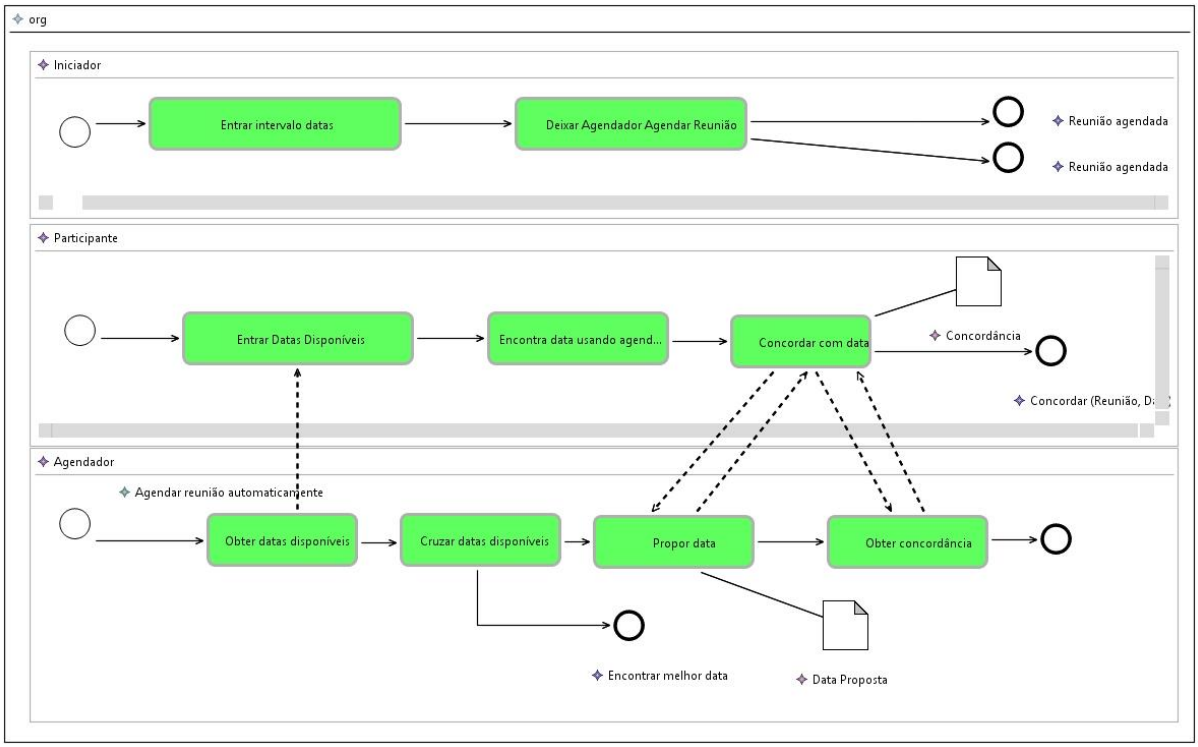


Figura 38 Modelo BPMN gerado pela transformação – exemplo agendador de reuniões

A rotina foi então transformada no evento inicial do participante *Agendador*. Cada dependência de recurso foi transformada em duas ligações de fluxo de mensagem, com um artefato gerado (artefatos *Concordância* e *Data Proposta*). O objetivo interno *Concordar (Reunião, Data)* foi transformado no evento final do ator *Participador*. A dependência de tarefa *Entrar Datas Disponíveis* foi incluída como uma atividade na *lane* do ator *dependee*, no caso o ator *Participador*. A dependência de objetivo *Reunião Agendada* foi transformada em um evento final do participante *Iniciador* e o objetivo interno *Reunião Agendada* também foi transformado em um evento final do mesmo participante. A tarefa *Cruzar datas disponíveis* foi inserida entre as atividades *Obter datas disponíveis* e *Propor data* (decisão feita pelo usuário durante o processo de transformação). O objetivo *Encontrar melhor data* foi transformado em um evento final.

Limitações na aplicação das regras i* para BPMN

Esta seção apresenta as limitações encontradas no processo de transformação de i* para BPMN, levando em conta a aplicação ao exemplo de agendamento de reuniões. Tais limitações devem ser corrigidas numa versão futura das heurísticas e da ferramenta.

O modelo esperado após a execução da transformação é apresentado na Figura 39. A atividade *Agendar reunião automaticamente* deveria estar presente no modelo BPMN destino gerado pela ferramenta, pois à tarefa que corresponde a essa atividade estão ligadas a dependência de objetivo *Reunião agendada* e a dependência de tarefa *Entrar intervalo datas*. Essa é uma limitação presente nas heurísticas de mapeamento propostas por Alves [5], que determina que a tarefa raiz da rotina deve ser transformada no evento inicial que desencadeia o processo (Regra IX da transformação i* para BPMN).

A seguir, destacamos as limitações encontradas na aplicação das heurísticas:

- De acordo com a regra VIII na transformação de i* para BPMN, um objetivo deve ser transformado em um evento final, sendo ele uma dependência ou elemento interno de um ator. No caso do processo de agendamento de reuniões, temos o objetivo *Reunião Agendada*, interno ao ator *Iniciador*; e o objetivo *Reunião Agendada*, como uma dependência entre a tarefa *Deixar Agendador Agendar Reunião*, do ator *Iniciador*, e a tarefa *Agendar reunião automaticamente*, do ator *Agendador*. Os dois objetivos foram incluídos no participante *Iniciador* do modelo BPMN destino (de acordo com a regra, embora tenham o mesmo nome).
- Não foi possível incluir uma ligação de fluxo de mensagem entre a tarefa *Agendar reunião automaticamente* e a tarefa *Deixar Agendador Agendar Reunião*, correspondente à dependência de objetivo *Reunião Agendada* (3ª regra de consistência na transformação i* para BPMN), pois a primeira é transformada em um evento inicial no modelo BPMN destino.
- A dependência de tarefa *Entrar intervalo datas* também foi perdida. Como *Agendar reunião automaticamente* é um evento inicial no modelo destino, não foi possível incluir uma ligação de mensagem de fluxo que corresponde à dependência. Esses problemas deixaram o participante *Iniciador* desconectado do processo.
- Após a tarefa *Cruzar datas disponíveis*, existem dois fluxos de sequência possíveis sem uma condição para que um deles seja escolhido. Isso deixa o modelo ambíguo.

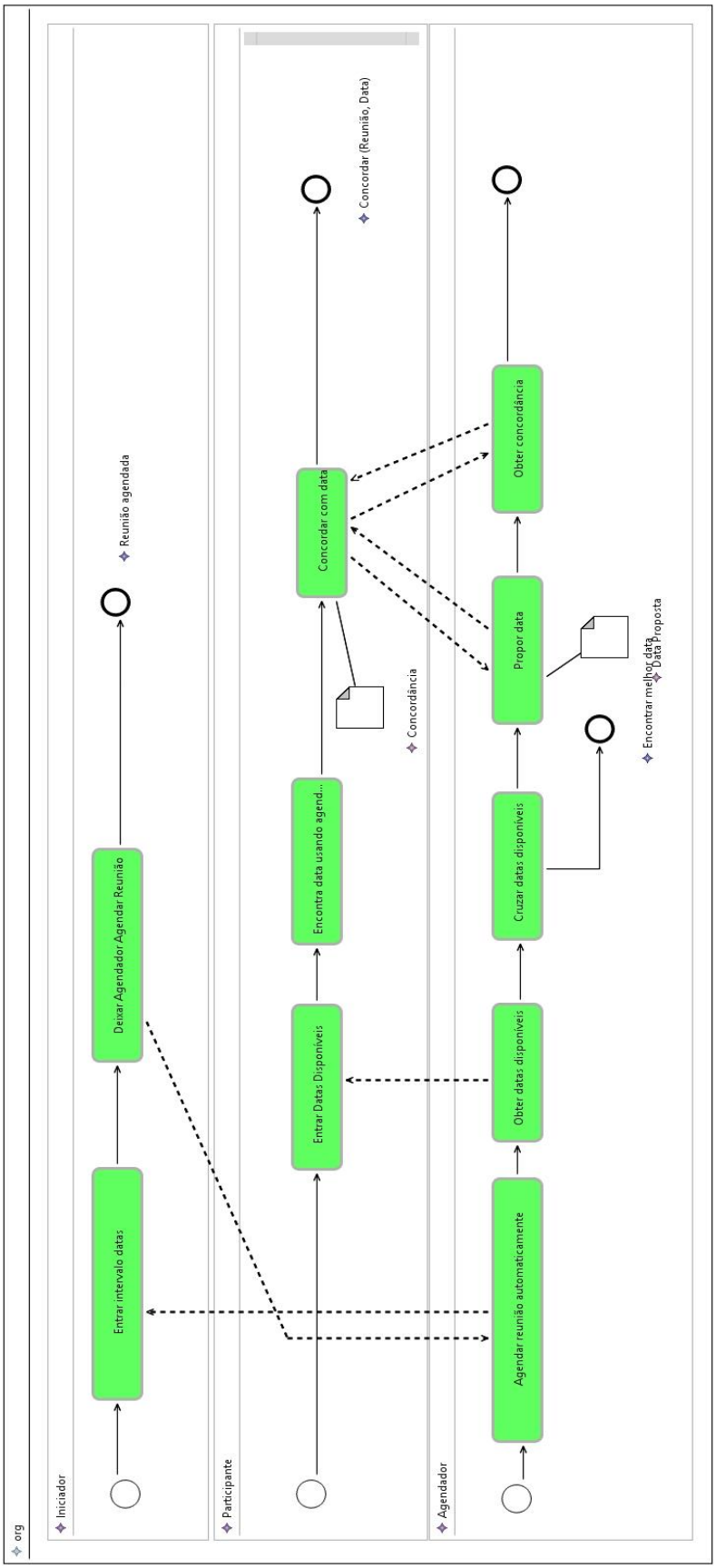


Figura 39 Modelo BPMN esperado após a transformação i* para BPMN – exemplo agendador de reuniões

4.3. Transformação de BPMN para i*

Para a transformação de BPMN para i*, usaremos o modelo BPMN que esperávamos obter na transformação de i* para BPMN (Figura 39) uma vez que o resultado dessa transformação não foi satisfatório. O modelo i* gerado é apresentado na Figura 40.

As atividades do modelo BPMN de origem foram transformadas em tarefas no modelo i* destino. As atividades que não possuem ligação de mensagem de fluxo ligada a elas foram transformadas em tarefas internas ao ator correspondente. As atividades que possuíam alguma ligação de mensagem de fluxo ligada a elas foram transformadas em dependências de tarefa. Os artefatos, juntamente com as ligações de fluxo de mensagem entre atividades que geram um artefato, foram transformados em dependências de recurso.

Limitações na aplicação das regras BPMN para i*

Esta seção apresenta as limitações encontradas nas heurísticas de mapeamento BPMN para i*. Tais limitações também devem ser corrigidas numa versão futura das heurísticas e da ferramenta.

No transformação BPMN para i*, a única limitação encontrada foi a que se refere à decomposição de tarefa correspondente à rotina do processo. O objetivo *Encontrar melhor data* deveria fazer parte dessa decomposição de tarefa. Isso decorre do fato de não existir uma heurística para incluir sub-objetivos no modelo i* destino. Esse objetivo foi então tratado como um objetivo interno do ator *Agendador*. Por outro lado, a tarefa *Cruzar datas disponíveis* foi incluída como uma subtarefa na decomposição da rotina, pois fazia parte da sequência de atividades principal do ator *Agendador* no modelo BPMN de origem.

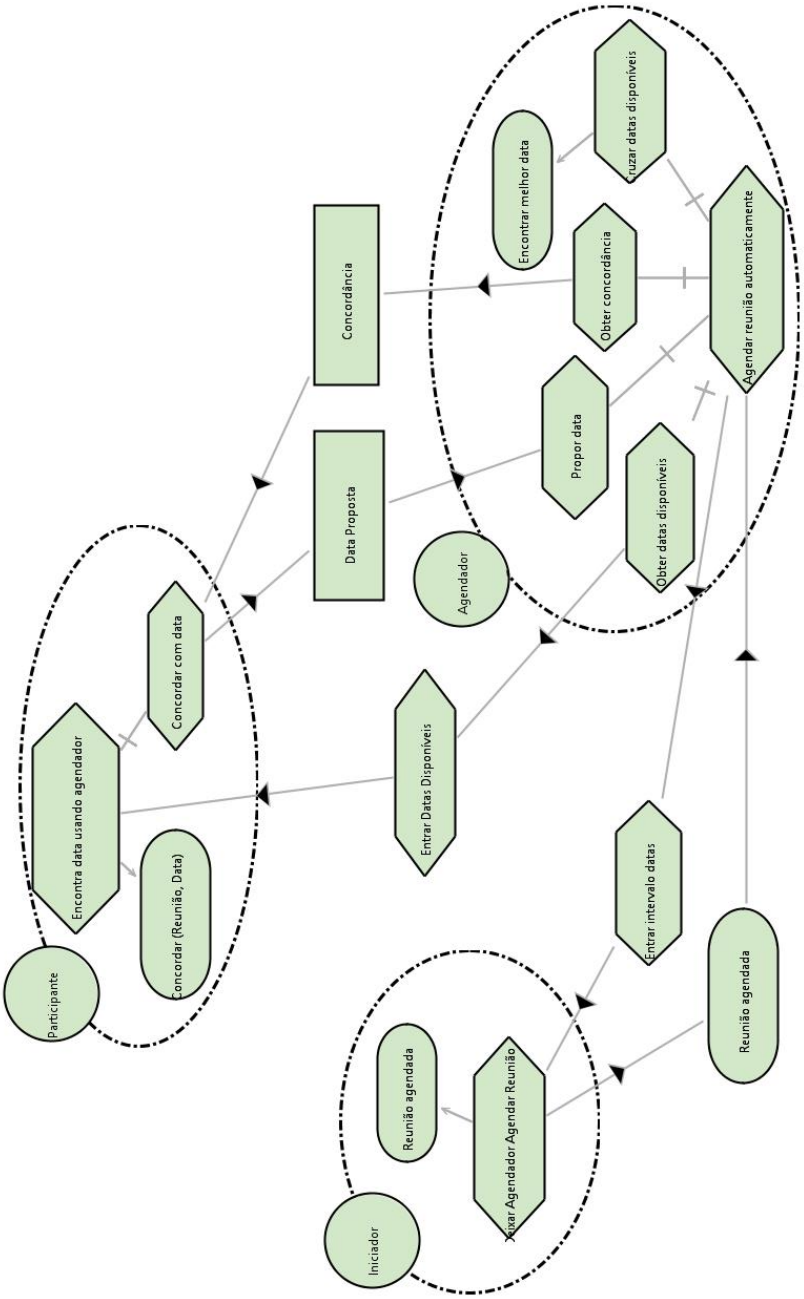


Figura 40 Modelo i* gerado na transformação BPMN para i* - exemplo do Agendador de reuniões

CAPÍTULO 5

Conclusão

Este capítulo apresenta as contribuições e limitações encontradas durante a elaboração deste trabalho. Além disso, faz-se um direcionamento para trabalhos futuros, a partir do que aqui foi desenvolvido, bem como apresenta-se as considerações finais desta monografia.

5.1. Contribuições e Limitações

O presente trabalho engloba a construção de uma ferramenta para transformação bi-direcional entre modelos i* e BPMN a partir das heurísticas definidas por Alves [5]. Essas heurísticas possibilitam a integração da abordagem orientada a objetivos com os modelos de processos de negócio, o que promove o alinhamento dos processos de uma organização com os objetivos estratégicos da mesma.

Como contribuição deste trabalho, a ferramenta *iStar2Bpmn* facilita a adoção conjunta dos modelos i* e BPMN na organização, uma vez que automatiza as heurísticas propostas por Alves [5], aumentando a produtividade na construção dos modelos e garantindo a consistência entre eles. Outra contribuição desse trabalho foi a aplicação das heurísticas a um exemplo de aplicação, o que permitiu identificar limitações nas heurísticas de mapeamento, além daquelas já reconhecidas por Alves [5].

Dentre as limitações deste trabalho, podemos citar como a principal o grande número de passos que devem ser seguidos pelo analista para transformar um modelo no outro, que depende da quantidade de elementos presente no modelo. Outra limitação é a ausência de uma interface gráfica mais elaborada que minimize o número de passos (telas para tomada de decisão) no processo de transformação. Além disso, os editores gráficos de modelos i* e BPMN ainda não possuem checadores de sintaxe. A ferramenta também ainda não é capaz de lidar com subprocessos do BPMN.

Consideradas as contribuições e limitações deste trabalho, a próxima seção apresenta as direções e perspectivas futuras.

5.2. Direções Futuras

Como direcionamentos futuros para a ferramenta *iStar2Bpmn*, podemos citar:

- Melhorar a usabilidade do processo de transformação para que a quantidade de passos para transformação de um modelo no outro seja mínima. Isto poderá ser alcançado com a implementação de uma interface gráfica mais elaborada, que permita a interação do usuário com o modelo na escolha dos elementos envolvidos em algumas regras de transformação que precisam da interferência do analista (i.e., semiautomáticas);
- Implementação da funcionalidade de validação de sintaxe para os editores gráficos do i* e do BPMN;
- Implementação de uma funcionalidade para verificação de consistência entre modelos i* e BPMN;
- Correções na paleta de ferramentas dos editores gráficos de i* e BPMN, para que a mesma contenha os símbolos dos elementos a serem modelados;
- Extensão da ferramenta para que a mesma lide com subprocessos do BPMN;
- Integração da ferramenta aos editores gráficos;
- Avaliar a ferramenta a partir de um questionário direcionado aos usuários, considerando aspectos como facilidade na construção dos modelos, usabilidade e satisfação;
- Escrita de um manual de utilização da ferramenta.

Além dos direcionamentos enumerados acima, a ferramenta deverá evoluir conjuntamente com as heurísticas de mapeamento definidas por [5]. O código da ferramenta será disponibilizado pelo autor em um repositório na web.

Na próxima seção são apresentadas as considerações finais desta monografia.

5.3. Considerações Finais

Este trabalho teve seus objetivos alcançados, pois possibilitou a implementação de uma ferramenta capaz de transformar modelos i* em modelos BPMN e vice-versa, com base nas heurísticas de mapeamento contidas no trabalho de Alves [5].

A produção deste trabalho foi muito importante para o desenvolvimento acadêmico e científico do autor, uma vez que permitiu ao mesmo o estudo da modelagem de processos em i* e BPMN, estudo da tecnologia para definição e criação de metamodelos para linguagens de modelagem, criação de editores gráficos no ambiente de desenvolvimento Eclipse, estudo e avaliação das heurísticas de mapeamento definidas por Alves [5], tradução dessas heurísticas para regras de transformação na linguagem ETL e automatização da transformação entre modelos i* e BPMN.

Dentre as dificuldades encontradas pelo autor na realização deste trabalho, cumpre destacar a alta curva de aprendizagem no estudo da tecnologia envolvida para a construção da ferramenta, a complexidade na implementação das regras de transformação e o curto tempo disponível para a realização deste trabalho. Alguns pontos ficarão como trabalhos futuros, como a implementação de checadores de sintaxe para os editores gráficos, por exemplo, como explicado na seção anterior. Essas melhorias requerem um maior tempo para serem estudadas e implementadas.

Referências Bibliográficas

[1] KANDT, R. K. **Software Requirements Engineering: Practices and Techniques**. Jet Propulsion Laboratory, California Institute of Technology, 2003.

[2] YU, Eric S. K. **Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering**, Proceedings of the Third IEEE International Symposium on, 1997.

[3] VARELA, J. P.; SANTANDER, V. F. A., Silva, I. F. **Integrando o Framework i* ao Processo de Gerência de Riscos**. Congresso Ibero-Americano em Engenharia de Software, 2012.

[4] DECREUS, K.; SNOECK, M.; POELS, G. **Practical Challenges for Methods Transforming i* Goal Models into Business Process Models**. In: 17th IEEE International Requirements Engineering Conference (RE '09), 2009.

[5] ALVES, R. S. **Integração do modelo i* com o BPMN para a obtenção de um processo de negócio melhorado a fim de alcançar as metas estratégicas da organização**. Trabalho de Graduação. Universidade Federal de Pernambuco, 2013.

[6] PRESSMAN, R. S. **Engenharia de software**. 6ª ed. Porto Alegre: Bookman, 2006.

[7] BROOKS, Frederick P. **No Silver Bullet – Essence of accidents in software engineering.** University of North Carolina at Chapel Hill. 1986.

[8] SOMMERVILLE, Ian. **Software Engineering**. 9th ed. Pearson. 2011.

[9] YU, E. **Social Modeling and i***. Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos, Springer-Verlag, Berlin, Heidelberg, 2009.

[10] YU, E. **Modeling Strategic Relationships for Process Reengineering**. Ph.D. Thesis, Graduate Dept. of Comp. Science, University of Toronto. 1995.

[11] WHITE, S. A. **Introduction to BPMN**. IBM Corporation, 2004. Disponível em: <www.bptrends.com>. Acesso em: 27/11/2013.

[12] KOLIADIS, G.; VRANESEVIC, A.; BHUIYAN, M.; KRISHNA, A.; GHOSE, A.: **Combining i* and BPMN for Business Process Model Lifecycle Management**. In: Business Process Management Workshops, 2006.

[13] The Eclipse Foundation. **Epsilon**. 2012. Disponível em: http://www.eclipse.org/epsilon/. Acesso em: 06/12/2013.

[14] The Eclipse Foundation. **EuGENia**. 2012. Disponível em: http://www.eclipse.org/epsilon/doc/eugenia/. Acesso em: 06/12/2013

[15] The Eclipse Foundation. **Emfatic Language Reference**. 2012. Disponível em: http://www.eclipse.org/epsilon/doc/articles/emfatic/. Acesso em: 06/12/2013.

[16] PAES, J. AGILE: **Uma Abordagem para Geração Automática de Linguagens i***. Dissertação de Mestrado em Ciência da Computação. Universidade Federal de Pernambuco, 2011.

[17] KOLOVOS, D.; ROSE, L.; GARCIA-DOMINGUEZ, A.; PAIGE, R. **The Epsilon Book**. Disponível para download em: https://www.eclipse.org/epsilon/doc/book/. Acesso em: 16/12/2013.

APÊNDICE A

Metamodelos em Emfatic

BPMN

```
@namespace(uri="bpmnmodel", prefix="BPMNModel")

@gmf
package bpmnmodel;

@gmf.diagram(model.extension="bpmn", diagram.extension="bpmndiagram")
class BPMNModel {
	attr String title;

	val BPMNPool [*] pools;                              // Model pools
	val BPMNMessageFlow [*] messageFlows;          // Model message flows
	val BPMNAssociation [*] associations;          // Model associations
}

@gmf.node(label="name", figure="rectangle", border.color="0,0,0")
class BPMNPool extends BPMNMessageElement {
	attr String name;                // Pool name

	@gmf.compartment
	val BPMNLane [*] lanes;    // Pool lanes
}

@gmf.node(label="name", figure="rectangle")
class BPMNLane {
	attr String name;

	@gmf.compartment
	val BPMNNode [*] nodes;                                          // Lane nodes

	val BPMNSequenceFlow [*] sequenceFlows;       // Lane sequence flows
}

@gmf.link(source="sourceA", target="targetA", color="0,0,0")
class BPMNAssociation {
	ref BPMNActivity sourceA;
	ref BPMNDataObject targetA;

	ref BPMNDataObject sourceB;
	ref BPMNActivity targetB;

	ref BPMNSequenceFlow sourceC;
	ref BPMNDataObject targetC;

	ref BPMNDataObject sourceD;
	ref BPMNSequenceFlow targetD;
}
```

```
@gmf.link(figure="br.ufpe.cin.bpmntool.figures.BPMNMessageFlowFigure",
label="label", source="source", target="target", target.decoration="arrow",
color="0,0,0")
class BPMNMessageFlow {
	attr String label;

	ref BPMNMessageElement source;
	ref BPMNMessageElement target;
}

class BPMNMessageElement {
	attr String label;
}

class BPMNNode extends BPMNMessageElement {
	// Has no attributes
}

class BPMNEvent extends BPMNNode {
	// Has no attributes
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNFinalEventFigure",
label="label", label.placement="external")
class BPMNFinalEvent extends BPMNEvent {
	// Has no attributes
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNInitialEventFigure",
label="label", label.placement="external", resizable="false")
class BPMNInitialEvent extends BPMNEvent {
	// Has no attributes
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNIntermediateEventFigure",
label="label", label.placement="external")
class BPMNIntermediateEvent extends BPMNEvent {
	// Has no attributes
}

class BPMNActivity extends BPMNNode {
	attr String name;
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNTaskFigure", size="30,60",
label="name")
class BPMNTask extends BPMNActivity {
	// Has no attributes
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNSubProcessFigure", size="3,2",
label="label", label.placement="external")
class BPMNSubProcess extends BPMNActivity {
	// Has no attributes
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNGatewayFigure", size="1,1",
label="condition")
class BPMNGateway extends BPMNNode {
```

```
	attr String condition;
}

@gmf.node(figure="br.ufpe.cin.bpmntool.figures.BPMNDataObjectFigure",
label="label", label.placement="external")
class BPMNDataObject extends BPMNNode {
	attr String name;
}

@gmf.link(figure="br.ufpe.cin.bpmntool.figures.BPMNSequenceFlowFigure",
label="label", label.pattern="{0}", source="source", target="target",
target.decoration="arrow", color="0,0,0")
class BPMNSequenceFlow {
	attr String label;

	ref BPMNNode source;
	ref BPMNNode target;
}
```

I*

```
@namespace(uri="IstarModel", prefix="IstarModel")

@gmf
package top;

@gmf.diagram(model.extension="istar", diagram.extension="istardiagram")
class IstarModel {
	attr String title;

	val IstarDependencyLink [*] dependencyLinks;
	val IstarActorLink [*] actorLinks;
	val IstarElement [*] elements;
	val IstarCompartment [*] compartments;
}

@gmf.node(label = "name", figure="figures.IstarElementFigure", label.icon="false")
class IstarElement extends IstarIntentionalElement {
	attr IstarElementType type;
}


@gmf.node(label = "name", figure="figures.IstarCompartmentFigure",
label.icon="false")
class IstarCompartment extends IstarNodeObject {
	attr IstarCompartmentType type;

	val IstarMeanEndLink [*] meanEndLinks;
	val IstarTaskDecomposition [*] tasksDecompositions;
	val IstarContributionLink [*] contributionLinks;

	@gmf.compartment(collapsible="true")
	val IstarElement [*] elements;
}
```

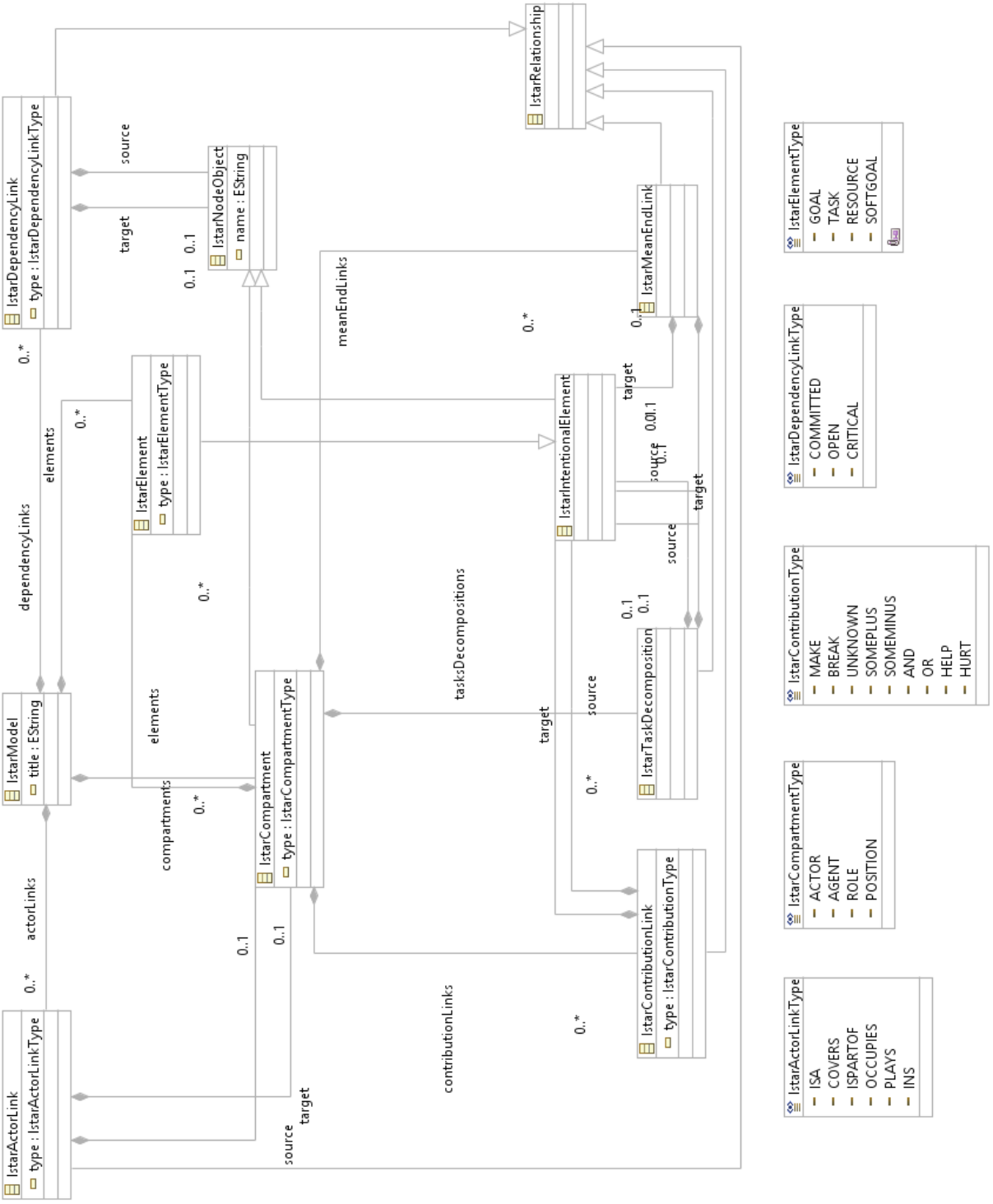
```
@gmf.link(figure="figures.IstarActorLinkFigure", source="source", target="target",
target.decoration="arrow", width="2", label="type")
class IstarActorLink extends IstarRelationship {
	attr IstarActorLinkType type;

	ref IstarCompartment source;
	ref IstarCompartment target;
}

@gmf.link(figure="figures.IstarDependencyLinkFigure", source="source",
target="target", target.decoration="arrow", width="2")
class IstarDependencyLink extends IstarRelationship {
	attr IstarDependencyLinkType type;

	ref IstarNodeObject source;
	ref IstarNodeObject target;
}

class IstarNodeObject {
	attr String name;
}

class IstarRelationship {

}

@gmf.link(figure="figures.IstarMeanEndLinkFigure", source="source",
target="target", target.decoration="arrow", width="2")
class IstarMeanEndLink extends IstarRelationship {
	ref IstarIntentionalElement source;
	ref IstarIntentionalElement target;
}

class IstarIntentionalElement extends IstarNodeObject {

}

@gmf.link(figure="figures.IstarTaskDecompositionFigure", source="source",
target="target", target.decoration="arrow", width="2")
class IstarTaskDecomposition extends IstarRelationship {
	ref IstarIntentionalElement source;
	ref IstarIntentionalElement target;
}

@gmf.link(figure="figures.IstarContributionLinkFigure", source="source",
target="target", target.decoration="arrow", width="2")
class IstarContributionLink extends IstarRelationship {
	attr IstarContributionType type;

	ref IstarIntentionalElement source;
	ref IstarIntentionalElement target;
}

enum IstarCompartmentType {
	ACTOR;
	AGENT;
	ROLE;
	POSITION;
}
```

```
enum IstarElementType {
	GOAL;
	TASK;
	RESOURCE;
	SOFTGOAL;
}

enum IstarDependencyLinkType {
	COMMITTED;
	OPEN;
	CRITICAL;
}

enum IstarActorLinkType {
	ISA;
	COVERS;
	ISPARTOF;
	OCCUPIES;
	PLAYS;
	INS;
}

enum IstarContributionType {
	MAKE;
	BREAK;
	UNKNOWN;
	SOMEPLUS;
	SOMEMINUS;
	AND;
	OR;
	HELP;
	HURT;
}
```

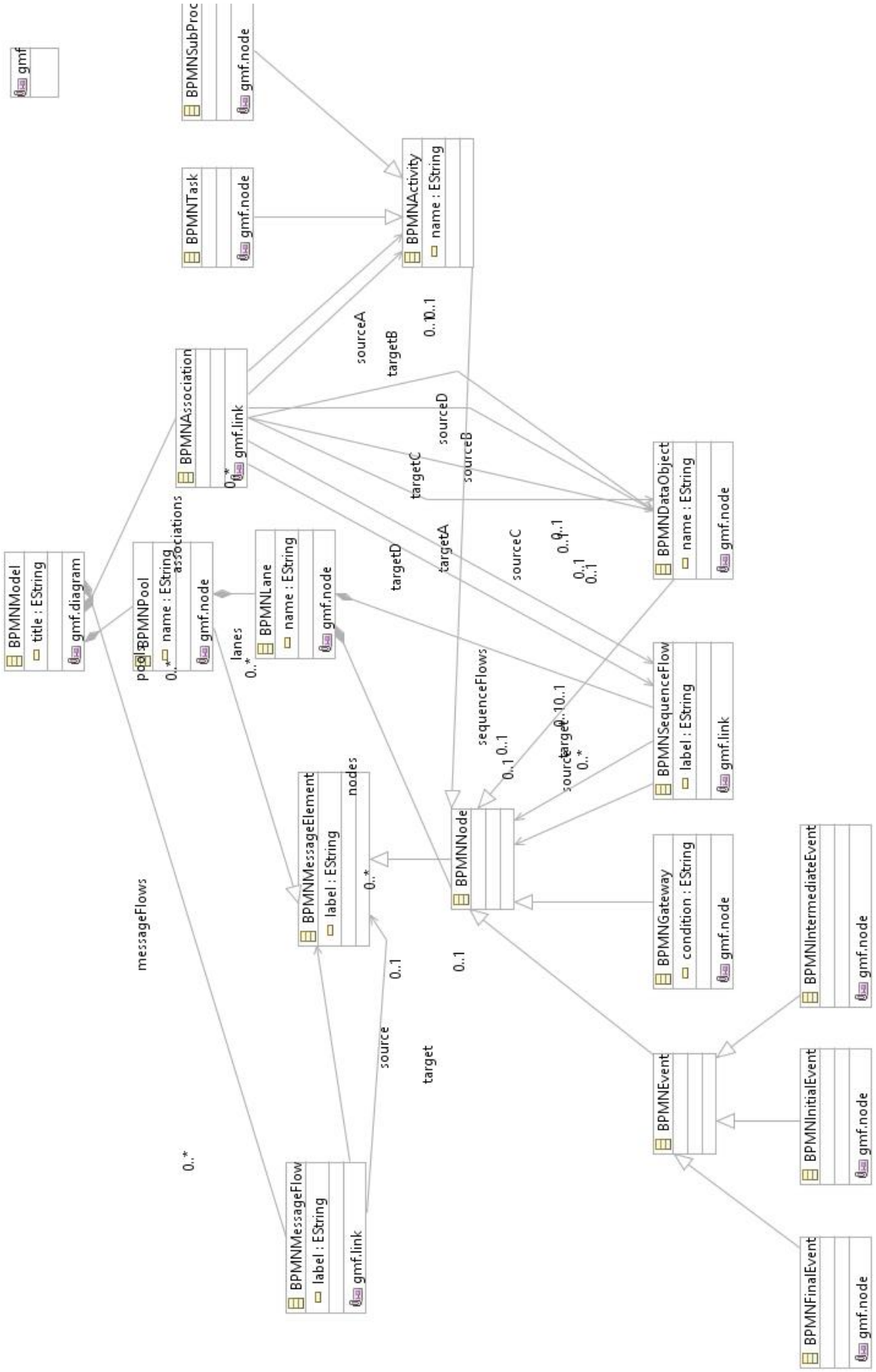
APÊNDICE B

Representação gráfica dos metamodelos

I*



BPMN


gmf
BPMNSubProc
gmf.node
BPMNTask
gmf.node
BPMNActivity
name : EString
BPMNAssociation
gmf.link
sourceA
targetB
0..1
0..1
sourceD
targetC
sourceB
BPMNDataObject
name : EString
gmf.node
BPMNModel
title : EString
gmf.diagram
BPMNPool
name : EString
gmf.node
associations
BPMNLane
name : EString
gmf.node
lanes
targetD
targetA
sourceC
0..1
0..1
0..1
BPMNSequenceFlow
label : EString
gmf.link
sequenceFlows
0..1
0..1
source
target
0..1
0..*
BPMNMessageElement
label : EString
nodes
0..*
BPMNNode
messageFlows
0..*
0..1
BPMNGateway
condition : EString
gmf.node
BPMNIntermediateEvent
gmf.node
BPMNEvent
BPMNInitialEvent
gmf.node
BPMNMessageFlow
label : EString
gmf.link
BPMNFinalEvent
gmf.node

APÊNDICE C

Arquivos XMI dos exemplos

Processo ***Managed Indemnity Insurance*** **– Modelo BPMN**

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:BPMNModel="bpmnmodel">
  <BPMNModel:BPMNModel>
    <pools name="People">
      <lanes name="Patient">
        <nodes xsi:type="BPMNModel:BPMNTask" name="GetTreated"/>
        <nodes xsi:type="BPMNModel:BPMNInitialEvent" label=""/>
        <nodes xsi:type="BPMNModel:BPMNTask" name="Take (Medication)"/>
        <nodes xsi:type="BPMNModel:BPMNFinalEvent" label="Treated (Sickness)"/>
        <sequenceFlows source="/0/@pools.0/@lanes.0/@nodes.1"
target="/0/@pools.0/@lanes.0/@nodes.2"/>
        <sequenceFlows source="/0/@pools.0/@lanes.0/@nodes.2"
target="/0/@pools.0/@lanes.0/@nodes.0"/>
        <sequenceFlows source="/0/@pools.0/@lanes.0/@nodes.0"
target="/0/@pools.0/@lanes.0/@nodes.3"/>
      </lanes>
      <lanes name="Physician">
        <nodes xsi:type="BPMNModel:BPMNInitialEvent" label="Treat Patient"/>
        <nodes xsi:type="BPMNModel:BPMNTask" name="Diagnose Sickness"/>
        <nodes xsi:type="BPMNModel:BPMNTask" name="Treat Sickness"/>
        <nodes xsi:type="BPMNModel:BPMNTask" name="BillInsurCo"/>
        <nodes xsi:type="BPMNModel:BPMNFinalEvent" label="PatientBeCured"/>
        <nodes xsi:type="BPMNModel:BPMNDataObject" label="Fee (Treatment)"/>
        <sequenceFlows label="" source="/0/@pools.0/@lanes.1/@nodes.0"
target="/0/@pools.0/@lanes.1/@nodes.1"/>
        <sequenceFlows label="" source="/0/@pools.0/@lanes.1/@nodes.1"
target="/0/@pools.0/@lanes.1/@nodes.2"/>
        <sequenceFlows label="" source="/0/@pools.0/@lanes.1/@nodes.2"
target="/0/@pools.0/@lanes.1/@nodes.3"/>
        <sequenceFlows source="/0/@pools.0/@lanes.1/@nodes.3"
target="/0/@pools.0/@lanes.1/@nodes.4"/>
      </lanes>
    </pools>
    <pools name="Companies">
      <lanes name="InsuranceCompany">
        <nodes xsi:type="BPMNModel:BPMNTask" name="ApproveTreatment"/>
        <nodes xsi:type="BPMNModel:BPMNTask" name="ReimburseTreatment"/>
        <nodes xsi:type="BPMNModel:BPMNDataObject" label="PreApproval
(Treatment)"/>
        <nodes xsi:type="BPMNModel:BPMNInitialEvent"/>
        <nodes xsi:type="BPMNModel:BPMNFinalEvent"/>
        <sequenceFlows label="" source="/0/@pools.1/@lanes.0/@nodes.3"
target="/0/@pools.1/@lanes.0/@nodes.0"/>
        <sequenceFlows label="" source="/0/@pools.1/@lanes.0/@nodes.0"
target="/0/@pools.1/@lanes.0/@nodes.1"/>
        <sequenceFlows source="/0/@pools.1/@lanes.0/@nodes.1"
target="/0/@pools.1/@lanes.0/@nodes.4"/>
```

```
      </lanes>
    </pools>
    <messageFlows source="/0/@pools.0/@lanes.1/@nodes.2"
target="/0/@pools.0/@lanes.0/@nodes.2"/>
    <messageFlows source="/0/@pools.0/@lanes.1/@nodes.2"
target="/0/@pools.1/@lanes.0/@nodes.0"/>
    <messageFlows source="/0/@pools.1/@lanes.0/@nodes.0"
target="/0/@pools.0/@lanes.1/@nodes.2"/>
    <messageFlows source="/0/@pools.1/@lanes.0/@nodes.1"
target="/0/@pools.0/@lanes.1/@nodes.3"/>
    <messageFlows source="/0/@pools.0/@lanes.1/@nodes.3"
target="/0/@pools.1/@lanes.0/@nodes.1"/>
    <associations sourceA="/0/@pools.1/@lanes.0/@nodes.0"
targetA="/0/@pools.1/@lanes.0/@nodes.2"/>
    <associations sourceA="/0/@pools.0/@lanes.1/@nodes.3"
targetA="/0/@pools.0/@lanes.1/@nodes.5"/>
  </BPMNModel:BPMNModel>
```

Processo ***Managed Indemnty Insurance*** **– Modelo I***

```
<?xml version="1.0" encoding="UTF-8"?>
<IstarModel:IstarModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IstarModel="IstarModel" title="ViablePractice">
  <dependencyLinks source="//@compartments.0/@elements.0" target="//@elements.1"/>
  <dependencyLinks source="//@elements.1" target="//@compartments.2/@elements.0"/>
  <dependencyLinks source="//@compartments.2/@elements.2" target="//@elements.0"/>
  <dependencyLinks source="//@elements.0" target="//@compartments.0/@elements.0"/>
  <dependencyLinks source="//@compartments.0/@elements.2" target="//@elements.2"/>
  <dependencyLinks source="//@elements.2" target="//@compartments.1/@elements.3"/>
  <dependencyLinks source="//@compartments.1/@elements.3" target="//@elements.3"/>
  <dependencyLinks source="//@elements.3" target="//@compartments.0/@elements.2"/>
  <dependencyLinks source="//@compartments.2/@elements.2" target="//@elements.4"/>
  <dependencyLinks source="//@elements.4" target="//@compartments.1/@elements.0"/>
  <dependencyLinks source="//@compartments.1/@elements.1" target="//@elements.5"/>
  <dependencyLinks source="//@elements.5" target="//@compartments.2/@elements.3"/>
  <elements name="Take (Medication)" type="TASK"/>
  <elements name="Treated (Sickness)"/>
  <elements name="Covered (Sickness)"/>
  <elements name="PremiumPayment" type="RESOURCE"/>
  <elements name="PreApproval (Treatment)" type="RESOURCE"/>
  <elements name="Fee (Treatment)" type="RESOURCE"/>
  <compartments name="Patient">
    <meanEndLinks source="//@compartments.0/@elements.1"
target="//@compartments.0/@elements.3"/>
    <tasksDecompositions source="//@compartments.0/@elements.1"
target="//@compartments.0/@elements.0"/>
    <tasksDecompositions source="//@compartments.0/@elements.1"
target="//@compartments.0/@elements.2"/>
    <elements name="GetTreated" type="TASK"/>
    <elements name="GetWellByHavingInsurance" type="TASK"/>
    <elements name="BuyInsurance" type="TASK"/>
    <elements name="BeWell"/>
  </compartments>
  <compartments name="InsuranceCompany">
    <meanEndLinks source="//@compartments.1/@elements.4"
target="//@compartments.1/@elements.5"/>
```

```
    <tasksDecompositions source="//@compartments.1/@elements.2"
target="//@compartments.1/@elements.0"/>
    <tasksDecompositions source="//@compartments.1/@elements.2"
target="//@compartments.1/@elements.1"/>
    <tasksDecompositions source="//@compartments.1/@elements.4"
target="//@compartments.1/@elements.3"/>
    <tasksDecompositions source="//@compartments.1/@elements.4"
target="//@compartments.1/@elements.2"/>
    <contributionLinks type="SOMEPLUS" source="//@compartments.1/@elements.0"
target="//@compartments.1/@elements.9"/>
    <contributionLinks type="SOMEPLUS" source="//@compartments.1/@elements.6"
target="//@compartments.1/@elements.8"/>
    <contributionLinks type="SOMEMINUS" source="//@compartments.1/@elements.0"
target="//@compartments.1/@elements.6"/>
    <contributionLinks type="SOMEPLUS" source="//@compartments.1/@elements.8"
target="//@compartments.1/@elements.7"/>
    <contributionLinks type="SOMEPLUS" source="//@compartments.1/@elements.9"
target="//@compartments.1/@elements.7"/>
    <elements name="ApproveTreatment" type="TASK"/>
    <elements name="ReimburseTreatment" type="TASK"/>
    <elements name="ProcessClaim" type="TASK"/>
    <elements name="SellPolicy" type="TASK"/>
    <elements name="RunMgdIndemnityInsuranceBusiness" type="TASK"/>
    <elements name="RunHealthInsuranceBusiness" type="TASK"/>
    <elements name="Fast" type="SOFTGOAL"/>
    <elements name="Profitable" type="SOFTGOAL"/>
    <elements name="Low Admin Costs" type="SOFTGOAL"/>
    <elements name="Controlled Medical Costs" type="SOFTGOAL"/>
  </compartments>
  <compartments name="Physician">
    <meanEndLinks source="//@compartments.2/@elements.0"
target="//@compartments.2/@elements.4"/>
    <tasksDecompositions source="//@compartments.2/@elements.4"
target="//@compartments.2/@elements.6"/>
    <tasksDecompositions source="//@compartments.2/@elements.6"
target="//@compartments.2/@elements.5"/>
    <tasksDecompositions source="//@compartments.2/@elements.0"
target="//@compartments.2/@elements.1"/>
    <tasksDecompositions source="//@compartments.2/@elements.0"
target="//@compartments.2/@elements.2"/>
    <tasksDecompositions source="//@compartments.2/@elements.0"
target="//@compartments.2/@elements.3"/>
    <elements name="Treat Patient" type="TASK"/>
    <elements name="Diagnose Sickness" type="TASK"/>
    <elements name="Treat Sickness" type="TASK"/>
    <elements name="BillInsurCo" type="TASK"/>
    <elements name="PatientBeCured"/>
    <elements name="ViablePractice" type="SOFTGOAL"/>
    <elements name="RunMedicalPractice" type="TASK"/>
  </compartments>
</IstarModel:IstarModel>
```