UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INTEGRAÇÃO DA FERRAMENTA ISTAR2BPMN COM OPENOME E BIZAGI

FELIPE DE MACÊDO RODRIGUES

TRABALHO DE GRADUAÇÃO

RECIFE
JULHO DE 2015
UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

FELIPE DE MACÊDO RODRIGUES

INTEGRAÇÃO DA FERRAMENTA ISTAR2BPMN COM OPENOME E BIZAGI

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientadora: Profª. Carla Silva

RECIFE
JULHO DE 2015

INTEGRAÇÃO DA FERRAMENTA ISTAR2BPMN COM OPENOME E BIZAGI

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 22 de julho de 2015.

Banca examinadora:

_____

Profª. Carla Silva
Orientadora
UFPE

_____

Prof. Jaelson Castro
UFPE

_____

Prof. Robson Fidalgo
UFPE

AGRADECIMENTOS

A Deus pela saúde e oportunidade concedida.
A meus familiares e amigos cujo suporte tornou possível a conclusão deste curso.
A minha namorada pelo apoio e motivação em todos os momentos.
A Profª. Carla Silva por eliminar todas as dificuldades.
Aos professores Jaelson Castro e Róbson Fidalgo pelo tempo dedicado.
A todos os colegas de curso.

“Torna-te aquilo que és.”.

Friedrich Nietzsche

RESUMO

Para que uma empresa possa ser adaptável às constantes mudanças exigidas pelo mercado, ela precisa ter uma modelagem simples e facilmente compreensível de suas entidades. A notação BPMN (Business Process Modeling Notation) tem como objetivo principal a representação de processos de negócios de modo a ser facilmente compreendida por todos [5]. A modelagem organizacional surge como uma necessidade para a compreensão do ambiente empresarial, ajudando a entender as complexas interações entre as organizações e as pessoas. [6] Nesse contexto surge a modelagem orientada a objetivos, abordagem que prioriza as metas que os usuários esperam do sistema, e que tem crescido como uma forma promissora de descrever sistemas que realmente satisfaçam os desejos dos stakeholders. Ela fornece uma maneira de identificar e especificar tanto os objetivos dos stakeholders com relação ao sistema pretendido, como as características do próprio sistema. [7] O framework i* é uma abordagem de Engenharia de Requisitos Orientada a Objetivos (GORE - Goal-Oriented Requirements Engineering) formulada para representar, modelar e analisar objetivos organizacionais [8]. Durante o ciclo de vida do modelo de processo de negócios, quando mudanças ocorrem em um modelo, estas devem se propagar a outros modelos empresariais de modo a manter a consistência. Combinações entre notações complementares (como i* e BPMN) oferecem um maior suporte conceitual a mudanças. Refletir alterações no contexto organizacional a mudanças no design de processos de negócio significa alinhar processos de negócios com objetivos organizacionais. Do mesmo modo, melhorias operacionais podem ser mapeadas de volta a objetivos organizacionais facilitando análise e garantindo a inexistência de conflitos com objetivos já existentes [10]. O iStar2BPMN [11] é uma ferramenta desenvolvida com o intuito de transformar modelos i* em modelos BPMN, e vice-versa. Ela inclui editores gráficos para i* e para o BPMN, e possibilita a transformação entre estes modelos a partir de diretrizes de mapeamento bem definidas [12]. No entanto, a ferramenta não permite importação e exportação de modelos construídos com outras ferramentas populares que dão suporte a modelagem i* e BPMN, como o OpenOME [14] e o Bizagi [18], respectivamente. O objetivo principal do presente trabalho é viabilizar o reconhecimento de diagramas, criados com a ferramenta Istar2Bpmn, por ferramentas populares como o OpenOME e o Bizagi e vice-versa.

Palavras-Chave: BPMN, Framework i*, Processos de negócios, XML, Java.

ABSTRACT

For a company to be adaptable to the constant changes required by the market, it must have a simple and easily understandable modeling of its entities. The BPMN notation (Business Process Modeling Notation) has as main objective the mode of representation of business processes to be easily understood by all. [5] The organizational modeling is a necessity for understanding the business environment, helping to understand the complex interactions between organizations and people. [6] In this context arises oriented modeling purposes, an approach that prioritizes the goals that users expect the system, and it has grown as a promising way to describe systems that actually fulfill the wishes of stakeholders. It provides a way to identify and specify both the objectives of the stakeholders with respect to the intended system, such as the characteristics of the system itself. [7] i * framework is an approach from GORE (Goal-Oriented Requirements Engineering) formulated to represent, model and analyze organizational objectives [8]. During the lifecycle of the business process model when changes occur in a model, they should spread to other business models in order to maintain consistency. Combinations of complementary notations (as i * and BPMN) offer a higher conceptual support to changes. To reflect changes in the organizational context to changes in business process design means aligning business processes with organizational goals. Similarly, operational improvements can be mapped back to organizational objectives facilitating analysis and assuring no conflict with existing targets [10]. The iStar2BPMN [11] is a tool developed for the purpose of turning i* models BPMN models, and vice versa. It includes graphical editors for i* and BPMN, and enables the transformation between these models from mapping guidelines clearly defined [12]. However, the tool does not allow import and export models built with other popular tools that support modeling i* and BPMN like OpenOME [14] and the Bizagi [18], respectively. The main objective of this work is to enable the recognition of diagrams created with the Istar2Bpmn tool for popular tools like OpenOME and Bizagi and vice versa.

Keywords: BPMN, i* Framework, business processes, XML, Java.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

O presente capítulo divide-se em quatro partes. Na primeira parte é exposto o contexto ao qual pertence este trabalho. Na segunda é explicada a motivação para o mesmo, seguida pelos objetivos do presente trabalho na terceira parte. Finalmente, a quarta e última parte contém a explicação de como o documento foi estruturado.

1.1. Contexto

De acordo com De Souza, um processo de negócio consiste em *"um conjunto de atividades estruturadas, executadas de forma sequencial ou paralela que adicionam valor aos seus insumos produzindo um resultado de valor para os seus clientes, sejam eles internos ou externos* [1]*."*

Todo trabalho importante realizado nas empresas faz parte de algum processo. Não existe um produto ou um serviço oferecido por uma empresa sem um processo empresarial [2]. O que torna os modelos de negócios complexos é a quantidade de diferentes necessidades dos usuários e estas mudarem constantemente [3].

Para que uma empresa possa ser adaptável às constantes mudanças exigidas pelo mercado, ela precisa ter uma modelagem simples e facilmente compreensível de suas entidades. Várias são as técnicas, metodologias e notações existentes para a modelagem dos processos de uma empresa [4].

A notação BPMN (Business Process Modeling Notation) tem dominado o cenário mundial da modelagem de processos e apresenta dentre outras vantagens o fato de ser uma notação de padrão aberto disponibilizada pela OMG (Object Management Group). Ela tem como objetivo principal a representação de processos de negócio de modo a ser facilmente compreendida por todos [5].

Cada organização apresenta missão, objetivos e processos próprios. A modelagem organizacional surge como uma necessidade para a compreensão do ambiente empresarial, ajudando a entender as complexas interações entre as organizações e as pessoas. [6] Nesse contexto surge a modelagem orientada a objetivos, abordagem que prioriza as metas que os usuários esperam do sistema, e que tem crescido como uma forma promissora de descrever sistemas que realmente satisfaçam os desejos dos stakeholders. Ela fornece uma maneira de identificar e especificar tanto os objetivos dos stakeholders com relação ao sistema pretendido, como as características do próprio sistema. [7]

O framework i* é uma abordagem de Engenharia de Requisitos Orientada a Objetivos (GORE - Goal-Oriented Requirements Engineering) formulada para representar, modelar e analisar objetivos organizacionais [8]. Ela tem sido aplicada na modelagem de organizações, processos de negócios e requisitos de sistemas. É uma abordagem centrada nos stakeholders do sistema e nas dependências entre eles [9].

1.2. Motivação

Durante o ciclo de vida do modelo de processo de negócios, quando mudanças ocorrem em um modelo, estas devem se propagar a outros modelos empresariais de modo a manter a consistência. Combinações entre notações complementares (como i* e BPMN) oferecem um maior suporte conceitual a mudanças. Refletir alterações no contexto organizacional a mudanças no design de processos de negócio significa alinhar processos de negócios com objetivos organizacionais. Do mesmo modo, melhorias operacionais podem ser mapeadas de volta a objetivos organizacionais facilitando análise e garantindo a inexistência de conflitos com objetivos já existentes [10]. O iStar2BPMN [11] é uma ferramenta desenvolvida com o intuito de transformar modelos i* em modelos BPMN, e vice-versa. Ela inclui editores gráficos para i* e para o BPMN, e possibilita a transformação entre estes modelos a partir de diretrizes de mapeamento bem definidas [12]. No entanto, a ferramenta não permite importação e exportação de modelos construídos com outras ferramentas populares que dão suporte a modelagem i* e BPMN, como o OpenOME [14] e o Bizagi [18], respectivamente.

1.3. Objetivos do trabalho

O objetivo principal do presente trabalho é implementar uma forma de viabilizar o reconhecimento de diagramas, criados com a ferramenta Istar2Bpmn, por ferramentas populares como o OpenOME e o Bizagi e vice-versa. O Istar2Bpmn é uma ferramenta criada [11] a partir de diretrizes de mapeamento bem definidas [12]. O mesmo é capaz de transformar modelos i* em modelos BPMN e vice-versa.

1.4. Estrutura do documento

O presente trabalho divide-se em 5 capítulos. O primeiro é este capítulo introdutório que menciona os objetivos do trabalho.

No segundo capítulo encontra-se a fundamentação teórica necessária à compreensão do documento, as heurísticas de mapeamento entre BPMN e i* e uma introdução à ferramenta iStar2BPMN.

No terceiro capítulo é explicada a classe Java ManipulaXml que serve como base para todos os procedimentos de conversões de formatos descritos nos apêndices. Ainda nos apêndices é possível encontrar o mapeamento entre código XML [19] e elementos de diagramas para as três ferramentas: OpenOME [14], Bizagi [18] e iStar2BPMN.

No capítulos 4 é mostrado um exemplo de aplicação, com o passo a passo de conversões desde o OpenOME até o Bizagi, passando pelo iStar2BPMN e em seguida o caminho inverso de volta ao OpenOME.

Por fim, o capítulo 5 contém considerações finais e expectativas para o futuro deste trabalho.

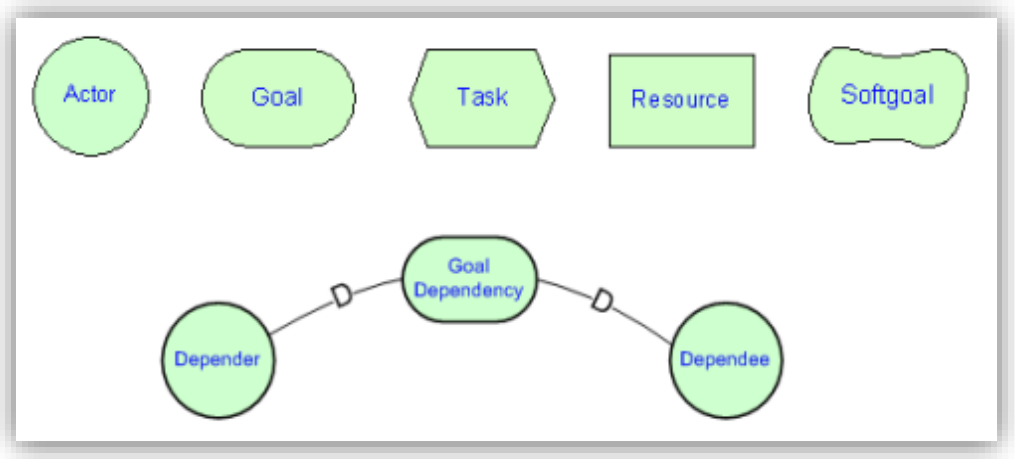
2. Fundamentação teórica

Neste capítulo será apresentado o modelo i* e seus elementos principais, seguido por um exemplo de aplicação do i*: o processo Managed Indemnity Insurance. Depois serão apresentadas as interfaces gráficas das ferramentas iStar2BPMN e OpenOME contendo a modelagem do Managed Indemnity Insurance. Em seguida será apresentada a notação BPMN com seus principais elementos e as interfaces gráficas do iStar2BPMN e do Bizagi contendo o Managed Indemnity Insurance modelado em BPMN. Ainda neste capítulo serão apresentadas as heurísticas de mapeamento, definidas no trabalho de Alves, do modelo i* para o modelo BPMN e vice-versa [12]. Por último será introduzida a ferramenta iStar2BPMN resultante do trabalho de Melo [11].

2.1. Modelagem i* no iStar2BPMN e no OpenOME

Um modelo i* é formado por componentes como atores, objetivos, softgoals, tarefas, recursos e dependências [13], todos ilustrados na Figura 2.1.

Figura 2.1 – Elementos do i*



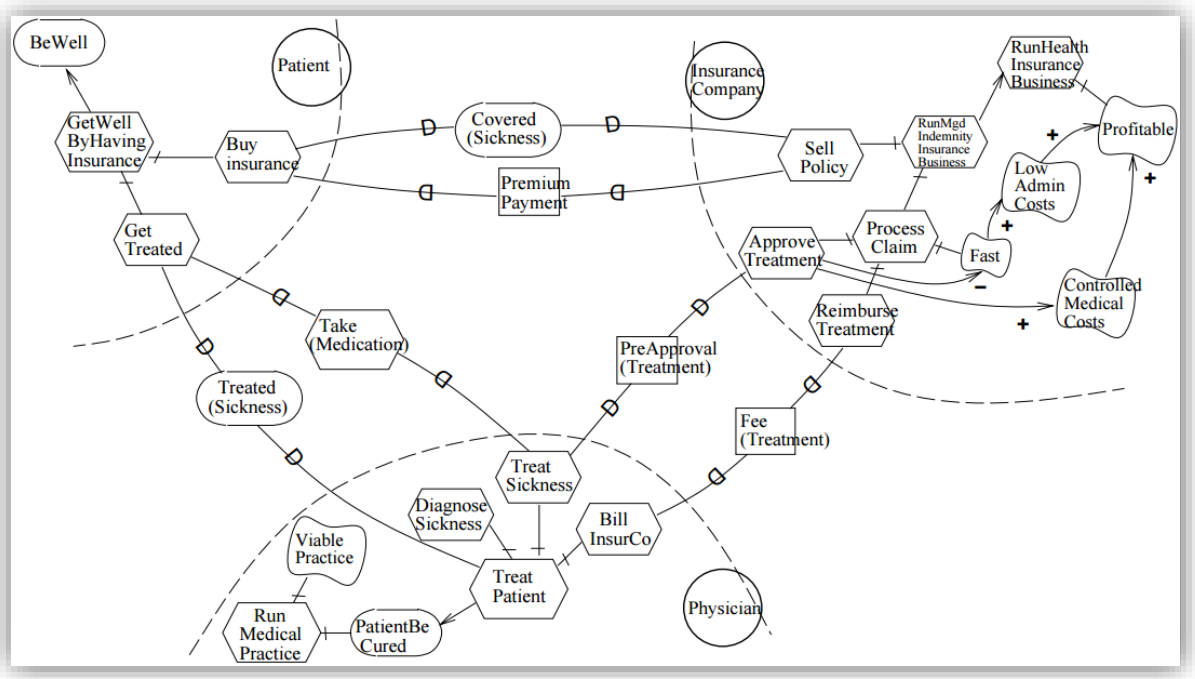
Fonte: ALVES, 2013, p. 19.

O i* possui dois modelos:

- O Modelo de dependência Estratégica (Strategic Dependency Model - SD), descreve as relações de dependência entre os atores;
- O Modelo de Razão Estratégica (Strategic Rationale Model - SR), em que os compartimentos dos atores são extendidos para que sejam mostrados os seus elementos internos;

Como exemplo de aplicação do i*, temos o processo Managed Indemnity Insurance [13], modelagem para gerenciamento de seguros de saúde, ilustrado na Figura 2.2.

Figura 2.2 - Modelo SR do processo de gerenciamento de seguros de saúde



Fonte: YU, 2011, p. 42.

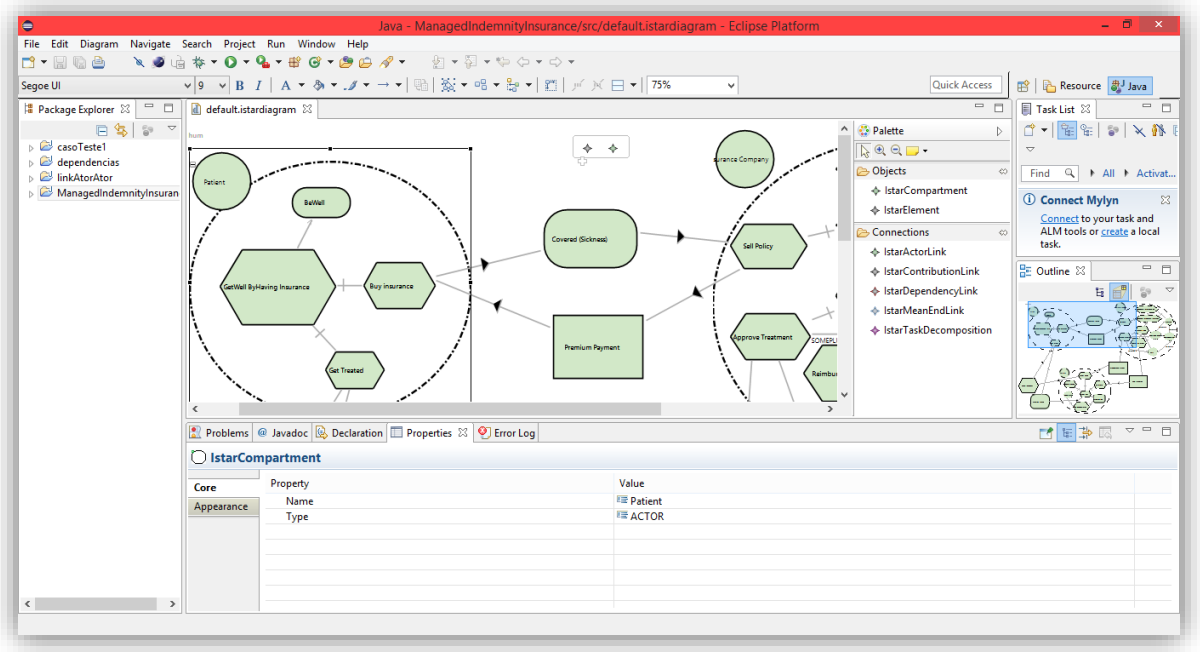
Nesse processo, pacientes (Patient), médicos (Physician) e companhias de seguros (Insurance Company) dependem uns dos outros para realizar tarefas e alcançar objetivos. O objetivo principal do paciente é estar bem (BeWell). Para alcançar esse objetivo, o mesmo precisa comprar um seguro de saúde (Buy 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 tem como objetivo geral a cura do paciente (PatientBeCured). Para isso ele deve tratar o paciente (Treat Patient), 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.

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.

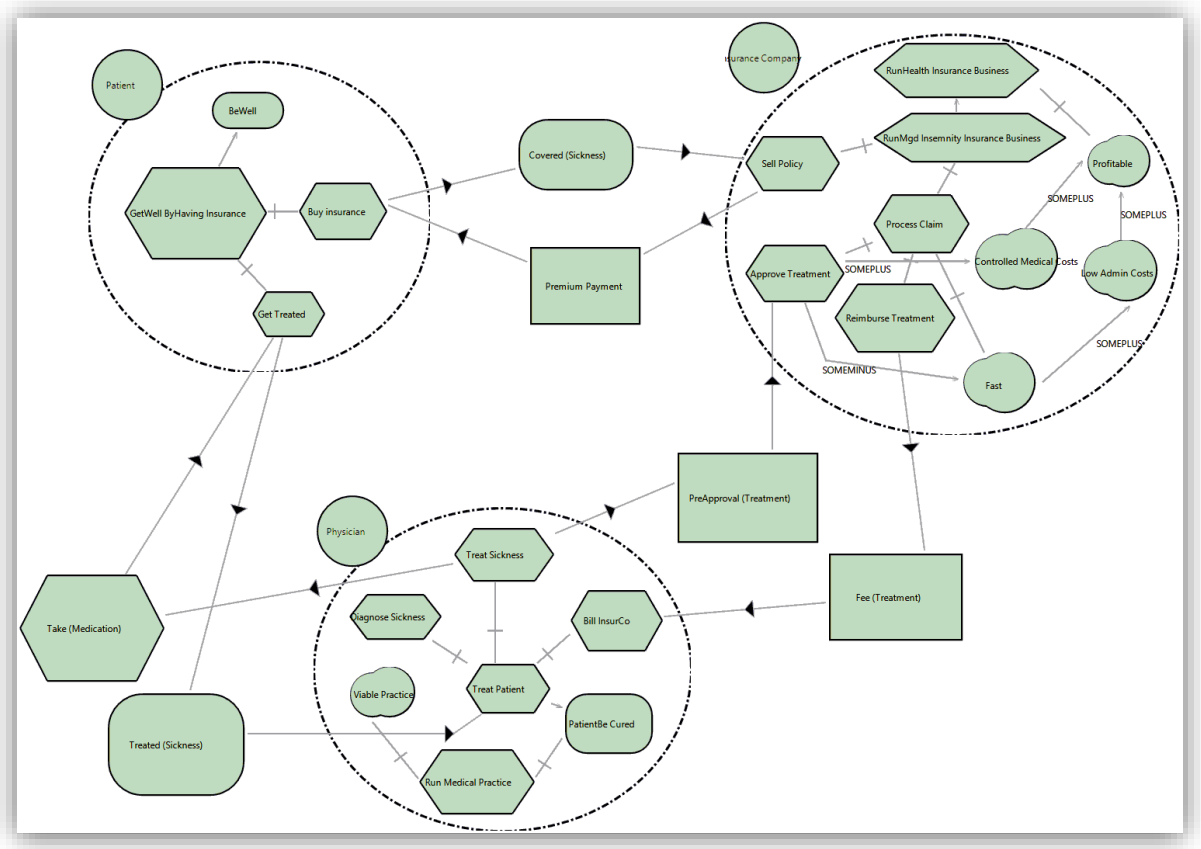
O iStar2BPMN [11] dispõe de um editor gráfico para i*. O mesmo está representado na Figura 2.3. Já na Figura 2.4 temos o processo Managed Indemnity Insurance completamente modelado na ferramenta iStar2BPMN.

Figura 2.3 - Editor gráfico i* do iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

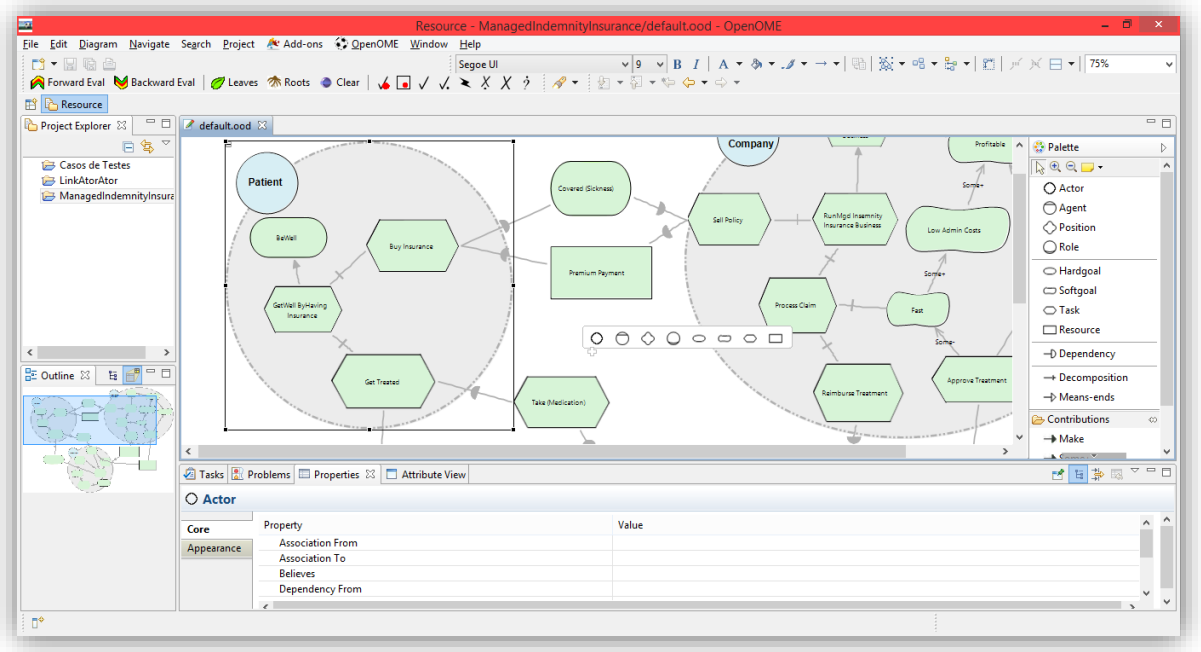
Figura 2.4 – O processo Managed Indemnity Insurance construído no iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Diversas ferramentas foram criadas para a construção e análise de modelos i*. Uma das mais populares é o OpenOME [14], ferramenta open-source baseada na IDE Eclipse e na linguagem Java. A interface gráfica do OpenOME (Figura 2.5) é bastante amigável e disponibiliza um verificador de sintaxe preparado para corrigir vários tipos de erros comuns.

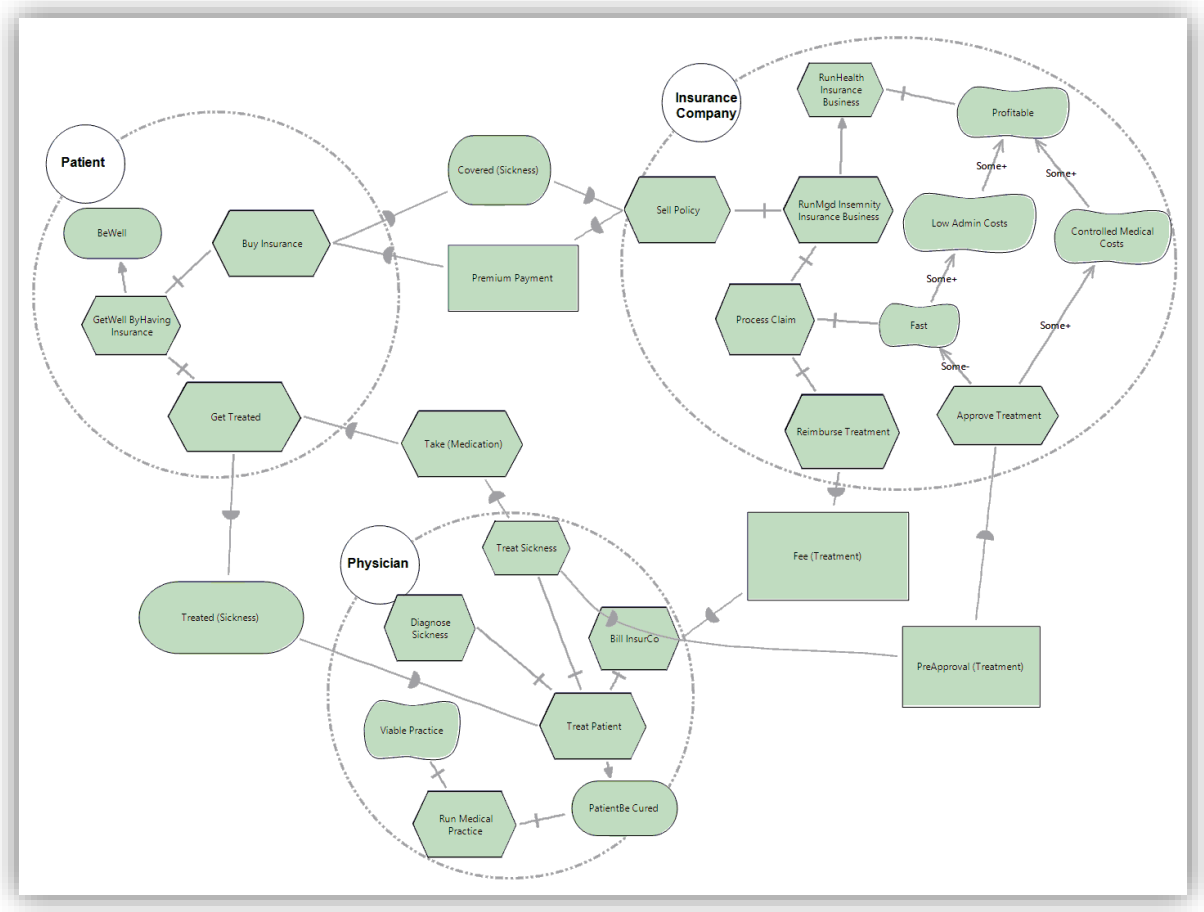
Figura 2.5 – Interface gráfica do OpenOME



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Como o iStar2BPMN não dispõe de importação de modelos criados fora da ferramenta, uma empresa que deseja examinar o impacto de suas alterações no contexto organizacional sobre os processos de negócios, precisaria reconstruir manualmente todos os seus modelos i* dentro do editor gráfico do iStar2BPMN, para então poder fazer a conversão de i* para BPMN, tarefa inviável por demanda excessiva de tempo e retrabalho. Na Figura 2.6 temos o processo Managed Indemnity Insurance construído no OpenOME.

Figura 2.6 – O processo Managed Indemnity Insurance construído no OpenOME

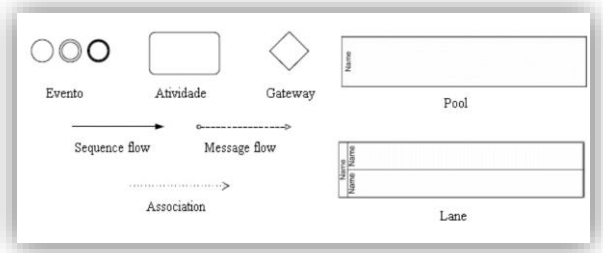


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

2.2. Modelagem BPMN no iStar2BPMN e no Bizagi

Um diagrama construído com a notação BPMN, ou seja um BPD (Business Process Diagram) é composto por quatro grupos de elementos [15]: objetos de fluxo (evento, atividade, gateway), objetos de conexão (fluxo de sequência, fluxo de mensagem, associação), swimlanes (pool, lane) e artefatos (objetos de dados, grupos e anotações), todos representados na Figura 2.7.

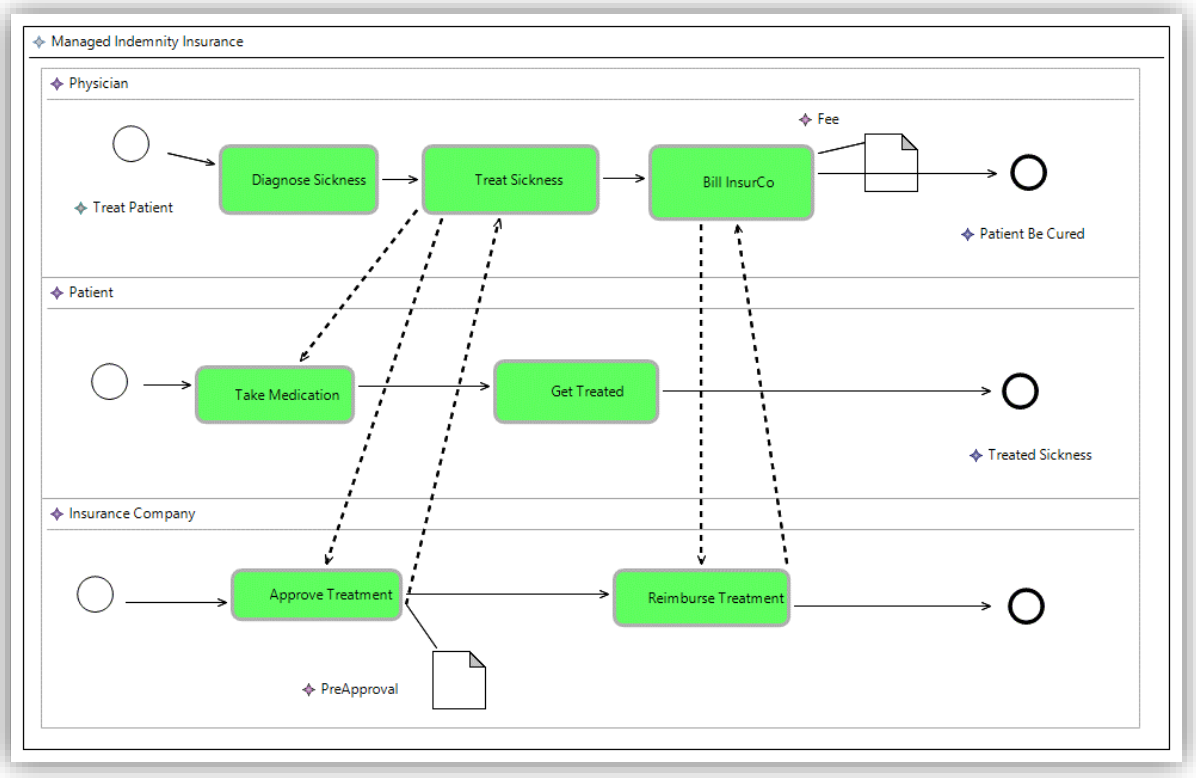
Figura 2.7 – Elementos do BPMN



Fonte: ALVES, 2013, p. 25.

Para modelar um processo de negócio em BPMN, primeiro é preciso identificar os eventos iniciais do processo, os processos que serão realizados e os resultados finais do fluxo. Em seguida, com o uso de desvios (gateways), as decisões e ramificações do fluxo são modeladas [16]. A Figura 2.8 representa um modelo BPMN do Managed Indemnity Insurance construído no iStar2BPMN via conversão a partir do diagrama i* do mesmo.

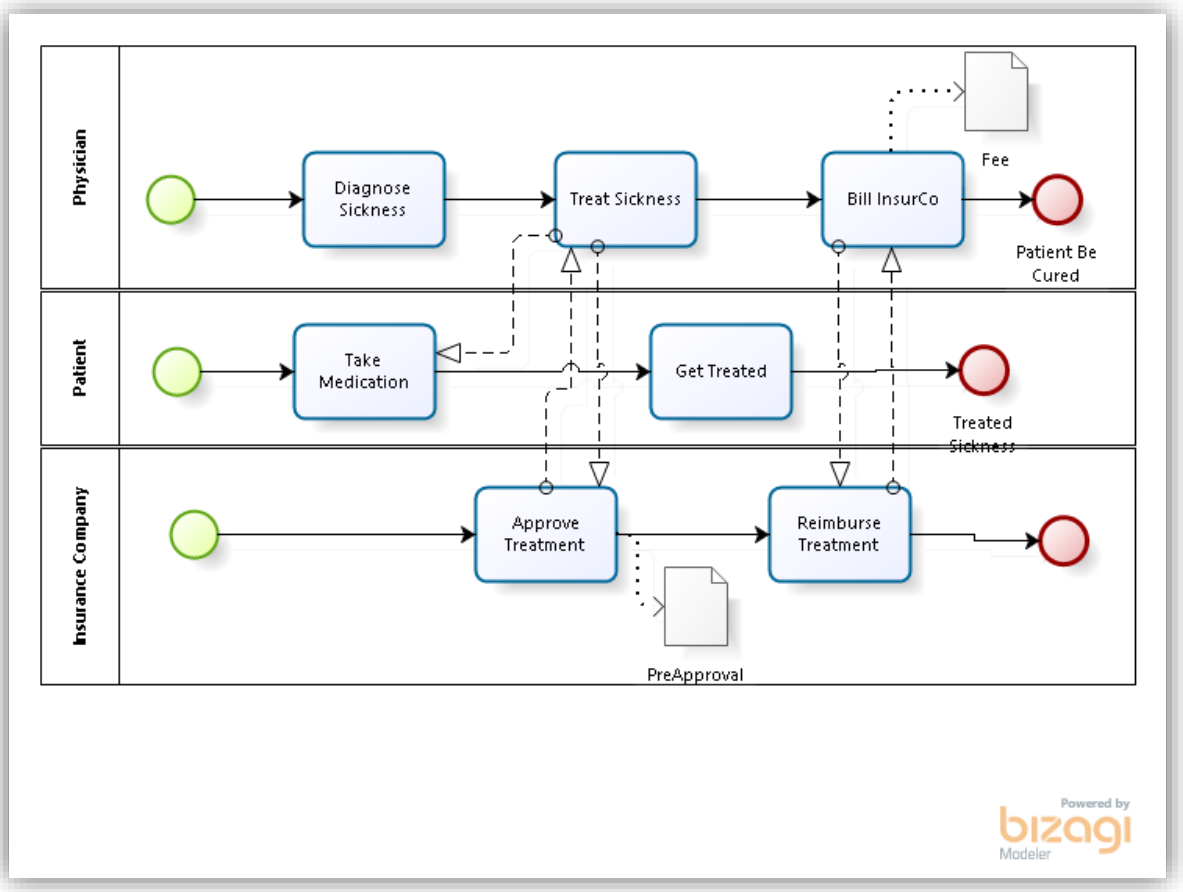
Figura 2.8 – O Managed Indemnity Insurance construído no iStar2BPMN em BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O Bizagi Process Modeler [18] é uma ferramenta que permite desenhar, documentar e compartilhar processos de trabalho usando a notação BPMN (Business Process Management Notation). Na Figura 2.9 temos o Managed Indemnity Insurance modelado em notação BPMN na ferramenta Bizagi.

Figura 2.9 – O Managed Indemnity Insurance construído no Bizagi



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Como o Bizagi é uma ferramenta muito difundida tanto no meio acadêmico quanto empresarial, é esperado que venha a surgir a necessidade de exportação do modelo construído/gerado no iStar2BPMN para o Bizagi, funcionalidade ainda não disponibilizada pela ferramenta.

A ausência das funcionalidades de importação e exportação de diagramas da ferramenta iStar2BPMN dificulta o reuso de modelos construídos na mesma, ou fora dela, e é um empecilho ao crescimento da ferramenta.

2.3. 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 [12] é uma melhoria do método de transformação proposto por Koliadis e outros [10], 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 [12] 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 [12], foram baseadas no método proposto por Koliadis e outros [10]. 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 [12]. A seguir é apresentado o conjunto de heurísticas de mapeamento definido por Alves [12] em seu trabalho.

2.3.1. Heurísticas de mapeamento do modelo i* para o modelo BPMN

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 [12] e Koliadis e outros [10] baseiam-se nos conceitos de rotina e escopo de um processo. Segundo Yu [13], 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 identifica-se 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 sentidos 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.

2.3.2. 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.

2.4. A ferramenta iStar2BPMN

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. [11]

Em seu trabalho, Melo [11] traduziu as heurísticas de mapeamento explicadas no tópico anterior para regras de transformação na linguagem ETL e automatizou a transformação entre modelos i* e BPMN.

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.

Os arquivos com terminação diagram são os que possibilitam a visualização da representação gráfica do modelo i* ou BPMN nos editores gráficos.

A ferramenta iStar2BPMN, proposta por Melo [11] em seu trabalho, 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. Ela torna mais sistemático o método de conversão bidirecional entre diagramas i* e BPMN, reduzindo a inferência humana durante o mapeamento dos modelos e, assim, possibilitando a geração automática de um modelo a partir de outro.

Como as heurísticas de mapeamento não são totalmente independentes da experiência do analista [12], a ferramenta transforma modelos de uma notação gráfica para outra de forma interativa, requisitando ao usuário informações necessárias para o processo de transformação.

3. Tecnologia

As seguintes atividades foram realizadas:

- Estudo da ferramenta Istar2Bpmn e da tecnologia empregada na construção da mesma. Esse estudo proporcionou um melhor entendimento de como funciona a ferramenta e resultou na descoberta de soluções para os problemas que surgiram durante o desenvolvimento deste trabalho.
- Listagem e estudo dos formatos de arquivos disponíveis para importação e exportação das ferramentas Istar2Bpmn, OpenOME e Bizagi. Durante análise dos formatos de arquivos disponíveis descobriu-se ao menos um formato que poderia ser utilizado para execução deste trabalho. Os demais formatos foram então ignorados ou descartados por inviabilidade.
- Implementação de procedimentos de conversões entre formatos de arquivos existentes. Durante a implementação foi criada a classe Java ManipulaXml, crucial para todos os procedimentos de conversão e que será explicada mais adiante. Para tornar o Istar2Bpmn compatível com as ferramentas OpenOME e Bizagi foi preciso:
  - Permitir que um diagrama i* gerado no OpenOME seja aberto pelo Istar2Bpmn;
  - Permitir que um diagrama i* gerado no Istar2Bpmn seja importado para o OpenOME;
  - Permitir que um diagrama BPMN gerado no Bizagi seja aberto pelo Istar2Bpmn;
  - Permitir que um diagrama BPMN gerado no Istar2Bpmn seja importado para o Bizagi;
- Testes e correções. Durante os testes foram identificados e corrigidos pequenos problemas com a implementação.

3.1. Conceitos iniciais referentes à tecnologia utilizada

A Extended Markup Language (XML) é uma linguagem de marcação recomendada pela W3C (World Wide Web Consortium) para criação de documentos com dados hierarquizados. Ela é extensível porque permite definir os elementos de marcação. [19]

A DOM (Document Object Model) é uma API (Application Programming Interface) para validação de documentos XML bem formados. Com ela é possível construir documentos XML e navegar suas estruturas modificando, adicionando ou removendo elementos e conteúdo.

Alguns conceitos simples serão utilizados extensivamente no presente trabalho, como:

1. Tag ou nó: elemento textual contido por “<” e “>” que pode incluir atributos, texto ou tags filhas. Na API DOM é representado por um objeto da classe org.w3c.dom.Element ou org.w3c.dom.Node.
2. Atributo: pertence sempre a alguma tag e está associado a algum valor contido entre aspas duplas (“). Na API DOM é representado por um objeto da classe org.w3c.dom.Attr.
3. Nó raíz, é a primeira ocorrência no documento XML de alguma tag. Durante o desenvolvimento deste trabalho ela se mostrou como aquela tag que contém todas as demais do documento como tag filhas.
4. Tag filha: nó pertencente a outro nó mais externo hierarquicamente.

5. Tag pai: nó que contém ao menos um nó mais interno hierarquicamente.

6. Tag filha direta: nó interno a outro nó, porém no primeiro nível da hierarquia de filhos do nó pai ao qual pertence.

7. Tag irmã: nó cujo nó pai é o mesmo de outro nó.

Como uma exigência da W3C, um importante objetivo para a DOM é prover uma interface de programação padrão que possa ser utilizada em uma variedade enorme de ambientes e aplicações. A DOM foi desenvolvida para ser utilizada com qualquer linguagem de programação.

A biblioteca javax.xml.parsers provê classes para a manipulação de documentos XML através da linguagem Java. Esta biblioteca foi utilizada extensivamente no presente trabalho para a criação da classe ManipulaXml explicada a seguir.

3.2. A classe java ManipulaXml

Para que o presente trabalho se tornasse possível foi desenvolvida uma classe Java capaz de realizar as seguintes ações:

- Carregar um arquivo XML bem formado para a memória do computador. (void carregaArquivoXml(String arquivo)).
- Armazenar referência rápida para o nó raíz do arquivo XML (void setNoRaiz(Element noRaiz)).
- Criar uma nova tag ou nó (Node newNode(String nome)).
- Dado um nó específico e o nome de seu atributo, retornar o valor do mesmo se existente (String getValue(Node node, String atributo)).
- Dado um nome de tag e o nome de seu atributo, buscar a primeira ocorrência de tag com o nome fornecido e retornar o valor do atributo procurado (String getValue(String tag, String atributo)).
- Capturar a lista de todos os nós do documento XML (NodeList listAllNodes()). Na API DOM isto é representado por um elemento da classe org.w3c.dom.NodeList ou org.w3c.dom.NodeSet.
- Capturar a lista de todos os nós do documento XML que apresentam um determinado nome (NodeList listAllNodes(String tag)).
- Capturar a lista de todos os nós do documento XML que apresentam um determinado nome e que possuem um certo atributo com um valor específico (NodeList listAllNodes(String tag, String atributo, String valor)).
- Capturar a partir de uma lista de nós aqueles que apresentam um certo atributo com um determinado valor (NodeSet listAllNodes(NodeSet nodeList, String atributo, String valor)).
- Obter a lista de filhos de um determinado nó (NodeSet listAllChildren(Node currentNode)).
- Obter da lista de filhos de um determinado nó, aqueles que apresentam um determinado nome (NodeSet listAllChildren(Node currentNode, String nome)).

- Obter a lista de filhos diretos de um determinado nó (NodeSet listAllChildrenDireto(Node currentNode)).
- Obter da lista de filhos diretos de um determinado nó, aqueles que apresentam um determinado nome (NodeSet listAllChildrenDireto(Node currentNode, String nome)).
- Obter a lista de nós irmãos de um determinado nó (NodeSet listAllBrothers(Node currentNode)).
- Obter a lista de nós irmãos de um determinado nó, cujo nome coincide com o fornecido (NodeSet listAllBrothers(Node currentNode, String nome)).
- Obter a junção de duas listas de nós (NodeSet uniaoDeListas(NodeList lista1, NodeList lista2)).
- Imprimir no console uma determinada lista de nós (void print(NodeList nodeList)).
- Imprimir no console uma determinada lista de atributos fornecida (void print(NamedNodeMap attrList)).
- Remover todos os atributos de todos os nós fornecidos (void removeAllAttr(NodeList nodeList)).
- Remover todos os atributos de um determinado nó (void removeAllAttr(Node node)).
- Remover um determinado nó, repassando todos os seus filhos ao nó pai (void remove(Node node)).
- Remover todos os nós que apresentam um determinado nome, ou seja remover todas as ocorrências de uma determinada tag (void remove(String tag)).
- Dados dois nós, substituir o “antigo” pelo “novo” nó, transferindo todos os nós filhos do primeiro para o segundo (void replace(Node antigo, Node novo)).
- Renomear todas as ocorrências de uma determinada tag (void rename(String nomeAntigo, String novoNome)).
- Adicionar a um determinado nó um atributo com um certo valor (void addAttribute(Node node, String atributo, String valor)).
- Adicionar a todas as ocorrências de uma determinada tag um atributo com um certo valor (void addAttribute(String tag, String atributo, String valor)).
- Remover um determinado atributo de todas as ocorrências de uma determinada tag (void removeAttribute(String tag, String atributo)).
- Informar se um determinado nó contém um determinado atributo (boolean hasAttribute(Node tag, String atributo)).
- Remover um atributo específico de um determinado nó (void removeAttribute(Node tag, String atributo)).
- Remover todos os nós filhos de um determinado nó (void removeAllChildren(Node tag)).
- Remover todos os nós filhos de um determinado nó que apresentam um determinado nome (void removeAllChildren(Node tag, String nome)).

- Contar a quantidade de tags filhas de um determinado nó (int countAllChildren(Node tag)).
- Contar a quantidade de tags filhas de um determinado nó que apresentam um determinado nome (int countAllChildren(Node tag, String nome)).
- Contar a quantidade de tags filhas de um determinado nó que apresentam um determinado nome e que são filhas diretas do nó, ou seja hierarquicamente estão no nível mais alto dentre os filhos do nó (int countAllChildrenDireto(Node tag, String nome)).
- Escrever o documento XML num arquivo (boolean writeXmlFile(Document doc, File arquivo)).
- Escrever o documento XML num arquivo sem questionar a pré-existência do mesmo no sistema (boolean writeXmlFile(String arquivo, String extensao)).
- Escrever o documento XML num arquivo perguntando ao usuário se o arquivo já existe (boolean writeXmlFile(String arquivo, String extensao, Shell shell)).

As duas primeiras ações são realizadas automaticamente assim que uma classe qualquer herda a classe ManipulaXml fornecendo à mesma o caminho referente ao arquivo XML que será manipulado (ex.: C:\Users\Felipe\Downloads\eclipse-epsilon-1.2-win32-x86_64\ArquivoXml.bpmn). As demais serão utilizadas nos capítulos de procedimentos de conversões.

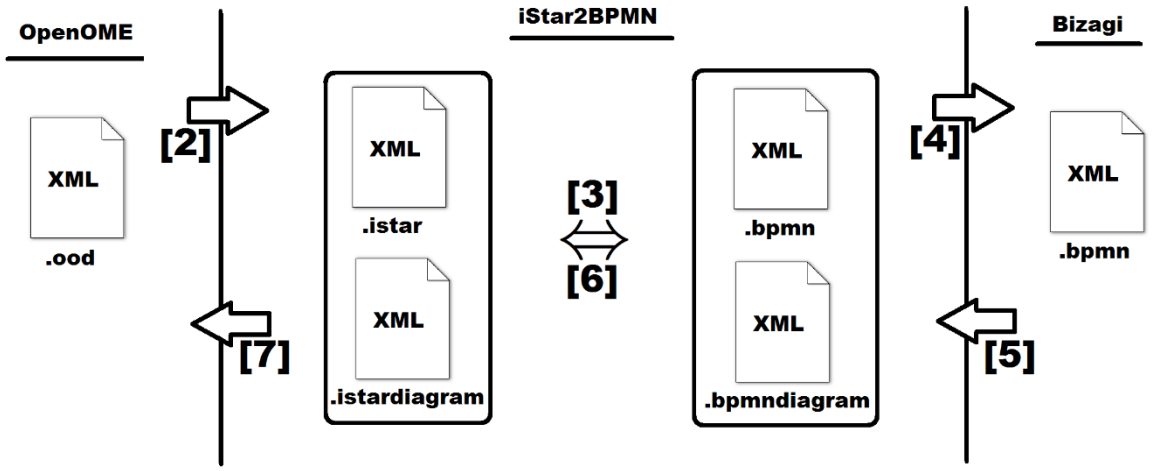
4. Exemplo de aplicação: o processo Managed Indemnity Insurance

Neste capítulo será apresentado o passo a passo da conversão de um diagrama i* construído no OpenOME que encontrará seu BPMN equivalente na ferramenta Bizagi, passando pela ferramenta iStar2BPMN que disponibiliza integração bidirecional entre os modelos i* e BPMN. Os seguintes passos serão realizados, nesta ordem:

1. Construção manual do diagrama i* referente ao processo Managed Indemnity Insurance na ferramenta OpenOME.
2. APÊNDICE E - Procedimento de conversão de um diagrama i* na ferramenta OpenOME em formato ".ood" para um diagrama i* na ferramenta iStar2BPMN.
3. Conversão do diagrama obtido no passo anterior para seu correspondente BPMN.
4. APÊNDICE G - Procedimento de conversão de um diagrama BPMN na ferramenta iStar2BPMN em formato ".bpmn" e ".bpmndiagram" para um diagrama BPMN na ferramenta Bizagi.
5. APÊNDICE H - Procedimento de conversão de um diagrama BPMN na ferramenta Bizagi em formato ".bpmn" para um diagrama BPMN na ferramenta iStar2BPMN.
6. Conversão do diagrama BPMN obtido no passo anterior para seu correspondente i*.
7. APÊNDICE F - Procedimento de conversão de um diagrama i* na ferramenta iStar2BPMN em formato ".istar" e ".istardiagram" para um diagrama i* na ferramenta OpenOME.

O procedimento completo envolve diferentes tipos de arquivos, cada um adequado a sua ferramenta, como ilustrado no esquema abaixo. Cada seta corresponde a um passo citado acima.

Figura 4.1 - Representação esquemática do processo de conversões entre diferentes formatos XML

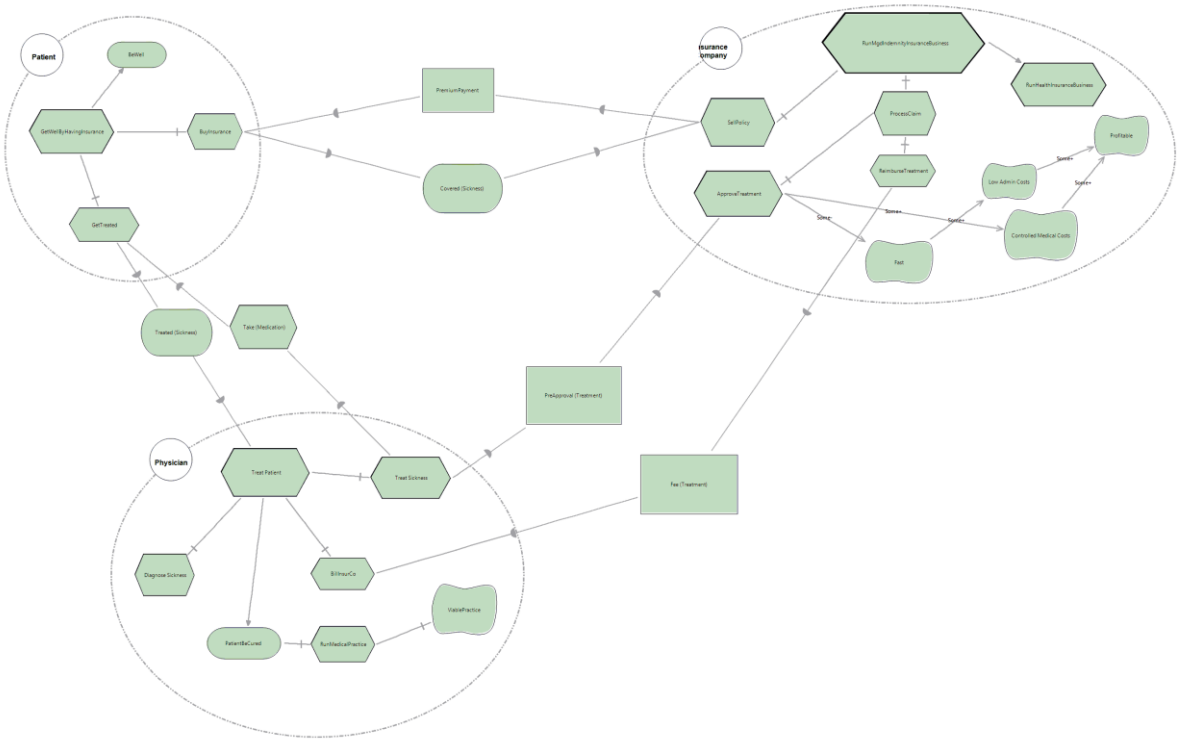


Fonte: imagem elaborada pelo autor

4.1. Passo 1 – Construção do diagrama i* no OpenOME

O primeiro passo consiste na construção de um diagrama i* na ferramenta OpenOME. Será utilizado o processo de gerenciamento de seguros de saúde (Managed Indemnity Insurance), como explicado no capítulo 2 do presente trabalho e ilutrado na Figura 2.2. Após construção do modelo utilizando-se da interface gráfica do OpenOME (Figura 2.5 – Interface gráfica do OpenOME) é obtido o diagrama correspondente ao processo Managed Indemnity Insurance. O mesmo encontra-se ilutrado abaixo (Figura 4.2).

Figura 4.2 – O processo Managed Indemnity Insurance construído no OpenOME

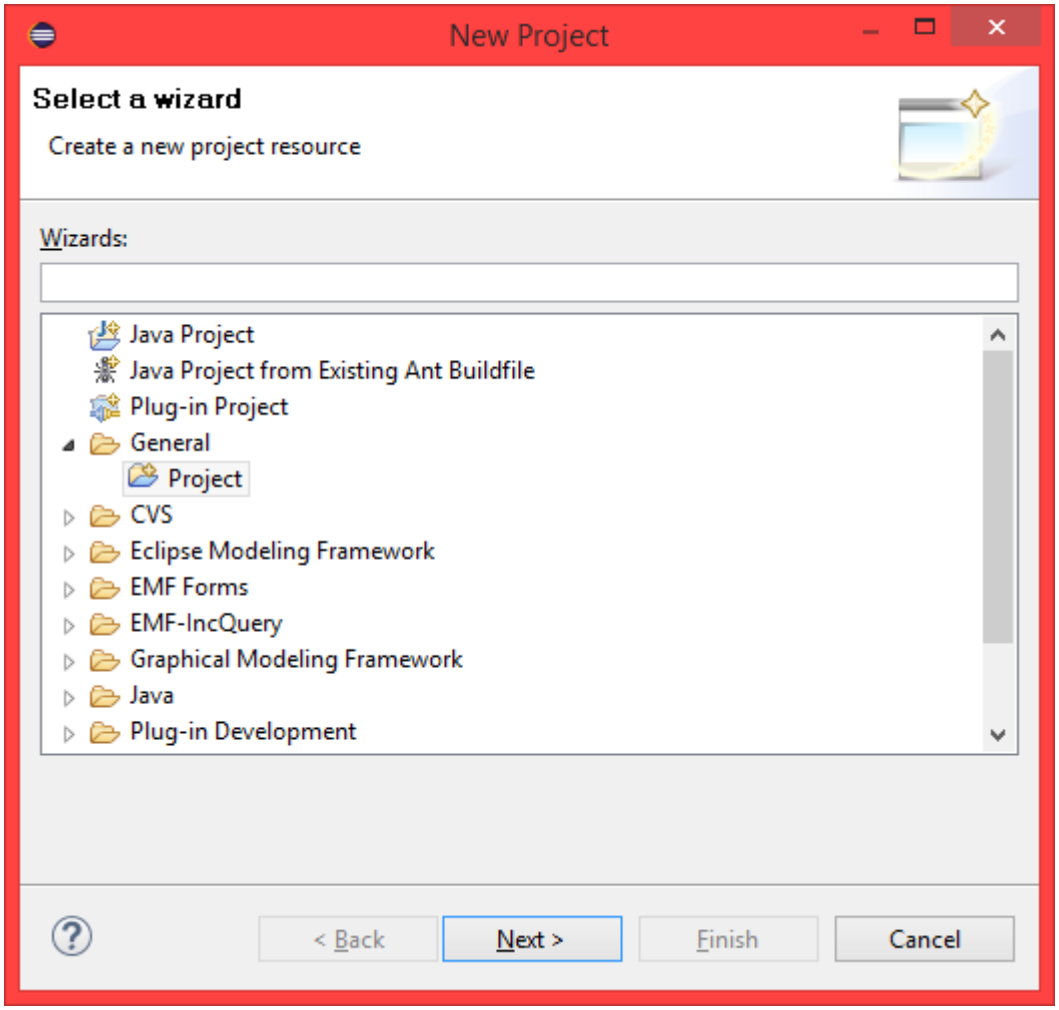


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Objetivando finalizar o primeiro passo deste capítulo, o diagrama será salvo no diretório “C:\OpenOME\win32.win32.x86_64\OpenOME\workspace\ManagedIndemnityInsurance”. Este diretório deve ser conhecido para a realização do próximo passo.

Antes de iniciar o segundo passo, é preciso que exista um projeto ao qual o diagrama importado pertencerá. Neste exemplo será criado um novo projeto. Na interface gráfica da ferramenta iStar2BPMN deve ser escolhida a opção “File” -> “New” -> “Project...”, gerando a caixa de diálogo ilustrada na Figura 4.3.

Figura 4.3 – Caixa de diálogo correspondente ao primeiro passo na criação de um novo projeto na ferramenta iStar2BPMN



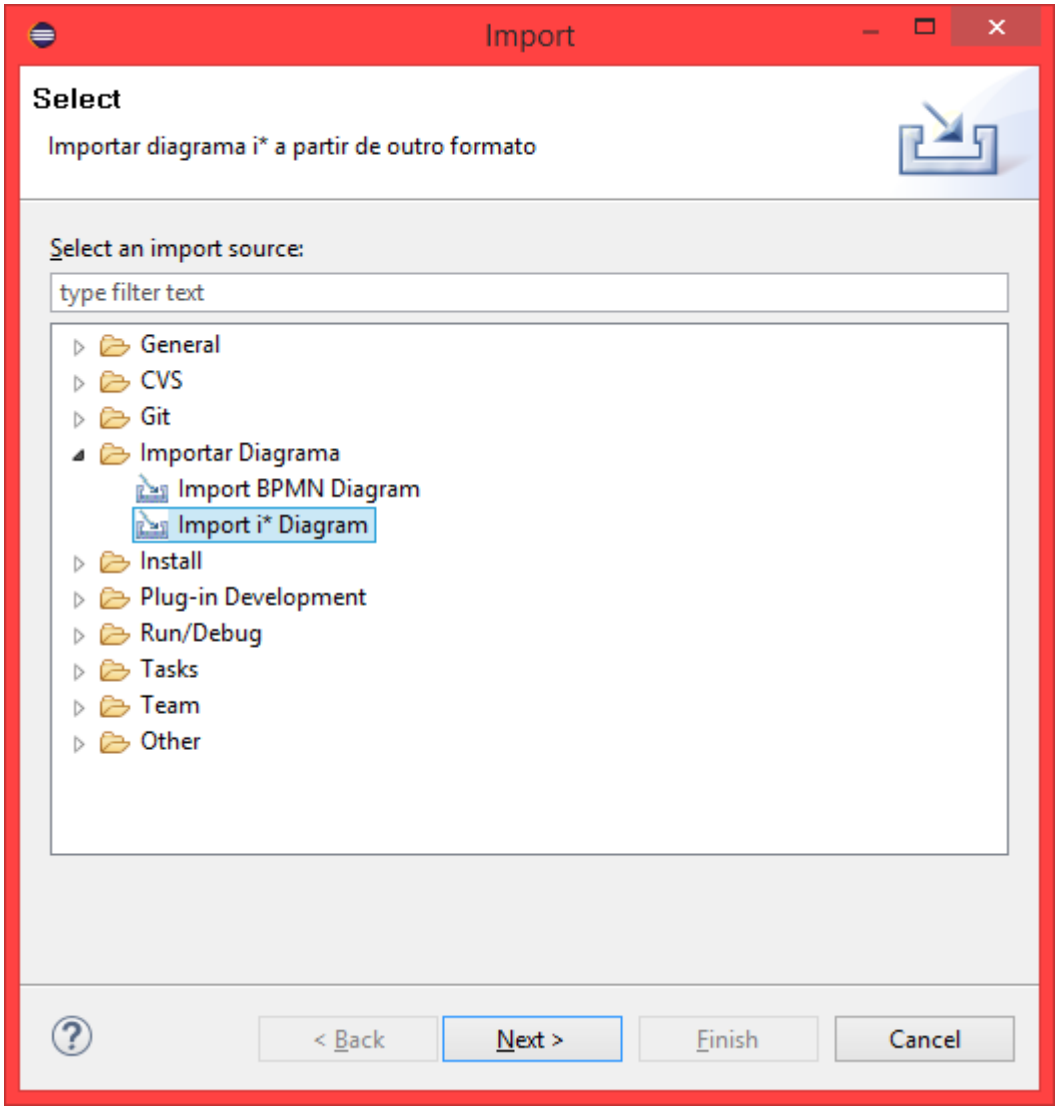
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Após selecionada a opção “Project” da pasta “General”, clica-se no botão “Next >”. Na tela seguinte deve-se inserir o nome do projeto no campo “Project name:” e clicar no botão “Finish”. Neste exemplo o nome do projeto será “ManagedIndemnityInsurance”. O diretório que irá conter os diagramas pertencentes a este novo projeto deve ser conhecido e neste exemplo será o “C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance”.

4.2. Passo 2 – Importação do diagrama i* do OpenOME para o iStar2BPMN

Como agora existe um projeto já criado na ferramenta, pode-se prosseguir com o passo dois, que consiste em importar o diagrama gerado no OpenOME (diagrama.ood) para a ferramenta iStar2BPMN. A importação se dá pela opção “File” -> “Import ...” da interface gráfica do iStar2BPMN. Na caixa de diálogo que se abre (Figura 4.4) seleciona-se a opção “Import i* Diagram” da pasta “Importar Diagrama” e clica-se em “Next >”.

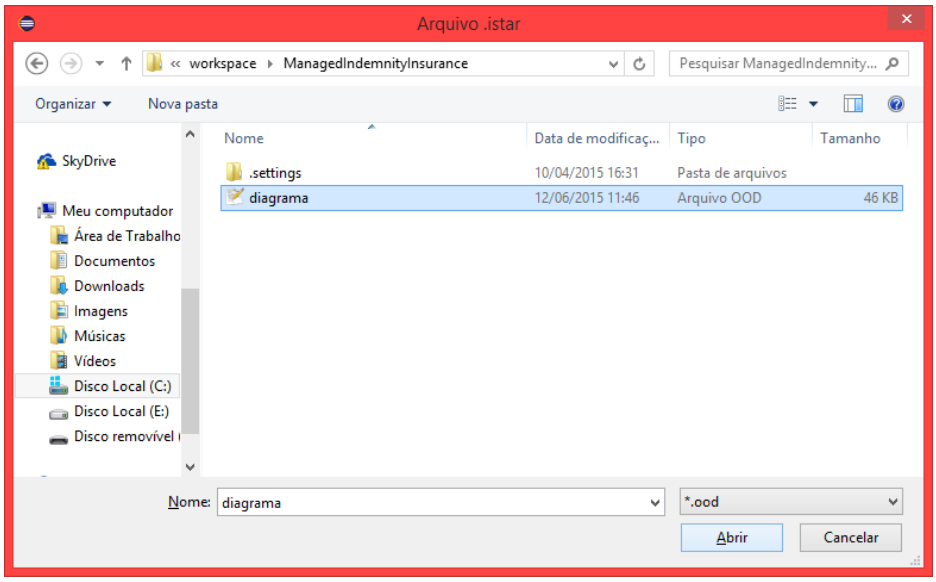
Figura 4.4 – Caixa de diálogo correspondente ao primeiro passo da importação de um diagrama i* na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Na tela seguinte seleciona-se a opção ".ood (OpenOME)" e clica-se no botão "Browse...". Em seguida é preciso navegar até o diretório que contém o diagrama a ser importado, selecionar o arquivo de extensão ood correspondente ao diagrama desejado e clicar no botão "Abrir", como ilutrado na Figura 4.5.

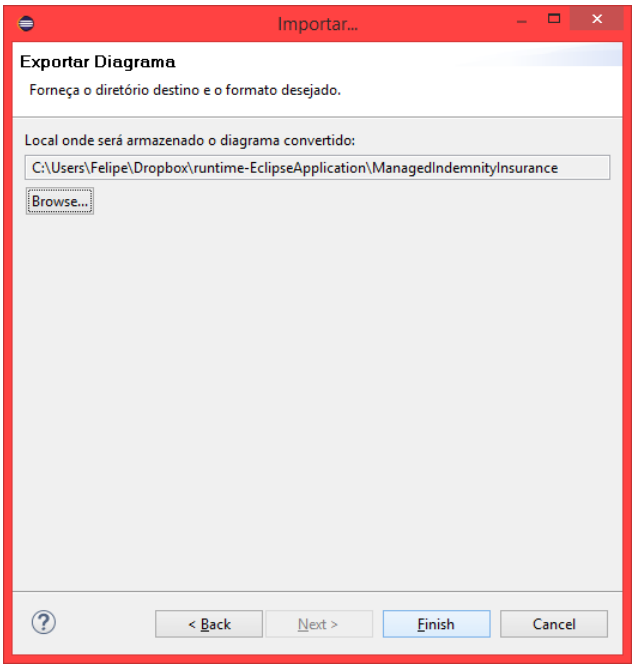
Figura 4.5 – Caixa de diálogo correspondente a seleção de um diagrama i* a ser importado na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

A seguir clica-se em “Next” e na próxima tela será escolhido o local de destino da importação, ou seja o local onde serão inseridos os arquivos XML correspondentes ao diagrama a ser importado. Após navegar até o diretório destino e obter a tela representada na Figura 4.6, clica-se no botão “Finish”.

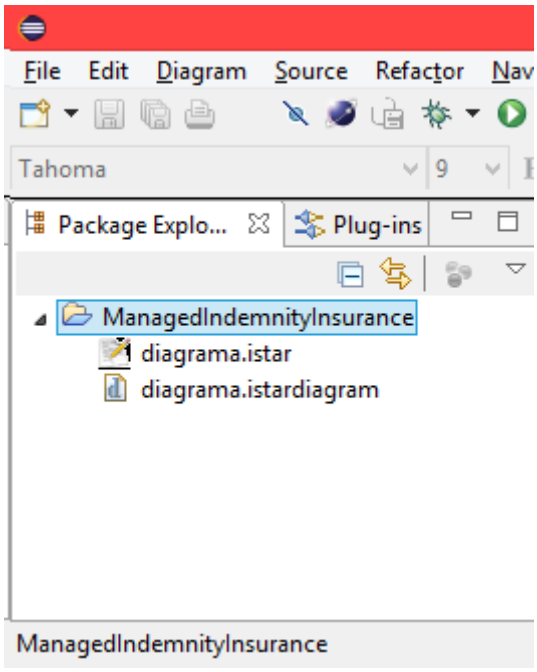
Figura 4.6 – Caixa de diálogo correspondente ao último passo na importação de um diagrama i* do OpenOME para a ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Para visualizar o resultado da importação basta ir em “File” -> “Open File...” e navegar até o arquivo de extensão istardiagram gerado após importação. Alternativamente pode-se selecionar o projeto que contém o diagrama importado na guia “Package Explorer” da interface gráfica do iStar2BPMN, como ilustra.a Figura 4.7 e pressionar F5 para atualizar e assim poder clicar duas vezes sobre o arquivo de extensão istardiagram.

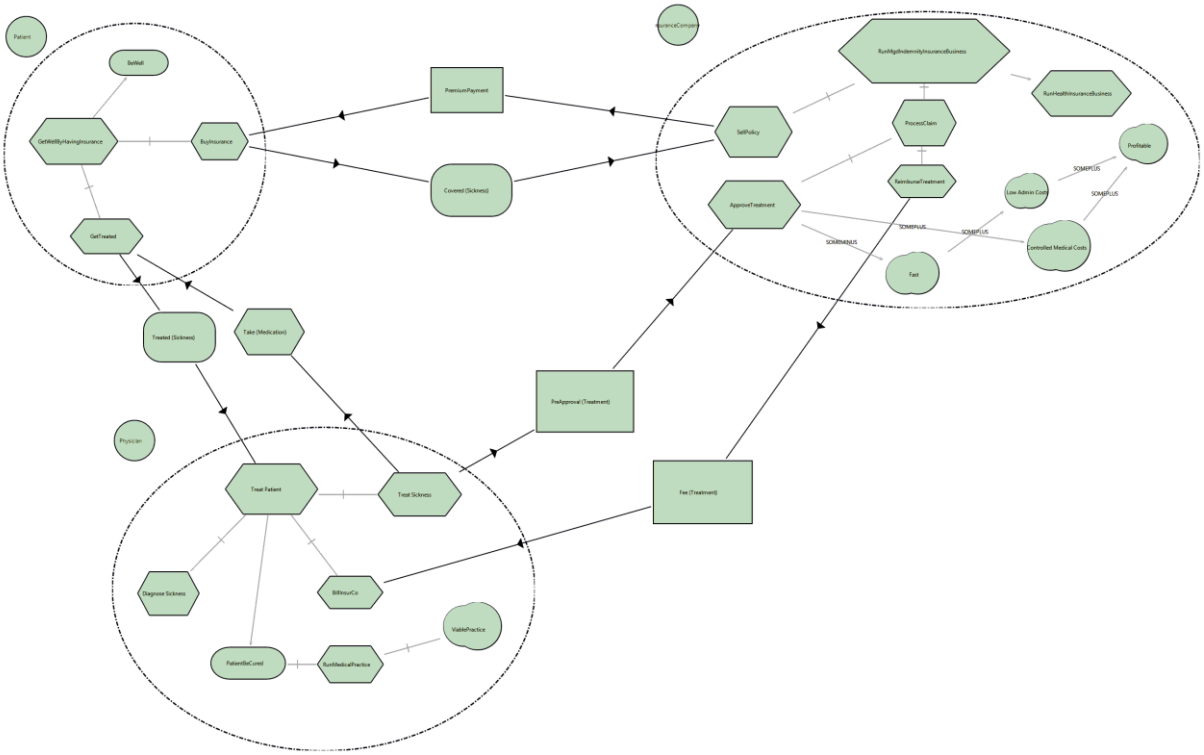
Figura 4.7 – Guia Package Explorer na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O diagrama resultante da conversão está ilutrado na Figura 4.8 abaixo. Isso conclui o passo dois.

Figura 4.8 – O processo Managed Indemnity Insurance importado do OpenOME e visualizado na interface gráfica do iStar2BPMN



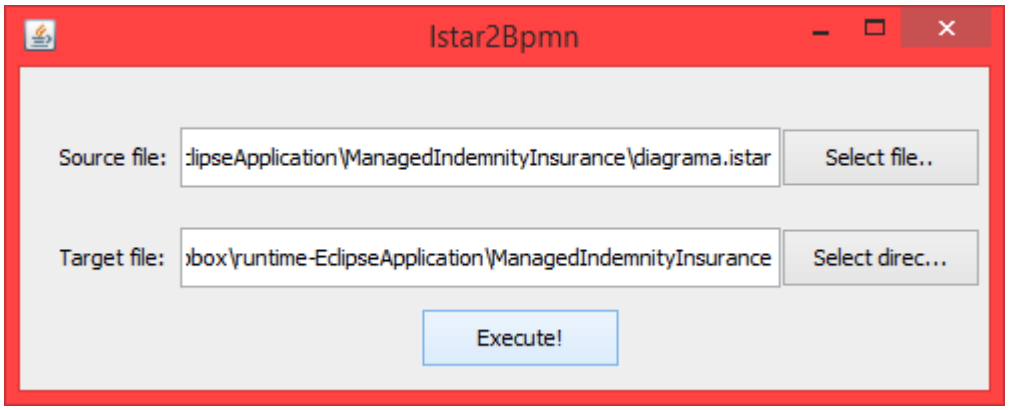
Fonte: *print screen* da aplicação iStar2BPMN no sistema operacional Windows 8.1

4.3. Passo 3 – Conversão do diagrama i* para BPMN

Os passos abaixo serão transcritos do trabalho de Melo [11] pois representam a conversão de um diagrama i* em um diagrama BPMN.

O processo de conversão entre modelos i* e BPMN é interativo. Primeiro o usuário deve escolher o arquivo de entrada e o diretório em que o arquivo gerado deve ser salvo, como ilutra a Figura 4.9.

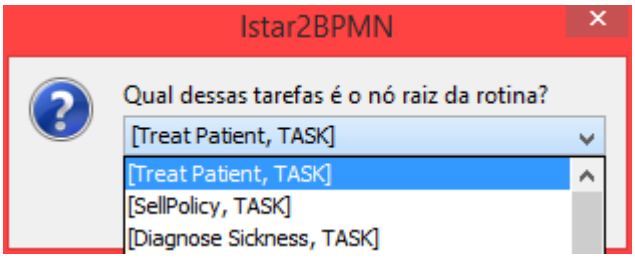
Figura 4.9 – Seleção do arquivo de entrada e do diretório destino no processo de conversão entre modelos i* e BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Em seguida deve-se escolher a 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 raíz da rotina, as suas subtarefas e as dependências ligadas a essas tarefas. A Figura 4.10 mostra a tela em que o usuário pode escolher a tarefa do modelo i* que será usada como rotina.

Figura 4.10 – Escolha da tarefa que será a rotina do processo na conversão entre modelos i* e BPMN

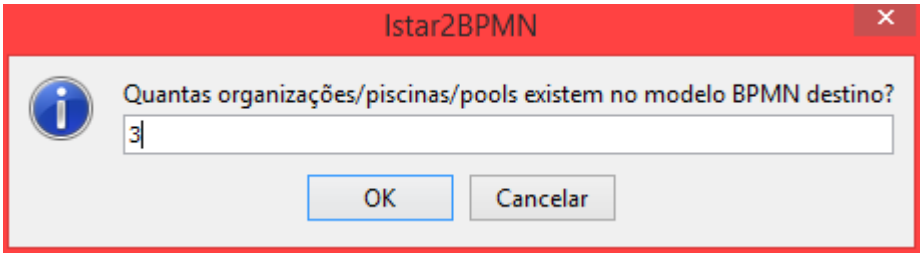


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

A seguir deve-se definir quantas organizações existem no modelo BPMN destino (Figura 4.11) e o nome de cada uma delas (Figura 4.12). Assumiremos que cada organização terá o mesmo nome do ator correspondente. Desta forma pode-se escolher a organização em que cada ator se encontra, como ilutrado na Figura 4.13.

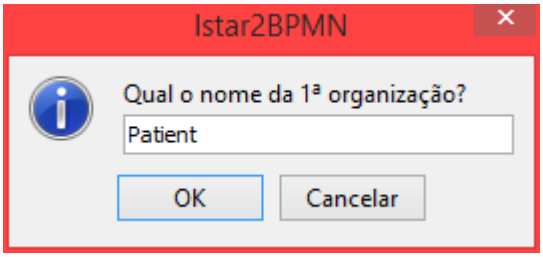
Na aplicação das heurísticas de mapeamento ao exemplo do processo Managed Indemnity Insurance, Alves [12] assume que todos os atores pertencem a organizações diferentes, e portanto cada qual está em uma piscina diferente no modelo BPMN destino (regra II no processo de transformação de i* para BPMN). 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.

Figura 4.11 – Definição de quantidade de organizações existentes no modelo no processo de conversão entre modelos i* e BPMN



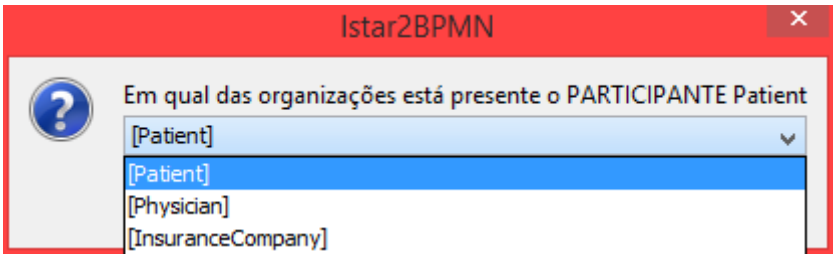
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.12 – Tela para informar o nome de cada organização/piscina no processo de conversão entre modelos i* e BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

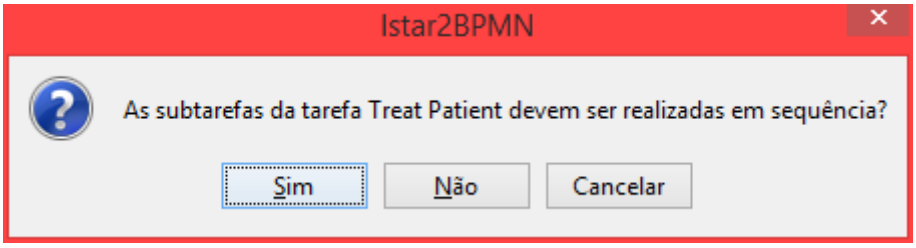
Figura 4.13 – Escolha da organização em que cada ator se encontra no processo de conversão entre modelos i* e BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

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 4.14). O usuário deve clicar em “Sim”, uma vez que no trabalho de Alves [12] o modelo BPMN destino possui essas atividades realizadas em sequência.

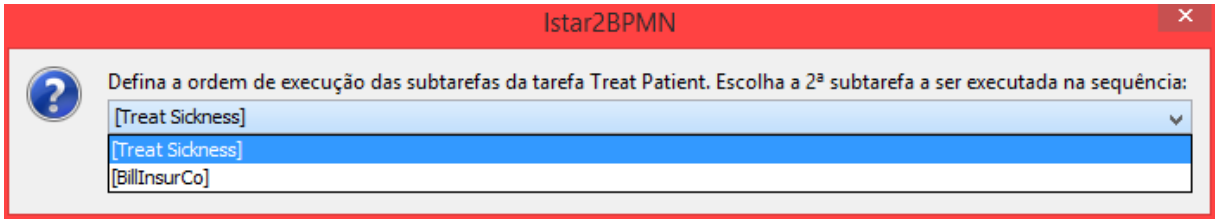
Figura 4.14 – Questionamento quanto a atividades sequenciais no processo de conversão entre modelos i* e BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Em seguida deve-se definir a ordem de execução destas 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 4.15). As atividades devem ser realizadas nesta ordem: Diagnose Sickness, Treat Sickness, BillInsurCo, conforme análise feita por Alves [12].

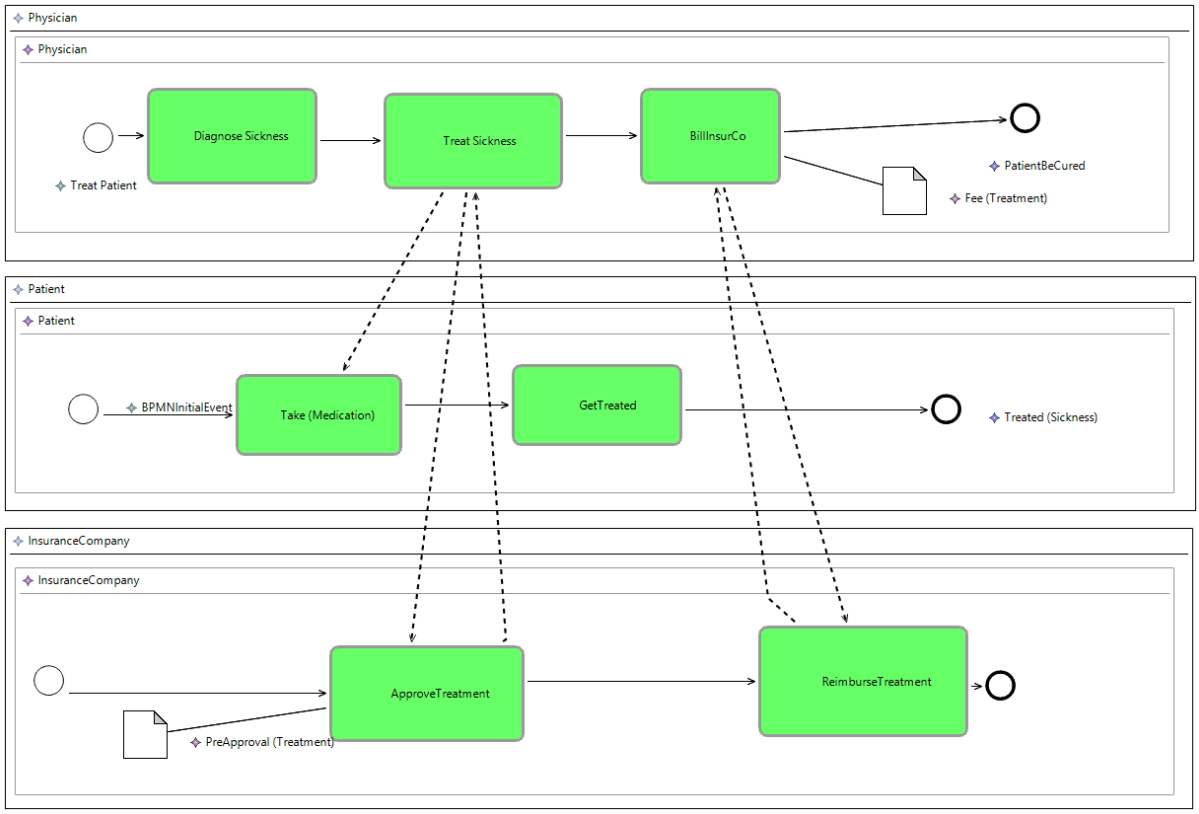
Figura 4.15 – Definição da ordem de execução de atividades no processo de conversão entre modelos i* e BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Finalmente, a Figura 4.16 ilustra o resultado do processo de conversão de um diagrama i* em um BPMN, concluindo assim o passo 3 deste capítulo. É **importante** saber que nenhuma das informações referentes a posicionamento dos elementos no diagrama, como valor de coordenada x, coordenada y, width e height será propagada ao arquivo XML a menos que o usuário interaja com os elementos redimensionando em ambos os eixos (x e y) e reposicionando cada um deles. Isso ocorre porque o XML captura essas informações durante a construção do diagrama, só que nesse caso o mesmo foi gerado automaticamente e não construído, portanto as informações não alcançam o XML e o mesmo ficará mal formado se não houver intervenção do usuário.

Figura 4.16 – Diagrama BPMN gerado pela transformação de um modelo i*



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

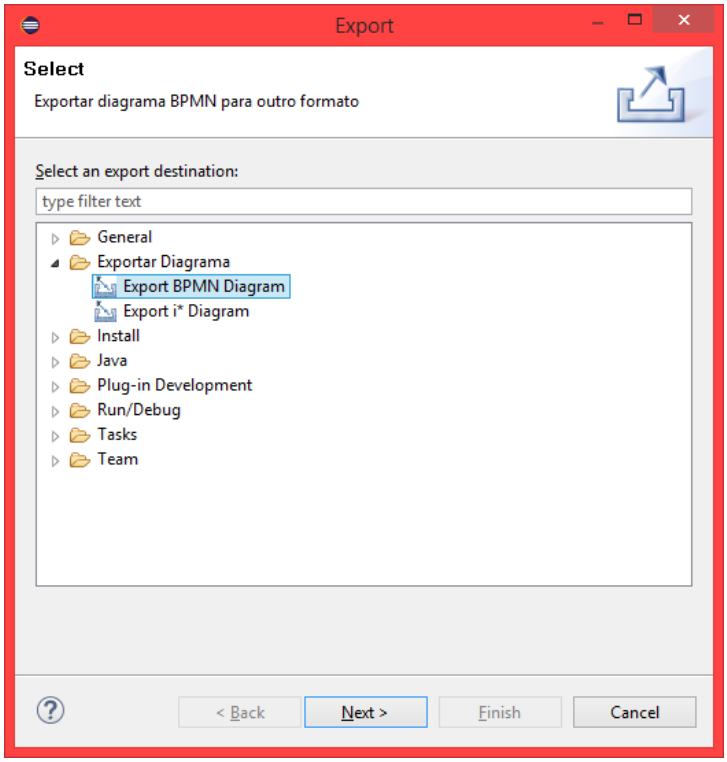
4.4. Passo 4 – Exportação do diagrama BPMN para o Bizagi

Agora o diagrama obtido ao final do passo anterior será exportado para a ferramenta Bizagi. Para isso, inicialmente será escolhida na interface gráfica da ferramenta iStar2BPMN a opção “File” -> “Export”. Na tela que surge deve-se escolher a opção “Export BPMN Diagram” presente na pasta “Exportar Diagrama” como ilutra a Figura 4.17**.**

Na próxima tela (Figura 4.18) deve-se fornecer o arquivo de extensão bpmn correspondente ao diagrama que será convertido. Neste exemplo o mesmo pôde ser encontrado na pasta “C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance\”, assim como o arquivo de extensão bpmndiagram presente no mesmo diretório e solicitado na próxima tela.

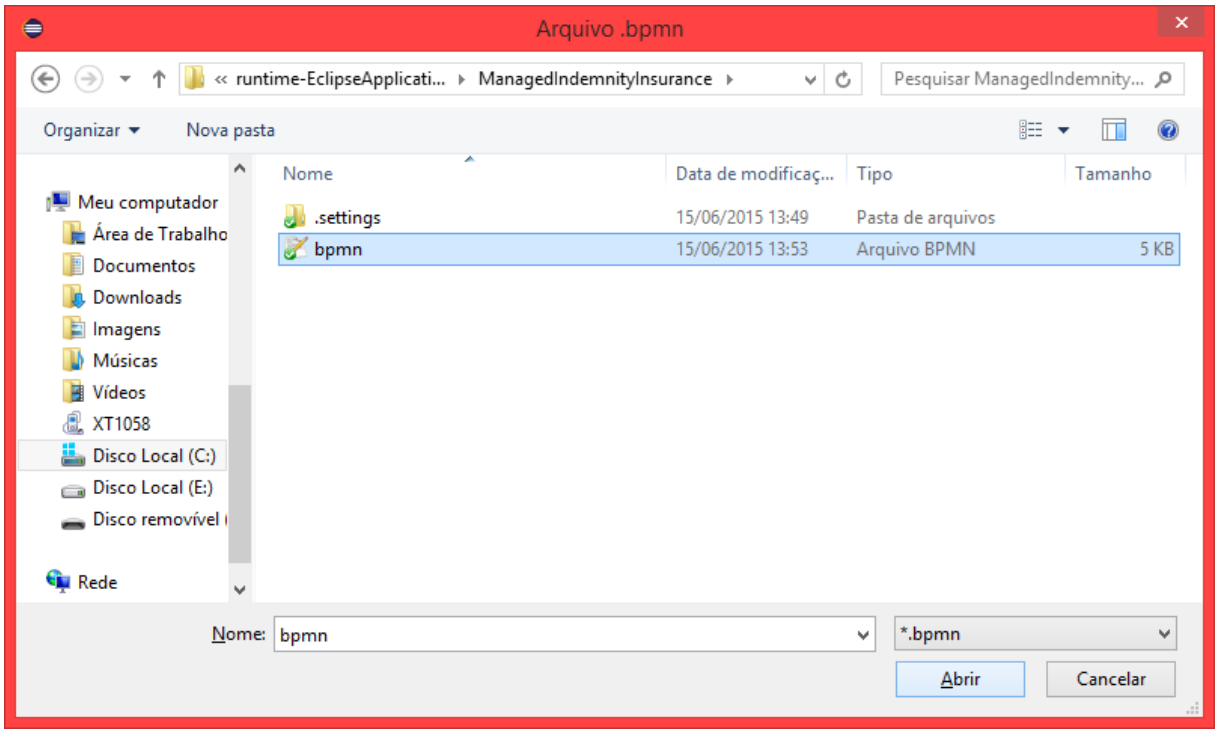
Em seguida é solicitado o diretório destino para a conversão. Neste exemplo utilizaremos o Desktop (C:\Users\Felipe\Desktop). A Figura 4.19 ilustra o último passo do procedimento de exportação. Agora o arquivo de extensão bpmn que pode ser lido pela ferramenta Bizagi encontra-se na área de trabalho (C:\Users\Felipe\Desktop).

Figura 4.17 – Caixa de diálogo correspondente ao primeiro passo da exportação de um diagrama BPMN na ferramenta iStar2BPMN



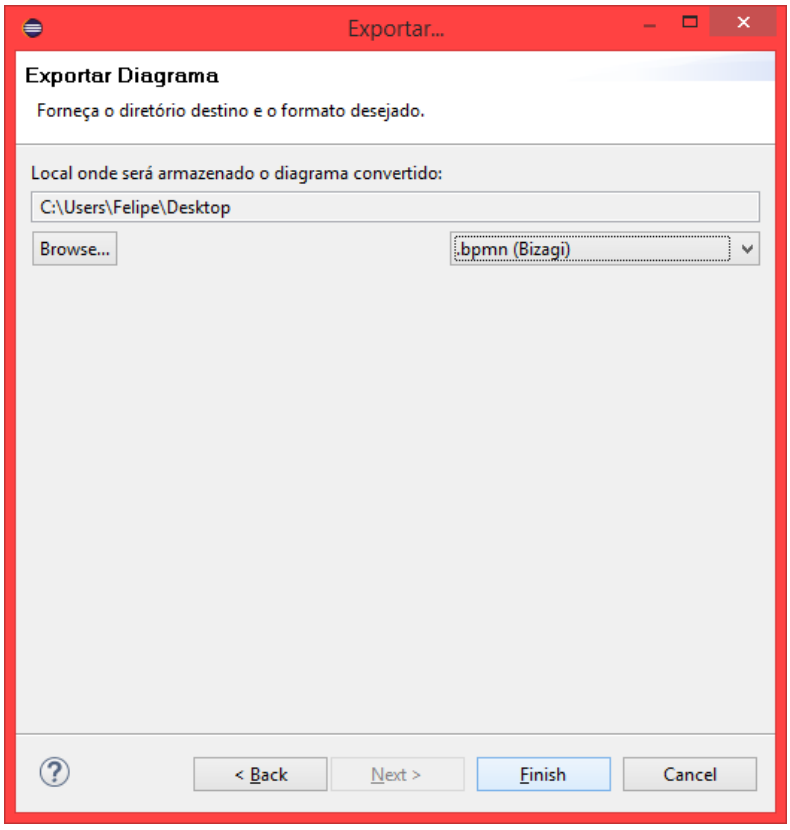
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.18 – Tela de solicitação de um arquivo de extensão BPMN na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

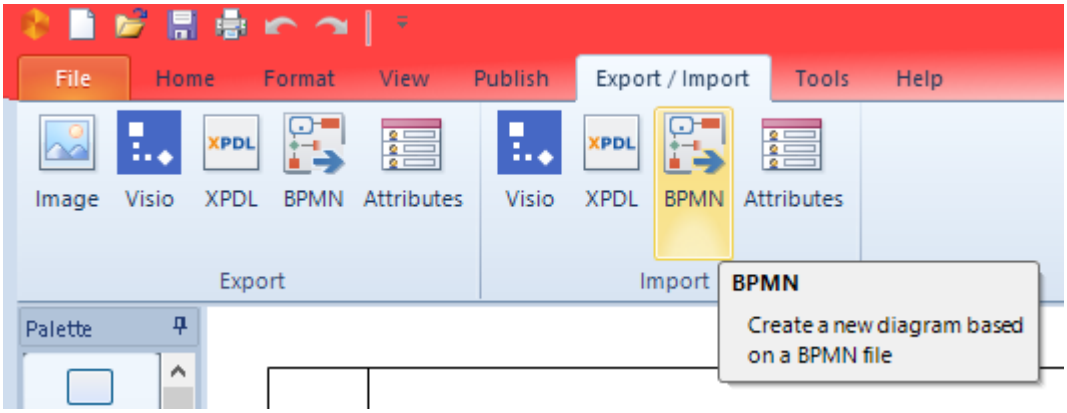
Figura 4.19 – Último passo da exportação para o Bizagi de um diagrama BPMN na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Já na interface gráfica da ferramenta Bizagi, é preciso importar o diagrama BPMN gerado e que se encontra na área de trabalho. A opção de importação de um arquivo de extensão bpmn (Bizagi) encontra-se na aba “Export / Import” como ilutra a Figura 4.20.

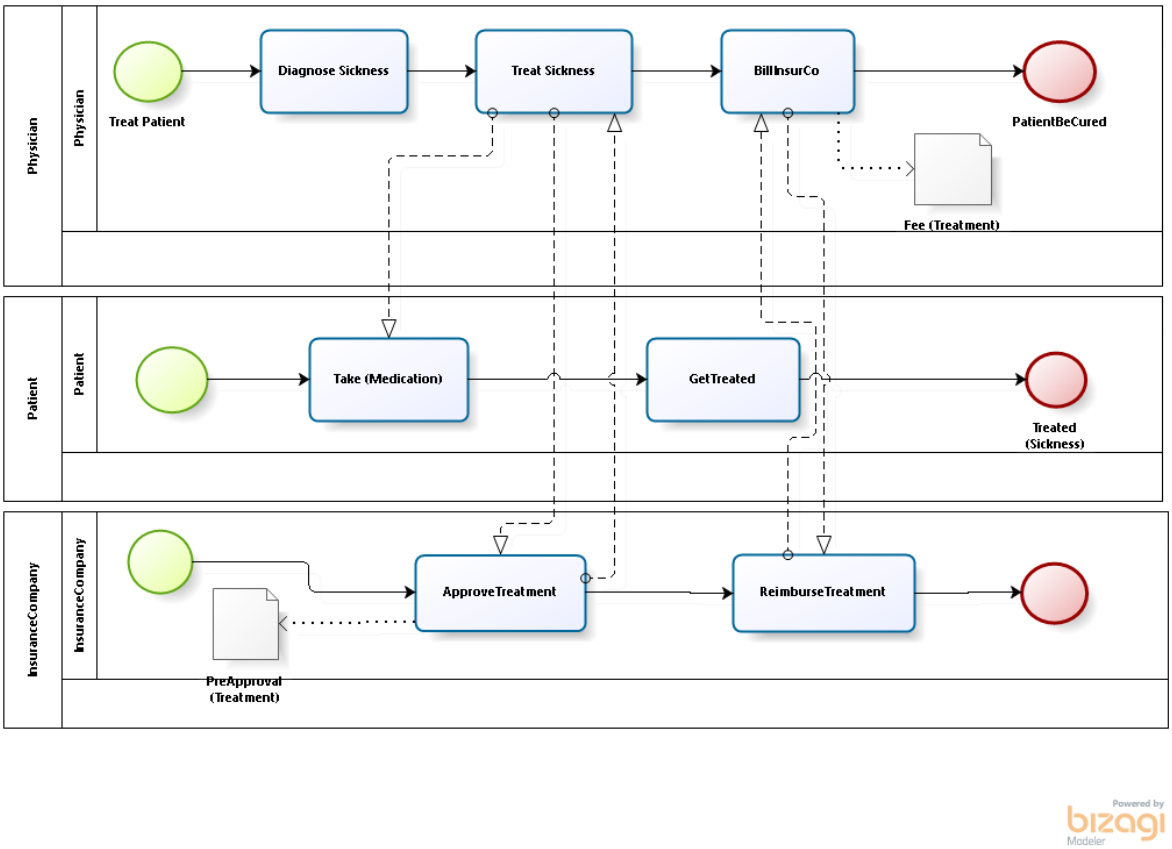
Figura 4.20 – Importação de diagrama na ferramenta Bizagi



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O diagrama resultante da importação encontra-se na Figura 4.21.

Figura 4.21 – Diagrama importado para a ferramenta Bizagi



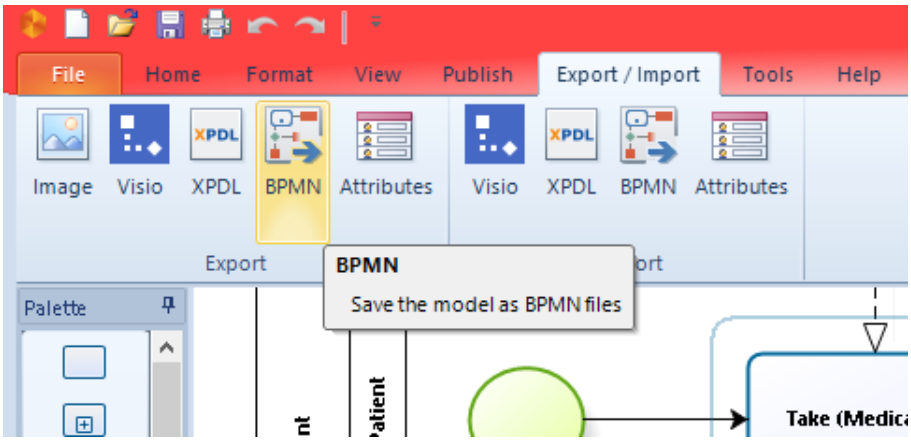
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

4.5. Passo 5 – Importação do diagrama BPMN para o iStar2BPMN

Neste ponto já foi possível importar para o iStar2BPMN um diagrama i* construído na ferramenta OpenOME. O mesmo foi então convertido para seu correspondente modelo BPMN e em seguida exportado para a ferramenta Bizagi.

Agora terá início o caminho inverso de todo o procedimento deste capítulo. Para importar o diagrama BPMN que está na interface gráfica da ferramenta Bizagi é preciso escolher a opção “Export BPMN” na guia “Export / Import” como ilutra a Figura 4.22.

Figura 4.22 – Exportação de diagrama na ferramenta Bizagi

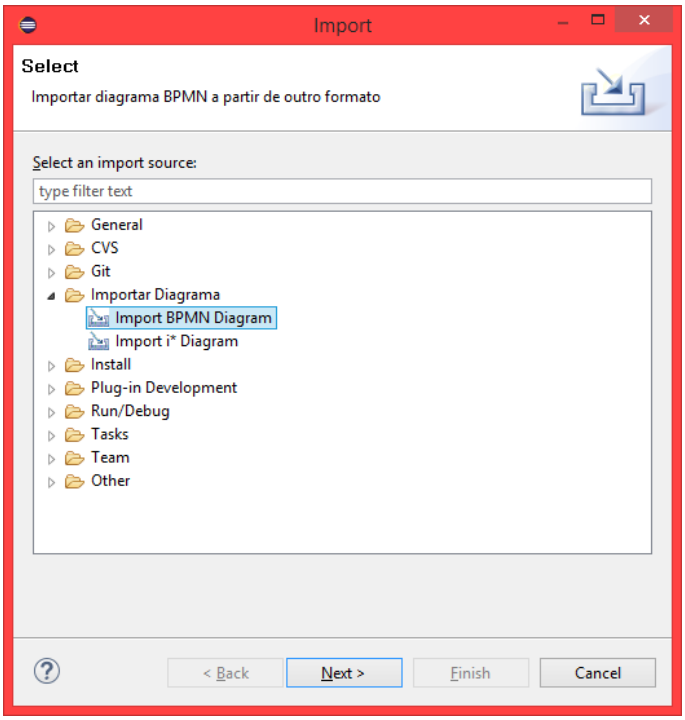


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

A pasta destino a ser utilizada neste exemplo será o diretório “C:\Users\Felipe\Desktop”. Após exportação será criado pelo Bizagi o arquivo de nome “diagrama” e extensão “bpmn” no diretório “C:\Users\Felipe\Desktop\New Model”.

Já na interface gráfica do iStar2BPMN, será importado o diagrama através da opção “File” -> “Import ...” -> “Import BPMN Diagram”, como ilutrado na Figura 4.23.

Figura 4.23 – Importação de diagrama BPMN na ferramenta iStar2BPMN

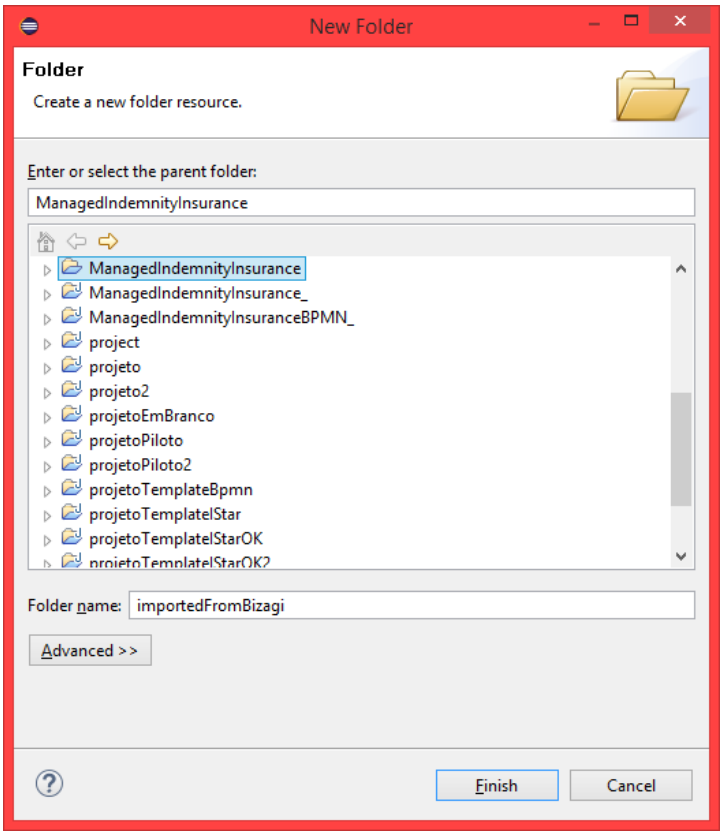


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Nas telas seguintes será escolhida a opção ".bpmn (Bizagi)" e solicitado o caminho para o arquivo (neste exemplo é o "C:\Users\Felipe\Desktop\New Model\diagrama.bpmn"). O usuário também deve definir o diretório onde será armazenado o diagrama convertido, neste exemplo será utilizado o "C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance\importedFromBizagi".

Anteriormente a importação, uma nova pasta (importedFromBizagi) foi criada dentro do projeto para facilitar a organização geral. Novas pastas podem ser criadas clicando com o botão direito do mouse sobre o projeto, escolhendo a opção "New" -> "Folder" e em seguida fornecendo o nome da nova pasta, como ilutrado na Figura 4.24.

Figura 4.24 – Importação de diagrama BPMN na ferramenta iStar2BPMN



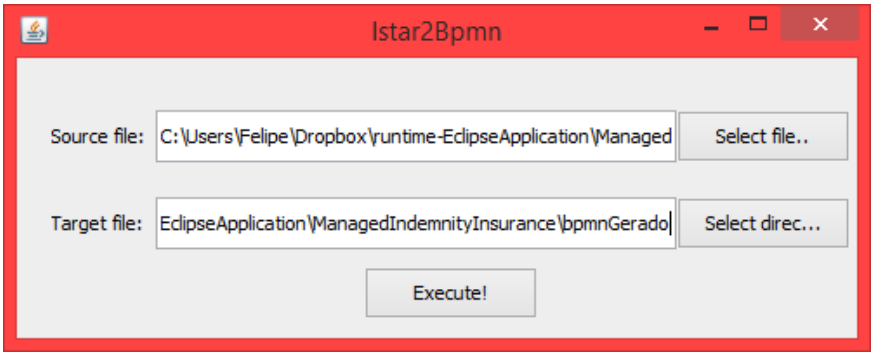
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O resultado da importação é análogo a Figura 4.16 e com isso chega-se ao final do passo 5.

4.6. Passo 6 – Conversão do diagrama BPMN para i*

Agora será gerado um modelo i* a partir do BPMN gerado no passo anterior. Como “Source file” será utilizado o “C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance\importedFromBizagi\diagrama.bpmn” e como “Target file” o diretório “C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance\bpmnGerado”. Os passos abaixo serão transcritos do trabalho de Melo [11] e representam a conversão de um diagrama BPMN em um diagrama i*.

Figura 4.25 – Seleção do arquivo de entrada e do diretório destino no processo de conversão entre modelos BPMN e i*

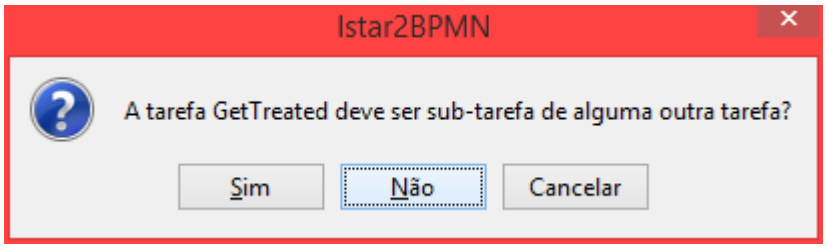


Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O primeiro questionamento gerado pela ferramenta (Figura 4.26) 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*). As 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.

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”).

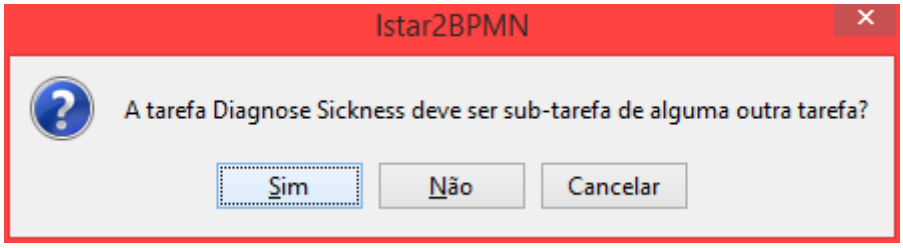
Figura 4.26 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa Get Treated)



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

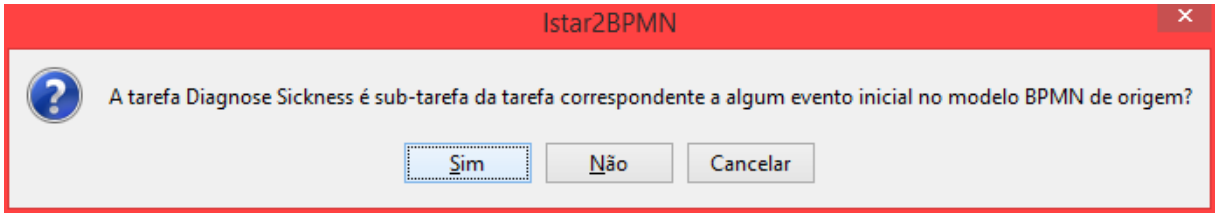
A ferramenta pergunta ainda sobre as tarefas Diagnose Sickness, Treat Sickness e BillInsurCo. Para cada uma delas, o usuário deve escolher a primeira opção (“Sim”), uma vez que elas devem fazer parte de uma decomposição de tarefa (Figura 4.27). Posteriormente, a ferramenta pergunta ao usuário se a tarefa a ser decomposta corresponde a algum evento inicial no modelo BPMN de origem (Figura 4.28). 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 4.29). As figuras Figura 4.30 a Figura 4.33 ilustram o processo.

Figura 4.27 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa Diagnose Sickness)



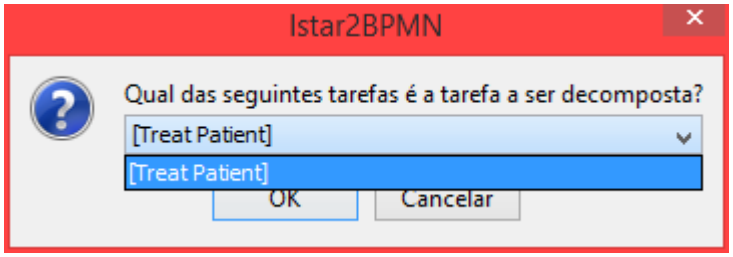
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.28 – Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa Diagnose Sickness)



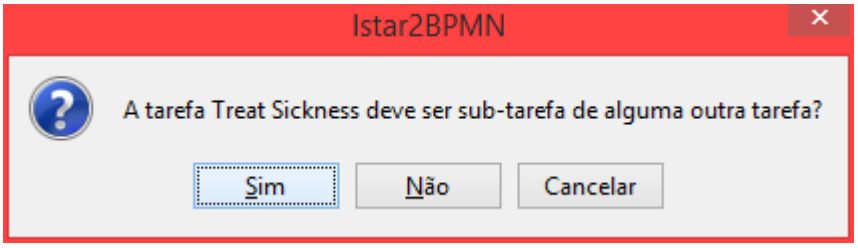
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.29 – Tela para escolha da tarefa a ser decomposta



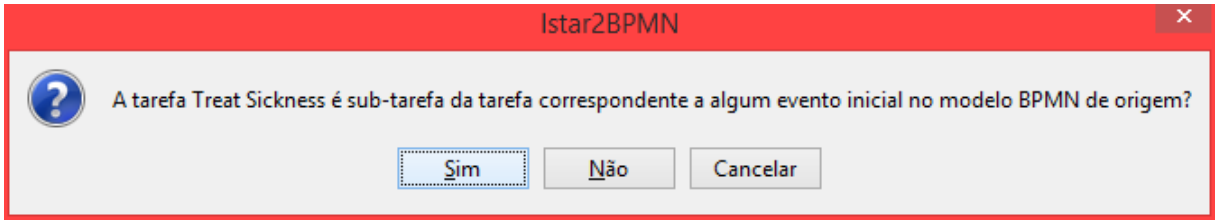
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.30 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa Treat Sickness)



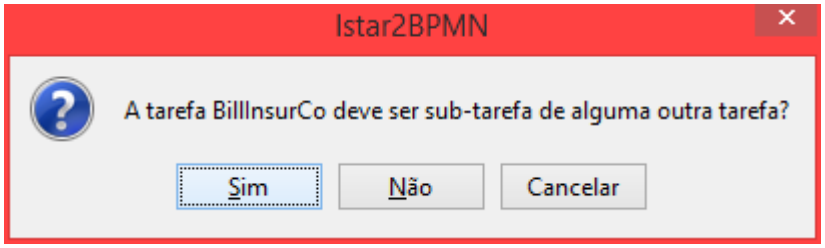
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.31 – Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa Treat Sickness)



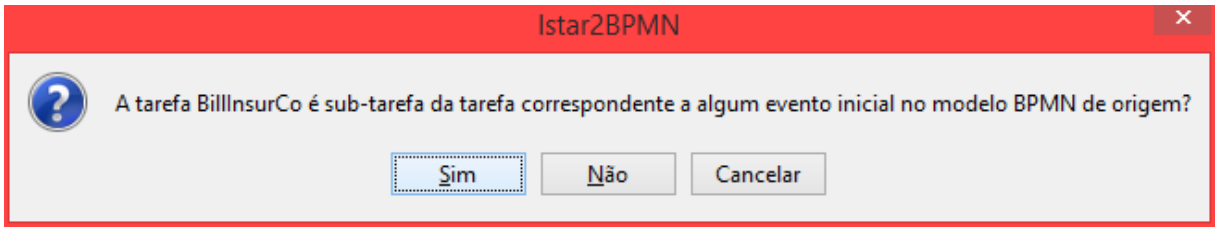
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.32 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa BillInsurCo)



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

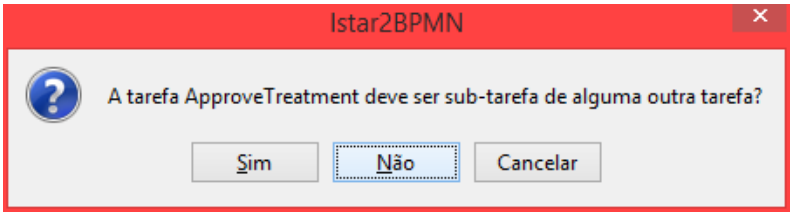
Figura 4.33 – Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa BillInsurCo)



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

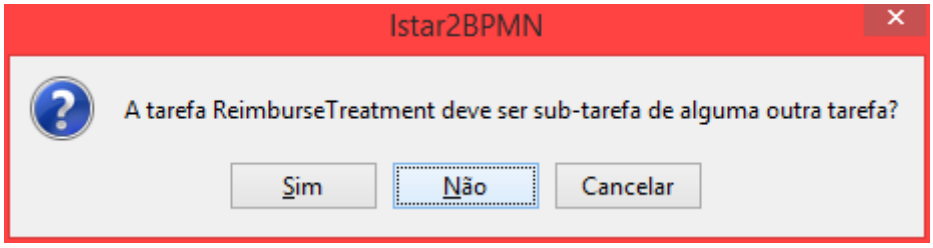
A mesma pergunta é feita para as atividades ApproveTreatment e ReimburseTreatment, do participante InsuranceCompany. Da mesma forma, como não pertencem a nenhuma decomposição de tarefa que está no escopo do processo, o usuário deve escolher a segunda opção (“Não”) como ilustrado nas figuras Figura 4.34 e Figura 4.35. 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 dependêndia de objetivo ou não.

Figura 4.34 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa ApproveTreatment)



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

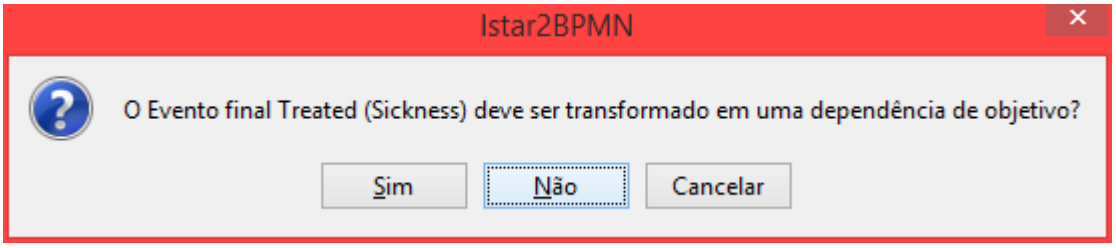
Figura 4.35 – Tela para decidir se determinada tarefa deve pertencer a uma decomposição de tarefa (tarefa ReimburseTreatment)



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

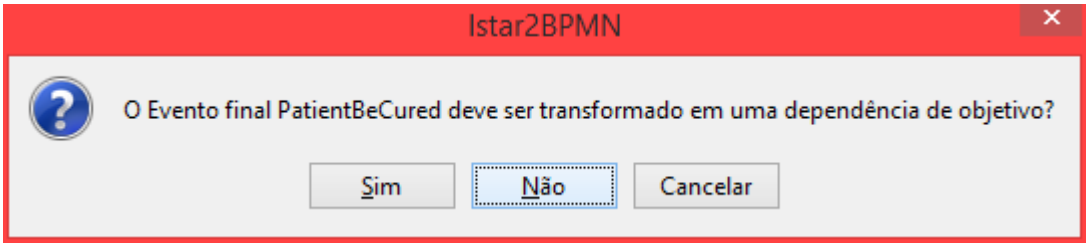
Caso o usuário escolha a primeira opção, ele deverá escolher a tarefa a qual essa dependência de objetivo deve estar ligada. Caso contrário, um objetivo e uma ligação meio-fim entre esse objetivo e a tarefa que corresponde à atividade imediatamente anterior ao evento final são incluídos no ator correspondente. 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 a um ator, e, portanto, o usuário deve escolher a opção “Não” (figuras Figura 4.36 e Figura 4.37).

Figura 4.36 – Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa Treated (Sickness))



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Figura 4.37 – Tela para decidir se uma tarefa é sub-tarefa da rotina do processo (tarefa PatientBeCured)



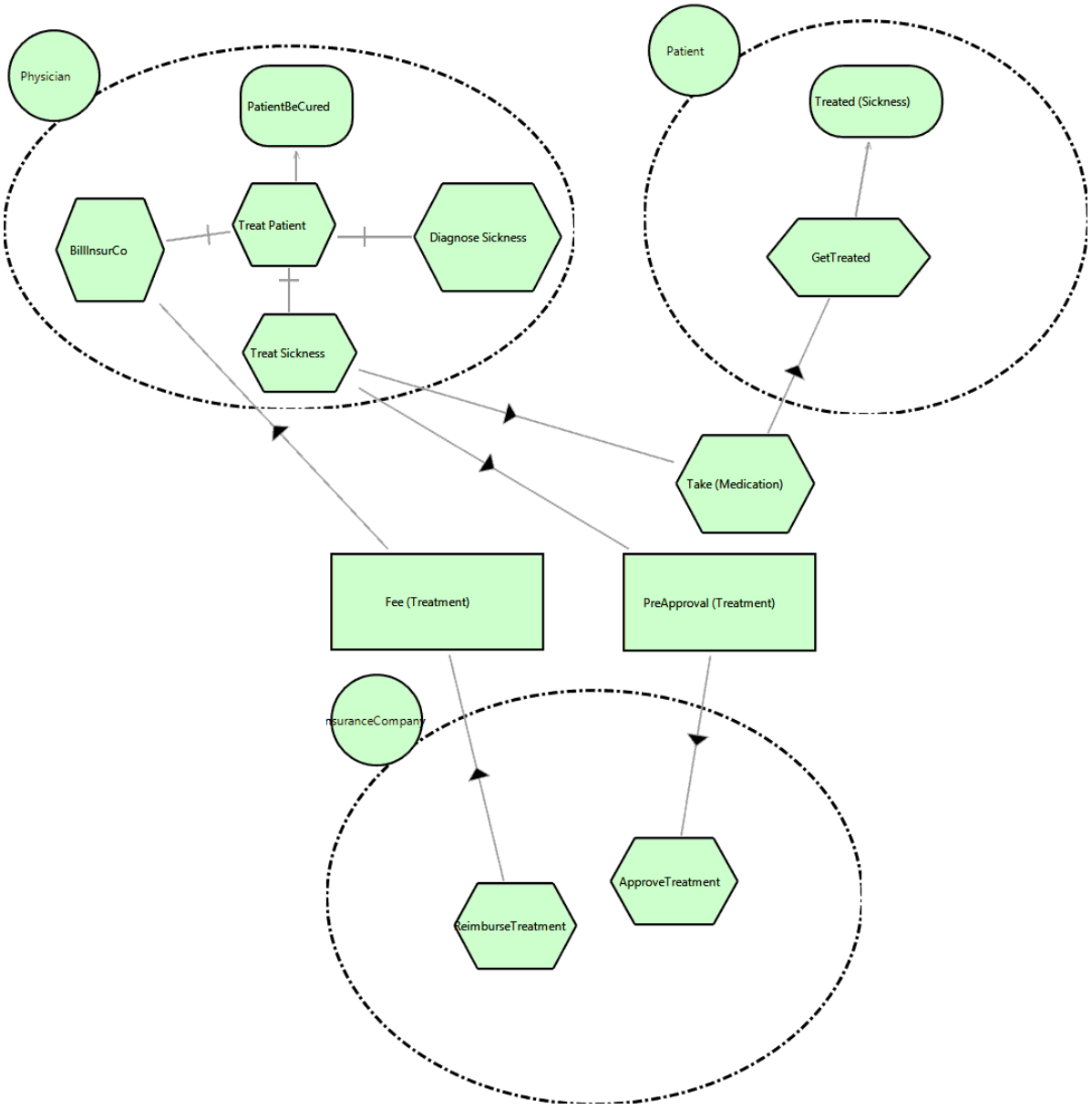
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

Com isso obtém-se o diagrama representado na Figura 4.38 e chega-se ao final do passo 6. 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 forma 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 [11].

Figura 4.38 – Modelo i* gerado pela transformação



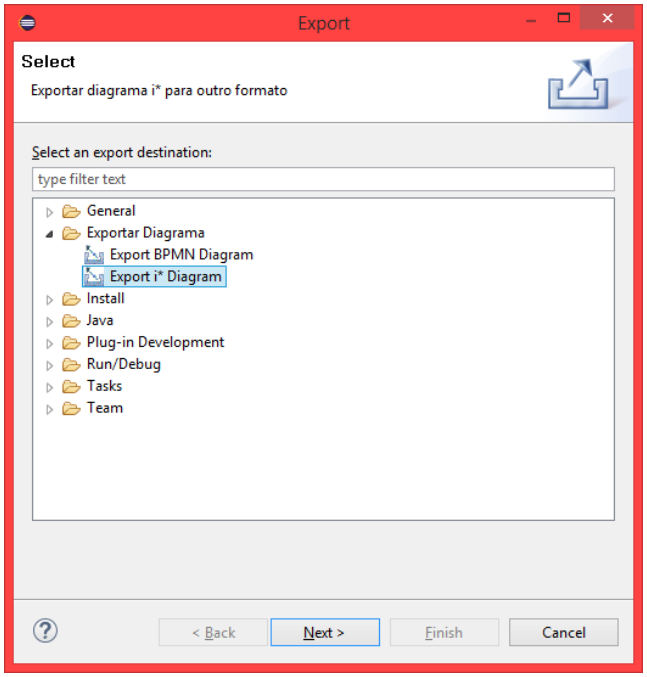
Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

4.7. Passo 7 – Exportação do diagrama i* do iStar2BPMN para o OpenOME

Neste último passo será gerado, por conversão de XMI, a partir do diagrama gerado no passo anterior e presente na ferramenta iStar2BPMN, um diagrama reconhecível pela ferramenta

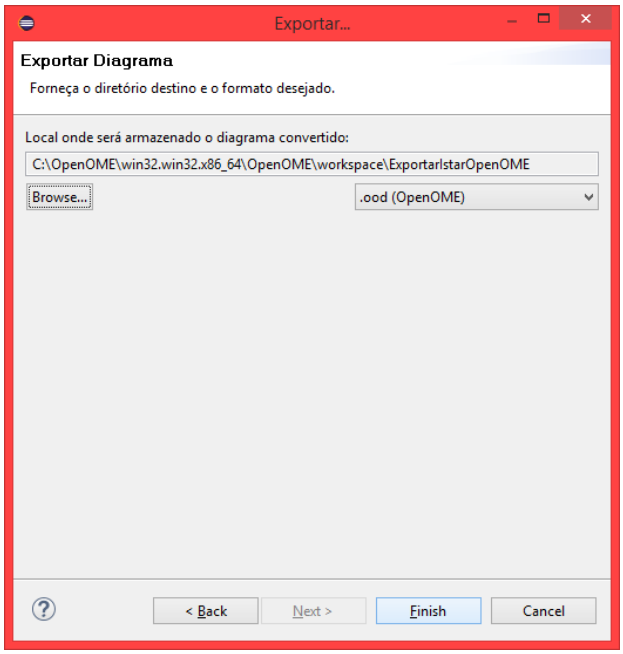
OpenOME. Para isso, na interface gráfica do iStar2BPMN seleciona-se a opção “File” -> “Export” -> “Export i* Diagram” (Figura 4.39) e nas telas seguintes serão fornecidos o caminho até o arquivo de extensão istar que corresponde ao modelo a ser exportado (C:\Users\Felipe\Dropbox\runtime-EclipseApplication\ManagedIndemnityInsurance\bpmnGerado\istar.istar), o arquivo de extensão istardiagram e o diretório destino da conversão (Figura 4.40).

Figura 4.39 – Tela inicial de exportação de modelo i* na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

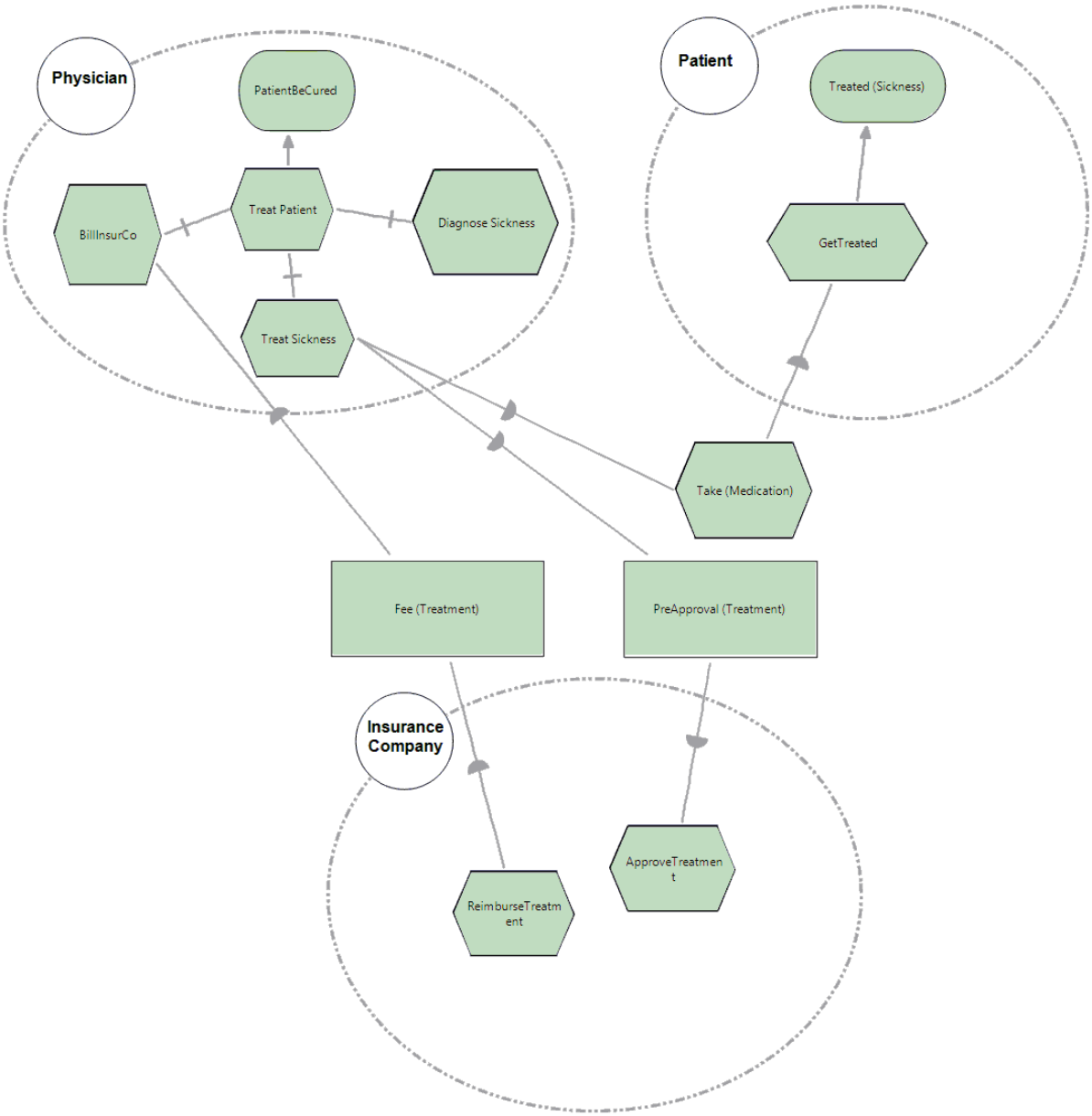
Figura 4.40 – Última tela da exportação de um diagrama i* na ferramenta iStar2BPMN



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

O resultado da exportação, já na interface gráfica do OpenOME está mostrado abaixo na Figura 4.41.

Figura 4.41 – Diagrama exportado do iStar2BPMN para o OpenOME



Fonte: *print screen* da aplicação no sistema operacional Windows 8.1

5. Conclusão

Aqui são apresentadas as contribuições e limitações encontradas durante o desenvolvimento deste trabalho. As considerações finais e possíveis direcionamentos para futuros trabalhos também se encontram neste capítulo.

5.1. Contribuições e limitações

Como contribuição principal deste trabalho, a ferramenta iStar2BPMN é agora capaz de importar modelos construídos no OpenOME ou Bizagi, assim como exportar para as ferramentas citadas diagramas construídos na própria interface gráfica do iStar2BPMN.

Outra contribuição foi a implementação e descrição de procedimentos de conversão entre formatos de arquivos XML, tendo como base uma classe facilitadora para a manipulação de arquivos XML também desenvolvida e descrita no presente trabalho: a classe Java ManipulaXml.

Dentre as dificuldades encontradas destaca-se a auxência de alguns atributos em alguma das ferramentas, ou seja nem todos os atributos de um elemento do diagrama encontram seu correspondente no diagrama convertido apenas por inexistência daquele tipo de atributo em outra ferramenta. Alguns atributos presentes na ferramenta OpenOME inexistem na ferramenta iStar2BPMN e portanto acabam sendo ignorados durante o procedimento de conversão de um diagrama i* no OpenOME para um diagrama i* no iStar2BPMN.

Outra dificuldade foi a inexistência de um software específico capaz de auxiliar na construção dos procedimentos de conversão entre formatos. Deste modo qualquer pequena falha no XML podia resultar em rejeição do diagrama inteiro por parte da ferramenta sendo utilizada (iStar2BPMN, OpenOME ou Bizagi) sem ao menos uma indicação de qual parte do código XML contém algum erro.

Finalmente, outra dificuldade foi o fato de que a ferramenta OpenOME permite a ligação entre elementos externos, ou seja elementos não pertencentes a algum compartment. O mesmo não é aceito na ferramenta iStar2BPMN por entrar em desacordo com a linguagem i* que não permite este tipo de situação. A solução encontra-se representada na Tabela 11.8 - Inserção de nota informativa num diagrama da ferramenta iStar2BPMN onde antes havia uma ligação inválida no diagrama do OpenOME fornecido para conversão".

A seguir estão transcritas do trabalho de Melo [11] algumas limitações encontradas anteriormente e que persistem:

- 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.
- A ausência de uma interface gráfica mais elaborada que minimize o número de passos (telas para tomada de decisão) no processo de transformação.
- Os editores gráficos de modelos i* e BPMN do iStar2BPMN ainda não possuem checadores de sintaxe.
- A ferramenta iStar2BPMN ainda não é capaz de lidar com subprocessos do BPMN.

5.2. Direções futuras

A transformação do XML de diagramas é um caminho para a compatibilização entre ferramentas, colaborando para o crescimento das mesmas. Como direcionamento futuro, espera-se que a ferramenta iStar2BPMN se torne compatível com outras ferramentas e outros formatos através da exportação e importação de diagramas escritos por exemplo em XPDL (XML Process Definition Language) [22], Q7 [20], iStarML [21], dentre outros formatos. O estudo de possíveis formatos a serem utilizados, suas peculiaridades e procedimentos de conversões ficarão como trabalhos futuros.

A classe Java ManipulaXml foi implementada visando a conversão de quaisquer formatos desde que escritos em XML bem formado. A mesma pode ser melhorada com a inclusão de novos métodos ou incrementando a eficiência dos já existentes. A criação de uma interface gráfica para a construção de procedimentos de conversões entre formatos XML diferentes, tendo como base a classe ManipulaXml, seria um gigantesco facilitador para a compatibilização entre ferramentas através de diversas linguagens diferentes. Esta classe pode ser um primeiro passo para uma grande ferramenta manipuladora de arquivos XML, capaz de criar rotinas de conversões entre formatos.

5.3. Considerações finais

Os objetivos do presente trabalho foram alcançados visto que após o término do mesmo tornou-se possível converter um diagrama i* construído no ambiente da ferramenta OpenOME para um formato reconhecido pela ferramenta iStar2BPMN, assim como um diagrama BPMN construído na ferramenta Bizagi também pode ser importado à ferramenta iStar2BPMN. Também foram implementadas as exportações, ou seja um diagrama i* construído na ferramenta iStar2BPMN pode ser convertido para um formato reconhecível pela ferramenta OpenOME e um diagrama BPMN construído no iStar2BPMN pode ser exportado para um formato reconhecível pelo Bizagi, completando assim a integração entre as três ferramentas.

Os procedimentos de conversão foram implementados na linguagem Java e incorporados ao código da ferramenta iStar2BPMN. O código atualizado da ferramenta será disponibilizado pelo autor em um repositório web.

A realização deste trabalho implicou em um grande crescimento acadêmico e pessoal do autor através de estudos sobre manipulação de arquivos XML e a linguagem Java, assim como a responsabilidade de desenvolver um projeto de longo prazo.

6. Referências

1. DE SOUZA, Bruno Nogueira. Integração de dispositivos móveis à ferramentas de workflow. Universidade de Brasilia Instituto de Ciências Exatas Departamento de Ciência da Computaçao. CEP, v. 70910, p. 900. Disponível em: <http://monografias.cic.unb.br/dspace/handle/123456789/51>. Acesso em: 08/04/2015.
2. GONÇALVES, José Ernesto Lima. As empresas são grandes coleções de processos. RAE, v. 40, n. 1, p. 7, 2000. Disponível em: <www.scielo.br/pdf/rae/v40n1/v40n1a02.pdf>. Acesso em: 08/04/2015.
3. KALPIC, B.; BERNUS, P. Business process modelling in industry: the powerful tool in enterprise management. Computers in Industry, v. 47, n. 3, p. 299-318, march 2002. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0166361501001518>. Acesso em: 08/04/2015
4. KIRIKOVA, M. Explanatory capability of enterprise models. Data & Knowledge Engineering, v. 33, p. 119-136, 2000. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169023X99000488>. Acesso em: 08/04/2015.
5. CAMPOS, André LN. Modelagem de Processos com BPMN. Brasport, 2014. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=zZAZBAAAQBAJ&oi=fnd&pg=PA25&ots=gDz_DMF0yr&sig=sFt8nD2GoNzSrcvBP7b-C-fQelo#v=onepage&q&f=false>. Acesso em: 06/04/2015.
6. PÁDUA, SID de; CAZARINI, Edson Walmir; INAMASU, Ricardo Yassushi. Modelagem organizacional: captura dos requisitos organizacionais no desenvolvimento de sistemas de informação. Gestão & Produção, v. 11, n. 2, p. 197-209, 2004. Disponível em: <http://www.scielo.br/pdf/gp/v11n2/a06v11n2>. Acesso em: 08/04/2015
7. SILVA, Carla TLL; BORBA, Clarissa; CASTRO, Jaelson. G2SPL: Um Processo de Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software. In: WER. 2010. Disponível em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER10/silva.pdf>. Acesso em: 08/04/2015.
8. G. Grau, E. Yu, J. Horkoff, S. Abdulhadi, “i* Guide”. Disponível em: <http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67>. Acesso em: 24/03/2015.
9. CUESTA, Lidia López. The notion of Specialization in the i* Framework. PhD THESIS. Universitat Politècnica de Catalunya, Barcelona, February 2013. Disponível em: <http://www.essi.upc.edu/~gessi/papers/TesisLidiaLopez.pdf>. Acesso em: 24/03/2015.
10. KOLIADIS, George et al. Combining i* and BPMN for Business Process Model lifecycle management. In: Business Process Management Workshops. Springer Berlin Heidelberg, 2006. p. 416-427. Disponível em: <http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1578&context=infopapers>. Acesso em: 10/04/2015.
11. MELO, Eduardo Bezerra de. Ferramenta para automatizar as transformações bidirecionais entre i* e BPMN. Trabalho de Graduação. Universidade Federal de Pernambuco, 2014. Disponível em: <http://www.cin.ufpe.br/~tg/2013-2/ebm2.pdf>. Acesso em: 10/04/2015.

12. ALVES, Rebeca de Souza. 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. Disponível em: <http://www.cin.ufpe.br/~tg/2012-2/rsa2.pdf>. Acesso em: 10/04/2015.
13. YU, Eric. Modelling strategic relationships for process reengineering. Social Modeling for Requirements Engineering, v. 11, p. 2011, 2011. Disponível em: <http://www.cs.utoronto.ca/pub/eric/DKBS-TR-94-6.pdf>. Acesso em: 10/04/2015.
14. HORKOFF, Jennifer; YU, Yijun; Yu, Eric. OpenOME: An Open-source Goal and Agent-Oriented Model Drawing and Analysis Tool. CEUR Proceedings of the 5th International i* Workshop (iStar 2011). Disponível em: <http://ceur-ws.org/Vol-766/paper27.pdf>. Acesso em: 10/04/2015.
15. WHITE, Stephen A. BPMN modeling and reference guide: understanding and using BPMN. Future Strategies Inc., 2008. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=0Z2Td3bCYW8C&oi=fnd&pg=PA9&dq=white+introduction+to+bpmn&ots=r5VLOptRTS&sig=SCG91xWnZzoMgu9EEF10pVIQ5X0>. Acesso em: 11/04/2015.
16. Owen, Martin, and Jog Raj. "BPMN and business process management." Introduction to the New Business Process Modeling Standard (2003). Disponível em: <http://da-rosi4management.googlecode.com/svn/trunk/material/bpmn/BPMN%20and%20Business%20Process%20Management.pdf>. Acesso em: 11/04/2015.
17. KOLOVOS, Dimitrios S.; PAIGE, Richard F.; POLACK, Fiona AC. Eclipse development tools for epsilon. In: Eclipse Summit Europe, Eclipse Modeling Symposium. 2006. p. 200. Disponível em: <http://www.researchgate.net/profile/Fiona_Polack/publication/228927046_Eclipse_Development_Tools_for_Epsilon/links/004635179605a3ec22000000.pdf>. Acesso em: 06/04/2015.
18. BizAgi Process Modeler. Disponível em: <http://www.bizagi.com/esp/productos/ba-modeler/modeler.html>. Acesso em: 24/03/2015.
19. ALMEIDA, Mauricio Barcillos. Uma introdução ao XML, sua utilização na Internet e alguns conceitos complementares. Ciência da Informação, v. 31, n. 2, p. 5-13, 2002. Disponível em: <http://www.scielo.br/pdf/ci/v31n2/12903>. Acesso em: 08/05/2015.
20. LEITE, Julio Cesar Sampaio do Prado; YU, Yijun; LIU, Lin; YU, Eric S. K.; MYLOPOULOS, John; Quality-Based Software Reuse. Disponível em: <http://www.cs.toronto.edu/~yijun/literature/paper/leite05caise.pdf>. Acesso em: 09/06/2015.
21. CARES, Carlos et al. Towards interoperability of i* models using iStarML. Computer Standards & Interfaces, v. 33, n. 1, p. 69-79, 2011. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0920548910000334>. Acesso em: 09/06/2015.
22. SHAPIRO, Robert. A technical comparison of XPDL, BPML and BPEL4WS. Cape Visions, 2002. Disponível em: <http://www.bptrends.com/bpt/wp-content/publicationfiles/Comparison%20of%20XPDL%20and%20BPML_BPEL%2012-8-02111.pdf>. Acesso em: 09/06/2015.

7. APÊNDICE A - Estudo do XMI de um diagrama i* no OpenOME

Todo diagrama i* na ferramenta OpenOME consiste em um arquivo de extensão “.ood” (OpenOME Diagram), escrito em XML. A seguir será feito um estudo a respeito desse formato de arquivo objetivando encontrar características relevantes a um processo de conversão de um diagrama i* da ferramenta OpenOME para um formato reconhecível pela ferramenta iStar2BPMN.

7.1. Estrutura Geral de um arquivo “.ood”

A ordem em que os elementos aparecem no XML do arquivo ”.ood” deve ser respeitada de acordo com a listagem abaixo. Obs.: Alguns atributos foram removidos para facilitar a visualização.

- Linha inicial do arquivo “.ood”:

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

- Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama. Apresenta obrigatoriamente como filhos diretos duas tags: uma tag “edu.toronto.cs.openome_model:Model” e uma tag “notation:Diagram”.

```
<xmi:XMI>
```

- Tag pai de todas as declarações de elementos do diagrama.

```
<edu.toronto.cs.openome_model:Model>
```

- Tag que representa a declaração de uma intention. Sabe-se que é uma intention externa quando ela é filha direta da tag <edu.toronto.cs.openome_model:Model>. Pode representar uma Task, um Recurso, um Goal ou um Softgoal. A diferenciação se faz pelo atributo “xmi:type” desta tag.

```
<intentions>
```

- Tag que representa a declaração de uma ligação de contribuição. A diferenciação entre tipos de contribution links se faz pelo atributo “xmi:type” desta tag. Apresenta um atributo “source” que informa o id do elemento de onde partiu a ligação e um atributo “target” que indica em que elemento a ligação chega.

```
<contributions>
```

- Tag que representa a declaração de uma ligação de dependência. Apresenta os atributos “dependencyFrom” e “dependencyTo” que indicam o target e o source respectivamente.

```
<dependencies>
```

- Tag que representa a declaração de uma ligação de decomposição. Apresenta um atributo “source” que informa o id do elemento de onde partiu a ligação e um atributo “target” que indica em que elemento a ligação chega.

```
<decompositions>
```

- Tag que representa a declaração de um container. Pode representar um Actor, um Agent, um Position ou um Role. A diferenciação se faz pelo atributo “xmi:type” desta tag.

```
<containers>
```

- Bloco que representa a declaração de um container com uma ou mais tags filhas <intentions> representando elementos pertencentes ao boundary do compartment, ou seja intentions internas ao mesmo.

```
<containers>
  <intentions>
  <intentions>
  [...]
<containers>
```

- Tag que representa a declaração de uma ligação de associação. A diferenciação entre tipos de association links se faz pelo atributo “xmi:type” desta tag. Apresenta um atributo “source” que informa o id do elemento de onde partiu a ligação e um atributo “target” que indica em que elemento a ligação chega.

```
<associations>
```

- Tag pai de todos os blocos de tags <children>. Cada bloco representa algum compartment, alguma intention ou algum compartment com intention(s) no diagrama. Apresenta o atributo “element” com valor igual ao do atributo “xmi:id” da tag <edu.toronto.cs.openome_model:Model>. Contém o atributo “name” cujo valor é o nome do arquivo “.ood”. Contém obrigatoriamente uma tag filha direta “styles” com atributo xmi:type de valor "notation:DiagramStyle".

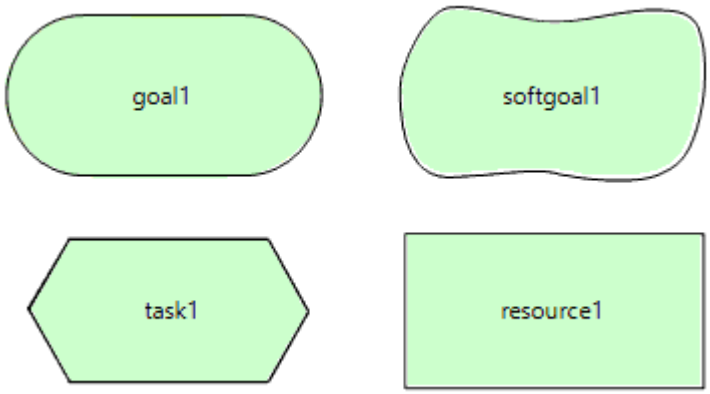
```
<notation:Diagram>
```

7.1.1.Intention

O bloco de tags <children> a seguir pode aparecer zero ou mais vezes, e representa uma intention (Goal, Softgoal, Task, Resource) no diagrama. Sabe-se que é uma intention externa (que não pertence a nenhum compartment) quando ao ser declarada ela foi filha direta da tag <edu.toronto.cs.openome_model:Model>. É possível encontrar onde foi declarada a tag <intention> correspondente a esse bloco buscando no XML a tag <intention> cujo atributo “xmi:id” apresenta o mesmo valor que o atributo “element” da tag <children> principal do bloco de tags “children”. O conjunto formado pela tag “children” com duas tags filhas “children”, seguidas pela tag “layoutConstraint” representa uma intention. A terceira tag <children> deve conter uma tag filha <layoutConstraint>. A Figura 7.1 ilustra os tipos de intentions existentes.

<children xmi:type="notation:Shape" type=[Ver Tabela 7.2]> <children xmi:type="notation:DecorationNode" type=[Ver Tabela 7.2]/> <children xmi:type="notation:DecorationNode" type=[Ver Tabela 7.2]> <layoutConstraint xmi:type="notation:Location"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Intention

Figura 7.1 – Intentions na ferramenta OpenOME



Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

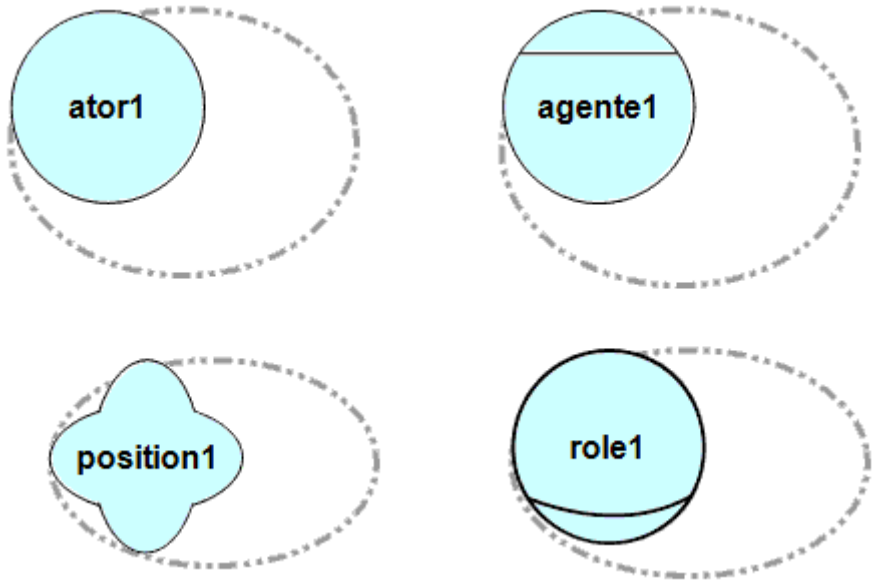
7.1.2.Compartment

O bloco a seguir pode aparecer zero ou mais vezes, e representa um compartment (Actor, Agent, Role, Position) no diagrama. O conjunto formado pela tag “children” com duas tags filhas “children”, seguidas pela tag “layoutConstraint” representa um compartment. A terceira tag “children” deve conter duas tags “styles”, a primeira com atributo “xmi:type” de valor "notation:SortingStyle" e a

segunda com atributo “xmi:type” de valor "notation:FilteringStyle". A Figura 7.2 ilustra os tipos de compartments existentes.

<children xmi:type="notation:Shape" type=[Ver Tabela 7.2]> <children xmi:type="notation:DecorationNode" type=[Ver Tabela 7.2]/> <children xmi:type="notation:BasicCompartment" type=[Ver Tabela 7.2]> <styles xmi:type="notation:SortingStyle"/> <styles xmi:type="notation:FilteringStyle"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment

Figura 7.2 - Compartments na ferramenta OpenOME



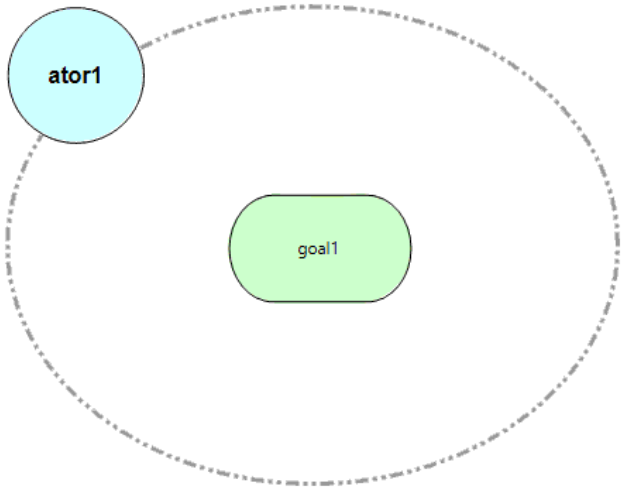
Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

7.1.3.Compartment com intention interna

O bloco a seguir pode aparecer zero ou mais vezes, sempre como filho direto da tag <notation:Diagram>, e representa um compartment (Actor, Agent, Role, Position) com uma intention interna a ele (Goal, Softgoal, Task, Resource) no diagrama. O conjunto formado pela tag “children” com duas tags filhas “children”, seguidas pela tag “layoutConstraint” representa um compartment. Enquanto O conjunto formado pela tag “children” com duas tags filhas “children”, seguidas pela tag “layoutConstraint” representa uma intention. A Figura 7.3 ilustra a situação.

<children xmi:type="notation:Shape" type=[Ver Tabela 7.2]> <children xmi:type="notation:DecorationNode" type=[Ver Tabela 7.2]/> <children xmi:type="notation:BasicCompartment" type=[Ver Tabela 7.2]>	Compartment Início
<children xmi:type="notation:Shape" type=[VER TABELA]> <children xmi:type="notation:DecorationNode" type=[VER TABELA]/> <children xmi:type="notation:DecorationNode" type=[VER TABELA]> <layoutConstraint xmi:type="notation:Location"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Intention interna
<styles xmi:type="notation:SortingStyle"/> <styles xmi:type="notation:FilteringStyle"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment Fim

Figura 7.3 - Intention interna a Compartment na ferramenta OpenOME



Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

- A tag a seguir deve aparecer no XML obrigatoriamente após todos os blocos de tags “children” correspondentes a compartments e intentions do diagrama. A mesma é filha direta da tag “notation:Diagram”.

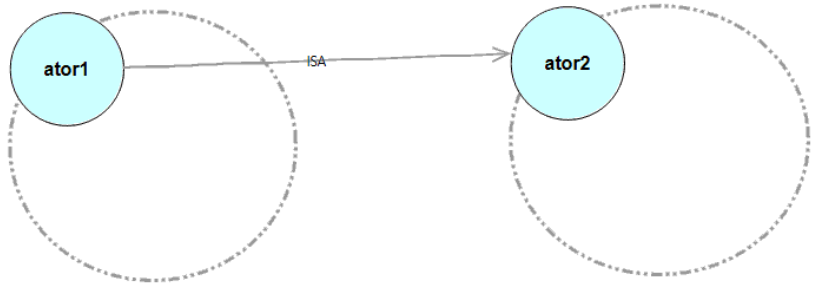
```
<styles xmi:type="notation:DiagramStyle"/>
```

7.1.4.Actor association

O bloco a seguir pode estar presente zero ou mais vezes e representa uma actor association. O conjunto formado pela tag “edges” e suas tags filhas: uma tag “children”, uma tag “styles” e uma tag “bendpoints” representa uma associação entre dois compartments. A tag “children” deve conter como filha uma tag “layoutConstraint”. A Figura 7.4 ilustra uma associação do tipo ISA.

<edges xmi:type="notation:Connector" type=[Ver Tabela 7.3] source="_w6ggkPHOEeS0MIfHkiFBbw" target="_x0byMPHOEeS0MIfHkiFBbw"> <children xmi:type="notation:DecorationNode" type=[VER TABELA]> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Actor association

Figura 7.4 - Actor association na ferramenta OpenOME



Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

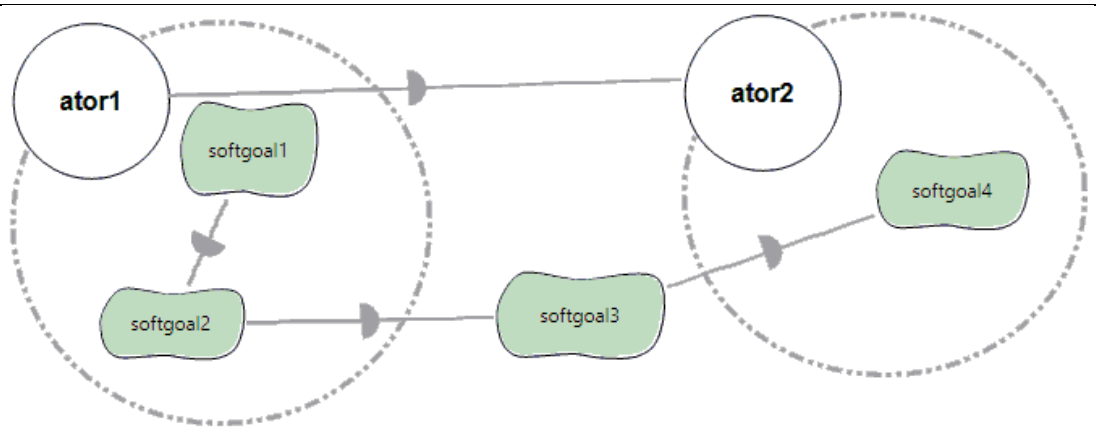
7.1.5.Dependency link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de dependência ou dependency link. O conjunto formado pela tag “edges” com atributo type de valor “4001” e suas tags filhas: uma tag “styles” e uma tag “bendpoints” representa uma ligação de dependência entre dois compartments ou duas intentions. A Figura 7.5 ilustra ligações de dependência.

<edges xmi:type="notation:Connector" type="4001" source="_vL3PofHdEeSyp9hXzpKFjg" target="_wEC00fHdEeSyp9hXzpKFjg"> <styles xmi:type="notation:FontStyle"/>	Dependency link

<bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	

Figura 7.5 - Dependency links na ferramenta OpenOME



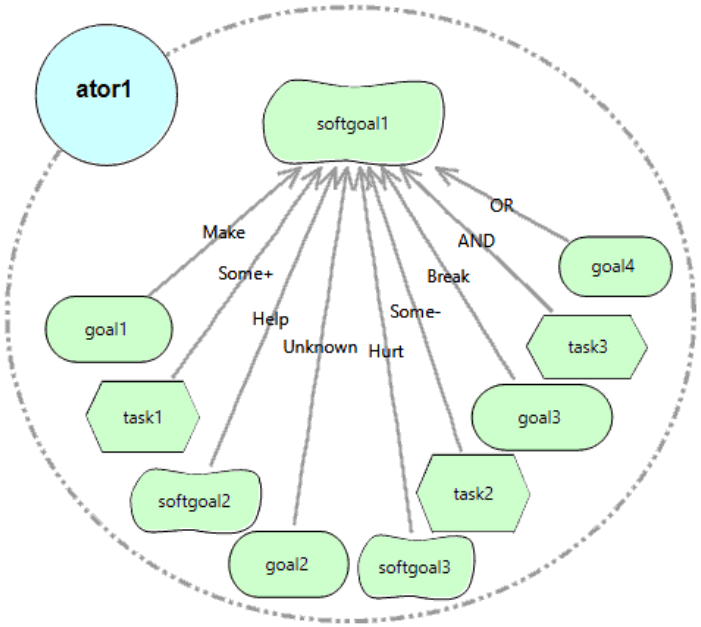
Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

7.1.6.Contribution link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de contribuição ou contribution link. O conjunto formado pela tag “edges e suas tags filhas: uma tag “children”, uma tag “styles” e uma tag “bendpoints” representa uma ligação de contribuição entre duas intentions. A tag “children” deve conter uma tag “layoutConstraint”. A Figura 7.6 ilustra ligações de contribuição.

<edges xmi:type="notation:Connector" type=[Ver Tabela 7.3] source="_s5kK8fHfEeSyp9hXzpKFjg" target="_PJB6YfHfEeSyp9hXzpKFjg"> <children xmi:type="notation:DecorationNode" type=[VER TABELA]> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Contribution link

Figura 7.6 - Contribution links na ferramenta OpenOME



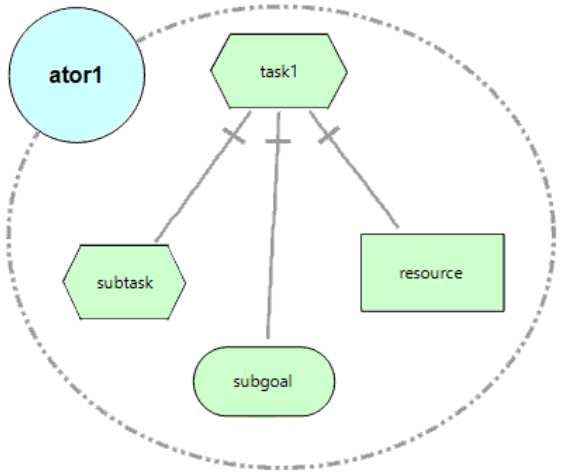
Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

7.1.7.Task decomposition link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de decomposição de tarefa ou task decomposition link. O conjunto formado pela tag “edges” com atributo type de valor “4002” e suas tags filhas: uma tag “styles” e uma tag “bendpoints” representa uma ligação de decomposição entre dois elements. A Figura 7.7 ilustra ligações de decomposição.

<edges xmi:type="notation:Connector" type="4002" source="_gzdp4fHhEeSyp9hXzpKFjg" target="_ecfLcfHhEeSyp9hXzpKFjg"> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Task Decomposition Link

Figura 7.7 - Task Decomposition Links na ferramenta OpenOME



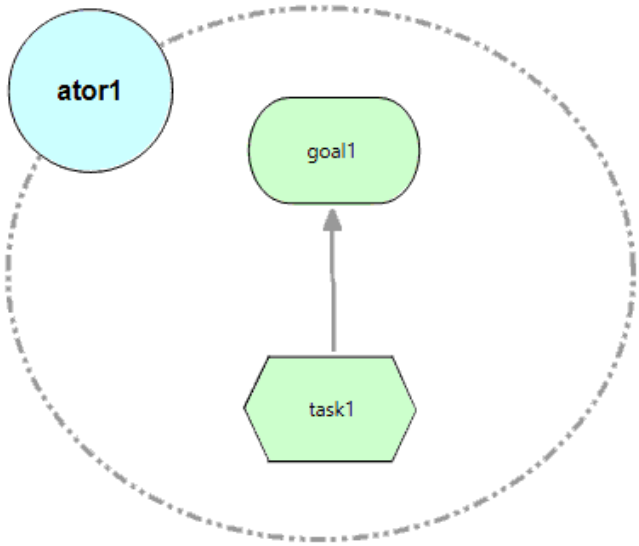
Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

7.1.8.Means-end link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de means-end. O conjunto formado pela tag “edges” com atributo type de valor “4003” e suas tags filhas: uma tag “styles” e uma tag “bendpoints” representa uma ligação de means-end entre duas intentions. A Figura 7.8 ilustra uma ligação de means-end.

<edges xmi:type="notation:Connector" type="4003" source="_VUG1MfHiEeSyp9hXzpKFjg" target="_UnVOwfHiEeSyp9hXzpKFjg"> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Means-End Link

Figura 7.8 - Means-End Link na ferramenta OpenOME



Fonte: screenshot da ferramenta OpenOME no sistema operacional Windows 8.1

7.2. Listagem completa de tags e atributos

A relação entre cada uma das tags de um ".ood" e seus atributos segue abaixo.

Tabela 7.1 - Listagem completa de tags e atributos de um ood na ferramenta OpenOME

Tag	Atributo	Valor
xml	version	1.0
	encoding	UTF-8
xmi:XMI	xmi:version	2.0
	xmlns:xmi	http://www.omg.org/XMI
	xmlns:edu.toronto.cs.openome_model	http:///edu/toronto/cs/openome_model.ecore
	xmlns:notation	http://www.eclipse.org/gmf/runtime/1.0.2/notation
edu.toronto.cs.openome_model:Model	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
intentions	xmi:type	edu.toronto.cs.openome_model:Goal
		edu.toronto.cs.openome_model:Softgoal
		edu.toronto.cs.openome_model:Task
		edu.toronto.cs.openome_model:Resource
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.

	dependencyTo (opcional)	Atributo que está presente apenas quando o xmi:id da intentions é igual ao valor do atributo dependencyFrom de pelo menos uma tag dependencies. O valor desse atributo é igual ao conjunto de valores xmi:id de todas as dependencies que terminam nessa intentions, separadas por um espaço em branco. Ex.: Se existe uma tag intentions com atributo xmi:id de valor "_0F7RgPG_EeS0MIfHkiFBbw", e duas tags dependencies cujos atributos dependencyFrom apresentam o mesmo valor "_0F7RgPG_EeS0MIfHkiFBbw". Seja o valor do atributo xmi:id de uma das tags dependencies igual a "_wiJMQPHAEeS0MIfHkiFBbw" e o valor do xmi:id da outra tag dependencies igual a "_zaM-gPHAEeS0MIfHkiFBbw". Então o atributo dependencyTo da tag intentions irá conter o valor "_wiJMQPHAEeS0MIfHkiFBbw _zaM-gPHAEeS0MIfHkiFBbw" que é a concatenação dos valores dos atributos xmi:id de cada tag dependencies.
	dependencyFrom (opcional)	Análogo ao atributo dependencyTo, porém o valor desse atributo é igual ao conjunto de valores xmi:id de todas as dependencies que começam nessa intentions, separadas por um espaço em branco. Diferentemente do dependencyTo que reunia as dependências que terminam nessa intentions.
	name	Nome fornecido ao intention (Goal, Softgoal, Task ou Resource)
	contributesFrom (opcional)	Concatenação de todos os valores xmi:id das tags contributions que apresentam o valor do atributo target idêntico ao valor do xmi:id desta intentions.
	contributesTo (opcional)	Concatenação de todos os valores xmi:id das tags contributions que apresentam o valor do atributo source idêntico ao valor do xmi:id desta intentions.
	parentDecompositions (opcional)	Atributo fornecido ao intentions onde chega no mínimo uma decomposition. Seu valor é igual a concatenação de todos os xmi:id de decompositions cujo atributo target apresenta valor igual ao xmi:id da tag intentions.

	decompositions (opcional)	Atributo fornecido ao intentions de onde parte no mínimo uma decomposition. Seu valor é igual a concatenação de todos os xmi:id de decompositions cujo atributo source apresenta valor igual ao xmi:id da tag intentions.
contributions	xmi:type	edu.toronto.cs.openome_model:MakeContribution
		edu.toronto.cs.openome_model:SomePlusContribution
		edu.toronto.cs.openome_model:HelpContribution
		edu.toronto.cs.openome_model:UnknownContribution
		edu.toronto.cs.openome_model:HurtContribution
		edu.toronto.cs.openome_model:SomeMinusContribution
		edu.toronto.cs.openome_model:BreakContribution
		edu.toronto.cs.openome_model:AndContribution
		edu.toronto.cs.openome_model:OrContribution
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	target	Mesmo valor do atributo xmi:id da tag intentions referente ao destino da ligação. Ou seja onde chega a ligação.
	source	Mesmo valor do atributo xmi:id da tag intentions referente a origem da ligação. Ou seja de onde se origina a ligação.
dependencies	xmi:type	edu.toronto.cs.openome_model:Dependency
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	dependencyFrom	Mesmo valor do atributo xmi:id da tag intentions ou compartments referente ao destino da ligação. Ou seja onde chega a ligação.
	dependencyTo	Mesmo valor do atributo xmi:id da tag intentions ou compartments referente a origem da ligação. Ou seja de onde se origina a ligação.
decompositions	xmi:type	edu.toronto.cs.openome_model:AndDecomposition
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	source	Mesmo valor do atributo xmi:id da tag intentions referente a origem da ligação. Ou seja de onde se

		origina a ligação.
	target	Mesmo valor do atributo xmi:id da tag intentions referente ao destino da ligação. Ou seja onde chega a ligação.
containers	xmi:type	edu.toronto.cs.openome_model:Actor
		edu.toronto.cs.openome_model:Agent
		edu.toronto.cs.openome_model:Position
		edu.toronto.cs.openome_model:Role
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	dependencyTo (opcional)	Atributo que está presente apenas quando o xmi:id da containers é igual ao valor do atributo dependencyFrom de pelo menos uma tag dependencies. O valor desse atributo é igual ao conjunto de valores xmi:id de todas as dependencies que terminam nessa containers, separadas por um espaço em branco.
	dependencyFrom (opcional)	Análogo ao atributo dependencyTo, porém o valor desse atributo é igual ao conjunto de valores xmi:id de todas as dependencies que começam nessa containers, separadas por um espaço em branco. Diferentemente do dependencyTo que reunia as dependências que terminam nessa containers.
	name	Nome fornecido ao container (Actor, Agent, Role ou Position).
associations	xmi:type	edu.toronto.cs.openome_model:IsAAssociation
		edu.toronto.cs.openome_model:CoversAssociation
		edu.toronto.cs.openome_model:IsPartOfAssociation
		edu.toronto.cs.openome_model:OccupiesAssociation
		edu.toronto.cs.openome_model:PlaysAssociation
		edu.toronto.cs.openome_model:INSAssociation
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	source	Mesmo valor do atributo xmi:id da tag containers referente a origem da ligação. Ou seja de onde se origina a ligação.
	target	Mesmo valor do atributo xmi:id da tag containers referente ao destino da ligação. Ou seja onde chega a ligação.

children	xmi:type	notation:Shape
		notation:DecorationNode
		notation:BasicCompartment
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	type	Ver Tabela 7.2 - Possíveis valores para o atributo type da tag <children>
	element	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag e apresenta o mesmo valor que o xmi:id de alguma tag filha direta de <edu.toronto.cs.openome_model:Model>. Ou seja essa tag children corresponde a algum elemento declarado anteriormente no arquivo XML. A referência entre eles é feita através do valor desse atributo element.
	fontName	Segoe UI
styles	xmi:type	notation:SortingStyle
		notation:FilteringStyle
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	fontName	Segoe UI
layoutConstraint	xmi: type	notation:Bounds
	xmi: id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	x	Valor correspondente a coordenada X do elemento no diagrama. Serve para posicionar visualmente o mesmo.
	y	Valor correspondente a coordenada Y do elemento no diagrama. Serve para posicionar visualmente o mesmo.
	width	Valor correspondente a largura do elemento no diagrama.
	height	Valor correspondente a altura do elemento no diagrama.
edges	xmi:type	notation:Connector
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	type	Ver Tabela 7.3 - Possíveis valores para o atributo type da tag <edges>

	element	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag e apresenta o mesmo valor que o xmi:id de alguma tag filha direta de <edu.toronto.cs.openome_model:Model>. Ou seja essa tag edges corresponde a algum elemento declarado anteriormente no arquivo XML. A referência entre eles é feita através do valor desse atributo element.
	source	Mesmo valor que o atributo xmi:id da tag children correspondente ao elemento de onde parte a ligação.
	target	Mesmo valor que o atributo xmi:id da tag children correspondente ao elemento onde chega a ligação.
bendpoints	xmi:type	notation:RelativeBendpoints
	xmi:id	String formada por 23 caracteres aleatórios. É utilizada como identificação única da tag.
	points	Coordenadas que determinam o trajeto da ligação. Ex.: [123, -25, -307, -20]$[326, -114, -104, -109]

7.3. Possíveis valores para o atributo type da tag <children>

Tabela 7.2 - Possíveis valores para o atributo type da tag <children> de um ood

Valores	**Descrição**
2001, 5005 e 7001	O conjunto formado por uma tag “children” com atributo type de valor “2001” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5005” e “7001” representa na ferramenta OpenOME um compartment do tipo Actor com ou sem elementos internos.
2002, 5010 e 7002	O conjunto formado por uma tag “children” com atributo type de valor “2002” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5010” e “7002” representa na ferramenta OpenOME um compartment do tipo Agent com ou sem elementos internos.
2003, 5015 e 7003	O conjunto formado por uma tag “children” com atributo type de valor “2003” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5015” e “7003” representa na ferramenta OpenOME um compartment do tipo Position com ou sem elementos internos.
2004, 5020 e 7004	O conjunto formado por uma tag “children” com atributo type de valor “2004” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5020” e “7004” representa na ferramenta OpenOME um compartment do tipo Role com ou sem elementos internos.

2005, 5021 e 5041	O conjunto formado por uma tag “children” com atributo type de valor “2005” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5021” e “5041” representa na ferramenta OpenOME uma intention do tipo Goal externa, ou seja que não pertence a algum compartment.
2006, 5022 e 5042	O conjunto formado por uma tag “children” com atributo type de valor “2006” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5022” e “5042” representa na ferramenta OpenOME uma intention do tipo Softgoal externa, ou seja que não pertence a algum compartment.
2007, 5023 e 5043	O conjunto formado por uma tag “children” com atributo type de valor “2007” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5023” e “5043” representa na ferramenta OpenOME uma intention do tipo Task externa, ou seja que não pertence a algum compartment.
2008, 5024 e 5044	O conjunto formado por uma tag “children” com atributo type de valor “2008” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5024” e “5044” representa na ferramenta OpenOME uma intention do tipo Resource externa, ou seja que não pertence a algum compartment.
3001, 5001 e 5025	O conjunto formado por uma tag “children” com atributo type de valor “3001” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5001” e “5025” representa na ferramenta OpenOME uma intention do tipo Goal interna, ou seja que pertence a algum compartment do tipo Actor.
3002, 5002 e 5026	O conjunto formado por uma tag “children” com atributo type de valor “3002” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5002” e “5026” representa na ferramenta OpenOME uma intention do tipo Softgoal interna, ou seja que pertence a algum compartment do tipo Actor.
3003, 5003 e 5027	O conjunto formado por uma tag “children” com atributo type de valor “3003” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5003” e “5027” representa na ferramenta OpenOME uma intention do tipo Resource interna, ou seja que pertence a algum compartment do tipo Actor.
3004, 5004 e 5028	O conjunto formado por uma tag “children” com atributo type de valor “3004” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5004” e “5028” representa na ferramenta OpenOME uma intention do tipo Task interna, ou seja que pertence a algum compartment do tipo Actor.
3005, 5006 e 5029	O conjunto formado por uma tag “children” com atributo type de valor “3005” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5006” e “5029” representa na ferramenta OpenOME uma intention do tipo Goal interna, ou seja que pertence a algum compartment do tipo Agent.
3006, 5007 e 5030	O conjunto formado por uma tag “children” com atributo type de valor “3006” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5007” e “5030” representa na ferramenta OpenOME uma intention do tipo

	Softgoal interna, ou seja que pertence a algum compartment do tipo Agent.
3007, 5008 e 5031	O conjunto formado por uma tag “children” com atributo type de valor “3007” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5008” e “5031” representa na ferramenta OpenOME uma intention do tipo Resource interna, ou seja que pertence a algum compartment do tipo Agent.
3008, 5009 e 5032	O conjunto formado por uma tag “children” com atributo type de valor “3008” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5009” e “5032” representa na ferramenta OpenOME uma intention do tipo Task interna, ou seja que pertence a algum compartment do tipo Agent.
3009, 5011 e 5033	O conjunto formado por uma tag “children” com atributo type de valor “3009” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5011” e “5033” representa na ferramenta OpenOME uma intention do tipo Goal interna, ou seja que pertence a algum compartment do tipo Position.
3010, 5012 e 5034	O conjunto formado por uma tag “children” com atributo type de valor “3010” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5012” e “5034” representa na ferramenta OpenOME uma intention do tipo Softgoal interna, ou seja que pertence a algum compartment do tipo Position.
3011, 5013 e 5035	O conjunto formado por uma tag “children” com atributo type de valor “3011” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5013” e “5035” representa na ferramenta OpenOME uma intention do tipo Resource interna, ou seja que pertence a algum compartment do tipo Position.
3012, 5014 e 5036	O conjunto formado por uma tag “children” com atributo type de valor “3012” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5014” e “5036” representa na ferramenta OpenOME uma intention do tipo Task interna, ou seja que pertence a algum compartment do tipo Position.
3013, 5016 e 5037	O conjunto formado por uma tag “children” com atributo type de valor “3013” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5016” e “5037” representa na ferramenta OpenOME uma intention do tipo Goal interna, ou seja que pertence a algum compartment do tipo Role.
3014, 5017 e 5038	O conjunto formado por uma tag “children” com atributo type de valor “3014” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5017” e “5038” representa na ferramenta OpenOME uma intention do tipo Softgoal interna, ou seja que pertence a algum compartment do tipo Role.
3015, 5018 e 5039	O conjunto formado por uma tag “children” com atributo type de valor “3015” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5018” e “5039” representa na ferramenta OpenOME uma intention do tipo Resource interna, ou seja que pertence a algum compartment do tipo Role.
3016, 5019 e	O conjunto formado por uma tag “children” com atributo type de valor “3016” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores

5040	“5019” e “5040” representa na ferramenta OpenOME uma intention do tipo Task interna, ou seja que pertence a algum compartment do tipo Role.
4005 e 6004	O conjunto formado por uma tag “edges” com atributo type de valor “4005” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6004” representa na ferramenta OpenOME uma contribuição do tipo HELP.
4006 e 6005	O conjunto formado por uma tag “edges” com atributo type de valor “4006” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6005” representa na ferramenta OpenOME uma contribuição do tipo HURT.
4007 e 6006	O conjunto formado por uma tag “edges” com atributo type de valor “4007” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6006” representa na ferramenta OpenOME uma contribuição do tipo MAKE.
4008 e 6007	O conjunto formado por uma tag “edges” com atributo type de valor “4008” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6007” representa na ferramenta OpenOME uma contribuição do tipo BREAK.
4009 e 6008	O conjunto formado por uma tag “edges” com atributo type de valor “4009” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6008” representa na ferramenta OpenOME uma contribuição do tipo SOMEPLUS.
4010 e 6009	O conjunto formado por uma tag “edges” com atributo type de valor “4010” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6009” representa na ferramenta OpenOME uma contribuição do tipo SOMEMINUS.
4011 e 6010	O conjunto formado por uma tag “edges” com atributo type de valor “4011” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6010” representa na ferramenta OpenOME uma contribuição do tipo UNKNOWN.
4012 e 6011	O conjunto formado por uma tag “edges” com atributo type de valor “4012” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6011” representa na ferramenta OpenOME uma contribuição do tipo AND.
4013 e 6012	O conjunto formado por uma tag “edges” com atributo type de valor “4013” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6012” representa na ferramenta OpenOME uma contribuição do tipo OR.
4014 e 6013	O conjunto formado por uma tag “edges” com atributo type de valor “4014” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6013” representa na ferramenta OpenOME uma associação do tipo ISA.
4015 e 6014	O conjunto formado por uma tag “edges” com atributo type de valor “4015” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6014” representa na ferramenta OpenOME uma associação do tipo COVERS.
4016 e 6015	O conjunto formado por uma tag “edges” com atributo type de valor “4016” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6015” representa na ferramenta OpenOME uma associação do tipo OCCUPIES.
4017 e	O conjunto formado por uma tag “edges” com atributo type de valor “4017” e

6016	uma tag filha direta “children” cujo atributo “type” apresenta valor “6016” representa na ferramenta OpenOME uma associação do tipo ISPARTOF.
4018 e 6017	O conjunto formado por uma tag “edges” com atributo type de valor “4018” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6017” representa na ferramenta OpenOME uma associação do tipo PLAYS.
4019 e 6018	O conjunto formado por uma tag “edges” com atributo type de valor “4019” e uma tag filha direta “children” cujo atributo “type” apresenta valor “6018” representa na ferramenta OpenOME uma associação do tipo INS.

7.4. Possíveis valores para o atributo type da tag <edges> na ferramenta OpenOME

Cada tag edges corresponde a algum filho direto da tag “edu.toronto.cs.openome_model:Model”. A referência é feita pelo valor do atributo element da tag edges e o valor do atributo xmi:id da tag filha de edu.toronto.cs.openome_model:Model. Para cada valor do atributo type de uma tag edges existe o correspondente valor do atributo xmi:type da tag filha direta de edu.toronto.cs.openome_model:Model. Essa correspondência deve ser respeitada para que se obtenha um diagrama reconhecido pela ferramenta.

Tabela 7.3 - Possíveis valores para o atributo type da tag <edges> de um ood

Valor	Tag correspondente	Valor do atributo xmi:type
4001	dependencies	edu.toronto.cs.openome_model:Dependency
4002	decompositions	edu.toronto.cs.openome_model:AndDecomposition
4003	decompositions	edu.toronto.cs.openome_model:OrDecomposition
4005	contributions	edu.toronto.cs.openome_model:HelpContribution
4006	contributions	edu.toronto.cs.openome_model:HurtContribution
4007	contributions	edu.toronto.cs.openome_model:MakeContribution
4008	contributions	edu.toronto.cs.openome_model:BreakContribution
4009	contributions	edu.toronto.cs.openome_model:SomePlusContribution
4010	contributions	edu.toronto.cs.openome_model:SomeMinusContribution
4011	contributions	edu.toronto.cs.openome_model:UnknownContribution
4012	contributions	edu.toronto.cs.openome_model:AndContribution
4013	contributions	edu.toronto.cs.openome_model:OrContribution
4014	associations	edu.toronto.cs.openome_model:IsAAssociation
4015	associations	edu.toronto.cs.openome_model:CoversAssociation
4016	associations	edu.toronto.cs.openome_model:OccupiesAssociation
4017	associations	edu.toronto.cs.openome_model:IsPartOfAssociation
4018	associations	edu.toronto.cs.openome_model:PlaysAssociation
4019	associations	edu.toronto.cs.openome_model:INSAssociation

8. APÊNDICE B - Estudo do XMI de um diagrama i* no iStar2BPMN

Todo diagrama i* na ferramenta iStar2BPMN consiste em dois arquivos: um arquivo de extensão “.istar” e outro de extensão “.istardiagram”. Escritos em XML, ambos devem estar presentes no mesmo diretório para que juntos formem um diagrama na ferramenta iStar2BPMN.

No XML do arquivo “.istardiagram” é possível encontrar referências ao arquivo “.istar”, portanto renomear o arquivo de extensão “.istar” sem corrigir o XML do “.istardiagram” pode implicar em erro na ferramenta. Algumas tags e atributos são opcionais, ou seja sua retirada do arquivo XML mantém o mesmo ainda reconhecível pela ferramenta. Entretanto há atributos que se ausentes impedem a correta leitura do arquivo.

8.1. Estrutura Geral de um arquivo “.istardiagram”

A ordem em que os elementos aparecem no XML do arquivo ”.istardiagram” deve ser respeitada de acordo com a listagem abaixo. Obs.: Alguns atributos foram removidos para facilitar a visualização.

8.1.1.Linha inicial

Linha inicial do arquivo “.istardiagram”:

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

8.1.2.Nó raíz

Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama. Apresenta obrigatoriamente como filhos diretos uma tag “styles” com xmi:type="notation:DiagramStyle" e xmi:id formado por uma string de 23 caracteres aleatórios, e uma tag “element” com atributos xmi:type="IstarModel:IstarModel" e href= nome do diagrama + ".istar#/". É nesse atributo href que se faz referência ao arquivo “.istar”.

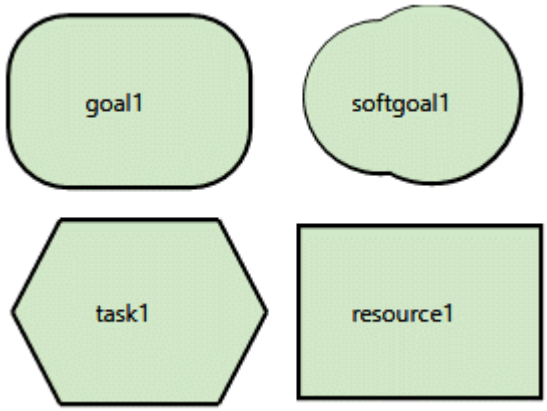
```
<notation:Diagram>
```

8.1.3.Element

O bloco a seguir pode aparecer zero ou mais vezes, e representa um element (Goal, Softgoal, Task, Resource) no diagrama que não pertence a nenhum compartment. O conjunto formado pela tag “children” com atributo type de valor “2001”, uma tag filha “children” de type “5001”, seguida pelas tags “element” e “layoutConstraint” representa um element. A Figura 8.1 ilustra os tipos de elements existentes. A diferenciação entre tipos se dá no XML do arquivo “.istar”.

<children xmi:type="notation:Shape" type="**2001**"> <children xmi:type="notation:DecorationNode" type="**5001**"/> <element xmi:type="IstarModel:IstarElement"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Element

Figura 8.1 – Elements na ferramenta iStar2BPMN



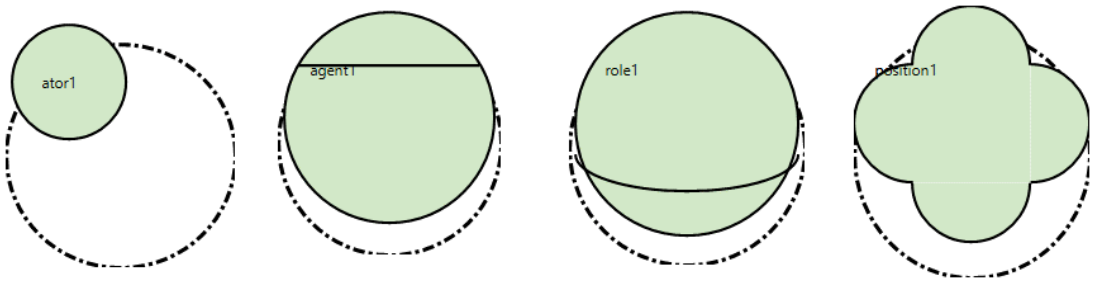
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.4.Compartment

O bloco a seguir pode aparecer zero ou mais vezes, e representa um compartment (Actor, Agent, Role, Position) no diagrama. O conjunto formado pela tag “children” com atributo type de valor “2002”, duas tags filhas “children” de type “5002” e “7001”, seguidas pelas tags “styles”, “element” e “layoutConstraint” representa um compartment. A Figura 8.2 ilustra os tipos de compartments existentes. A diferenciação entre tipos se dá no XML do arquivo “.istar”.

<children xmi:type="notation:Shape" type="**2002**"> <children xmi:type="notation:DecorationNode" type="**5002**"/> <children xmi:type="notation:BasicCompartment" type="**7001**"/> <styles xmi:type="notation:HintedDiagramLinkStyle"/> <element xmi:type="IstarModel:IstarCompartment"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment

Figura 8.2 - Compartments na ferramenta iStar2BPMN



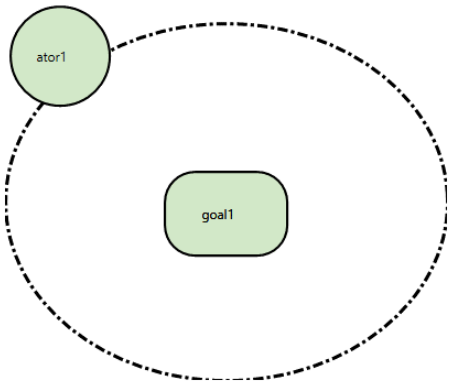
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.5.Compartment com element interno

O bloco a seguir pode aparecer zero ou mais vezes, sempre como filho direto da tag <notation:Diagram>, e representa um compartment (Actor, Agent, Role, Position) com apenas um element interno a ele (Goal, Softgoal, Task, Resource) no diagrama. O conjunto formado pela tag “children” com atributo type de valor “2002”, duas tags filhas “children” de type “5002” e “7001”, seguidas pelas tags “styles”, “element” e “layoutConstraint” representa um compartment. Enquanto o conjunto formado por uma tag “children” de type “3001” cujas tags filhas são uma tag “children” de type “5003”, uma tag “element” e uma “layoutConstraint” representa um element interno. A Figura 8.3 ilustra a situação.

<children xmi:type="notation:Shape" type="**2002**"> <children xmi:type="notation:DecorationNode" type="**5002**"/> <children xmi:type="notation:BasicCompartment" type="**7001**">	Compartment Início
<children xmi:type="notation:Shape" type="**3001**"> <children xmi:type="notation:DecorationNode" type="**5003**"/> <element xmi:type="IstarModel:IstarElement"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Element interno
</children> <styles xmi:type="notation:HintedDiagramLinkStyle"/> <element xmi:type="IstarModel:IstarCompartment"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment Fim

Figura 8.3 - Element interno a Compartment na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

- As duas tags a seguir devem aparecer no XML obrigatoriamente após todos os compartments e elements do diagrama. No atributo href da tag <element> nota-se a referência que se faz no arquivo ".istardiagram" ao arquivo ".istar".

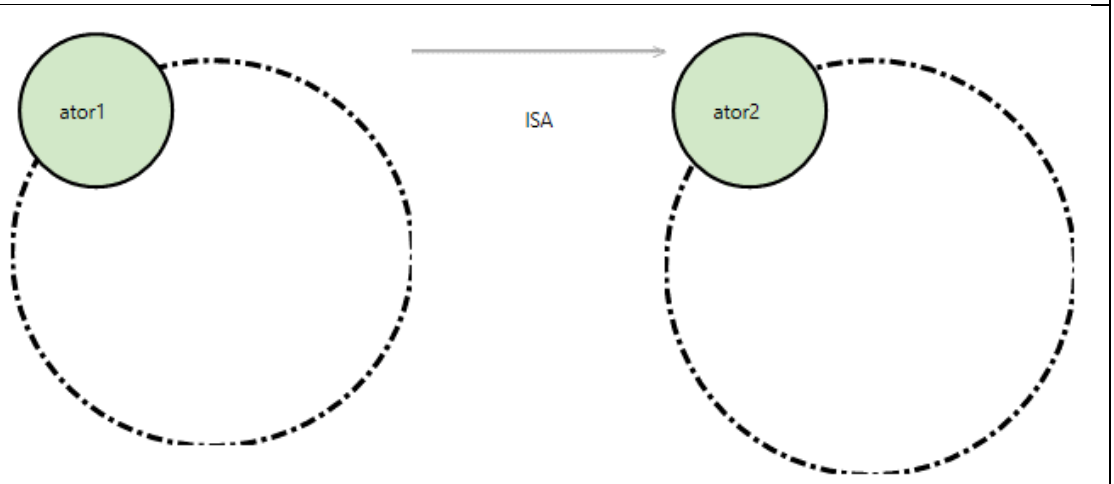
```
<styles xmi:type="notation:DiagramStyle" xmi:id="_p97K0fDPEeSWd4Dhn1BrBw"/>
<element xmi:type="IstarModel:IstarModel" href="nome do arquivo.istar#/"/>
```

8.1.6. Actor association

O bloco a seguir pode estar presente zero ou mais vezes e representa uma actor association. O conjunto formado pela tag "edges" com atributo type de valor "4001" e suas tags filhas: uma tag "children" com atributo type de valor "6001", uma tag "styles", uma tag "element", uma tag "bendpoints" e as opcionais "sourceAnchor" e "targetAnchor" representa uma associação entre dois compartments. A diferenciação entre tipos de associações se dá no XML do arquivo ".istar". A Figura 8.4 ilustra uma associação do tipo ISA.

<edges xmi:type="notation:Connector" type="**4001**" source="_rjah8PDPEeSWd4Dhn1BrBw" target="_s-pJ0PDPEeSWd4Dhn1BrBw"> <children xmi:type="notation:DecorationNode" type="**6001**"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarActorLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/>	Actor association
<sourceAnchor xmi:type="notation:IdentityAnchor"/> <targetAnchor xmi:type="notation:IdentityAnchor"/>	Tags opcionais
</edges>	

Figura 8.4 - Actor association na ferramenta iStar2BPMN



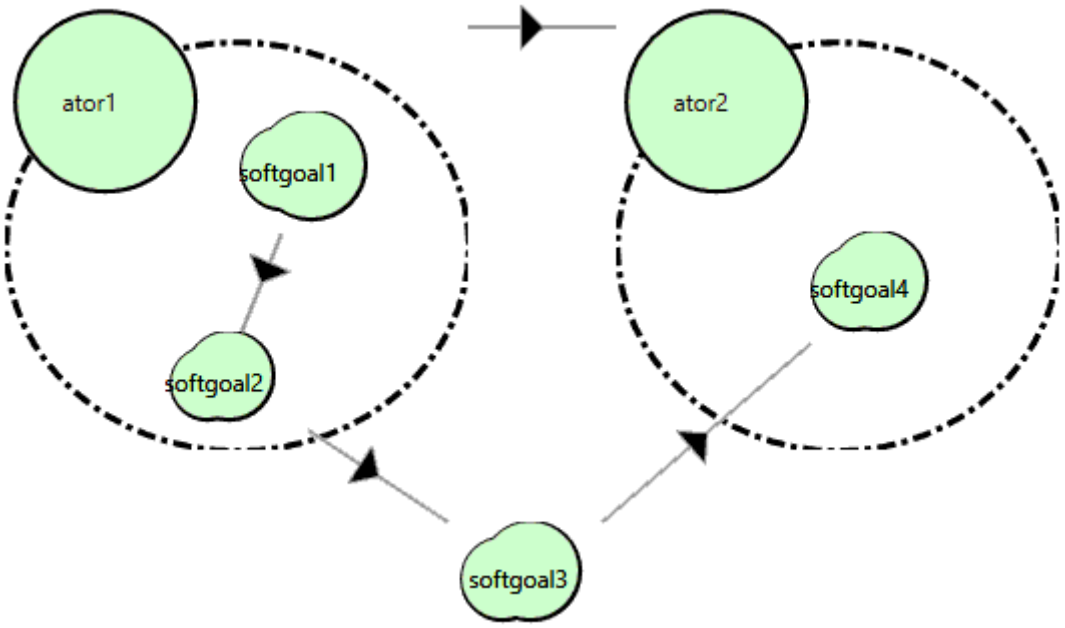
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.7.Dependency link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de dependência ou dependency link. O conjunto formado pela tag “edges” com atributo type de valor “4002” e suas tags filhas: uma tag “styles”, uma tag “element”, uma tag “bendpoints” e as opcionais “sourceAnchor” e “targetAnchor” representa uma ligação de dependência entre dois compartments ou dois elements distintos. A diferenciação entre tipos de dependências se dá no XML do arquivo “.istar”. A Figura 8.5 ilustra uma ligação de dependência.

<edges xmi:type="notation:Connector" type="**4002**" source="_gG3rgPDiEeSb8NZPdPkW0g" target="_hRHREPDiEeSb8NZPdPkW0g"> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarDependencyLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/>	Dependency link
<sourceAnchor xmi:type="notation:IdentityAnchor"/> <targetAnchor xmi:type="notation:IdentityAnchor"/>	Tags opcionais
</edges>	

Figura 8.5 - Dependency links na ferramenta iStar2BPMN



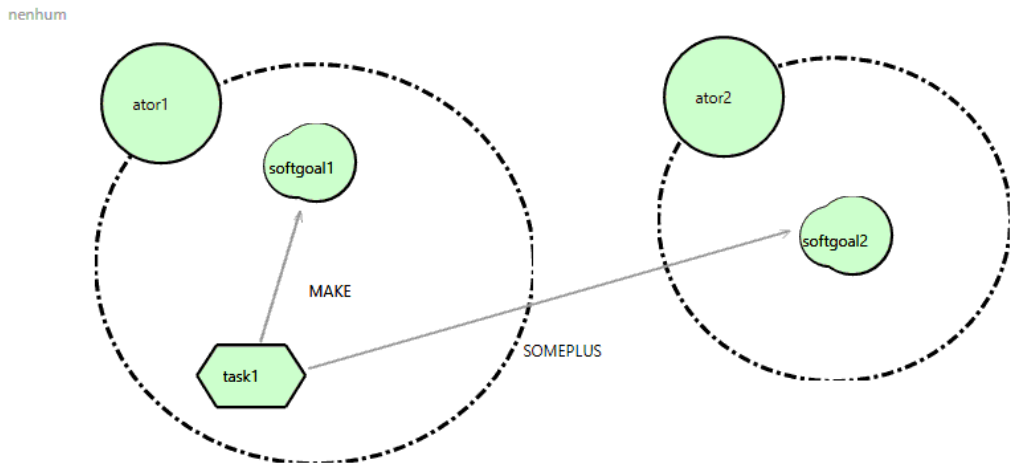
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.8.Contribution link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de contribuição ou contribution link. O conjunto formado pela tag “edges” com atributo type de valor “4005” e suas tags filhas: uma tag “children” com atributo type de valor “6005”, uma tag “styles”, uma tag “element”, uma tag “bendpoints” e as opcionais “sourceAnchor” e “targetAnchor” representa uma ligação de contribuição entre dois elements. A diferenciação entre tipos de contribuições se dá no XML do arquivo “.istar”. A Figura 8.6 ilustra ligações de contribuição.

<edges xmi:type="notation:Connector" type="**4005**" source="_rjah8PDPEeSWd4Dhn1BrBw" target="_s- pJ0PDPEeSWd4Dhn1BrBw"> <children xmi:type="notation:DecorationNode" type="**6005**"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarContributionLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/>	Contribution link
<sourceAnchor xmi:type="notation:IdentityAnchor"/> <targetAnchor xmi:type="notation:IdentityAnchor"/>	Tags opcionais
</edges>	

Figura 8.6 - Contribution links na ferramenta iStar2BPMN



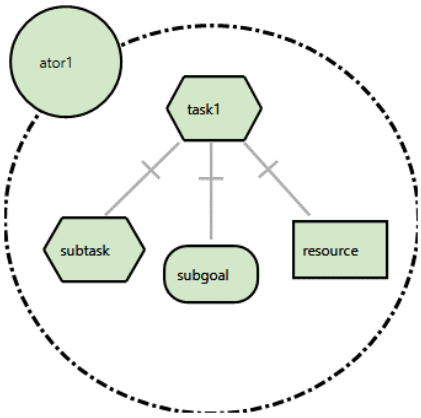
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.9.Task Decomposition link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de decomposição de tarefa ou task decomposition link. O conjunto formado pela tag “edges” com atributo type de valor “4004” e suas tags filhas: uma tag “styles”, uma tag “element”, uma tag “bendpoints” e as opcionais “sourceAnchor” e “targetAnchor” representa uma ligação de decomposição entre dois elements. A Figura 8.7 ilustra ligações de decomposição.

<edges xmi:type="notation:Connector" type="**4004**" source="_rjah8PDPEeSWd4Dhn1BrBw" target="_s-pJ0PDPEeSWd4Dhn1BrBw"> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarTaskDecomposition**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/>	Task Decomposition Link
<sourceAnchor xmi:type="notation:IdentityAnchor"/> <targetAnchor xmi:type="notation:IdentityAnchor"/>	Tags opcionais
</edges>	

Figura 8.7 - Task Decomposition Links na ferramenta iStar2BPMN



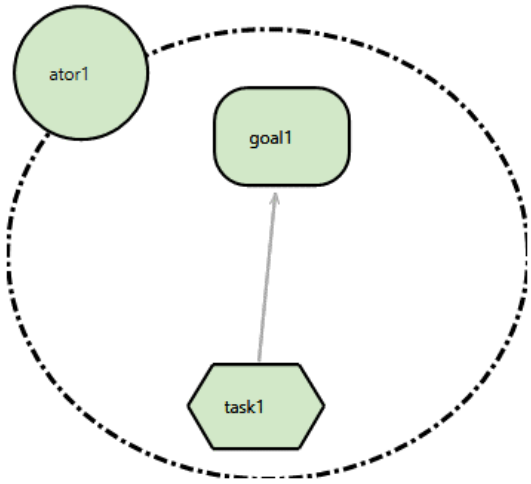
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.1.10. Means-end link

O bloco a seguir pode estar presente zero ou mais vezes e representa uma ligação de means-end. O conjunto formado pela tag “edges” com atributo type de valor “4003” e suas tags filhas: uma tag “styles”, uma tag “element”, uma tag “bendpoints” e as opcionais “sourceAnchor” e “targetAnchor” representa uma ligação de means-end entre dois elements. A Figura 8.8 ilustra uma ligação de means-end.

<edges xmi:type="notation:Connector" type="**4003**" source="_rjah8PDPEeSWd4Dhn1BrBw" target="_s-pJ0PDPEeSWd4Dhn1BrBw"> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarMeanEndLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/>	Means-End Link
<sourceAnchor xmi:type="notation:IdentityAnchor"/> <targetAnchor xmi:type="notation:IdentityAnchor"/>	Tags opcionais
</edges>	

Figura 8.8 - Means-End Link na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

8.2. Listagem completa de tags e atributos

A relação entre cada uma das tags de um ".istardiagram" e seus atributos segue abaixo.

Tabela 8.1 - Listagem completa de tags e atributos de um istardiagram

Tag	Atributo	Valor
xml	version	1.0
	encoding	UTF-8
notation:Diagram	xmi:version	2.0
	xmlns:xmi	http://www.omg.org/XMI
	xmlns:IstarModel	IstarModel
	xmlns:notation	http://www.eclipse.org/gmf/runtime/1.0.2/notation
	xmi:id	String de 23 caracteres aleatórios
	type	Top
	name	Nome do diagrama + ".istardiagram"
	measurementUnit	Pixel
children	xmi:type	notation:Shape
		notation:DecorationNode
		notation:BasicCompartment
	xmi:id	String de 23 caracteres aleatórios
	type	**Tabela 8.2 - Possíveis valores para o atributo type da tag <children>**
	fontName	Segoe UI
element	xmi:type	IstarModel:IstarElement
		IstarModel:IstarCompartment

		IstarModel:IstarActorLink
		IstarModel:IstarContributionLink
		IstarModel:IstarDependencyLink
		IstarModel:IstarTaskDecomposition
		IstarModel:IstarMeanEndLink
	href	**Tabela 8.3 - Possíveis valores para o atributo href da tag <element>**
layoutConstraint	xmi:type	notation:Bounds
		notation:Location
	xmi:id	String de 23 caracteres aleatórios
	x	Valor numérico que indica uma posição x no diagrama
	y	Valor numérico que indica uma posição y no diagrama
	width	Valor numérico que indica a largura do elemento
	height	Valor numérico que indica a altura do elemento
styles	xmi:type	notation:HintedDiagramLinkStyle
		notation:FontStyle
	xmi:id	String de 23 caracteres aleatórios
	fontName	Segoe UI
edges	xmi:type	notation:Connector
	xmi:id	String de 23 caracteres aleatórios
	type	Tabela 8.4 - Possíveis valores para o atributo type da tag <edges>
	source	String de 23 caracteres aleatórios
	target	String de 23 caracteres aleatórios
bendpoints	xmi:type	notation:RelativeBendpoints
	xmi:id	String de 23 caracteres aleatórios
	points	Coordenadas que determinam o trajeto da ligação. Ex.: [123, -25, -307, -20]$[326, -114, -104, -109]
sourceAnchor (opcional)	xmi:type	notation:IdentityAnchor
	xmi:id	String de 23 caracteres aleatórios
	id	Par de valores decimais entre 0 e 1. Ex.: (0.2076271186440678,0.04048582995951417)
targetAnchor (opcional)	xmi:type	notation:IdentityAnchor
	xmi:id	String de 23 caracteres aleatórios
	id	Par de valores decimais entre 0 e 1. Ex.: (0.2076271186440678,0.04048582995951417)

8.3. Possíveis valores para o atributo type da tag <children>

Tabela 8.2 - Possíveis valores para o atributo type da tag <children> de um istardiagram	
Valores	**Descrição**
2002, 5002 e 7001	O conjunto formado por uma tag “children” com atributo type de valor “2002” e duas tags filhas diretas “children” cujos atributos “type” apresentam valores “5002” e “7001” representa na ferramenta iStar2BPMN um compartment (Actor, Agent, Role, Position) com ou sem elementos internos.
3001 e 5003	O conjunto formado por uma tag “children” com atributo type de valor “3001” e uma tag filha direta “children” cujo atributo “type” apresenta valor “5003” representa na ferramenta iStar2BPMN um element (Goal, Softgoal, Task, Resource) interno a algum compartment.
2001 e 5001	O conjunto formado por uma tag “children” com atributo type de valor “2001” e uma tag filha direta “children” cujo atributo “type” apresenta valor “5001” representa na ferramenta iStar2BPMN um element (Goal, Softgoal, Task, Resource) externo, ou seja não pertencente a algum compartment.

8.4. Possíveis valores para o atributo href da tag <element>

O atributo href faz referência a elementos do XML do arquivo “.istar”. Ele é composto por uma string que sempre inicia com o nome do diagrama seguido por “.istar#//” e algum dos valores na tabela abaixo. Os números X e Y são encontrados pela ordem em que aparecem no XML, sendo o primeiro a aparecer o número zero, o segundo o número um e assim sucessivamente. A contagem inicia em zero a partir do topo do documento XML e também a partir da primeira tag interna a alguma tag compartments.

Tabela 8.3 - Possíveis valores para o atributo href da tag <element> de um istardiagram

Valor	Descrição
@compartments.X	Referência ao compartment de número X.
@elements.X	Referência ao element de número X.
@compartments.X/@elements.Y	Referência ao element de número Y pertencente ao compartment de número X.
@actorLinks.X	Referência ao actor link de número X.
@dependencyLinks.X	Referência ao dependency link de número X.
@compartments.X/@contributionLinks.Y	Referência ao contribution link de número Y interno ao compartment de número X.
@compartments.X/@tasksDecompositions.Y	Referência a task decomposition de número Y interna ao compartment de número X.
@compartments.X/@meanEndLinks.Y	Referência ao means-end link de número Y

interno ao compartment de número X.

8.5. Possíveis valores para o atributo type da tag <edges>

Para cada valor do atributo type de uma tag “edges” existe o correspondente valor do atributo xmi:type da tag filha direta “element”. Essa correspondência deve ser respeitada para que se obtenha um diagrama reconhecido pela ferramenta.

Tabela 8.4 - Possíveis valores para o atributo type da tag <edges> de um istardiagram

Valor	Valor do atributo xmi:type da tag filha element
4001	IstarModel:IstarActorLink
4002	IstarModel:IstarDependencyLink
4003	IstarModel:IstarMeanEndLink
4004	IstarModel:IstarTaskDecomposition
4005	IstarModel:IstarContributionLink

8.6. Estrutura Geral de um arquivo “.istar”

A ordem em que os elementos aparecem no XML do arquivo ”.istar” deve ser respeitada de acordo com a listagem abaixo. Obs.: Alguns atributos foram removidos para facilitar a visualização, porém a lista completa de atributos e suas respectivas tags encontra-se na Tabela 8.5.

8.6.1.Linha inicial

Primeira linha de código XML do arquivo “.istar”:

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

8.6.2.Nó raíz

Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama.

```
<IstarModel:IstarModel>
```

8.6.3.Dependency link

A tag a seguir aparece zero ou mais vezes e corresponde a uma ligação de dependência. O atributo type indica qual o tipo da dependência. O mesmo só estará presente quando a dependência

não for do tipo COMMITTED. Esta tag deve estar presente antes (acima) da primeira tag “elements” do XML, se existente. Senão, ela aparece acima da primeira tag compartments.

```
<dependencyLinks type="CRITICAL" source="//@compartments.0/@elements.0"
target="//@compartments.0/@elements.1"/>
```

8.6.4.Actor association

A tag a seguir aparece zero ou mais vezes e corresponde a uma associação no diagrama entre dois compartments. O atributo type indica qual o tipo da associação. O mesmo só estará presente quando a associação não for do tipo ISA. Esta tag deve estar presente antes (acima) da primeira tag “elements” do XML, se existente. Senão, ela aparece acima da primeira tag compartments.

```
<actorLinks type="COVERS" source="//@compartments.0" target="//@compartments.1"/>
```

8.6.5.Element

A tag a seguir pode aparecer zero ou mais vezes, sempre como filho direto da tag <IstarModel:IstarModel>, e representa um element externo (Goal, Softgoal, Task, Resource) no diagrama, ou seja um element que não é interno a algum compartment. O atributo type indica qual o tipo do element. O mesmo só estará presente quando o element não for do tipo Goal.

<elements name="task1" type="TASK"/>	Element

8.6.6.Compartment

A tag a seguir pode aparecer zero ou mais vezes, sempre como filho direto da tag <IstarModel:IstarModel>, e representa um compartment (Actor, Agent, Role, Position) no diagrama. O atributo type indica qual o tipo do compartment. O mesmo só estará presente quando o compartment não for do tipo Actor.

<compartments name="agente1" type="AGENT">	Compartment

8.6.7.Compartment com element interno

O bloco a seguir pode aparecer zero ou mais vezes, sempre como filho direto da tag <IstarModel:IstarModel>, e representa um compartment (Actor, Agent, Role, Position) com um elemento interno (Goal, Softgoal, Task, Resource) no diagrama. Esse bloco pode vir antes ou depois do bloco anterior que representa um compartment vazio.

<compartments name="role1" type="ROLE">	Compartment
<elements name="softgoal1" type="SOFTGOAL"/>	Element
</compartments>	

8.6.8.Contribution link

No bloco a seguir está presente a tag contributionLinks que pode aparecer zero ou mais vezes, sempre como filho direto de alguma tag <compartments>, e representa uma ligação de contribuição entre elements no diagrama. O atributo type indica qual o tipo da contribuição. O mesmo só estará presente quando a contribuição não for do tipo MAKE.

<compartments name="ator1">	Compartment Início
<contributionLinks type="SOMEPLUS" source="//@compartments.0/@elements.0" target="//@compartments.0/@elements.1"/>	Contribution Link
<elements name="softgoal1" type="SOFTGOAL"/> <elements name="task1" type="TASK"/>	Elements internos
</compartments>	Compartment Fim

8.6.9.Task decomposition link

No bloco a seguir está presente a tag tasksDecompositions que pode aparecer zero ou mais vezes, sempre como filho direto de alguma tag <compartments>, e representa uma ligação de decomposição entre elements no diagrama.

<compartments name="ator1">	Compartment Início
<tasksDecompositions source="//@compartments.0/@elements.0" target="//@compartments.0/@elements.1"/>	Tasks Decompositions Link
<elements name="softgoal1" type="SOFTGOAL"/> <elements name="task1" type="TASK"/>	Elements internos
</compartments>	Compartment Fim

8.6.10. Means-end link

No bloco a seguir está presente a tag meanEndLinks que pode aparecer zero ou mais vezes, sempre como filho direto de alguma tag <compartments>, e representa uma ligação de Means-Ends entre elements no diagrama.

<compartments name="ator1">	Compartment Início
<meanEndLinks source="//@compartments.0/@elements.0" target="//@compartments.0/@elements.1"/>	Means-Ends Links
<elements name="softgoal1" type="SOFTGOAL"/> <elements name="task1" type="TASK"/>	Elements internos
</compartments>	Compartment Fim

8.7. Listagem completa de tags e atributos

A lista de todas as possíveis tags de um ".istar" e seus atributos segue abaixo.

Tabela 8.5 - Listagem completa de tags e atributos de um istar

Tag	Atributo	Valor
xml	version	1.0
	encoding	UTF-8
IstarModel:IstarModel	xmi:version	2.0
	xmlns:xmi	http://www.omg.org/XMI
	xmlns:IstarModel	IstarModel
elements	name	Nome fornecido ao element no diagrama
	type	SOFTGOAL
		TASK
		RESOURCE
compartments	name	Nome fornecido ao compartment no diagrama
	type	AGENT
		ROLE
		POSITION
dependencyLinks	type	OPEN
		CRITICAL
	source	Informa de qual element parte a ligação. Ex.: source="//@compartments.0/@elements.0"
	target	Informa em qual dos elements chega a ligação. Ex.: target="//@compartments.0/@elements.1"
actorLinks	type	COVERS
		ISPARTOF
		OCCUPIES
		PLAYS
		INS
	source	Informa de qual dos compartments parte a

		ligação. Ex.: source="//@compartments.0"
	target	Informa em qual dos compartments chega a ligação. Ex.: target="//@compartments.1"
contributionLinks	type	SOMEPLUS
		HELP
		UNKNOWN
		HURT
		SOMEMINUS
		BREAK
		AND
		OR
	source	Informa de qual element parte a ligação. Ex.: source="//@compartments.0/@elements.0"
	target	Informa em qual dos elements chega a ligação. Ex.: target="//@compartments.0/@elements.1"
tasksDecompositions	source	Informa de qual element parte a ligação. Ex.: source="//@compartments.0/@elements.0"
	target	Informa em qual dos elements chega a ligação. Ex.: target="//@compartments.0/@elements.1"
meanEndLinks	source	Informa de qual element parte a ligação. Ex.: source="//@compartments.0/@elements.0"
	target	Informa em qual dos elements chega a ligação. Ex.: target="//@compartments.0/@elements.1"

9. APÊNDICE C - Estudo do XMI de um diagrama BPMN no iStar2BPMN

Todo diagrama BPMN na ferramenta iStar2BPMN consiste em dois arquivos: um arquivo de extensão “.bpmn” e outro de extensão “.bpmndiagram”. Escritos em XML, ambos devem estar presentes no mesmo diretório para que juntos formem um diagrama na ferramenta iStar2BPMN.

No XML do arquivo “.bpmndiagram” é possível encontrar referências ao arquivo “.bpmn”, portanto renomear o arquivo de extensão “.bpmn” sem corrigir o XML do “.bpmndiagram” implica em erro na ferramenta. Algumas tags e atributos são opcionais, ou seja sua retirada do arquivo XML mantém o mesmo ainda reconhecível pela ferramenta. Entretanto há atributos que se ausentes impedem a correta leitura do arquivo seção.

9.1. Estrutura Geral de um arquivo bpmndiagram no iStar2BPMN

A ordem em que os elementos aparecem no XML do arquivo ”.bpmndiagram” deve ser respeitada de acordo com a listagem abaixo.

9.1.1.Linha Inicial

Linha inicial do arquivo “.bpmndiagram”.

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

9.1.2.Nó raíz

Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama. Apresenta obrigatoriamente como filhos diretos uma tag “styles” com xmi:type="notation:DiagramStyle" e xmi:id formado por uma string de 23 caracteres aleatórios, e uma tag “element” com atributos xmi:type="BPMNModel:BPMNModel" e href= nome do diagrama + ".bpmn#/". É nesse atributo href que se faz referência ao arquivo “.bpmn”.

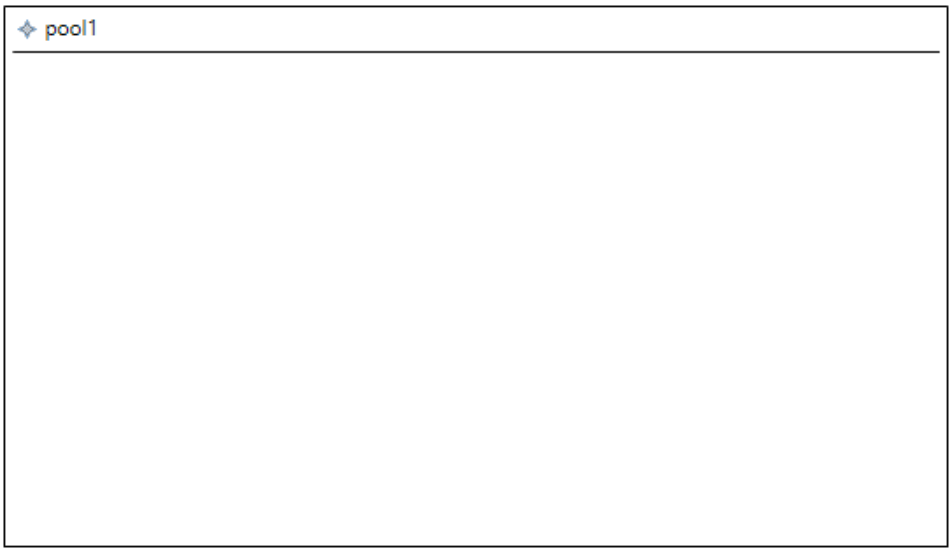
```
<notation:Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:BPMNModel="bpmnmodel"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation"
xmi:id="_w91ycP71EeSupLsaLvlLMw" type="Bpmnmodel" name="caso12_1.bpmndiagram"
measurementUnit="Pixel">
```

9.1.3.Pool

O bloco a seguir representa uma Pool no diagrama. O conjunto formado pela tag “children” com atributo type de valor “2001”, duas tags filhas “children” de type “5008” e “7001”, seguidas por quatro tags “styles”, uma “element” e uma “layoutConstraint” representa uma Pool, como ilustrado na Figura 9.1. Esse bloco deve ser adicionado como filho da tag “notation:Diagram” antes da primeira ocorrência de tag “styles”.

<children xmi:type="**notation:Node**" xmi:id="**_IFUL0P1rEeSkfvHvb38sxg**" type="**2001**"> <children xmi:type="**notation:DecorationNode**" xmi:id="**_IFUL1v1rEeSkfvHvb38sxg**" type="**5008**"/> <children xmi:type="**notation:BasicCompartment**" xmi:id="**_IFUL1_1rEeSkfvHvb38sxg**" type="**7001**"/> <styles xmi:type="**notation:DescriptionStyle**" xmi:id="**_IFUL0f1rEeSkfvHvb38sxg**"/> <styles xmi:type="**notation:FontStyle**" xmi:id="**_IFUL0v1rEeSkfvHvb38sxg**" fontName="**Segoe UI**"/> <styles xmi:type="**notation:FillStyle**" xmi:id="**_IFUL0_1rEeSkfvHvb38sxg**"/> <styles xmi:type="**notation:HintedDiagramLinkStyle**" xmi:id="**_IFUL1P1rEeSkfvHvb38sxg**"/> <element xmi:type="**BPMNModel:BPMNPool**" href="**caso0.bpmn#//@pools.0**"/> <layoutConstraint xmi:type="**notation:Bounds**" xmi:id="**_IFUL1f1rEeSkfvHvb38sxg**" x="**102**" y="**37**" width="**539**" height="**309**"/> </children>	Pool

Figura 9.1 – Pool na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.4.Lane

O bloco a seguir representa uma Lane no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3001”, duas tags filhas “children” de type “5007” e “7002”, seguidas por uma tag “styles”, uma “element” e uma “layoutConstraint” representa uma Lane, como ilustrado na Figura 9.2. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Pool. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7001”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_DxH9UP1sEeSkfvHvb38sxg"** type=**"3001"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_DxH9U_1sEeSkfvHvb38sxg"** type=**"5007"**/> <children xmi:type=**"notation:BasicCompartment"** xmi:id=**"_DxH9VP1sEeSkfvHvb38sxg"** type=**"7002"**/> <styles xmi:type=**"notation:HintedDiagramLinkStyle"** xmi:id=**"_DxH9Uf1sEeSkfvHvb38sxg"**/> <element xmi:type=**"BPMNModel:BPMNLane"** href=**"caso2.bpmn#//@pools.0/@lanes.0"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_DxH9Uv1sEeSkfvHvb38sxg"** y=**"3"** width=**"666"** height=**"151"**/> </children>	Lane

Figura 9.2 - Lane na ferramenta iStar2BPMN

Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.5.Start Event

O bloco a seguir representa um Start Event no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3003”, uma tag filha “children” de type “5002” contendo uma tag “layoutConstraint”, seguida por uma tag “element” e uma “layoutConstraint” representa um Start Event, como ilustrado na Figura 9.3. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja, será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type="**notation:Shape**" xmi:id="**_pzs6kP2CEeSxyOp2P4UWWA**" type="**3003**" fontName="**Segoe UI**"> <children xmi:type="**notation:DecorationNode**" xmi:id="**_pzs6kv2CEeSxyOp2P4UWWA**" type="**5002**"> <layoutConstraint xmi:type="**notation:Location**" xmi:id="**_pzs6k_2CEeSxyOp2P4UWWA**" x="**30**" y="**96**"/> </children> <element xmi:type="**BPMNModel:BPMNInitialEvent**" href="**caso3.bpmn#//@pools.0/@lanes.0/@nodes.0**"/> <layoutConstraint xmi:type="**notation:Bounds**" xmi:id="**_pzs6kf2CEeSxyOp2P4UWWA**" x="**25**" y="**18**" width="**71**" height="**76**"/> </children>	Start Event

Figura 9.3 - Start Event na ferramenta iStar2BPMN

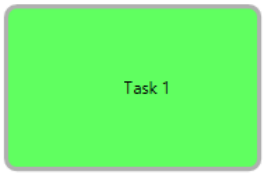
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.6.Task

O bloco a seguir representa uma Task no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3005”, uma tag filha “children” de type “5004”, seguida por uma tag “element” e uma “layoutConstraint” representa uma Task, como ilustrado na Figura 9.4. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type="**notation:Shape**" xmi:id="**_zRPDYP4vEeSOkIn8Fo6qdQ**" type="**3005**" fontName="**Segoe UI**"> <children xmi:type="**notation:DecorationNode**" xmi:id="**_zRPqcP4vEeSOkIn8Fo6qdQ**" type="**5004**"/> <element xmi:type="**BPMNModel:BPMNTask**" href="**caso4.bpmn#//@pools.0/@lanes.0/@nodes.1**"/> <layoutConstraint xmi:type="**notation:Bounds**" xmi:id="**_zRPDYf4vEeSOkIn8Fo6qdQ**" x="**205**" y="**18**" width="**111**" height="**71**"/> </children>	Task

Figura 9.4 - Task na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.7.Intermediate Event

O bloco a seguir representa um Intermediate Event no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3004”, uma tag filha “children” de type “5003” contendo uma tag “layoutConstraint”, seguida por uma tag “element” e uma “layoutConstraint” representa um Intermediate Event, como ilustrado na Figura 9.5. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_D_QJwP45EeSOkIn8Fo6qdQ"** type=**"3004"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_D_RX4P45EeSOkIn8Fo6qdQ"** type=**"5003"**> <layoutConstraint xmi:type=**"notation:Location"** xmi:id=**"_D_RX4f45EeSOkIn8Fo6qdQ"** y=**"5"**/> </children> <element xmi:type=**"BPMNModel:BPMNIntermediateEvent"** href=**"caso5.bpmn#//@pools.0/@lanes.0/@nodes.2"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_D_QJwf45EeSOkIn8Fo6qdQ"** x=**"375"** y=**"28"** width=**"61"** height=**"56"**/> </children>	Intermediate Event

Figura 9.5 - Intermediate Event na ferramenta iStar2BPMN

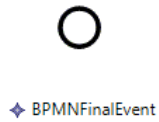
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.8.Final Event

O bloco a seguir representa um Final Event no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3002”, uma tag filha “children” de type “5001” contendo uma tag “layoutConstraint”, seguida por uma tag “element” e uma “layoutConstraint” representa um Final Event, como ilustrado na Figura 9.6. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_-2LsQP46EeSOkIn8Fo6qdQ"** type=**"3002"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_-2MTUP46EeSOkIn8Fo6qdQ"** type=**"5001"**> <layoutConstraint xmi:type=**"notation:Location"** xmi:id=**"_-2MTUf46EeSOkIn8Fo6qdQ"** y=**"5"**/> </children> <element xmi:type=**"BPMNModel:BPMNFinalEvent"** href=**"caso6.bpmn#//@pools.0/@lanes.0/@nodes.0"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_-2LsQf46EeSOkIn8Fo6qdQ"** x=**"70"** y=**"13"** width=**"111"** height=**"91"**/> </children>	Final Event

Figura 9.6 - Final Event na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.9.Gateway

O bloco a seguir representa um Gateway no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3007”, uma tag filha “children” de type “5006”, seguida por uma tag “element” e uma “layoutConstraint” representa um Gateway, como ilustrado na Figura 9.7. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_oV71oP4-EeSOkIn8Fo6qdQ"** type=**"3007"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_oV8csP4-**	Gateway

EeSOkIn8Fo6qdQ" type=**"5006"**/> <element xmi:type=**"BPMNModel:BPMNGateway"** href=**"caso7.bpmn#//@pools.0/@lanes.0/@nodes.0"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_oV71of4-EeSOkIn8Fo6qdQ"** x=**"235"** y=**"18"** width=**"91"** height=**"96"**/> </children>	

Figura 9.7 - Gateway na ferramenta iStar2BPMN

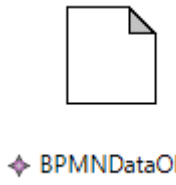
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.10. Data Object

O bloco a seguir representa um Data Object no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3008”, uma tag filha “children” de type “5009” contendo uma tag “layoutConstraint”, seguida por uma tag “element” e uma “layoutConstraint” representa um Data Object, como ilustrado na Figura 9.8. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_avcDYP5BEeSOkIn8Fo6qdQ"** type=**"3008"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_avcqcP5BEeSOkIn8Fo6qdQ"** type=**"5009"**> <layoutConstraint xmi:type=**"notation:Location"** xmi:id=**"_avcqcf5BEeSOkIn8Fo6qdQ"** y=**"5"**/> </children> <element xmi:type=**"BPMNModel:BPMNDataObject"** href=**"caso8.bpmn#//@pools.0/@lanes.0/@nodes.0"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_avcDYf5BEeSOkIn8Fo6qdQ"** x=**"165"** y=**"3"** width=**"166"** height=**"161"**/> </children>	Data Object

Figura 9.8 - Data Object na ferramenta iStar2BPMN



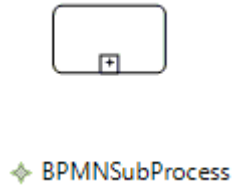
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.11. Sub-Process

O bloco a seguir representa um Sub-Process no diagrama. O conjunto formado pela tag “children” com atributo type de valor “3006”, uma tag filha “children” de type “5005” contendo uma tag “layoutConstraint”, seguida por uma tag “element” e uma “layoutConstraint” representa um Sub-Process, como ilustrado na Figura 9.9. Esse bloco deve ser adicionado como filho da terceira tag “children” de uma Lane. Ou seja será filho da tag cujo atributo “type” apresenta valor igual a “7002”.

<children xmi:type=**"notation:Shape"** xmi:id=**"_IU4tcP5eEeSuLItin-El6A"** type=**"3006"** fontName=**"Segoe UI"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_IU5UgP5eEeSuLItin-El6A"** type=**"5005"**> <layoutConstraint xmi:type=**"notation:Location"** xmi:id=**"_IU5Ugf5eEeSuLItin-El6A"** y=**"5"**/> </children> <element xmi:type=**"BPMNModel:BPMNSubProcess"** href=**"caso9.bpmn#//@pools.0/@lanes.0/@nodes.0"**/> <layoutConstraint xmi:type=**"notation:Bounds"** xmi:id=**"_IU4tcf5eEeSuLItin-El6A"** x=**"105"** y=**"6"** width=**"256"** height=**"136"**/> </children>	Sub-Process

Figura 9.9 - Sub-Process na ferramenta iStar2BPMN



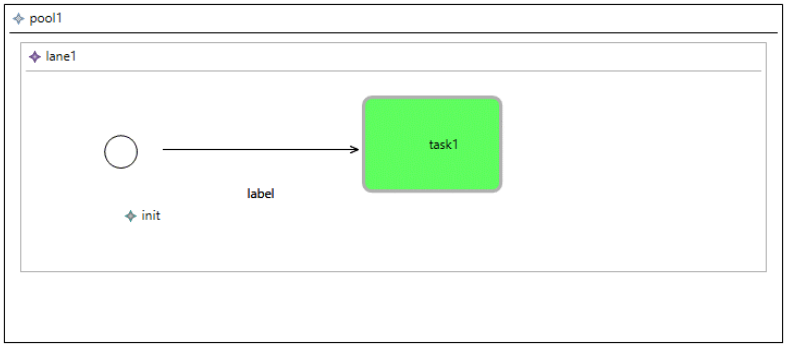
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.12. Sequence Flow

O bloco a seguir representa um Sequence Flow no diagrama. O conjunto formado pela tag “edges” com atributo type de valor “4003”, uma tag filha “children” de type “6002” contendo uma tag “layoutConstraint”, seguida por duas tags “styles”, uma tag “element” e uma “bendpoints” representa um Sequence Flow, como ilustrado na Figura 9.10. Esse bloco deve ser adicionado como filho da tag “notation:Diagram” após a ocorrência de tag “element” filha direta de “notation:Diagram”.

<edges xmi:type=**"notation:Edge"** xmi:id=**"_Zm6nMP5kEeSuLItin-El6A"** type=**"4003"** source=**"_XxbsQP5kEeSuLItin-El6A"** target=**"_YdQQcP5kEeSuLItin-El6A"**> <children xmi:type=**"notation:DecorationNode"** xmi:id=**"_Zm6nNP5kEeSuLItin-El6A"** type=**"6002"**> <layoutConstraint xmi:type=**"notation:Location"** xmi:id=**"_Zm6nNf5kEeSuLItin-El6A"** y=**"40"**/> </children> <styles xmi:type=**"notation:RoutingStyle"** xmi:id=**"_Zm6nMf5kEeSuLItin-El6A"**/> <styles xmi:type=**"notation:FontStyle"** xmi:id=**"_Zm6nMv5kEeSuLItin-El6A"** fontName=**"Segoe UI"**/> <element xmi:type=**"BPMNModel:BPMNSequenceFlow"** href=**"caso10_1.bpmn#//@pools.0/@lanes.0/@sequenceFlows.0"**/> <bendpoints xmi:type=**"notation:RelativeBendpoints"** xmi:id=**"_Zm6nM_5kEeSuLItin-El6A"** points=**"[38, -2, -205, 9]$[217, -14, -26, -3]"**/> <targetAnchor xmi:type=**"notation:IdentityAnchor"** xmi:id=**"_ZnZlUP5kEeSuLItin-El6A"** id=**"(0.19117647058823528,0.4583333333333333)"**/> </edges>	Sequence Flow

Figura 9.10 - Sequence Flow na ferramenta iStar2BPMN



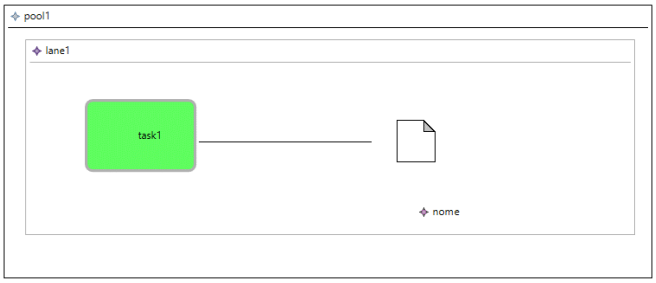
Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.1.13. Association

O bloco a seguir representa uma Association no diagrama. O conjunto formado pela tag “edges” com atributo type de valor “4001”, seguida por duas tags “styles”, uma tag “element” e uma “bendpoints” representa uma Association, como ilustrado na Figura 9.11. Esse bloco deve ser adicionado como filho da tag “notation:Diagram” após a ocorrência de tag “element” filha direta de “notation:Diagram”.

<edges xmi:type=**"notation:Edge"** xmi:id=**"_fhuCoP5tEeSuLItin-El6A"** type=**"4001"** source=**"_dhc0AP5tEeSuLItin-El6A"** target=**"_eSCjcP5tEeSuLItin-El6A"**> <styles xmi:type=**"notation:RoutingStyle"** xmi:id=**"_fhuCof5tEeSuLItin-El6A"**/> <styles xmi:type=**"notation:FontStyle"** xmi:id=**"_fhuCov5tEeSuLItin-El6A"** fontName=**"Segoe UI"**/> <element xmi:type=**"BPMNModel:BPMNAssociation"** href=**"caso11_1.bpmn#//@associations.0"**/> <bendpoints xmi:type=**"notation:RelativeBendpoints"** xmi:id=**"_fhuCo_5tEeSuLItin-El6A"** points=**"[68, -2, -252, -10]$[315, -45, -5, -53]"**/> </edges>	Association

Figura 9.11 - Association na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

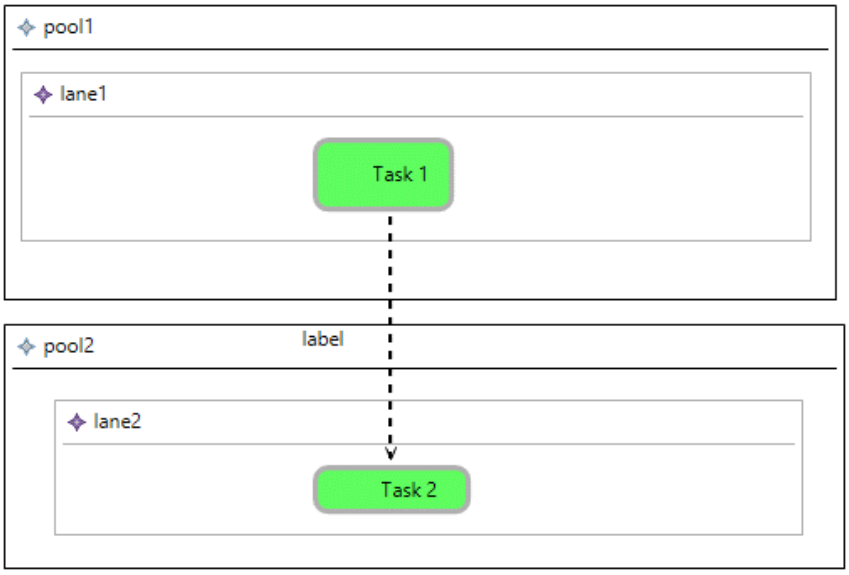
9.1.14. Message Flow

O bloco a seguir representa um Message Flow no diagrama. O conjunto formado pela tag “edges” com atributo type de valor “4002”, uma tag filha “children” de type “6001” contendo uma tag “layoutConstraint”, seguida por duas tags “styles”, uma tag “element” e uma “bendpoints” representa um Message Flow, como ilustrado na Figura 9.12. Esse bloco deve ser adicionado como filho da tag “notation:Diagram”, após a única ocorrência de tag filha “element”.

<edges xmi:type=**"notation:Edge"** xmi:id=**"_7O39YP71EeSupLsaLvlLMw"** type=**"4002"** source=**"_3mloEP71EeSupLsaLvlLMw"**	Message Flow

```
target="_5s_68P71EeSupLsaLvlLMw">
   <children xmi:type="notation:DecorationNode"
xmi:id="_7O4kcP71EeSupLsaLvlLMw" type="6001">
      <layoutConstraint xmi:type="notation:Location"
xmi:id="_7O4kcf71EeSupLsaLvlLMw" y="40"/>
   </children>
   <styles xmi:type="notation:RoutingStyle"
xmi:id="_7O39Yf71EeSupLsaLvlLMw"/>
   <styles xmi:type="notation:FontStyle" xmi:id="_7O39Yv71EeSupLsaLvlLMw"
fontName="Segoe UI"/>
   <element xmi:type="BPMNModel:BPMNMessageFlow"
href="caso12_1.bpmn#//@messageFlows.0"/>
   <bendpoints xmi:type="notation:RelativeBendpoints"
xmi:id="_7O39Y_71EeSupLsaLvlLMw" points="[7, 26, 12, -154]$[46, 178, 51, -2]"/>
</edges>
```

Figura 9.12 - Message Flow na ferramenta iStar2BPMN



Fonte: screenshot da ferramenta iStar2BPMN no sistema operacional Windows 8.1

9.2. Estrutura Geral de um arquivo bpmn no iStar2BPMN

A ordem em que os elementos aparecem no XML do arquivo ".bpmn" deve ser respeitada de acordo com a listagem abaixo.

9.2.1.Linha Inicial

Linha inicial do arquivo ".bpmn":

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

9.2.2.Nó raíz

Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama.

<BPMNModel:BPMNModel xmi:version=**"2.0"** xmlns:xmi=**"http://www.omg.org/XMI"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xmlns:BPMNModel=**"bpmnmodel"**>

9.2.3.Pool

A tag a seguir aparece zero ou mais vezes como filha da tag "BPMNModel:BPMNModel" e corresponde a uma Pool no diagrama.

<pools name=**"pool1"**>

9.2.4.Lane

A tag a seguir aparece zero ou mais vezes como filha de alguma ocorrência de tag "lanes" e representa uma Lane no diagrama.

<lanes name=**"lane1"**>

9.2.5.Start Event

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag "lanes" e representa um Start Event no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNInitialEvent"** label=**"init"**/>	Start Event

9.2.6.Task

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa uma Task no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNTask"** label=**"label"** name=**"task1"**/>	Task

9.2.7.Intermediate Event

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa um Intermediate Event no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNIntermediateEvent"** label=**"nome"**/>	Intermediate Event

9.2.8.Final Event

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa um Final Event no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNFinalEvent"** label**="final"**/>	Final Event

9.2.9.Gateway

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa um Gateway no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNGateway"** label=**"label"** condition=**"condition"**/>	Gateway

9.2.10. Data Object

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa um Data Object no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNDataObject"** label=**"label"** name=**"nome"**/>	Data Object

9.2.11. Sub-Process

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa um Sub-Process no diagrama.

<nodes xsi:type=**"BPMNModel:BPMNSubProcess"** label=**"label"** name=**"nome"**/>	Sub-Process

9.2.12. Sequence Flow

A tag a seguir pode aparecer zero ou mais vezes como filha de alguma ocorrência de tag “lanes” e representa uma ligação do tipo Sequence Flow no diagrama.

<sequenceFlows label=**"label"** source=**"//@pools.0/@lanes.0/@nodes.0"** target=**"//@pools.0/@lanes.0/@nodes.1"**/>	Sequence Flow

9.2.13. Association

A tag a seguir pode aparecer zero ou mais vezes como filha da tag “BPMNModel:BPMNModel” e representa uma ligação do tipo Association no diagrama.

<associations sourceA=**"//@pools.0/@lanes.0/@nodes.0"** targetA=**"//@pools.0/@lanes.0/@nodes.1"**/>	Association

9.2.14. Message Flow

A tag a seguir pode aparecer zero ou mais vezes como filha da tag “BPMNModel:BPMNModel”, após todas as ocorrências de tag “pools” e representa uma ligação do tipo Message Flow no diagrama.

<messageFlows label=**"label"** source=**"//@pools.0/@lanes.0/@nodes.0"** target=**"//@pools.1/@lanes.0/@nodes.0"**/>	Message Flow

10. APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi

Um diagrama BPMN na ferramenta Bizagi pode ser exportado para o formato “BPMN” gerando um arquivo de extensão “.bpmn”, escrito em XML. A seguir será feito um estudo a respeito desse formato de arquivo objetivando encontrar características relevantes a um processo de conversão de um diagrama BPMN da ferramenta Bizagi para um formato reconhecível pela ferramenta iStar2BPMN.

A ordem em que os elementos aparecem no XML do arquivo ”.bpmn” deve ser respeitada de acordo com a listagem abaixo.

- Linha inicial do arquivo “.bpmn”:

```
<?xml version="1.0"?>
```

- Tag correspondente ao diagrama como um todo. É o nó pai de todos os elementos do diagrama, portanto é considerado também o nó raíz do diagrama. Apresenta obrigatoriamente como filhos diretos no mínimo três tags: uma “process”, uma tag “collaboration” e uma “BPMNDiagram”. O valor “**20100524**”, conjunto de 8 caracteres numéricos, presente no atributo “xmlns” da tag “definitions” estará presente também em todas as ocorrências de atributo “xmlns”.

```
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="_2015052008317"
targetNamespace="http://www.bizagi.com/definitions/_2015052008317"
xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL">
```

- Aparece uma única vez. Representaria uma Pool no diagrama se não tivesse o atributo “name”.

```
 <process id="Id_3b4187b9-0536-4819-a8f0-40b9640cc875" name="Main Process">
  <documentation />
 </process>
```

- Tag pai de toda ocorrência de tag “participant”. Inicialmente tem-se apenas uma tag “participant” correspondente ao process de atributo name igual a “Main Process”. O atributo “processRef” da tag “participant” deve conter o mesmo valor que o atributo “id” da tag “process” cujo name é igual a “Main Process”. No atributo name da tag “collaboration” tem-se o nome do diagrama.

```
<collaboration id="Id_a90d7f37-dc70-4699-9397-544c9c226223" name="Diagram 1">
  <participant id="Id_4c491030-aa7a-4513-92de-737209ae0b05" name="Main Process"
processRef="Id_3b4187b9-0536-4819-a8f0-40b9640cc875">
    <extensionElements>
      <bizagi:BizagiExtensions xmlns:bizagi="http://www.bizagi.com/bpmn20">
        <bizagi:BizagiProperties>
          <bizagi:BizagiProperty name="bgColor" value="White" />
          <bizagi:BizagiProperty name="borderColor" value="Black" />
          <bizagi:BizagiProperty name="isMainParticipant" />
        </bizagi:BizagiProperties>
      </bizagi:BizagiExtensions>
    </extensionElements>
  </participant>
</collaboration>
```

- A tag “BPMNDiagram” contém como filhas uma tag “BPMNPlane” e uma ou mais “BPMNLabelStyle”. O atributo “bpmnElement” da tag “BPMNPlane” deve conter o mesmo valor que o atributo “id” da tag “collaboration”. O atributo “bpmnElement” da tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da tag “participant”. O atributo “labelStyle” da tag “BPMNLabel” deve conter o mesmo valor que o atributo “id” da tag “BPMNLabelStyle”.

```
<BPMNDiagram id="Diagram_0c4a572c-4d76-4c75-aca0-894266a44581"
xmlns="http://www.omg.org/spec/BPMN/20100524/DI">
  <BPMNPlane id="DiagramElement_2c7ef70e-9930-401c-87b0-2ecaf53f5d7d"
bpmnElement="Id_a90d7f37-dc70-4699-9397-544c9c226223">
    <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
    <BPMNShape id="DiagramElement_38212f35-145b-4df9-a4a1-5e73841d55f6"
bpmnElement="Id_4c491030-aa7a-4513-92de-737209ae0b05" isHorizontal="true">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="30" y="30" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      <BPMNLabel id="DiagramElement_f3fa967c-e66b-44bb-8cbd-729e33217df3"
labelStyle="Style_79e23c31-0d64-40fe-b996-1da6367442ff">
        <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
        <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      </BPMNLabel>
    </BPMNShape>
  </BPMNPlane>
```

```
   <BPMNLabelStyle id="Style_79e23c31-0d64-40fe-b996-1da6367442ff">
     <Font name="Segoe UI" size="10" isBold="true" isItalic="false" isUnderline="false"
isStrikeThrough="false" xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   </BPMNLabelStyle>
  </BPMNDiagram>
</definitions>
```

10.1. Estrutura geral de um arquivo bpmn no Bizagi

Cada um dos tópicos a seguir representa o acréscimo de algum elemento a um diagrama qualquer reconhecível pela ferramenta Bizagi. Cada adição resulta no acréscimo de linhas de código ao arquivo XML.

10.1.1. Pool

A adição de um Artifact do tipo Pool em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1]
`<process id="Id_bee6bc56-7efd-4df8-85ed-30c611e2492c">`
`<documentation />`
`</process>`
 	
[2]
`<participant id="Id_6b08251c-e69b-4c92-858e-c918dc3aefe7" name="pool1"`
`processRef="Id_bee6bc56-7efd-4df8-85ed-30c611e2492c">`
`<extensionElements>`
`<bizagi:BizagiExtensions xmlns:bizagi="http://www.bizagi.com/bpmn20">`
`<bizagi:BizagiProperties>`
`<bizagi:BizagiProperty name="bgColor" value="White" />`
`<bizagi:BizagiProperty name="borderColor" value="Black" />`
`</bizagi:BizagiProperties>`
`</bizagi:BizagiExtensions>`
`</extensionElements>`
`</participant>`
 	
[3]
`<BPMNShape id="DiagramElement_6b950928-efba-482d-b3d3-7c3a1683d807"`
`bpmnElement="Id_6b08251c-e69b-4c92-858e-c918dc3aefe7" isHorizontal="true">`
	Pool

```
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="153" y="0" width="700" height="350"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      <BPMNLabel id="DiagramElement_e28f63e5-f69d-422b-8404-578126ada4be"
labelStyle="Style_4391551b-e256-4378-bf88-6e6f42037226">
        <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
        <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      </BPMNLabel>
    </BPMNShape>

[4]
  <BPMNLabelStyle id="Style_4391551b-e256-4378-bf88-6e6f42037226">
    <Font name="Segoe UI" size="8" isBold="true" isItalic="false" isUnderline="false"
isStrikeThrough="false" xmlns="http://www.omg.org/spec/DD/20100524/DC" />
  </BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “definitions” antes da primeira ocorrência de tag “collaboration”. O bloco de número 2 é filho da tag “collaboration”. O bloco 3 é filho da tag “BPMNPlane” ocorrendo após a primeira ocorrência de tag “extension”. E finalmente o quarto bloco é filho direto da tag “BPMNDiagram” ocorrendo após todas as ocorrências de tag “BPMNPlane”. A figura abaixo ilustra a adição de um elemento Pool.

Figura 10.1 – Pool na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.2. Lane

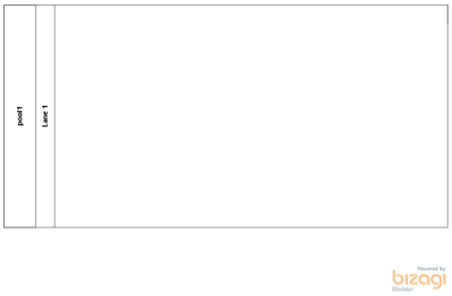
A adição de um Artifact do tipo Lane em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

	Lane
[1] <laneSet id=**"Id_a17b1619-8661-4f45-846e-94aa0e9655bc"**>	Apenas para a primeira inserção
<lane id=**"Id_a0c9a27d-4ad4-4467-b2e8-861424d754ee"** name=**"Lane 1"**> <childLaneSet id=**"Id_bb3d1f8d-54f6-4292-851a-ff6a4532699a"** /> </lane>	
</laneSet>	Apenas para a primeira inserção
[2] <BPMNShape id=**"DiagramElement_1fdb822a-09b8-41da-9482-2559cb9aaeec"** bpmnElement=**"Id_a0c9a27d-4ad4-4467-b2e8-861424d754ee"** isHorizontal=**"true"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"50"** y=**"0"** width=**"650"** height=**"350"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> <BPMNLabel id=**"DiagramElement_e7146d5f-b977-40f8-9c50-1929cc2d2acb"** labelStyle=**"Style_f5129d79-fa75-4837-9e16-2e64c8827f2d"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"0"** y=**"0"** width=**"0"** height=**"0"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabel> </BPMNShape> **[3]** <BPMNLabelStyle id=**"Style_f5129d79-fa75-4837-9e16-2e64c8827f2d"**> <Font name=**"Segoe UI"** size=**"8"** isBold=**"true"** isItalic=**"false"** isUnderline=**"false"** isStrikeThrough=**"false"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabelStyle>	

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde essa Lane foi adicionada. A tag “laneSet” representa um conjunto de Lanes podendo conter uma ou mais Lanes no diagrama. A

mesma será adicionada apenas ao inserir a primeira ocorrência de tag “lane” para uma determinada tag “process”, resultando em um conjunto de lanes (“laneSet”) por Pool (“process”). O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “lane”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.2 - Lane na ferramenta Bizagi



Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.3. Start Event

A adição de um Start Event em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <startEvent id=**"Id_af551326-08ac-420a-96b1-f2cf97fd1749"** name=**"init"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties> <bizagi:BizagiProperty name=**"bgColor"** value=**"#E6FF97"** /> <bizagi:BizagiProperty name=**"borderColor"** value=**"#62A716"** /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </startEvent> **[2]** <BPMNShape id=**"DiagramElement_fb04d1d2-0455-4666-9c52-b8b56898a980"** bpmnElement=**"Id_af551326-08ac-420a-96b1-f2cf97fd1749"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"273"** y=**"111"** width=**"30"** height=**"30"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** />	Start Event

```
   <BPMNLabel id="DiagramElement_86174e38-addf-4c5a-80c1-8c28e32cd23d"
labelStyle="Style_67ad4380-a865-4a0f-bfd9-a66218c6e9d2">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   </BPMNLabel>
</BPMNShape>

[3]
<BPMNLabelStyle id="Style_67ad4380-a865-4a0f-bfd9-a66218c6e9d2">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false" isUnderline="false"
isStrikeThrough="false" xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Start Event foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “startEvent”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.3 - Start Event na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

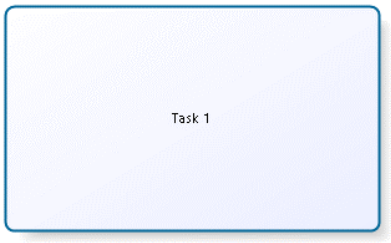
10.1.4. Task

A adição de uma Task em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <task id=**"Id_ec3d6937-2d10-4fbb-a0c2-59b23328dee3"** name=**"Task 1"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties> <bizagi:BizagiProperty name=**"bgColor"** value=**"#ECEFFF"** /> <bizagi:BizagiProperty name=**"borderColor"** value=**"#03689A"** /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </task> **[2]** <BPMNShape id=**"DiagramElement_98f5b05f-9fa6-46cf-b3bd-c78b19a708ac"** bpmnElement=**"Id_ec3d6937-2d10-4fbb-a0c2-59b23328dee3"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"290"** y=**"142"** width=**"90"** height=**"60"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> <BPMNLabel id=**"DiagramElement_194aafee-fd4e-44fb-9113-40d76b48088d"** labelStyle=**"Style_88d878a1-efa1-44b0-9621-52615173a26e"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"0"** y=**"0"** width=**"0"** height=**"0"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabel> </BPMNShape> **[3]** <BPMNLabelStyle id=**"Style_88d878a1-efa1-44b0-9621-52615173a26e"**> <Font name=**"Segoe UI"** size=**"8.25"** isBold=**"false"** isItalic=**"false"** isUnderline=**"false"** isStrikeThrough=**"false"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabelStyle>	Task

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Start Event foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “task”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.4 – Task na ferramenta Bizagi



Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.5. Intermediate Event

A adição de um Intermediate Event em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <intermediateThrowEvent id=**"Id_b33fa564-8c0a-47bb-ad89-29030161f4bd"** name=**"nome"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties> <bizagi:BizagiProperty name=**"bgColor"** value=**"#FEFAEF"** /> <bizagi:BizagiProperty name=**"borderColor"** value=**"#969149"** /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </intermediateThrowEvent> **[2]** <BPMNShape id=**"DiagramElement_d0f6a841-b4a9-483e-901f-07eeff5409f1"** bpmnElement=**"Id_b33fa564-8c0a-47bb-ad89-29030161f4bd"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"339"** y=**"144"** width=**"30"** height=**"30"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> <BPMNLabel id=**"DiagramElement_4921421c-0711-4307-842e-62f9dc293372"** labelStyle=**"Style_d2210149-03df-446e-b03d-b3a9a75eabb8"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"0"** y=**"0"** width=**"0"** height=**"0"**	Intermediate Event

xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabel> </BPMNShape> **[3]** <BPMNLabelStyle id=**"Style_d2210149-03df-446e-b03d-b3a9a75eabb8"**> <Font name=**"Segoe UI"** size=**"8.25"** isBold=**"false"** isItalic=**"false"** isUnderline=**"false"** isStrikeThrough=**"false"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabelStyle>	

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Intermediate Event foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “intermediateThrowEvent”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.5 – Intermediate Event na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.6. Final Event

A adição de um Final Event em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <endEvent id=**"Id_c0cecc00-ca14-420f-867a-065d85fb5f7e"** name=**"final"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties>	Final Event

```
            <bizagi:BizagiProperty name="bgColor" value="#EEAAAA" />
            <bizagi:BizagiProperty name="borderColor" value="#990000" />
         </bizagi:BizagiProperties>
      </bizagi:BizagiExtensions>
   </extensionElements>
</endEvent>

[2]
<BPMNShape id="DiagramElement_691872eb-9749-4ddd-9e08-0de8a84cef9b"
bpmnElement="Id_c0cecc00-ca14-420f-867a-065d85fb5f7e">
   <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
   <Bounds x="374" y="151" width="30" height="30"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   <BPMNLabel id="DiagramElement_316e28ae-799a-4423-b54e-84ab0a509542"
labelStyle="Style_af3b7ecb-7387-4dfe-ae37-d8280e255f02">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   </BPMNLabel>
</BPMNShape>

[3]
<BPMNLabelStyle id="Style_af3b7ecb-7387-4dfe-ae37-d8280e255f02">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false"
isUnderline="false" isStrikeThrough="false"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Final Event foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “endEvent”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.6 – Final Event na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.7. Gateway

A adição de um Gateway em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <exclusiveGateway id=**"Id_7ede0ad6-d5e4-4e4a-96db-60c10a88484c"** name=**"nome"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties> <bizagi:BizagiProperty name=**"bgColor"** value=**"#FFFFCC"** /> <bizagi:BizagiProperty name=**"borderColor"** value=**"#A6A61D"** /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </exclusiveGateway> **[2]** <BPMNShape id=**"DiagramElement_0e17ad11-5054-49d8-b73f-ba5fc0122d29"** bpmnElement=**"Id_7ede0ad6-d5e4-4e4a-96db-60c10a88484c"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"333"** y=**"106"** width=**"60"** height=**"60"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> <BPMNLabel id=**"DiagramElement_99466592-d712-4838-9978-df13d57e571d"** labelStyle=**"Style_d1b47eb2-3675-4174-baf3-ac44e069be59"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"0"** y=**"0"** width=**"0"** height=**"0"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** />	Gateway

```
   </BPMNLabel>
</BPMNShape>

[3]
<BPMNLabelStyle id="Style_d1b47eb2-3675-4174-baf3-ac44e069be59">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false"
isUnderline="false" isStrikeThrough="false"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Gateway foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “exclusiveGateway”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.7 – Gateway na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.8. Data Object

A adição de um Data Object em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

```
[1]                                                                    Data
<dataObject id="Id_872a7b6b-a5e9-4806-937f-724eaa139d20" name="nome">  Object
   <documentation />
   <extensionElements>
      <bizagi:BizagiExtensions xmlns:bizagi="http://www.bizagi.com/bpmn20">
```

```
        <bizagi:BizagiProperties>
          <bizagi:BizagiProperty name="bgColor" value="#F0F0F0" />
          <bizagi:BizagiProperty name="borderColor" value="#666666" />
        </bizagi:BizagiProperties>
      </bizagi:BizagiExtensions>
   </extensionElements>
   <dataState id="Id_24bcd967-8b45-4191-a74a-e83b644dd219" name="" />
</dataObject>

[2]
<BPMNShape id="DiagramElement_d3ef624c-d70b-4700-a91f-6ca3d4d8b34e"
bpmnElement="Id_872a7b6b-a5e9-4806-937f-724eaa139d20">
   <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
   <Bounds x="303" y="83" width="40" height="50"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   <BPMNLabel id="DiagramElement_b9936efb-448a-4d63-98e8-f2ae7562378b"
labelStyle="Style_a5daea5e-77b0-4e45-a3ce-ae0829298ec7">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   </BPMNLabel>
</BPMNShape>

[3]
<BPMNLabelStyle id="Style_a5daea5e-77b0-4e45-a3ce-ae0829298ec7">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false"
isUnderline="false" isStrikeThrough="false"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Data Object foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “dataObject”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.8 – Data Object na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.9. Sub-Process

A adição de um Sub-Process em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <subProcess id=**"Id_d471072e-7a7f-426b-88e3-2a01b7c768f7"** name=**"nome"**> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi=**"http://www.bizagi.com/bpmn20"**> <bizagi:BizagiProperties> <bizagi:BizagiProperty name=**"bgColor"** value=**"#ECEFFF"** /> <bizagi:BizagiProperty name=**"borderColor"** value=**"#03689A"** /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </subProcess> **[2]** <BPMNShape id=**"DiagramElement_0df5d99b-435a-4480-869f-c2bc01fdecbd"** bpmnElement=**"Id_d471072e-7a7f-426b-88e3-2a01b7c768f7"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"328"** y=**"84"** width=**"90"** height=**"60"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> <BPMNLabel id=**"DiagramElement_4eff9de2-47b3-461a-a190-31d0f15f30f9"** labelStyle=**"Style_d47b1491-24f1-4f40-8370-d002cf8dfe51"**> <extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"0"** y=**"0"** width=**"0"** height=**"0"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabel> </BPMNShape>	Sub-Process

[3] <BPMNLabelStyle id=**"Style_d47b1491-24f1-4f40-8370-d002cf8dfe51"**> <Font name=**"Segoe UI"** size=**"8.25"** isBold=**"false"** isItalic=**"false"** isUnderline=**"false"** isStrikeThrough=**"false"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabelStyle>	

O primeiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Sub-Process foi adicionado. O segundo bloco representa um novo bloco “BPMNShape” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNShape” deve conter o mesmo valor que o atributo “id” da nova tag “subProcess”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.9 – Sub-Process na ferramenta Bizagi

Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.10. Sequence Flow

A adição de um Sequence Flow em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

[1] <outgoing>**Id_83d5bbe2-e1f9-40a1-9588-32937cd60a63**</outgoing> **[2]** <incoming>**Id_83d5bbe2-e1f9-40a1-9588-32937cd60a63**</incoming> **[3]**	Sequence Flow

```
<sequenceFlow id="Id_83d5bbe2-e1f9-40a1-9588-32937cd60a63" name="nome"
sourceRef="Id_dbacb933-3e92-45a8-bf42-4e17e47f3072"
targetRef="Id_a732a02b-7083-4d04-b4f3-d3e19daa3da7">
   <documentation />
   <extensionElements>
      <bizagi:BizagiExtensions xmlns:bizagi="http://www.bizagi.com/bpmn20">
         <bizagi:BizagiProperties>
            <bizagi:BizagiProperty name="bgColor" value="White" />
            <bizagi:BizagiProperty name="borderColor" value="Black" />
         </bizagi:BizagiProperties>
      </bizagi:BizagiExtensions>
   </extensionElements>
</sequenceFlow>

[4]
<BPMNEdge id="DiagramElement_41a7bf40-3aeb-4a78-8cd8-49f62296d96b"
bpmnElement="Id_83d5bbe2-e1f9-40a1-9588-32937cd60a63">
   <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
   <waypoint x="161" y="150" xmlns="http://www.omg.org/spec/DD/20100524/DI"
/>
   <waypoint x="328" y="150" xmlns="http://www.omg.org/spec/DD/20100524/DI"
/>
   <BPMNLabel id="DiagramElement_687762a5-731d-4fb0-836b-83d18b1f14f7"
labelStyle="Style_9bd3d32e-db05-4391-8d3f-ec0a8bdebed6">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="245" y="150" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
   </BPMNLabel>
</BPMNEdge>

[5]
<BPMNLabelStyle id="Style_9bd3d32e-db05-4391-8d3f-ec0a8bdebed6">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false"
isUnderline="false" isStrikeThrough="false"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é composto por uma tag “outgoing” que representa a existência de uma ligação saindo deste objeto. Em outras palavras, se houver por exemplo uma ligação do tipo

Sequence Flow saindo de um Start Event, esse bloco pertencerá à tag “startEvent” como forma de representar a existência de um Sequence Flow partindo deste elemento (Start Event).

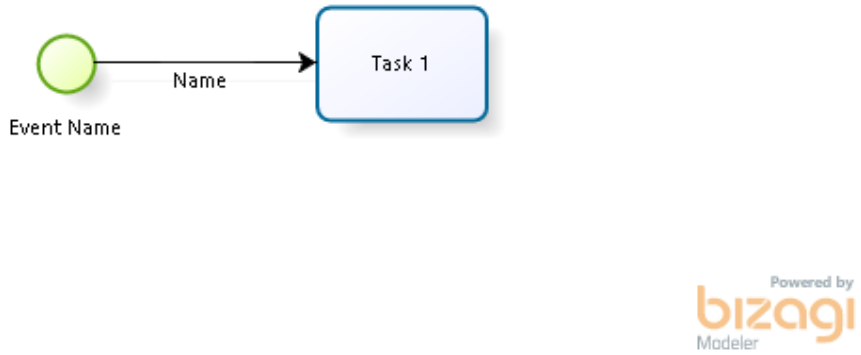
O segundo bloco é análogo ao primeiro, porém representa a existência de uma ligação que alcança um elemento.

O terceiro bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde esse Sequence Flow foi adicionado.

O quarto bloco representa um novo bloco “BPMNEdge” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNEdge” deve conter o mesmo valor que o atributo “id” da nova tag “sequenceFlow”. Esse valor será igual também ao inserido dentro das tags “outgoing” e “incoming”.

O quinto bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.10 – Sequence Flow na ferramenta Bizagi



Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.11. Association

A adição de uma Association em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

	Association
[1] <ioSpecification id=**"Id_a6720501-5f26-411c-ad59-d7366081e56a"**>	Apenas para a primeira inserção
<dataOutput id=**"Id_d7960b5c-4e97-4e77-b681-e50e12ee1dec"** />	
<inputSet id=**"Id_ef490d2c-b99e-45a4-80dd-59f0e58fdd10"** /> <outputSet id=**"Id_e8c1298b-3044-472f-89a4-6c55db477955"**>	Apenas para a primeira inserção
<dataOutputRefs>**Id_d7960b5c-4e97-4e77-b681-e50e12ee1dec**</dataOutputRefs>	

</outputSet> </ioSpecification>	Apenas para a primeira inserção
<dataOutputAssociation id="**Id_ea1ac9d0-06bd-4f2c-abe7-71f21ec735b0**"> <sourceRef>**Id_d7960b5c-4e97-4e77-b681-e50e12ee1dec**</sourceRef> <targetRef>**Id_a89a8bf9-79b7-44cc-a3c1-c61f532606f3**</targetRef> </dataOutputAssociation> **[2]** <association id="**Id_839a0759-709f-4e33-bfd7-f082adda21c7**" sourceRef="**Id_94908ac7-7603-4fab-bca5-7f8cebcd326d**" targetRef="**Id_a89a8bf9-79b7-44cc-a3c1-c61f532606f3**"> <documentation /> <extensionElements> <bizagi:BizagiExtensions xmlns:bizagi="**http://www.bizagi.com/bpmn20**"> <bizagi:BizagiProperties> <bizagi:BizagiProperty name="**bgColor**" value="**White**" /> <bizagi:BizagiProperty name="**borderColor**" value="**Black**" /> </bizagi:BizagiProperties> </bizagi:BizagiExtensions> </extensionElements> </association> **[3]** <BPMNEdge id="**DiagramElement_d1f26497-bdf7-4d0e-87d9-7261ec5dd8c4**" bpmnElement="**Id_839a0759-709f-4e33-bfd7-f082adda21c7**"> <extension xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**297**" y="**138**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**351.5**" y="**138**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**351.5**" y="**161**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**531**" y="**161**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <BPMNLabel id="**DiagramElement_86590b36-5264-41b9-8071-dd8e94446d27**" labelStyle="**Style_e035c282-336f-4882-87c4-2d0db41d1d7d**">	

<extension xmlns=**"http://www.omg.org/spec/DD/20100524/DI"** /> <Bounds x=**"352"** y=**"150"** width=**"0"** height=**"0"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabel> </BPMNEdge> **[4]** <BPMNLabelStyle id=**"Style_e035c282-336f-4882-87c4-2d0db41d1d7d"**> <Font name=**"Segoe UI"** size=**"8.25"** isBold=**"false"** isItalic=**"false"** isUnderline=**"false"** isStrikeThrough=**"false"** xmlns=**"http://www.omg.org/spec/DD/20100524/DC"** /> </BPMNLabelStyle>	

Dado um diagrama com a presença de uma Task e um Data Object, a adição de uma Association que inicia na Task e termina no Data Object resulta na inserção do bloco de número um como filho da tag “task” cujo atributo “id” apresenta valor igual ao do atributo “sourceRef” da tag “association” do bloco de número dois. Essa inserção do primeiro bloco será feita após a primeira ocorrência de tag “extensionElements”. Para cada nova Association inserida após a primeira inserção alguns trechos do código não serão repetidos. Os mesmos estão marcados com “Apenas para a primeira inserção” na tabela acima.

O segundo bloco é adicionado como filho da tag “process” cujo “id” apresenta o mesmo valor que o atributo “processRef” da tag “participant” correspondente a Pool onde essa Association foi criada.

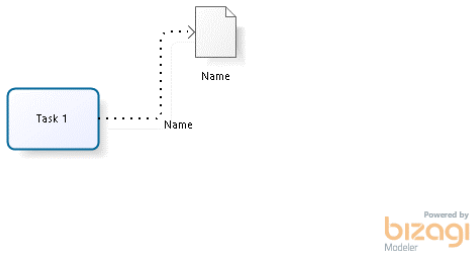
O terceiro bloco representa um novo bloco “BPMNEdge” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNEdge” deve conter o mesmo valor que o atributo “id” da nova tag “association”.

O quarto bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

O atributo “id” da tag “dataOutput” deve conter o mesmo valor que o inserido entre as tags “dataOutputRefs” e “sourceRef”.

O valor do atributo “targetRef” da tag “association” deve ser igual ao valor inserido entre as tags “targetRef” filhas de “dataOutputAssociation”.

Figura 10.11 – Association na ferramenta Bizagi



Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

10.1.12. Message Flow

A adição de um Message Flow em um diagrama BPMN na ferramenta Bizagi provoca o acréscimo das seguintes linhas de código XML (divididas em blocos):

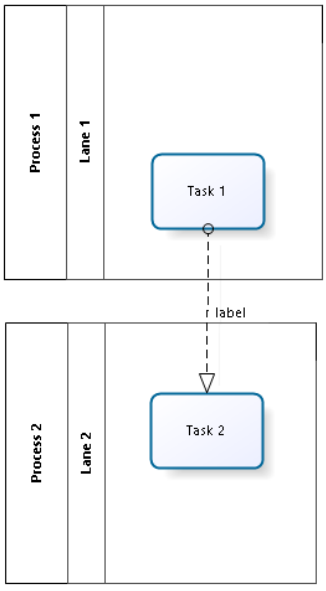
[1] <messageFlow id="**Id_152e38d6-6012-44ac-b367-6718d479f737**" name="**label**" sourceRef="**Id_7b681e60-a0cf-4466-bbb4-2fea6c898a14**" targetRef="**Id_4da92d81-b4e2-496e-aa90-3a02cae877cc**"> <documentation /> </messageFlow> **[2]** <BPMNEdge id="**DiagramElement_f0350947-407d-434e-9d29-e4a45fb14c60**" bpmnElement="**Id_152e38d6-6012-44ac-b367-6718d479f737**"> <extension xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**283**" y="**301**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**283**" y="**376.5**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**279**" y="**376.5**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <waypoint x="**279**" y="**439**" xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <BPMNLabel id="**DiagramElement_3bec6940-d9be-4cc5-bab8-88a46e1a054c**" labelStyle="**Style_b6874e0e-0fde-402b-bd01-7bcb8a38fbd7**"> <extension xmlns="**http://www.omg.org/spec/DD/20100524/DI**" /> <Bounds x="**281**" y="**376.5**" width="**0**" height="**0**" xmlns="**http://www.omg.org/spec/DD/20100524/DC**" /> </BPMNLabel>	Message Flow

```
</BPMNEdge>


[3]
<BPMNLabelStyle id="Style_b6874e0e-0fde-402b-bd01-7bcb8a38fbd7">
   <Font name="Segoe UI" size="8.25" isBold="false" isItalic="false"
isUnderline="false" isStrikeThrough="false"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
</BPMNLabelStyle>
```

O primeiro bloco é adicionado como filho da tag “collaboration” logo após todas as ocorrências de “participant”. O segundo bloco representa um novo bloco “BPMNEdge” que será adicionado como filho da tag “BPMNPlane”. O atributo “bpmnElement” da nova tag “BPMNEdge” deve conter o mesmo valor que o atributo “id” da nova tag “messageFlow”. O terceiro bloco é uma tag “BPMNLabelStyle” cujo atributo “id” apresenta o mesmo valor que o atributo “labelStyle” da tag “BPMNLabel” recém adicionada.

Figura 10.12 – Message Flow na ferramenta Bizagi



Fonte: screenshot da ferramenta Bizagi no sistema operacional Windows 8.1

11. APÊNDICE E - Procedimento de conversão de um diagrama i* na ferramenta OpenOME em formato ".ood" para um diagrama i* na ferramenta iStar2BPMN

Dado um arquivo de extensão ".ood", escrito em XML, que represente um diagrama qualquer reconhecível pela ferramenta OpenOME. O procedimento de conversão se inicia com a captura e armazenamento do nome fornecido ao diagrama, ou seja o nome deste arquivo de extensão ".ood". O mesmo deve ser encontrado no atributo "name" da tag "notation:Diagram" do XML do arquivo ".ood".

Em seguida são criados os arquivos de extensão ".istar" e ".istardiagram" utilizando o nome do diagrama, capturado anteriormente, para nomear os arquivos. Ambos receberão o mesmo nome.

O arquivo "nome do diagrama.istar" inicialmente contém apenas uma tag "IstarModel:IstarModel" com atributos "xmi:version", "xmlns:xmi" e "xmlns:IstarModel", resultando nas seguintes linhas de código:

```
<?xml version="1.0" encoding="UTF-8"?>
<IstarModel:IstarModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IstarModel="IstarModel"/>
```

O arquivo "nome do diagrama.istardiagram" inicial contém apenas o seguinte código (variando apenas o valor do atributo "xmi:id" da tag "notation:Diagram"):

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IstarModel="IstarModel"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation"
xmi:id="_S6Tf8OKyEeSbqKfxMPV-XQ" type="Top" name="caso0.istardiagram"
measurementUnit="Pixel">
  <styles xmi:type="notation:DiagramStyle" xmi:id="_S6Tf8eKyEeSbqKfxMPV-XQ"/>
  <element xmi:type="IstarModel:IstarModel" href="caso0.istar#/"/>
</notation:Diagram>
```

Juntos, esses dois arquivos representam um diagrama vazio (sem elementos) na ferramenta iStar2BPMN. Estes códigos são padrões, ou seja todo diagrama vazio recém-criado na ferramenta irá conter os códigos acima. Eles servirão de base para receber tudo o que for convertido a partir do arquivo ".ood", sendo povoados durante o processo de conversão, construindo assim um diagrama reconhecível pela ferramenta iStar2BPMN.

11.1. Conversão – Etapa 1 (Containers com ou sem Elements internos)

Nessa primeira etapa todas as ocorrências de containers (Actor, Agent, Role, Position) serão tratadas. Cada ocorrência é determinada por uma tag <containers> no XML do arquivo ".ood". Para cada ocorrência será feito o conjunto de passos a seguir:

1. Para cada tag <containers> encontrada:
   1.1. Capturar e armazenar o valor do atributo "name".
   1.2. Capturar e armazenar o valor do atributo "xmi:id".
   1.3. Capturar e armazenar o valor do atributo "xmi:type".
   1.4. Criar no arquivo ".istar" uma nova tag <compartments> com atributos "name", "xmi:id" e "xmi:type" cujos valores serão os capturados nos passos 1.1, 1.2 e 1.3 respectivamente.
   1.5. Adicionar à tag <compartments>, criada no passo anterior, o atributo "num" de valor igual a quantidade de tags <compartments> já existentes no arquivo ".istar" imediatamente antes do momento da criação desta nova.
   1.6. Adicionar a tag <compartments> recém-criada ao nó raiz do XML do arquivo ".istar", como filha direta do mesmo.
   1.7. O seguinte bloco de código XML, que corresponde a um compartment (Actor, Agent, Role, Position) na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo ".istardiagram", respeitando a hierarquia:

<children xmi:type="notation:Shape"> <children xmi:type="notation:DecorationNode"/> <children xmi:type="notation:BasicCompartment"/> <styles xmi:type="notation:HintedDiagramLinkStyle"/> <element xmi:type="IstarModel:IstarCompartment"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.1 - Relação de tags e atributos de um compartment na ferramenta iStar2BPMN

Tag	Atributo	Valor
<children>	type	2002
	xmi:id	O mesmo valor que o capturado no passo 1.2
	xmi:type	notation:Shape
	fontName	Segoe UI
<children>	type	5002

	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<children>	type	7001
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:BasicCompartment
<styles>	xmi:type	notation:HintedDiagramLinkStyle
	xmi:id	Cadeia de 23 caracteres
<element>	xmi:type	IstarModel:IstarCompartment
	href	nome do diagrama + ".istar#//@compartments." + num (valor capturado no passo 1.5)
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados a partir da tag <layoutConstraint> do arquivo ".ood" que é filha da tag <children> cujo atributo "element" apresenta o mesmo valor que o capturado no passo 1.2
	y	
	width	
	height	

1.8. Se a tag <containers> atual possui filhos, para cada tag <intentions> filha de <containers>:

1.8.1. Capturar e armazenar o valor do atributo "name".

1.8.2. Capturar e armazenar o valor do atributo "xmi:id".

1.8.3. Capturar e armazenar o valor do atributo "xmi:type".

1.8.4. Criar no arquivo ".istar" uma nova tag <elements> com atributos "name", "xmi:id" e "xmi:type" cujos valores serão os capturados nos passos 1.8.1, 1.8.2 e 1.8.3 respectivamente.

1.8.5. Adicionar à tag <elements>, criada no passo anterior, o atributo "num". O valor de "num" será igual à quantidade existente de tags <elements> filhas da tag <compartments> criada no passo 1.4. A tag <elements> recém-criada será adicionada como tag filha da <compartments>.

1.8.6. O seguinte bloco de código XML, que corresponde a um element (Goal, Softgoal, Task, Resource) na ferramenta iStar2BPMN, será criado e adicionado como filho da tag <children> de atributo type = "7001" criada no ".istardiagram" no passo 1.7:

<children xmi:type="notation:Shape" type="**3001**"> <children xmi:type="notation:DecorationNode" type="**5003**"/> <element xmi:type="IstarModel:IstarElement"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Element

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.2 - Relação de tags e atributos de um element interno a um compartment na ferramenta iStar2BPMN		
Tag	**Atributo**	**Valor**
<children>	type	3001
	xmi:id	O mesmo valor que o capturado no passo 1.8.2
	xmi:type	notation:Shape
	fontName	Segoe UI
<children>	type	5003
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<element>	xmi:type	IstarModel:IstarElement
	href	href da tag <element> do passo 1.7 + "/@elements." + num (valor capturado no passo 1.8.5)
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados a partir da tag <layoutConstraint> do arquivo ".ood" que é filha da tag <children> cujo atributo "element" apresenta o mesmo valor que o capturado no passo 1.8.2
	y	
	width	
	height	

11.2. Conversão – Etapa 2 (Elements externos)

Intentions da ferramenta OpenOME correspondem a Elements no iStar2BPMN. Podem ser do tipo Goal, Softgoal, Task ou Resource. Nessa etapa serão convertidos os elements que não pertencem a algum container (Actor, Agent, Role, Position), ou seja elements externos. Cada ocorrência é determinada por uma tag <intentions> no XML do arquivo ".ood". Para cada ocorrência será feito o conjunto de passos a seguir:

1. Para cada tag <intentions> encontrada que não é filha de uma tag <containers>:
    1.1. Capturar e armazenar o valor do atributo "name".
    1.2. Capturar e armazenar o valor do atributo "xmi:id".
    1.3. Capturar e armazenar o valor do atributo "xmi:type".
    1.4. Criar no arquivo ".istar" uma nova tag <elements> com atributos "name", "xmi:id" e "xmi:type" cujos valores serão os capturados nos passos 1.1, 1.2 e 1.3 respectivamente.

1.5. Adicionar à tag <elements>, criada no passo anterior, o atributo “num” de valor igual a quantidade de tags <elements> já existentes no arquivo “.istar”, que são filhas diretas (primeiro grau) do nó raiz.

1.6. Adicionar a tag <elements> recém-criada ao nó raiz do XML do arquivo “.istar”, como filha direta do mesmo, acima da primeira ocorrência de tag <compartments> se existente.

1.9. O seguinte bloco de código XML, que corresponde a um element (Goal, Softgoal, Task, Resource) na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo “.istardiagram”, respeitando a hierarquia:

<children xmi:type="notation:Shape" type="**2001**"> <children xmi:type="notation:DecorationNode" type="**5001**"/> <element xmi:type="IstarModel:IstarElement"/> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Element

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.3 - Relação de tags e atributos de um element externo na ferramenta iStar2BPMN

Tag	**Atributo**	**Valor**
<children>	type	2001
	xmi:id	O mesmo valor que o capturado no passo 1.2
	xmi:type	notation:Shape
	fontName	Segoe UI
<children>	type	5001
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<element>	xmi:type	IstarModel:IstarElement
	href	nome do diagrama + ".istar#//@elements." + num (valor capturado no passo 1.5)
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados a partir da tag <layoutConstraint> do arquivo “.ood” que é filha da tag <children> cujo atributo “element” apresenta o mesmo valor que o capturado no passo 1.2
	y	
	width	
	height	

11.3. Conversão – Etapa 3 (Association Links)

Nesta etapa serão convertidas todas as ligações entre containers, ou seja todas as actors associations. As mesmas podem ser do tipo ISA, COVERS, ISPARTOF, OCCUPIES, PLAYS e INS. Na ferramenta OpenOME esse tipo de ligação é representada no XML de um diagrama por uma tag <associations>, portanto para cada ocorrência desta tag será realizado o conjunto de passos a seguir:

1. Para cada tag <associations> encontrada:
   1.1. Capturar e armazenar o valor do atributo “xmi:type” e remover a substring "edu.toronto.cs.openome_model:".
   1.2. Capturar e armazenar o valor do atributo “xmi:id”.
   1.3. Capturar e armazenar o valor do atributo “source”.
   1.4. Capturar e armazenar o valor do atributo “target”.
   1.5. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.3 a partir do atributo “source”.
   1.6. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.4 a partir do atributo “target”.
   1.7. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.5 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.8. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.6 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.9. Criar nova tag <actorLinks> no arquivo “.istar”. Essa tag será inserida no XML antes da primeira ocorrência de tag <elements>. Caso não haja, ela será inserida antes da primeira ocorrência de tag <compartments>.
   1.10. Adicionar o atributo “type” à tag <actorLinks>. Seu valor será fornecido de acordo com o valor capturado no passo 1.1 e a tabela abaixo:

Tabela 11.4 - Possíveis valores para o atributo type da tag actorLinks de um arquivo istar

Valor capturado no passo 1.1	**Valor do atributo “type”**
IsAAssociation	
CoversAssociation	COVERS
IsPartOfAssociation	ISPARTOF
OccupiesAssociation	OCCUPIES
PlaysAssociation	PLAYS
INSAssociation	INS

   1.11. Adicionar o atributo “source” à tag <actorLinks>. Seu valor será o armazenado no passo 1.7.

1.12. Adicionar o atributo “target” à tag <actorLinks>. Seu valor será o armazenado no passo 1.8.

1.13. Adicionar o atributo “xmi:id” à tag <actorLinks>. Seu valor será o armazenado no passo 1.2.

1.14. Adicionar o atributo “num” à tag <actorLinks>. Seu valor será igual a quantidade de tags <actorLinks> já existentes no arquivo “.istar”.

1.15. O seguinte bloco de código XML, que corresponde a uma actor association na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo “.istardiagram”:

<edges xmi:type="notation:Connector" type="**4001**" source="_rjah8PDPEeSWd4Dhn1BrBw" target="_s-pJ0PDPEeSWd4Dhn1BrBw"> <children xmi:type="notation:DecorationNode" type="**6001**"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarActorLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Actor association

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.5 - Relação de tags e atributos de um association link na ferramenta iStar2BPMN

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	O mesmo valor que o capturado no passo 1.2
	type	4001
	source	O mesmo valor que o capturado no passo 1.3
	target	O mesmo valor que o capturado no passo 1.4
<children>	type	6001
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres

	fontName	Segoe UI
<element>	xmi:type	IstarModel:IstarActorLink
	href	nome do diagrama + ".istar#//@actorLinks." + num (valor capturado no passo 1.14)
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor capturado a partir do atributo “points” da tag <bendpoints> do arquivo “.ood” que é filha da tag <edges> cujo atributo “element” apresenta o mesmo valor que o capturado no passo 1.2.

11.4. Conversão – Etapa 4 (Dependency Links)

Nesta etapa serão convertidas todas as ligações de dependência, ou seja todas as dependency links. Na ferramenta OpenOME esse tipo de ligação é representada no XML de um diagrama por uma tag <dependencies>, portanto para cada ocorrência desta tag será realizado o conjunto de passos a seguir:

1. Para cada tag <dependencies> encontrada:
   1.1. Capturar e armazenar o valor do atributo “xmi:type” e remover a substring "edu.toronto.cs.openome_model:".
   1.2. Capturar e armazenar o valor do atributo “xmi:id”.
   1.3. Capturar e armazenar o valor do atributo “dependencyTo”.
   1.4. Capturar e armazenar o valor do atributo “dependencyFrom”.
   1.5. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.3 a partir do atributo “dependencyTo”.
   1.6. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.4 a partir do atributo “dependencyFrom”.
   1.7. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.5 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.8. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.6 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.9. Criar nova tag <dependencyLinks> no arquivo “.istar”. Essa tag será inserida no XML antes da primeira ocorrência de tag <elements> que não seja filha de uma tag <compartments>. Caso não haja, ela será inserida antes da primeira ocorrência de tag <compartments>.
   1.10. Adicionar o atributo “source” à tag <dependencyLinks>. Seu valor será o armazenado no passo 1.7.
   1.11. Adicionar o atributo “target” à tag <dependencyLinks>. Seu valor será o armazenado no passo 1.8.
   1.12. Adicionar o atributo “xmi:id” à tag <dependencyLinks>. Seu valor será o armazenado no passo 1.2.

1.13. Adicionar o atributo “num” à tag <dependencyLinks>. Seu valor será igual a quantidade de tags <dependencyLinks> já existentes no arquivo “.istar”.

1.14. O seguinte bloco de código XML, que corresponde a uma dependency link na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo “.istardiagram”:

<edges xmi:type="notation:Connector" type="**4002**" source="_gG3rgPDiEeSb8NZPdPkW0g" target="_hRHREPDiEeSb8NZPdPkW0g"> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarDependencyLink**"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Dependency link

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.6 - Relação de tags e atributos de um dependency link na ferramenta iStar2BPMN

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	O mesmo valor que o capturado no passo 1.2
	type	4002
	source	O mesmo valor que o capturado no passo 1.3
	target	O mesmo valor que o capturado no passo 1.4
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<element>	xmi:type	IstarModel:IstarDependencyLink
	href	nome do diagrama + ".istar#//@dependencyLinks." + num (valor capturado no passo 1.13)
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor capturado a partir do atributo “points” da tag <bendpoints> do arquivo “.ood” que é filha da tag <edges> cujo atributo “element” apresenta o mesmo valor que o capturado no passo 1.2.

11.5. Conversão – Etapa 5 (Task Decompositions e Means-End Links)

Na ferramenta OpenOME, tanto as ligações do tipo task decompositions quanto as do tipo means-end links são representadas no XML do diagrama pela tag de mesmo nome: <decompositions>. A diferenciação se dá então pelo atributo “xmi:type” da tag, de acordo com a tabela abaixo.

Tabela 11.7 - Tipo de ligação adequada ao valor do atributo “xmi:type” da tag decompositions de um arquivo ood

Valor do atributo “xmi:type” da tag <decompositions>	**Tipo de ligação correspondente**
edu.toronto.cs.openome_model:AndDecomposition	task decomposition
edu.toronto.cs.openome_model:OrDecomposition	means-end link

Nesta etapa serão convertidas todas as task decompositions e means-end links, portanto para cada ocorrência da tag <decompositions> será realizado o conjunto de passos a seguir:

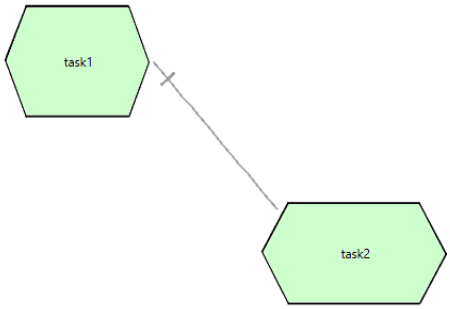
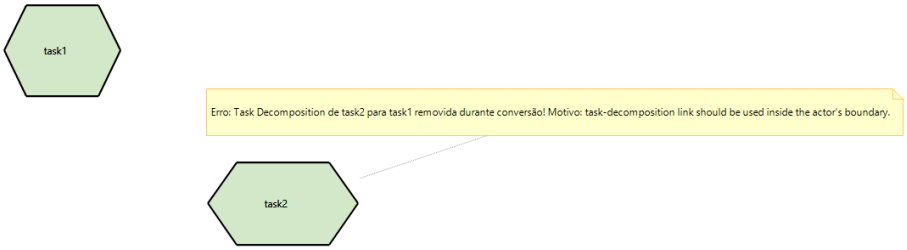
1. Para cada tag <decompositions> encontrada:
   1.1. Capturar e armazenar o valor do atributo “xmi:type” e remover a substring "edu.toronto.cs.openome_model:".
   1.2. Capturar e armazenar o valor do atributo “xmi:id”.
   1.3. Capturar e armazenar o valor do atributo “source”.
   1.4. Capturar e armazenar o valor do atributo “target”.
   1.5. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.3 a partir do atributo “source”.
   1.6. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.4 a partir do atributo “target”.
   1.7. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.5 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.8. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.6 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.9. Criar nova tag <tasksDecompositions>, se o valor capturado no passo 1.1 for igual a “AndDecomposition”. Senão, se o valor capturado no passo 1.1 for igual a “OrDecomposition” então o nome da tag a ser criada no arquivo “.istar” será <meanEndLinks>.
   1.10. Adicionar o atributo “source” à tag <tasksDecompositions> ou <meanEndLinks>. Seu valor será o armazenado no passo 1.7.
   1.11. Adicionar o atributo “target” à tag <tasksDecompositions> ou <meanEndLinks>. Seu valor será o armazenado no passo 1.8.
   1.12. Adicionar o atributo “xmi:id” à tag <tasksDecompositions> ou <meanEndLinks>. Seu valor será o armazenado no passo 1.2.

1.13. Capturar o nó pai da tag <intentions> cujo atributo “xmi:id” apresenta valor igual ao capturado no passo 1.3.

1.14. Se a tag capturada no passo anterior não for uma tag <containers>, significa que a ligação foi realizada entre elementos externos, ou seja não pertencentes a um compartment. Esta situação é permitida na ferramenta OpenOME, o que é considerada uma falha pois entra em desacordo com a linguagem i* que não permite esse tipo de situação.

1.14.1. Nesse caso a ligação será retirada e em seu lugar será criada uma nota informativa no diagrama gerado através da conversão. O resultado pode ser melhor compreendido através da tabela a seguir, onde uma ligação inválida em OpenOME é removida e surge em seu lugar uma nota informativa.

Tabela 11.8 - Inserção de nota informativa num diagrama da ferramenta iStar2BPMN onde antes havia uma ligação inválida no diagrama do OpenOME fornecido para conversão

Antes (OpenOME)	
Depois (iStar2BPMN)	

1.15. Senão, se a tag capturada no passo anterior for uma tag <containers>:

1.15.1. Capturar o valor do atributo “xmi:id” da tag <containers> capturada no passo 1.13.

1.15.2. Capturar a tag <compartments> do arquivo “.istar”, cujo atributo “xmi:id” apresenta valor igual ao capturado e armazenado no passo anterior.

1.15.3. Sobre a tag <compartments> do arquivo “.istar” obtida no passo anterior:

1.15.3.1. Capturar o valor do atributo “num”.

1.15.3.2. Obter a quantidade de filhos <tasksDecompositions> ou <meanEndLinks> da tag.

1.16. Adicionar o atributo “num” à tag <tasksDecompositions> ou <meanEndLinks>. Seu valor foi obtido no passo 1.15.3.2.

1.17. A tag criada no passo 1.9 será inserida no XML do ".istar" antes da primeira ocorrência de tag <elements> que seja filha da tag <compartments> obtida no passo 1.15.2. A tag <tasksDecompositions> ou <meanEndLinks> será então a nova tag filha da tag <compartments> no arquivo ".istar".

1.18. O seguinte bloco de código XML, que corresponde a uma task decomposition ou a um means-end link na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo ".istardiagram":

<edges xmi:type="notation:Connector"> <styles xmi:type="notation:FontStyle"/> <element/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Task Decomposition ou Means-End Link

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.9 - Relação de tags e atributos de uma task-decomposition ou um means-end link

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	O mesmo valor que o capturado no passo 1.2
	type	4003 (Means-End Link) ou 4004 (Task Decomposition Link)
	source	O mesmo valor que o capturado no passo 1.3
	target	O mesmo valor que o capturado no passo 1.4
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<element>	xmi:type	IstarModel:IstarMeanEndLink (Means-End Link) ou IstarModel:IstarTaskDecomposition (Task Decomposition Link)
	href	nome do diagrama + ".istar#//@compartments." + num (valor capturado no passo 1.15.3.1) + "/@tasksDecompositions." ou "/@meanEndLinks." + num (valor capturado no passo 1.15.3.2)
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres

	points	Valor capturado a partir do atributo “points” da tag <bendpoints> do arquivo “.ood” que é filha da tag <edges> cujo atributo “element” apresenta o mesmo valor que o capturado no passo 1.2.

11.6. Conversão – Etapa 6 (Contribution Links)

Nesta etapa serão convertidas todas as contributions. As mesmas podem ser do tipo SOMEPLUS, HELP, UNKNOWN, HURT, SOMEMINUS, BREAK, AND, OR. Na ferramenta OpenOME esse tipo de ligação é representada no XML de um diagrama por uma tag <contributions>, portanto para cada ocorrência desta tag será realizado o conjunto de passos a seguir:

1. Para cada tag <contributions> encontrada:
   1.1. Capturar e armazenar o valor do atributo “xmi:type” e remover a substring "edu.toronto.cs.openome_model:".
   1.2. Capturar e armazenar o valor do atributo “xmi:id”.
   1.3. Capturar e armazenar o valor do atributo “source”.
   1.4. Capturar e armazenar o valor do atributo “target”.
   1.5. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.3 a partir do atributo “source”.
   1.6. No “.istardiagram”, buscar e capturar a tag <children> cujo “xmi:id” apresenta o mesmo valor que o armazenado no passo 1.4 a partir do atributo “target”.
   1.7. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.5 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.8. Buscar e capturar a tag <element> filha da <children> encontrada no passo 1.6 e armazenar o valor de seu atributo “href”. Em seguida remover a substring nome do diagrama + ".istar#".
   1.9. Criar nova tag <contributionLinks> no arquivo “.istar”.
   1.10. Adicionar o atributo “type” à tag <contributionLinks> recém-criada. Seu valor será fornecido de acordo com o valor capturado no passo 1.1 e a tabela abaixo:

Tabela 11.10 - Possíveis valores para o atributo type da tag contributionLinks de um arquivo istar

Valor capturado no passo 1.1	**Valor do atributo “type”**
MakeContribution	
SomePlusContribution	SOMEPLUS
HelpContribution	HELP
UnknownContribution	UNKNOWN
HurtContribution	HURT
SomeMinusContribution	SOMEMINUS
BreakContribution	BREAK

AndContribution	AND
OrContribution	OR

1.11. Adicionar o atributo “source” à tag <contributionLinks>. Seu valor será o armazenado no passo 1.7.

1.12. Adicionar o atributo “target” à tag <contributionLinks>. Seu valor será o armazenado no passo 1.8.

1.13. Adicionar o atributo “xmi:id” à tag <contributionLinks>. Seu valor será o armazenado no passo 1.2.

1.14. Capturar o nó pai da tag <intentions> cujo atributo “xmi:id” apresenta valor igual ao capturado no passo 1.3.

1.15. Se a tag capturada no passo anterior não for uma tag <containers>, significa que a ligação partiu de um elemento externo, ou seja não pertencente a um compartment. Esta situação é permitida na ferramenta OpenOME mas é considerada uma falha pois entra em desacordo com a linguagem i* que não permite esse tipo de situação.

1.15.1. Nesse caso a ligação será retirada e em seu lugar será criada uma nota informativa no novo diagrama gerado através da conversão.

1.16. Senão, se a tag capturada no passo anterior for uma tag <containers>:

1.16.1. Capturar o valor do atributo “xmi:id” da tag <containers> capturada no passo 1.13.

1.16.2. Capturar a tag <compartments> do arquivo “.istar”, cujo atributo “xmi:id” apresenta valor igual ao capturado e armazenado no passo anterior.

1.16.3. Sobre a tag <compartments> do arquivo “.istar” obtida no passo anterior:

1.16.3.1. Capturar o valor do atributo “num”.

1.16.3.2. Obter a quantidade de filhos <contributionLinks> da tag.

1.17. Adicionar o atributo “num” à tag <contributionLinks>. Seu valor foi obtido no passo 1.16.3.2.

1.18. A tag criada no passo 1.9 será inserida no XML do “.istar” antes da primeira ocorrência de tag <elements> que seja filha da tag <compartments> obtida no passo 1.16.2. A tag <contributionLinks> será então a nova tag filha desta tag <compartments> no arquivo “.istar”.

1.19. O seguinte bloco de código XML, que corresponde a uma contribution link na ferramenta iStar2BPMN, será criado e adicionado como filho do nó raiz do arquivo “.istardiagram”:

<edges xmi:type="notation:Connector" type="**4005**"> <children xmi:type="notation:DecorationNode" type="**6005**"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <element xmi:type="**IstarModel:IstarContributionLink**"/>	Contribution link

<bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada uma das tags listadas e seus respectivos valores.

Tabela 11.11 - Relação de tags e atributos de uma contribution link na ferramenta iStar2BPMN

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	O mesmo valor que o capturado no passo 1.2
	type	4005
	source	O mesmo valor que o capturado no passo 1.3
	target	O mesmo valor que o capturado no passo 1.4
<children>	type	6005
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<element>	xmi:type	IstarModel:IstarContributionLink
	href	nome do diagrama + ".istar#//@compartments." + num (valor capturado no passo 1.16.3.1) + "/@contributionLinks." + num (valor capturado no passo 1.16.3.2)
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor capturado a partir do atributo “points” da tag <bendpoints> do arquivo “.ood” que é filha da tag <edges> cujo atributo “element” apresenta o mesmo valor que o capturado no passo 1.2.

11.7. Conversão – Etapa 7 (Limpeza final)

Nesta etapa final serão removidos do arquivo ".istar" todos os atributos criados temporariamente apenas para facilitar o procedimento de conversão. Além disso serão feitas adaptações referentes ao atributo "xmi:type" que não deve existir no arquivo ".istar".

1. De todas as ocorrências da tag <contributionLinks>:
   1.1. Remover o atributo "num".
   1.2. Remover o atributo "xmi:id".
2. De todas as ocorrências da tag <compartments>:
   2.1. Remover o atributo "num".
   2.2. Remover o atributo "xmi:id".
   2.3. Se a string correspondente ao valor do atributo "xmi:type" termina com uma substring diferente de "Actor", adicionar o atributo "type" com esse valor em uppercase.
   2.4. Remover o atributo "xmi:type".
3. De todas as ocorrências da tag <elements>:
   3.1. Remover o atributo "num".
   3.2. Remover o atributo "xmi:id".
   3.3. Se a string correspondente ao valor do atributo "xmi:type" termina com uma substring diferente de "Goal", adicionar o atributo "type" com esse valor em uppercase.
   3.4. Remover o atributo "xmi:type".

12. APÊNDICE F - Procedimento de conversão de um diagrama i* na ferramenta iStar2BPMN em formato ".istar" e ".istardiagram" para um diagrama i* na ferramenta OpenOME

Todo diagrama construído na ferramenta iStar2BPMN consiste de uma dupla de arquivos escritos em XML, sendo um de extensão ".istar" e o outro de extensão ".istardiagram". O procedimento de conversão se inicia recebendo esses dois arquivos e capturando o nome fornecido ao diagrama, ou seja o nome deste arquivo de extensão ".istar". O mesmo deve ser encontrado também no atributo "name" da tag "notation:Diagram" do XML do arquivo ".istar".

Em seguida será criado o arquivo de extensão ".ood" utilizando o nome do diagrama, capturado anteriormente, para nomear o arquivo.

O arquivo "nome do diagrama.ood" inicialmente irá conter apenas as tags mostradas abaixo, com seus respectivos atributos, resultando no seguinte XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:edu.toronto.cs.openome_model="http:///edu/toronto/cs/openome_model.ecore"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation">
  <edu.toronto.cs.openome_model:Model xmi:id="_0njCMeKoEeS5fq73HiF_gg"/>
  <notation:Diagram xmi:id="_0nszMOKoEeS5fq73HiF_gg" type="openome_model"
element="_0njCMeKoEeS5fq73HiF_gg" name="nome do diagrama.ood"
measurementUnit="Pixel">
    <styles xmi:type="notation:DiagramStyle" xmi:id="_0nszMeKoEeS5fq73HiF_gg"/>
  </notation:Diagram>
</xmi:XMI>
```

Esse arquivo ".ood" representa um diagrama vazio (sem elementos) na ferramenta OpenOME. Este código é padrão, ou seja todo diagrama vazio recém-criado na ferramenta irá conter um código semelhante ao acima, variando apenas os valores dos atributos "xmi:id", "element" e o atributo "name" da tag <notation:Diagram>. Ele servirá de base para receber tudo o que for convertido a partir dos arquivos ".istar" e ".istardiagram", sendo povoado durante o processo de conversão, construindo assim um diagrama reconhecível pela ferramenta OpenOME.

12.1. Pré-Conversão

Antes de iniciar o procedimento de conversão propriamente dito, será preciso realizar alguns ajustes no XML do arquivo ".istar". A hierarquia do XML nesse arquivo funciona da seguinte forma: existe um índice para cada tipo de tag, e esse índice é reiniciado se a ocorrência for interna a uma tag que não seja o nó raiz. Ou seja, se existem cinco ocorrências de uma mesma tag filha do nó

raiz, a primeira (lendo o arquivo XML a partir da primeira linha, ou seja de cima para baixo) receberá o índice zero, a segunda receberá o índice um e assim sucessivamente até a última ocorrência ou o final do arquivo. Porém se a mesma tag ao invés de pertencer ao nó raiz, for filha de uma tag <compartments>, a contagem começa em zero para todos os filhos da tag <compartments> como ilutrado abaixo na Tabela 12.1.

Tabela 12.1 - Sistema de índices do XML de um arquivo istar

<?xml version="1.0" encoding="UTF-8"?>	
<IstarModel:IstarModel>	
<actorLinks />.....	índice="0"
<actorLinks type="COVERS"/>.....	índice="1"
<dependencyLinks/>.....	índice="0"
<dependencyLinks/>.....	índice="1"
<compartments name="ator1">.....	índice="0"
<meanEndLinks/>.....	índice="0"
<meanEndLinks/>.....	índice="1"
<elements name="goal ator 1"/>.....	índice="0"
</compartments>	
<compartments name="ator2">.....	índice="1"
<tasksDecompositions/>.....	índice="0"
<tasksDecompositions/>.....	índice="1"
<contributionLinks />.....	índice="0"
<contributionLinks type="SOMEPLUS"/>.....	índice="1"
<contributionLinks type="HELP"/>.....	índice="2"
<contributionLinks type="UNKNOWN"/>.....	índice="3"
<contributionLinks type="HURT"/>.....	índice="4"
<contributionLinks type="SOMEMINUS"/>.....	índice="5"
<elements name="goal 2"/>.....	índice="0"
<elements name="goal1"/>.....	índice="1"
<elements name="task1" type="TASK"/>.....	índice="2"
<elements name="task2" type="TASK"/>.....	índice="3"
<elements name="softgoal2" type="SOFTGOAL"/>..	índice="4"
<elements name="resource1" type="RESOURCE"/>	índice="5"
</compartments>	
</IstarModel:IstarModel>	

Todas as referências feitas no arquivo “.istardiagram” a tags pertencentes ao arquivo “.istar” têm como base este sistema de índices. Por exemplo se no XML do arquivo “.istardiagram” for encontrado o seguinte bloco de código:

```
<children fontName="Segoe UI" type="2002" xmi:id="_9meegPPfEeSd-68uRMmb1A"
xmi:type="notation:Shape">
  <children type="5002" xmi:id="_aujx2Xh8O4yopOWi50hOn_"
xmi:type="notation:DecorationNode"/>
  <children type="7001" xmi:id="_LCZAHq6s_Z0JiBvsq5ZxcD"
xmi:type="notation:BasicCompartment"/>
  <styles xmi:id="_9Ce_O7P3XtOIsTq2odPiX9" xmi:type="notation:HintedDiagramLinkStyle"/>
  <element href="nome do diagrama.istar#//@compartments.1"
xmi:type="IstarModel:IstarCompartment"/>
  <layoutConstraint height="111" width="156" x="620" xmi:id="_vnxN8ad1gvZSE2ozhAtmjs"
xmi:type="notation:Bounds" y="15"/>
</children>
```

Sabe-se pelo atributo “xmi:type” da tag <element> que o conjunto de tags e atributos acima se trata de um Compartment, porém o tipo do mesmo (Actor, Agent, Role, Position) e seu nome são informações presentes apenas no arquivo “.istar”. Como o elemento a ser buscado é um Compartment, deve ser encontrada a ocorrência de tag <compartments> no arquivo “.istar” cujo índice é obtido através do atributo “href” da tag <element>. Nesse atributo é possível encontrar o nome do arquivo “.istar”, o nome da tag referenciada e seu índice.

Nesta etapa de pré-conversão, será adicionada a cada nó pertencente ao arquivo “.istar” um atributo de nome “href” cujo valor é igual a concatenação do nome do diagrama com “.istar#//@”, seguido pelo nome da tag, um ponto (“.”) e o valor de seu índice no “.istar”, se for um Compartment ou Element não pertencente a um Compartment. Senão, a concatenação será feita inicialmente pelo nome do diagrama com “.istar#//@compartments.”, seguido pelo índice do Compartment e “/@” seguido pelo nome da tag, um ponto (“.”) e o valor de seu índice no “.istar”.

Ainda nesta etapa será também adicionado a cada nó “compartments” ou “elements” o valor do atributo “xmi:id” presente no primeiro nó “children” do bloco no istardiagram que representa esse element ou compartment. Uma melhor compreensão pode ser obtida a partir do exemplo a seguir que mostra a situação do arquivo istar antes e depois das modificações realizadas nessa etapa da conversão.

Exemplo de um arquivo istar antes da etapa de pré-conversão:

```
<?xml version="1.0" encoding="UTF-8"?>
<IstarModel:IstarModel xmi:version="2.0" xmlns:IstarModel="IstarModel"
xmlns:xmi="http://www.omg.org/XMI">
   <compartments name="ator1">
      <elements name="goal1"/>
      <elements name="task1" type="TASK"/>
      <elements name="resource1" type="RESOURCE"/>
      <elements name="softgoal1" type="SOFTGOAL"/>
   </compartments>
</IstarModel:IstarModel>
```

O mesmo arquivo istar após a etapa de pré-conversão:

```
<?xml version="1.0" encoding="UTF-8"?>
<IstarModel:IstarModel xmi:version="2.0" xmlns:IstarModel="IstarModel"
xmlns:xmi="http://www.omg.org/XMI">
   <compartments href="caso3.istar#//@compartments.0" name="ator1"
      num="0" xmi:id="_rSy3UPi3EeSdj_L_g8dRbw">
      <elements href="caso3.istar#//@compartments.0/@elements.0"
         name="goal1" num="0" xmi:id="_sJP-MPi3EeSdj_L_g8dRbw"/>
      <elements href="caso3.istar#//@compartments.0/@elements.1"
         name="task1" num="1" type="TASK" xmi:id="_tLRb4Pi3EeSdj_L_g8dRbw"/>
      <elements href="caso3.istar#//@compartments.0/@elements.2"
         name="resource1" num="2" type="RESOURCE"
xmi:id="_t9aDcPi3EeSdj_L_g8dRbw"/>
      <elements href="caso3.istar#//@compartments.0/@elements.3"
         name="softgoal1" num="3" type="SOFTGOAL" xmi:id="_u-
tvcPi3EeSdj_L_g8dRbw"/>
   </compartments>
</IstarModel:IstarModel>
```

O arquivo istardiagram correspondente:

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram measurementUnit="Pixel" name="caso3.istardiagram"
   type="Top" xmi:id="_qsQbwPi3EeSdj_L_g8dRbw" xmi:version="2.0"
   xmlns:IstarModel="IstarModel"
   xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation"
xmlns:xmi="http://www.omg.org/XMI">
   <children fontName="Segoe UI" type="2002"
      xmi:id="_rSy3UPi3EeSdj_L_g8dRbw" xmi:type="notation:Shape">
      <children type="5002" xmi:id="_rS8BQPi3EeSdj_L_g8dRbw"
xmi:type="notation:DecorationNode"/>
      <children type="7001" xmi:id="_rS8oUPi3EeSdj_L_g8dRbw"
xmi:type="notation:BasicCompartment">
         <children fontName="Segoe UI" type="3001"
            xmi:id="_sJP-MPi3EeSdj_L_g8dRbw" xmi:type="notation:Shape">
            <children type="5003" xmi:id="_sJP-Mvi3EeSdj_L_g8dRbw"
xmi:type="notation:DecorationNode"/>
            <element
               href="caso3.istar#//@compartments.0/@elements.0"
xmi:type="IstarModel:IstarElement"/>
            <layoutConstraint height="56" width="76" x="100"
```

```
            xmi:id="_sJP-Mfi3EeSdj_L_g8dRbw"
            xmi:type="notation:Bounds" y="45"/>
        </children>
        <children fontName="Segoe UI" type="3001"
          xmi:id="_tLRb4Pi3EeSdj_L_g8dRbw" xmi:type="notation:Shape">
          <children type="5003" xmi:id="_tLRb4vi3EeSdj_L_g8dRbw"
xmi:type="notation:DecorationNode"/>
          <element
            href="caso3.istar#//@compartments.0/@elements.1"
xmi:type="IstarModel:IstarElement"/>
          <layoutConstraint height="66" width="81" x="20"
            xmi:id="_tLRb4fi3EeSdj_L_g8dRbw"
            xmi:type="notation:Bounds" y="120"/>
        </children>
        <children fontName="Segoe UI" type="3001"
          xmi:id="_t9aDcPi3EeSdj_L_g8dRbw" xmi:type="notation:Shape">
          <children type="5003" xmi:id="_t9aDcvi3EeSdj_L_g8dRbw"
xmi:type="notation:DecorationNode"/>
          <element
            href="caso3.istar#//@compartments.0/@elements.2"
xmi:type="IstarModel:IstarElement"/>
          <layoutConstraint height="61" width="91" x="130"
            xmi:id="_t9aDcfi3EeSdj_L_g8dRbw"
            xmi:type="notation:Bounds" y="110"/>
        </children>
        <children fontName="Segoe UI" type="3001"
          xmi:id="_u-tvcPi3EeSdj_L_g8dRbw" xmi:type="notation:Shape">
          <children type="5003" xmi:id="_u-tvcvi3EeSdj_L_g8dRbw"
xmi:type="notation:DecorationNode"/>
          <element
            href="caso3.istar#//@compartments.0/@elements.3"
xmi:type="IstarModel:IstarElement"/>
          <layoutConstraint height="61" width="86" x="90"
            xmi:id="_u-tvcfi3EeSdj_L_g8dRbw"
            xmi:type="notation:Bounds" y="190"/>
        </children>
      </children>
      <styles xmi:id="_rSy3Ufi3EeSdj_L_g8dRbw"
xmi:type="notation:HintedDiagramLinkStyle"/>
      <element href="caso3.istar#//@compartments.0"
```

```
xmi:type="IstarModel:IstarCompartment"/>
      <layoutConstraint height="276" width="256" x="50"
        xmi:id="_rSy3Uvi3EeSdj_L_g8dRbw" xmi:type="notation:Bounds" y="15"/>
   </children>
   <styles xmi:id="_qsQbwfi3EeSdj_L_g8dRbw" xmi:type="notation:DiagramStyle"/>
   <element href="caso3.istar#/" xmi:type="IstarModel:IstarModel"/>
</notation:Diagram>
```

12.2. Conversão – Etapa 1 (Containers com ou sem Elements internos)

Cada ocorrência de nó “compartments” no arquivo istar representa um Compartment no diagrama, com seu nome e tipo (Actor, Agent, Role, Position). No arquivo istardiagram encontram-se as informações visuais referentes a posição (coordenadas x e y) do Compartment, assim como sua altura(height) e largura (width).

1. Para cada ocorrência de nó “compartments” no arquivo istar:
   1.1. Capturar e armazenar o valor do atributo “name”.
   1.2. Capturar e armazenar o valor do atributo “type”.
   1.3. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.5. Sobre o nó “element” encontrado no passo anterior:
      1.5.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.
      1.5.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:
         1.5.2.1. Capturar e armazenar o valor do atributo “x”.
         1.5.2.2. Capturar e armazenar o valor do atributo “y”.
         1.5.2.3. Capturar e armazenar o valor do atributo “width”.
         1.5.2.4. Capturar e armazenar o valor do atributo “height”.
   1.6. De posse de todos os dados capturados e armazenados, construir o nó de nome “containers” que será filho de “edu.toronto.cs.openome_model:Model” no arquivo ood. A seguir estão descritos os passos para a construção do mesmo.
      1.6.1. Sobre o nó de nome “containers” recém-criado no arquivo ood:
         1.6.1.1. Adicionar o atributo “xmi:type” cujo valor será obtido de acordo com a Tabela 12.2 abaixo.

Tabela 12.2 - Possíveis valores para o atributo xmi:type da tag containers de um arquivo ood

Valor capturado no passo 1.2	Valor adequado ao atributo “xmi:type”
-	edu.toronto.cs.openome_model:Actor
AGENT	edu.toronto.cs.openome_model:Agent
ROLE	edu.toronto.cs.openome_model:Role
POSITION	edu.toronto.cs.openome_model:Position

1.6.1.2. Adicionar o atributo “xmi:id” cujo valor foi obtido na etapa de pré-conversão.

1.6.1.3. Adicionar o atributo “name” cujo valor foi capturado e armazenado no passo 1.1.

1.6.1.4. O seguinte bloco de código XML, que corresponde a um compartment (Actor, Agent, Role, Position) na ferramenta OpenOME, será criado no arquivo ood e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (isto é, será inserido acima da primeira ocorrência de nó “styles”):

<children xmi:type="notation:Shape"> <children xmi:type="notation:DecorationNode"/> <children xmi:type="notation:BasicCompartment"> <styles xmi:type="notation:SortingStyle"/> <styles xmi:type="notation:FilteringStyle"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Compartment

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.3 - Relação de tags e atributos de um compartment na ferramenta OpenOME

Tag	Atributo	Valor
<children> (primeiro children)	type	2001 (Actor), 2002 (Agent), 2003 (Position) ou 2004 (Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:Shape
	fontName	Segoe UI
	element	Mesmo valor que o do atributo “xmi:id” do nó “containers” gerado no passo 1.6.1.2
<children> (segundo children)	type	5005 (Actor), 5010 (Agent), 5015 (Position) ou 5020 (Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<children> (terceiro children)	type	7001 (Actor), 7002 (Agent), 7003 (Position) ou 7004 (Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:BasicCompartment
<styles>	xmi:type	notation:SortingStyle
	xmi:id	Cadeia de 23 caracteres
<styles>	xmi:type	notation:FilteringStyle
	xmi:id	Cadeia de 23 caracteres
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados nos passos 1.5.2.1 a 1.5.2.4

	y	
	width	
	height	

1.6.1.5. Se o nó “compartments” atual possui filhos, para cada nó “elements” filho de “compartments”:

1.6.1.5.1. Capturar e armazenar o valor do atributo “name”.

1.6.1.5.2. Capturar e armazenar o valor do atributo “type”.

1.6.1.5.3. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.

1.6.1.5.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.

1.6.1.5.5. Sobre o nó “element” encontrado no passo anterior:

1.6.1.5.5.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.

1.6.1.5.5.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:

1.6.1.5.5.2.1. Capturar e armazenar o valor do atributo “x”.

1.6.1.5.5.2.2. Capturar e armazenar o valor do atributo “y”.

1.6.1.5.5.2.3. Capturar e armazenar o valor do atributo “width”.

1.6.1.5.5.2.4. Capturar e armazenar o valor do atributo “height”.

1.6.1.5.6. De posse de todos os dados capturados e armazenados, construir o nó de nome “intentions” que será filho de “containers” criado no passo 1.6. A seguir estão descritos os passos para a construção do mesmo.

1.6.1.5.6.1. Sobre o nó de nome “intentions” recém-criado no arquivo ood:

1.6.1.5.6.1.1. Adicionar o atributo “xmi:type” cujo valor será obtido de acordo com a Tabela 12.4 abaixo.

Tabela 12.4 - Possíveis valores para o atributo xmi:type da tag intentions de um arquivo ood

Valor capturado	Valor adequado ao atributo “xmi:type”
-	edu.toronto.cs.openome_model:Goal
TASK	edu.toronto.cs.openome_model:Task
SOFTGOAL	edu.toronto.cs.openome_model:Softgoal
RESOURCE	edu.toronto.cs.openome_model:Resource

1.6.1.5.6.1.2. Adicionar o atributo “xmi:id” cujo valor foi obtido na etapa de pré-conversão.

1.6.1.5.6.1.3. Adicionar o atributo “name” cujo valor foi capturado e armazenado no passo 1.6.1.5.1.

1.6.1.5.6.1.4. O seguinte bloco de código XML, que corresponde a uma intention (Goal, Softgoal, Task, Resource) na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”,

respeitando a hierarquia (ou seja, será inserido acima da primeira ocorrência de nó “styles”):

<children xmi:type="notation:Shape"> <children xmi:type="notation:DecorationNode"/> <children xmi:type="notation:DecorationNode"> <layoutConstraint xmi:type="notation:Location"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Intention

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.5 - Relação de tags e atributos de uma intention interna na ferramenta OpenOME

Tag	Atributo	Valor
<children> (primeiro children)	type	3001(Goal interna a Actor), 3002(Softgoal interna a Actor), 3003(Resource interna a Actor), 3004(Task interna a Actor), 3005(Goal interna a Agent), 3006(Softgoal interna a Agent), 3007(Resource interna a Agent), 3008(Task interna a Agent), 3009(Goal interna a Position), 3010(Softgoal interna a Position), 3011(Resource interna a Position), 3012(Task interna a Position), 3013(Goal interna a Role), 3014(Softgoal interna a Role), 3015(Resource interna a Role) e 3016(Task interna a Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:Shape
	fontName	Segoe UI
	element	Mesmo valor que o do atributo “xmi:id” do nó “intentions”. Valor gerado no passo 1.6.1.5.6.1.2.
<children> (segundo children)	type	5001(Goal interna a Actor), 5002(Softgoal interna a Actor), 5003(Resource interna a Actor), 5004(Task interna a Actor), 5006(Goal interna a Agent), 5007(Softgoal interna Agent), 5008(Resource interna a Agent), 5009(Task interna a Agent), 5011(Goal interna a Position),

		5012(Softgoal interna a Position), 5013(Resource interna a Position), 5014(Task interna a Position), 5016(Goal interna a Role), 5017(Softgoal interna a Role), 5018(Resource interna a Role) e 5019(Task interna a Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<children> (terceiro children)	type	5025(Goal interna a Actor), 5026(Softgoal interna a Actor), 5027(Resource interna a Actor), 5028(Task interna a Actor), 5029(Goal interna a Agent), 5030(Softgoal interna Agent), 5031(Resource interna a Agent), 5032(Task interna a Agent), 5033(Goal interna a Position), 5034(Softgoal interna a Position), 5035(Resource interna a Position), 5036(Task interna a Position), 5037(Goal interna a Role), 5038(Softgoal interna a Role), 5039(Resource interna a Role) e 5040(Task interna a Role)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados nos passos 1.6.1.5.5.2.1 a 1.6.1.5.5.2.4
	y	
	width	
	height	

12.3. Conversão – Etapa 2 (Elements externos)

Elements externos correspondem a nós de nome “intentions” que não são internos a algum nó “containers”.

1. Para cada nó “elements” que não é filho de algum “compartments”:
   1.1. Capturar e armazenar o valor do atributo “name”.
   1.2. Capturar e armazenar o valor do atributo “type”.
   1.3. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.4.1. Sobre o nó “element” encontrado no passo anterior:

1.4.2. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.

1.4.3. Sobre o nó “layoutConstraint” encontrado no passo anterior:

1.4.3.1. Capturar e armazenar o valor do atributo “x”.

1.4.3.2. Capturar e armazenar o valor do atributo “y”.

1.4.3.3. Capturar e armazenar o valor do atributo “width”.

1.4.3.4. Capturar e armazenar o valor do atributo “height”.

1.5. De posse de todos os dados capturados e armazenados, construir o nó de nome “intentions” que será filho do nó “edu.toronto.cs.openome_model:Model” do ood. A seguir estão descritos os passos para a construção do mesmo.

1.5.1. Sobre o nó de nome “intentions” recém-criado no arquivo ood:

1.5.1.1. Adicionar o atributo “xmi:type” cujo valor será obtido de acordo com a Tabela 12.4.

1.5.1.2. Adicionar o atributo “xmi:id” cujo valor foi obtido na etapa de pré-conversão.

1.5.1.3. Adicionar o atributo “name” cujo valor foi capturado e armazenado no passo 1.1.

1.5.1.4. O seguinte bloco de código XML, que corresponde a uma intention externa (Goal, Softgoal, Task, Resource) na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (ou seja, será inserido acima da primeira ocorrência de nó “styles”):

<children xmi:type="notation:Shape"> <children xmi:type="notation:DecorationNode"/> <children xmi:type="notation:DecorationNode"> <layoutConstraint xmi:type="notation:Location"/> </children> <layoutConstraint xmi:type="notation:Bounds"/> </children>	Intention

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.6 - Relação de tags e atributos de uma intention externa na ferramenta OpenOME

Tag	Atributo	Valor
<children> (primeiro children)	type	2005(Goal), 2006(Softgoal), 2007(Task) e 2008(Resource)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:Shape
	fontName	Segoe UI
	element	Mesmo valor que o do atributo “xmi:id” do nó

		“intentions”. Valor gerado no passo 1.5.1.2
<children> (segundo children)	type	5021(Goal), 5022(Softgoal), 5023(Task) e 5024(Resource)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<children> (terceiro children)	type	5041(Goal), 5042(Softgoal), 5043(Task) e 5044(Resource)
	xmi:id	Cadeia de 23 caracteres
	xmi:type	notation:DecorationNode
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<layoutConstraint>	xmi:type	notation:Bounds
	xmi:id	Cadeia de 23 caracteres
	x	Valores capturados nos passos 1.4.3.1 a 1.4.3.4
	y	
	width	
	height	

12.4. Conversão – Etapa 3 (Actor Associations)

1. Para cada nó “actorLinks” encontrado no arquivo istar:
   1.1. Capturar e armazenar o valor do atributo “type”.
   1.2. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “actorLinks” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “actorLinks” na etapa de pré-conversão.
   1.4. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
   1.5. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.6. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
   1.7. De posse de todos os dados capturados e armazenados, construir o nó de nome “associations” que será filho do nó “edu.toronto.cs.openome_model:Model” do ood. A seguir estão descritos os passos para a construção do mesmo.
      1.7.1. Sobre o nó de nome “associations” recém-criado no arquivo ood:
         1.7.1.1. Adicionar o atributo “xmi:type” cujo valor será obtido de acordo com a Tabela 12.7 abaixo.

Tabela 12.7 - Possíveis valores para o atributo xmi:type da tag associations de um arquivo ood	
Valor capturado	Valor adequado ao atributo “xmi:type”
-	edu.toronto.cs.openome_model:IsAAssociation
COVERS	edu.toronto.cs.openome_model:CoversAssociation
ISPARTOF	edu.toronto.cs.openome_model:IsPartOfAssociation
OCCUPIES	edu.toronto.cs.openome_model:OccupiesAssociation
PLAYS	edu.toronto.cs.openome_model:PlaysAssociation
INS	edu.toronto.cs.openome_model:INSAssociation

1.7.1.2. Adicionar o atributo “xmi:id” cujo valor é uma sequência de 23 caracteres.

1.7.1.3. Adicionar o atributo “source”, preparado no passo 1.2.

1.7.1.4. Adicionar o atributo “target”, preparado no passo 1.3.

1.8. O seguinte bloco de código XML, que corresponde a uma actor association na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (ou seja, será inserido abaixo da primeira ocorrência de nó “styles”):

<edges xmi:type="notation:Connector"> <children xmi:type="notation:DecorationNode"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Actor association

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.8 - Relação de tags e atributos de uma actor association na ferramenta OpenOME		
Tag	**Atributo**	**Valor**
<edges>	xmi:type	notation:Connector
	xmi:id	Cadeia de 23 caracteres
	type	4014(ISA), 4015(COVERS), 4016(OCCUPIES), 4017(ISPARTOF), 4018(PLAYS), 4019(INS)
	element	Mesmo valor obtido no passo 1.7.1.2
	source	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “source” da tag “associations” criada no passo 1.7.
	target	Valor do atributo “xmi:id” do nó “children” cujo

		atributo “element” apresenta valor igual ao do atributo “target” da tag “associations” criada no passo 1.7.
<children>	xmi:type	notation:DecorationNode
	xmi:id	Cadeia de 23 caracteres
	type	6013(ISA), 6014(COVERS), 6015(OCCUPIES), 6016(ISPARTOF), 6017(PLAYS), 6018(INS)
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor obtido no passo 1.6

12.5. Conversão – Etapa 4 (Dependency Links)

1. Para cada nó “dependencyLinks” do arquivo istar:
   1.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “dependencyLinks” na etapa de pré-conversão.
   1.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “dependencyLinks” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.5. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
   1.6. De posse de todos os dados capturados e armazenados, construir o nó de nome “dependencies” que será filho do nó “edu.toronto.cs.openome_model:Model” do ood. A seguir estão descritos os passos para a construção do mesmo.
      1.6.1. Sobre o nó de nome “dependencies” recém-criado no arquivo ood:
         1.6.1.1. Adicionar o atributo “xmi:type” de valor igual a “edu.toronto.cs.openome_model:Dependency”.
         1.6.1.2. Adicionar o atributo “xmi:id” cujo valor é uma sequência de 23 caracteres.
         1.6.1.3. Adicionar o atributo “dependencyTo”, cujo valor foi preparado no passo 1.1.

1.6.1.4. Adicionar o atributo “dependencyFrom”, cujo valor foi preparado no passo 1.2.

1.7. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “dependencyTo” do nó “dependencies” criado no passo 1.6.

1.8. Adicionar ao nó encontrado no passo anterior o atributo “dependencyFrom” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.9. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “dependencyFrom” do nó “dependencies” criado no passo 1.6.

1.10. Adicionar ao nó encontrado no passo anterior o atributo “dependencyTo” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.11. O seguinte bloco de código XML, que corresponde a um dependency link na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (ou seja, será inserido abaixo da primeira ocorrência de nó “styles”):

<edges xmi:type="notation:Connector" type="4001"> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Dependency link

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.9 - Relação de tags e atributos de um dependency link na ferramenta OpenOME

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	Cadeia de 23 caracteres
	type	4001 (DEPENDENCY LINK)
	element	Mesmo valor obtido no passo 1.6.1.2
	source	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “source” da tag “dependencies” criada no passo 1.6.
	target	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “target” da tag “dependencies” criada no

		passo 1.6.
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor obtido no passo 1.5

12.6. Conversão – Etapa 5 (Task Decompositions e Means-End Links)

1. Para cada ocorrência de tag “tasksDecompositions” ou “meanEndLinks” no arquivo istar:
   1.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “tasksDecompositions” ou “meanEndLinks” na etapa de pré-conversão.
   1.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “tasksDecompositions” ou “meanEndLinks” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.5. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
   1.6. De posse de todos os dados capturados e armazenados, construir o nó de nome “decompositions” que será filho do nó “edu.toronto.cs.openome_model:Model” do ood. A seguir estão descritos os passos para a construção do mesmo.
      1.6.1. Sobre o nó de nome “decompositions” recém-criado no arquivo ood:
         1.6.1.1. Adicionar o atributo “xmi:type” de valor igual a “edu.toronto.cs.openome_model:AndDecomposition” quando se tratar de uma “tasksDecompositions” ou “edu.toronto.cs.openome_model:OrDecomposition” quando se tratar de uma “meanEndLinks”.
         1.6.1.2. Adicionar o atributo “xmi:id” cujo valor é uma sequência de 23 caracteres.
         1.6.1.3. Adicionar o atributo “source”, cujo valor foi preparado no passo 1.1.
         1.6.1.4. Adicionar o atributo “target”, cujo valor foi preparado no passo 1.2.
   1.7. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “source” do nó “contributions” criado no passo 1.6.

1.8. Adicionar ao nó encontrado no passo anterior o atributo “decompositions” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.9. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “target” do nó “contributions” criado no passo 1.6.

1.10. Adicionar ao nó encontrado no passo anterior o atributo “parentDecompositions” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.11. O seguinte bloco de código XML, que corresponde a um task decomposition link ou um means-end link na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (ou seja, será inserido abaixo da primeira ocorrência de nó “styles”):

<edges xmi:type="notation:Connector"> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Task Decomposition Link ou Means-End Link

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.10 - Relação de tags e atributos de uma task decomposition link ou um means-end link na ferramenta OpenOME

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	Cadeia de 23 caracteres
	type	4002 (TASK DECOMPOSITION LINK) ou 4003 (MEANS-END LINK)
	element	Mesmo valor obtido no passo 1.6.1.2
	source	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “source” da tag “decompositions” criada no passo 1.6.
	target	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “target” da tag “decompositions” criada no passo 1.6.
<styles>	xmi:type	notation:FontStyle

	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor obtido no passo 1.5

12.7. Conversão – Etapa 6 (Contribution Links)

1. Para cada ocorrência de tag “contributionLinks” no arquivo istar:
   1.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “contributionLinks” na etapa de pré-conversão.
   1.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do istardiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “contributionLinks” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo istardiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.5. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
   1.6. De posse de todos os dados capturados e armazenados, construir o nó de nome “contributions” que será filho do nó “edu.toronto.cs.openome_model:Model” do ood. A seguir estão descritos os passos para a construção do mesmo.
      1.6.1. Sobre o nó de nome “contributions” recém-criado no arquivo ood:
         1.6.1.1. Adicionar o atributo “xmi:type” cujo valor será obtido de acordo com a Tabela 12.11 abaixo.

Tabela 12.11 - Possíveis valores para o atributo xmi:type da tag contributions de um arquivo ood

Valor capturado	**Valor adequado ao atributo “xmi:type”**
-	edu.toronto.cs.openome_model:MakeContribution
BREAK	edu.toronto.cs.openome_model:BreakContribution
UNKNOWN	edu.toronto.cs.openome_model:UnknownContribution
SOMEPLUS	edu.toronto.cs.openome_model:SomePlusContribution
SOMEMINUS	edu.toronto.cs.openome_model:SomeMinusContribution
AND	edu.toronto.cs.openome_model:AndContribution
OR	edu.toronto.cs.openome_model:OrContribution
HELP	edu.toronto.cs.openome_model:HelpContribution
HURT	edu.toronto.cs.openome_model:HurtContribution

         1.6.1.2. Adicionar o atributo “xmi:id” cujo valor é uma sequência de 23 caracteres.

1.6.1.3. Adicionar o atributo “source”, cujo valor foi preparado no passo 1.1.

1.6.1.4. Adicionar o atributo “target”, cujo valor foi preparado no passo 1.2.

1.7. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “source” do nó “contributions” criado no passo 1.6.

1.8. Adicionar ao nó encontrado no passo anterior o atributo “contributesFrom” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.9. Buscar e capturar no parcialmente criado arquivo ood o nó de nome “intentions” ou “containers” cujo atributo “xmi:id” apresenta valor igual ao valor do atributo “target” do nó “contributions” criado no passo 1.6.

1.10. Adicionar ao nó encontrado no passo anterior o atributo “contributesTo” cujo valor é igual ao gerado no passo 1.6.1.2. Se o mesmo já possui esse atributo, então concatenar o valor separando por um espaço em branco (“ “).

1.11. O seguinte bloco de código XML, que corresponde a um contribution link na ferramenta OpenOME, será criado e adicionado como filho do nó “notation:Diagram”, respeitando a hierarquia (ou seja, será inserido abaixo da primeira ocorrência de nó “styles”):

<edges xmi:type="notation:Connector"> <children xmi:type="notation:DecorationNode"> <layoutConstraint xmi:type="notation:Location"/> </children> <styles xmi:type="notation:FontStyle"/> <bendpoints xmi:type="notation:RelativeBendpoints"/> </edges>	Contribution link

Como alguns atributos foram removidos para facilitar a visualização, segue abaixo a relação completa dos atributos pertencentes a cada um dos nós listados e seus respectivos valores.

Tabela 12.12 - Relação de tags e atributos de um contribution link na ferramenta OpenOME

Tag	Atributo	Valor
<edges>	xmi:type	notation:Connector
	xmi:id	Cadeia de 23 caracteres
	type	4005(HELP), 4006(HURT), 4007(MAKE), 4008(BREAK), 4009(SOMEPLUS), 4010(SOMEMINUS), 4011(UNKNOWN), 4012(AND), 4013(OR)
	element	Mesmo valor obtido no passo 1.6.1.2
	source	Valor do atributo “xmi:id” do nó “children” cujo

		atributo “element” apresenta valor igual ao do atributo “source” da tag “associations” criada no passo 1.6.
	target	Valor do atributo “xmi:id” do nó “children” cujo atributo “element” apresenta valor igual ao do atributo “target” da tag “associations” criada no passo 1.6.
<children>	xmi:type	notation:DecorationNode
	xmi:id	Cadeia de 23 caracteres
	type	6004(HELP), 6005(HURT), 6006(MAKE), 6007(BREAK), 6008(SOMEPLUS), 6009(SOMEMINUS), 6010(UNKNOWN), 6011(AND), 6012(OR)
<layoutConstraint>	xmi:type	notation:Location
	xmi:id	Cadeia de 23 caracteres
<styles>	xmi:type	notation:FontStyle
	xmi:id	Cadeia de 23 caracteres
	fontName	Segoe UI
<bendpoints>	xmi:type	notation:RelativeBendpoints
	xmi:id	Cadeia de 23 caracteres
	points	Valor obtido no passo 1.5

13. APÊNDICE G - Procedimento de conversão de um diagrama BPMN na ferramenta iStar2BPMN em formato ".bpmn" e ".bpmndiagram" para um diagrama BPMN na ferramenta Bizagi

Todo diagrama BPMN construído na ferramenta iStar2BPMN consiste de uma dupla de arquivos escritos em XML, sendo um de extensão ".bpmn" e o outro de extensão ".bpmndiagram". O procedimento de conversão se inicia recebendo esses dois arquivos e capturando o nome fornecido ao diagrama, ou seja o nome deste arquivo de extensão ".bpmn". O mesmo deve ser encontrado também no atributo "name" da tag "notation:Diagram" do XML do arquivo ".bpmn".

Em seguida será criado o arquivo de extensão ".bpmn" (Bizagi) utilizando o nome do diagrama, capturado anteriormente, para nomear o arquivo.

O arquivo "nome do diagrama.bpmn" (Bizagi) inicialmente irá conter apenas as tags mostradas abaixo, com seus respectivos atributos, resultando no seguinte XML:

```
<?xml version="1.0"?>
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="_2015052008317"
targetNamespace="http://www.bizagi.com/definitions/_2015052008317"
xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL">
  <process id="Id_3b4187b9-0536-4819-a8f0-40b9640cc875" name="Main Process">
    <documentation />
  </process>
  <collaboration id="Id_a90d7f37-dc70-4699-9397-544c9c226223" name="Diagram 1">
    <participant id="Id_4c491030-aa7a-4513-92de-737209ae0b05" name="Main Process"
processRef="Id_3b4187b9-0536-4819-a8f0-40b9640cc875">
      <extensionElements>
        <bizagi:BizagiExtensions xmlns:bizagi="http://www.bizagi.com/bpmn20">
          <bizagi:BizagiProperties>
            <bizagi:BizagiProperty name="bgColor" value="White" />
            <bizagi:BizagiProperty name="borderColor" value="Black" />
            <bizagi:BizagiProperty name="isMainParticipant" />
          </bizagi:BizagiProperties>
        </bizagi:BizagiExtensions>
      </extensionElements>
    </participant>
  </collaboration>
  <BPMNDiagram id="Diagram_0c4a572c-4d76-4c75-aca0-894266a44581"
xmlns="http://www.omg.org/spec/BPMN/20100524/DI">
    <BPMNPlane id="DiagramElement_2c7ef70e-9930-401c-87b0-2ecaf53f5d7d"
```

```
bpmnElement="Id_a90d7f37-dc70-4699-9397-544c9c226223">
    <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
    <BPMNShape id="DiagramElement_38212f35-145b-4df9-a4a1-5e73841d55f6"
bpmnElement="Id_4c491030-aa7a-4513-92de-737209ae0b05" isHorizontal="true">
      <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
      <Bounds x="30" y="30" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      <BPMNLabel id="DiagramElement_f3fa967c-e66b-44bb-8cbd-729e33217df3"
labelStyle="Style_79e23c31-0d64-40fe-b996-1da6367442ff">
        <extension xmlns="http://www.omg.org/spec/DD/20100524/DI" />
        <Bounds x="0" y="0" width="0" height="0"
xmlns="http://www.omg.org/spec/DD/20100524/DC" />
      </BPMNLabel>
    </BPMNShape>
  </BPMNPlane>
  <BPMNLabelStyle id="Style_79e23c31-0d64-40fe-b996-1da6367442ff">
    <Font name="Segoe UI" size="10" isBold="true" isItalic="false" isUnderline="false"
isStrikeThrough="false" xmlns="http://www.omg.org/spec/DD/20100524/DC" />
  </BPMNLabelStyle>
 </BPMNDiagram>
</definitions>
```

Esse arquivo representa um diagrama vazio (sem elementos) na ferramenta Bizagi. Este código é padrão, ou seja todo diagrama vazio recém-criado na ferramenta irá conter um código semelhante ao acima, variando apenas os valores dos atributos “id” e “targetNamespace” da tag “definitions”, “id” da tag “process”, “id” da tag “collaboration”, “id” e “processRef” da tag “participant”, “bpmnElement” da tag “BPMNPlane”, “bpmnElement” da tag “BPMNShape”. Ele servirá de base para receber tudo o que for convertido a partir dos arquivos “.bpmn” (iStar2BPMN) e “.bpmndiagram”, sendo povoado durante o processo de conversão, construindo assim um diagrama reconhecível pela ferramenta Bizagi.

13.1. Pré-Conversão

Antes de iniciar o procedimento de conversão propriamente dito, será preciso realizar alguns ajustes no XML do arquivo “.bpmn”. A hierarquia do XML nesse arquivo funciona de forma análoga a de um arquivo .istar, ou seja existe um índice para cada tipo de tag, e esse índice é reiniciado se a ocorrência for interna a uma tag que não seja o nó raiz. Todas as referências feitas no arquivo “.bpmndiagram” a tags pertencentes ao arquivo “.bpmn” têm como base este sistema de índices (raciocínio análogo ao explicado na Tabela 12.1 - Sistema de índices do XML de um arquivo istar).

Nesta etapa de pré-conversão, será adicionada a cada nó pertencente ao arquivo ".bpmn" um atributo de nome "href" cujo valor é igual a concatenação do nome do diagrama com ".bpmn#//@", seguido pelo nome da tag, um ponto (".") e o valor de seu índice no ".bpmn", se for uma tag do tipo "pools", "messageFlows" ou "associations". Senão, a concatenação será feita inicialmente pelo nome do diagrama com ".bpmn#//@pools.", seguido pelo índice da Pool e "/@" seguido pelo nome da tag, um ponto (".") e o valor de seu índice no ".bpmn".

Ainda nesta etapa será também adicionado a cada nó "pools" ou "lanes" o valor do atributo "xmi:id" presente no primeiro nó "children" do bloco no istardiagram que representa essa Pool ou essa Lane.

13.2. Conversão – Etapa 1 (Pools e Lanes)

Cada ocorrência de nó "pools" no arquivo bpmn representa uma Pool no diagrama. No arquivo bpmndiagram encontram-se as informações visuais referentes a posição (coordenadas x e y) da Pool, assim como sua altura(height) e largura (width).

2. Para cada ocorrência de nó "pools" no arquivo bpmn:
   - 2.1. Capturar e armazenar o valor do atributo "name".
   - 2.2. Capturar e armazenar o valor do atributo "href" criado na etapa de pré-conversão.
   - 2.3. Buscar o nó de nome "element" do arquivo bpmndiagram cujo atributo "href" apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   - 2.4. Sobre o nó "element" encontrado no passo anterior:
     - 2.4.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome "layoutConstraint".
     - 2.4.2. Sobre o nó "layoutConstraint" encontrado no passo anterior:
       - 2.4.2.1. Capturar e armazenar o valor do atributo "x".
       - 2.4.2.2. Capturar e armazenar o valor do atributo "y".
       - 2.4.2.3. Capturar e armazenar o valor do atributo "width".
       - 2.4.2.4. Capturar e armazenar o valor do atributo "height".
   - 2.5. De posse de todos os dados capturados e armazenados, inserir no arquivo ".bpmn" do Bizagi que está sendo montado, os blocos de código XML que juntos formam uma Pool na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico "Estudo do XMI de um diagrama BPMN no Bizagi" na subseção "Pool".
   - 2.6. Se o nó "pools" atual contém filhos, para cada nó "lanes" filho de "pools":
     - 2.6.1. Capturar e armazenar o valor do atributo "name".
     - 2.6.2. Capturar e armazenar o valor do atributo "href" criado na etapa de pré-conversão.
     - 2.6.3. Buscar o nó de nome "element" do arquivo istardiagram cujo atributo "href" apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
     - 2.6.4. Sobre o nó "element" encontrado no passo anterior:
     - 2.6.5. Encontrar o nó irmão (filho de mesmo nó pai) de nome "layoutConstraint".
       - 2.6.5.1. Sobre o nó "layoutConstraint" encontrado no passo anterior:
         - 2.6.5.1.1. Capturar e armazenar o valor do atributo "x".

2.6.5.1.2. Capturar e armazenar o valor do atributo “y”.

2.6.5.1.3. Capturar e armazenar o valor do atributo “width”.

2.6.5.1.4. Capturar e armazenar o valor do atributo “height”.

2.6.6. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam uma Lane na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” na subseção “**Lane**” do presente trabalho.

13.3. Conversão – Etapa 2 (Tasks)

1. No arquivo “.bpmn” do diagrama iStar2BPMN, para cada ocorrência de nó “nodes” cujo atributo “xsi:type” apresenta valor igual a "BPMNModel:BPMNTask":
   1.1. Capturar e armazenar o valor do atributo “name”.
   1.2. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
   1.3. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.4. Sobre o nó “element” encontrado no passo anterior:
      1.4.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.
      1.4.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:
         1.4.2.1. Capturar e armazenar o valor do atributo “x”.
         1.4.2.2. Capturar e armazenar o valor do atributo “y”.
         1.4.2.3. Capturar e armazenar o valor do atributo “width”.
         1.4.2.4. Capturar e armazenar o valor do atributo “height”.
   2.7. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam uma Task na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” na subseção “**Task**”.

13.4. Conversão – Etapa 3 (Events)

1. No arquivo “.bpmn” do diagrama iStar2BPMN, para cada ocorrência de nó “nodes” cujo atributo “xsi:type” apresenta valor igual a "BPMNModel:BPMNInitialEvent", "BPMNModel:BPMNIntermediateEvent" ou "BPMNModel:BPMNFinalEvent":
   1.1. Capturar e armazenar o valor do atributo “label”.
   1.2. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
   1.3. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.4. Sobre o nó “element” encontrado no passo anterior:

1.4.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.

1.4.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:

1.4.2.1. Capturar e armazenar o valor do atributo “x”.

1.4.2.2. Capturar e armazenar o valor do atributo “y”.

1.4.2.3. Capturar e armazenar o valor do atributo “width”.

1.4.2.4. Capturar e armazenar o valor do atributo “height”.

1.5. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam um Event na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” nas subseções “**Start Event**”, “**Intermediate Event**” e “**Final Event**”. O tipo do Event (startEvent, intermediateThrowEvent ou endEvent) depende do valor do atributo “xsi:type” capturado no passo 1.

13.5. Conversão – Etapa 4 (Gateway)

1. No arquivo “.bpmn” do diagrama iStar2BPMN, para cada ocorrência de nó “nodes” cujo atributo “xsi:type” apresenta valor igual a "BPMNModel:BPMNGateway":

1.1. Capturar e armazenar o valor do atributo “label”.

1.2. Capturar e armazenar o valor do atributo “condition”.

1.3. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.

1.4. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.

1.5. Sobre o nó “element” encontrado no passo anterior:

1.5.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.

1.5.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:

1.5.2.1. Capturar e armazenar o valor do atributo “x”.

1.5.2.2. Capturar e armazenar o valor do atributo “y”.

1.5.2.3. Capturar e armazenar o valor do atributo “width”.

1.5.2.4. Capturar e armazenar o valor do atributo “height”.

2.8. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam um Gateway na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” na subseção “**Gateway**”.

13.6. Conversão – Etapa 5 (Data Object e Sub-Process)

1. No arquivo “.bpmn” do diagrama iStar2BPMN, para cada ocorrência de nó “nodes” cujo atributo “xsi:type” apresenta valor igual a "BPMNModel:BPMNDataObject" ou "BPMNModel:BPMNSubProcess":

1.1. Capturar e armazenar o valor do atributo “label”.
1.2. Capturar e armazenar o valor do atributo “name”.
1.3. Capturar e armazenar o valor do atributo “href” criado na etapa de pré-conversão.
1.4. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
1.5. Sobre o nó “element” encontrado no passo anterior:
1.5.1. Encontrar o nó irmão (filho de mesmo nó pai) de nome “layoutConstraint”.
1.5.2. Sobre o nó “layoutConstraint” encontrado no passo anterior:
1.5.2.1. Capturar e armazenar o valor do atributo “x”.
1.5.2.2. Capturar e armazenar o valor do atributo “y”.
1.5.2.3. Capturar e armazenar o valor do atributo “width”.
1.5.2.4. Capturar e armazenar o valor do atributo “height”.
2.9. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam um Data Object ou um Sub-Process na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” nas subseções “**Data Object**” e “**Sub-Process**”. O tipo da tag a ser criada (dataObject ou subProcess) depende do valor do atributo “xsi:type” capturado no passo 1.

13.7. Conversão – Etapa 6 (Message Flows)

2. Para cada ocorrência de tag “messageFlows” no arquivo bpmn:
2.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
2.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
2.3. Capturar e armazenar o valor do atributo “label”.
2.4. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
2.5. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
2.6. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
2.7. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam um Message Flow na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” na subseção “**Message Flow**”.

13.8. Conversão – Etapa 7 (Associations)

1. Para cada ocorrência de tag “associations” no arquivo bpmn:
   1.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
   1.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
   1.4. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.5. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.
   1.6. De posse de todos os dados capturados e armazenados, inserir no arquivo “.bpmn” do Bizagi que está sendo montado, os blocos de código XML que juntos formam uma Association na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico “APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi” na subseção “**Association**”.

13.9. Conversão – Etapa 8 (Sequence Flows)

1. Para cada ocorrência de tag “sequenceFlows” no arquivo bpmn:
   1.1. Capturar e armazenar o valor do atributo “source”. Esse valor será substituído pelo valor do atributo “source” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
   1.2. Capturar e armazenar o valor do atributo “target”. Esse valor será substituído pelo valor do atributo “target” do nó “edges” do bpmndiagram que contém um nó filho “element” de atributo “href” com valor igual ao do atributo “href” que foi adicionado ao “sequenceFlows” na etapa de pré-conversão.
   1.3. Capturar e armazenar o valor do atributo “label”.
   1.4. Capturar e armazenar o valor do atributo “href” gerado na etapa de pré-conversão.
   1.5. Buscar o nó de nome “element” do arquivo bpmndiagram cujo atributo “href” apresenta o mesmo valor que foi capturado e armazenado no passo anterior.
   1.6. Capturar e armazenar o valor do atributo “points” do nó “bendpoints” irmão do nó “element” capturado no passo anterior.

1.7. De posse de todos os dados capturados e armazenados, inserir no arquivo ".bpmn" do Bizagi que está sendo montado, os blocos de código XML que juntos formam um Sequence Flow na ferramenta Bizagi. Os mesmos podem ser encontrados no tópico "APÊNDICE D - Estudo do XMI de um diagrama BPMN no Bizagi" na subseção "**Sequence Flow**".

14. APÊNDICE H - Procedimento de conversão de um diagrama BPMN na ferramenta Bizagi em formato ".bpmn" para um diagrama BPMN na ferramenta iStar2BPMN

Dado um arquivo de extensão ".bpmn", escrito em XML, que represente um diagrama qualquer reconhecível pela ferramenta Bizagi. O procedimento de conversão se inicia com a captura e armazenamento do nome fornecido ao diagrama, ou seja o nome deste arquivo de extensão ".bpmn". O mesmo deve ser encontrado no atributo "name" da tag "collaboration" do XML do arquivo ".bpmn".

Em seguida são criados os arquivos de extensão ".bpmn" e ".bpmndiagram" utilizando o nome do diagrama, capturado anteriormente, para nomear os arquivos. Ambos receberão o mesmo nome.

O arquivo "nome do diagrama.bpmn" inicialmente contém apenas uma tag "BPMNModel:BPMNModel" com atributos "xmi:version", "xmlns:xmi" e "xmlns:IstarModel", resultando nas seguintes linhas de código:

```
<?xml version="1.0" encoding="UTF-8"?>
<BPMNModel:BPMNModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:BPMNModel="bpmnmodel"/>
```

O arquivo "nome do diagrama.bpmndiagram" inicial contém apenas o seguinte código (variando apenas o valor do atributo "xmi:id" da tag "notation:Diagram"):

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:BPMNModel="bpmnmodel"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation"
xmi:id="_0gFnYARqEeWgM9BG1iY8Fg" type="Bpmnmodel"
name="blankDiagram.bpmndiagram" measurementUnit="Pixel">
  <styles xmi:type="notation:DiagramStyle" xmi:id="_0gFnYQRqEeWgM9BG1iY8Fg"/>
  <element xmi:type="BPMNModel:BPMNModel" href="blankDiagram.bpmn#/"/>
</notation:Diagram>
```

Juntos, esses dois arquivos representam um diagrama vazio (sem elementos) na ferramenta iStar2BPMN. Estes códigos são padrões, ou seja todo diagrama vazio recém-criado na ferramenta irá conter os códigos acima. Eles servirão de base para receber tudo o que for convertido a partir do arquivo ".bpmn" (Bizagi), sendo povoados durante o processo de conversão, construindo assim um diagrama reconhecível pela ferramenta iStar2BPMN.

14.1. Conversão – Etapa 1 (Pools e Lanes)

Nessa primeira etapa todas as ocorrências de Pools serão tratadas. Cada ocorrência é determinada por uma tag <process> no XML do arquivo “.bpmn” (Bizagi). Para cada Pool no diagrama pode vir a existir uma ou mais Lanes, ou seja para cada tag “process” pode existir uma lista de tags “lanes” não vazia.

Para cada ocorrência de tags process e lane será feito o conjunto de passos a seguir:

1. Para cada tag process (que não apresenta atributo "name" com valor "Main Process"):
   - 1.1. Capturar e armazenar o valor do atributo "id" de process;
   - 1.2. Buscar tag participant cujo atributo "processRef" apresenta valor igual ao capturado no passo anterior;
   - 1.3. Sobre a tag participant encontrada no passo anterior:
     - 1.3.1. Capturar e armazenar o valor do atributo "name" de participant;
     - 1.3.2. Capturar e armazenar o valor do atributo "id" de participant;
   - 1.4. Buscar tag BPMNShape cujo atributo bpmnElement apresenta valor igual ao capturado no passo anterior (1.3);
   - 1.5. Buscar tag filha direta de nome Bounds;
   - 1.6. Armazenar da tag encontrada no passo anterior os valores dos atributos "x", "y", "width" e "height";
   - 1.7. Criar o código XML correspondente a uma Pool na ferramenta iStar2BPMN. O mesmo pode ser encontrado na seção “**Pool**” de “APÊNDICE C - Estudo do XMI de um diagrama BPMN no ;
   - 1.8. Se a tag process atual contém filhos, ou seja se ocorre a tag “laneSet”, então para cada tag “lane” filha de process:
     - 1.8.1. Capturar e armazenar o valor do atributo "name" de lane;
     - 1.8.2. Capturar e armazenar o valor do atributo "id" de lane;
     - 1.8.3. Buscar tag BPMNShape cujo atributo bpmnElement apresenta valor igual ao capturado no passo anterior;
     - 1.8.4. Buscar tag filha direta de nome Bounds;
     - 1.8.5. Armazenar da tag encontrada no passo anterior os valores dos atributos "x", "y", "width" e "height";
     - 1.8.6. Serão subtraídos 50 pixels do valor de “x” capturado no passo anterior. Esta correção ocorre devido a existência de uma borda no Bizagi que não existe no iStar2BPMN.
     - 1.8.7. Será subtraído do valor de “y” da lane o valor de “y” da pool a que pertence esta lane, pois o “y” da lane é relativo e não absoluto como seria no Bizagi.
     - 1.8.8. Criar o código XML correspondente a uma Lane na ferramenta iStar2BPMN. O mesmo pode ser encontrado na seção “**Lane**” de “APÊNDICE C - Estudo do XMI de um diagrama BPMN no ”;

14.2. Conversão – Etapa 2 (Elementos)

Aqui serão convertidos os elementos, ou seja todos os start events, intermediate events, end events, tasks, gateways, data objects e sub-processes. O início do procedimento é o mesmo para todos: nele serão capturadas informações relevantes quanto a Pool que contém o elemento.

1. Para cada nó “process” existente:
    1.1. Capturar e armazenar o valor do atributo “id”;
    1.2. Localizar o nó “participant” cujo atributo “processRef” apresenta valor igual ao capturado no passo anterior. Capturar e armazenar o valor do atributo “id” deste nó “participant” encontrado.
    1.3. Localizar o nó “BPMNShape” cujo atributo “bpmnElement” apresenta valor igual ao capturado no passo anterior.
    1.4. Capturar e armazenar os valores dos atributos “x” e “y” do nó “Bounds” filho direto do nó “BPMNShape” encontrado no passo anterior.
    1.5. De posse do nó “process” referente à Pool, seu id, e as coordenadas x e y:
        1.5.1. Para cada nó “startEvent”, “intermediateThrowEvent”, “endEvent”, “task”, “exclusiveGateway”, “dataObject” ou “subProcess” filho do nó “process” atual:
            1.5.1.1. Capturar e armazenar o atributo “name” (se houver);
            1.5.1.2. Capturar e armazenar o atributo “id”;
            1.5.1.3. Localizar o nó “BPMNShape” cujo atributo “bpmnElement” apresenta valor igual ao capturado no passo anterior.
            1.5.1.4. Capturar o nó “Bounds” filho direto do “BPMNShape” capturado no passo anterior e a partir dele capturar e armazenar os valores de “x”, “y”, “width” e “height” referentes ao presente elemento.
            1.5.1.5. A partir do valor “y” capturado no passo anterior é possível determinar a que Lane pertence o elemento. A Lane pai será a que contiver o maior valor “y” que seja menor do que o valor “y” do elemento.
            1.5.1.6. Capturar e armazenar os valores “x” e “y” da Lane pai do presente elemento. A mesma foi encontrada no passo anterior.
        1.8.9. De posse do “id”, “name”, “x”, “y”, “width” e “height” do elemento, além das coordenadas “x” e “y” da pool e da lane a que o elemento pertence, o XML referente ao mesmo será criado de acordo com o formato adequado, como explicado no tópico “APÊNDICE C - Estudo do XMI de um diagrama BPMN no ”;

14.3. Conversão – Etapa 3 (Links)

Nesta etapa serão convertidas todas as ligações entre elementos, ou seja todos os sequence flows, message flows e associations. Os nós “sequenceFlow” e “association” são filhos de algum nó “process” correspondente a Pool onde se iniciam, enquanto o nó “messageFlow” será sempre filho de algum nó “collaboration”.

1. Para cada nó “sequenceFlow” existente:
   1.1. Capturar e armazenar o atributo “name” (se houver);
   1.2. Capturar e armazenar o atributo “id”;
   1.3. Capturar e armazenar o valor do atributo “sourceRef”;
   1.4. Localizar o nó “nodes”, no arquivo bpmn (iStar2BPMN) que está sendo criado, cujo atributo temporário “id” apresenta o mesmo valor que o capturado no passo anterior em “sourceRef”. Em outras palavras, localizar o “nodes” correspondente à origem do Link.
   1.5. Sobre o nó “nodes” encontrado no passo anterior:
      1.5.1. Capturar e armazenar o atributo temporário “href”.
   1.6. Localizar o nó “element”, no arquivo bpmndiagram (iStar2BPMN) que está sendo criado, cujo atributo “href” apresenta o mesmo valor que o capturado no passo anterior (1.5.1). Ou seja encontrar o nó “element” correspondente ao “nodes” que representa a origem do Link.
   1.7. A partir do nó “children” que é pai do nó “element” encontrado no passo anterior, capturar e armazenar o atributo “xmi:id” do mesmo.
   1.8. Capturar e armazenar o valor do atributo “targetRef”;
   1.9. Localizar o nó “nodes”, no arquivo bpmn (iStar2BPMN) que está sendo criado, cujo atributo temporário “id” apresenta o mesmo valor que o capturado no passo anterior em “targetRef”. Em outras palavras, localizar o “nodes” correspondente ao destino do Link.
   1.10. Sobre o nó “nodes” encontrado no passo anterior:
      1.10.1. Capturar e armazenar o atributo temporário “href”.
   1.11. Localizar o nó “element”, no arquivo bpmndiagram (iStar2BPMN) que está sendo criado, cujo atributo “href” apresenta o mesmo valor que o capturado no passo anterior (1.9.1). Ou seja encontrar o nó “element” correspondente ao “nodes” que representa o destino do Link.
   1.12. A partir do nó “children” que é pai do nó “element” encontrado no passo anterior, capturar e armazenar o atributo “xmi:id” do mesmo.
   1.13. Obter os índices da Pool onde se inicia o Link, da Lane onde se inicia o Link e do próprio Link através de contagem da quantidade de ocorrências da tag “sequenceFlow” até o presente momento.
   1.14. De posse do valor do atributo “xmi:id” do nó “children” referente ao nó de origem do link, do valor de “xmi:id” do nó “children” referente ao destino do link, do “id” e “name” do link e dos valores dos atributos “href” do nó origem e do nó destino, montar o código XML correspondente a um sequence flow no iStar2BPMN. O mesmo pode ser encontrado na seção “**Sequence Flow**” de “APÊNDICE C - Estudo do XMI de um diagrama BPMN no ”;
2. Para cada nó “association” ou “messageFlow” existente:
   2.1. Capturar e armazenar o atributo “name” (se houver);
   2.2. Capturar e armazenar o atributo “id”;
   2.3. Capturar e armazenar o valor do atributo “sourceRef”;

2.4. Localizar o nó “nodes”, no arquivo bpmn (iStar2BPMN) que está sendo criado, cujo atributo temporário “id” apresenta o mesmo valor que o capturado no passo anterior em “sourceRef”. Em outras palavras, localizar o “nodes” correspondente à origem do Link.

2.5. Sobre o nó “nodes” encontrado no passo anterior:

2.5.1. Capturar e armazenar o atributo temporário “href”.

2.6. Localizar o nó “element”, no arquivo bpmndiagram (iStar2BPMN) que está sendo criado, cujo atributo “href” apresenta o mesmo valor que o capturado no passo anterior (1.5.1). Ou seja encontrar o nó “element” correspondente ao “nodes” que representa a origem do Link.

2.7. A partir do nó “children” que é pai do nó “element” encontrado no passo anterior, capturar e armazenar o atributo “xmi:id” do mesmo.

2.8. Capturar e armazenar o valor do atributo “targetRef”;

2.9. Localizar o nó “nodes”, no arquivo bpmn (iStar2BPMN) que está sendo criado, cujo atributo temporário “id” apresenta o mesmo valor que o capturado no passo anterior em “targetRef”. Em outras palavras, localizar o “nodes” correspondente ao destino do Link.

2.10. Sobre o nó “nodes” encontrado no passo anterior:

2.10.1. Capturar e armazenar o atributo temporário “href”.

2.11. Localizar o nó “element”, no arquivo bpmndiagram (iStar2BPMN) que está sendo criado, cujo atributo “href” apresenta o mesmo valor que o capturado no passo anterior (1.9.1). Ou seja encontrar o nó “element” correspondente ao “nodes” que representa o destino do Link.

2.12. A partir do nó “children” que é pai do nó “element” encontrado no passo anterior, capturar e armazenar o atributo “xmi:id” do mesmo.

2.13. Obter os índices da Pool onde se inicia o Link, da Lane onde se inicia o Link e do próprio Link através de contagem da quantidade de ocorrências da tag “association” ou “messageFlow” até o presente momento.

2.14. De posse do valor do atributo “xmi:id” do nó “children” referente ao nó de origem do link, do valor de “xmi:id” do nó “children” referente ao destino do link, do “id” e “name” do link e dos valores dos atributos “href” do nó origem e do nó destino, montar o código XML correspondente a uma association ou message flow no iStar2BPMN. O mesmo pode ser encontrado na seção “**Association**” ou “**Message Flow**” de “APÊNDICE C - Estudo do XMI de um diagrama BPMN no ”. Diferentemente de um Sequence Flow, o código de uma Association ou de um Message Flow em um arquivo bpmn (iStar2BPMN) será adicionado como filho do nó “BPMNModel:BPMNModel” e não pertencerá a uma Lane.

14.4. Conversão – Etapa 4 (Limpeza final)

Nesta etapa final serão removidos do arquivo “.bpmn” (iStar2BPMN) todos os atributos criados temporariamente apenas para facilitar o procedimento de conversão.

1. De todas as ocorrências da tag <pools>:

1.1. Remover o atributo “id”.

1.2. Remover o atributo “href”.
1.3. Remover o atributo “indice”.
1.4. Remover o atributo “x”.
1.5. Remover o atributo “y”.

2. De todas as ocorrências da tag <lanes>:
2.1. Remover o atributo “id”.
2.2. Remover o atributo “href”.
2.3. Remover o atributo “indice”.
2.4. Remover o atributo “x”.
2.5. Remover o atributo “y”.

3. De todas as ocorrências da tag <nodes>:
3.1. Remover o atributo “id”.
3.2. Remover o atributo “href”.