PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 18/12

Aplicação da Engenharia Reversa e Reengenharia de Software no Desenvolvimento de *plugins* para a Ferramenta *Oryx*

Henrique Prado Sousa
Julio Cesar Sampaio do Prado Leite

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 18/12 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dezembro 2012

Aplicação da Engenharia Reversa e Reengenharia de Software no Desenvolvimento de *Plugins* para a Ferramenta *Oryx*

Henrique Prado Sousa
hsousa@inf.puc-rio.br

Julio Cesar Sampaio do Prado Leite
julio@inf.puc-rio.br

Abstract. This paper presents the application of software reengineering on the *Oryx* tool with the objective of developing plugins that provide new features. First reverse engineering was applied in order to obtain the necessary knowledge about the tool, enabling the subsequent execution of reengineering, which implements the additional new features (plugins), as well as adapts existing ones. The developed plugins aims to offer the capacity of modeling using the i* language together the BPMN notation in the same model. This document registers the reverse engineering and details the reengineering of implementation. We conclude showing the tests results used to help validate the new plugins of Oryx tool.

Keywords: iStar, BPMN, goal modeling, business process modeling, reverse engineering, reengineering, *Oryx*.

Resumo: Este trabalho apresenta a aplicação da reengenharia de software na ferramenta *Oryx* com o objetivo de desenvolver *plugins* que ofereçam novos recursos. Primeiramente aplicamos a engenharia reversa de forma a obtermos o conhecimento necessário sobre a ferramenta, possibilitando a posterior aplicação da reengenharia de software, na qual implementos novos recursos (*plugins)* adicionais, bem como adaptamos os existentes. Os *plugins* desenvolvidos visam oferecer a capacidade de modelagem das duas linguagens i* e notação BPMN em um mesmo modelo. Neste documento estão registrados os conhecimentos extraídos a partir da engenharia reversa e detalhado o processo de implementação a partir da reengenharia. Ao final são apresentados os testes para validar os novos *plugins* em funcionamento na ferramenta *Oryx*.

Palavras-chave: iStar, BPMN, modelagem de objetivos, modelagem de processos de negócio, engenharia reversa, reengenharia, *Oryx*.

Responsável por publicações:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1. Motivação

Este trabalho é o esforço inicial para o desenvolvimento de uma ferramenta e definição de uma linguagem de modelagem que ofereça recursos para a integração de modelos de processos de negócio e objetivos. Neste trabalho os esforços foram concentrados na implementação de *plugins* que contenham os elementos de linguagem necessários para a posterior análise da integração.

Para o desenvolvimento deste software, foi selecionada como base (reuso) a ferramenta *Oryx*, que possui código aberto e oferece suporte para múltiplas linguagens de modelagem, além de permitir o desenvolvimento de modelos customizáveis a partir da descrição de elementos no padrão json (*http://www.json.org/).* Apesar de ser aberta, pouca documentação referente à infraestrutura do software é oferecida e a documentação disponível encontra-se em outras línguas como inglês e alemão.

Este trabalho utilizou estratégias de engenharia reversa e reengenharia de software na evolução da ferramenta de modelagem *Oryx*, baseado na proposta de [Leite, 1996]. Durante o trabalho foi recuperado o desenho da ferramenta a partir das descobertas possibilitadas pela engenharia reversa, especificados os novos requisitos, reprojetado o desenho da ferramenta, e então concluída a reengenharia ao implementar *plugins* que adicionam a capacidade de criação de diagramas com elementos de modelagem de objetivos juntamente com elementos de modelagem de negócio. Essa funcionalidade foi implementada utilizando a linguagem de modelagem de objetivos i* (*iStar*/iEstrela) e a notação BPMN.

Este trabalho oferece um exemplo de aplicação da técnica de reengenharia de software utilizando a engenharia reversa, seguindo o método proposto por [Leite, 1996]. Também serve como uma fonte de informações sobre a ferramenta *Oryx,* composta por explicações detalhadas sobre sua instalação, infraestrutura e passos para customizá-la. A partir das informações sobre a customização da ferramenta presentes neste trabalho, é possível criar novos *plugins* (ou estender *plugins* existentes), permitindo assim, o desenvolvimento de novos recursos de modelagem e linguagens com muito menos esforço, uma vez que todo o núcleo da ferramenta *Oryx* é reutilizado.

Também se encontram registrados comentários referentes às dificuldades enfrentadas durante as atividades que compõem este trabalho, bem como as soluções encontradas para contorná-las. Além disso, descrevemos de forma resumida o *framework* de modelagem intencional i* e a notação de modelagem de processos de negócio BPMN.

1.2. Apresentação dos elementos envolvidos

Nesta seção são brevemente apresentadas as notações i*, BPMN e a ferramenta *Oryx*, que sevem como base para este trabalho.

1.2.1. iEstrela (iStar / i*)

O *Framework* i* é uma linguagem para modelagem de objetivos que foi projetada para modelar contextos organizacionais com base nos relacionamentos de dependência

entre os atores participantes. A ideia central do i* é representar através de modelos os atores e as dependências que eles têm uns com os outros para que metas próprias sejam atingidas [Oliveira *et al*, 2008].

Diferentemente dos processos de modelagem típicos, tais como diagrama de fluxo de dados ou diagramas de atividades, os quais focam no fluxo de informação ou controle de fluxo, os modelos em i* possuem um nível de abstração maior, abordando uma dimensão estratégica sem detalhes sobre como as tarefas são realizadas. O propósito desta linguagem é, assim, outro, permitindo uma análise aprofundada sobre os objetivos organizacionais e como os objetivos provocam impactos uns nos outros e nas tarefas executadas pela organização.

O conceito mais relevante no i* é o conceito de objetivo: "Um objetivo é uma condição ou estado de interesse no mundo que um ator gostaria de alcançar" [Tradução livre, [Yu, 1995] e [Yu, 2001] *apud* [Oliveira *et al*, 2008]]. Desta forma, atores dependem uns dos outros para que seus objetivos sejam alcançados, recursos sejam fornecidos, tarefas sejam realizadas e *softgoals* sejam "razoavelmente satisfeitos" [Oliveira *et al*, 2008].

Os objetivos podem ser classificados em duas formas: *hardgoal* e *sofgoal*. Os *Hardgoals* são definidos como objetivos que, ou são satisfeitos plenamente, ou não são satisfeitos. Já os *softgoals* são o oposto, uma vez que estão sujeitos a interpretação por serem imprecisos e subjetivos, ou seja, podem ser considerados satisfeitos em diferentes escalas, de acordo com a interpretação e/ou contexto.

Os diagramas principais do i* são o modelo SD – modelo de dependências estratégicas (*Strategic Dependency Model*), e o modelo SR – modelo de razões estratégicas (*Strategic Reasoning Model*).

No modelo SD, são expressos exclusivamente os relacionamentos de dependência estratégica entre os atores enquanto que no modelo SR são detalhadas as razões estratégicas internas dos atores. As subseções seguintes detalham os modelo SR e SD.

1.2.1.1. Modelo SD

No modelo SD, os atores possuem dependência entre si para alcançar objetivos, executar tarefas e fornecer recursos. Ao depender de outro ator, o ator dependente obtém vantagens das oportunidades que são disponibilizadas através dos atores que prestam o suporte [Yu, 2009]. Ao mesmo tempo em que o dependente é beneficiado, ele torna-se vulnerável caso não sejam atendidas as suas expectativas. Essas dependências são consideradas estratégicas para os atores interessados, já que eles podem ser beneficiados ou prejudicados em seu bem-estar. Atores poderiam escolher que dependências ter, de acordo com seu julgamento em relação a potenciais ganhos e perdas [Yu, 2009].

Um modelo SD é um grafo onde os nós representam atores (agentes, posições, papéis), e cada dependência representa um relacionamento de cooperação entre dois atores, onde um ator chamado de *depender* depende de outro chamado de *dependee*. O elo da dependência, chamado de *dependum*, é o objeto da dependência que pode ser: um objetivo, um *softgoal*, uma tarefa ou um recurso, e esse é sempre uma entidade física ou informacional. Conforme mencionado anteriormente, as relações de dependência mostram as vulnerabilidades do *depender*, pois o *dependee* pode falhar no cumprimento do acordo e, por isso, cada dependência tem um grau de importância ("*critical*", "*committed*" ou "*open*", em ordem decrescente de importância) para refletir sua relevância [Oliveira *et al*, 2008].

Dentro do modelo SD, é possível expressar quatro tipos de dependências estratégicas: dependência de meta; dependência de tarefa; dependência de recurso; e dependência de *softgoal*. A Figura 1 ilustra os quatro tipos de relacionamento:

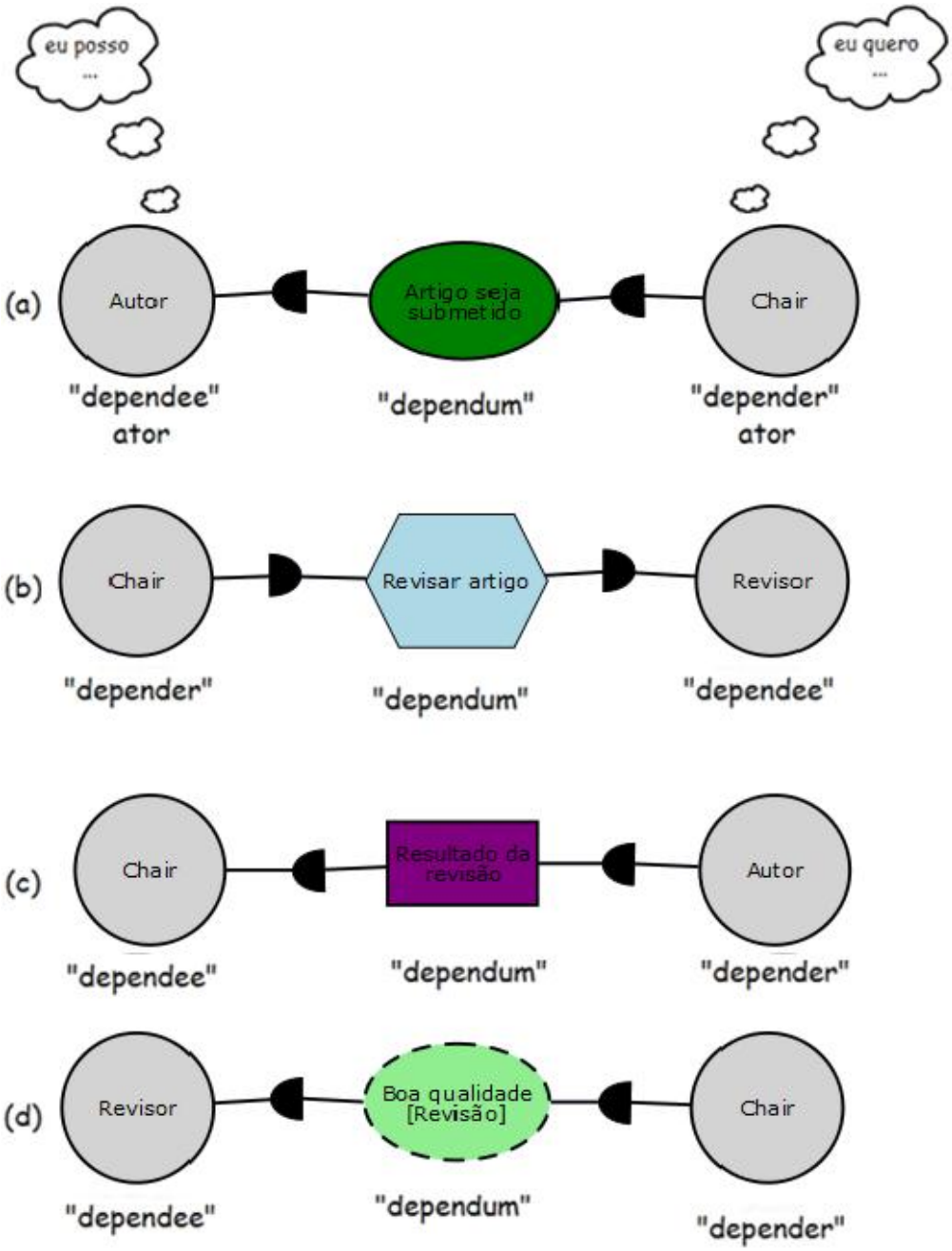


Figura 1 – Relacionamentos de dependência do modelo SD [Adaptação de 7 *apud* 5]

O exemplo (a) faz uma ilustração do significado de uma dependência entre dois atores: um ator "quer" alguma coisa que outro ator "pode" suprir. Essa é uma representação de "dependência de objetivo": o CHAIR da conferência (*depender*) depende do AUTOR (*dependee*) para que "Artigo seja Submetido".

O exemplo (b) faz a representação de uma "dependência por tarefa". O ator CHAIR (*depender*) depende do ator REVISOR (*dependee*) executar a tarefa "Revisar Artigo".

O exemplo (c) faz a representação de uma "dependência por recurso". O AUTOR (*depender*) depende do CHAIR (*dependee*) fornecer o "Resultado da Revisão".

O exemplo (d) faz a representação de uma "dependência por *softgoal*". O ator CHAIR (*depender*) depende do ator REVISOR (*dependee*) executar a revisão com boa qualidade. Observe que neste exemplo está indicada a direção de dependência.

Os tipos das dependências refletem o grau de liberdade existente no relacionamento. Na dependência por objetivo, cabe ao *dependee* toda e qualquer decisão para o cumprimento do objetivo. Na dependência por tarefa, o *dependee* executa a tarefa da maneira como o *depender* deseja e, por isso, cabe ao *depender* as decisões de como a tarefa deve ser executada. Na dependência por recurso, o grau de liberdade é nulo, ou seja, o *dependee* deve fornecer o recurso exatamente como o *depender* deseja. Na dependência por softgoal, o *depender* tem a decisão final de aceitar ou não a meta alcançada, porém ele usa o beneficio do "*know-how*" (habilidades e conhecimentos) do *dependee* [Oliveira *et al*, 2008].

1.2.1.2. Modelo SR

O modelo SR é um grafo que pode conter metas, metas-flexíveis, tarefas e recursos. O modelo SR representa o raciocínio estratégico dentro dos limites de um ator [Oliveira *et al*, 2010]. Nesse modelo, são atribuídos objetivos, tarefas, recursos e *softgoals* para cada ator, desta vez como elementos intencionais internos que o ator deseja alcançar [Cardoso *et al*, 2011].

Os objetivos e planos podem ser decompostos em um conjunto de outros planos ou objetivos. Esses elementos também podem ser relacionados com *softgoals* através de relacionamentos qualitativos classificados com "+", para representar um relacionamento de contribuição positiva ou "-", para representar um relacionamento de contribuição negativa, ambos relacionados à satisfação do *softgoal*. A relação meios-fim (*means-end*) é mapeada graficamente por um vetor direcionado para o nó fim, o qual geralmente é um objetivo, sendo os nós meio sempre tarefas (mais de uma tarefa no caso de existirem alternativas mutuamente exclusivas - tipo "XOR" - para o alcance do objetivo). Os modelos SR aparecem dentro de um campo delimitador de elementos, que está diretamente associado a um ator, o qual os objetivos e planos estão sendo analisados para determinar como eles podem ser satisfeitos/alcançados (Figura 2) [Susi *et al*, 2005].

A decomposição de tarefa é mapeada graficamente pela representação dos nós subcomponentes da tarefa, os quais são ligados ao nó principal (a tarefa maior) através de um elo representado por uma reta com um pequeno segmento de reta que corta esse elo. Os nós dos subcomponentes podem ser: objetivos, tarefas, recursos e *softgoals*. Esses subcomponentes têm o mesmo significado daqueles já utilizados no modelo SD.

Um objetivo é uma condição ou estado de desejo no mundo que o ator pretende alcançar e uma tarefa especifica um modo particular de realizar algo. Quando uma tarefa é especificada como uma subtarefa de outra, a subtarefa restringe a tarefa maior para um curso de ação em particular.

Um recurso é uma entidade física ou informacional. As principais características a serem consideradas neste elemento são sua disponibilidade e por quem foi disponibilizado, no caso de uma dependência externa. Um *softgoal* também é uma condição no mundo que o ator deseja alcançar, mas diferentemente de um objetivo concreto (ou simplesmente objetivo), o critério para essa condição ser atingida não é definido objetivamente, sendo sujeito à interpretação.

A Figura 2 exemplifica um modelo SR desenhado na ferramenta *Oryx* (que é produto deste trabalho). Este modelo captura parte do raciocínio (*rationale*) envolvido

em um de sistema de acionamento de seguro. Um médico deve submeter o plano de tratamento para a companhia de seguros para aprovação (*Approval of treatment*) prévia ou o tratamento não poderá ser reembolsado. A companhia de seguros verifica se o tipo de tratamento é coberto pelo contrato e se o tratamento proposto está de acordo com a opinião médica. "The *Claims manager*" está apto a aprovar o tratamento a partir da tarefa "*Approve treatment*". Esta tarefa consiste em dois componentes: o subobjetivo "*Treatment be assessed*" e a subtarefa "*Sign approval document*". Observe que estão modeladas somente as tarefas que são consideradas importantes o suficiente para serem preocupações estratégicas do ator. Uma forma de ter o plano de tratamento avaliado é permitir ao ator "*Claims clerk*" fazê-lo. Outra forma é o próprio "*Claims manager*" realizar o trabalho. Esta alternativa requer que o "*Claims manager*" verifique no contrato (*Assess treatment*) quais as condições médicas e o plano de tratamento são cobertos para o paciente, as políticas em vigor e quais regras para planejar um tratamento alinhado a posterior avaliação da seguradora (*Verify patient policy*). O plano de tratamento alinhado pode ser executado por alguém com conhecimentos médicos especiais, como o Medical assessor ou pode ser executado pelo próprio "*Claims manager*", a partir do acesso a informações históricas de pacientes (*Revier paient medical history*) em repositórios (*Medical reports department*).

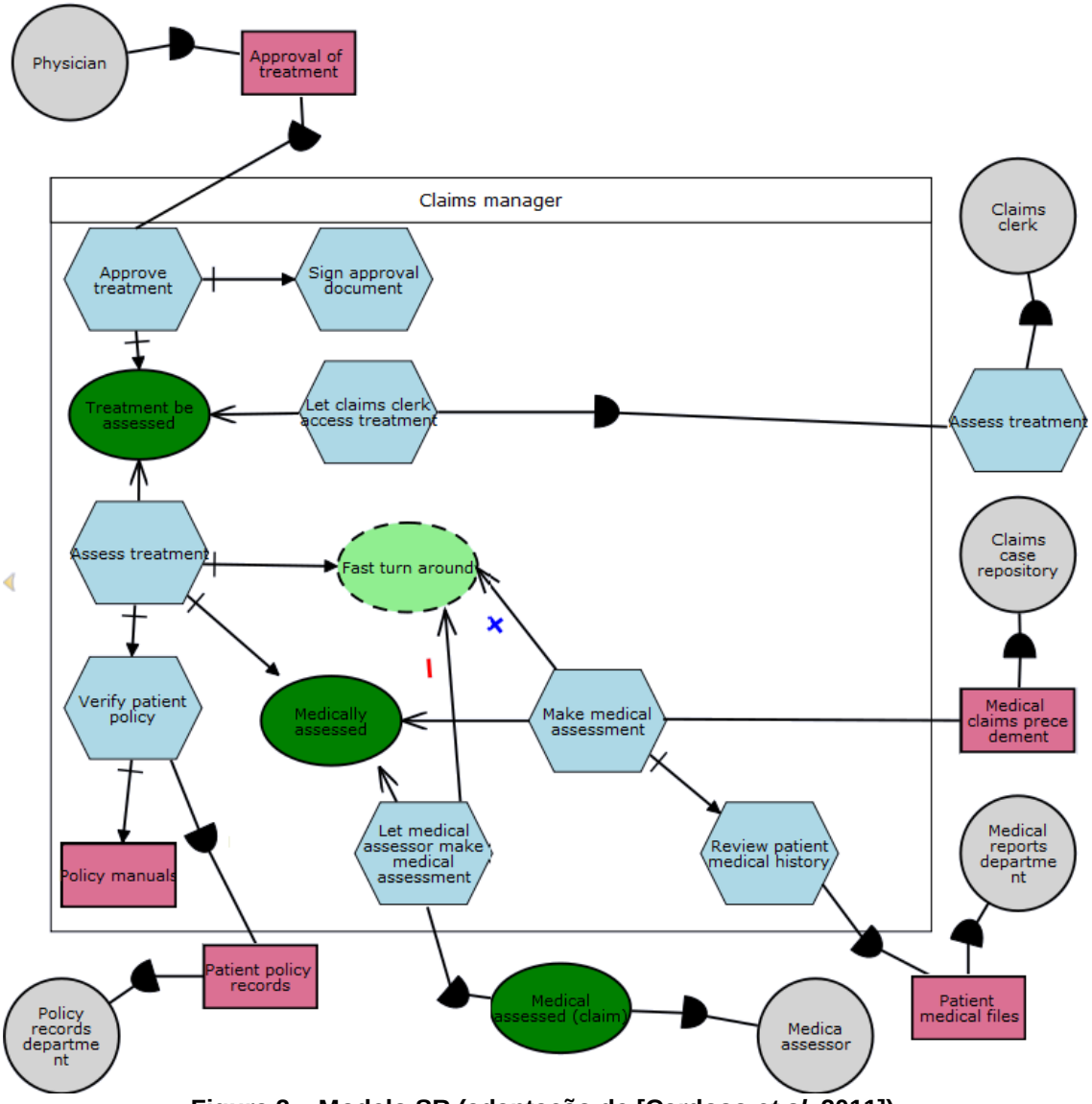


Figura 2 – Modelo SR (adaptação de [Cardoso *et al*, 2011]).

1.2.2. BPMN

Esta seção resume os aspectos referentes à notação BPMN. Todo o conteúdo desta seção é baseado no documento oficial da OMG [OMG, 2010], contendo resumos e traduções livres do texto original. É importante salientar que no momento do desenvolvimento deste documento a notação BPMN 2.0 ainda encontra-se na versão BETA2, assim como o seu documento oficial.

BPMN (*Business Process Model and Notation*) é um padrão desenvolvido pela OMG (*Object Management Group*) com o principal objetivo de oferecer uma notação que fosse legível e compreensível por todos os usuários do negócio, desde os analistas de negócio, que criam os rascunhos iniciais do processo, aos desenvolvedores técnicos, responsáveis por implementar a tecnologia que irá suportar o processo e, por fim, as pessoas do negócio que irão gerenciar e monitorar estes processos. Desta forma, BPMN cria uma ponte padronizada para a lacuna entre o desenho do processo de negócio e sua implementação.

A intenção da BPMN é padronizar o modelo de processo de negócio e sua notação em face das diversas notações de modelagem e pontos de vista distintos. Fazendo isto, a notação irá oferecer uma forma simples de comunicação das informações do processo a outros usuários do negócio, implementadores do processo, consumidores e fornecedores.

Os membros da OMG trouxeram conhecimentos e experiências em diversas notações e procuraram consolidar as melhores ideias a partir delas para alcançar uma única notação padrão. Exemplos de outras notações ou metodologias que foram revistas são *UML Activity Diagram, UML EDOC Business Process, IDEF, ebCML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM* e *Event-Process Chains (EPCs)*.

Escopo da notação BPMN

Esta especificação oferece modelo e notação para processos de negócio em um formato padronizado que pode ser usado entre diferentes ferramentas para troca de definições de arquivos de processos BPMN. O objetivo da especificação é habilitar a portabilidade de definições de processos, permitindo que os usuários possam pegar as definições de um processo criado em um ambiente de um fabricante e utilizá-lo em um ambiente de outro fabricante. A especificação BPMN 2.0 estende o escopo e a capacidade da versão BPMN 1.2 em algumas áreas:

- Formaliza a semântica de execução para todos os elementos BPMN
- Define um mecanismo extensível, tanto para extensões de modelos de processo como para extensões de gráficos
- Refina a composição de eventos e correlações
- Estende a definição de interações humanas
- Define um modelo de coreografia

Esta especificação também resolve problemas conhecidos da BPMN 1.2, tais como inconsistências e ambiguidades. BPMN suporta somente conceitos de modelagem que são aplicáveis a processos de negócio. Isso significa que outros tipos de modelagem realizados por organizações, mesmo com propósitos do negócio, estão fora do escopo da BPMN. Os seguintes aspectos estão fora do escopo da especificação:

- Definição de modelos organizacionais e recursos.
- Modelagem de repartições funcionais
- Modelagem de dados e informação
- Modelagem de estratégia
- Modelagem de regras de negócio

Apesar da BPMN mostrar o fluxo de dados (mensagens) e a associação de artefatos de dados a atividades, ele não é uma linguagem para modelagem de fluxo de dados. Além disso, simulação operacional, monitoramento e o posicionamento estratégico de processos de negócio também estão fora de escopo da especificação 2.0.

BPMN 2.0 pode ser mapeado para mais de uma plataforma dependente de linguagem de modelagem de processos, por exemplo, WS-BPEL 2.0. O mapeamento para outros padrões emergentes são considerados como esforços isolados. A especificação 2.0 utiliza outros padrões para definir tipos de dados e operações de serviços, tais como XML *Schema*, *XPath* e WSDL.

Elementos da BPMN

Deve ser enfatizado que uma das direções para o desenvolvimento do BPMN é criar um mecanismo simples e compreensível para desenvolver modelos de processos de negócio e, ao mesmo tempo, ser capaz de interligar com a complexidade inerente aos processos de negócio. A abordagem adotada para lidar com esses dois requisitos conflitantes foi organizar aspectos gráficos da notação em categorias especificas. Isso fornece um pequeno conjunto de categorias de notação que o leitor de um diagrama BPMN pode facilmente reconhecer os tipos básicos de elementos e compreender o diagrama. Dentro das categorias básicas dos elementos, variação e informações adicionais podem ser adicionadas para suportar os requisitos de complexidade sem mudar dramaticamente o visual básico e percepção do diagrama. As cinco categorias básicas de elementos são: Fluxo de objetos; Dados; Objetos de conexão; Raias (*Swimlanes)* e Artefatos. Cada categoria é descrita a seguir:

Fluxo de Objetos – São os elementos gráficos principais que definem o comportamento do processo de negócio. Existem 3 tipos de Fluxos de Objetos:

- Eventos
- Atividades
- Operadores lógicos

Dados – É representado por quarto elementos:

- Objetos de dados
- Entrada de dados
- Saída de dados
- Depósitos de dados

Objetos de conexão – São objetos que conectam semanticamente elementos do modelo. Existem quatro formas de conectar os Objetos de Fluxo entre si e com outras informações. Existem quatro Objetos de Conexão:

- Fluxos de sequência
- Fluxos de mensagens
- Associações
- Associações de dados

Raias – São objetos que organizam os elementos de um modelo, agrupando-os. Existem duas formas de agrupar os elementos de modelagem primários através das "Raias":

- Grupos (*Pools)*
- Faixas (*Lanes)*

Artefatos - São usados para oferecer informação adicional sobre o processo. Existem dois artefatos padronizados, mas os modeladores ou as ferramentas de modelagem estão livres para adicionar quantos Artefatos sejam necessários. O atual conjunto de Artefatos inclui:

- Grupo
- Anotação de texto

A Figura 3 apresenta um exemplo de modelo de processo de negócio. O modelo apresenta o fluxo de atividades do Engenheiro de software e do Gerente de configuração referente ao subprocesso "Gestão de Mudança", que compõem o processo superior "Desenvolvimento de Software".

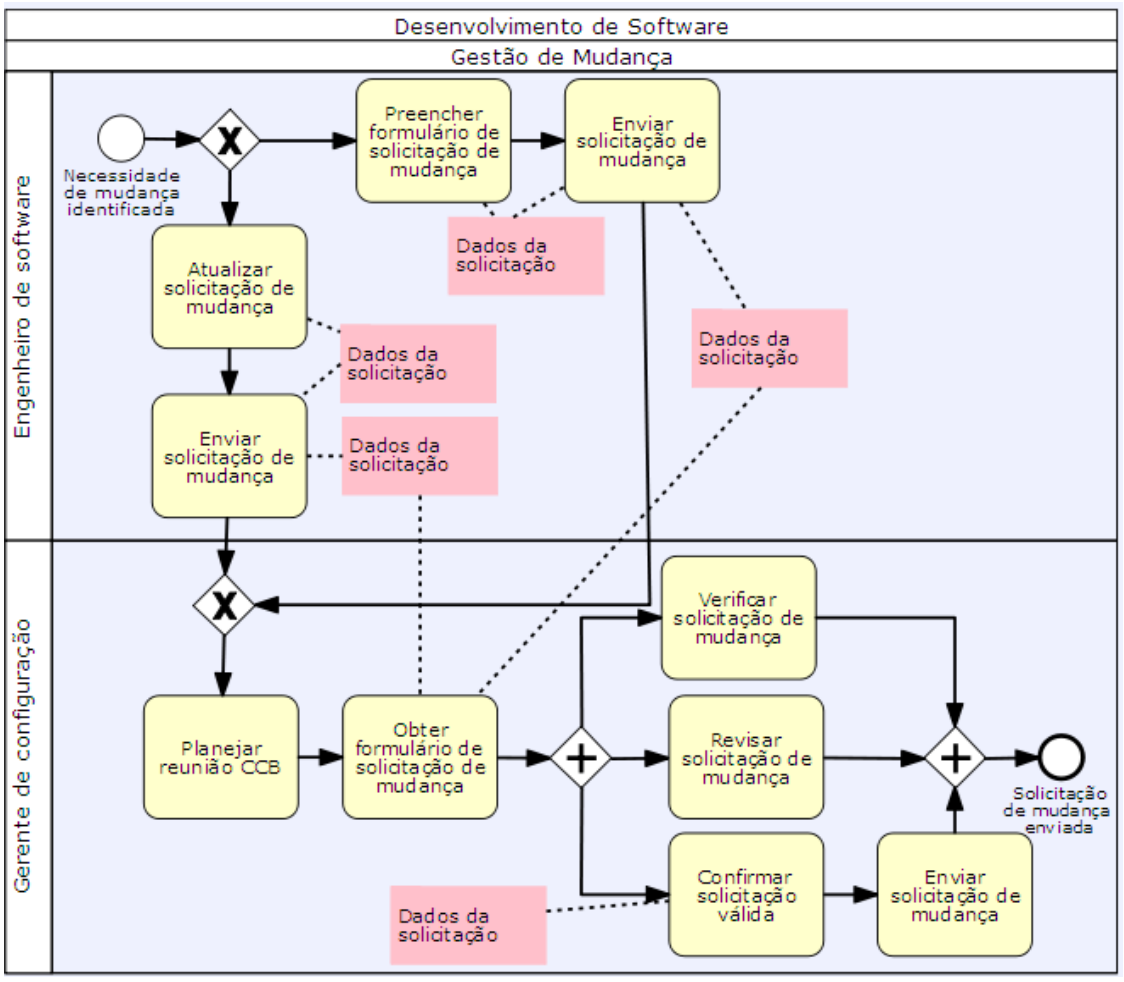


Figura 3 – Exemplo de processo de negócio na notação BPMN

1.2.3. Ferramenta *Oryx*

A ferramenta *Oryx* é um *framework* acadêmico de código aberto inicialmente desenvolvido para oferecer a modelagem gráfica de processos de negócio. Sua tecnologia é baseada em *web*, sendo executado através do navegador (*browser*), o que elimina a necessidade de instalação do software. A ferramenta oferece várias linguagens para modelagem, tais como BPMN (*Business Process Model and Notation*) e EPC (*Event-driven Process Chain*), além disso, é extensível a novas funções através da adição de *plugins* [*Oryx*, 2010].

Sua arquitetura é bem definida e já foi reutilizada para o desenvolvimento de soluções comerciais como, por exemplo, a ferramenta Signavio [Kunze&Weske, 2010]. Além disso, a ferramenta *Oryx* encontra-se em uso por grandes empresas, por exemplo, a Serpro [Serpro, 2011], que realiza customizações para adequá-la às suas necessidades.

A Figura 4 apresenta alguns diagramas disponíveis para uso através da ferramenta *Oryx*. Atualmente a ferramenta encontra-se na versão Beta e pode ser acessada pelo site oficial do *Oryx*: "*http://bpt.hpi.uni-potsdam.de/Oryx/Research*". Mais informações descobertas sobre a ferramenta *Oryx* durante este trabalho são descritas nos capítulos subsequentes.

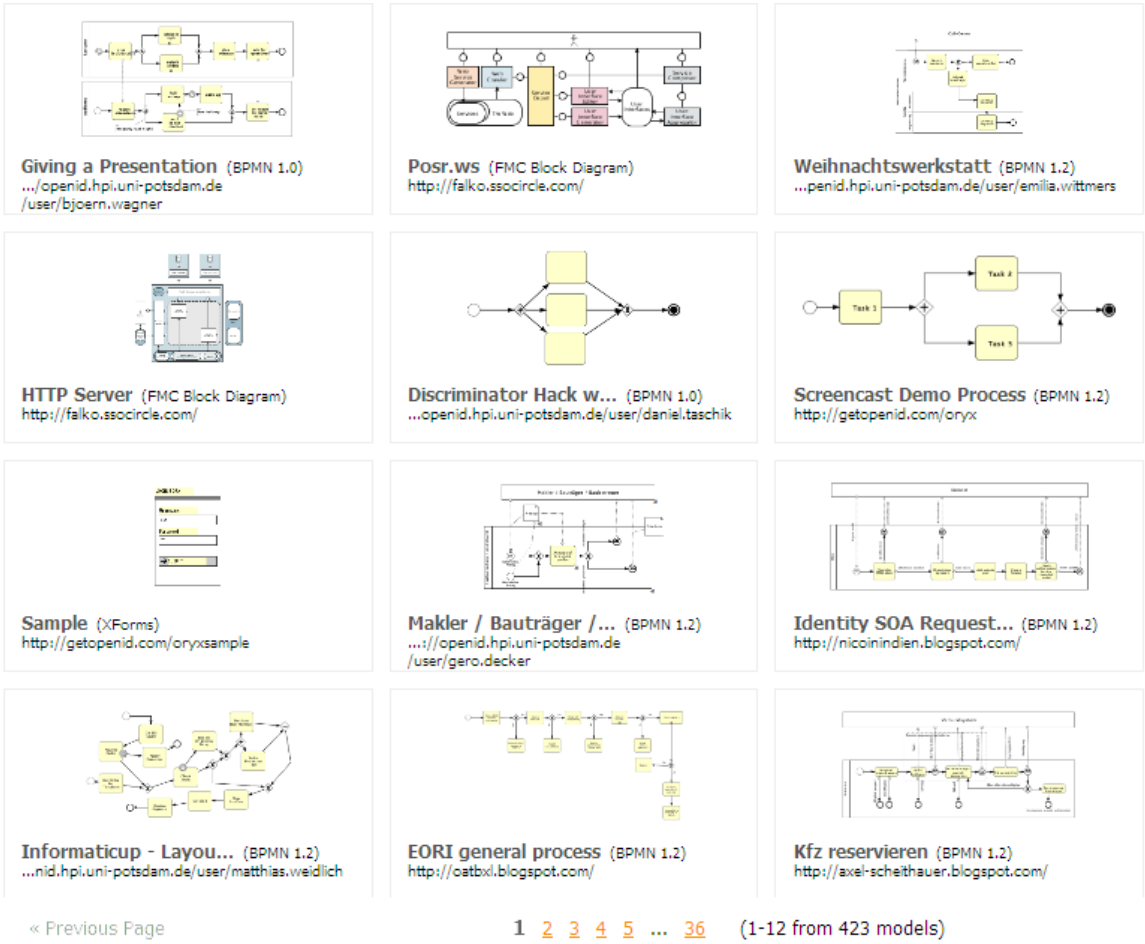


Figura 4 – Painel com diversos exemplos de tipos de modelos que podem ser modelados na ferramenta [*Oryx*, 2010]

1.3. Considerações iniciais do projeto

Esta seção apresenta os objetivos que devem ser satisfeitos neste trabalho e o conhecimento sobre a ferramenta no estágio inicial, antes de começarem os estudos efetivos.

1.3.1. Objetivos

O objetivo deste trabalho é criar uma ferramenta que permita a modelagem de objetivos juntamente com a modelagem de processos. Para alcançar este objetivo a ferramenta *Oryx* é reutilizada e alterada de forma a oferecer um modelo que contenha os elementos e regras de modelagem da linguagem i* e da notação BPMN.

1.3.2. Situação inicial

O conhecimento inicial deste trabalho em relação aos objetivos que se almeja alcançar é suficiente para nortear o início do projeto. Sabe-se que a ferramenta *Oryx* tem capacidade de ser customizada em função da experiência adquirida no uso da ferramenta. Diversos modelos com elementos e regras bem distintos são oferecidos na ferramenta através de seu site oficial. Além disso, usamos o conhecimento de outros [Daniel *et al* 2007] que já trabalharam com a customização da ferramenta.

Outras fontes de conhecimento identificadas foram: fórum dedicado a discussão da ferramenta *Oryx*, publicação de documentos, escrito em inglês e alemão.

1.3.3. Lista de Requisitos iniciais

A lista de requisitos desenvolvida nesta seção apresenta a visão macro e inicial dos requisitos antes de iniciar o trabalho de pesquisa e desenvolvimento. Durante as pesquisas, é esperado que os requisitos aumentem e/ou sejam mais granulares, motivado tanto pelo conhecimento que será adquirido durante o processo como pela necessidade de alinhamento a novas necessidades e natural evolução.

- Criar *plugin* contendo elementos do i* e da BPMN

1.4. Estrutura do trabalho

Este trabalho está dividido da seguinte forma:

O capítulo 0 apresenta uma introdução e está dividido em quatro seções, a seção 1.1 apresenta a motivação do projeto, explicando brevemente sobre a ideia de desenvolver *plugins* para a ferramenta *Oryx*, a seção 1.2 detalha a linguagem i*, a notação BPMN e a ferramenta *Oryx*, que são os elementos envolvidos no trabalho, a seção 1.3 apresenta uma breve apreciação de registros sobre o estado inicial do projeto, momento em que as informações são limitadas, a seção 1.4 é a presente seção, resumindo a estrutura do documento.

O capítulo 2 descreve detalhadamente a aplicação de técnicas da engenharia de software, dividido em quatro seções. A seção 2.1 apresenta a introdução do capítulo, a seção 2.2 aborda a reengenharia de software e descreve a aplicação da engenharia reversa, a reprojeção dos requisitos e a reimplementação do código.

O capítulo 3 é dedicado à execução dos testes, baseados nos requisitos implementados. Os testes seguem um roteiro predefinido que contrapõem os resultados esperados com os resultados práticos. Este capítulo é dividido em três seções: a seção 3.1 apresenta o roteiro de teste de forma genérica, a seção 3.2 descreve o resultado dos testes para o modelo principal, diagrama iStar SD, diagrama iStar SR e o teste de chamada das diferentes perspectivas. A seção 3.3 apresenta um conjunto de *bugs* da ferramenta *Oryx,* de forma a alertar ao usuário que estes problemas não possuem ligação com os *plugins* desenvolvidos neste trabalho.

O capítulo 4 apresenta a conclusão do trabalho.

O Apêndice apresenta um conjunto de informações necessárias como primeiros passos para utilizar o *Oryx*. Ele encontra-se dividido da seguinte forma: instalação da ferramenta, uso de perspectivas e os aspectos e opções da ferramenta utilizados na modelagem.

2. Aplicando a Engenharia de Software

Este capítulo detalha a aplicação de técnicas da engenharia de software que foram utilizadas para possibilitar o desenvolvimento de um novo *plugin* para a ferramenta *Oryx*, partindo do conhecimento restrito sobre sua infraestrutura.

2.1. Introdução

Após definir os objetivos e necessidades básicos do projeto, iniciaram-se as atividades de planejamento. Baseado no conteúdo limitado sobre a ferramenta torna-se imprescindível o aprofundamento no conhecimento de sua arquitetura e linguagem de codificação.

Considerando os recursos iniciais que se limitam à própria ferramenta e algumas fontes de informação limitadas, buscamos dentro das técnicas existentes na engenharia de software uma solução que norteasse as próximas atividades, considerando o perfil de problema abordado. O principal insumo inicial é o próprio software desenvolvido que possui seu código livre e, em busca de informações da fase de projeção e codificação do sistema, caminharemos na direção inversa do fluxo de desenvolvimento do software.

Partindo das características deste projeto, decidimos aplicar a reengenharia de software. As próximas subseções detalham o método e sua aplicação neste projeto.

2.2. Reengenharia de software

A reengenharia de software, conhecida também como renovação ou reconstrução, é o exame e alteração de um sistema de software, para reconstituí-lo em uma nova forma, e a subsequente implementação dessa nova forma. Um processo de reengenharia geralmente inclui alguma forma de engenharia reversa, seguida por uma forma de engenharia progressiva ou reestruturação [Chikofsky&Cross II, 1990&IEEE CS-TCSE, 1997&GT-REG, 1998 *apud* Piekarski&Quinaia, 1995].

Do mesmo modo que [Chikofsky&Cross II, 1990], [Warden, 1992] consideram que a reengenharia pode ser dividida em duas fases principais: a Engenharia Reversa e a Engenharia Progressiva, cada uma destas fases também podem ser divididas em uma série de atividades [Piekarski&Quinaia, 1995]. Desta forma, este trabalho foi realizado

aplicando a engenharia reversa para obtermos informações sobre a ferramenta, possibilitando a posterior reimplementação do software, com o objetivo de realizar todas as alterações necessárias para que seja possível oferecer a modelagem utilizando as linguagens BPMN e i* de forma conjunta.

Para nortear este trabalho de forma sistemática, decidimos instanciar o método de reengenharia de software proposto por [Leite, 2006]. O método é baseado em um metaprocesso que considera a existência de quatro subprocessos necessários para a execução da reengenharia de um dado artefato, que são: recuperar, especificar, reprojetar e reimplementar. A Figura 5 detalha em um modelo SADT cada um dos subprocessos e respectivos elementos envolvidos (entrada, saída, controle e mecanismo).

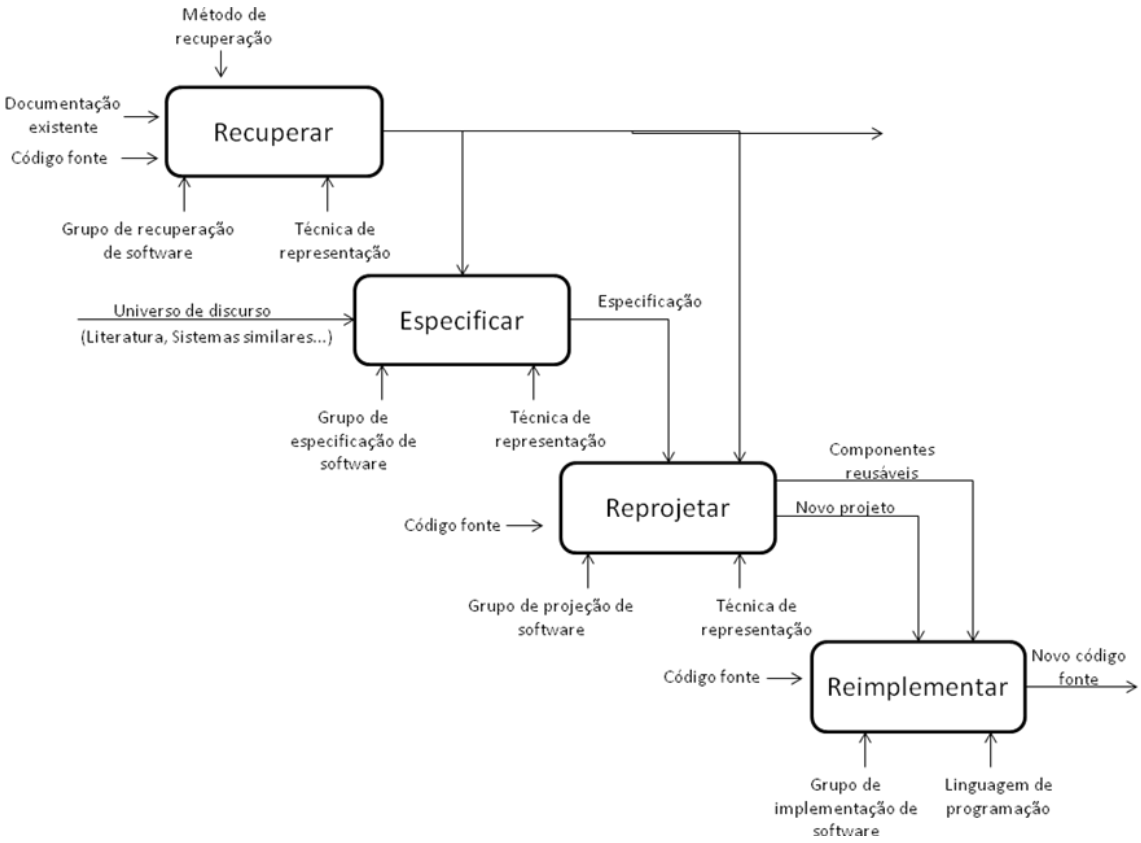


Figura 5 – Método de reengenharia de software (baseado em [Leite, 1996]).

Nosso primeiro passo consistiu em caminhar "para trás", partindo do software implementado em direção à sua especificação inicial. Para isso é necessário buscar informações sobre a ferramenta com o objetivo de identificar como é o funcionamento de sua infraestrutura, o que permite posteriormente gerar modelos de especificação do sistema. A partir disso, é possível reprojetar os requisitos, adicionando as novas necessidades (que motivaram a reengenharia da ferramenta) e reimplementar o software com o máximo de reuso, ou seja, alterando o código fonte o mínimo possível para corresponder aos novos requisitos.

A primeira atividade a ser realizada neste trabalho é representada pelo subprocesso "Recuperar" (Figura 5). Neste subprocesso utilizamos como "método de recuperação" a Engenharia Reversa, que é uma forma de estudo de um software que visa derivar informações sobre o projeto ou especificação de um sistema, partindo de seu código fonte [Sommerville, 1995].

Os detalhes deste trabalho sobre a execução das atividades que foram realizadas em cada um dos subprocessos do método são descritos nas próximas seções. A próxima seção detalha a aplicação da engenharia reversa no software Oryx.

2.2.1. Aplicação da Engenharia reversa

A engenharia reversa tem como principio a "desmontagem" das caixas pretas do software, de seus segredos, de trás pra frente, ou seja, o processo de recuperação do projeto com especificação e documentação procedimental, arquitetura e dados [Pressman, 1995 *apud* Rezende, 2005].

A engenharia reversa também pode ser elaborada na criação de novos sistemas a partir de sistemas antigos [Heuser, 2000 *apud* Rezende, 2005], utilizando as informações recuperadas para alterar ou reconstituir o sistema existente, num esforço para melhorar sua qualidade global, reimplementando a função do sistema, adicionando novas funções ou melhora de desempenho global [Rezende, 2005].

Para aplicarmos a engenharia reversa na ferramenta *Oryx,* definimos um plano composto por 3 fases que, por sua vez, são compostas por um conjunto de atividades, conforme descrito na Tabela 1. Essas fases decompõem o subprocesso "Recuperar", do método (Figura 5).

Tabela 1 – Projeção de macroatividades da engenharia reversa da ferramenta *Oryx*

Fase 1	Fase 2	Fase 3
Identificar como obter o código fonte	Identificar documentação da ferramenta	Estudar o código fonte
Identificar os procedimentos para a instalação da base de codificação	Estudar a arquitetura da ferramenta	Identificar os conceitos arquiteturais na codificação
Configurar o ambiente de desenvolvimento	Identificar os procedimentos para desenvolvimento de *plugins*	Avaliar outras necessidades

Todas as atividades planejadas para a aplicação da engenharia reversa na ferramenta *Oryx* se incluem na atividade mais abstrata "Adquirir conhecimento sobre customização da ferramenta". O código fonte e a documentação existente servem de insumo para a engenharia reversa, de acordo com o método aplicado (Tabela 1). A execução das 3 fases são descritas nas subseções seguintes.

2.2.1.1. Execução da primeira fase

A primeira fase do processo de engenharia reversa foi iniciada com uma pesquisa online para obter informações gerais sobre como proceder com a instalação do ambiente de desenvolvimento *Oryx*. Durante as buscas foram identificados muitos sites com o assunto "*Oryx*", porém o site oficial do projeto *Oryx* (http://code.google.com/p/*Oryx*-editor/) publicado no "*Google Codes*" ofereceu o conjunto de informações necessárias para realizar as atividades da primeira fase. Na sequencia são resumidos todos os passo necessários para criar o ambiente de desenvolvimento da ferramenta *Oryx*:

Passo 1 – Instalação dos softwares necessários:

- Mozilla Firefox – Disponível em http://www.mozilla.com/pt-PT/firefox/;
- *Java Development Kit* (JDK) – Disponível em "http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html";
- Eclipse IDE for Java EE *Developers* – Disponível em "http://www.eclipse.org/downloads/";
- Python 2.5 ou maior – "Disponível em http://www.python.org/download/";
- Postgres 8.3 ou maior, desde que compatível – Disponível em http://www.postgresql.org/download/; No Windows, execute a instalação como administrador e configure a instalação do Postgres para instalar também PL/Python, se houver esta opção.
- Tomcat – Disponível em "http://tomcat.apache.org/download-60.cgi";

Após a instalação dos softwares, é necessário realizar um conjunto de configurações que serão descritos a seguir.

- O primeiro passo consiste em instalar os softwares necessários que serão utilizados em conjunto com o ambiente de programação. Os softwares são:
  - Mozilla Firefox – Instalar *addon* 1843, disponível em "https://addons.mozilla.org/de/firefox/addon/1843";
  - Criar variável de ambiente para o Java. No Windows, o seguinte artigo mostra como proceder: "http://technet.microsoft.com/pt-br/library/cc668471.aspx"; A variável de ambiente deve ter a seguinte configuração: Nome da variável: **JAVA_HOME**; Caminho da variável (no Windows): "**C:\Arquivos de Programas\Java\jdk1.6.0_11"**, ou o caminho referente a versão do JDK instalado.
  - No Eclipse, instalar os seguintes *plugins* a partir do menu "*Help/Install new softwares"* ou semelhante, no caso de diferentes versões*:* Subclipse – disponível em "http://subclipse.tigris.org/update_1.4.x"; Aptana Studio – disponível em *Aptana Studio Update Site* – "http://download.aptana.com/tools/studio/*plugin*/install/studio";
- O segundo passo é baixar através de SVN a codificação do *Oryx*. No Eclipse, ir em "**File/New/Other../SVN/Checkout Projects from SVN**" e clicar em "*Next"*. Configurar o seguinte repositório: "**http://*Oryx*-editor.googlecode.com/svn/trunk"** e finalizar o procedimento. A ferramenta irá baixar todos os arquivos da pasta "*trunk*".
- O terceiro passo é configurar o banco de dados e outros arquivos do sistema. A configuração do banco de dados deve ser feita da seguinte forma: no *prompt* de comando (CMD, no Windows), realizar os seguintes comandos:
  - createuser –U postgres –-echo –-pwprompt –-encrypted TTP
  - TTP d –U postgres –-echo –-encoding utf8 –-owner poem poem
  - psql –U postgres –-dbname TTP –-file T_schema.sql

  Editar o arquivo **tomcat-users.xml**, presente dentro da pasta **conf**, no diretório onde o programa foi instalado. O conteúdo do arquivo deve ser o seguinte:

  <?xml version='1.0' encoding='utf-8'?>

<*tomcat-users*>

<*user username="Oryx" password="Oryx" roles="admin, manager" />*

</*tomcat-users*>

Nos arquivos do *Oryx* recentemente adquiridos via SVN, editar o arquivo **build.properties.** A linha **deploymentdir** deve conter o caminho correto da pasta de publicação de projetos **webapps** do tomcat. Ex: "C:/Program Files/Apache Software Foundation/Tomcat 6.0/webapps".

Ainda nos arquivos do *Oryx*, vá em "editor/client/scripts/*Oryx*.js" e configure a linha **PATH** como "`PATH: "/Oryx/",`". Em " TTP-jvm/etc/hibernate.cfg.xml", configure as configurações de acesso ao banco como:

```
<!-Database connection settings →
<property name="connection.driver_class">org.postgresql.Driver</property>
<property name="connection.url">jdbc:postgresql://localhost/poem</property>
<property name="connection.username">poem</property>
<property name="connection.password">Oryx</property>
```

No arquivo **build.xml** acesse o "*External Tools Configuration*", selecionar os seguintes *Targets*: **build-with-xhtml-test-files-flag, build-all, dep loy-all**.
Executar o *build* para compilação. Caso ocorra algum erro, rever os passos deste manual. O procedimento de compilação pode demorar. Após execução com sucesso dos procedimentos descritos, o ambiente está pronto.

Após as execuções destas atividades, o ambiente de desenvolvimento da aplicação foi implantado e a codificação encontra-se acessível. Com isso, a fase 1 (Tabela 1) da engenharia reversa do software *Oryx* foi finalizada. A próxima subseção descreve a execução da segunda fase.

2.2.1.2. Execução da segunda fase

A segunda fase da engenharia reversa foi iniciada com a busca de documentos que descrevessem a arquitetura da ferramenta *Oryx*. Alguns documentos [Daniel *et al*, 2007], [Peters *et al*, 2007], [Tscheschner *et al*, 2007], foram encontrados no site oficial *Oryx*-editor, no *Google Codes* ("http://code.google.com/p/*Oryx*-editor/"). Estes documentos são monografias desenvolvidas por alunos que estão envolvidos no grupo de pesquisa da ferramenta *Oryx*. Os trabalhos possuem um conteúdo extenso sobre a infraestrutura da ferramenta e o desenvolvimento de *plugins*, no entanto, todas as documentações estão em língua estrangeira como o inglês e o alemão.

Para solucionar parcialmente a dificuldade de leitura em alemão, os arquivos em PDF deveriam ser traduzidos. Então foi feita uma pesquisa para identificar ferramentas que realizassem esse trabalho. A maioria das ferramentas encontradas não traduziam arquivos com extensão PDF, e as que traduziam eram caras, além disso, as traduções em softwares gratuitos e demonstrativos eram limitadas a poucas linhas.

Outra busca foi realizada para verificar softwares de tradução de arquivos do tipo DOC, e diversos aplicativos foram encontrados. A partir disso foi iniciada outra busca por programas capazes de traduzir arquivos PDF para arquivos DOC. Vários aplicativos foram encontrados e a conversão foi feita, no entanto, a conversão foi insatisfatória, já que inseriu no documento resultante caracteres de forma aleatória, alterando a formatação e afetando o texto.

Novamente iniciou-se a busca por tradutores de PDF, e desta vez foi identificada uma ferramenta *online* de tradução baseada no tradutor do *Google* (*Google Translate* –

"http://translate.google.com.br/"), disponível em "http://www.onlinedoctranslator.com/translator.html". Este ferramenta traduziu o documento integralmente, mas também inseriu muitos erros, no entanto, foi a melhor solução identificada e o produto da tradução era aceitável.

Com a leitura destes documentos foi possível obter o conjunto de informações necessárias sobre a arquitetura da ferramenta, compreender no nível teórico como é a organização de arquivos do *Oryx* e os arquivos chaves para a construção de *plugins*.

Em sequencia serão descritos, de forma resumida, os principais conceitos da ferramenta *Oryx* descobertos a partir do estudo dos documentos identificados na pesquisa.

A ferramenta tem um projeto típico "Cliente-Servidor" onde mantém a codificação principal sendo processada no lado do servidor, enquanto parte da codificação permanece no lado cliente. A codificação desenvolvida para o lado servidor não foi estudada neste trabalho, já que este conhecimento não é necessário para o desenvolvimento de *plugins*. A estrutura da ferramenta foi desenvolvida para ser capaz de interpretar arquivos que são compostos pela definição de elementos de modelagem e suas regras de interação. O desenvolvimento destes arquivos independe da complexidade do núcleo da ferramenta *Oryx*, o que demonstra a modularidade dos componentes que compõem o "núcleo de sistema" e os "*plugins*".

O foco do trabalho será na parte do cliente, mais especificamente no desenvolvimento dos "*Stencils*". Um *stencil* é a descrição de um objeto gráfico e suas regras, que definem como esse objeto irá se relacionar com os outros no diagrama. Um conjunto de *stencils* podem ser carregados para uso no editor *Oryx* para construir modelos [Peters *et al*, 2007]. No entanto os *stencil*s não serão carregados em separado, eles serão agrupados em um mesmo arquivo formando um *stencil set*.

Uma importante característica do *stencil* é que ele não define a representação gráfica dos elementos, mas somente as propriedades referentes ao comportamento do objeto no diagrama. O desenvolvimento da parte gráfica dos componentes será abordado mais a frente.

Os *stencils set* são armazenados em arquivos do tipo JSON (*JavaScript Object Notation* – "http://json.org/"). O arquivo JSON que define um *plugin* para o *Oryx* é composto por um cabeçalho, conjunto de *stencil*s e conjunto de regras.

A Figura 6 exemplifica o código de um *stencil set*. Esse fragmento é composto por um título que nomeia o *stencil set*, um *namespace* que é um identificador único, uma descrição, e o conjunto de *stencils* e regras.

```
{
  "title":"Workflow Nets",
  "namespace":"http://www.example.org/workflownets",
  "description":"Simple stencil set for Workflow Nets."
  "stencils": [/*...*/],
  "rules": {/*...*/}
}
```

Figura 6 – Fragmento de um *stencil set*

No interior da *tag* de *stencils* são inseridas as definições dos elementos de modelagem. Não há restrição de numero de *stencils* a serem inseridos no conjunto. A Figura 7 exemplifica o código de um *stencil*, presente dentro da *tag "Stencils": [ ]*, ele inicia e termina entre as chaves ({ }).

O *stencil* é formado pelos atributos: *type*, que define qual é o tipo de elemento do *stencil,* por exemplo, um nó ou relacionamento; *id*, que é um identificador único para o elemento; *title*, nomeia o objeto no diagrama; *description*, define uma descrição do elemento que será mostrada no gráfico; *view*, é o *path* para o arquivo gráfico que será desenhado no diagrama; *icon*, é o *path* para o arquivo gráfico que possuirá o pequeno ícone que representará o elemento; *roles*, especificam regras definidas que podem ser aplicadas a elementos específicos; *properties*, oferecem ao usuário informações e campos para serem preenchidos referentes a propriedades do elemento. Não existe restrição ao numero de propriedades inseridas no *stencil*.

```
"stencils":
[
  {
    "type":"node",
    "id":"Activity",
    "title":"Activity",
    "description":"An atomic activity.",
    "view":"activity.svg",
    "icon":"activity.png",
    "roles": ["activitySource", "activityTarget"],
    "properties": [/*...*/]
  }/*,
  ...*/
]/*,
```

Figura 7 – Fragmento de um *stencil*

As propriedades de um *stencil* podem conter diversos parâmetros. Cada propriedade descrita irá gerar um campo no gráfico seja para permitir a inserção de texto, seleção de opção ou conteúdo pré-definido. A Figura 8 exemplifica o código das *properties* contendo apenas uma propriedade e seus parâmetros, limitados pelos "{ }".

```
"properties": [
  {
    "id":"caption",
    "type":"String",
    "title":"Caption",
    "value":"",
    "description":"",
    "readonly":false,
    "optional":true,
    "refToView":"caption",
    "wrapLines":true
  }/*,
  ...*/ ]
```

Figura 8 – Fragmento das propriedades de um *stencil*

Por fim, as *rules* definem um conjunto de regras referente aos relacionamentos entre os elementos no diagrama. A Figura 9 mostra um fragmento de código contendo três tipos de regras: *connectionRules, cardinalityRules* e *containmentRules*. Ainda existem outros tipos de regras, porém somente essas serão detalhadas devido à sua predominância de aplicação nos *stencils set* produzidos neste trabalho.

```
"rules":
{
  "connectionRules":  [/*...*/],
  "cardinalityRules": [/*...*/],
  "containmentRules": [/*...*/]
}
```

Figura 9 – Fragmento de código das regras de um *stencil set*

O tipo de regra *"connectionRules"* consiste em definir quais elementos podem ser interligados a partir de um relacionamento, ou seja, mais especificamente é a configuração de um determinado objeto do tipo relacionamento, em que define-se os elementos os quais ele pode interligar.

A Figura 10 apresenta um fragmento de código que define regras para relacionamento entre elementos. Neste exemplo, o elemento de relacionamento *"controlflow"* está configurado com duas regras de conexão. A primeira define que pode existir uma conexão saindo do elemento *"activitySource"* em direção ao elemento *"conditionTarget"*. A segunda regra define a conexão saindo do elemento *"conditionSource"* para o elemento *"activityTarget"*.

```
"connectionRules": [
  {
    "role":"controlflow",
    "connects": [
      {
        "from":"activitySource",
        "to":"conditionTarget"
      },
      {
        "from":"conditionSource",
        "to":"activityTarget"
      }
    ]
  }
]
```

Figura 10 – Fragmento de código das regras de conexão

O tipo de regra *"cardinalityRules"* define o número de conexões de entrada/saída que um elemento pode possuir. A Figura 11 exemplifica um fragmento contendo duas regras de cardinalidade. Essa regra não está definida para um elemento específico como descrito nas regras de conexão, exemplificada anteriormente, mas define um papel (*role*) que deve ser configurado no *stencil* para adquirir a regra. No exemplo, duas *roles* são definidas: a *"inputcondition"* e a *"outputcondition"*. A *role "inputcondition"* define que o objeto programado para obedecer esta regra deverá obrigatoriamente ter um elemento de relacionamento "entrando". A *"role outputcondition"* também especifica que é obrigatório um relacionamento de saída do elemento.

```
"cardinalityRules": [
  {
    "role":"inputcondition",
    "minimumOccurrence":1,
    "maximumOccurrence":1
  },
  {
    "role":"outputcondition",
    "minimumOccurence":1,
    "maximumOccurrence":1
  },
]
```

Figura 11 – Fragmento de código de regras de cardinalidade

O tipo de regra *"containmentRules"* define quais elementos podem estar contidos dentro do elemento configurado para obedecer esta regra. Esta regra é mais adequada a grupos (*pools*) onde os elementos são inseridos. A Figura 12 exemplifica um fragmento de código contendo uma regra de conteúdo que cria uma *role "diagram"*

configurada para permitir os elementos que possuem o id "*activity*", "*condition*" e as regras de conexão "*inputcondition*" e "*outputcondition*" serem modelados dentro do corpo dos objetos que forem configurados para obedecê-la.

```
"containmentRules": [
  {
   "role":"diagram",
   "contains": [
     "activity",
     "condition",
     "inputcondition",
     "outputcondition"
   ]
  }
]
```

Figura 12 – Fragmento de código de regras de conteúdo

Uma vez definido o *stencil set*, todas as informações e regras sobre os elementos estarão disponíveis para interpretação do *Oryx*. O próximo passo é definir os elementos gráficos que representarão os *stencils* definidos no *stencil set*. Dois elementos gráficos são necessários para representação no modelo: o ícone e o objeto gráfico.

O ícone é uma versão reduzida do objeto gráfico que serve para a ferramenta apresentar ao usuário para que ele selecione no painel e inclua no diagrama. Os ícones devem estar no formato de arquivo PNG com dimensões de 16 x 16 pixels. Essas figuras podem ser desenvolvidas, por exemplo, utilizando o *MsPaint*, no *Windows*.

O elemento gráfico que representa o objeto que será desenhado no diagrama é definido a partir do formato SVG (*Scalable Vetor Graphics*).

Neste formato os desenhos são criados a partir de códigos do padrão SVG. Existem vários comandos que podem ser utilizados para gerar diversos tipos de grafos. Esses comandos podem ser estudados no site do W3C ("http://www.w3.org/Graphics/SVG/"). Existem vários tipos de arquivo SVG, por exemplo, é possível criar um arquivo SVG a partir de outras figuras existentes, criando apenas uma ancora para o outro arquivo gráfico, no entanto, este tipo de arquivo SVG não funciona corretamente no *Oryx*, sendo recomendado apenas o uso dos arquivos SVG descritivos, em que as imagens são geradas a partir da especificação de código na linguagem oferecida pelo padrão.

Além da codificação do padrão SVG, devem ser inseridos no arquivo outras *tags* que serão utilizadas pelo *Oryx*. Por exemplo, devem ser definidos no arquivo SVG a localização do campo de texto do elemento, cor e tamanho da fonte e outros elementos, como os *Oryx Magnets*, que definem pontos no desenho onde os relacionamentos podem se ligar ao objeto.

A Figura 13 demonstra um exemplo de codificação de arquivo SVG contendo *tags* do *Oryx*. Neste exemplo, nota-se que o cabeçalho possui a definição do *namespace* do *Oryx*, necessário para reconhecer dessas *tags*: "*xmlns:Oryx=" TTP://www.b3mn.org/Oryx*".

A *tag* <*Oryx*:magnets> demarca o inicio da configuração dos pontos que estarão disponíveis no objeto para a ligação de relacionamentos. Cada ponto é definido pela sua coordenada x e y.

Entre as *tags* <g> </g> estão configurados os elementos que definirão o visual gráfico do elemento. A *tag* <radialGradient> </radialGradient> define um fundo para ser utilizado no objeto.

Esta imagem é composta por um retângulo <rect> que está ancorado ao elemento (*Oryx*:anchors) pelo topo, esquerda e direita, o que significa que ao ampliar o objeto no diagrama, o retângulo crescerá para estas direções, exceto para baixo. O mesmo ocorre para o círculo <*circle*> que também compõem a imagem. Na *tag* <*text*> é definido o campo para inserção de texto no elemento, incluindo seu tamanho da fonte, coordenada x,y e alinhamento (*align*) em relação ao objeto. O texto é encaixado no retângulo que é desenhado no gráfico (*fittoelem).*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
   xmlns="http://www.w3.org/2000/svg"
   xmlns:oryx="http://www.b3mn.org/oryx"
   width="100"
   height="100"
   version="1.0">

  <oryx:magnets>
    <oryx:magnet oryx:cx="0" oryx:cy="60" oryx:anchors="left" />
    <oryx:magnet oryx:cx="75" oryx:cy="80" oryx:anchors="bottom" />
    <oryx:magnet oryx:cx="100" oryx:cy="60" oryx:anchors="right" />
    <oryx:magnet oryx:cx="75" oryx:cy="0" oryx:anchors="top" />
    <oryx:magnet oryx:cx="50" oryx:cy="40" oryx:default="yes" />
  </oryx:magnets>

  <g pointer-events="fill">

    <defs>
        <radialGradient id="background" cx="50%" cy="50%" r="100%" fx="50%" fy="50%">
            <stop offset="0%" stop-color="lightgray" stop-opacity="1"/>
            <stop id="fill_el" offset="100%" stop-color="lightgray" stop-opacity="1"/>
        </radialGradient>
    </defs>

    <rect id="text_frame" oryx:anchors="bottom top right left" x="25" y="25"
    width="70" height="50" rx="10" ry="10" stroke="black" stroke-width="0" fill="none" />

    <circle id="bg_frame" oryx:resize="vertical horizontal" cx="50" cy="40" r="45"
    stroke="black" fill="url(#background) white" stroke-width="2"/>

    <text font-size="12"
        id="text_name"
        x="50" y="40"
        oryx:align="middle center"
        oryx:fittoelem="text_frame"
        stroke="black">
    </text>
  </g>
</svg>
```

Figura 13 – Exemplo de código de imagem SVG

Em suma, para desenvolver o *plugin* desejado, é necessário definir o *stencil set* contendo todos os elementos que se deseja modelar e suas regras, os ícones e arquivos SVG referentes a cada um dos elementos. Esse conhecimento é suficiente para finalizar a Fase 2 (Tabela 1) do processo de engenharia reversa.

Na próxima subseção, são descritos os procedimentos realizados na terceira fase em que serão estudados os conceitos descobertos na segunda fase diretamente na codificação da ferramenta.

2.2.1.3. Execução da terceira fase

Na terceira fase do processo de engenharia reversa, o alvo de estudo é a codificação e o ambiente de programação do *Oryx*. Grande parte do código já foi estudada nos documentos utilizados na segunda fase, porém neste momento todos os conceitos serão revistos de forma prática nos arquivos do projeto do *Oryx*.

Entre o conjunto de pastas e arquivos do projeto do *Oryx*, estão codificações referentes ao lado servidor e cliente da aplicação. Como dito antes, o estudo realizado tem foco no lado do cliente. Os arquivos dos *stencils*, os quais serão de fato editados, estão presentes no seguinte *path:* "*Oryx*/editor/data/*stencil*sets" (Figura 14).

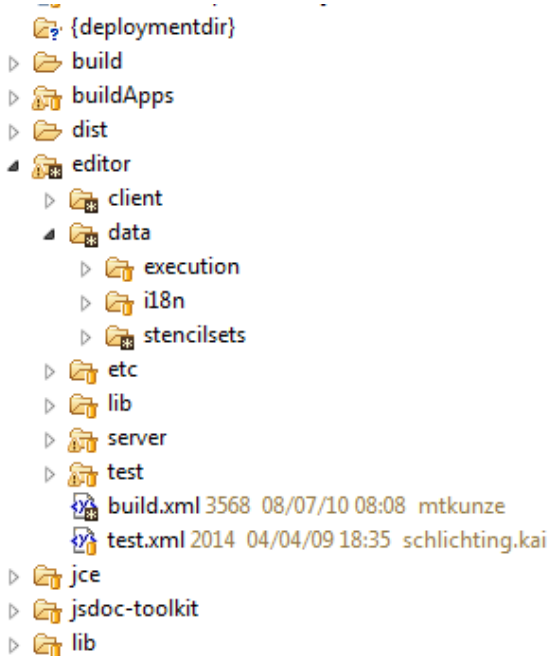


Figura 14 – Infraestrutura de pastas do ambiente de desenvolvimento do *Oryx*

Abrindo a pasta *stencilset* (Figura 15) é possível encontrar um grande conjunto de *stencils* presentes nas subpastas e também o arquivo "*stencil*sets.json". Neste arquivo são registrados todos os *stencils sets* presentes na pasta *stencilsets*. Portanto é necessário registrar o novo *stencil* neste arquivo. A Figura 16 exemplifica o registro de um *stencil set* no arquivo *stencilsets.jason*. As informações que compõem o registro são título do *stencil* (*title)*, *namespace,* descrição (*description)*, *path* onde está o arquivo JSON do *stencilset* que se deseja registrar (*uri)*, *path* do ícone para representar o *stencil set* (*icon_url*) e a opção que torna o *stencil set* visível na ferramenta ou não (*visible).*

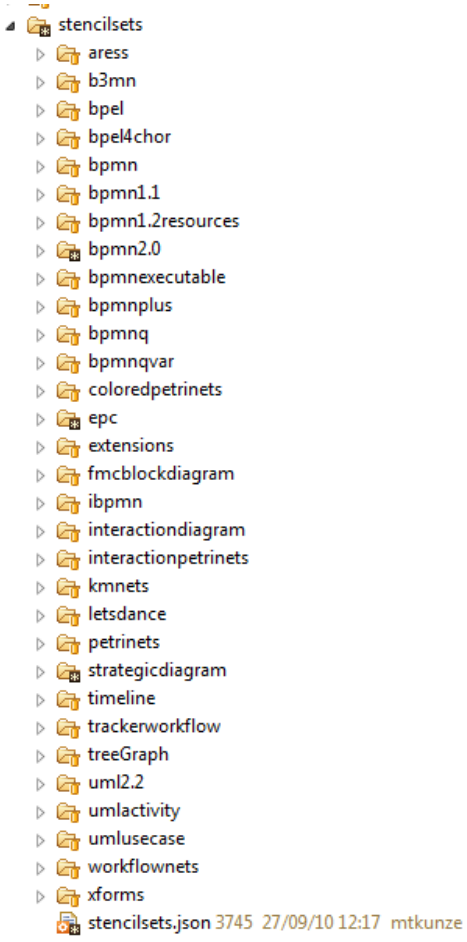


Figura 15 – Detalhamento da pasta *stencilsets*

```
{
    "title":"EPC",
    "namespace":"http://b3mn.org/stencilset/epc#",
    "description":"Event-driven Process Chain specification.",
    "uri":"/epc/epc.json",
    "icon_url":"/epc/epc.png",
    "visible":true
},
```

Figura 16 – Exemplo de registro de *stencil set*

Ao entrar em uma pasta de um *stencil set*, identifica-se uma estrutura comum a todos as pastas dos *stencils set* que é composta por uma subpasta dedicada aos ícones dos elementos (*icons)*, uma subpasta dedicada aos arquivos SVG que representam os elementos no modelo (*view),* o arquivo JSON que contem todas as configurações dos *stencils* e suas regras, e opcionalmente um arquivo do tipo PNG contendo o ícone que representa o *stencil set*.

Figura 17 – Estrutura de pastas do *stencil set* "*petrinets*"

É justamente essa estrutura que deverá ser implementada neste projeto para a criação dos novos *stencils set*. Todo o conhecimento sobre a infraestrutura da

ferramenta *Oryx* necessário para a continuação do projeto foi identificado, finalizando o processo de engenharia reversa (Tabela 1).

A próxima subseção apresenta o diagrama conceitual da ferramenta desenvolvido utilizando o conhecimento extraído com a aplicação da engenharia reversa. A atividade de definição deste diagrama representa o subprocesso "Especificar", do método de reengenharia de software (Figura 5).

2.2.1.4. Resultados

Utilizando o conhecimento obtido na engenharia reversa, foi possível identificar os principais elementos e seus relacionamentos que compõem a estrutura da ferramenta *Oryx*. O diagrama conceitual ilustrado na Figura 18 apresenta esses elementos, sendo o elemento principal o *stencil set*.

O *stencil set* pode conter zero ou mais *stencils* e *rules*. Um *stencil* contém zero ou mais *properties* e *roles*. Uma *rule* pode ser de três tipos: C*onnectionRules, CardinalityRules* e *ContainmentRules.* Cada *rule* define uma *role* que pode ser referenciada pelo *stencil* que passa a respeitar a regra.

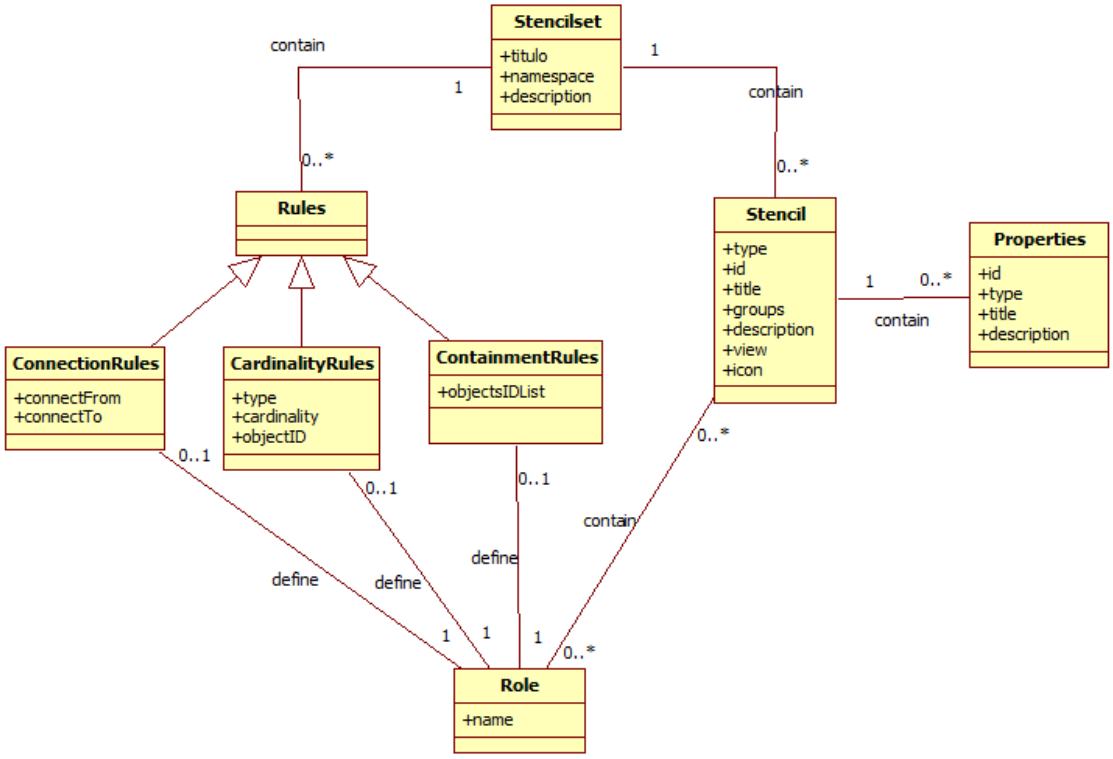


Figura 18 – Diagrama conceitual contendo elementos básicos de um *stencil set*

A definição deste modelo consolida o conhecimento sobre a infraestrutura da ferramenta *Oryx* e serve como insumo para a projeção de novos requisitos que necessários para executar a reengenharia na ferramenta *Oryx*. A partir deste momento, os esforços no desenvolvimento do software "caminham" em seu fluxo normal, partindo das especificações para a implementação, demarcando o fim da engenharia reversa e o inicio da engenharia progressiva. A próxima subseção detalha a nova projeção de requisitos, que se refere ao subprocesso "Reprojetar", do método de reengenharia de software (Figura 5).

2.2.2. Reprojeção de requisitos

Após finalizar a engenharia reversa e identificar os elementos necessários para a implementação dos *plugins*, a visão das reais necessidades para alcançar os objetivos do projeto tornou-se mais clara e a lista de requisitos pôde ser detalhada melhor. A segunda versão da lista está descrita a seguir:

- Criar *plugin* contendo i*
    - Criar elementos SVG com formato gráfico compatível ao i*
    - Criar ícones para cada elemento SVG
    - Implementar o arquivo *stencil set*
        - Criar *stencil* para cada elemento
        - Criar regras de acordo com as regras do i*
        - Aplicar as regras aos elementos de acordo com as regras do i*
- Completar *plugin* com a BPMN
    - Criar elementos SVG com formato gráfico compatível ao BPMN
    - Criar ícones para cada elemento SVG
    - Implementar o arquivo *stencil set*
        - Criar *stencils* para cada elemento do BPMN
        - Criar regras de acordo com as regras do BPMN
        - Aplicar as regras aos elementos de acordo com as regras do BPMN
- Seguir padrão de nomenclatura de outros *plugins* da ferramenta
- Seguir padrão de estrutura de outros *plugins* da ferramenta
- Permitir o carregamento mútuo dos elementos do BPMN e do i*

Partindo do conhecimento obtido até o momento e da projeção dos novos requisitos, sabemos que devemos implementar novas funcionalidades e reimplementar funcionalidades antigas para incluir os novos requisitos no sistema e realizar alterações/adaptações necessárias. A reimplementação do código faz parte do processo de reengenharia de software, que consiste em reutilizar os esforços de desenvolvimento passados [Premerlani&Blaha, 1994] para reorganizar e modificar sistemas de software existentes, parcial ou totalmente [Sommerville, 1995].

A próxima seção descreve sobre a reengenharia, sua aplicação na ferramenta *Oryx* e as novas necessidades que surgiram após gerar a versão 1.0 da ferramenta.

2.2.3. Reimplementação

Conforme dito anteriormente, o código do *Oryx* disponibilizado para *download* já possui uma implementação do BPMN 2.0, permitindo o seu reuso, o que consequentemente irá reduzir bastante o esforço projetado para a definição deste *stencil set*. Entretanto, é necessário incluir novos *stencils* do i* e suas regras, além de definir os elementos gráficos e realizar as alterações necessárias para adaptar o i* ao *stencil set* do BPMN.

O *stencil set* BPMN + i* foi projetado conforme o esquema apresentado na Figura 19. A maior dificuldade encontrada na implementação foi a ausência de ferramenta de *debug*, já que não há como acompanhar a execução do código de um *stencil set*, e um pequeno erro de uma virgula ausente já faz com que a ferramenta não funcione corretamente.

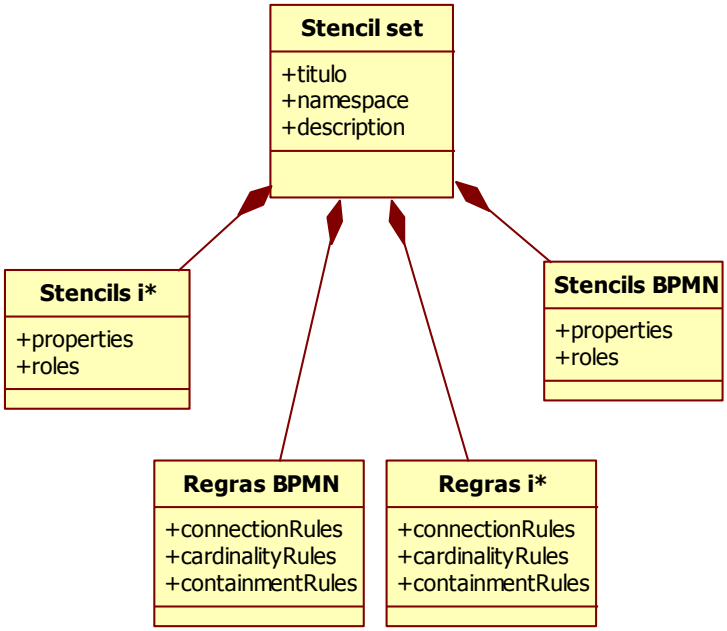


Figura 19 – Esquema de formação do *Stencil set* BPMN + i*

Durante as atividades de implementação, diversas vezes foi necessário apagar grande parte da codificação desenvolvida e reinserir novamente em partes, realizando testes de execução em paralelo para garantir que nenhum detalhe pudesse ficar ausente. Neste tipo de programação, onde se trabalha com um arquivo simples de *tags* e sem um sistema de *debug*, não é seguro aplicar um grande número de modificações para posteriormente realizar testes, já que um erro inserido em algum ponto do código demandará esforços maiores para ser rastreado e identificado.

Os sintomas que ocorrem por problemas de codificação são os mesmos para qualquer problema inserido no *stencil set*. O sistema não apresenta os elementos do *stencil set* no ambiente *Oryx* ou a ferramenta simplesmente não carrega integralmente, independente de carregar os elementos do *stencil set*.

Com a finalização do código, o projeto foi avaliado e revisto, gerando a necessidade de novo conjunto de modificações no *stencil set*. Além de modificações estritamente visuais, também definimos a necessidade de alteração na forma de apresentação dos elementos que consistiu em oferecer ao usuário a possibilidade de utilizar unicamente o i*, apresentando separadamente o modelo SD e SR; utilizar unicamente a BPMN; unir BPMN com SD; unir BPMN com SR. Essas alterações incluíram ao escopo do trabalho novos requisitos que demandariam tempo considerável, uma vez que, para implementar os *stencils* desta forma, seria necessária alterações no esquema do *stencil set* (Figura 19), bem como alterações em configurações dos arquivos do *Oryx*. A necessidade de inclusão de novos requisitos em momento posterior ao subprocesso "Reprojetar" não é abordado no método que utilizamos (Figura 5), entretanto, neste trabalho apenas retornamos a este subprocesso e reprojetamos os requisitos para serem implementados.

O *check list* de requisitos foi alterado conforme segue:

- Criar *plugin* contendo i*
    - Criar elementos SVG com formato gráfico compatível ao i*
    - Criar ícones para cada elemento SVG
    - Implementar o arquivo *stencil set*
        - Criar *stencil* para cada elemento
        - Criar regras de acordo com as regras do i*
        - Aplicar as regras aos elementos de acordo com as regras do i*
- Completar *plugin* com a BPMN
    - Criar elementos SVG com formato gráfico compatível ao BPMN
    - Criar ícones para cada elemento SVG
    - Implementar o arquivo *stencil set*
        - Criar *stencils* para cada elemento do BPMN
        - Criar regras de acordo com as regras do BPMN
        - Aplicar as regras aos elementos de acordo com as regras do BPMN
- Seguir padrão de nomenclatura de outros *plugins* da ferramenta
- Seguir padrão de estrutura de outros *plugins* da ferramenta
- **Permitir o carregamento do diagrama i* SD**
- **Permitir o carregamento do diagrama i* SR**
- **Permitir o carregamento do diagrama BPMN**
- **Permitir o carregamento do diagrama BPMN + SD**
- **Permitir o carregamento do diagrama BPMN + SR**

Para implementar esses requisitos, foram extraídos todos os elementos do *stencil set* implementado até o momento. Cada diagrama (i* SD, i* SR e BPMN) foi transformado em um único *stencil set* e foram criadas "perspectivas" na ferramenta para que o usuário possa selecionar e facilmente trocar entre os diferentes *plugins*.

A separação dos novos *stencil sets* não trazem nenhuma novidade ao trabalho, a não ser pela mudança do esquema de implementação (Figura 19), no entanto, a criação de perspectivas necessitou de alteração em um novo arquivo que define as perspectivas da ferramenta, bem como na estrutura de pastas dos arquivos.

O *Oryx* possui um arquivo chamado *extension.json* (Figura 20), onde são configurados *stencil sets* que estendem a capacidade de outros *stencil set* considerados como principais. Estas extensões são aplicadas ao modelo quando o usuário solicita a inclusão do *plugin* manualmente ou quando seleciona a perspectiva predefinida que contém *plugins* de extensão.

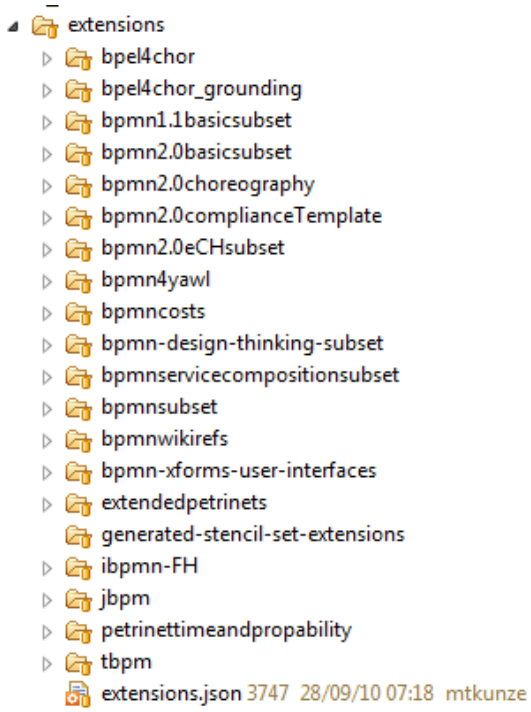


Figura 20 – Pasta "*extensions*" contendo o arquivo "*extensions.json*"

Esta estrutura de extensões foi utilizada como estratégia para a implementação das visões. Um *stencil set* genérico foi desenvolvido contendo o *stencil* básico do diagrama e o conjunto de regras genéricas que são usadas pelos *stencils set* que irão estendê-lo. Esse *stencil set* principal é iniciado na ferramenta, e as perspectivas definem os elementos que vão ser apresentados neste *stencil set* principal. Portanto cada *stencil set* implementado a partir da separação mencionada anteriormente (i* SD, i* SR e BPMN) foi configurado como uma extensão. A Figura 21 ilustra o esquema de formação do *stencil set* principal i*BPMN e seu relacionamento com suas extensões.

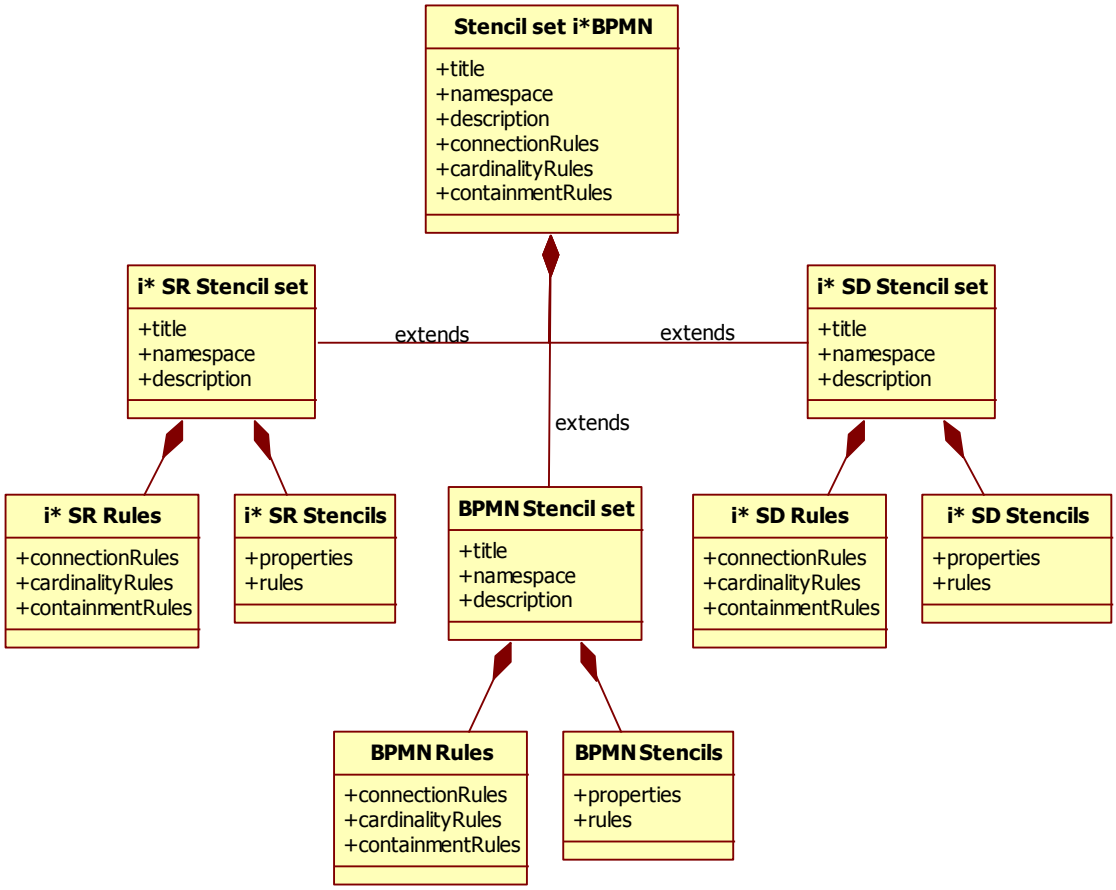


Figura 21 – Esquema de formação do *stencil set* i*BPMN a partir de extensões

O arquivo *extension.json* é formado pelo registro do conjunto de *stencils set* que são utilizados como extensão e o registro das perspectivas. O registro de um *stencil set* é formado por titulo, *namespace*, descrição, definição (*path* para o *stencil set* da extensão) e o campo de extensão, onde se define o *path* em que se encontra o *stencil set* a ser estendido. O registro de uma perspectiva é composto por titulo, *namespace*, descrição, *stencil set* que vai ser estendido e o campo onde se registra quais *stencils set* serão incluídos na perspectiva e outro campo onde se registra quais *stencils set* serão removidos da perspectiva (Figura 22).

```
{"extensions":
 [
     {
         "title":"iStar SD Extension",
         "namespace":"http://oryx-editor.org/stencilsets/extensions/iStar_SD_extension#",
         "description":"Include in the diagram the iStar SD elements",
         "definition":"iStar_SD_extension/iStar_SD_extension.json",
         "extends":"http://b3mn.org/stencilset/iStarBPMN#"
     }
 ],
 "perspectives":
 [
     {
         "title":"iStar SD",
         "namespace":"http://oryx-editor.org/stencilsets/perspectives/iStarSD#",
         "description":"iStar SD perspective. Features the complete standard.",
         "stencilset":"http://b3mn.org/stencilset/iStarBPMN#",
         "removeExtensions" : [ ],
         "addExtensions" : [ ]
     }
 ]}
```

Figura 22 – Exemplo do arquivo *extension.json*

Com essas alterações também se finaliza a etapa de implementação dos *plugins.* A próxima atividade é referente aos testes e correções do código e não possui mais relação com o processo de reengenharia de software adotado (Figura 5), entretanto consideramos que somente após esta etapa, a reengenharia da ferramenta estará finalizada. Por ter um conteúdo extenso, definimos um novo capítulo detalhando a definição do roteiro de testes e seus resultados.

3. Testes

Este capítulo apresenta informações de testes referentes ao *plugin* desenvolvido neste trabalho. Primeiramente é definido o roteiro a ser aplicado nos testes e posteriormente apresentados os respectivos resultados. Também são descritos um conjunto de problemas que são próprios da ferramenta *Oryx* (*bugs*) para evidenciar que certos comportamentos não possuem relação com os *plugins* testados.

3.1. Roteiro de testes

A codificação desenvolvida neste trabalho é um conjunto de especificações que são interpretadas pelo núcleo da ferramenta *Oryx* e geram um resultado gráfico em seu ambiente de modelagem. Para testar a codificação desenvolvida, foi escolhido o método de testes assistido, baseado em um roteiro onde são pré-registrados os resultados esperados por cada funcionalidade a ser testada. Uma vez que os resultados esperados são atingidos através do uso da funcionalidade, a função é considerada satisfatória. Caso contrário, é identificado o defeito que deve ser consertado e novamente testado ou registrada a existência do problema, caso seja avaliado e justificado que o código não deve ser alterado neste caso.

Os testes foram projetados a partir de roteiros onde se definem especificações que devem ser integralmente cumpridas ao utilizar a função na ferramenta. Os elementos alvo dos testes foram cada objeto e cada regra aplicada ao objeto.

Como um *plugin* é formado por *stencils* e regras, basicamente serão estes elementos que serão testados. A especificação dos elementos é repassada para a codificação e o resultado esperado é o elemento gráfico e regras de acordo com o esperado.

O *stencil set* referente ao BPMN 2.0, apesar de ter seu código reutilizado e reestruturado para adaptar-se à infraestrutura aplicada neste trabalho, já estava presente na aplicação *Oryx* e é produto de um trabalho de projeto final de graduação [Daniel *et al* 2007], portanto, espera-se de forma mínima que funcione corretamente e, por isso, este *stencil set* não será incluído nos testes.

Para cada *stencil* serão testados:

- No caso de ser um nó, o elemento gráfico esperado, localização do texto no objeto, efeito de ampliação do objeto, regras específicas.
- No caso de ser um relacionamento, o elemento gráfico esperado, localização do texto no objeto, efeito de ampliação do objeto, regras específicas e regras de conexão.

Nem todos os testes serão aplicáveis a todos os *stencils* e regras, sendo neste caso, marcado como não aplicável. A sintaxe geral do programa é automaticamente verificada pelo *Oryx*, que somente abre o *stencil set* para modelagem se todas as regras de sintaxe estiverem corretas.

As regras serão testadas separadamente para cada situação esperada por um objeto em relação aos outros elementos que fazem parte do domínio da regra. Para estes testes serão preenchidos os modelos ilustrados na

Tabela 2, Tabela 3, Tabela 4 e Tabela 5.

O objetivo do teste gráfico (

Tabela 2) é verificar se o elemento está de acordo com as especificações esperadas. A verificação de aceite do teste é visual e comparativa. A especificação padrão dos elementos é composta dos seguintes campos: Elemento – nome do elemento na ferramenta; Formato – formato gráfico do elemento; Cor – cor do corpo do elemento; Texto – Configuração do texto em relação ao objeto;

O campo Resultado, mostra a figura que é resultante do código que foi implementado. Esta figura deverá ser comparada com a especificação para verificar se o objeto obedece a todas as especificações, o que caracteriza um teste com sucesso. Uma vez que o elemento passou no teste, o campo Teste é preenchido com "OK".

Tabela 2 – Teste gráfico

Teste gráfico	
Especificação	**Resultado**
Elemento:	
Formato:	
Cor:	
Texto:	
Teste:	

Os Testes de regras seguem o padrão do Teste gráfico, porém possuem parâmetros que são específicos de acordo com as características da regra. O Teste de regras do tipo *Containment Rules* possui apenas 2 parâmetros: o Elemento – nome do elemento que é o alvo da regra; Conteúdo – lista de elementos que podem conter no elemento principal.

Tabela 3 – Teste de regras do tipo Containment Rules

Teste de regras (*Containment Rules*)	
Especificação	**Resultado**
Elemento:	
Conteúdo:	
Teste:	

O Teste de regras do tipo *Cardinality Rules* possuem os seguintes parâmetros: elemento – nome do elemento o qual a regra está atribuída; elemento de entrada – nome do relacionamento que chega ao elemento; Card. Entrada – Cardinalidade do elemento de entrada; Elemento de saída – nome do relacionamento que sai do elemento; Card. Saída – Cardinalidade do elemento de saída.

Tabela 4 – Teste de regras do tipo *Cardinality Rules*

Teste de regras (*Cardinality Rules*)	
Especificação	**Resultado**
Elemento:	
Elemento de entrada	
Card. Entrada:	
Elemento de saída:	
Card. Saída:	
Teste:	

O teste de regras do tipo *Connection Rules* possui os seguintes parâmetros: Elemento – nome do relacionamento que será avaliado; Elem. Inicial – nome do elemento que deve estar na ponta inicial do relacionamento; Elem. Final – nome do elemento que deve estar na ponta final do relacionamento.

Tabela 5 – Teste de regras do tipo *Connection Rules*

Teste de regras (*Connection Rules*)	
Especificação	**Resultado**
Elemento:	
Elem. Inicial:	
Elem. Final:	
Teste:	

Testes de "negação" das regras não serão aplicados já que o *Oryx* automaticamente impede a execução de regras que não estão explicitas. Por exemplo, se definir somente uma regra que relaciona o objeto A ao objeto B nesta direção, a partir do relacionamento X, não é necessário testar a conexão para a direção inversa porque ela não acontecerá, já que não está definida.

A próxima subseção apresenta o resultado dos testes aplicados nos padrões de teste definidos nesta seção.

3.2. Resultado dos testes

Os resultados aceitáveis dos testes subsequentes correspondem à satisfação dos seguintes requisitos contidos na lista de requisitos:

- Criar *plugin* contendo i*
    - Criar elementos SVG com formato gráfico compatível ao i*
    - Criar ícones para cada elemento SVG
    - Implementar o arquivo *stencil set*

        - Criar *stencil* para cada elemento
        - Criar regras de acordo com as regras do i*
        - Aplicar as regras aos elementos de acordo com as regras do i*
- Permitir o carregamento do diagrama i* SD
- Permitir o carregamento do diagrama i* SR
- Permitir o carregamento do diagrama BPMN
- Permitir o carregamento do diagrama BPMN + SD
- Permitir o carregamento do diagrama BPMN + SR

É válido lembrar que os requisitos referentes ao *plugin* da BPMN são considerados satisfeitos devido ao reuso de seu código, conforme descrito anteriormente. Esses requisitos são listados abaixo:

- Criar elementos SVG com formato gráfico compatível ao BPMN
- Criar ícones para cada elemento SVG
- Implementar o arquivo *stencil set*
    - Criar *stencils* para cada elemento do BPMN
    - Criar regras de acordo com as regras do BPMN
    - Aplicar as regras aos elementos de acordo com as regras do BPMN

3.2.1. Diagrama principal (iStarBPMN)

Os testes subsequentes são referentes ao diagrama principal do sistema, que é responsável por receber os outros elementos que o estendem.

Tabela 6 – Teste gráfico para o elemento iStarBPMNDiagram (Diagrama principal)

Teste gráfico	
Especificação	**Resultado**
Elemento: iStarBPMNDiagram	
Formato: Ocupa todo o campo disponível	
Cor: azul claro	
Texto: N/A	
Teste: OK	

Tabela 7 – Teste da regra *Containment Rules* para o elemento iStarBPMNDiagram (Diagrama principal)

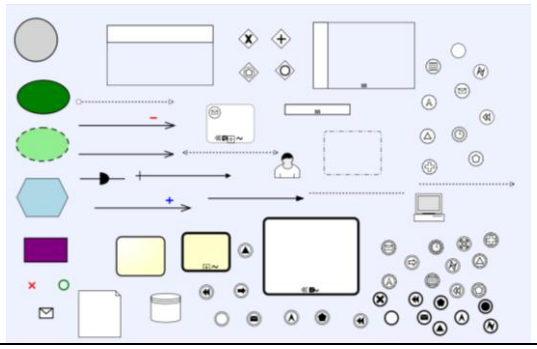
Teste de regras (*Containment Rules*)	
Especificação	**Resultado**
Elemento: iStarBPMNDiagram	
Conteúdo: Todos os elementos do i* e da BPMN	
Teste: OK	

Tabela 8 – Teste da regra *Cardinality Rules* para os elementos do tipo Evento inicial (Diagrama principal)

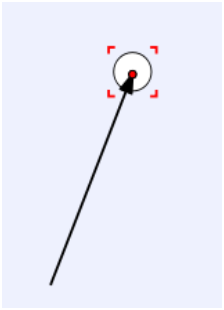
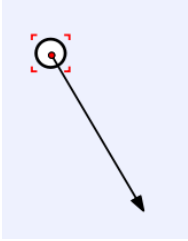
Teste de regras (*Cardinality Rules*)	
Especificação	**Resultado**
Elemento: Regra genérica (Eventos iniciais)	
Elemento de entrada: SequenceFlow	
Card. Entrada: 0	
Elemento de saída: N/A	
Card. Saída: N/A	
Teste: OK	

Tabela 9 – Teste da regra *Cardinality Rules* para o elemento do tipo Evento final (Diagrama principal)

Teste de regras (*Cardinality Rules*)	
Especificação	**Resultado**
Elemento: Regra genérica (Eventos finais)	
Elemento de entrada: N/A	
Card. Entrada: N/A	
Elemento de saída: SequenceFlow	
Card. Saída: 0	
Teste: OK	

3.2.2. Diagrama iStar SD

Os testes subsequentes são referentes aos elementos *Agente, Agent Lane, Hardgoal, Softgoal, Task, Resource, Critical Mark, Open Mark* e *Dependency* que compõem o *stencil set* iStar SD:

Tabela 10 – Teste gráfico para o elemento *Agent* (Diagrama SD)

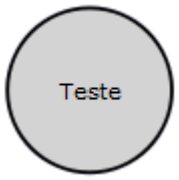
Teste gráfico	
Especificação	**Resultado**
Elemento: *Agent*	
Formato: circular	
Cor: cinza	
Texto: centralizado	
Teste: OK	

Tabela 11 – Teste gráfico para o elemento *Hardgoal* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Hardgoal*	
Formato: oval	
Cor: verde	
Texto: centralizado	
Teste: OK	

Tabela 12 – Teste gráfico para o elemento *Softgoal* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Softgoal*	
Formato: oval, com desenho tracejado.	
Cor: verde claro	
Texto: centralizado	
Teste: OK	

Tabela 13 – Teste gráfico para o elemento *Task* (Diagrama SD)

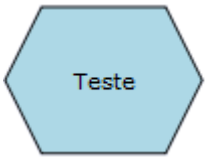
Teste gráfico	
Especificação	**Resultado**
Elemento: *Task*	
Formato: Hexagonal, com topo e fundo reto. Lados equivalentes e equidistantes (topo=fundo, esquerda=direita).	
Cor: azul	
Texto: centralizado	
Teste: OK	

Tabela 14 – Teste gráfico para o elemento *Resource* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Resource*	
Formato: retângulo	
Cor: roxo	
Texto: centralizado	
Teste: OK	

Tabela 15 – Teste gráfico para o elemento *AgentLane* (Diagrama SD)

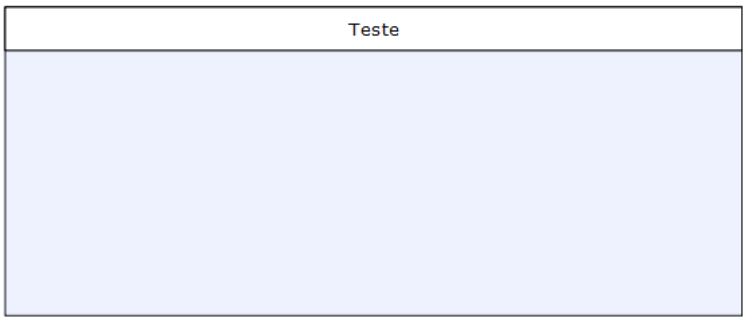
Teste gráfico	
Especificação	**Resultado**
Elemento: *AgentLane*	
Formato: retângulo, com campo de texto no canto superior também retangular.	
Cor: campo de texto branco; corpo azul claro.	
Texto: centralizado	
Teste: OK	

Tabela 16 – Teste gráfico para o elemento *Critical Mark* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Critical Mark*	
Formato: X	
Cor: vermelho	
Texto: N/A	
Teste: OK	

Tabela 17 – Teste gráfico para o elemento *Open Mark* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Open mark*	
Formato: circular	
Cor: Borda verde, fundo branco	
Texto: N/A	
Teste: OK	

Tabela 18 – Teste gráfico para o elemento *Dependency* (Diagrama SD)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Dependency*	
Formato: reta, com elemento central com formado de "D".	
Cor: preto	
Texto: N/A	
Teste: OK	

Tabela 19 – Teste da regra *Containment Rule* para o elemento *AgentLane* (Diagrama SD)

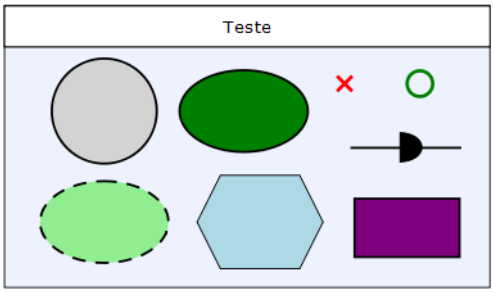
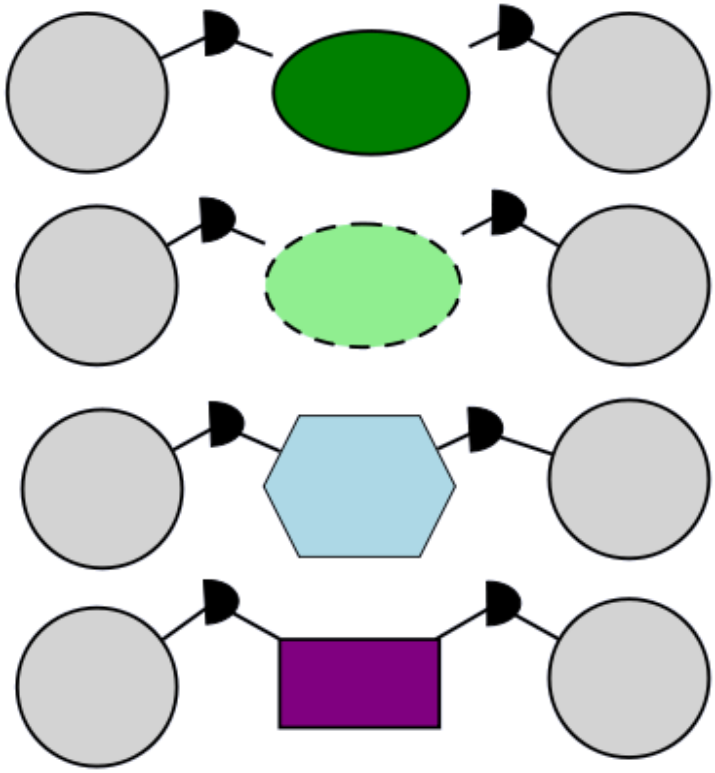
Teste de regras (*Containment Rules*)	
Especificação	**Resultado**
Elemento: *AgentLane*	
Conteúdo: *Agent, Hardgoal, Softgoal, Task, Resource, Critical Mark, Open Mark, Dependency.*	
Teste: OK	

Tabela 20 – Teste da regra *Connection Rules* para os elementos que utilizam o relacionamento *Dependency* (Diagrama SD)

Teste de regras (*Connection Rules*)	
Especificação	**Resultado**
Elemento: *Dependecy*	
Elem. Inicial: *Agent*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Agent*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Agent*	
Elem. Final: *Task*	
Elem. Inicial: *Agent*	
Elem. Final: *Resource*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Agent*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Agent*	
Elem. Inicial: *Task*	
Elem. Final: *Agent*	
Elem. Inicial: *Resource*	
Elem. Final: *Agent*	

3.2.3. Diagrama iStar SR

Os testes subsequentes são referentes aos elementos *Agente, Agent Lane, Hardgoal, Softgoal, Task, Resource, Critical Mark, Open Mark*, *Dependency, Task Decomposition, Means-Ends, Positive Contribuition* e *Negative Contribuition,* que compõem o *stencil set* iStar SD. A repetição dos testes do conjunto de elementos que estão presentes no diagrama iStar SD se justifica porque, apesar de serem iguais, são *stencils set* distintos, e a codificação encontra-se em arquivos diferentes.

Tabela 21 – Teste gráfico para o elemento *Agent* (Diagrama SR)

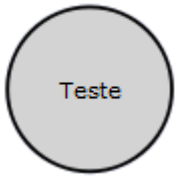
Teste gráfico	
Especificação	**Resultado**
Elemento: *Agent*	
Formato: circular	
Cor: cinza	
Texto: centralizado	
Teste: OK	

Tabela 22 – Teste gráfico para o elemento *Hardgoal* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Hardgoal*	
Formato: oval	
Cor: verde	
Texto: centralizado	
Teste: OK	

Tabela 23 – Teste gráfico para o elemento *Softgoal* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Softgoal*	
Formato: oval, com desenho tracejado.	
Cor: verde claro	
Texto: centralizado	
Teste: OK	

Tabela 24 – Teste gráfico para o elemento *Task* (Diagrama SR)

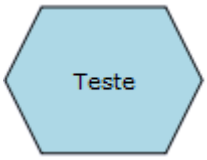
Teste gráfico	
Especificação	**Resultado**
Elemento: *Task*	
Formato: Hexagonal, com topo e fundo reto. Lados equivalentes e equidistantes (topo=fundo, esquerda=direita).	
Cor: azul	
Texto: centralizado	
Teste: OK	

Tabela 25 – Teste gráfico para o elemento *Resource* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Resource*	
Formato: retângulo	
Cor: roxo	
Texto: centralizado	
Teste: OK	

Tabela 26 – Teste gráfico para o elemento *AgentLane* (Diagrama SR)

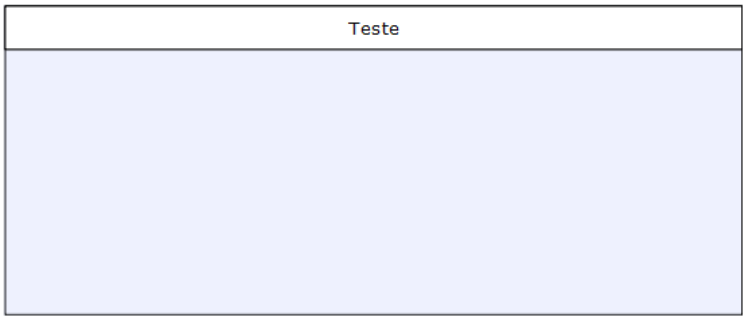
Teste gráfico	
Especificação	**Resultado**
Elemento: *AgentLane*	
Formato: retângulo, com campo de texto no canto superior também retangular.	
Cor: campo de texto branco; corpo azul claro.	
Texto: centralizado	
Teste: OK	

Tabela 27 – Teste gráfico para o elemento *Critical Mark* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Critical Mark*	
Formato: X	
Cor: vermelho	
Texto: N/A	
Teste: OK	

Tabela 28 – Teste gráfico para o elemento *Open Mark* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Open Mark*	
Formato: circular	
Cor: Borda verde, fundo branco.	
Texto: N/A	
Teste: OK	

Tabela 29 – Teste gráfico para o elemento *Dependency* (Diagrama SR)

Teste gráfico	
Especificação	**Resultado**
Elemento: *Dependency*	
Formato: reta, com elemento central com formado de "D".	
Cor: preto	
Texto: N/A	
Teste: OK	

Tabela 30 – Teste gráfico para o elemento *Task Decomposition* (Diagrama SR)

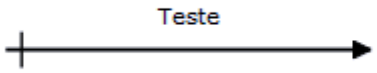
Teste gráfico	
Especificação	**Resultado**
Elemento: *Task Decomposition*	
Formato: reta, com pequena reta transversal na parte posterior e seta em formato de triangulo, pequeno, preto, opaco, na parte posterior.	
Cor: preto	
Texto: Parte superior, centralizado.	
Teste: OK	

Tabela 31 – Teste gráfico para o elemento *Means – Ends* (Diagrama SR)

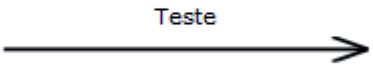
Teste gráfico	
Especificação	**Resultado**
Elemento: *Means – Ends*	
Formato: reta, com seta aberta na parte posterior.	
Cor: preto	
Texto: parte superior, centralizado.	
Teste: OK	

Tabela 32 – Teste gráfico para o elemento *Positive Contribuition* (Diagrama SR)

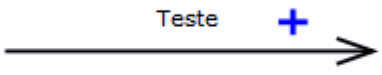
Teste gráfico	
Especificação	**Resultado**
Elemento: *Positive Contribuition*	
Formato: reta, com seta aberta na parte posterior e elemento em forma de cruz, azul também na parte posterior, representando a contribuição positiva.	
Cor: preto	
Texto: parte superior, centralizado.	
Teste: OK	

Tabela 33 – Teste gráfico para o elemento *Negative Contribuition* (Diagrama SR)

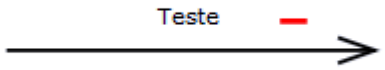
Teste gráfico	
Especificação	**Resultado**
Elemento: *Negative Contribuition*	
Formato: reta, com seta aberta na parte posterior e elemento em forma de traço, vermelho também na parte posterior, representando a contribuição negativa.	
Cor: preto	
Texto: parte superior, centralizado.	
Teste: OK	

Tabela 34 – Teste da regra *Containment Rules* para o elemento *AgentLane* (Diagrama SR)

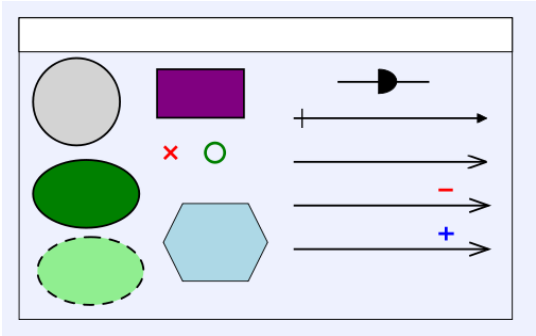
Teste de regras (*Containment Rules*)	
Especificação	**Resultado**
Elemento: *AgentLane*	
Conteúdo: *Agent, Hardgoal, Softgoal, Task, Resource, Critical Mark, Open Mark, Dependency, Task Decomposition, Means-Ends, Positive Contribuition, Negative Contribuition*.	
Teste: OK	

Tabela 35 – Teste da regra *Connection Rules* com o relacionamento *Dependency* (SR)

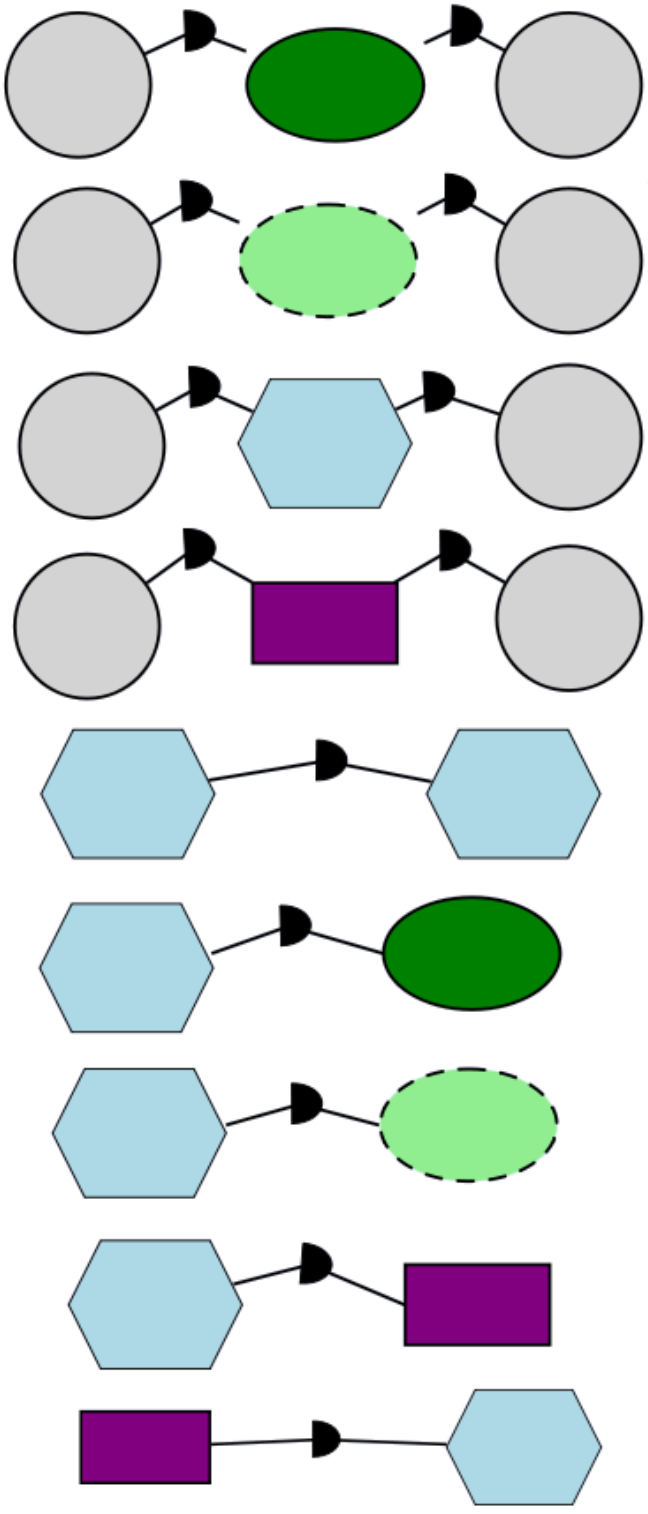
Teste de regras (*Connection Rules*)	
Especificação	**Resultado**
Elemento: *Dependecy*	
Elem. Inicial: *Agent*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Agent*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Agent*	
Elem. Final: *Task*	
Elem. Inicial: *Agent*	
Elem. Final: *Resource*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Agent*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Agent*	
Elem. Inicial: *Task*	
Elem. Final: *Agent*	
Elem. Inicial: *Resource*	
Elem. Final: *Agent*	
Elem. Inicial: *Task*	
Elem. Final: *Task*	
Elem. Inicial: *Task*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Resource*	
Elem. Inicial: *Resource*	
Elem. Final: *Task*	
Teste: OK	

Tabela 36 – Teste da regra *Connection Rules* para os elementos que utilizam o relacionamento *Task Decomposition* (Diagrama SR)

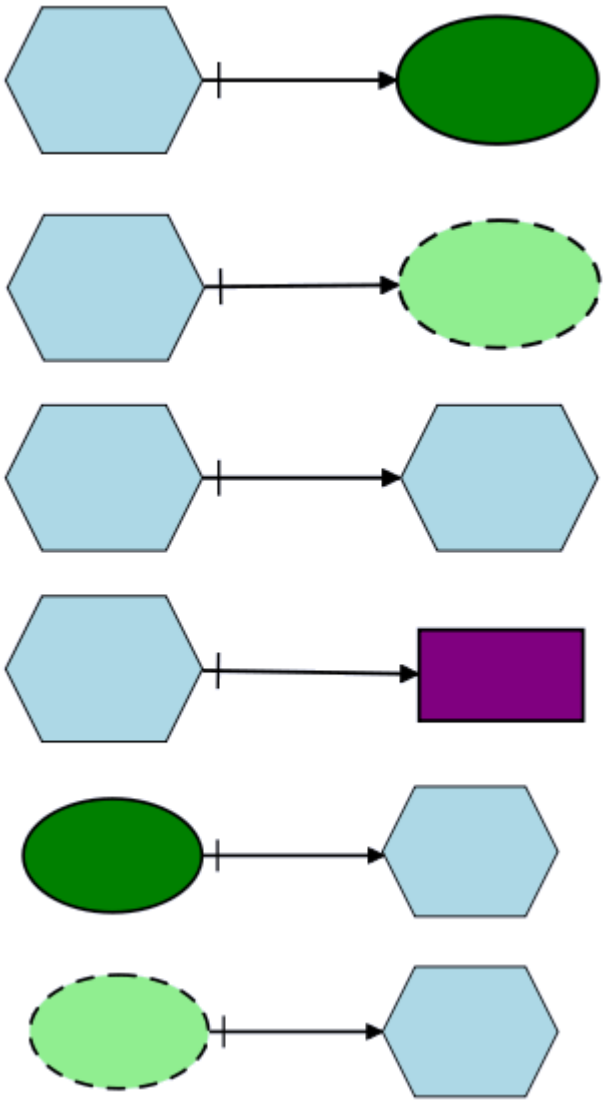
Teste de regras (Connection Rules)	
Especificação	**Resultado**
Elemento: *Task Decomposition*	
Elem. Inicial: *Task*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Task*	
Elem. Inicial: *Task*	
Elem. Final: *Resource*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Task*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Task*	
Teste: OK	

Tabela 37 – Teste da regra *Connection Rules* para os elementos que utilizam o relacionamento *Means-Ends* (Diagrama SR)

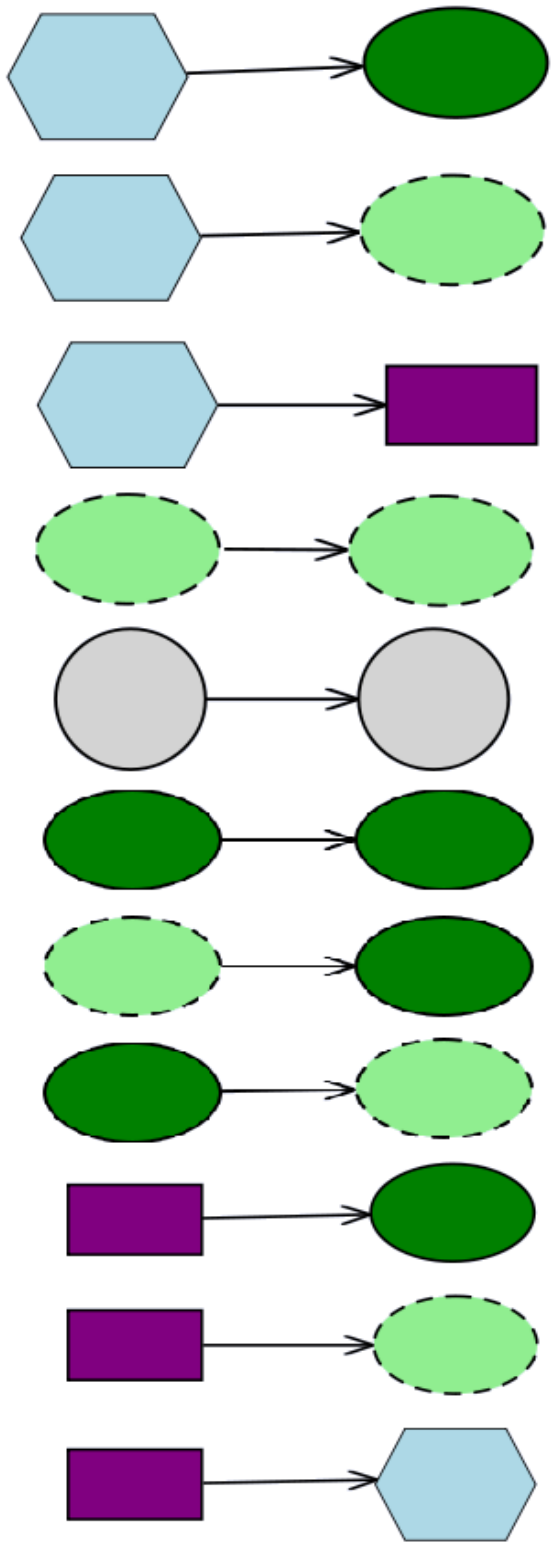
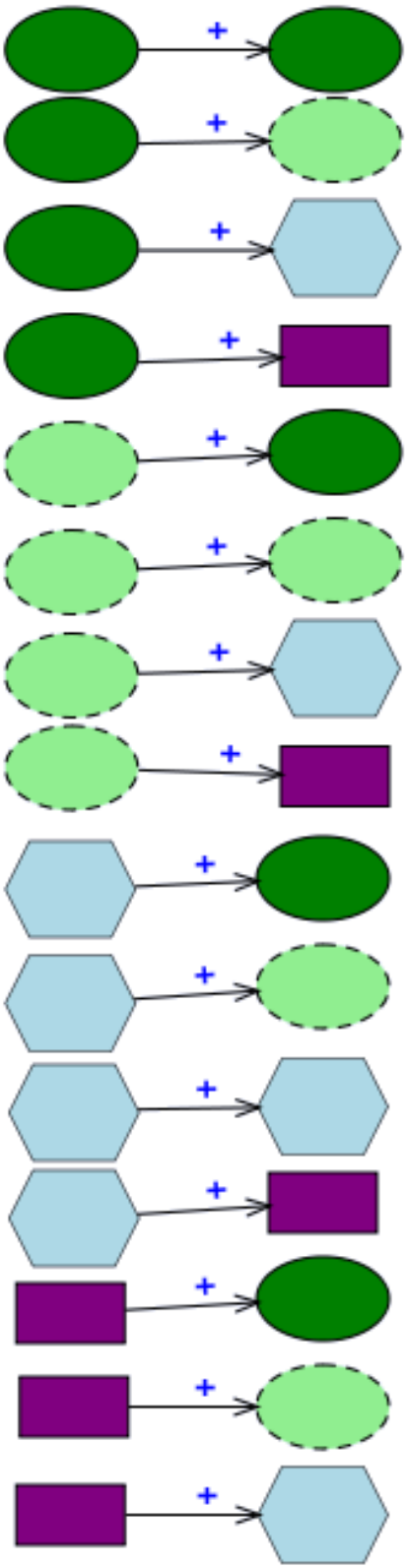
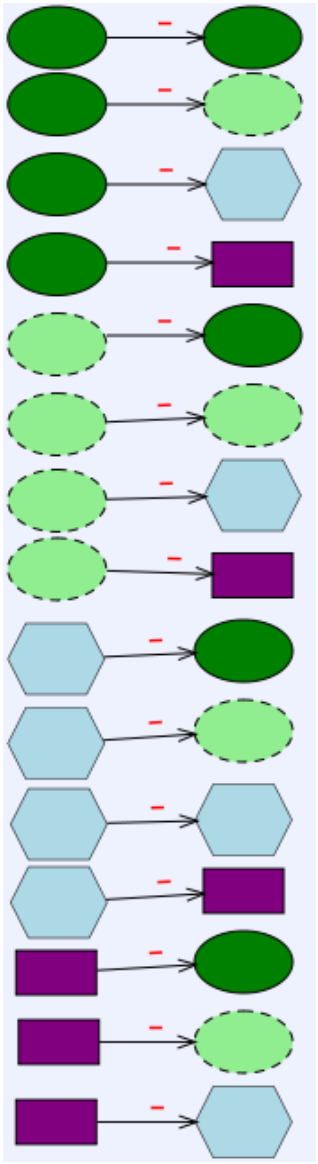
Teste de regras (Connection Rules)	
Especificação	**Resultado**
Elemento: *Means – Ends*	
Elem. Inicial: *Task Decomposition*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Task Decomposition*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Task Decomposition*	
Elem. Final: *Resource*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Agent*	
Elem. Final: *Agent*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Task*	
Teste: OK	

Tabela 38 – Teste da regra *Connection Rules* para os elementos que utilizam o relacionamento *Positive Contribuition* (Diagrama SR)

Teste de regras (Connection Rules)	
Especificação	**Resultado**
Elemento: *Positive Contribuition*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Task*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Resource*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Task*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Resource*	
Elem. Inicial: *Task*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Task*	
Elem. Inicial: *Task*	
Elem. Final: *Resource*	
Elem. Inicial: *Resource*	

Elem. Final: *Hardgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Task*	
Teste: OK	

Tabela 39 – Teste da regra *Connection Rules* para os elementos que utilizam o relacionamento *Negative Contribuition* (Diagrama SR)

Teste de regras (Connection Rules)	
Especificação	**Resultado**
Elemento: *Negative Contribuition*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Task*	
Elem. Inicial: *Hardgoal*	
Elem. Final: *Resource*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Task*	
Elem. Inicial: *Softgoal*	
Elem. Final: *Resource*	
Elem. Inicial: *Task*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Task*	

Elem. Final: *Softgoal*	
Elem. Inicial: *Task*	
Elem. Final: *Task*	
Elem. Inicial: *Task*	
Elem. Final: *Resource*	
Elem. Inicial: *Resource*	
Elem. Final: *Hardgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Softgoal*	
Elem. Inicial: *Resource*	
Elem. Final: *Task*	
Teste: OK	

3.2.4. Teste de chamada das perspectivas

O teste de chamada das perspectivas consiste em verificar se as perspectivas programadas estão importando os *stencils set* corretos. Este teste é simples, bastando apenas selecionar as perspectivas e verificar se os elementos dos *stencils set* tornam-se disponíveis na tela para uso. As figuras abaixo demonstram a execução correta de todas as perspectivas (Figura 23, Figura 24, Figura 25,

Figura 26 e Figura 27):

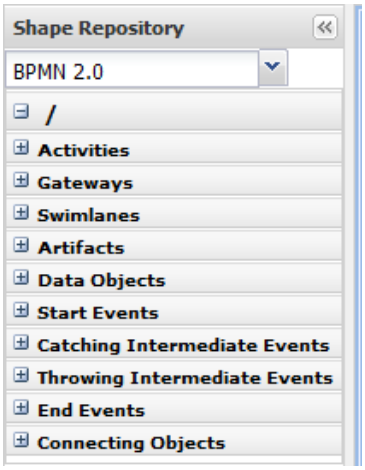


Figura 23 – Perspectiva BPMN 2.0

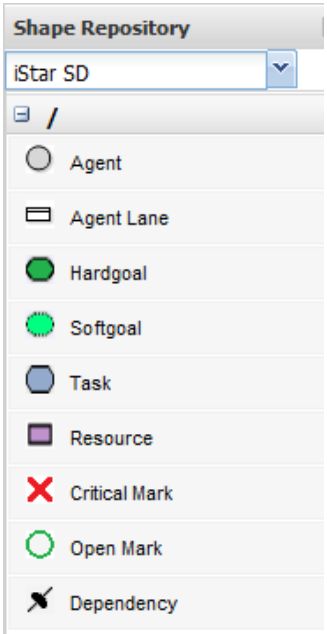


Figura 24 – Perspectiva iStar SD

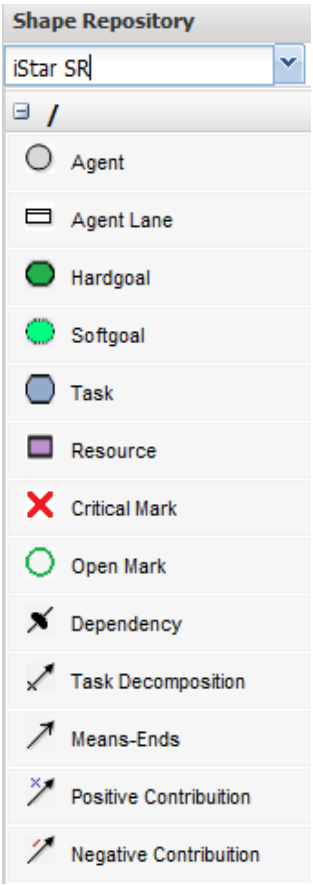


Figura 25 – Perspectiva iStar SR

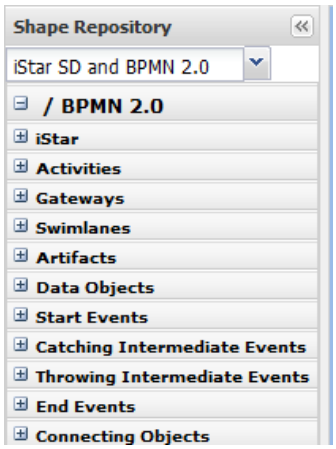


Figura 26 – Perspectiva iStar SD and BPMN 2.0

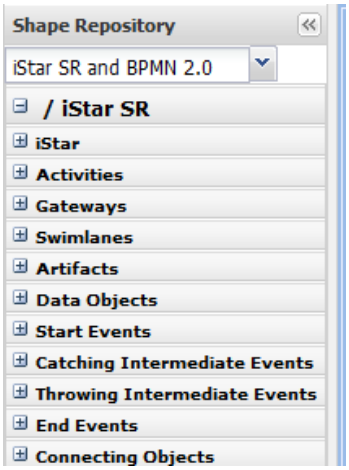


Figura 27 – Perspectiva iStar SR *and* BPMN 2.0

Um problema foi identificado ao executar os testes de perspectivas. Ao trocar de perspectiva ou simplesmente eliminar o *stencil set* do programa pela caixa de seleção de adição de *plugins*, o software eliminava todos os *stencil*s, porém deixava o rastro dos elementos do tipo *EDGE*. Qualquer elemento deste tipo permanecia no modelo ao retirar seu *stencil set*. Como a execução dos procedimentos para inserir e retirar os *stencils* (baseado nos comandos do usuário) é de responsabilidade do código presente no núcleo da ferramenta (não abordado neste trabalho), não foi possível identificar o problema para solucioná-lo. Uma saída encontrada para contornar o problema foi a implementação de duas novas extensões que, quando instaladas, apenas retiram esses elementos. Então a única configuração realizada foi aplicar essas extensões estrategicamente no momento de trocar a opção das perspectivas. Com essa solução, o problema foi eliminado de forma aceitável.

Todos os problemas encontrados foram corrigidos, incluindo o problema ao executar as perspectivas, conforme apresentado anteriormente. Diversos *bugs* da ferramenta foram identificados durante o seu uso, e foram registrados na subseção seguinte. Esses problemas não possuem relação com o código desenvolvido neste trabalho.

3.3. *Bugs* conhecidos

Esta subseção apresenta uma lista de *bugs* conhecidos nas funcionalidades e usabilidade da ferramenta, e não foram levados em consideração nos testes, já que são erros intrínsecos à versão da ferramenta *Oryx*. A lista de problema é apresentada a seguir:

- Não funciona corretamente no Internet Explorer
- Não permite copiar e colar objetos em modelos diferentes
- Não permite salvar modelos localmente
- Ao importar modelos em linguagens padrão aceitos pela ferramenta, sempre ocorre algum desvio de localização de elementos ao restaurar o modelo exportado.
- Ao exportar modelos em BPMN 2.0 DI XML, a ferramenta apresentou em diversos testes um aviso de erro de validação. Ao importar o arquivo, a atributo o qual a ferramenta reportou o erro no momento da exportação encontra-se ausente no modelo importado.
- Após algumas operações na ferramenta, é possível que a opção "Desfazer" e "Refazer" parem de funcionar corretamente.
- A opção "*Connect several nodes by marking then in the desired order"* não funcionou corretamente. Ao selecionar o primeiro objeto, e o segundo objeto, a ferramenta trava a seleção de novos objetos e não permite mais a troca do foco do elemento.
- As opções "*Fades tbpm foto in and out*" e "*Evaluate Partial Process Model"* não funcionaram durante os testes da ferramenta.
- O modelo trava constantemente ao manipular objetos (principalmente setas), utilizar as funções "Desfazer" e "Refazer" e copiar e colar objetos. O travamento é permanente, obrigando o usuário a refazer alterações que não foram salvas.
- As setas que possuem gráficos centrais (por exemplo, o relacionamento "*Dependency"* do i*) são transformadas em linhas retas ao desenhar o objeto de forma retilínea.

Após a finalização da etapa de testes e correções, encerra-se a reengenharia da ferramenta *Oryx*, tendo como resultado os *plugins* que contém as definições das linguagens i* e BPMN, podendo ser configurados para uso separados ou em conjunto. Os testes garantiram que os modelos obedecem às respectivas regras de sintaxe e funcionam de forma satisfatória. O próximo capitulo apresenta as conclusões deste trabalho.

4. Conclusão

O presente trabalho apresentou a extensão da ferramenta *Oryx* através do desenvolvimento de *plugins* de forma a permitir a modelagem de diagramas como o *Strategic Dependency (SD), Strategic Rationale (SR)* do i* e BPMM. Os diagramas das respectivas linguagens e notações podem ser acessados tanto em sua forma padrão como em conjunto com outras, tais como, um diagrama contendo elementos i*SD e BPMN ou i*SR e BPMN.

Para a execução deste trabalho foi necessário realizar estudos em busca de conhecimento sobre como implementar *plugins* para a ferramenta *Oryx* e posteriormente gerar o código abrangendo as novas funcionalidades. Para isso foi aplicada a reengenharia de software através do método proposto em [Leite, 2006], o que possibilitou estender a ferramenta com os novos *plugins*.

A execução deste trabalho permitiu o estudo dos processos e técnicas da engenharia de software através a implementação de novos *stencil set* na ferramenta Oryx. As atividades desenvolvidas se diferenciam da programação tradicional, uma vez que os *stencils* possuem sua codificação baseada na especificação das características de elementos que compõem a dada linguagem, diferente de projetos padrões.

Outro fator importante foi o conhecimento da estrutura de arquivos SVG, que permitem a criação de diversas figuras a partir de especificação por código. Esta estrutura padronizada permite o desenho de elementos mais precisos, diferente do uso de ferramentas conhecidas, como o *MSPaint* ou o *Corel Draw*.

O estudo da infraestrutura do *Oryx* também é importante já que além de agregar conhecimento sobre a forma de implementação dos *stencils set*, também se transforma em um recurso para futuras extensões de linguagens.

Nos trabalhos futuros serão realizados estudos do núcleo da ferramenta Oryx para a implementação de melhorias de suas funcionalidades e correções dos *bugs* citados neste trabalho.

Referências

[Cardoso *et al*, 2011] Cardoso, E.C.S.; Guizzardi, R.S.S.; Almeida, J.P.A.; "Aligning Goal Analysis and Business Process Modelling: A Case Study in Health Care", International Journal of Business Process Integration and Management, 2011.

[Chikofsky&Cross II, 1990] Chikofsky, E. J.; Cross II, J. H.; "Reverse Engeneering and Design Recovery": a Taxonomy", IEEE Software, v.7, n.1, 1990, p. 13-17.

[Daniel *et al*, 2007] Daniel, P., Weske, M., Overdick, H., Decker, G., "*Oryx* BPMN Stencil Set Implementation", Bachelor Thesis, Hasso Plattner Institut, 30/06/2007, disponível em "http://*Oryx*-project.org/research".

[Decker *et al*, 2008] Decker, G., Overdick, H., Weske, M., "*Oryx* – An Open Modeling Platform for the BPM Community", In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, BPM, volume 5240 of Lecture Notes in Computer Science, pages 382–385. Springer, 2008.

[GT-RED, 1998] GT-RED (Georgia Tech's – Reverse Engineering Group), "Glossary of Reengineering Terms", Georgia, "www.cc.gatech.edu/reverse/glossary.html, 1998.

[Heuser, 2000] Heuser, C. A., "Projeto de Banco de Dados", sexta edição, Editora Artmed, ISBN: 9788577803828.

[IEEE CS-TCSE, 1997] IEEE CS-TCSE (IEEE Computer Society- Technical Council on Software Engeneering); "Reengineering and Reverse Engineering Terminology", Washington, "www.tcse.org/revengr/taxonomy.html, 1997.

[Kunze&Weske, 2010] Kunze, M., Weske, M., "Signavio-*Oryx* Academic Initiative", Demo Session of the 8th International Conference on Business Process Management (BPM 2010). Hoboken, NJ, September 2010.

[Leite, 1996] Leite, J.C.S.P., "Working Results on Software Re-Engineering", ACM SIGSOFT, Software Engineering Notes vol. 21 n° 2, page 39-44, march, 1996.

[Oliveira *et al*, 2008] Oliveira, A.; Padua A.; Leite, J. C. S. P; Cysneiros, L. M.; "Engenharia de requisitos intencional: um método de elicitação, modelagem e análise de requisitos", tese de doutorado, PUC-RIO, 03/2008.

[Oliveira *et al*, 2010] Oliveira, A.; Padua A.; Leite, J. C. S. P; Cysneiros, L. M.; "Using i* Meta Modeling for Verifying i* Models", Proceedings of the 4th International i* Workshop – iStar10, 76-80, Hammamet, Tunisia, 2010.

[OMG, 2010] OMG, "Business Process Model and Notation Version 2.0", 2010, disponível em "http://www.bpmn.org/".

[*Oryx*, 2010] *Oryx*, Site oficial *Oryx*, disponível em " TTP://*Oryx*-project.org/research", acessado em "20/12/2010".

[Peters *et al*, 2007] Peters, N., Weske, M., Overdick, H., Decker, G., "*Oryx* Stencil Set Specification", Final Bachelor's Page, Hasso Plattner Institut, 30/06/2007, disponível em "http://*Oryx*-project.org/research".

[Piekarski&Quinaia, 1995] Piekarski, A.E.T., Quinaia, M.A.; "Reengenharia de software: o que, por quê e como", PRESSMAN, R. S. Engenharia de Software. São Paulo:

Makron Books, 1995.

[Premerlani&Blaha, 1994] Premerlani, W. J., Blaha, M. R., "An approach for Reverse Engineering of Relational Databases", Communications of the ACM, V.37, n.5, 1994, p.42-49.

[Pressman, 1995] Pressman, R. S., "Engenharia de Software", São Paulo: Makron Books, ISBN : 8534602379 ISBN-13: 9788534602372 3ª Edição - 1995 - 1090 pág.

[Rezende, 2005] Rezende, D. A., "Engenharia de software e sistemas de informação", terceira edição, Rio de Janeiro, Brasport, 2005, ISBN: 85-7452-215-5, CDD:005.1.

[Susi *et al*, 2005] Susi, A., Perini, A., Mylopoulos, J; Giorgini, P.; "The Tropos Metamodel and its Use", Informatical Journal, 5/2005.

[Serpro, 2011] Serpro, "Revista tema", n 208, setembro/outubro, 2011. ISSN: 0100-5227.

[Sommerville, 1995] Sommerville, I.; "Software Engineering", International Computer Science Series, 5 edição, Editora: Addison-Wesley, 1995.

[Tscheschner *et al*, 2007] Tscheschner, W., Weske, M., Overdick, H., Decker, G., "*Oryx* Dokumentation", Bachelorabeit, Hasso Plattner Institut, 30/06/2007, disponível em "http://*Oryx*-project.org/research".

[Warden, 1992] Warden, R. "Re-enginnering – A Pratical Methodology with Commercial Applications", Applied Information Tecnology 12 (Software Reuse and Reverse Engineering in Pratice), Chapman@Hall, 1992.

[Yu, 1995] Yu, E., "Modelling Strategic Relationships for Process Reengineering". Phd Thesis, Graduate Department of Computer Science, University of Toronto, Toronto, Canada, 1995, pp.124.

[Yu, 2001] Yu, Eric, "Agent-Oriented Modelling: Software versus the world in agent-oriented software engineering", AOSE-2001, Workshop Proceedings, Montreal, Canada, 29/03/2001, LNCS 2222.

[Yu, 2009] Yu, E., "Social Modeling and i*". In: Borgida, A. T., V. Chaudhri, P. Giorgini, E. S. Yu (eds.): Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos. LNCS volume 5600, 2009. Springer. ISBN 978-3-642-02462-7.

ANEXO I - Manual da Ferramenta

Instalação

Em posse dos arquivos referentes a codificação do *Oryx* e seus respectivos *stencils* completos, são necessários realizar os seguintes passos:

- Instalação do software Tomcat, versão 6.0 ou superior.
- Instalação do *browser* Firefox, versão 3.0 ou superior.

Ao finalizar as instalações (se necessárias), copiar a pasta completa do *Oryx* para a pasta "*webapps*" do Tomcat. No Windows, por padrão de instalação esta pasta fica em "C:\Arquivos de Programas\Apache Software Foundation\Tomcat 6.0\webapps". Após copiar os arquivos, inicie o servidor Tomcat.

Em seguida abra o *browser* Firefox e digite o seguinte endereço: "http://localhost:8080/*Oryx*/iStarBPMN_editor.xhtml". O aplicativo deverá abrir corretamente. Em caso de falha, rever a instalação do Tomcat, já que ele é o principal responsável pela publicação do software na máquina local.

Passos básicos

Ao abrir o aplicativo, a seguinte tela será mostrada (Figura 28):

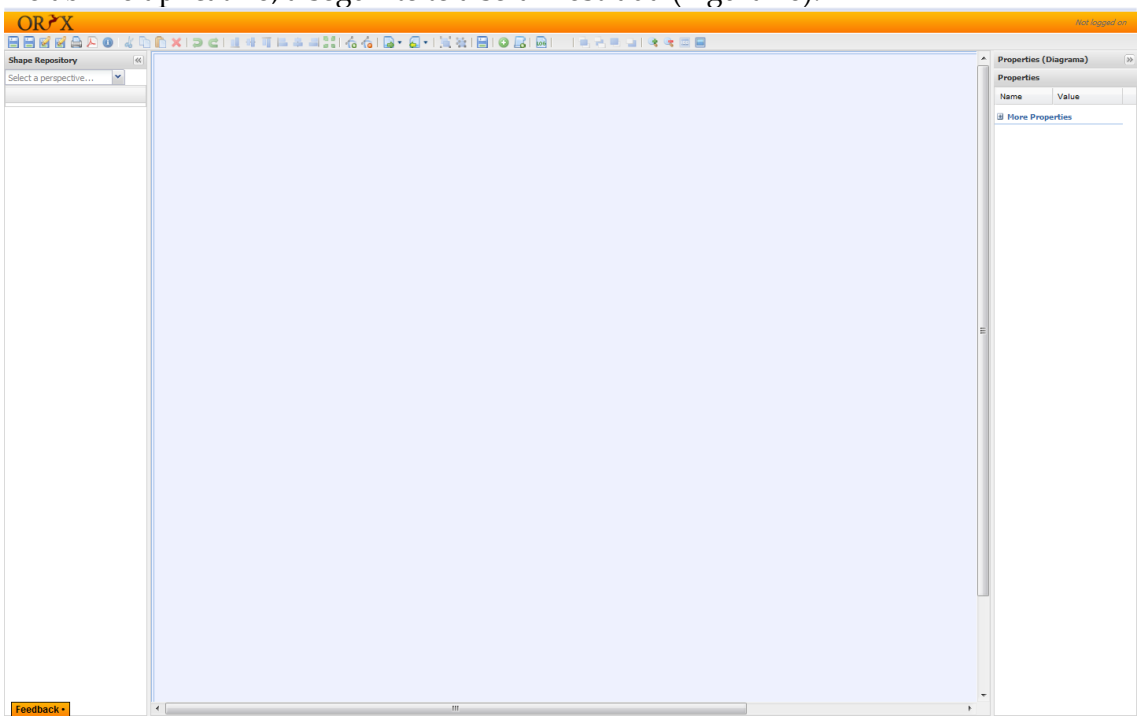


Figura 28 – Ambiente *Oryx*

Nesta tela principal nenhum "*stencil*" foi carregado, portanto nenhum elemento de modelagem é exibido. Para iniciar a modelagem é necessário primeiramente selecionar a perspectiva que contém as linguagens que serão utilizadas. Para isso, é necessário clicar na caixa "*Select a perspective...*", no canto superior direito do ambiente, conforme ilustra a Figura 29.

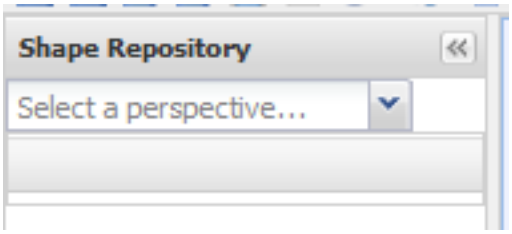


Figura 29 – Seleção de perspectiva

Após a seleção da perspectiva, a ferramenta disponibilizará os elementos de modelagem de acordo com as linguagens escolhidas. Caso deseje selecionar outra perspectiva, somente é necessário trocar a opção na mesma caixa de seleção, no entanto, recomenda-se o reinicio da ferramenta, por exemplo, atualizando o *browser* ou realizando outra chamada ao link, para limpar a memória da aplicação que é compartilhada com o *browser*. Em caso de situações inesperadas como erros ou mensagens na tela, recomenda-se a limpeza da memória *cache* do *browser* e reinicio da ferramenta.

Ambiente de modelagem

O ambiente de modelagem é composto por três painéis e uma tela central, para a edição dos modelos (Figura 28). No painel à esquerda encontram-se agrupados os elementos disponíveis pela ferramenta para modelagem (Figura 30). Clicando no "+" ao lado do nome do agrupamento (por exemplo, A*ctivities*), ele expande deixando mostrar os respectivos elementos (ver o agrupamento *gateways*).

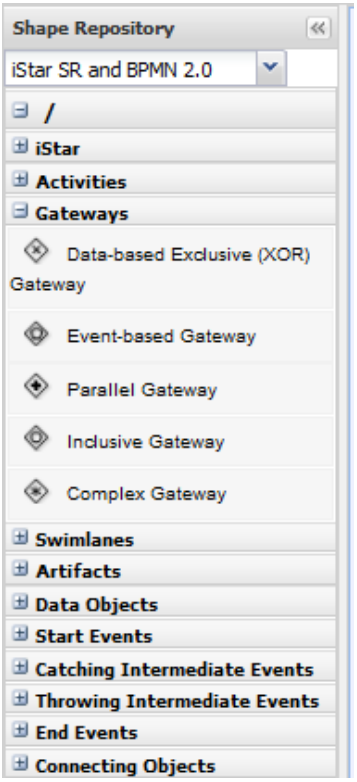


Figura 30 – Painel de elementos para modelagem

No painel à direita encontram-se as propriedades de um elemento do diagrama que se encontra selecionado (no caso de nenhum elemento selecionado, a ferramenta seleciona o diagrama). A partir destas propriedades é possível armazenar informações sobre o objeto bem como alterá-las. Para cada objeto selecionado, o painel de

propriedades irá apresentar um conjunto de opções inerente aquele objeto. A Figura 31 ilustra o conjunto de propriedades pertencentes a um objeto do tipo Tarefa (*Task)*, conforme visto entre parênteses na primeira linha do painel.

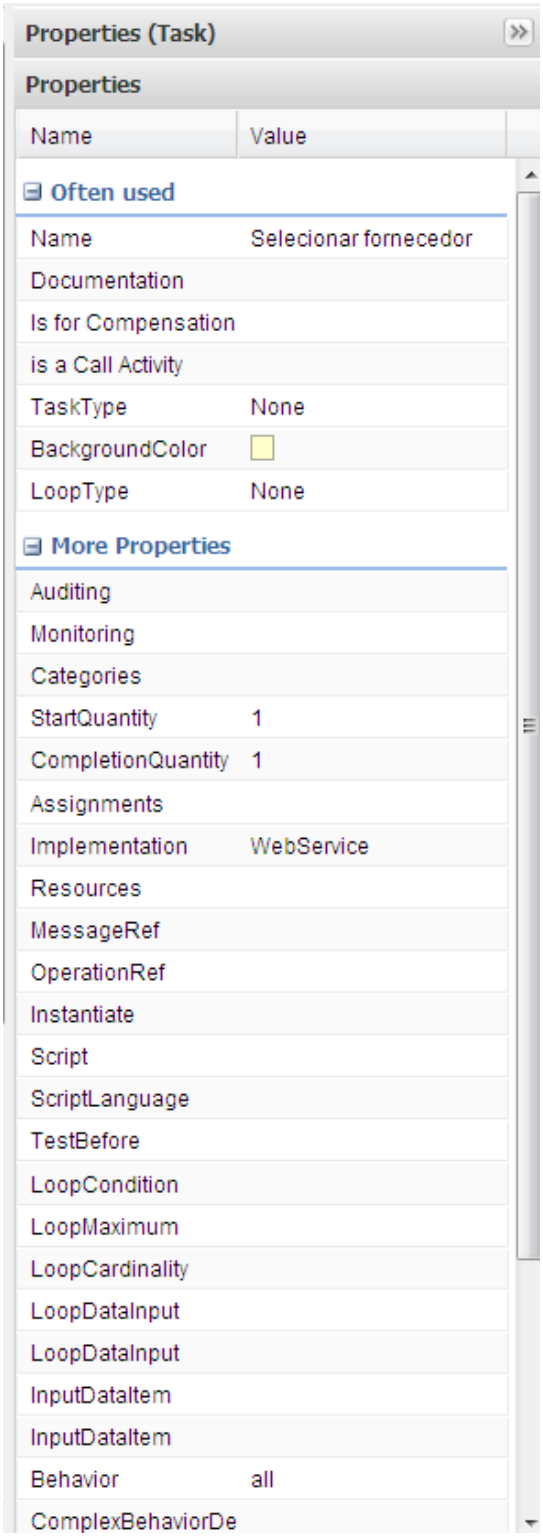


Figura 31 – Painel de propriedades do elemento

No painel superior central encontram-se diversas opções conforme descritos a seguir, respectivamente os botões a partir da esquerda para a direita:

- *Save e Save as...* – Salvam o modelo corrente. Na ferramenta Cross*Oryx*, essas funcionalidades não estão operacionais.

- *Print* – Imprime o modelo corrente. Ao clicar nesta opção, a ferramenta apresenta uma mensagem recomendando exportar primeiramente para PDF e posteriormente imprimir seu conteúdo, uma vez que podem ocorrer erros na impressão direta.

- *Export as PDF* – Exporta o modelo em um arquivo do tipo PDF

- *Info* – Apresenta informações de copyright e a versão da ferramenta

- *Cut, Copy e Paste* – Referente aos comandos recortar, copiar e colar, respectivamente.

- *Undo e Redo* – Referentes aos comandos desfazer e refazer, respectivamente.

- *Botton, Middle, Top, Left, Center, Right, Same Size* – Referentes a comandos que auto-organizam elementos no modelo alinhando respectivamente da seguinte forma: abaixo, ao meio, acima, à esquerda, ao centro, à direita, no mesmo tamanho.

- *Add a Docker to an edge, Delete a docker* – Referentes ao comando que insere ou apaga, respectivamente, pontos em qualquer tipos de setas para serem manipulados e criarem quebras/curvas nestes elementos (Figura 32).

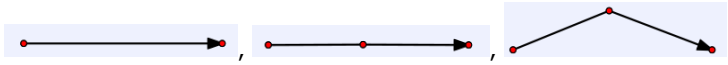


Figura 32 – Inserindo pontos de articulação em setas

- Exportar – A ferramenta oferece os seguintes padrões para exportar os modelos: ERDF, JSON, RDF, BPMN 2.0 DI XML, XPDL 2.2.

- Importar – A ferramenta oferece os seguintes padrões para importar modelos: ERDF, JSON, BPMN 2.0 DI XML e Visio BPMN.

- *Group all selected Shape, Delete the goup of all selected Shapes* – Referente ao comando que agrupa ou desagrupa, respectivamente, o conjunto de elementos selecionados. Por exemplo, ao agrupar 3 objetos, a partir deste momento, ao clicar em qualquer um dos 3 objetos, serão selecionados todos. Ao desagrupar, os objetos voltam a ser independentes ao serem selecionados.

- *Overlay test* – O modelo é pintado de vermelho para aplicação de testes.

- *Add a stencil set extension* – Oferece um conjunto de *templates* contendo os subconjuntos básicos de elementos para modelagens específicas ou simplificadas (por exemplo, "Subconjunto Básico BPMN 2.0" e "Coreografia BPMN 2.0"). Por padrão, os elementos de coreografia não constam no painel de elementos, sendo necessário adicionar sua extensão através desta opção.

- *Add additional Plugins dynamically* – Esta opção abre uma lista de *plugins* da ferramenta para seleção ou remoção.

- *Connect several nodes* – Esta funcionalidade conecta elementos ao clicá-los na ordem desejada.

- *Imports a model from a TBPM Photo* – Importa um modelo a partir de um arquivo .PNG ou .jpg, do tipo TBPM.

- *Generates a Process Log in MXML* – Cria um log do modelo corrente do tipo MXML e oferece para download.

- *Check Syntax* – A ferramenta analisa o modelo sintaticamente em busca de erros. Ao identificar um erro, marca no modelo e apresenta textualmente o motivo do problema.

- *Bring to front, Bring to back, Bring to Forward, Bring Backward* – Referentes aos comandos que traz um objeto para frente, para trás, para frente do próximo e para trás do anterior respectivamente, em relação aos outros objetos no modelo.

- *Zoom into the model, Zoom out the model, Zoom to the standard level e Zoom to fit the model size* – Referentes aos comandos que aumentam ou diminuem a visualização do modelo, alteram seu tamanho para o padrão ou alteram o seu tamanho de acordo com o tamanho da tela, respectivamente.

* As opções "*Fades tbpm foto in and out*" e "*Evaluate Partial Process Model*" não estão incluídas na lista de funcionalidades presentes no painel superior central da ferramenta porque não funcionaram.

O uso da ferramenta *Oryx* é simples e intuitivo. Sua interface é bem dividida acompanhando o padrão utilizado em diversas ferramentas do mercado. Sua capacidade de alterar os objetos através de seus parâmetros é baixa, no entanto a ferramenta possui seu código aberto, permitindo alterações diretamente no código fonte.

A modelagem na ferramenta *Oryx* oferece a funcionalidade "*Drag and drop*" permitindo inserir elementos no modelo ao arrastar o objeto desejado a partir dos painéis. Também é possível dar continuidade a modelagem a partir dos elementos já inseridos no modelo. Por exemplo, ao clicar em uma atividade, a ferramenta apresenta imediatamente os objetos que podem ser criados e interligados a partir dela. Com isso, não é necessário arrastar mais um elemento dos painéis para o modelo e posteriormente interligá-lo a outro elemento pré-posto. Basta somente clicar nas opções que surgem a partir do elemento clicado que a ferramenta já criará o novo objeto já interligado.

Outros pontos positivos de usabilidade oferecidos pela ferramenta é a capacidade de reorganizar as setas de fluxo de acordo com o movimento dos elementos que ela interliga e a capacidade de quebrá-las em curvas para adaptá-las à complexidade gráfica do modelo sem necessitar sobrepor-se a outros elementos. No entanto, não há opção de criar uma curva perfeita, ou seja, não é possível desenhar uma seta semiarredondada por exemplo. Apenas é possível "quebrá-la" em algum ponto para fazer a curva.