Compartilhamento de Modelos iStar 2.0 em Ferramentas de Modelagem Online - o Caso da piStar

Sharing iStar 2.0 Models on Online Modeling Tools – The piStar Case

Max Guenes[1]

João Pimetel[2]

1 Escola Politécnica de Pernambuco

2 Universidade Federal Rural de Pernambuco

E-mail: Autor Principal Max Guenesmaxguenes@gmail.com

Resumo

Para facilitar e ajudar na tomada de decisões durante a fase de engenharia de requisitos foi proposta a linguagem de modelos iStar 2.0 com uma visão social, onde os atores de um modelo se relacionam entre si, através de relações de dependência. Atualmente existe uma ferramenta gráfica para criação de tais modelos, a piStar. Nela, o modelo é criado em um navegador de internet e salvo na máquina do usuário. Um grande risco é de que este arquivo, salvo localmente na máquina, possa ser perdido ou até mesmo corrompido. Neste trabalho foi desenvolvida uma solução para ajudar no compartilhamento desses modelos através de um web service. O web service possibilita o armazenamento dos modelos criados e a disponibilização dos mesmos através de uma URL. Foi feita a implementação em Java de um web service que permitisse o piStar compartilhar os modelos criados.

Palavras-Chave:iStar; Engenharia de Requisitos; Ferramenta de Apoio; WebService.

Abstract

To make it easier and assist in the decision making during the requirements engineering phase, the iStar 2.0 modelling language was proposed, with a social vision where the actors of a model relate to each other through dependency links. Currently, there is a graphical tool available that was designed to create those models, the piStar tool. In that tool, the model is created in an inter-net browser and saved in the user computer. A major risk is that the file, stored in the machine, could be lost or even be corrupted. In this work it was developed a solution to help in sharing those models through a web service. The web service provides the saving and sharing of the created models though an URL. A web service that allows piStar to share the created models was implemented with Java.

Keywords: *iStar; Requirement Engineering; Supporting Tool; Webservice.*

http://dx.doi.org/10.25286/repa.v3i2.885

1 Introdução

Em muitas áreas de sistema de informação e engenharia, as ideias, as premissas e os objetivos são apresentados em modelos conceituais que refletem o conhecimento de um ambiente **[1]**. Partes desses modelos são transformados em dados e operações que podem ser representados por uma máquina.

Em **[2]** é explicado que um dos grandes desafios na área de requisitos é como identificar a real necessidade das pessoas e como representá-las em um modelo. Esses desafios precisam ser conquistados para que se possa desenvolver um sistema que realmente atenda às necessidades dos clientes.

Devido ao avanço e consolidação dos sistemas de informação na vida das pessoas, se tornou mais do que crucial a representação de suas necessidades e seus relacionamentos nos modelos ao se desenvolver tais sistemas **[3]**.

A linguagem i* é bastante aceita e usada pela comunidade acadêmica **[4]** da área de engenharia de requisitos e modelagem de negócios, além de outras áreas como telecomunicação, controle de tráfego aéreo, agricultura e entidades do governo, como visto em **[5]**.

Por falta de padronização dentro da própria comunidade, existe uma grande dificuldade na propagação de conhecimento sobre o i* e suas diversas variações. Visando mitigar esse problema, foi realizado um processo de padronização do i*, sendo criado a sua segunda versão, o iStar 2.0 **[4]**.

Foi desenvolvida uma ferramenta gráfica para criação de modelos iStar 2.0, a piStar, seguindo os requisitos definidos em **[6]**. Uma ferramenta leve, open source, que usa frameworks de Javascript, a piStar é capaz de realizar todas as operações através de um navegador de internet.

A ferramenta é capaz de salvar os modelos em arquivos de texto, onde é escrito todo seu conteúdo. Uma vez salvo, este arquivo está sujeito a ser perdido, modificado ou corrompido, visto que é mantido na máquina do usuário.

Este trabalho tem como objetivo a implementação de um módulo de webservices para guarda e recuperação de modelos iStar 2.0, facilitando o compartilhamento do modelo pela web.

A próxima seção, 2, explica o iStar 2.0 e introduz alguns conceitos relevantes para a implementação da solução proposta: a ferramenta piStar original, identificadores únicos, e técnicas de invasão. A seção 3 mostra como foi arquitetado e implementado o serviço. A seção 4 explica as validações realizadas contra tipos de ataques conhecidos e o ambiente de testes. Por último, a seção 5 apresenta a conclusão e trabalhos futuros.

2 Fundamentação Teórica

Nesta seção serão apresentados alguns fundamentos teóricos abordados durante este trabalho.

2.1 Framework *i** e iStar 2.0

O i* é uma linguagem de modelagem e framework de raciocínio **[4]**. Proposto nos anos noventa por Eric Yu **[7, 4]**, foi rapidamente aceito pela comunidade acadêmica e, pela sua natureza aberta, várias extensões foram propostas. Essas extensões provêm um refinamento de conceitos existentes ou definem novas ideias.

Acredita-se que essa flexibilidade, e a consequente falta de padronização, tenha impactado negativamente o crescimento e a propagação do conhecimento sobre o i* fora da comunidade acadêmica **[4]**, da seguinte forma:

Novatos – Achavam difícil aprender as peculiaridades da linguagem;

Educadores – Não havia uma base de conhecimento compartilhada para ensinar;

Praticantes – Não possuíam uma referência estabelecida de como usar o i* em seus projetos;

Fornecedor de tecnologia – Não sabia qual extensão doi* a ser implementada e quais técnicas deveram ser utilizadas em cima da extensão.

Para balancear a sua natureza aberta e as dificuldades de adoção foram realizados vários encontros, que culminaram na padronização de uma nova versão do i*. Para se diferenciar do seu predecessor, além de facilitar a indexação por motores de buscas, a linguagem foi nomeada de iStar 2.0.

2.1.1 Atores e seus tipos

Atores são entidades ativas, autônomas e que desejam alcançar um objetivo, exercendo seu know-how em colaboração com outros atores do modelo.
Existem duas especializações de atores no iStar 2.0:

Agente (Agent) –Ator com manifestação concreta, física, como por exemplo, uma organização ou uma pessoa;

Papel (Role) –Caracterização abstrata de um comportamento de um ator, dentro de um contexto ou domínio.

Para o iStar 2.0, sempre que não é relevante distinguir o tipo de ator, é utilizada a notação do ator genérico, sem especialização. Os tipos de atores são representados na Figura 1.
O ator possui uma fronteira, onde é definido seus elementos intencionais (objetivos, tarefas, etc.). Na Figura 2, o ator (*Researcher*) apresenta três elementos intencionais: "Good quality", "i* models created", e "Use piStar".

Figura 1: Exemplo de atores no modelo iStar 2.0
Fonte: Os autores.

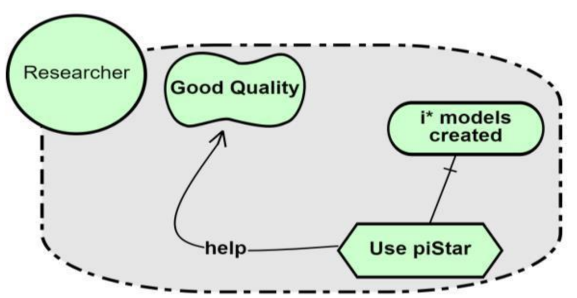

Figura 2: Exemplo de modelo do piStar de pesquisador que usa modelos piStar.
Fonte: Os autores.

2.1.2 Associações de atores

Atores dos modelos geralmente, possuem algum tipo de associação entre eles. Estas relações são descritas como associações entre atores.
Os dois tipos de associações de atores são:

é um (*is-a*) - Representa a generalização/especiali-zação de papéis com papéis (especialização)

participa-em (*participates-in*) - Representa qualquer tipo de associação entre dois atores; não existe restrição entre os tipos dos atores conectados nesta relação.

2.1.3 Elementos intencionais

Um elemento intencional é algo que um ator deseja. Esses elementos aparecem dentro das fronteiras dos atores, representando algo que o ator almeja.

Os seguintes tipos de elementos são definidos no iStar 2.0:

Objetivo (*Goal*) - Estado do mundo que um atordeseja alcançar, tendo um critério claro de conquista.

Qualidade (*Quality*) - Elemento que representa um atributo com diferentes níveis de satisfação que um ator deseja alcançar.

Tarefa (*Task*) - Representa as ações que um ator quer/pode exercer, geralmente com o propósito de alcançar um objetivo.

Recurso (*Resource*) - Entidade física ou informação necessária para que o ator execute uma tarefa.

2.1.4 Relação de elementos intencionais

Existem quatro tipos de relações entre elementos intencionais: Refinamento, Requisitado por, Contribuição, e Qualificação.

Refinamento (*Refinement*) - Associação feita entre objetivos e tarefas do ator, para demonstrar

http://dx.doi.org/10.25286/repa.v3i2.885

hierarquia de elementos filhos para um pai. É dividido em dois tipos:

– E (*AND*) - Para atender o pai, todos os filhos devem ser atendidos.

– OU inclusivo (*Inclusive OR*) - Para atender o pai, pelo menos um dos filhos deve ser atendido.

Requisitado por (*Needed By*) - Relação entre uma tarefa e um recurso, indicando o que é necessário para um ator executar tal tarefa.

Contribuição (*Contribution*) - Representam o efeito de um elemento intencional em qualidades. São cruciais para a análise durante o processo de tomada de decisão entre alternativas de objetivos e tarefas. Associações de contribuições ajudam a acumular evidências para qualidades. São os tipos de contribuições:

– Faz (*Make*) - A fonte provê evidências positivas fortes para satisfazer a qualidade destino;

– Ajuda (*Help*) - A fonte provê evidências positivas fracas para satisfazer a qualidade destino;

– Machuca (*Hurt*) - A fonte provê evidências negativas fracas para satisfazer a qualidade destino;

– Quebra (*Break*) - A fonte provê evidências negativas fortes para satisfazer a qualidade destino;

Qualificação (*Qualification*) –Esta relação refere uma qualidade ao seu sujeito (tarefa, objetivo ou recurso). Expressa uma qualidade desejada durante a execução de uma tarefa, conquista de um objetivo ou provisão de um recurso.

2.1.5 Dependências sociais

Dependências são as representações das relações sociais nos modelos iStar 2.0. A dependência poder ser definida por 5 argumentos:

***Depender* –** O ator que depende de algo a ser provido (*Dependum*);

***DependerElmt* –** O elemento intencional, dentro da fronteira do ator Depender, de onde surge a dependência, explicando sua existência

***Dependum* –** Elemento intencional que representa o objeto de dependência;

***Dependee* –** O ator que provê o *Dependum*

***DependeeElmt* –** O elemento intencional que explica como o *Dependee* provê o *Dependum*.

2.2 Ferramenta piStar

A piStar é uma ferramenta para a criação de modelos iStar 2.0. Utiliza frameworks de Javascript para exibir, editar, salvar e carregar modelos. A Figura 2 mostra um exemplo de modelo construído utilizando a ferramenta piStar.

No exemplo da Figura 2, o ator Pesquisador precisa alcançar um objetivo, criar modelos i*. Para alcançar este objetivo, ele usa o piStar e consequentemente, alcançando boa qualidade.

Os modelos salvos são disponibilizados em um arquivo de texto (extensão .txt”) contendo o conteúdo completo do modelo no formato JSON (*JavaScriptObject Notation***[8]**). Ao carregar um arquivo“.txt” de modelo piStar, o mesmo é exibido novamente na ferramenta. O código-fonte dessa ferramenta está disponível em https://github.com/jhcp/pistar.

Na i* wiki **[20]** são listadas outras ferramentas com a proposta parecida com a do piStar, como por exemplo o Leaf 2.0 e i*-REST (ferramenta web para criação de modelos iStar 2.0).

2.3 UUID

Os UUID, ou *UniversallyUniqueIdentifiers*, são sequências de 128 bits utilizadas para identificação de informações **[9]**. Eles foram adotados neste trabalho para permitir a individualização dos modelos salvos através do *webservice* desenvolvido.

Seu formato possui 16 octetos, 32 dígitos hexadecimais, separados em 5 grupos, seguindo o padrão de 8-4-4-4-12 caracteres, tendo em seu total, 36 caracteres. Por ex: "4fadf6cf-89ef-46d5-b743-66ce07e0eb69".

A implementação em Java (versão 8) de UUID adotada neste trabalho está documentada em **[10]**. Tal implementação utiliza a versão 4 do UUID, onde para alcançar uma chance de 50% de

probabilidade de colisão de nomes, é preciso gerar em torno de 2,71 x 10^{18} de UUIDS aleatórios.

2.4 Técnicas de invasão

Com o crescimento do uso da internet as aplicações web se tornaram comum na vida das pessoas (Internet banking, *e-mail*, compras *online*). Por estarem expostas em um endereço público, tais aplicações estão sujeitas a ataques **[11]**. Para que o *webservice* aqui proposto possa ser disponibilizado ao público, foi preciso estudar as principais técnicas de invasão que podem ser utilizadas contra esse tipo de sistemas.

Em **[11]** é visto que as principais falhas de segurança em sistemas Web são por conta de XSS (*Crosssite Scripting*) e SQL *Injection* (Injeção de SQL).

O XSS é o tipo de invasão que ocorre quando um código malicioso é injetado em um site confiável**[12]**. Eles acontecem pela falta de limpeza da entrada de dados de usuários, ou pela limpeza incorreta, levando o site estar exposta a tal vulnerabilidade.

SQL *injection* ou injeção de SQL é uma das vulnerabilidades mais perigosas em aplicações web **[13]**. Aplicações vulneráveis a SQL *injection* permitem que a base de dados esteja completamente acessível para um invasor. Novamente, é resultado da falha ou ausência de validação de entradas de dados do usuário.

3 O piStar *backend*

Este trabalho tem como objetivo fornecer a funcionalidade de compartilhamento de modelos para a ferramenta de modelagem de requisitos piStar. Para o compartilhamento de modelos é necessário prover duas funções básicas: salvar o modelo e recuperá-lo. Para implementar essa funcionalidade foi decidido criar serviços REST (*Re-presentational State Transfer* **[14]**) para armazenamento e recuperação de modelos iStar 2.0.

Foi escolhida a implementação REST do serviço pela facilidade de integração com a interface gráfica piStar, por usar Javascript, além do formato de dado já utilizado, JSON, ser facilmente transferido por este tipo de serviço. Uma das bibliotecas já utilizadas no piStar, o JQuery, possui uma API (*Application Programming Interface*) de chamadas assíncronas para chamadas a serviços REST.

O *webservice* deve prover uma interface REST adequada, utilizando formatação JSON. Cada modelo salvo deve ser identificado através de um identificador único, utilizado para a recuperação do mesmo, quando consultado.

Foi desenvolvida a aplicação *backend*, implementada em Java, assim como algumas modificações na ferramenta piStar para receber esta nova funcionalidade, além de deixa-la retrocompatível com o modo *offline* (modo original do piStar).

3.1 Arquitetura

Foi utilizado o padrão de projeto MVC (*Model, View, Controller***[15]**) para desenvolvimento do serviço. A Figura 3 ilustra os módulos do serviço. Os modelos são definidos dentro do módulo de persistência. A visualização contém somente a interface de *webservices* REST.

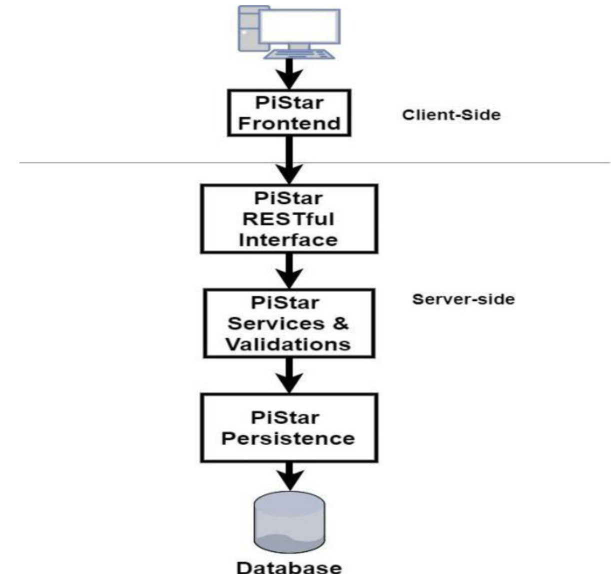

Figura 3: Diagrama de arquitetura do projeto
Fonte:Os autores.

O projeto foi dividido em 3 módulos principais: o módulo de *web services* (piStar *RESTful* Interface), o módulo de regras de negócios (*piStar Services & Validations*), e o módulo de persistência (*piStar Persistence*). O projeto do

http://dx.doi.org/10.25286/repa.v3i2.885

cliente piStar (*piStar Frontend*) foi mantido separado do projeto *backend*.

O módulo de *web services* é responsável por expor uma interface RESTful com os métodos do piStar *backend*, além de receber os parâmetros necessários para o módulo de regras de negócios.

O módulo de regras de negócios é responsável por fazer as validações dos dados e por tratar os parâmetros recebidos e retornos enviados.

O módulo de persistência está responsável por armazenar e recuperar os dados no banco de dados.

A implementação foi feita em Java, utilizando o *frameworkSpring* 4 **[16]**. O *Spring* é um *frameworkopen source* que oferece um conjunto diverso de bibliotecas para desenvolvimento de aplicações Java. São utilizados no projeto os módulos de MVC, persistência (JPA) e injeção de dependências.

O Maven foi utilizado como ferramenta de controle de dependências e construção do projeto Java. Maven é uma ferramenta *open-source*, mantido pela Apache decontrole de dependências, plug-ins e hierarquia de projetos. Uma vez que um projeto possui projeto parente, acaba obtendo suas dependências, plug-ins e propriedades.

O projeto Maven foi montado em cima do projeto *parent* do *Spring Boot*, "*spring-boot-starter-parent*". A Tabela 1 mostra as dependências utilizadas no projeto.

Nome	Versão
spring-boot-starter-parent	1.5.4.RELEASE
spring-boot-starter-data-jpa	1.5.4.RELEASE
spring-boot-starter-web	1.5.4.RELEASE
spring-boot-starter-test	1.5.4.RELEASE
mybatis-spring-boot-starter	1.3.0
HikariCP	2.6.0
h2	1.4.196
fyway-core	4.2.0
validation-api	1.1.0.Final

Quadro 1: Tabela de dependências Java.
Fonte: Os autores.

3.2 Serviços

Nesta seção são descritos os serviços disponíveis, os método HTTP e os conteúdos das requisições, utilizando o padrão RESTful.

3.2.1 Salvar modelo

Método POST para a URL **http://<host>/model/save** o campo "*host*"é o domínioonde está instalada a aplicação. O corpo da requisição deve conter o conteúdo JSON, seguindo modelo da Figura 4.

O retorno deste método é uma *String* com o UUID do modelo salvo, que poderá ser utilizada para recuperar o mesmo.

O UUID do modelo salvo é baseado no seu conteúdo JSON. Após as validações do modelo, é utilizado o método *PistarModelServiceImpl.getUniqueHash* (*String content*) para gerar o seu identificador. Para modelos com o mesmo conteúdo, ignorando sua data de inclusão, é retornado o mesmo UUID.

3.2.2 Recuperar modelo

Método GET para a URL **http://<host>/model/<hash>**, o campo "*host*" é o *host* de onde está instalado a aplicação e "*hash*" é o identificador do modelo a ser recuperado. O retorno do método é o conteúdo JSON do modelo salvo, no identificador passado, seguindo o modelo da Figura 4.


```
{
  "actors": [
    {
      "id": "940a8359-070d-42d7-9780-a9d7b5d70399",
      "text": "Researcher",
      "type": "istar.Actor",
      "x": 119,
      "y": 49,
      "nodes": [
        {
          "id": "afdf8da0-fc98-4952-ae8e-244e2b11164c",
          "text": "i* models created",
          "type": "istar.Goal",
          "x": 320,
          "y": 88
        },
        {
          "id": "fed0e59a-093c-4294-a21e-27f285cf6d91",
          "text": "Use piStar",
          "type": "istar.Task",
          "x": 291,
          "y": 170
        },
        {
          "id": "817b65fd-9693-443a-9431-a9bf1f73d4e0",
          "text": "Good Quality",
          "type": "istar.Quality",
          "x": 190,
          "y": 65
        }
      ]
    }
  ],
  "dependencies": [],
  "links": [
    {
      "id": "ff72ca31-ae66-43fd-a6b6-4add07ba12fe",
      "type": "istar.AndRefinementLink",
      "source": "fed0e59a-093c-4294-a21e-27f285cf6d91",
      "target": "afdf8da0-fc98-4952-ae8e-244e2b11164c"
    },
    {
      "id": "771654a0-c79d-4430-93fd-6299cf6787db",
      "type": "istar.ContributionLink",
      "source": "fed0e59a-093c-4294-a21e-27f285cf6d91",
      "target": "817b65fd-9693-443a-9431-a9bf1f73d4e0",
      "label": "help"
    }
  ],
  "tool": "pistar.1.0.0",
  "istar": "2.0",
  "saveDate": "Tue, 18 Jul 2017 09:35:58 GMT",
  "diagram": {
    "width": 2232,
    "height": 1172
  }
}
```


Figura 4: Exemplo JSON piStar.
Fonte: Os autores.

3.2.3 Versão da aplicação

Método GET para a URL http://<host>/version o campo "*host*" é o host de onde está instalado a aplicação. Não recebe nenhum tipo de parâmetro.

Retorna uma String contendo a versão da aplicação instalada no host. Este método é utilizado para verificar se a aplicação está no ar.

3.3 Modificações na interface gráfica

Algumas modificações foram feitas na interface gráfica piStar, para adicionar a funcionalidade de salvar e de carregar um modelo salvo. Foi adicionado um botão para compartilhar o modelo, que envia o conteúdo JSON do modelo para que o servidor *backend* salve seu conteúdo.

As alterações no código foram realizadas de forma que fosse compatível com sua versão original.

Inicialmente, o piStar verifica se o servidor está respondendo com sucesso, checando o método de recuperar versão. Caso o método responda com um número de versão do piStar *Backend*, é aceito que existe o suporte para compartilhamento de modelos e o botão "*Share model*" aparece na tela, conforme ilustrado na Figura 5.

Figura 5: Botão de compartilhar modelos.
Fonte: Os autores.

Após realizar as modificações necessárias no modelo, o mesmo pode ser salvo através do botão de compartilhar. Ao enviar o modelo ao servidor, é retornado o identificador do mesmo, usado para consulta. A interface gráfica exibe o link para ser compartilhado, ilustrado na Figura 6.

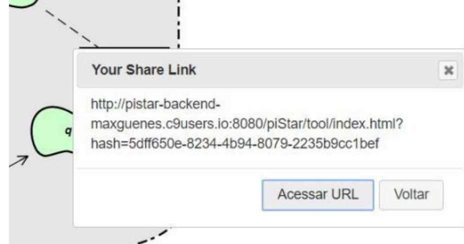

Figura 6: Link de consulta de modelo salvo.
Fonte: Os autores.

O link de compartilhamento de modelos segue o formato:

http://<host>?hash=<hash>

O valor "*host*" é o domínio onde a aplicação *frontend* está instalada e "*hash*" é o valor do identificador do modelo salvo no *backend.*

Ao acessar um link com um parâmetro "hash", é consultado no backend se existe um modelo salvo com este identificador. Caso o servidor retorne um modelo, o conteúdo deste é carregado na biblioteca piStar e exibido no editor.

Foi adicionado na biblioteca principal do piStar, uma sobrecarga da função "*loadModel*". Era recebido o valor texto do conteúdo JSON, ao carregar modelo (utilizado quando era enviado um arquivo de texto para a interface). O método foi alterado para fazer a transformação do conteúdo em um objeto JSON e enviar para a sobrecarga que recebe o objeto JSON para exibir na interface gráfica. Esta mesma sobrecarga é utilizada quando é consultado um modelo existente no *backend*, onde o conteúdo retornado (formato JSON) é enviado direto para o método.

3.4 Esquema do banco de dados

Durante a primeira execução da aplicação, a biblioteca *Flyway* fica responsável por criar o esquema da tabela do banco de dados. *Flyway* é uma biblioteca open source utilizada para garantir a integridade da estrutura de tabelas do banco de dados. É configurada uma sequência de scripts SQL que são executados no momento em que a biblioteca é chamada, em momento de execução. Ele verifica a existência da tabela e caso não esteja criada, faz a criação da tabela, Figura 7.

O esquema da tabela *pistar_model* possui o HASH, como um UUID único, *JSON_CONTENT* como o conteúdo, *REMOTE_ADDR* para o endereço remoto do responsável por criar o modelo e *INSERTED_DATE* para a data de criação do modelo.

A definição das classes de persistências, *Mappers,* usando o *Mybatis* v3.4.4, **[17]**, é a seguinte:

```
CREATE TABLE 'pistar_model' (
  'HASH' UUID NOT NULL,
  'JSON_CONTENT' LONGTEXT NOT NULL,
  'REMOTE_ADDR' LONGTEXT NOT NULL,
  'INSERTED_DATE' TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY ('hash')
) DEFAULT CHARSET=utf8;
```

Figura 7: SQL de criação de tabela de modelos.
Fonte: Os autores.

http://dx.doi.org/10.25286/repa.v3i2.885

3.5 Limitações do projeto

Pela forma na qual o projeto foi implementado, algumas limitações foram identificadas, que podem ser ajustadas em trabalhos futuros.

3.5.1 Modificações no modelo salvo

Por não possuir um controle de acesso ou autenticação, o serviço assume qualquer modelo enviado, seja o inicial ou uma modificação de um já existente, como sendo um modelo novo. Ou seja, todo modelo enviado para o serviço é salvo com um identificador novo, mantendo o registro do modelo original salvo no banco, no caso de uma modificação. Ao realizar várias operações de edição, o modelo antigo nunca é removido, gerando registros não mais consultados na base de dados.

3.5.2 Ausência de criptografia

Não foi implementada a camada de criptografia ou segurança das requisições e do modelo salvo no banco de dados. Todos os conteúdos são enviados e recebidos através de protocolo HTTP (*Hipertext Transfer Protocol*)**[24]**. Quando não é utilizado o protocolo seguro, como por exemplo HTTPS (*Hipertext Transfer Protocol Secure*) os dados transferidos pela rede estão sujeitos a serem interceptados no caminho e alterado seu conteúdo.

4 Validação

Para garantir a integridade e funcionamento do serviço, são feitas algumas validações das entradas de usuários. Além disso, nesta seção também são descritas algumas etapas de testes visando garantir a qualidade do *web service* desenvolvido.

4.1 Validação de modelos e entradas

Todas as entradas de usuários salvas no servidor são validadas para garantir a consistência dos modelos salvos, além de evitar tipos de ataques de sistemas conhecidos, como por exemplo XSS e SQL *Injection*.

Foi utilizada a biblioteca de validação *javax.validation:validation-api* para fazer uma validação inicial do modelo, retornando um erro ao envio de um modelo inválido. Após essa validação inicial é realizada outra varredura no modelo completo, garantindo a integridade do modelo, como por exemplo, validações de tipos válidos e *target* e *sources* existentes no modelo.

Foi criado uma interface Java *PiStarValidObject*, Figura 8, para fazer as validações do modelo e suas referências. Cada objeto do modelo (Atores, nós, dependências, links, diagrama e o próprio modelo) implementam esta interface em sua hierarquia de classe. Existe um método, chamado *checkValidObject* que é chamado para validar o objeto.

```
package com.pistar.jpa.model;


public interface PiStarValidObject {
    void checkValidObject();
}
```

Figura 8: Interface *PiStarValidObject.*
Fonte: Os autores.

Para o caso de elementos dos modelos que são identificáveis, através de um ID e tipo, eles estendem a classe abstrata *IdentifiedObject*, Figura 9, que implementa *PistarValidObject*. Nesta classe, é verificado se o ID e o tipo não são nulos e que o tipo é um tipo válido para objeto em questão. Para o uso dessa classe é preciso implementar o método abstrato *getValidTypes*, onde é retornado o conjunto de tipos válidos para o objeto.

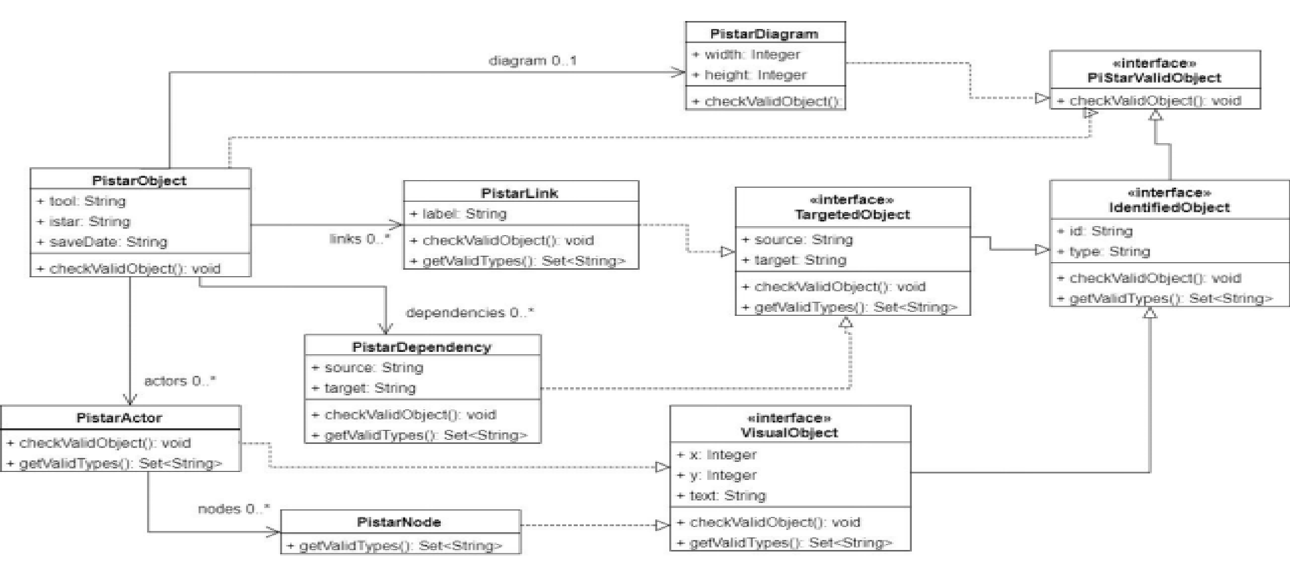

Figura 9: Classe abstrata *Identified Object* .

Para links e dependências é necessário verificar se os campos *target* e source existem. Para isso, todos os objetos válidos são guardados e durante a validação de links e dependências, é verificado se as referências existem. A Figura 10 ilustra o diagrama de classes dos objetos modelos do *piStar backend*.

```
public abstract class IdentifiedObject
    implements PiStarValidObject {

   @NotNull(message = "Empty ID value")
   public String id;

   @NotNull(message = "Empty Type value")
   public String type;

   @Override
   public void checkValidObject() {
       Assert.notNull(id, "Empty ID value");
       Assert.notNull(type, "Empty Type value");
       Assert.isTrue(getValidTypes().
           contains(type),
           "Invalid Node Type "+type+
           " in valid types "+getValidTypes());
   }

   abstract Set<String> getValidTypes();


   ...
}
```

Figura 10: Diagrama de classes de modelos piStar backend.

Para links e dependências é necessário verificar se os campos *target* e *source* existem. Para isso, todos os objetos válidos são guardados e durante a validação de links e dependências, é verificado se as referências existem. A Figura 9 ilustra o diagrama de classes dos objetos modelos do piStar backend.

4.2 Filtro XSS

O controle de conteúdo recebido no *backend*, é feito através do *framework Spring* MVC. É usada a implementação da interface Java *javax.servlet.Filter* para filtrar o conteúdo de uma requisição. Foi criada a classe com.*pistar.filter.XSSFilter* para tratar os dados do cabeçalho e parâmetros, utilizando a classe *wrapper XSSRequestWrapper* para envolver uma requisição maliciosa **[18]**.

Os valores do cabeçalho e parâmetros passam, caso existam, por um método *stripXSS*, Figura 11, que verifica se possui conteúdo de HTML e o remove do texto. Após a limpeza do conteúdo, o texto utilizado no sistema está livre de XSS.

```
public static String stripXSS(String value) {
    if (value != null) {
        value = value.trim().replaceAll("\n",
            "");
        value = value.replaceAll("\\<.*?>","");
    }
    return value;
}
```

Figura 11:Método stripXSS.

http://dx.doi.org/10.25286/repa.v3i2.885

Foi criado o teste unitário do método, que verifica conteúdo HTML no texto, enviando para o método alguns exemplos comuns de XSS. A Figura 12 demonstra parte do resultado retirado do teste unitário do projeto, que checa a existência de conteúdo HTML.

4.3 Filtro SQL Injection

A biblioteca de persistência, *MyBatis* v3.4.4, possui um mecanismo de tradução de um *template* XML para criação de um comando SQL. A própria biblioteca detém um controle de conteúdo para evitar SQL *Injection***[17]**.

```
Antes - Um texto qualquer, sem conteudo
    malicioso
Depois - Um texto qualquer, sem conteudo
    malicioso

--------------------------------
Antes - <SCRIPT>alert("Cookie")
</SCRIPT>
Depois - alert("Cookie")
--------------------------------
Antes - <body onload=alert('test1')>
Depois -
--------------------------------
Antes - <b onmouseover=alert('Wufff!')>click
    me!</b>
Depois - click me!
--------------------------------
Antes - <img src="http://url.to/xss"
onerror=alert(document.cookie);>
Depois -
--------------------------------
Antes - <IMG SRC=j&#X41vascript:alert('test2')>

Depois -
```

Figura 12: Resultados do teste unitário.

Ao usar a sintaxe correta, o MyBatis faz o controle do conteúdo enviado como parâmetro deixando-o de forma segura contra tipos de invasão.

É indicado pela própria documentação da biblioteca, utilizar a sintaxe de "#parametro", para evitar SQL *Injection*. Desta forma, o conteúdo é transformado em um valor texto, sendo tratado dentro do comando SQL somente como um valor, ao contrário de outras abordagens que acabam concatenando o valor enviado no comando final.

Foi criado o mapeamento do método de inserção, Figura 13, utilizando os padrões corretos, definidos pela biblioteca.

```
<insert id="insertModel">
    INSERT
    INTO
        PISTAR_MODEL("HASH", "JSON_CONTENT", "REMOTE_ADDR")
    VALUES
        (#{hash}, #{jsonContent}, #{remoteAddr})
</insert>
```

Figura 13: Mapeamento de inserção de modelo.
Fonte:Os autores.

4.4 Ambiente de testes

Os testes foram realizados, utilizando o ambiente remoto Cloud9 (https:/c9.io/) usando um ambiente com as configurações descritas na Tabela 2.

O código foi desenvolvido localmente, enviado para o Github, e implantado na máquina remota Cloud9. Era feita a instalação do Maven e suas dependências.

SO	Ubuntu 14.04.3 LTS
RAM	X86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4

Quadro 2: Ambiente Cloud9.
Fonte: Os autores.

A aplicação era executada em uma JDK 8 e o build era executado com o Maven 2.

A URL dos projetos no *Github* estão disponíveisem:

piStar *frontend* -
https://github.com/maxguenes/pistar

piStar *backend* -
https://github.com/maxguenes/pistar-backend

5 Conclusão

Foi proposto neste trabalho a implementação um *web service* para gestão dos modelos salvos na ferramenta piStar. Foram utilizados boas práticas de projeto Java, além de *frameworksopen source* bem documentados para prover este serviço.

Ademais, o cliente da ferramenta foi estendido de forma a comportar a funcionalidade de compartilhar e carregar modelos através de uma URL.

As validações realizadas no serviço buscam garantir consistência do que foi salvo, além de limpeza das entradas de dados de usuários, evitando ataques de XSS e SQL *Injection*, que são os tipos de ataques mais comuns **[11]** e**[23]**.

Para trabalhos futuros, a implementação dos serviços *backend* poderia ser estendida adicionando-se uma camada de autenticação, onde os modelos seriam atrelados a um perfil de usuário.

Um dos pontos não abordado no projeto foi o teste de desempenho – em particular, testes de cargas, que visariam garantir que o serviço consegue escalar para grandes quantidades de usuários.

Outra possibilidade de trabalho futuro seria a análise de potenciais conflitos entre identificadores UUID, podendo ser modificado para outra implementação de identificadores únicos.

Referências

[1] S. Y. Eric. Social modeling and i*. In: **Conceptual Modeling**:Foundations and Applications, p. 99–121, Springer, 2009.

[2] ERIC, S. **Social modeling for requirements engineering**. Mit Press, 2011.

[3] MYLOPOULOS, J. Information modeling in the time of the revolution. **Information systems,** v. 23, n. 3-4, p. 127–155, 1998.

[4] DALPIAZ, F.; FRANCH, X.; HORKOFF,J**. Istar 2.0 language guide**,” arXiv preprint: 1605.07767, 2016.

[5] LIMA, P. et al. Scalability of istar: a systematic mapping study. In: Workshop em Engenharia de Requisito (WER), 19., 2016, Quito. **Anais**…Quito: WER, 2016.

[6] PIMENTEL,J.; VILELA, J.; CASTRO, J. Web tool for goal modelling and statechart derivation, In: **Requirements Engineering Conference** (RE), 2015 IEEE 23rd International, pp. 292– 293, IEEE, 2015.

[7] YU, E. **Modelling strategic relationships for process reengineering**. Social Modeling for Requirements Engineering, vol. 11, p. 2011,2011

[8] BRAY, T T**he javascript object notation (json) data interchange format**. 2014.

[9] LEACH, P. J.; MEALLING, M.; SALZ, R. A univesally unique identifier (uuid) urn namespace. 2005.

[10] Java 8 class uuid documentation. Disponível em: http://docs.oracle.com/javase/8/docs/a-pi/java/util/UUID.html Acessado: 25 jul. 2017.

[11] BARBOSA, Eber Donizeti; CASTRO, R. d. O. Desenvolvimento de Software Seguro: Conhecendo e Prevenindo Ataques Sql Injection e Crosssite Scripting (XSS). **Revista TIS**, v. 4, 2016.

[12] HYDARA, I. et al. Current state of research on cross-site scripting (xss)–a systematic literature review. **Information and Software Technology**, v. 58, pp. 170–186, 2015.

[13] HALFOND, W.G.; VIEGAS, J.; ORSO, A. A classification of sqlinjection attacks and countermeasures. In: Proceedings of the IEEE

 http://dx.doi.org/10.25286/repa.v3i2.885

International Symposium on Secure Software Engineering, 17., 2006, Raleigh. **Anais**… Raleigh: IEEE, 2006. P. 13-15

[14] R. L. Costello et al., Rest (representational state transfer) **last updated Jun**, v. 26, 2002.

[15] YENER, M.; THEEDOM, A. **Professional Java EE design patterns**. Jhon Wiley &Sons, 2014.

[16] Spring Framework. Disponível em: http://projects.spring.io/spring-framework/ Acessado em: 25 jul. 2017.

[17] How to fix sql injection using mybatis. Disponível em: https://software-security.sans.org/developer-how-to/fix-sql-injection-in-java-mybatis Acessado em: 21 jul. 2017.

[18] Spring security cross-site scripting. Disponível em: https://defensivecode.wordpress.com/2013/09/03/spring- security-xss/ Acessado: 22 jul. 2017.

[19] HORKOFF, T. et. al. Taking goal models downstream: a systematic roadmap. In: Research Challenges in Information Science (RCIS), IEEE Eighth International Conference on, 8., 2014, Marrocos. **Anais**… Marrocos, 2014. p. 1–12.

[20] "istar wiki." http://istarwiki.org/. Acessado: 23Jul. 2017.

[21] JAQUEIRA, M. Lucena; F. M. Alencar, J. CASTRO; ARANHA, E "Using i* models to enrich user stories" iStar, v. 13, pp. 55–60, 2013.

[22] WEN, X.; JIANHUA,G. Research of web application framework based on spring mvc and mybatis. **Microcomputer Applications**, v. 7, p. 1– 4, 2012.

[23]J. FONSECA, M. Vieira; MADEIRA, H. Testing and comparing web vulnerability scanning tools for sql injection and xss attacks. In**:** Dependa- ble Computing, Pacific Rim International Symposium on, 13., 2007, Meulborne. **Anais**…Meulborne: IEEE, 2007. p. 365–372.

[24] FIELDING, Roy, et al. **Hypertext transfer protocol-- HTTP/1.1. No. RFC 2616**. 1999.

[25] RESCORLA, Eric; SCHIFFMAN, A. **The secure hyper-text transfer protocol**. 1999.