Webservice para integração de dados entre a ferramenta de simulação MASPN e seu portal Web

Éderson Guterres Costa[1], **Alexandre de O. Zamberlan**[1 2]

[1]Curso de Sistemas de Informação – Universidade Franciscana (UFN)
Santa Maria – RS

[2]Laboratório de Práticas – Universidade Franciscana (UFN)
Santa Maria – RS

{edersonguterres,alexz}@ufn.edu.br

Resumo. *Este trabalho faz parte das pesquisas realizadas no Multi-agent System for Polymeric Nanoparticles (MASPN), que é composto por uma ferramenta de simulação e um website. Dessa forma, este trabalho tem o foco de projetar e implementar uma integração dos dois ambientes, um feito para Desktop (Java) e outro para Internet (Python, Django e MySQL). O portal Web gerencia fármacos, pesquisadores, polímeros, experimentos com nanopartículas poliméricas, já na ferramenta MASPN, a partir desses dados cadastrados no portal, possibilita a simulação do comportamento das nanopartículas poliméricas. Para a construção da integração, webservice, foram utilizadas as tecnologias JSON, mysqldump, mysql2sqlite, dumpdata, Python, Django e Java, todas contextualizadas em boas práticas do framework REST.*

Abstract. *This work is part of the research carried out in the Multi-agent System for Polymeric Nanoparticles (MASPN), which consists of a simulation tool and a website. Thus, this work aims to design and implement the integration of the two environments, one built for Desktop (Java) and the other for Internet (Python, Django and MySQL). The website manages drugs, researchers, polymers, experiments with polymeric nanoparticles, in the MASPN tool (based on the data registered in the portal), it allows the simulation of the behavior of polymeric nanoparticles. For the design of the integration were used JSON, mysqldump, mysql2sqlite, dumpdata, Python, Django and Java, all obeying the best practices of the REST framework.*

1. Introdução

O presente trabalho está inserido na pesquisa referente à ferramenta de simulação MASPN [Zamberlan et al. 2016], [Zamberlan 2018], [Zamberlan et al. 2019], [Zamberlan et al. 2020] e seu portal Web [Pereira et al. 2018].

Os ambientes de simulação e portal, até a realização deste trabalho, não estavam interligados. Toda a simulação realizada era digitada na ferramenta *Desktop*, a partir dos dados cadastrados e consultados no portal MASPN. Portanto, integrar os ambientes foi fundamental para a continuidade da pesquisa da ferramenta MASPN.

Decidiu-se usar um *Middleware* conhecido como *webservice*, que é uma solução utilizada para integrar sistemas e para a comunicação entre aplicações diferentes ou heterogêneas (nível de sistema operacional, linguagens de programação, banco de dados, por

exemplo). Dessa forma, é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis [Coulouris et al. 2003] e [Tanenbaum 1994].

De acordo com [W3School 2020b], SOAP é um protocolo de comunicação entre aplicações heterogêneas, com um formato baseado em XML para enviar e receber mensagens, sob uma plataforma independente e recomendado pela W3C. Já, segundo [Codecademy 2020], REST é um estilo de arquitetura cliente-servidor que fornece padrões de comunicação entre computadores conectados na Web. Contudo, tanto a implementação do cliente quanto a do servidor podem ser feitas de forma independente, sem que cada um conheça o outro. Por fim, segundo [DevMedia 2020b], o uso de serviços REST tem crescido devido às empresas Google, Facebook, Yahoo!, Amazon, eBay e Microsoft.

O objetivo geral do trabalho foi projetar e implementar a integração dos ambientes simulador MASPN e seu portal.

Os objetivos específicos do trabalho foram:

- Entender e projetar módulos com tecnologias para *webservices* (linguagens de programação, pacotes/bibliotecas, *plugins* para ambientes de desenvolvimento);
- Definir uma arquitetura e fluxo de trabalho para o *webservice* (relação sistema *Desktop* com sistema Web);
- Realizar adequações necessárias na ferramenta de simulação e no portal Web para que o *webservice* entre em funcionamento.

2. Revisão bibliográfica

Nesta seção, são abordados a ferramenta de simulação MASPN e seu portal Web. Além disso, são discutidos conceitos, técnicas e ferramentas para projeto e implementação de *webservice*.

2.1. Pesquisa MASPN

O ambiente de simulação MASPN (Figura 1) está no universo de pesquisas em Nanociências e auxilia na produção e caracterização de nanopartículas poliméricas, por meio de simulação com a técnica de Inteligência Artificial de sistemas multiagentes. O ambiente de simulação MASPN busca reduzir tempo e custos em laboratório [Zamberlan 2018].

O projeto MASPN é composto pelo ambiente de simulação (projetado e implementado em Java para ambiente *Desktop*) e pelo seu portal Web para gestão de fármacos, polímeros e experimentos, os quais podem ser utilizados na ferramenta de simulação [Pereira et al. 2018]. O portal (Figuras 2 e 3) foi projetado e implementado via Pyton-Django e banco de dados MySQL.

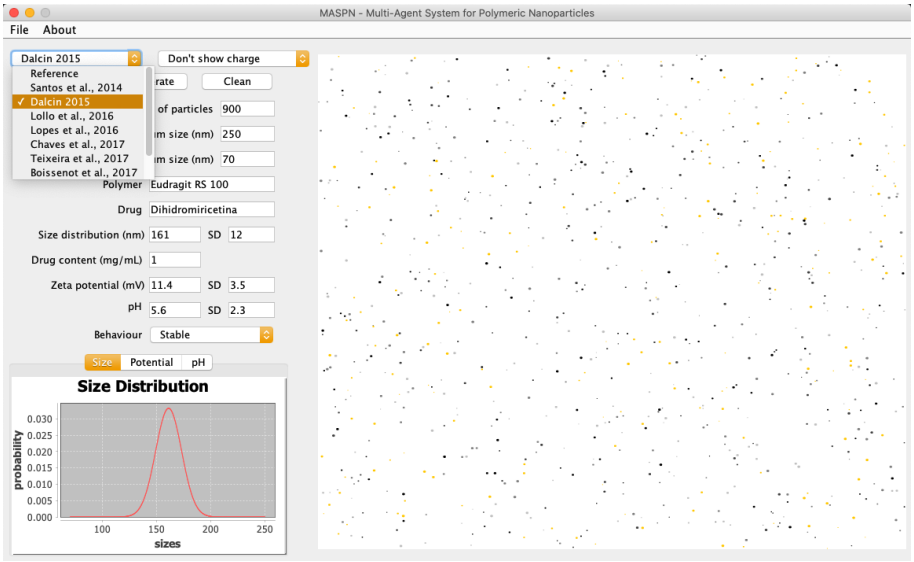

Figura 1. Interface *Desktop* para a simulação de sistemas nanoparticulados.

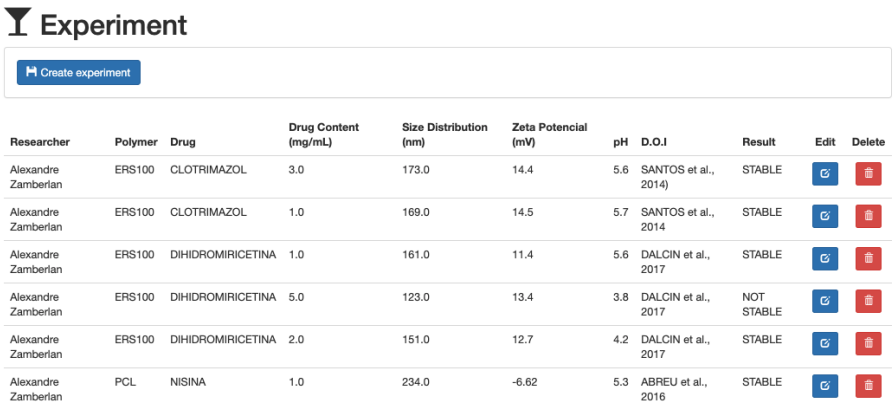

Figura 2. Interface de gestão de experimentos no portal Web.

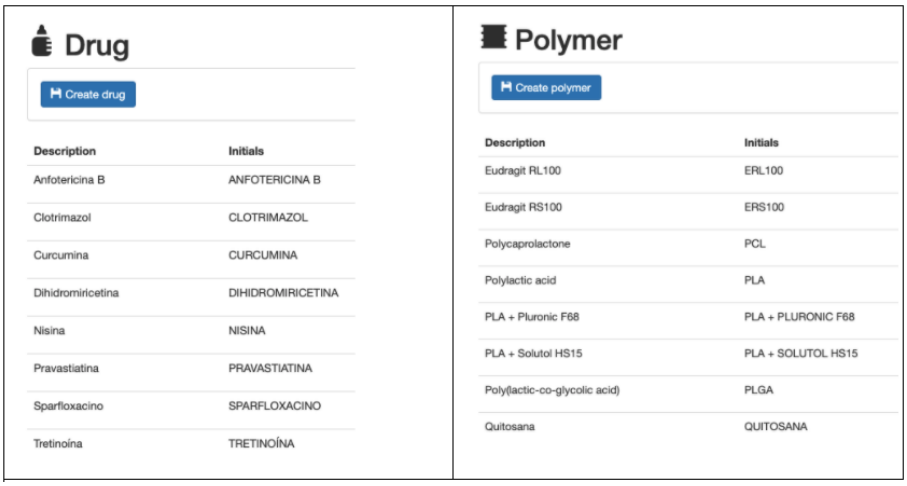

Figura 3. Interfaces de gestão de fármacos e de polímeros no portal Web.

2.2. Webservice

O *webservice* é formado por componentes de software ou hardware em que as aplicações podem trocar dados entre si [Coulouris et al. 2003] e [Tanenbaum 1994]. Cada aplicação pode ter a sua própria ”linguagem”, que é traduzida para uma linguagem universal, ou seja, em um formato intermediário como *Extensible Markup Language* (XML), *JavaScript Object Notation* (JSON), *Comma-Separated Values* (CSV), entre outros.

Para as empresas, *webservices*, além da agilidade nos processos e eficiência na comunicação, há questões embutidas de segurança entre os dados trocados ou compartilhados. Além disso, toda a comunicação entre sistemas é dinâmica, pois não há intervenção humana [Coulouris et al. 2003].

2.2.1. SOAP e REST

SOAP é um protocolo de comunicação entre aplicações heterogêneas, com um formato baseado em *Extensible Markup Language* (XML) para enviar e receber mensagens, sob uma plataforma independente e recomendado pela W3C [W3School 2020b]. Por muitos anos, foi o protocolo oficial para construção de *webservices*.

REST é um estilo de arquitetura cliente-servidor que fornece padrões de comunicação entre computadores conectados na Web. Contudo, tanto a implementação do cliente quanto a do servidor podem ser feitas de forma independente, sem que cada um conheça o outro [DevMedia 2020b]. Sistemas no padrão REST fazem que o código no lado do cliente possa ser alterado a qualquer momento sem afetar a operação do servidor, e o código no lado do servidor possa ser alterado sem afetar a operação do cliente.

Em [DevMedia 2020b], há uma análise importante de quando usar SOAP ou REST. De acordo com relato, a implementação de *webservices*, em 99% dos casos, é por meio do SOAP, e isso se deve ao fato de que o protocolo foi, e de certa forma ainda continua sendo, o mais utilizado para a criação de *webservices*, de um modo tal que para muitos o termo *webservice* está necessariamente atrelado ao protocolo SOAP [DevMedia 2020b].

Como já mencionado no texto, REST vem ganhando espaço no mercado e têm sido utilizado por empresas como Google, Facebook, Yahoo!, Amazon, eBay, Microsoft [DevMedia 2020b], asssociado com *JavaScript Object Notation* (JSON).

2.2.2. XML e Java

XML é uma linguagem de marcação que define um conjunto de regras para codificação de documentos em um formato que pode ser lido por humanos e por computadores [Stackabuse 2020]. Ela tem uma estrutura de marcação semelhante ao HTML e foi projetada para armazenar e transportar dados entre sistemas heterogêneos (como o que ocorre nesta pesquisa). XML é comumente usado para trocar dados entre serviços da Web, *Back-End* e *Front-End* [Stackabuse 2020].

Os elementos de XML podem ser definidos como blocos ou recipientes para guardar texto, elementos/objetos, atributos. Cada XML contém um ou vários elementos, e eles são delimitados por etiquetas (de início e de fim de bloco), ou para elementos vazios

por uma etiqueta do vazio-elemento.

Java Document Object Model (JDOM) é um modelo de documento em forma de objeto baseado na linguagem de programação Java. De fato, é uma representação de um documento XML em Java, que fornece uma maneira de representar esse documento para leitura com manipulação e escrita fáceis e eficientes, com uma API (*Application Program Interface*) simples. Entretanto, para construção desses documentos, é preciso utilizar um analisador sintático (*parser*) externo [JDOM 2020].

2.2.3. *JavaScript Object Notation* (JSON)

JSON é um formato de representação de dados no estilo texto, baseado na linguagem de programação Javascript. É compacto para troca simples e rápida de dados entre sistemas (heterogêneos), sendo uma alternativa ao XML, uma vez que é mais rápido [DevMedia 2020c]. Esse fato justifica que a Google utilize o formato para a troca de grandes volumes de dados. Ou seja, o *parser* do JSON é mais eficiente que o *parser* XML. Apesar de estar associado à Javascript, pode ser utilizado com outras linguagens [DevMedia 2020c].

De acordo com [DevMedia 2020c], JSON possui uma sintaxe muito simples. Para cada valor que se deseja representar é preciso atribuir um nome ou rótulo que detalhe seu significado (semântica), muito parecido com a sintaxe do JavaScript. Na Figura 4, é possível visualizar uma lista de livros, em que cada livro tem os atributos *título*, *resumo*, *ano*, *palavras-chave*, *publicado* e seus valores respectivos, circundados por aspas-duplas. Note que uma lista é formada com dois colchetes ”[ ]”, como em Python e Javascript, representando uma tabela do banco de dados, por exemplo. Já o uso de duas chaves ”{ }”, representa um dicionário ou uma tupla (linha) do banco de dados, por exemplo.

```
4   [
5       {
6           "titulo"            : "JSON versus XML",
7           "resumo"            : "o duelo de dois modelos de representação de informações",
8           "ano"               : 2020,
9           "palavras-chave"    : ["computação", "webservice", "integração"],
10          "publicado"         : true
11      },
12      {
13          "titulo"            : "Webservice",
14          "resumo"            : "entenda o conceito de integração de sistemas",
15          "ano"               : 2019,
16          "palavras-chave"    : ["computação", "heterogêneo", "internet"],
17          "publicado"         : false
18      }
19  ]
```

Figura 4. Exemplo de código JSON com dois objetos (livros) e seus atributos.

Assim como JDOM está para XML e Java, existe a biblioteca GSON de código aberto que integra JSON com a linguagem Java. GSON foi criado pela empresa Google e é usada para converter objetos Java em sua representação JSON, também sendo usado para converter uma *string* JSON em um objeto Java equivalente [Google 2020]. Dessa forma, GSON é uma biblioteca que serializa e deserializa objetos Java à JSON.

Sendo assim, JSON é uma espécie de “concorrente” da XML na área de troca

de informações, possuindo algumas semelhanças e diferenças para a representação de informações [DevMedia 2020a]. Destacam-se algumas semelhanças [DevMedia 2020a]:

- os dois modelos representam informações no formato texto;
- ambos possuem natureza auto-descritiva;
- ambos são capazes de representar informação complexa, difícil de representar no formato tabular, como objetos compostos (objetos dentro de objetos), hierarquia, atributos multi valorados, matrizes, dados ausentes, etc;
- são utilizados para transportar informações em aplicações AJAX;
- podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627;
- ambos são independentes de linguagem. Dados em XML e JSON podem ser acessados por qualquer linguagem de programação, por meio de API específica.

Já as diferenças, de acordo com [DevMedia 2020a], são:

- JSON não é uma linguagem de marcação, pois não possui *tags* de abertura e fechamento;
- JSON representa as informações de forma mais compacta;
- JSON não permite a execução de instruções de processamento, algo possível em XML;
- JSON é destinado para a troca de informações, já o XML possui mais aplicações.

2.3. Trabalhos relacionados

Nesta subseção, buscou-se apresentar e discutir trabalhos com tecnologias ou metodologias que pudessem dar suporte para esta pesquisa.

2.4. Webservice para acesso a dados da aplicação Caronas

O trabalho foi uma implementação *webservice* que funciona de *backend* da aplicação de Caronas Solidárias [dos Santos Conceição 2017]. O *webservice* garante a 'ponte' entre os bancos de dados do Centro de Processamento de Dados da Universidade Federal do Rio Grande do Sul (UFRGS) e aplicações móveis de clientes (alunos, professores e funcionários da instituição) que desejam oferecer ou receber caronas para o trânsito até os diferentes *campi* da universidade. A implementação é baseada na arquitetura REST, e responde em formato JSON e XML. Há um processo de autenticação que impede o acesso por usuários não identificados, ou seja, alunos, professores e funcionários da universidade [dos Santos Conceição 2017].

2.5. Desenvolvimento de uma API REST para um Sistema Acadêmico de Terceiros

Neste artigo, foram apresentados os passos do desenvolvimento de uma *Application Program Interface* (API) de consulta de dados em sistema acadêmico de terceiros (sistema contratado e pago) implantado no Instituto Federal de Alagoas. A principal razão para o desenvolvimento da API foi permitir que alunos, professores e analistas da área da Tecnologia da Informação pudessem desenvolver novos sistemas que utilizassem dados do sistema terceirizado [Ana Silva Ferreira 2018].

O trabalho gerou uma aplicação protótipo, direcionada para os coordenadores de curso do instituto, que permite a visualização das notas de um determinado aluno em um formato de Gráfico de Radar. A ideia é que a partir de uma única visualização

gráfica o coordenador possa ter uma visão ampla do desempenho quantitativo do aluno [Ana Silva Ferreira 2018].

Analisando os trabalhos e comparando com esta proposta, percebe-se que a utilização da arquitetura REST vem tendo uma crescente utilização para a construção de *webservices*, por ser de fácil uso, flexível, multiplataforma, multilinguagem, entre outras. Todos os trabalhos relacionados utilizaram a ideia de *webservice* justamente para que outro sistema, em diferente plataforma, linguagem, localização, pudesse consumir dados de um sistema dito servidor, ou seja, um sistema de terceiro, um sistema Web, etc.

3. Proposta de trabalho

O sistema de banco de dados do portal MASPN é MySQL. Dessa forma, é importante, para a construção do *webservice*, definir a exportação dos dados do banco para JSON e para script MySQL e SQLite, pois se mostraram mais interessantes para serem usados nesta proposta de trabalho. Registra-se que a opção de uso do banco de dados SQLite dá-se porque muitos usuários do simulador são usuários Windows, com nenhuma habilidade para instalar e configurar o banco de dados MySQL.

3.1. Materiais e métodos

Este trabalho é baseado em pesquisa exploratória com revisão bibliográfica amparado com estudo de caso. Já no projeto e desenvolvimento da solução foram utilizados a metodologia *Scrum* com a técnica *Kanban*.

As ferramentas utilizadas foram:

- Trello: ferramenta online para operacionalizar a técnica Kanban, que gerencia o cronograma de atividades deste trabalho;
- Astah: ferramenta gratuita para a diagramação de aspectos funcionais e estruturais da solução proposta;
- Linguagem Python, *framework* Django e o ambiente de desenvolvimento Visual Studio Code;
- linguagem JSON;
- Pacotes de integração *mysqldump*, *dumpdata* e *mysql2sqlite*.

3.2. Modelagem e Projeto do webservice MASPN

Na Figura 5, é possível visualizar o papel do *webservice* na relação simulador e portal. Toda vez que o banco é alterado no portal[1], um arquivo JSON e dois arquivos *scripts* (MySQL e SQLite) de consumo são exportados/gerados para que pesquisadores via simulador possam utilizar experimentos. O destaque é o pacote *webservice* que possui as funcionalidades para exportação do banco de dados (JSON, MySQL e SQLite).

Este trabalho foi realizado em concomitância com o trabalho final de graduação de Pierre Fenner [Fenner and Zamberlan 2021]. Enquanto este gera o *webservice*, o trabalho de Pierre consome o *webservice* na aplicação Java *Desktop* para simulação de nanopartículas poliméricas (MASPN). Por se tratar de uma aplicação *Desktop* para usuários

[1]Também há a possibilidade de geração do *webservice* no crontab do servidor Linux.

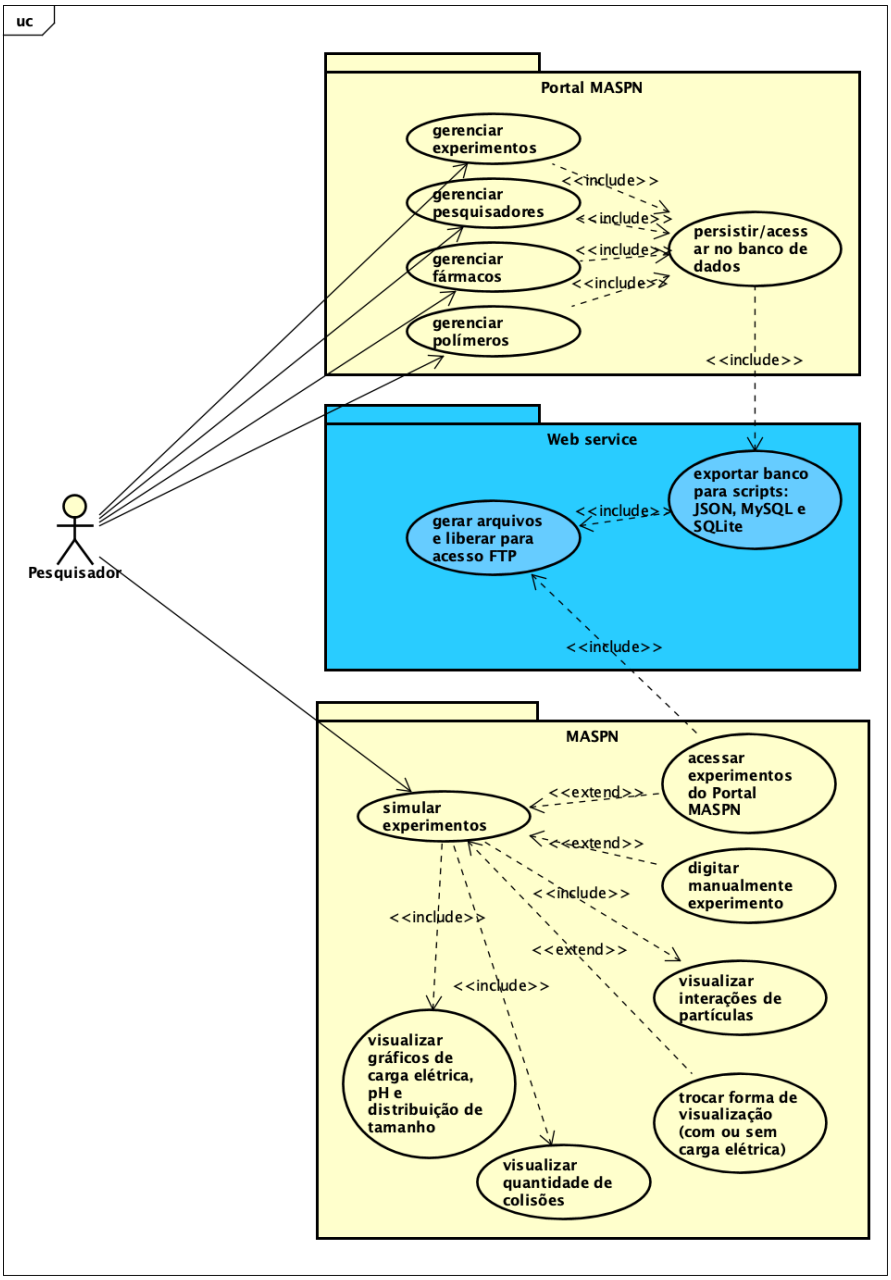

Figura 5. Diagrama de Casos de Uso do Simulador, Portal e *webservice*.

da área de saúde (farmacêuticos, biomédicos, químicos e engenheiros químicos), a ferramenta de simulação vai ter como banco de dados oficial o SQLite. Mas, com a possibilidade de usuários mais experientes trocarem o banco de dados para MySQL (já que são necessárias etapas de instalação e configuração desse banco).

Em relação aos aspectos estruturais do projeto MASPN, já com a ideia do *webservice* integrado ao portal e ao simulador, a Figura 6 ilustra como os sistemas fazem a relação entre eles. O destaque é o pacote portal em que a classe $webservice$ deve possuir no mínimo um experimento e todos os experimentos são enviados ou disponibilizados ao simulador. E todo experimento deve conter polímero, fármaco (*drug*) e pesquisador

responsável (*researcher*).

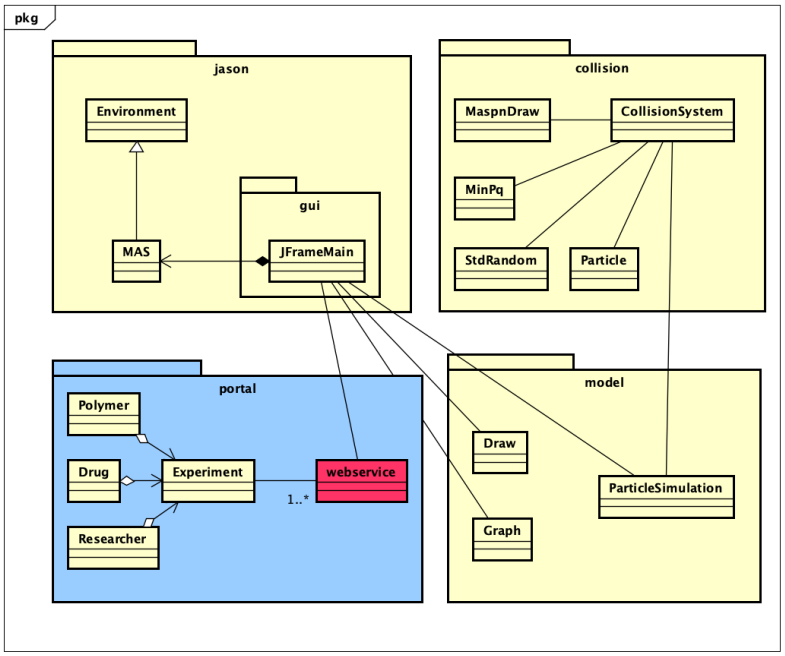

Figura 6. Diagrama de Domínio do Projeto MASPN, adaptado de [Zamberlan 2018].

A Figura 7 ilustra as tabelas no banco de dados do portal, em que *drug* representa o fármaco, *polymer* representa o polímero e *researcher* e *experiment* representam, respectivamente, dados de pesquisador e de um experimento.

Para melhor compreensão da dinâmica de geração e consumo, a Figura 8 mostra o fluxo de como o portal gera o *webservice* e como ele pode ser consumido pela ferramenta de simulação *desktop*. Observe que há duas possibilidades da geração do *webservice*: uma realizada toda vez que o banco de dados do portal for atualizado (gestão de pesquisador, fármaco, polímero e experimento); outra, agendada diariamente, às 5 horas, no serviço *crontab* do servidor portal.

A vantagem de se gerar o *webservice* quando o banco de dados é alterado nas gestões básicas (usuário, fármaco, polímero e experimento), é a possibilidade do simulador acessar o banco atualizado a qualquer momento. Porém, há a desvantagem de se criar um gargalo de processamento caso o banco de dados esteja volumoso. Por esse motivo, decidiu-se usar a geração do *webservice* de forma agendada e programada no *crontab* do servidor.

O *crontab* deve ser alterado no usuário *root* ou super usuário do servidor, para que o processo seja garantido pelo sistema. Seguem os comandos necessários no *crontab*.

```
0 5 * * * mysqldump -u usuario -psenha nome_banco > /home/webservice/webservice.sql
0 5 * * * mysql2sqlite -d nome_banco -u usuario -p senha
          -f /home/webservice/webservice.sqlite

0 6 * * * chmod +r /home/webservice/webservice.sql
0 6 * * * chmod +r /home/webservice/webservice.sqlite
```

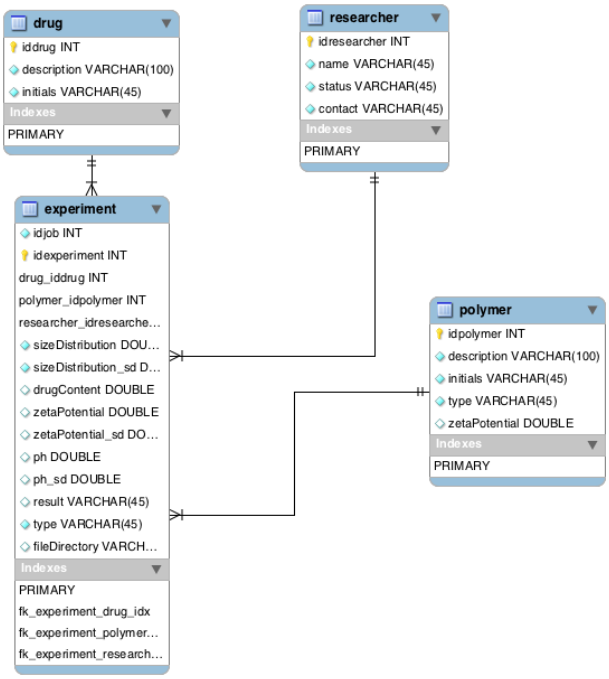

Figura 7. Diagrama Entidade e Relacionamento do Portal e *webservice* [Zamberlan 2018].

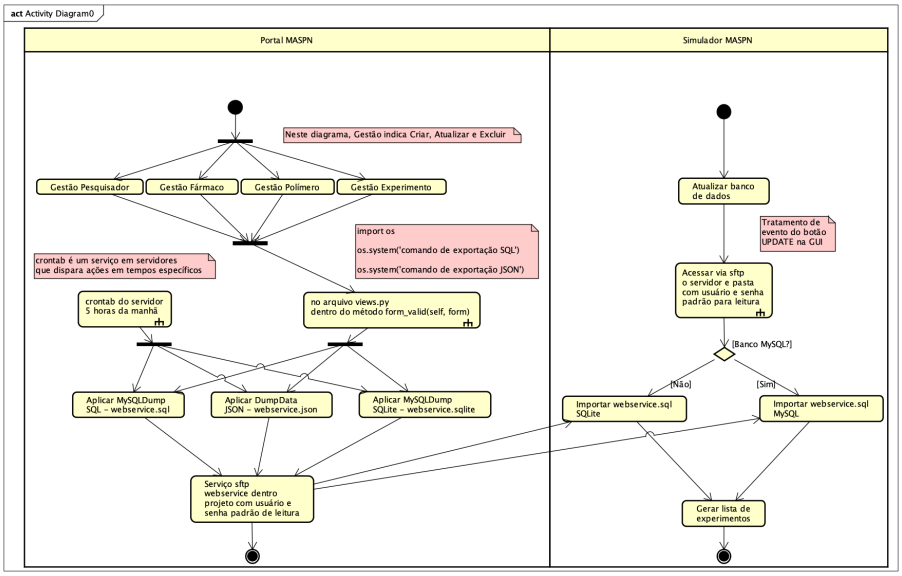

Figura 8. Diagrama de atividades de geração e consumo do webservice.

A primeira linha do *crontab* gera o arquivo para consumo de simulador, enquanto a segunda linha gera permissão de acesso do arquivo, uma hora depois, para que a ferramenta *Desktop* possa fazer um acesso remoto via serviço *sftp* (Figura 8).

Outro ponto importante no servidor, é a criação de um usuário sem permissões de administrador, somente como a permissão de suportar conexões *sftp*.

A geração automática do *webservice*, após alteração no portal, pode ser visuali-

zada na Figura 9 contendo as classes *Create* e *Update* de um experimento. Dessa forma, no método *form_valid* são disparados as gerações dos arquivos (linhas 21, 22 e 23; 39, 40 e 41).

```
15  class ExperimentCreateView(LoginRequiredMixin, OrdinaryRequiredMixin, CreateView):
16      model = Experiment
17      fields = ['researcher', 'polymer', 'drug', 'drug_content', 'size_distribution','size_distribution_standard_deviation',
18                'zeta_potencial', 'zeta_potential_standard_deviation','ph','ph_standard_deviation', 'doi', 'result']
19      success_url = 'experiment_list'
20
21      def form_valid(self, form):
22          try:
23              os.system('comando de exportação MySQL')
24              os.system('comando de exportação JSON')
25              os.system('comando de exportação SQLite')
26              return super().form_valid(form)
27          except Exception as e:
28              messages.error(self.request, 'Web service não gerado neste momento')
29              return super().form_invalid(form)
30
    You, seconds ago | 2 authors (You and others)
31  class ExperimentUpdateView(LoginRequiredMixin, OrdinaryRequiredMixin, UpdateView):
32      model = Experiment
33      fields = ['researcher', 'polymer', 'drug', 'drug_content', 'size_distribution','size_distribution_standard_deviation',
34                'zeta_potencial', 'zeta_potential_standard_deviation','ph','ph_standard_deviation', 'doi', 'result']
35      success_url = 'experiment_list'
36
37      def form_valid(self, form):
38          try:
39              os.system('comando de exportação MySQL')
40              os.system('comando de exportação JSON')
41              os.system('comando de exportação SQLite')
42              return super().form_valid(form)
43          except Exception as e:
44              messages.error(self.request, 'Web service não gerado neste momento')
45              return super().form_invalid(form)
46
```

Figura 9. Classes de Criação e Atualização de um experimento.

Finalmente, na Figura 10, há o simulador *desktop* MASPN que tem um *comboBox* com os experimentos cadastrados no portal e consumidos via acesso remoto.

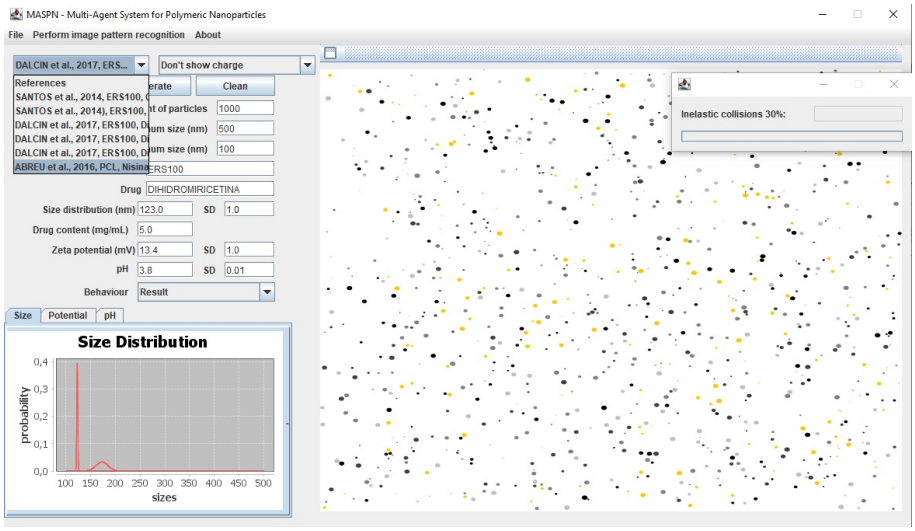

Figura 10. Interface principal do simulador MASPN com a relação dos experimentos consumidos via *webservice* [Fenner and Zamberlan 2021].

4. Conclusões

O texto apresentou a proposta do trabalho e a relação do simulador (ferramenta *Desktop*) com o portal MASPN (sistema Web). Foi realizada uma revisão sobre *webservice* e tecnologias para sua construção. Destaca-se que, até recentemente, o padrão SOAP era o

mais utilizado, uma vez que tem excelente relação com XML. Contudo, com a chegada de JSON e da disponibilidade do *framework* GSON, a implementação de *webservices* também pode ser realizada com outras tecnologias.

Este trabalho utilizou-se do recurso *crontab*, do sistema operacional do servidor, para agendar a geração do *webservice* em horários específicos. O *webservice* é composto por 3 arquivos: *webservice.sql* (para o banco de dados MySQL), *webservice.sqlite* (para o banco SQLite) e *webservice.json* (de uso genérico para o simulador). Para isso, pesquisou-se como gerar o *webservice*, em que, em primeiro momento, essa geração seria sempre no momento em que o banco da aplicação fosse alterado. Contudo, percebeu-se que haveria um gargalo de processamento quando o sistema do portal tivesse muitos dados. Ou seja, o processo de geração do banco poderia prejudicar o servidor. Dessa forma, decidiu-se colocar essa funcionalidade no agendamento do sistema operacional em um horário em que o servidor é pouco utilizado.

Além disso, foram necessários algumas alterações de infra-estrutura do servidor, como por exemplo, a criação de um usuário *webservice* com permissões restritas (somente acesso seguro para transferência de arquivos - *sftp*), em que o simulador pudesse acessar o servidor via esse usuário para realizar download dos arquivos *webservices*.

Cabe ressaltar que esse processo de geração automatizada do *webservice* vai ser útil e importante para a geração automática de *backups* do Portal MASPN.

Referências

Ana Silva Ferreira, Jalves Mendonça Nicacio, G. V. F. G. M. S. F. V. S. R. (2018). Desenvolvimento de uma API REST para um Sistema Acadêmico de Terceiros. https://www.researchgate.net/. Acessado em Abril de 2021.

Codecademy (2020). What is REST? https://www.codecademy.com/articles/what-is-rest. Acessado em outubro de 2020.

Coulouris, G., Dollimore, J., and Kindenberg, T. (2003). *Distributed Systems: Concepts and Design*. Addison-Wesley, New York, 3rd edition.

DevMedia (2020a). JSOM Tutorial. https://www.devmedia.com.br/json-tutorial/25275. Acessado em setembro de 2020.

DevMedia (2020b). REST Tutorial. https://www.devmedia.com.br/rest-tutorial/28912. Acessado em outubro de 2020.

DevMedia (2020c). Uma introdução ao JSON. https://www.devmedia.com.br/json-tutorial/25275. Acessado em setembro de 2020.

Django.cowhite (2020). Exportando e importando dados no Django usando mysqldump no MySQL. https://django.cowhite.com/blog/exporting-and-importing-data-in-django-using-pg_dump-in-postgresql-mysqldump-in-mysql-dumpdata-and-loaddata-commands-in-django. Acessado em Outubro de 2020.

dos Santos Conceição, L. (2017). Webservice para acesso a dados da aplicação Caronas. https://www.lume.ufrgs.br/. Acessado em março de 2021.

Fenner, P. C. and Zamberlan, A. (2021). *Refatoração da ferramenta MASPN em JASON*. Universidade Franciscana, Santa Maria.

Google (2020). Guia do usuário Gson. https://sites.google.com/site/gson/gson-user-guide. Acessado em setembro de 2020.

JDOM (2020). JDOM: site de referência. http://jdom.org. Acessado em setembro de 2020.

Pereira, G. G., Brum, J. V. R., Vieira, S., Fagan5, S., Laporta, L., and Zamberlan, A. (2018). Portal Web para simulação no ambiente MASPN. In *XXII Simpósio de Ensino, Pesquisa e Extensão*, Santa Maria. Universidade Franciscana, Universidade Franciscana.

Sqlshack (2020). Como fazer backup e restaurar bancos de dados MySQL usando o comando mysqldump. https:https://www.sqlshack.com/how-to-backup-and-restore-mysql-databases-using-the-mysqldump-command/. Acessado em setembro de 2020.

Stackabuse (2020). Ler e escrever XML em Java. https://stackabuse.com/reading-and-writing-xml-in-java/. Acessado em setembro de 2020.

Tanenbaum, A. S. (1994). *Distributed Operating Systems*. Prentice Hall, New York.

W3School (2020a). SQL Introduction. https://www.w3schools.com/sql/sql_intro.asp. Acessado em outubro de 2020.

W3School (2020b). XML Soap. https://www.w3schools.com/xml/xml_soap.asp. Acessado em outubro de 2020.

Zamberlan, A. (2018). *Sistema Multiagente para avaliação do efeito de aglomeração em nanopartículas poliméricas*. PhD thesis, Universidade Franciscana - UFN, Santa Maria.

Zamberlan, A., Bordini, R., Kurtz, G., and Fagan, S. (2020). Multi-agent systems, simulation and nanotechnology. In *Multi Agent Systems-Strategies and Applications*. IntechOpen.

Zamberlan, A., Dalcin, A. J., Kurtz, G., Bordini, R., Raffin, R., and Fagan, S. (2016). Simulation environment for polymeric nanoparticle: experiment database. *Disciplinarum Scientia*, 17(3):429–446.

Zamberlan, A. d. O., Kurtz, G. C., Gomes, T. L., Bordini, R. H., and Fagan, S. B. (2019). A simulation environment for polymeric nanoparticles based on multi-agent systems. *Journal of Molecular Modeling*, 25(1):5.

5. Apêndice

Esta seção traz tutoriais para exportar e importar o banco de dados em diferentes *scripts*.

5.1. Exportar MySQL para JSON e SQLite

No tutorial em [Django.cowhite 2020], há as instruções, em linha de comando, via o software Mysqldump, para importar e exportar dados MySQL por meio do Django.

Mysqldump é um aplicativo de linha de comando usado para gerar o *backup* lógico do banco de dados MySQL, produzindo instruções SQL[2] que para recriar os objetos e dados do banco de dados. O comando também pode ser usado para gerar a saída no formato XML, JSON ou CSV. [Sqlshack 2020].

5.1.1. Exportando e importando o banco de dados para SQL - MySQL

Para a exportação de dados gerais, utiliza-se o comando:

```
mysqldump -u db_username  db_name  table_name --password  >  export_mysql_data.sql
```

Registra-se que o arquivo gerado é o 'export_mysql_data.sql' e que o parâmetro '–password' vai solicitar que o usuário digite a senha.

Para exportar os dados com o usuário do banco de dados 'myuser', nome do banco de dados 'db_name', endereço de rede (host) 'localhost' ou 127.0.0.1 ou qualquer endereço remoto, o comando é:

```
mysqldump -u myuser db_username -h localhost --password > export_mysql_data.sql
```

Se o endereço for um IP remoto, então utiliza-se:

```
mysqldump -u myuser db_name -h 1.2.3.4 --password > export_mysql_data.sql
```

A exportação de uma única tabela pode ser feita da seguinte maneira:

```
mysqldump -u myuser db_name table_name -h 1.2.3.4 > export_mysql_data.sql
```

Já para a importação dos dados do banco, deve-se utilizar o comando mysql, ao invés do mysqldump. Perceba também que a importação é sobre o arquivo 'export_mysql_data.sql':

```
mysql  -u  db_username  db_name  -h  host  --password < export_mysql_data.sql
```

Um exemplo para importar com o usuário do banco de dados 'myuser', nome do banco de dados 'db_name', endereço de rede 'localhost':

```
mysql -u myuser db_name -h localhost --password < export_mysql_data.sql
```

Se o endereço for um IP remoto, então usa-se:

```
mysql -u myuser db_name -h 1.2.3.4 --password < export_mysql_data.sql
```

```
Usage: mysql2sqlite [OPTIONS]

  Transfer MySQL to SQLite using the provided CLI options.

Options:
  -f, --sqlite-file PATH       SQLite3 database file  [required]
  -d, --mysql-database TEXT    MySQL database name  [required]
  -u, --mysql-user TEXT        MySQL user  [required]
  -p, --prompt-mysql-password  Prompt for MySQL password
  --mysql-password TEXT        MySQL password
  -t, --mysql-tables TEXT      Transfer only these specific tables (space
                               separated table names). Implies --without-
                               foreign-keys which inhibits the transfer of
                               foreign keys.

  -X, --without-foreign-keys   Do not transfer foreign keys.
  -h, --mysql-host TEXT        MySQL host. Defaults to localhost.
  -P, --mysql-port INTEGER     MySQL port. Defaults to 3306.
  -S, --skip-ssl               Disable MySQL connection encryption.
  -c, --chunk INTEGER          Chunk reading/writing SQL records
  -l, --log-file PATH          Log file
  -V, --vacuum                 Use the VACUUM command to rebuild the SQLite
                               database file, repacking it into a minimal
                               amount of disk space

  --use-buffered-cursors       Use MySQLCursorBuffered for reading the MySQL
                               database. This can be useful in situations
                               where multiple queries, with small result sets,
                               need to be combined or computed with each
                               other.

  -q, --quiet                  Quiet. Display only errors.
  --version                    Show the version and exit.
  --help                       Show this message and exit.
```

Figura 11. Opções de comandos da ferramenta.

5.1.2. Exportando e importando o banco de dados para SQL - SQLite

No site https://pypi.org/project/mysql-to-sqlite3/, há a descrição e a exemplificação da ferramenta Python *mysql-to-sqlite3* para converter MySQL em SQLite.

Para isso, *mysql-to-sqlite3* precisa estar instalado no servidor via comando *pip install mysql-to-sqlite3*. A Figura 11 mostra todas as opções da ferramenta.

Para a exportação de dados gerais, utiliza-se o comando:

```
mysql2sqlite -d nome_banco -u usuario -p senha  -f /caminho/arquivo.sqlite
```

5.1.3. Exportando e importando o banco de dados para JSON

Uma vez que o banco foi exportado para a linguagem SQL, é possível converte-lo para JSON. Na Internet há inúmeros *scripts* gratuitos. O utilizado neste trabalho é o disponibilizado em [Django.cowhite 2020] e que foi construído em Python. A execução do *script* é:

```
python manage.py dumpdata > export_mysql_data.json
```

O comando 'dumpdata' converte os dados SQL (existentes na aplicação Django) para o formato JSON (arquivo 'export_mysql_data.json'), independentemente do banco de dados utilizado.

[2]Linguagem para acessar e manipular banco de dados relacionais [W3School 2020a].

Por sua vez, o parâmetro '–indent' exporta um JSON legível com o recuo especificado, que no exemplo é 4.

```
python manage.py dumpdata --indent = 4 > export_mysql_data.json
```

Já o comando de gerenciamento 'loaddata' faz o caminho contrário, isto é, carrega os dados exportados em um arquivo para o banco da aplicação Django.

```
python manage.py loaddata < export_mysql_data.json
```