Refatoração da ferramenta de simulação MASPN

Pierre C. Fenner[1], **Alexandre Zamberlan**[1]

[1]Universidade Franciscana(UFN)
Santa Maria – RS

pierre.fenner@ufn.edu.br, alexz@ufn.edu.br

Resumo. *Este estudo faz parte da pesquisa do ambiente de simulação para sistemas nanoparticulados poliméricos (MASPN) e do seu portal Web para gestão de experimentos. O objetivo do trabalho foi refatorar a ferramenta MASPN nas tecnologias AgentSpeak(L) e JASON, deixando-a aderente à teoria de Sistemas Multiagentes Inteligentes. Além disso, adequar os pacotes de simulação algs4 e jFreeChart e refatorar o esquema de banco de dados utilizado pelo simulador. A pesquisa é exploratória com revisão bibliográfica apoiada em um estudo de caso. Já em relação à metodologia de desenvolvimento do sistema, foram utilizadas a metodologia ágil Scrum e a técnica de gestão de atividades Kanban. Os resultados mais significativos foram uma interface gráfica com novas funcionalidades, principalmente em relação ao interpretador JASON e a forma como os dados de experimentos são consumidos pelo simulador.*

Abstract. *This study is part of the research related to the simulation environment for polymeric nanoparticulate systems (MASPN) and its Website, which manages experiments. The main goal was to refactor the MASPN tool using AgentSpeak(L) and JASON technologies, ensuring better adherence to the Intelligent Multiagent Systems theory. In addition, adapt the algs4 and jFreeChart simulation packages and refactor the database scheme used by the simulator. The research is exploratory with a literature review supported by a case study. Regarding the system development methodology, the Scrum agile methodology and the Kanban activity management technique were used. The most significant results were a graphical interface with new functionalities, mainly in relation to the JASON interpreter and the way the experiment data is consumed by the simulator.*

1. Introdução

A Ciência da Computação possui ferramentas, metodologias e boas práticas que dão suporte às diferentes áreas do saber, como por exemplo à Nanociência. Uma das contribuições da área da Computação é a modelagem e a simulação computacional por meio de Sistemas Multiagentes [Zamberlan 2018]. Dessa forma, este trabalho é baseado na pesquisa realizada e publicada por [Zamberlan et al. 2016], [Zamberlan 2018] e [Zamberlan et al. 2020] em que foi projetado e implementado uma ferramenta de simulação para nanopartículas poliméricas, conhecida como MASPN (*Multi-Agent System for Polymeric Nanoparticles*) [Zamberlan et al. 2019], com uso da linguagem de programação Java, do pacote de computação científica *algs4*, partes do interpretador JASON, ambiente de desenvolvimento NetBeans, metodologia *Feature Driven-Development*

(FDD), metodologia PROMETHEUS. Porém, a ferramenta para possuir um melhor desempenho junto à teoria de Sistemas Multiagentes, deveria ser totalmente projetada e construída usando as linguagens AgentSpeak(L) e Java (de forma integrada), bem como o interpretador JASON, com implementação pelo ambiente de programação Eclipse, que possui os *plugins* de desenvolvimento JASON disponíveis (diferente do ambiente NetBeans).

O objetivo geral do trabalho foi refatorar[1] a ferramenta MASPN nas tecnologias AgentSpeak(L) e JASON, deixando-a aderente à teoria de Sistemas Multiagentes Inteligentes. Para atingir o objetivo geral, alguns objetivos específicos foram elencados: i) estudar a pesquisa realizada em [Zamberlan 2018]; ii) estudar a ferramenta Eclipse e *plugins* JASON; iii) integrar o pacote de computação científica *algs4* no ambiente Eclipse para garantir o movimento Browniano das partículas (direção e velocidade de movimento, massa e volume de partículas; colisões elásticas e inelásticas); iv) programar (refatorar) os processos de simulação das partículas por JASON; v) programar (refatorar) as animações de colisão entre partículas. Para um melhor entendimento do proposto, o trabalho está dividido em seções. Há a seção 2 com a revisão bibliográfica que contextualiza o leitor no contexto do trabalho. A seção 3 apresenta a proposta de refatoração do sistema, com a remodelagem estrutural e funcional do simulador exclusivamente dentro do paradigma Sistemas Multiagentes em JASON, e os resultados alcançados. Finalmente, a conclusão e as referências do trabalho.

2. Revisão bibliográfica

Nesta seção, são discutidos conceitos e definições dentro do projeto MASPN (simulador e portal), além da teoria Sistemas Multiagentes com ênfase na linguagem AgentSpeak(L) e seu interpretador JASON. Por fim, é discutido o tema refatoração de sistemas.

2.1. Projeto MASPN: ferramenta de simulação

Conforme apresentado em [Zamberlan et al. 2019], o projeto *Multi-agent System for Polymeric Nanoparticle* (MASPN) é um ambiente de simulação inovador e original com recursos para demonstrar interações de nano partículas poliméricas, a partir de parâmetros físico-químicos, garantindo o movimento *browniano* das partículas e reduzindo tempo e gastos em laboratórios com experimentos reais. O fluxo básico de funcionamento da ferramenta segue: i) parâmetros são coletados a partir de dados reais ou hipotéticos de um experimento; ii) são inseridos na ferramenta, que simula (calculando e animando) as interações entre partículas; iii) informando, ao final, se esses parâmetros geram ou não um sistema estável (não aglomerado).

O ambiente foi projetado e construído de acordo a metodologia *Feature-Driven Development* (FDD) e a abordagem de sistemas multiagentes. Também, foi utilizado o pacote de simulação orientado a eventos *algs4*[2]. O ambiente MASPN, como ferramenta de simulação, possui interface gráfica, gestão de parâmetros físico-químicos integrados, animação do movimento *browniano*, gráficos de distribuição para tamanho, potencial zeta

[1] Alterar a estrutura interna de um software de forma a torná-lo mais fácil de perceber sem modificar o seu comportamento observável [Porto 2020].

[2] Pacote com classes e métodos de computação científica e que garante o comportamento real das colisões entre partículas, de acordo com leis da Física Clássica [Zamberlan 2018].

(carga elétrica) e pH de partículas. Toda a ferramenta foi construída sob a teoria de orientação a objetos na linguagem Java. Até o momento da refatoração, as principais funcionalidades do simulador são gerenciar experimentos e simular.

2.2. Portal MASPN

Segundo discutido em [Pereira et al. 2018], o projeto do sistema Web apresenta um portal para gestão de pesquisadores, fármacos, polímeros e experimentos de nanopartículas poliméricas para o uso no ambiente de simulação MASPN. A implementação foi realizada com a linguagem de programação Python, *framework* Django de desenvolvimento de sistemas Web; *framework* Bootstrap para a interface do sistema. Também foi utilizado *Model-View-Controller* (MVC), que é um padrão arquitetural de software para implementação de interfaces de usuário em computadores.

A dinâmica de funcionamento entre o simulador e o portal é de que pesquisadores inserem dados sobre experimentos no portal e podem simular na ferramenta *desktop*. Cabe ressaltar que um *webservice* [Costa and Zamberlan 2021] em JSON, MySQLDump e MySQL2SQLite foi projetado e construído concomitantemente a este trabalho, justamente para facilitar a integração do portal com o simulador.

2.3. Sistemas Multiagentes

Os SMA formam uma área de investigação na Inteligência Artificial Distribuída (IAD), tratando os aspectos referentes à computação distribuída em sistemas de inteligência artificial [Wooldridge 2001]. Um agente é um sistema de computador em que está situado em um ambiente, e é capaz de realizar ações autônomas no intuito de realizar os objetivos designados. Assim, SMA é formado de vários agentes que interagem em ambiente compartilhado com o objetivo de individual ou coletivo.

A definição da arquitetura interna do agente está associada com o tipo de tarefa que o agente irá realizar e qual o seu papel na sociedade multiagente. Dessa forma, o que caracteriza um agente e a sociedade que ele está inserido são as interações com o ambiente e os processos internos que possibilitam a realização dessas interações [Wooldridge 2001]. A especificação de quais e como são esses processos internos é chamada de arquitetura. Diferentes arquiteturas têm sido propostas com o objetivo de caracterizar os agentes com um nível de inteligência e de autonomia.

De acordo com [Zamberlan 2018], os ambientes de simulação de nanoestruturas são ideais para arquiteturas reativas de agentes. O uso de SMA possibilita projetar sistemas complexos de simulação, por exemplo, de maneira naturalmente distribuída (descentralizada e focada no agente) e *bottom-up* (construção a partir das funcionalidades do agente até chegar no sistema como um todo). De acordo com um dos autores da ferramenta JASON, Jomi Hübner, *apud* [Zamberlan 2018], o paradigma de SMA baseia-se em sistemas naturais, como ocorre em um formigueiro (onde a formiga seria um agente e o formigueiro seria o ambiente, sendo que este último possui a inteligência coletiva). Uma característica significativa na teoria orientada a agentes é a autonomia, também existente em estruturas de sistemas nanoestruturados. Em relação à organização de um sistema multiagente reativo ou cognitivo, há eventos, restrições e interações, o que também ocorre num ambiente na escala nano[3]. Também conforme [Zamberlan 2018], SMA pos-

[3]Unidade de medida que refere $10^{-9} metros$, ou seja, a bilionésima parte da unidade metro

suem, logo, agentes que têm um comportamento de acordo com o ciclo perceber, planejar e atuar. Por exemplo, uma partícula seria um agente que interage com outras partículas e com a parede de seu ambiente (caixa). A caixa seria o ambiente contendo o solvente (que dissiparia ou não a energia de interação das partículas). Assim, os agentes partículas devem perceber outras partículas, as paredes da caixa e o solvente do ambiente.

2.3.1. AgentSpeak(L) e JASON: *a Java-based interpreter for an extended version of AgentSpeak*

Programação Orientada a Agentes (POA) é um paradigma de programação em que o desenvolvimento do software é por meio de agentes (programas, rotinas ou métodos). Dentro da área de linguagens de programação orientado a agentes, AgentSpeak(L) tem sido a linguagem mais influente entre todas com base na arquitetura BDI (Belief-Desire-Intention) [Bordini et al. 2007]. É uma linguagem declarativa do paradigma lógico, trabalhando com proposições lógicas, ou seja, cada crença, cada desejo e cada intenção (planos), ao invés de serem tratados como métodos, são regras de transição [Bordini et al. 2007].

Os tipos de agentes implementados com AgentSpeak(L), as vezes são chamados de sistemas de planejamento reativo. Assim, JASON surge como o principal interpretador da linguagem AgentSpeak(L). Outra característica importante sobre JASON é que comparado a outros sistemas de agentes BDI, a linguagem é implementada em Java e é software livre. Além dessas características, JASON possui uma integração com o ambiente de programação Eclipse via *plugin* específico JASON [Hübner and Bordini 2020].

2.4. Refatoração de Sistemas

Refatoração é uma mudança feita na estrutura interna de um sistema para torná-lo mais fácil de ser entendido e modificável, sem mudar o seu comportamento observado [Fowler 2018].

Alguns profissionais da área, utilizam da "refatoração" sempre que fazem qualquer limpeza no código [Fowler 2018]. Contudo, refatorar possui uma abordagem além de limpeza de código. Refatoração é sobre aplicar pequenos ajustes e, assim, realizar uma mudança significativa ao unificar essas melhorias [Fowler 2018]. Refatorar é similar a otimização de desempenho, em que há alteração de código sem alteração de funcionalidades do programa. De acordo com [Fowler 2018], a grande diferença é que refatorar promove deixar o código mais "fácil de entender e simples de modificar". Enquanto, a otimização de desempenho, apenas se importa em agilizar o processamento de código. Um aspecto importante de refatorar sistemas é eliminar código duplicado, para melhor entendê-lo e para facilmente modificá-lo [Fowler 2018]. Ajudar a entender o código também significa ajudar a encontrar *bugs* mais rapidamente. Algumas técnicas para refatorar são: i) *Extract function (Method)* que é uma técnica em que é analisado um fragmento de código dentro de uma função e criado uma função a partir dele. É utilizado quando a sua função (método) possui muitas linhas de código, o que pode ser difícil de entender a lógica da função. Algumas pessoas possuem uma preocupação com pequenas funções por que podem ter um alto custo de desempenho. Porém, a otimização dos compiladores atuais trabalham melhor com funções reduzidas, pelo motivo conseguir tratar dados em

memória *cache* mais facilmente; ii) *Extract variable* é utilizada quando há uma expressão e é necessário quebra-lá em variáveis. Expressões podem se tornar complexas, dificultando a leitura, por isso a sua redução em variáveis, que também pode ser útil em reusos; iii) *Inline function (Method)* é uma técnica que é o inverso da *extract function*, em que é capturar uma função, analisar, e trazer o seu conteúdo para o lugar que a está chamando. É utilizada frequentemente quando um código tem funções que delegam a outras funções repetidas vezes. Por exemplo, função A que chama função B que chama função C, assim por diante; *Inline variable* é uma técnica quando a variável não consegue comunicar mais que a expressão em si, se tornando apenas uma ponte entre a expressão e o resultado. Ou seja, é o inverso da *extract variable*.

No IDE Eclipse, há um modulo de refatoração, onde já foram implementadas as técnicas apresentadas e outras que foram descritas [Fowler 2018]. Além da refatoração ser utilizada de forma interativa, também pode ser usada para *scripts* de refatoração, em que as refatorações (técnicas) mais usadas são armazenadas em um histórico de refatorações. Inclusive há o *Refactoring Wizard*, que auxilia no processo de refatoração, bem como o nome expressa [Eclipse 2003a].

2.5. Eclipse, *plugin* Windows Builder e *plugin* JASON

Os ambientes de desenvolvimento integrado (IDE) ajudam os desenvolvedores a programar novas aplicações de forma rápida, já que as várias funcionalidades presentes no IDE não precisam ser ajustadas e integradas manualmente durante a instalação e configuração do ambiente [Hat 2020]. No IDE Eclipse existem *plugins* que possibilitam a integração com vários pacotes [Eclipse 2020], como exemplo, pacotes para diagramação UML, para design gráfico de sistemas, controle de versionamento de código. Dessa forma, *plugins* contribuem com funcionalidades especificas e que são ativadas quando são necessárias [Melhem and Glozic 2003].

Neste projeto, é utilizado o *plugin* WindowBuilder, que é composto pelos pacotes *SWT Designer* e *Swing Designer*. Isso, faz com que no Eclipse seja possível a criação (clicar-arrastar-soltar) de aplicações com interface gráfica em Java, sem a necessidade de escrever, literalmente, o código [Eclipse 2003b]. No ambiente Eclipse, pode-se utilizar *plugin* JASON, em que é possível a criação de projetos SMA (arquivo *.mas2j*), adicionar agentes (arquivo *.asl*), incluir o *mind inspector*, que ajuda a debugar e analisar o sistema desenvolvido [Hübner and Bordini 2020]. Além disso, há todo o recurso de *IntelliSense* do Eclipse (auto-completar e o acesso ao Java Docs), possibilidade de adicionar diferentes pacotes e *plugins* (*Windows Builder* para Interfaces Gráficas), integrar sistemas heterogêneos, etc.

2.6. Trabalhos relacionados

Nesta subseção, são discutidos os trabalhos relacionados que apresentam resultados em refatoração de sistemas.

O trabalho realizado em [Melo Junior et al. 2021] apresentou a ferramenta RefaX-AOP genérica para refatoração de códigos orientados a aspectos. Esse ambiente auxilia o processo de melhoria de código da linguagem AspectJ. Porém, Refax-AOP passou por um processo de refatoração, que melhorou a ferramenta original Refax. Nesse trabalho, foi refatorado a arquitetura do Refax: o componente *refaX core* foi adaptado, o pacote

CodeAccessFunction foi trocado pelo *RefactoringDefinition*. As técnicas de refatoração aplicadas foram estruturais e funcionais (*Extract function, Extract variable, Inline function e Inline variable*). E como ferramentas e tecnologias foram utilizadas XSLT, Java, AspectJ RefaX e AspectJML. O autor relata que a ferramenta facilitou a definição de novas refatorações por meio da solução proposta, ou seja, a codificação em XSLT, em que(revisar) foi possível extrair as facilidades de uma linguagem estruturada e baseada em *templates*.

O trabalho em [França and Soares 2013] apresenta uma análise sobre a refatoração por meio da programação orientada a aspectos, utilizando como estudo de caso um sistema de controle de monografia (SCM). A refatoração alterou a estrutura do código fonte, reescrevendo as classes, métodos e condições dentro das classes. As técnicas utilizadas foram converter código procedural em orientado a objetos, renomear métodos, extrair métodos e reverter condicionais. Portanto pode-se afirmar que houve refatoração estrutural e funcional (*Extract function, Extract variable, Inline function e Inline variable*). As tecnologias para o desenvolvimento da ferramenta foram a linguagem de programação C# e a plataforma *dot* NET. Foi observado uma melhoria no software, sejam elas em eficiência, reutilização, simplicidade e manutenção, após as várias refatorações, assim como uma redução de linhas de código.

Como apresentado em [Melo Junior et al. 2021], que utilizou refatoração arquitetural, neste estudo também se utilizou essa técnica. Já no trabalho [França and Soares 2013], foi observado a utilização de técnicas funcionais, com foco na alteração do código fonte, trazendo um maior desempenho para o software. Dessa forma, esta pesquisa também tem refatoração funcional, principalmente relacionado a comunicação ao portal (banco de dados) e a utilização do interpretador JASON. Finalmente, destaque-se que as técnicas de refatoração utilizadas neste trabalho foram: i) refatoração funcional (*extract and inline function*): consumo de experimentos é via um acesso remoto (*webservice*) no servidor que mantém o Portal MASPN. Ou seja, a forma de armazenamento e tratamento dos dados foi modificada; ii) refatoração arquitetural/estrutural (*extract and inline variable*): reorganização do modelo MVC, em termos de pastas do projeto, uma vez que os dados de experimentos não são mais produzidos no simulador, mas sim no portal. Inserção do banco de dados SQLite como alternativa ao banco MySQL, logo, adição do *driver* de conexão do SQLite; iii) refatoração funcional e estrutural: integração do projeto MAS no projeto MASPN, ou seja, criação do projeto MASPN dentro do projeto MAS em JASON.

3. Refatoração do simulador

O projeto MASPN, no que diz respeito a ferramenta de simulação *desktop*, é o que deve receber o processo de refatoração. Ou seja, a refatoração do simulador foi realizada no IDE Eclipse com os *plugins* Windows Builder e JASON. E a principal justificativa desse processo de refatoração no ambiente Eclipse é por conta do seu suporte ao *plugin* JASON. Dessa forma, foi possível integrar a teoria BDI de Sistemas Multiagentes com o interpretador JASON, que não acontecia no antigo simulador. Havia a teoria BDI implementada, mas a dinâmica de funcionamento estava dividida e não integrada.

3.1. Materiais e métodos

Este trabalho é uma pesquisa exploratória com revisão bibliográfica apoiada em um estudo de caso, concomitante à pesquisa realizada por [Costa and Zamberlan 2021]. Em relação à metodologia de desenvolvimento desta pesquisa, é utilizada metodologia ágil *Scrum* [Pressman 2010] [Wykowski and Wykowska 2019] e a técnica de gestão de atividades *Kanban*. As ferramentas e materiais utilizados foram: Trello (ambiente para gestão de atividades (técnica Kanban)); Astah (ambiente para diagramação de aspectos funcionais e estruturais da solução); Github (repositório para códigos construídos e gestão de atividades); AgentSpeak (linguagem para programação na teoria *BDI (belief, design, intention)* [Bordini et al. 2007]); JASON (interpretador para a linguagem AgentSpeak [Bordini et al. 2007]); Eclipse (ambiente de desenvolvimento *(IDE - Integrated Development Enviroment)* com *plugins* JASON.

3.2. Modelagem do projeto

Devido ao uso da metodologia Scrum (iterativa e incremental), as funcionalidades e seus requisitos, periodicamente, são convertidos em protótipos. A Figura 1 apresenta um mapa mental da ideia de como estava o simulador e o que se realizou no processo de refatoração.

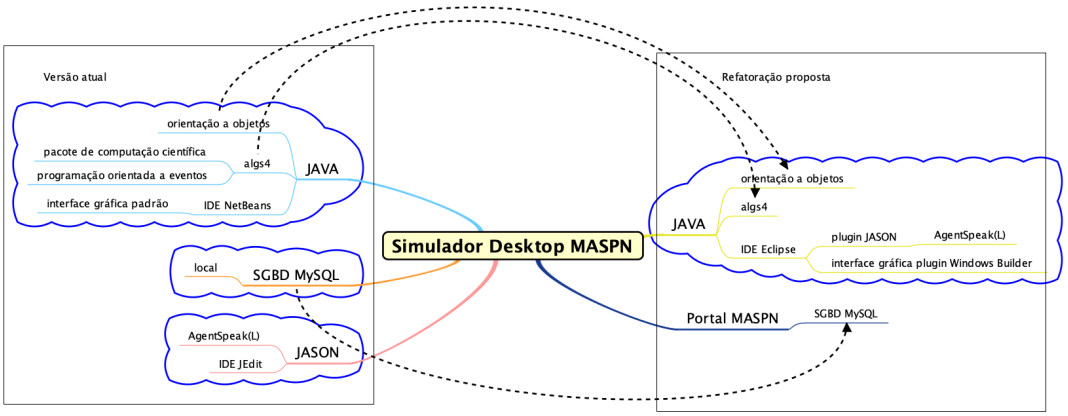

Figura 1. Mapa mental da refatoração desejada.

Em termos específicos, a refatoração atingiu:

- questões de banco de dados (aspecto estrutural), que antes eram em dois bancos (um no portal e outro no *desktop*), mas que agora a base concentra-se no portal com o consumo dos dados pelo simulador via *webservice*;
- a reorganização dos pacotes e das principais classes do simulador (aspecto estrutural), com a ideia de acessar os dados cadastrados no Portal MASPN;
- a reorganização de pastas e arquivos do projeto do simulador (aspectos estrutural e funcional). Nos retângulos de destaque Figura 2, é possível visualizar a refatoração ocorrida: i) na pasta *asl*, o código *particle.asl* recebeu alteração funcional para influenciar a simulação com dados de pH (que antes não existia); ii) a pasta *factory* recebeu adição de arquivos/classes e métodos para realizar o download do *webservice* do servidor e converter o arquivo *webservice.sql* para *webservice.sqlite* (conferindo refatoração em aspectos funcionais); na pasta *mas*, foi

adicionada uma funcionalidade para que o SMA contemplasse simulações com agentes/partículas com dados de pH; iv) na pasta *model*, houve uma alteração estrutural na classe *Particle*; v) na pasta *sqlite* (adicionada ao projeto), há o surgimento do banco local (*sqlite3*) populado pelos dados do *webservice*; vi) e na pasta *webservice*, surge para tratar os dados vindos do servidor;

- reprogramação das funcionalidades acessar experimentos do portal MASPN e executar SMA JASON, como mostra a Figura 3, pacote MASPN (em verde).

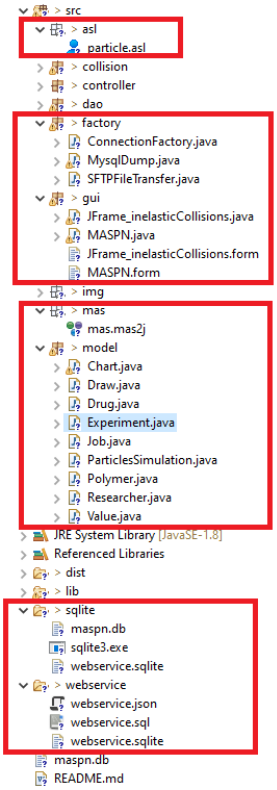

Figura 2. Refatoração estrutural de pastas e arquivos do simulador.

Na refatoração do sistema, em relação ao uso do banco de dados no simulador, há um fluxo bem diferente de produção, armazenamento e consumo de experimentos entre Portal MASPN e Simulador MASPN, como ilustrado na Figura 4. Cabe atenção tanto no *pool* Portal MASPN, quanto no *pool* Simulador MASPN. Enquanto que o portal fornece três opções de arquivo contendo dados do banco, o simulador também tem duas opções de consumo desses dados: MySQL ou SQLite.

3.2.1. Resultados da refatoração

Na Figura 5, é possível visualizar a interface refatorada do MASPN dentro do IDE Eclipse, via o *plugin* Windows Builder que garantiu que o simulador mantivesse a mesma estrutura gráfica. Uma refatoração gerada foi a troca do botão '*Manage Experiments*' pelo botão '*Update*' (refatoração estrutural e funcional).

A funcionalidade associada ao botão *Update*, pode ser visualizada na Figura 6, em há 3 processos principais: executar o download do *webservice*; atualizar o banco SQLite

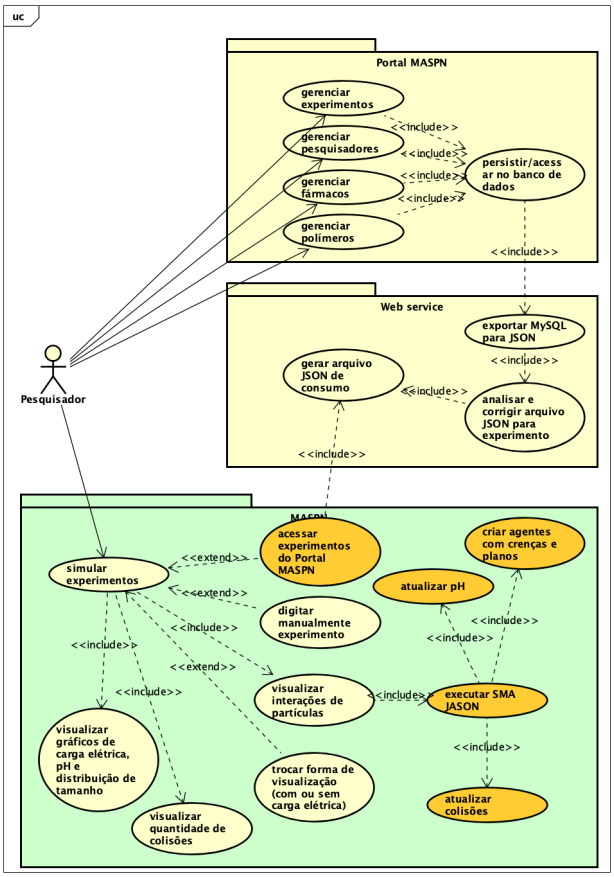

Figura 3. Diagrama de casos de uso refatorado.

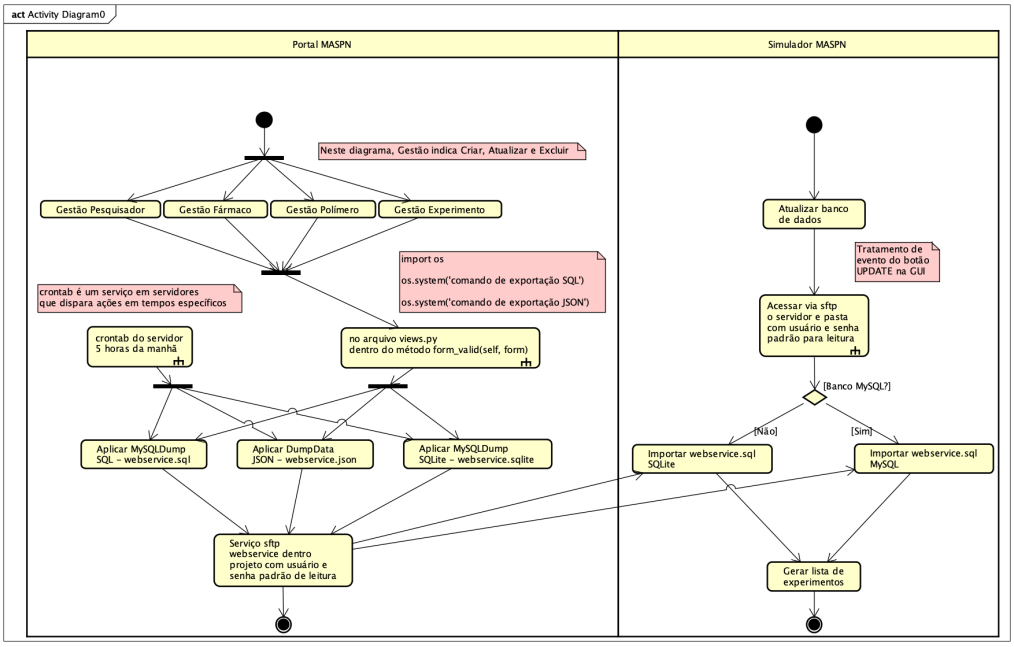

Figura 4. Diagrama de atividades da relação Portal e Simulador.

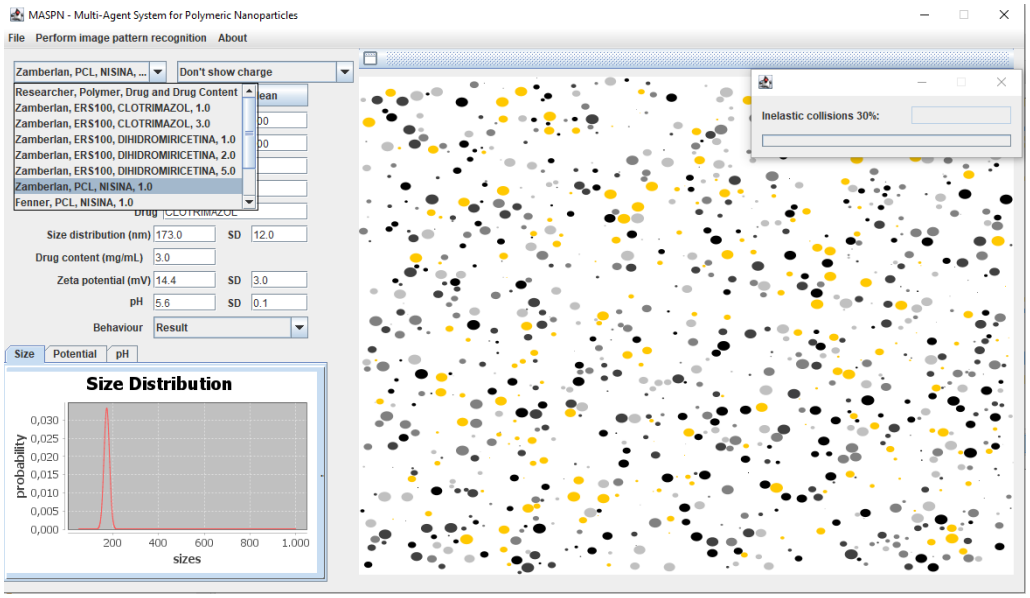

Figura 5. Versão do MASPN refatorada no Eclipse.

com novos dados do portal; preencher o componente *combo box*[4] da interface gráfica com a relação de experimentos do banco.

```
652    private void jButton_manageJobsActionPerformed(java.awt.event.ActionEvent evt){//GEN-FIRST:event_jButton_manageJobsActionPerforr
653        if (JOptionPane.showConfirmDialog(rootPane, "You will update the MASPN database!", "Confirm", JOptionPane.OK_CANCEL_OPTION)
654            |
655            //Downloading webservice file
656            SFTPFileTransfer.downloadSftp();
657            JOptionPane.showMessageDialog(this,"Download Finished", "Confirm", JOptionPane.INFORMATION_MESSAGE);
658            //execute sqlite file
659            MysqlDump.convertWebService();
660            //Filling comboBox
661           fillComboBoxExperiments();
662
663        }
664
```

Figura 6. Função de ação do botão *update*.

Na Figura 7, é possível visualizar a classe *SFTPFileTransfer* criada para conectar com o servidor, em uma pasta específica para o serviço *sftp*, de um usuário com permissões limitadas. Nela, há o construtor, que estabelece a conexão do usuário ao servidor. Entre as linhas 41 e 71 (Figura 8), há o método de classe *downloadSftp* para realizar o download dos arquivos *webservices* do servidor. A Figura 9 é a continuação do código da classe *SFTPFileTransfer* e na imagem é possível acompanhar como os arquivos SQLite e JSON são criados e armazenados.

3.2.2. Resultados da refatoração - SMA

Em relação à incorporação do simulador ao JASON, a propriedade pH influencia as simulações, uma vez que uma ambiente mais ou menos ácido faz com que as partículas agitem mais ou agitem menos.

Na Figura 10, ilustra o processo de refatoração do atributo pH em relação à geração do arquivo do projeto *.mas2j*. Ou seja, o projeto do SMA no JASON deve con-

[4]Um tipo de caixa de diálogo que contém controles do tipo deslizantes, caixas de texto e listas suspensas.

```java
public class SFTPFileTransfer {
    private static final String REMOTE_HOST = "lapinf.ufn.edu.br";
    private static final String USERNAME = [redacted];
    private static final String PASSWORD = [redacted];
    private static final int SESSION_TIMEOUT = 10000;
    private static final String remoteFileMysql = "/home/webservice/webservice.sql";
    private static final String remoteFileSqlite = "/home/webservice/webservice.sqlite";
    private static final String remoteFileJson = "/home/webservice/webservice.json";

    private static ChannelSftp setupJsch() throws JSchException {
        String username = USERNAME;
        String password = PASSWORD;
        String host = REMOTE_HOST;
        JSch jsch = new JSch();
        Session jschSession = jsch.getSession(username, host);
        java.util.Properties config = new java.util.Properties();
        config.put("StrictHostKeyChecking", "no");
        jschSession.setConfig(config);
        jschSession.setPassword(password);
        jschSession.connect();
        return (ChannelSftp) jschSession.openChannel("sftp");
     }
```

Figura 7. Parte 1 - Classe *SFTPFileTransfer*.

```java
public static boolean downloadSftp() {
    ChannelSftp channelSftp = null;
    try {
      channelSftp = setupJsch();
      channelSftp.connect(SESSION_TIMEOUT);
    } catch (JSchException e) {
      JOptionPane.showMessageDialog(null,"Fail to connect to database!", "Atention", JOptionPane.WARNING_MESSAGE);
    }

    try{
        FileWriter fwMysql = null;
        FileWriter fwSqlite = null;
        FileWriter fwJson = null;

        fwMysql = new FileWriter("webservice"+File.separator+"webservice.sql", false);
        fwSqlite = new FileWriter("webservice"+File.separator+"webservice.sqlite", false);
        fwJson = new FileWriter("webservice"+File.separator+"webservice.json", false);

        InputStream inputFile;
        String line;
        BufferedReader br;

        //leitura e criaÃ§Ã£o do arquivo local do mysql
        inputFile = channelSftp.get(remoteFileMysql);
        br = new BufferedReader(new InputStreamReader(inputFile));
        while ((line = br.readLine()) != null)
        {
            fwMysql.write(line+"\n");
        }
        br.close();
        fwMysql.close();
```

Figura 8. Parte 2 - Classe *SFTPFileTransfer*.

```java
        //Read and creating the sqlite local file
        inputFile = channelSftp.get(remoteFileSqlite);
        br = new BufferedReader(new InputStreamReader(inputFile));
        while ((line = br.readLine()) != null)
        {
            fwSqlite.write(line+"\n");
        }
        br.close();
        fwSqlite.close();

        //Read and creating the json local file

        inputFile = channelSftp.get(remoteFileJson);

        br = new BufferedReader(new InputStreamReader(inputFile));
        while ((line = br.readLine()) != null)
        {
            fwJson.write(line+"\n");
        }
        br.close();
        fwJson.close();


    } catch (IOException|SftpException e) {
        JOptionPane.showMessageDialog(null,"Fail to download the database files (mysql, or sqlite, or json)", "Atention",
    }
    channelSftp.exit();
    return true;
  }
}
```

Figura 9. Parte 3 - Classe *SFTPFileTransfer*.

ter todas as partículas da lista de partículas e todos as suas propriedades, convertidas em crenças.

```
        StringBuilder lines = new StringBuilder();
        lines.append("MAS maspn {\n");
        lines.append("\t infrastructure: Centralised\n");
        lines.append("\t environment: MAS\n");
        lines.append("\t agents:\n");
        Particle[] particles = new Particle[amountParticles];
        DecimalFormat df = new DecimalFormat("0.00");
        double rx, ry, vx, vy, dMinCorrelated, dMaxCorrelated,radiusUniform,massTemp,charge,new_pH;
        Color color;
        for (int i = 0; i < amountParticles; i++) {
            try {
                rx = StdRandom.uniform(0.0, 1.0);
                ry = StdRandom.uniform(0.0, 1.0);
                vx = StdRandom.uniform(-0.0009, 0.0009);
                vy = StdRandom.uniform(-0.0009, 0.0009);
                //double dMinCorrelated = dMin * 0.0005 / 50;
                //double dMaxCorrelated = dMax * 0.010 / 1000;
                //1k particles (0.0005 means 50, 0.010 -> maximum value means 1000)
                dMinCorrelated = dMin * 0.001 / 100;
                dMaxCorrelated = dMax * 0.010 / 1000;

                radiusUniform = StdRandom.uniform(dMinCorrelated, dMaxCorrelated);
                massTemp = StdRandom.uniform(0.2, mass);  //atention here.....
                color = colorsVector[StdRandom.uniform(colorsVector.length)];
                charge = StdRandom.gaussian(zetaPotential, zetaPotentialSD);
                new_pH = StdRandom.gaussian(pH, pHSD);

                if (new_pH <= 0) {
                    new_pH = 0.1;
                }
                particles[i] = new Particle(rx, ry, vx, vy, radiusUniform, massTemp, color, charge, pH);
                lines.append("\t\tparticle" + i + "[beliefs=\"charge(" + df.format(charge) + "),mass(" + df.format(massTemp) + "),pH("+df.format(new_pH)+")\"];\n");
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "Some parameters are out of range!\n"
                        + "Pay attention to the text boxes! ", "Atention", JOptionPane.WARNING_MESSAGE);
                return;
            }
        }
        lines.append("\t aslSourcePath: \n");
        lines.append("\t\t \"src/asl\"; \n");
        lines.append("}\n");
        FileWriter arq;
        try {
            arq = new FileWriter("src/mas/mas.mas2j");
            arq.append(lines);
            arq.close();
        } catch (IOException e) {
```

Figura 10. Função criar animação e projeto *.mas*.

3.2.3. Avaliação da ferramenta MASPN refatorada

Destaca-se que o processo de avaliação da refatoração da ferramenta de simulação teve como base os trabalhos (experimentos) utilizados na construção do MASPN, que foram elencados em [Zamberlan 2018]. A Figura 11 mostra 4 experimentos cadastrados no portal e avaliados na versão antiga do MASPN e na versão refatorada.

Experimentos	Quantidade máxima de partículas		Tempo início da simulação	
	Versão antiga	Versão nova	Versão antiga	Versão nova
SANTOS et al., 2014)	5700	6000	~2 segundos	~2 segundos
DALCIN et al., 2017	5500	6000	~3 segundos	~2 segundos
ABRE et al., 2016	1100	1100	~1 segundo	~4 segundos
OURIQUE et al., 2008	4500	4500	~2 segundos	~1 segundo

Figura 11. Experimentos utilizados em [Zamberlan 2018].

Nesse processo de avaliação, percebeu-se que houve um desempenho superior às simulações realizadas na versão antiga (no que se refere a quantidade de partículas e tempo de simulação). Os testes foram realizados em um hardware com processador Intel I5, memória RAM de 8 GB, velocidade de barramento de 2133 Mhz, Java *Runtime* 1.8.

Outros pontos de destaque da refatoração foram:

- botão de acesso ao banco do portal;
- *parsing* do arquivo *webservice.sqlite* para o banco SQLite;
- o interpretador JASON contém o simulador.

4. Conclusões

Registra-se que de todos os objetivos assumidos, somente o objetivo específico *programar (refatorar) as animações de colisão entre partículas* não foi atingido. Descobriu-se que o processo de animação é uma tarefa da área de Computação Gráfica e de alta complexidade, e decidiu-se deixá-lo como trabalho futuro.

Entretanto, esta pesquisa alcançou resultados significativos para o projeto MASPN, uma vez que foi realizada a conversão da interface gráfica construída em Netbeans para o IDE Eclipse, totalmente fidedigna e funcional como a original, pronta para uso. Destaca-se a instalação e configuração do JASON no Eclipse e o projeto *.mas2j* dentro do simulador. Além disso, todos os pacotes utilizados na versão original foram instalados, configurados e testados no sistema refatorado, como por exemplo pacote *algs4*[5], *jFreeChart*[6] e *drive* do MySQL. Também foi instalado, configurado e testado o *drive* do SQLite, já que o simulador terá a possibilidade de trabalhar com os dois bancos de dados.

4.1. Limitações

No processo de conversão do *webservice* para o banco SQLite, a ferramenta de simulação refatorada executa nos três sistemas operacionais (Windows, Linux e Mac OS). Porém, para a execução em Linux e Mac Os é preciso seguir alguns passos diferentes e específicos desses sistemas operacionais, que é a execução do arquivo *.jar* obrigatoriamente em terminal. Também, nesses dois sistemas operacionais a função atualizar banco de dados não está finalizada.

Sugestões de trabalhos futuros:

- dar a opção do usuário, após instalar o MASPN, escolher qual banco deseja utilizar: MySQL (configurações adicionais por parte do usuário) ou SQLite (embutido na ferramenta);
- estender a função atualizar banco de dados do simulador para que funcionem no sistema operacional Mac OS;
- garantir no processo de animação o efeito de colisões inelásticas, ou seja, quando partículas de cargas opostas colidirem, elas precisam manter-se aglomeradas.

Referências

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, Liverpol, UK.

Costa, E. G. and Zamberlan, A. (2021). *Web service para integração de dados entre a ferramenta de simulação MASPN e seu portal Web*. Universidade Franciscana, Santa Maria.

Eclipse (2003a). Refactoring support. https://help.eclipse.org/2020-09/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Freference%2Fref-menu-refactor.htm. Acessado em novembro de 2020.

Eclipse (2003b). Windows Builder. https://www.eclipse.org/windowbuilder/. Acessado em outubro de 2020.

[5]Com classes e métodos de computação científica, inclusive cálculos de movimento browniano de partículas.

[6]Para geração de gráficos.

Eclipse (2020). Desktop IDEs. https://www.eclipse.org/ide/. Acessado em outubro de 2020.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2th edition.

França, J. M. and Soares, M. S. (2013). Avalição de refatoração de software com programação orientada a aspectos usando metricas. https://repositorio.ufu.br/bitstream/123456789/12535/1/JoyceMeire.pdf. Acessado em março de 2021.

Hat, R. (2020). O que é ide? https://www.redhat.com/pt-br/topics/middleware/what-is-ide. Acessado em outrubro de 2020.

Hübner, J. F. and Bordini, R. H. (2020). Site oficial do JASON. http://jason.sourceforge.net/wp/description/. Acessado em setembro de 2020.

Melhem, W. and Glozic, D. (2003). Pde does plug-ins. https://www.eclipse.org/articles/Article-YourAcessado em Outubro de 2020.

Melo Junior, L. S., Mendonça, N. C., and de Menezes, R. S. (2021). Uma ferramenta de refatoração para AspectJ utilizando AspectJML e XSLT. https://www.researchgate.net/publication/229046345_Uma_Ferramenta_de_Refatoracao_para_AspectJ_utilizando_AspectJML_e_XSLT. Acessado em março de 2021.

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.

Porto (2020). Infopédia. https://www.infopedia.pt/dicionarios/lingua-portuguesa/refatorar. Acessado em agosto de 2020.

Pressman, R. S. (2010). *Software Engineering: a Practitioner's Approach*. McGraw-Hill Education, New York, 7th edition.

Wooldridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Wykowski, T. and Wykowska, J. (2019). Lessons learned: Using Scrum in non-technical teams. https://www.agilealliance.org/resources/experience-reports/lessons-learned-using-scrum-in-non-technical-teams. Acessado em agosto 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.