



# PUC

ISSN 0103-9741

Monografias em Ciência da Computação  
n° 14/2017

## **Capturando e Analisando Proveniência de Dados em Sistemas Multiagentes**

**Tassio Ferenzini Martins Sirqueira  
Carlos José Pereira de Lucena**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900**

**RIO DE JANEIRO - BRASIL**

## Capturando e Analisando Proveniência de Dados em Sistemas Multiagentes

Tassio Ferenzini Martins Sirqueira<sup>1,2</sup>, Carlos José Pereira de Lucena<sup>1</sup>

<sup>1</sup>Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

<sup>2</sup>Instituto Vianna Junior (FIVJ)

tmsirqueira@vianna.edu.br, lucena@inf.puc-rio.br

**Abstract.** This paper presents the development of FProvW3C framework, next to a multi-agent system where the framework is responsible for collecting and storing the provenance data. We describe the data structures of the framework, following the W3C PROV model, the UML diagrams of the framework and the application developed with the use of the framework, the standards of projects used and the tests that were carried out to validate the framework and the built system. At the end the next steps and some results that have been obtained so far are presented.

**Keywords:** Multiagent Systems; Data Provenance; Provenance in SMA.

**Resumo.** Esse trabalho apresenta o desenvolvimento do *framework* FProvW3C, junto a um sistema multiagente, onde o *framework* é responsável por coletar e armazenar os dados de proveniência. São descritas as estruturas de dados do *framework*, seguindo o modelo PROV da W3C, os diagramas UMLs do *framework* e da aplicação desenvolvida com o uso deste, os padrões de projetos empregados e os testes que foram realizados para validar o *framework* e o sistema construído. Ao fim são apresentados os próximos passos e alguns resultados que foram obtidos até o momento.

**Palavras-chave:** Sistemas Multiagentes; Proveniência de dados; Proveniência em SMA.

---

**Responsável por publicações:**

Rosane Teles Lins Castilho  
Assessoria de Biblioteca, Documentação e Informação  
PUC-Rio Departamento de Informática  
Rua Marquês de São Vicente, 225 - Gávea  
22453-900 Rio de Janeiro RJ Brasil  
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530  
E-mail: [bib-di@inf.puc-rio.br](mailto:bib-di@inf.puc-rio.br)

# Sumário

1	Introdução	1
2	Descrição do Problema	1
2.1	Visão Geral	2
2.2	Propósito	2
2.3	Análise de Domínio	3
2.4	Padrões de Projeto Utilizados	4
2.5	Descrição dos Agentes	7
3	Modelos	8
3.1	Análise do Domínio	8
3.1.1	Modelo de <i>Features</i>	8
3.1.1.1	Casos de Uso	9
3.1.1.1.1	Lista de atores	9
3.1.1.1.2	Diagramas	9
3.1.1.1.3	Detalhamento	9
3.1.2	Projeto do Domínio	11
3.1.2.1	Classe	11
3.1.2.2	Sequências	11
4	Descrição da aplicabilidade do <i>Framework</i>	13
4.1	Guia de Instalação e Configuração	13
4.2	Descrição da instanciação de produtos	14
5	Teste	15
6	Considerações finais	17
	Referências	17

## 1 Introdução

A engenharia de software habitualmente utiliza validação empírica para comprovar teorias, avaliar novas invenções ou fundamentar hipóteses. Como abordado por Nunes *et al.* (2012), os seres humanos estão envolvidos de alguma forma na maioria dos processos de software e, as decisões que são tomadas contribuem para uma explicação dos efeitos dos processos.

Conhecer os passos executados durante um processo pode auxiliar na avaliação dos resultados atingidos. Entretanto, estudar fenômenos do mundo real envolvem a obtenção de informações e essa atividade deve ser bem pensada, planejada e executada, para não ocorrer desvios durante a coleta dos dados. Outro fator que pode dificultar a avaliação e entendimento de um software é o uso de sistemas autônomos, onde ações são executadas por agentes de software, muitas das vezes complexas de serem observadas.

Partindo do princípio que dados de proveniência podem auxiliar na tomada de decisão e na validação das ações tomadas por agentes de software, esse trabalho busca descrever o *framework* FProvW3C. Este *framework* tem como objetivo auxiliar na modelagem e captura dos dados de proveniência em aplicações computacionais. Além disto é apresentado um sistema multiagente que será utilizado para demonstrar o funcionamento do *framework* e um sistema web para apresentação e análise dos dados coletados.

Além desta introdução, a seção 2 apresenta os problemas com a falta de informações de proveniência e os objetivos com o desenvolvimento do *framework*. A seção 3 descreve as características do *framework* e a da aplicação desenvolvida para demonstrar o funcionamento do mesmo, com as técnicas de engenharia de software aplicadas e sua documentação. Na seção 4 é descrita a aplicabilidade do trabalho proposto e os passos de execução da aplicação desenvolvida utilizando o *framework* FProvW3C. A seção 5 apresenta os testes realizados para avaliar o sistema desenvolvido com uso do *framework*. Por fim, a seção 6 apresenta as considerações finais a respeito do *framework* proposto, suas possíveis expansões e os resultados já obtidos até o momento.

## 2 Descrição do Problema

Determinar os passos executados por um sistema computacional não é trivial quando ocorrem de forma autônoma, como demonstrado em Sirqueira *et al.* (2017). Sistemas multiagentes (SMA) vem se expandindo e informações dos dados manipulados pelos agentes, bem como os passos executados por cada agente podem auxiliar na rastreabilidade e compreensão de um determinado estado no software, baseado no histórico.

Como abordado por Travassos (2007), a engenharia de software empírica baseia-se na observação ou experiência, entretanto, a falta de formalismo não possibilita um consenso de uma teoria, a validação de uma hipótese, a replicação do experimento não é fácil, além de não criar uma base de conhecimento sobre o assunto investigado. Price e Shanks (2011) mostraram que informações sobre os dados influenciam não apenas a tomada de decisão sobre tais dados, mas também o processo de tomada de decisão geral. Informações de proveniência tendem a enriquecer dados utilizados, processados ou produzidos por aplicações computacionais quando estes são avaliados em um estudo.

O problema a ser atacada é a aplicação de um modelo padrão para captura de dados de proveniência, de forma que estes dados possam agregar valor, facilitando a replicação do estudo e permitindo a rastreabilidade das ações dos agentes em um SMA.

## 2.1 Visão Geral

Existem diversos modelos de proveniência de dados, entres os principais destacam-se o OPM (Moreau et al., 2008) e o PROV (Missier et al., 2013). O PROV é padronizado pela W3C, possui maior representatividade que o modelo OPM, trata a proveniência retrospectiva e prospectiva dos dados e possui formas de representação também padronizada para XML, JSON e para ontologia.

No *framework* FProvW3C foi empregado o modelo PROV que será apresentado a seguir.

## 2.2 Propósito

O propósito do *framework* FProvW3C é facilitar a captura de dados, baseado no modelo PROV de proveniência, onde a modelagem e as definições das entidades de persistência encontram-se pré-definidas, reduzindo o trabalho dos pesquisadores para implementar um sistema de coleta de dados e utilizando como arcabouço o modelo padronizado pela W3C.

O modelo PROV apresenta 3 vértices principais que são: i) entidade; ii) agente e; iii) atividade, além de um conjunto de relacionamentos primários e secundários entre estes vértices, conforme as figuras 1 e 2.

		Object			
		Entity		Activity	Agent
Subject	Entity	<b>WasDerivedFrom</b> Revision Quotation PrimarySource AlternateOf SpecializationOf HadMember	<b>WasGeneratedBy</b> WasInvalidatedBy	$\begin{matrix} R \\ T \\ L \end{matrix}$	<b>WasAttributedTo</b>
	Activity	<b>Used</b> WasStartedBy WasEndedBy	<b>WasInformedBy</b>		
	Agent	—	—	—	<b>ActedOnBehalfOf</b>

Figura 1: Relacionamentos Primários do PROV.

		Secondary Object		
		Entity	Activity	Agent
Subject	Entity	—	WasDerivedFrom (activity)	—
	Activity	WasAssociatedWith (plan)	WasStartedBy (starter) WasEndedBy (ender)	—
	Agent	—	ActedOnBehalfOf (activity)	—

Figura 2: Relacionamentos Secundários do PROV.

O FProvW3C apresenta além dos vértices principais, os relacionamentos primários e secundários, os atributos contidos em cada vértice e relação, seguindo estritamente a especificação do modelo de dados do PROV, denominado PROV-DM.

## 2.3 Análise de Domínio

Apesar do domínio de aplicação do *framework* ser extensa, foram propostas 3 aplicações:

1. Sistemas autônomos de compra e venda de mercadorias utilizando como base o menor preço;
2. Sistemas autônomos de compra e venda de ações, simulando à bolsa de valores, utilizando como base o histórico de preços;
3. Sistema autônomo de identificação de problemas (maus cheiros, alta volatilidade, forte acoplamento e etc.) em código fonte, utilizando dados do repositório de código GITHUB<sup>1</sup>;

Para demonstrar o uso do *framework* foi desenvolvida a primeira aplicação listada. Por se tratar de um *framework*, os pontos fixos são definidos pelo próprio modelo PROV, e como pontos extensíveis podemos destacar a inserção de novos atributos nos vértices, nas relações causais e novas relações causais. Os exemplos de pontos de extensões aplicáveis a cada sistema proposto pode ser visto nas figuras 3, 4 e 5 respectivamente.

	Entity	Activity	Agent
Entity	WasInfluenceBy	-	-
Activity	-	-	-
Agent	-	-	WasStartedBy WasEndedBy

Figura 3: Relacionamento extensíveis no 1º sistema.

	Entity	Activity	Agent
Entity	WasInfluenceBy	-	-
Activity	-	Used	-
Agent	-	-	WasStartedBy WasEndedBy

Figura 4: Relacionamento extensíveis no 2º sistema.

	Entity	Activity	Agent
Entity	WasInfluenceBy	-	WasGeneratedBy
Activity	-	Used	-
Agent	WasInfluencedBy	-	WasStartedBy WasEndedBy

Figura 5: Relacionamento extensíveis no 3º sistema.

Os pontos de extensão referentes aos atributos dependem da aplicação que será utilizado. No 1º sistema foram estendidos os atributos “name” na classe Agent, “description” em Activity e “price” e “title” em Entity.

---

<sup>1</sup> GITHUB: <http://github.com/>

## 2.4 Padrões de Projeto Utilizados

No sistema utilizado para demonstrar o *framework* proposto foram utilizados 4 padrões. Os padrões foram o MVC, DAO, *Singleton* e *Factory Method*.

O padrão arquitetural *Model-View-Controller* (MVC) é uma forma de separar uma aplicação, ou até mesmo um pedaço da interface de uma aplicação, em três partes: o modelo, a visão e o controlador. Essa separação pode ser vista na Figura 6.

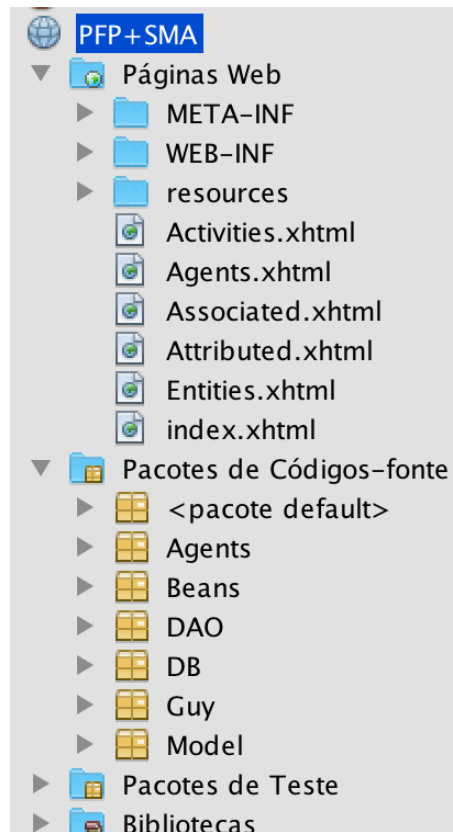


Figura 6: Padrão MVC utilizado no sistema.

O padrão DAO é um padrão de projeto que abstrai e encapsula os mecanismos de acesso a dados, escondendo os detalhes da execução da origem dos dados. Eles criam uma camada de independência entre a lógica de negócios e o acesso a banco, permitindo a substituição do sistema gerenciador de banco de dados (SGBD) de forma fácil, desde que seja utilizada a linguagem *Java Persistence Query Language* (JPQL). O uso do padrão pode ser visto nas figuras 7 e 8 respectivamente.

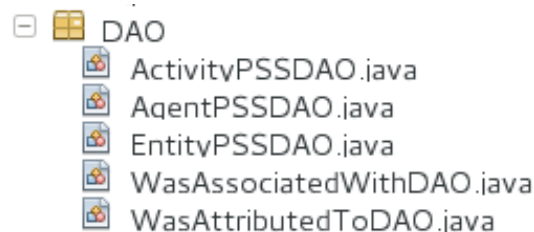


Figura 7: Padrão DAO utilizado no sistema.



```

public List<ActivityPSS> getAll() {
    logger.trace("Start Method");
    try {
        session = PersistenceUtil.getSession();
        Query query = session.createQuery("from ProvActivity");
        List<ActivityPSS> list = query.list();
        session.close();
        return list;
    } catch (Exception ex) {
        logger.error("Unexpected error", ex);
    }
    logger.trace("Ended Method");
    return null;
}

```

Figura 8: Exemplo do JPQL utilizado no sistema.

O padrão *Singleton* é extremamente simples e tem como objetivo garantir que uma classe tenha apenas uma instância e fornece um ponto global de acesso à mesma. Seu exemplo pode ser visto na figura 9.

```

public static ActivityPSSDAO getInstance() {
    if (activityPSSDAO == null) {
        activityPSSDAO = new ActivityPSSDAO();
    }
    return activityPSSDAO;
}

```

Figura 9: Padrão *Singleton* utilizado no sistema.

O último padrão aplicado foi o *Factory Method*, que define uma interface ou classe abstrata para criar um objeto e permite adiar a instanciação para subclasses. O uso do padrão pode ser visto nas figuras 10 e 11 respectivamente.

```

public abstract class PersistenceUtil {

    private static SessionFactory sessionFactory;

```

Figura 10: Padrão *Factory Method* utilizado no sistema.

```

    session = (Session) PersistenceUtil.getSession();

```

Figura 11: Instanciação do padrão *Factory Method* utilizado no sistema.

Já no *framework* FProvW3C foi aplicada a *Java Persistence API* (JPA), que descreve uma interface comum para *frameworks* de persistência de dados. A JPA define um meio de mapeamento objeto-relacional para objetos Java. Todas as classes do *framework* pertencem ao mesmo pacote (PROV-DM), sua estrutura e o uso da JPA podem ser vistos nas figuras 12 e 13 respectivamente.



Figura 12: Estrutura do *framework*.

```

@Table(name = "ProvWasDerivedFrom")
public abstract class ProvWasDerivedFrom implements Serializable {

    @Id
    @GeneratedValue
    @Column
    private Integer idwasDerivedFrom;
    @Column
    private String generation;
    @Column
    private String usage;
    @JoinColumn
    @ManyToOne
    private ProvActivity activity;
    @JoinColumn
    @ManyToOne
    private PROV.DM.ProvEntity generatedEntity;
    @JoinColumn
    @ManyToOne
    private PROV.DM.ProvEntity usedEntity;
}

```

Figura 13: Exemplo de notação JPA aplicado no *framework*.

## 2.5 Descrição dos Agentes

Na aplicação foram desenvolvidos 2 agentes. O agente de venda, que possui um catálogo de produtos com os respectivos preços, sendo alguns dos produtos e preços cadastrados durante a iniciação do agente e outros podendo ser adicionados pelo usuário, conforme figura 14.

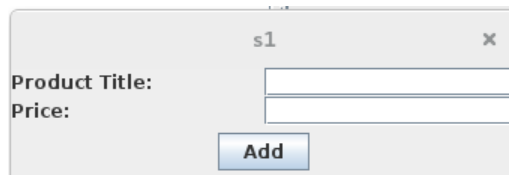


Figura 14: Cadastro de produto e preço no sistema.

Já o agente de comprar recebe como parâmetro na inicialização do sistema o produto que deve comprar, realizando a interação com o agente de vende para cumprir com seu objetivo.

Os agentes são iniciados na instanciação do sistema, conforme figura 15.

```
public static void main(String[] arg) throws StaleProxyException, InterruptedException {  
  
    Logger logger = Logger.getLogger("Main");  
    logger.trace("Start Method");  
    String[] args = {"java", "php", "c"};  
  
    AgentSell s1 = new AgentSell();  
    InitAgent.init(s1, "s1", "Seller");  
    Thread.currentThread().sleep(5000);  
    s1.updateCatalogue("c", 50);  
    s1.updateCatalogue("php", 30);  
    s1.updateCatalogue("java", 80);  
  
    AgentSell s2 = new AgentSell();  
    InitAgent.init(s2, "s2", "Seller2");  
    Thread.currentThread().sleep(10000);  
    s2.updateCatalogue("c", 50);  
    s2.updateCatalogue("php", 80);  
    s2.updateCatalogue("java", 50);  
  
    AgentBuy b1 = new AgentBuy();  
    b1.setArguments(args);  
    InitAgent.init(b1, "b1", "Buyer1");  
    Thread.currentThread().sleep(15000);  
  
    AgentBuy b2 = new AgentBuy();  
    b2.setArguments(args);  
    InitAgent.init(b2, "b2", "Buyer2");  
  
    logger.trace("Ended Method");  
}
```

Figura 15: Iniciação dos agentes no sistema.

O sistema apesar de possuir dois agentes desenvolvidos, um de compra e um de venda, permite instanciar quando os agentes foram requisitados, conforme a Figura 15.

Os agentes de venda trabalham como uma estratégia para cálculo do valor do produto baseado no valor definido na inicialização. Essa estratégia consiste em baixar o preço em 10% caso o agente não tenha realizado a venda para o agente de compra na negociação e aumentar em 20 % caso a venda tenha sido realizada com sucesso. O trecho de código dessas funções pode ser visto na figura 16.

```

if (price != null && price > 0) {
    reply.setPerformative(ACLMessage.PROPOSE);
    reply.setContent(String.valueOf(price.intValue()));
    if (price < 200) {
        int nprice = (int) (price * 1.20);
        catalogue.replace(title, price, nprice);
    } else {
        int nprice = (int) (price * 0.90);
        catalogue.replace(title, price, nprice);
    }
} else {
    reply.setPerformative(ACLMessage.REFUSE);
    reply.setContent("not-available");
}
}

```

Figura 16: Renegociação de preços dos agentes no sistema.

### 3 Modelos

#### 3.1 Análise do Domínio

##### 3.1.1 Modelo de Features

O modelo de *Features* (figura 17) descreve a estrutura do PROV-O (LEBO *et al.*, 2013) representado na figura 18.

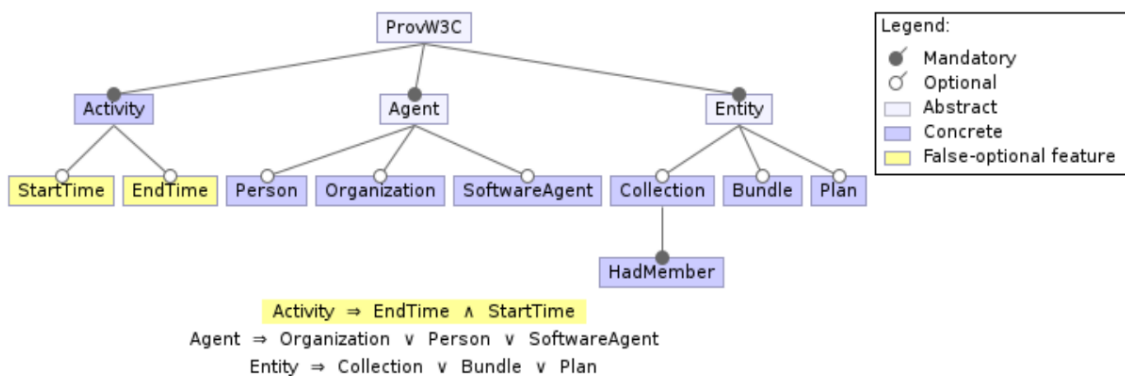


Figura 17: Modelo de *Features* do *framework*.

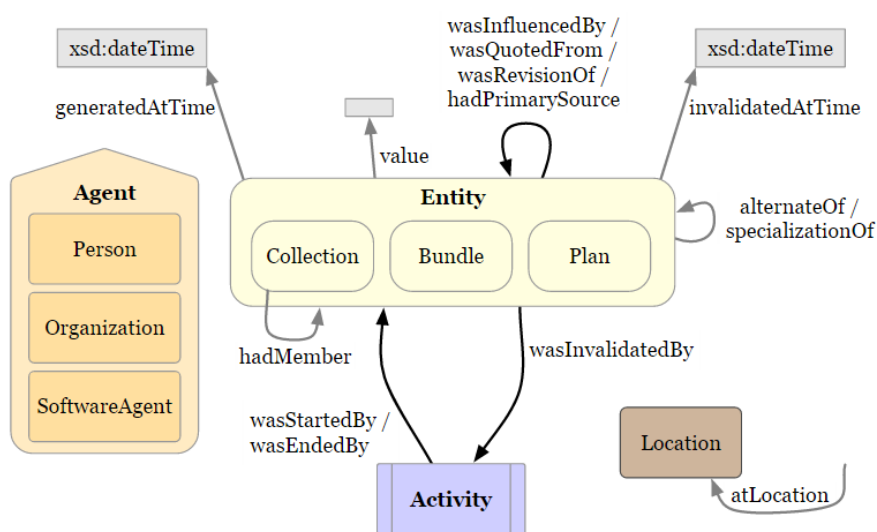


Figura 18: Modelo PROV-O.

### 3.1.1.1 Casos de Uso

#### 3.1.1.1.1 Lista de atores

- Agentes de software: Os agentes de software são atores no sistema por realizarem as ações de compra e vende de produtos de forma autônoma;
- Usuário: As pessoas que fazem uso do sistema, tem como papel o cadastro dos produtos de cada agente de venda e podem fazer consulta dos dados de proveniência capturados ao longo da execução do sistema.

#### 3.1.1.1.2 Diagramas

O diagrama de caso de uso representando a aplicação pode ser visto na figura 19.

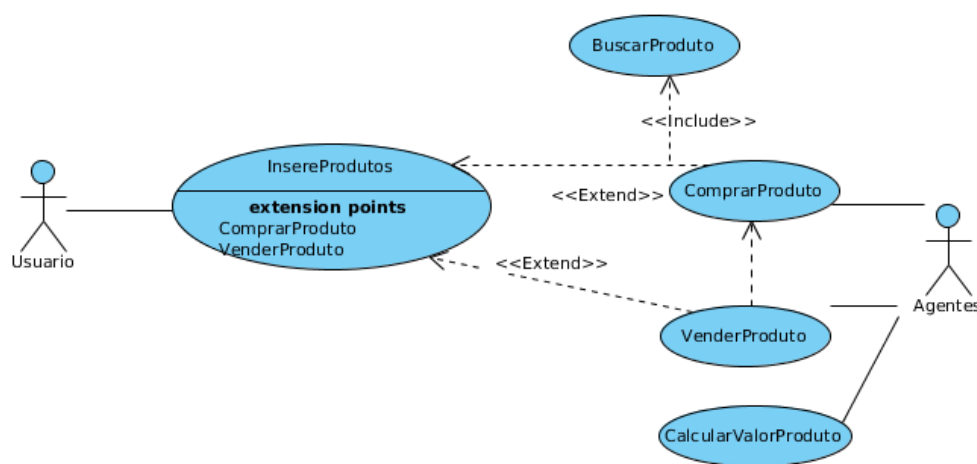


Figura 19: Diagrama de Caso de Uso da aplicação utilizando o *framework*.

#### 3.1.1.1.3 Detalhamento

Caso de uso 01

Nome
Cadastro de produto
Descrição
Usuário cadastra os produtos com os valores dos agentes de venda
Atores
Usuário e agente de venda
Pré-condições
Não existe
Fluxo Básico
<ol style="list-style-type: none"><li>1. O agente de venda solicita o cadastro dos produtos</li><li>2. O usuário informa o produto e o valor</li><li>3. O usuário clica em cadastrar</li><li>4. O agente de venda registra o produto</li></ol>
Regras de negócio
RN 01: O produto deve possui valor entre 1 e 200 unidades monetárias

#### Caso de uso 02

Nome
Compra de produto
Descrição
O agente de compra busca os agentes vendedores e realiza a compra do produto
Atores
Agente de compra e agente de venda
Pré-condições
O agente de venda possui o produto solicitado
Fluxo Básico
<ol style="list-style-type: none"> <li>1. O agente de compra busca os agentes de venda disponíveis no sistema</li> <li>2. O sistema retorna todos os agentes de venda</li> <li>3. O agente de compra informa o produto solicitado para os agentes de venda</li> <li>4. Os agentes de venda informam se possuem o produto e seu respectivo valor</li> <li>5. O agente de compra efetua a negociação com o agente de venda que apresenta o produto disponível com o menor valor</li> <li>6. Todos os agentes atualizam os valores de seus produtos baseados na descrição do tópico 3.4 apresentado anteriormente</li> </ol>
Regras de negócio
RN 02: O agente de compra deve informar qual o produto solicitado
RN 03: O agente de venda deve possuir o produto solicitado

#### Caso de uso 03

Nome
Venda de produto
Descrição
O agente de venda oferece o produto para os agentes de compra
Atores
Agente de venda e agente de compra
Pré-condições
O agente de venda possui produtos para oferecer
Fluxo Básico
<ol style="list-style-type: none"> <li>1. O agente de venda busca os agentes de compra disponíveis no sistema</li> <li>2. O sistema retorna todos os agentes de compra</li> <li>3. O agente de compra oferece os produtos para os agentes de compra</li> <li>4. O agente de compra determina se aceita ou rejeita a negociação</li> <li>5. O agente de compra efetua a negociação com o agente de venda que ofereceu o produto</li> <li>6. Todos os agentes atualizam os valores de seus produtos baseados na descrição do tópico 3.4 apresentado anteriormente</li> </ol>
Regras de negócio
RN 04: O agente de venda deve possuir produtos para venda
RN 05: O agente de compra pode aceitar ou rejeitar a compra

#### Caso de uso 04

Nome
Lista de ações executadas no sistema
Descrição
Consulta todas as operações realizadas pelas agentes no sistema

Atores
Usuário
Pré-condições
Não existe
Fluxo Básico
<ol style="list-style-type: none"> <li>1. O usuário seleciona o menu superior de consulta</li> <li>2. O usuário seleciona a opção desejada no menu de consulta</li> <li>3. O sistema exibi o histórico registrado pelo <i>framework</i> FProvW3C</li> </ol>
Regras de negócio
RN 06: O sistema deve permitir a rastreabilidade dos agentes do sistema

### 3.1.2 Projeto do Domínio

#### 3.1.2.1 Classe

O diagrama de classe representando a aplicação pode ser visto na figura 20.

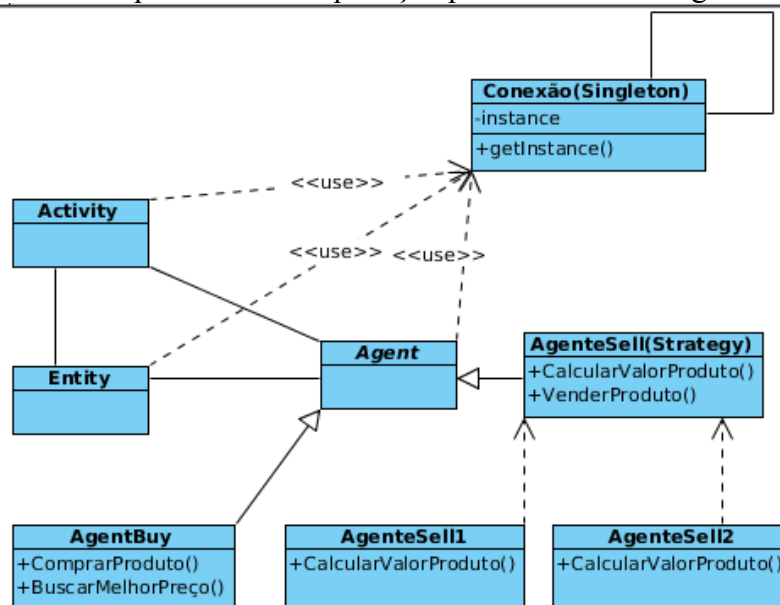


Figura 20: Diagrama de Classes da aplicação utilizando o *framework*.

#### 3.1.2.2 Sequências

O diagrama de sequência representando a interação do usuário na aplicação pode ser visto na figura 21.

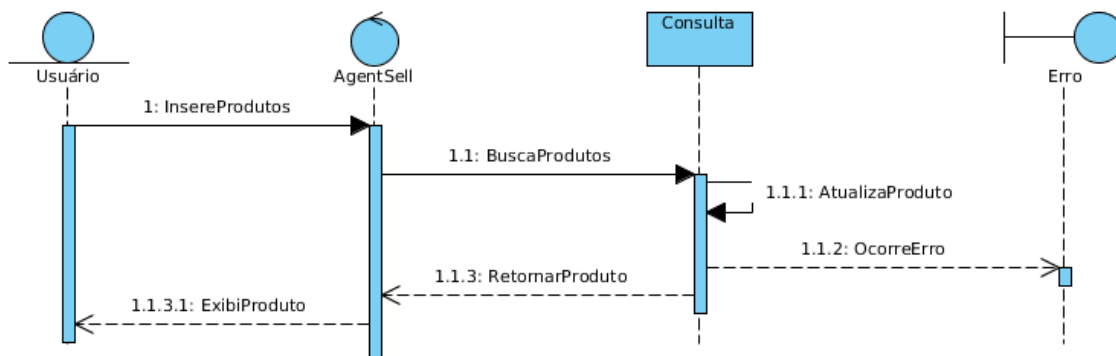


Figura 21: Diagrama de sequência do usuário.

Este diagrama (figura 21) representa a interação do usuário no sistema, onde o Usuário cadastra cada produto com o respectivo preço no sistema. Caso o produto já exista o preço é atualizado.

Já a figura 22 apresenta o diagrama de sequência do agente de compra, onde o mesmo faz a busca pelos agentes de vende, ao ser retornado a lista com os agentes com o produto solicitado, o mesmo efetua a compra com o agente que apresenta o melhor valor e todos os agentes de venda fazem a atualização o valor da venda do produto.

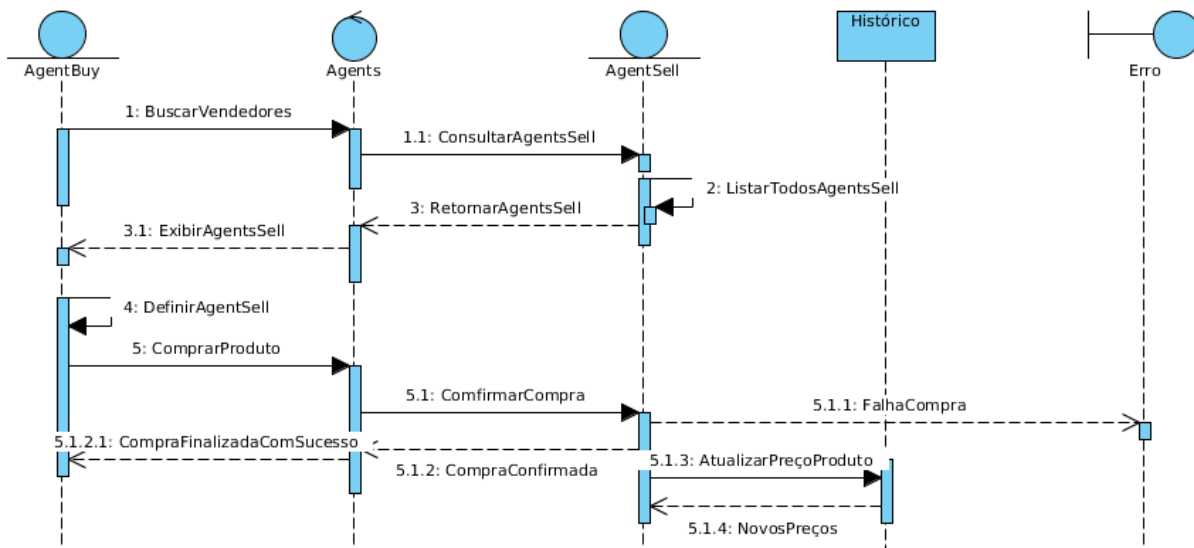


Figura 22: Diagrama de sequência do agente de compra buscando vendedor.

O diagrama de sequência do agente do venda pode ser visto na figura 23. O agente de venda busca todos os agentes de compra conectados ao sistema e oferece os produtos aos mesmo, caso a venda seja aceita pelo agente de compra e iniciado o diagrama de sequência disponível na figura 24.

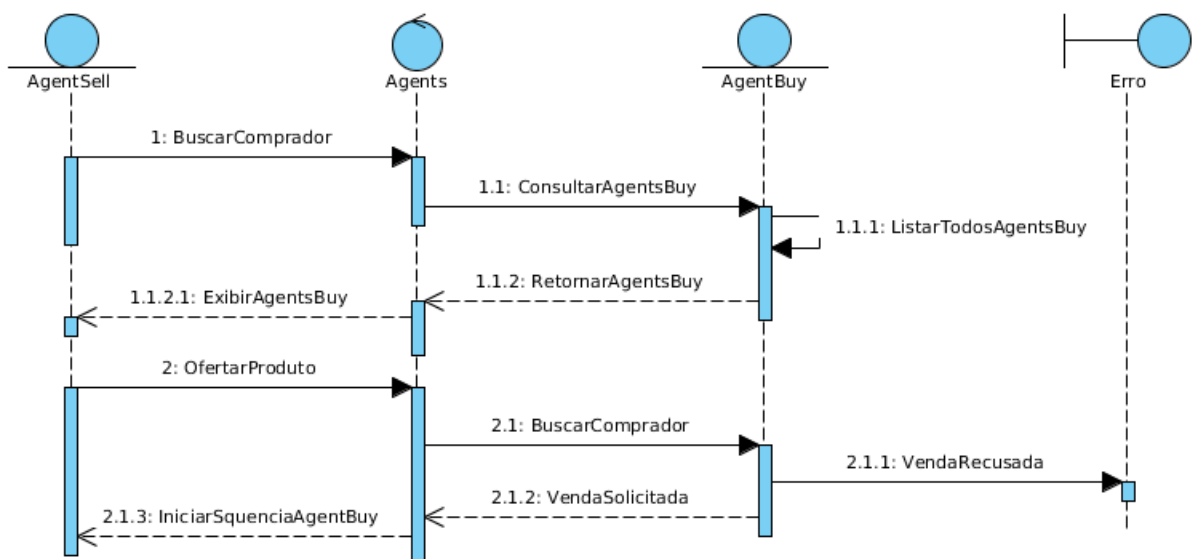


Figura 23: Diagrama de sequência do agente de venda buscando comprador.



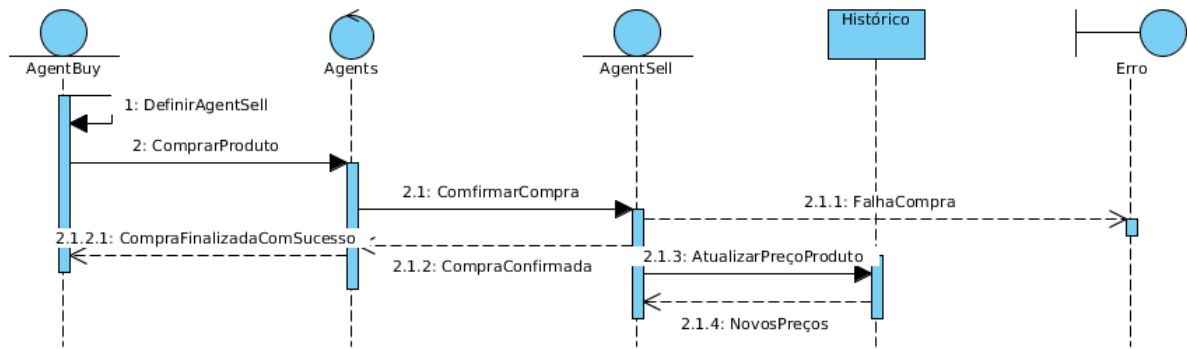


Figura 24: Diagrama de sequência do agente de compra efetivando a venda.

A figura 24 apresenta o diagrama de sequência de continuidade do digrama 24, onde o agente de compra vai efetiar a compra baseado no agente de vende que ofereceu o produto requisitado. Após a conclusão da operação o agente de venda atualiza o preço do produto para a próxima venda.

## 4 Descrição da aplicabilidade do *Framework*

### 4.1 Guia de Instalação e Configuração

- 1) Configure o arquivo “hibernate.cfg.xml” que encontra dentro de “Pacotes de Código-fonte”, conforme figura 25.

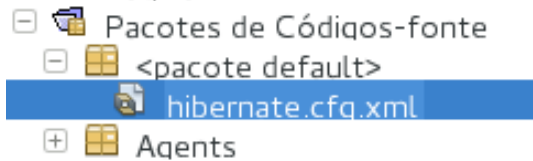


Figura 25: Local de configuração do arquivo “hibernate.cfg.xml”.

- 2) Altere no arquivo “hibernate.cfg.xml”, substituindo as linhas “url”, “username” e “password” por um banco de dados válido com seu respectivo usuário e senha de acesso, conforme a figura 26.

```

<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/sma</property>
    <property name="connection.username">sma</property>
    <property name="connection.password">sma2017</property>
    <property name="connection.pool_size">1</property>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="current_session_context_class">thread</property>
    <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    <property name="show_sql">>false</property>
    <property name="hbm2ddl.auto">validate</property>
  
```

Figura 26: Configuração do acesso ao banco de dados no arquivo “hibernate.cfg.xml”.

- 3) Agora execute a aplicação Java Web e posteriormente o arquivo “Main.java” que encontra-se dentro do pacote “Guy”, conforme figura 27.

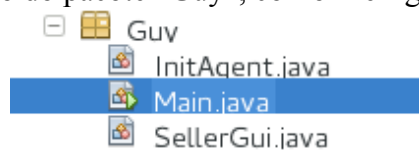


Figura 27: Arquivo de iniciação dos agentes.

- 4) Serão apresentadas duas interfaces de cadastro de produto, conforme figura 14. Informe os produtos e valores de cada item. Ex: Java, php, c, python.
- 5) Os agentes de compra estão configurados para buscar pelos produtos listados no exemplo anterior, essa configuração pode ser alterada no arquivo “Main.java”, especificamente na linha da variável “args”, conforme figura 28.
- 6) Feito esta configuração o sistema encontra-se executando dois agentes de compra e dois agentes de venda e todas as ações dos agentes estão sendo registradas no banco por meio da coleta de proveniência de dados com uso do FProvW3C.

```

public static void main(String[] arg) throws StaleProxyException, InterruptedException {
    Logger logger = Logger.getLogger("Main");
    logger.trace("Start Method");
    String[] args = {"java", "php", "c", "python"};

    AgentSell s1 = new AgentSell();
    InitAgent.init(s1, "s1", "Seller");
    Thread.currentThread().sleep(5000);
    s1.updateCatalogue("c", 50);
    s1.updateCatalogue("php", 30);
    s1.updateCatalogue("java", 80);

    AgentSell s2 = new AgentSell();
    InitAgent.init(s2, "s2", "Seller2");
    Thread.currentThread().sleep(10000);
    s2.updateCatalogue("c", 50);
    s2.updateCatalogue("python", 80);
    s2.updateCatalogue("java", 50);

    AgentBuy b1 = new AgentBuy();
    b1.setArguments(args);
    InitAgent.init(b1, "b1", "Buyer1");
    Thread.currentThread().sleep(15000);

    AgentBuy b2 = new AgentBuy();
    b2.setArguments(args);
    InitAgent.init(b2, "b2", "Buyer2");

    logger.trace("Ended Method");
}

```

Figura 28: Configuração dos produtos a serem buscados pelos agentes de compra.

## 4.2 Descrição da instanciação de produtos

Ao executar os passos descritos no item 4.1, foi aberto o sistema no browser apresentando uma tela conforme a figura 29. No menu superior encontra-se o menu de consultas (figura 30), onde o sistema permite visualizar o histórico de ações dos agentes registrados no sistema, com base na opção desejada.

Um exemplo do registro histórico das atividades executadas pelos agentes pode ser visualizado na figura 31.



Figura 29: Tela inicial do sistema.

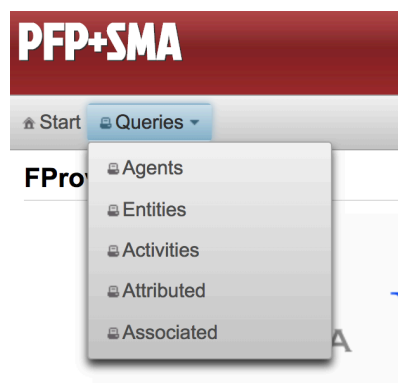


Figura 30: Menu de consultas das ações dos agentes.

ID Activity	Description	Start Time	End Time
1	The Agentb1@192.168.0.102:1099/JADE send the ACLMessage to all sellers	2017-11-18 10:27:30.0	2017-11-18 10:27:30.0
2	The Agent b1@192.168.0.102:1099/JADE receive all proposals/refusals from seller agents	2017-11-18 10:27:31.0	2017-11-18 10:27:31.0
3	The Agent b1@192.168.0.102:1099/JADE receive all proposals/refusals from seller agents	2017-11-18 10:27:31.0	2017-11-18 10:27:31.0
4	The Agentb1@192.168.0.102:1099/JADE send the purchase order to the seller that provided the best offer	2017-11-18 10:27:31.0	2017-11-18 10:27:31.0
5	The Agent Buyerb1@192.168.0.102:1099/JADEReceive the purchase order reply	2017-11-18 10:27:31.0	2017-11-18 10:27:31.0

Figura 31: Listagem das atividades executadas pelos agentes do sistema.

## 5 Teste

Para demonstrar a consistência do framework em armazenar os dados no banco, foram criados 6 testes para as classes ActivityPSS, AgentPSS, EntityPSS, WasAssociatedWith e WasAttributedTo, representados nas firas de 32 à 36.

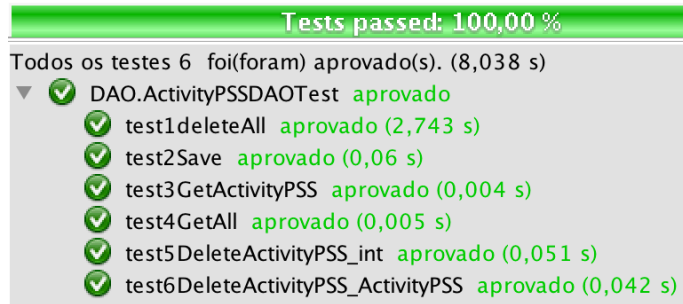


Figura 32: Teste de persistência na classe *ActivityPSS*.

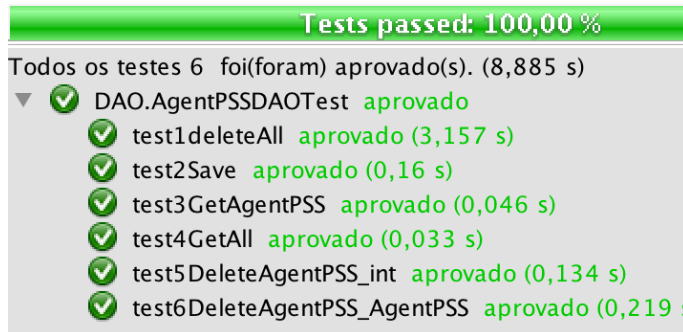


Figura 33: Teste de persistência na classe *AgentPSS*.

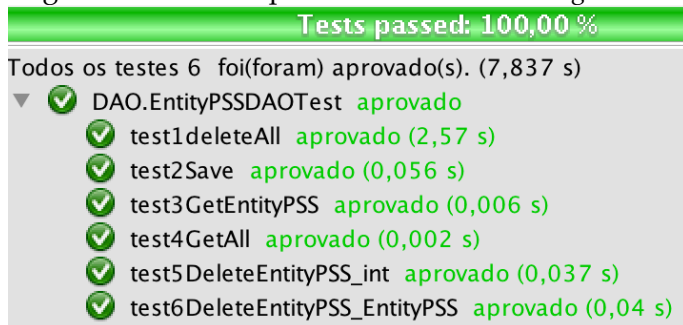


Figura 34: Teste de persistência na classe *EntityPSS*.

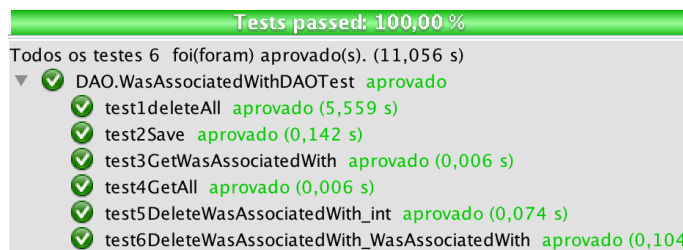


Figura 35: Teste de persistência na classe *WasAssociatedWith*.

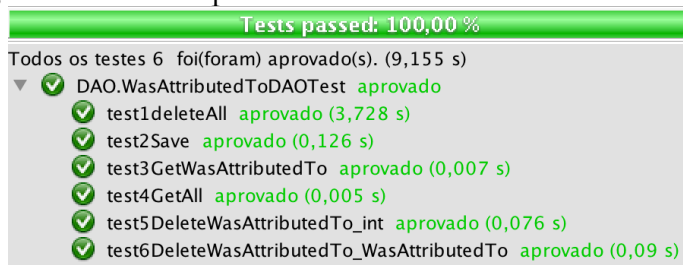


Figura 36: Teste de persistência na classe *WasAttributedTo*.

## 6 Considerações finais

A proveniência de dados permite criar uma base história de acontecimentos do sistema. Essa base pode contribuir para o entendimento das ações tomadas pelos agentes e pelos usuários, bem como rastrear ações e obter resultados utilizando mineração de dados e máquinas de inferência.

A proveniência de dados ainda é algo pouco explorado em SMA, o que permite amplas pesquisas e formação de uma nova base do assunto.

Como resultado até o momento com o uso do *framework* temos um artigo (Sirqueira et al., 2017) e dois relatórios técnicos (Sirqueira et al., 2017a, Sirqueira et al., 2017b), que estão submetidos a revistas.

Como objetivo futuro, pretende-se aplicar o *framework* ao BDI4JADE (Nunes et al., 2011), para captura dos dados de proveniência em SMA. Também estão sendo consideradas outras formas de capturas menos intrusivas a plataforma, mantendo a utilização do *framework*.

O *framework2* e a aplicação<sup>3</sup> encontram-se disponíveis no GitHub.

## Referências

LEBO, Timothy et al. Prov-o: The prov ontology. **W3C recommendation**, v. 30, 2013.

MISSIER, Paolo; BELHAJJAME, Khalid; CHENEY, James. The W3C PROV family of specifications for modelling provenance metadata. In: **Proceedings of the 16th International Conference on Extending Database Technology**. ACM, 2013. p. 773-776.

MOREAU, Luc et al. The open provenance model: An overview. In: **International Provenance and Annotation Workshop**. Springer Berlin Heidelberg, 2008. p. 323-326.

NUNES, Ingrid et al. Transparent Provenance Derivation for User Decisions. In: **IPAW**. 2012. p. 111-125.

NUNES, Ingrid; LUCENA, C. J. P. D.; LUCK, Michael. BDI4JADE: a BDI layer on top of JADE. **ProMAS 2011**, p. 88-103, 2011.

PRICE, Rosanne; SHANKS, Graeme. The impact of data quality tags on decision-making outcomes and process. **Journal of the Association for Information Systems**, v. 12, n. 4, p. 323, 2011.

SIRQUEIRA, Tassio et al. A Software Framework for Data Provenance. In: **29th International Conference on Software Engineering & Knowledge Engineering (SEKE'2017)**. SEKE/Knowledge Systems Institute. 2017. p. 615-619.

SIRQUEIRA, Tassio Ferenzini Martins; VIANA, Marx Leles; DE LUCENA, Carlos José Pereira. Proveniência de Dados em Sistemas Multiagentes. **Monografias em Ciência da Computação – PUC-Rio**. 2017. p. 11.a

SIRQUEIRA, Tassio Ferenzini Martins; VIANA, Marx Leles; DE LUCENA, Carlos José Pereira. Capturando Proveniência de Dados em Sistemas Multiagentes. **Monografias em Ciência da Computação – PUC-Rio**. 2017. p. 10.b

TRAVASSOS, Guilherme H. From Silver Bullets to Philosophers' Stones: Who Wants to Be Just an Empiricist?. In: **Empirical Software Engineering Issues. Critical Assessment and Future Directions**. Springer Berlin Heidelberg, 2007. p. 39-39.

---

2 FProvW3C: <https://github.com/tassioferenzini/FrameworkFProvW3C>

3 PFP+SMA: <https://github.com/tassioferenzini/PFP-SMA>