HOLOS

ISSN: 1518-1634

holos@ifrn.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Brasil

Medeiros, H. B.; Souza Neto, P. A.; Hallais Neto, R. S.

PLUGIN EXTRATOR PARA VERIFICAÇÃO DE COMPOSIÇÃO PEWS

HOLOS, vol. 3, 2012, pp. 84-106

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Natal, Brasil

HOLOS
ISSN 1807 - 1600

PLUGIN EXTRATOR PARA VERIFICAÇÃO DE COMPOSIÇÃO PEWS

H. B. Medeiros[1], P. A. Souza Neto[1], R. S. Hallais Neto[1]
[1]Instituto Federal do Rio Grande do Norte
handersonmedeiros@gmail.com, placido.neto@gmail.com, robertohneto@gmail.com

Artigo submetido em agosto/2011 e aceito em julho/2012

RESUMO

Os serviços Web se baseiam em troca de mensagens como forma de integração de sistemas distintos que independem de protocolo de comunicação, plataforma e linguagem de programação. O crescimento da quantidade de componentes de software disponíveis na forma de serviços Web faz com que novos sistemas possam ser concebidos a partir da composição dos diversos serviços disponíveis, pois muitas vezes um só serviço não atende às requisições dos usuários, fazendo surgir o SOA - arquitetura orientada a serviços. A linguagem PEWS (Predicate path-Expression for Web Services) é uma linguagem utilizada para especificar a ordem e as restrições condicionais em que as operações de serviços web são executadas A proposta desse trabalho é a da criação de um plugin Extrator que faça parte do PEWS Editor, tendo como principal função, verificar se a relação ente as operações e serviços usadas na composição pela ferramenta front-end são válidas. Após implementação, análise e validação o plugin verificou com sucesso as operações do estudo de caso e que é importante que exista essa verificação, pois, saber se o serviço utilizado na composição realmente disponibiliza aquela operação demostra que a estrutura da composição não está comprometida com operações inexistentes nos serviços.

PALAVRAS-CHAVE: Serviços WEB, Composição de Serviços, Verificação de métodos, PEWS.

PEWS TOOL EXTENSION FOR VERIFICATION OF WEB SERVICES COMPOSITION

ABSTRACT

Web services are based on message exchange as a means of integrating disparate systems that are independent of communication protocol, platform and programming language. The growing number of software components available as Web services makes new systems can be designed from the composition of the various services available, because often one service does not meet user requirements, giving rise to SOA - Architecture service-oriented. The language pews (Predicate path-Expression for Web Services) is a language used to specify the order and conditional restrictions on which web service operations are performed. The purpose of this work is the creation of an Extractor plugin that is part of the pews Editor, with the primary function, verify that the relation between the operations and services used in the composition by front-end tool are valid. After implementation, analysis and validation we concluded that the plugin has successfully verified the operations of the case study and it is important to have this check, because, whether the service actually used in the composition provides that operation demonstrates that the structure of the composition is not committed to operations in non-existent services.

KEY-WORDS: WEB Services, Service Composition, Verification methods, PEWS.

PLUGIN EXTRATOR PARA VERIFICAÇÃO DE COMPOSIÇÃO PEWS

INTRODUÇÃO

Os serviços Web são funcionalidades de sistemas de informação disponíveis na Internet e desenvolvidos utilizando diferentes linguagens. Essa tecnologia surgiu como proposta de resolver os problemas na interoperabilidade de sistemas heterogêneos. Tais serviços surgiram como um conjunto especificações de padrões de sistemas abertos baseados em troca de mensagens utilizando documentos XML e é independente de protocolo de comunicação, plataforma e linguagem de programação.

A interoperabilidade de sistemas heterogêneos é realizada através do encapsulamento de API's de serviços a serem integrados. Especificações que incluem linguagens como SOAP, WSDL e UDDI estabelecem padrões para a representação e troca de mensagens acessadas por serviços remotos.

À medida que cresce a quantidade de componentes de software disponibilizados na forma de serviços web, novos sistemas podem ser construídos a partir da composição dos diversos serviços disponíveis. SOA, Arquitetura Orientada a Serviços (*Service Oriented Architecture*) é uma arquitetura baseada em serviços. De acordo com (SAMPAIO, 2006) seu paradigma objetiva criar módulos funcionais chamados de serviços, com baixo acoplamento que permitem a reutilização de código.

A arquitetura SOA foi proposta no intuito da interoperabilidade de sistemas por meio de um conjunto de interfaces de serviços (API's), transformando aplicações monolíticas rígidas em componente de softwares flexíveis. Esses componentes não necessitam de detalhes técnicos da plataforma dos outros serviços para a troca de informações a serem realizadas.

Os aplicativos e rotinas são disponibilizados como serviços em uma rede de computadores (Internet ou Intranet), permitindo uma comunicação por meio de padrões abertos. Por esse motivo, os serviços web são recursos bastante utilizados para a criação dessa estrutura.

A arquitetura SOA básica consiste de um serviço consumidor e um provedor de serviços, onde um consumidor faz uma requisição para um provedor de serviços, que responde com o resultado para o consumidor; o provedor de serviços pode também ser um consumidor de serviços.

Muitas vezes um só serviço não atende às requisições dos usuários, sendo assim, torna-se necessário a composição de diversos serviços Web com finalidade de atingir um objetivo em particular. Seguindo esse conceito, aplicações completas poderiam ser tratadas como uma junção de diversos serviços e, não mais, serem escritas individualmente, implicando num aprimoramento da reusabilidade dos serviços Web. Um aspecto valioso nesse modelo de serviços é que novos serviços web podem ser criados a partir da existência de outros, sem desconsiderar o paradigma da arquitetura orientada a serviços, compondo ou recompondo aplicações sem tocar em código já constituído e devidamente testado.

PEWS *Predicate path-Expression for Web Services* é uma linguagem utilizada para especificar a ordem em que as operações de serviços web são executadas. A linguagem pode ser utilizada não apenas na especificação de serviços web simples ou compostos, mas também como uma forma sintática na qual é possível compreender as propriedades dos serviços web Ba et al. (2005a). Dado um conjunto de operações (ou métodos) já implementados, um novo serviço web pode ser implementado seguindo uma descrição do comportamento apresentados.

Potrich (2006) criou o PEWS Editor na forma de um *plugin* para a plataforma Eclipse com o objetivo de facilitar a utilização e edição de composições PEWS, realizando análise léxica, sintática, semântica e por fim, geração de código na versão do PEWS em XML(XPEWS).

As aplicações que utilizam a tecnologia de composições de serviços se tornam dependentes dos serviços usados, pois na maioria dos casos, grande parte das funcionalidades estará nesses serviços. É importante saber se os métodos usados na composição estão relacionados de maneira correta a seus serviços. Caso alguns deles não estiverem operando ou a referência a ele estiver errada, seja por meio de digitação ou confusão na hora de relacioná-lo a um serviço que o fornece, poderá vir a comprometer o funcionamento da aplicação.

Ter uma ferramenta que venha a fazer a validação dos métodos para que sejam identificados quais estão descritos ou não nos serviços web, auxiliaria assim, o desenvolvedor no momento da criação dessas composições.

Diante desse contexto, este trabalho propõe o desenvolvimento, na linguagem Java, do Plugin Extrator, que tem a função de auxiliar o desenvolvedor em sua solução de composição de serviços. Nada garante que as operações descritas nos arquivos PEWS, e posteriormente no XPEWS, estejam realmente declaradas no(s) WSDL(s) do serviço. O Plugin Extrator, tem como função principal conferir se os métodos descritos no arquivo XPEWS realmente estão declarados no(s) WSDL(s) do serviço, garantindo assim a integridade dos métodos utilizados na composição feita na ferramenta *front-end* PEWS Editor através da comparação sintática das operações.

O artigo está estruturado para apresentar uma visão geral de como é o plugin desenvolvido pelo Potrich (2006), mostrando a estrutura e funcionalidades disponibilizadas pelo plugin, bem como, uma limitação encontrada. Na Sessão seguinte é apresentado o Plugin Extrator, mostrando sua estrutura, funções, funcionamento, especificações, modelagem e implementação. Posteriormente, um estudo de caso usando composição de serviços será apresentado a fim de testar o Plugin Extrator. Por fim, a sessão com os resultados dos testes feitos com o plugin para validar as operações usadas no estudo de caso e a sessão de conclusão contendo os objetivos alcançados e trabalhos futuros a serem desenvolvidos no projeto.

HOLOS
ISSN 1807 - 1600

PEWS EDITOR

O PEWS Editor[1] é um *plugin* para a plataforma de desenvolvimento Eclipse na linguagem Java desenvolvido pelo Potrich (2006), que provê melhor análise do uso de instrumento prático da linguagem PEWS, permitindo posteriores contribuições à mesma.

Potrich apresenta em seu trabalho o desenvolvimento de um *front-end* cuja função é facilitar a criação e edição de composições PEWS. Esse *front-end* realiza análise léxica, sintática e semântica das composições, verificação de erros de codificação. Por fim, é gerado um código XPEWS, através de um compilador, aumentando a produtividade do desenvolvedor em relação ao tempo de desenvolvimento das composições. Logo, permitindo maior integração com outras ferramentas, como por exemplo, editores XML e WSDL.

Conforme Potrich (2006) afirma em seu trabalho, a premissa do PEWS editor tem como função a escrita de composições PEWS, através de facilidades como:

- Assistente para criação de novas composições, trazendo uma estrutura mínima para a nova composição;
- Sintaxe colorida, facilitando a identificação de palavras reservadas;
- Análise automática do código da composição, realizada a cada vez que a composição é salva;
- Identificações dos erros encontrados a partir da análise do código, apresentada tanto no editor (diretamente na linha que contém o erro), quanto no visualizador de problemas;
    - Facilitando a busca de erros até em outras composições PEWS (o visualizador informa os erros de composições que não estão abertas no momento);
- Facilidade na geração de código XPEWS, através de um menu de contexto;
- Integração com outras ferramentas disponibilizadas pela plataforma Eclipse, uma vez que o plug-in permite realizar uma composição em qualquer projeto existente (Java, por exemplo).

O PEWS Editor (Figura 1) é composto por:

1. Sintaxe Colorida: O item 2 da Figura 1, mostra os recursos com sintaxe colorida para que sejam visualizados palavras reservadas da linguagem, *strings* e comentários adicionados à composição de maneira diferenciada.
2. Marcação de erros: marcação para facilitar a localização de erros encontrados durante a análise da composição, como se pode ver no item 3 da Figura 1.
3. Menu de contexto: É visualizado ao clicarmos com o botão direito do mouse sobre uma composição PEWS no explorador de pacotes ou sobre um arquivo, como esta sendo exibido na Figura 1 pelo item 1. O menu fica encarregado de executar o compilador para verificar a composição e gerar o arquivo XPEWS.
4. Construtor: Cada vez que os projetos contendo a composição PEWS são alterados ou salvos um construtor é acionado executando o compilador a procura de erros. Havendo a ocorrência, os erros são notificados tanto no editor como no visualizador de problemas como mostra o item 3 da Figura 1.

[1] Disponível para download em http://svn6.assembla.com/svn/pews-project/tool/front-end/

5. Compilador: responsável pelas analises léxicas, sintáticas, semânticas e a tradução do código.

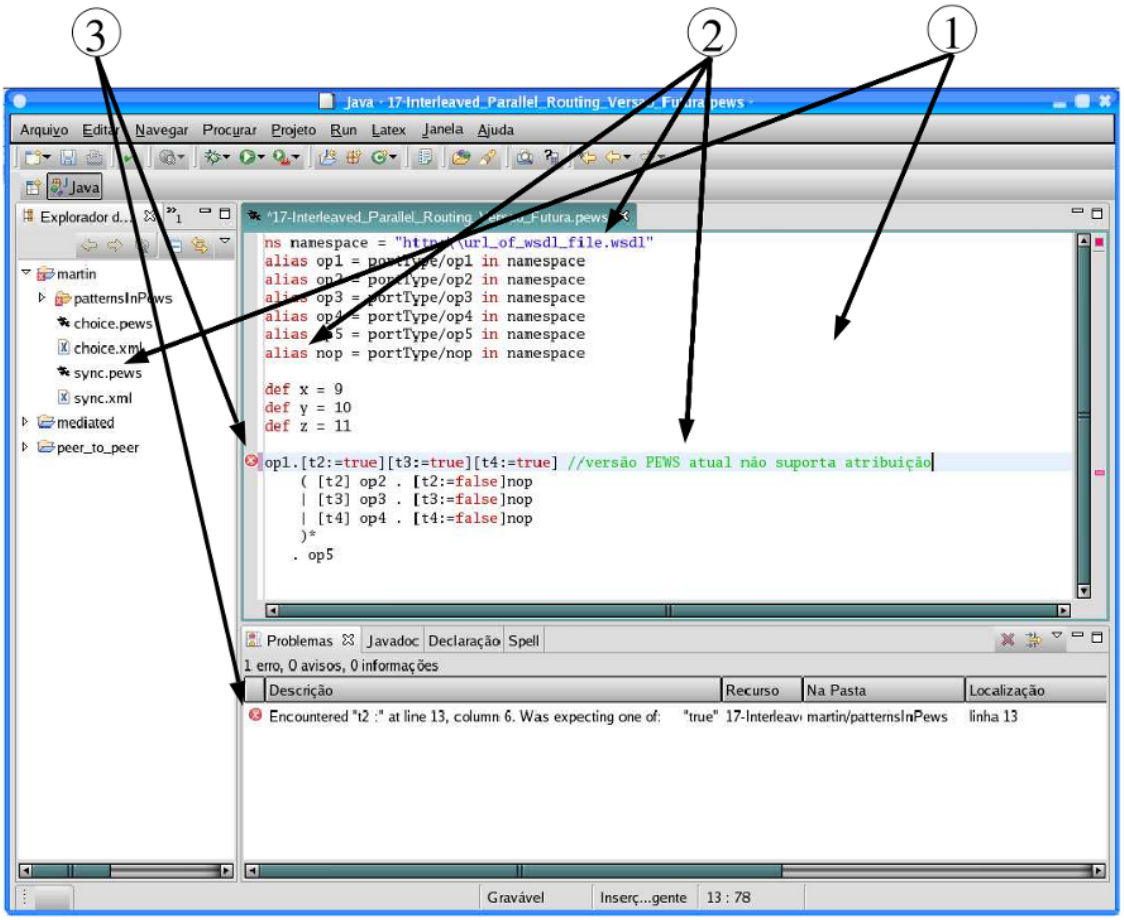

Figura 1: Plugin PEWS Editor

No atual estado PEWS Editor foi percebido uma limitação: pode-se relacionar operações a serviços, sem que a operação pertença ao serviço relacionado, causando assim, problemas de inconsistência e não há uma funcionalidade que indique este tipo de erro.

Essa relação inconsistente pode ser gerada através de uma confusão do desenvolvedor ou, até mesmo, por erro de digitação. Para reparar essa necessidade de verificação, teve início a construção de um *plugin* que faça esse trabalho de forma a vir mostrar ao usuário do *plugin*, se existe ou não relações inconsistentes entre operações e serviços.

PLUGIN ECLIPSE PARA EXTRAÇÃO E COMPARAÇÃO DE DADOS WSDL

O Plugin Extrator surge como uma ferramenta de auxílio para essas composições, facilitando no momento da especificação da composição. Ele verifica a operabilidade e mostra para o desenvolvedor quais serviços e métodos relatados no(s) WSDL(s) da composição estão funcionando, com isso, agregando valor ao Editor PEWS desenvolvido por Potrich (2006).

Esse *plugin* checa a consistência dos *namespaces* e operações dos serviços descritas no(s) WSDL(s) com as operações selecionadas para a composição no arquivo XPEWS. A proposta do *plugin* Extrator é a de que, em qualquer momento da criação ou edição da

composição, ao ser gerado o documento XPEWS, se tenha uma forma de visualizar quais métodos da composição estão descritos em seus respectivos serviços.

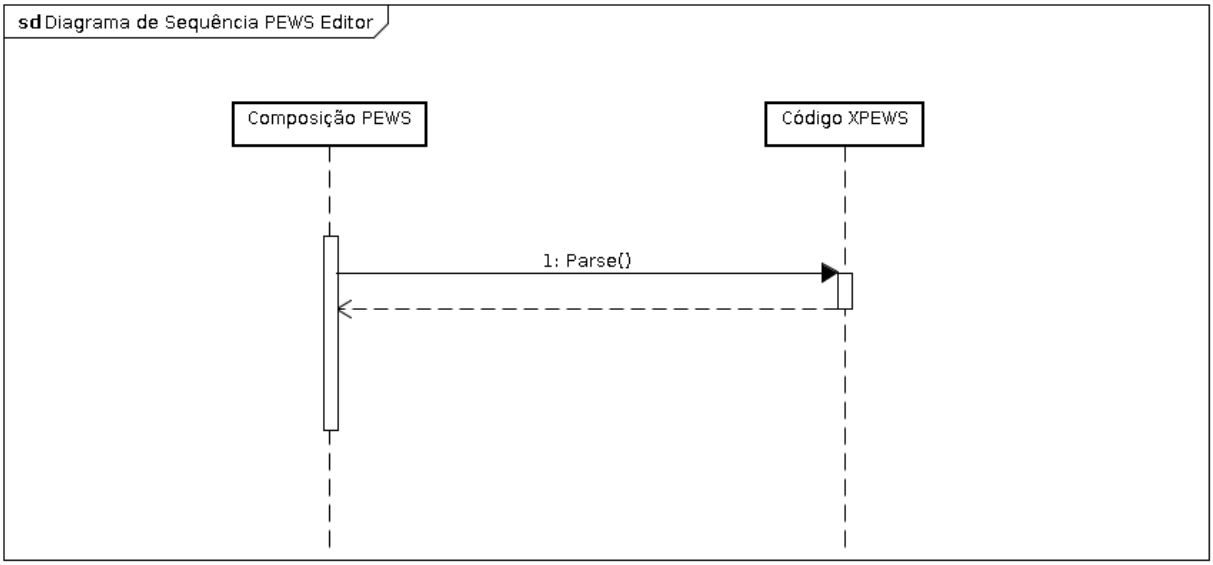

Figura 2: Visualização macro do Diagrama de sequência do PEWS Editor em UML**.**

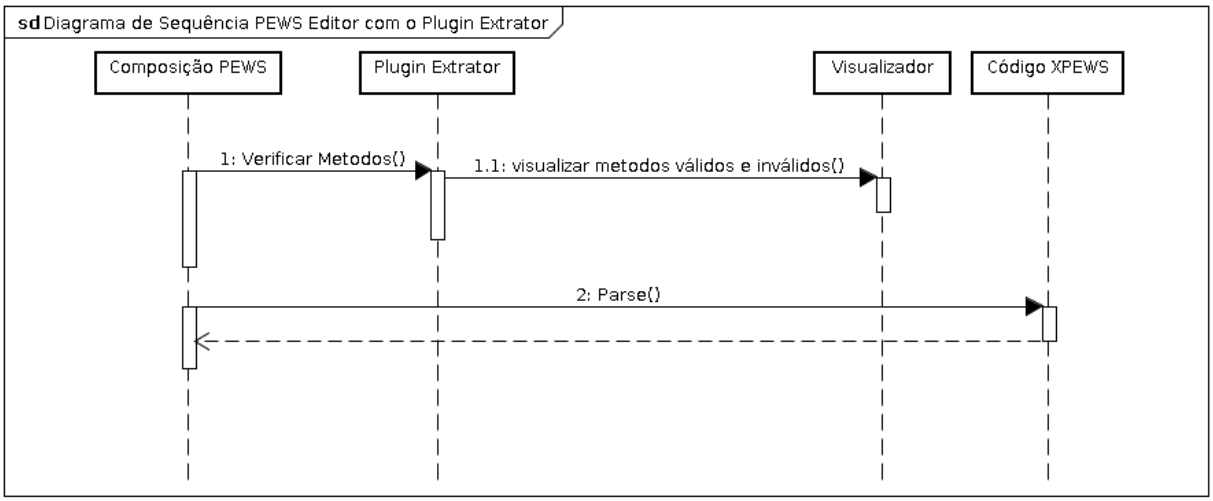

Figura 3: Visualização macro do Diagrama de sequência da proposta pelo Extrator em UML**.**

No estado atual a sequência do PEWS Editor (Figura 2) é feita através da criação da composição e em seguida a geração do código XPEWS sem a verificação das operações. O Plugin Extrator propõem adicionar mais um passo nessa sequência: visualização das operações usadas na composição e verificação de quais estão relacionadas corretamente (Figura 3).

A verificação dos métodos feita pelo Plugin Extrator é realizada pela seguinte forma:

- Captura e leitura do arquivo XPEWS;
- Leitura do(s) WSDL(s) empregado(s) na composição;
- Download do(s) WDSL(s) para a máquina do usuário do *plugin*;
- Utilização do DOM para criação da árvore DOM para que seja possível a manipulação tanto do documento XPEWS quanto do(s) WSDL(s);
- Captura das *operations* dos XPEWS e do(s) WSDL(s);
- Comparação das *operations*;
- Visualização dos métodos (*operations*) utilizados na composição do serviço, com a diferenciação dos que estão ou não descritos no(s) WSDL(s).

Estrutura do Plugin

Em paralelo ao universo dos serviços Web, a plataforma de desenvolvimento Eclipse, lançada no ano 2001, se tornou um dos principais IDE's para integrar diferentes tipos de aplicações. O que torna o Eclipse uma plataforma de desenvolvimento especial, além do fato de ela possuir código-aberto e da grande variedade de *plugins* existentes, é a flexibilidade na qual podem ser combinados seus recursos (*views* e *editors*).

É possível reorganizar o ambiente de desenvolvimento que atenda as características necessárias ao desenvolvedor. O desenvolvimento de *plugins* para o Eclipse estende este conceito de personalização do ambiente de desenvolvimento dando, ao usuário do Eclipse, a possibilidade de criar e publicar *plugins*, com as mais diversas funcionalidades.

O Plugin Extrator utiliza alguns pontos de extensão que a plataforma Eclipse disponibiliza para a criação de sua interface, agregando novas funcionalidades e características. São eles:

- org.eclipse.ui.perspectiveExtensions, para que o *plugin* seja adicionado à perspectiva;
- org.eclipse.ui.views, utilizada para mostrar a adição de uma *view* na perspectiva, apresentando os métodos usados na composição de serviços que estão válidos e inválidos;
- org.eclipse.ui.popupMenus, Menus que são acionados ao clicarem com o botão direito em um determinado componente, chamando as actionsSets;
- org.eclipse.ui.actionSets, as ações executadas através do clique nos pop-ups menus, chamando as ações para a verificação dos métodos;

HOLOS
ISSN 1807 - 1600

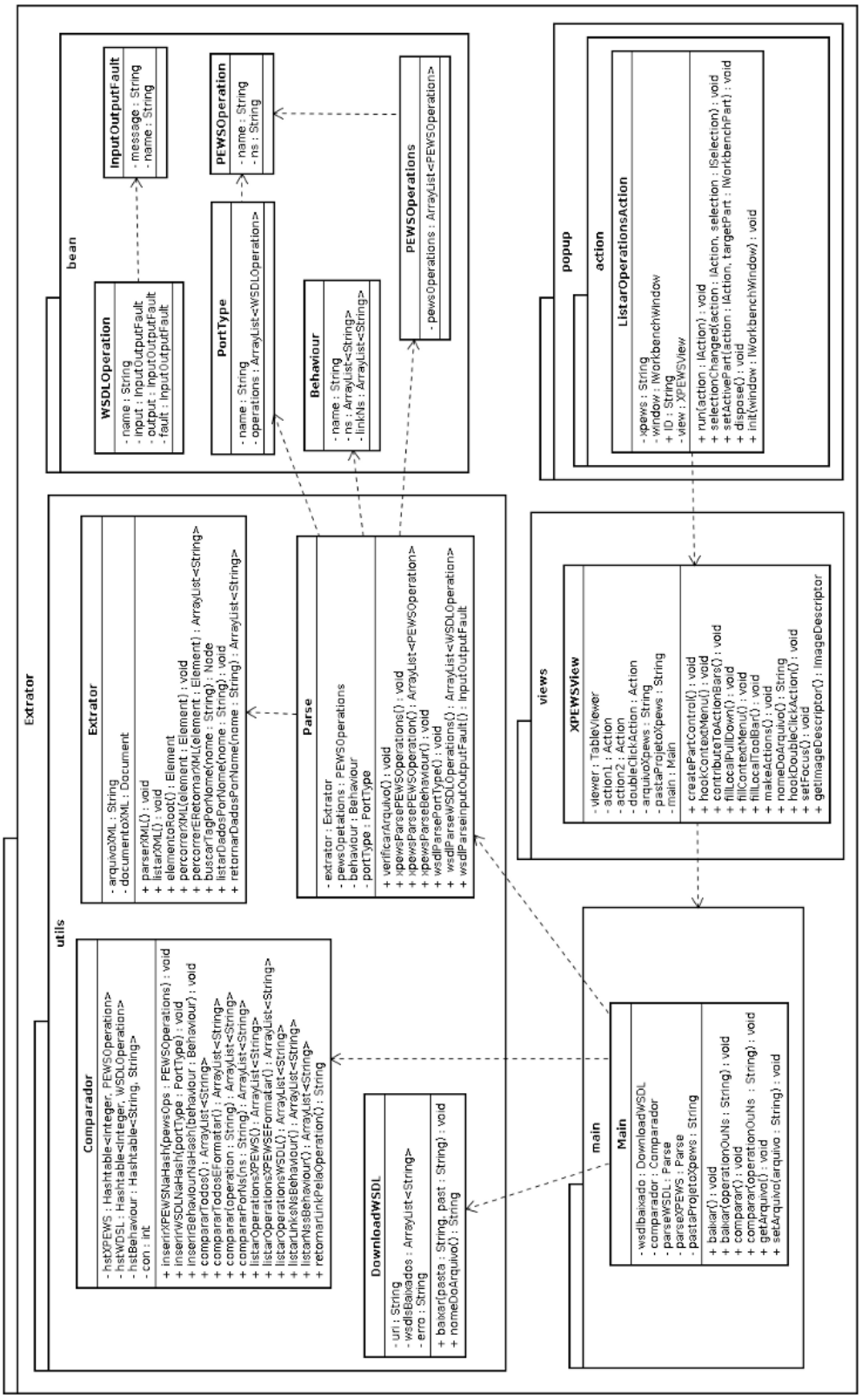

Figura 4: Diagrama de Classe do Plugin Extrator em UML**..**

HOLOS
ISSN 1807 - 1600

A figura 4 mostra através do diagrama de classe como está estruturado o *plugin*. Para ter uma visão mais detalhada dessa estrutura, com suas classes e métodos, pode-se ressaltar:

Download WSDL

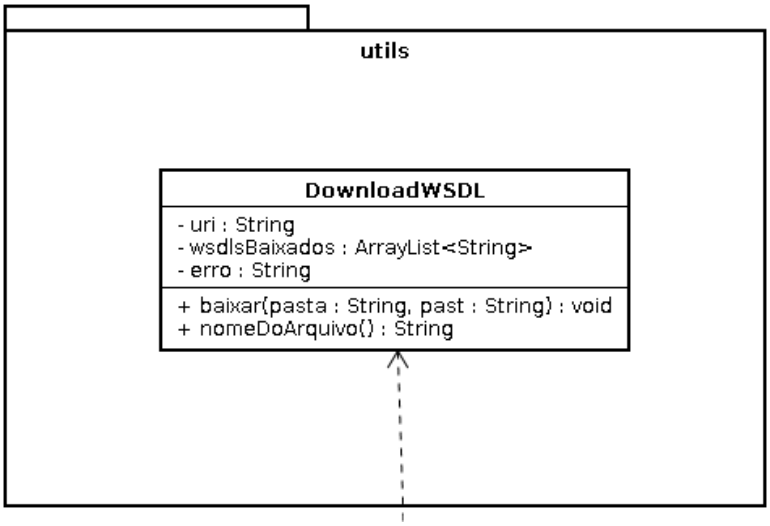

Figura 5: Classe Download WSDL em UML.

A verificação das operações se dá inicialmente através da obtenção do arquivo WSDL disponibilizado pelo serviço. O *download* do WSDL é necessário para que o *plugin* saiba exatamente quais são os métodos disponibilizados.

A classe extrator.utils.DownloadWSDL (Figura 5) é responsável pelo download desse arquivo ao capturar quais serviços irão ser utilizados na composição, o *plugin* baixa o(s) WSDL(s) para a pasta do projeto, para a máquina do usuário que usa o *plugin*, na qual esteja o arquivo XPEWS, bem como, guarda a referência do(s) WSDL(s). Após finalizado o *download*, é realizado o processo de parser no(s) XML(s).

Parse

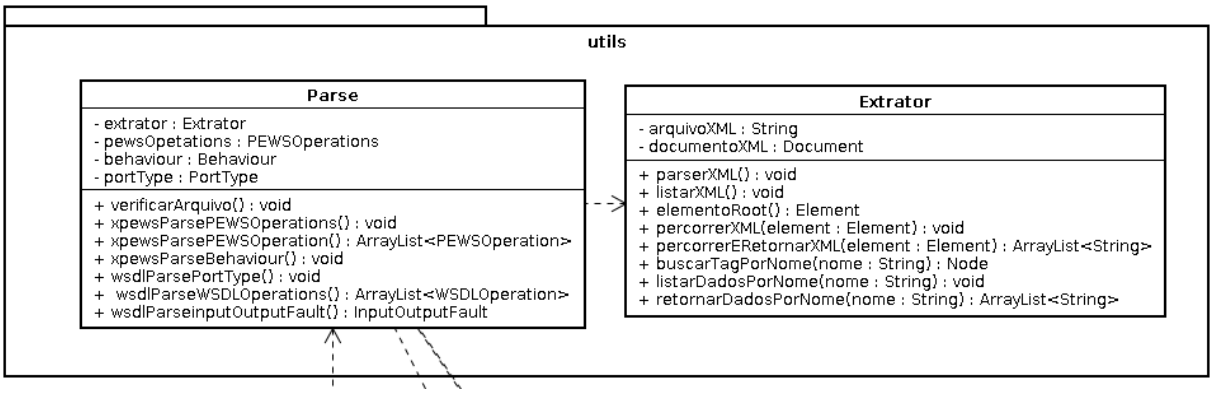

Figura 6: Classe Parser e Extrator em UML..

O Parse é o encarregado de extrair os dados dos arquivos XPEWS e WSDL, os transformando numa arvore através da tecnologia Document Object Model (DOM -

Modelo de Objetos para Documentos) usando a classe extrato.utils.Extrator (Figura 6). O Parse percorre toda a árvore, retorna os elementos pertencentes a um determinado nó especificado pelo desenvolvedor em um arquivo XML. Depois de extraídas as informações, a classe Parse coloca esses dados em seus respectivos objetos, extraindo do XPEWS as informações dos serviços utilizados através do nó *behaviour* e extraindo também os métodos (*operation*) usados na composição e os adicionando na classe "PEWSOperation".

Nos arquivos WSDL, o extrator.utils.Parser (Figura 6) extrai as *operations* adicionando-as na classe WSDLOperation. Tanto as *operations* extraídas do XPEWS como as *operations* extraídas do(s) WSDL(s) serão usadas para que seja feito a comparação mostrando quais das descritas nesses arquivos são comuns aos dois. A partir desse passo, o Plugin Extrator já tem a capacidade para fazer a verificação dos métodos da composição.

Verificação de métodos

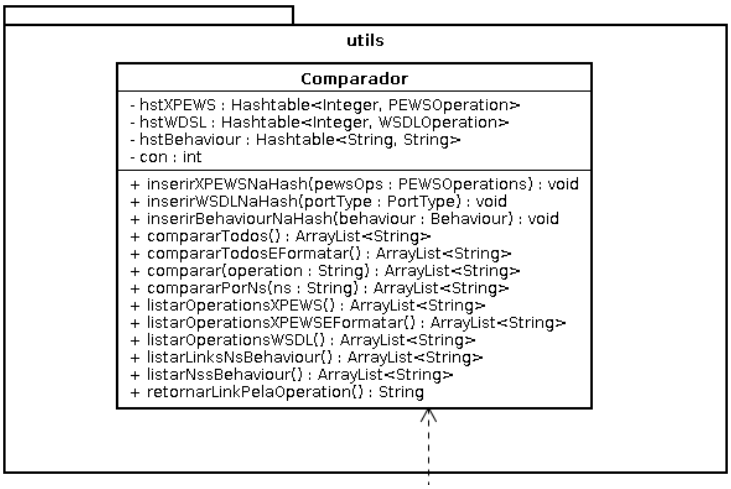

Figura 7: Classe Comparador em UML**.**

A partir da obtenção do(s) WSDL(s) que estão sendo usados na composição, se tem a capacidade de verificar os métodos adotados na mesma. A classe extrator.utils.Comparador (Figura 7) é a classe encarregada de fazer a verificação dos métodos do arquivo XPEWS com os métodos dos serviços descritos no WSDL. As classes "PEWSOperation" e "WSDLOperation" contêm as operações do documento XPEWS e do(s) documento(s) WSDL(s), eles são inseridos em tabelas hashs, para que seja feita a comparação.

Essa comparação feita pelo Plugin Extrator é puramente sintática, mostrando quais estão válidos para serem usados na composição. O *plugin* verifica o nome do método de um determinado serviço no documento XPEWS e analisa o WSDL desse serviço, certificando que o método do XPEWS esteja no WSDL do serviço.

Visualização dos Métodos Listados

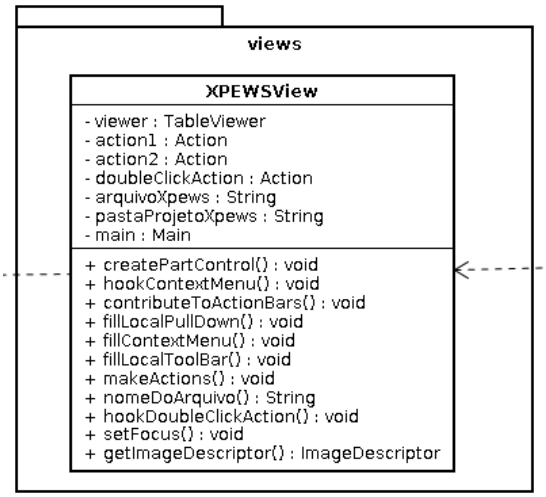

Figura 8: Classe View em UML**..**

Após todos os passos anteriores, uma *view* é instanciada através da classe extrator.utils.XPEWSView (Figura 8) ficando encarregada de apresentar ao usuário que está usando o *plugin*, através de uma mostragem visual, quais são os métodos usados na composição que estão relatados no(s) WSDL(s). Essa mostragem se dá na seguinte forma:

- Criação e abertura de uma view chamada PEWS Operations no *workspace* da plataforma Eclipse, como visualizado no item 1 da Figura 9;
- Visualização de todos os métodos usados na composição, separados em grupos;
    - Cada grupo se refere ao(s) serviços(s) utilizado(s) na composição, item 2 da Figura 9;
    - Os métodos que estão descritos no XPEWS e que estão sinalizados com um ícone com a letra “V” indicam que estes estão descritos no(s) WSDL(s), item 3 da Figura 9;
    - Os métodos que estão descritos no XPEWS e que estão sinalizados com um ícone com a letra “X” indicam que estes não estão descritos no(s) WSDL(s), item 4 da Figura 9.

A *view* também disponibiliza atalhos para que determinadas ações sejam chamadas:

- O item 5 da Figura 9 faz novamente a verificação dos métodos do arquivo XPEWS abertos na PEWS Operation view.
- O item 6 da Figura 9 mostrar apenas listagem dos métodos usados na composição.
- O item 7 da Figura 9 é o menu acessível a partir do clique com o botão direito na view PEWS Operations, mostrando as opções de visualização ou listagem dos métodos utilizados na composição dos serviços.

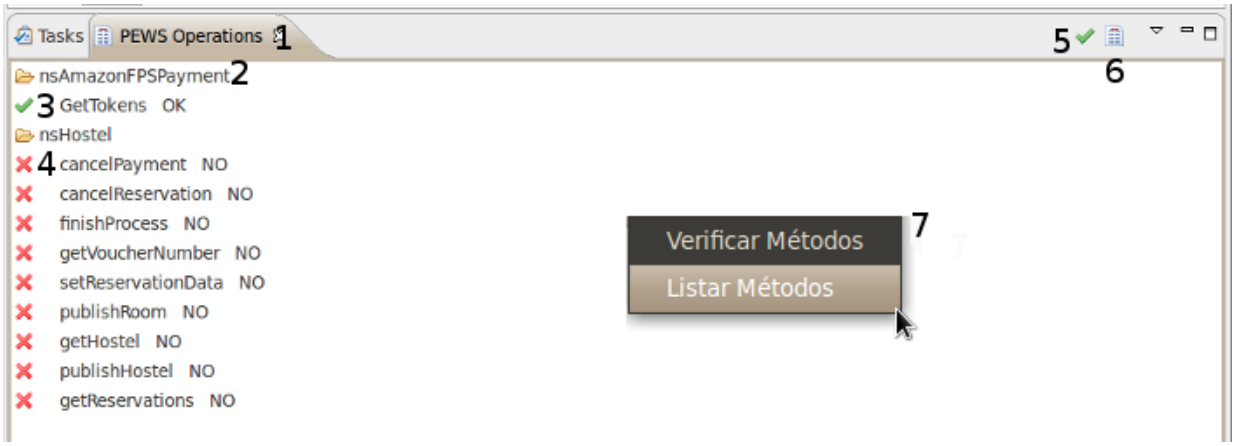

Figura 9: PEWS Operations.

Formas de acesso para as validações.

O Plugin Extrator disponibiliza menus para a verificação dos métodos XPEWS, acessíveis através do *workbench*, da plataforma Eclipse chamando as ações (actionSets). São formas que o Extrator emprega ao reagir com as interações do usuário do *plugin*:

Esses menus estão dispostos da seguinte forma:

- Adição de um menu na barra de menus superior do Eclipse como mostrado no item 3 da Figura 10, contendo dois submenus para a listagem e verificação dos métodos da composição no arquivo XPEWS, item 4 e 5 da Figura 10;
- Ao clicar com o botão em cima de um arquivo.xml, para que seja visualizado no menu *popup* com uma nova opção: "XPEWS", item 1 da Figura 10, com um submenu Verificar Métodos, item 3 da Figura 10. A partir dele é instanciado a *view* PEWS Operation para a visualização dos métodos da composição no arquivo XPEWS.

Orquestração expressa no XPEWS

O arquivo XPEWS segue o padrão de XML, é nele onde se encontra toda a descrição dos serviços web utilizados na composição, quais dos métodos serão utilizados e como será a composição. Os nós ***behaviour*** e ***operation*** presentes no arquivo XPEWS serão os nós utilizados na análise e verificação dos métodos.

O elemento ***behaviour*** (Figura 11) especifica o modo em que cada cliente pode interagir com o serviço web, sendo composto por dois atributos: **xmlns:** declara um ***namespace*** que referencia a um documento WSDL, podendo assim, usar vários provedores de conteúdo em uma única composição de serviços; e o atributo ***name:*** que é o nome da composição.

O elemento ***operation*** (Figura 11) declara quais operações serão utilizados na composição, é composto pelo atributo ***name***. O atributo ***portType*** informa qual ***portType*** contém a operação e o atributo ***refersTo*** associa a operação a um método de um ***namespace*** declarado no elemento ***behavior***.

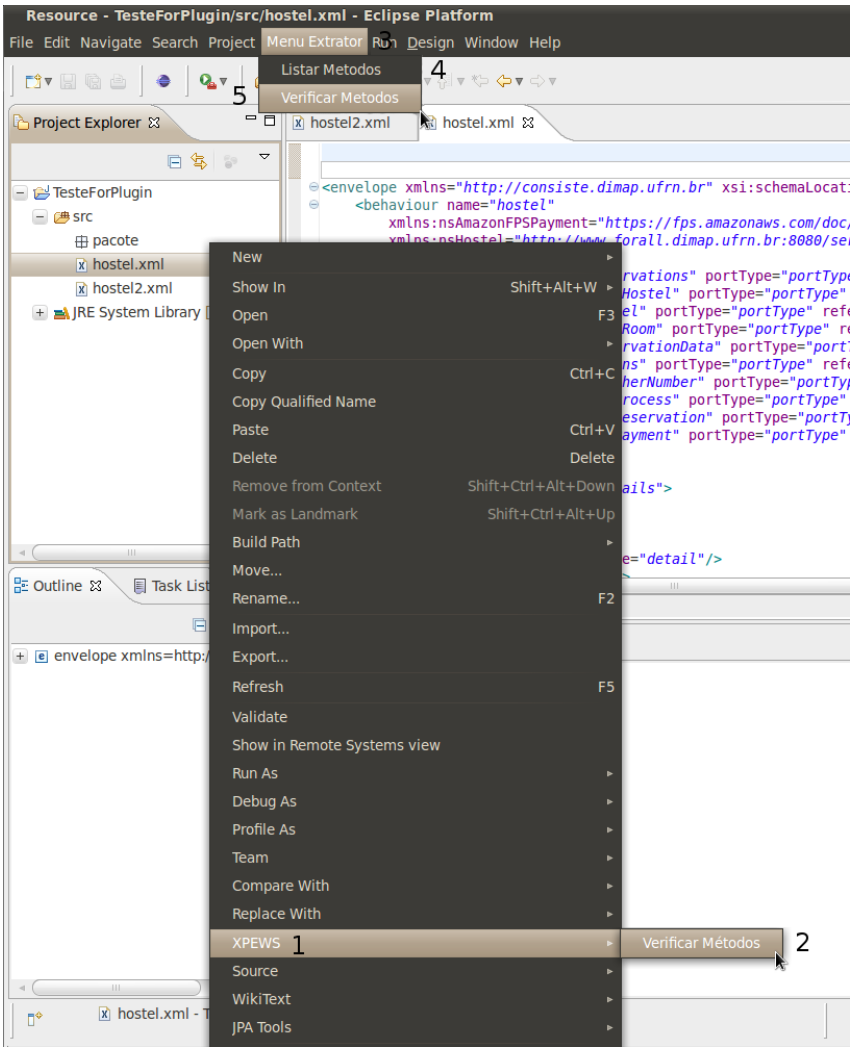

Figura 10: Menus Plugin Extrator.

As operações descritas no XPEWS que são usadas na composição do serviços estão na *tag operation* no documento WSDL que especifica como acessar o serviço web, suas operações ou métodos disponíveis.

O *plugin* tem que se certificar para que seja feita a verificação das operações usadas na composição no arquivo XPEWS com as operações disponibilizadas pelo serviço WEB através do documento WSDL. O Plugin Extrator precisa comparar os dois para verificar quais métodos estão operáveis e quais não estão, checando se as *operations* do XPEWS estão descritas nas *operations* do(s) WSDL(s). Com isso, é mostrando ao usuário do *plugin* quais das operações realmente são válidas, ou seja, existem tanto na descrição da composição quanto na descrição do serviço. O método ***GetTokens*** sublinhado na Figura 11 e 12 é o método que realmente existe na descrição do WSDL, os outros portanto são inválidos.

```
<envelope>
    <behaviour name="hostel"
        xmlns:nsAmazonFPSPayment="https://fps.amazonaws.com/doc/2008-09-17/AmazonFPS.wsdl"
        xmlns:nsLocadora="http://localhost:8080/WebServiceProject/services/Locadora?wsdl"
        xmlns:nsHostel="http://www.forall.dimap.ufrn.br:8080/services/HostelService.wsdl">
        <operations>
            <operation name="getReservations" portType="portType" refersTo="nsHostel:getReservations"/>
            <operation name="publishHostel" portType="portType" refersTo="nsHostel:publishHostel"/>
            <operation name="getHostel" portType="portType" refersTo="nsHostel:getHostel"/>
            <operation name="publishRoom" portType="portType" refersTo="nsHostel:publishRoom"/>
            <operation name="setReservationData" portType="portType" refersTo="nsHostel:setReservationData"/>
            <operation name="GetTokens" portType="portType" refersTo="nsAmazonFPSPayment:GetTokens"/>
            <operation name="getVoucherNumber" portType="portType" refersTo="nsHostel:getVoucherNumber"/>
            <operation name="finishProcess" portType="portType" refersTo="nsHostel:finishReservation"/>
            <operation name="cancelReservation" portType="portType" refersTo="nsHostel:cancelReservation"/>
            <operation name="cancelPayment" portType="portType" refersTo="nsHostel:cancelPayment"/>
        </operations>
        <services> ... </services>
        <varDef>...</varDef>
        <pathExp>... </pathExp>
    </behaviour>
    <contract_behaviour>...</contract_behaviour>
</envelope>
```

Figura 11: Fragmento de um arquivo XPEWS, métodos que irão ser comparados na verificação.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions >
    <wsdl:types>
      <xs:schema> ... <xs:schema>
    </wsdl:types>
    ...
    <wsdl:portType name="AmazonFPSPortType">
        <wsdl:operation name="CancelToken">
            <wsdl:documentation>
                Cancels any token installed by the calling application on its own account.
            </wsdl:documentation>
            <wsdl:input message="tns:CancelTokenRequestMsg" wsa:Action="urn:CancelToken"/>
            <wsdl:output message="tns:CancelTokenResponseMsg" wsa:Action="urn:CancelToken:Response"/>
        </wsdl:operation>

        <wsdl:operation name="Cancel">
            <wsdl:documentation>
                Cancels an ongoing transaction and puts it in cancelled state.
            </wsdl:documentation>
            <wsdl:input message="tns:CancelRequestMsg" wsa:Action="urn:Cancel"/>
            <wsdl:output message="tns:CancelResponseMsg" wsa:Action="urn:Cancel:Response"/>
        </wsdl:operation>

        <wsdl:operation name="FundPrepaid">
            <wsdl:documentation>
                Funds the prepaid balance on the given prepaid instrument.
            </wsdl:documentation>
            <wsdl:input message="tns:FundPrepaidRequestMsg" wsa:Action="urn:FundPrepaid"/>
            <wsdl:output message="tns:FundPrepaidResponseMsg" wsa:Action="urn:FundPrepaid:Response"/>
        </wsdl:operation>

        <wsdl:operation name="GetTokens">
            <wsdl:documentation>
                Returns a list of tokens installed on the given account.
            </wsdl:documentation>
            <wsdl:input message="tns:GetTokensRequestMsg" wsa:Action="urn:GetTokens"/>
            <wsdl:output message="tns:GetTokensResponseMsg" wsa:Action="urn:GetTokens:Response"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

Figura 12: Fragmento de um arquivo WSDL, métodos que irão ser comparados na verificação.

A Figura 13 é uma captura de tela exibindo a *view* PEWS Operation mostrando o resultado da verificação das operações do código XPEWS (Figura 11) com os métodos

HOLOS
ISSN 1807 - 1600

descritos no serviço através do código do WSDL (Figura 12) indicando quais estão válidos para a realização da composição.

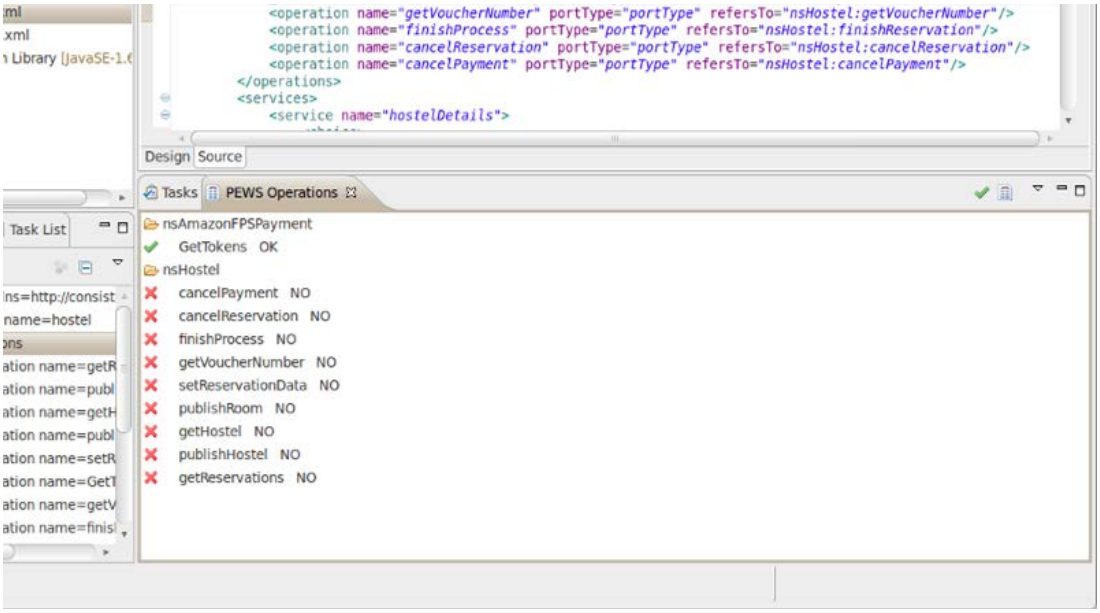

Figura 13: Visualização da comparação das *operations* na aba PEWS Operations.

ESTUDO DE CASO

Este estudo de caso foi proposto para mostrar o funcionamento do *plugin* numa situação simulando uma composição de serviço através de uma aplicação de rádio online. A rádio online funciona da seguinte maneira: a aplicação tem uma ferramenta de busca que tem como principal função encontrar um determinado artista exibindo sua discografia e as informações dos álbuns, tais como: ano de lançamento, faixas do álbum, tempo total do álbum, ano, gênero etc. O usuário da rádio *online* tem a possibilidade de criar uma lista de músicas de acordo com sua preferência e visualizar se aquele álbum ou música esta disponível para venda, caso seja de interesse dele, é realizado a venda dos produtos escolhidos.

Foram utilizados 4 serviços web diferentes na composição acima, Figura 15:

- http://www.flash-db.com/services/ws/flashCDDB.wsdl, tem como *namespace* buscarMusic;
- http://ws.contentlib.mweb.co.th/ContentLibrary.WS/WSMusic.asmx?wsdl, tem como *namespace* ouvirMusic;
- http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl , cujo *namespace* é comprarMusic;
- https://fps.amazonaws.com/doc/2008-09-17/AmazonFPS.wsdl, cujo *namespace* é pagarMusic.

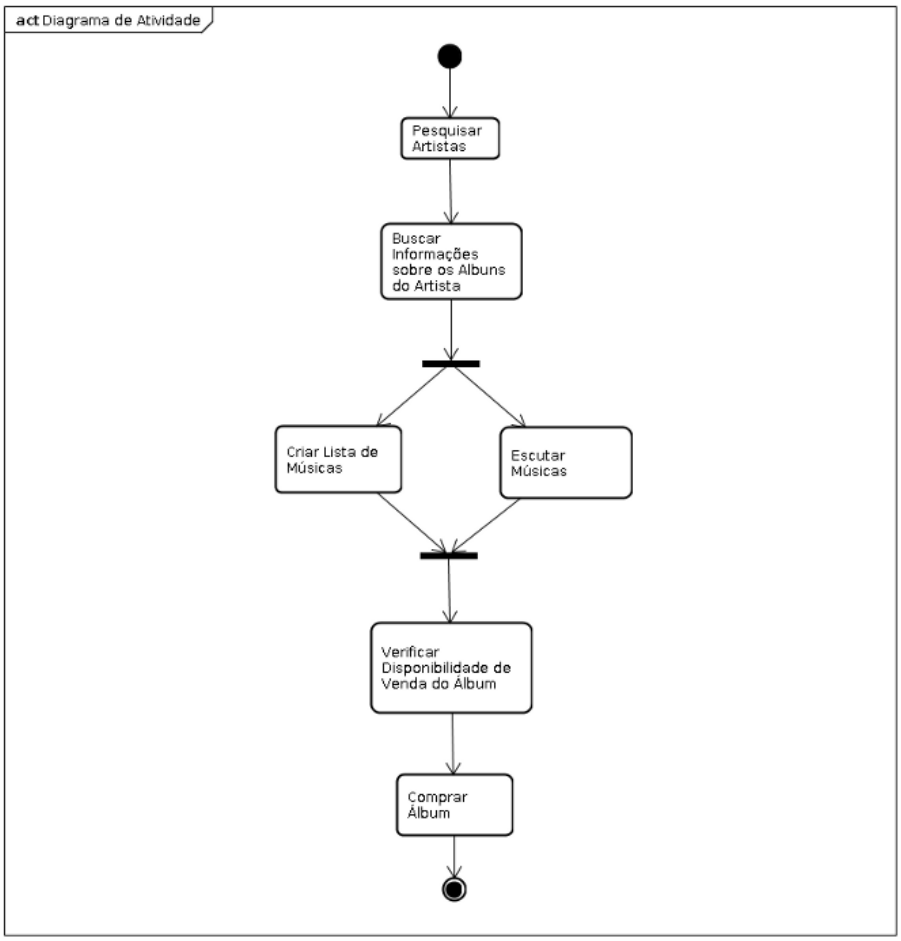

Figura 14: Diagrama de atividade da Radio Online em UML.

A partir desse cenário da Figura 14, foi proposto um seguinte esquema para a composição de serviço:

```
ns buscarMusic = "http://www.flash-db.com/services/ws/flashCDDB.wsdl"
ns ouvirMusic = "http://ws.contentlib.mweb.co.th/ContentLibrary.WS/WSMusic.asmx?wsdl"
ns comprarMusic = "http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl"
ns pagarMusic = "https://fps.amazonaws.com/doc/2008-09-17/AmazonFPS.wsdl"

alias searchArtistList = portype/searchArtistList in buscarMusic
alias getAlbumInfo = portype/getAlbumInfo in buscarMusic

alias Music_Songlist_Select = portype/Music_Songlist_Select in ouvirMusic
alias Music_Song_Select = portype/Music_Song_Select in ouvirMusic

alias ItemSearch = portype/ItemSearch in comprarMusic
alias SellerListingSearch = portype/SellerListingSearch in comprarMusic

alias GetPaymentInstruction = portype/GetPaymentInstruction in pagarMusic
alias Pay = portype/Pay in pagarMusic

(searchArtistList . getAlbumInfo. Music_Song_Select. Music_Songlist_Select .
ItemSearch . SellerListingSearch . GetPaymentInstruction . Pay)*
```

Figura 15: Radio Online PEWS.

O *namespace* buscarMusic pesquisa os artistas e mostrar as informações dos seus álbuns:

- O alias *searchArtistList* encarregado de listar os artistas pesquisados;
- O alias *getAlbumInfo* retorna a informação dos álbuns do artista.

O *namespace* ouvirMusic é responsável por trazer para o usuário da rádio as musicas que ele escolheu para escutar:

- *Music_Song_Select* é o alias encarregado da visualização das musicas do artista;
- *Music_Songlist_Select* é o alias que cria a lista que músicas escolhidas pelo usuário da rádio para ouvir.

O *namespace comprarMusic* irá buscar se aquele álbum ouvido pelo usuário da rádio esta disponível para sua compra:

- ItemSearch é o alias que se encarrega com a busca do álbum ou música para ver se está disponível ou não para a venda
- O alias SellerListingSearch fica responsável pela lista de produtos a serem vendidos.

O *namespace* pagarMusic é responsável por fazer a transação de venda:

- GetPaymentInstruction é o alias que tem como função mostrar quais são as instruções para realizar o pagamento da compra: tipos de pagamentos, parcelas e etc;
- O alias Pay tem como função finalizar o pagamento, e conseguinte, a compra das musicas feita pelo usuário da rádio.

A partir desse arquivo PEWS conseguimos obter o XPEWS (Figura 16) através do *plugin front-end* desenvolvido pelo Potrich (2006), o resultado é o arquivo mostrado abaixo:

```
<envelope xmlns="http://portal.ifrn.edu.br/" xsi:schemaLocation="http://aquarius.inf.ufpr.br pews.xsd">
 <behaviour name="Listem Buy Music"
  xmlns:buscarMusic="http://www.flash-db.com/services/ws/flashCDDB.wsdl"
  xmlns:ouvirMusic = "http://ws.contentlib.mweb.co.th/ContentLibrary.WS/WSMusic.asmx?wsdl"
  xmlns:comprarMusic = "http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl"
  xmlns:pagarMusic = "https://fps.amazonaws.com/doc/2008-09-17/AmazonFPS.wsdl">

  <operations>
   <operation name="searchArtistList" portType="portType" refersTo="buscarMusic:searchArtistList"/>
   <operation name="getAlbumInfo" portType="portType" refersTo="buscarMusic:getAlbumInfo"/>
   <operation name="ItemSearch" portType="portType" refersTo="comprarMusic:ItemSearch"/>
   <operation name="SellerListingSearch" portType="portType" refersTo="comprarMusic:SellerListingSearch"/>
   <operation name="GetPaymentInstruction" portType="portType" refersTo="pagarMusic:GetPaymentInstruction"/>
   <operation name="Pay" portType="portType" refersTo="pagarMusic:Pay"/>
   <operation name="Music_Songlist_Select" portType="portType" refersTo="ouvirMusic:Music_Songlist_Select"/>
   <operation name="Music_Song_Select" portType="portType" refersTo="ouvirMusic:Music_Song_Select"/>
  </operations>
  <pathExp>...</pathExp>
 </behaviour>
</envelope>
```

Figura 26: Radio Online XPEWS.

RESULTADOS

Deve-se acessar uma das duas formas mostradas na Figura 10 para analisar as operações da composição de serviço pelo Plugin Extrator. No caso dos testes foi usado a opção localizada no menu superior do eclipse, como visto na Figura 17.

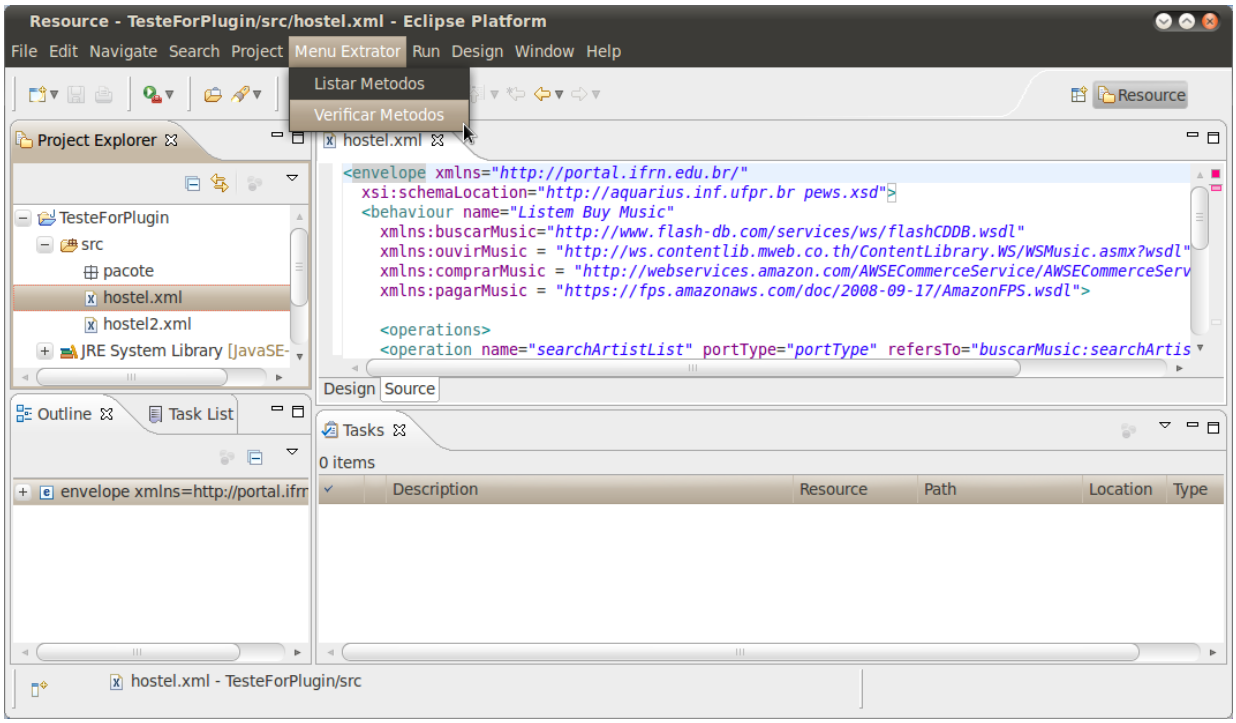

Figura 17 Acessando o Plugin Extrator para analise das operações

A Figura 18 mostra a visualização da lista contendo os *namespaces* dos serviços e seus respectivos métodos (Tabela 1) que estão sendo utilizados na composição da Radio Online. O Plugin Extrator verificou, através da comparação sintática, os métodos descritos na composição contidos no arquivo XPEWS com os métodos relatados nas interfaces de descrição de serviços no WSDL. Com isso, o usuário do *plugin* tem a noção antes mesmo de aplicação está finalizada de quais métodos descritos estão ou não nos *namespaces*.

Como forma de testar a eficiência do *plugin* foi elaborado mais um teste que simula a verificação dos métodos. Da mesma forma que no primeiro teste para analisar as operações da composição, foi usado novamente a opção localizada no menu superior do eclipse, Figura 17.

A diferença desse teste para o primeiro é que algumas das operações foram alteradas para que ocorra propositalmente o erro na validação dos métodos, devido ao fato de tais métodos não estarem descritos em seu serviço, mostrado para o usuário do *plugin* que esses métodos não estão válidos para serem usados na composição. Foi utilizado nesse teste o código XPEWS da Figura 19:

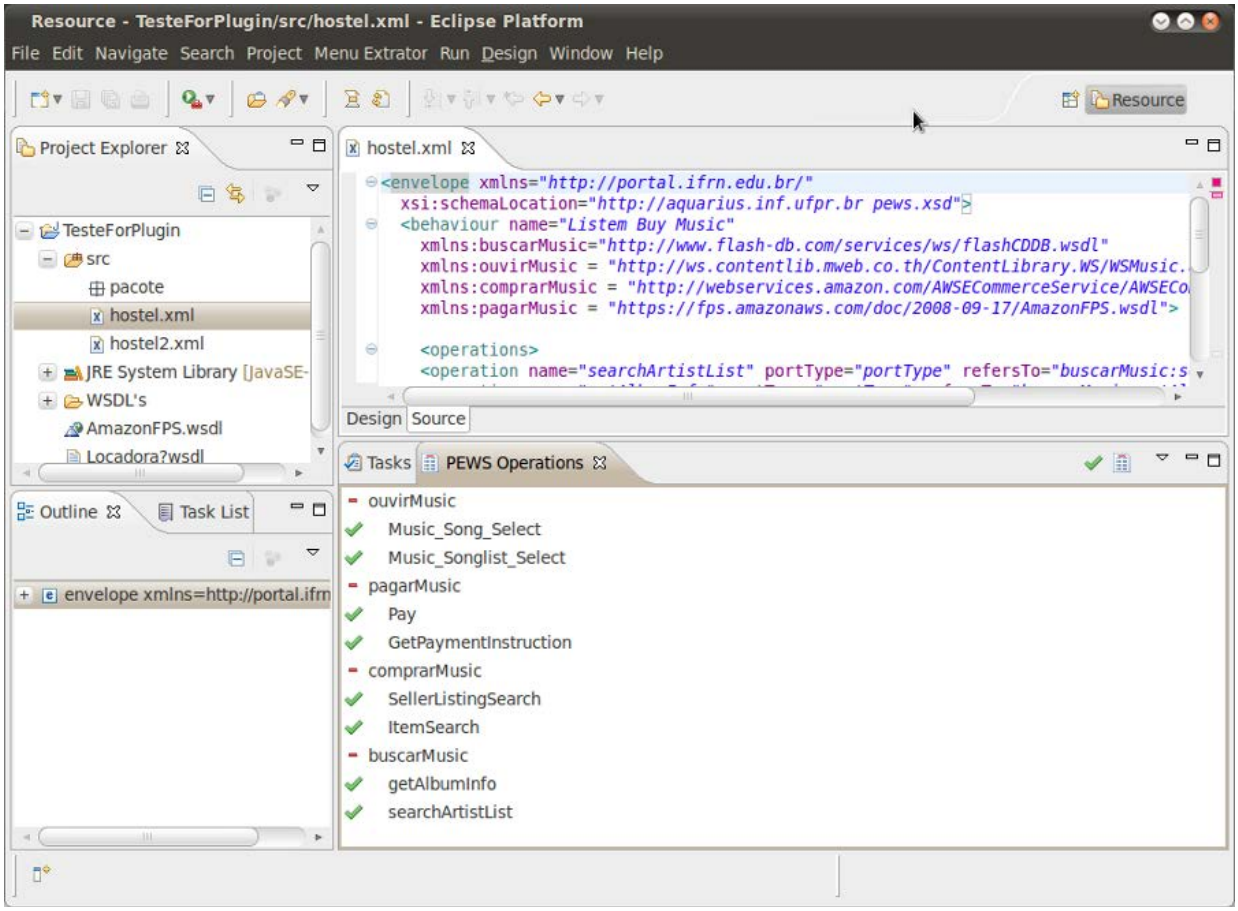

Figura 18: Resultado da verificação dos métodos da composição na aba PEWS Operations da Figura 16.

```
<envelope xmlns="http://portal.ifrn.edu.br/"
  xsi:schemaLocation="http://aquarius.inf.ufpr.br pews.xsd">
  <behaviour name="Listem Buy Music"
    xmlns:buscarMusic="http://www.flash-db.com/services/ws/flashCDDB.wsdl"
    xmlns:ouvirMusic = "http://ws.contentlib.mweb.co.th/ContentLibrary.WS/WSMusic.asmx?wsdl"
    xmlns:comprarMusic = "http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl"
    xmlns:pagarMusic = "https://fps.amazonaws.com/doc/2008-09-17/AmazonFPS.wsdl">

    <operations>
    <operation name="searchArtistList" portType="portType" refersTo="buscarMusic:searchArtistList"/>
    <operation name="GetAlbumInfo" portType="portType" refersTo="buscarMusic:GetAlbumInfo"/>
    <operation name="ItemSearch" portType="portType" refersTo="buscarMusic:ItemSearch"/>
    <operation name="SellerListingSearch" portType="portType" refersTo="comprarMusic:SellerListingSearch"/>
    <operation name="GetPaymentInstruction" portType="portType" refersTo="pagarMusic:GetPaymentInstruction"/>
    <operation name="Pay" portType="portType" refersTo="pagarMusic:Pay"/>
    <operation name="Music_Songlist_Select" portType="portType" refersTo="ouvirMusic:Music_Songlist_Select"/>
    <operation name="Music_Song_Select" portType="portType" refersTo="ouvirMusic:Music_Song_Select"/>
    <operation name="Music_artista_Select" portType="portType" refersTo="ouvirMusic:Music_artista_Select"/>
    </operations>
    <pathExp>...</pathExp>
  </behaviour>
</envelope>
```

Figura 19: Radio Online XPEWS apresentando métodos inválidos.

A Figura 19 difere do código XPEWS da Figura 16 nos seguintes aspectos:

- A adição de um método inexistente chamado Music_artista_select pertencente ao serviço “ouvirMusic”;

- Alteração da assinatura do método getAlbumInfo para GetAlbumInfo para que seja apresentado mais um erro de validação;
- O método ItemSearch que pertence ao serviço comprarMusic foi alterado como se ele fosse pertencendo ao serviço buscarMusic, provocando novamente outro aviso de erro de validação de método.

O Resultado final do teste na Figura 20 mostra quais das operações usadas na composição não estão presentes nos serviços relacionados:

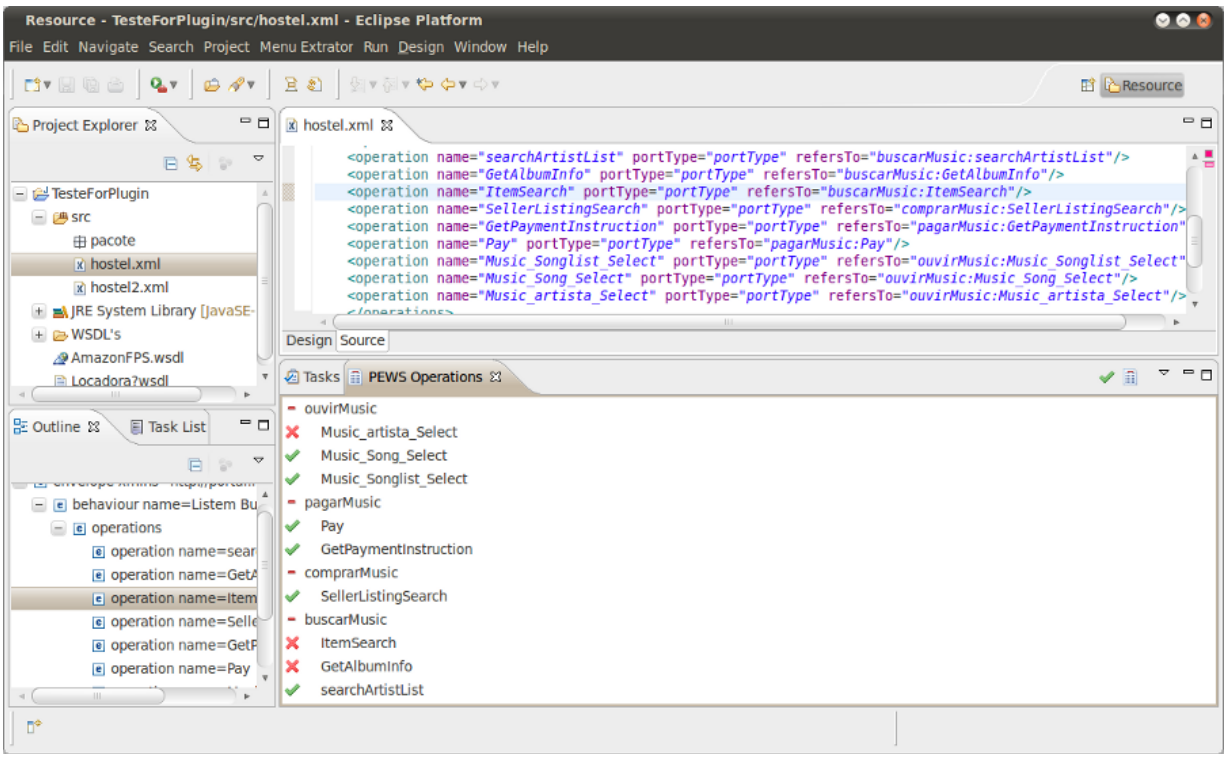

Figura 20: Resultado da verificação dos métodos da composição com erros de validação representados com o "X" vermelho.

A Tabela 1 e Tabela 2 mostram os serviços e operações usados no primeiro e segundo teste respectivamente. Como visto na Figura 18, o Plugin Extrator mostrou que todas as operações usadas na composição pertencem ao serviço cujo qual foi relacionado de maneira correta, assinalado com o símbolo em forma de "V".

Já na Figura 20, O *plugin* mostrou as operações relacionadas de maneira errada com o serviço com o símbolo em forma de "X", devido aos métodos não existirem nos serviços aos quais foram relacionados.

Tabela 1: *Namespaces* dos serviços e operações usadas no primeiro teste.

***NAMESPACES* DOS SERVIÇOS**	**OPERAÇÕES**	**STATUS**

buscarMusic	searchArtistList	✔
	getAlbumInfo	✔
ouvirMusic	Music_Song_Select	✔
	Music_Songlist_Select	✔
comprarMusic	itemSearch	✔
	SellerListingSearch	✔
pagarMusic	GetPaymentInstruction	✔
	Pay	✔

***NAMESPACES* DOS SERVIÇOS**	**OPERAÇÕES**	**STATUS**
buscarMusic	itemSearch	✘
	searchArtistList	✔
	GetAlbumInfo	✘
ouvirMusic	Music_artista_Select	✘
	Music_Songlist_Select	✔
comprarMusic	SellerListingSearch	✔
pagarMusic	GetPaymentInstruction	✔
	Pay	✔

Tabela2: *Namespaces* dos serviços e operações usadas no segundo teste.

CONSIDERAÇÕES FINAIS

Este artigo apresentou uma contribuição ao trabalho desenvolvido pelo Potrich (2006), como uma forma de agregador de valor a tal ferramenta. Tendo como objetivo principal validar os métodos utilizados na composição, pertencentes na especificação XPEWS de serviço, visando verificar operabilidade das operações utilizadas na composição. Para alcançar esse objetivo foi proposta a criação de um *plugin* para a plataforma de desenvolvimento Eclipse, aproveitando a potencialidade dessa ferramenta que já trabalha com o desenvolvimento de serviços Web.

Neste trabalho foram apresentados detalhes de como ocorre o processo de composição de serviços através da linguagem PEWS que usa *path-Expression* para a descrição de comportamento dos métodos das interfaces dos serviços web, como por exemplo, a ordem em que as operações de serviços web são executadas.

O *plugin* apresentado tem como função, auxiliar o desenvolvedor na hora da criação de composições de serviço utilizando a linguagem PEWS pela da ferramenta PEWS Editor, Resolvida a limitação desse editor através da validação dos métodos usados, para que o desenvolvedor tenha certeza de que os métodos usados nessa composição estão referenciados ao seu legitimo serviço. Para isso, foi criado um ambiente para melhor

visualização desse processo de validação, onde é mostrando ao desenvolvedor quais serviços e métodos estão sendo usados na composição e quais estão válidos ou não.

Por fim, foi apresentado um caso de uso de uma Radio Online para mostrar como se dá a criação das composições de serviços, os passos utilizados nos PEWS para execução dessa composição e a validação dos métodos através dos Plugin Extrator.

O trabalho é uma primeira versão de um protótipo que ainda carece de evolução para poder entrar em funcionamento de forma estável em ambientes de desenvolvimentos reais. Na fase atual, algumas melhorias podem ser vislumbradas no Plugin Extrator em torno do seu funcionamento, sendo algumas destacadas aqui como trabalhos futuros:

- Adicionar ao console as informações de download do(s) WSDL(S) para a máquina do usuário do Plugin Extrator para que o ele possa ter noção de quais estão ou foram baixados;
- Organizar de forma mais intuitiva a visualização dos erros.
- Disponibilizar o código fonte do plugin para que a comunidade possa utilizar como objeto de estudo e estender suas funcionalidades.

É também visado como trabalho futuro que após essas melhorias serem feitas, o projeto seja submetido ao Potrich (2006) para uma avaliação, visando para que o Plugin Extrator seja integrado ao PEWS Editor, para que os objetivos proposto por esse trabalho seja alcançado.

Por fim, espera-se que este trabalho possa contribuir para futuros projetos de pesquisa, para a ciência, como também, para literatura, vindo a servir como material de consulta e de referências.

REFERÊNCIAS BIBLIOGRÁFICAS

1. BA, C. et al. Building web services interfaces usingn predicate path expressions. IX Brazilian Symposium on Programming Languages, Recife - Brazil: University of Pernambuco, n. 7, p.4, 28 jul. 2005.
2. BA, C. et al. Pews: A new language for building web service interfaces. Journal of Universal Computer Science, São Paulo, n. 7, p. 4, 28 jul. 2005.
3. CLARO ALBERS, P. J.-K. H. D. B. Comparison between ws-bpel and owl-s. [S.l.]: Internacional Conference on Enterprise Information Systems, 2005.
4. COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Sistemas Distribuidos – Conceitos E Projeto. [S.l.]: BOOKMAN COMPANHIA ED, 2007. ISBN 9788560031498.
5. DEITEL, H. Xml como Programar. [S.l.]: Bookman Companhia Ed, 2001. ISBN 9788536301471
6. ECLIPSE. Eclipse. 2005. Disponível em: <http://www.ibm.com/developerworks/br/library/os-ecplug/>. Acesso em: Maio de 2011.
7. FONSECA, A. de A. Desenvolvimento de um plugin para Composição de Serviços Web utilizando a Plataforma Eclipse. Salvador: [s.n.], 2008.

8. GALLARDO, D. Desenvolvendo Plug-ins do Eclipse. 2002. Disponível em <http://www.ibm.com/developerworks/br/library/os-ecplug/>. Acesso em: Maio de 2011.
9. GRONER, L. Java e XML. 2009. Disponível em: <http://www.loiane.com/2009/04/java-e-xml/>. Acesso em: Maio de 2011.
10. MACHINES, I. B. Eclipse documentation - Current Release. 2003. Disponível em: <http://tinyurl.com/6xjyz23>. Acesso em: Junho de 2011.
11. POTRICH, E. Pews Editor, Um Front-End para a Linguagem Pews. Curitiba: Universidade Federal do Paraná, 2006.
12. SAMPAIO, C. SOA e Web Services em Java. [S.l.]: Brasport, 2006. ISBN 9788574522678.
13. SILVA, L. M. Aplicando Composição e Orquestração de Serviços na Organização de Sistemas. São Paulo – SP: [S.n.], 2007. Disponível em: <www.ic.unicamp.br/beatriz/cursos/mo809/2008/referencias/monografia.pdf>. Acesso em: Fevereiro de 2011.
14. SITE, E. P. Introduction to Eclipse Plugin Development. 2008. Disponível em: <http://www.eclipsepluginsite.com/index.html>. Acesso em: Maio de 2011.
15. VELOSO, R. R. Java e XML, Processamento de documentos XML com Java. São Paulo - SP: Novatec Editora Ltda, 2007. ISBN 9788575221112.