Integração do Mecanismo de Self-healing na Execução das Composição de Sistemas de Informação através dos Serviços Web Semânticos*

Manuele Ferreira[1], **Daniela Barreiro Claro**[1], **Denivaldo Lopes**[2]

[1]FORMAS - Grupo de Pesquisa em Formalismos e Aplicações Semânticas
LaSiD/DCC/IM/UFBA
Av. Adhemar de Barros, s/n, Ondina – Salvador – BA – Brasil
manueleferreira@gmail.com, dclaro@ufba.br

[2]Laboratório de Engenharia de Software e Redes de Computadores (LESERC)
Campus do Bacanga - CCET – São Luís – MA – Brasil
dlopes@dee.ufma.br

Abstract. *The use of Web services has grown due to the increase of interoperable environment, low cost, integrated and loosely coupled systems. However, an atomic service, at times, does not achieve the goal given by a user, thus it should be composed. Compositions of web services could become available in large and complex environments, for example the Internet, which leverage to faults during the execution process. In order to provide greater availability of these compositions, mechanisms of detection and failure recovery are necessary and were integrated into a development environment to improve reliability. Self-manageable features should also be included due to the complexity of the environment. Thus, this work proposes the incorporation of mechanisms for self-healing as an Eclipse plugin to manage these web service compositions. Experiments were conducted to validate our proposal and generate a comparison with previous work.*

Resumo. *O uso de serviços web cresceu em decorrência do aumento da necessidade de prover serviços em um ambiente interoperável, com baixo custo, integrado e com baixo acoplamento. No entanto, um serviço isolado, em determinados momentos, não atende ao objetivo definido pelo usuário, sendo assim necessário compor estes serviços. Composições de serviços web podem estar disponibilizadas em ambientes complexos e grandes, por exemplo a Internet, o que as torna suscetíveis a falhas durante o processo de execução. Com o intuito de prover uma maior disponibilidade destas composições, mecanismos de detecção e recuperação de falhas são necessários e foram integrados em um ambiente de desenvolvimento para aumentar a confiabilidade. Estes mecanismos ainda devem possuir características auto-gerenciáveis devido à complexidade do ambiente. Assim, o presente trabalho propõe a incorporação de mecanismos de auto-cura como um plugin do Eclipse para o gerenciamento destas composições de serviços Web. Experimentos foram realizados visando validar a proposta e gerar um comparativo com os trabalhos relacionados.*

*O autor Denivaldo Lopes agradece à FAPEMA pela bolsa de pesquisa concedida BEPP 0431/2010

1. Introdução

O crescente desenvolvimento de sistemas de informação (SI) vem ampliando a utilização de ferramentas amigáveis e funcionais. Diversos serviços destes sistemas de informação têm sido publicado através da Web, utilizando um formato padronizado denominado Serviços Web(*Web Services*). Este formato padronizado vem garantindo interoperabilidade entre SI heterogêneos, facilitando a composição e consequentemente ampliando o fornecimento de novas funcionalidades. Ferramentas amigáveis e funcionais no processo de desenvolvimento de serviços web já foram incorporadas em diversos ambientes integrados de desenvolvimento (*IDE - Integrated Development Environment*) como a IDE Eclipse. Essa incorporação propicia o aumento da produtividade e diminui a curva de aprendizado de um desenvolvedor Web.

Descrições semânticas vêm sendo incorporadas em serviços Web e também nestes ambientes integrados de desenvolvimento (IDE) com o intuito de facilitar a utilização por desenvolvedores. Dentre as linguagens semânticas, a OWL-S (*Ontology Web Language for Services*) permite descobrir e selecionar serviços Web automaticamente, pois adiciona características não ambíguas e interpretáveis computacionalmente. A OWL-S é baseado na OWL(*Web Ontology Language*) que é uma linguagem para descrição de ontologias [McGuinness and van Harmelen 2004]. Os serviços Web que utilizam características semânticas na sua descrição são denominados Serviços Web Semânticos (SWS).

Esses serviços web semânticos são normalmente publicados em ambientes complexos e grandes, por exemplo, a Internet, o que os torna suscetíveis a falhas durante o processo de execução, principalmente, por indisponibilidade. Visando minimizar esses problemas, propõe-se a utilização de sistemas autonômicos, que permitem o auto-gerenciamento de sistemas computacionais. Existem quatro principais propriedades de autogerenciamento: auto-configuração (*self-configuring*), auto-otimização (*self-optimizing*), auto-proteção (*self-protecting*) e a auto-cura (*self-healing*) [Horn 2001]. O mecanismo de auto-cura auxilia no aumento da disponibilidade de serviços, pois prevê mecanismos para detectar, prover diagnósticos e a correção de falhas de forma autonômica, mantendo o sistema consistente.

Diante desse contexto, o objetivo do presente trabalho é integrar mecanismos de auto-cura (*self-healing*) na etapa de execução de composições de serviços web geradas pelo *plugin* OWL-S Composer [Sena et al. 2010]. A adição dessas novas funcionalidades permitem a geração, pelo *plugin*, de composições de serviços com maior disponibilidade.

Este artigo encontra-se estruturado da seguinte forma: a seção 2 apresenta os trabalhos relacionados; a seção 3 apresenta uma visão geral do *plugin* OWL-S Composer. A seção 4 descreve o trabalho proposto e a seção 5 apresenta os experimentos realizados. A seção 6 apresenta os resultados obtidos e a seção 7 finaliza com as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Durante as pesquisas, três trabalhos relacionados à proposta deste artigo foram evidenciados. O *Self-healing Integrator for Web Services* (SHIWS) [Denaro et al. 2007] é um *plugin* para a plataforma Eclipse que explora a auto-adaptação dos sistemas autonômicos, provendo um mecanismo dinâmico que identifica e adapta as incompatibilidades entre os

solicitantes dos serviços e seus fornecedores. Porém, o processo de modelagem desta ferramenta é manual, tornando custosa a etapa de inserção de um novo serviço e impossibilitando a reutilização de serviços distribuídos na grande rede. Além disso, a automatização do processo é inviabilizada visto que uma linguagem de descrição semântica não é utilizada.

O SH-BPEL [Modafferi et al. 2006] é um *plugin* para WS-BPEL que busca prover uma arquitetura para monitoramento, geração de diagnóstico e recuperação de falhas identificada na execução das composições de serviços web modelados por essa linguagem. Um dos pontos negativos nessa abordagem é a utilização de uma linguagem que não oferece suporte a semântica, o que não permite a busca e seleção automática de serviços web.

Autores em [Wiesner et al. 2008] definem um modelo de recuperação de falhas em SWS, objetivando ampliar a flexibilidade, consistência e confiabilidade para serviços web. As idéias apresentadas em [Wiesner et al. 2008] são próximas à proposta deste trabalho, porém não há evidências de implementação do mecanismo proposto pelos autores. Além do mais, o mecanismo citado trata de serviços web, mas não aborda uma composição de serviços web, diferentemente do trabalho proposto neste artigo.

Autores em [Sena et al. 2010] propuseram um *plugin* denominado *OWL-S Composer 2.0* para o ambiente Eclipse que permite a composição manual e visual de SWS, incluindo a geração de composições semanticamente similares. Um dos principais pontos negativos do *OWL-S Composer 2.0* é que este *plugin* não publica, nem executa e nem monitora uma composição de serviços web desenvolvida neste ambiente.

Dessa forma, este trabalho tem como objetivo principal propor a integração de mecanismos de auto-cura na etapa de execução de composições de SWS. Estes mecanismos visam permitir a adaptação relativa às mudanças no ambiente, por exemplo, na presença de falhas, garantindo o alcance do objetivo na execução de composições de SWS. Além disso, este trabalho incorpora uma linguagem de descrição semântica aos mecanismos de recuperação, tornando-os mais flexíveis e robustos. Assim, o OWL-S Composer 3.0 foi desenvolvido para atender todo o ciclo de vida de uma composição OWL-S, publicando, executando e monitorando esta execução.

3. Visão Geral do OWL-S Composer 3.0

O *OWL-S Composer 1.0* [Fonseca et al. 2009] foi desenvolvido com o objetivo de prover um ambiente que permitisse a modelagem visual de composições de SWS através de diagramas. O desenvolvimento utilizando a plataforma Eclipse permitiu que ele fosse integrado com outras ferramentas que facilitassem as tarefas de criação e manutenção de serviços.

Por meio da utilização da ferramenta *OWL-S Discovery* que realiza descoberta semântica de serviços Web atômicos, foi desenvolvido o *OWL-S Composer 2.0* [Sena et al. 2010] que permite a descoberta e geração visual de composições de SWS semanticamente similares àquelas que foram modeladas manualmente.

A versão 3.0 do *plugin* é apresentada neste trabalho e integra mecanismos de autocura durante a fase de execução que permite aumentar a confiabilidade nas composições executadas neste ambiente.

3.1. Estratégias de recuperação utilizadas no mecanismo de Auto-cura

Dentre as ações de recuperação, o presente trabalho utilizou estratégias macros para recuperação em caso de falha conforme descrito a seguir:

- **Reexecutar (*Retry*):** Reexecuta o processo;

- **Substituição**

  - Por uma réplica (*ReplaceByEqual*): Efetua uma busca em servidores pré-definidos visando selecionar um serviço alternativo que seja uma réplica daquele onde foi identificada uma falha. Esse novo serviço selecionado será executado e o seu resultado utilizado no fluxo de execução da composição;

  - Por um serviço equivalente (*ReplaceByEquivalent*): Essa estratégia tem como objetivo selecionar serviços que são semanticamente similares àquele onde a falha foi identificada. É utilizada a ferramenta *OWL-S Discovery* para identificação de serviços semanticamente similares. O novo serviço é executado e o resultado retornado ao fluxo de execução da composição;

- **Saltar (*SkipOptionalService*):** Essa estratégia tem como objetivo deixar de executar aqueles serviços que não são essenciais para alcançar o objetivo de uma composição. Para tal, é identificado se o serviço que falhou é obrigatório ou facultativo baseado em uma análise dos seus *outputs*. Se a quantidade de *outputs* for igual a zero ou não houverem interligações entre esses *outputs* e outros serviços, esse será considerado facultativo e não será executado. Caso contrário, é obrigatório.

Estas estratégias de recuperação foram utilizadas para definir os mecanismos de auto-cura necessários a serem incorporados ao OWL-S Composer a fim de torná-lo mais confiável e robusto.

4. Mecanismo de auto-Cura proposto incorporado no OWL-S Composer

A fase de auto-cura foi incorporada no OWL-S Composer no processo de execução de composições de serviços web. A estrutura MDR (Monitoramento, Diagnóstico e Recuperação), tal como mostrada na Figura 1, foi gerada baseada na estrutura do MAPE-K [Huebscher and McCann 2008]. Há três fases básicas no MDR: *Monitoramento*, onde é feito o processo de avaliação e identificação de uma falha gerada pelo elemento que está sendo gerenciado; *Diagnóstico*, cujo objetivo é avaliar as falhas identificadas e selecionar uma estratégia de recuperação e a *Recuperação*, onde é executada a estratégia de recuperação que irá atuar no elemento gerenciado. É importante ressaltar que esse processo é feito de forma automática e dinâmica, não necessitando do usuário qualquer modificação nos seus arquivos OWL-S ou criação de outros arquivos auxiliares.

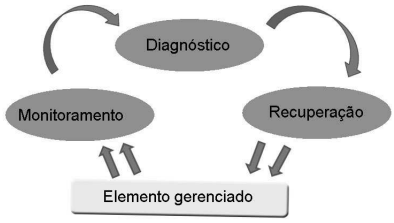

Figura 1. Estrutura MDR

As classes *ReplaceByEqual*, *SkipOptionalService*, *Retry* e *ReplaceByEquivalent* são as classes que irão receber a requisição com as informações da falha no processo de recuperação (Figura 2). O monitoramento detecta quando ocorrem falhas e realiza uma requisição à classe *SHMonitor* que vai identificar a falha. A classe *SHMonitor* tem como objetivo servir de interface entre as classes de execução de serviços da OWL-S API e o mecanismo de auto-cura.

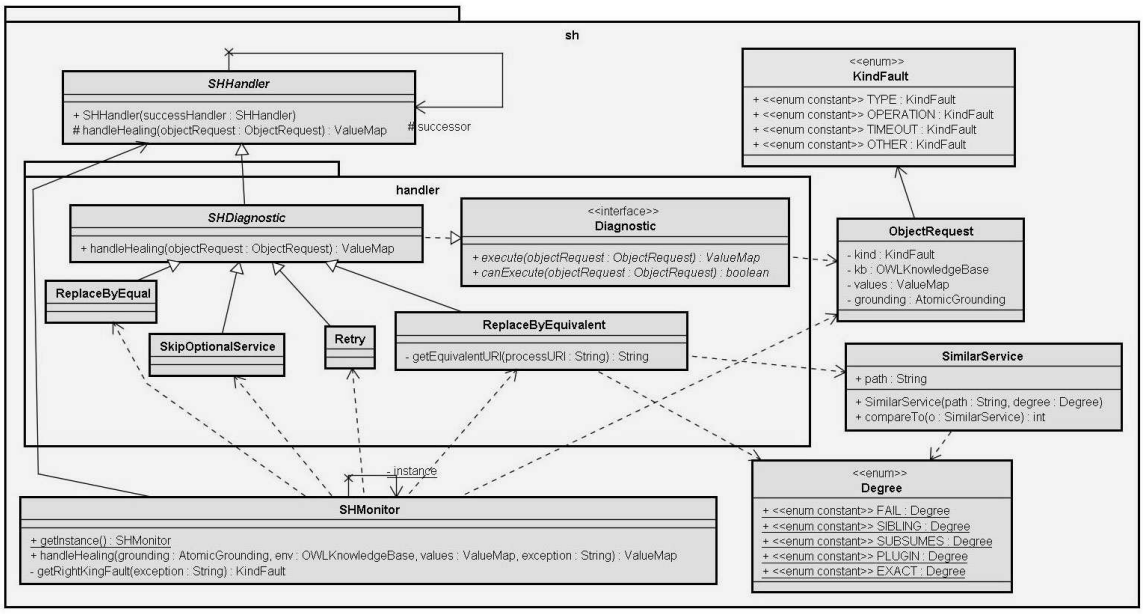

Figura 2. Diagrama de Classes do mecanismo de Auto-cura

Essa sequência pode ser melhor visualizada através do diagrama de sequência definido nas figuras 3 e 4. Na presença de falhas, o primeiro mecanismo de recuperação a ser acionado é o *Retry*. Caso seja possível reexecutar o serviço, este mecanismo realiza a recuperação e retorna o valor esperado pela execução do processo. Em caso negativo ou se ocorrer falhas durante o processo de recuperação, a classe *SHDiagnostic* aciona o mecanismo sucessor que é *ReplaceByEqual* e assim sucessivamente. *ReplaceByEqual* implementa a estratégia de 'substituição por uma réplica', *ReplaceByEquivalent* implementa a estratégia de 'substituição por um serviço similar', *Retry* implementa a estratégia de 'reexecutar' e *SkipOptionalService* implementa a 'saltar'. *SHDiagnostic* é uma classe abstrata que implementa métodos comuns às classes concretas e através da interface *Diagnostic* obriga as subclasses implementarem métodos importantes para a verificação e

execução da estratégia. Em especial, a estratégia de recuperação *ReplaceByEquivalent* necessita de outras classes para efetuar o processo de identificação de similaridade entre os serviços que são as classes *Degree* e *SimilarService*.

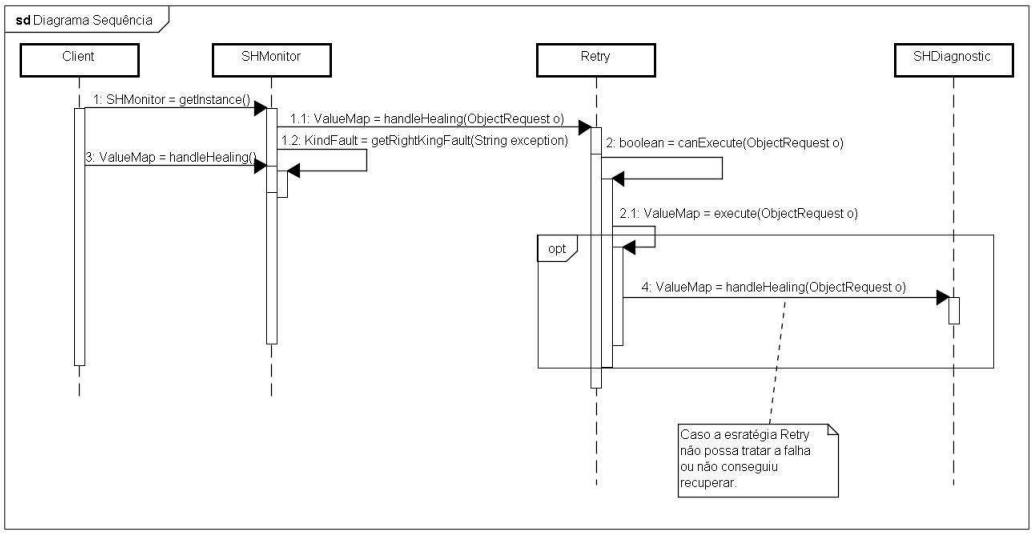

Figura 3. Diagrama de sequência do mecanismo de Auto-cura com início da sequência

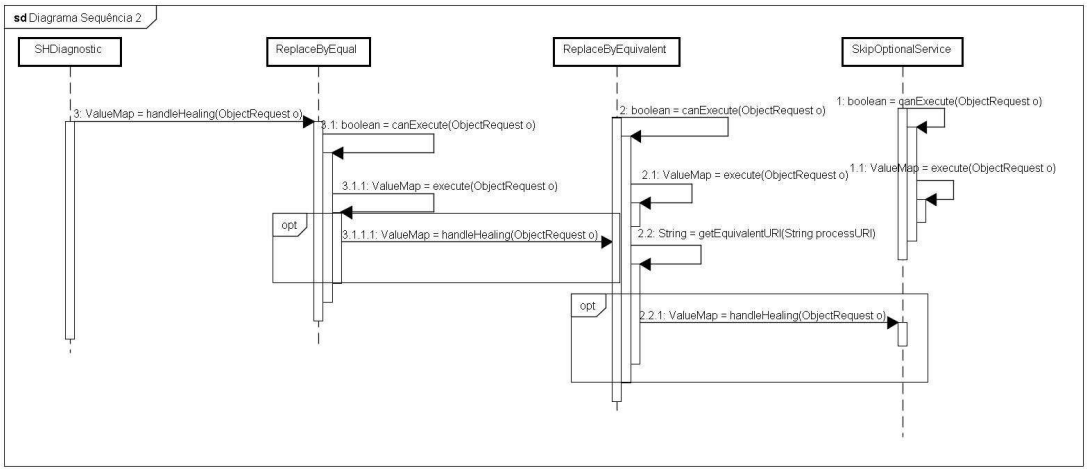

Figura 4. Diagrama de sequência do mecanismo de Auto-cura com continuação da sequência

5. Experimentos

Com o intuito de validar o mecanismo de auto-cura no OWL-S Composer 3.0, seis experimentos foram realizados. Os serviços utilizados foram obtidos do *Online Portal for Semantic Services* [Krug et al. 2009], um portal colaborativo sobre SWS que contêm um repositório de serviços Web semânticos.

5.1. Ambiente de desenvolvimento

O *OWL-S Composer 3.0* foi desenvolvido na versão Galileo 3.5.2 do Eclipse, utilizando os plugins EMF 2.5.0, GMF 1.2.0, GEF 3.5.1 e JET 1.1.0. As ferramentas integradas ao

ambiente e independentes da plataforma Eclipse utilizadas foram: a OWL-S API versão 1.1.0 e o *OWL-S Discovery* versão 1.0. Além disso, foi utilizado o JAX-SA com o objetivo de efetuar a conversão do arquivo WSDL para o OWL-S. No entanto, foi necessário efetuar algumas modificações na versão 1.6 que vinha sendo utilizada pelo *plugin*, visando corrigir um defeito encontrado nesse processo de conversão.

5.2. Serviços e Ontologias

Cada parâmetro utilizado nos serviços web é mapeado em uma classe de uma ontologia de domínio. É importante ressaltar que a utilização de ontologias é essencial para a fase de substituição por serviços semanticamente similares, pois é por meio delas que serão efetuadas as inferências necessárias para identificar correlação entre classes, conforme ilustra a Figura 5. *Carro* possui *Sedan* como subclasse e *Veículo* como superclasse.

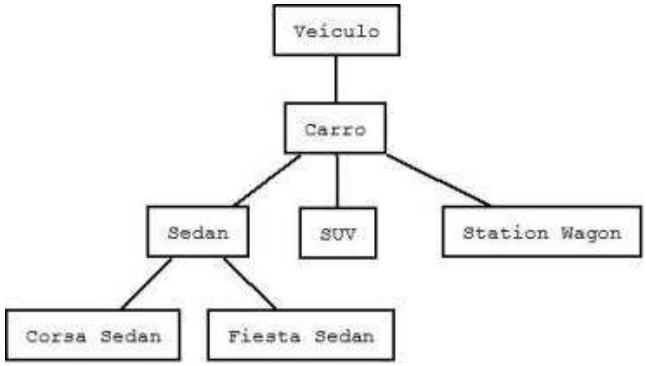

Figura 5. Ontologia *Auto*

5.3. Experimentos Realizados

O primeiro experimento realizado consiste na manipulação do plugin com o mecanismo de auto-cura sem falhas. O objetivo deste experimento foi analisar o funcionamento normal da composição de serviços web.

O segundo experimento analisou o funcionamento do mecanismo *Reexecução* (*Retry*), simulando o estado do sistema na presença de falha por temporização ou por omissão (de recebimento e envio), supondo que essas falhas sejam transientes ou intermitentes. Para simular a indisponibilidade temporária de um serviço, foi inserido um "loop infinito" dentro do serviço descrito para simular uma falhar por *timeout*. O tempo máximo (timeout) definido empiricamente foi de 60 segundos. Ao executar o estudo de caso, a falha por timeout foi acionada automaticamente pelo sistema e posteriormente o mecanismo de recuperação foi executado e o serviço reexecutado, ocasionando assim no sucesso da composição.

O terceiro experimento se refere à substituição por réplica (*ReplaceByEqual*), objetivando simular o estado do sistema na presença de falha por parada, timeout ou por omissão (de recebimento e envio), supondo que essas falhas sejam permanentes. Para simular uma parada no funcionando do serviço requisitado, o *loop* inserido no serviço como descrito anteriormente não foi removido. Uma réplica deste serviço foi inserida em um outro servidor sem o *loop infinito*. Ao executar o serviço, a falha por *timeout* foi acionada automaticamente e o mecanismo de recuperação tentou reexecutar o serviço,

porém houve reincidência da falha, culminando em um problema no acesso ao primeiro servidor. O segundo mecanismo foi analisado *ReplaceByEqual* e identificou que existia uma réplica. Dessa forma, realizou a substituição e retornou o resultado correto para a continuação da execução da composição.

O quarto experimento refere-se à substituição por similaridade (*ReplaceByEquivalent*), simulando o estado do sistema na presença de falha por parada, timeout ou por omissão (de recebimento e envio), supondo que essas falhas sejam permanentes. Há uma reincidência do erro no serviço, e na ausência da réplica, o mecanismo *ReplaceByEquivalent* é acionado. Uma busca foi realizada no repositório definido e identificou-se um serviço similar ao requisitado. Ao realizar a substituição, foi efetuada a execução, consequentemente o sucesso da composição.

O quinto experimento refere-se à substituição por réplica (*ReplaceByEquivalent*) com falha de resposta. Este experimento utiliza o mesmo mecanismo do experimento anterior, no entanto, simula o estado do sistema na presença de falha de resposta (de valor), supondo que essas falhas sejam permanentes. Um erro de tipo de retorno (de String para int) foi inserido no programa com o intuito de forçar que a falha ocorresse. Os mecanismos de *Retry* e *ReplaceByEqual* não foram acionados, uma vez que houve a execução correta e como a falha é específica da definição do serviço, a falha irá reincidir se for utilizada uma réplica. Neste caso, foi realizada uma busca em um repositório e identificou um serviço similar que possui como retorno o tipo String, como esperado pela composição. A execução após a substituição ocorreu com sucesso.

O sexto experimento retrata o Saltar (*SkipOptionalService*) simulando o estado do sistema na presença de falha por parada, timeout ou por omissão (de recebimento e envio), supondo que essas falhas sejam permanentes. Um parâmetro de saída da composição foi retirado demonstrando que um dos serviços da composição não será mais obrigatório para a sua execução final. Assim, os parâmetros de saída do serviço requisitado foram analisados e foi identificado automaticamente que apesar desse serviço possuir parâmetros de saída, esses não tinham ligação com qualquer outro serviço. Assim, este serviço é caracterizado automaticamente como opcional e o mecanismo "Saltar" permitiu que a execução da composição continuasse sem a obrigatoriedade do resultado desse serviço.

6. Resultados obtidos

A avaliação do *plugin OWL-S Composer 3.0* foi baseada nas funcionalidades implementadas e divididas em dois contextos: Ferramenta e Estrutura do mecanismo de auto-cura.

6.1. Ferramenta

a. Critérios analisados

Baseado em [Chafle et al. 2007], alguns requisitos mínimos para uma ferramenta de composição de serviços Web foram definidos: *Funcionalidade*, *Interface*, *Usabilidade* e *Integração*. Além desses, foi proposto por [Fonseca et al. 2009] o requisito *Legibilidade* e por [Sena et al. 2010] o requisito *Grau de similaridade*. Para avaliar a execução de serviços e o mecanismo de auto-cura, o presente trabalho também propõe mais dois requisitos *Execução* e *Monitoramento* que estão atrelados ao critério *Funcionalidade*, conforme abaixo:

1. **Funcionalidade:** O ambiente de desenvolvimento da ferramenta deve permitir o suporte de todo o ciclo de vida de uma composição de serviços Web, desde a criação até a publicação;
   1.1 **Execução:** A ferramenta deve permitir que o usuário execute composições, fornecendo o seu resultado final;
   1.2 **Monitoramento:** A ferramenta deve fornecer de forma transparente ao usuário mecanismos que permitam o aumento da confiabilidade das composições durante o processo de execução;
2. **Interface:** A interface deve se mostrar simples e funcional ao usuário, não necessitando que ele utilize recursos extras para implementar a composição de serviços;
3. **Usabilidade:** A ferramenta deve possibilitar que o usuário acesse de forma intuitiva todas as funcionalidades, através da utilização de recursos que permitem facilitar o uso, como botões e *menus*, além da intuitiva disponibilização na tela;
4. **Integração:** As funcionalidades que contribuem para a composição de serviços Web devem estar integradas, permitindo que os recursos da ferramenta sejam aproveitados;
5. **Legibilidade:** O código OWL-S gerado pela ferramenta deve ser legível e corresponder sintaticamente e semanticamente a um escrito de forma manual;
6. **Grau de similaridade:** A ferramenta deve possibilitar ao usuário selecionar os graus de similaridade que deseja e, além disso, deve apresentar qual o grau da composição resultante.

b. Avaliação destes Critérios no *OWL-S Composer 3.0*

O *OWL-S Composer 3.0* atendeu aos critérios descritos no tópico anterior como descrito a seguir:

1. **Funcionalidade:** A versão 3.0 do *OWL-S Composer* incorpora mais duas etapas do ciclo de vida de uma composição de serviço Web que são a invocação e monitoramento. A publicação ainda não é suportada, consequentemente, esse requisito ainda está sendo parcialmente atendido.
   1.1 **Execução:** Esse critério é atendido, pois a ferramenta permite a execução de composições de serviços gerados ou não pelo *plugin*.
   1.2 **Monitoramento:** Com a integração do mecanismo de auto-cura que permite a identificação dinâmica de falhas, seu diagnóstico e recuperação esse requisito foi atendido.
2. **Usabilidade:** Esse requisito é atendido pois são utilizados menus e botões padrões do ambiente Eclipse. Além disso, a disposição dos elementos gráficos nas janelas é padronizada assim como as novas funcionalidades integradas na versão 3.0.
3. **Interface:** Critério atendido pelo *OWL-S Composer 3.0*, pois não é necessária a utilização de ferramentas extras.
4. **Integração:** Critério atendido pelo *plugin* uma vez que são utilizados outros plugins do ambiente Eclipse na ferramenta proposta.
5. **Legibilidade:** Para avaliar esse requisito, foram gerados códigos OWL-S de composições de serviços e comparados manualmente com a especificação do padrão. Além disso, foi utilizado o método de validação de arquivos OWL-S disponibilizado na API e a própria execução do arquivo, que permite a identificação de possíveis inconsistências.

6. **Grau de similaridade:** Esse critério é atendido, pois o *plugin* oferece suporte ao filtro dos resultados da geração de composições semanticamente similares pelo grau de similaridade definido pelo usuário. Os graus de similaridade disponíveis são: *Exact*, *Plugin*, *Subsumes* e *Sibiling*.

c. Comparação com os trabalhos relacionados (vide seção 2)

Durante a avaliação do *OWL-S Composer 3.0* no contexto Ferramenta foram selecionados dois trabalhos relacionados: o *SHIWS* e o *OWL-S Composer 2.0*.

O *SHIWS* não oferece suporte ao ciclo de vida do desenvolvimento de composições de serviços web semânticos visto que o mesmo não possui informações semânticas. Além disso, não gera arquivos OWL-S e nem trabalha com os graus de similaridade. No entanto, *SHIWS* aumenta a confiabilidade dos serviços executados por possuir um mecanismo de auto-cura, apesar de não ser totalmente transparente ao usuário. É importante ressaltar também que como a forma de avaliação dos serviços é feita de forma manual para geração de arquivos auxiliares, o critério *usabilidade* fica comprometido.

A Tabela 1 apresenta o resultado da análise comparativa entre as ferramentas baseadas nos critérios apresentados. Diante desse contexto, é possível afirmar que o *OWL-S Composer 3.0* é uma ferramenta adequada para desenvolvimento de composições de serviços permitindo a sua execução com uma maior confiabilidade.

	SHIWS	*OWL-S Composer 2.0*	*OWL-S Composer 3.0*
1. Funcionalidade	Parcial	Parcial	Parcial
1.1 Execução	Sim	Não	Sim
1.2 Monitoramento	Parcial	Não	Sim
2. Interface	Sim	Sim	Sim
3. Usabilidade	Não	Sim	Sim
4. Integração	Sim	Sim	Sim
5. Legibilidade	Não	Sim	Sim
6. Grau de similaridade	Não	Sim	Sim

Tabela 1. Tabela comparativa no contexto ferramenta

6.2. Estrutura do mecanismo de auto-cura

a. Critérios analisados

Alguns requisitos foram selecionados para efetuar um comparativo em relação ao mecanismo de auto-cura desenvolvido.

1. **Estrutura extensível:** Visa analisar se a arquitetura permite que novas funcionalidades relacionadas ao mecanismo de auto-cura sejam inseridas de forma centralizada e que não necessite de muito conhecimento do sistema, facilitando a sua manutenção. Além disso, que possua baixo acoplamento e alta coesão entre classes.
2. **Abordagem prática:** Tenha ocorrido implementação e experimentos visando validar a abordagem.
3. **Transparência ao usuário:** Permite que o usuário usufrua de todos os mecanismos de auto-cura sem a necessidade deste realizar qualquer procedimento extra ao uso normal do sistema.

4. **Descoberta por similaridade:** Permite que o sistema flexibilize o processo de substituição de serviços utilizando artifícios semânticos, visando alcançar um maior número de serviços desejáveis.

b. Avaliação destes Critérios no *OWL-S Composer 3.0*

O *OWL-S Composer 3.0* atendeu aos critérios descritos no tópico anterior como descrito a seguir:

1. **Estrutura extensível:** A arquitetura foi projetada utilizando os padrões de projeto com o intuito de gerar uma estrutura sólida e de fácil manutenção.
2. **Abordagem prática:** Foi desenvolvido um estudo de caso com o objetivo de avaliar de uma maneira prática a abordagem proposta.
3. **Transparência ao usuário:** Não é necessário o uso de qualquer arquivo externo ou modificações pelo usuário visando usufruir dos mecanismos de auto-cura.
4. **Descoberta por similaridade:** Foi desenvolvido o mecanismo de *ReplaceByEquivalent* que permite usufruir das informações semânticas dos arquivos visando efetuar o processo de substituição destes serviços.

c. Comparação com os trabalhos relacionados (vide seção 2)

Dois trabalhos para análise do contexto do mecanismo de auto-cura foram selecionados: o *SH-BPEL* e o "Mecanismos de recuperação para SWS". Através do comparativo com essas ferramentas, foi possível evidenciar as vantagens do uso da versão 3.0 do *plugin* OWL-S Composer.

O *SH-BPEL* possui uma arquitetura que tem como objetivo facilitar a inserção de novos mecanismos e a realização de manutenções. No entanto, não possui suporte a semântica e necessita que o usuário efetue mapeamentos extras no arquivo do serviço. Essa mesma interferência é necessária em "Mecanismos de recuperação para SWS" e, adicionalmente, este trabalho foi teórico e o algoritmo proposto não se preocupa com a estrutura do mecanismo.

A Tabela 2 apresenta o resultado da análise comparativa entre os trabalhos através dos critérios apresentados anteriormente. Diante desse contexto, é possível afirmar que o *OWL-S Composer 3.0* possui um mecanismo de auto-cura que é adequado para o aumento da confiabilidade das composições de serviços Web.

	SH-BPEL	Mecanismos de recuperação	OWL-S Composer 3.0
1. Estrutura extensível	Sim	Não	Sim
2. Abordagem prática	Sim	Não	Sim
3. Transparência ao usuário	Não	Não	Sim
4. Descoberta por similaridade	Não	Sim	Sim

Tabela 2. Tabela comparativa no contexto estrutura do mecanismo de auto-cura

A versão 3.0 do *plugin* e as anteriores estão disponíveis para download no site https://sourceforge.net/projects/owl-scomposer/.

7. Conclusão

A incorporação de características semânticas aos serviços web facilita o processo de descoberta e contribui, principalmente, para as estratégias de substituição de serviços,

visando o aumento da confiabilidade e disponibilidade na execução de composições de serviços web. Os sistemas autonômicos, especificamente, auto-cura(self-healing) vem sendo integrada a sistemas de informação com o intuito de aumentar a confiabilidade, recuperar as falhas e principalmente manter o sistema em funcionamento de maneira correta.

Este trabalho propôs uma nova versão do *OWL-S Composer* que integra mecanismos de auto-cura ao processo de execução de composições de SWS. Desta forma, o *OWL-S Composer 3.0* oferece suporte a mais duas etapas do ciclo de vida de uma composição de serviços Web: a execução e o monitoramento.

Como trabalho futuro, pretende-se desenvolver uma modelagem com suporte às composições automáticas no *OWL-S Composer* e ampliar os testes de desempenho do mecanismo de auto-cura desenvolvido.

Referências

Chafle, G., Das, G., Dasgupta, K., Kumar, A., Mittal, S., Mukherjea, S., and Srivastava, B. (2007). An integrated development enviroment for web service composition. pages 839–847, Salt Lake City, Utah, USA.

Denaro, G., Pezze, M., and Tosi, D. (2007). Shiws: A self-healing integrator for web services. In *ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 55–56, Washington, DC, USA. IEEE Computer Society.

Fonseca, A. A., Claro, D. B., and Lopes, D. C. P. (2009). Gerenciando o desenvolvimento de uma composição de serviços web semânticos através do owl-s composer. In *Proceedings of the 5th Brazilian Symposium of Information Systems (SBSI2009)*, Brasilia, DF, Brazil.

Horn, P. (2001). Autonomic computing: Ibm's perspective on the state of information technology. http://www.research.ibm.com/autonomic/.

Huebscher, M. C. and McCann, J. A. (2008). A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28.

Krug, A., Küster, U., and Ruhmer, C. (2009). Opossum online portal for semantic services. http://fusion.cs.uni-jena.de/opossum/index.php. Último acesso em 03 de julho de 2010.

McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language. http://www.w3.org/TR/owl-features/. Último acesso em 20 de novembro de 2009.

Modafferi, S., Mussi, E., and Pernici, B. (2006). Sh-bpel: a self-healing plug-in for ws-bpel engines. In *MW4SOC '06: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 48–53, Melbourne, Australia. ACM.

Sena, V. A., Claro, D. B., Amorim, R., and Lopes, D. (2010). Similaridade semântica na composição de sistemas de informação através dos serviços web. In *VI Simpósio Brasileiro de Sistemas de Informação (SBSI 2010)*, Marabá/PA.

Wiesner, K., Vaculín, R., Kollingbaum, M. J., and Sycara, K. P. (2008). Recovery mechanisms for semantic web services. In *DAIS*, pages 100–105, Oslo, Norway.