# Uma Arquitetura para Composição de Serviços com Modelos de Interação Heterogêneos

Alexis Huf
Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Florianópolis, SC - Brasil
alexis.huf@posgrad.ufsc.br

Frank Siqueira
Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Florianópolis, SC - Brasil
frank.siqueira@ufsc.br

## ABSTRACT

Current Web-based Services are highly heterogeneous not only on data but also with regard to service interaction. Despite their heterogeneity, composition of these services is required in order to achieve additional functionality. Description languages and composition algorithms for heterogeneous services have been recently proposed. However, existing techniques do not take the Publish/Subscribe paradigm into consideration or do not offer sufficient support for interaction through hypermedia controls as required by the REST architectural style. Analyzing current heterogeneous composition techniques, this dissertation identifies limitations hindering automatic composition of these heterogeneous services. It proposes an architecture and two techniques, adaptiveness and replication, that when combined with a graph-based composition algorithm, enable planning and execution of heterogeneous compositions. In addition to supporting heterogeneity, our prototype achieved similar or better performance than two state-of-the-art homogeneous composition algorithms in most cases.

## KEYWORDS

service composition, REST, publish-subscribe, events, heterogeneous systems

## 1 INTRODUÇÃO

No contexto de sistemas computacionais distribuídos, serviços são elementos computacionais auto-descritivos que permitem que organizações exponham suas competências programaticamente, viabilizando a composição de aplicações distribuídas com baixo custo [19]. Uma categoria amplamente utilizada de serviços são os *Web Services*, assim denominados pelo uso de tecnologia Web para hospedagem e comunicação. Os *Web Services* podem ser subdivididos em duas categorias: aqueles que utilizam o protocolo de comunicação SOAP nos termos do *Web Services Basic Profile*[30]; e os que adotam (total ou parcialmente) o estilo arquitetural REST (*REpresentational State Transfer*) [7] empregando diretamente o protocolo HTTP.

Analisando palavras-chave associadas aos serviços registrados no ProgrammableWeb[1], um diretório comunitário listando serviços, constatou-se que apenas 10,03% dos serviços utilizam o protocolo SOAP, enquanto 73,63% são fornecidos diretamente sobre HTTP (os 16,34% restantes não puderam ser classificados)[2]. No entanto, as restrições impostas pelo estilo arquitetural REST são frequentemente aplicadas parcialmente [16]. Essa prática comum levou ao desenvolvimento de modelos de maturidade para serviços REST [26], que classificam os serviços com base na conformidade para com as restrições impostas por esse estilo arquitetural.

Dentre as restrições REST, a restrição HATEOAS (*Hypermedia As The Engine Of Application State*) implica em uma diferença drástica quanto à interação se comparada com serviços SOAP: serviços devem expor as próximas ações possíveis ao cliente através de controles hipermídia (e.g., *links* ou formulários), que devem ser considerados pelo cliente ao determinar sua próxima interação com o serviço [7]. Essa restrição é essencial para alcançar as vantagens proporcionadas pelo estilo REST, como manutenibilidade e interoperabilidade. Além disso, ignorar esses controles corresponde a ignorar a documentação do serviço, podendo levar a resultados incompletos ou incorretos.

Além de SOAP e REST, outro tipo de serviço comum são os Serviços Orientados a Eventos (SOE). Um evento representa uma mudança de estado em alguma entidade ou algum fato relevante. Eventos são disseminados através de paradigmas de comunicação como *Publish/Subscribe* e Notificação [6]. Esse tipo de serviço é altamente heterogêneo. Em um contexto SOAP, podem ser suportados por um ESB (*Enterprise Service Bus*) ou por padrões [10]. No contexto de serviços REST, pode-se citar *polling*, padrões específicos [8, 11] e funcionalidades de *streaming* do protocolo HTTP 2 [5].

Dados e funcionalidades de diferentes serviços podem ser combinados em um serviço composto (ou composição de serviço) [19]. É possível construir manualmente uma composição combinando serviços desses três tipos. No entanto, essa abordagem demanda recursos humanos qualificados para implementação e manutenção da composição. A construção automática de composições de serviços heterogêneos busca atenuar esses custos, mas ainda é um problema em aberto. Há trabalhos na literatura propondo descrições comuns de serviços [25], geração de *proxies* [22, 31], *Middlewares* [9, 12] e linguagens de processo ou extensões [20, 28].

Visto a popularidade de serviços REST, os cenários que se beneficiam do paradigma *Publish/Subscribe* e o foco da literatura em composição de serviços SOAP, há demanda para técnicas de composição heterogênea. No entanto, SOAP, REST e SOE impõem modelos de interação, que restringem não apenas protocolos de comunicação, mas também as sequências de interação válidas entre um cliente e um serviço e a forma de descrição do serviço ao cliente. A análise

[1] http://www.programmableweb.com/

[2] https://github.com/IvanGoncharov/ProgrammableWeb

In:XVII Workshop de Teses de Dissertações (WTD 2017), Gramado, Brasil. Anais do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2017.

ISBN 978-85-7669-380-2.

das abordagens de composição heterogênea revela duas limitações. Primeiro, a restrição HATEOAS é violada ao ser ignorada ou ao construir composições assumindo a presença de controles hipermídia durante a execução do serviço composto. Segundo, nenhuma abordagem tenta simultaneamente cobrir SOAP, REST e SOE.

Considerando apenas REST, Verborgh et al. [32] apresentam um algoritmo de composição que suporta HATEOAS adequadamente, intercalando execução e planejamento da composição. Já no caso de SOE, há abordagens que contemplam composições heterogêneas com SOAP [9, 28]. Embora uma revisão da literatura tratando de composições heterogêneas tenha encontrado apenas duas técnicas de composição automática dentre 30 levantadas, há extensa bibliografia considerando apenas serviços SOAP [1, 24]. Frente a essas observações, este trabalho busca confirmar a seguinte hipótese:

**Hipótese**: Composições de serviços heterogêneos, especificamente SOAP, REST e SOE, podem ser construídas e executadas automaticamente, garantindo que as interações realizadas com os serviços existentes estejam em conformidade com os respectivos modelos de interação.

## 2 CONTRIBUIÇÕES ESPERADAS

O objetivo deste trabalho consiste em definir uma arquitetura que permita o planejamento e execução de composições de serviços com modelos de interação heterogêneos (SOAP, REST e SOE), respeitando todas as restrições impostas pelos modelos de interação.

Além da própria arquitetura, as seguintes contribuições são esperadas como resultado desta pesquisa.

(1) Análise quanto aos tipos de heterogeneidades suportadas pela arquitetura, e como elas são suportadas;
(2) Algoritmos para planejamento e execução de composições heterogêneas;
(3) Viabilização do uso de composição automática de serviços, não apenas na academia, mas também na indústria, através de uma implementação da arquitetura.

## 3 MÉTODO

Esta pesquisa busca confirmar a hipótese enunciada na Seção 1 atingindo os objetivos através dos seguintes passos:

(1) Conduzir uma RSL (Revisão Sistemática da Literatura) com escopo em composições heterogêneas. Essa revisão fornecerá subsídios para a catalogação das heterogeneidades e resultará na classificação das técnicas existentes para sua solução (Seção 5);
(2) Categorizar as heterogeneidades, considerando classificações prévias encontradas na literatura. Tal categorização permitirá relacionar heterogeneidades e técnicas;
(3) Levantar na literatura de composição automática de serviços recente requisitos funcionais e não funcionais: a expressividade de linguagens de descrição de serviços ou de composição utilizadas, o tempo necessário para o planejamento de composições e o método de avaliação utilizado nas propostas;
(4) Definir uma arquitetura, aplicando um subconjunto das técnicas identificadas, para suportar composições heterogêneas (Subseção 6.1);
(5) Projetar algoritmos para planejamento e execução de composições heterogêneas na arquitetura definida, com expressividade equivalente ao estado da arte (Subseção 6.2);
(6) Implementar um protótipo da arquitetura e dos algoritmos (Seção 7);
(7) Avaliar experimentalmente a viabilidade prática da arquitetura e do algoritmo, comparando o tempo de planejamento (cf. item 3) com o estado da arte (Seção 7);
(8) Avaliar a expressividade e aplicabilidade (cf. item 3) dos algoritmos através de estudos de caso.

No estágio atual do trabalho, as etapas 3 e 8 estão em uma fase inicial de desenvolvimento, enquanto as demais etapas estão sendo executadas de forma iterativa, mas não foram finalizadas.

## 4 FUNDAMENTAÇÃO TEÓRICA

Atualmente, SOA (*Service Oriented Architecture*) [19] é a principal referência para construção de aplicações distribuídas com serviços. Esta arquitetura identifica três elementos de processamento: o Serviço, que publica sua descrição em um repositório; o Repositório que permite buscas nas descrições de serviços registrados; e o Cliente, que localiza o serviço no repositório e o utiliza dinamicamente.

Um dos principais usos de SOA é viabilizar a composição de serviços. Uma das maneiras para construção de serviços compostos é através de um processo, onde serviços são invocados de acordo com uma estrutura lógica de controle (*if*, *sequence*, *repeat*, etc.) e um fluxo de dados bem definidos. Um exemplo de linguagem para esse fim é BPEL (*Business Process Execution Language*) [18].

Outra forma de obter serviços compostos é através de composição automática de serviços [17]. Algoritmos para esse fim tomam como entrada descrições dos serviços disponíveis e um objetivo a ser realizado (ou uma descrição do serviço composto). Como resultado, obtêm-se uma especificação de processo, um serviço ou os resultados de sua execução. Técnicas de composição levam em consideração ao menos um de três aspectos dos serviços [4]: QoS (seleção de serviços existentes para maximização de QoS do serviço composto), I/O (como obter saídas desejadas a partir de entradas conhecidas) e pré/pós-condições (como obter pós-condições, expressas em alguma lógica a partir das condições atuais).

Considerando os três tipos de serviço abordados nesta pesquisa, existem diferentes restrições quanto à interação entre um cliente e o serviço. No presente trabalho, adota-se o termo "modelo de interação" para referir-se a essas restrições.

Um serviço usualmente implementa regras de negócio. Um aspecto do serviço a ser modelado são as sequências válidas de operações. Por simplicidade, seja $\mathcal{L} = (Q, q_0, \Sigma, \delta)$ um LTS (*Labeled Transition System*) que documenta a interação com um serviço qualquer. $Q$ é conjunto de estados, $q_0$ é o estado inicial, $\Sigma$ representa o conjunto de ações possíveis no serviço e $\delta(q, \alpha)$ indica o estado sucessor de $q$ após execução da ação $\alpha$. No caso de serviços SOAP, $\delta$ deve ser formalizado como parte da descrição do serviço ou, mais frequentemente, ser parte da programação de seus clientes. No caso de serviços REST (mesmo que não adotem HATEOAS), uma ação $\alpha$ tem a forma $(v, R, e)$ onde $v$ é um verbo HTTP, $R$ um recurso identificado por URI e $e$ uma representação opcional cuja necessidade é determinada por $v$. Adicionalmente, denota-se a resposta obtida por $\rho(\alpha)$. No caso da restrição HATEOAS, $\delta$ não é exposto ao cliente.

Ao invés disso, após uma ação $\alpha$, todo $\alpha'$ para o qual $\delta(q, \alpha')$ está definido é incluído em $\rho(\alpha)$ na forma de controles hipermídia.

Eugster et al. [6] detalham os paradigmas de comunicação *Publish/Subscribe* e Notificação. No contexto de serviços, a adoção de um desses paradigmas implica que o SOE produzirá eventos a serem consumidos por clientes interessados. Em SOA não há mecanismos para que os serviços busquem clientes, logo, clientes interessados nesses eventos devem se registrar no serviço, que então tomará a iniciativa de produzir os eventos. O modelo de interação associado a SOE consiste em uma requisição de assinatura, seguida por um número indeterminado de respostas. O modelo de interação se aplica mesmo que no mecanismo de entrega o cliente tome a iniciativa (e.g., *polling*).

## 5 TRABALHOS RELACIONADOS

Foi realizada uma RSL (Revisão Sistemática da Literatura) buscando responder a seguinte pergunta: "Quais são as abordagens utilizadas para realizar ou viabilizar a composição de serviços heterogêneas, considerando serviços SOAP, REST e orientados a eventos?". A *string* de busca é dividida em duas partes e se aplica a título, resumo e palavras-chave. Na primeira parte busca-se artigos que explicitamente mencionam modelo de interação através da ocorrência simultânea de três palavras-chave: *service*, *heterogeneous* e *Interaction Model*. Já a segunda parte busca trabalhos que tratem composição, descrição ou descoberta de serviços em mais de um tipo de serviço, utilizando combinações de palavras-chave que caracterizam serviços SOAP, REST e orientado a eventos.

A *string* de busca foi executada em três bases de dados[3] visando artigos publicados em conferências ou periódicos, submetidos a revisão por pares, em língua inglesa, publicados desde janeiro de 2008. Foram obtidos 197 documentos não duplicados. Esses 197 documentos foram selecionados manualmente com base em seus resumos (153 exclusões) e posteriormente pelo texto completo (14 exclusões), resultando em apenas 30 documentos com contribuição para a pergunta de pesquisa. Desses 30 documentos, foram identificados 5 grupos principais de técnicas propostas para tratamento da heterogeneidade, que serão descritos a seguir.

(1) **Descrição comum** Uma única linguagem para descrição de serviços heterogêneos, viabilizando a construção de um algoritmo de composição heterogêneo [25]. No entanto, ainda é necessário resolver heterogeneidades durante a invocação. As abordagens se subdividem em:
   (a) **Semântica** Baseadas em ontologias.
   (b) **Sintática** Descrições que não incluem capacidade de inferência e não almejam ser auto-descritivas.
   (c) **Metamodelo** Descrições de serviços são vistas como modelos dos serviços e modeladas por um metamodelo [27]. Metamodelos viabilizam traduções entre diferentes linguagens de descrição.

(2) **Proxies** São criados serviços cuja função é acesso a serviços com outro modelo de interação. Nessa categoria, os trabalhos tratam apenas de heterogeneidade SOAP/REST.

(3) **Middleware** Um *middleware* seleciona componentes adequados ao tipo de serviço a ser invocado.

(4) **Linguagem de processo** Se subdivide em:
   (a) **Extensões** Uma linguagem de processo, como BPEL [18], pode ser estendida para suportar diretamente outros tipos de serviço. Extensões para REST adicionam instruções correspondentes a métodos HTTP [13, 20] e extensões para SOE, instruções para disparo ou recepção de eventos.
   (b) **Nova linguagem** Similar a Extensões, no entanto toda uma linguagem de processo é proposta.
   (c) **Linguagem existente** BPEL [18] oferece suporte básico a eventos [2], e alguns trabalhos contornam limitações com serviços de suporte.

(5) **Implementação Direta** É possível implementar a composição diretamente em uma linguagem de programação.

Roman et al. [25] apresentam o uso de uma descrição comum semântica como solução para heterogeneidade SOAP/REST. Descrições de serviço recebem anotações semânticas usando SAWSDL [15] ou hRESTS [14] e são interpretadas de acordo com o MSM (*Minimal Service Model*) [21]. O resultado é uma descrição semântica uniforme, que dado um *framework* capaz de invocar adequadamente cada tipo de serviço usando as descrições originais, possibilita o projeto de algoritmos de descoberta e composição. No entanto, não há suporte para descrição de controles hipermídia, o que impede que um algoritmo de composição os considere.

Upadhyaya et al. [31] tratam da geração de serviços REST como *proxies* para serviços SOAP. Aplicando processamento de linguagem natural aos documentos WSDL, a ferramenta extrai sugestões de recursos a serem expostos. As sugestões são apresentadas para o usuário, que pode editar a configuração de como os recursos extraídos serão expostos pelo *proxy*. Há suporte para *links* entre classes de recursos, mas não é detalhado como construir *links* para recursos individuais (URIs com parâmetros). Além disso, a aderência do serviço *proxy* aos princípios REST depende da revisão do especialista, que deve considerar o funcionamento do serviço SOAP.

Pautasso [20] propõe extensões à linguagem BPEL para composição de serviços REST. *Activities* correspondentes aos métodos HTTP permitem a invocação de serviços. Adicionalmente, *handlers* permitem que um processo BPEL seja exposto como um recurso. Variáveis podem ser usadas para extrair URIs de respostas e as utilizar em requisições subsequentes. A restrição HATEOAS é violada pois o processo considera apenas os controles hipermídia presentes em sua definição, gerando acoplamento entre processo e serviços. A possível ausência de controles durante a execução precisa ser considerada no processo, e controles não previstos são ignorados.

As abordagens que contemplam SOE são majoritariamente baseadas em *middleware*, em alguns casos com extensões na linguagem de processo ou com linguagens novas. Georgantas et al. [9] propõem a integração entre três paradigmas de comunicação, requisição-resposta, espaço de tuplas e *Publish/Subscribe*. São propostos modelos para cada paradigma e um quarto modelo básico contendo apenas as primitivas *post* (envio) e *get* (recepção). Um ESB (*Enterprise Service Bus*) capaz de executar *workflows* envolvendo esses três paradigmas é proposto. Como é o caso de todos os trabalhos tratando de eventos, REST não é considerado.

A Tabela 1 analisa a cobertura dos trabalhos selecionados na RSL quanto à heterogeneidade dos modelos de interação. Na Tabela 1, HATEOAS é considerado separadamente devido aos poucos

[3] http://scopus.com/, http://dl.acm.org/ e http://ieeexplore.ieee.org/

**Tabela 1: Modelos de interação suportados.**

| Quantidade | SOAP | REST | Eventos | HATEOAS |
|---|---|---|---|---|
| 18 | ✓ | ✓ | x | x |
| 4 | ✓ | ✓ | x | Estático |
| 2 | ✓ | ✓ | x | Evita |
| 6 | ✓ | x | ✓ | |

trabalhos que suportam essa restrição, mesmo que parcialmente, enquanto que a posição declarada pelo próprio trabalho em relação ao suporte a REST é apresentada na coluna homônima. Duas conclusões seguem da Tabela 1. Primeiro, não há suporte simultaneamente aos três tipos de serviço. Foram identificados 18 trabalhos que suportam composição entre SOAP e REST, sem considerar a restrição HATEOAS, enquanto 6 trabalhos suportam composição entre SOAP e Eventos. Segundo, HATEOAS não é completamente suportado. Os quatro trabalhos marcados com "Estático" consideram controles hipermídia estaticamente, não permitindo que a presença ou ausência desses controles altere o plano da composição durante a execução. Já as duas abordagens indicadas com "Evita" não tratam de composição, mas sua aplicação não impede a realização de composições que suportem HATEOAS.

Além dos trabalhos da RSL, há dois trabalhos na literatura que não consideram modelos de interação heterogêneos, mas cujas contribuições são incorporadas neste trabalho.

Verborgh et al. [32] apresentam um algoritmo de composição chamado *Pragmatic Proof* que efetivamente considera a restrição HATEOAS. Descrições de serviço RESTdesc [34], ao serem processadas pelo *reasoner* N3 EYE [33] permitem que o problema de planejar uma composição seja reduzido ao de prova de teorema na lógica N3. Para suporte a HATEOAS, após cada interação com um serviço, os resultados obtidos são confrontados com os resultados documentados, e a composição é replanejada. O replanejamento garante que a interação seja dirigida pelos controles hipermídia. No entanto, o replanejamento completo e o uso do *reasoner* N3 são fatores que afetam negativamente o desempenho.

Rodriguez-Mier et al. [23] apresentam o ComposIT, que reduz o problema de composição de *Web Services* ao de busca de um caminho de custo mínimo em grafos, modelando cada vértice como um conjunto de serviços. O bom desempenho do algoritmo, avaliado com o *benchmark* WSC'08 [3], é obtido através da identificação de serviços e estados de busca redundantes. Embora o algoritmo tenha sido aplicado [24] em conjunto com uma abordagem de descrição comum, o MSM [25], não há suporte a HATEOAS ou a invocação de serviços heterogêneos. Outra limitação do algoritmo é o uso apenas de entradas e saídas dos serviços no planejamento, sem considerar pré- e pós-condições ou ações associadas aos serviços.

## 6 ESTADO ATUAL DO TRABALHO

### 6.1 Arquitetura

Tendo em vista a heterogeneidade dos modelos de interação entre serviços existentes, propõe-se uma arquitetura de composição e execução, ilustrada na Figura 1. *Tradutores* convertem descrições de serviço existentes em uma descrição intermediária baseada em RDF. Devido ao uso de RDF, não há prejuízo em utilizar outros

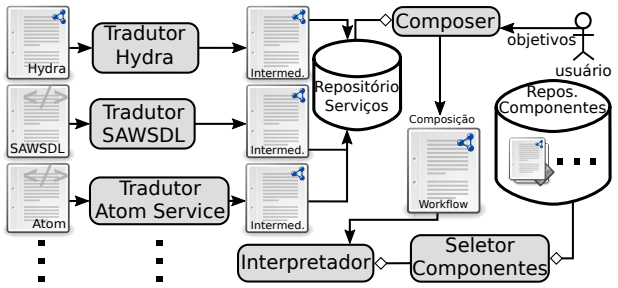



**Figura 1: Arquitetura de Composição.**

vocabulários, como HTTP-in-RDF[4] para descrever aspectos não cobertos pela linguagem intermediária. Essas descrições RDF são armazenadas em um *Repositório de Serviços*, onde o *Composer*, considerando apenas a linguagem intermediária, as busca para construir composições que atendam os objetivos de um *Usuário*. As composições resultantes são representadas em uma linguagem de *workflow* interpretável pela própria arquitetura.

A arquitetura aplica duas das cinco técnicas identificadas na Seção 5. Os componentes constituem uma abordagem de *middleware*, ao proverem um conjunto de funcionalidades básicas: (des)serialização de dados e envio/recebimento de mensagens. O *Interpretador*, ao processar o *workflow*, utiliza o *Seletor de Componentes* para selecionar componentes adequados considerando os serviços envolvidos na ação. Por exemplo, pode ser selecionado um componente específico para o envio de requisições HTTP, devido às especificidades introduzidas pelo *Tradutor*, mas ignoradas pelo *Composer* ao incluir a ação *Send* no *workflow*.

O segundo tipo de abordagem aplicada é a de descrição comum. Em conjunto com OWL, a linguagem de descrição intermediária permite a resolução de heterogeneidades na descrição das interações possíveis com um serviço e no seu protocolo de aplicação [29]. No entanto, esta cobre apenas aspectos centrais e, como em outras abordagens encontradas na literatura, a formalização de ações e de protocolos é delegada a ontologias externas[5].

A linguagem intermediária captura as interações possíveis com um serviço na forma de mensagens. Essa linguagem consiste em uma ontologia cujas principais classes são *Variable* e *Message*. Uma mensagem possui remetente e destinatário, condição de ocorrência, e variáveis que fazem parte da mensagem. A condição de ocorrência pode ser espontânea ou uma reação, com uma certa cardinalidade à outra mensagem. Já as variáveis possuem um tipo, uma representação (conceito equivalente ao usado em REST), um valor, e podem ser constituídas recursivamente por outras variáveis. Por exemplo, em um serviço REST cada método HTTP aplicável origina uma mensagem para a resposta tendo como condição uma reação à mensagem de requisição. Já em uma interação com um SOE, a mensagem que representa a notificação tem como condição de ocorrência uma reação de cardinalidade *many*.

Uma terceira técnica, não identificada na RSL da Seção 5, consiste em adaptar o algoritmo de composição aos modelos de interação heterogêneos. Considerando os modelos de interação e a discussão

[4] https://www.w3.org/TR/HTTP-in-RDF/

[5] A conversão entre descrições desses aspectos é desafiadora, uma vez que diferentes formalizações não possuem expressividade equivalente.

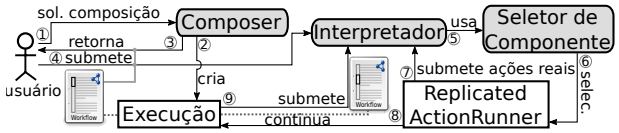



**Figura 2: Fluxo da composição replicada e adaptativa.**

em [32], HATEOAS demanda que o algoritmo de composição seja adaptativo, considerando os resultados de cada interação ao determinar os próximos passos. De modo análogo, um SOE necessita que o algoritmo de composição considere a possibilidade de um número desconhecido, ou ilimitado, de respostas a uma única mensagem.

## 6.2 Replicação e Adaptatividade

Para suportar as heterogeneidades discutidas anteriormente, este trabalho propõe, respectivamente, *Replicação* da execução e *Adaptatividade* da composição. A técnica de Adaptatividade demanda um algoritmo de composição que opere em um *grafo de estados* que satisfaça três características: (1) Estados (nós) consistem em um conjunto de serviços; (2) A aresta $(u_1, u_2)$ corresponde a invocar os serviços do estado $u_1$, atingindo as condições necessárias para que os serviços em $u_2$ possam ser posteriormente invocados; (3) Há listas, possivelmente vazias, de alternativas para serviços e estados.

Tal algoritmo de composição baseado em grafo de estados produz como resultado uma lista de arestas a serem seguidas. Cada aresta corresponde a uma sequência contínua de ações executáveis pela arquitetura, que são dividas em *slices*. Um *slice* inclui no máximo uma ação de recepção, que, se presente, será sempre a última.

A Figura 2 mostra uma visão geral da composição adaptativa. Após submissão dos objetivos e entradas da composição, o *Composer* cria um objeto *Execução* para manter o estado (representado através da descrição intermediária) e o vincula ao *workflow* retornado, que contém em vez da composição inteira, apenas as ações do primeiro *slice*, encapsuladas em uma ação "replicada". Ao encontrar essa ação especial, o *Interpretador* seleciona um componente apropriado que submete as verdadeiras ações do *slice* para execução, e replica o objeto execução durante interações com um SOE (nas quais a ação de recebimento resulta em múltiplas mensagens). Outro papel desse componente é continuar o objeto *Execução*. Esta operação consiste em validar os resultados obtidos frente aos planejados. Caso a validação seja negativa, o algoritmo de composição é utilizado para replanejar a composição partindo do estado atual da execução.

A validação verifica se todas as pós condições de uma aresta no grafo de estado são válidas, e é executada a cada *slice* dessa. Ao detectar ao menos uma falha irrecuperável, o algoritmo constrói o conjunto $F$ de pares de mensagens responsáveis pelas falhas. A adaptação do grafo consiste em substituir $u_1$ por um novo estado $\alpha$.

Há três possibilidades para $\alpha$. Caso todos os serviços em $F$ tenham alternativas, estas constituirão o novo estado. Caso contrário, cogita-se utilizar um estado alternativo como $\alpha$. Como último recurso, um estado vazio é usado como $\alpha$. Após a substituição de $u_1$, as arestas de entrada e saída de $\alpha$ são reavaliadas e removidas caso alguma restrição seja violada com $\alpha$. Caso $\alpha$ seja vazio, permanecem apenas as arestas que retornam para estados predecessores.

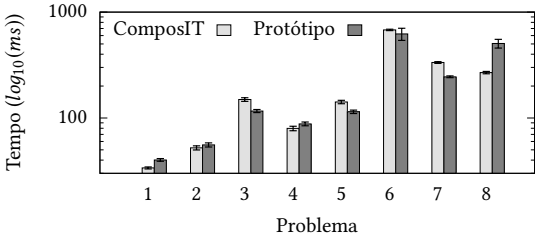
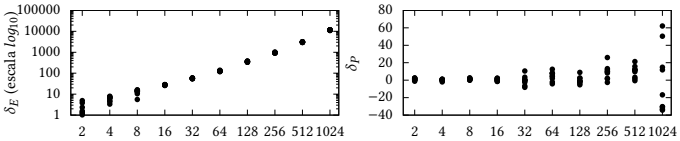



**Figura 3: Tempo de planejamento por problema do WSC'08.**




**(a) Efeito para o EYE** **(b) Efeito para o Protótipo**

**Figura 4: Efeito no tempo ao mudar de 1 para 3 parâmetros.**

## 7 RESULTADOS PRELIMINARES

A arquitetura foi avaliada através de experimentos executados em um Intel i5 a 2.5GHz com 4GB disponíveis para *heap* da JVM.

O primeiro conjunto de experimentos, assim como o ComposIT, utilizou o *Web Services Challenge 2008* (WSC'08) [3]. O WSC consiste de 8 problemas, cada um contendo um repositório de serviços, um conjunto de entradas e um conjunto de saídas desejadas. Algoritmos de composição devem planejar uma série de invocações aos serviços para obter as saídas a partir das entradas. ComposIT e Protótipo foram comparados pelo Intervalo de Confiança (IC) da média dos tempos de planejamento. Visando uma margem de erro de até 5% da média, foram realizadas 48 execuções para o ComposIT e 31 para o Protótipo. Esses valores foram calculados a partir de uma amostra piloto de 9 execuções. Não foi possível alcançar a margem de erro desejada nos problemas 6 e 8 (Protótipo) e 1, 2 e 4 (ComposIT), pois para tal seria necessário um alto número de execuções.

A Figura 3 sumariza a média do tempo necessário pelo ComposIT e pelo Protótipo para obter a solução de cada um dos 8 problemas no WSC'08. As barras de erro em cada coluna representam o IC de 95%. Devido a melhorias na implementação, a média do protótipo é menor em 3 problemas, e os ICs se intersectam para outros 2 problemas. Para os problemas 1, 4 e 8 o protótipo apresenta média maior. No entanto, as diferenças em valores absolutos são aceitáveis, frente a possibilidade de tratar composições heterogêneas. Portanto, conclui-se que a arquitetura possui desempenho viável.

O segundo experimento consistiu na execução do *benchmark* com o qual o *Pragmatic Proof* foi avaliado [32]. Nesse experimento, 9 execuções foram suficientes para garantir o IC de 95%. O *benchmark* possui três cenários, sempre variando o número de serviços ($n$) no repositório em potências de 2 (de 2 a 1024). No cenário 1, cada serviço possui um parâmetro de entrada e um de saída, enquanto no cenário 2, são três entradas e três saídas. Nesses dois cenários a composição que satisfaz o objetivo é um encadeamento de todos os $n$ serviços. O cenário 3 é similar ao cenário 1, mas a solução

não envolve os $n$ serviços, mas sim outros 32 serviços adicionais. Foi comparado o tempo do EYE [33] ($t_E$), *reasoner* utilizado no *Pragmatic Proof*, com o tempo de planejamento do protótipo ($t_P$).

Para os 3 cenários, há um $n_0$ tal que para todo $n_i \geq n_0$, $t_P \leq t_E$. Comparando os cenários 1 e 2, é possível analisar o efeito do número de parâmetros dos serviços no tempo de planejamento. Denota-se $t_{E1}$ e $t_{E3}$ os tempos do EYE no cenário 1 e 2, respectivamente. A Figura 4 mostra o comportamento de $\delta_E$ (a) e $\delta_P$ (b), onde $\delta_E = t_{E3} - t_{E1}$, sendo $\delta_P$, $t_{E1}$ e $t_{P3}$ definidos de forma análoga. Os gráficos indicam que o protótipo é pouco afetado pelo número de parâmetros. De fato, uma transformação logarítmica aplicada simultaneamente a $n$, $t_{E3}$ e $t_{E1}$ (para que suposições estatísticas quanto ao erro experimental sejam satisfeitas) permite obter um modelo quadrático para o *speedup* obtido ao reduzir o número de parâmetros por serviço de 3 para 1, expresso por $log_2(\frac{t_{E3}}{t_{E1}})$, a partir de $log_2(n)$. No entanto, o mesmo não é possível para $\delta_P$ ou $\frac{t_{P3}}{t_{P1}}$ (com ou sem transformações), uma vez que o erro experimental tem mais impacto que o *speedup* hipotético. Ou seja, nesse *benchmark*, o protótipo não é afetado pelo número de parâmetros dos serviços.

## 8 CONSIDERAÇÕES FINAIS

O presente trabalho apresentou uma arquitetura para composição automática de serviços heterogêneos, especificamente SOAP, REST e SOE. Embora no estágio atual a viabilidade da arquitetura já tenha sido demonstrada, a avaliação de expressividade e de aplicabilidade ainda não foi realizada e a terceira contribuição esperada ainda não foi alcançada. Essa última é importante por se tratar de uma contribuição além da academia, com o resultado esperado de, em parte, simplificar o desenvolvimento e manutenção de aplicações web distribuídas. Pretende-se, a partir do protótipo, construir uma ferramenta de código aberto que possua integração com ferramentas já utilizadas em produção e que permita automatizar partes de composições hoje construídas manualmente.

No que tange à expressividade, utilizou-se um algoritmo baseado apenas em I/O dos serviços. Embora essa classe de algoritmos apresente bom desempenho, problemas de composição mais complexos necessitam de algoritmos mais expressivos. Para solucionar esta limitação, pretende-se aprimorar o algoritmo de modo a considerar pré- e pós-condições dos serviços durante a composição.

## REFERÊNCIAS

[1] Jhonatan Alves, Jerusa Marchi, Renato Fileto, and Mario A R Dantas. 2016. Resilient composition of Web services through nondeterministic planning. In *2016 IEEE Symp. on Computers and Comm. (ISCC)*. IEEE, Messina, Italy, 895–900.

[2] Pierpaolo Baglietto, Massimo Maresca, Michele Stecca, Antonio Manzalini, Roberto Minerva, and Corrado Moiso. 2010. Analysis of design patterns for composite telco services. In *Intelligence in Next Generation Networks (ICIN), 14th Internat. Conf. on*. IEEE, Berlin, Germany.

[3] Ajay Bansal, M Brian Blake, Srividya Kona, Steffen Bleul, Thomas Weise, and Michael C Jaeger. 2008. WSC-08: Continuing the Web Services Challenge. In *2008 10th IEEE Conf. on E-Commerce Technol. and the Fifth IEEE Conf. on Enterprise Comput., E-Commerce and E-Services*. IEEE, Washington, DC, USA, 351–354.

[4] Peter Bartalos and Mária Bieliková. 2011. Automatic dynamic web service composition: A survey and problem formalization. *Computing and Informatics* 30, 4 (2011), 793–827.

[5] Mike Belshe, Peon Roberto, and Thomson Martin. 2015. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. RFC Editor.

[6] Patrick Th. Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.* 35, 2 (2003), 114–131.

[7] Roy T Fielding and Richard N Taylor. 2002. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.* 2, 2 (2002), 115–150.

[8] B. Fitzpatrick, B. Slatkin, M. Atkins, and J. Genestoux. 2014. *PubSubHubbub Core 0.4*. Working Draft. RFC Editor. https://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html

[9] Nikolaos Georgantas, Georgios Bouloukakis, Sandrine Beauche, and Valérie Issarny. 2013. Service-oriented distributed applications in the future internet: The case for interaction paradigm interoperability. *European Conf. on Service-Oriented and Cloud Computing* 8135 LNCS (2013), 134–148.

[10] Steve Graham, Hull Hull, and Bryan Murray. 2006. *Web Services Base Notification 1.3*. OASIS Standard. OASIS.

[11] J. Gregorio and B. de Hora. 2007. *The Atom Publishing Protocol*. RFC 5023.

[12] Feifei Hang and Liping Zhao. 2013. HyperMash: A Heterogeneous Service Composition Approach for Better Support of the End Users. In *Proceedings of the 2013 IEEE 20th International Conference on Web Services (ICWS '13)*. IEEE Computer Society, Washington, DC, USA, 435–442.

[13] Florian Haupt, Markus Fischer, Dimka Karastoyanova, Frank Leymann, and Karolina Vukojevic-Haupt. 2014. Service Composition for REST. In *Proc. IEEE 18th Internat. Enterprise Distributed Object Computing Conf.* IEEE, Ulm, Germany, 110–119.

[14] Jacek Kopecký, Karthik Gomadam, and Tomas Vitvar. 2008. hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Web Intelligence and Intelligent Agent Technol., 2008. WI-IAT '08. IEEE/WIC/ACM Internat. Conf. on*, Vol. 1. IEEE, Sydney, Australia, 619–625.

[15] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. 2007. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Comput.* 11, 6 (2007), 60–67.

[16] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. 2010. Investigating Web APIs on the World Wide Web. In *Web Services (ECOWS), 2010 IEEE 8th European Conf. on*. IEEE, Ayia Napa, Cyprus, 107–114.

[17] Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. 2001. Semantic Web services. *IEEE Intelligent Systems* 16, 2 (2001), 46–53.

[18] OASIS. 2007. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. OASIS.

[19] Mike P Papazoglou. 2003. Service-oriented computing: concepts, characteristics and directions. In *Proc. of the Fourth Internat. Conf. on Web Information Systems Engineering*. IEEE, Rome, Italy, 3–12.

[20] Cesare Pautasso. 2009. RESTful Web Service Composition with BPEL for REST. *Data Knowl. Eng.* 68, 9 (sep 2009), 851–866.

[21] Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky, and John Domingue. 2010. iServe: a linked services publishing platform. In *Proc. of the 1st Workshop on Ontology Repositories and Editors for the Semantic Web*, Vol. 596. CEUR Workshop Proc., Hersonissos, Greece, 71–82.

[22] Yu-Yen Peng, Shang-Pin Ma, and Jonathan Lee. 2009. REST2SOAP: A framework to integrate SOAP services and RESTful services. In *IEEE Internat. Conf. on Service-Oriented Computing and Applications, SOCA' 09*. IEEE, Taipei, Taiwan, 106–109.

[23] Pablo Rodriguez-Mier, Manuel Mucientes, Juan C Vidal, and Manuel Lama. 2012. An Optimal and Complete Algorithm for Automatic Web Service Composition. *Internat. Journal of Web Services Research* 9, 2 (2012), 1–20.

[24] Pablo Rodriguez-Mier, Carlos Pedrinaci, Manuel Lama, and Manuel Mucientes. 2016. An Integrated Semantic Web Service Discovery and Composition Framework. *IEEE Trans. on Services Comput.* 9, 4 (2016), 537–550.

[25] Dumitru Roman, Jacek Kopecký, Tomas Vitvar, John Domingue, and Dieter Fensel. 2015. WSMO-Lite and hRESTS. *Web Semant.* 31 (March 2015), 39–58.

[26] Ivan. Salvadori and Frank Siqueira. 2015. A Maturity Model for Semantic RESTful Web APIs. In *Web Services, IEEE Internat. Conf. on*. IEEE, New York, NY, 703–710.

[27] Edwin Seidewitz. 2003. What models mean. (2003), 26–32 pages.

[28] Siniša Srbljić, Dejan Škvorc, and Daniel Skrobo. 2010. Programming Language Design for Event-driven Service Composition. *Automatika* 51, 4 (2010), 374–386.

[29] Thomas Strang and Claudia Linnhof-Popien. 2003. Service Interoperability on Context Level in Ubiquitous Computing Environments. In *Proceedings of the 2003 International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet*.

[30] Rutt Tom, Hainline Micah, Jeyaraman Ram, and Durand Jacques. 2014. *Web Services Basic Profile 1.2*. Technical Report. OASIS.

[31] Bipin Upadhyaya, Ying Zou, Hua Xiao, Joanna Ng, and Alex Lau. 2011. Migration of SOAP-based services to RESTful services. In *13th IEEE International Symposium on Web Systems Evolution (WSE)*. IEEE, Williamsburg, VI, USA, 105–114.

[32] Ruben Verborgh, Dörthe Arndt, Sofie Van Hoecke, Jos De Roo, Giovanni Mels, Thomas Steiner, and Joaquim Gabarro. 2016. The pragmatic proof: Hypermedia API composition and execution. *Theory and Practice of Logic Programming* 17, 1 (2016), 1–48.

[33] Ruben Verborgh and Jos De Roo. 2015. Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. *IEEE Software* 32, 3 (2015), 23–27.

[34] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Jos De Roo, Rik de Van Walle, and Joaquim Gabarró Vallés. 2013. Capturing the functionality of Web services with functional descriptions. *Multimedia Tools and Applications* 64, 2 (2013), 365–387.