UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE – UERN
***CAMPUS* AVANÇADO DE NATAL**
FACULDADE DE CIÊNCIAS EXATAS E NATURAIS – FANAT
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE DE MEDEIROS BARROS

WEB-SERVICES COMPOSTOS: UM ESTUDO SOBRE AS ABORDAGENS DE VERIFICAÇÃO

NATAL/RN

2015

PEDRO HENRIQUE DE MEDEIROS BARROS

WEB-SERVICES COMPOSTOS: UM ESTUDO SOBRE AS ABORDAGENS DE VERIFICAÇÃO

Monografia apresentada à Universidade do Estado do Rio Grande do Norte – UERN como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Dr. Felipe Denis Mendonça de Oliveira

NATAL/RN

2015

PEDRO HENRIQUE DE MEDEIROS BARROS

WEB-SERVICES COMPOSTOS: UM ESTUDO SOBRE AS ABORDAGENS DE VERIFICAÇÃO

Monografia apresentada à Universidade do Estado do Rio Grande do Norte – UERN como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em _____/_____/_____

Banca Examinadora

_____

Prof. Dr. Felipe Denis Mendonça de Oliveira

_____

Profª. Me. Camila de Araújo Sena

_____

Prof. Dr. Carlos Alberto de Albuquerque Silva

Catalogação da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.

Barros, Pedro Henrique de Medeiros

Web-services compostos: um estudo sobre as abordagens de verificação. / Pedro Henrique de Medeiros Barros. – Natal, RN, 2015.

76 f.

Orientador: Prof. Dr. Felipe Denis Mendonça de Oliveira

Monografia (Graduação em Ciência da Computação). Universidade do Estado do Rio Grande do Norte.

Bibliotecária: Jocelania Marinho Maia de Oliveira – CRB 15 319

A minha família; meu companheiro; meus amigos e a todos que colaboraram com essa conquista.

AGRADECIMENTOS

Foi um caminho longo e difícil para chegar até aqui. Muitos anos de estudo, dedicação, esforço e empenho foram necessários. Não conseguiria concluir este curso sozinho e, devido a isso, venho aqui prestar meus agradecimentos.

Agradeço primeiramente a Deus, pela força, saúde, vida e cuidado que tem comigo. Obrigado Senhor! Se não fosse a Sua mão eu não teria conseguido.

Agradeço ao meu orientador, Professor Felipe Denis, por acreditar no meu trabalho e sempre ter se mantido disponível nos momentos que precisei. Muito obrigado!

Aos professores da UERN, funcionários, pela 'família UERN' que sempre me incentivou à conclusão do curso.

Agradeço a minha família, em especial aos meus pais, minha irmã, a vovó Bezinha e vovô Zé Lucio. Por todo o amor, compreensão, ajuda, puxões de orelha, cuidado e dedicação. Sem vocês eu não seria quem sou, sem vocês nada disso seria possível. Meu muito mais que obrigado, amo vocês!

Ao meu companheiro, Samuray Freire, por acreditar em mim, por me ajudar sempre quando precisei, pelos conselhos e ombro amigo. Você é peça fundamental na minha vida. Muito obrigado de todo o coração.

Agradeço aos amigos que fiz durante o curso: Lidiane, Katiane, Maria Helena, Jacyara, Gustavo, Franklin, Suzyane, entre outros. Obrigado pelas risadas e choros compartilhados, vocês são incríveis.

Obrigado aos meus amigos de vida, pessoas mais que especiais que cruzaram o meu caminho e que me enchem de amor, energia positiva e luz.

Enfim, agradeço a todos que, diretamente ou indiretamente contribuíram para a conclusão do meu curso de graduação.

[...] Porque Deus disse: “faça por onde que Eu te ajudarei”; então vambora fazendo!

Inês Brasil

RESUMO

Diversos tipos de serviços estão disponíveis através da internet apresentando funções que agilizam e movimentam o mercado mundial. Em função desse cenário as organizações de TI estão tendo a necessidade de lidar com os serviços, fazendo com que a necessidade, tempo/agilidade, a pressão para a redução de custos e a adaptabilidade sejam características predominantes. SOC utiliza serviços como elementos fundamentais para desenvolver aplicações e soluções. Associado a esse paradigma existe a SOA, a arquitetura orientada a serviços que tem como principal exemplificação os *Webservices*. Serviços complexos são criados através de composição entre serviços pré-existentes e, para isso, é necessário que existam parâmetros e critérios de comunicação em comum (multiplataforma) conhecidos como *gates* e que, no repositório UDDI, conste a descrição em alguma linguagem sintática desses *Webservices* para assim, existir uma composição correta, sem erros, sem pontos de *deadlock*. Para comprovar que uma composição está correta são utilizadas metodologias de especificação formal para realizar o processo de verificação operacionais, confiáveis, robustas e livre de erros. Este trabalho tem como resultante o relato com as técnicas transacionais utilizadas e suas respectivas variações propostas com o objetivo de garantir o bom funcionamento da composição de *Webservices* - tanto por orquestração e coreografia - ao realizar a sua verificação antes da sua execução.

Palavras-Chave: *Webservices*. Composição de Serviços. Verificação de Serviços Compostos. Abordagens Transacionais.

ABSTRACT

Several types of services are available through the internet featuring functions that speed and power the global market. Due to this scenario, IT organizations are taking the need to deal with the services, causing the need, time/speed, pressure to reduce costs and adaptability are predominant features. SOC utilizes services as fundamental elements for developing applications and solutions. Associated with this paradigm there is the SOA, service-oriented architecture whose main exemplification the Webservices. Complex services are created through composition between pre-existing services and for this it is necessary that there are parameters and common reporting criteria (cross-platform) known as gates and that in the UDDI repository, bearing the description in some syntactic language of these Webservices for thus there is a correct composition, without error, deadlock points. To prove that a composition is correct formal specification methods are used to carry out the process of operational check, reliable, robust and error free. This work is resulting reporting with the use transactional techniques and their variations proposals in order to ensure the proper functioning of Webservices composition - both orchestration and choreography - to hold its checked prior to the execution.

Keywords: Webservices. Service Composition. Compounds Service Verification. Transactional Approaches.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

TI - Tecnologia da Informação

SOC - *Service-Oriented Computing*

SOA - *Service-Oriented Architecture*

SOAP - *Simple Object Access Protocol*

HTTP - *Hypertext Transfer Protocol*

SMTP - *Simple Mail Transfer Protocol*

FTP - *File Transfer Protocol*

UDDI - *Universal Description, Discovery and Integration*

BPEL - *Business Process Execution Language*

WS-CDL - *Web Service Choreography Definition Language*

XLANG - *XML LANGuage*

BPML - *Business Process Modeling Language*

WSCI - *Web Service Choreography Interface*

XML - *eXtensible Markup Language*

WSDL - *Web Services Description Language*

MIME - *Multipurpose Internet Mail Extensions*

URL - *Uniform Resource Locator*

TCP - *Transmission Control Protocol*

MSMQ - *Message Queuing*

POP3 - *Post Office Protocol*

IMAP - *Internet Message Access Protocol*

WSFL - *Web Services Flow Language*

OWL-S - *Web Ontology Language for Service*

CCS - *Calculus of Communicating Systems*

ACP - *Algebra of Communicating Processes*

PROMELA - *Process or Protocol Meta Language*

LTL - *Linear Temporal Logic*

UML - *Unified Modeling Language*

UML-AD - *Activity Diagram UML*

LOTOS - *Language of Temporal Ordering Specification*

LTS - *Labelled Transition Systems*

STS - *Symbolic Transition System*

SFA - *Symbolic Finite Automaton*

AFD - Autômato Finito Determinístico

SUMÁRIO

1. INTRODUÇÃO

Os padrões e normas da sociedade mundial vivem em constante mudança de acordo com a evolução do tempo, desenvolvendo-se em diversos aspectos e parâmetros: política, religião, educação, economia. Em todos os âmbitos pode-se observar diversas características mutáveis e como elas são afetadas por vários tipos de fatores (CARVALHO, 1997).

De acordo com a história algumas situações foram determinantes e marcantes para que alterações notáveis fossem observadas na nossa instituição social, dentre elas: a descoberta da escrita, a integração entre as sociedades, a mercantilização, a revolução industrial, as guerras mundiais, dentre outros. Em toda esta contextualização de acontecimentos o que mais se aproxima da atualidade e que veio gerar uma mudança significativa no comportamento social se trata da facilidade de troca de informações em alta velocidade, viabilizada por meios de comunicação criados para esta finalidade, a partir da internet. A velocidade e a alta taxa de dados trocados por todos os usuários da internet no mundo proporcionaram para a sociedade contemporânea e hipermoderna[1] o título de sociedade da informação (TAKAHASHI, 2000).

A intitulada sociedade da informação é definida por Beninger (1976) como o modelo social no qual a criação, distribuição, uso, integração e manipulação de informações se caracteriza enquanto atividade econômica, política e cultural de extrema importância. A qualidade, produtividade, a competição, a demanda de consumidores elevada, a evolução rápida e constante, a tecnologia, a inovação e globalização são palavras que definem o mundo dos negócios nos dias atuais SENAC (2011). O mundo está envolto por serviços e gerenciamento de negociações.

Toda evolução traz desafios para a sua manutenção. Esses desafios podem ser vistos nas negociações nas quais os processos de comercialização estão cada vez mais rápidos e, além da velocidade, a adaptação dos produtos e serviços, de

[1] Hipermodernidade é o termo criado pelo filósofo francês Gilles Lipovetsky. Lipovetsky, compreendendo que os termos pós-modernidade ou mesmo contemporaneidade não dava mais conta de todas as transformações sofridas pelas sociedades do século XXI, buscou delimitar o momento atual da sociedade humana com essa nova terminologia. A utilização do termo “hiper” teve como objetivo referenciar uma exacerbação dos valores criados na Modernidade, atualmente elevados de forma exponencial. Para maiores aprofundamentos, indico a obra intitulada “Os tempos hipermodernos” de Gilles Lipovetsky e Sébastien Charles.

acordo com a necessidade dos consumidores, faz com que seja necessário que os mesmos estejam disponíveis de acordo com o desejo do consumidor e independentemente da plataforma que o mesmo utiliza para acessá-los. De acordo com Ferreira (2011) serviços podem ser definidos como uma atividade econômica que não geram produtos tangíveis.

Diversos tipos de serviços estão disponíveis através da internet e, como dito anteriormente, os mesmos não apresentam resultados e produtos de forma concreta, mas apresentam diversos tipos de funções que agilizam e movimentam o mercado mundial. Em função desse novo cenário as organizações de tecnologia da informação (TI) estão tendo a necessidade de lidar, de forma especial, com os serviços, fazendo com que a necessidade, tempo/agilidade, a pressão para a redução de custos e a adaptabilidade sejam características predominantes para os serviços oferecidos.

Como alternativa para se adequar a essa nova caracterização de mercado surge um novo paradigma computacional intitulado de *Service-Oriented Cumputing* (SOC). SOC utiliza serviços como elementos fundamentais para desenvolver aplicações e soluções. Associado a esse paradigma existe a *Service-Oriented Archteture* (SOA), que se trata da implementação, por processos, de forma bem definida, com funções reutilizáveis e flexíveis, que são os serviços. Segundo Papazoglou e Georgakopoulos (2003) “SOA é um modelo implementativo que é simplesmente caracterizado como uma forma de reorganizar aplicações de infraestrutura em um conjunto de serviços para que eles interajam entre si”. A funcionalidade da aplicação é descrita de uma forma que permite o uso para integrar e adquirir funcionalidades em outras aplicações, criando assim, possíveis composições de serviços (DAMASCENO, 2009).

Webservices, segundo Kova *et al.* (2009) estão em expansão e são tipos de serviços utilizados para as mais diversas finalidades, como por exemplo a integração entre aplicações, facilitando assim colaborações empresariais e criando serviços complexos, *business-to-business*, entre outros. A arquitetura de desenvolvimento de um *Webservice* traz parâmetros de comunicação como SOAP, protocolos de transporte como o HTTP, SMTP, FTP, etc.; repositório de *Webservices* em uma camada UDDI e a descrição da sua funcionalidade em linguagens como BPEL, WS-CDL, XLANG, BPML, WSCI.

Serviços complexos são criados através de composição entre serviços pré-existentes e, para isso, é necessário que existam parâmetros e critérios de comunicação em comum (multiplataforma) conhecidos como *gates* e que, no repositório UDDI, conste a descrição em alguma linguagem sintática desses *Webservices* para assim, existir uma composição correta, sem erros, sem pontos de *deadlock* (W3C, 2004).

Para comprovar que uma composição está correta, alguns pesquisadores – dentre os quais podemos citar: Lohmann (2007), Ouyang (2007) [2], Tan (2009) e Beek (2007); afirmam que metodologias de especificação formal podem ser utilizadas para ajudar no processo de verificação das mesmas, fazendo com que elas sejam operacionais, confiáveis, robustas e livre de erros, retornando assim, ao consumidor todas, as funcionalidades desejadas no exato momento em que o mesmo solicitar.

Visando então acompanhar as técnicas de validação e verificação das composições de *Webservices*, foram analisados neste trabalho diversos artigos, entre monografias, dissertações e trabalhos acadêmicos, tratando sobre métodos de tradução, transformação, especificação automática, entre outras metodologias, para resolver a problemática de composição de serviços complexos no tocante a sua corretude. Como resultante do mesmo, tem-se o relato com as técnicas transacionais utilizadas e suas respectivas variações propostas com o objetivo garantir o bom funcionamento da composição de *Webservices* - tanto por orquestração e coreografia - ao realizar a sua verificação antes da sua execução.

[2] Lohmann e Ouyang realizam tal afirmação propondo o desenvolvimento de uma ferramenta que realiza o mapeamento do código BPEL de uma composição para Redes de Petri, objetivando realizar a verificação de composições de *Webservices*.

2. OBJETIVOS

2.1. OBJETIVO GERAL

Estudar os *Webservices*, em especial os *Webservices* Compostos, fazendo um levantamento sobre as técnicas e abordagens utilizadas para a verificação de composições automáticas, com relação a seu funcionamento perante as diversas situações.

2.2. OBJETIVOS ESPECÍFICOS

- Entender o funcionamento de um *Webservice* composto;
- Levantar dados bibliográficos com relação as abordagens de verificação das composições;
- Identificar as técnicas de especificação mais utilizadas para *Webservices* compostos;
- Relacionar as técnicas de especificação mais utilizadas;

3. WEBSERVICES EM SUA CONSTITUIÇÃO TEÓRICA

Em se tratando de serviços computacionais, os quais estão disponíveis, em sua maioria, através da internet, os mesmos não apresentam resultados e produtos de forma concreta, mas trazem diversos tipos de funções que agilizam e movimentam o mercado mundial. Assim, as organizações de Tecnologia da Informação (TI) estão tendo a necessidade de lidar de forma diferenciada com estes serviços, fazendo com que o tempo/agilidade, a pressão para a redução de custos e a adaptabilidade sejam características predominantes para os mesmos.

Com esse objetivo surge um novo paradigma computacional denominado *Service-Oriented Compunting* (SOC) que, de acordo com Papazoglou

> *Service-Oriented Computing* (SOC) é um novo paradigma computacional que utiliza serviços como construtores básicos para apoiar o desenvolvimento de forma rápida, com baixo custo e fácil composição de aplicações distribuídas mesmo em ambientes heterogêneos. (2008, p.1, tradução nossa)

Como via implementativa do paradigma supracitado, tem-se como técnica principal a *Service-Oriented Architecture* (SOA) que utiliza os conceitos, princípios e métodos computacionais existentes em SOC, os quais serão detalhados a seguir.

3.1. SERVICE-ORIENTED ARCHITECTURE (SOA)

Com a configuração do cenário atual relacionada a produção de sistemas computacionais que, em sua grande maioria, são voltados para a internet, viu-se a necessidade da utilização de uma arquitetura baseada nos princípios da computação distribuída, visando a comunicação entre os sistemas heterogêneos, em plataformas distintas e desenvolvidas por empresas diferentes.

A necessidade dos mais diversos tipos de sistemas e a grande vastidão de aplicações complexas existentes atualmente, são provas de que a implementação de sistemas deste tipo é inviável se feita por apenas um único grupo desenvolvedor. Devido a isto, vê-se a necessidade de utilização de uma arquitetura de software que possa trazer aos sistemas métodos que proporcionem o particionamento de sistemas

complexos em sistemas mais simples, em que seja possível o acoplamento dos mesmos para formar o sistema final.

Doravante, surge a SOA como uma maneira de implementar os processos de forma bem definida com funções reutilizáveis e flexíveis. Papazoglou (*idem*) relata que SOA é um modelo implementativo simplesmente caracterizado como uma forma de reorganizar aplicações de infraestrutura em um conjunto de serviços, para que eles interajam entre si. A funcionalidade de sua aplicação é descrita de uma forma que permite o uso para integrar e adquirir funcionalidades em outras aplicações, criando assim possíveis composições de serviços (DAMASCENO, 2009). Ainda segundo Damasceno

> Em SOA a unidade mais básica é o serviço, que são entidades que provêm alguma funcionalidade. A partir desses serviços e de suas interfaces novos serviços podem ser construídos. [...] *Web Services* são um exemplo de como esta arquitetura está se difundindo rapidamente. E isso se deve ao fato dessa tecnologia usar de soluções simples para resolver problemas relacionados a integração entre aplicações. [...] SOA é mais do que apenas um conjunto de tecnologias e não está diretamente relacionada com qualquer tecnologia, embora seja mais frequentemente implementada com *Web Services.* Os *Web Services* são a tecnologia mais apropriada para a realização de SOA. (*idem*, p.3)

Segundo Soares (2007), existem algumas características chave de benefícios trazidos pela SOA, que estão descritos a seguir:

- Interoperabilidade: esta propriedade garante que é possível haver a comunicação entre serviços independentes, contando com o compartilhamento dos mesmos padrões de comunicação, como por exemplos os protocolos HTTP, SMTP, FTP, etc;

- Visibilidade: é uma característica de extrema importância pois, através dela, os consumidores podem enxergar os serviços disponíveis para utilização, bem como os próprios serviços podem se enxergar para assim estabelecerem uma comunicação, onde o resultado final é um serviço composto e complexo formado pela associação dos participantes;

- Reutilização: como os serviços podem ser associados para formarem outros serviços compostos, existe a possibilidade de reaproveitamento das unidades já implementadas para compor novos serviços complexos, diminuindo assim os custos na implementação e reduzindo o tempo da mesma;

- Adaptabilidade: com as rápidas mudanças nos ambientes de negócios, faz-se necessário o dinamismo das aplicações e a adaptabilidade das mesmas para com outras plataformas. SOA traz a possibilidade da adaptação a novas regras, como por exemplo, a utilização de serviços com funções complementares para montarem composições com objetivos diferentes.

Ante estas colocações, pode-se dizer, então, que a SOA se trata de um conjunto de tecnologias e abordagens que não são diretamente relacionadas com técnicas determinadas, mas se trata da implementação baseada em conceitos específicos que, geralmente, são notados por intermédio da construção dos webservices.

3.2. WEBSERVICES

Webservices, segundo Kova (2009), é uma tecnologia que está em expansão, nos quais os seus serviços são implementados, destarte, utilizados para as mais diversas finalidades. A exemplo, tem-se a integração entre aplicações distintas devido a facilidade das colaborações entre as funcionalidades, haja vista sua estrutura permitir a heterogeneidade de plataforma, a visibilidade, a acessibilidade, bem como a comunicação entre esses *Webservices*, podendo assim criar serviços mais complexos.

Alonso (2004, p.124, tradução nossa) corrobora dizendo que os "*Webservices* são vistos como uma aplicação acessível para outras aplicações por intermédio da web". Ou seja, os webservices podem ser entendidos como componentes que possuem funcionalidades disponíveis e acessíveis pela internet, apresentando a disponibilidade das suas operações, funcionalidades e descrições dos seus serviços, tanto para os seus consumidores quanto para os outros *Webservices.*

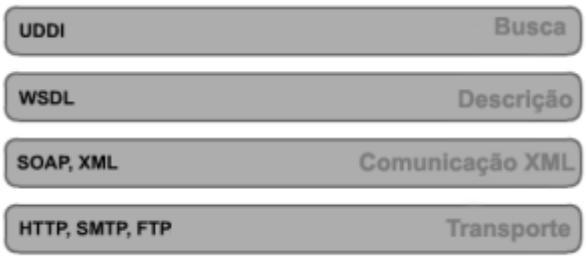
Teixeira (2012, p.34) complementa afirmando que os

> *Webservice* é uma representação padrão para algum recurso computacional ou de informações que pode ser usado por outros programas, os quais podem ser recursos de informações, como um catálogo de peças, recursos de computador, tais como um processador especializado, ou recursos de armazenamento. A essência de um serviço é que o fornecimento do serviço é independente da aplicação que o usa. Os provedores de serviços podem desenvolver serviços especializados e oferecê-los para uma variedade de usuários de diferentes organizações.

Para que os *Webservices* se enquadrem como exemplificação da implementação da SOA, é preciso que eles utilizem conceitos definidos por esta tipologia arquitetural, ou seja, que tenham a descrição dos serviços, que haja comunicação por intermédio de suas interfaces, que eles se comuniquem entre si, que exista um baixo acoplamento entre eles e que eles sejam o mais independente possível.

Outrossim, associadas aos *Webservices* existe uma série de protocolos responsáveis por sua descrição, por sua localização, seu desenvolvimento e pelo tipo de comunicação que eles estabelecem entre si. Segundo Damasceno (*ibidem*), pode separar estes protocolos em 4 tipos de camadas diferentes, que são: Camada de busca, camada de descrição, camada de comunicação e camada de transporte, conforme exibido na Figura 1 e, a posteriori, elucubrados:

Figura 1 – Camadas de protocolos dos *Webservices*



Fonte: Damasceno (*idem*, p.6)

Dada a importância das camadas supracitadas para a composição e atuação dos *Webservices*, compreende-se a importância de realizar uma explanação, em linhas gerais, sobre as mesmas, trazendo suas descrições e como elas se agrupam à formação de um *Webservice.*

3.2.1. Arquitetura

3.2.1.1. Camada de Busca

Quando ocorre o desenvolvimento de um *Webservice* é necessário que ele seja acessado por um cliente ou por outro serviço em algum lugar da internet. Para que isto ocorra, existe um padrão de diretórios disponível para a busca e a publicação dos serviços, funcionando assim como um mediador entre os requisitantes e os desenvolvedores.

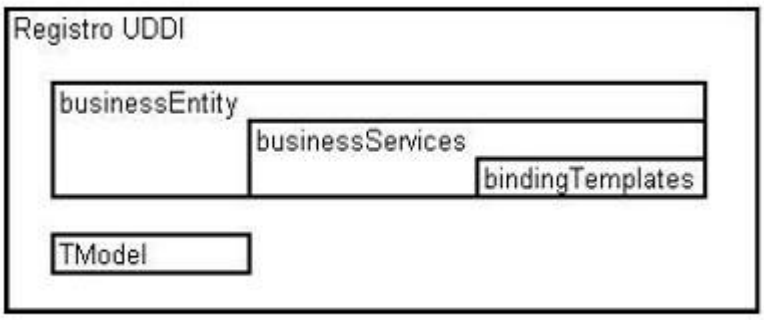
Nesse sentido, é na camada de busca que ficam armazenadas as publicações dos *Webservices*. Uma das tecnologias mais conhecidas para desempenhar tal função é a UDDI (*Universal Description, Discovery, and Integration).* Segundo Reckziegel (2006), a UDDI é uma especificação que descobre, descreve e integra os *Webservices*. A ideia do seu funcionamento é a de uma base de dados que possui a descrição dos *Webservices* nele contidos. O consumidor de um *Webservice* tem a UDDI como uma unidade de consulta dos serviços desejados.

Segundo Dalton,

> Um diretório UDDI é um arquivo XML que descreve o negócio e os serviços. O registro tem três partes: Páginas brancas: descrevem a companhia: nome, endereço e contatos; Páginas amarelas: incluem as categorias, baseada em taxonomias padrões. Páginas verdes: descrevem a interface para o serviço, em nível de detalhe suficiente para se escrever uma aplicação que use o *Web Service*. (2006, p.13)

Ainda segundo Reckziegel (2006), pode-se definir uma hierarquia dentro da estrutura de um registro UDDI, que é definido conforme imagem abaixo:

Figura 2 – Estrutura de um registro UDDI



Fonte: Reckziegel (*idem*)

Sobre essa hierarquia dentro da estrutura de um UDDI, Dalton (*idem*) e Reckziegel (*idem*) trarão a seguinte definição sobre a mesma:

- *BusinessEntity* é um elemento que representa o provedor de um *Webservice*. É nele que são apresentadas informações como dados de contato, categoria, serviços oferecidos e identificadores de negócio de uma determinada organização/empresa;
- *BusinessService* é um elemento filho do elemento *businessEntity*, o qual descreve a função de negócio de um serviço;
- *BindingTemplate* é um elemento que referencia os detalhes técnicos do serviço, como a sua interface e/ou API;
- *TModels* ou *Typemodel* é um elemento que trata da definição dos serviços e, em alguns casos, contém o arquivo WSDL que descreve as interfaces de comunicação dos *Webservices*.

3.2.1.2. Camada de Descrição

Descrever serviços de maneira flexível, extensível e compatível é um dos objetivos mais centrais da SOA. E como os *Webservices* são a maior exemplificação desta arquitetura, vê-se que a integração e o reuso dos mesmos são características chave para um bom desenvolvimento.

Webservices são multiplataforma e, para que eles se comuniquem, faz-se necessária a padronização de sua descrição. Afim de atender essa necessidade, surgiu a linguagem denominada *Web Services Description Language* (WSDL). WSDL é baseado no formato *eXtensible Markup Language* (XML) para descrever *Webservices* em um conjunto de operações que podem ser orientadas a documentos ou a procedimentos (CHRISTENSEN *et al.*, 2001).

Christensen *et al.* ainda relata que

> WSDL é baseado no formato XML para descrever webservices com um conjunto de operações com objetivos específicos através de mensagens orientadas a documentos ou a procedimentos. As operações e mensagens são descritas de forma abstrata, e então são conduzidas através de

> protocolos de rede em mensagens com o formato que define os *endpoints* dos webservices. *Endpoints* concretos relacionados são combinados com *endpoints* abstratos (serviços). WSDL é extensível para permitir a descrição dos *gates* e das mensagens independentemente do formato da mensagem ou do protocolo de rede usado para a comunicação, uma vez que, as únicas ligações descritas neste documento dizem qual a utilização do WSDL em conjunto com SOAP 1.1, HTTP GET/POST e MIME. (*ibidem*)

Uma descrição em WSDL pode ser separada em dois tipos: concreta e abstrata, que também são denominadas por alguns autores como funcionais e não-funcionais (DAMASCENO, *ibidem*). A interface concreta de um WSDL fala sobre a especificação de detalhes no que diz respeito a sua localização. Já a interface abstrata de um WSDL trata de uma definição genérica sobre o *Webservice* especificado, indicando informativos como tipo de dados, metadados, tipo de mensagens de troca de informações, e outros tipos de dados reusáveis.

Segundo Damasceno (*idem*) e Soares (*ibidem*) o WSDL 1.1, versão atual que é utilizada nos *Webservices*, é dividida nos seguintes campos descritores:

- <*definitions*>: este campo engloba todos os campos do WebService e é através dele que existem a definição de nome do mesmo e a declaração dos *namespaces* dos campos a seguir.

- *Interface* concreta

  - <*bindding*>: este campo detalha os protocolos e os formatos dos dados que serão utilizados na comunicação entre os consumidores e os *Webservices*. Também há a definição entre a identificação (atributo *name*), a atribuição da referência entre o *portType* e o *name* e a definição de cada operação feita pelos protocolos e os formatos de dados;

  - <*service*>: este campo trata da definição de atributos que referenciam o endereço de invocação de determinado WebService. Entre os atributos existentes podemos destacar: *name* que se refere a identificação; o elemento *port* que se refere a URL do *Webservice*; e o *bindding* que referencia o atributo *name* existente no campo *bindding*. É neste campo em que diferentes endereços podem ser referenciados para o mesmo serviço.

- Interface abstrata

  - <*message*>: este campo define quais os dados que são trocados em determinada operação, estabelecendo assim, para cada uma delas, os argumentos de entrada e valores de retorno, mais conhecidos como *gates*;

  - <*operation*>: definição abstrata de uma operação para a mensagem, como o nome da operação/método, assim como as mensagens de entrada/saída referentes aos parâmetros da operação do serviço;

  - <*portType*>: este campo trata das operações que podem ser atendidas e das mensagens envolvidas para atender a estas operações. Cada descrição define os tipos de mensagens e quais são elas a serem usadas para cada operação, como por exemplo, mensagens de entrada e mensagens de saída. Seus elementos definem a identificação (*name*), as operações (*operations*) e os tipos de mensagens, estas últimas que podem ser de dois tipos: mensagens de entrada (*input*) e mensagens de saída (*output*).

3.2.1.3. Camada de Comunicação

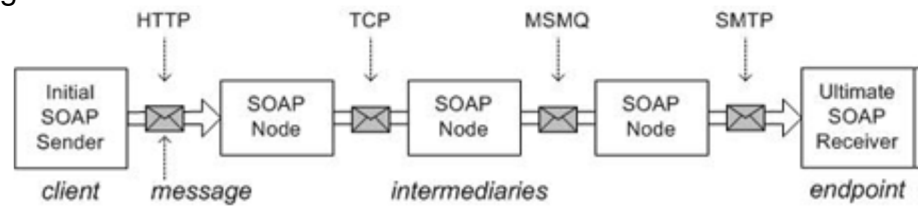
Quando se fala sobre *Webservices* e sua camada de comunicação, se refere diretamente ao protocolo *Simple Object Access Protocol* (SOAP). Protocolo este baseado no padrão de dados XML que permite a comunicação entre clientes e servidores.

O SOAP pode ser associado a quaisquer protocolos da camada de transporte, tendo com principalmente o *Hyper Text Tranfer Protocol* (HTTP). Segundo Sant'Anna (2004) o SOAP é um protocolo elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante à invocação de páginas Web.

Como as mensagens SOAP utilizam, geralmente, o protocolo HTTP com pacotes idênticos, virtualmente, para a sua comunicação, as suas mensagens atravessam automaticamente os *firewalls* e roteadores, devido serem confundidas

com mensagens HTTP da internet. Na Figura 3, pode ser observado um exemplo de fluxo de dados em uma comunicação que utiliza a tecnologia SOAP, na qual tem-se um cliente comunicando-se com um *endpoint,* valendo-se de diversos tipos de protocolos de transporte.

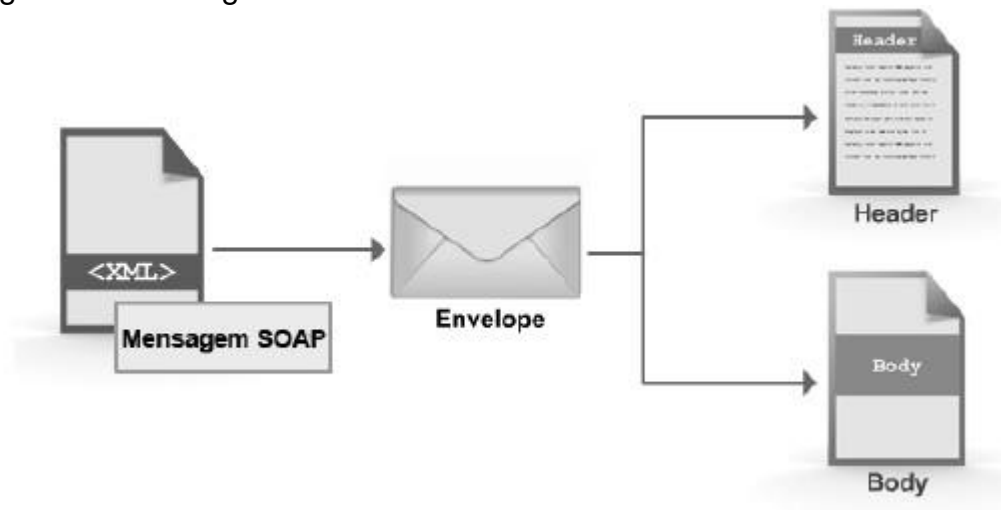
Figura 3 – Fluxo de dados SOAP



Fonte: Gudgin *et al.* (2007)

Segundo Damasceno (*ibidem*, p.9), uma mensagem SOAP tem dois elementos filhos: o *Header*, que possui informações e controle e o *Body*, que detém o conteúdo da mensagem, descrita em XML. Segue abaixo a representação gráfica da mensagem SOAP.

Figura 4 – Mensagem SOAP



Fonte: Damasceno (*idem*, p.9)

3.2.1.4. Camada de Transporte

Na estrutura arquitetural de um *Webservice,* a camada de mais baixo nível chama-se camada de transporte, a qual coincide com a camada de aplicação da internet (TANENBAUM, 2011). Nesta camada, pode ser observado diversos protocolos, como por exemplo HTTP, SMTP (*Simple Mail Transfer Protocol*), FTP (*File*

Transfer Protocol), POP3 (*Post Office Protocol*), IMAP (*Internet Message Access Protocol*), dentre outros.

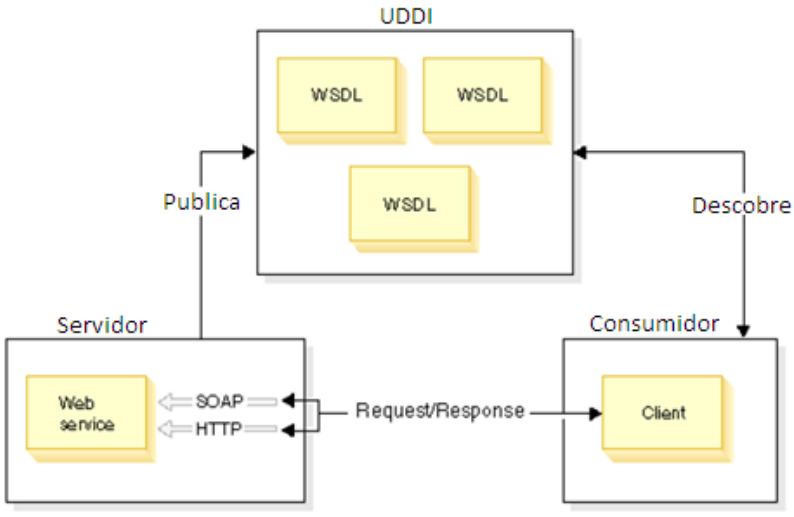
Como a grande maioria dos serviços são disponíveis por intermédio de sistemas de internet, serviços de e-mail, ou ainda pela disponibilização de arquivos, o protocolo mais comum e mais utilizado é o HTTP, que é um protocolo de comunicação baseado em *request-response*, que é feito através de navegadores web pelos clientes para os servidores (TANENBAUM, *ibidem*).

No entanto, outros protocolos podem ser utilizados, porém, em menor volume, são eles: o SMTP, que atua sobre clientes e servidores de e-mails, configurando redes ponto a ponto estabelecendo a comunicação entre ambos; e o FTP, que é simplesmente uma forma rápida e versátil de permitir a um cliente ter acesso a um determinado disco e assim transferir as suas informações deste servidor para a quem o solicita.

3.3. FUNCIONAMENTO WEBSERVICE

O esquema exposto através da Figura 5 descreve o funcionamento da interação cliente-webservice:

Figura 5 – Funcionamento das camadas



Fonte: Leandro (2011, p.5)

Um servidor de serviços web publica em determinada unidade UDDI os *Webservices* que ele detém. Esta publicação é feita na linguagem de descrição WSDL

que, especifica e facilita, para o cliente, localizar qual o *Webservice* que atende a sua necessidade. Ao descobrir o *Webservice* desejado, o cliente, por intermédio da comunicação viabilizada através do SOAP e HTTP, consegue obter o serviço desejado.

3.4. COMPOSIÇÃO DE SERVIÇOS (WEBSERVICES)

Uma das principais características de SOA é a possibilidade de reuso de componentes de softwares. No que concerne ao referido assunto, lida-se diretamente com a computação distribuída, no qual aplicações são desenvolvidas através de *Webservices* que, por intermédio de recursos de rede podem se comunicar e apresentar uma nova funcionalidade gerando, assim, um aproveitamento de código de serviços já implementados e ressignificando à sua funcionalidade. Neste sentido, essa comunicação de *Webservices* que é, necessariamente, formada a partir da aglutinação de dois ou mais serviços, recebe o nome de composição de serviços.

A grande maioria das composições de serviços possuem característica complexa e composição automática, haja vista surgirem a partir das necessidades de consumo apresentadas por clientes que as desejam. Segundo Rao e Su (2004), a definição de serviços compostos inclui um conjunto de *Webservices* atômicos juntos, onde há um controle e fluxo de dados entre os serviços componentes, bem como um *workflow* para especificar o fluxo por cada um dos seus itens.

Segundo Alergus e Jipa

> A interconexão de webservices para encontrar outros processos de negócios (*business process*) é chamada de Composição de Webservices. Esta composição pode ser vista como a integração de unidades autônomas ou outras composições de webservices. A composição está exatamente no fato de Webservices existentes serem combinados juntos sob as regras básicas definidas entre eles para suprir a demanda que um serviço simples não pode realizar. Estas regras são especificadas pelos próprios componentes da composição no momento em que os mesmos são invocados e as condições que existem nestas regras determinarão quais os serviços poderão ou não ser invocados. (2009, p.2, tradução nossa)

As composições de serviços podem ser feitas tanto de forma manual quanto automática. E sobre a diferenciação dessas duas formas, Silva (*ibidem*) traz as seguintes conceituações:

- Manual: a composição é feita através do usuário que escolhe e solicita os *Webservices* para atenderem a sua necessidade, organizando assim, a sua composição.
- Automática. Neste tipo de composição, o usuário deve especificar apenas o que precisa e então, o sistema automaticamente inicia o processo de busca e seleção de *Webservices* e os integram de forma que o objetivo seja atingido, criando assim, um serviço composto.

Para exemplificar os tipos de composição, tem-se o seguinte cenário: Um usuário precisa fazer a conversão de uma determinada quantia, em Real, para o seu respectivo valor em Euro. Quando esse procedimento é realizado por intermédio de uma composição automática, o usuário apenas informa a conversão que precisa, dizendo o seu valor em Real e os serviços individuais, necessários para este processo (um intermediário que converta de Real para Dólar e outro que converta de Dólar para Euro), se aglutinarão para gerar o resultado final, que é ofertar a conversão do valor de Real para Euro. Nesse processo o usuário não enxerga esta composição. Caso a integração seja feita de forma manual, é o usuário quem fica responsável por todo o processo, desde encontrar um *Webservice* que converta o respectivo valor de Real para Dólar até procurar e utilizar outro serviço, que tenha como parâmetro de entrada um valor em Dólar, que retorne esse novo valor para Euro.

Ainda sobre composição manual e automática Murtagh (2004) vai dizer que a composição automática apresenta uma série de vantagens quando comparada com a manual, dentre elas:

- Composição de serviços automática pode ter uma série de vantagens relacionada a novos serviços. Se a composição se comporta de forma dinâmica, os *Webservices* que estarão envolvidos se apresentarão de forma multável/moldável a qualquer momento quando necessitado incluir novos *Webservices* para atingir os objetivos de forma mais rápida, mais barata, ou de melhor desempenho.

- Composição de serviços automática pode se ajustar para encontrar a melhor maneira de corresponder o pedido do usuário (escolher a melhor forma de pagamento é um dos exemplos).
- Composição de serviços automática pode copiar *Webservices* que estão indisponíveis. Se um dos *Webservices* da composição estiver indisponível ele pode ser substituído por algum outro equivalente.

E nisso, Murtagh (idem), continuando a discussão sobre composição de *Webservices* automática, acrescenta que o fluxo do processo de composição se completa por intermédio dos seguintes passos:

I. Descoberta automática dos serviços: localizando através da unidade UDDI quais os *Webservices* candidatos à composição;

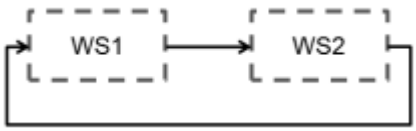
II. Composição automática de serviços: determinando quais os *Webservices* a serem executados (através da sua descrição WSDL); qual a sequência e a ordem da execução para a composição;

III. Execução automática: invocação dos serviços e o retorno do resultado para o usuário.

Webservices podem ser combinados das mais diversas maneiras em uma composição e, segundo Damasceno (*ibidem*, p.10), podem ser:

- Sequencial: existe uma sequência lógica para a execução dos serviços, onde um só pode ser realizado após o outro:

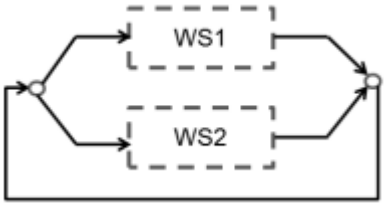
Figura 6 – Combinação sequencial



Fonte: Damasceno (*ibidem*, p.10)

- Paralela: a execução dos dois serviços acontece de forma concomitante, onde a composição precisa dos resultados dos dois *Webservices*, ao mesmo, tempo para dar continuidade da composição:

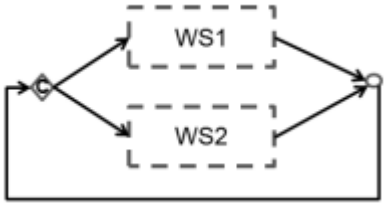
Figura 7 – Combinação paralela



Fonte: Damasceno (*ibidem*, p.10)

- Com escolha: dependendo do comportamento da composição, um ou outro serviço pode ser solicitado, sendo assim, existe a condicional que determina qual serviço seja executado no fluxo da composição.

Figura 8 – Composição com escolha



Fonte: Damasceno (*ibidem*, p.10)

3.5. MODELOS DE COMPOSIÇÃO DE SERVIÇOS

A medida que uma composição automática de serviços é executada, existem alguns passos envolvidos na mesma para que ela ocorra. Eles vão desde a seleção dos *Webservices* que irão compor, a comunicação entre os serviços e as iterações automáticas que ocorrerão entre esses serviços para que exista, como resultado, um objetivo final de acordo com a solicitação do cliente.

Existem dois principais modelos de organização destes serviços, orquestração e coreografia que, apesar de aparentarem similaridade, possuem configurações distintas em seu caráter organizacional.

3.5.1. Orquestração

É dado o nome de orquestração às composições de serviços que possuem um ponto central coordenador desta composição. Assim como sugerido em seu nome, as composições automáticas por orquestração apresentam uma unidade orquestradora (a exemplo de um maestro) que atua regendo a composição, ativando os serviços conforme a sua necessidade.

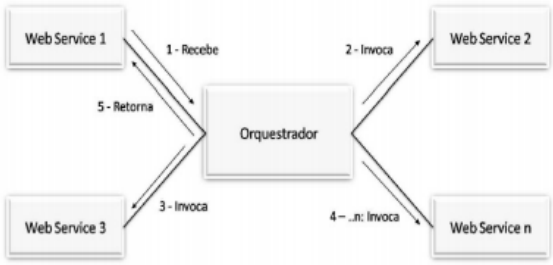
Conforme Michaels e Fileto,

> Orquestração se refere a um processo de negócio executável que pode interagir tanto com serviços internos quanto externos, sendo que essa interação é sempre representada sob a perspectiva de uma das partes, que é a que detém o controle do processo [...] A orquestração foca no comportamento sob o ponto de vista de um único participante, agindo como um controlador de um processo e conduzindo a execução desse processo seguindo sua definição. (2007, p.3, tradução nossa)

Na orquestração podem ser encontrados dois tipos de serviços:

- Processo Mestre: processo responsável pelo controle da composição de processos, organizando assim a orquestração (orquestrador). Este serviço é o responsável pela coordenação dos serviços participantes. Ele é o responsável pela coordenção das interações, da sincronia, do *workflow* de gerenciamento de transações de negócios e monitoramento de processos de negócios.
- Processo Participante: processo passivo dentro de uma composição por orquestração. Este serviço não tem conhecimento relacionado aos demais serviços participantes, sabendo apenas da existência do serviço mestre.

Figura 9 - Composição por orquestração



Fonte: Silva (2008, p.6)

A composição acima trata-se de uma típica orquestração, apresentando o serviço “orquestrador” como o processo mestre, o qual determina o *bussiness process* de execução da composição, definindo os momentos em que os *Webservices* (processos participantes) serão invocados.

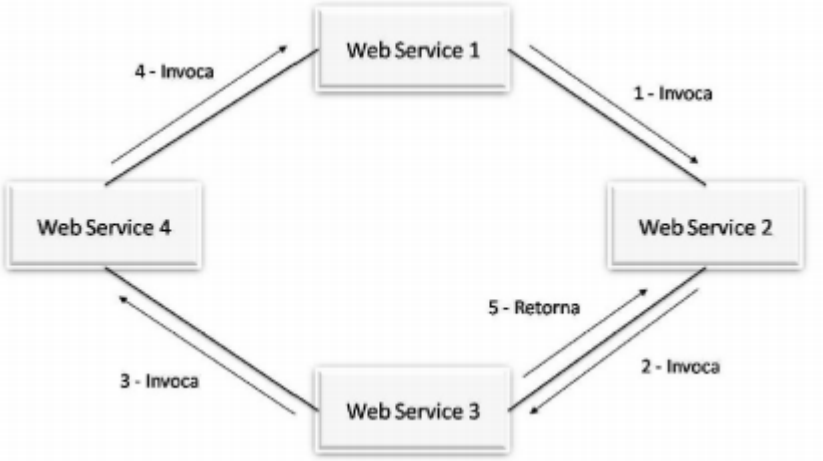
De acordo com Silva (*idem*) a grande vantagem da orquestração é que ela não requer a necessidade de refazer funcionalidades de sistemas já existentes, deixando na responsabilidade do processo mestre o retorno das mensagens e as invocações dos participantes.

O padrão mais conhecido para a descrição semântica da Composição de serviços por Orquestração é o BPEL (*Business Process Execution Language*), que será discorrido ulteriormente.

3.5.2. Coreografia

Diferentemente da orquestração, a coreografia trata-se de um modelo de composição que descreve a colaboração entre os serviços como um todo, independente de uma unidade controladora (processo mestre), detalhando a comunicação ponto-a-ponto entre os serviços que montam esta composição. Todos os serviços conhecem uns aos outros dentro da composição e, para que ocorra o seu funcionamento, é necessária a troca de mensagens entre os serviços participantes. O controle e o cumprimento do fluxo dos dados são responsabilidade de todos os serviços que a compõe:

Figura 10 - Composição por coreografia



Fonte: Silva (2008, p.6)

Segundo Silva

> A comunicação é modelada através de conexões permanentes e com manutenção de estado. Trata dos detalhes da interação dos participantes do processo executado em colaboração, descrevendo e restringindo as mensagens que cada participante pode enviar e quais respostas são esperadas. (*idem*, p.6)

A grande vantagem da utilização da coreografia na composição de serviços é a possibilidade de se ter uma visão global do comportamento da composição e das interações entre os serviços participantes, fazendo com que os mesmos sejam desenvolvidos, ou adaptados, conscientes do seu papel na composição de serviços. Nisso, diferencia-se da orquestração, onde existe o comprometimento da composição, caso os serviços não sejam desenvolvidos e projetados para trabalhar em conjunto.

A linguagem mais conhecida para a especificação semântica da composição de serviços por coreografia é a WS-CDL (*Web Services Choreography Description Language*), que também será discutida no tópico a seguir.

3.6. DESCRIÇÃO SEMÂNTICA DE COMPOSIÇÃO DE SERVIÇOS

3.6.1. Business Process Execution Language (BPEL)

BPEL é uma linguagem padrão para especificar processos de negócio e estados de processos totalmente baseada na arquitetura XML. Também conhecida como WSBPEL ou BPEL4WS, esta linguagem surgiu após um consórcio criado em 2003 entre as empresas: IBM (WSFL), Microsoft (XLANG), SAP e Siebel Systems; com o objetivo de criar uma linguagem para padronizar a integração de serviços compostos ou serviços isolados (OASIS, 2007).

De acordo com Wohed (2003), o BPEL é capaz de especificar dois tipos de processos diferentes: os processos executáveis e os processos abstratos. Processos abstratos tratam da especificação do comportamento da troca de mensagens entre as partes da composição, sem revelar para os seus parceiros o seu comportamento interno individual. Já os processos executáveis dizem da especificação da ordem de

execução entre as atividades a serem realizadas pelos serviços; dos serviços envolvidos na execução da tarefa; da troca de mensagens entre esses parceiros; e dos erros e tratamento das exceções.

O resultado da especificação BPEL funciona como um fluxograma que descreve as atividades que ocorrerão na composição, as quais podem ser divididas como atividades primitivas e estruturadas.

As atividades primitivas são as seguintes:

- *Invoke* (Invoca uma operação descrita em WSDL);
- *Recieve* (Aguarda retorno de outro serviço);
- *Reply* (Dá retorno a outro serviço);
- *Assign* (Copia o conteúdo dos dados de uma variável e atribui a outra);
- *Wait* (Fica inativo até ser reinvocado);
- *Throw* (Indica erros na composição);
- *Terminate* (Encerra a composição);
- *Empty* (Não executa nada);
- *Compensate* (Desfaz alterações em caso de erro).

Para conseguir especificar estruturas mais complexas, tem-se os seguintes operadores estruturados que determinam sequência, paralelismo, *loops* e outros:

- *Sequence* (Determina a ordem de execução dos serviços);
- *Switch* (Condicional da sequência);
- *While* (Loop na sequência);
- *Pick* (Determina a condição para a continuação da execução);
- *Flow* (Execução em paralelo de dois ou mais serviços);
- *Scope* (Trata do grupamento de serviços para receberem o mesmo retorno);
- *Link* (Determina a relação de dependência entre duas ou mais atividades).

Como exemplificação, segue abaixo um código BPEL:

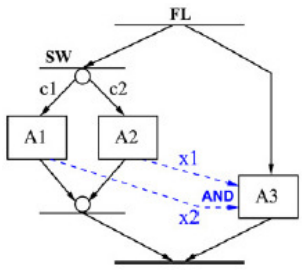
Figura 11 – Código BPEL

```
<process name="unreachableTask">
 <flow name="FL" suppressJoinFailure="yes">
  <links>
    <link name="x1"/>
    <link name="x2"/>
  </links>
  <switch name="SW">
    <case>
      <invoke name="A1">
        <sources> <source linkName="x1"/>  </sources>
      </invoke>
    </case>
    <otherwise>
      <invoke name="A2">
        <sources> <source linkName="x2"/>  </sources>
      </invoke>
    </otherwise>
  </switch>
  <invoke name = "A3">
   <targets>
    <joinCondition>
      bpws:getLinkStatus('x1') and bpws:getLinkStatus('x2')
    </joinCondition>
    <target linkName="x1"/>
    <target linkName="x2"/>
   </targets>
  </invoke>
 </flow>
</process>
```

Fonte: Ouyang (2006, p.165)

A codificação acima trata de uma composição de serviço denominada de "*Unreachable Task*", na qual existem dois links, denominados de X1 e X2. O link X1 será invocado quando a condicional existente no fluxo for para o caso A1, da mesma forma que o link X2 será invocado quando a condicional for para o caso A2. Ambos os links levam a composição para a atividade A3 que, quando executada, encerra o fluxo da composição. Assim, apresenta-se a equivalência do código acima através do seguinte *workflow*.

Figura 12 - *Workflow*



Fonte: Ouyang (2006, p.165)

3.6.2. Web Services Choreography Description Language (WS-CDL)

WS-CDL é uma linguagem também baseada em XML, que descreve a comunicação, colaboração e interação de serviços sem que haja um serviço centralizador (orquestrador), dando uma noção geral sobre o comportamento da composição entre os serviços.

Segundo Rani

> WS-CDL é uma linguagem de esperificação peer -to-peer, onde cada parte deseja permanecer autónoma e em que nenhum participante tem domínio sobre qualquer outro - ou seja, não ponto de centralização . A descrição peer-to-peer baseia-se no que denominamos de um conjunto ordenado de interações, onde uma interação é definida como a troca de mensagens entre os participantes. (2005, p.1)

A descrição em WS-CDL pode ser dividida em duas partes: estática e dinâmica. A parte estática determina as relações que são invariantes, como por exemplo, os tipos de dados, os participantes da comunicação e os canais de comunicação. Já a parte dinâmica da especificação determina as características mutáveis na composição, que podem ser os tipos de interações entre os participantes, bem como sua sincronia (paralelismo ou sequencia).

De acordo com Fredlund (2006), a parte estática da codificação WS-CDL pode ser dividida em:

- Tipos de função (*Role Types*): falam quais os tipos de funções que os participantes terão na coreografia. Como exemplo podemos citar um serviço de venda que pode ser tanto atacado como varejo;
- Tipos de participantes (*Participant Types*): falam que tipos de caráter que os participantes poderão assumir na coreografia, assumindo assim tipologias diferentes mesmo sendo, a priori, o mesmo serviço;
- Tipos de relações (*Relationship Types*): determina o tipo de relação que dois serviços terão em uma coreografia;
- Tipos de canais (*Channel Types*): descreve o meio usado para a comunicação entre os serviços compostos.

Ainda em Fredlund (2006) pode ser visto que a parte dinâmica da especificação é onde o coração da coreografia vai estar definido. Este local também é responsável por descrever o conjunto de relações e interações entre os diversos tipos de funções e por definir o conjunto de variáveis existentes na composição para realizar a análise do comportamento dos dados na coreografia. Ele pode ser dividido em:

- *Basic Activities*: determina as atividades e os tipos de interações que os serviços podem ter um com o outro, como por exemplo: *reply-response*;
- *Workunits*: fala dos fatores condicionais e *loops* na comunicação entre os serviços;
- *Structural Activities*: fala sobre os fatores sequenciais (paralelismo e sequências).

3.7 WEB ONTOLOGY LANGUAGE FOR SERVICES (OWL-S)

OWL-S é uma linguagem de descrição semântica baseada em OWL (*Web Ontology Language*) dessarte, em XML, utilizada para definir e instanciar ontologias[3] de *Websevices*. O grande objetivo desta descrição é atribuir um sentido semântico para serviços ou composição de serviços pois, quando se tem apenas a descrição em WSDL têm somente conhecimentos com relação aos métodos e *gates* de entrada e saída destes *Webservices*. Como é necessário a existência de serviços não ambíguos, com resultados exatos, precisa-se conhecer, também, o comportamento semântico e a exatidão dos resultados das solicitações realizadas a estes *Webservices.*

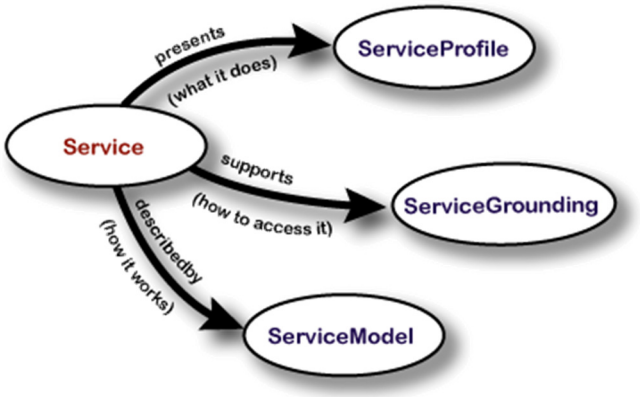
De acordo com Murtagh (2004) uma composição de serviços deve ser descrita de forma que não haja ambiguidades e que os seus seguintes estados estejam bem descritos e acessíveis a quaisquer que os solicitem.

- Propriedades e recursos – para descoberta automática;

[3] Uma ontologia é a especificação de uma conceitualização. Ela define um conjunto de primitivas representações com as quais podemos modelar um domínio ou discutir sobre ele. Essa representação primitiva é tipicamente dividida em classe, atributos e relacionamentos. Para maiores aprofundamento indicamos a obra de Gruber intitulada “Ontology” de 2009.

- Pré-condições, efeitos, entradas e saídas – para composições automáticas;
- Interfaces, finalizações e protocolos – para execução automática.

Figura 13 – Níveis da ontologia de um serviço



Fonte: http://www.daml.org/services/owl-s/1.1/overview/

A partir da imagem a cima pode ser percebido que a ontologia se dá mediante 3 níveis diferentes – *ServiceProfile, ServiceGrounding* e *ServiceModel* – mas que se articulam e trabalham em conjunto para descrever o comportamento semântico de cada serviço ou composição de serviços.

De forma mais detalhada, Martin (2007) fala que cada nível trabalha da/na seguinte perspectiva:

- *ServiceProfile:* é a subontologia que permite a descrição das propriedades e características do serviço, possibilitando, assim, a otimização da descoberta automática, uma vez que a torna mais fácil e ágil.
- *SeviceModel (ProcessModel):* é a subontologia que vai se encarregar pela descrição do funcionamento do serviço, onde faz um detalhamento semântico do conteúdo do mesmo, delineando informações como processo sequencial, resoluções, condicionais, *loops*, dentre outros. Ele é tido como o coração da descrição semântica, pois é nele que temos as seguintes estruturas de controle: *Sequence, Split, Split+Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While* e o último deles *Repeat-Until.*
- *ServiceGrounding:* é a subontologia que determina como o serviço será acessado, detalhando informações como os protocolos de trocas de

mensagens, serializações, transporte e endereçamento. Pode também ser visto como uma forma de mapeamento do abstrato para uma especificação concreta, fazendo com que os elementos da descrição sejam vistos como objetos acessíveis para a interação. Portanto, é possível verificar que a parte mais concreta da ontologia está contida na subontologia em questão, já que o *ServiceProfile* e o *ServiceModel* tratam de informações de níveis mais abstrato (fluxos, descrições e especificações).

4. TÉCNICAS DE VERIFICAÇÃO DE WEBSERVICES COMPOSTOS: RESULTADOS E SUAS RESPCTIVAS DISCUSSÕES

Composições de *Webservices*, como já mencionado, é um aglutinado de *Webservices* mais simples que têm como objetivo a criação de uma funcionalidade mais complexa, visando atender as mais diversas solicitações dos clientes. As composições apresentam um conjunto de técnicas e tecnologias base que permitem o seu reuso, a sua composição automática, a sua ressignificação e adequação, tendo sempre que estar disponível para o cliente (W3C, 2004).

As composições, em especiais as de caráter automático, apesar de serem fortemente acopladas, na tentativa de não apresentar falhas e erros, não garantem que, durante o seu funcionamento, os mais diversos problemas possam ocorrer. Dentre esses problemas podemos citar, como exemplos, os estados de *deadlock*, as composições ambíguas, não exatas, inconsistências, os problemas de corretude, de logica temporal e confiabilidade.

Segundo Rodrigues (2009), a falta de especificações em padrões atuais – por exemplo: BPEL e WSCDL –, o aumento da complexidade e requisitos de confiabilidade mais rigorosas em serviços de negócios, fomentam a necessidade de abordagens de confiança para design de serviços.

Assim, vê-se que existe a necessidade de aplicar técnicas de verificação e validação destas composições para que as mesmas possam, a partir de uma articulação precisa, desempenhar suas funções de forma exitosa. Para tal, foram visualizadas a utilização de técnicas de abordagens transacionais, ou seja, modelos que utilizam transições para descrever um sistema computacional através de mudança de estados.

Para realizar a verificação das composições através de abordagens transacionais é preciso que exista a análise dos serviços que a compõem e o mapeamento/tradução desta composição para alguma abordagem transacional, visando descrever o seu comportamento, os seus estados e transações, culminando numa validação matemática.

Para realizar o levantamento das abordagens existentes para solucionar a problemática da necessidade da validação das composições de *Webservices*, foi

realizada a pesquisa em caráter quanli-quanti, *survey*, usufruindo da coleta de artigos científicos publicados nas seguintes bases de dados: ACM, Elsivier e IEEE. A busca foi realizada na biblioteca de dados *Scholar Google* e, para sua concretização, foram utilizadas palavras-chave como: *Webservice composition verification* e *verification approach to webservice composition*.

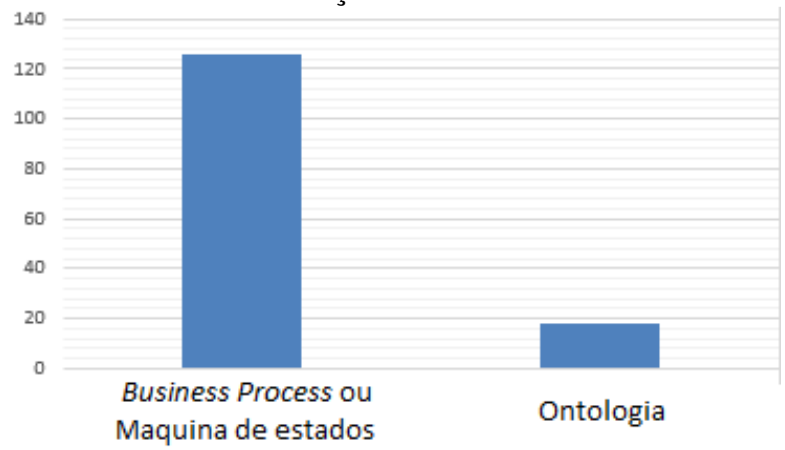
Após a busca, foi necessária a aplicação de um filtro de dados através da leitura dos títulos e resumos, visando assim a realização da pesquisa somente com artigos que tratassem unicamente da temática em questão.

Ao término da coleta de dados, deu-se segmento com a leitura das publicações que se enquadravam no perfil da pesquisa. Doravante, foi preenchido um questionário com as seguintes informações de cada publicação: ano, tipo de abordagem, tipo de solução e estudos de caso.

Ao realizar-se as leituras dos artigos mapeados dentro dos parâmetros estabelecidos (verificação de *Webservices* compostos), foi observado que existem duas possibilidades de mapeamento/tradução das composições. Essas possibilidades materializam-se nas formas de análise semântica (ontologia) da composição e análise do processo de negócio (*Bussiness Process*).

Foi visualizada a seguinte distribuição de análises:

Gráfico 1 – Gráfico de distribuição de análises



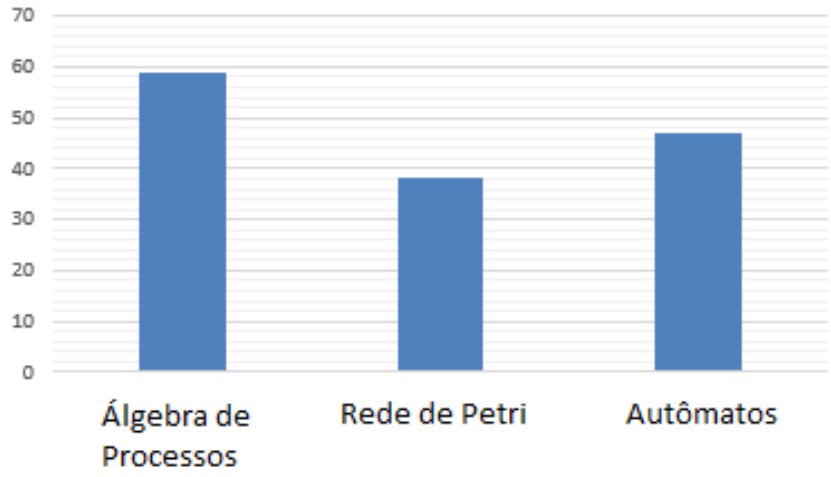
Ou seja, dos 144 artigos analisados (*vide* Apêndices A, B e C), 18 autores tiveram a preferência pela tradução e verificação de ontologias e 126 preferiram analisar o *Bussiness Process* ou os estados da composição. Essa preferência se dá, de acordo

com Wombacher (2004), porque a utilização das linguagens de *Bussiness Process* permite a visualização dos estados internos dos processos (serviços) e facilita não só em sua tradução para algum método transacional, mas também descreve todas as possibilidades de estados que o serviço ou a composição possa vir a assumir.

Para tanto, apesar da ontologia ser utilizada em menor escala, Vidal (2011) esclarece o porquê de ainda se necessitar desse tipo de análise. Ele diz que a utilização das ontologias, para a tradução, oferta um conjunto de descritores e informações, nele contidas, que facilita a compreensão com relação a composição de serviço como um todo, otimizando a sua descoberta, a sua descrição comportamental e sua invocação. Enquanto o *Bussiness Process* trata da descrição dos estados dos serviços, a ontologia visará relatar o comportamento da composição.

Dentre as diversas técnicas encontradas para verificar as falhas que, por ventura venham se apresentar no processo de execução das composições de serviços, identificou-se 3 tipos de abordagens para subsequentes soluções: Redes de Petri, Álgebra de Processos e Autômatos. Tem-se abaixo a seguinte tabela que ilustra o quadro quantitativo com relação as soluções.

Gráfico 2 – Gráfico quantitativo das abordagens transacionais apresentadas como técnicas para validação e verificação de webservices compostos.



As abordagens realizam os mapeamentos das codificações sejam em BPEL, WS-CDL ou OWL-S para as linguagens acima citadas, ou vice-versa, realizando assim a tradução de seus operadores para as linguagens formais escolhidas e

usando, em sua grande maioria, ferramentas para análise deste mapeamento, destarte, verificando estas composições.

Neste sentido, serão analisadas as abordagens e os formalismos utilizados para verificação das composições, oriundos do levantamento realizado via técnica survey, explicitando assim os motivos à sua utilização, bem como as tipologias utilizadas e alguns estudos de caso.

4.1. REDE DE PETRI

A Rede de Petri é um formalismo que permite a modelagem de sistemas dinâmicos possuindo, através de seus métodos matemáticos e gráficos, grande expressividade para representação de relações de diversos tipos, como: sequência, conflitos, concorrências e sincronia (LOBO, 1999). Rede de Petri é uma técnica de especificação formal bem estabelecida, largamente difundida e adequada à modelagem de sistemas que tenham atividades paralelas, concorrentes, assíncronas e não-determinísticas (MACIEL, 1996).

Este tipo de solução é aplicado em diversas problemáticas e, uma delas, é na composição de *Webservices*, na tentativa de realizar a validação e verificação de sistemas complexos, devido ao fato de ser um formalismo que descreve os seus estados e eventos. Cada evento possui pré e pós-condições, bem como barras que simbolizam as transições entre eles.

Assim, percebeu-se a utilização, por alguns pesquisadores como Hinz (2005), Verbeek (2006) e Vidal (2011), desta abordagem para solucionar a problemática em questão, onde usufruíram deste formalismo para validar e verificar as composições de *Webservices*.

A exemplo, a abordagem realizada por Fan (2013), utiliza a Rede de Petri como forma de descrever a formalização da composição do serviço e o relacionamento entre os *Webservices* (onde são verificadas as transações, confiança e as falhas são identificadas). Em seguida, é proposta uma estratégia de composição de serviços confiáveis, fazendo com que o comportamento seja verificado na fase da

análise, ou na própria execução, para assim reparar os erros do projeto. Os passos são:

- Descrição de controle de fluxo (estados aceitos e confiabilidade) e a proposta da composição do serviço;
- Análise entre os estados aceitáveis e os que são possíveis de serem encontrados na composição dos serviços proposta;
- Criação de uma estratégia confiável e o algoritmo de execução, utilizando técnicas de redundância para que o serviço esteja sempre disponível e confiável;
- Formalização com Rede de Petri, para verificar a eficácia do método.

Além da abordagem padrão da Rede de Petri, verificou-se que duas outras extensões foram utilizadas no mapeamento, sendo elas:

- Rede de Petri Colorida: são redes que utilizam cores para poderem representar diferentes processos ou recursos em uma mesma rede, diferindo apenas na cor que as mesmas possuem – foi utilizada por Ni (2010) e por Cardinale (2010). Esta extensão tem como objetivo diminuir o tamanho da rede complexa e permitir que os estados sejam individualizados em função das cores que os mesmos possuem (FRANCES, 2003);

- Rede de Petri Temporizada: apresentam tratativas com relação a tempo, facilitando na representação de atividades concorrentes, síncronas e assíncronas – foi utilizada por Gralet (2011) e por Boukadi (2006). A figura de tempo pode ser atribuída a estados, transações e a quando poderá ser disparada uma transação (FRANCES, *idem*).

Como exemplificação dessa abordagem, observa-se, no estudo de caso sobre uma proposta de mapeamento, feita por Verbeek (2006), a exemplificação de uma verificação realizada por intermédio da especificação da seguinte composição de serviços composta por 5 atividades: *getBuyerInformation, getSellerInformation, settleTrade, confirmBuyer* e *confirmSeller*. O fluxo do processo de negócio trata de uma transação comercial e segue o seguinte padrão: os serviços *getBuyerInformation* e *getSellerInformation* podem rodar concomitantemente e, quando ambos estiverem

concluídos, o *settleTrade* se inicia após a finalização dos serviços anteriores, onde após a sua conclusão, os serviços *confirmBuyer* e *confirmSeller* são iniciados. Caso seja concluído corretamente, termina-se a transação.

Para a definição de sua especificação temos o seguinte fluxo descrito em BPEL:

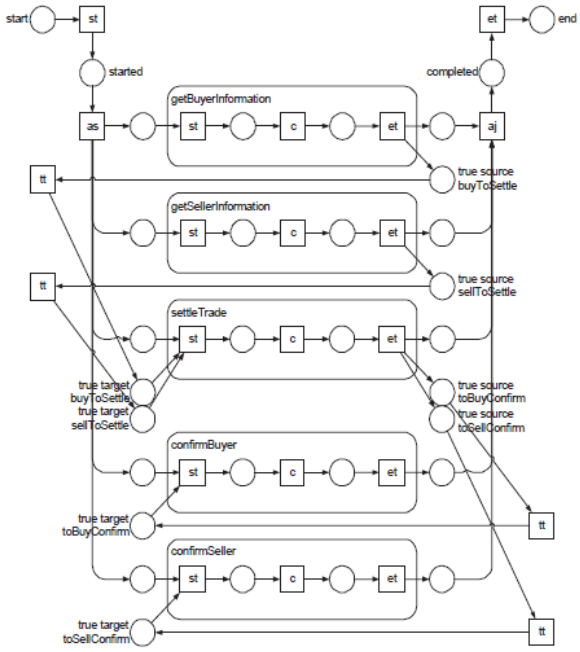
Figura 14 – Código BPEL de uma composição de serviço: serviço de compra

```
<flow suppressJoinFailure="yes">
  <links>
    <link name="buyToSettle"/>
    <link name="sellToSettle"/>
    <link name="toBuyConfirm"/>
    <link name="toSellConfirm"/>
  </links>
  <receive name="getBuyerInformation">
    <source linkName="buyToSettle"/>
  </receive>
  <receive name="getSellerInformation">
    <source linkName="sellToSettle"/>
  </receive>
  <invoke name="settleTrade"
          joinCondition="bpws:getLinkStatus('buyToSettle') and
                         bpws:getLinkStatus('sellToSettle')">
    <target linkName="getBuyerInformation"/>
    <target linkName="getSellerInformation"/>
    <source linkName="toBuyConfirm"/>
    <source linkName="toSellConfirm"/>
  </invoke>
  <reply name="confirmBuyer">
    <target linkName="toBuyConfirm"/>
  </reply>
  <reply name="confirmSeller">
    <target linkName="toSellConfirm"/>
  </reply>
</flow>
```

Fonte: Verbeek (2006, p.10)

Para verificar a composição acima, é realizado o mapeamento de forma manual, através da leitura do código BPEL acima, para a seguinte Rede de Petri:

Figura 15 – Rede de Petri da composição de serviço: serviço de compra



Fonte: Verbeek (2006, p.11)

Para auxiliar na especificação, além dos estados dos processos, são utilizadas transições, como: st (*startTrue*), c (*condition*), et (*endTrue*), tt (*trueTrue*), as (*AND-slit*) e aj (*AND-join*).

Após o estado inicial, as condições são abertas através da transição ‘as’ e se inicia a execução dos serviços *getBuyerInformation* e *getSellerInformation*. A posteriori, existe uma transição associada a finalização correta dos dois serviços, representado por dois ‘tt’s que são os pré-requisitos para o disparo da execução do estado *buytoSettle/selltoSettle* que, quando encerrado, aciona as transições ‘tt’s que invocam os serviços *confirmBuyer* e *confirmSeller*, os quais finalizam a composição e enceram a atividade após a transição ‘aj’ final.

A verificação desta Rede de Petri é realizada através da ferramenta *Woflan*, a ser utilizada junto do *Oracle BPEL Process Manager*, obtendo assim o diagnóstico da composição em questão.

4.2. ÁLGEBRA DE PROCESSOS

A álgebra de processos é um estudo de sistemas distribuídos, ou paralelos, de forma algébrica. Quando se fala de álgebra, fala-se que as abordagens que se enquadram nesta tipologia analisam o comportamento destes sistemas supracitados de forma algébrica e axiomática. A álgebra de processos pode ser definida como uma estrutura matemática que satisfaz os axiomas dados através de operadores básicos (BAETEN, 2006).

Nicola (2011) complementa dizendo que o componente primordial de uma álgebra de processo é a sua sintaxe, que é determinada pela combinação bem formada de operadores e mais alguns termos elementares.

De acordo com Bergstra e Klop (1984), as álgebras de processos são baseadas em um conjunto de operadores e ações básicas que definem a execução alternativa, sequencial e paralela. Elas são usadas para obter um resultado algébrico mais satisfatório que simplifica a análise e a representação de um sistema computacional.

Existe uma série de álgebras de processos que são propostas, dentre elas, SALAÜN (2004) fala que as mais importantes são: a *Calculus of Communicating Systems* (CCS), *Algebra of Communicating Process* (ACP), *Communicating Sequential Process* (CSP). Ele também coloca nesse rol as mais recentes, como o π-*calculus*, LOTOS, *Process or Protocol Meta Language* (Promela) e *Timed* CSP. Estas linguagens compartilham dos mesmos ingredientes: construtores simples para descrever comportamentos dinâmicos, modelos de composição, operadores semânticos (or/and), a análise comportamental via *model checking* e equivalência de processos.

Entretanto, dentre as abordagens de análise de composições de *webservices*, foram encontradas apenas abordagens que utilizam: Promela, CCS, Lotos e π-calculus.

Salaün (2004) e Camara (2005) preferem utilizar o CCS como linguagem de especificação devido a pequena quantidade de operadores, embora esses operadores estejam em pequena quantidade, isso não reduz o nível de expressividade na realização de um mapeamento de processo de negócios/ontologia presente em uma composição de serviços. Esta linguagem é muito usada devido a sua especificação permitir a avaliação da corretude de propriedades, como também *deadlocks* e *livelocks*.

Todica (2012) usufrui da Promela como linguagem utilizada para o mapeamento de um modelo de composição de serviço, realizando o *model checking* da composição através da utilização do *framework* SPIN. Esta ferramenta é capaz de gerar um *Linear Temporal Logic* (LTL) equivalente. Ao se obter o código BPEL da composição, o framework Promela realiza a tradução para a linguagem Promela, onde há, junto ao SPIN, a realização do *model checking* e a verificação de propriedades como a corretude, *deadlocks* e *livelocks*. Após esse processo, pode ser realizada a geração do LTL equivalente para validar a composição.

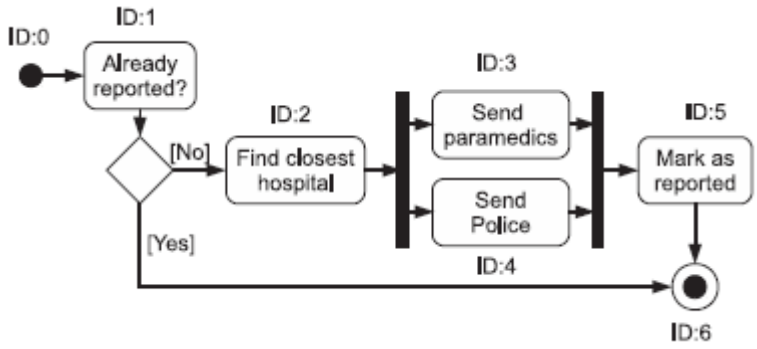
Salaün e Chirichiello (2004), Ferrara (2004) e Dumez (2013) utilizam o LOTOS e o CADP como linguagem e ferramenta de especificação para a resolução da tradução e verificação da composição de webservices. Este tipo de linguagem combina dois tipos de modelo de especificação: aspectos estáticos (dados e operadores) e os aspectos dinâmicos (processos).

Utilizar-se-á agora o estudo de caso, realizado por Dumez (2013), que expressa a abertura de um chamado de emergência. Para tal, existe uma composição de serviços com 4 webservices, que são:

- *HospitalLocator* – serviço que encontra o hospital mais próximo ao local do acidente;
- *Paramedics* – serviço que envia médicos para o local;
- *PoliceDispatch* – serviço que encaminha a patrulha para o local;
- *ReportsDatabase* – serviço que checa se a emergência já foi registrada e, caso não tenha sido, realiza o registro da mesma (dois métodos).

Para visualizar o seu *workflow* temos o seguinte diagrama de atividade:

Figura 16 – Diagrama de atividade UML-AD: emergência médica



Fonte: Dumez (2013, p.6)

Ao se deparar com a realização de um chamado de emergência, primeiro identifica-se se o mesmo já foi comunicado e, caso já tenha sido, a composição entra em estado final. Caso não, primeiro se localiza qual o hospital mais próximo ao local do acidente, posteriormente, invoca o envio da equipe médica e dos policiais; somente quando ambos são solicitados, ocorre a marcação de chamado comunicado e o processo entra em estado final da composição.

Para traduzir o diagrama, o primeiro passo a ser realizado é criar um processo para cada estado da atividade (incluindo o estado inicial e final) e assim dar um identificador diferente a cada um deles. Em seguida deve ser pensado na concorrência e sincronia entre eles. Assim, com o auxílio da ferramenta CADP, a especificação equivalente para o diagrama cima exposto seria:

Figura 17 – Especificação full lotos: emergência médica

```
specification EmergencyResponse [SEND, RECV]: noexit
behavior
  (
  Init [SEND, RECV](0) |||
  CheckIfReported [SEND, RECV](1) |||
  FindHospital [SEND, RECV] (2) |||
  SendParamedics [SEND, RECV](3) |||
  SendPolice [SEND, RECV](4) |||
  MarkAsReported [SEND, RECV](5) |||
  Final [SEND, RECV](6)
  )
  |[SEND,RECV]|
  BUS [SEND,RECV] ( < > )
where
  (* Processes definition *)
endspec

process Service1 [SEND, RECV] (Id:Int) : exit :=
  (* Do work *)
  Sequence [SEND, RECV] (Id,2) > > exit
  where
  process Sequence [SEND, RECV] (Id:Int, Id_dst:Int): exit :=
SEND !Id_dst !Id !RUN !void; exit
  endproc
endproc

process Service2 [SEND, RECV] (Id:Int) : exit :=
  (* Wait for message *)
  RECV !Id ?Sender:Int !RUN !void;
  (* Do work *)
endproc

process ParallelSplit [SEND, RECV] (Id:Int, Ids_dst:IntSet) :
  exit :=
  [empty(Ids_dst)] - > exit
  []
  [not(empty(Ids_dst))] - >
    (let Dest:Int = pick(Ids_dst) in
      SEND !Dest !Id !RUN !void;
      ParallelSplit[SEND, RECV](Id, remove(Dest, Ids_dst))
  )
endproc

process Synchronization [SEND, RECV] (Ids_src:IntSet, Id:Int)
    : exit :=
  [empty(Ids_src)] - > exit
  []   [not(empty(Ids_src))] - >
    RECV !Id ?Id_src:Int !RUN !void [Id_src isin Ids_src];
    Synchronization [SEND, RECV] (remove(Id_src, Ids_src),
Id)
endproc

process ExclusiveChoice [SEND, RECV] (Id:Int,
    Ids_dst:IntSet): exit :=
  (choice Dest:Int []
    [Dest isin Ids_dst] - >
      SEND !Dest !Id !RUN !void;
      exit)
endproc

process SimpleMerge [SEND, RECV] (Ids_src:IntSet, Id:Int) :
    exit :=
  RECV !Id ?Id_src:Int !RUN !void [Id_src isin Ids_src];
  exit
endproc

process Init [SEND, RECV] (Id:Int) : exit :=
  Sequence [SEND, RECV] (Id, 1)
  > > exit
endproc

process CheckIfReported [SEND, RECV] (Id:Int) : exit :=
  RECV !Id !0 !RUN !void;
  ExclusiveChoice [SEND, RECV] (Id, insert(6, insert(2, {})))
  > > exit
endproc

process FindHospital [SEND, RECV] (Id:Int) : exit :=
  RECV !Id !1 !RUN !void;
  ParallelSplit [SEND, RECV] (Id, insert(4, insert(3, {})))
  > > exit
endproc

process SendParamedics [SEND, RECV] (Id:Int) : exit :=
  RECV !Id !2 !RUN !void;
  Sequence [SEND, RECV] (Id,5) > > exit
endproc

process MarkAsReported [SEND, RECV] (Id:Int) : exit :=
  Synchronization [SEND, RECV] (insert(4, insert(3, {})), Id)
  > >
  Sequence [SEND, RECV] (Id,6) > > exit
endproc

process Final [SEND, RECV] (Id:Int) : exit :=
  SimpleMerge [SEND, RECV] (insert(5, insert(1, {})), id)
  > > exit
endproc
```

Fonte: Dumez (2013, p.6)

Para validar a especificação acima exposta, é necessária a geração do LTS equivalente obtendo assim, a verificação da corretude e de todos os estados em que a composição pode assumir. O LTS é gerado através do CADP:

Figura 18 – Gráfico LTS: emergência médica

Fonte: Dumez (2013, p.8)

Posteriormente a verificação da composição ter sido concluída, pode-se então gerar um código BPEL equivalente, após tradução do código LOTOS obtido e verificado partindo do UML-AD da composição. Segue a codificação em BPEL:

Figura 19 – Código BPEL: emergência médica

```
< process name = EmergencyResponse >
  < sequence >
    < receive operation = ReportEmergency    / >
    < invoke name = "wasAlreadyReported" . . . / >
    < if name = "notAlreadyReported" >
      < condition > not_reported < /condition >
      < sequence >
        < flow name = "Parallel" >
          < invoke name = "SendParamedics" . . . / >
          < invoke name = "SendPolice"    / >
        < /flow >
        < invoke name = "markAsReported"    / >
      < /sequence >
    < /if >
    < reply operation = ReportEmergency . . . / >
  < /sequence >
< /process >'
```

Fonte: Dumez (2013, p.8)

4.3. AUTÔMATOS

Um autômato finito determinístico é um modelo matemático computacional utilizado para definir sistemas computacionais e sequências lógicas. Eles servem como técnica de especificação formal em diversos tipos de situações e, um dos fatores que contribuem para isso, é o fato da sua possível representação através de grafos que podem ser utilizados para prover interfaces de programação (MACHADO, 2000).

Um autômato finito determinístico é dividido em: símbolo de entrada, conjunto finito dos estados do autômato, funções de transição, estado inicial e o conjunto de estados finais (BOECHAT, 2008). Em se tratando de composição de *Webservices*, pode-se fazer a seguinte analogia: os símbolos de entrada determinarão quais as entradas/saídas de cada estado; os estados da composição determinarão em que ponto uma composição se encontra e qual estado ela assume em um contexto geral; as transições determinam as operações a serem realizadas; o estado inicial seria a invocação da composição, assim como o final seria a conclusão da mesma.

Nas abordagens onde foram utilizados os autômatos finitos determinísticos como solução, viu-se dois tipos utilizados: os *Timed Automata* e os *Symbolic Finite Automata* (SFA).

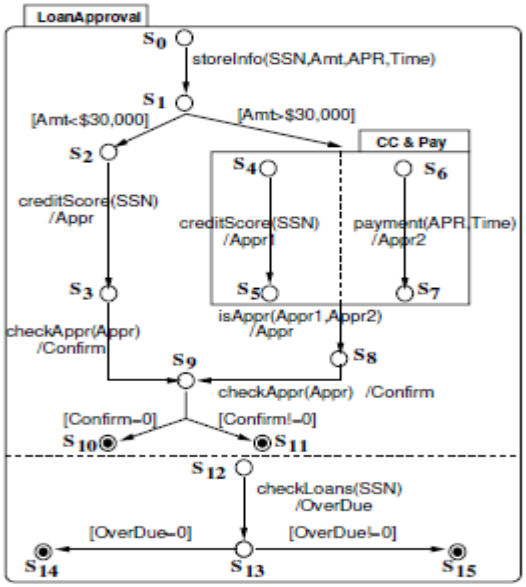
Phatak (2006) e Ouederni (2011), entre outros, utilizam o SFA, em especial com a extensão Symbolic Transition Systems (STS) para inserir informações na sua tupla de especificação que não são encontradas em um AFD comum, como: nome da

mensagem trocada pelos estados, direção da comunicação (envio/recepção) e a lista do tipo de dados que serão recebidos/enviados por esse estado.

Já Diaz *et al.* (2006) e Pu *et al.* (2006), preferem utilizar os *Timed Automata*, que são poderosos descritores de ferramentas de tempo-real pois, devido a existência de controladores temporais (relógios) em transições entre estados, podem definir fatores sequenciais, concorrentes e paralelos. Para este tipo de autômato, existe uma ferramenta de verificação chamada UPPAAL.

Como caso de estudo, vamos analisar Pathak (2006), que trata de uma aprovação de operação de empréstimo. Para tal, tem-se uma composição de serviços no qual existem as informações do cliente (*storeInfo*) e o pedido de empréstimo. Para a realização, é informado o valor da operação e, caso seja menor que 30 mil dólares, a operação é verificada (*creditScore*) e aprovada (*checkAppr*). Caso seja maior que 30 mil dórares, é feita a análise do crédito (*creditScore*) e a verificação dos pagamentos (*payment*) para, depois, outorgar a aprovação (creditScore). Como parâmetro de entrada para a verificação do autor, apresenta-se a máquina de estados, em UML, com as funções que acontecem em cada transação de estados:

Figura 20 – Maquina de estados UML: aprovação de crédito

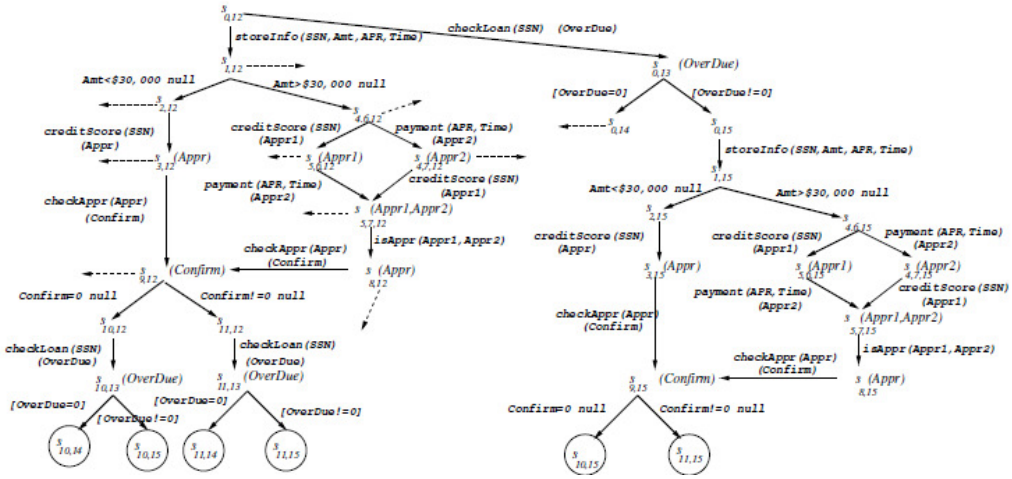


Fonte: Pathack (2006, p.5)

Para verificar a composição, é necessário que haja a tradução da máquina de estados para um autômato finito determinístico temporal, visando descrever o

comportamento desta composição e verificar a sua validade perante a sua execução. E, para o caso acima citado, tem-se a figura abaixo:

Figura 21 – STS: empréstimo



Fonte: Pathack (2006, p.6)

Após verificar o autômato, o autor propõe a criação de uma ferramenta capaz de transformar o AFN em OWL-S, com o objetivo de ter uma análise semântica sobre a composição.

Poucas abordagens são completas, ou seja: nem todos os descritores das linguagens de descrição de webservices compostos ou da sua ontologia podem ser descritos, fazendo assim com que algumas abordagens sejam parciais, não conseguindo solucionar, de forma completa, o problema das composições.

5. CONSIDERAÇÕES FINAIS

Este trabalho aborda a utilização de serviços como unidades autônomas, reusáveis e a sua utilização em grande escala para implementação de atividades, desde simples e/ou complexas. Estes serviços podem ser agrupados devido as técnicas fornecidas através da arquitetura SOA e vem sendo difundido em grande escala devido a velocidade de troca de informações e a necessidade destas aplicações estarem disponíveis para os seus consumidores através de *Webservices.*

Para desempenhar tarefas complexas, os *Webservices* podem se organizar conforme a sua necessidade, formando assim composições de serviços com o objetivo de atender as demandas de consumo. Estes serviços que irão formar os serviços compostos se comunicam por intermédio do protocolo SOAP, e as suas descrições individuais em WSDL estão contidas em unidades repositórias de dados UDDI.

Ao formarem uma composição de forma automática, não necessariamente temos um serviço composto que corresponda a atividade desejada pelo seu consumidor. Problemas de correture, *deadlocks*, indisponibilidade e composições ambíguas podem ser geradas tornando, assim, falha a operação. Para tal, vê-se a necessidade da análise da composição de serviços gerada, objetivando realizar a sua verificação.

Para analisar as composições, são utilizadas abordagens transacionais para descrever os estados em que esta composição possa vir a assumir e quais as transações que existem entre esses estados. Elas realizam o mapeamento das linguagens de descrição do processo de negócio (BPEL, WS-CDL) ou ontologias (OWL-S) para algum método transacional, realizando assim a verificação.

Através da pesquisa realizada por estre trabalho foi possível identificar 3 métodos objetivando verificar as composições: a utilização de Rede de Petri, Álgebra de Processos e Autômatos. Verificou-se ainda a utilização de algumas extensões, das técnicas acima citadas, para solucionar deficiências destas abordagens, como por exemplo: *Timed Automatas* para a representação de tempo através de implementação de relógios em suas transições; Rede de Petri colorida, visando atribuir valores diferentes para um mesmo estado, na tentativa de simplificar a análise dos seus

ramos; e *Full Lotos* que busca descrever, além dos seus atributos temporais, também as trocas de mensagens e as entradas/saídas de cada processo.

Como parâmetros para verificação, foi constatado a partir da análise dos artigos, a utilização de diagramas de atividades, diagramas de estados, descrição de processo de negócio e análise semântica para a tradução e, posterior verificação de corretude de uma composição**.** Não obstante, também foi visualizado o método reverso, onde se é gerada uma composição de serviços a partir de uma especificação transacional correta, obtendo como resultado um processo de negócio ou ontologia gerado a partir do mapeamento.

Embora existam diversas técnicas para solucionar o problema, poucas conseguiram realizar a proposta de mapeamento total de uma linguagem de descrição de uma composição – seja ela de processo de negócio ou ontológica – para o formalismo utilizado à verificação. E nenhuma delas foi capaz de propor a tradução para quaisquer tipos de descrição para formalismo, como por exemplo: conseguir mapear tanto orquestrações quanto coreografias, em BPEL ou WS-CDL respectivamente, ou utilizando a ontologia da composição para alguma abordagem transacional.

Portanto, junto com esta proposta completa existe, também, a necessidade de uma ferramenta que realize, no momento da composição, sua tradução para um método transacional a fim de verifica-la e, caso esteja falha, seja corrigida antes de sua execução.

Vê-se então a necessidade de criação de uma técnica adaptável, que consiga solucionar a problemática em questão de forma completa, mapeando/traduzindo composições dos mais diversos tipos e descritas em suas linguagens específicas. Junto com esta técnica, também é necessária a criação de uma ferramenta de tradução para realizar este mapeamento e, posteriormente, a sua verificação. Para tal, tem-se estes objetivos acima citados como proposição para trabalhos futuros.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ALERGUS,Eldorina-Andreea; JIPA, Andreea-Paula. **Web Service Composition**. Disponível em: <http://docslide.us/search/?q=Automatic+Semantic+Web+Services+Composition>. Acessado em: 2 de dezembro de 2015.

ALONSO, Gustavo; CASATI, Fabio; KUNO, Harumi; MACHIRAJU, Vijay**. Web Service – Concepts, Architetures and Applications.** Disponível em: <https://www.inf.ethz.ch/personal/alonso/Web-book/Chapter-5.pdf>. Acessado em: 2 de dezembro de 2015.

BAETEN, J.C.M.; BEEK, D.A. van; ROODA, J.E**. Process Algebra.** Disponível em: <http://mate.tue.nl/mate/pdfs/8509.pdf>. Acessado em: 2 de dezembro de 2015.

BEEK, Maurice H. ter; BUCCHIARONE, Antonio; GNESI, Stefania. **A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods**. Disponível em: <http://www.emse.fr/~beaune/websem/cours2008_2009/AlainLeger/Articles/ASurveyOnServiceCompositionApproaches-FromIndustrialStandardsToFormalMethods_2006.pdf>. Acessado em: 6 de novembro de 2015.

BENIGER, James R. **The Control Revolution: Technological and Economic Origins of the Information Society.** Disponível em: <http://en.wikipedia.org/wiki/Information_society#cite_note-Beniger_1986-1>. Acessado em: 4 de abril de 2015.

BERGSTRA, J. A.; KLOP, J. W. **Process Algebra for Synchronous Communication**. Disponível em: <http://www.sciencedirect.com/science/article/pii/S001999588480025X>. Acessado em: 2 de dezembro de 2015.

BOECHAT, Glaucya Carreiro. **Investigação de um Modelo de Arquitetura Biométrica Multimodal para Identificação Pessoal.** Disponível em: <http://www.cin.ufpe.br/~gcb/Dissertacao_GLAUCYA_BOECHAT.pdf>. Acessado em: 2 de dezembro de 2015.

BOUKADI, Khouloud; GHEDIRA, Chirine; MAAMAR, Zakaria; BOUCHENEB, Hanifa. **Specification and Verification of Views over Composite Web Services Using High Level Petri-Nets.** Disponíve em: <http://liris.cnrs.fr/Documents/Liris-2429.pdf>. Acessado em: 2 de dezembro de 2015.

CAMARA, Javier; CANAL, Carlos; CUBO, Javier; VALLECILLO, Antonio. **Formalizing WSBPEL Business Processes Using Process Algebra.** Disponível em: <http://dl.acm.org/citation.cfm?id=1706772>. Acessado em: 2 de dezembro de 2015.

CARDINALE, Yudith; HADDADB, Joyce E; MANOUVRIERB, Maude; RUKOZB, Marta. **Web service selection for transactional composition.** Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050910003030>. Acessado em: 2 de dezembro de 2015.

CARVALHO, Marilia Gomes. **Tecnologia, desenvolvimento social e Educação Tecnológica.** Disponível em: <http://revistas.utfpr.edu.br/pb/index.php/revedutec-ct/article/view/1011>. Acessado em: 06 de novembro de 2015.

CHRISTENSEN, Erik; CURBERA, Francisco; MEREDITH, Greg; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL) 1.1.** Disponível em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acessado em: 2 de dezembro de 2015.

DALTON, Luiz Marcilio. **Análise de Linguagens de Composição para Web Services**. Disponível em: <http://www.bibliotecavirtual.celepar.pr.gov.br/arquivos/File/MonografiaseArtigos/Dissertacao_Dalton.pdf>. Acessado em: 2 de dezembro de 2015.

DAMASCENO, Julio Cesar. **Introdução à Composição de Serviços Web.** Disponível em: <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc8.pdf>. Acessado em: 1 de abril de 2015.

DAML. **OWL-S: Semantic Markup for Web Services.** Disponível em: <http://www.daml.org/services/owl-s/1.1/overview/>. Acessado em: 2 de dezembro de 2015.

DIAZ, Gregorio; PARDO, Juan J.; CAMBRONERO, Marıa E.; VALERO, Valentın; CUARTERO, Fernando. **Verification of Web Services with Timed Automata.** Disponível em: <https://investigacion.uclm.es/documentos/fi_1268855655-ENTCS06%20-%20Verification%20of%20Web%20Services%20with%20Timed%20Automata.pdf>. Acessado em: 2 de dezembro de 2015.

DUMEZ, C.; BAKHOUYA, M.; GABER, J.; WACK M.; LORENZ, P. **Model-driven approach supporting formal verification for the web service composition protocols.** Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804513000180>. Acessado em: 2 de dezembro de 2015.

FAN, Guisheng; YUA, Huiqun; CHENC, Liqiong; LIUA, Dongmei. **Petri net based techniques for constructing reliable service composition.** Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121212003214>. Acessado em: 2 de dezembro de 2015.

FERRARA, Andrea. **Web Services: A Process Algebra Approach.** Disponível em: <http://lists.w3.org/Archives/Public/public-ws-chor/2004Oct/att-0000/ferraraPA.pdf>. Acessado em: 2 de dezembro de 2015.

FERREIRA, Aurélio Buarque de Holanda [1910-1989]. **Miniaurélio século XXI Escolar: O minidicionário da língua portuguesa.** Rio de Janeiro: Nofa Fronteira, 2001.

FRANCES, Carlos R. L. **Introdução a Redes de Petri.** Disponível em: <http://www.dca.ufrn.br/~affonso/DCA0409/pdf/redes_de_petri.pdf>. Acessado em: 2 de dezembro de 2015.

FREDLUND, Lars-Ake. Implementing WS-CDL. Disponível em: <http://babel.ls.fi.upm.es/~fred/papers/jsweb2006.pdf>. Acessado em: 2 de dezembro de 2015.

GRAIET, Mohamed; HAMEL, Lazhar; MARAOUI, Raoudha; GAALOUL, Walid; KMIMECH, Mourad; BHIRI, Mohamed T. **Towards an approach of formal verification of Web Service Composition.** Disponível em: <http://www.emeraldinsight.com/doi/abs/10.1108/17440081211222582?journalCode=ijwis>. Acessado em: 2 de dezembro de 2015.

GRUBER, Tom. **Ontology.** Disponível em: <http://tomgruber.org/writing/ontology-definition-2007.htm>. Acessado em: 2 de dezembro de 2015.

GUDGIN, Martin; HADLEY, Marc; ROGERS, Tony; YALCINALP, Umit. **Web Services Addressing 1.0 – Metadata.** Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.7231&rep=rep1&type=pdf>. Acessado em: 2 de dezembro de 2015.

HINZ, Sebastian; SCHMIDT, Karsten; STAHL, Christian. **Transforming BPEL to Petri Nets.** Disponível em: <http://www2.informatik.hu-berlin.de/top/download/publications/HinzSS2005_lncs3649.pdf>. Acessado em: 2 de dezembro de 2015.

KOVA, Melissa; BENTAHAR, Jamal; MAAMAR, Zakaria; YAHYAOUI, Hamdi. **A Formal Verification Approach of Conversations in Composite Web Services using NuSMV.** Disponível em: <http://dl.acm.org/citation.cfm?id=1659327>. Acessado em: 10 de maio de 2015.

LOBO, Adilton. **Redes de Petri, a prática em Sistemas de Manufatura.** Disponível em: <http://ftp.udesc.br/~r4al/ARTREDPE.HTM>. Acessado em: 2 de dezembro de 2015.

LOHMANN, Niels. **A Feature-Complete Petri Net Semantics for WS-BPEL 2.0 and its Compiler BPEL2oWFN.** Disponível em: <http://www2.informatik.hu-berlin.de/top/download/publications/Lohmann2007_hub_tr212.pdf>. Acessado em: 6 de novembro de 2015.

MACHADO, Julio H. A. P. **Hyper-Automaton:Hipertextos e Cursos na Web Usando Autômatos Finitos com Saída.** Disponível em: <http://www.researchgate.net/publication/228987664_Hyper_Automaton_hipertextos_e_cursos_na_web_usando_autmatos_finitos_com_sada>. Acessado em: 2 de dezembro de 2015.

MACIEL, Paulo R.M. **Introdução às redes de Petri e aplicações.** Disponível em: <http://www.researchgate.net/publication/247935030>. Acessado em: 2 de dezembro de 2015.

MICHELS, Paulo Henrique; FILETO, Renato. **Coreografia de Serviços Web.** Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_621/artigo.pdf>. Acessado em: 2 de dezembro de 2015.

MURTAGH, Dónal. **Automated Web Service Composition.** Disponível em: <https://www.scss.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-14.pdf>. Acessado em: 2 de dezembro de 2015.

NICOLA, de Rocco. **A gentle introduction to Process Algebra.** Disponível em: <https://www.pst.ifi.lmu.de/Lehre/sose-2013/formale-spezifikation-und-erifikation/intro-to-pa.pdf>. Acessado em: 2 de dezembro de 2015.

NI, Yue; FAN, Yushun. **Model transformation and formal verification for Semantic Web Services composition.** Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997810000207>. Acessado em: 2 de dezembro de 2015.

OASIS. **Web Services Business Process Execution Language Version 2.0.** Disponível em: <https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>. Acessado em: 2 de dezembro de 2015.

OUEDERNI, Meriem; SALAUN, Gwen; PIMENTEL, Ernesto. **Measuring the Compatibility of Service Interaction Protocols.** Disponível em: <https://hal.archives-ouvertes.fr/hal-00650529/document>. Acessado em: 2 de dezembro de 2015.

OUYANG, Chun; VERBEEK, Eric; AALST, Wil M.P. van der; BREUTEL, Stephan; DUMAS, Marlon; HOFSTEDE, Arthur H.M. ter. **Formal semantics and analysis of control flow in WS-BPEL.** Disponível em: <http://ac.els-cdn.com/S0167642307000500/1-s2.0-S0167642307000500-ain.pdf?_tid=1606e11 a-84c2-11e5-a9d9-00000aab0f27&acdnat=1446840663_fd3d315db32a46b46b9c 4 fc33c86e8a4>. Acessado em: 6 de novembro de 2015.

OUYANG, Chun; VERBEEK, Eric; DUMASA, Marlon; AALSTA, W.M.P. van der; HOFSTEDEA, Arthur H. M. ter; BREUTELA, Stephan. **Formal semantics and analysis of control flow in WS-BPEL.** Disponível em: <http://repository.tue.nl/641507>. Acessado em: 2 de dezembro de 2015.

PAPAZOGLOU, Michael El P.; TRAVERSO, Paolo; DUSTDAR, Schahram; LEYMANN, Frank. **Service-Oriented Computing: A Reseach Roadmap.** Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S0218843008001816>. Acessado em: 2 de dezembro de 2015.

PAPAZOGLOU, Michael; GEORGAKOPOULOS, D. **Service-Oriented Computing.** Disponível em: <http://www.gsic.uva.es/wikis/juaase/images/a/ac/WebServices-Papazoglou-CACM-2003.pdf>. Acessado em: 6 de abril de 2015.

PATHAK, Ajyotishman; BASU, Samik; HONAVAR, Vasant. **Modeling Web Service Composition using Symbolic Transition Systems.** Disponível em: <https://www.aaai.org/Papers/Workshops/2006/WS-06-01/WS06-01-006.pdf>. Acessado em: 2 de dezembro de 2015.

PU, Geguang; XIANGPENG, Zhao; SHULING, Wang, ZONGYAN, Qiu. **Towards the Semantics and Verification of BPEL4WS.** Disponível em: <http://www.sciencedirect.com/science/article/pii/S1571066106003355>. Acessado em: 2 de dezembro de 2015.

RANI, Meenu; CHAWLA, Amit K.; BATRA, Shalini. **Web Service Choreography Description Language (WS-CDL): Goals and Benefits.** Disponível em: <http://www.researchgate.net/publication/267546305_Web_Service_Choreography_

Description_Language_(WS-CDL)_Goals_and_Benefits>. Acessado em: 2 de dezembro de 2015.

RAO, Jinghai; SU, Xiaomeng. **A Survey of Automated Web Service Composition Methods.** Disponível em: <http://www.cs.cmu.edu/~jinghai/papers/survey_rao.pdf>. Acessado em: 2 de dezembro de 2015.

RECKZIEGEL, Mauricio. **Descrevendo, descobrindo e integrando Web Services – UDDI.** Disponível em: <http://imasters.com.br/artigo/4474/web-services/descrevendo-descobrindo-e-integrando-web-services-uddi/>. Acessado em: 2 de dezembro de 2015.

RODRIGUES, Douglas; ESTRELLA, Júlio C.; BRANCO, Kalinka R.L.J.C. **Segurança Computacional no Desenvolvimento de Web Services.** Disponível em: <http://www2.fc.unesp.br/erispo2010/MC-02.pdf>. Acessado em: 2 de dezembro de 2015.

SALAUN, Gwen; FERRARA, Andrea; CHIRICHIELLO, Antonella. **Negotiation among web services using LOTOS/CADP.** Disponível em: <http://www.dis.uniroma1.it/chiri/research/papers/TechReports/2004/TechnicalReport_1304.pdf>. Acessado em: 2 de dezembro de 2015.

SALAUN, Gwen; BORDEAUX, Lucas; SCHAERF, Marco. **Describing and Reasoning on Web Services Using Process Algebra.** Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1314722&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1314722>. Acessado em: 2 de dezembro de 2015.

SANT'ANNA, Mauro. **SOAP e WebServices**. Disponível em: <http://www.linhadecodigo.com.br/artigo/38/soap-e-webservices.aspx>. Acessado em: 2 de dezembro de 2015.

SENAC. DN. **Qualidade em prestação de serviços.** Rio de Janeiro: Senac Nacional, 2011.

SILVA, Alexandro Nascimento da. **Composição Automática de Web Services: Avaliando os Planejadores JSHOP2 e JESS.** Disponível em: <http://homes.dcc.ufba.br/~dclaro/download/Monografia%20-%20Final%20II%20-%20AlexandroNascimento.pdf>. Acessado em: 2 de dezembro de 2015.

SILVA, Leo Moreira. **Aplicando Composição e Orquestração de Serviços na Organização de Sistemas.** Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_838/Aplicando%20Composi%E7%E3o%20e%20Orquestra%E7%E3o%20de%20Servi%E7os%20na%20Organiza%E7%E3o%20de%20Sistemas.pdf>. Acessado em: 2 de dezembro de 2015.

SOARES, Fabiana Govea. **Arquitetura orientada a Serviços**. Disponível em: <http://www.escavador.com/pessoas/3989159>. Acessado em: 2 de dezembro de 2015.

TAKAHASHI, Tadao. **Sociedade da informação no Brasil: livro verde**. Brasília: Ministério da Ciência e Tecnologia, 2000.

TANENBAUM, Andrew S**. Computer Networks**. Tradução: Vandenberg D. de Souza Amsterdan: Editora Campus, 2011.

TAN, Wei; ZHOU, Mengchu. **A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language.** Disponível em: http://www.researchgate.net/profile/MengchuZhou/publication/224307265APetriNetBaseMethodforCompatibilityAnalysisandCompositionofWebServicesinBusinessProcessExecutionLanguage%20/links/0fcfd508ab0aa3fa78000000.pdf>. Acessado em: 6 de novembro de 2015.

TEIXEIRA, Nara Sueina. **Análise da Compatibilidade de Componentes Especificados em UML.** Disponível em: <https://repositorio.ufsc.br/handle/123456789/99418>. Acessado em: 2 de dezembro de 2015.

TODICA, Valeriu; VAIDA, Mircea-Florin; CREMENE, Marcel. **Formal Verification in Web Services Composition.** Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6237702&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6237702>. Acessado em: 2 de dezembro de 2015.

VERBEEK, H.M.W; AALST, W.M.P van der. **Analysing BPEL processes using Petri Nets.** Disponível em: <http://web.student.tuwien.ac.at/~e0052074/WST/p263.pdf>. Acessado em: 2 de dezembro de 2015.

VIDAL, Juan C.; LAMA, Manuel; BUGARIN, Alberto. **Toward the use of Petri nets for the formalization of OWL-S choreographies**. Disponível em: <http://link.springer.com/article/10.1007%2Fs10115-011-0451-z>. Acessado em: 2 de dezembro de 2015.

WOHED, Petia; AALST, W. M. P. van der; DUMAS, Marlon; HOFSTEDE, Arthur H. M. ter. **Analysis of Web Services Composition Languages: The Case of BPEL4WS.** Disponível em: <http://link.springer.com/chapter/10.1007/978-3-540-39648-2_18#page-1>. Acessado em: 2 de dezembro de 2015.

WOMBACHER, Andreas; FANKHAUSER, Peter; NEUHOLD, Erich. **Transforming BPEL into annotated Deterministic Finite State Automata for Service Discovery.** Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1314753&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F9185%2F29136%2F01314753.pdf%3Farnumber%3D1314753>; Acessado em: 2 de dezembro de 2015.

W3C**. Web Services Architecture**. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acessado em: 6 de novembro de 2015.

APENDICES

Apêndice A – Lista de Artigos – Rede de Petri

ANO	TITULO DO ARTIGO	AUTORES
2013	Model-driven approach supporting formal verification for the web service composition protocols	DUMEZ, C.; BAKHOUYA, M.; GABER, J.; WACK M.; LORENZ, P.
2010	A model checker for WS-CDL	WANGA, Hongbing; KANGA, Zuling; ZHOUA, Ning; LI, Li
2011	Towards an approach of formal verification of Web Service Composition	GRAIET, Mohamed; HAMEL, Lazhar; MARAOUI, Raoudha; GAALOUL, Walid; KMIMECH, Mourad; BHIRI, Mohamed T.
2010	Towards Formalizing Web Service Composition in Maude's Strategy Language	MEROUANI, Hamza; MOKHATI, Farid; SERIDI-BOUCHELAGHEM, Hassina
2008	A Calculus for Generation, Verification and Refinement of BPEL Specifications	ABOUZAID, Faisal; MULLINS, John
2009	Implementing Rigorous Web Services with Process Algebra	RODRIGUES, Mauricio C. R.; MALKOWSKI, Simon; FERREIRA, Joao E.
2004	Web Services: A Process Algebra Approach	FERRARA, Andrea
2004	Towards a Formal Foundation to Orchestration Languages	VIROLI, Mirko
2006	A Semantical Framework for the Orchestration and Choreography of Web Services	PAHL, Claus; ZHU, Yaoling
2012	Formal Verification in Web Services Composition	TODICA, Valeriu; VAIDA, Mircea-Florin; CREMENE, Marcel
2009	Ensuring Coordination of Multi-business Interactions	YUAN, Min; HUANG, Zhiqiu; HU, Jun; LI, Xiang; ZHU, Yi
2004	Analysis of Interacting BPEL Web Services	FU, Xiang; BULTAN, Tevfik; SU, Jianwen
2011	Towards an approach of formal verification of mediation protocol based on Web services	GRAIET, Mohamed; MARAOUI, Raoudha; KMIMECH, Mourad; BHITI, Mohamed T.; GAALOUL, Walid
2006	Formalizing the specification and execution	CICEKLI, Nihan K.; CICEKLI, Ilyas
2010	Um Modelo Formal para Composição de Serviços Web usando a Linguagem SAWSDL	MENEGAZZO, Cinara T.; VIEIRA, Késia V.
2007	A Formal Framework for Web Services Coordination	GUIDI, Claudio; LUCCHI, Roberto; MAZZARA, Manuel
2004	From BPEL to SRML: A Formal Transformational Approach	BOCCHI, Laura; HONG, Yi; LOPES, Antónia; FIADEIRO, José L.

2004	Describing and Reasoning on Web Services Using Process Algebra	SALAUN, Gwen; BORDEAUX, Lucas; SCHAERF, Marco
2006	Pattern Based Property Specification and Verification for Service Composition	YU, Jian; MANH, Tan P.; HAN, Jun; JUN, Yan; HAN, Yanbo; WANG, Jianwu
2011	Using formal methods to develop WS-BPEL applicatinos	LAPADULA, Alessandro; PUGLIESE, Rosario; TIEZZI, Francesco
2004	Negotiation among web services using LOTOS/CADP	SALAUN, Gwen; FERRARA, Andrea; CHIRICHIELLO, Antonella
2013	QoS Composition and Analysis in Reconfigurable Web Services Choreographies	KATTEPUR, Ajay; GEORGANTAS, Nikolaos; ISSARNY, Valérie
2006	Unified Service-Composition for BPEL	SHAO, Bing
2008	Validation of Web Service Compositions	BARESI, Luciano; BIANCULLI, Domenico; GHEZZI, Carlo; GUINEA, Sam; SPOLETINI, Paola
2007	A Calculus for Orchestration of Web Services	LAPADULA, Alessandro; PUGLIESE, Rosario; TIEZZI, Francesco
2008	Automated Composition of Web Services: the ASTRO Approach	MARCONI, Annapaola; PISTORE, Marco; TRAVERSO, Paolo
2012	The Tale of SOLOIST: a Specification Language for Service Compositions Interactions	BIANCULLI, Domenico; GHEZZI, Carlo; PIETRO, Pierluigi S.
2013	VerChor: A Framework for Verifying Choreographies	GUDEMANN, Matthias; POIZAT, Pascal; SALAUN, Gwen; DUMONT, Alexandre
2010	Declarative Specification and Verification of Service Choreographies	MONTALI, Marco; PESIC, Maja; AALST, Wil M. P. V. der; CHESANI, Federico; MELLO, Paola; STORARI, Sergio
2005	Formalizing WSBPEL Business Processes Using Process Algebra	CAMARA, Javier; CANAL, Carlos; CUBO, Javier; VALLECILLO, Antonio
2004	Compatibility Verification for Web Service Choreography	FOSTER, Howard; UCHITEL, Sebastian; MAGEE, Magee; KRAMER, Jeff
2010	Analyzing Behavioral Substitution of Web Services based on λ-calculus	LI, Kuang; XIA, Yingjie; DENG, Shuiguang; WU, Jian
2006	A Formal Model for BPEL4 WS Description of Web Service Composition	GU, Xiwu; LU, Zhengding
2007	Efficient Analysis of BPEL 2.0 Processes using π-Calculus	WEIDLICH, Matthias; DECKER, Gero; WESKE, Mathias
2004	Formalizing Web Service Choreographies	BROGI, Antonio; CANAL, Carlos; PIMENTEL, Ernesto; VALLECILLO, Antonio
2009	A Chu Spaces Semantics of Control Flow in BPEL	DU, Xutao; XING, Chunxiao; ZHOU, Lizhu
2009	Computing compatibility in dynamic service composition	WU, Zhaohui; DENG, Shuiguang; LI, Ying, WU, Jian

2010	Formal Specification and Verification of Service Composition using LOTOS	DUMEZ, Christophe; BAKHOUYA, Mohamed; GABER, Jaafar; WACK, Maxime
2004	Towards a formal framework for Choreography	BUSI, Nadia; GORRIERI, Roberto; GUIDI, Claudio; LUCCHI, Roberto; ZAVATTARO, Gianluigi
2005	A Formal Model for Web Service Choreography Description Language (WS-CDL)	YANG, Hongli; ZHAO, Xiangpeng; QIU, Zongyan; PU, Geguang; WANG, Shuling
2010	Formal Analysis of Behavioural Equivalence for Trustworthy and Composite Web Services	ZHAO, Yongwang; HU, Chunyang; LIU, Min; MA, Dianfu
2010	A Formal Model for Service Choreography with Exception Handling and Finalization	ZHAO, Yongxin; WANG, Zheng; PU, Geguang; ZHU, Huibiao
2004	Semantics of BPEL4WS-like Fault and Compensation Handling	QIU, Zongyan; WANG, Shuling; PU, Geguang; ZHAO, Xiangpeng
2010	Model checking web services choreography in process analysis toolkit	XU, Dong; LEI, Zhou; LI, Wei-min; ZHANG, Bo-feng
2010	Verifcation of Timed BPEL 2.0 Models	FARES, Elie; BODEVEIX, Jean-Paul; FILALI, Mamoun
2008	A Formal Analysis of Behavioral Equivalence for Web Services	KUANG, Li
2006	Modeling and Verifying Web Services Choreography using Process Algebra	LI, Jing; HE, Jifeng; ZHU, Huibiao; PU, Geguang
2005	A pi-calculus based semantics for WS-BPEL	LUCCHI, Roberto; Manuel Mazzara
2003	Model-based Verification of Web Service Compositions	FOSTER, Howard; UCHITEL, Sebastian; MAGEE, Jeff; KRAMER, Jeff
2011	A formal and visual modeling approach to choreography based web services composition and conformance verification	YEUNG, W.L.
2006	Formalizing WSBPEL Business Processes Using Process Algebra	CAMARA, Javier; CANAL, Carlos; CUBOA, Javier; VALLECILLO, Antonio
2005	LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography	FOSTER, Howard; UCHITEL, Sebastian; MAGEE, Jeff; KRAMER, Jeff
2009	Modeling the WorkUnit of WS-CDL Based on Process Algebra	LI, Shenghong
2008	Modeling the Patterns of WS-CDL Interactions Based on Process Algebra	LI, Shenghong; MIAO, Huaikou
2012	Realizability of Choreographies Using Process Algebra Encodings	SALAUN, Gwen; BULTAN, Tevfik; ROOHI, Nima
2012	Using Formal Methods to develop WS-BPEL applications	LAPADULA, Alessandro; PUGLIESE, Rosario; TIEZZI, Francesco

2012	Research on Formal Modeling and Verification of BPEL based web service	ZHAO, Huiqun; WANG, Wenwen; SUN, Jing; WEI, Ying
2009	A barred operational semantics for a subset of WS-CDL with time restrictions	VALERO, Valentín; DIAZ, Gregorio; CAMBRONERO, María E.; MACIA, Hermenegilda
2009	Model Checking web services orchestrations using BP-Calculus	ABOUZAID, Faisal ; MULLINS, John

]

Apêndice B – Lista de Artigos – Álgebra de Processos

ANO	TITULO DO ARTIGO	AUTORES
2012	A high-level Petri net based approach for modeling and composition of web services	CHEMAA, Sofiane; BACHTARZI, Faycal; CHAOUI, Allaoua
2007	Petri Net Semantics for OWL-S Service Choreography	VIDAL, Juan C.; LAMA, Manuel; BUGARIN, Alberto
2013	Petri net based techniques for constructing reliable service composition	FANA, Guisheng; YUA, Huiqun; CHENC, Liqiong; LIUA, Dongmei
2010	Verifying Web Services of OWL-S with Petri Net	YU, Bo; LI, Duluo
2010	Model transformation and formal verification for Semantic Web Services composition	NI, Yue; FAN, Yushun
2010	Web service selection for transactional composition	CARDINALE, Yudith; HADDADB, Joyce E; MANOUVRIERB, Maude; RUKOZB, Marta
2011	Toward the use of Petri nets for the formalization of OWL-S choreographies	VIDAL, Juan C.; LAMA, Manuel; BUGARIN, Alberto
2006	Specification and Verification of Views over Composite Web Services Using High Level Petri-Nets	BOUKADI, Khouloud; GHEDIRA, Chirine; MAAMAR, Zakaria; BOUCHENEB, Hanifa
2005	Modeling and Model Checking Web Services	SCHLINGLOFF, Holger; MARTENS, Axel; SCHMIDT, Karsten
2004	A CP-nets-based Design and Verification Framework for Web Services Composition	YI, Xiaochuan; KOCHUT, Krys J.
2008	Conformance Checking of Service Behavior	AALST, W. M. P. Van der; DUMAS, Marlon; OUYANG, Chun; ROZINAT, Anne; VERBEEK, Eric
2004	Triana: A Graphical Web Service Composition and Execution Toolkit	MAJITHIA, Shalil; SHIELDS, Matthew; TAYLOR, Ian; WANG, Ian
2005	WSDL-Based Automatic Test Case Generation for Web Services Testing	BAI, Xiaoying; DONG, Wenli; TSAI, Wei-Tek; CHEN, Yinong
2006	Analysing BPEL processes using Petri Nets	VERBEEK, H.M.W; AALST, W.M.P van der
2005	Transforming BPEL to Petri Nets	HINZ, Sebastian; SCHMIDT, Karsten; STAHL, Christian
2007	Modeling and Verification of Web Services Composition based on CPN	KANG, Hui; YANG, Xiuli; YUAN, Sinmiao

2005	Transformation BPEL to CP-Nets for Verifying Web Services Composition	YANG,YanPing; TAN, QingPing; YU, JinShan, LIU, Feng
2009	A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language	TAN, Wei; FAN, Yushun; ZHOU, MengChu
2011	A Petri Net Approach to Analyzing Behavioral Compatibility and Similarity of Web Services	LI, Xitong; FAN, Yushun; SHENG, Quan Z.; MAAMAR, Zakaria ZHU, Hongwei
2010	A Petri Net Approach to Analysis and Composition of Web Services	XIONG, PengCheng, FAN, YuShun; ZHOU, MengChu Zhou
2012	A Petri-net and QoS Based Model for Automatic Web Service Composition	LI, Bin; XU, Yan; WU, Jun; ZHU, Junwu
2003	On Compatibility of Web Services	MARTENS, Axel
2005	Simulation and Equivalence between BPEL Process Models	MARTENS, Axel
2006	Analyzing Compatibility of BPEL Processes	MARTENS, Axel; MOSER, Simon; GERHARDT, Achim; FUNK, Karoline
2006	Formal Semantics and Analysis of Control Flow in WS-BPEL	OUYANG, Chun; VERBEEK, Eric; AALST, W. M.P. van der; BREUTEL, Stephen; DUMAS, Marlon; HOFSTEDE, Arthur H.M. ter
2003	WS-Net: A Petri-net Based Specification Model for Web Services	ZHANG, Jia; CHANG, Carl K.; CHUNG, Jen-Yao; KIM, Seong W.
2010	Composition of Petri Nets Models in Service-oriented Industrial Automation	MENDES, J. M.; LEITAO, Paulo; RESTIVO, Francisco; COLOMBO, Armando W.
2007	From OWL-S descriptions to Petri nets	BROGI, Antonio; CORFINI, Sara; IARDELLA, Stefano
2005	Consistency between Executable and Abstract Processes	MARTENS, Axel
2012	A Petri Net Semantics for Web Service in Choreography Description Language	LI, Jieying; ZHANG, Huijuan; DUN, Haiqiang
2010	BPEL2DENEB: Translation of BPEL Processes to Executable High-level Petri Nets	FABRAE, Javier; ALVAREZ, Pedro
2003	A Petri Net-based Model for Web Service Composition	HAMADI, Rachid; BENATALLAH, Boualem
2010	A pattern-based approach to protocol mediation for web services composition	LI, Xitong; FAN, Yushun; MADNICK, Stuart; SHENG, Quan Z.

2010	A Petri Nets based functional validation for services composition	YOO, Taejong; JEONG, Buhwan; CHO, Hyunbo
2009	A Petri net approach for the design and analysis of Web Services Choreographies	VALERO, Valentín; CAMBRONERO, M. Emilia; DIAZ, Gregorio; MACIA, Hermenegilda
2008	A formal modeling platform for composing web services	CHI, Yu-Liang; LEE, Hsun-Ming
2007	RESTFul Petri Net Execution	DECKER, Gero; LUDERS, Alexander; OVERDICK, Hagen; SCHLICHTING, Kai; WESKE, Mathias
2012	Transforming Web Services Choreographies with priorities and time constraints into prioritized-time colored petri nets	VALERO, Valentín; MACIA, Hermenegilda; PARDO, Juan J.; CAMBRONERO, María E.; DIAZ, Gregorio

Apêndice C – Lista de Artigos – Autômatos

ANO	TITULO DO ARTIGO	AUTORES
2004	Transforming BPEL into annotated Deterministic Finite State Automata for Service Discovery	WOMBACHER, Andreas; FANKHAUSER, Peter; NEUHOLD, Erich
2006	Designing a BPEL Orchestration Engine Based on ReSpecT Tuple Centres	CABANO, Michele; DENTI, Enrico; RICCI, Alessandro; VIROLI, Mirko
2009	A Formal Verification Approach of Conversations in Composite Web Services using NuSMV	KOVA, Melissa; BENTAHAR, Jamal; MAAMAR, Zakaria; YAHYAOUI, Hamdi
2012	A novel architecture for Web service composition	KARUNAMURTHY, Rajesh; KHENDEK, Ferhat; GLITHO, Roch H.
2011	Measuring the Compatibility of Service Interaction Protocols	OUEDERNI, Meriem; SALAUN, Gwen; PIMENTEL, Ernesto
2006	Towards the Semantics and Verification of BPEL4WS	PU, Geguang; XIANGPENG, Zhao; SHULING, Wang, ZONGYAN, Qiu
2006	Verification of Web Services with Timed Automata	DIAZ, Gregorio; PARDO, Juan J.; CAMBRONERO, Marıa E.; VALERO, Valentın; CUARTERO, Fernando
2009	Algorithms and Complexity of Automata Synthesis by Asynhcronous Orchestration With Applications to Web Services Composition	BALBIANI, Philippe; CHEIKH, Fahima; FEUILLADE, Guillaume
2012	A centralized and a decentralized method to automatically derive choreography-conforming web service systems	RODRIGUEZ, Ismael, DIAZ, Gregorio; RABANAL, Pablo; MATEO, Jose A.
2004	Behaviour Analysis And Verification of Web Service Compositions	FOSTER, Howard
2006	Modeling Web Service Composition using Symbolic Transition Systems	PATHAK, Ajyotishman; BASU, Samik; HONAVAR, Vasant
2006	A formal model for semantic Web service composition	LECUE, Freddy; LEGER, Alain
2012	Model-Checking Web Services Business Activity Protocols	MARQUES, Abinoam P. Jr.; RAVN, Anders P.; SRBA, Jiri; VIGHIO, Saleem
2003	Semi-automatic Composition of Web Services using Semantic Descriptions	SIRIN, Evren; HENDLER, James; PARSIA, Bijan

2003	WST: A tool for the translation od WS-CDL into timed automata	CAMBRONERO, Maria E.; DIAZ, Gregorio; MARTINEZ, Enrique; VALERO, Valentın; TOBARRA, Llanos
2006	Checking Compatibility and Replaceability in Web Services Business Protocols with Access Control	ELABD, Emad; COQUERY, Emmanuel; HACID, Mohand-Said
2004	Analysis of Interacting BPEL Web Services	FU, Xiang; BULTAN, Tevfik; SU, Jianwen
2005	Automatic Translation of WS-CDL Choreographies to Timed Automata	DIAZ, Gregorio; PARDO, Juan J.; CAMBRONERO, María E.; VALERO, Valentín; CUARTERO, Fernando
2009	Design and Verification of Web Services Compositions	MARTINEZ, Enrique; CAMBRONERO, María E.; DIAZ, Gregorio; VALERO, Valentin
2006	Timed Modelling and Analysis in Web Service Compositions	KAZHAMIAKIN, Raman; PANDYA, Paritosh; PISTORE, Marco
2006	Verification of Computation Orchestration Via Timed Automata	DONG, Jin S.; LIU, Yang; SU, Jun; ZHANG, Xian
2006	Verifying correctness of Web services choreography	TAREK, Melliti; BOUTROUS-SAAD, Celine; RAMPACEK, Sylvain
2010	A Verification Method for Web Service Composition based on Discrete Event System Modeling and Simulation	LIM, Seong Y.; BAIK, Jongmoon; CHOI, Ho-Jin; LEE, Dan H.
2004	AUTOMATIC SERVICE COMPOSITION BASED ON BEHAVIORAL DESCRIPTIONS	BERARDI, Daniela; CALVANESE, Diego; GIACOMO, Giuseppe de; LENZERINI, Maurizio; MECELLA, Massimo
2010	Separating Operational and Control Behaviors	SHENG, Quan Z.; MAAMAR, Sakaria; YAHYAOUNI, Hamdi; BENTAHAR, Jamal; BOUKADI, Khouloud
2003	FINITE STATE AUTOMATA AS CONCEPTUAL MODEL FOR E-SERVICES	BERARDI, Daniela; ROSA, Fabio D.; SANTIS, Luca D.; MECELLA, Massimo
2004	Formal Verification of BPEL4WS Business Collaborations	FISTEUS, Jesús A.; FERNANDEZ, Luis S.; KLOOS, Carlos
2009	Model Checking Service Component Composition By SPIN	DING, Zuohua; JIANG, Mingyue
2006	Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN	GARCIA-FANJUL, José; TUYA, Javier; RIVA, Claudio de la
2009	Verification Web Services Composition Based on OWL-S	XIA, Hong; LI, Zengzhi
2006	Model-Checking Behavioral Specification of BPEL Applications	NAKAJIMA, Shin
2006	Towards Model-based Verification of BPEL with Model Checking	CAO, Honghua; YING, Shi; DU, Dehui
2005	Automated Synthesis of Composite BPEL4WS Web Services	PISTORE, M.. TRAVERSO, P.; BERTOLI, P.; MARCONI, A.

2008	Model-checking of Web Services Choreography	HONGLI, Yang; XIANGPENG, Zhao; CAO, Cai; ZONGYAN, Qiu
2004	A Parametric Communication Model for the Verification of BPEL4WS Compositions	KAZHAMIAKIN, Raman; PISTORE, Marco
2009	Formal Modeling and Conformance Validation for WS-CDL using Reo and CASM	TASHAROFIA, Samira; SIRJANIA, Marjan
2010	Dynamic Web Services Composition Based on Linear Temporal Logic	HAO, Shengang; ZHANG, Li
2008	Specifying and Monitoring Temporal Properties in Web services Compositions	KALLEL, Slim; CHARFI, Anis; DINKELAKER, Tom; MEZINI, Mira; JMAIEL, Mohamed
2008	A Tool for Choreography Analysis Using Collaboration Diagrams	BULTAN, Tevfik; FERGUSON, Chris; FU, Xiang
2010	Modeling and Reasoning about Service Behaviors and their Compositions	CAUSEVIC, Aida; SECELEANU, Cristina; PETTERSSON, Paul
2008	Simulation and Formal Analysis of Workflow Models	KOVACS, Mata; GONCZY, Laszlo
2010	An Integrated Workbench for Model-Based Engineering of Service Compositions	FOSTER, Howard; UCHITEL, Sebastian;MAGEE, Jeff; KRAMER, Jeff
2011	Validation and verification ofWeb services choreographies by using timed automata	CAMBRONERO, Maria E.; DIAZ, Gregorio; VALERO, Valentín; MARTINEZ, Enrique
2006	Verification of Web Services with Timed Automata	DIAZ, Gregorio; PARDO, Juan J.; CAMBRONERO, Marıa E.; VALERO, Valentın; CUARTERO, Fernando
2013	Symbolic model checking composite Web services using operational and control behaviors	BENTAHAR, Jamal; YAHYAOUI, Hamdi; KOVA, Melissa; MAAMAR, Zakaria
2008	Modeling Web Service Interactions Using the Coordination Language Reo	TASHAROFI, Samira; VAKILIAN, Mohsen; MOGHADDAM, Roshanak Z.; SIRJANI, Marjan
2006	Formal Verification of Web Service Behavioural Conformance through Testing	DRANIDIS, Dimitris; KOURTESIS, Dimitris; RAMOLLARI, Ervin