Alexis Armin Huf

An Architecture for Composition of Services with Heterogeneous Interaction Models

Florianópolis
2018

Alexis Armin Huf

AN ARCHITECTURE FOR COMPOSITION OF SERVICES WITH HETEROGENEOUS INTERACTION MODELS

Dissertação de Mestrado submetida ao Programa de Pós Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis
2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Huf, Alexis Armin
An architecture for composition of services with heterogeneous interaction models / Alexis Armin Huf ; orientador, Frank Augusto Siqueira, 2018.
250 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Ciência da Computação, Florianópolis, 2018.

Inclui referências.

1. Ciência da Computação. 2. Serviços Web. 3. Arquitetura de Software. 4. Composição de Serviços. I. Siqueira, Frank Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Alexis Armin Huf

AN ARCHITECTURE FOR COMPOSITION OF SERVICES WITH HETEROGENEOUS INTERACTION MODELS

Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 20 de fevereiro de 2018.

Prof. José Luís Almada Güntzel, Dr.
Coordenador do Programa

Banca Examinadora:

Prof. Frank Augusto Siqueira, Dr.
Universidade Federal de Santa Catarina
Orientador

Prof. Vítor Estêvão Silva Souza, Dr.
Universidade Federal do Espírito Santo (Videoconferência)

Prof. Gustavo Medeiros de Araújo, Dr.
Universidade Federal de Santa Catarina

Prof. Elder Rizzon Santos, Dr.
Universidade Federal de Santa Catarina

Prof. Roberto Willrich, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado
aos meus pais e professores

AGRADECIMENTOS

Agradeço primeiramente à minha família, em especial meus pais, por terem ensinado o valor do conhecimento desde cedo.

Aos mestres, pelos valiosos ensinamentos e dedicação. Sem o apoio e dedicação dessas pessoas, certamente não teria concluído as jornadas mais importantes da minha vida. Em especial, agradeço ao professor Frank Siqueira pelas ideias, revisões detalhadas e disponibilidade. Aos membros da banca sou grato pelos comentários e contribuições que foram muito valiosas na confecção da versão final desta dissertação e que terão impacto nos meus próximos trabalhos. Em particular, Prof. Roberto Willrich e Prof. Elder Rizzon Santos já haviam participado do Exame de Qualificação e naquele momento já forneceram críticas que contribuíram para o desenvolvimento desta dissertação. Ainda antes, em uma disciplina, foram menções do Prof. Elder que me levaram ao D*-Lite. Tive também a oportunidade de compartilhar breves discussões com Vítor Estêvão Silva Souza no *Workshop* de Teses e Dissertações do WebMedia que motivaram profundas melhorias na fundamentação teórica desta dissertação.

Aos muitos amigos que fiz no LaPeSD ocidental que contribuíram de várias formas para esta dissertação. O embrião do tema desta dissertação surgiu de uma conversa com Ivan Luiz Salvadori sobre composição (ou não) de serviços RESTful. Bruno César Neves de Oliveira colaborou em vários artigos e ajudou a formular melhores perguntas. Eduardo Camilo Inácio, Alyson Deives Pereira, Eliza Gomes e Pedro Penna forneceram críticas em vários seminários e me ajudaram a sobreviver a fase das disciplinas.

À coordenação do programa, pelo apoio administrativo e a sua dedicação para o sucesso do programa e dos alunos.

Por fim, agradeço à CAPES pelo apoio financeiro durante o mestrado.

Premature optimization is the root of all
evil (or at least most of it) in programming.
(KNUTH, 1974)

RESUMO

Serviços Web atuais são altamente heterogêneos não apenas em termos de formato de dados, mas também em relação à maneira como ocorre a interação com o serviço. Apesar da existência de heterogeneidade, a composição desses serviços se torna necessária para realização de atividades complexas, que não são atendidas por um único serviço. Existem na literatura propostas de algoritmos de composição para serviços heterogêneos baseados em descrições semânticas da funcionalidade dos serviços. No entanto, essas propostas não suportam um dos principais modelos de interação existentes ou apresentam limitações no suporte a algum destes modelos. Esta dissertação apresenta uma arquitetura de software e duas técnicas, denominadas *forking* e adaptação, para composição automática de serviços heterogêneos. Diferentemente de propostas existentes, ao combinar uma descrição comum, um middleware e um algoritmo de composição, todas as restrições de interação impostas por serviços REST e por serviços orientados a eventos são respeitadas. Foram realizados experimentos comparando um protótipo da arquitetura com uma abordagem de composição de serviços SOAP e com a única abordagem de composição RESTful identificada na literatura que efetivamente suporta a restrição HATEOAS. No primeiro experimento, composto por oito cenários, o protótipo apresentou média do tempo de composição menor em sete dos oito cenários sendo mais lento apenas no cenário menos complexo. O segundo experimento avalia a escalabilidade em três cenários. Dentro de cada cenário, o aumento da complexidade dos problemas teve um efeito menor sobre o protótipo do que na implementação do algoritmo de composição RESTful.

Palavras-chave: Serviços Web, Composição de Serviços, Planejamento Automático, Arquitetura de Software, Hipermídia, Eventos, Publish/Subscribe.

RESUMO ESTENDIDO

INTRODUÇÃO

Serviços, no contexto de sistemas distribuídos, são elementos computacionais que realizam funções (PAPAZOGLOU, 2003). Uma das princípais atividades viabilizadas pelo uso de serviços é o projeto de novos serviços que combinam dados e funcionalidades de serviços existentes. Embora os primeiros serviços a serem chamados de Serviços Web (TOM et al., 2014) sejam aqueles que utilizam o protocolo SOAP (e que neste trabalho são denominados serviços SOAP), há atualmente uma grande quantidade de serviços que utilizam diretamente o protocolo HTTP, denominados Web APIs. As diferenças entre os dois grupos não se limitam ao protocolo SOAP. Dentro do segundo grupo, o estilo arquitetural REST (FIELDING; TAYLOR, 2002) possui popularidade considerável, apesar de não ser aplicado integralmente por todos os serviços (MALESHKOVA; PEDRINACI; DOMINGUE, 2010). Um terceiro tipo é composto por serviços que notificam eventos a clientes interessados, podendo esses ser implementados como serviços RESTful, Web APIs ou serviços SOAP. Cada um dos tipos de serviço mencionados impõem um conjunto de restrições à interação com os serviços daquele tipo, que esta dissertação denomina modelo de interação básico. A dissertação toma como hipótese a afirmação de que é possível, por meio de algoritmos, projetar e executar serviços compostos a partir de serviços com modelos de interação heterogêneos (SOAP, RESTful, incluindo variantes degeneradas que não respeitam todas as restrições REST, e serviços orientados a eventos) respeitando cada um desses modelos.

Objetivo: Definir uma arquitetura de software para composição automática de serviços com modelos de interação heterogêneos. Especificamente, (1) devem ser considerados os modelos de interação de serviços SOAP, RESTful e orientados a eventos; (2) a performance deve ser similar a abordagens estado-da-arte para composição homogênea.

Contribuições

- Uma categorização das heterogeneidades observadas no contexto de composição de serviços;
- Um algoritmo que em conjunto com a arquitetura possibilita a composição de serviços com modelos de interação heterogêneos.

Método:

1. Conduzir uma Revisão Sistemática da Literatura (KITCHENHAM; CHARTERS, 2007) com escopo em composição de serviços heterogêneos;
2. Categorizar heterogeneidades entre modelos de interação básicos;
3. Levantar na literatura algoritmos de composição eficientes e algoritmos que suportam serviços RESTful ou orientados a eventos;
4. Definir uma arquitetura para composição de serviços heterogêneos;
5. Obter algoritmos para composição de serviços heterogêneos;
6. Implementar um protótipo;
7. Avaliar o suporte a heterogeneidades;
8. Avaliar a expressividade dos serviços compostos pela arquitetura e a aplicabilidade prática da arquitetura através de um cenário;
9. Avaliar, através de experimentos controlados, a viabilidade e a performance da arquitetura.

FUNDAMENTAÇÃO TEÓRICA

Para sustentar etapas posteriores do trabalho, as heterogeneidades entre os modelos de interação básicos são enumeradas utilizando um modelo de protocolo de serviço para a categorização quanto à origem dessas heterogeneidades. Esse modelo de protocolo busca superar limitações de modelos anteriores (STRANG; LINNHOF-POPIEN, 2003) para que possa ser aplicado aos tipos de serviço considerados nessa dissertação. O modelo de protocolo define um protocolo como os sequenciamentos válidos de interações, baseado em um paradigma de comunicação (EUGSTER et al., 2003). Cada interação, consistindo de uma ação e um *payload* de dados, é envelopada em outro protocolo. As heterogeneidades mais problemáticas com relação a abordagens existentes para composição heterogênea têm origem no paradigma de comunicação, na descrição do protocolo e nas ações (métodos HTTP).

TRABALHOS RELACIONADOS

Uma Revisão Sistemática da Literatura foi realizada para sumarizar abordagens existentes para composição de serviços heterogêneos. Foram identificadas seis categorias de abordagens: Descrição comum, *Proxies*, *Middleware*, Linguagens de processo, Processadores de eventos e Implementação Direta. Nenhum desses métodos, com exceção da implementação direta, é capaz de suportar os três modelos de interação básicos (ainda assim teoricamente, ou seja, com base em casos relatados na literatura). Combinações entre os demais métodos também não são suficientes para suporte a todos os modelos de interação, especificamente pela ausência de suporte à restrição *Hypermedia as the Engine of Application State* (HATEOAS), parte do estilo arquitetural REST. Sob essa restrição, um cliente deve determinar sua próxima ação a partir de controles hipermídia (isto é, *links*, *forms* e *URI templates*) ofertados pelo serviço.

No escopo mais amplo de composição de serviços, a maioria das abordagens ignora aspectos de serviços RESTful ou orientados a eventos, sendo aplicável apenas a serviços SOAP. Além das abordagens incluídas na revisão sistemática, podem ser mencionadas abordagens para *Complex Event Processing* (CUGOLA; MARGARA, 2012). Para serviços RESTful, há várias abordagens, no entanto apenas o algoritmo Pragmatic Proof (VERBORGH et al., 2016) efetivamente suporta a restrição HATEOAS. Em se tratando de serviços SOAP, foco da literatura de composição automática, há algoritimos eficientes, podendo ser destacado o ComposIT (RODRIGUEZ-MIER et al., 2012).

PROPOSTA

A arquitetura proposta, chamada Unserved, combina os métodos de descrição comum e Middleware com um algoritmo de composição capaz de suportar completamente as heterogeneidades não suportadas pelos dois métodos. A descrição comum é realizada através de uma ontologia que modela a troca de mensagens entre clientes e o serviço. A arquitetura contempla a tradução de descrições existentes para a descrição comum; a indexação das descrições de serviços; o planejamento de serviços compostos a partir de objetivos do usuário; e a execução dos serviços compostos.

O algoritmo de composição obtido tem como base o algoritmo ComposIT (RODRIGUEZ-MIER et al., 2012). A técnica de *forking* permite que a execução de um serviço composto continue independentemente para cada evento recebido. A técnica de adaptação permite que controles hipermídia sejam considerados durante a execução, alterando o plano do serviço composto conforme necessário. A expressividade do algoritmo ComposIT é estendida adicionando suporte a pré- e pós-condições, descritas com consultas SPARQL (SEABORNE; HARRIS, 2013), utilizando a sintaxe SPIN (KNUBLAUCH; HENDLER; IDEHEN, 2011).

RESULTADOS

A proposta é avaliada usando quatro dos método de avaliação listados em Hevner et al. (2004): análise arquitetural, análise estática, cenário e experimentos controlados. Os três primeiros métodos avaliam, respectivamente, o suporte a heterogeneidade, a expressividade e a aplicabilidade do protótipo sem extrair medidas a partir de sua execução. Já o último método avalia exclusivamente medidas de performance obtidas a partir da execução do protótipo as comparando com resultados de abordagens do estado da arte.
A identificação explícita das heterogeneidades entre os modelos de interação através do modelo de protocolo fornece subsídios para análise do suporte à heterogeneidades. As heterogeneidades entre os modelos de interação básicos são suportadas. Algumas heterogeneidades adicionais, como descrição explícita de protocolos de aplicação, para as quais não há padrões *de jure* ou *de facto*, e por isso não são incluídas nos modelos de interação básicos, também são suportadas. Um cenário demonstra que a arquitetura tem uso prático. No entanto, extensões desse cenário revelam limitações oriundas (1) da baixa expressividade das descrições existentes para serviços orientados a eventos; e (2) da ausência de descoberta de serviço durante execução do serviço composto.
Foram realizados experimentos comparando um protótipo da arquitetura Unserved com o ComposIT (RODRIGUEZ-MIER et al., 2012), um algoritmo de composição de serviços SOAP e com o Pragmatic Proof(VERBORGH et al., 2016), a única abordagem de composição RESTful que efetivamente suporta a restrição HATEOAS. No primeiro experimento, composto por oito cenários, o protótipo apresentou média de tempo de composição menor em sete dos oito cenários, sendo mais lento apenas no cenário menos complexo. O segundo experimento avalia a escalabilidade em três cenários. Dentro de cada cenário, o aumento da complexidade dos problemas teve um efeito menor sobre o protótipo do que na implementação do algoritmo de composição RESTful.

CONCLUSÕES

Como confirmado pelos resultados, a arquitetura suporta os três modelos de interação básicos (SOAP, RESTful e orientados a eventos) e possui aplicação prática viável, representando um avanço em relação ao estado da arte, tanto em termos de funcionalidade – na comparação com abordagens de composição heterogênea – quanto em termos de desempenho – na comparação com o ComposIT (RODRIGUEZ-MIER

et al., 2012) e com o *Pragmatic Proof* (VERBORGH et al., 2016). Ainda assim, diversos aspectos não foram considerados nessa dissertação, entre eles: distribuição da arquitetura (para maior escalabilidade), heterogeneidade de dados e Qualidade de Serviço. Os experimentos realizados com o protótipo focam apenas na comparação com dois trabalhos fortemente relacionados. Aspectos como usabilidade e efeito na produtividade de programadores não são avaliados. Como trabalhos futuros específicos da arquitetura, podem ser listados: representação explícita dos modelos de interação dentro da arquitetura; novos modelos de interação; integração com motores de processamento de eventos complexos; melhorias na expressividade dos serviços compostos viabilizada pela arquitetura e pelo algoritmo de composição.

Palavras-chave: Serviços Web, Composição de Serviços, Planejamento Automático, Arquitetura de Software, Hipermídia, Eventos, Publish/Subscribe.

ABSTRACT

Current Web-based Services are highly heterogeneous not only on data but also with respect to service interaction. Despite their heterogeneity, composition of these services is required in order to achieve additional functionality. Semantic descriptions and composition algorithms for heterogeneous services have been proposed. However, existing techniques either ignore event-oriented services, which employ Publish/Subscribe or related paradigms, or do not offer sufficient support for interaction through hypermedia controls (i.e., links, forms and URI templates) as required in the REST architectural style. This dissertation presents a software architecture and two techniques, forking and adaptation, for automatic composition of heterogeneous services. Unlike current proposals, by combining a intermediary description, a composition algorithm and a middleware, all interaction constraints related to RESTful and event-oriented services are respected. Two experiments with a prototype implementation of the architecture were performed. The first compared it against a fast algorithm for composition of SOAP services, and the second compared it against the only algorithm found in the literature that supports the Hypermedia As The Engine Of Application State (HATEOAS) constraint of RESTful services. In the first experiment, consisting in eight scenarios, the prototype was faster in 7 scenarios. The second experiment showed that the prototype is more scalable than the RESTful services composition algorithm and less sensitive to the number of I/O parameters.

Keywords: Web Services, Service Composition, Automated planning, Software Architecture, Hypermedia, Events, Publish/Subscribe.

LIST OF FIGURES

LIST OF LISTINGS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ACL	Agent Comminucation Language. 172
API	Application Programming Interface. 40, 136, 213
APP	Atom Publishing Protocol (GREGORIO; HORA, 2007). 40, 63, 71
BGP	Basic Graph Pattern (SEABORNE; HARRIS, 2013). 116, 245
BPEL	Business Process Execution Language (JORDAN et al., 2007). 65, 80, 81, 88, 91, 93, 94, 95, 97, 134, 169, 172, 220
BPMN	Business Process Model and Notation (OMG, 2011). 93, 220
CEP	Complex Event Processing. 91, 103, 172
CI	Confidence Interval. 147, 148, 154, 157, 158, 160, 163, 167, 229
CoAP	Constrained Application Protocol (SHELBY; HARTKE; BORMANN, 2014). 60
CORBA	Common Object Request Broker (OMG, 2015a). 78
CSV	Comma Separated Values. 217
DAML-S	DARPA Agent Markup Language for Services. 100
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer (FENSEL et al., 2001). 52
DBMS	DataBase Management System. 52
DHT	Distributed Hash Table. 70
DNSSD	DNS based Service Discovery (CHESHIRE; KROCHMAL, 2013). 70
DSL	Domain-Specific Language. 93
ECA	Event-Condition-Action. 88, 94, 97
EOS	Event-Oriented Service. 11, 40, 41, 42, 44, 62, 63, 71, 82, 85, 86, 87, 91, 93, 94, 95, 97, 98, 99, 100, 103, 105, 106, 114, 118, 127, 128, 132, 133, 134, 135, 138, 139, 140, 169, 217, 220, 229, 231, 237
FIPA	Foundation for Intelligent Physical Agents. 172
GC	Garbage Collector. 142, 152

GUID	Globally Unique Identifier. 151
HAL	JSON Hypertext Application Language (KELLY, 2016). 60
HATEOAS	Hypermedia As The Engine Of Application State (FIELDING; TAYLOR, 2002). 17, 39, 40, 42, 58, 61, 62, 89, 90, 91, 92, 93, 95, 97, 98, 100, 102, 103, 105, 114, 127, 128, 131, 132, 135, 139, 161, 168, 172, 213, 215, 228, 231
HTML	HyperText Markup Language (BERNERS-LEE; CONNOLLY, 1995; LEITHEAD et al., 2017). 50, 58, 60
HTN	Hierarchical Task Network (EROL; HENDLER; NAU, 1994). 97, 100
HTTP	HyperText Transfer Protocol (FIELDING et al., 1999; BELSHE; ROBERTO; MARTIN, 2015). 39, 40, 41, 42, 57, 59, 60, 61, 62, 64, 73, 77, 81, 82, 90, 93, 94, 102, 108, 109, 111, 112, 113, 114, 133, 134, 139, 213, 238, 247
IANA	Internet Assigned Numbers Authority. 70
ICMP	Internet Control Message Protocol (POSTEL, 1981). 157
IDL	Intermediate Description Language. 78
IOPE	Input, Output, Preconditions and Effects. 54, 133
IoT	Internet of Things. 127, 140
IRI	Internationalized Resource Identifier (DÜERST; SUIGNARD, 2005). 50, 51, 61, 93, 239
JIT	Just In Time. 142, 152
JSON	JavaScript Object Notation (THE..., 2013). 51, 60, 73, 107
JSON-LD	JSON - Linked Data (LANTHALER; KELLOGG; SPORNY, 2014). 51
JVM	Java Virtual Machine. 141, 142, 151, 152, 155
LAN	Local Area Network. 57
MAS	Multi-Agent System (WOOLDRIDGE; JENNINGS, 1995). 91, 92
MCS	Multi-Context System (BREWKA; EITER, 2007). 133, 172

MQTT	Message Queuing telemetry Transport (BANKS; GUPTA, 2014)....................................60
NFP	Non-Functional Property.................. 71, 72, 237
OWL	Web Ontology Language (MOTIK; PATEL-SCHNEIDER; PARSIA, 2012). 44, 52, 72, 73, 107, 108, 111, 115, 116, 125, 143, 173, 239
OWL-S	OWL semantic markup for Services (MARTIN et al., 2007)................................72, 91, 100, 101
PaaS	Platform as a Service (MELL; GRANCE et al., 2011). 59
PDDL	Planning Domain Definition Language (MCDERMOTT et al., 1998)................... 101, 134, 135, 136, 172
PDF	Probability Density Function.............150, 155, 159
PICOC	Population, Intervention, Comparison, Outcome, Context (PETTICREW; ROBERTS, 2008)............217, 218
POX	Plain Old XML...................61, 81, 93, 128, 167
PuSH	PubSubHubbub (FITZPATRICK et al., 2014).......64
QoS	Quality of Service.................54, 74, 76, 101, 168
RAM	Random Access Memory.........................141
RDF	Resource Description Framework (CYGANIAK; WOOD; LANTHALER, 2014)...........19, 27, 44, 50, 51, 52, 59, 60, 72, 73, 106, 107, 108, 109, 112, 115, 123, 124, 125, 135, 141, 154, 168, 237, 238, 239, 247, 248
RDF/XML	RDF XML Syntax (GANDON; SCHREIBER, 2014).51
RDFS	RDF Schema (BRICKLEY; GUHA, 2014)...51, 72, 73, 108, 115, 116
REST	REpresentational State Transfer (FIELDING; TAYLOR, 2002)....39, 40, 41, 42, 44, 56, 57, 58, 59, 60, 61, 62, 77, 78, 82, 86, 87, 90, 91, 92, 95, 128, 213, 214, 215, 217, 218, 224
RMM	Richardson Maturity Model (RICHARDSON, 2008; WEBBER; PARASTATIDIS; ROBINSON, 2010)...61, 62, 81, 90, 91, 93, 94, 95, 100, 103, 128, 133, 139, 167, 224, 226, 231
RPC	Remote Procedure Call...........41, 62, 78, 79, 80, 96
RPM	Revolutions Per Minute..........................141

RQ	Research Question	85, 217, 218, 220, 227
RSS	Realy Simple Syndication (WINER, 2003)	40
SATA	Serial AT Attachment	141
SAWSDL	Semantic Annotations for WSDL and XML Schema (KOPECKÝ et al., 2007)	44, 72, 73, 80, 81, 90, 101, 106, 134, 239
SLA	Service Level Agreement	229
SLR	Systematic Literature Review	42, 43, 44, 45, 56, 61, 66, 85, 86, 95, 103, 105, 106, 127, 128, 134, 139, 167, 169, 170, 217, 223, 224, 229
SMS	Systematic Mapping Study	85
SOA	Service Oriented Architecture (PAPAZOGLOU, 2003)	40, 64
SOC	Service-Oriented Computing (PAPAZOGLOU, 2003)	68
SPARQL	SPARQL Protocol and RDF Query Language (SEABORNE; HARRIS, 2013; FEIGENBAUM et al., 2013)	52, 59, 113, 115, 124, 125, 173
SPIN	SPARQL Inferencing Notation (KNUBLAUCH; HENDLER; IDEHEN, 2011)	52, 112, 113, 123, 124, 125, 133, 136, 154, 240
SQL	Structured Query Language (INFORMATION..., 2016)	213
STS	State Transition Systems	68
SUMO	Suggested Upper Merged ontology (NILES; PEASE, 2001)	48
SWRL	Semantic Web Rule Language swrl	102
TAK	Title or Abstract or Keywords	222, 223
UFO	Unified Foundational Ontology (GUIZZARDI, 2005)	48, 111
UML	Unified Markup Language (OMG, 2015b)	65, 107, 108
URI	Universal Resource Identifier (BERNERS-LEE; FIELDING; MASINTER, 2005)	50, 51, 59, 60, 72, 73, 81, 90, 91, 94, 108, 109, 128, 139, 213, 239
URL	Universal Resource Locator (BERNERS-LEE; FIELDING; MASINTER, 2005)	63
W3C	World Wide Web Consortium	55, 60, 140
WADL	Web Application Description Language (HADLEY, 2009)	71, 73, 91, 93

WoT	Web of Things (KAJIMOTO; KOVATSCH; DAVULURU, 2017)....................................60, 140
WSC	Web Services Challenge (BANSAL et al., 2008)....101, 120, 127, 143, 144, 143, 144, 145, 144, 146, 147, 148, 150, 154, 157, 159, 160, 162, 167, 168
WS-CDL	Web Services Choreography Description Language (BARRETO et al., 2005)........................78, 80
WSDL	Web Services Description Language (WEERAWARANA et al., 2007)..56, 71, 72, 78, 80, 81, 91, 93, 100, 101, 106, 114, 169, 238, 239
WSMO	Web Service Modeling Ontology (ROMAN et al., 2005). 72, 90, 91, 112
XML	eXtensible Markup Language (COWAN et al., 2006).50, 51, 56, 72, 73, 79, 81, 91, 107, 108, 238, 239
XMPP	eXtensible Messaging and Presence Protocol (SAINT-ANDRE, 2011)....................................60
XSD	XML Schema Definition (BEECH et al., 2012)..50, 71, 72, 81, 134

LIST OF IRI PREFIXES

`atom`	`http://www.w3.org/2005/Atom#`
`ex`	`http://example.org/ns#`
`foaf`	`http://xmlns.com/foaf/0.1/`
`http`	`http://www.w3.org/2011/http#`
`httpm`	`http://www.w3.org/2011/http-methods#`
`hydra`	`http://www.w3.org/ns/hydra/core#`
`log`	`http://www.w3.org/2000/10/swap/log#`
`math`	`http://www.w3.org/2000/10/swap/math#`
`om`	`http://www.wurvoc.org/vocabularies/om-1.8/`
`owl`	`http://www.w3.org/2002/07/owl#`
`rdf`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs`	`http://www.w3.org/2000/01/rdf-schema#`
`saref`	`http://uri.etsi.org/m2m/saref#`
`sp`	`http://spinrdf.org/sp#`
`u`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved.ttl#`
`ua`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/atom.ttl#`
`uc`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-c.ttl#`
`uh`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/http.ttl#`
`uj`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-j.ttl#`
`umt`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/media-type.ttl#`
`un`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-n.ttl#`
`uo`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-o.ttl#`
`up`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-p.ttl#`
`ur`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/rdf.ttl#`
`us`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/sawsdl.ttl#`
`uw2`	`https://alexishuf.bitbucket.io/2016/04/unserved/unserved-x/wsdl20.ttl#`
`wgs`	`http://www.w3.org/2003/01/geo/wgs84_pos#`
`wot`	`http://iot.linkeddata.es/def/wot#`
`xsd`	`http://www.w3.org/2001/XMLSchema#`

LIST OF SYMBOLS

$\leftarrow$	Assignment
$\exists$	Existential quantification
$\rightarrow$	Implication
$\wedge$	Logical and
$\vee$	Logical or
$\mapsto$	Maps to
$\sqsubseteq$	Subclass (in ontologies)
$\subseteq$	Subset: $\forall x \,.\, x \in A \rightarrow x \in B$
$\langle \ldots \rangle$	Tuple
$\forall$	Universal quantification

CONTENTS

1 INTRODUCTION

In the context of distributed computing systems, services are computational elements that perform functions and are used to compose distributed applications at a lower cost (PAPAZOGLOU, 2003). Three properties of services listed by Papazoglou (2003) distinguish them from software reuse approaches (KRUEGER, 1992). First, services must be accessed by the lowest common denominator technology (technology neutrality). Second, no contextual information of the provider should be required of the consumer and *vice versa* (loose coupling). Third, the definitions and location of services should be discoverable (location transparency).

The environment where the notion of service as a computing paradigm found its largest impact is the Web. The flexibility of HyperText Transfer Protocol (HTTP) and related technologies, over which the Web is constructed, made for an attractive common ground for delivering services. One of the early standardization efforts led to the development of the SOAP protocol and a related set of technologies and standards then known as “Web Services” (TOM et al., 2014), but here referred to as SOAP services. Later, motivated by performance concerns (MIZOUNI et al., 2011; ARROQUI et al., 2012; AIJAZ et al., 2009; MULLIGAN; GRAČANIN, 2009; UPADHYAYA et al., 2011a), industry started exploring the implementation of services directly over HTTP, ditching SOAP and related technologies.

In rejecting SOAP, the REpresentational State Transfer (REST) architectural style proposed by (FIELDING, 2000) served as a common influence towards using HTTP directly. This exploration outside of the highly standardized SOAP services was seemingly chaotic, leading to some confusing terminology both in industry and academia (FIELDING, 2008; MALESHKOVA; PEDRINACI; DOMINGUE, 2010; RENZEL; SCHLEBUSCH; KLAMMA, 2012; SALVADORI; SIQUEIRA, 2015). A common misunderstanding is that the usage of HTTP alone, without SOAP, is sufficient for complying to the REST architectural style. This movement by industry yielded two new types of service. The simple direct usage of HTTP, without additional protocols stacked over it, yields Web APIs. The application of REST constraints, usually over HTTP, yields RESTful services.

The REST architectural style (FIELDING; TAYLOR, 2002) imposes five constraints: Client-Stateless-Server, Cache, Layered System, Code on Demand, and Uniform interface. The Uniform Interface is the most distinctive of these, and consists of four other constraints. First, adopt a single scheme for identifying resources. Second, manipulate those resources through their representations. Third, all messages should be self-descriptive in the sense that their meaning can be determined by intermediaries without knowledge of previous interactions. And lastly, abide to the Hypermedia As The Engine Of Application State (HATEOAS) constraint. That is, the representations contain hypermedia controls, such as links and forms that allow a user to transition its state by following a link to other resource or changing the state of resources when submitting a form. Ignoring the controls in favor of purely client-generated resource identifiers and requests, in addition to coupling service and client implementations, amounts to ignoring the application protocol of the service, possibly leading to failures or incomplete/incorrect results.

Parallel to the schism between SOAP services and Web APIs, many service designers encountered the need of handling server-originating events. HTTP is based on the Client-Server paradigm, where the server passively awaits for client-issued requests. In the realm of SOAP services, WS-Notification (GRAHAM; HULL; MURRAY, 2006) and WS-Eventing (BOX et al., 2006) were proposed as solutions, neither being the definitive standard. For Web APIs, several solutions and conventions were developed, such as Atom Publishing Protocol (APP) (GREGORIO; HORA, 2007), long polling (LORETO et al., 2011, p. 4) and HTTP 2 server push (BELSHE; ROBERTO; MARTIN, 2015, p. 60).

Figure 1 shows the percentages of service types in ProgrammableWeb[1], a primarily human-readable public directory of services, as of September 2017. The granularity of services in ProgrammableWeb is large (e.g., Amazon EC2 API is a single entry), therefore some entries represent both a SOAP and an REST implementation. As ProgrammableWeb contains mostly Internet-accessible and product-embedded Web Services, Web APIs dominate. SOAP services are also often offered in parallel with RESTful services (3.77%), but still 8.58% of services are exclusively SOAP. Nevertheless, HATEOAS presents the most significant challenges to service composition, yet services employing this constraint are nearly half of all services. ProgrammableWeb has no identification of Event-Oriented Service (EOS). However, quick inspection of popular services reveals event-related functionality, such as Realy Simple Syndication (RSS) feeds (WINER, 2003) in the Eventful API[2] and streams in the Twitter API[3].

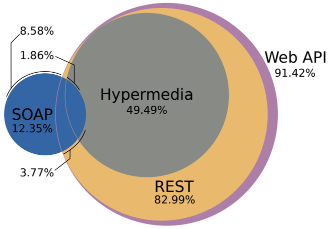
---

[1] <https://www.programmableweb.com/>

[2] <http://api.eventful.com/docs/venues/rss>

[3] <https://dev.twitter.com/streaming/overview>

Figure 1 – Venn diagram for types of services in ProgrammableWeb as of September 2017, identified though keyword analysis.



Source: the author, Appendix A.

Data and functionality of services can be aggregated by composite services, which can be constructed as any other service or can be designed by algorithms. The design and execution of composite services constitute the process of service composition (BERARDI et al., 2003), which also names the research area. The classic software architecture for service composition is Service Oriented Architecture (SOA), which envisions the discovery and description of services. Potential service consumers rely on discovery to find and select the best possible services for the required purposes according to their descriptions.

The differences between the mentioned service types is not simply a matter of heterogeneous communication protocols (i.e., SOAP versus HTTP). EOSs impose an heterogeneity of interaction paradigms: such services will employ notification or Publish/Subscribe paradigms (EUGSTER et al., 2003), which contrast against the Client-Server paradigm (ANDREWS, 1991; SINHA, 1992) on the fact that the server has the initiative. RESTful services forgo the concept of service-specific operations, a core element in SOAP service interfaces. RESTful services also expose the application protocol through hypermedia controls, which contrast with the prevalent strategy of designing a composite service and then executing it (VERBORGH et al., 2016).

The interaction paradigm, the possible operations, their possible orderings, the data formats and the transport protocol stack, together form the interaction model of a service. Interaction models can be partially specified by some constraints, e.g., the interaction model imposed by REST constraints, or that imposed by Remote Procedure Call (RPC). Given the discussed heterogeneity of service types, there is a need for taking heterogeneity of interaction models into account during service composition. There are proposals in the literature with this goal, defining common descriptions (ROMAN et al., 2015), proxy services generation (UPADHYAYA et al., 2011b; PENG; MA; LEE, 2009), middlewares (GEORGANTAS et al., 2013; HANG; ZHAO, 2013) and workflow languages or extensions (PAUTASSO, 2009; SRBLJIĆ; ŠKVORC; SKROBO, 2010).

The problem tackled in this dissertation, composition of services with heterogeneous interaction models, remains an open issue. A Systematic Literature Review (SLR), performed as part of this research, identified 54 original works. None supports the HATEOAS constraint, as a consequence of either ignoring the constraint or of building static composite service specifications that assume the offer of specific hypermedia controls. In addition, only four approaches do not require the composite service specification to be given as input. Three works (ARDISSONO et al., 2009; CHENG et al., 2017; LIU et al., 2014) claim to support RESTful services, SOAP services and EOS. However, these approaches violate HATEOAS, do not present evidence of respecting semantics of HTTP verbs, and (ARDISSONO et al., 2009) in particular does not support the notion of resources, central to the REST architectural style.

While heterogeneous composition approaches fail to fully support REST, there is a composition algorithm that fully supports it, including the HATEOAS constraint (VERBORGH et al., 2016). Services which are only partially RESTful (i.e., do not abide to all constraints of the architectural style), including Web APIs, are still an important parcel of existing services and should also be considered. In the case of SOAP and EOS heterogeneity, there are composition approaches that adequately cover the heterogeneities between the interaction models (SRBLJIĆ; ŠKVORC; SKROBO, 2010; GEORGANTAS et al., 2013). As for SOAP, there is extensive literature of automatic composition algorithms available (RODRIGUEZ-MIER et al., 2016; ALVES et al., 2016; CHATTOPADHYAY; BANERJEE; BANERJEE, 2015). Considering these observations, this dissertation aims to confirm the following hypothesis:

Hypothesis: Heterogeneous composite services, specifically those involving SOAP services, RESTful services (as well as degenerate variations) and EOSs, can be designed and executed by algorithms, in conformance to each of the corresponding interaction models.

1.1 OBJECTIVE

The goal of this dissertation is to define an architecture for the automatic composition of heterogeneous composite services. Both the methods for design of a composite service specification and for its execution must ensure (1) conformance with the basic interaction models implied by SOAP services, RESTful services and Event-Oriented Services; and (2) performance close to state-of-the-art approaches for homogeneous composition. Special attention must be given to the HATEOAS constraint, which is the most characteristic constraint of the REST architectural style, and while not being optional, has been neglected by existing approaches.

1.2 CONTRIBUTIONS

This dissertation puts forth the following contributions:

1. A categorization of heterogeneities observed in the context of service composition through the notion of interaction models and analysis of shortcomings in current approaches;
2. An abstract design-and-execution algorithm that, in combination with the architecture, enables heterogeneous service composition.

1.3 METHOD

This research aims to confirm the hypothesis stated earlier by achieving the objective (Section 1.1). The following research method was adopted:

1. Conduct an SLR with scope in heterogeneous service compositions. This SLR will include a categorization and analysis of heterogeneities support (Chapter 3);
2. Categorization of heterogeneities between the basic interaction models (Section 2.5);
3. Locate, in recent functional service composition literature, time-efficient algorithms for composite service design and algorithms supporting composition of basic interaction models other than SOAP services (Section 3.2);
4. Define an architecture for automatic heterogeneous compositions. A subset of techniques identified in the SLR should be used (Chapter 4);
5. Design planning and execution algorithms for heterogeneous composite service design using the defined architecture (Section 4.2);

6. Implement a prototype of the architecture algorithms;
7. Evaluate, through architecture analysis (HEVNER et al., 2004), the architecture support for heterogeneities between the basic interaction models (Subsection 5.1.1);
8. Evaluate expressivity of composite service automatically designed by the architecture and the practical applicability of the architecture through a scenario (Section 5.2 and Subsection 5.1.2);
9. Evaluate through experiments the practical viability of the architecture and the algorithms, and verify if required time for planning compositions is close or smaller under the prototype when compared against state of the art design algorithms (Section 5.3).

1.4 SCOPE DELIMITATION

Although the definition of EOS and REST applies to services other than Web Services, the scope of this dissertation, as well as for 9 out of 48 works identified in the SLR of Section 3.1, is restricted to Web Services. For readability and consistency, the term “service” is used (e.g., RESTful service), while “Web Service” is reserved for stating something true for Web Services but not necessarily true for other types of service.

A Web Service can serve data of any nature. However, the software artifacts which are a byproduct of this dissertation only consider data in RDF syntaxes. RDF is the *de facto* abstract syntax (there are multiple concrete syntaxes) for use in the Semantic Web. As a related restriction, the implementation only considers services with descriptions referencing classes and properties in ontologies written in Web Ontology Language (OWL). These restrictions are not without precedent in the literature. Both are used in the composition algorithm by (VERBORGH et al., 2016). The need of semantic service description is also common in service composition (ALVES et al., 2016; HOBOLD; SIQUEIRA, 2012; RODRIGUEZ-MIER et al., 2012) and there are several service description ontologies (LANTHALER; GÜTL, 2013; VERBORGH et al., 2013) and languages (KOPECKÝ et al., 2007). Non-RDF content could be converted to and from RDF using techniques such as the lifting and lowering mappings defined on SAWSDL (KOPECKÝ et al., 2007).

As a final restriction on scope, while the focus of this dissertation is on heterogeneity, and ontologies are adopted, ontology heterogeneity beyond consideration of `owl:sameAs`, `owl:equivalentProperty` and `owl:equivalentClass` OWL properties is not dealt with. Semantic Web open issues, such as ontology alignment (OTERO-CERDEIRA;

RODRÍGUEZ-MARTÍNEZ; GÓMEZ-RODRÍGUEZ, 2015), are considered out of scope.

1.5 DOCUMENT STRUCTURE

Chapter 2 defines background concepts and justifies used definitions when there are conflicts in relevant literature. The main outputs of this chapter are a taxonomy of service types Subsection 2.3.2 and the associated basic interaction models and their heterogeneities. Chapter 3 conducts an SLR in order to identify and summarize related work. This SLR is an improved version of the one presented in (HUF; SIQUEIRA, 2017). A brief survey on automatic service composition of SOAP and RESTful services complements the SLR. Chapter 4 presents the Unserved software architecture, which was intially propsed in (HUF; SALVADORI; SIQUEIRA, 2017) but received improvements (pre/post-conditions) for this dissertation and is described in more detail. The architecture has its composite service expressivity and support for heterogeneities analyzed in Section 5.1; its application to a practical scenario discussed in Section 5.2; and its performance compared to related work in Section 5.3. Conclusions, limitations, threats to validity and future work are presented in Chapter 6.

This dissertation also includes a glossary (p. 245) with terms whose definitions are not inline in the main text. In the case of terms defined in the main text, there is an index (p. 249) pointing to pages where the terms are defined (margin notes mark position inside the page). The PDF version includes hyperlinks to said definitions.

2 BACKGROUND

This chapter establishes a common terminology that will be adopted in this document for describing the proposed architecture. It is important to notice, though, that some concepts present in this terminology are subject to varying levels of controversy in the literature.

2.1 ONTOLOGY

The word ontology originates from philosophy, where it is a discipline of study aiming to answer questions such as "what there is?" and "how the things there are relate to each other", among others (HOFWEBER, 2017). In Computer Science the term has come to mean a software engineering artifact, composed by a vocabulary and explicit assumption about its intended meaning (GUARINO, 1998). Guizzardi (2007) attributes to Hayes (1978) the introduction of the term in Artificial Intelligence, which borrows its meaning to all Computer Science disciplines.

The most cited[1] definition of ontology is that provided by Gruber (1993), where an ontology is "an explicit specification of a conceptualization", with a conceptualization being the body of "objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them" (GRUBER, 1993 apud GENESERETH; NILSSON, 1987, p. 199). Giaretta & Guarino (1995) are critical of this definition. The main issue being that under the cited definition of conceptualization, relations are defined extensionally, i.e., a relation $above(x, y)$ is defined by the set of pairs (x, y) for which it is true. If the world state changes, so does the relations, the conceptualization and the ontology.

Giaretta & Guarino (1995) propose an alternative notion of conceptualization and revise the definition to "an ontology is a partial account of a conceptualization". This is the definition adopted in this dissertation, usually accompanied by an ontology language used to specify the ontology. In other words, an ontology consists of a domain of discourse and a set of constraints on which relations should hold in which possible worlds. Since any set of constraints, specified in a logical lan-

---

[1] All works with terms "ontology" and "ontologies" that Scopus reports as having at least half of the 6950 citations of (GRUBER, 1993) were evaluated. Except for GRUBER, all other works are from biology fields. These biology papers present specific domain ontologies and only (ASHBURNER et al., 2000) had a definition of the term ontology. It is unlikely that citations to (ASHBURNER et al., 2000) are refering to such definition instead of the domain ontology.

guage, may have more than one logical model, the set will have more than one corresponding conceptualization. From this stems the introduction of “a partial account” in the definition. This revised definition makes explicit the distinction between a conceptualization (the philosophical notion of ontology), the ontology (an explicit but approximate description of the intended conceptualization) and the knowledge base (assertions using the notions made explicit by the ontology).

As a step before reaching the revised definition of ontology, Giaretta & Guarino (1995) examine two forms of specifying the constraints which are part of an ontology. One using first-order logic and another able to represent finer constraints based on modal logic, each form with different proposed uses. This notion is expanded on later works (GUARINO, 1997; GUARINO, 1998; GUIZZARDI, 2007; FALBO, 2014), and originates a classification according to the amount and structure of the conceptualization:

1. **Reference ontologies** (a.k.a. documenting/off-line ontologies) contain extensive formal descriptions of relation constraints and have the purpose of establishing a consensus about the intended conceptualization. An example is the UFO-S ontology (NARDI et al., 2015)
2. **Operational ontologies** (or shareable/on-line/lightweight ontologies) contain simplified descriptions of the conceptualization in languages that have desirable computational properties, such as decidability or polynomial time reasoning algorithms. In this case, expressivity is sacrificed under the assumption that agents have previously agreed upon the intended conceptualization. The Minimal Service Model (PEDRINACI et al., 2010; ROMAN et al., 2015), the pizza ontology[2] and the intermediary description proposed in Subsection 4.1.1 are examples of operational ontologies.

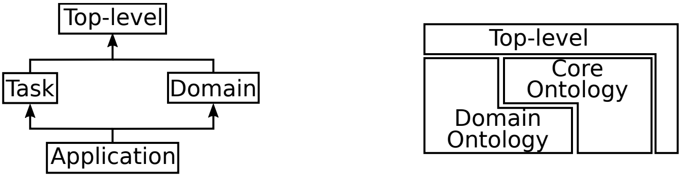
A second dimension for classification of ontologies is the subject of the underlying conceptualization. Guarino (1998) identify 4 such categories (Figure 2a). Scherp et al. (2011) additionally identifies a category of core ontologies (Figure 2b). The resulting categories are listed below:

1. **Top-level ontologies** (a.k.a. upper/foundational ontologies): Describe concepts independent of domain, task and application, such as space, time, events, actions, participation, numbers, etc.

---

[2] https://protege.stanford.edu/ontologies/pizza/pizza.owl

Figure 2 – Types of ontologies according to subject of the conceptualization.



(a) Types of ontology (GUARINO, 1998)

(b) Role of core ontologies (SCHERP et al., 2011).

Source: the author, adapted from respective sources.

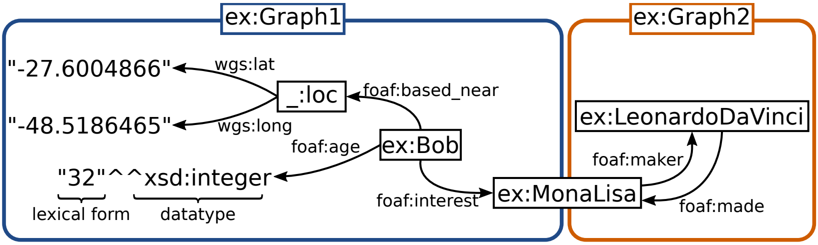
Examples are Suggested Upper Merged ontology (SUMO) (NILES; PEASE, 2001) and Unified Foundational Ontology (UFO) (GUIZZARDI, 2005);

2. **Domain ontologies**: Describe concepts and relations specific to a domain of knowledge, such as chemicals (HASTINGS et al., 2013), or genes (ASHBURNER et al., 2000);
3. **Task ontologies**: Describe concepts specific to a particular task, such as medical diagnosis;
4. **Application ontologies**: Describe concepts tied both to a domain and a task. For example, (BERTAUD-GOUNOT; DUVAUFERRIER; BURGUN, 2012) describes an ontology with concepts such as "Spondyloarthritis Clinical Finding", tied to a domain (the disease) and a task (diagnosis);
5. **Core ontologies**: A core ontology describes concepts from a field which spans multiple domains. As Figure 2b summarizes, it is between top-level and domain/task ontologies in terms of generality. A Core Ontology is expected to refine a top-level ontology and to be used by domain/task ontologies or directly by applications or application ontologies. Examples of such cross-domain fields are multimedia annotation (ARNDT et al., 2007) or services (NARDI et al., 2015; NORTON et al., 2008).

2.2 SEMANTIC WEB

The Semantic Web envisions a Web of machine-readable data alongside the traditional document-based Web (BERNERS-LEE et al.,

Figure 3 – Sample RDF 1.1 graphs.



Source: the author, adapted from (RAIMOND; SCHREIBER, 2014, s. 3).

2001). To achieve the machine-readable requirement, the data semantics must be made explicit, a task which is the role of ontologies.

The core technology in the Semantic Web is RDF (CYGANIAK; WOOD; LANTHALER, 2014), whose first draft dates back to 1997 (LASSILA; SWICK, 1999). RDF itself describes an abstract syntax based on *subject-predicate-object* triples. A set of triples forms a graph, where subjects and objects are nodes and predicates are the edges. Graphs, identified by a unique Internationalized Resource Identifier (IRI)[3] are part of a Dataset, which must contain at least one graph named the *default graph*.

Figure 3 shows an example with two RDF graphs highlighting several concepts of the RDF abstract syntax (also referred to as data model). Triples are represented graphically as connection between nodes, such as `ex:Bob foaf:interest ex:MonaLisa`. The values used as subject, predicate and object of triples are called terms. The main type of RDF term are IRIs, like `ex:MonaLisa` (`ex:` is an abbreviation of an IRI prefix cf. p. 31). As an IRI is universal, `ex:MonaLisa` can be present in different graphs but still is interpreted a single resource. Due to this universality, predicates of RDF triples must always be IRIs. In some situations a universal identifier is not required, for example when describing an unnamed point in Earth with latitude and longitude. *Blank nodes*, such as `_:loc` fulfill this role: `loc` is a name whose scope is restricted to the RDF graph.

---

[3] RDF employs IRIs, which serve the same purpose as Universal Resource Identifiers (URIs), but can be composed of Unicode characters. A mapping from IRIs to URIs exists (DÜERST; SUIGNARD, 2005, p. 10–15), so there is no difference in the set of identifiable resources.

IRIs and blank nodes are already sufficient for building graphs albeit not with practical descriptions of the nodes. RDF also allows the use of literals in graphs, but restricts their use to objects in triples. In RDF 1.1, all literals have a lexical form and a datatype which defines how the string should be interpreted, as shown in Figure 3 for Bob's age. RDF uses the datatypes defined in XML Schema Definition (XSD) (PETERSON et al., 2012) as well as two additional datatype for eXtensible Markup Language (XML) and HyperText Markup Language (HTML). If the datatype of a literal is omitted, as is the case for longitude and latitude in Figure 3, it is assumed to be `xsd:string`.

This abstract syntax has multiple concrete syntaxes, such as RDF XML Syntax (RDF/XML), JSON - Linked Data (JSON-LD) and Turtle. Each of these has specific advantages, such as compatibility with existing tools and practices (XML processing for RDF/XML and JavaScript Object Notation (JSON) prevalence in Web APIs for JSON-LD). A common feature present in most[4] concrete syntaxes is IRI prefixing. This consists in using a mnemonic to avoid repeating long substrings shared by many URIs in a document. The IRI prefixes used in the code fragments of this dissertation are listed in p. 31.

Together with RDF (LASSILA; SWICK, 1999), a simple ontology language was developed, called RDF Schema (RDFS) (BRICKLEY; GUHA, 2014). RDFS provides means to describe class and property[5] subsumption hierarchies as well as domain and range of properties. Hierarchy definitions are restricted to `subClassOf` and `subPropertyOf`, with the semantics that if x `subClassOf` y, then the extension[6] of x is a subclass of the extension of y (and analogously for `subPropertyOf`). There are also constructs for describing collections and reification, which have no strict semantic defined (PATEL-SCHNEIDER; HAYES, 2014, appx. D).

Listing 1 shows an example RDF graph. All predicates, but `a` , which is an alias for `rdf:type`, are properties from the FOAF[7] ontology. Lines 1-4 describe that the RDF document itself (`<>` is a relative IRI) is a `foaf:PersonalProfileDocument` about and made by `<#me>`. Line 7 contains a triple with a literal as object. Lines 10-

---

[4] N-Triples(CAROTHERS; SEABORNE, 2014) and N-Quads (CAROTHERS, 2014) are concrete syntaxes without IRI prefix support.

[6] A triple has a *predicate*, which will always be an RDF *property* as per pattern *rdfD2* in (PATEL-SCHNEIDER; HAYES, 2014). The former term denotes a position, the latter a class of resources.

[6] The extension of a class A is the set of x nodes for which there is a triple x `rdf:type` A stated in the graph or inferred. For a property P, the extension is the set (x, y) where likewise x A y is stated or inferred (PATEL-SCHNEIDER; HAYES, 2014).

Listing 1 – Example RDF graph using the FOAF ontology.

```
 1  <> a foaf:PersonalProfileDocument;
 2    a foaf:Document;
 3    foaf:maker <#me>;
 4    foaf:primaryTopic <#me>.
 5
 6  <#me> a foaf:Person;
 7    foaf:name "John Doe";
 8    foaf:knows <http://jane.example.org/home/about#me>.
 9
10  _:acc foaf:accountName "john";
11    foaf:accountServiceHomepage <https://messenger.com>.
```

Source: the author.

11 describe a blank node representing an instant messaging account. Under RDFS semantics the triple in line 2 could be inferred from line 1 and the fact that `foaf:PersonalProfileDocument` is a subclass of `foaf:Document`. The triple `_:acc a foaf:OnlineAccount` is implicit as it could be inferred from line 10 and the fact that `foaf:accountName` has `rdfs:domain` in `foaf:OnlineAccount`.

OWL (MOTIK; PATEL-SCHNEIDER; PARSIA, 2012; PATEL-SCHNEIDER; HORROCKS; HARMELEN, 2002) is an ontology language succeeding the DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) project. OWL is based on description logics (BAADER, 2010) and allows for more complex constraints when defining an ontology, while still being representable in RDF. As examples, one could formalize the notion that a `foaf:Person` with an `foaf:OnlineAccount` is a `ex:ReachablePerson` and that an instance of `foaf:Person` cannot also be an instance of `foaf:Document`. In the context of the graph in Listing 1, the former constraint implies that the triple `<#me> a ex:ReachablePerson` is true. The latter would raise an inconsistency if the triple `<#me> owl:sameAs <>`, asserting that the person and the document are the same resource, were added to that graph.

In addition to its representation in documents, RDF can be stored in a specially designed DataBase Management System (DBMS) called triple store, which is optimized for storing RDF triples and querying them, possibly with support for inference at query time. Queries to triple stores are written using the SPARQL Protocol and RDF Query Language (SPARQL). The main type of queries are select queries, where a query such as `SELECT ?p ?n WHERE {?p foaf:name ?n.}`,

---

[7] <http://xmlns.com/foaf/spec/>

if posed against the RDF in Listing 1, would return ⟨`?p=<#me> ?n="John Doe"`⟩ as the only possible solution. SPARQL queries can also be embedded in RDF itself using an alternative syntax, named SPARQL Inferencing Notation (SPIN). SPIN can also be used to directly describe elements of the group basic graph pattern (the contents of the `WHERE` clause) independently from a command such as `SELECT` or `ASK`. This enables the use of SPARQL as a language for arbitrary logical expressions that in our proposal is used to express pre/post-conditions (Subsubsection 4.1.1.1).

2.3 SERVICES

As discussed by Nardi et al. (2015), the term "service" is subject to a phenomenon named *systematic polysemy.* This means that the term is used to refer to multiple distinct, albeit related, concepts. Therefore, even in the same document, the term may have its meaning changing according to context, and may at times assume more than one meaning simultaneously.

As part of evaluating their proposed core reference ontology, Nardi et al. (2015) collected in the literature four different perspectives on service in addition to the one used by themselves. Three of these five perspectives are used in the literature to define what a service *is.*

Service as behavior Some authors view "service" as behavior of the service provider. Frequently such behavior is described in terms of interaction between the provider and consumer. Service can therefore be defined as an atomic interaction that produces an outcome (QUARTEL et al., 2007); a process that can be changed according to consumer inputs (SAMPSON, 2010); or as a transaction, which is a sequence of coordinating interactions to negotiate the service, a sequence of production actions that result in a material or immaterial good, and a sequence of interactions to evaluate the service results (TERLOUW; ALBANI, 2013).

Service as competence Another perspective is on the competences of the service. Vargo & Lusch (2004, p. 2) define service, in the context of marketing and economy, as the application of specialized competences in benefit of another entity. In a context closer to computing, MacKenzie et al. (2006) define a service as a mechanism for accessing competences. While a distinction between competence and the mean to access it is done in this specification, it is not mandated. Although not citing this permission, several works in scientific or technical literature explicitly use the term service to refer to the competence (REZGUI et al., 2002; ACAMPORA et al., 2010; ZSEBY; ZANDER; CARLE, 2002; PREIST, 2004).

Service as commitment This is the perspective of the ontology proposed by Nardi et al. (2015). A service is a sequence of commitments and corresponding claims (demands) established between the service provider and target customers. For example, *Hotel Inc.* commits to host Bob from the 20^{th} to the 25^{th} of March 2018 in a "executive room" with breakfast and daily room cleaning. Bob commits to pay $400 up front and to leave by 12:00 PM on the 25^{th}.

Two remaining perspectives mentioned by Nardi et al. (2015) are not definitions of services, but account for contexts in which services are used and how services are characterized in that context.

Services for value co-creation This view originates from marketing and from the view of service as competence application. Vargo & Lusch (2004, p. 10), explore the definition of service as competence application and argue that the unit of exchange in economy is not goods, but services. Therefore, the purpose of a service is mutual creation of value, involving and benefiting both provider and consumer. Spohrer et al. (2008) later propose an abstraction called the service system, a configuration of people, technologies and resources, that interact with other service systems to create value.

Computational services Under this category, Nardi et al. (2015) mention the usage of services as a software design paradigm and their use in the context of communication protocols. In the former, Nardi et al. (2015) argue that services are mainly characterized by their Input, Output, Preconditions and Effects (IOPE). In the latter, as discussed by Vissers & Logrippo (1986), the main characteristic of a communication service is hiding implementation detail. The so called "service concept" (VISSERS; LOGRIPPO, 1986) allows one to abstract how data is communicated using the service provided through a lower-layer communication protocol.

An addendum must be offered to the discussion of computational services presented by (NARDI et al., 2015). With the goal of evaluating their own proposal, the authors do not discuss what is a computational service and concentrate on what characteristics of computational services are commonly accounted for. This is understand-able, as in a sense, all other perspectives are considered in works that deal with computational services. For example, (ROMAN et al., 2005) distinguish services from Web Services, defining the former under the perspective of competence application and the latter simultaneously under the perspectives of behavior (choreography), competence (capability) and commitment (non-functional properties). Another example is (CANFORA et al., 2005), which distinguishes abstract and corresponding concrete services. The former are defined by competence, and for the latter, only their commitments regarding Quality of Service (QoS) attributes are of importance.

2.3.1 Service in this Dissertation

The work of (NARDI et al., 2015) reveals a generalized lack of consensus on the meaning of “service”. However, one of the senses of the word is of central importance to this dissertation. In order to prevent misunderstanding and unnecessarily coining a new term for something that is not novel, a classical definition is adopted, with some aspects clarified.

This dissertation is concerned with services as viewed in the context of distributed computing systems. A classical definition is given by Papazoglou (2003), who defines services as computational elements that perform functions and are used to compose distributed applications at lower cost. Services have three fundamental properties: (1) Access through standardized, lowest common denominator technology; (2) Loose coupling, i.e., no contextual information of the provider is required from the consumer and vice-versa; and (3) Definition and location of service are discoverable, providing location transparency. This definition of service is consistent with that of the World Wide Web Consortium (W3C) (BOOTH et al., 2004), which defines a service as an abstract resource representing the capability of performing tasks that represent coherent functionality from the perspective of both provider and requester.

The adopted meaning of the term service in this dissertation is given by Papazoglou (2003). The term service is used as a computational element offered by a specific provider to a consumer, both computational elements themselves. Their nature (e.g., a server or an object in the memory of a running program) is defined by the context where “service” is used, if needed. The distinction between a person or organization and its corresponding software agent, as present in (BOOTH et al., 2004), is not relevant for the purpose of this dissertation. The term “agent” is therefore omitted to improve readability.

This use of the term service is consistent with the suggestion by (NARDI et al., 2015, p. 286) of *service offering* as the core meaning of the term. The term “service” denotes an element, not an action. It is not reasonable for a service to exist without the intention of being used. Therefore, a (software) agent being aware of the existence of a service, implies the existence of an offering of that service. It is still possible that the offering is directed to a specific group of consumers of which that agent is not a member.

2.3.2 Service Types

There are multiple forms in which a computational element can exist. In the case of services, terms such as “Web Service” and “REST service” are used to denote types of service. Similarly to what occurs with service and ontology, some of these terms lack a broadly accepted definition. Given the goals of this dissertation, a clear classification of service types is needed.

The construction of the classification that will be presented in Subsubsection 2.3.2.5 started from the three interaction models mentioned in Chapter 1. Once these classes of service were clearly defined, more general classes were sought. The definitions adopted for each class try to rely on characteristics widely accepted in the literature. This principle was violated only when necessary for the hierarchy consistency.

2.3.2.1 Web Services

In the context of distributed systems, the most popular refinement of the concept of service are Web Services. Similarly to “service”, although not to the same extent, there is no universal consensus on what is a Web Service. Some important specifications (TOM et al., 2014; JORDAN et al., 2007; BOOTH et al., 2004) bind the Web Service concept to specific technologies, such as Web Services Description Language (WSDL) and SOAP. However, many influential academic works list services that do not use any of those technologies under the term “Web Service” (GARRIGA et al., 2016; MAXIMILIEN et al., 2007b; PAUTASSO, 2009; LANTHALER; GüTL, 2010; BARTALOS; BIELIKOVA, 2011).

A particularly relevant example of inconsistency with “Web Service” as defined in specifications, is the term “REST Web Service”.

Table 2 – Works with "RESTful service" related terms in metadata. Queries executed on October 30, 2017. Duplicate detection was automated using the same method as described in Section B.4.

Regular Expression	Scop.	IEEE	ACM	Total Unique
REST(ful)? services?	392	268	228	465 (25.05%)
REST(ful)? Web services?	635	157	364	778 (41.92%)
REST(ful)? (web)? APIs?	503	202	244	613 (33.03%)

Source: the author.

Table 2 shows the number of works in the literature using variations of "REST service", "REST Web Service" and "REST API". Despite being the most frequent term, there is no evidence[8] that so called "REST Web Service" are instances of the REST architectural style over SOAP and XML.

Although "Web Service" in "REST Web Service" differs from the definition in specifications, the term Web Service is seldomly defined in these works. From the above mentioned highly cited works using the term in a broader sense, only (BARTALOS; BIELIKOVA, 2011) presents a definition of Web Service, as an "abstract entity providing its functionality on the Web". A similar re-definition is provided by Bouguettaya, Sheng & Daniel (2014, p. VII), where a Web Service " (...) exposes features programmatically over the Internet".

Defining "Web Service" in terms of SOAP or Web is problematic. The former is inconsistent with much of REST-oriented and heterogeneous service composition literature, as discussed in Section 3.1. The latter is inconsistent with services operating inside a corporate intranet or a Local Area Network (LAN). To avoid such inconsistencies, this dissertation defines Web Services as services that are accessed through HTTP (BELSHE; ROBERTO; MARTIN, 2015).

Adopting a broader definition of the term "Web Service" creates the need for the self-explanatory SOAP service term to indicate a service that employs the SOAP protocol. Similarly, SOAP Web Service, restricts the scope to only SOAP services stacked over the HTTP protocol.

---

[8] Such evidence should have been found during the SLR in Section 3.1. While it is possible that some works would only reveal usage of SOAP in their full text, the result from title, abstract and keywords shows these are a minority.

2.3.2.2 REST services and Web APIs

The term REST has a well-defined origin in the PhD thesis of Fielding (2000), also published as a highly influential article (FIELDING; TAYLOR, 2002). The REST architectural style focuses on manipulation of resource states through their representations using a uniform interface where interactions are stateless and interceptable by intermediaries. A *resource* is anything (e.g., information, objects, abstract concepts) that can be named by a *resource identifier*. A *representation* is a sequence of bytes that specifies the current or intended state of a resource.

REST is derived (FIELDING; TAYLOR, 2002) from five other architectural styles (Client-Server[9], Client-Stateless-Server, Cache, Layered System and Code on Demand, which is optional) and an additional set of constraints named *uniform interface*. The uniform interface is subdivided into four constraints: (1) identification of resources, (2) manipulation of resources through their representations, (3) self-descriptive messages and (4) Hypermedia As The Engine Of Application State (HATEOAS). The first two constraints imply that operations on resources involve the identifier, which is persistent, and representations of the resource state, which are specific to a point in time. Self-descriptive messages allow any intermediary processing element, such as a proxy or a gateway, to intercept the message and fully understand its semantics without knowledge of previous interactions between that user agent and the origin server[10]. The HATEOAS constraint provides a way of specifying the application protocol. The user agent determines its actions by examining and choosing the appropriate hypermedia controls (e.g., links and forms in case of HTML) among those present in the received representation.

An example provides a better overview of role of the HATEOAS constraint in an interaction. Consider a music player software. It offers several controls (buttons), such as play, pause, and skip to the next or previous track in queue. Each of these is only available when the player state allows for their execution. A control (button) that changes its label from “play” to “pause” when music starts playing or a “skip to next” control (button) that becomes disabled when the

---

[9] Although the original source uses the term Client-Server paradigm (SINHA, 1992; ANDREWS, 1991), the described paradigm can also be accounted as a architectural style. This is done in (FIELDING, 2000, p. 45) when describing styles from which REST is derived.

[10] In the REST architectural style, the term user agent and origin server are used to identify the client and the server, distinguishing them from intermediary proxies or gateways (FIELDING; TAYLOR, 2002).

last track is playing are examples. While the music player has no hypermedia controls, it is, by analogy, employing HATEOAS. Had it not been employing it, all controls would always be enabled and the software manual would list the conditions required for each control to be activated. This example can be easily transposed to RESTful web applications, where the controls would by hypermedia controls (i.e., links and HTML forms).

The term Web API, on the other hand has an unclear origin, and possibly originated from industry informally. From industry, its possible earliest mention is from a talk by a Salesforce representative[11] . In this context, the term Web API may have emerged naturally as meaning "an API on the web". In scientific literature, there are mentions as early as the one in (JAKOBOVITS; MODAYUR; BRINKLEY, 1996), were a "Web API" component inside the Web server was responsible for generating HTML code. Closer to its current meanings, the first mention probably is the one by Carroll et al. (2004) when referring to an RDF "NetAPI" implementation (SEABORNE, 2002), a predecessor to the SPARQL protocol (FEIGENBAUM et al., 2013).

Similarly to "Web Service", there is no precise and widely accepted definition of Web API. In academic literature, some representative definitions are given below:

- Panziera et al. (2011): Web APIs are another name for RESTful services;
- Heath & Bizer (2011): Web APIs have as goal providing access to data, typically use HTTP and widely adopt the REST architectural style;
- Maleshkova, Pedrinaci & Domingue (2010): Web APIs are services using the HTTP protocol and URIs for resource identification, which may not conform to the REST architectural style. Web APIs are antagonistic to SOAP services: a service cannot both be a SOAP service and a Web API;
- Cao et al. (2013): Web APIs are typically a set of HTTP request messages and typically considered a synonym of "web service", but applications have moved away from SOAP towards REST;
- Chen et al. (2009): Web APIs enable direct access to the website functionality and "are based on the Web Services architecture". This particular work classifies SOAP services as also instances of the concept of Web API.

---

[11] LANE, K. *History of APIs*. 2017. Available at: <https://history.apievangelist.com/>. Access on: 08 nov. 2017.

- Jayathilaka, Krintz & Wolski (2015): Web API is a “web-accessible API exported by an application hosted on a Platform as a Service (PaaS) platform”

These definitions are counterproductive due to excessive generality (overlapping with other service types) or excessive specificity. For the purposes of this dissertation the adopted definition of Web APIs is that of services provided over HTTP without layering an additional transport protocol (e.g., SOAP) over it. This definition is consistent with practice in industry, where Web API and SOAP are treated as disjoint concepts. In academia, it is consistent with REST definition, as many Web APIs do not fully conform to the REST architectural style.

REST constraints can be applied to services, yielding RESTful services, or to the corresponding service type (e.g., an REST Web Service). In the literature, the main related terms to RESTful service are those resulting from the regular expressions in Table 2. While there are three different terms being used, REST is majoritarily applied directly over the HTTP protocol. There are very few instantiations of REST over protocols other than HTTP. As examples, an instantiation over eXtensible Messaging and Presence Protocol (XMPP) (STANIK; KAO, 2016) and the possible[12] instantiation over Constrained Application Protocol (CoAP), Message Queuing telemetry Transport (MQTT) and WebSockets in the W3C Web of Things (WoT) upcoming standard (KAJIMOTO; KOVATSCH; DAVULURU, 2017). This dissertation adopts almost exclusively the term RESTful service, as the protocol over which REST is instantiated is often of no interest.

While most RESTful services employ HTTP, they differ from web applications by using machine-readable representations in place of HTML. This leads to a change in the nature of hypermedia controls. Hyperlinks are often found in representation formats used by RESTful services. RDF, for example, natively supports hyperlinks, as predicates correspond to the HTML link relation type and the object is the link target. In JSON there are draft standards for representing hypermedia controls. JSON Hypertext Application Language (HAL) (KELLY, 2016), for instance, defines how to embed hyperlinks in JSON. Hyperschema (ANDREWS; WRIGHT, 2018) goes beyond and describes hypermedia controls that take input either as URI template parameters (GREGORIO et al., 2012) or as an HTTP POST request body. Hydra takes yet another approach, where the hypermedia controls (links, URI templates and operations, sending input in the request body) are

---

[12] The WoT standard suite is not yet complete.

described *a priori* but the link targets are extracted from the RDF representations returned by the service.

A commonly used rationale for adoption of Web APIs (and RESTful services) is performance gain, specially reduced message payload size, response time and memory usage. Several works evaluate implementations of SOAP services against their Web APIs counterparts and observe reduction of these metrics (MIZOUNI et al., 2011; ARROQUI et al., 2012; AIJAZ et al., 2009; MULLIGAN; GRAČANIN, 2009; UPADHYAYA et al., 2011a). Although these works affirm to compare RESTful services to SOAP services, none analyzes the impact of the HATEOAS constraint.

A widely spread inconsistency is that often the term RESTful service is used to refer to what this dissertation classifies as Web APIs. This phenomenon occurs both in industry and academy. In the former case, it was first diagnosed by REST's creator, Fielding, informally[13] and confirmed by Maleshkova, Pedrinaci & Domingue (2010) when analyzing descriptions of Web APIs in the ProgrammableWeb[14] repository. Academically, this issue is observed in works selected for the SLR in Section 3.1, which among other issues, have no proper support for HATEOAS despite claiming to support REST.

Maturity models can be used to assess software quality as well to guide its development (WENDLER, 2012). REST maturity models all share the latter aim. Salvadori & Siqueira (2015), which themselves propose a model considering semantic support of RESTful services, also provide a coverage of existing maturity models. The first of such models to be proposed was the Richardson Maturity Model (RMM) (RICHARDSON, 2008; WEBBER; PARASTATIDIS; ROBINSON, 2010). The RMM consists of four levels:

0. Plain Old XML (POX): A service which handles a single resource, exposing a single IRI, also using a single HTTP verb. This is the case of SOAP services;
1. Resources: A service with multiple resources, each identified by a single IRI. However, action information is still encoded in representations and IRIs. For example, a `GET` request to `/room/123/reserve?name=Jon+Doe` encodes the act of creating a resource reservation within the resource identifier, violating the message self-description constraint;

---

[13] FIELDING, R. T. *REST APIs must be hypertext-driven.* 2008. Available at: <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. Access on: 3 nov. 2017.

[14] <https://www.programmableweb.com/>

2. HTTP verb: This level implies the semantic of HTTP verb and status codes (or of the protocol used in place of HTTP) are respected. To achieve this level, the example request used in Level 1 should be a `POST` to `/reservations/` with room number and guest name in the payload;
3. Hypermedia/HATEOAS: This level corresponds to enforcing the HATEOAS constraint. The service includes hypermedia controls in representations, from which the client determines its next actions and consequent state transitions.

Maleshkova, Pedrinaci & Domingue (2010) do not apply the RMM. Instead, they define three categories[15] of Web APIs: RPC-Style, corresponding to Level 1 (Resources) in RMM; RESTful, corresponding to Level 2 (HTTP verb) in RMM; Hybrid, which is the case of services for which some parts conform to Level 2, but some only conform to Level 1. The authors evaluated 18% of listed APIs and classified 47.8% as RPC-Style, 32.4% as RESTful and 19.8% as Hybrid. A more recent analysis on ProgrammableWeb, covering data from 2016 to 2017, is presented in Appendix A. However, this analysis is based only on keyword analysis in ProgrammableWeb descriptions, unlike Maleshkova, Pedrinaci & Domingue (2010), which analyzed the authoritative service descriptions.

This dissertation employs the RMM for evaluation of REST conformance (particularly in Section 3.1). While the WS^3 model proposed by (SALVADORI; SIQUEIRA, 2015) additionally covers Semantic Web adoption and additional service description methods, these aspects are (1) not originally considered by Fielding & Taylor (2002) and (2) largely not addressed by the approaches later evaluated in this dissertation.

2.3.2.3 Event Oriented Services (EOS)

Event-oriented services (EOSs) are services which serve events to interested clients. Events, according to Eugster et al. (2003), describe state changes or relevant facts. A client interacting with an EOS often will only take the initiative in communicating its interest in types of events to the service. Unlike other discussed service types, it is the EOS which will then take the initiative of notifying the client when events occur.

Eugster et al. (2003) lists two main interaction paradigms that fit this scenario. The notification paradigm corresponds to the Observer

---

[15] It is unclear if SOAP services were not listed in ProgrammableWeb when the study was conducted or if they were ignored. Such services would not fit the proposed classification.

pattern (GAMMA et al., 1995), and requires that both the event producer and consumer know each other network addresses and have communicated before the events to be delivered occur. Eugster et al. (2003) names these two requirements space coupling (the agents know each other address) and time coupling (events prior to the client subscription are not delivered). An interaction paradigm that avoids these two types of coupling is the Publish/Subscribe paradigm. This is done by introducing and intermediary agent between producer and consumer, which queues events and routes them to interested consumers.

The Publish/Subscribe and Notification paradigms can also be implemented by Web Services. It is important to distinguish the application of these interaction paradigms from the particular event delivery mechanism. The delivery mechanisms used by such services can be categorized as *push* or *pull*. In the former the server is able to deliver (push) messages to the client, while in the latter the client must periodically query the server (poll) in order to fetch (pull) events produced since its last poll. Even in cases where an EOS delivers events trough pull, the application logic to be executed by the client is determined by the event-oriented nature of the service: instead of fetching a single event, the client will repeatedly poll the service and process the received events, and not just the first one.

Heterogeneity of EOS implementations goes beyond the push/pull divide, into technological aspects. Technologies can be divided according to the service type used for event delivery. The following list summarizes some of such technologies:

- SOAP service:
  - **WS-Eventing** (BOX et al., 2006): implements the notification paradigm over SOAP. Pushes events to interested clients.
  - **WS-Notification** (GRAHAM; HULL; MURRAY, 2006): implements notification, also over SOAP. An extension, WS-BrokeredNotification (CHAPPELL; LIU, 2006) implements Publish/Subscribe. Events can be delivered with push or pull.
- Web APIs:
  - **Web callback**: The client provides the service an Universal Resource Locator (URL), to where the EOS will send a request to notify about a specific event or the existence of new events to be polled. This method is also known as Web hook.

  - **Atom Publishing Protocol (APP)** (GREGORIO; HORA, 2007): A server implementing this protocol acts as the intermediary in the Publish/Subscribe paradigm. Clients poll the Atom Feeds (NOTTINGHAM; SAYRE, 2005) to receive the events. The APP itself conforms to the REST architectural style.
  - **PubSubHubbub (PuSH)** (FITZPATRICK et al., 2014): Specifies push delivery mechanism for use with Atom feeds. Clients subscribe to a hub (intermediary service) to receive Web callbacks when a particular feed has new entries added. This specification does not cover the method trough which publishers notify the hub about newly added entries.
  - **HTTP 2 Server Push** (BELSHE; ROBERTO; MARTIN, 2015, p. 60): While a client must still open the connection, the server can push responses to the client, without corresponding requests sent by the client. To prevent race conditions, the server sends the client a server self-issued "promise" request preceding every pushed response.
  - **Long polling** (LORETO et al., 2011, p. 4): The HTTP server delays responding to a polling request until an event or a timeout occurs.
- Other:
  - **WebSocket**[16] (FETTE; MELNIKOV, 2011): This protocol allows bidirectional communication, allowing push delivery over the Internet. Its main feature is employing a handshake protocol that is equivalent of an HTTP protocol upgrade (FIELDING et al., 1999, p. 57).

2.3.2.4 Composite Services

In the same sense that subroutines in a program source code are used by subroutines that perform more complex functions, services can also use other services. The three fundamental properties attributed by Papazoglou (2003) to services (cf. p. 55) as well as the basic Service Oriented Architecture (SOA) and the extended SOA proposed described in that same document are directed towards the creation of *composite services*. Composite services are defined by Papazoglou (2003) as services that "access and combine information and functions from possibly

---

[16] Despite having the Web as its application scenario, this dissertation does not classify it as a Web Service (cf. Subsubsection 2.3.2.1). The rationale behind this is that the protocol is not an HTTP extension (FETTE; MELNIKOV, 2011, p. 11).

multiple services", here called component services. Unlike other service types, such definition can be taken as universally accepted. Other definitions for composite service are highly similar, such as "Web Service implemented by invoking other Web Services" given by Alonso et al. (2004).

A composite service may exist as any of the previously enumerated service types. Another common form is as a specification in languages created specially for describing composite services. Business Process Execution Language (BPEL) (JORDAN et al., 2007) is one of these languages and allows the specification of a workflow, which defines an ordered sequence of stages to complete a task using constructs common in general-purpose programming languages, such as loops, branches, parallel flows and sequences. BPEL specifications are interpreted and executed by systems called workflow engines. Under such systems, events such as a message receipt, can be related to a specific deployed process and start the execution of a new process instance. Such instances represent an execution of the composite service and are monitored by the workflow engine for exchanged messages, exceptional situations and performance. Workflow engines and BPEL processes are useful in scenarios with complex, long-running business-level processes that may involve activities performed by humans (CLÉMENT et al., 2010a; CLÉMENT et al., 2010b).

Composite service specifications can also be *ad-hoc*. Hobold & Siqueira (2012), for example, create propose a composition algorithm that represents composite service in a graph-based structure specific to the implemented prototype. In addition to *ad-hoc*, a composite service specification can also be ephemeral. Verborgh et al. (2016) use this strategy and discard the composite service specification after executing only one operation defined in it. In their algorithm, the specification is continuously replaced by an updated specification of the yet-to-be-executed composite service.

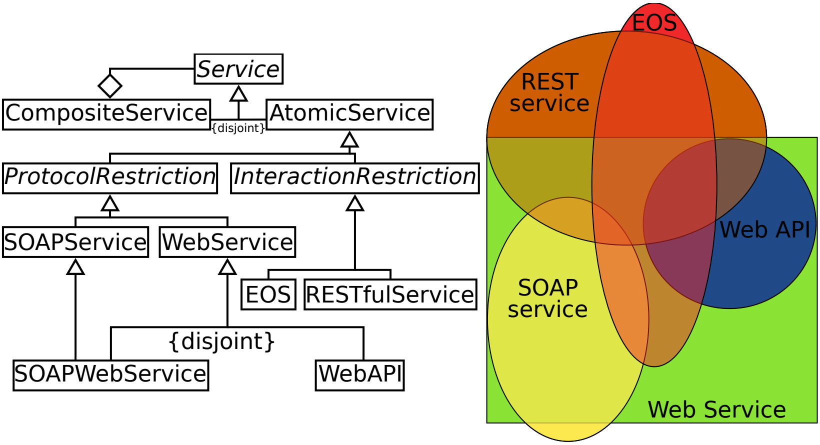
2.3.2.5 Class Hierarchy

The preceding definitions of service types, allow modeling the service types as an Unified Markup Language (UML) class hierarchy, as shown in Figure 4a. In this diagram, generalizations are, unless otherwise noted, incomplete[17] and overlapping[18]. A Service must be either Composite or Atomic, which is defined as the complement of Composite and is used only in this diagram for completeness. Atomic Services can be specialized according to their communication *Protocol* or according to richer interaction constraints that must necessarily be explained by

Figure 4 – Graphical visualizations of the service type hierarchy adopted in this dissertation.

(a) UML class diagram.

(b) Venn diagram, not proportional area and not including composite services.



Source: the author.

Interaction models. Specializations by communication protocol include SOAP services and Web Services, while specializations by interaction model include RESTful and event-oriented services. Among the Web Services specializations considered, Web APIs and SOAP Web Services are disjoint, per the definition of Web APIs.

The full range of possible class combinations can be observed by the intersections of the classes' instance sets, as shown in Figure 4b, where the depicted area of sets is not proportional to the number of existing services[18].

---

[16] Under incomplete generalization, if A is specialized by B and C, A may have instances that are instances of neither B nor C. Were the generalization complete, all instances of A must either be instances of B or C. If *A* is abstract (denoted by italics), then generalization is complete.

[17] An overlapping generalization of B and C into A allows an object to be an instance of both B and C. A disjoint generalization forbids so.

[18] For an analysis on proportion of service types, see Appendix A

2.4 SERVICE COMPOSITION

Yet again, as it was the case with "service", "service composition" also suffers the semantic polysemy phenomenon. This is not only inherited from "service", as in usage of the term "service composition" often occurs in a context where "service" has the sense of a computational element. Instead, "composition" alone presents multiple senses, most notably the sense of an artifact or the sense of a process. As a statistic, the SLR in Section 3.1 found that 43% of works using the term "composition" relied on both senses of the word.

For readability, this dissertation uses the term composition with the sense of an action, and as discussed in Subsubsection 2.3.2.4, adopts the term composite service when referring to the artifact sense. Setting the issue of artifact versus composition aside, the precise definition of the composition action deserves a finer analysis of existing definitions. The following are explicit definitions of the term obtained from within the 15 most cited papers in Scopus[19] and from papers that attempt to establish a consensus (ZHANG, 2008; BOUGUETTAYA et al., 2017).

1. Papazoglou (2003, p. 7): The term is used to denote both an action and an artifact. When used as an action, it is the action of consolidating "multiple services into a single composite service". Composite services are services that combine data and functionality from other services (PAPAZOGLOU, 2003, p. 3). In a later work (PAPAZOGLOU et al., 2008), the authors clarify that "aggregators", which create the composite service, are also responsible for its execution;
2. McIlraith & Son (2001, p.47): "composition and interoperation" aim to "perform a task, given a high-level description of the task's objective". The authors use interoperation in the sense of resolving heterogeneity issues and explicitly state that execution is part of "composition and interoperation";
3. Sheng et al. (2014, p. 219) "the process of aggregating multiple services into a single service in order to perform more complex functions". Executing the composite service that aggregates the services is explicitly listed as part of the process;
4. Zeng et al. (2004, p. 311): composition is the process of "interconnecting Web Services provided by multiple business partners according to some business process";

---

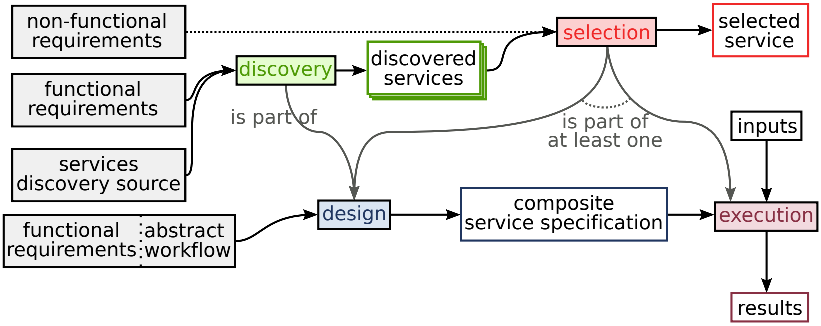
[19] Two queries were performed, one for `"service composition"` and another for `"service composition" AND NOT QoS`.

5. Berardi et al. (2003, p. 44): “Composition involves two different issues. (...) *composition* synthesis, or simply composition, is concerned with synthesizing a new composite e-Service, thus producing a specification of how to coordinate the component e-Services to obtain the composite e-Service. (...) *orchestration* is concerned with coordinating the various component e-Services according to some given specification, and also monitoring control and data flow among the involved e-Services”;
6. Dustdar & Schreiner (2005): “The process of developing a composite service”, where a composite service is a “Web Service implemented by invoking other Web Services” (ALONSO et al., 2004)
7. Singh (2001, p. 6): “using existing Web Services and building new customized services out of them.”;
8. Milanovic & Malek (2004, p. 51): Developers use composition to “solve complex problems by combining available basic services and ordering them to best suit their problem requirements”;
9. Zhang (2008, p. 66): In an editorial proposing sub-areas of interest in Service-Oriented Computing (SOC) research, the author defines the service composition action as a “way of creating value-added services or applications based on existing services”;

One notion common to all such definitions is that of combining or aggregating services, what can be translated into creating composite services. However, not all definitions (e.g., 6-9) include execution of the composite service as part of composition. Given that there are classic and current original works including (ZENG et al., 2004; LEE; LEE; WANG, 2015a) and not including (RAO; KUNGAS; MATSKIN, 2004; RODRIGUEZ-MIER et al., 2012; VERBORGH et al., 2016) execution in their concept of composition, the definition by Berardi et al. (2003) is the most comprehensive.

The definition given by (BERARDI et al., 2003) contemplates all other presented definitions, despite two minor issues. Firstly, although it has a general meaning in the presented definition of composition, the term synthesis is used mainly by works that, like (BERARDI et al., 2003), describe algorithms for synthesizing State Transition Systemss (STSs). This is also evident in a comprehensive survey on service composition by (LEMOS; DANIEL; BENATALLAH, 2015), where synthesis is one among other methods for automatic design of composite services. Such usage is inconsistent with man-made composite services and with composite services created by other methods such as planning algorithms. Secondly, in the literature, the term orchestration

Figure 5 – Service composition overview.



Source: the author.

can mean the specification of a composite service from a central point of view (PELTZ, 2003; ROMAN et al., 2005) as well as the action of executing the composite service according to the aforementioned specification (DIJKMAN; DUMAS, 2004; PAIK et al., 2017).

Given the current uses of synthesis and orchestration in service composition literature, this dissertation employs the more generic terms design and execution. Therefore, service composition is an action divided in two sub-activities: composite service design is the action of creating the specification of a composite service; composite service execution is the action of coordinating component services according to such specification. As “design” and “execution” are not technical terms, they are frequent in the literature to describe phases of the composition process or of the life cycle of a composite service. Both terms are, for example, used in (DUSTDAR; SCHREINER, 2005; BARESI; GUINEA, 2011).

Design and execution can be further detailed, as shown in Figure 5. Synthesis usually takes three inputs: a set of functional requirements or an abstract composite service specification, a source for discovery of services and, optionally, a set of non-functional requirements. The process of designing a composite service involves discovering candidate component services and selecting the most suitable candidates. Discovery and selection can also occur during execution. The following sections discuss further details.

2.4.1 Service Discovery

Service discovery, also known as matchmaking, is the process of locating services that satisfy functional requirements. (GLINZ, 2007) defines[20] a functional requirement as a requirement whose "matter of interest is primarily the expected behavior of a system or system component in terms of its reaction to given input stimuli and the functions and data required for processing the stimuli and producing the reaction." In contrast, a non-functional requirement is a required "attribute of or a constraint on a system". Glinz (2007) lists three possible non-functional requirements: requirements on performance (timing, speed, volume or throughput), on specific qualities (ISO, 2001), or constraints (which constrain the solution space beyond what is necessary for meeting all other requirements). Non-functional requirements are the concern of a later process, service selection.

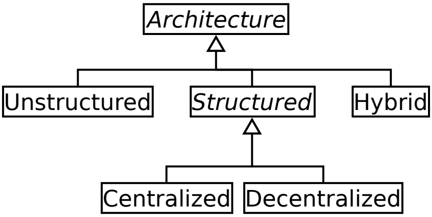
The discovery process will use one or more sources, from where information about services is obtained. Meshkova et al. (2008) reviews literature on service discovery and identified the classes of architecture displayed on Figure 6. An unstructured architecture does not differentiate between nodes that provide services and nodes that act as part of a directory, neither defines how repository information is distributed. Gnutella (KLINGBERG; MANFREDI, 2002) is an example of an unstructured architecture. A structured architecture, on the contrary, may present these two characteristics. The former (distinct node roles) is observed in Centralized architectures, such as DNS (MOCKAPETRIS; DUNLAP, 1988; CHESHIRE; KROCHMAL, 2013) and Jini (ARNOLD et al., 1999), where specific nodes answer directory queries. The latter (well-defined information partitioning) is observed in Decentralized architectures, usually based on Distributed Hash Table (DHT) as is the case of SPiDeR (SAHIN et al., 2005). Finally, characteristics of structured and unstructured architectures can be combined in Hybrid architectures, such as (NEJDL et al., 2002).

Service discovery as a research area is independent from service composition. As such, not all approaches covered by extensive surveys on service discovery (MESHKOVA et al., 2008; RAMBOLD et al., 2009; AHMED; BOUTABA, 2011) are applicable to composition. An important distinction concerns the discovered artifact. Some approaches simply discover the address of a suitable service. This is the

---

[20] As discussed by (GLINZ, 2007), there is some controversy on the precise border between functional and non-functional requirements. The ISO/IEEE 24765 (IEEE, 2010) software engineering vocabulary, for example, adopts a more lax definition of non-functional requirement as "describes not what the software will do but how the software will do it".

Figure 6 – Possible architectures for service discovery sources.



Source: the author, adapted from (MESHKOVA et al., 2008, p. 2100)

case of Eureka[21], a replicated service repository developed to be used in load balancing. A richer form of description is that used by DNS based Service Discovery (DNSSD). The service record contains a service name managed by Internet Assigned Numbers Authority (IANA)[22] and a set of key-value pairs specific to that service name.

Of particular interest to composition are discovery approaches that return *service descriptions*, machine-readable data artifacts that describe functionality and details for using the service (address, data formats, etc.). Fanjiang et al. (2017) surveys existing service descriptions, and analyzes the descriptions used by service composition algorithms. At a conceptual level, the authors generalize the description used by composition algorithms to a 6-tuple $\langle I, O, P, E, NF, ON \rangle$

1. I: The set of inputs;
2. O: The set of outputs;
3. P: The set of preconditions, i.e., logical statements that must be true before the service can be invoked;
4. E: The set of postconditions, i.e., logical statements that are expected to be true after the service is invoked;
5. NF: A set of Non-Functional Propertys (NFPs) of the service.
6. ON is an ontology that provides formal semantics for terms used in the previous elements.

Although papers presenting composition algorithms often use tuples to describe services (FANJIANG et al., 2017), actual services use more concrete forms of service description, usually tied to technical details. For SOAP services, the canonical description language is WSDL,

---

[22] <https://github.com/Netflix/eureka>

[23] <https://www.iana.org/assignments/service-names-port-numbers/>

which uses XSD to describe data. For Web APIs there are multiple alternative languages, some of which are: Web Application Description Language (WADL) (HADLEY, 2009), OpenAPI/Swagger[24], API Blueprint[25], hRESTS (KOPECKÝ; GOMADAM; VITVAR, 2008), Hydra (LANTHALER; GÜTL, 2013) and RESTdesc (VERBORGH et al., 2013). None of these is a *de facto* nor a *de jure* standard, and unlike SOAP services, some popular Web APIs, such as Twitter's, [26] have no official machine-readable description. In the case of EOS there are few dedicated description languages; service documents in APP (GREGORIO; HORA, 2007) can be considered an example. Other than through APP, Web APIs that deliver events are often described in natural language, as is the case of timeline streams in Twitter[27].

WSDL describes only the syntax of messages exchanged with the service, using XSD. In contrast, a semantic description would make the semantics of both service functionality and data semantics machine-readable. This is accomplished by describing the service data and functionality using operational ontologies. In practice, RDFS and OWL ontologies (often domain ontologies or application ontologies) are used to describe data and functionality of a service. Services themselves are described by core ontologies that cover aspects such as requests and operations. An ontology for service description can also be considered to be a language whose syntax is that of the ontology language used in its definition.

Semantic Annotations for WSDL and XML Schema (SAWSDL) (KOPECKÝ et al., 2007) extends WSDL and XSD so that WSDL elements (interfaces, operations and faults) and XSD elements (elements, types and attributes) can receive attributes linking them to concepts in an ontology. The annotation is done by adding a `modelReference` attribute with an URI identifying semantic concepts (e.g., an OWL class, property or instance). SAWSDL adds two other attributes to XSD: `loweringSchemaMapping` and `liftingSchemaMapping`. A lowering mapping specifies how to transform semantic data (e.g., RDF) into non-semantic data (e.g., XML). The lifting mapping does the reverse transformation.

There is no restriction on which concepts can be referenced using `sawsdl:modelReference`. In the case of XSD types, OWL or RDFS classes are a natural choice. However, for operations and inter-

---

[23] <https://www.openapis.org/>

[24] <https://apiblueprint.org/>

[25] <https://developer.twitter.com/en/docs>

[26] <https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html>

faces, a question of how to semantically describe services arises. There are ontologies developed to this end that can be used with SAWSDL. OWL-S (MARTIN et al., 2007) describes a service in terms of its profile (corresponding to $\langle I, O, P, E, ON \rangle$), its model (a workflow invoking other services that achieves the functionality described in the profile) and its grounding (a relation to a WSDL file or other syntactic service description). While OWL-S precedes SAWSDL, the latter can be used to replace the grounding mechanism (MARTIN; PAOLUCCI; WAGNER, 2007). Another ontology is Web Service Modeling Ontology (WSMO)-Lite (VITVAR et al., 2007), which is a simplification of WSMO (ROMAN et al., 2005). Unlike OWL-S, WSMO-Lite does not support specifying a service as a workflow and includes a class for describing NFPs.

In the case of Web APIs, syntactic languages such as OpenAPI (up to version 3.0), Blueprints and WADL contain no mechanism or extension similar to SAWSDL for relating elements of the description to concepts in a ontology. All= these languages support specifying Media Types and XML (WADL) or JSON schemas[27] (OpenAPI, API Blueprints). OpenAPI contains a `format` that can receive use-defined type identifiers, but no usage with URIs of ontology concepts is reported.

Semantic description of Web APIs is only possible with RDFS and OWL ontologies, such as hRESTS (KOPECKÝ; GOMADAM; VITVAR, 2008), Hydra (LANTHALER; GÜTL, 2013) and RESTdesc (VERBORGH et al., 2013). hRESTS models a service as a set of operations, each with an optional input message, an HTTP method and an output message. Hydra and RESTdesc, on the other hand, assume services use RDF as representation and uses this fact to describe hypermedia controls.

Hydra (LANTHALER; GÜTL, 2013) defines Web APIs in terms of supported RDFS classes and RDF properties. A special type of RDF property is introduced to represent dereferenceable hyperlinks. Listing 2 shows a fragment of an Hydra description, highlighting its main features. Line 2 states one supported class, `ex:Issue`, which as stated in line 5 supports the `ex:raisedBy` property. Such property is, as stated in line 6, a dereferenceable hyperlink to a resource of type `ex:User`. The Hydra description also describes `ex:Issue` resources as supporting the HTTP DELETE method.

RESTdesc (VERBORGH et al., 2013), on the other hand, models a service as an N3 logic (BERNERS-LEE et al., 2008) implication

---

[27] <http://json-schema.org/>

Listing 2 – Fragment of an Hydra Web API description containing an hyperlink description, in Turtle syntax.

```
ex:doc a hydra:ApiDocumentation;
  hydra:supportedClass ex:Issue.
ex:Issue a hydra:Class;
  hydra:supportedOperation [ hydra:method "DELETE" ];
  hydra:supportedProperty ex:raisedBy .
ex:raisedBy a hydra:Link;
  rdfs:domain ex:Issue;
  rdfs:range ex:User.
```

rule. In the $\langle I, O, P, E, NF, ON\rangle$ scheme abstraction described previously, the antecedent of the rule corresponds to I and P, and the rule consequent corresponds to O and E. The consequent of such rules also contains descriptions[28] of the HTTP request message and of the expected response.

2.4.2 Service Selection

There are several distinct definitions in the literature for service selection (BONATTI; FESTA, 2005; WANG; VASSILEVA, 2007; SUN; KU, 2014). However, these variations are not in disagreement, but complement each other. Service Selection can be summarized as the process of choosing a single service from each of possibly multiple sets of candidate services, based on one or more attributes of each candidate and seeking to minimize or maximize some attributes of the composite service yielded by the selected services. The candidate sets given as input to the selection process are the services discovered for each activity that is part of the composite service specification being designed or executed (if selection occurs at runtime).

Service selection is performed taking into consideration attributes of the candidate services. The vast majority of these attributes are QoS attributes, of which (RAN, 2003) provides an overview. However, it is possible that the discovery process was approximate (PAOLUCCI et al., 2002), and therefore a *matching degree* can also be taken as an attribute. In its simplest form, a service selection approach will receive a single set of candidate services with similar functionality that could be used as a particular service within a composite service. From within this candidate set, the selection approach will select a single service with the best single attribute, e.g., matching degree or average

---

[28] This description uses the HTTP vocabulary in RDF (KOCH; ACKERMANN; VELASCO, 2017)

response time. A broader overview on proposed methods for service selection can be obtained in (ALABOOL; MAHMOOD, 2013) and Jatoth, Gangadharan & Buyya (2015).

Optimization problems are problems where one must find an optimal feasible solution, i.e., a solution with maximal or minimal value attribute by an objective function (DEB; KALYANMOY, 2001). Service selection is an optimization problem, and as such there are three main aspects that determine variations of the service selection problem: locality, number of optimized attributes and support for constraints.

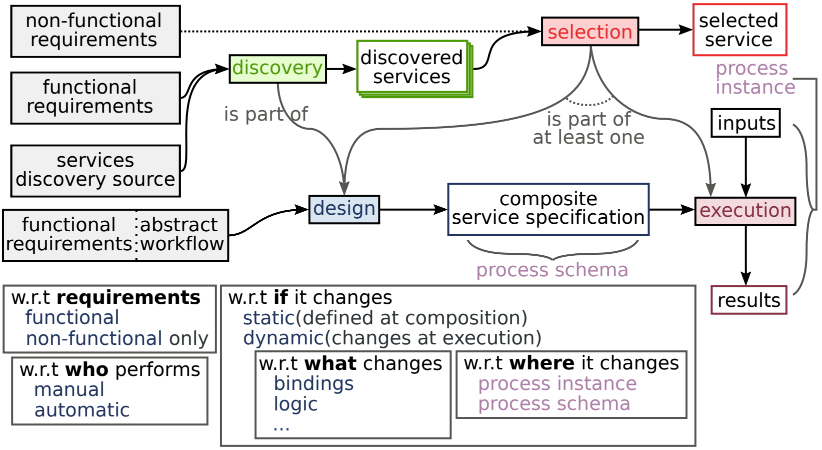
In general, global optimization seeks the minimization (analogously maximization) of an objective function $f : X \to R$ over the set of constrained solutions $C \subseteq X$, consists in finding x such that $\forall \bar{x}\, f(x) \leq f(\bar{x})$. Local optimization requires only that $\bar{x} \in C \cap N$, where N is a set values in the neighborhood of x (HIRIART-URRUTY, 2013). In service selection, the meaning of local selection is that selection occurs independently in each candidate set. Likewise, the meaning of global selection is that the selection is not independent and the best service choices within all candidate sets are done at once (ZENG et al., 2004). The selection-specific definition of local optimization can only find locally optimal solutions in the definition given by (HIRIART-URRUTY, 2013). The conditions on whether global service selection find a global optimal solution depend on the specific optimization technique employed.

An optimization problem may seek the optimization of one or multiple objectives (DEB; KALYANMOY, 2001). This is true also for service selection. Finally, arbitrary constraints might restrict feasible solutions. For example, one might impose a maximum cost and a minimum availability while optimizing for cost, availability and response time (FAN; FANG; JIANG, 2011). Such constraints can also be applied to segments of the workflow of a composite service. For example, Deng et al. (2016) considers the need to perform optical character recognition service and subsequent translation with a stricter response time requirement than the rest of the composite service.

2.4.3 Composite Service Design Qualifiers

Multiple qualifiers can be attributed to the design process. Secondary sources often elaborate on qualifiers of the design process as well of its sub-processes (GARRIGA et al., 2016; MURGUZUR et al., 2014; SHENG et al., 2014; AHMED; BOUTABA, 2011). For the purposes of this dissertation, qualifiers relating to requirements of composite ser-

Figure 7 – Main qualifiers of service design.



Source: the author.

vice, to who performs the design, and when design is performed, are the most important. Figure 7 augments Figure 5 with qualifiers of interest.

Design can be qualified according to the type of requirements taken into consideration. Functional design takes functional requirements as input, and as a consequence, determining how to coordinate the services (i.e., plan control and data flows) is part of design. Non-functional design takes control and data flows as input instead of functional requirements. In this case, design amounts to discovering and selecting the optimal services for each abstract service present in the workflow[29]. The discovery will consider functional requirements specified for the abstract services of the given abstract workflow.

The second pair of design qualifiers concerns who performs the design. Automatic design is performed entirely by an algorithm. As this qualifier is orthogonal to the type of requirements, approaches which receive an abstract workflow, such as Deng et al. (2016), are still considered automatic for using an algorithm to select services. When the composite service specification is not product of an algorithm, the term manual design is used. Approaches that focus solely on execution are classified as employing manual design.

---

[29] In the literature, it is customary to use terms such as QoS-based and QoS-aware composition, even tough the approach does not include discovery (CANFORA et al., 2005; ALRIFAI; SKOUTAS; RISSE, 2010).

The third and last qualifier pertains to the moment when design is performed. Figure 7 shows design and execution as two separate processes. Designing and executing a composite service in distinct moments characterizes a static design. When this distinction of moment is removed, and design occurs immediately before or interleaving with execution, one performs dynamic design. The notion of static or dynamic can also be applied to discovery and selection, in this case orthogonally to the remainder of the design process . If selection is the only dynamic process, only bindings to services will change. If the whole design is dynamic, then the logic (control and data flows) of the composite service is subject to change at runtime.

Dynamic design has possibly two targets: a single process instance or the process schema. Both terms originate from literature on process flexibility (MURGUZUR et al., 2014). A change on the process instance is restricted to that particular execution of the composite service. Future instances may change differently or stick to the original process schema if the composition approach makes such distinction. On the other hand, changing the process schema amounts to changing the composite service specification, affecting future process instances or already running instances if process instance evolution is supported (MURGUZUR et al., 2014).

2.5 INTERACTION MODELS

An interaction model is a set of constraints that determines which interactions and sequences of interactions with a service are valid. Constraints of the interaction model may be in any level, covering business rules (e.g., the shipment requires an address and must occur after payment) and technical details such as use of HTTP and supported authentication methods. The term interaction model is used instead of architectural style to emphasize that restrictions apply only to the interaction between service provider and consumer.

Each of the service types targeted by this dissertation imply a basic interaction model. That is, all services of that type conform to the basic interaction model. Nevertheless, each particular service of a given type may still enforce additional constraints layered on top of those of the basic interaction model.

Basic interaction models are derived from the service types. Among the three service types focused on this dissertation (SOAP, RESTful and event-oriented), the RESTful basic interaction model is the most complex. All data (i.e., resource, representation, etc.) and connecting elements (i.e., cache, resolver, tunnel, etc.) of REST are also member of the basic interaction model. Processing elements are replaced with roles: the *client*, which corresponds to the *user agent*; and the *server*, which combines the *proxy*, *gateway* and *origin server* processing elements. In other words, from the point of view of the interaction model, the Layered System is seen as indivisible.

Due to the combination of processing elements into two roles, some REST constraints have no effect in the interaction model. The uniform interface constraint is fully applicable to the basic interaction model and is the most important constraint. Among constraints of the Client-Stateless-Server style, only the need for the client to send all necessary state for processing a message has effect on the interaction model. Among cache constraints, only the labeling of cacheable/non-cacheable response representations has effect on the interaction model.

2.5.1 Protocol Model

Interoperability is classically defined as the ability of two systems to cooperate despite different languages, interfaces and execution platforms (WEGNER, 1996). Strang & Linnhof-Popien (2003), considering this definition and contemporary works (VALLECILLO; HERNÁNDEZ; TROYA, 2000; HEILER, 1995; MURER; SCHERER; WÜRTZ, 1996), propose the three-layered interoperability model[30] shown in Figure 8. Levels in this model represent aspects that could be heterogeneous among systems, and therefore should be precisely documented or made homogeneous during design of these systems.

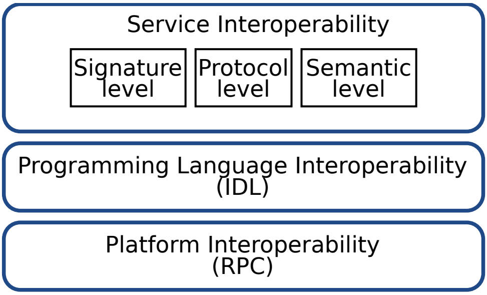
The first level, platform, solves interoperability issues related to different (and multiple) hardware. The second level, programming language, deals with heterogeneity of data and code formats that arise when different programming languages or environments (Java, C++, POSIX) are used. As solutions for the first and second level the authors list, respectively, RPC (which has several implementations) and Intermediate Description Language (IDL) approaches similar to Common Object Request Broker (CORBA) (OMG, 2015a).

The third level in Figure 8 deals with service and applications interoperability. Signature level covers the available operations, their names, parameter types and names, and passing order. Protocol refers to the valid sequences in which operations of a service can be invoked. Semantic level refers to the understanding and interpretation of in-

---

[30] Strang & Linnhof-Popien (2003) propose an extension of this model with a context level dealing with issues of ubiquitous computing. These additional interoperability issues are not of interest to the interoperability of interaction models considered in this Section.

Figure 8 – Interoperability model by Strang & Linnhof-Popien (2003).


Service Interoperability
Signature level
Protocol level
Semantic level
Programming Language Interoperability (IDL)
Platform Interoperability (RPC)

Source: adapted from (STRANG; LINNHOF-POPIEN, 2003).

formation and actions by interoperating systems and their designers. In the context of Web Services, (STRANG; LINNHOF-POPIEN, 2003) lists as solutions for uniform description of aspects in each level, respectively, WSDL (WEERAWARANA et al., 2007), Web Services Choreography Description Language (WS-CDL)[31] (BARRETO et al., 2005), and Ontologies.

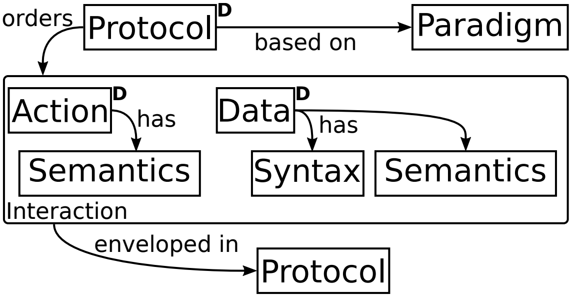
Strang & Linnhof-Popien (2003) are biased towards an RPC interaction model. As a consequence, heterogeneity issues introduced by RESTful services cannot be clearly represented with this model. The same occurs with extensions of this model, such as the one proposed in (TSALGATIDOU; ATHANASOPOULOS; PANTAZOGLOU, 2008). In order to identify sources of heterogeneity, this dissertation proposes a protocol model, which can be recursively applied. That is, the protocol is defined as a set of operations, their possible orderings and a transport protocol for invoking the operations, described by the same model. An overview is shown in Figure 9.

Protocols are usually based on paradigms, such as Client-Server, RPC or Publish/Subscribe, which determine basic patterns of interactions (e.g., request-response) over which the protocol is constructed. Each interaction can be seen as an action with a data payload. The data payload has a syntax (e.g., XML) that determines how data is represented and semantics, that determine how to interpret it. Actions, on the other hand, are considered to be finite within a protocol, and therefore do not need a syntax. However, actions have a semantic that must be shared among interacting entities. Each interaction occurs

---

[31] Strang & Linnhof-Popien (2003) cite a discontinued predecessor of WS-CDL.

Figure 9 – Protocol model. “D” stands for description of the model element.



Source: the author.

over another protocol, typically by wrapping action and data within the data payload of this lower-level protocol.

Successful interaction requires that the whole protocol, specially data syntax and semantics of action and data, be shared among interacting software systems. While this sharing can be result of tightly coupled design, services provide descriptions that can be discovered and ideally avoid tight coupling. The semantics of actions and data can also be shared through ontologies. In Figure 9 the possibility of description and discovery is represented by the letter “D”.

2.5.2 Heterogeneities in Basic Interaction Models

Below, heterogeneous aspects of the basic interaction models are discussed within the protocol model of Subsection 2.5.1. In some cases, the basic interaction model does not define any constraint on an element of the protocol model. Nevertheless, if there are proposed solutions in the context of that service type, they are discussed.

SOAP services:

- Paradigm: Client-Server and RPC[32];
- Protocol description: There is no *de facto* nor *de jure* description. WS-CDL (BARRETO et al., 2005) is an example but its use is not common. BPEL allows describing processes and binding them to roles in *partner links* (JORDAN et al., 2007). This allows the selection of a service able to assume one of the roles in a partner link;

- Action description: actions are named in WSDL and have I/O parameters. Semantics can be added through semantic annotation with SAWSDL;
- Data description: Syntax is always XML and can be further specified using XSD (BEECH et al., 2012). Semantics, likewise, can be added with SAWSDL;
- Lower protocol: SOAP, which can itself be layered on HTTP (the usual) or SMTP (LAFON; MITRA, 2007).

Web APIs:

- Action & data description: There are multiple alternative Web API description languages. Some were discussed in p. 71;
- Lower Protocol: HTTP (FIELDING et al., 1999; BERNERS-LEE; CONNOLLY, 1995).

RESTful services: As discussed in Subsubsection 2.3.2.2 (p. 61), often not all REST architectural style constraints are adopted by services. Each level in the Richardson Maturity Model (RMM) implies in additional restrictions to the protocol model from Figure 9. A proper RESTful service, when implemented over HTTP, conforms to all restrictions.

0. POX:
    - Lower Protocol: HTTP. Actions are encoded as part of the URI (if the HTTP verb is GET) or of HTTP content body.
1. Resources:
    - Data: all actions apply to resource, therefore the URI of such resource is always part of the data payload, and is properly set in the HTTP message.
2. HTTP verbs:
    - Action: There are only actions corresponding to HTTP verbs, and the semantic of the actions is in conformance with the HTTP specification. There is no more need to encode actions as part of the URI or HTTP content body.
3. Hypermedia:

---

[32] WSDL 1.1 (CHRISTENSEN et al., 2001) listed notification and solicit-response (server-initiated requests) *message exchange patterns*. These were not included in WSDL 2.0 and are forbidden in the Web Services Basic Profile (TOM et al., 2014) and in BPEL (JORDAN et al., 2007) specifications.

- Protocol description: Application protocol must be exposed to clients through hypermedia controls.

The RMM main goal is to guide development of RESTful services, and focuses on common mistakes. The REST architectural style implies some additional constraints in the protocol model that are not present when cascading the previously listed constraints for Level 0 through 3:

- Paradigm: Client-Stateless-Server. This implies that requests must include all data necessary for their processing;
- Data: servers should properly state validity of data trough appropriate HTTP headers (FIELDING; NOTTINGHAM; RESCHKE, 2014);
- Data description: HTTP content negotiation;

As this dissertation considers, for all practical purposes, that RESTful services are implemented over HTTP, an overlap exists between the basic interaction model of Web APIs and RESTful services. It can therefore be concluded that Web APIs are a degenerate form of RESTful services. Although it is not specified by the definition of Web APIs which REST constraints are violated, support for RESTful services at any maturity level also implies support for Web APIs.

Event-oriented Services:

- Paradigm: notification or Publish/Subscribe;
- Lower Protocol: Should support server-initiated interactions for event delivery or predict polling by the client.

2.6 SUMMARY OF DEFINITIONS

This chapter defined several terms which are object of some conflicts in literature. Table 3 summarizes some of such terms with their summarized definitions that will be used in following chapters.

Table 3 – Summary of service type definitions.

Term	**p.**	**Definition**
Ontology	47	Partial account of a conceptualization
Service	55	Computational elements that perform functions and are accessed through standardized interfaces, are loosely coupled and have both their definition and location discoverable
Composite service	64	A service that accesses and combines information and functions from others
RESTful service	60	Fully conforms to the REST style
Web Service	57	Is accessed through HTTP
SOAP service	57	Is accessed through SOAP
SOAP Web Service	57	Is accessed through SOAP over HTTP
Web API	59	Employs no transport over HTTP
EOS	62	Delivers events to interested clients
Service Composition	69	Action divided in two sub-activities: composite service design and execution
Composite service Design	69	Process of creating the specification of a composite service
Composite service Execution	69	Coordinating component services according to a composite service specification
Service Discovery	70	Process of locating services that satisfy functional requirements
Service Selection	74	Process of choosing a single service from each set of candidate services, based on one or more attributes of each candidate and seeking to minimize or maximize some attributes of the composite service yielded by the selected services

Source: the author

3 RELATED WORK

The related work is divided in two groups. The first group, discussed in Section 3.1, concerns original research dealing with composition of services with heterogeneous interaction models. This section corresponds to the step 1 from Section 1.3. A Systematic Literature Review (SLR) procedure is adopted in order to confirm the relevance of this dissertation goal. The second group, discussed in Section 3.2, corresponds to step 3 from Section 1.3. It covers recent automatic design approaches which provide a foundation for the proposal (Chapter 4) and experiments (Chapter 5).

3.1 HETEROGENEOUS SERVICE COMPOSITION

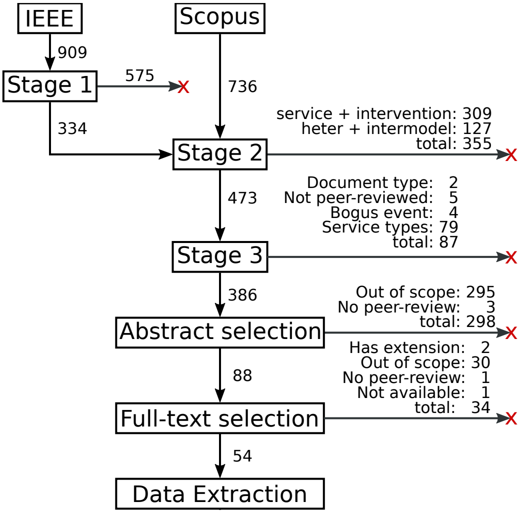
Kitchenham & Charters (2007) presents guidelines for planning and conducting SLRs, which are defined by the authors as a means for identifying, interpreting and evaluating relevant research available on a particular topic, Research Question (RQ) or phenomenon. In addition to SLR Kitchenham & Charters (2007) also mention two additional closely related literature review types: Systematic Mapping Study (SMS) and tertiary (or meta) reviews. The former aims to provide a coarse-grained overview of evidence pertaining to a topic or phenomenon. The latter is characterized by taking other reviews as input.

The choice for an SLR is justified by the scope of the single RQ of interest. Heterogeneous service composition is a narrow but not overly specific topic, which makes viable the realization of an SLR. In addition, step 1 from Section 1.3 requires deeper analysis of each primary work. These observations discard the application of an SMS. A tertiary review is also not appropriate as the topic is not as broad and as consolidated as service composition, for which there are reviews available.

3.1.1 Protocol & Execution

The RQ to be answered by the SLR is “What are the approaches for designing or executing composite services with heterogeneous interaction models, specifically those implied by EOS, SOAP services and RESTful services?”. The full protocol, designed according to the guidelines by Kitchenham & Charters (2007), is described in Appendix B. A broad overview of the protocol, which suffices for understanding the SLR results, follows.

Figure 10 – Overview of the studies selection process, cf. Appendix B.

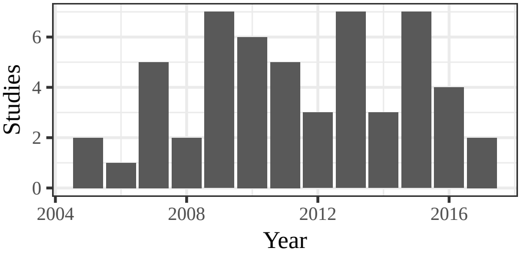


Source: the author.

Figure 10 shows an overview of the selection process described in Section B.4 with counts of studies processed in each step. The sources, IEEE and Scopus, were queried on November 19, 2017. Stages 1 to 3 are performed by a GNU Makefile script, as discussed in Section B.4. IEEE has a limitation of 15 keywords on search queries. Therefore, more relaxed queries are used on IEEE and Stage 1 consists in filtering results locally as specified in the Scopus queries. Stage 2 enforces that certain groups of keywords occur in the same sentence or keyword, i.e., “composition” and “service” must occur in the same sentence in order to characterize a match. Stage 3 performs several checks to exclude certainly irrelevant studies. Given this automation, only 386 out of the 1398 unique studies retrieved from IEEE and Scopus had their abstracts manually analyzed.

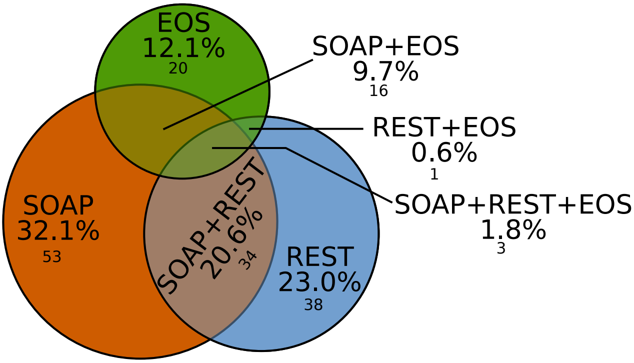
After selection by full-text, which occurred concurrently with the data extraction process, 54 papers were selected. Studies were found covering all four combinations of SOAP services, RESTful services and EOSs, with REST/SOAP heterogeneity being the most frequent. Most of these papers were published in conference proceedings, and only 13 originate from journals. The SLR protocol did not enforce a publication

Figure 11 – Number of selected studies per publication year.



Source: the author

Figure 12 – Service types considered in heterogeneous composition proposals.



Source: the author

time window. As shown in Figure 11, selected studies were published between 2005 and 2017.

3.1.2 Categorizations

Figure 12 shows the number of studies whose proposed methods target the three service types and their combinations. In this figure, like in most of this section, studies are categorized under REST by their claims alone. The precise level of REST architectural style support is evaluated apart. SOAP services are the most frequently targeted service type, followed by RESTful services. SOAP+RESTful heterogeneity is the most studied pair, followed by SOAP+EOS. In comparison, RESTful+EOS heterogeneity is a niche topic, to the point that only one study (LIECHTI et al., 2015) tackles exclusively REST+EOS heterogeneity.

The proposals made by studies for supporting heterogeneity of service types in service composition were distributed in six top-level categories. The following list presents definitions of these categories.

Common description A single description language is used to describe services of different types. The rationale for this method is that some heterogeneous aspects can be ignored during composition. All proposals assume one of the following three forms:

- A **semantic description language** (cf. p. 72);
- A **syntactic description language**, without machine-readable semantics;
- A **metamodel** (SEIDEWITZ, 2003), that is used to map transformations between different description languages.

Proxies A new service of type x is created. This service redirects all interactions with its clients to a service of type y (and *vice-versa*). This category can be broken down by the service type of the proxy service;

Middleware Actions such as invoking a service or receiving a message can be seen as abstract functionality realized differently according to the current environment or platform. Middleware techniques create such abstractions and solve heterogeneity by delegating realization of such actions to components specific to each service type;

Workflow languages A language for procedural specification of composite service is used to solve heterogeneity issues. Approaches subdivide into categories according to the origin of the language:

- An **existing language**, such as BPEL (JORDAN et al., 2007), can provide support for some heterogeneous aspects;
- **Language extensions** add capabilities to a language, allowing services of different types to be invoked, or description of additional interaction paradigms;
- An entirely **New language** can be proposed.

Event processor A composite service can be specified in terms of events and executed in an event processor;

- One common type of event processor architecture is **Event-Condition-Action (ECA)** and variants. In ECA, events

Table 4 – Categories of heterogeneous composition methods proposed by selected studies.

Top-level Method Category	Subcategory	Studies
Common Description	Syntactic	6
	Semantic	3
	Metamodel	1
Direct Implementation	–	2
Event Processor	ECA	5
	Event Processor	4
Middleware	–	20
Proxy	SOAP	7
	REST	4
	EOS	3
	Other	2
	Proxy	1
Workflow Language	Existing	12
	Extension	8
	New	7

Source: the author.

trigger the evaluation of conditions, that if satisfied, trigger the execution of actions.

Direct implementation Since a composite service can be designed as any other service of any type, a developer can manually implement a composite service that uses heterogeneous services;

An overview of the amount of selected studies according to categories of methods can be seen in Table 4. As a study may propose a combination of methods, or more than one method, the sum of category counts is greater than the number of studies. Workflow languages and middlewares are the most frequently proposed methods, together accounting for 33 of the 54 studies. These two categories are also highly correlated, given that many studies employ a middleware for implementing workflow engines.

Another categorization created from selected studies concerns specifically with violations to the HATEOAS constraint:

Ignored The study ignores the HATEOAS constraint in the conception and evaluation of the proposed methods. Two outcomes are possible:

- Most studies leave the constraint **unsupported**;
- Some approaches ignore HATEOAS, but their application has **no effect** on supporting the constraint.

Static The approach executes a predefined workflow. Some details of this workflow, such as URIs of HTTP requests is obtained at run time from hypermedia controls. Although hypermedia controls are considered, the application protocol is not driven by them;

Unsupported A final possibility is that a study explicitly states that HATEOAS is not supported.

Finally, the Richardson Maturity Model (RMM) provides four levels of REST architectural style conformance for services, which are described in p. 61. When evaluating selected studies against RMM (Section B.6), some studies provided insufficient evidence to evaluate their maturity level. These were classified as **unclear**. Other studies made claims that imply a particular maturity level, yet provided no evidence. These studies had their claims taken seriously, but received an **(unclear)** tag.

The complete list of the 54 selected studies, including their method categories, their RMM level and the classification of HATEOAS violation can be found in Table 19 in Appendix C. No selected study fully supports the HATEOAS constraint. Therefore, no study achieves the *hypermedia* level in RMM.

3.1.3 Method Summaries

For brevity reasons, summaries of representative higher quality studies for each category of methods are presented. Such summaries are structured in order to provide a complete overview of the heterogeneity support method, including its applicability scenarios.

Common description: Roman et al. (2015) present the use of the WSMO-Lite ontology for describing heterogeneous services. They demonstrate that such usage allows one to create algorithms able to design composite services from a goal and a set of available services described with the ontology. The execution of such composite service is not in the scope of the paper. The ontology abstracts differences between services by employing a Minimal Service Model, where REST and SOAP services are both viewed as sets of operations, each consisting of an input and an output message. Descriptions in WSMO-Lite originate from processing semantic annotations in service description languages, such as SAWSDL for SOAP and hRESTS for RESTful services.

The handling of hypermedia controls in (ROMAN et al., 2015) is explicit, but still insufficient for full HATEOAS support. Even though the paper does not describe a process language, the extraction of semantic description from annotated service descriptions and the composite service design algorithm imply static support for HATEOAS. The authors also point to tools that assist human specialists for creating descriptions (MALESHKOVA; PEDRINACI; DOMINGUE, 2009). The tool in particular for description of RESTful services focuses on describing operations, without relation to hypermedia controls. As a consequence, services described using this tool also lack a proper description of hypermedia control that could be offered by the service.

Common descriptions need not be take the form of an ontology (e.g., OWL-S, WSMO) nor the form of traditional service descriptions (e.g., Hydra, WSDL). Fokaefs & Stroulia (2013) define a metamodel (SEIDEWITZ, 2003) to which elements of WADL and WSDL (both XML-based languages) are mapped. This allows translation between WADL and WSDL descriptions. The authors suggest this could be used to generate *proxy* services. The paper is unclear as to how WADL URI templates (GREGORIO et al., 2012) are represented in the metamodel (and in WSDL), which restricts RMM evaluation to *resources-unclear*. HATEOAS is ignored.

All proposed common description methods focus on SOAP and REST heterogeneity. The main use of common description is to abstract heterogeneity during composite service design. As drawbacks, heterogeneity remains a problem during execution (requiring proxies or middlewares) and this method alone is not able to guarantee HATEOAS conformance.

Proxies: Giorgio, Ripa & Zuccalà (2010) propose using SOAP and RESTful services as proxies to access services of the other type. This proposal is made in the context of replacing failed component services in a composite service. To achieve this, a common description is used to describe both service types. The common descriptions allows dynamically generating a proxy so that the service client only needs to have the URI of the replacement service updated.

There are also proxying approaches involving other types of services. Braubach & Pokahr (2013), in the context of Multi-Agent

Systems (MASs), propose exposing agent functionality as SOAP or RESTful services. The reverse is also discussed: exposing services as agents in the agent platform. Blumbergs & Kravcevs (2011) employ an EOS proxy, implemented using WS-Notification (GRAHAM; HULL; MURRAY, 2006), that allows integrating a Complex Event Processing (CEP) engine into BPEL processes.

Middleware: Pico-Valencia & Holgado-Terriza (2016) propose an architecture, based on the JADE MAS platform (BELLIFEMINE; POGGI; RIMASSA, 1999), for dynamically generating behavior classes that make an agent interact with Web Services. The agents interact among themselves to negotiate and achieve goals by composing the services. All service types are considered as a set of operations, and an agent behavior can only invoke a single operation. In the case of RESTful services, this means invoking a single method on a single resource.

Bromberg et al. (2011a) propose the *Starlink* middleware, which is the only study to focus on heterogeneous application protocols. In this framework, services have their application protocols modeled by state machines. Interaction with a service is done according to the state machine over a lower-level communication protocol. SOAP and REST are considered examples of such protocols. The middleware weaves the state machines of two services with equivalent functionality but different application protocols according to state and transition mappings and translates messages during interaction.

The two middleware approaches discussed belong to a minority of studies (8 out of 20) that do not employ the middleware technique to implement workflow engines. The other 12 approaches that combine middleware and workflow languages are left for discussion under the category of Workflow languages.

Workflow languages: Georgantas et al. (2013) propose the integration of three interaction paradigms (EUGSTER et al., 2003): request-response, tuple space and Publish/Subscribe. The study proposes a model for each paradigm and then generalizes all models in a Generic Application (GA) model which contains only send (*post*) and receive (*get*) primitives applied on messages. This latter model is used to implement a workflow engine capable of executing services that each conform to one of the three interaction paradigms. A specific workflow language with constructs for all three interaction paradigms is proposed. The workflow engine transforms the workflow specification into a workflow internally described only according to the GA model. This approach does not consider RESTful services, and due to static specification of the workflow, would not fully support HATEOAS if extended to consider them.

Support for EOSs is often achieved using a workflow language (by 13 out of 20 studies that support EOS). BPEL activities – such as `<receive>`, `<pick>` and `<onEvent>` – allow a process to wait or react to notifications sent by other processes or services. A total of 9 studies employ only BPEL for integration with EOSs. Highly influential examples are (AMNUAYKANJANASIN; NUPAIROJ, 2005) and (LIN; LIN, 2007).

Two studies propose support EOS mainly through BPEL extensions: (JURIC, 2010) and (SRBLJIĆ; ŠKVORC; SKROBO, 2010). Both add extensions that increase programmer productivity. The first study adds the concept of events, event sources and event sinks to WSDL and a trigger action to BPEL. The latter adds the notion of message queues and Publish/Subscribe brokers and proposes a new workflow language.

A total of five studies propose extensions to the BPEL language for composing RESTful services. The earliest and most influential is the one by Pautasso (2009), which creates new activities for each HTTP method, allows extracting hypermedia controls for determining IRIs of subsequent requests and allows defining RESTful services using BPEL itself. Haupt et al. (2014) extend this general approach addressing details, such as content negotiation and HTTP caching headers. Both achieve *static* HATEOAS support.

Three studies attempt to support RESTful services using existing workflow languages. Thakar, Tiwari & Varma (2016) and Wu, Lin & Chen (2013) describe RESTful services using WSDL and compose them using BPEL. As a result, both achieve the lowest maturity level of RMM (i.e., *POX*) as their WSDL descriptions cannot describe different resources on a server. Bergweiler (2015), on the other hand, uses Business Process Model and Notation (BPMN) (OMG, 2011), but retains WADL descriptions of RESTful services. However, insufficient details are provided to certainly affirm that protocol level in RMM is achieved.

Five studies propose new workflow languages. Three present languages highly similar to BPEL (IM; JEONG, 2016; SUPAVETCH; CHUNITHIPAISAN, 2011; VASKO; DUSTDAR, 2007). Hang & Zhao (2013) present a tool for visually designing composite services by connecting I/Os of services. Maximilien et al. (2007a) also connects service by the I/Os, but do so using a Domain-Specific Language (DSL), which allows converting data formats as necessary.

Event processor: Ardissono et al. (2009) is an example of event-driven composition. Each service is wrapped by a component, called “Communication Interface” (CI), that mediates communication between the service and the main coordinator. The coordinator is essentially a message router that delivers events to which the CIs have previously subscribed. In this architecture, each CI monitors the context and controls its wrapped service, invoking operations when synchronization information and data to do so becomes available. Another responsibility of the CI is to forward events and results from the wrapped service to the coordinator.

Liechti et al. (2015) describe an Event-Condition-Action (ECA) system, named iFLUX, implemented as a Web API without hypermedia controls. EOSs must push events to the system using HTTP POST. The description used for action targets also restricts RESTful services to use the POST method and does not include URI templates (GREGORIO et al., 2012), which restricts RMM level to *POX*.

Direct implementation: (BAGLIETTO et al., 2010) identifies four interaction patterns in the context of Telecom services and proposes that they can be fulfilled using BPEL or by directly implementing the patterns using Java. The authors list direct implementation merely as an alternative, and instead focus on BPEL. (DREWNIOK et al., 2009) is an evaluation paper that compares four different implementations for an event-driven composite service. One of the direct implementations is a SOAP service that receives notifications and notifies other services as specified in the workflow. The second implementation consists in making each SOAP service directly invoke the next service in the workflow. Both implementations by (DREWNIOK et al., 2009) introduce coupling between services, but in experiments presented lower latency in event notification than a BPEL engine.

The two selected studies that propose this method are concerned only with SOAP and EOS heterogeneity. Nevertheless, the method is, by its definition, applicable to any heterogeneity scenario. Its drawback is lack of generality (solutions are not reusable) and considerable effort. HATEOAS can be supported by using automatic composition (VERBORGH et al., 2016), tools such as the one described in (SALVADORI et al., 2017) or by executing the composite service through human interaction via web browsers.

3.1.4 Service Type versus Methods

Three types of relations between a methods and service types were identified. A method *supports* a service type if its application does not imply in violating the interaction model imposed by that service type. The method *enables* a service type if applying that method alone to a composition system or architecture that currently supports only services with types other than the one evaluated, allows services of that type to be used as component of composite services without violating its interaction model. As an example, consider a system that composes SOAP services: simply adding a common description layer or an event processor will not enable support for RESTful services. Finally, the use of a method with a service type was *observed* if some selected study employed that method as part of a solution supporting the service type. For the purposes of filling Table 5, a workflow language is considered to be accompanied by a workflow engine.

RESTful services concentrate most limitations, in particular due to insufficient or nonexistent support for the HATEOAS constraint. In Table 5, P stands for "up to *Protocol* level in RMM", S stands for "only *static* support for HATEOAS", and N stands for "no effect on HATEOAS". Direct implementation is the only method found in this SLR that could theoretically enable heterogeneous composition with RESTful services.

One method is incompatible with HATEOAS. Workflow languages imply the interaction is driven by the workflow, not by hypermedia controls. Non-RESTful Proxies hide the RESTful nature of the service and therefore imply violation of HATEOAS, which could only be supported if the proxies were themselves RESTful. In this latter situation, a method that enables RESTful services composition could be combined with the proxying approach. Only (LOMOTEY; DETERS, 2013) takes this approach, but no method is proposed in addition to proxying (resulting in the observed N in Table 5).

Figure 13 shows, for each top-level method, which elements of the protocol model from Subsection 2.5.1 have their heterogeneities supported. From this overview, these methods can be roughly sorted according to their heterogeneity handling power, from the most powerful to the least powerful method:

1. Direct implementation;
2. Proxy;
3. Workflow language or common description combined with middleware;
4. Common description;
5. Middleware.

Direct implementation and proxy solve heterogeneities from any source within the protocol model. For the latter only a subset of the

Table 5 – Supported, enabled and observed service types for each method.

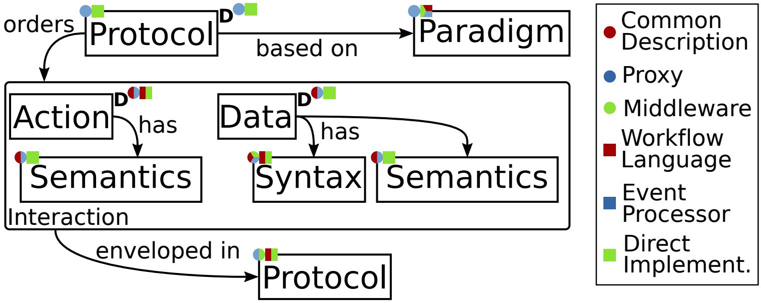
	Supported			**Enabled**			**Observed**		
Method	**SOAP**	**REST**	**EOS**	**SOAP**	**REST**	**EOS**	**SOAP**	**REST**	**EOS**
Common description	✓	✓	✓				✓	S	
Proxy	✓	✓	✓	✓	P		✓	PN	✓
Middleware	✓	✓	✓	✓	P	✓	✓	S	✓
Workflow language	✓	S	✓	✓	S	✓	✓	S	✓
Event processor	✓	✓	✓			✓	✓	P	✓
Direct implementation	✓	✓	✓	✓	✓	✓	✓		✓

Source: the author.

heterogeneities can be handled at any time. Only EOS proxies can support interaction paradigm heterogeneity. Protocol description heterogeneity, due to HATEOAS, is only supported (but not enabled) by RESTful proxies. Protocol heterogeneity was not demonstrated with proxies in the SLR, but is theoretically possible (BROMBERG et al., 2011b). Workflow languages (again assuming existence of the corresponding workflow engine) may add support for heterogeneities in action description and on interaction paradigm (EUGSTER et al., 2003) (e.g., BPEL supports for events).

A common description is able to handle only heterogeneities in the description of syntax and semantics of actions and data. Protocol description is not included because description of hypermedia controls does not guarantee their consideration during execution. Middlewares only deal with heterogeneities in the lower level protocol and on data syntax. The common description and middleware methods can be combined to achieve a level of interaction model heterogeneity support similar to that of workflow languages. Finally, the event processors method only add support for heterogeneity between RPC and event-oriented paradigms.

Figure 13 – Sources of heterogeneities whose support is enabled by each of the methods.



Source: the author.

3.1.5 Composite Service Design

Three attributes of the composite service design process were evaluated: automatic and functional design; dynamic and functional design; and dynamic selection of services.

Automatic design is not common in the context of heterogeneous service composition as only three studies propose this (LALIWALA; CHAUDHARY, 2006; ROMAN et al., 2015; UPADHYAYA; ZOU, 2012). Laliwala & Chaudhary (2006) propose an ECA architecture supporting SOAP services and EOS. Actions of a triggered ECA rule are input to an algorithm that computes the closure of preconditions to be satisfied and generates a BPEL process that is executed as consequence of the ECA rule. (LALIWALA; CHAUDHARY, 2006) is also the only selected study to propose dynamic functional design.

Roman et al. (2015) and Upadhyaya & Zou (2012) employ common descriptions of RESTful and SOAP services to create heterogeneous composite services. The algorithm used in (ROMAN et al., 2015) was originally proposed by (HOFFMANN et al., 2008) and plans a linear invocation of services described by logical pre- and post-conditions to achieve a logical goal state. Upadhyaya & Zou (2012) employ planning using Hierarchical Task Network (HTN) decomposition (EROL; HENDLER; NAU, 1994). Services are associated to concepts (e.g., *vacation*), which are decomposed into more specific concepts (e.g., *vacation* involves *hotel* and *travel*) using datasets such as DBPedia (AUER et al., 2007) and WordNet (MILLER, 1995). These two studies per-

Table 6 – Evaluation per service types.

Service Types	Evaluation Method	Count
REST+EOS	Case Study	1
SOAP+EOS	Informed Argument	10
	Architecture Analysis	5
	Scenario	7
	Case Study	1
SOAP+REST	Informed Argument	28
	Architecture Analysis	4
	Scenario	14
	Case Study	1
	Controlled Experiment	1
SOAP+REST+EOS	Informed Argument	3
	Scenario	1

Source: the author.

form composition strictly before execution, and therefore only consider HATEOAS statically when the workflow is planned.

The third aspect, dynamic selection, is considered by both Giorgio, Ripa & Zuccalà (2010) and Rajaram, Babu & Ganesan (2015). However, both merely set their proposals in a context where a workflow engine dynamically selects services, without proposing selection methods.

3.1.6 Evaluation Methods

The evaluation presented by selected studies, with respect to how well heterogeneity is supported, is generally unsatisfactory. This is correlated with the general failure in fully supporting the HATEOAS constraint. Table 6 shows the relation between combinations of service types and the evaluation methods listed in (HEVNER et al., 2004), while Table 7 shows the attributed quality for each evaluation method (Section B.5).

22 studies adopt only the *informed argument* (HEVNER et al., 2004) method, describing the heterogeneity problem, the proposed solution and related work. From this group, only (AMNUAYKANJANASIN; NUPAIROJ, 2005) achieves *mostly credible* status, as the

Table 7 – Quality per evaluation method.

Evaluation Method	Evaluation Quality	Count
Architecture Analysis	Insufficient	2
	Mostly Credible	4
	Credible	3
Case Study	Insufficient	1
	Credible	2
Controlled Experiment	Mostly Credible	1
Informed Argument	Insufficient	34
	Mostly Credible	3
	Credible	4
Scenario	Insufficient	15
	Mostly Credible	3
	Credible	4

Source: the author.

study supports a specific type of EOS and the argumentation is detailed and explicit (not simply a description of the approach). The remaining 16 studies of the 38 employing informed argument also employ other methods, and when achieve higher quality evaluation own this to these additional methods. Informed argument is not an adequate method as it is vulnerable to bias that can lead to important aspects of interaction models, such as HATEOAS, being ignored. It is also vulnerable to bias already present in the particular research area from which related works were selected.

Scenarios, which Hevner et al. (2004) classifies together with informed argument under descriptive evaluation methods, still suffer from bias as such scenarios are chosen and implemented by their authors. A total of 17 studies employ this method, 15 in combination with informed argument. Only (BAGLIETTO et al., 2010; BLUMBERGS; KRAVCEVS, 2011; SRBLJIĆ; ŠKVORC; SKROBO, 2010), employing both methods and targeting specific scenarios of SOAP and EOSs integration, present credible evaluations. Furthermore, no evaluation used multiple scenarios, or variations of a scenario, to determine limitations of the heterogeneous composition approach[1].

A case study avoids the bias introduced by the authors defining the scenario. However, there is still a possibility of bias due to the selection of which case study is to be conducted. Nevertheless, case studies allow researches to observe phenomena related to the evaluated artifact that only occur in real business environments. Only Wu et al. (2012) employed a case study, but with ambiguous results related to HATEOAS. The study proposes a SOAP proxy that provides at most protocol level in RMM, but violates HATEOAS. However, the case study employs *JOpera* (PAUTASSO, 2009) supporting HATEOAS statically.

A single study (UPADHYAYA; ZOU, 2012) employs a controlled experiment as part of heterogeneity support evaluation. Since the study involves creation of RESTful proxies by identifying resources within WSDL descriptions, an experiment evaluated the accuracy of the proposed resource extraction method. This experiment does not properly describe the origin of WSDL files and does not evaluate recall, as would be common. Albeit not a complete evaluation of heterogeneity support, this is one of the most well evaluated proposals.

The evaluation method most related to heterogeneity support is architecture analysis. Yet, only 9 studies employ it with credible quality being achieved only for the scenario of SOAP and EOS heterogeneity (GEORGANTAS et al., 2013; MINGUEZ; ZOR; REIMANN, 2011; JURIC, 2010). In the case of SOAP and RESTful services heterogeneity, Vasko & Dustdar (2007) employed this method, but ignored HATEOAS. Other studies in the same category (HAUPT et al., 2014; ROMAN et al., 2015; PAUTASSO, 2009) employ this technique, but their proposals are only able to support HATEOAS statically.

3.2 AUTOMATIC SERVICE COMPOSITION

Automatic service composition has been a topic of interest nearly for as long as the concept of service composition has emerged. A classical work proposing functional service composition is (MCILRAITH; SON, 2001). The authors first describe the DAML-S Web Service description ontology, which later evolved into OWL-S (MARTIN et al., 2007). Automatic composition is achieved using a reasoner (ConGolog (GIACOMO; LESPÉRANCE; LEVESQUE, 2000)) that realizes com-

---

[1] The closest case is of (BAGLIETTO et al., 2010), which uses a single scenario to compare two approaches to support heterogeneity.

posite processes described in DAML-S considering user preferences and constraints. A similar approach, using the SHOP2 (NAU et al., 2003) HTN planner, is proposed in (SIRIN et al., 2004).

Peer (2004), in contrast to the approaches in (MCILRAITH; SON, 2001; SIRIN et al., 2004), converts WSDL descriptions with semantic annotations in an ad-hoc language to Planning Domain Definition Language (PDDL) (MCDERMOTT et al., 1998), which is understood by several planners (HOFFMANN; NEBEL, 2001; DO; KAMBHAMPATI, 2003). Unlike OWL-S, this approach does not contemplate the definition of composite services. A similar conversion to PDDL approach is adopted in (KLUSCH; GERBER, 2005), where OWL-S descriptions are used, supporting hierarchical tasks. The approach uses version 2.1 of PDDL (FOX; LONG, 2003), which covers reasoning about duration of actions and conditions that hold during the execution of actions. However, the approach considers, as far as planning is concerned, that service invocation has no duration.

Another general approach for service composition is through graphs. Hobold & Siqueira (2012) model the composition problem as finding minimal cost paths in a graph, in which each service operation (SAWSDL is used as description) is represented as a node, and nodes are linked if outputs of the source node satisfy at least one input of the target node. By finding a path for each desired output, one can build a workflow for the composite service that produces all desired outputs. The authors also propose storing compositions in a repository for later reuse. The performance of the algorithm is evaluated with a controlled experiment designed by the authors themselves.

Rodriguez-Mier et al. (2012) propose the ComposIT algorithm, which reduces composition to finding a single minimal-cost path in a graph. This graph uses sets of services as nodes, which are placed in layers. A node a_i from layer i has an outgoing edge to node a_{i+1} if, and only if, the set of input parameters for a_{i+1} can be satisfied with outputs from a_i or from any node in layer $i-1$. The algorithm itself is an A* (HART; NILSSON; RAPHAEL, 1968) search on the reverse (edges directions) graph. The novelty of the approach lies in optimizations that discard services and nodes from the search graph. Evaluation in (RODRIGUEZ-MIER et al., 2012) is done with the Web Services Challenge (WSC)'08 benchmark (BANSAL et al., 2008). An extension (RODRIGUEZ-MIER et al., 2016) of this research employs the common description proposed by Roman et al. (2015). However, the focus is in service discovery and no heterogeneity issue is addressed.

The ComposIT algorithm proposed by Rodriguez-Mier et al. (2012) is extended by Chattopadhyay, Banerjee & Banerjee (2015) with a greedy approach, i.e., it employs local selection during composite service design. In addition to service count and solution length, the approach also considers QoS attributes such as response time and throughput. The paper reports composition time to be under 1 millisecond, but three issues arise involving these results. First, it is not clear exactly which part of the composition process is measured. Second, as will be further discussed in the context of experiments (Subsubsection 5.3.1.4), it was not possible to replicate results, since neither the prototype implementation nor the modified evaluation dataset are provided. Furthermore, results attributed to the ComposIT algorithm are inconsistent with those reported on (RODRIGUEZ-MIER et al., 2012). Results disagree in scale and in relationship between the problems that constitute the benchmark (e.g., problems 3 and 5 are reported as slower than problems 6 and 8).

There are composition approaches specific to RESTful services, beyond approaches like Haupt et al. (2014), Pautasso (2009), Maximilien et al. (2007b) discussed in Section 3.1. Garriga et al. (2016) provide a review on manual and automatic composition approaches for RESTful services. Here we limit ourselves to some specific recent studies proposing automatic RESTful services composition.

Two automatic composition approaches are listed in (GARRIGA et al., 2016). Zhao & Doshi (2009) describe services using an ontology and Semantic Web Rule Language (SWRL) (HORROCKS et al., 2004). First the paper describes formal semantics for all HTTP methods as well as for SWRL rules, in the context of situation calculus (MCCARTHY; HAYES, 1969). The intent of the paper is to automate composition through planning in situation calculus (similarly to (MCILRAITH; SON, 2001) and subsequent works). The second approach, by Zhao et al. (2011), formally specifies composite services using π-calculus (MILNER, 1999) and employs a mapping between π-calculus and Linear Logic (GIRARD, 1987) in order to employ reasoners for the latter to automate design of composite services. Zhao & Doshi (2009) ignore HATEOAS. (ZHAO et al., 2011), like others in Section 3.1 (HAUPT et al., 2014; PAUTASSO; WILDE, 2011), consider hypermedia controls in the workflows of composite services, but the workflow itself does not change during execution.

Verborgh et al. (2016) present a composition algorithm, named Pragmatic Proof, that properly takes HATEOAS into account. It employs RESTdesc descriptions (VERBORGH et al., 2013) which describe the service as implications in a logic called N3 logic (BERNERS-LEE et al., 2008). The authors use an N3 logic reasoner, such as EYE (VERBORGH; ROO, 2015), to prove a formula using the RESTdesc descriptions (which are implication rules). The proof gives the specification of a composite service that achieves the proved formula. To support HATEOAS, the goal formula is proven after every interaction with services, incorporating the results of the previous interaction. In this way, the availability of hypermedia controls is taken into account and changes the workflow executed at runtime. Evaluation of performance is done with experiments designed by the authors themselves and focuses on required times by the reasoner to obtain the first proof. However, the complete re-plan after each interaction, discarding already proven facts, is a potential performance bottleneck.

Since an EOS is typically implemented over technologies that would classify it as a SOAP or RESTful service, most research that covers EOS "composition" adds EOS support to SOAP or RESTful composition techniques. Section 3.1 already provided a discussion of such works. An implicit case of pure EOS composition (i.e, there is no dependency on SOAP or RESTful service composition) occurs in CEP (CUGOLA; MARGARA, 2012), where the focus is to combine lower-level events into higher-level events (e.g., suspicious credit card transactions combined into a credit card fraud event).

3.3 CONCLUSION

The SLR from Section 3.1 has shown that composition of services with heterogeneous interaction models remains an open issue when SOAP services, RESTful services and EOSs are considered. In particular, despite the considerable amount of existing studies, there is no proper support for HATEOAS. The small number of automatic composition approaches is related to this lack of support for HATEOAS, since as discussed in (VERBORGH et al., 2016), HATEOAS support requires automatic composition. Furthermore, studies that currently target the three service types have weak evaluations and low RMM maturity levels (either POX or unclear evidence of *protocol* maturity).

The main outcomes of the SLR are summarized by Table 5 (relating identified methods for heterogeneity support to the three service types) and Figure 13 (relating the same methods to elements of the protocol model from Subsection 2.5.1). No method is able to support RESTful services, except for *direct implementation.* However, by its very definition, this method is not a general solution and no evidence of its application to RESTful services was found by the SLR. Furthermore, there is no combination of methods able to support all heterogeneities with origin in all elements of the protocol model.

The additional review on automatic composition shows that HATEOAS can be fully supported (VERBORGH et al., 2016) without resorting to direct implementation. Nevertheless, some approaches targeting RESTful services repeat HATEOAS violations observed in the SLR (ZHAO; DOSHI, 2009; ZHAO et al., 2011). This overview also shows a historical trend in research on SOAP-oriented automatic composite service design moving away from highly expressive algorithms towards simpler, but faster, I/O-based algorithms.

4 PROPOSAL

As shown by Chapter 3, composition of SOAP services, RESTful services, and EOSs remains an open challenge. This chapter proposes a software architecture, named Unserved[1], as a solution. This architecture employs a composition algorithm as a method for supporting heterogeneities introduced by the basic interaction model of RESTful services (HATEOAS) and by the basic interaction model of EOSs, an approach that was not observed among studies selected in the SLR from Section 3.1.

4.1 HETEROGENEOUS COMPOSITION ARCHITECTURE

Perry & Wolf (1992) define a software architecture as having data, processing and connecting elements with form and a rationale. Form consists of properties of elements and relationships among them. Properties constrain viable alternatives and the implementation of elements. Relationships constrain where elements are placed and how they interact. The rationale justifies the choices made on the design of the architecture. This section presents the *Unserved* architecture for service composition supporting heterogeneous interaction models.

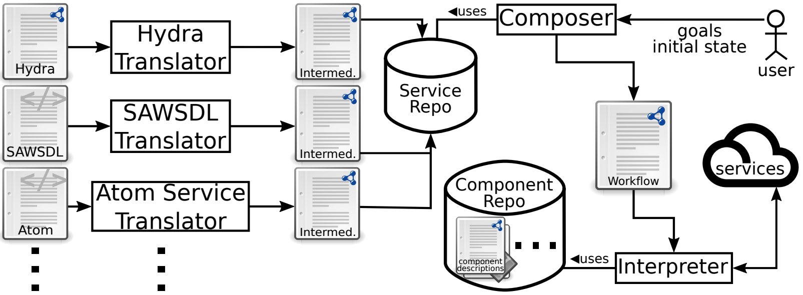
Figure 14 shows an overview of the data flow in Unserved. Existing service descriptions are translated into an intermediary description and stored in a repository. In order to achieve user-provided goals, a composer element discovers and selects services to design a composite service as a workflow. This workflow is interpreted by the interpreter, which uses components from a component repository when doing tasks such as sending requests to services.

In the following sections, data (Subsection 4.1.1) and processing elements (Subsection 4.1.2) are discussed in detail, including weighted properties, relationships and rationales. The data flow illustrated by Figure 14 is expanded as elements are described. Unserved was designed to support a single process: service composition, of which processes such as discovery and execution are parts (as defined in Section 2.4). The process view of Unserved is described in Section 4.2. Connecting elements (and connector view) have no dedicated section as all connection elements are procedure calls and parameters, in principle, restricting the Unserved architecture to a single operating system process.

---

[1] The name Unserved originated from an unpublished previous version of this research, where it was an acronym for unSERVED is Not a SERVicE Description and named what here is known as intermediary description.

Figure 14 – Overview of the service composition architecture. Rectangles and cylinders denote processing elements and documents denote data elements. Lines indicate association between processing elements. Arrows indicate the direction of data flow.



Source: Adapted from (HUF; SALVADORI; SIQUEIRA, 2017).

4.1.1 Data Elements

Unserved contains five data elements:

1. Service descriptions
2. Service data
3. Intermediary descriptions
4. User goals
5. Workflow
6. Component Descriptions

Service descriptions, which were discussed in Subsection 2.4.1 are one of the main inputs of Unserved. While SOAP services have WSDL as the main service description language, RESTful services and Web APIs as well as EOS have no such widely accepted description language. The SLR in Section 3.1 revealed works that proposed a common description language for all types of services. However, no common description proposal covered EOS (as shown by Table 5).

Services heterogeneity also includes data heterogeneity, the latter being present also in services that are otherwise similar (i.e., same basic interaction model and application domain). Data heterogeneity issues are challenging and are out of the scope of this dissertation. Nevertheless, as previous proposal of common description (ROMAN et al., 2015), this architecture assumes that data originating from services is either expressed in an RDF syntax or is converted (lifted) from service-specific formats to RDF and converted (lowered) from RDF to service-specific formats during service invocation according to provided mappings (e.g., the mechanism defined in SAWSDL (KOPECKÝ et al., 2007) and discussed in p. 72).

The use of RDF to solve data heterogeneity demands infrastructure that can be reused if the architecture employs an OWL ontology as common description for services. If an ad-hoc language based on JSON or XML was adopted as common description, it would need to be mapped either to data structures in the programming language under which Unserved is implemented or to RDF resources. Since an ontology is used for service description, the state kept during the execution phase of composition can be stored in RDF. Furthermore, the use of features such as Apache Jena enhanced nodes[2] ease implementation.

Given the role such common description has in the architecture, it is named intermediary description. The ontology[3] is classified as a core ontology and as an operational ontology according to the classifications by Guarino (1997) and (FALBO, 2014) (cf. Section 2.1). It is a core ontology as the object of the ontology are causally-linked messages, which is the perspective used to model services. The classification as a core ontology follows from two characteristics. First, both such messages and services can be found in multiple domains (e.g., financial, scientific, commerce, media, and so on), which precludes its classification as a domain ontology. Second, the ontology has a narrow application and was neither designed nor it is useful in defining new domain ontologies. Its classification as an operational ontology is due to the fact that not all semantics is explicit in its OWL description, and depends on previous agreement between applications (i.e., Unserved architecture implementations) that use the ontology. For this same reason the operational ontology is not intended as a widespread replacement for existing service descriptions.

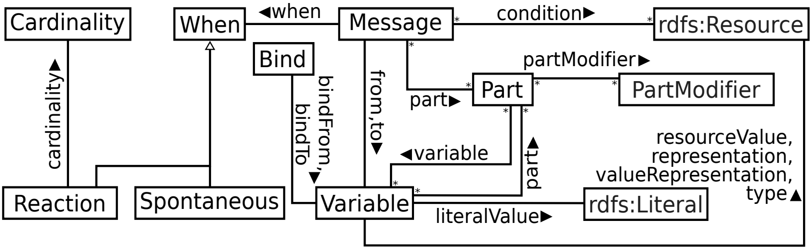
The intermediary description ontology contains two modules. The core module, called Unserved, is shown in Figure 15 and covers high-level interaction from the point of view of messages. While the ontology is expressed in RDF, the notation of Figure 15 is a UML class diagram for compactness. Another module, called unserved-x, relates lower-level details – such as media types and RDF structure – to the higher-level concepts in the core module. Both modules are publicly available online[4]. In RDF listings, `u:` is the prefix of the core module,

---

[2] <https://jena.apache.org/documentation/notes/jena-internals.html>

[3] <https://alexishuf.bitbucket.io/2016/04/unserved/unserved.ttl>

Figure 15 – Intermediary description ontology, displayed as an UML class diagram. Association multiplicity is `*` ► `0..1` unless stated otherwise. Multiple associations share the same line for compactness.



Source: the author.

while `ur:`, `uh:` and `umt:` are prefixes of unserved-x sub-modules. All prefixes are defined in p. 31 and an overview of unserved-x is given in Appendix D.

The two main concepts of the ontology are *Message* and *Variable*. *Message* instances represent the types of messages exchanged during a service invocation. Messages contain technical information that pertains to the protocols involved and are composed of parts. Each part is usually bound to a *Variable* instance and to a part modifier, which determines how the data bound to the variable can be extracted or inserted into the message. Part modifiers are not part of the intermediary description and are officially implementation dependent. However, part modifiers for binding to objects of RDF properties and to parts of HTTP messages are provided in unserved-x.

The *Part* mechanism can also be applied to describe the structure of values attributed to variables. In this way, a message can have its hierarchical structure described up to an arbitrary level of detail as allowed by the part modifiers understood by an Unserved implementation. An important use of parts related to heterogeneity of interaction models is the ability to describe the presence of hypermedia controls embedded in messages.

A *Variable* instance may have a type, a representation and a value. The type of a variable often (but not necessarily) is an RDFS or OWL class of which a *Variable* value is instance of, and representation denotes the format of the value. This distinction is necessary specially

---

[4] <https://alexishuf.bitbucket.io/2016/04/unserved/index.html>

in RESTful services, in which a resource of a semantic type may be represented by an URI, XML document or with an RDF syntax, among others.

The intermediary description has no notion of service. To describe interaction between services and clients, causal relations between messages are defined using the `u:when` property and the `u:When` subclasses. The core Unserved ontology defines two possible causes. A *spontaneous* message can be sent by an agent interpreting the description at any moment. A *reaction* message occurs in response to a previous message and in some cardinality. Request-response interactions consist of a spontaneous message and a reaction message with *single* cardinality. On the other hand, a Publish/Subscribe interaction will consist of a spontaneous subscription message and a reaction with *many* as cardinality.

As the intermediary description has no notion of a service, there is no difference in representation between pre- and post-conditions. Conditions are bound to messages using the `u:condition` property. Pre-conditions are bound to spontaneous messages (the requests or subscriptions) and post-conditions are bound to reaction messages (the responses or notifications). All conditions bound to a message are considered equally truthful under conjunction. In order to send a message, all conditions must hold, and for received messages all conditions are expected to hold.

The process used to design the ontology was *ad-hoc*. The first step in this process was an informal review of current service descriptions techniques and how they could (or could not) model the heterogeneities between the targeted interaction models as discussed in Subsection 2.5.2. From this, came the realization of causally-linked messages with detailed structure as the main abstraction that enables modeling both RESTful and event-oriented services. The remainder of the ontology was developed incrementally as further requirements were identified in the course of development of the Unserved architecture.

Listing 3 shows a fragment of a Hydra service description and the corresponding intermediary description. The Hydra description fragment (Listing 3a) lists a single supported class, `ex:User`, for which a single operation, an HTTP GET, is supported. An HTTP GET on a `ex:User` takes no input and returns a User representation (lines 4-7). Lines 8-10 describe a link that may be present in `ex:User` representations referring to a collection of issues open by that user.

Listing 3b presents the Hydra description translated into the intermediary description, using the core Unserved ontology and the HTTP (`uh:`) and RDF (`ur:`) modules of unserved-x. Both the request

Listing 3 – Example intermediary description for an RESTful service response message.

(a) Hydra description fragment (Turtle).

```
ex:doc a hydra:ApiDocumentation;
  hydra:supportedClass ex:User.
ex:User a hydra:Class;
  hydra:supportedOperation [
    a hydra:Operation;
    hydra:method "GET";
    hydra:returns ex:User ];
  hydra:supportedProperty
    ex:User-raisedIssues .
ex:User-raisedIssues a hydra:Link;
  rdfs:domain ex:User;
  rdfs:range hydra:Collection.
```

(b) intermediary description fragment.

```
1  _:userURI a u:Variable;
2    u:type ex:User;
3    u:representation xsd:anyURI.
4  _:request a http:Request, u:Message;
5    http:mthd httpm:GET;
6    u:when u:spontaneous;
7    u:part [
8      u:variable _:userURI;
9      u:partModifier [ a uh:AsHttpProperty;
10       uh:httpProperty http:absoluteURI ]
11   ].
12 _:response a http:Response, u:Message;
13   u:part [ u:variable _:user;
14     u:partModifier [ a ux-http:PartModifier;
15       ux-http:httpProperty http:body;
16       ux-http:httpResource _:response ] ];
17   u:bind [ u:bindTo _:issueUser;
18     u:bindFrom _:user ];
19   u:when [ u:reactionTo _:request;
20            u:cardinality u:one ].
21 _:user
22   u:type ex:User;
23   u:representation [ a ux-rdf:RDF;
24     ux-rdf:root _:userURI ];
25   u:part [
26     u:variable [ a u:Variable;
27       u:type ex:User-raisedIssues;
28       u:representation xsd:anyURI; ];
29     u:partModifier [ a ux-rdf:AsProperty;
30       ux-rdf:property ex:User-raisedIssues;
31       ux-rdf:propertyOf _:user ] ].
```

Source: the author, Hydra description adapted from an online example[5].

(lines 4-11) and response (12-20) HTTP messages are described. The absolute URI of this request is bound to the `_:userURI` variable (lines 7-11), whose value will be the result of another interaction or given by the user. The response is described as a reaction to the request (lines 19-20) and as containing an `ex:User` representation in its body (lines 13-16). Lines 21-31 state that such representation is in RDF and the same URI of the request identifies the `ex:User` resource.

Variables such as `_:user` in Listing 3b may also have parts. In this case, the `ex:User-raisedIssues` hyperlink is described as part of the `ex:User` representation. This link has a specific type and is required in order to send subsequent requests for the issues raised by the user. Since this type of link may be obtained from multiple alternative interactions, lines 17-18 use the `u:bind` property to bind

---

[5] <http://www.markus-lanthaler.com/hydra/api-demo/vocab>

the `_:user` value to another variable, `_:issueUser`. The latter is not part of `_:response`, but is used in post-conditions of the interaction enabled by the `ex:User-raisedIssues` hyperlink.

In comparison to UFO-S (NARDI et al., 2015), a reference ontology, the intermediary description ontology focus is the description of service delivery. In fact, the intermediary description aims to describe only the interactions between customer and hired service provider (NARDI et al., 2015). A secondary role is service offer, which is implicitly made by the existence of a service description. In the intermediary description, the service provider commits to reaction messages, their variables and to the truth of formulas involving these variables. Service negotiation is not covered in the intermediary description.

User goals are the functional requirements for which a composite service is designed to fulfill. In this architecture, user goals are documented the same as outputs of the desired composite service. That is, goals are `u:Variable` instances that may be part of SPIN elements. Unserved could be extended with other pre/post-condition description mechanism and the user goals descriptions could be decoupled from these, provided that the composition algorithm is extended accordingly.

In line with the intermediary description, an ontology for describing workflows is also defined. The workflow is defined as a sequence of message exchanges, with four possible activity classes: `up:Send`, `up:Receive`, `up:Copy` and `up:Sequence`. The `up:Send` and `up:Receive` activities are associated with an `u:Message` instance and the number of messages received is defined by the message description, not by the activity instance. A copy activity copies values from an `u:Variable` instance from another. Finally, an `up:Sequence` activity contains an `rdf:List` of other activities that are executed in that order.

Activities of workflows are executed by components selected according to the `u:Message` or `u:Variable` instance to which the activity applies. Components are themselves kept in a repository and described with a specific ontology (with prefix `uc:`). Three types of components are described: `uc:ActionRunner`, which executes a workflow activity; `uc:Parser`, which is used to transform actual data exchanges with services into values of `u:Variable` instances; and `uc:Renderer`, which transforms the value of an `u:Variable` instance into bytes that can be sent to actual services.

The description of components uses OWL class expressions (MOTIK; PATEL-SCHNEIDER; PARSIA, 2012, Section 8) to describe the activities or variables to which a component applies. Listing 4 shows the description of a component able to send HTTP requests. The top-

Listing 4 – Description of a component that sends HTTP requests.

```
1   [ a uc:ActionRunner;
2     uc:actionClass [ a owl:Class;
3       owl:intersectionOf ( up:Send
4         [ a owl:Class, owl:Restriction ;
5           owl:onProperty up:message ;
6           owl:someValuesFrom http:Request ] )
7     ] ].
```

Source: the author.

level blank node is the component description. Lines 2-7 contain a class expression that can be read as "an `up:Send` whose `up:message` is an `http:Request`". Any `up:Send` instance that matches this description will be processed by this component. The link between description and code that implements the component is a implementation detail[6].

4.1.1.1 SPIN Conditions

As it stands, the Unserved ontology is able to describe only inputs and outputs of services. Given that, at runtime, service data is also represented in RDF, it is natural to express pre- and post-conditions in terms of RDF. Among description approaches that target RESTful services, only RESTdesc, used in (VERBORGH et al., 2016), and WSMO-Lite (ROMAN et al., 2015) cover pre/post-conditions. However, in WSMO-Lite no logical language is specified for describing such conditions. In RESTdesc, N3 logic (BERNERS-LEE et al., 2008) is used, and the expressivity of pre-conditions amounts to applying variable substitution to a conjunction of formulas against the N3 logic entailment of the current world state.

The key to adding pre- and post-conditions to intermediary descriptions is binding them to some external language. Unserved places no requirement on such language. Nevertheless, for the purposes of this dissertation, SPIN (KNUBLAUCH; HENDLER; IDEHEN, 2011) is used (see p. 53). The interpretation of pre- and post-conditions within composite service design algorithms are described in Subsection 4.2.2. The supported subset consists of *triple patterns* and *unions*. Since SPIN is expressed in RDF, instances of Unserved `u:Variable` can also be made instances of `sp:Variable` and included in SPIN expressions.

---

[6] The prototype uses an additional property to hold the fully qualified implementation class name.

Listing 5 – SPIN elements pre/post-conditions.

(a) A post-condition describing issues raised by a user.

```
1   _:issueUser sp:varName "issueUser".
2   _:issue sp:varName "issue".
3   _:responseIssues a u:Message;
4     u:condition [ sp:subject _:issue;
5       sp:predicate ex:Issue-raisedBy;
6       sp:object _:isssueUser ].
```

(b) A pre-condition for a “POST User” operation.

```
1   _:pUser sp:varName "pUser".
2   _:postUser a u:Message;
3     u:condition [ a sp:Union;
4       sp:elements (
5         ([ sp:subject _:pUser;
6            sp:predicate ex:User-name;
7            sp:object [sp:varName "name"] ])
8         ([ sp:subject _:pUser;
9            sp:predicate ex:User-email;
10           sp:object [sp:varName "email"] ]) ) ].
```

Source: the author.

Listing 5 shows hypothetical[7] examples of SPIN elements. Listing 5a shows a post condition bound to the response message (`_:requestIssues`) corresponding to the `ex:User-raisedIssues` hypermedia control. The same `_:issueUser` variable from Listing 3b is used in the post-condition of the response to the request made by following the link. The condition itself (lines 5-7) is a simple *triple pattern* stating that issues on the response were raised by the user from where the link was followed.

Listing 5b describes a pre-condition for creating a new User through a POST operation. The submitted representation must contain either a name, an email, or both. The instance of `sp:Union` represents this disjunction between the two *triple patterns* that are members of the list under `sp:elements`. `_:pUser` is an instance of `u:Variable` that is bound to the body of the HTTP POST request.

In most situations, `u:Variable` instances will contain resources as values, which makes their use in SPIN elements intuitive, as previously shown. However, another common situation are literal values. The condition that a variable `_:x` has the value 9 can be represented in SPARQL as `?x u:literalValue 9.` and encoded as a SPIN triple pattern. This is enabled by forbidding the use of Unserved properties that have `u:variable` as their `rdfs:domain` in SPIN conditions. If those predicates are used in a triple pattern, the semantics is that, during SPARQL evaluation, the subject of the triple pattern will be bound to the `u:Variable` instance, not to its `u:resourceValue`.

---

[7] The original Hydra description used as example in this section does not impose these conditions.

4.1.2 Processing Elements

The Unserved architecture (Figure 14) contains five processing elements:

1. Translators
2. Service repository
3. Composer
4. Interpreter
5. Component repository
6. Components

Translators are responsible for converting existing service descriptions into the intermediary description (Subsection 4.1.1) upon request by an external agent (e.g., an end user or a crawler). Translators also employ specific vocabularies in order to annotate Unserved descriptions with information specific to the service type or to protocols involved. While the core Unserved ontology is a common description, agnostic to many heterogeneities, several details, such as construction of HTTP messages and WSDL operations, must be preserved for use during service interaction.

The *service repository* stores intermediary descriptions of services. Its main functions are indexing messages and providing fast querying of `u:Message` and `u:Variable` instances. Precise indexing needs depend on the composition algorithm used. The composition algorithm used in this dissertation, based on ComposIT (Subsection 4.2.1) requires an index from classes to `u:Message` instances that have as parts variables with the key as their `u:type`. The most frequent query done by ComposIT is "what services have some input satisfied by this class?".

The *composer* component encapsulates a composite service design algorithm. The architecture does not mandate a specific algorithm, but requires that the algorithm provides support for the HATEOAS constraint and for EOSs. Section 4.2 proposes two techniques that, when applied to an existing algorithm, yield an algorithm that conforms to HATEOAS and with the EOS interaction model. The composer takes as input user goals and discovers required services from the service repository. The output of composition is a workflow.

Workflows are executed by the *interpreter*. Its main responsibility is managing execution and delivery of results. Actual execution of each activity in the workflow is delegated to a component stored in the *component repository*. This is so due to the fact that service heterogeneities hidden by the intermediary language still exist and must be

```
SELECT ?component WHERE {
  ?component uc:actionClass ?class.
  <activity> a ?class.
}
```

Listing 6 – SPARQL query for selecting a component for a workflow activity.

taken into account during execution. As workflows have a tree structure, the Interpreter selects a component for the workflow root activity and executes the selected action runner component. In the case of activities such as `up:Sequence`, the action runner component may select other components and execute them in the context managed by the *interpreter*.

The *component repository* manages and indexes component descriptions (cf. Subsection 4.1.1) as well as their instantiation upon request. The main task of the repository is selecting the component for a specific workflow activity instance. In OWL jargon, component selection problem consists of concurrently instance-checking the activity instance against all class expressions of component descriptions. Under OWL inference, considering `<activity>` to be the activity instance, this could be satisfied with the SPARQL query in Listing 6.

4.1.2.1 Ontology Reasoning

The use of RDF for handling service data and defining the intermediary description as an ontology motivates the exploitation of implicit knowledge through OWL or RDFS inference. The Unserved architecture itself does not mandate this, but a discussion on implementation alternatives – and the approach taken in the official prototype, used for evaluation (Chapter 5) – is pertinent.

State-of-the-art reasoners, such as Pellet (SIRIN et al., 2007) and FaCT++ (TSARKOV; HORROCKS, 2006), albeit powerful and fairly efficient, have their performance often measured in the scale of seconds (KANG et al., 2014), while the performance for composition algorithms is often measured in milliseconds (RODRIGUEZ-MIER et al., 2012; VERBORGH et al., 2016; CHATTOPADHYAY; BANERJEE; BANERJEE, 2015). The direct use of these reasoners is an obstacle for achieving performance close to state-of-the-art service composition approaches.

The prototype of the Unserved architecture employs reasoning during composition design and execution, using three different strategies at the same time. At composition design, a customized rule-based reasoner is applied to all user-provided input and the intermediary descriptions (including the domain ontologies referred by these). This customized reasoner contains:

- Class inference rules manually extracted from the Unserved intermediary description ontology (e.g, if `?x` has a `u:type`, then `?x a u:Variable`);
- OWL equivalence rules: `owl:equivalentClass`, `owl:equivalentProperty` and `owl:sameAs`;
- OWL everything is a `owl:Thing` rule;
- RDFS `rdfs:subPropertyOf` rule.

The second strategy intertwines class hierarchy rules (i.e., transitivity of `rdfs:subClassOf`) with the composite service design algorithm, whose designer can define an inference strategy better than a generic reasoner could. The algorithm must manually traverse `rdfs:subClassOf` links between classes, and take these into consideration when building data structures from the intermediary descriptions data. Algorithm implementations may also employ closed-world semantics when evaluating constraints of the intermediary descriptions.

The third reasoning strategy is responsible for component selection and consists in a breadth-first exploration of class expressions used with `uc:actionClass`. Each class expression is converted into a list of simple queries, involving only Basic Graph Patterns (BGPs) (SEABORNE; HARRIS, 2013), that may provide conclusive evidence on whether the activity satisfies, or not, that expression. One query from each list is evaluated, breadth-first. If a query implies that the instance does not satisfy the expression, that list is removed from further evaluations. Otherwise, if membership is implied, the matching component is returned. Finally, if the query does not match (inconclusive evidence), evaluation of queries continues.

4.2 FORKED AND ADAPTIVE COMPOSITION

This section presents two techniques applied to a composition algorithm. The first, Forking, provides basic support for composition of Publish/Subscribe services. All composite service execution state is stored on an object that can be duplicated and used to spawn an independent execution thread for each notification message of Publish/Subscribe services. For example, consider a fictitious scientific publisher system. As researchers submit articles to journals, events are generated on a Publish/Subscribe service. Forking allows a composition algorithm to be planned so that for each submission (event), a service composition for assigning at least three reviewers is designed and executed. The second approach, Adaptation, adds support for the HATEOAS constraint. The composition plan is re-evaluated against the actual hypermedia controls, after every service interaction. Adaptation requires a graph-based composition algorithm, which operates on a *state graph* that has the following characteristics:

1. States contain a set with one or more services;
2. The (u_1, u_2) edge corresponds to invoking u_1 services, achieving the necessary conditions so that u_2 services may be later invoked;
3. There are lists of alternatives to services and states.

An overview of a composition using an algorithm with these techniques is depicted in Figure 16. Upon client request, the composer creates an execution object from the composition problem with all necessary data structures (including the state graph) for planning the composition. A plan is obtained from the state graph, as a list of state graph edges[8]. If no plan can be obtained, the user is notified and composition ends. Each edge originates a sequence of actions executable by Unserved (copy, send, receive), and this sequence is further divided into *slices*. A slice contains at most one receive action, that if present, is the last action.

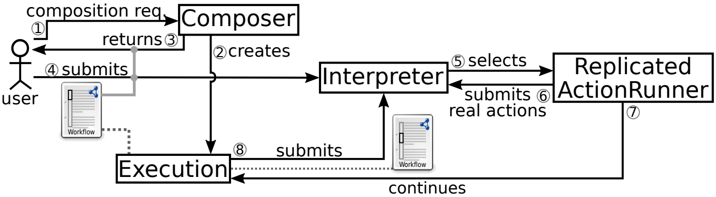
The descriptions of all services and other elements of the problem (e.g., known inputs and wanted outputs) are stored on a single RDF graph that serves as working memory of the composition. Service inputs and outputs are present in the graph as unbound variables, which will be bound to values as Action Runners (described in Section 4.1) that execute the planned actions. After every received message, results must be confronted with expectations so that the plan is adapted, if necessary. Therefore, only the first slice must be executed before control returns to the composition algorithm. The composer returns a workflow with only the first slice, wrapped with a fork action, and with a pointer back to the execution object. When interpreting this action, as depicted in Figure 16, the interpreter will select the *Fork action Runner*, which will return control to the composition algorithm for validation, adaptation and execution of the next slice.

The runner selected by the interpreter for a fork action works as shown in Listing 7. First, all actions except the receive (r) are executed, producing side effects on the execution context (line 2). If there is a receive and it will result in multiple messages (line 5), each

---

[8] The list of edges may constitute a path or a tree. Either way, the last edge will lead to the goal state, concluding the composition.

Figure 16 – Overview of adaptive composition execution.



Source: Adapted from (HUF; SALVADORI; SIQUEIRA, 2017).

Listing 7 – Fork action Runner implementation.

```
1:  procedure RunWrapper($w, c$)
2:      $r \leftarrow$ ExecuteUntilReceive($w, context(e)$)
3:      $e \leftarrow execution(w)$
4:      SetInterperter($e, interpreter(c)$)
5:      if $r \neq$ null $\wedge\ cardinality(r) \neq$ one then
6:          for all $1 \leq i \leq cardinality(r)$ do
7:              $e' \leftarrow$ Fork($e$)
8:              Run($interpreter(c), r, context(e')$)
9:              begin thread
10:                 Resume($e'$)
11:             end thread
12:     else
13:         if $r \neq$ null then
14:             Run($interpreter(c), r, context(e)$)
15:         Resume($e$)
```

Source: the author.

one is received into a forked execution (line 8) and resumed in another thread (line 10). Single-shot messages are received into the current context and resumed directly (lines 13-15).

The implementation suggested in Listing 7 is not the most efficient, since threads incur some management and context-switching overhead. In addition, the Fork may also impose a limit on scalability if it copies the execution state during a fork. If a composite service executing under the algorithm from Listing 7 involves an EOS with high-frequency notification (e.g., a notification for each transaction on an e-commerce or on a stock exchange) a fixed thread pool should be used and FORK should promote sharing of resources between forked executions by employing Copy-On-Write[9] at a high granularity level.

The RESUME procedure of the execution object displayed in Listing 8 closes the cycle, validating the results against the current plan of the execution object (e). Validation consists of checking if the current composite service execution state satisfies all preconditions of the next stages in the planned workflow. If the edge of the state graph had its last slice executed, the effects on $context(e)$ are validated against the executed edge (line 4), resulting in a set of failed services. If valid, the execution object advances to the next edge of the composition path. Otherwise, the state graph is adapted in order to re-plan the composition (lines 7-8). If the PLAN fails at this stage, execution ends and the user is notified by a callback that this particular forked execution thread failed. If the last slice was not the last one for the current edge ($remainingEdgeActions(e) \neq \emptyset$), the composition advances to the next slice (line 10). When ADVANCEEDGE meets the end of the path, it sets the *isEnded* flag, which causes the composition execution to deliver a result and finish (lines 12-13). If there is still work to do, the current slice is wrapped and executed with the current interpreter (line 4 of Listing 7).

The procedure for adapting the state graph upon edge failure is shown in Listing 9, where G stands for the state graph, (u_1, u_2) is the failed edge, and $F \subseteq u_1$ is the set of failed services. A state a is constructed to replace u_1, without including services from F. Each failed service has a list of alternatives, denoted by $alts(f)$. Since not all alternatives are equivalent, a helper function COMPAT$(alts(f), u_1)$ filters $alts(f)$ and returns only alternative services that can be invoked using the already bound inputs (and other state) of u_1.

There are three possible constitutions of the alternative state a. First, if all failed services in u_1 have invocable alternatives, they will constitute a (line 6). Second (lines 7-13), if there is a suitable alternative state u_a to u_1 which does not include any node from the set $\hat{F}_{u_1}$. This set is the union of all sets F for u_1 and all previous states (alternatives and the original) that were considered in this position of the state graph. Third, a will be an empty state distinct from any other empty state already in G (line 15).

Once a has been computed, the inputs of its services are bound with values already present for inputs of u_1 (line 17). The incoming (I) and outgoing edges (O) of u_1 originate edges to and from a (lines 18-19),

---

[9] Copy-on-Write is a programming idiom where data structures are shared between different objects, but are copied by any object that needs to modify them. Also known as *implicit sharing* or *shadowing*.

Listing 8 – Resume Execution algorithm.

```
procedure RESUME($e$)
    if $remainingEdgeActions(e) = \emptyset$ then
        $F \leftarrow$ VALIDATE($e, currEdge(e)$)
        if $F = \emptyset$ then
            ADVANCEEDGE($e$)
        else
            ADAPT($e, F$)
            PLAN($e$)
    else
        ADVANCESLICE($e$)
    if $isEnded(e)$ then
        $r \leftarrow$ CREATERESULT($e$)
        DELIVERRESULT($interpreter(e), r$)
    $c \leftarrow$ WRAPASREPLICATED($e, currSlice(e)$)
    RUN($interpreter(e), c, context(e)$)
```

Source: the author

this time adapting the assignments so that a's inputs and outputs are used in place of those of u_1. ADAPTEDGES also verifies any additional criteria that applies to edges of the state graph[10].

Depending on the alternative chosen by *chs* in Listing 9, more or less edges may be lost (l). For brevity, we omit exploration of these alternatives with a **minimize** block (line 5) to symbolize that only the side effects of the iteration that yielded the smallest l remain. That is, the actual implementation of *chs* chooses the alternative services that, together, yield the smallest number of lost edges due to I/O parameter incompatibility. Specific properties of the composition algorithm to which the adaptation technique is applied can be used to forgo an actual minimization algorithm by a faster strategy. This is done, for example, for ComposIT in Subsection 4.2.1.

4.2.1 Adapting a graph-based composition algorithm

We apply the aforementioned techniques to the ComposIT algorithm proposed by Rodriguez-Mier et al. (2012) within the composition

---

[10] For example, if edges are determined by pre-/post-conditions, the satisfaction of pre-conditions must be rechecked during edge adaptation

Listing 9 – Algorithm for graph adaptation.

```
procedure Adapt($G, (u_1, u_2), F$)
    $I \leftarrow \{(u_0, u_1)$ s.t. $(u_0, u_1) \in E(G)\}$
    $O \leftarrow \{(u_1, u_3)$ s.t. $(u_1, u_3) \in E(G)\}$
    $l \leftarrow \infty$
    minimize $l$
        $a \leftarrow \{chs(\textsc{Compat}(alts(f), u_1))$ s.t.$f \in F\}$
        if null $\in a$ then
            $a \leftarrow$ null
            for all $u_a \in alts(u_1)$ s.t. $u_a \cap \hat{F}_{u_1} = \emptyset$ do
                $a \leftarrow \textsc{Compat}(u_a, u_1)$
                if $a \neq$ null $\wedge |a| = |u_a|$ then
                    break
                $a \leftarrow$ null
        if $a =$ null then
            $a \leftarrow \{\textsc{CreateUniqueEmptyNode}(G)\}$
        else
            AssignInputs($a, u_1$)
            $l \leftarrow \textsc{AdaptEdges}(a, u_1, I)$
            $l \leftarrow l + \textsc{AdaptEdges}(a, u_1, O)$
    end minimize
    CreateNoOp($\{(a, u_0)$ s.t. $(u_0, u_1) \in I\}$)
    RemoveNode($u_1$)
```

Source: the author

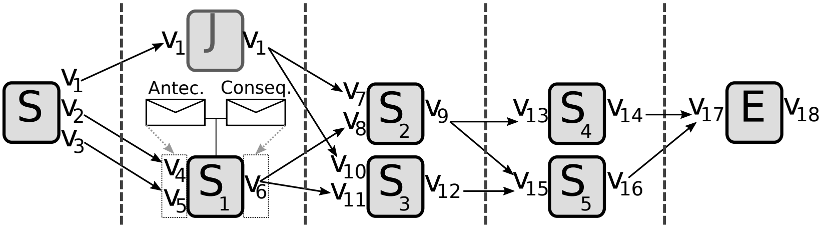
architecture of Section 4.1. In this algorithm, a composition problem consists of a set of services, a set of known inputs (bound variables in the intermediary description) and wanted outputs (unbound variables). ComposIT was chosen due to good results obtained in the WSC'08 benchmark (BANSAL et al., 2008) and due to availability of source code for its prototype implementation[11].

The ComposIT algorithm has three phases: (1) Construction of a dependency graph; (2) its optimization; and (3) planning the composite service workflow. An example of the dependency graph is shown in Figure 17, where variables v_1, v_2, v_3 are known inputs and v_{18} is the wanted output. Each node S_i in this graph corresponds to a service, which in the intermediary description is a pair of antecedent and con-

---

[11] <https://github.com/citiususc/composit>

Figure 17 – An example service dependency graph.



Source: the author.

Listing 10 – Algorithm for execution validation.

```
function VALIDATE($e, (u_1, u_2)$)
    $F \leftarrow \emptyset$                                    ▷ set of failed services
    for all $a \in actions((u_1, u_2))$ s.t. $a : Copy \wedge target(a) \in I(u_1)$ do
        if $\neg isBound(target(a))$ then
            $r \leftarrow$ FINDREPLACEMENT($target(a), u_1$)
            if $r \neq$ null then
                COPY($r, target(a)$)
            else
                $F \leftarrow F \cup \{provider(source(a))\}$
    REVERTASSIGNMENTS($F$)
    return $F$
```

Source: the author.

sequent messages. Variables that are part of the antecedent form the inputs of the node and those that are part of the consequent, the outputs. Edges connect the output variables in a layer with compatible inputs of the next layer: an output o with type t_o and representation r_o matches an input of type t_i and representation r_i if, and only if, $t_o \sqsubseteq t_i \wedge r_o = r_i$. Special jump nodes (J) ensure that only nodes of neighboring layers are connected. However, the insertion of such nodes is more efficient if performed during later phases, on demand. As part of a series of changes to improve efficiency under Unserved, a jump node is created for every output not in the previous layer, while in the ComposIT prototype a jump node is created per service).

The optimization phase removes nodes from the dependency graph. If two nodes s_1 and s_2 have the same predecessors, but s_2 successors are a subset of those of s_1, s_1 is said to dominate s_2 and s_2 can be removed. If instead, their successors sets are equal, only one of them must remain in the dependency graph. These removals, named *Offline Service Compression* (RODRIGUEZ-MIER et al., 2012), have no effect on the ability to find the optimal composition path. However, at runtime, the selected service may be unavailable, it might not contain a hypermedia control, or it may miss some relevant output. A topological sort of the *dominates* relation is used to identify a representative node and compile a sorted list of its *alternatives* allowing the implementation to forgo the **minimize** block in Listing 9. Additionally, as all alternatives have the same predecessors, COMPAT is always true.

The state graph for this composition algorithm is also a layered graph, where a state is a combination of services that share the same layer on the dependency graph. An edge (u_1, u_2) exists between two states if, and only if, all outputs of u_2 can be assigned from outputs of u_1. The planning phase of ComposIT consists in searching a path in the state graph. The graph is materialized on demand during backward search, and the *Online Node Reduction* optimization (RODRIGUEZ-MIER et al., 2012) detects and removes equivalent states. As in *Offline Service Compression*, removed states are kept as *alternatives* of the representative state, but no sorting is required.

Validation of an execution context against an edge (u_1, u_2), shown in Listing 10, checks for the inputs of u_2 that were targets of a copy action but remained unbound (lines 3-4). For any unbound input, the algorithm attempts to find a replacement source (lines 5-7), or otherwise adds the original provider service to the set F of failed services. Finally, assignments from failed services are undone (line 10).

4.2.2 Adding Support for SPIN Pre/Post-conditions

The ComposIT algorithm can be extended to support the pre-/post-conditions documentation with SPIN proposed in Subsection 4.1.1. However, two syntactical restrictions must be enforced on SPIN elements. First, only conjunction (RDF lists) and disjunction (`sp:Union`) of `sp:TriplePattern` are allowed. Second, all variables of triple patterns must also be instances of `u:Variable` bound to or from the `u:Message` to which the SPIN condition applies.

The rules for existence of an edge (s_1, s_2) in the service dependency graph remain the same and are determined exclusively by inputs

Listing 11 – SPARQL query for constructing SPIN triple patterns implied by hierarchical output variables.

```
CONSTRUCT {
  _:tp sp:subject   ?outer;
       sp:predicate ?pred;
       sp:object    ?inner;
} WHERE {
  ?outer u:part ?part.
  ?part u:variable ?inner;
        u:partModifier ?mod.
  ?mod a ur:property ?pred.
}
```

Source: the author.

and outputs. However, for efficient determination of valid states during the planning phase of ComposIT, a data structure should be built for fast determination of which post-conditions of a predecessor service satisfy which pre-conditions of a successor service.

During the planning phase, states are generated using the same criteria as in regular ComposIT. After elimination of equivalent states (Online Node Reduction), a state v will have many *candidate* incoming edges. Only candidate edges that are condition consistent will be permanently included in the state graph and considered during the path search. Before defining the criteria for condition-consistency, some preliminary definitions are required.

Pre-conditions of a service s are denoted by Φ_s. They have the form of triple patterns combined by conjunction or disjunction, respectively represented by RDF lists and `sp:Union` elements. In this text, conjunction and disjunction of triples are denoted by symbols $\wedge$ and $\vee$, respectively. Post-conditions of the service s are denoted by Ψ_s. Both Φ and Ψ are determined by `u:condition` triples involving, respectively, the spontaneous and reaction messages that constitute the service abstraction.

An Intermediary description also carries an implicit type of post-condition, denoted by $P(s)$, which is implied by hierarchical definition of `u:Variable` instances using `u:part`. In the most common cases that a variable has parts defined using a `ur:AsProperty` part modifier, a triple pattern post-condition is implied. This modifier states that the inner variable, which is part of the outer, is the object of a specific RDF property. This inference can alternatively be defined using a SPARQL construct query, as shown in Listing 11.

The flattening function, F, replaces disjunction with conjunction and is defined as follows:

$$F(\Gamma) = \begin{cases} \langle s\,p\,o\rangle & \text{if } \Gamma = \langle s\,p\,o\rangle \text{ is a triple pattern} \\ F(\Gamma_1) \cup F(\Gamma_2) & \text{if } \Gamma = \Gamma_1 \vee \Gamma_2 \\ F(\Gamma_1) \cup F(\Gamma_2) & \text{if } \Gamma = \Gamma_1 \wedge \Gamma_2 \end{cases}$$

The graph function, G, is a transformation that obtains an RDF graph from a set of SPIN triple patterns by treating variables as plain RDF nodes and transforming each triple pattern in an RDF triple.

Finally, $\Gamma_1 \vDash_S \Gamma_2$ is an entailment relation from a set of triple patterns, Γ_1 to a conjunction of SPIN conditions. Such relation holds when there is a substitution $\sigma = \{x_1 \mapsto y_1, \ldots\}$ such that the following two conditions are met:

1. The SPARQL query Γ_2', which is obtained by considering all variables in Γ_2 as RDF nodes instead of variables, has a result in the graph $G(F(\Gamma_1\sigma))$; and
2. If $x \mapsto y \in \sigma$ with t_x and t_y being the `u:type` of x and y, and with r_x and r_y being the `u:representation` of x and y, then $t_x \sqsubseteq t_y \wedge r_x \sqsubseteq r_y$.

With the previous definitions, the condition-consistency property can be defined. If $e = (u, v)$ is a candidate edge, for inclusion on the state graph, then e is condition consistent, if, and only if, $\bigcup_s^u (F(\Psi_s) \cup P(s)) \vDash \Psi_s$. In other words, in order to consider SPIN conditions, edges of the state graph must be further validated to ensure pre-conditions are satisfied.

The criteria for *Offline Service Compression* and *Online Node Reduction* remain unaltered, as both are defined only in terms of edges in the dependency and the state graphs.

4.3 FINAL REMARKS

This chapter described Unserved as well as the process that is executed during forking and adaptive composition of services with heterogeneous interaction models. The discussion of the architectural elements and of the composition algorithm also covers aspects, such as preconditions using SPIN and the use of RDF and OWL ontologies for several purposes - from service data representation to component selection. These aspects are not strongly linked to the goal of the architecture. Nevertheless, they are included as part of the Unserved architecture for their rationales mentioned in the previous sections and to approach the discussion in this dissertation from the implemented prototype. These additional aspects are also considered during evaluation in Chapter 5, both in comparison to the objectives of this work and in comparison to the related work.

5 EVALUATION

This chapter evaluates the proposed architecture. The goal of proposing a architecture that enables composition of the three basic interaction models directly requires evaluation of whether the heterogeneities between these models are supported. Secondly, composition process should be able to solve problems with practical relevance. This motivates evaluation of expressivity of intermediary description and the produced composite service specifications. In addition to solving composition problems, this solution should not have a prohibitive cost in terms of performance, which motivates performance comparison with state-of-the-art approaches.

Hevner et al. (2004) presents a taxonomy of artifact evaluation methods. Evaluation of heterogeneity support between basic interaction models demands an analytical evaluation. In this category, architecture analysis (for asserting conformance of Unserved to each basic interaction model) and static analysis (for assessing other properties of the architecture) are the most appropriate methods. These analyses are not based on the evaluations performed by studies selected in the SLR (Section 3.1), given that only 3 (GEORGANTAS et al., 2013; MINGUEZ; ZOR; REIMANN, 2011; JURIC, 2010) out of 54 studies present adequate architecture analyses. In addition, these three works are limited to SOAP and EOS heterogeneity (more details can be found in Subsection 3.1.6).

Section 5.2 details an Internet of Things (IoT) scenario. This is done to evaluate the applicability of Unserved, with the ComposIT algorithm extended with SPIN conditions, in a practical composition problem. A scenario is only one of many possible situations where a automatic composition tool could be useful. While all studies selected in the SLR from (Section 3.1) discuss a single successful scenario (Subsection 3.1.6), Section 5.2 also discusses variations of the scenario that reveal limitations in the applicability of Unserved as presented in Chapter 4. Such limitations have solutions proposed.

In related work covered by the SLR in Section 3.1, only three studies propose automatic composite service design. Of these, only (UPADHYAYA; ZOU, 2012) includes a controlled experiment for performance evaluation. Furthermore, these three works are not representative of the state of the art in automatic composition. This chapter compares performance with two approaches listed in Section 3.2. Subsection 5.3.1 employs the WSC'08 benchmark (BANSAL et al., 2008) to compare the original ComposIT algorithm with its forking and adaptive version implemented on Unserved.

Composition of RESTful services often neglects the HATEOAS constraint or supports it statically, as discussed in Subsection 3.1.2. Among the identified composition algorithms that target RESTful services, only the Pragmatic Proof (VERBORGH et al., 2016) does support the HATEOAS constraint. Subsection 5.3.2 uses the same benchmark used in (VERBORGH et al., 2016) to compare the Unserved prototype with the Pragmatic Proof algorithm.

5.1 ANALYSIS

Architectural analysis is conducted in Subsection 5.1.1 to assess the support of Unserved for heterogeneities between the three basic interaction models implied by SOAP services, RESTful services and EOSs. In this evaluation, results are contrasted with conclusions from Subsection 3.1.4. Static analysis is conducted in Subsection 5.1.2, where the composition algorithm obtained by applying forking, adaptation and SPIN conditions to ComposIT (RODRIGUEZ-MIER et al., 2012) has its expressivity evaluated and compared to state-of-the-art approaches for automatic composition (Section 3.2). While this last evaluation is based on the prototype, it only evaluates if some features are supported by the prototype, and does not run experiments.

5.1.1 Heterogeneity support

Table 8 compares the coverage of service types by Unserved to that provided by existing heterogeneous composition approaches identified in the SLR from Section 3.1. The existing approaches are presented in three groups, separated by horizontal lines. The first group approaches target the three service types but largely to properly support RESTful services. Liu et al. (2014) maps REST services to an action & I/O abstraction limiting method to POST and not allowing parametric URIs. The other two approaches (CHENG et al., 2017; ARDISSONO et al., 2009) do not present evidence of achieving protocol maturity, but also have no evidence of the contrary. The second group contains studies that support HATEOAS statically. That is, application state is driven by a statically defined workflow, not by hypermedia controls. The third group summarizes all remaining observed combinations of values for the columns. More details for all studies can be obtained in Section 3.1.

Table 8 shows that Unserved is the only approach to target all three service types and to fully the RESTful services basic interaction model. Unserved employs two heterogeneity support methods identi-

Table 8 – Supported service types by approaches in the SLR compared with Unserved.

Approach	**SOAP**	**REST**	**EOS**	**RMM level**
Liu et al. (2014)	✓	✓	✓	POX
Cheng et al. (2017)	✓	✓	✓	protocol (unclear)
Ardissono et al. (2009)	✓	✓	✓	protocol (unclear)
Haupt et al. (2014)	✓	✓	✗	HATEOAS (static)
Pautasso (2009)	✓	✓	✗	HATEOAS (static)
Roman et al. (2015)	✓	✓	✗	HATEOAS (static)
Upadhyaya & Zou (2012)	✓	✓	✗	HATEOAS (static)
Ramtohul & Ogunleye (2017)	✓	✓	✗	HATEOAS (static)
7 studies	✓	✓	✗	POX
4 studies	✓	✓	✗	resources
14 studies	✓	✓	✗	protocol
4 studies	✓	✓	✗	unclear
Liechti et al. (2015)	✗	✓	✓	POX
16 studies	✓	✗	✓	-
Unserved	✓	✓	✓	HATEOAS

Source: the author.

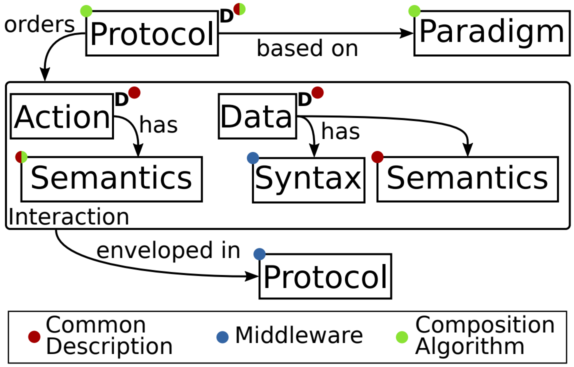
fied in Section 3.1: Common description and Middleware[1]. As discussed in Subsection 3.1.2, these methods are not sufficient for support of HATEOAS nor EOSs even theoretically. Therefore, Unserved also employs the forking and adaptive composition algorithm discussed in Section 4.2 as a method for supporting heterogeneity.

Figure 18 depicts which sources of heterogeneity (in the protocol model of Subsection 2.5.1) are covered by which data or processing elements of the Unserved architecture. Borrowing terminology from logics, the unserved architecture elements, the forking and the adaptation techniques could be said to be monotonic with respect to the heterogeneities between the basic interaction models considered in this dissertation. That is, the introduction of adaptation, for example, does not retract the support for any previously supported heterogeneity.

---

[1] Although the architecture includes a workflow language as a data element, it does not contribute to heterogeneity support.

Figure 18 – Relation of architectural elements to heterogeneity support in Figure 9.



Source: the author.

Protocol & Paradigm

The composition algorithm is responsible for dealing with paradigm and protocol heterogeneities between the three basic interaction models. Altogether the three interaction models cover four paradigms, which are supported by the composition algorithm:

1. Client-Server: This paradigm demands service and client to communicate over a network. This is trivially supported by the architecture and all works cited in Chapter 3;
2. Client-Stateless-Server: This paradigm adds the two restrictions to Client-Server. First, the server holds no client-specific state. Second, each message has all necessary information for its processing, requiring no such client-specific state stored on the server. These are a matter of server implementation and domain-specific service description;
3. Notification: In this paradigm, the server notifies the client about events to which it has shown interest previously. If the client has not demonstrated interest in such event, there is no need for service composition, as somehow the service is coupled to the client or there is some binding mechanism already in place. Considering the pattern of subscription+notification, this is represented in the intermediary language as a spontaneous message and a reaction message to the first with many-valued cardinality. During execution, the interpreter will indefinitely receive notifications,

Listing 12 – A Request message depending on a hypermedia control from `_:response` in Listing 3b.

```
_:requestIssues a u:Message, http:Request;
  http:mthd httpm:GET;
  u:when u:spontaneous;
  u:part [ u:variable [ a u:Variable;
      u:type ex:User-raisedIssues;
      u:representation xsd:anyURI ];
    u:partModifier [ a uh:AsHttpProperty;
      uh:httpProperty http:absoluteURI ] ].
```

Source: the author.

spawning an independent fork of the composite service for each received message;

4. Publish/Subscribe: This differentiates from notification by including an event broker and by variations on how the subscription and events are described (topic, content or type) (EUGSTER et al., 2003). The first difference is not perceivable from the standpoint of service composition. The second is a matter of domain-specific service description.

The protocol of a service is specific to that service. Conformance of a composite service to the protocols of component services must be ensured at design-time, be the design performed beforehand or concurrently with execution. In the case of manual design, it is responsibility of the human developers to ensure conformance. In the case of automatic design, the responsibility lies in the algorithm.

HATEOAS is a form of protocol description and discovery which implies either automatic design or human-computer interaction during execution of the composite service. Support for HATEOAS in a composition algorithm has been shown by (VERBORGH et al., 2016). The adaptiveness of composition (Section 4.2) is analogous to the Pragmatic Proof algorithm. In Unserved, hypermedia controls are described as parts of `u:Message` instances and are required in order to send further request messages. Listing 12 shows an example of such request message, which requires th `ex:User-raisedIssues` link described as part of the `_:request` message in Listing 3b to be sent. This requirement is considered by the planning algorithm and, like in Pragmatic Proof, validated during execution.

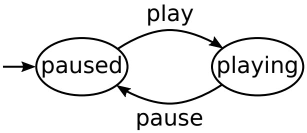
For SOAP services, whether and how application protocol is described is not a consensus. Therefore, these aspects are not part of

Figure 19 – Music player control service state machine description. Messages are not fully described for readability.

(a) intermediary description.

```
1   _:state a u:Variable; u:type ex:State;
2     u:literalValue "paused".
3
4   _:playRequest a u:Message;
5     u:condition [ sp:subject   _:state;
6       sp:predicate u:literalValue;
7       sp:object    "paused" ];
8     u:when u:spontaneous.
9   _:playResponse a u:Message;
10    u:bind [ u:bindTo _:state;
11      u:bindFrom [ a u:Variable; u:type ex:State;
12      u:literalValue "playing"] ];
13    u:condition [ sp:subject   _:state;
14      sp:predicate _:literalValue;
15      sp:object   "playing" ];
16    u:when [ u:reactionTo _:playRequest; u:cardinality u:single ].
17
18  _:pauseRequest a u:Message;
19    u:condition [ sp:subject   _:state;
20      sp:predicate u:literalValue;
21      sp:object "playing" ];
22    u:when u:spontaneous.
23  _:pauseResponse a u:Message;
24    u:bind [ u:bindTo _:state;
25      u:bindFrom [ a u:Variable; u:type ex:State;
26      u:literalValue "paused"] ];
27    u:condition [ sp:subject   _:state;
28      sp:predicate _:literalValue;
29      sp:object   "paused" ];
30    u:when [ u:reactionTo _:pauseRequest; u:cardinality u:single ].
```

(b) State machine diagram.



Source: the author.

the basic SOAP service interaction model. Nevertheless, state machine protocols can be described using (1) the `u:condition` property in intermediary descriptions with SPIN elements; (2) an `u:Variable` instance to hold the current state; (3) and `u:bind` directives to update that state. An example of a music player control service is shown in Figure 19. An `u:Variable` (lines 1-2) instance controls the service state which initially is paused. A play command has the precondition that the state is *paused* (lines 5-7) and the post-condition that it is *playing* (lines 13-15). The `u:bind` directive in lines 10-12 causes the value of `_:state` to change accordingly. The pause operation (lines 18-30) is described analogously.

Finally, as the basic interaction model of an EOS does not mention presence of a protocol description beyond the Publish/Subscribe and notification paradigms. However, the set of EOSs is not disjoint from any other service type. Therefore, the arguments presented for the case of HATEOAS and statically described protocols in SOAP services also apply to each forked execution that spawns from a notification receipt.

Action

All three basic interaction models use names for their actions. SOAP services and Web APIs with maturity lower than protocol level of RMM use application-specific sets of operations. Services with protocol maturity or higher use the HTTP methods (GET, PUT, POST, DELETE, HEAD, PATCH) as their actions. EOS will invariably include an action with *subscribe* semantics, but will typically also include other actions following the conventions of other service types.

Action names are supported by the intermediary description which describes them as part of the request messages. Description of action semantics is supported through their I/Os, described with ontology classes and with their pre/post-conditions. The semantics of actions is described by the intermediary description and is respected by the composition algorithm both during design and execution phases.

Action semantics is described as I/Os (when SPIN conditions are not employed) or as IOPE with expressivity on a par with Pragmatic Proof (VERBORGH et al., 2016). Description of action semantics is specially important for SOAP services, but there is no consensus on the formalisms or expressivity, beyond I/Os. Heterogeneity in description of action semantics is again solved by translators, which convert heterogeneous descriptions to the intermediary description.

An alternative unexplored in Unserved are Multi-Context Systems (MCSs) (BREWKA; EITER, 2007). These allow the use of multiple contexts, each employing a different logic. Information flow between different contexts is allowed using bridge rules. Unlike the approach taken in Unserved, in a MCS there is no main context (or logic) to which the different contexts are mapped. A bridge rule can refer to facts from any context in its premise and its head applies to one of the multiple contexts. As discussed in (BREWKA; EITER, 2007), contexts and bridge rules may be non-monotonic[2]. However, bridge rules only add facts to contexts. (BREWKA et al., 2011) extend the theoretical framework so that bridge rules perform arbitrary actions, including deletion of facts.

SPIN conditions were used in place of MCS as these reuse many of the Unserved architecture and composition algorithm infrastructure. At the same time, the goal of the architecture (support for the three

---

[2] Let K_1 and K_2 be sets of formulas in a logic L such that $K_1 \subseteq K_2$. L is said to be *monotonic* if, and only if, for any formula Φ, $K_1 \vDash \Phi \implies K_2 \vDash \Phi$.

basic interaction models) is already achieved, as no basic interaction model mandates any description of action semantic more specific than I/O or finite actions (HTTP methods and Publish/Subscribe). Furthermore, although the SPIN conditions are not formally presented as a logic, inference rules are defined and the knowledge base maintained during design and execution could be integrated into an MCS.

Data

Data heterogeneity is a large issue, which itself demands research efforts greater than those already spent on this research itself. As discussed in Section 1.4, data heterogeneity is considered out of scope. Among the basic interaction models, only SOAP services mention data by requiring their syntax to be defined with XSD.

As discussed in Section 1.4, Unserved, like many previous attempts (ROMAN et al., 2015; UPADHYAYA et al., 2011a; RODRIGUEZ-MIER et al., 2016; VERBORGH et al., 2016), employs ontologies to deal with data heterogeneity. Nevertheless, Unserved does not explore semantic heterogeneity issues, such as ontology alignment (EUZENAT; SHVAIKO, 2013; OLIVEIRA et al., 2017). As for syntactic heterogeneity, a technique similar to lowering/lifting mappings from SAWSDL (KOPECKÝ et al., 2007) is employed through *parsers* and *renderers* components Subsection 4.1.2.

Lower-level protocol

The three basic interaction models mandate two protocols: HTTP (RESTful services, cf. Section 1.4) and SOAP (SOAP services). Also, cf. Section 1.4, only these two protocols are considered for EOSs. Like virtually all studies that deal with this heterogeneity that were identified in the SLR (Section 3.1), the middleware method is employed. In Unserved this corresponds to the selection of action runners for the `up:Send` and `up:Receive` activities. Although this was not explored, this same approach could be extended to protocols other than HTTP and SOAP.

5.1.2 Expressivity

In the absence of a standard benchmark for evaluating expressivity in service composition, this section surveys expressivity capabilities of the Unserved prototype (including SPIN conditions, cf. Subsection 4.2.2), PDDL-based composition approaches, Pragmatic Proof and the ComposIT prototype, in order to identify which capabilities are supported by each approach. manual composition with BPEL is left outside of this comparison given that, as being performed by a human specialist, it offers any expressivity capability that could be defined.

Expressivity capabilities can be divided in three features: service description, conditions and composite service specification. The first feature – service description– has three capabilities that can be identified: (1) the capability to describe inputs and outputs; (2) the capability to describe message structure; (3) the capability to describe Publish/Subscribe services. The first is supported by all considered approaches and is also required by all basic interaction models. The second is not supported only by ComposIT, whose prototype is oblivious to nesting of outputs. This second capability is a prerequisite to support the HATEOAS constraint and, therefore, to support the RESTful services basic interaction model. The third capability is supported only by Unserved and is a perquisite for supporting the EOS basic interaction model.

The second feature contains the most capabilities and, in the Unserved architecture, is provided using SPIN conditions. SPIN conditions, limited to conjunction and disjunction, have comparable expressivity to RESTdesc descriptions (used by Pragmatic Proof). The latter extends the RDF abstract syntax with formulas and support for additional magic[3] properties such as `log:implies` or `log:semantics`. N3 reasoners such as CWM (BERNERS-LEE et al., 2008) and EYE (VERBORGH; ROO, 2015) also support built-in predicates for string (e.g., `string:concatenate`) and math operations (e.g., `math:greaterThan`) [4].

N3 logic includes a predicate similar to negation, `log:notIncludes`, which tests if a N3 formula node includes the contents of another. This construct cannot be used in RESTdesc descriptions to model services that retract triples from the world state nor to forbid post-conditions from another service. The Pragmatic Proof algorithm also requires that universals in post-conditions also occur in pre-conditions (VERBORGH et al., 2016, p.20). This effectively limits the post-conditions to conjunction (as F enforces in Subsection 4.2.2). Disjunction in preconditions cannot be expressed in N3 logic. However, alternative RESTdesc implications can be used to achieve the same result of a single implication with disjunctive pre-condition.

---

[3] Such properties are said to be magic because they do exist in the RDF graphs processed by N3 reasoners but can still be queried as if they were always present.

[4] <http://www.w3.org/2000/10/swap/doc/CwmBuiltins.html>

Some automatic composition algorithms employ PDDL planners (PEER, 2004; KLUSCH; GERBER, 2005). Both of these works only cover features of classic PDDL version 1.2 (MCDERMOTT et al., 1998). PDDL includes three logical constructs not supported with the subset of the previously described SPIN elements (Subsection 4.2.2): negation (`not`), implication (`imply`) and universals (`forall`). PDDL also includes predicates for dealing with numeric values (`<`, `>=`), which are not supported by Unserved as it stands.

Effects in PDDL are closer to effects with SPIN conditions. The only features unsupported by the latter are universally quantified effects (`:conditional-effects`) and built-in predicates for numeric comparison. Conditional effects (`when` in PDDL) must be encoded in the conditions of the spontaneous message.

The last feature of expressivity capabilities is that of composite service specifications, where the only capability surveyed is specification of loops. Among all approaches, only PDDL supports this.

Among all expressivity capabilities, Unserved does not support only three: Negation, Math operations and Loops. Math operations can be added with an strategy akin to that of N3 logic, used by Pragmatic Proof. The other two have the use of ComposIT as a limitation. Negation would break the assumption over which the dependency graph and the state graph are built: a service cannot revert the effects of previous services. As for loops, they clash with the assumption in ComposIT that a service needs to be invoked at most once. If ComposIT were replaced by another algorithm without these limitations, negation could be added in SPIN conditions using `FILTER NOT EXISTS` constructs. The support for loops, on the other hand, must be provided by the composition algorithm itself.

Table 9 summarizes the previously discussed expressivity capabilities of PDDL, Pragmatic Proof, ComposIT and the Unserved prototype (including SPIN conditions, cf. Subsection 4.2.2). Of all expressivity capabilities, only *Message structure* and Publish/Subscribe are necessary conditions for the support of basic interaction models. The other capabilities increase the range of application scenarios that can be handled using an approach that supports them.

All expressivity limitations stem from the ComposIT algorithm or from the particular subset of SPIN elements supported in SPIN conditions. No limitation is implied by properties or relationships of the Unserved architecture that have a strong architectural weight (i.e., are constraints required for achieving heterogeneous composition). Therefore, the above limitations in expressivity can be overcome by future implementations of the Unserved architecture.

Table 9 – Expressivity capabilities of related composition approaches and Unserved.

Layer	**Capability**	**PDDL**	**Pragmatic Proof**	**ComposIT**	**Unserved**
Service	I/Os	✓	✓	✓	✓
	Message structure	✓	✓	✗	✓
	Publish/Subscribe	✗	✗	✗	✓
Conditions	And	✓	✓	✗	✓
	Or	✓	✓	✗	✓
	Negation	✓	✗	✗	✗
	Conditional Effects	✓	✓	✗	✓
	Math operations	✓	✓	✗	✗
Composite service	Loops	✓	✗	✗	✗

Source: the author.

5.2 SCENARIO

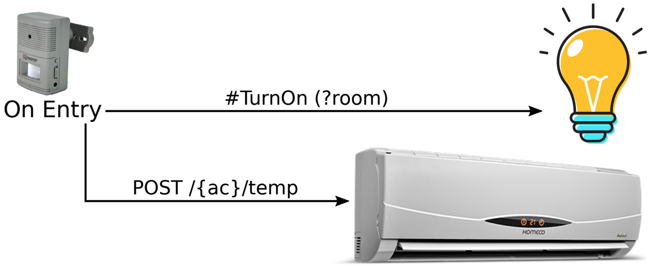
To evaluate applicability of the architecture, a smart house service composition scenario is used[5,6]. Such house has presence sensors, a remote controlled lighting and air conditioning system. All these systems are exposed as services. Each room has a presence sensor which publishes entries to an Atom feed (NOTTINGHAM; SAYRE, 2005) whenever someone enters or leaves the room. The lights are controlled by a central automation SOAP service. The air conditioning is controlled by a centralized gateway that, as an RESTful service gateway, lists all available air conditioning units and allows turning them on/off. Air conditioning units are controlled using the Mozilla Web Thing API

---

[5] Neither the house, nor the devices were physically used. Service descriptions were fed into the architecture and interaction was done with dummy implementations of these services

[6] This section limits itself to high-level discussion of the involved services and of aspects that enable their composition under Unserved with the ComposIT algorithm extended with SPIN conditions. More details can be obtained in Appendix E.

Figure 20 – The evaluation scenario.



Source: the author

(FRANCIS, 2017). The goal in this scenario is to turn on air conditioning and lighting of any room whenever a person enters that room.

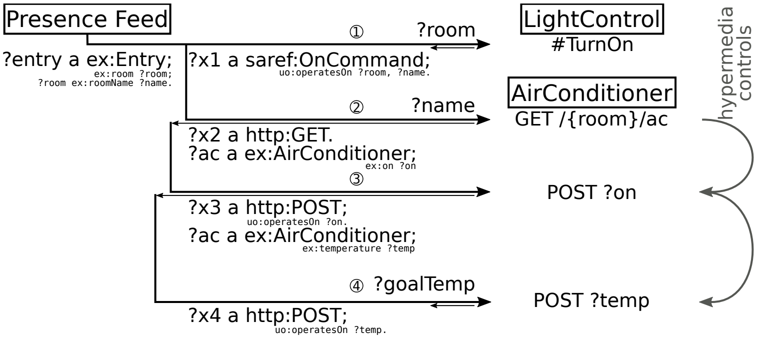
Figure 20 shows an overview of the desired composite service. Every time someone enters a room, the light is turned on and the air conditioner is set at a specific temperature. The user gives as input the desired temperature for the air conditioner and the description of composition goals. These goals take the form of `u:Variable` instances that are also instances of `u:Wanted` and have SPIN conditions describing their desired relations. At a high level, the goals can be summarized as:

1. An entry event, with a room resource;
2. An instance of `saref:OnCommand` that operates on the room;
3. An instance of `httpm:POST` that operates on the `ex:temperature` of an air conditioner and on the desired temperature.

The execution of the desired composite service is shown in Figure 21. The Presence EOS is implemented as an Atom Feed with a single category with `atom:term` `ex:Entry`. From the domain ontology, it is known that this class is domain of `ex:room`, which has range in `saref:BuildingSpace` class, which is domain of `ex:roomName`. The composite service listens for events originating from this service and uses the `ex:room` and `ex:roomName` properties in the next steps of the execution.

The SOAP service responsible for controlling the lights takes the `saref:BuildingSpace` present in `ex:Entry` instances and turns on the lights on that room (① in Figure 21). The most relevant output of this service is not data, but the action of turning lights on. This

Figure 21 – The composite service execution under the scenario.



Source: the author

is represented with a surrogate output that has `saref:OnCommand` as its type and with the `uo:operatesOn` assertion for all inputs of the service.

The RESTful service that controls the air conditioning system handles multiple air conditioners. First, one must obtain the air conditioner for a room using an URI template that takes the room name (`ex:roomName`) as parameter (② in Figure 21). An air conditioner resource describes the device state and offers two hypermedia controls: one for powering the device on or off, and another for setting its temperature. If the device is not powered on, the hypermedia control for setting the temperature will not be offered. All HTTP operations also generate surrogate outputs. However, the semantic of the operation is expressed by the HTTP method itself, forgoing the need for a domain-specific action ontology. In this scenario, the device is powered off. Therefore, during execution, as result of adaptation, the device is first turned on (③ in Figure 21) and then has the temperature set (④ in Figure 21).

This scenario demonstrates composition of an EOS, a SOAP service and an RESTful service that employs both HATEOAS and an URI template (common in Web APIs that achieve at most protocol level in RMM). However, an often neglected perspective of scenario evaluations (as is the case of studies in the SLR, cf. Subsection 3.1.6) is when methods fail to handle a given scenario. Variations of this scenario can be created, making the current Unserved architecture implementation, the current composition algorithm or the service description languages unsuitable. In the following, some of these problems are discussed, together with proposed solutions.

Limitation 1. Incomplete EOS description. Usually, the event-oriented nature is attributed by natural language documentation. Atom feeds, on the other hand, do not describe the event contents further than their Media Type an category. W3C WoT Things Description (KAEBISCH; KAMIYA, 2017) describe event contents (i.e., which properties are actually included), but are designed for a specific context (IoT).

- *Source*: Descriptions for EOS in use;
- *Consequences*: If the room name was defined using `rdfs:label` instead of `ex:roomName`, composition would not be possible. If `saref:BuildingSpace` were the domain of hundreds of irrelevant properties, intermediary description by the Atom translator would be wasteful;
- *General Consequences*: A service may be incorrectly translated as event-oriented or non-event-oriented. Inaccurate intermediary descriptions w.r.t properties included;
- *Solutions*: Crete new languages for description of EOS. Wide acceptance of such languages may be a challenge, as service composition is only one among concerns of service designers.

Limitation 2. Discovery during composition. Most standardization efforts towards WoT concentrate on discovery and description of devices and their hosted services. Typical IoT composition scenarios involve only a few services selected according to context associated with their devices (e.g., location). Therefore, it is more efficient to discover these devices and services dynamically, during or immediately before composite service design, rather than building and maintaining a service repository with all services of all devices. Unserved employs the latter approach, static discovery.

- *Source*: The Unserved architecture;
- *Consequences*: If the air conditioner control provided only a list of W3C's WoT thing descriptions, composition would not be possible;
- *General Consequences*: All possibly relevant services must be added to the service repository before composition is requested;
- *Solutions*: Request a composition whose goal is to find service descriptions, feed all resulting descriptions back into the service repository and request a new composition with the actual goal.

This workflow could be integrated into the Unserved architecture itself for added convenience.

Limitation 3. Retry service after satisfying violated preconditions. If the state of affairs known to the interpreter differs from that known by a service, the service may fail due to pre-condition violations. The interpreter, however, will believe the failure is irrecoverable and will not try to update its state and achieve precondition.

- *Source*: Interpreter in the Unserved architecture;
- *Consequences*: If the hypermedia control for setting the air conditioner temperature were to be always offered, or if the service was a SOAP service, then the composer would likely plan to directly invoke such operation. If the device is not yet powered on, composition would fail permanently, without trying to power the device on;
- *General Consequences*: If the world state changes during execution of a composite service or in between the composition request and its execution, the user goals may never be achieved as the execution does not differentiate between recoverable failures (pre-condition violations) and irrecoverable failures (unavailable services);
- *Solutions*: As a first step to any solution, all pre/post-conditions should be present in service descriptions. The first alternative is to ensure that composition requests either include accurate information about state or that they are made in such way that enforces the composer to plan their querying from respective services. This solution relies mostly on the Unserved end user. The second alternative would be to somehow determine if the service failed due to a precondition violation or due to an actual error. If the failure is a precondition violation, the interpreter would then determine which condition failed, either by parsing the service error message or by spawning a second composition that fetches updated state from services. With the preconditions state updated, the interpreter would notify the composer that the service previous failure should be ignored and request an adaptation of the current plan.

5.3 CONTROLLED EXPERIMENTS

For controlled experiments, a prototype of Unserved was implemented in Java version 1.8[7]. The Java language was chosen due to its wide popularity and due to the availability RDF tooling. All experiments were conducted on an Intel i5-3230 processor running at 2.5GHz. The machine had 8GB of RAM, but the single JVM process running the prototype had a limit of 4GB heap memory. The disk unit was magnetic, with 5400 RPM and a SATA 3.0 interface. Write speed for a 2 GB file in blocks of 8 KB was 65.1 MB/s, while read speed was 55.2 MB/s. Source code, test data and binaries, not including maven repository cache, amount to 895MB. Results and a helper script to replicate the results is available online[8]

The JVM includes non-deterministic processes, i.e., Garbage Collector (GC) and Just In Time (JIT) compilation, that can interfere with timing collection. Every measurement occurs in a JVM which will be used only for that single measurement. Before the actual measurement, two preheat runs of the whole process under measurement are run. After preheat, loaded data and caches are flushed (asserted by near-zero heap usage after each preheat), GC is run and system disk buffers are flushed with the `sync` UNIX utility. This strategy is applied to both the ComposIT and Unserved prototypes[8].

The prototype includes a forking and adaptive implementation of variations of the ComposIT algorithm. Each variation consists of a choice for each of the following factors:

1. SPIN condition support: (Yes/No);
2. Cost function: Service count (used in ComposIT (RODRIGUEZ-MIER et al., 2012)) or response time;
3. Search algorithm:
    - A* (HART; NILSSON; RAPHAEL, 1968);
    - D*-Lite (KOENIG; LIKHACHEV, 2002; KOENIG; LIKHACHEV; FURCY, 2004);
    - *Greedy without adaptation* (CHATTOPADHYAY; BANERJEE; BANERJEE, 2015). This algorithm generates only the best possible state when exploring the state graph, thus restricting the adaptation to a single retry of failing services;
    - *Greedy*: the search algorithm is as described by Chattopadhyay, Banerjee & Banerjee (2015). However, all preceding states are generated, not only the locally optimal.

When comparisons are made with the original ComposIT prototype, the implementation without SPIN conditions, using A* and

---

[6] <https://bitbucket.org/alexishuf/unserved-testbench>

[7] <https://bitbucket.org/alexishuf/unserved-testbench-dissertation>

[8] The scripts for running experiments apply an instrumentation patch, including these procedures, to the ComposIT prototype.

service count as cost function is used. When comparing to Pragmatic Proof, the variation with D*-Lite and SPIN conditions optimizing service count is used.

The reimplementation under Unserved was not a translation of the existing code, but a new implementation of the same algorithm with details adjusted to Unserved. These changes apply to all variants, including Greedy without optimization. Four changes to implementation details, listed below,had a positive impact on performance in comparison to the ComposIT prototype:.

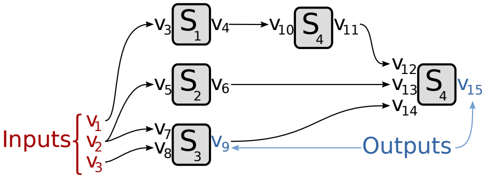
1. Construction: Storing predecessor information per service node;
2. Construction: Storing successor information per service node;
3. Construction and Planning: No subclass listing;
4. Planning: *Jump nodes* per input, not per-service.

The first two changes simply store information that is obtained during dependency graph construction, but that in the original ComposIT implementation is re-computed, if needed. This change is also important, since, in order to support execution, the mapping between outputs and inputs, not only between services, will be used in the output of the planning phase. The second change has an important impact specifically in the optimization phase.

The ComposIT original prototype lists sub classes of a class during Online Node Reduction. This has a negative impact on performance amplified by the number of classes on the taxonomy and the height of the taxonomy. The implementation under unserved avoids this by reusing an index of service outputs populated during the construction phase using only super-class queries.

The final change consists in creating one jump node for each input not satisfied by services in the preceding layer. The original implementation did so for each service nor in the preceding layer. In the benchmark used for evaluating ComposIT, WSC'08 (BANSAL et al., 2008), this had an expressive impact on problem 6. However, SPIN conditions require a single jump node for the whole service. This requirement is only valid when the service for which the jump node is a predecessor has preconditions defined. The implementation using SPIN conditions uses this fact to decide which jump node strategy to employ for each service. Both the ComposIT implementation and that under unserved create jump nodes on demand while walking through the state graph.

Figure 22 – A simplification of a solution to problem 2 in WSC’08.



Source: the author.

5.3.1 Web Services Challenge 2008

The Web Services Challenge (WSC)’08 benchmark (BANSAL et al., 2008) is currently the most widely accepted benchmark that focuses on functional composite service design. This benchmark was also used for evaluating ComposIT (RODRIGUEZ-MIER et al., 2012; RODRIGUEZ-MIER et al., 2016). The benchmark consists of 8 problems, each characterized by a set of services, a set of known inputs and of goal outputs. Expressivity of service descriptions is limited to I/O: each service is described by a name, the set of required inputs and the set of produced outputs. I/Os are described with types within a type hierarchy, similarly to an ontology. Outputs or known inputs can be use as input of a service if, and only if, they are of the same type or are a sub-type of the type of the service’s input. The same rule applies to goal outputs. In the experiments executed with Unserved, the type hierarchy of each problem is converted into an OWL ontology.

Solutions in WSC’08 are workflows that simply connect inputs and outputs of services. Figure 22 shows a simplification of the solution to problem 2 (several inputs outputs were discarded for readability). Variables v_1, v_2, v_3, are given as problem inputs and v_9, v_{15} are the goal outputs. Executing services from left to right with the mapped input assignments yields values for the goal outputs. The solution in Figure 22 uses five services and has a length of three (as S_1, S_2 and S_3 can be invoked in parallel).

Table 10 shows numeric properties of each WSC’08 problem as well as properties of the official solutions provided with WSC’08 (two last columns). The counts for inputs and outputs is the sum for all services: 5 services, each with 3 inputs amount to 15 inputs. For all problems, the type hierarchy has a single root. The height of the type hierarchy is the number of nodes (including start and end node) of the longest path following `rdfs:subClassOf` edges in the types ontology.

Table 10 – Numeric properties for each WSC'08 problem.

Problem #	Problem					Solution	
	Services	Inputs	Outputs	Types	Hierarchy height	Services	Length
1	158	735	778	1540	14	10	3
2	558	2972	2890	1565	12	5	3
3	604	3254	3129	3089	15	40	23
4	1041	5781	5611	3135	18	10	5
5	1090	5816	5953	3067	14	20	8
6	2198	12218	11831	12468	16	40	9
7	4113	22324	22392	3075	14	20	12
8	8119	44569	44628	12337	16	30	20

Source: the author, computed from the WSC'08 dataset[9]. Last two columns were reported in (RODRIGUEZ-MIER et al., 2012).

In general, WSC problem complexity increases with each new problem. However, this increase does not have a linear relationship with the problem number. Among the properties shown in Table 10, problem properties have a larger impact on problem difficulty than solution properties. This holds for both the original ComposIT prototype and for the Unserved prototype using any configuration of algorithm.

As Table 10 shows, problem 1 is the simplest. Problem 2 increases the number of services, but reduces the number of services in the solution, keeping the number of types similar. Problem 3 increases the type count and both the solution length and size. Problem 4 increases the number of services and grows the type hierarchy, but simplifies the solution. Problem 5 introduces the applicability of Online Node Reduction, which remains in all subsequent problems. Problem 6 nearly doubles the number of services and increases the number of types by four times, requiring a solution with many services. Problem 6 is also notable for requiring many jump nodes. Problem 7 returns the number of types and hierarchy length to levels similar to problem 5, but increases the number of services by four. Problem 8 doubles the number of services in problem 7, decreases the number of services in the solution by a third and nearly doubles its length.

---

[9] <https://github.com/kmi/sws-test-collections>

Table 11 – Desired comparisons between algorithm variations and ComposIT. RT stands for Response Time.

Algorithm A	Algorithm B	Metric
ComposIT	A*	all phases
A*	D*-Lite	planning
A*	D*-Lite	first re-plan
A*	A* with SPIN conditions	all phases
A*	Greedy	planning
A*	Greedy without adaptation	planning
Greedy	Greedy without adaptation	planning
A* (RT)	Greedy (RT)	planning
A* (RT)	Greedy (RT) without adaptation	planning
Greedy (RT)	Greedy (RT) without adaptation	planning

Source: the author.

5.3.1.1 Experimental Design

Given the large number of algorithm variations, a strategy for planning experiments had to be devised in order to draw conclusions with available resources. Table 11 shows the desired comparisons between composition algorithm variants implemented under Unserved and with the ComposIT prototype. Between pairs of variations that change only the search algorithm, the desired comparison metric is the time of the planning phase. The comparison between A* and its counterpart supporting SPIN conditions encompasses all phases, since there are changes in data structures at the dependency graph construction phase that propagate to all other phases. Finally, A* and D*-Lite are also compared in the time to re-plan a failed composition. The D*-Lite algorithm is notified of changes in the state graph and is designed to reuse already found optimal paths if the previously optimal path is broken or becomes sub-optimal.

Each combination of an algorithm variation and a WSC'08 problem defines a population whose confidence intervals for the mean should be identified with sufficient precision to identify which variations have better performance. In order to determine sample size for each population, a pilot study with 9 samples for each population was conducted. With these results, the sample sizes were calculated using the standard textbook equation for an infinite population:

$$n_0 = \frac{z_\gamma^2 \sigma^2}{E_o^2} \tag{5.1}$$

The desired confidence level is 95%, which yields $z_\gamma = 1.96$ and σ is aproximated using the standard deviation of the pilot sample. E_0 should be the largest number such that, for any metrics under which an algorithm α is evaluated, E_0 is greater than 5% of the point estimate for that metric, the confidence intervals of compared algorithms are apart by at least 40% of the distance between metrics, and n_0 forthe samples of both algorith is the least necessary. The 5% threshold serves to avoid prohibitively large sample sizes required by when metrics have close values. The 40% threshold is a safeguard against any possible bias due to the pilot sample size, which is small.

In order to compute n_0 for algorithm α and WSC'08 problem p, n_{α_1} and n_{α_2} are computed for each comparison between algorithms α_1 and α_2 in Table 11. These values are computed by starting with $n_{\alpha_1} \leftarrow 9$ and $n_{\alpha_2} \leftarrow 9$, and then successively increasing n_{α_1} and n_{α_2} until the 40% error distance condition is met or the error reaches the 5% limit. If the 5% limit is reached for one of the algorithms, the other continues having n incremented. After these computations, n_0 for algorithm α and problem p is set as the maximum n_α observed. Due to technical details, ComposIT has samples with size 35 for all problems. This number is the largest n required to ensure an error margin of ±5% for any problem, following the same equation mentioned above.

Previous designs of the experiments with WSC reported in this section have shown that in most cases, normality of population for all metrics cannot be assumed. Textbook Confidence Intervals (CIs), obtained with a t-test, assume that population follows an approximate normal distribution (since the test is applied to a single mean). Lumley et al. (2002) argue that the normality requirement can be safely ignored if the sample size is large enough. However, while some variations and problem combinations (problems 5 and 6 mostly) had large sample sizes, only 15% of all population samples have a size larger than 30. Given this situation, CIs are computed through the adjusted bootstrap percentile non-parametric (i.e., does not rely on assumptions about the population) method (EFRON, 1987). For all analysis conducted in the following sections, CIs have a confidence level of 95% computed through bootstrapping with 4000 replications.

Given the small sample sizes for most populations, the pilot samples were added to the newly obtained samples, to increase the reliability of conclusions. Table 12, shows for each algorithm variation,

Table 12 – Sample sizes for algorithm variations and WSC'08 problem.

Algorithm Variation	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ComposIT	35	35	35	35	35	35	35	35
A*	18	19	19	22	372	631	22	19
D* Lite	27	27	31	27	27	32	32	27
A* (with re-plan)	30	33	19	23	21	78	19	18
D* Lite (with re-plan)	29	39	18	23	281	36	19	19
A* with SPIN conditions	18	18	19	22	32	54	22	19
Greedy	22	18	18	18	22	34	18	18
Greedy w/o adaptation	18	18	18	19	18	19	19	19
A* (RT)	18	18	18	19	18	22	42	18
Greedy (RT)	18	18	19	19	18	20	18	19
Greedy (RT) w/o adaptation	18	18	18	18	19	19	18	18
D*-Lite (RT)	18	18	22	18	18	23	23	18

Source: the author.

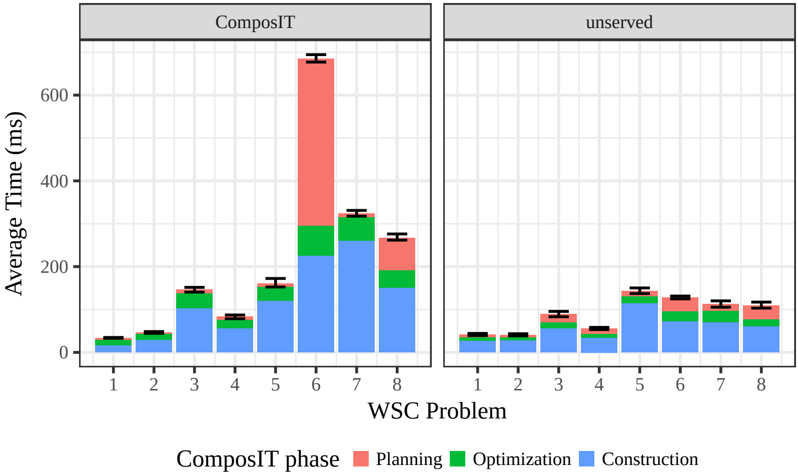
the minimum and maximum sample size, along with the corresponding WSC'08 problem.

5.3.1.2 ComposIT under unserved

The first experiment compares performance of the original ComposIT implementation to its forking and adaptive implementation under Unserved, without the use of SPIN conditions. Figure 23 shows the averages for each WSC'08 problem, decomposed by phase of the ComposIT algorithm. Error bars show the CI for the mean time taken by the whole design process. For all problems the CIs of ComposIT and its reimplementation do not overlap, which indicates that the whether mean values increase or decrease under Unserved can be generalized. Only problem 1 has the mean increasing under Unserved, while for all other problems the mean decreases.

Figure 24 shows the changes in means for all ComposIT phases. In problem 1, the mean increases by 23.14% (or 7.83 ms) under Unserved. For all other problems the decrease ranges from 10.90% (or 17.58 ms) to 81.34% (or 557.72 ms). The most expressive decreases in the mean, both in percent and on absolute values are observed on the problems with the highest difficulty from the standpoint of ComposIT.

Figure 23 – Breakdown by phase for ComposIT and its reimplementation under Unserved.

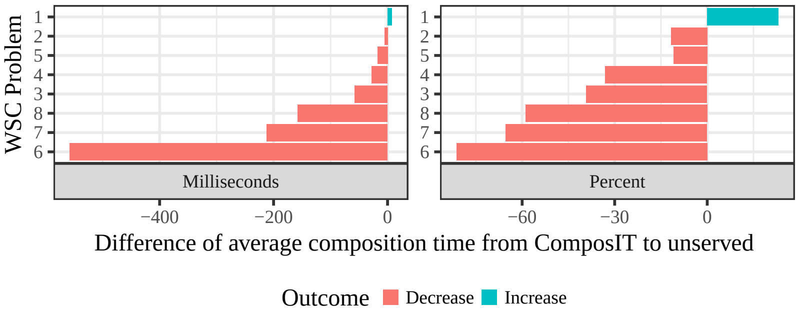


Source: the author.

Figure 25 details the decrease (or increase) in mean per ComposIT phase. While planning under Unserved is slightly more complex (since mappings between inputs and outputs must be preserved for execution), problems 8 and 6 benefit from changes introduced in the implementation under Unserved. Problem 6 owes the performance gain mostly to the use of a jump node per input, instead of per service. Both problems benefit from the absence of subclass listing during Online Node Reduction, which takes place only on problems 5, 6, 7 and 8. In all problems but 6 and 8, planning time increases. However, the absolute values of the increases are small, ranging from 2.39 ms (problem 2) to 9.71 ms (problem 3). A final observation about increases, is that the largest absolute value increases occur in the problems with the longest solution path: problems 3 and 7. Problem 3 involves no Online Node Reduction, while problem 7 involves, but has a taxonomy of classes roughly four times smaller than problems 6 and 8.

The breakdown of mean differences per phases from Figure 25 also reveals the efficacy of storing successor information during construction phase on reducing the time required for the optimization phase. The performance gains range from 40.76% (or 5.41 ms) in problem 1 to 65.92% (or 46.55 ms) in problem 6.

Figure 24 – Change in mean time of each WSC problem for ComposIT reimplementation under Unserved in comparison to the original ComposIT prototype.



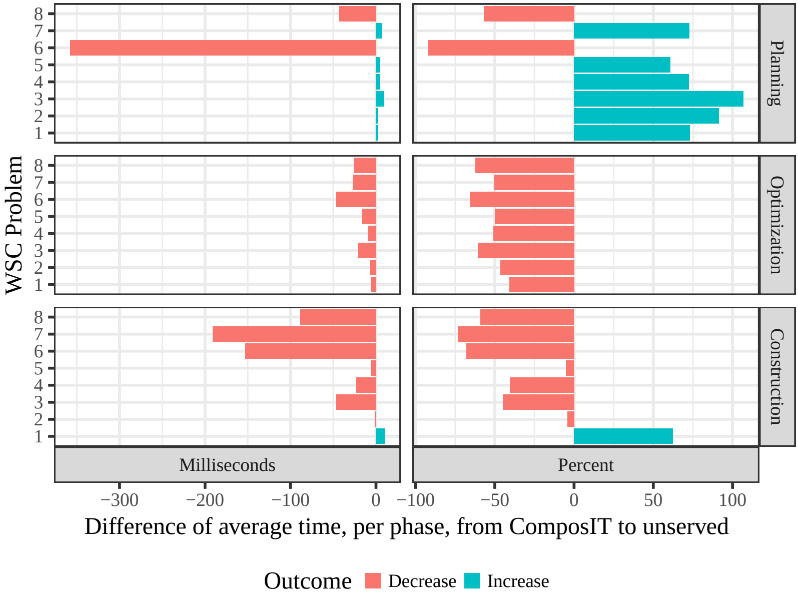
Source: the author.

As for construction, while changes described in Section 5.3 transfer workload to this phase, all problems but problem 1 had a decrease in mean time. The precise origins of this decrease are diffuse. In general the implementation under Unserved performs nearly no redundant access to type taxonomies, e.g. super-classes are only computed once for any type. In addition this phase received intensive lower-level performance tuning, without changes that alone had impact comparable to the other changes discussed.

Figure 26 shows the Probability Density Functions (PDFs) for each sample of total time for each problem for both the ComposIT prototype and its Unserved reimplementation. For each sample, bullets mark the mean and lines mark the percentiles that would correspond to $\mu - \sigma$, median and $\mu + \sigma$ in a theoretically normal distribution with same variance. Although normality cannot be assumed, the percentiles that correspond to the 68.27% coverage of standard deviation provide an adequate benchmark for comparison.

Regarding the 68.27% coverage,for five of the problems (3, 4, 6, 7 and 8), the interval for unserved does not intersect with that of ComposIT. This indicates that in addition to the means being smaller, individual observations will often be smaller than observations with the ComposIT implementation. As for the problems where these intervals intersect, problems 1 and 2 have small absolute values and problems 1 and 5 have clearly bimodal distributions.

Figure 25 – Change in mean time per phase for ComposIT reimplementation under Unserved.



Source: the author.

Traces of bimodality can be observed more clearly in the PDFs of problems 1, 3, 5 and 6. Problem 5 is the one most affected by this anomaly: while its mean under Unserved is smaller than under ComposIT, half of the observations are above the mean under ComposIT. The cause of this bimodality was not identified, but it was traced to the particular task of obtaining the nodes that constitute the next layer during dependency graph construction[10]. Neither prototype explicitly employs randomized algorithms (CORMEN et al., 2009, p. 122). The cause is not likely to be external, since this anomaly is present on replications on the same machine or on another machine[11].

Two hypothesis for the increased variability in some problems under Unserved can be constructed. The first is erratic behavior of hash sets and hash maps, whose repeatable behavior across JVM instances

---

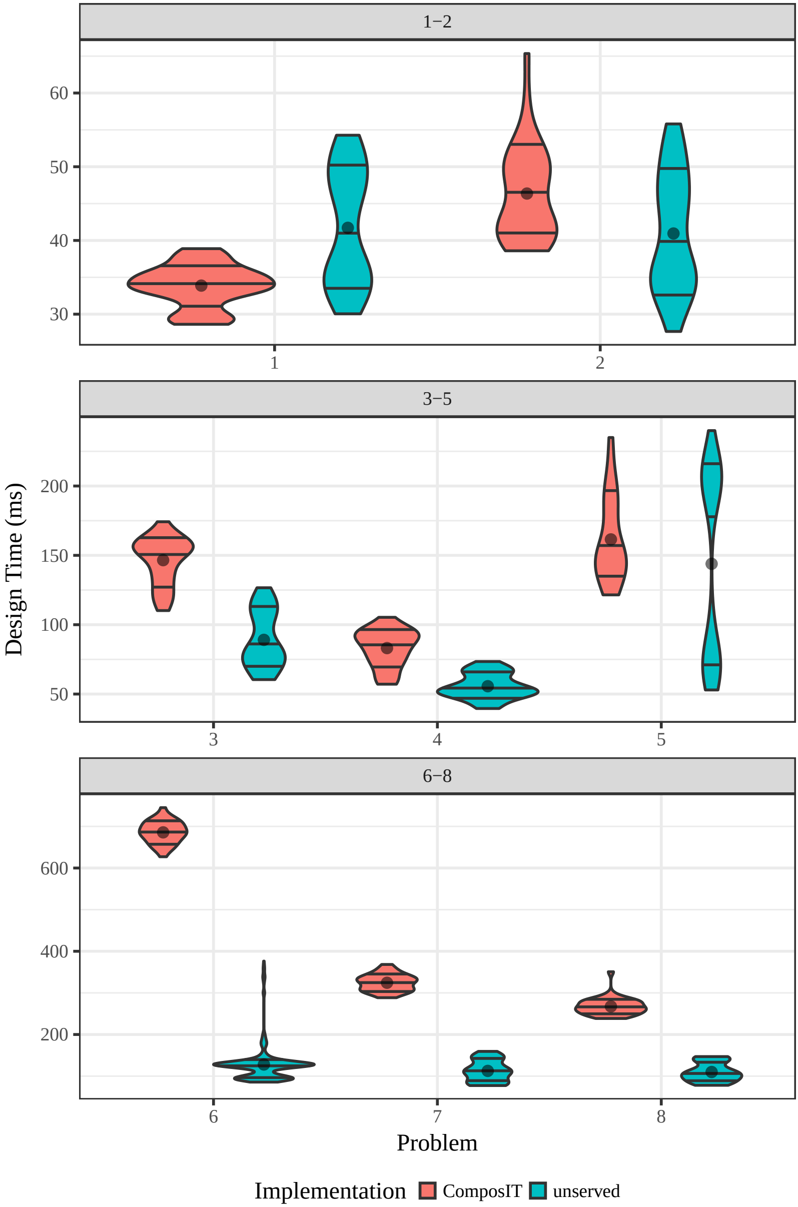
[9] Method `FastIOLayersState.MyNodesSupplier::getNodes`

[10] This machine has an earlier i5-2400 limited to 2.5GHz, 4GB a and a faster disk (123 MB/s write and 113 MB/s read).

depends on repeatable object hash codes. Under Unserved most hash codes are not repeatable since they depend on the hash code of blank nodes, prevalent in intermediary descriptions. Blank nodes as implemented in Apache Jena have their hash code derived from a randomly assigned Globally Unique Identifier (GUID).

The second hypothesis for increased variability is interference from JVM JIT compiler and/or GC. While the same countermeasures to avoid this interference have been applied to the ComposIT and to the Unserved prototypes, the Unserved prototype contains a larger code base active during composition and uses more heap memory. These two factors may increase the chance of GC or JIT running during the activities that are being timed. Results obtained in Subsubsection 5.3.1.3 favor the latter hypothesis.

Figure 26 – PDFs for each WSC problem and implementation sample. Horizontal lines are the 15.87% percentile, median and 84.13% percentile. Transparent bullets mark the mean.



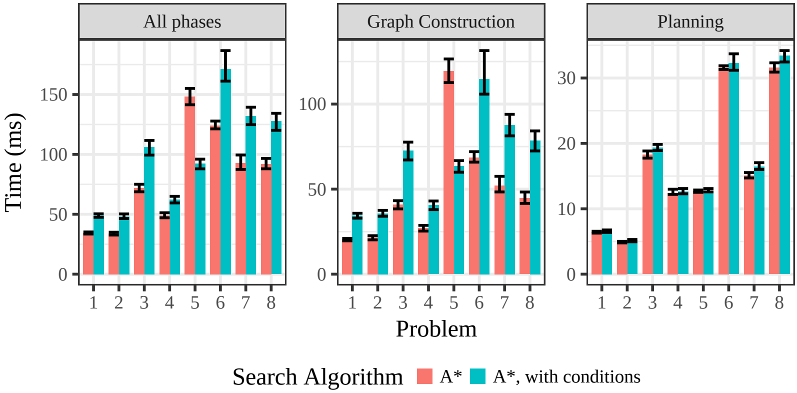
Source: the author.

5.3.1.3 A* with SPIN conditions

A second comparison is between A* and A* with support for SPIN conditions, both implemented under Unserved and optimizing for service count. The goal of this comparison is to evaluate the performance impact incurred by the additional data structures and processing required to support SPIN conditions. However, as WSC'08 does not include pre-conditions in its services, their evaluation during planning is not evaluated. An evaluation that includes preconditions is conducted in comparison with Pragmatic Proof, in Subsection 5.3.2.

The changes introduced to support SPIN conditions concentrate on the dependency graph construction phase and on the planning phase. During construction, post-conditions are indexed and pre-conditions are mapped to suitable post-conditions. During planning, states are filtered according to whether they satisfy pre-conditions of their succeeding state. In the case of WSC'08 there are only post-conditions, derived from the hierarchical structure of WSC'08 service descriptions. Under unserved, WSC'08 services are modeled as returning a single result resource associated through a bogus RDF property to all other resources which are the actual outputs of the service. As a result, every service with n inputs also has a conjunction of n triple patterns as post-condition.

Figure 27 – Mean times for the whole design process and for construction and planning phases, for A* without and with SPIN conditions.



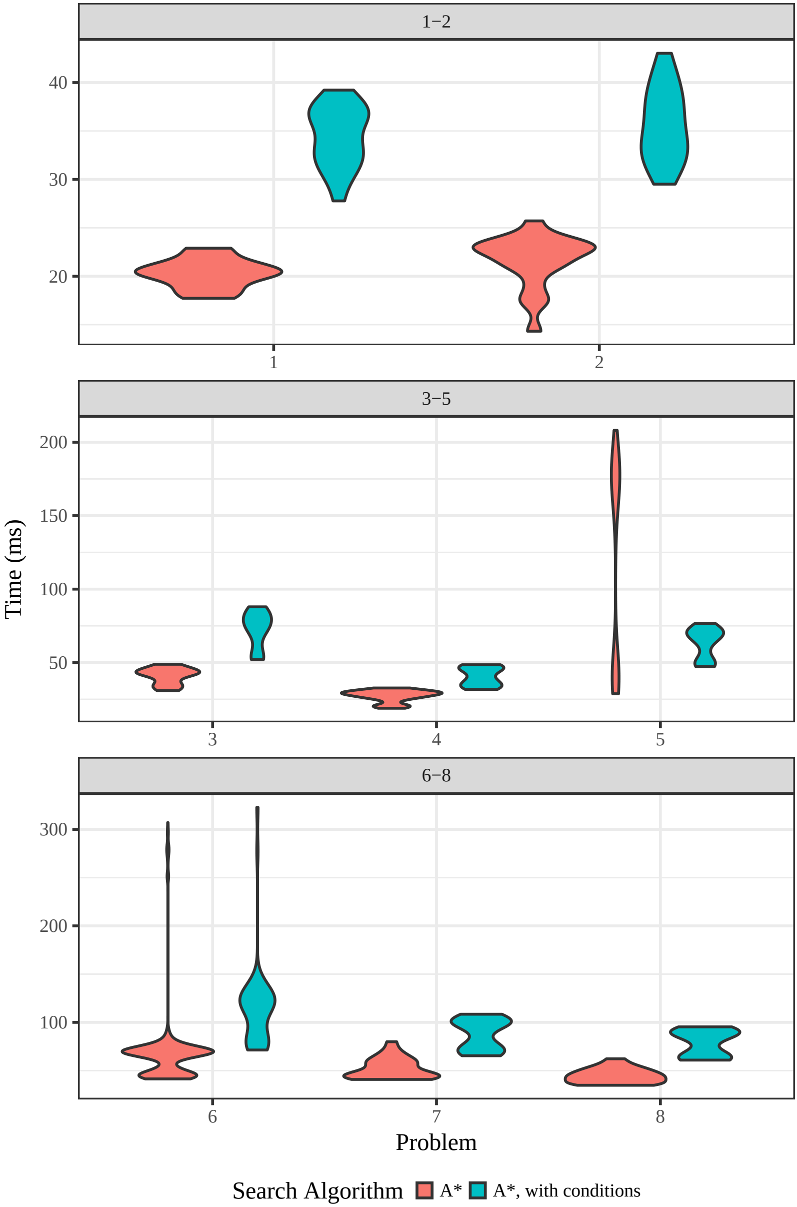
Source: the author.

Figure 27 shows the means, with bootstrapped 95% CIs for the whole design process as well as for the construction and planning phases. In the case of the planning phase, problems 3, 7 and 8 have their mean planning time increased by the SPIN conditions machinery. However absolute mean differences are small, between 5.77% (1.82 ms) and 5.83% (1.05 ms).

The differences in total design time are originate are determined by the changes in the construction phase. For all but problem 5, the mean in the construction phase increases. This creates a increase, in overall design time ranging between 26.8% (13.15 ms) and 37.9% (47.07 ms). In the case of problem 5, there is a decrease in construction that causes the overall design time to decrease by 37.81% (56.09 ms).

The decrease in problem 5 is counter-intuitive, but its cause can be traced to the absence of bimodality, as shown in Figure 28. However, adding support for SPIN conditions does not reduce variability or bimodality. In contrast, problems 1, 3, 7 and 8 have bimodality more evidenced when SPIN conditions support is present. These results support the hypothesis of JVM non-deterministic process interfering with variability but is not a proof.

Figure 28 – PDFs for total design time using A* with and without SPIN conditions.



Source: the author.

5.3.1.4 Other algorithms

Figure 29 compares all search algorithm alternatives under Unserved, without SPIN conditions and minimizing the service count. Search algorithms can be roughly sorted in decreasing order according to the planning time: D*-Lite $\geq$ A* $\geq$ Greedy $\geq$ Greedy without adaptation. CIs for D*-Lite and A* mean time only do not intersect in problems 1 (easy planning) and 3 (long path). The difference between means in this case ranges from 5.4% (0.347 ms, problem 1) to 6.74% (1.231 ms, problem 3). The Greedy algorithm is faster than A* in most problems, with its mean being between 13.82% (0.679 ms, problem 2) and 38.64% (12.202 ms, problem 8) smaller. Greedy and A* have intersecting CIs only in problems 1 and 7.

Removing the generation of all predecessor states, so that only the best state is generated, as is described by (CHATTOPADHYAY; BANERJEE; BANERJEE, 2015), provides the best performance results for all WSC'08 problems. However, the implementation of the Greedy algorithm without adaptation under unserved failed to reproduce results shown in (CHATTOPADHYAY; BANERJEE; BANERJEE, 2015), where all times remained below 1 ms. Furthermore, relationships between WSC problems are different under Unserved. The results originally reported point problem 1 as the most difficult, while problem 2 is slower than problems 4, 5, 7 and 8. These results are inconsistent not only with the results of this dissertation, but also with the results in (RODRIGUEZ-MIER et al., 2012) and (RODRIGUEZ-MIER et al., 2016). As the greedy algorithm paper provides only a high-level description of the algorithm and no prototype could be found, the reasons for this disagreement remain unknown.

Comparing the mean time for all three phases, differences between algorithms are not so evident. CIs intersect for A* and D*-Lite in all problems. Greedy only retains non-intersecting CIs with A* in problems 4 and 8. Greedy without adaptation retains non-intersecting CIs with Greedy only for problems 1, 2 and 6.

The paper that proposes the Greedy (without adaptation) algorithm (CHATTOPADHYAY; BANERJEE; BANERJEE, 2015) also evaluated the algorithm for optimizing response time, using a modified version of the WSC'08 dataset with randomly assigned response times for each service. However, as the dataset could not be obtained, a new version had to be built. Being a new experiment, assuming all response times for all services conform to a single normal distribution is unrealistic. While not a definitive solution, response time was assigned using five different normal distributions, whose parameters where ob-

Figure 29 – Mean time for whole design process and planning phase, for A*, D*-Lite and Greedy search algorithm, when optimizing service count.

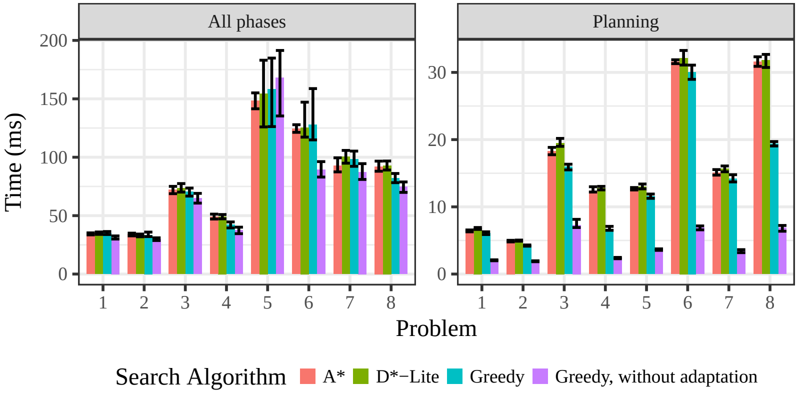


Table 13 – Service Response Time classes

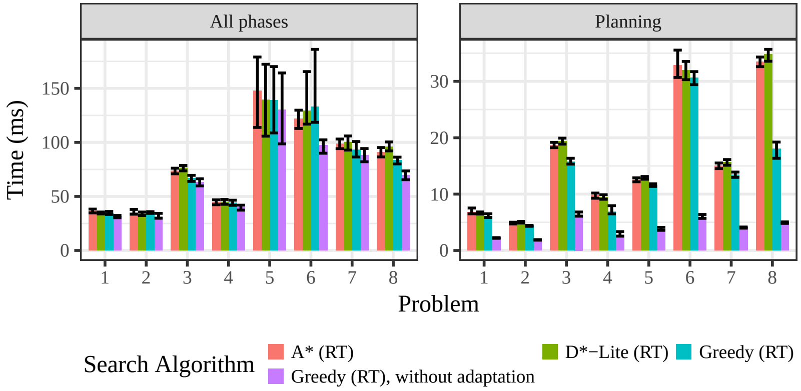
Class	**Host**	**Mean** (μ)	**Std. Dev.** (σ)
1	www.ufsc.br	22.253	4.507
2	www.adu.edu.az	34.082	12.452
3	roundtablepizza.mn	208.057	10.043
4	www.rtc.cv	274.967	2.146
5	www.btc.gov.bd	422.442	5.893

Source: the author.

tained with Internet Control Message Protocol (ICMP) *ping* requests to different services (to model diverse routes in the Internet). The five distributions, chosen with equal probability are shown in Table 13.

Figure 30 shows the results for the same search algorithms as in Figure 29 but minimizing response time. All confidence intervals from Figure 30 intersect with the corresponding intervals in Figure 29. The rough ordering of algorithms by mean time remains unaltered except for a statistical tie between A* and D*-Lite in all scenarios. A* and Greedy algorithms only have intersecting CIs in problems 3 and 8. The Greedy algorithm without adaptation has lower mean than and non-intersecting CIs with the plain Greedy algorithm for all problems. This

Figure 30 – Mean time for whole design process and planning phase, for A* and Greedy search algorithm, when optimizing response time.



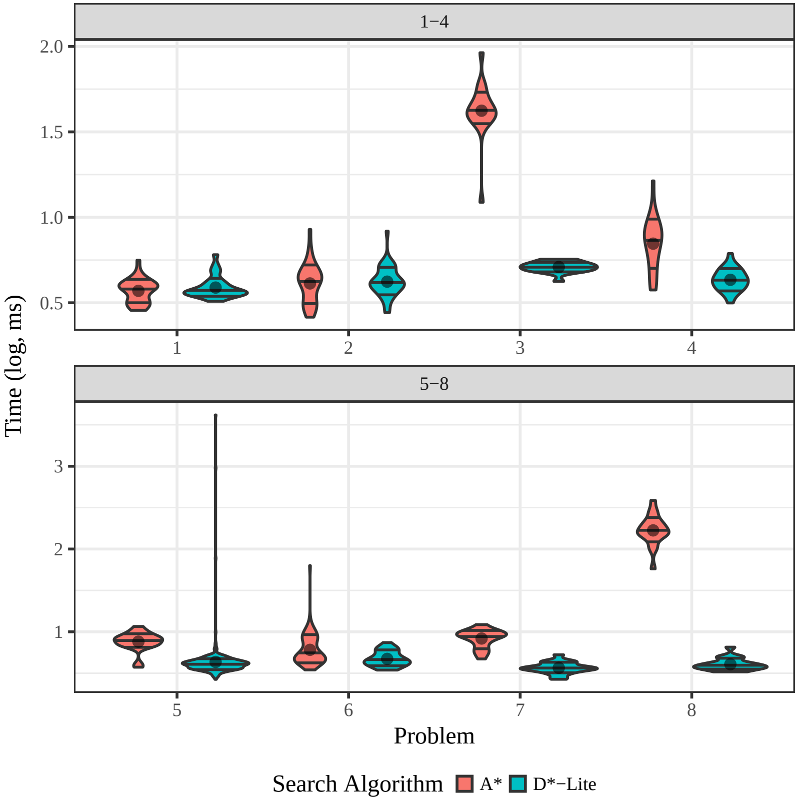
advantage of Greedy without adaptation is only visible for the mean of the whole design process in problems 1, 4, 6 and 8.

5.3.1.5 Re-planning

This experiment compares the variations of ComposIT implemented under the Unserved that use the A* search algorithm (like the original) and the D*-Lite algorithm. To compare both algorithms, a failure was induced during execution so that the first invoked service did not return any result. This forced a plan adaptation, in order to select an alternative service. In the following discussion, only the time required by each algorithm is compared. Since re-planning occurs after the state graph has been partially materialized in memory, observed times are substantially smaller than those for the planning phase in Figure 23.

Figure 31 shows the PDFs of each sample. In contrast to A*, the PDFs of all D*-Lite samples have lower amplitude. Skewness is for D*-Lite samples is positive (tail upwards), for all but problem 3. Problems 2 and 7 have only a slightly positive skewness value and are nearly symmetrical. No sample provides reasonable evidence of a normally distributed population, as was the case in previous experiments. For the same reason as earlier, the 68.27% coverage around the median was computed. In problems 3, 5, 7 and 8, the D*-Lite interval is below

Figure 31 – PDF of plan adaptation time for each WSC problem and graph search algorithm combination. Horizontal lines are the 15.87% percentile, median and 84.13% percentile. Transparent bullets mark the mean.

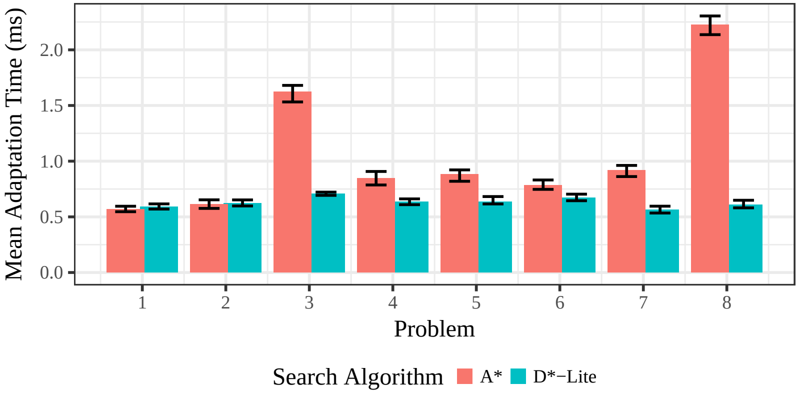


Source: the author.

the A* and does not intersect it. For all other problems the intervals intersect.

Figure 32 shows the means for both algorithms with error bars denoting their 95% CIs, obtained with bootstrapping. For most problems (3 to 8), D*-Lite is definitely faster (with non-intersecting CIs). The decrease in mean provided by D*-Lite ranges from 13.97% (0.109 ms, problem 6) to 72.66% (1.617 ms, problem 8), while CIs overlap in problems 1 and 2. These latter problems have a state graph with

Figure 32 – Average time for plan adaptation with different algorithms for WSC problems



Source: the author.

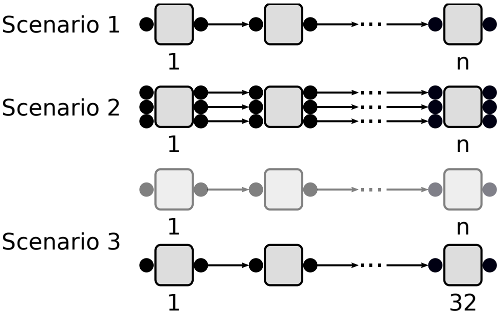
only 3 layers, which makes a full re-plan of such short paths as fast as adapting the current path.

5.3.2 Pragmatic Proof

In Chapter 3, the only composition algorithm found that supports the HATEOAS constraint was the Pragmatic Proof algorithm (VERBORGH et al., 2016). Verborgh et al. (2016) evaluated the algorithm using a benchmark designed by the authors themselves. This benchmark defines three scenarios, each varying a variable n, as shown in Figure 33. Scenario 1 has n services in the repository, each with an input and an output. The first service requires a triple pattern `?x ex:rel1 ?y` as its only input and produces another triple `?x ex:rel2 ?y`, which matches the input of the second service, and so on. The last service outputs a triple with the `ex:relGoal` property. Finally, the goal given to the Pragmatic Proof algorithm, in N3 logic, is `?x ex:relGoal ?y`. Scenario 2 examines the number of input conditions. It extends scenario 1 by requiring simultaneously three different triple patterns at each service. Scenario 3 examines the presence of irrelevant services by extending scenario 1 with 32 additional services and using a goal that is fulfilled using only these additional services.

In (VERBORGH et al., 2016), n varies from 4 to 1024 by powers of 2 (4, 8, 16, ...). The benchmark is used to evaluate only the time

Figure 33 – Pragmatic Proof benchmark scenarios.



Source: the author.

required to produce the first proof required by the Pragmatic Proof algorithm using two N3 logic reasoners: CWM (BERNERS-LEE et al., 2008) and EYE (VERBORGH; ROO, 2015). This time is broken down into parsing descriptions and reasoning. Measurement is done by first running the reasoner with reasoning disabled, and later with it enabled.

This benchmark can be used to compare Pragmatic Proof and the ComposIT reimplementation under Unserved with SPIN conditions. For Pragmatic Proof, the EYE reasoner is used[11], since it presented best results in (VERBORGH et al., 2016). For ComposIT, D*-Lite is used, given the results from Subsubsection 5.3.1.5. Preconditions as described in Subsection 4.2.2. Post-conditions are inferred from `u:part` relation between outputs. The time measurement considered covers all three phases (optimization is disabled as it has no effect in this benchmark).

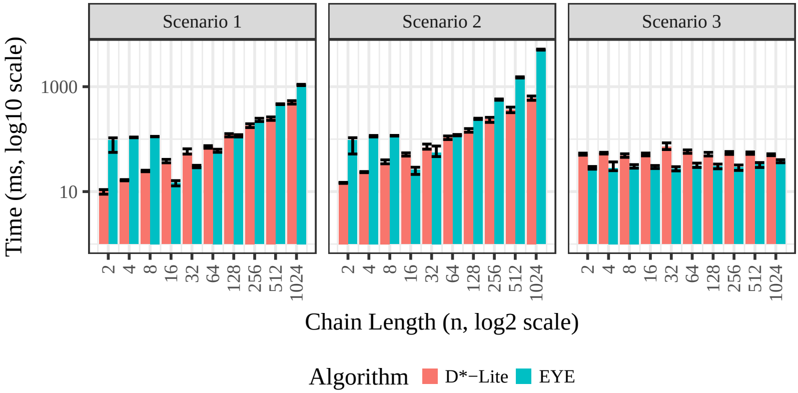
For the experiments here reported, each factors combination had a sample size of 9. As differences between composition algorithms were expressive, statistically significant conclusions could be drawn without further sampling. In addition to the small sample size, most samples have shapes that indicate non-normal distributions. Therefore, as in the WSC'08 experiments, the adjusted bootstrap percentile method was used with 4000 replications.

Figure 34 shows the averages for EYE reasoning and for ComposIT design time, per benchmark scenario and per value of n. For scenario 1, the Unserved implementation is faster for $n \leq 8$ and $n \geq 256$. For scenario 2, Unserved is faster for $n \leq 8$ and $n \geq 64$. In scenario

---

[11] Version 18.0117.1550.

Figure 34 – Averages and CI for EYE and ComposIT (D*-Lite) time in the Pragmatic Proof benchmark scenarios.



Source: the author.

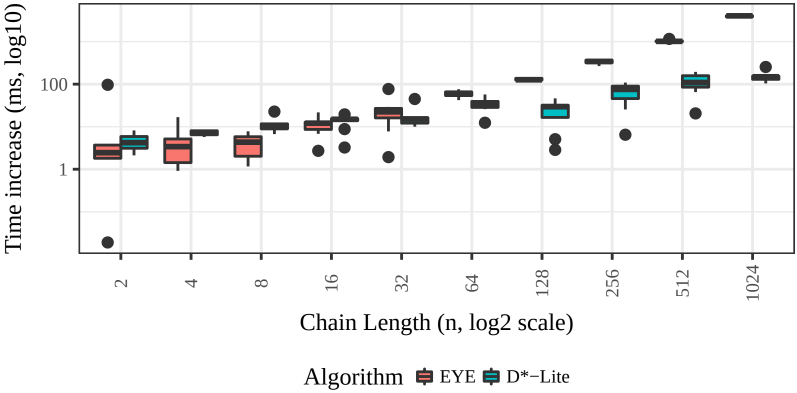
3 EYE is consistently faster than the Unserved implementation for all values n. The KENDALL (1938) τ correlation coefficient[12] between n and time reveals that the strength of correlation is nearly the same for both implementations in scenario 1 ($\tau = 0.8321$ for EYE and $\tau = 0.8047$ for Unserved) and scenario 2 ($\tau = 0.9045$ and $\tau = 0.8889$). In scenario 3 a weaker correlation is found for EYE ($\tau = 0.3159$) while no statistical evidence of correlation was found for Unserved.

Figure 34 also hints that EYE is more sensitive to increases in I/O count. This trend is highlighted in Figure 35, that shows box plots for the increase in time from scenario 1 to scenario 2. KENDALL (1938) τ correlation coefficient shows that both EYE and ComposIT under Unserved are affected by increasing I/O count ($p < 2e-16$ for both cases). EYE presents $\tau = 0.867$, close to the maximum, 1. On the other hand, the Unserved implementation presents $\tau = 0.618$, indicating correlation exists, but is weaker. Within the Unserved implementation, this correlation originates mostly from the construction phase: If only the planning phase is considered the method gives no conclusive evidence of correlation.

---

[12] τ does not assume normality and it does indicate strength of correlation. The first characteristic is important as 66% of populations under study (defined by combination of n, scenario and implementation) have their normality rejected by the Shapiro-Wilk test.

Figure 35 – Box-plot for increase in design time due to increase in I/O count from 1 to 3. Boxes represent 25% and 75% percentiles; horizontal line is the mean; outwards vertical lines extend to the farthest value no farther inside the outlier boundaries (25% or 75% percentile plus 1.5 the inter-quartile range).



Source: the author.

In practice, the largest performance drawback of Pragmatic Proof may be the need to spawn a reasoner process multiple time during composition execution. As an example, for scenario 3, with $n = 512$ the average time required to spawn the reasoner and parse descriptions is 398.2 ms. In the experiments here reported, the time required only for spawning the EYE reasoner averaged 39.1 ms.

Another issue with EYE is scalability in the presence of existential quantification in consequences of RESTdesc rules. The original benchmark generates RESTdesc rules with the form of `{?x ex:rel1 ?y.} => {?x ex:rel2 ?y. ?x ex:rel2 ?z.}`. In such rule, only `?y` is interpreted as existentially quantified (i.e., a blank node). While intermediary descriptions can use the same variable as both input and output, ComposIT does not support such use. Therefore the generated descriptions for ComposIT use different variables for inputs and outputs, but retain other characteristics of the benchmark. If the original benchmark is modified to distinguish inputs from outputs, any comparison becomes impossible, as no composition is produced in less than 5 minutes with $n \geq 8$ for scenario 2. This limitation is observed in

EYE versions Autumn15-12212144Z, 16.0919.1034, 17.0915.1303 and 18.0117.1550. EYE itself is still under active development.

6 CONCLUSION

Service computing (PAPAZOGLOU, 2003) has become a popular paradigm for creation of distributed applications, particularly over the Internet. Despite not being a recent paradigm, there are several open issues in service design, composition and selection (BOUGUETTAYA et al., 2017). One of such challenges is heterogeneity of service types (Subsection 2.3.2. SOAP, RESTful and event-oriented each induce basic interaction models, which place constraints that must be respected during service composition. This dissertation evaluated simultaneous support for the basic interaction models by existing composition methods and proposed the Unserved architecture, which is able to compose services with heterogeneous interaction models.

A first challenge met during the research that led to this dissertation was the not yet consolidated literature on heterogeneous compositions. The relevant studies identified by the SLR from Section 3.1 did not have a theoretical background for stating the problem of heterogeneous composition, neither identified which heterogeneities should be tackled. The consolidation of service types, the notion of interaction models and the identified methods in the SLR contribute to filling this gap in the literature.

In comparison to previous proposals identified in the SLR of Section 3.1, Unserved introduces the use of the composition algorithm with forking and adaptation as a technique for interaction model heterogeneity support. This allows the support of heterogeneities previously only possible theoretically through proxies, which preclude simultaneous support for the three basic interaction models. Unserved completely supports the three basic interaction models. Approaches in the SLR, on the other hand, did not present evidence of RESTful services support beyond POX (the lowest level in RMM).

Experiments comparing ComposIT original prototype and its reimplementation under Unserved found that the mean time for 7 out of 8 WSC problems decreases range from 10.90% (17.58 ms) to 81.34% (557.72 ms), with non-intersecting CIs. A small increase of 23.14% (7.83 ms) was observed only in problem 1. When SPIN conditions are added time increases range from 26.8% (12.15 ms,problem 4) to 37.9% (47.07 ms, problem 6). With respect to planning phase and the several investigated search algorithms, the algorithms can be sorted by decreasing planning time: D*-Lite $\geq$ A* $\geq$ Greedy $\geq$ Greedy without adaptation. This ordering holds for the two cost functions evaluated (service count and response time). D*-Lite and A* have close means (only 2 problems have non-intersecting CIs), while for other algorithms differences are statically significant in most problems. When re-planning occurs, D*-Lite was faster than A* in most problems (3 to 8).

In comparison to the only previous algorithm supporting HATEOAS, Pragmatic Proof, the ComposIT reimplementation was faster for small (chain length ≤ 8) and large problems (chain length ≥ 256 in scenario 1 or 64 in scenario 2). The comparison between these two scenarios has shown that the Unserved implementation is less sensitive to the number of I/O parameters. A final advantage of Unserved is that it avoids time-consuming tasks such as generating and parsing N3 and spawning the reasoner for each interaction in a composite service.

In summary, experimental results demonstrate that composition algorithms implemented in Unserved surpass, in most cases, the performance of current state-of-the-art algorithms. This demonstrates that an implementation of the architecture is viable in practice.

6.1 LIMITATIONS

Unserved was designed assuming reliability of connecting elements. This is the case the prototype used in the experiments. As results of problem 8 from the WSC'08 benchmark indicate, a very large number of services interacting in complex ways may be a threat to wide application of the architecture. This could be mitigated distributing processing between different machines. In their current form, neither the architecture nor algorithms involved can be easily distributed or parallelized. There are three challenges that may be either tackled or avoided during distribution: (1) high-frequency interaction between composer and service repository; (2) high-frequency interaction between interpreter and component repository; and (3) considerable amount of state during composite service execution.

Heterogeneity of data syntax and semantics was left out of scope, but remain relevant for practical applications. Issues such as ontology alignment are still open for investigation (OTERO-CERDEIRA; RODRÍGUEZ-MARTÍNEZ; GÓMEZ-RODRÍGUEZ, 2015). Semantic service descriptions as well as translation of service data to and from RDF are tasks that may require manually constructed mappings or adapters. There are approaches towards this end (OLIVEIRA et al., 2017), but the issue remains open.

The support for Non-functional requirements and properties is limited. Although the intermediary description allows assigning these to messages, the composition algorithms use a single cost function, which in experiments considered a single QoS property each time.

Other related issues, such as trust and reputation, were also ignored. However, interaction models do not mandate specific requirements or strategies for QoS. In addition, most existing solutions could be adapted into Unserved, as ComposIT was.

The scenario evaluation also identified some limitations of the prototype. One limitation is the lack of dynamic service discovery. A challenge in introducing this capability is that without knowing that services exists, the initial workflow cannot be designed by the composition algorithm. Therefore special cues must be introduced so that services are discovered before the composition algorithm deems a composition to be impossible. A second limitation is that currently all service faults are considered irrecoverable. However a fault may occur due to an unsatisfied pre-condition and such situation could be fixed invoking other services before the one that failed.

Many definitions provided in Chapter 2 were an attempt to solve conflicting definitions existing in the literature. Although extensive literature review was involved in this conflict-resolution process, it was not systematic. In a similar line, the protocol model of Subsection 2.5.1 can contemplate interaction models other than those considered in this dissertation. However, neither its universal application nor coverage were formally evaluated.

6.2 THREATS TO VALIDITY

The use of a systematic review protocol (Appendix B) seeks to mitigate all identified risks to the validity of conclusions in a SLR (KITCHENHAM; CHARTERS, 2007). However, some threats remain in study selection, data extraction and data synthesis.

Study selection criteria and processes require that works either mention at least two types of services or mention the goal of supporting service heterogeneity. Since SOAP services are also named simply Web Services in some sources, technologies such as BPEL and WSDL imply consideration of SOAP services. Considering “web service” as a SOAP services synonym or including all possible technologies for the three service types was found impractical during piloting. Nevertheless, The issue with SOAP services terminology appears to not apply to RESTful services and EOSs. In addition, no study that should have been included (but was not) is known.

A risk to validity of SLR results is that a single researcher conducted study selection and data extraction procedures. Several countermeasures were adopted to mitigate risk of human error. First, significant part of selection was automated (to avoid human error). Second, selection decisions could be done at any moment, even after the start of data extraction (fixing false positive inclusions). Third, quality of exclusion decisions was evaluated with random samples considering abstracts and full-text (estimating total amount of false negatives and fixing those identified). Fourth, data synthesis included confirmation of extracted data in the full-text of studies (fixing errors in data extraction).

6.3 ADDITIONAL RESEARCH OUTPUT

In addition to the main contributions of this dissertation, technical artifacts were also produced. The prototype implementation of the Unserved architecture is publicly available[1], as well as the datasets of our experiments[2]. A tool developed for the SLR presented in Section 3.1, but which may be useful for other researchers conducting an SLR, also has its source publicly available[3]. The SLR will be submitted to a journal, once the manuscript is finished.

This research had as output two publications, whose content overlap with this dissertation.

1. HUF, A.; SALVADORI, I. L.; SIQUEIRA, F. Planning and execution of heterogeneous service compositions. In: *2017 IEEE Symposium on Computers and Communications, ISCC 2017, Heraklion, Greece, July 3-6, 2017.* Washington, USA: IEEE, 2017. p. 987–993. Available at: <https://doi.org/10.1109/ISCC.2017-.8024654>.
2. HUF, A.; SIQUEIRA, F. Uma arquitetura para composição de serviços com modelos de interação heterogêneos. In: *Anais do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres, 17 a 20 de outubro, 2017, Gramado, Rio Grande do Sul.* Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2017. p. 11–16. ISBN 978-85-7669-380-2.

Most of the content in Chapter 4 and preliminary version of experiments in Section 5.3 first appeared in (HUF; SALVADORI; SIQUEIRA, 2017). Chapter 4 provides more details and also presents new elements introduced in the architecture for pre/post-condition support. Experiments from Section 5.3 first appeared in (HUF; SALVADORI; SIQUEIRA, 2017), before an improvement in their design

---

[1] <https://bitbucket.org/alexishuf/unserved-testbench>

[2] <https://bitbucket.org/alexishuf/unserved-testbench-dissertation>

[3] <https://github.com/alexishuf/slrpk>

that was introduced in (HUF; SIQUEIRA, 2017). In Section 5.3 more details are provided and more solid statistical methods were used for analysis. (HUF; SIQUEIRA, 2017) also contained a previous version of the SLR described in Section 3.1. This dissertation significantly improved the SLR protocol, resulting in 54 selected works, against the previous 30.

During the 2-year period of work towards this dissertation, effort was also invested in collaborations with other students from the Distributed Systems Research Lab also working with service composition and Semantic Web. These publications are listed below.

1. HUF, A.; SALVADORI, I. L.; SIQUEIRA, F. A service-oriented approach for integrating broadcast facilities. In: *IEEE International Conference on Services Computing, SCC 2016, San Francisco, CA, USA, June 27 - July 2, 2016*. Washington, USA: IEEE, 2016. p. 705–712. Available at: <https://doi.org/10-.1109/SCC.2016.97>.
2. SALVADORI, I.; HUF, A.; SIQUEIRA, F. An agent-based composition model for semantic microservices. In: *15th International Conference on WWW/INTERNET 2016, 2016, Mannheim. Proceedings of the IADIS International Conference WWW/Internet*. Manheim, Germany: IADIS, 2016. p. 75–82. ISBN 978-989-8533-57-9.
3. OLIVEIRA, B. C. N. et al. A platform to enrich, expand and publish linked data of police reports. In: *15th International Conference on WWW/INTERNET 2016, 2016, Mannheim. Proceedings of the IADIS International Conference WWW/Internet*. Manheim, Germany: IADIS, 2016. p. 118–118. ISBN 978-989-8533-57-9.
4. SALVADORI, I. L. et al. Publishing linked data through semantic microservices composition. In: *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, iiWAS 2016, Singapore, November 28-30, 2016*. New York, USA: ACM, 2016. p. 443–452. Available at: <http://doi.org/10.1145/3011141.3011155>.
5. SALVADORI, I. L. et al. Improving entity linking with ontology alignment for semantic microservices composition. *International Journal of Web Information Systems*, v. 13, n. 3, p. 302–323, 2017. Available at: <https://doi.org/10.1108/IJWIS-04-2017-0029>.
6. OLIVEIRA, B. C. N. et al. Automatic semantic enrichment of data services. In: *iiWAS '17: The 19th International Conference*

on Information Integration and Web-based Applications & Services, December 4–6, 2017, Salzburg, Austria. New York, USA: ACM, 2017. p. 415–424. Available at: <https://doi.org/10.1145/3151759.3151783>.

7. SALVADORI, I. et al. An ontology alignment framework for data-driven microservices. In: *iiWAS '17: The 19th International Conference on Information Integration and Web-based Applications & Services, December 4–6, 2017, Salzburg, Austria.* New York, USA: ACM, 2017. p. 425–433. Available at: <https://doi.org/10.1145/3151759.3151793>.

6.4 FUTURE WORK

The notion of interaction model, introduced in Section 2.5, is only used to formalize the problem targeted by Unserved and for its evaluation. Unserved was designed for all three basic interaction models. As a consequence, adaptation will be employed even if no service employing HATEOAS is used in the composition. In future work, interaction models could be promoted to data elements of Unserved, allowing their relation to intermediary descriptions. This would allow selection of the execution methods after composite service design. If the selected component services do not employ HATEOAS, one can forgo adaptive execution. Likewise, if there is no event-oriented component service, there is no need for slicing and for execution forking.

Another possibility opened up by making interaction models explicit is adding support for other interaction models not considered in this dissertation. For example, one could add support for Foundation for Intelligent Physical Agents (FIPA) agents. Such agents envelop their interactions in Agent Comminucation Language (ACL) (FIPA, 2002a), employ actions from a library of communicative acts (FIPA, 2002b) and may implement some pre-determined protocols (e.g., brokering (FIPA, 2002c)).

Another direction of research is integration with existing manual composition mechanisms, such as BPEL engines and CEP engines (CUGOLA; MARGARA, 2012). This would allow established manually-designed composite service specifications to be extended including services whose interaction models are not fully supported in those environments (e.g., RESTful services).

With respect to the composition algorithm used with Unserved, although it has expressivity comparable to that of Pragmatic Proof, it does not support some constructs supported by other manual or automatic composition approaches. In future work, pre- and post-conditions could be extended so that negation is supported, as is the case in PDDL (MCDERMOTT et al., 1998). Another possibility is employing a MCS, so that pre/post-conditions abiding to different logics can simultaneously coexist without requiring mapping to SPIN conditions *a priori*. As for composite service specifications, ComposIT invokes each component service at most once. In future work, the ability to plan for loops could be added by replacing ComposIT or extending it with this capability.

Currently Unserved and most composition simply notify the end user when a composite service achieving requested goals could not be designed. Returning instead a suggestion of changes to the goals in such way that would allow the design of a composite service would be more helpful to users. One step in this direction, for ComposIT under Unserved, would be to identify wanted variables that could only be satisfied under the *subsumes* matching degree described by (PAOLUCCI et al., 2002), and compute a modification to the user-provided wanted variable.

The evaluation of Unserved, like most related work often do (as shown in Subsection 3.1.6 and Section 3.2), consisted of analysis methods (HEVNER et al., 2004) and controlled experiments with artificial benchmarks. Related to performance alone, three aspects relevant to production environments could be evaluated in future works: First, the use of a triple store and the relation between changes to ComposIT introduced in Section 5.3 and the framework proposed by (RODRIGUEZ-MIER et al., 2016) for discovering services from a SPARQL endpoint. Second, the use of of-the-shelf OWL reasoners in scenarios that demand more powerful reasoning. Third, the performance of composite service execution under Unserved using real services instead of dummy implementations.

Beyond Unserved, there is opportunity for further experiments that evaluate automatic service composition in real business environments. This dissertation, like all related work that propose an automatic composition algorithm, evaluates performance using benchmarks whose artificial workloads seek to stress-test algorithms. Another gap in literature is the realization of experiments with control groups that evaluate programmer productivity gains obtained by the use of automatic composition tools.

BIBLIOGRAPHY

ACAMPORA, G. et al. Interoperable and adaptive fuzzy services for ambient intelligence applications. *ACM Transactions Autonomous Adaptative Systems*, ACM, New York, USA, v. 5, n. 2, p. 8:1–8:26, may 2010. ISSN 1556-4665. Available at: <https://doi.org/10.1145/1740600.1740604>.

AHMED, R.; BOUTABA, R. A Survey of Distributed Search Techniques in Large Scale Distributed Systems. *IEEE Communications Surveys & Tutorials*, v. 13, n. 2, p. 150–167, 2011. ISSN 1553-877X. Available at: <https://doi.org/10.1109-/SURV.2011.040410.00097>.

AIJAZ, F. et al. Enabling High Performance Mobile Web Services Provisioning. In: *2009 IEEE 70th Vehicular Technology Conference Fall*. Washington, USA: IEEE, 2009. p. 1–6. Available at: <https://doi.org/10.1109/VETECF.2009.5378949>.

ALABOOL, H. M.; MAHMOOD, A. K. Review on cloud service evaluation and selection methods. In: *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)*. Washington, USA: IEEE, 2013. p. 61–66. ISBN 2324-8149. Available at: <https://doi.org/10.1109/ICRIIS.2013-.6716686>.

ALONSO, G. et al. Web Services. In: _____. *Web Services: Concepts, Architectures and Applications*. Berlin, Germany: Springer, 2004. cap. 5, p. 123–149. ISBN 978-3-662-10876-5. Available at: <https://doi.org/10.1007/978-3-662-10876-5_5>.

ALRIFAI, M.; SKOUTAS, D.; RISSE, T. Selecting skyline services for qos-based web service composition. In: *Proceedings of the 19th International Conference on World Wide Web*. New York, USA: ACM, 2010. (WWW '10), p. 11–20. ISBN 978-1-60558-799-8. Available at: <https://doi.org/10.1145/1772690.1772693>.

ALVES, J. et al. Resilient composition of Web services through nondeterministic planning. In: *2016 IEEE Symp. on Computers and Comm. (ISCC)*. [s.n.], 2016. p. 895–900. Available at: <https://doi.org/10.1109/ISCC.2016.7543850>.

AMNUAYKANJANASIN, P.; NUPAIROJ, N. The BPEL orchestrating framework for secured grid services. In: *International Conference on Information Technology: Coding and Computing,* *ITCC*. [s.n.], 2005. v. 1, p. 348–377. Available at: <https://doi.org/10.1109/ITCC.2005.271>.

ANDREWS, G. R. Paradigms for Process Interaction in Distributed Programs. *ACM Computing Surv.*, ACM, New York, USA, v. 23, n. 1, p. 49–90, mar 1991. ISSN 0360-0300. Available at: <https://doi.org/10.1145/103162.103164>.

ANDREWS, H.; WRIGHT, A. *JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON*. [S.l.], 2018. Available at: <https://tools.ietf.org/html/draft-handrews-json-schema-hyperschema-01>.

ARDISSONO, L. et al. Syncfr: Synchronization collaboration framework. In: *Proceedings of the 2009 4th International Conference on Internet and Web Applications and Services, ICIW 2009*. Washington, USA: IEEE, 2009. p. 18–23. Available at: <https://doi.org/10.1109/ICIW.2009.11>.

ARNDT, R. et al. COMM: Designing a well-founded multimedia ontology for the web. In: ABERER, K. et al. (Ed.). *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*. Berlin, Germany: Springer, 2007. p. 30–43. ISBN 978-3-540-76298-0. Available at: <https://doi.org/10.1007/978-3-540-76298-0_3>.

ARNOLD, K. et al. *Jini Specification*. 1st. ed. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0201616343.

ARROQUI, M. et al. RESTful Web Services improve the efficiency of data transfer of a whole-farm simulator accessed by Android smartphones. *Computers and Electronics in Agriculture*, v. 87, n. Supplement C, p. 14–18, 2012. ISSN 0168-1699. Available at: <https://doi.org/10.1016/j.compag.2012.05.016>.

ASHBURNER, M. et al. Gene ontology: tool for the unification of biology. *Nature Genetics*, Nature America Inc., v. 25, n. 1, p. 25–29, may 2000. ISSN 1061-4036. Available at: <https://doi.org/10.1038/75556 http://10.0.4.14/75556>.

AUER, S. et al. Dbpedia: A nucleus for a web of open data. In: ABERER, K. et al. (Ed.). *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic*

Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings. Berlin, Germany: Springer, 2007. p. 722–735. ISBN 978-3-540-76298-0. Available at: <https://doi.org/10.1007/978-3-540-76298-0_52>.

BAADER, F. *The description logic handbook: Theory, implementation and applications*. 2nd. ed. [S.l.]: Cambridge university press, 2010. ISBN 9780521150118.

BAGLIETTO, P. et al. Analysis of design patterns for composite telco services. In: *14th Int. Conference on Intelligence in Next Generation Networks: "Weaving Applications Into the Network Fabric", ICIN 2010 - 2nd Int. Workshop on Business Models for Mobile Platforms, BMMP 10*. [s.n.], 2010. Available at: <https://doi.org/10.1109/ICIN.2010.5640904>.

BAGNASCO, A. et al. Application of web services to heterogeneous networks of small devices. In: *Proceedings of the 8th WSEAS International Conference on Automatic Control, Modeling & Simulation*. Stevens Point, USA: World Scientific and Engineering Academy and Society (WSEAS), 2006. (ACMOS'06), p. 146–151. ISBN 960-8457-42-4.

BANKS, A.; GUPTA, R. *MQTT Version 3.1.1*. [S.l.], 2014. Available at: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os-/mqtt-v3.1.1-os.html>.

BANSAL, A. et al. WSC-08: Continuing the Web Services Challenge. In: *2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*. [s.n.], 2008. p. 351–354. ISBN 2378-1963. Available at: <https://doi.org/10.1109-/CECandEEE.2008.146>.

BARESI, L.; GUINEA, S. Self-supervising BPEL processes. *IEEE Transactions on Software Engineering*, v. 37, n. 2, p. 247–263, mar 2011. ISSN 0098-5589. Available at: <https://doi.org/10.1109-/TSE.2010.37>.

BARRETO, C. et al. *Web Services Choreography Description Language Version 1.0*. [S.l.], 2005. Available at: <http://www.w3-.org/TR/2005/CR-ws-cdl-10-20051109/>.

BARTALOS, P.; BIELIKOVA, M. Automatic dynamic web service composition: A survey and problem formalization. *Computing and Informatics*, v. 30, n. 4, p. 793–827, 2011. ISSN 1335-9150.

BEECH, D. et al. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. [S.l.], 2012. Available at: <http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Jade–a FIPA-compliant agent framework. In: LONDON. *Proceedings of PAAM*. [S.l.], 1999. v. 99, n. 97-108, p. 33.

BELSHE, M.; ROBERTO, P.; MARTIN, T. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. [S.l.], 2015. Available at: <http://www.rfc-editor.org/rfc/rfc7540.txt>.

BERARDI, D. et al. Automatic composition of e-services that export their behavior. In: ORLOWSKA, M. E. et al. (Ed.). *Service-Oriented Computing - ICSOC 2003: First International Conference, Trento, Italy, December 15-18, 2003. Proceedings*. Berlin, Germany: Springer, 2003. p. 43–58. ISBN 978-3-540-24593-3. Available at: <https://doi.org/10.1007/978-3-540-24593-3_4>.

BERGWEILER, S. A flexible framework for adaptive knowledge retrieval and fusion for kiosk systems and mobile clients. In: *UBICOMM 2014 - 8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*. Nice, France: IARIA XPS Press, 2015. p. 164–171.

BERNERS-LEE, T.; CONNOLLY, D. W. *Hypertext Markup Language – 2.0*. [S.l.], 1995. Available at: <https://doi.org/10-.17487/rfc1866>.

BERNERS-LEE, T.; FIELDING, R. T.; MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax*. [S.l.], 2005. Available at: <https://doi.org/10.17487/rfc3986>.

BERNERS-LEE, T.; FISCHETTI, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. [S.l.]: HarperInformation, 2000. ISBN 006251587X.

BERNERS-LEE, T. et al. The semantic web. *Scientific american*, New York, USA:, v. 284, n. 5, p. 28–37, 2001.

BERNERS-LEE, T. I. M. et al. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, v. 8, n. 03, p. 249–269, 2008. ISSN 1475-3081. Available at: <https://doi.org/10.1017/S1471068407003213>.

BERTAUD-GOUNOT, V.; DUVAUFERRIER, R.; BURGUN, A. Ontology and medical diagnosis. *Informatics for Health and Social Care*, Taylor & Francis, v. 37, n. 2, p. 51–61, 2012.

BERTOLI, P. et al. Design and analysis of the composed telecom services. Springer, Berlin, Germany, v. 4907 LNCS, p. 282–294, 2009. Available at: <https://doi.org/10.1007/978-3-540-93851-4_28>.

BLUMBERGS, N.; KRAVCEVS, M. Wsrf usage for BPM and CEP systems integration. *Frontiers in Artificial Intelligence and Applications*, v. 224, p. 240–253, 2011. Available at: <https://doi.org/10.3233/978-1-60750-688-1-240>.

BO, Y. et al. The design of an orchestrated execution environment based on jbi. In: *IFCSTA 2009 Proceedings - 2009 International Forum on Computer Science-Technology and Applications*. [s.n.], 2009. v. 3, p. 367–371. Available at: <https://doi.org/10.1109/IFCSTA.2009.328>.

BONATTI, P. A.; FESTA, P. On optimal service selection. In: *Proceedings of the 14th International Conference on World Wide Web*. New York, USA: ACM, 2005. (WWW '05), p. 530–538. ISBN 1-59593-046-9. Available at: <https://doi.org/10.1145/1060745.1060823>.

BOOTH, D. et al. *Web Services Architecture*. [S.l.], 2004. Available at: <https://www.w3.org/TR/ws-arch/>.

BOUGUETTAYA, A.; SHENG, Q. Z.; DANIEL, F. *Web services foundations*. New York: Springer, 2014. ISBN 978-1-4614-7518-7. Available at: <https://doi.org/10.1007/978-1-4614-7518-7>.

BOUGUETTAYA, A. et al. A Service Computing Manifesto: The Next 10 Years. *Communications of the ACM*, ACM, New York, USA, v. 60, n. 4, p. 64–72, mar 2017. ISSN 0001-0782. Available at: <https://doi.org/10.1145/2983528>.

BOX, D. et al. *Web Services Eventing (WS-Eventing)*. [S.l.], 2006.

BRAUBACH, L.; POKAHR, A. Conceptual integration of agents with WSDL and RESTful web services. Springer, Berlin, Germany, v. 7837 LNAI, p. 17–34, 2013. Available at: <https://doi.org/10.1007/978-3-642-38700-5_2>.

BREWKA, G.; EITER, T. Equilibria in heterogeneous nonmonotonic multi-context systems. In: *Proceedings of the AAAI*. Vancouver, Canada: AAAI, 2007. v. 7, p. 385–390.

BREWKA, G. et al. Managed multi-context systems. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Barcelona, Spain: AAAI, 2011. v. 22, p. 786.

BRICKLEY, D.; GUHA, R. *RDF Schema 1.1*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.

BROMBERG, Y.-D. et al. Bridging the Interoperability Gap: Overcoming Combined Application and Middleware Heterogeneity. In: *Proceedings of the 12th International Middleware Conference*. Laxenburg, Austria, Austria: International Federation for Information Processing, 2011. (Middleware '11), p. 380–399. ISBN 978-3-642-25820-6. Available at: <http://dl.acm.org/citation.cfm?id=2414338.2414365>.

BROMBERG, Y.-D. et al. Bridging the interoperability gap: Overcoming combined application and middleware heterogeneity. Springer, Berlin, Germany, v. 7049 LNCS, p. 390–409, 2011. Available at: <https://doi.org/10.1007/978-3-642-25821-3_20>.

CANFORA, G. et al. An approach for qos-aware service composition based on genetic algorithms. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. New York, USA: ACM, 2005. (GECCO '05), p. 1069–1075. ISBN 1-59593-010-8. Available at: <https://doi.org/10.1145/1068009.1068189>.

CAO, B. et al. Mashup service recommendation based on user interest and social network. In: *2013 IEEE 20th International Conference on Web Services*. Washington, USA: IEEE, 2013. p. 99–106. Available at: <https://doi.org/10.1109/ICWS.2013.23>.

CAROTHERS, G. *RDF 1.1 N-Quads*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-n-quads-20140225/>.

CAROTHERS, G.; SEABORNE, A. *RDF 1.1 N-Triples*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-n-triples-20140225/>.

CARROLL, J. J. et al. Jena: Implementing the semantic web recommendations. In: *Proceedings of the 13th International World*

Wide Web Conference on Alternate Track Papers &Amp; Posters. New York, USA: ACM, 2004. (WWW Alt. '04), p. 74–83. ISBN 1-58113-912-8. Available at: <https://doi.org/10.1145/1013367.1013381>.

CHAPPELL, D.; LIU, L. *Web Services Brokered Notification 1.3*. [S.l.], 2006.

CHATTOPADHYAY, S.; BANERJEE, A.; BANERJEE, N. A Scalable and Approximate Mechanism for Web Service Composition. In: *2015 IEEE International Conference on Web Services*. Washington, USA: IEEE, 2015. p. 9–16. Available at: <https://doi.org/10.1109/ICWS.2015.12>.

CHELLAS, B. F. *Modal logic: an introduction*. Cambridge, United Kingdom: Cambridge University Press, 1980. ISBN 978-0-52-129515-4.

CHEN, H. et al. Using open web APIs in teaching web mining. *IEEE Transactions on Education*, v. 52, n. 4, p. 482–490, november 2009. ISSN 0018-9359. Available at: <https://doi.org/10.1109/TE.2008.930509>.

CHENG, B. et al. Lightweight mashup middleware for coal mine safety monitoring and control automation. *IEEE Transactions on Automation Science and Engineering*, v. 14, n. 2, p. 1245–1255, 2017. Available at: <https://doi.org/10.1109/TASE.2016.2518677>.

CHESHIRE, S.; KROCHMAL, M. *DNS-Based Service Discovery*. [S.l.], 2013. Available at: <https://doi.org/10.17487/rfc6763>.

CHOU, W.; LI, L.; LIU, F. Web service enablement of communication services. In: *IEEE International Conference on Web Services, ICWS 2005*. Washington, USA: IEEE, 2005. v. 2005, p. 393–400. Available at: <https://doi.org/10.1109/ICWS.2005.130>.

CHRISTENSEN, E. et al. *Web Services Description Language (WSDL) 1.1*. [S.l.], 2001. Available at: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>.

CIPOLLA, D. et al. Web service based asynchronous service execution environment. Springer, Berlin, Germany, v. 4907 LNCS, p. 304–316, 2009. Available at: <https://doi.org/10.1007/978-3-540-93851-4_30>.

CLÉMENT, L. et al. *Web Services – Human Task (WS-HumanTask) Specification Version 1.1*. [S.l.], 2010. Available at: <http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.html>.

CLÉMENT, L. et al. *WS-BPEL Extension for People (BPEL4People) Specification Version 1.1*. [S.l.], 2010. Available at: <http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cs-01.html>.

CLOPPER, C. J.; PEARSON, E. S. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, Oxford University Press, Biometrika Trust, Oxford, UK, v. 26, n. 4, p. 404–413, 1934. ISSN 00063444. Available at: <https://doi.org/10.2307/2331986>.

CORMEN, T. H. et al. *Introduction to Algorithms, Third Edition*. 3rd. ed. Cambridge, USA: The MIT Press, 2009. ISBN 0262033844, 9780262033848.

COWAN, J. et al. *Extensible Markup Language (XML) 1.1 (Second Edition)*. [S.l.], 2006. Available at: <http://www.w3.org/TR/2006-/REC-xml11-20060816>.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, ACM, New York, USA, v. 44, n. 3, p. 1–62, june 2012. ISSN 0360-0300. Available at: <https://doi.org/10.1145/2187671-.2187677>.

CYBENKO, G. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, v. 7, n. 2, p. 279 – 301, 1989. ISSN 0743-7315. Available at: <https://doi.org/10.1016/0743-7315(89)90021-X>.

CYGANIAK, R.; WOOD, D.; LANTHALER, M. *RDF 1.1 Concepts and Abstract Syntax*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.

DALY, J. et al. A hierarchy of evidence for assessing qualitative health research. *Journal of Clinical Epidemiology*, v. 60, n. 1, p. 43–49, 2007. ISSN 0895-4356. Available at: <https://doi.org/10.1016/j.jclinepi.2006.03.014>.

DEB, K.; KALYANMOY, D. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, USA: John Wiley & Sons, Inc., 2001. ISBN 047187339X.

DENG, S. et al. Constraints-Driven Service Composition in Mobile Cloud Computing. In: *2016 IEEE International Conference on Web Services (ICWS)*. Washington, USA: IEEE, 2016. p. 228–235. Available at: <https://doi.org/10.1109/ICWS.2016.37>.

Dictionary.com. paradigm. In: _____. *Dictionary.com Unabridged*. Random House Inc., 2017. Available at: <http://www.dictionary-.com/browse/paradigm>. Access on: 3 nov. 2017.

DIJKMAN, R.; DUMAS, M. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, v. 13, n. 04, p. 337–368, 2004. Available at: <https://doi.org/10.1142/S0218843004001012>.

DO, M. B.; KAMBHAMPATI, S. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, v. 20, n. 1, p. 155–194, 2003. Available at: <https://doi.org/10.1613-/jair.1156>.

DOMINGOS, D.; MARTINS, F.; ĈANDIDO, C. Internet of things aware WS-BPEL business process. In: *ICEIS 2013 - Proceedings of the 15th International Conference on Enterprise Information Systems*. Setúbal, Portugal: SciTePress, 2013. v. 2, p. 505–512. Available at: <https://doi.org/10.5220/0004449905050512>.

DREWNIOK, M. et al. Experiments and performance evaluation of event driven mashups. In: *IEEE Symposium on Computers and Communications*. Washington, USA: IEEE, 2009. p. 19–22. Available at: <https://doi.org/10.1109/ISCC.2009.5202397>.

DÜERST, M.; SUIGNARD, M. *Internationalized Resource Identifiers (IRIs)*. [S.l.], 2005. Available at: <https://doi.org/10-.17487/rfc3987>.

DUSTDAR, S.; SCHREINER, W. A Survey on Web Services Composition. *International Journal of Web and Grid Services*, Inderscience Publishers, Geneva, Switzerland, v. 1, n. 1, p. 1–30, august 2005. ISSN 1741-1106. Available at: <https://doi.org/10-.1504/IJWGS.2005.007545>.

EFRON, B. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, Taylor & Francis, v. 82, n. 397, p. 171–185, 1987. Available at: <https://doi.org/10.1080/01621459-.1987.10478410>.

EROL, K.; HENDLER, J.; NAU, D. S. Htn planning: Complexity and expressivity. In: *Proceedings of the 12th National Conference on Artificial Intelligencee*. Seattle, USA: AAAI, 1994. v. 2, p. 1123–1128.

EUGSTER, P. T. et al. The Many Faces of Publish/Subscribe. *ACM Computing Surv.*, ACM, New York, USA, v. 35, n. 2, p. 114–131, 2003. ISSN 0360-0300. Available at: <https://doi.org/10-.1145/857076.857078>.

EUZENAT, J.; SHVAIKO, P. *Ontology Matching*. Berlin, Germany: Springer, 2013. ISBN 978-3-642-38721-0.

FALBO, R. d. A. SABiO: Systematic Approach for Building Ontologies. In: *Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering co-located with 8th International Conference on Formal Ontology in Information Systems (FOIS 2014)*. Rio de Janeiro, Brazil: CEUR-WS.org, 2014. p. 16—-29.

FAN, X.-Q.; FANG, X.-W.; JIANG, C.-J. Research on web service selection based on cooperative evolution. *Expert Systems with Applications*, v. 38, n. 8, p. 9736 – 9743, 2011. ISSN 0957-4174. Available at: <https://doi.org/10.1016/j.eswa.2011.02.026>.

FANJIANG, Y.-Y. et al. An overview and classification of service description approaches in automated service composition research. *IEEE Transactions on Services Computing*, v. 10, n. 2, p. 176–189, mar 2017. ISSN 1939-1374. Available at: <https://doi.org/10.1109/TSC.2015.2461538>.

FEIGENBAUM, L. et al. *SPARQL 1.1 Protocol*. [S.l.], 2013. Available at: <http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/>.

FENSEL, D. et al. OIL: an ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, v. 16, n. 2, p. 38–45, 2001. Available at: <https://doi.org/10.1109/5254.920598>.

FETTE, I.; MELNIKOV, A. *The WebSocket Protocol*. [S.l.], 2011. Available at: <http://www.rfc-editor.org/rfc/rfc2616.txt>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000.

FIELDING, R. T. *REST APIs must be hypertext-driven*. 2008. Available at: <http://roy.gbiv.com/untangled/2008-/rest-apis-must-be-hypertext-driven>. Access on: 3 nov. 2017.

FIELDING, R. T. et al. *Hypertext Transfer Protocol – HTTP/1.1*. [S.l.], 1999. Available at: <http://www.rfc-editor.org/rfc/rfc2616-.txt>.

FIELDING, R. T.; NOTTINGHAM, M.; RESCHKE, J. F. *Hypertext Transfer Protocol (HTTP/1.1): Caching*. [S.l.], 2014. Available at: <http://www.rfc-editor.org/rfc/rfc7234.txt>.

FIELDING, R. T.; TAYLOR, R. N. Principled Design of the Modern Web Architecture. *ACM Transactions Internet Technology*, ACM, New York, USA, v. 2, n. 2, p. 115–150, 2002. ISSN 1533-5399. Available at: <https://doi.org/10.1145/514183.514185>.

FITZPATRICK, B. et al. *PubSubHubbub Core 0.4*. [S.l.], 2014. Available at: <https://pubsubhubbub.github.io/PubSubHubbub-/pubsubhubbub-core-0.4.html>.

FOKAEFS, M.; STROULIA, E. Wsmeta: A meta-model for web services to compare service interfaces. In: *Proceedings of the 17th Panhellenic Conference on Informatics*. [s.n.], 2013. (ACM International Conference Proceeding Series), p. 1–8. Available at: <https://doi.org/10.1145/2491845.2491860>.

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. *FIPA ACL Message Structure Specification*. Geneva, Switzerland, 2002.

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. *FIPA Communicative Act Library Specification*. Geneva, Switzerland, 2002.

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. *FIPA Communicative Act Library Specification*. Geneva, Switzerland, 2002.

FOX, M.; LONG, D. Pddl2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 20, n. 1, p. 61–124, december 2003. ISSN 1076-9757.

FRANCIS, B. *Web Thing API.* [S.l.], 2017. Available at: <https://www.w3.org/Submission/2017/Member-SUBM-WoT-20171208/>.

FREED, N.; KLENSIN, J. C. *Media Type Specifications and Registration Procedures.* [S.l.], 2005. Available at: <https://doi-.org/10.17487/rfc4288>.

FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. *IEEE Transactions on Software Engineering*, v. 24, n. 5, p. 342–361, may 1998. ISSN 0098-5589. Available at: <https://doi.org/10.1109/32.685258>.

GAEDKE, M.; HäRTZER, D.; HEIL, A. Webcomposition/DGS: Dynamic service components for web 2.0 development. In: *MoMM2008 - The 6th International Conference on Advances in Mobile Computing and Multimedia.* New York, USA: ACM, 2008. p. 456–459. Available at: <https://doi.org/10.1145/1497185-.1497282>.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software.* Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GANDON, F.; SCHREIBER, G. *RDF 1.1 XML Syntax.* [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/>.

GARLAN, D.; SHAW, M. An introduction to software architecture. In: ______. *Advances in Software Engineering and Knowledge Engineering.* Singapore: World Scientific Publishing, 1993. v. 2, p. 1–40. ISBN 981-02-1594-0.

GARRIGA, M. et al. RESTful service composition at a glance: A survey. *Journal of Network and Computer Applications*, v. 60, n. Supplement C, p. 32 – 53, 2016. ISSN 1084-8045. Available at: <https://doi.org/10.1016/j.jnca.2015.11.020>.

GENESERETH, M. R.; NILSSON, N. J. *Logical Foundations of Artificial Intelligence.* San Francisco, USA: Morgan Kaufmann Publishers Inc., 1987. ISBN 0-934613-31-1.

GEORGANTAS, N. et al. Service-oriented distributed applications in the future internet: The case for interaction paradigm interoperability. In: *Service-Oriented and Cloud Computing: Second European Conference, ESOCC 2013, Málaga, Spain, September 11-13, 2013. Proceedings*. Berlin, Germany: Springer, 2013. (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v. 8135 LNCS), p. 134–148. ISBN 978-3-642-40651-5. Available at: <https://doi.org/10.1007/978-3-642-40651-5_11>.

GIACOMO, G. D.; LESPÉRANCE, Y.; LEVESQUE, H. J. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, v. 121, n. 1, p. 109 – 169, 2000. ISSN 0004-3702. Available at: <https://doi.org/10.1016-/S0004-3702(00)00031-X>.

GIARETTA, P.; GUARINO, N. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, v. 25, p. 32, 1995.

GIORGIO, T. D.; RIPA, G.; ZUCCALà, M. An approach to enable replacement of SOAP services and REST services in lightweight processes. In: *Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*. Berlin, Germany: Springer, 2010. (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v. 6385 LNCS), p. 338–346. ISBN 978-3-642-16985-4. Available at: <https://doi.org/10.1007/978-3-642-16985-4_30>.

GIRARD, J.-Y. Linear logic. *Theoretical Computer Science*, v. 50, n. 1, p. 1 – 101, 1987. ISSN 0304-3975. Available at: <https://doi.org/10.1016/0304-3975(87)90045-4>.

GLINZ, M. On Non-Functional Requirements. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. Washington, USA: IEEE, 2007. p. 21–26. ISBN 1090-705X. Available at: <https://doi.org/10.1109/RE.2007.45>.

GRAHAM, S.; HULL, H.; MURRAY, B. *Web Services Base Notification 1.3*. [S.l.], 2006.

GREGORIO, J. et al. *URI Template*. [S.l.], 2012. Available at: <https://doi.org/10.17487/rfc6570>.

GREGORIO, J.; HORA, B. de. *The Atom Publishing Protocol*. [S.l.], 2007. Available at: <https://doi.org/10.17487/rfc5023>.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, v. 5, n. 2, p. 199–220, 1993. ISSN 1042-8143. Available at: <https://doi.org/10.1006/knac-.1993.1008>.

GUARINO, N. Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, v. 46, n. 2, p. 293–310, 1997. ISSN 1071-5819. Available at: <https://doi.org/10-.1006/ijhc.1996.0091>.

GUARINO, N. Formal ontology and information systems. In: *Formal Ontology in Information Systems. Proceedings of FOIS'98*. Amsterdam, The Netherlands: IOS Press, 1998. p. 3–15.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. Tese (Doutorado) — Universiteit Twente, october 2005.

GUIZZARDI, G. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, IOS Press, v. 155, p. 18—-39, 2007.

HADLEY, M. *Web Application Description Language*. [S.l.], 2009. Available at: <https://www.w3.org/Submission/wadl/>.

HANG, F.; ZHAO, L. Hypermash: A heterogeneous service composition approach for better support of the end users. In: *2013 IEEE 20th International Conference on Web Services*. Washington, USA: IEEE, 2013. (ICWS '13), p. 435–442. Available at: <https://doi.org/10.1109/ICWS.2013.65>.

HART, P.; NILSSON, N.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. v. 4, n. 2, p. 100–107, 1968. ISSN 0536-1567. Available at: <https://doi.org/10.1109/TSSC.1968.300136>.

HASTINGS, J. et al. The chebi reference database and ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Research*, v. 41, n. D1, p. D456–D463, 2013. Available at: <https://doi.org/10.1093/nar/gks1146>.

HAUPT, F. et al. Service composition for REST. In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference*. [s.n.], 2014. v. 2014-December, n. December, p. 110–119. Available at: <https://doi.org/10.1109/EDOC.2014.24>.

HAYES, P. J. The naive physics manifesto. In: MICHIE, D. (Ed.). *Expert Systems in the Micro-Electronic Age*. Edinburgh, United Kingdom: Edinburgh University Press, 1978.

HE, K. Integration and orchestration of heterogeneous services. In: *2009 Joint Conferences on Pervasive Computing (JCPC)*. Washington, USA: IEEE, 2009. p. 467–470. Available at: <https://doi.org/10.1109/JCPC.2009.5420139>.

HEATH, T.; BIZER, C. Semantic annotation and retrieval: Web of data. In: _____. *Handbook of Semantic Web Technologies*. Berlin, Germany: Springer, 2011. p. 191–229. ISBN 978-3-540-92913-0. Available at: <https://doi.org/10.1007/978-3-540-92913-0_6>.

HEILER, S. Semantic Interoperability. *ACM Computing Surv.*, ACM, New York, USA, v. 27, n. 2, p. 271–273, june 1995. ISSN 0360-0300. Available at: <https://doi.org/10.1145/210376.210392>.

HEVNER, A. R. et al. Design Science in Information Systems Research. *MIS Quarterly*, Management Information Systems Research Center, University of Minnesota, v. 28, n. 1, p. 75–105, 2004. ISSN 02767783. Available at: <http://www.jstor.org/stable/25148625>.

HIRIART-URRUTY, J.-B. Conditions for global optimality. In: _____. *Handbook of Global Optimization*. Boston, USA: Springer, 2013. cap. 1, p. 1–26. ISBN 978-1-4615-2025-2. Available at: <https://doi.org/10.1007/978-1-4615-2025-2_1>.

HOBOLD, G. C.; SIQUEIRA, F. Discovery of Semantic Web Services Compositions Based on SAWSDL Annotations. In: *IEEE 19th International Conference on Web Services*. Washington, USA: IEEE, 2012. p. 280–287. Available at: <https://doi.org/10.1109/ICWS.2012.97>.

HODGES, W. Elementary predicate logic. In: _____. *Handbook of Philosophical Logic: Volume I: Elements of Classical Logic*. Dordrecht, Germany: Springer, 1983. p. 1–131. ISBN 978-94-009-7066-3. Available at: <https://doi.org/10.1007/978-94-009-7066-3_1>.

HOFFMANN, J.; NEBEL, B. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, v. 14, p. 253–302, 2001. Available at: <https://doi.org/10.1613/jair.855>.

HOFFMANN, J. et al. Combining scalability and expressivity in the automatic composition of semantic web services. In: *2008 Eighth International Conference on Web Engineering*. Washington, USA: IEEE, 2008. p. 98–107. Available at: <https://doi.org/10.1109/ICWE.2008.8>.

HOFWEBER, T. Logic and ontology. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2017. Metaphysics Research Lab, Stanford University, 2017. Available at: <https://plato.stanford.edu/archives/win2017/entries/logic-ontology/>.

HORROCKS, I. et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. [S.l.], 2004. Available at: <https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.

HUF, A.; SALVADORI, I. L.; SIQUEIRA, F. A service-oriented approach for integrating broadcast facilities. In: *IEEE International Conference on Services Computing, SCC 2016, San Francisco, CA, USA, June 27 - July 2, 2016*. Washington, USA: IEEE, 2016. p. 705–712. Available at: <https://doi.org/10.1109/SCC.2016.97>.

HUF, A.; SALVADORI, I. L.; SIQUEIRA, F. Planning and execution of heterogeneous service compositions. In: *2017 IEEE Symposium on Computers and Communications, ISCC 2017, Heraklion, Greece, July 3-6, 2017*. Washington, USA: IEEE, 2017. p. 987–993. Available at: <https://doi.org/10.1109/ISCC.2017.8024654>.

HUF, A.; SIQUEIRA, F. Uma arquitetura para composição de serviços com modelos de interação heterogêneos. In: *Anais do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres, 17 a 20 de outubro, 2017, Gramado, Rio Grande do Sul*. Porto Alegre, Brazil: Sociedade Brasileira de Computação, 2017. p. 11–16. ISBN 978-85-7669-380-2.

IEEE. *Systems and software engineering – Vocabulary*. [S.l.], 2010. 1–418 p. Available at: <https://doi.org/10.1109/IEEESTD.2010.57338>.

IM, C.; JEONG, C. Isomp: An instant service-orchestration mobile m2m platform. *Mobile Information Systems*, v. 2016, 2016. Available at: <https://doi.org/10.1155/2016/7263729>.

INFORMATION technology – Database languages – SQL – Part 1: Framework (SQL/Framework). Geneva, CH, 2016. v. 2016.

ISO. *Software engineering – Product quality.* [S.l.], 2001.

JAKOBOVITS, R. M.; MODAYUR, B.; BRINKLEY, J. F. A Web-based repository manager for brain mapping data. *Proceedings of the AMIA Annual Fall Symposium*, American Medical Informatics Association, p. 309–313, 1996. ISSN 1091-8280.

JATOTH, C. b.; GANGADHARAN, G.; BUYYA, R. Computational intelligence based QoS-aware web service composition: A systematic literature review. *IEEE Transactions on Services Computing*, PP, n. 99, 2015. Available at: <https://doi.org/10.1109/TSC.2015.2473840>.

JAYATHILAKA, H.; KRINTZ, C.; WOLSKI, R. Response time service level agreements for cloud-hosted web applications. In: *Proceedings of the Sixth ACM Symposium on Cloud Computing.* New York, USA: ACM, 2015. (SoCC '15), p. 315–328. ISBN 978-1-4503-3651-2. Available at: <https://doi.org/10.1145-/2806777.2806842>.

JINGYONG, L. et al. Middleware-based distributed systems software process. In: *Proceedings of the 2009 International Conference on Hybrid Information Technology.* New York, USA: ACM, 2009. (ICHIT '09), p. 345–348. ISBN 978-1-60558-662-5. Available at: <https://doi.org/10.1145/1644993.1645058>.

JORDAN, D. et al. *Web Services Business Process Execution Language Version 2.0.* [S.l.], 2007. Available at: <http://docs-.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.

JURIC, M. Wsdl and BPEL extensions for event driven architecture. *Information and Software Technology*, Butterworth-Heinemann, Newton, USA, v. 52, n. 10, p. 1023–1043, 2010. Available at: <https://doi.org/10.1016/j.infsof.2010.04.005>.

KAEBISCH, S.; KAMIYA, T. *Web of Things (WoT) Thing Description.* [S.l.], 2017. Available at: <https://www.w3.org/TR-/2017/WD-wot-thing-description-20170914/>.

KAJIMOTO, K.; KOVATSCH, M.; DAVULURU, U. *Web of Things (WoT) Architecture*. [S.l.], 2017. Available at: <https://www.w3.org/TR/2017/WD-wot-architecture-20170914/>.

KANG, Y.-B. et al. How long will it take? accurate prediction of ontology reasoning performance. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. Palo Alto, USA: AAAI Press, 2014. (AAAI'14), p. 80–86. Available at: <http://dl.acm.org/citation.cfm?id=2893873.2893888>.

KELLY, M. *JSON Hypertext Application Language*. [S.l.], 2016. Available at: <https://tools.ietf.org/html/draft-kelly-json-hal-08>.

KENDALL, M. G. A NEW MEASURE OF RANK correlation. *Biometrika*, v. 30, n. 1-2, p. 81–93, 1938. Available at: <https://doi.org/10.1093/biomet/30.1-2.81>.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3*. [S.l.], 2007. 1-57 p.

KLIMKO, G. Knowledge management and maturity models: Building common understanding. In: *Proceedings of the 2nd European Conference on Knowledge Management*. Bled, Slovenia: MCIL, 2001. p. 269–278.

KLINGBERG, T.; MANFREDI, R. *Gnutella 0.6*. [S.l.], 2002. Available at: <http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html>.

KLUSCH, M.; GERBER, A. Semantic web service composition planning with OWLS-xplan. In: *Proceedings of the 1st Int. AAAI Fall Symp. on Agents and the Semantic Web*. Arlington, USA: AAAI, 2005. p. 55–62.

KNUBLAUCH, H.; HENDLER, J. A.; IDEHEN, K. *SPIN - Overview and Motivation*. [S.l.], 2011. Available at: <http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/>.

KNUTH, D. E. Computer programming as an art. *Communications of the ACM*, ACM, New York, USA, v. 17, n. 12, p. 667–673, december 1974. ISSN 0001-0782. Available at: <https://doi.org/10.1145/361604.361612>.

KOCH, J.; ACKERMANN, P.; VELASCO, C. A. *HTTP Vocabulary in RDF 1.0*. [S.l.], 2017. Available at: <https://www.w3.org/TR/2017/NOTE-HTTP-in-RDF10-20170202/>.

KOENIG, S.; LIKHACHEV, M. Improved fast replanning for robot navigation in unknown terrain. In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*. [s.n.], 2002. v. 1, p. 968–975 vol.1. ISBN 1. Available at: <https://doi.org/10.1109/ROBOT.2002.1013481>.

KOENIG, S.; LIKHACHEV, M.; FURCY, D. Lifelong Planning A*. *Artificial Intelligence*, v. 155, n. 1, p. 93–146, 2004. ISSN 0004-3702. Available at: <https://doi.org/10.1016/j.artint.2003.12.001>.

KOPECKÝ, J.; GOMADAM, K.; VITVAR, T. hRESTS: An HTML Microformat for Describing RESTful Web Services. In: *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*. Washington, USA: IEEE, 2008. v. 1, p. 619–625. ISBN 1. Available at: <https://doi.org/10.1109/WIIAT.2008.379>.

KOPECKÝ, J. et al. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, v. 11, n. 6, p. 60–67, 2007. ISSN 1089-7801. Available at: <https://doi.org/10.1109/MIC.2007.134>.

KRUEGER, C. W. Software Reuse. *ACM Computing Surv.*, ACM, New York, USA, v. 24, n. 2, p. 131–183, june 1992. ISSN 0360-0300. Available at: <https://doi.org/10.1145/130844.130856>.

LAFON, Y.; MITRA, N. *SOAP Version 1.2 Part 0: Primer (Second Edition)*. [S.l.], 2007. Available at: <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.

LALIWALA, Z.; CHAUDHARY, S. Event-driven dynamic web services composition: From modeling to implementation. In: . [s.n.], 2006. Available at: <https://doi.org/10.1109/INNOVATIONS.2006.301884>.

LANE, K. *History of APIs*. 2017. Available at: <https://history.apievangelist.com/>. Access on: 08 nov. 2017.

LANTHALER, M.; GÜTL, C. Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In: *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013) at the 22nd*

International World Wide Web Conference (WWW2013). Rio de Janeiro, Brazil: CEUR-WS.org, 2013.

LANTHALER, M.; GüTL, C. Towards a RESTful service ecosystem. In: *4th IEEE International Conference on Digital Ecosystems and Technologies*. Washington, USA: IEEE, 2010. p. 209–214. ISSN 2150-4938. Available at: <https://doi.org/10.1109-/DEST.2010.5610644>.

LANTHALER, M.; KELLOGG, G.; SPORNY, M. *JSON-LD 1.0*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-json-ld-20140116/>.

LASSILA, O.; SWICK, R. R. *Resource Description Framework (RDF) Model and Syntax Specification*. [S.l.], 1999. Available at: <https://www.w3.org/TR/1999/REC-rdf-syntax-19990222>.

LEE, J.; LEE, S.-j.; WANG, P.-f. A Framework for Composing SOAP, Non-SOAP and Non-Web Services. *IEEE Transactions on Services Computing*, v. 8, n. 2, p. 240–250, 2015. ISSN 1939-1374. Available at: <https://doi.org/10.1109/TSC.2014.2310213>.

LEE, J.; LEE, S.-J.; WANG, P.-F. A framework for composing SOAP, non-SOAP and non-web services. *IEEE Transactions on Services Computing*, v. 8, n. 2, p. 240–250, 2015. Available at: <https://doi.org/10.1109/TSC.2014.2310213>.

LEE, W.-S.; LEE, S.-Y.; LEE, K.-C. Conflict detection and resolution method in WS-ECA framework. In: *International Conference on Advanced Communication Technology, ICACT*. Washington, USA: IEEE, 2007. v. 1, p. 786–791. Available at: <https://doi.org/10.1109/ICACT.2007.358468>.

LEITHEAD, T. et al. *HTML 5.1 2nd Edition*. [S.l.], 2017. Available at: <https://www.w3.org/TR/2017/REC-html51-20171003/>.

LEMOS, A. L.; DANIEL, F.; BENATALLAH, B. Web Service Composition: A Survey of Techniques and Tools. *ACM Computing Surveys*, v. 48, n. 3, p. 1–41, 2015. ISSN 03600300. Available at: <https://doi.org/10.1145/2831270>.

LI, L.; CHOU, W. Infoset for service abstraction and lightweight message processing. In: . Washington, USA: IEEE, 2009. p. 703–710. Available at: <https://doi.org/10.1109/ICWS.2009.120>.

LIECHTI, O. et al. Enabling reactive cities with the iflux middleware. In: *Proceedings of the 6th International Workshop on the Web of Things*. [s.n.], 2015. (ACM International Conference Proceeding Series, v. 26-October-2015). Available at: <https://doi.org/10.1145/2834791.2834794>.

LIN, L.; LIN, P. Orchestration in web services and real-time communications. *IEEE Communications Magazine*, v. 45, n. 7, p. 44–50, 2007. Available at: <https://doi.org/10.1109/MCOM.2007-.382659>.

LIU, X. et al. imashup: A mashup-based framework for service composition. *Science China Information Sciences*, v. 57, n. 1, p. 1–20, 2014. Available at: <https://doi.org/10.1007/s11432-013-4782-0>.

LOMOTEY, R. K.; DETERS, R. Composition of the electronic health record: Mobile efficiency in mhealth. In: *IEEE 6th International Conference on Service-Oriented Computing and Applications, SOCA 2013*. [s.n.], 2013. p. 146–153. Available at: <https://doi.org/10.1109/SOCA.2013.18>.

LORETO, S. et al. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. [S.l.], 2011. Available at: <http://www.rfc-editor.org/rfc/rfc2616.txt>.

LUMLEY, T. et al. The importance of the normality assumption in large public health data sets. *Annual Review of Public Health*, v. 23, n. 1, p. 151–169, 2002. PMID: 11910059. Available at: <https://doi.org/10.1146/annurev.publhealth.23.100901.140546>.

LUO, S.; XU, B.; SUN, K. Compose real web services with context. In: *2010 IEEE 8th International Conference on Web Services*. Washington, USA: IEEE, 2010. p. 630–631. Available at: <https://doi.org/10.1109/ICWS.2010.63>.

MACKENZIE, C. M. et al. *Reference Model for Service Oriented Architecture 1.0*. [S.l.], 2006.

MAJDA, E.; AHMED, E. Using cloud saas to ensure interoperability and standardization in heterogeneous cloud based environment. In: *Proceedings of the 2015 5th World Congress on Information and Communication Technologies, WICT 2015*. [s.n.], 2016. p. 29–34. Available at: <https://doi.org/10.1109/WICT-.2015.7489640>.

MALESHKOVA, M.; PEDRINACI, C.; DOMINGUE, J. Semantically annotating RESTful services with SWEET. In: *The 8th International Semantic Web Conference (ISWC 2009).* Washington, USA: [s.n.], 2009.

MALESHKOVA, M.; PEDRINACI, C.; DOMINGUE, J. Investigating Web APIs on the World Wide Web. In: *Web Services (ECOWS), 2010 IEEE 8th European Conference on.* Washington, USA: IEEE, 2010. p. 107–114. Available at: <https://doi.org/10.1109/ECOWS.2010.9>.

MANNAVA, V.; RAMESH, T. A novel way of invoking web services and RESTful web services using aspect oriented programming. *Communications in Computer and Information Science*, v. 197 CCIS, p. 665–674, 2011. Available at: <https://doi.org/10.1007/978-3-642-22543-7_68>.

MARTIN, D. et al. *OWL-S: Semantic Markup for Web Services.* [S.l.], 2007. Http://www.daml.org/services/owl-s/1.2/overview/.

MARTIN, D.; PAOLUCCI, M.; WAGNER, M. Bringing semantic annotations to web services: OWL-S from the SAWSDL perspective. In: *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings.* Berlin, Germany: Springer, 2007. p. 340–352. ISBN 978-3-540-76298-0. Available at: <https://doi.org/10.1007/978-3-540-76298-0_25>.

MAXIMILIEN, E. et al. A domain-specific language for web APIs and services mashups. Springer, Berlin, Germany, v. 4749 LNCS, p. 13–26, 2007. Available at: <https://doi.org/10.1007/978-3-540-74974-5_2>.

MAXIMILIEN, E. M. et al. A domain-specific language for web APIs and services mashups. In: KRÄMER, B. J.; LIN, K.-J.; NARASIMHAN, P. (Ed.). *Service-Oriented Computing – ICSOC 2007: Fifth International Conference, Vienna, Austria, September 17-20, 2007. Proceedings.* Berlin, Germany: Springer, 2007. p. 13–26. ISBN 978-3-540-74974-5. Available at: <https://doi.org/10.1007/978-3-540-74974-5_2>.

MCCARTHY, J.; HAYES, P. J. Some philosophical problems from the standpoint of artificial intelligence. In: _____. Los Altos, CA: Morgan Kaufmann, 1969. v. 4, p. 431–450.

MCDERMOTT, D. et al. *PDDL - The Planning Domain Definition Language*. [S.l.], 1998.

MCILRAITH, S.; SON, T. Semantic Web services. *IEEE Intelligent Systems*, v. 16, n. 2, p. 46–53, 2001. ISSN 1541-1672. Available at: <https://doi.org/10.1109/5254.920599>.

MELL, P.; GRANCE, T. et al. *The NIST Definition of Cloud Computing*. Gaithersburg, USA, 2011.

MENG, Q. et al. A multiform SWIM service delivery platform for air traffic management environment. *International Journal of Digital Content Technology and its Applications*, v. 6, n. 18, p. 393–400, 2012. Available at: <https://doi.org/10.4156/jdcta.vol6-.issue18.47>.

MESHKOVA, E. et al. A Survey on Resource Discovery Mechanisms, Peer-to-peer and Service Discovery Frameworks. *Computer Networks*, Elsevier North-Holland, Inc., New York, USA, v. 52, n. 11, p. 2097–2128, august 2008. ISSN 1389-1286. Available at: <https://doi.org/10.1016/j.comnet.2008.03.006>.

MILANOVIC, N.; MALEK, M. Current solutions for web service composition. *IEEE Internet Computing*, v. 8, n. 6, p. 51–59, november 2004. ISSN 1089-7801. Available at: <https://doi.org/10.1109/MIC.2004.58>.

MILLER, G. A. Wordnet: A lexical database for english. *Communications of the ACM*, ACM, New York, USA, v. 38, n. 11, p. 39–41, november 1995. ISSN 0001-0782. Available at: <https://doi.org/10.1145/219717.219748>.

MILNER, R. *Communicating and mobile systems: the π calculus*. New York, USA: Cambridge university press, 1999. ISBN 0-521-65869-1.

MINGUEZ, J.; ZOR, S.; REIMANN, P. Event-driven business process management in engineer-to-order supply chains. In: *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2011*. [s.n.], 2011. p. 624–631. Available at: <https://doi.org/10.1109-/CSCWD.2011.5960183>.

MIZOUNI, R. et al. Performance Evaluation of Mobile Web Services. In: *2011 IEEE Ninth European Conference on Web*

Services. Washington, USA: IEEE, 2011. p. 184–191. Available at: <https://doi.org/10.1109/ECOWS.2011.12>.

MOCKAPETRIS, P.; DUNLAP, K. J. Development of the domain name system. In: *Symposium Proceedings on Communications Architectures and Protocols*. New York, USA: ACM, 1988. (SIGCOMM '88), p. 123–133. ISBN 0-89791-279-9. Available at: <https://doi.org/10.1145/52324.52338>.

MOTIK, B.; PATEL-SCHNEIDER, P. F.; PARSIA, B. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. [S.l.], 2012. Available at: <http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>.

MULLIGAN, G.; GRAČANIN, D. A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. Washington, USA: IEEE, 2009. p. 1423–1432. ISSN 0891-7736. Available at: <https://doi.org/10.1109/WSC.2009.5429290>.

MURER, T.; SCHERER, D.; WÜRTZ, A. Improving component interoperability information. In: *Workshop on Component-Oriented Programming at ECOOP'96 (WCOP-96)*. [S.l.: s.n.], 1996.

MURGUZUR, A. et al. Process flexibility in service orchestration: A systematic literature review. *International Journal of Cooperative Information Systems*, v. 23, n. 3, 2014. Available at: <https://doi.org/10.1142/S0218843014300010>.

NARDI, J. C. et al. A commitment-based reference ontology for services. *Information Systems*, v. 54, n. Supplement C, p. 263–288, 2015. ISSN 0306-4379. Available at: <https://doi.org/10.1016/j.is-.2015.01.012>.

NAU, D. et al. Shop2: An HTN planning system. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 20, n. 1, p. 379–404, december 2003. ISSN 1076-9757.

NEJDL, W. et al. Edutella: A p2p networking infrastructure based on rdf. In: *Proceedings of the 11th International Conference on World Wide Web*. New York, USA: ACM, 2002. (WWW '02), p. 604–615. ISBN 1-58113-449-5. Available at: <https://doi.org/10.1145/511446.511525>.

NELSON, T. H. Complex information processing: A file structure for the complex, the changing and the indeterminate. In: *Proceedings of the 1965 20th National Conference*. New York, USA: ACM, 1965. (ACM '65), p. 84–100. Available at: <https://doi.org/10.1145/800197.806036>.

NILES, I.; PEASE, A. Towards a standard upper ontology. In: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*. New York, USA: ACM, 2001. (FOIS '01), p. 2–9. ISBN 1-58113-377-4. Available at: <https://doi.org/10.1145/505168.505170>.

NORTON, B. et al. *Reference Ontology for Semantic Service Oriented Architectures*. [S.l.], 2008.

NOTTINGHAM, M.; SAYRE, R. *The Atom Syndication Format*. [S.l.], 2005. Available at: <https://doi.org/10.17487/rfc4287>.

TOM, R. et al. (Ed.). *Web Services Basic Profile 1.2*. [S.l.], 2014.

OBJECT MANAGEMENT GROUP. *Business Process Model and Notation (BPMN)*: Version 2.0. [S.l.], 2011. Available at: <http://www.omg.org/spec/BPMN/2.0>.

OBJECT MANAGEMENT GROUP. *CORBA Component Model Specification*. [S.l.], 2015.

OBJECT MANAGEMENT GROUP. *Unified Modeling Language*. [S.l.], 2015.

OLIVEIRA, B. C. N. et al. Automatic semantic enrichment of data services. In: *iiWAS '17: The 19th International Conference on Information Integration and Web-based Applications & Services, December 4–6, 2017, Salzburg, Austria*. New York, USA: ACM, 2017. p. 415–424. Available at: <https://doi.org/10.1145/3151759.3151783>.

OLIVEIRA, B. C. N. et al. A platform to enrich, expand and publish linked data of police reports. In: *15th International Conference on WWW/INTERNET 2016, 2016, Mannheim. Proceedings of the IADIS International Conference WWW/Internet*. Manheim, Germany: IADIS, 2016. p. 118–118. ISBN 978-989-8533-57-9.

OTERO-CERDEIRA, L.; RODRÍGUEZ-MARTÍNEZ, F. J.; GÓMEZ-RODRÍGUEZ, A. Ontology matching: A literature review. *Expert Systems with Applications*, Elsevier Ltd, v. 42, n. 2, p. 949–971, 2015. ISSN 09574174. Available at: <https://doi.org/10.1016/j.eswa.2014.08.032>.

PAIK, H. young et al. *Web Service Implementation and Composition Techniques*. Cham, Switzerland: Springer, 2017. ISBN 978-3-319-55542-3.

PANZIERA, L. et al. Distributed matchmaking and ranking of web APIs exploiting descriptions from web sources. In: *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. Washington, USA: IEEE, 2011. p. 1–8. ISSN 2163-2871. Available at: <https://doi.org/10.1109/SOCA.2011-.6166201>.

PAOLUCCI, M. et al. Semantic matching of web services capabilities. In: HORROCKS, I.; HENDLER, J. (Ed.). *The Semantic Web — ISWC 2002: First International Semantic Web Conference Sardinia, Italy, June 9–12, 2002 Proceedings*. Berlin, Germany: Springer, 2002. p. 333–347. ISBN 978-3-540-48005-1. Available at: <https://doi.org/10.1007/3-540-48005-6_26>.

PAPAZOGLOU, M. P. Service-oriented computing: concepts, characteristics and directions. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* [s.n.], 2003. p. 3–12. Available at: <https://doi.org/10.1109/WISE.2003.1254461>.

PAPAZOGLOU, M. P. et al. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, World Scientific, v. 17, n. 02, p. 223–255, 2008.

PATEL-SCHNEIDER, P.; HAYES, P. *RDF 1.1 Semantics*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>.

PATEL-SCHNEIDER, P. F.; HORROCKS, I.; HARMELEN, F. van. *OWL Web Ontology Language 1.0 Abstract Syntax*. [S.l.], 2002. Available at: <http://www.w3.org/TR/2002/WD-owl-absyn-20020729/>.

PAUTASSO, C. Bpel for rest. In: DUMAS, M.; REICHERT, M.; SHAN, M.-C. (Ed.). *Business Process Management. 6th International Conference, BPM 2008, Milan, Italy, September 2-4,* *2008. Proceedings.* Berlin, Germany: Springer, 2008. p. 278–293. ISBN 978-3-540-85758-7. Available at: <https://doi.org/10.1007-/978-3-540-85758-7_21>.

PAUTASSO, C. Restful web service composition with BPEL for REST. *Data and Knowledge Engineering*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 68, n. 9, p. 851–866, 2009. Available at: <https://doi.org/10.1016/j.datak-.2009.02.016>.

PAUTASSO, C.; WILDE, E. Push-enabling RESTful business processes. In: *Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings.* Berlin, Germany: Springer, 2011. (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v. 7084 LNCS), p. 32–46. ISBN 978-3-642-25535-9. Available at: <https://doi.org/10.1007/978-3-642-25535-9_3>.

PEDRINACI, C. et al. iServe: a linked services publishing platform. In: *Proceedings of the 1st Workshop on Ontology Repositories and Editors for the Semantic Web.* Hersonissos, Greece: CEUR-WS.org, 2010. v. 596.

PEER, J. A PDDL based tool for automatic web service composition. In: OHLBACH, H. J.; SCHAFFERT, S. (Ed.). *Principles and Practice of Semantic Web Reasoning: Second International Workshop, PPSWR 2004, St. Malo, France, September 6-10, 2004. Proceedings.* Berlin, Germany: Springer, 2004. p. 149–163. ISBN 978-3-540-30122-6. Available at: <https://doi.org/10.1007/978-3-540-30122-6_11>.

PELTZ, C. Web services orchestration and choreography. *Computer*, v. 36, n. 10, p. 46–52, october 2003. ISSN 0018-9162. Available at: <https://doi.org/10.1109/MC.2003.1236471>.

PENG, Y. Y.; MA, S. P.; LEE, J. Rest2SOAP: A framework to integrate SOAP services and RESTful services. In: *IEEE International Conference on Service-Oriented Computing and Applications, SOCA' 09.* [s.n.], 2009. p. 106–109. Available at: <https://doi.org/10.1109/SOCA.2009.5410458>.

PERRY, D. E.; WOLF, A. L. Foundations for the Study of Software Architecture. *SIGSOFT Softw. Eng. Notes*, ACM, New

York, USA, v. 17, n. 4, p. 40–52, october 1992. ISSN 0163-5948. Available at: <https://doi.org/10.1145/141874.141884>.

PETERSON, D. et al. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. [S.l.], 2012. Available at: <http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>.

PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide*. Malden, USA: John Wiley & Sons, 2008. ISBN 978-1-4051-2110-1.

PICO-VALENCIA, P.; HOLGADO-TERRIZA, J. An agent middleware for supporting ecosystems of heterogeneous web services. In: *The 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops*. Amsterdam, The Netherlands: Elsevier, 2016. (Procedia Computer Science, v. 94), p. 121–128. Available at: <https://doi.org/10.1016/j.procs.2016.08.020>.

PIETZUCH, P. R. *Hermes: A scalable event-based middleware*. [S.l.], 2004. Available at: <http://www.cl.cam.ac.uk/techreports-/UCAM-CL-TR-590.pdf>.

POSTEL, J. *Internet Control Message Protocol*. [S.l.], 1981. Available at: <https://doi.org/10.17487/rfc792>.

PREIST, C. A conceptual architecture for semantic web services. In: MCILRAITH, S. A.; PLEXOUSAKIS, D.; HARMELEN, F. van (Ed.). *The Semantic Web – ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*. Berlin, Germany: Springer, 2004. p. 395–409. ISBN 978-3-540-30475-3. Available at: <https://doi.org/10.1007/978-3-540-30475-3_28>.

PUDER, A.; RÖMER, K.; PILHOFER, F. *Distributed systems architecture: a middleware approach*. San Francisco, USA: Elsevier, 2006. ISBN 978-1-55860-648-7.

QUARTEL, D. A. C. et al. COSMO: A conceptual framework for service modelling and refinement. *Information Systems Frontiers*, v. 9, n. 2, p. 225–244, july 2007. ISSN 1572-9419. Available at: <https://doi.org/10.1007/s10796-007-9034-7>.

RAIMOND, Y.; SCHREIBER, G. *RDF 1.1 Primer*. [S.l.], 2014. Available at: <http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/>.

RAJARAM, K.; BABU, C.; GANESAN, A. DILT: A Hybrid Model for Dynamic Composition and Execution of Heterogeneous Web Services. In: *Proceedings of the 11th International Conference on Distributed Computing and Internet Technology - Volume 8956*. New York, USA: Springer, 2015. (ICDCIT 2015), p. 239–244. Available at: <https://doi.org/10.1007/978-3-319-14977-6_20>.

RAMBOLD, M. et al. Towards Autonomic Service Discovery A Survey and Comparison. In: *Proceedings of the 2009 IEEE International Conference on Services Computing*. Washington, USA: IEEE, 2009. p. 192–201. ISBN 978-0-7695-3811-2. Available at: <https://doi.org/10.1109/SCC.2009.59>.

RAMTOHUL, A.; OGUNLEYE, B. Rest and BPEL for e-government shared services. In: *Proceedings of the European Conference on e-Government, ECEG*. Reading, United Kingdom: ACPI, 2017. Part F129463, p. 339–348.

RAN, S. A model for web services discovery with qos. *SIGecom Exch.*, ACM, New York, USA, v. 4, n. 1, p. 1–10, mar 2003. ISSN 1551-9031. Available at: <https://doi.org/10.1145/844357-.844360>.

RAO, J.; KUNGAS, P.; MATSKIN, M. Logic-based web services composition: from service description to process model. In: *Proceedings of the IEEE International Conference on Web Services, 2004*. Washington, USA: IEEE, 2004. p. 446–453. Available at: <https://doi.org/10.1109/ICWS.2004.1314769>.

RENZEL, D.; SCHLEBUSCH, P.; KLAMMA, R. Today's Top "RESTful" Services and Why They Are Not RESTful. In: WANG, X. S. et al. (Ed.). *Web Information Systems Engineering - WISE 2012: 13th International Conference, Paphos, Cyprus, November 28-30, 2012. Proceedings*. Berlin, Germany: Springer, 2012. p. 354–367. ISBN 978-3-642-35063-4. Available at: <https://doi.org/10.1007/978-3-642-35063-4_26>.

REZGUI, A. et al. Preserving privacy in web services. In: *Proceedings of the 4th International Workshop on Web Information and Data Management*. New York, USA: ACM,

2002. (WIDM ’02), p. 56–62. ISBN 1-58113-593-9. Available at: <https://doi.org/10.1145/584931.584944>.

RICHARDSON, L. *Justice Will Take Us Millions Of Intricate Moves*. 2008. Talk presented at QCon 2008. Access on: 31 oct 2017.

RODRIGUEZ-MIER, P. et al. An Optimal and Complete Algorithm for Automatic Web Service Composition. *International Journal of Web Services Research*, IGI Global, v. 9, n. 2, p. 1–20, 2012. ISSN 1545-7362. Available at: <https://doi.org/10.4018-/jwsr.2012040101>.

RODRIGUEZ-MIER, P. et al. An Integrated Semantic Web Service Discovery and Composition Framework. *IEEE Transactions on Services Computing*, v. 9, n. 4, p. 537–550, 2016. Available at: <https://doi.org/10.1109/TSC.2015.2402679>.

ROMAN, D. et al. Web service modeling ontology. *Applied ontology*, IOS Press, v. 1, n. 1, p. 77–106, 2005. ISSN 1570-5838.

ROMAN, D. et al. Wsmo-lite and hRESTs: Lightweight semantic annotations for web services and RESTful APIs. *Journal of Web Semantics*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 31, p. 39–58, 2015. Available at: <https://doi.org/10.1016/j.websem.2014.11.006>.

SAHIN, O. D. et al. Spider: P2p-based web service discovery. In: BENATALLAH, B.; CASATI, F.; TRAVERSO, P. (Ed.). *Service-Oriented Computing - ICSOC 2005: Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005. Proceedings*. Berlin, Germany: Springer, 2005. p. 157–169. ISBN 978-3-540-32294-8. Available at: <https://doi.org/10.1007-/11596141_13>.

SAINT-ANDRE, P. *Extensible Messaging and Presence Protocol (XMPP): Core*. [S.l.], 2011. Available at: <https://doi.org/10-.17487/rfc6120>.

SALVADORI, I.; HUF, A.; SIQUEIRA, F. An agent-based composition model for semantic microservices. In: *15th International Conference on WWW/INTERNET 2016, 2016, Mannheim. Proceedings of the IADIS International Conference WWW/Internet*. Manheim, Germany: IADIS, 2016. p. 75–82. ISBN 978-989-8533-57-9.

SALVADORI, I. et al. An ontology alignment framework for data-driven microservices. In: *iiWAS '17: The 19th International Conference on Information Integration and Web-based Applications & Services, December 4–6, 2017, Salzburg, Austria*. New York, USA: ACM, 2017. p. 425–433. Available at: <https://doi.org/10.1145/3151759.3151793>.

SALVADORI, I.; SIQUEIRA, F. A maturity model for semantic RESTful web APIs. In: *Web Services (ICWS), 2015 IEEE International Conference on*. Washington, USA: IEEE, 2015. p. 703–710. Available at: <https://doi.org/10.1109/ICWS.2015.98>.

SALVADORI, I. L. et al. Publishing linked data through semantic microservices composition. In: *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, iiWAS 2016, Singapore, November 28-30, 2016*. New York, USA: ACM, 2016. p. 443–452. Available at: <http://doi.org/10.1145/3011141.3011155>.

SALVADORI, I. L. et al. Improving entity linking with ontology alignment for semantic microservices composition. *International Journal of Web Information Systems*, v. 13, n. 3, p. 302–323, 2017. Available at: <https://doi.org/10.1108/IJWIS-04-2017-0029>.

SAMPSON, S. E. The unified service theory. In: _____. *Handbook of Service Science*. Boston, MA: Springer, 2010. p. 107–131. ISBN 978-1-4419-1628-0. Available at: <https://doi.org/10.1007/978-1-4419-1628-0_7>.

SCHERP, A. et al. Designing core ontologies. *Applied Ontology*, IOS Press, v. 6, n. 3, p. 177–221, 2011.

SEABORNE, A. An RDF netAPI. In: HORROCKS, I.; HENDLER, J. (Ed.). *The Semantic Web — ISWC 2002: First International Semantic Web Conference Sardinia, Italy, June 9–12, 2002 Proceedings*. Berlin, Germany: Springer, 2002. p. 399–403. ISBN 978-3-540-48005-1. Available at: <https://doi.org/10.1007/3-540-48005-6_31>.

SEABORNE, A.; HARRIS, S. *SPARQL 1.1 Query Language*. [S.l.], 2013. Available at: <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.

SEIDEWITZ, E. *What models mean*. 2003. 26–32 p. Available at: <https://doi.org/10.1109/MS.2003.1231147>.

SHAPIRO, M. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In: IEEE. *Int. Conference on Distr. Comp. Sys. (ICDCS)*. Cambridge, MA, USA, United States, 1986. (Int. Conference on Distr. Comp. Sys. (ICDCS)), p. 198–204. Available at: <https://hal.inria.fr/inria-00444651>.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The Constrained Application Protocol (CoAP)*. [S.l.], 2014. Available at: <https://doi.org/10.17487/rfc7252>.

SHENG, Q. Z. et al. Web services composition: A decade's overview. *Information Sciences*, v. 280, p. 218–238, october 2014. ISSN 00200255. Available at: <https://doi.org/10.1016/j.ins.2014-.04.054>.

SIBIRTSEVA, A. et al. Sswim: A semantic service, wrapper and invocation manager. In: *10th IEEE Joint Conference on E-Commerce Technology and the 5th Enterprise Computing, E-Commerce and E-Services, CEC 2008 and EEE 2008*. Washington, USA: IEEE, 2008. (CECANDEEE'08), p. 175–182. Available at: <https://doi.org/10.1109/CECandEEE.2008.120>.

SINGH, M. P. Being Interactive: Physics of Service Composition. *IEEE Internet Computing*, v. 5, n. 3, p. 6–7, 2001. Available at: <https://doi.org/10.1109/MIC.2001.10014>.

SINHA, A. Client-server Computing. *Communications of the ACM*, ACM, New York, USA, v. 35, n. 7, p. 77–98, 1992. ISSN 0001-0782. Available at: <https://doi.org/10.1145/129902.129908>.

SIRIN, E. et al. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 5, n. 2, p. 51 – 53, 2007. ISSN 1570-8268. Available at: <https://doi.org/10.1016/j.websem.2007.03.004>.

SIRIN, E. et al. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 1, n. 4, p. 377 – 396, 2004. ISSN 1570-8268. International Semantic Web Conference 2003. Available at: <https://doi.org/10.1016/j.websem.2004.06.005>.

SPOHRER, J. et al. The service system is the basic abstraction of service science. In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. Waikoloa, USA: IEEE, 2008. p. 104–113. ISBN 978-0-7695-3075-8. Available at: <https://doi.org/10.1109/HICSS.2008.451>.

SRBLJIĆ, S.; ŠKVORC, D.; SKROBO, D. Programming language design for event-driven service composition [Oblikovanje programskih jezika za dogaajima poticanu kompoziciju usluga]. *Automatika*, v. 51, n. 4, p. 374–386, 2010.

STANIK, A.; KAO, O. A proposal for REST with XMPP as base protocol for intercloud communication. In: *2016 7th International Conference on Information, Intelligence, Systems Applications (IISA)*. Washington, USA: IEEE, 2016. p. 1–6. Available at: <https://doi.org/10.1109/IISA.2016.7785436>.

STRANG, T.; LINNHOF-POPIEN, C. Service Interoperability on Context Level in Ubiquitous Computing Environments. In: *Proceedings of the 2003 International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet.* [s.n.], 2003. Available at: <http://elib.dlr.de/7267/>.

SUN, P.-L.; KU, C.-Y. Review of threats on trust and reputation models. *Industrial Management and Data Systems*, v. 114, n. 3, p. 472–483, 2014. Available at: <https://doi.org/10.1108/IMDS-11-2013-0470>.

SUPAVETCH, S.; CHUNITHIPAISAN, S. Interface independent geospatial services orchestration. *Information Technology Journal*, v. 10, n. 6, p. 1126–1137, 2011. Available at: <https://doi.org/10.3923/itj.2011.1126.1137>.

TAHER, Y. et al. A causal tracking monitoring approach for business transaction management. In: *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science*. Setúbal, Portugal: SciTePress, 2013. p. 140–144. ISBN 978-989-8565-52-5. Available at: <https://doi.org/10.5220/0004380601400144>.

TAO, Y. et al. Scientific experimental platform based on BPEL and hadoop and its application to sound activity recognition. *Journal of Information and Computational Science*, v. 12, n. 18, p. 6999–7009, 2015. Available at: <https://doi.org/10.12733/jics20150398>.

TERLOUW, L. I.; ALBANI, A. An enterprise ontology-based approach to service specification. *IEEE Transactions on Services Computing*, v. 6, n. 1, p. 89–101, june 2013. ISSN 1939-1374. Available at: <https://doi.org/10.1109/TSC.2011.38>.

THAKAR, U.; TIWARI, A.; VARMA, S. On composition of SOAP based and RESTful services. In: *6th International Advanced Computing Conference, IACC 2016*. [s.n.], 2016. p. 500–505. Available at: <https://doi.org/10.1109/IACC.2016.99>.

THE JSON Data Interchange Format. [S.l.], 2013. Https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

TSALGATIDOU, A.; ATHANASOPOULOS, G.; PANTAZOGLOU, M. Interoperability Among Heterogeneous Services: The Case of Integration of P2P Services with Web Services. *International Journal of Web Services Research*, v. 5, n. 4, p. 79–110, october 2008. Available at: <https://search.proquest.com-/docview/203138491?accountid=26642>.

TSARKOV, D.; HORROCKS, I. FaCT++ description logic reasoner: System description. In: FURBACH, U.; SHANKAR, N. (Ed.). *Automated Reasoning: Third International Joint Conference, IJCAR 2006 Seattle, WA, USA, August 17-20, 2006 Proceedings*. Berlin, Germany: Springer, 2006. p. 292–297. ISBN 978-3-540-37188-5. Available at: <https://doi.org/10.1007-/11814771_26>.

UPADHYAYA, B.; ZOU, Y. Integrating heterogeneous web services from an end user perspective. In: *Proceedings of the 9th Middleware Doctoral Symposium of the 13th ACM/IFIP/USENIX International Middleware Conference, MIDDLEWARE 2012*. New York, USA: ACM, 2012. (MIDDLEWARE ’12). Available at: <https://doi.org/10.1145/2405688.2405695>.

UPADHYAYA, B. et al. Migration of SOAP-based services to RESTful services. In: *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*. Washington, USA: IEEE, 2011. p. 105–114. ISSN 1550-4441. Available at: <https://doi.org/10.1109/WSE.2011.6081828>.

UPADHYAYA, B. et al. Migration of SOAP-based services to RESTful services. In: *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*. [s.n.], 2011. p. 105–114. Available at: <https://doi.org/10.1109/WSE.2011.6081828>.

VALLECILLO, A.; HERNÁNDEZ, J.; TROYA, J. M. New issues in object interoperability. In: SPRINGER. *European Conference on Object-Oriented Programming*. [S.l.], 2000. p. 256–269.

Van Hoecke, S. et al. SAMuS: Service-oriented architecture for multisensor surveillance in smart homes. *The Scientific World Journal*, Hindawi, London, UK, v. 2014, 2014. ISSN 1537744X. Available at: <https://doi.org/10.1155/2014/150696>.

VARGO, S. L.; LUSCH, R. F. Evolving to a new dominant logic for marketing. *Journal of Marketing*, American Marketing Association, v. 68, n. 1, p. 1–17, january 2004. ISSN 0022-2429. Available at: <https://doi.org/10.1509/jmkg.68.1.1.24036>.

VASKO, M.; DUSTDAR, S. Introducing collaborative service mashup design. In: *Proceedings of the First International Workshop on Lightweight Integration on the Web (ComposableWeb'09)*. San Sebastian, Spain: CEUR-WS.org, 2007. (CEUR Workshop Proceedings, v. 470), p. 51–62.

VERBORGH, R. et al. The pragmatic proof: Hypermedia API composition and execution. *Theory and Practice of Logic Programming*, v. 17, p. 1–48, 2016. ISSN 1475-3081. Available at: <https://doi.org/10.1017/S1471068416000016>.

VERBORGH, R.; ROO, J. D. Drawing conclusions from linked data on the web: The EYE reasoner. *IEEE Software*, v. 32, n. 3, p. 23–27, 2015. ISSN 0740-7459. Available at: <https://doi.org/10.1109/MS.2015.63>.

VERBORGH, R. et al. Capturing the functionality of Web services with functional descriptions. *Multimedia Tools and Applications*, v. 64, n. 2, p. 365–387, 2013. ISSN 1573-7721. Available at: <https://doi.org/10.1007/s11042-012-1004-5>.

VISSERS, C. A.; LOGRIPPO, L. The importance of the service concept in the design of data communications protocols. In: *Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification*. Amsterdam, The Netherlands: Elsevier North-Holland, 1986. p. 3–17.

VITVAR, T. et al. WSMO-Lite: lightweight semantic descriptions for services on the web. In: *Web Services, 2007. ECOWS '07. Fifth European Conference on*. Washington, USA: IEEE, 2007. p. 77–86. Available at: <https://doi.org/10.1109/ECOWS.2007.30>.

WANG, Y.; VASSILEVA, J. A review on trust and reputation for web service selection. In: *Distributed Computing Systems Workshops, 2007. ICDCSW '07. 27th International Conference* *on*. Washington, USA: IEEE, 2007. p. 25–25. ISSN 1545-0678. Available at: <https://doi.org/10.1109/ICDCSW.2007.16>.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. *REST in practice: Hypermedia and systems architecture*. [S.l.]: O'Reilly Media, Inc., 2010. ISBN 978-0-596-80582-1.

WEERAWARANA, S. et al. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. [S.l.], 2007. Available at: <http://www.w3.org/TR/2007/REC-wsdl20-20070626>.

WEGNER, P. Interoperability. *ACM Computing Surv.*, ACM, New York, USA, v. 28, n. 1, p. 285–287, mar 1996. ISSN 0360-0300. Available at: <https://doi.org/10.1145/234313.234424>.

WENDLER, R. The maturity of maturity model research: A systematic mapping study. *Information and Software Technology*, v. 54, n. 12, p. 1317–1339, 2012. ISSN 0950-5849. Available at: <https://doi.org/10.1016/j.infsof.2012.07.007>.

WINER, D. *RSS 2.0 Specification*. 2003. Available at: <https://cyber.harvard.edu/rss/rss.html>. Access on: 01 sep 2017.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. *The knowledge engineering review*, Cambridge University Press, v. 10, n. 2, p. 115–152, 1995. Available at: <https://doi.org/10.1017/S0269888900008122>.

WU, B.; LIN, R.; CHEN, J. Integrating RESTful service into BPEL business process on service generation system. In: *IEEE 10th International Conference on Services Computing, SCC 2013*. Washington, USA: IEEE, 2013. (SCC '13), p. 527–534. Available at: <https://doi.org/10.1109/SCC.2013.70>.

WU, Z. et al. A web-based two-layered integration framework for smart devices. *Eurasip Journal on Wireless Communications and Networking*, v. 2012, 2012. Available at: <https://doi.org/10.1186-/1687-1499-2012-150>.

YOO, H.; PARK, Y.; BAE, H. Semi-automatic semantic service annotation for SOAP and REST web services. *Communications in Computer and Information Science*, v. 194 CCIS, p. 70–77, 2011. Available at: <https://doi.org/10.1007/978-3-642-22603-8_7>.

YU, G. et al. Sensor web service integration for pandemic disease spread simulation. In: *2009 17th International Conference on Geoinformatics*. Washington, USA: IEEE, 2009. ISBN 978-1-4244-4563-9. ISSN 2161-024X. Available at: <https://doi.org/10.1109/GEOINFORMATICS.2009.5293497>.

ZENG, L. et al. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, v. 30, n. 5, p. 311–327, may 2004. ISSN 0098-5589. Available at: <https://doi.org/10.1109/TSE.2004.11>.

ZHANG, L. et al. Research on IOT RESTful web service asynchronous composition based on bpel. In: *6th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2014*. Washington, USA: IEEE, 2014. v. 1, p. 62–65. Available at: <https://doi.org/10.1109/IHMSC.2014.23>.

ZHANG, L. J. *EIC Editorial: Introduction to the Knowledge Areas of Services Computing*. 2008. 62–74 p. Available at: <https://doi.org/10.1109/TSC.2008.15>.

ZHAO, H.; DOSHI, P. Towards Automated RESTful Web Service Composition. In: *2009 IEEE International Conference on Web Services*. Washington, USA: IEEE, 2009. p. 189–196. Available at: <https://doi.org/10.1109/ICWS.2009.111>.

ZHAO, X. et al. RESTful web service composition: Extracting a process model from Linear Logic theorem proving. In: *2011 7th International Conference on Next Generation Web Services Practices*. Washington, USA: IEEE, 2011. p. 398–403. Available at: <https://doi.org/10.1109/NWeSP.2011.6088212>.

ZSEBY, T.; ZANDER, S.; CARLE, G. *Policy-Based Accounting*. [S.l.], 2002. Available at: <https://doi.org/10.17487/rfc3334>.

APPENDIX A – PROGRAMMABLEWEB ANALYSIS

This appendix discusses the method used for the keyword analysis whose results were presented in Figure 1 and presents some additional results.

The keyword analysis on ProgrammableWeb[1] data was done using a dump generated by a scrapper[2]. The scrapper ran on august 12, 2016 and September 1st, 2017 and retrieved 13,892 services in 2016 and 16,473 in 2017. The services were identified by their ProgrammableWeb page URI, as we assume ProgrammableWeb has no duplicates. ProgrammableWeb data appears to not strictly adhere to a hierarchy of service types. Therefore the set of Structured Query Language (SQL) queries shown in Listing 13 was run against the SQLite[3] databases produced by the scrapper to classify APIs within the hierarchy used by this dissertation as defined in Chapter 1.

To identify REST services, the UPDATE on lines 14-22 of Listing 13 looks for the keyword REST in several fields related to protocols, formats and architectural style. Any service which declares to be an “Hypermedia API” is also considered as a REST service. Although HATEOAS does not strictly implies full adoption of REST, it is reasonable to expect so in practice, and the keyword-based method of identifying REST services does not support any conclusion on actual conformance to all REST constraints. The UPDATE on lines 24-26 identifies Hypermedia API using the specific field.

As for SOAP services, their identification is similar to REST and is done by the UPDATE on lines 28-35 in Listing 13. We assume that due to the very purpose of ProgrammableWeb, any service that is not identified as a SOAP service and that has data about its protocol or supported formats is a Web API (lines 37-47). This assumption seems reasonable, as the query for services with missing data (lines 49-56) found no matches in 2017 data.

The adopted method is vulnerable to the quality of keywords in ProgrammableWeb. In fact, Maleshkova, Pedrinaci & Domingue (2010) concluded that by 2010, 67.6% of services listed as REST in ProgrammableWeb actually did not respected the semantics of the HTTP methods (e.g., GET requests were not safe), violating the uniform interface constraint. This analysis relied on manual inspection of a sample set of services, and the study had no follow-up. Nevertheless, this

---

[3] <https://www.programmableweb.com/>
[3] <https://github.com/IvanGoncharov/ProgrammableWeb>
[3] <https://sqlite.org/>

```sql
DROP TABLE IF EXISTS tmp;
CREATE TABLE tmp (url TEXT,
                  isREST INTEGER,
                  isHypermedia INTEGER,
                  isSOAP INTEGER,
                  usesXML INTEGER,
                  isWebApi INTEGER,
                  noData INTEGER,
                  PRIMARY KEY(url));

INSERT INTO tmp(url, isREST, isSOAP, usesXML, isWebApi, noData)
    SELECT url, 0, 0, 0, 0, 0 FROM apis;

UPDATE tmp SET isREST = 1 WHERE url IN (
        SELECT url FROM apis WHERE UPPER(Protocol_Formats) LIKE '%REST%' OR
                                   UPPER(Other_options) LIKE '%REST%' OR
                                   UPPER(Supported_Request_Formats) LIKE '%REST%' OR
                                   UPPER(Supported_Response_Formats) LIKE '%REST%' OR
                                   UPPER(Architectural_Style) LIKE '%REST%' OR
                                   UPPER(Type) LIKE '%REST%' OR
                                   UPPER(Hypermedia_API) LIKE '%YES%'
);

UPDATE tmp SET isHypermedia = 1 WHERE url IN (
        SELECT url FROM apis WHERE UPPER(Hypermedia_API) LIKE '%YES%'
);

UPDATE tmp SET isSOAP = 1 WHERE url IN (
        SELECT url FROM apis WHERE UPPER(Protocol_Formats) LIKE '%SOAP%' OR
                                   UPPER(Other_options) LIKE '%SOAP%' OR
                                   UPPER(Supported_Request_Formats) LIKE '%SOAP%' OR
                                   UPPER(Supported_Response_Formats) LIKE '%SOAP%' OR
                                   UPPER(Architectural_Style) LIKE '%SOAP%' OR
                                   UPPER(Type) LIKE '%SOAP%'
);

UPDATE tmp SET isWebApi = 1 WHERE url IN (
    SELECT tmp.url FROM tmp, apis WHERE tmp.url=apis.url AND (
                                  isREST = 1 OR  (
                                      isSOAP = 0 AND (
                                          apis.Protocol_Formats NOT NULL OR
                                          apis.Supported_Request_Formats NOT NULL OR
                                          apis.Supported_Response_Formats NOT NULL
                                      )
                                  )
                               )
);

UPDATE tmp SET noData = 1 WHERE url IN (
        SELECT url FROM apis WHERE UPPER(Protocol_Formats) IS NULL AND
                                   UPPER(Other_options) IS NULL AND
                                   UPPER(Supported_Request_Formats) IS NULL AND
                                   UPPER(Supported_Response_Formats) IS NULL AND
                                   UPPER(Architectural_Style) IS NULL AND
                                   UPPER(Type) IS NULL
);
```

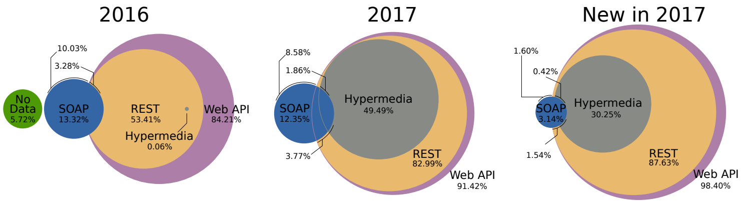
Listing 13 – SQL code used in analysis of ProgrammableWeb data.

method is an adequate approximation. In addition, the term “Hypermedia API” as a synonym for HATEOAS-compliant service is not as prone to misunderstanding[4] as REST.

---

[4] <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

Figure 36 – Venn diagrams with types of services identified by keywords in Programmable Web in 2016 and 2017



Source: the author.

Figure 36 shows the proportions of each type of service in 12 August 2016, 1st September 2017, and among the new services inserted in this interval. In 2016 SOAP services amounted to 13.32% of services, a quantity that decreased to 12.35% in 2017. This decrease can also be observed among the new services added, of which, only 3.14% were SOAP services, nearly half of them with dual REST/SOAP implementations.

Expressive changes can be observed between 2016 and 2017 in Figure 36 with respect to the number of hypermedia APIs and REST services. Such change should be interpreted not as 8,143 new HATEOAS conforming services in a year, but as an update on ProgrammableWeb directory with an improvement on data quality. Other evidences of such improvement is the elimination of the "No Data" category and the reduction on the number of Web APIs not classified as REST services. In addition, 5 services could not be classified by the queries in Listing 13 in the 2016 database (and were excluded from the Venn diagram for clarity) were classified in the 2017 database.

The diagram of Figure 36 showing only services added in between the two runs of the scrapper is not affected by the database update. It shows significant declared adoption of the HATEOAS constraint, at the same time that new SOAP services continued to be added to ProgrammableWeb.

APPENDIX B – SLR PROTOCOL

Kitchenham & Charters (2007, p. 13) proposes 10 steps for an SLR protocol. We detail the first 8 phases, as dissemination and project timetable are of no interest to this dissertation. The first review on the topic of heterogeneous service composition was not systematic and was continuously updated during 2016. An abridged version of this version appears on the related work section in (HUF; SALVADORI; SIQUEIRA, 2017, p. 991). The first SLR was conducted in March 2017 and included in an unpublished previous version of this document. Bias in the search strategy of this second review prompted the development of the current protocol.

B.1 RESEARCH QUESTION

This SLR has only a single RQ: “What are the approaches for designing or executing composite services with heterogeneous interaction models, specifically those implied by EOS, SOAP services and RESTful services.”.

B.2 SEARCH STRATEGY

Search is conducted on two databases of scientific publications:

1. Scopus[1]
2. IEEE Xplore[2]

Another source commonly used is ACM Digital Library[3], which is not considered due to technical details. Although it contains abstract and keyword data, accessible from the results Web page, such data is not included when downloading the search results in machine-readable formats such as BibTeX or CSV. Such data is necessary, since part of the search process is performed locally by a script.

Kitchenham & Charters (2007) suggests structuring an RQ in a scheme such as Population, Intervention, Comparison, Outcome, Context (PICOC), proposed by (PETTICREW; ROBERTS, 2008). Once the RQ is structured, synonyms can be identified, and a search string constructed connecting synonyms with `OR` and PICOC criteria with `AND`. This method is not appropriate to the present RQ for two reasons. First, only intervention and a population are present in the RQ.

---

[1] <http://scopus.com/>

[2] <http://ieeexplore.ieee.org/>

[3] <http://dl.acm.org/>

Therefore, filtering by other aspects would introduce bias. Second, primary works may list terms related to REST, SOAP and event in association with composition, without mentioning heterogeneous or interaction model. In this case, just connecting all keywords related to population with `OR` would cause an excessive amount of irrelevant matches.

Two search strings are built as part of the search strategy. To build such strings, similarly to the method in (KITCHENHAM; CHARTERS, 2007), groups of related keywords (synonyms w.r.t the search strategy) are identified in Table 14. The first group, intervention, corresponds to its homonyms in PICOC. Service composition is the intervention of interest, but both more specific (orchestration) and more generic (integration) keywords were included. The second group contains terms which could have meaning and usage similar to those of interaction model, as defined in p. 77.

SOAP services are identified in Table 14 using several related technologies, in addition to SOAP. The term “web service” could not be considered as a SOAP-related keywords as its use in the literature does not imply a SOAP service (cf. Subsubsection 2.3.2.1). Including “web service” as a SOAP-related keyword increases the number of results by a factor of 3.102 in Scopus alone. Piloting the protocol, it was found that usually the decision to include or exclude these additional works could only be accurate with access to the full text, yet no such work was found to be relevant to the RQ.

RESTful services and Web APIs are identified in a single group. Although the RQ mentions only RESTful services, from authors’ experience, literature (MALESHKOVA; PEDRINACI; DOMINGUE, 2010; RENZEL; SCHLEBUSCH; KLAMMA, 2012) and previous version of this review, it is known that most approaches claiming to support RESTful services support Web APIs, but not all REST architectural style constraints. Therefore, the search strategy also includes Web APIs. Whether REST is properly supported is a concern in the data extraction phase.

Since Scopus has the most expressive query language, query strings are presented with its syntax. For readability, the syntax is extended with bash-style variable substitutions (`${groupname}`) that refer to keyword groups defined in Table 14 (with `OR` inserted between the keywords). Two queries were built. The first, shown in Listing 14a aims to find works that propose techniques for composition of two or more of the three service types. The second query (Listing 14b) searches for the same intervention, but uses heterogeneity of interaction model to define the population. Common to both queries is the restriction

Table 14 – Groups of alternative keywords

Group	Keywords
intervention	synthesis synthesized synthesizing composition composing composes composite choreography choreographing choreographed choreographes orchestration orchestrating orchestrated orchestrates mashup interoperation interoperating interoperated interoperates interoperability integration integrating integrated integrates
interaction model	"communication protocol" "communication paradigm" "interaction paradigm" "interaction model" "communication protocols" "communication paradigms" "interaction paradigms" "interaction models"
SOAP	SOAP WSDL "WS-" BPEL UDDI SAWSDL WSMO "OWL-S"
REST	REST RESTful WebAPI "Web API" "Web-API" WebAPIs "Web-APIs" "Web APIs"
EOS	events events eventful eventing "publish-subscribe" "publish/subscribe" notification notifications

Source: the author.

Listing 14 – Templates for queries on Scopus.

(a) Query combining service types.

```
TITLE-ABS-KEY(
  service AND (${intervention})
  AND (
       ( (${SOAP}) AND (${REST})  )
    OR ( (${SOAP}) AND (${event}) )
    OR ( (${REST}) AND (${event}) )
  )
) AND DOCTYPE(cp or ar)
```

(b) Query for explicit mentions of heterogeneity.

```
TITLE-ABS-KEY(
  service AND (${intervention})
  AND (heterogeneous OR heterogeneity)
  AND (${interaction model})

) AND DOCTYPE(cp or ar)
```

Source: the author.

to conference or journal papers and the term “service”. The latter is needed to remove spurious matches caused by terms like “rest”, “event” and “integration”, which frequently occur in English, without any relation to services.

IEEE Xplore has no equivalent to `TITLE-ABS-KEY`, therefore the query is run against all metadata fields. Furthermore, it limits the number of search terms to 15, a count that is already smaller than the number of keywords for intervention. As a consequence of this limitation, only "service", and no intervention keywords, is used for translations of both queries in Listing 14. The query in Listing 14a is further divided into three sub-queries, each containing only a single combination of two service types (i.e., SOAP+REST, SOAP+EOS, REST+EOS).

B.3 SELECTION CRITERIA

A single inclusion criteria and 4 exclusion criteria are defined. Failing to satisfy the inclusion criteria also implies exclusion. Selection criteria is applied initially to title and abstract. Only works included by abstract and title have the criteria evaluated against the full text. The inclusion criteria is derived from the RQ:

1. Proposes a method for composition, i.e., design or execution (Section 2.4) of heterogeneous services contemplating at least two of the following service types: SOAP services, RESTful services or EOSs.

Exclusion Criteria:

1. Intervention (composition, integration, etc.) only on things other than service (e.g., device or data);
2. The following three conditions hold:
   I. The intervention method is implemented with, employs services of a specific type or employs concepts of this this service type;
   II. There is no evidence that the method aims to compose services of the aforementioned type is supported as a type of service to be composed;
   III. Excluding the aforementioned service type, no other two service types for which condition II does not hold remain.
3. The following three conditions hold:
   a) The intervention method is an extension, implementation or is influenced by BPEL or BPMN;
   b) Events or notifications are handled or supported by the method;

    c) There is no indication that events originate from outside components executing the business process.

4. The study has no proposal more relevant than event delivery through SOAP or RESTful services;
5. The study proposes a composition approach for one type of service alone and compares the approach or composite service to counterparts using other service types;
6. The study states to propose a intervention method but does not give any information about how heterogeneous services are supported;
7. The study proposes an architecture or describes a system that employs different service types that assume different roles, but the design of the proposal is such that the it does not handle heterogeneity of service types since heterogeneity is not an issue in the architecture or system;
8. The study is a preliminary publication of this dissertation: Huf, Salvadori & Siqueira (2017);
9. The study has an extended version which presents the same intervention method with more detail;
10. The study is not in English;
11. The study was not published in a conference or journal;
12. The study is a survey;
13. The study was not subject to scientific peer-review, e.g., monographs, technical reports, standards, editorials, invited papers and keynotes;
14. The study full-text could not be obtained using all means available, including its request by email.

B.4 SELECTION PROCEDURES

A single researcher, Alexis is responsible for study selection. A completely manual selection from the results of queries submitted to IEEE Xplore and Scopus is highly susceptible to human error. Therefore, a software was developed to support the selection process, named slrpk[4]. This tool manipulates references as sets with built-in deduplication and record-merging. Its main usage is to evaluate expressions in a custom algebra based on set theory. The automated phase of the selection process realized by a GNU Makefile[5]. Studies not removed

Table 15 – Additional regular expressions

Name	Expression
EOS	`event[\Wisf]|publish\Wsubscribe|notification`
service	`(service|(REST(ful)?|web|service)\W*API)`
heterogeneous	`heterogene(ous|ity|ities)`
proceedings	`survey|systematicliteraturereview|review|slr`
survey	`proceedings|workshop|conference`

Source: the author.

during this phase later undergo a manual selection by their abstracts.

For brevity in exposition, let ⟨`name`⟩ denote the case-insensitive regular expression in Table 15 or a expression that matches terms in Table 14, in both cases, enclosed by parenthesis. The automated part of the selection process has three chained steps. The first stage is a countermeasure to the limited queries submitted to IEEE Xplore. The second stage checks co-occurrence of keywords within the same sentence:

1. For all works originating from IEEE:
   - Require ⟨`intervention`⟩ as substring in Title or Abstract or Keywords (TAK);
   - Require ⟨`REST`⟩ as substring in TAK if the work originated only from a query by REST services.
2. Let ⟨`space`⟩ be `.*` if evaluated in title, `[^.]*` if evaluated in the abstract or `[^,.;]*` if evaluated in keywords. For all works:
   - Require either ⟨`service`⟩⟨`space`⟩⟨`intervention`⟩ or ⟨`intervention`⟩⟨`space`⟩⟨`service`⟩ to match substrings of TAK;
   - Let ⟨`space`⟩ be `.*` if evaluated in title, retaining the definitions above for other fields. If the work matched only a query for heterogeneous interaction model, require either ⟨`heterogeneous`⟩⟨`space`⟩⟨`interactionmodel`⟩ or

---

[5] <https://github.com/alexishuf/slrpk>

[5] <https://bitbucket.org/alexishuf/dissertacao/src/extras/revisao/v4/>

`⟨interactionmodel⟩⟨space⟩⟨heterogeneous⟩` to match substrings of TAK.

The third stage evaluates some exclusion criteria:

BibTex type For all works, remove those with the following BibTeX entry types: `@manual`, `@book`, `@booklet`, `@techreport`, `@thesis`, `@phdthesis`;

Document type For all works, remove any for which a substring of the title matches `⟨survey⟩|⟨proceedings⟩`

Not peer-reviewed For all works, remove any for which a substring of its title matches `tutorial|keynote|editorial`;

Bogus event For all works originating from the service combinations queries (Listing 14a), remove those that match `event[^isf ,.;\])\-]` in TAK, but do not match `⟨EOS⟩` nor `(⟨SOAP⟩.*⟨REST⟩)|(⟨REST⟩.*⟨SOAP⟩)` in TAK;

Service types : For all works originating from service combinations queries (Listing 14a), remove all for which only one among `⟨REST⟩`, `⟨SOAP⟩` and `⟨EOS⟩` match as a substring of the abstract.

The manual selection is performed only on studies not excluded during he automated phase. In the case of a false positives, i.e., the work should not have been selected, the error is detected during data extraction. Wrongly selected works are then excluded and this event is registered as an exclusion motivated by the full-text.

The expected result of the selection phase is exclusion of most studies. Double-checking all exclusions by abstract is not viable due to limited resources. However, accuracy of manual exclusion decisions by abstract can be estimated from random samples of excluded studies. To avoid bias by the human reviewer's memory, a interval of one month between selection by abstract and accuracy evaluation is adopted. Accuracy evaluation has the following steps

1. Obtain a sample with 25% of studies not automatically excluded;
2. Evaluate, based only on their abstracts (other fields are hidden, to avoid confirmation bias), whether sampled studies should be included or excluded;
3. Assuming the selection performed on this sample to be correct, identify *candidate false exclusions*, i.e., works that were excluded during the SLR but were not in the sample;
4. For every candidate false exclusion, obtain the study full-text and determine if it should have been excluded or included;
5. Obtain the *definitive false exclusion* count.

B.5 QUALITY ASSESSMENT

Given the goal of this SLR, there is no justification to exclude studies based on their quality or to discard extracted data from low-quality studies. Nevertheless, there are risks to the validity of the results of this SLR stemming from the quality of selected studies. There are two main issues: poor exposition of research and conclusions without theoretical or empirical support. The former is tackled allowing data extraction questions to be answered with “unclear”. The latter are handled by employing the checklist from Table 16, whose component questions are subsequently discussed in detail.

One quality issue is poor evaluation by studies of their proposed heterogeneous composition methods. A classification of such methods, presented by Hevner et al. (2004, p. 85–87), is used to collect the evaluation methods employed by studies to demonstrate that their proposal effectively supports heterogeneity in composition. During data synthesis, this allows evaluation of whether adequate evaluation methods have been used for the identified categories of heterogeneous composition methods.

Methods such as field study, static analysis and architecture analysis (HEVNER et al., 2004) are the most adequate for evaluating heterogeneity support in service composition. Yet, in piloting the SLR, many works employed such methods unsatisfactorily. This can happen if the treatment is highly informal, if evidences are not properly presented or if important aspects, such as REST constraints, are not evaluated. Guidelines listed by (KITCHENHAM; CHARTERS, 2007) are mostly concerned with reviews that aggregate quantitative data. Even when concerned with qualitative data, guidelines for clinical and social research (DALY et al., 2007) still focus on population sampling. Therefore, a 3-level subjective scale was adopted as displayed in Table 16

Another expected quality issue is the violation of REST architectural style constraints (cf. p. 61). These violations can compromise the reliability of the whole data synthesis process. Therefore, a REST maturity model must be applied on the evidence provided by the study to determine precisely what is the level of support for the REST architectural style. The RMM (RICHARDSON, 2008; WEBBER; PARASTATIDIS; ROBINSON, 2010) was found to be the most adequate maturity model during a pilot phase of this SLR.

Table 16 – Quality assessment checklist.

What is the...	Possible answers	Use
evaluation method?	(HEVNER et al., 2004, Table 2)	Evaluate method adequacy, after synthesis
evaluation quality?	Insufficient; Mostly credible, but with minor issues; Credible.	Evaluate quality, after synthesis
level of conformance to REST?	RMM (RICHARDSON, 2008; WEBBER; PARASTATIDIS; ROBINSON, 2010)	Replace claims of REST support during data synthesis

Source: the author.

B.6 DATA EXTRACTION

The data extraction process, like selection is performed by a single reviewer. To minimize possibility of human error the process is divided in two phases. In the first phase, all questions of the data extraction form, except for evaluation method and quality, are answered. This phase also serves a double-check of inclusion decisions (Section B.4). Since selection criteria and data extraction procedures are tightly coupled, mistakenly included studies will be detected during this phase. In the second phase, the researcher answers the remaining questions and reviews if the already answered questions are correct. Any possible error during data extraction that remains after the second phase should be detected and fixed during data synthesis, which confirms extracted data against the full-text.

Data extraction is conducted on a spreadsheet with basis on the full text of selected studies. All questions can be answered with “unclear”. The abbreviation NA stands for “does Not Apply” in answers. For the first phase, the questions in and possible answers are the following:

1. What is the explicit or implicit meaning of “composition” in most of its occurrences? (artifact/action/NA);
2. There were occurrences of composition using both meanings? (Yes/No)
3. When the term composition is used to mean an action, execution is considered part of composition? (Yes/No/NA);

4. REST services are considered a Web Services? (Yes/No/NA);
5. What service types are supported? (REST,SOAP,EOS)
6. The approach described for heterogeneous composition includes or requires:
    a) ... automatic composite service design (cf. Subsection 2.4.3? (Yes/No)
    b) ... dynamic functional design of composite services? (Yes/No)
    c) ... the workflow (planned sequence of service invocations, including control structures) to be dynamic? A Yes answer to Question 8 implies this to be true. Even without dynamic composite service design, the composite service specification can be loose and allow different workflows to be planned or enacted during execution (MURGUZUR et al., 2014). (Yes/No)
    d) ... dynamic service selection? (Yes/No)
7. With respect to REST support:
    a) What is the level of REST support according to the RMM maturity model? For this evaluation, only evidences present in the paper itself should be considered[6]. Possible answers are NA, POX, Resources, Protocol, Hypermedia, unclear and *level*-unclear. The latter is used when a study presents evidence, that despite inconclusive, hints that *level* is supported.
8. The study proposes a:
    a) Language or method for service description? (Yes/No)
    b) Composite service specification language? (Yes/No)
    c) Method or approach for Composite service design? (Yes/No)

For each study, a summary was written, describing:

1. The methods proposed by the study in order to support composition of heterogeneous services;
2. If the study claims to support RESTful services but does not achieve the Hypermedia RMM level, what is the reason for the failure;
3. Any clarification or justification deemed necessary by the reviewer concerning any data extraction question;
4. Anything deemed important or worthy of note by the reviewer.

---

[6] Obtaining source code and artifacts of all selected studies would not be possible with available resources.

The second phase questions follow the same extraction method. These questions cover the evaluation methods employed to validate proposals made by the selected studies. The questions are:

1. What are the evaluation methods used to demonstrate correctness of heterogeneity support? Possible values are those in the classification provided by Hevner et al. (2004, Table 2);;
2. What is the quality of evaluation cf. Table 16? (Insufficient/Mostly credible/Credible);
3. What are the methods used to evaluate performance of proposed approaches? Like Question 1, this must use the classification proposed in (HEVNER et al., 2004);

Similarly to the first phase, a summary is written for each study. Such summary covers the evaluations presented by the study concerning heterogeneity support and performance. The answer to Question 2 must be justified in this summary. Justification for why an evaluation was classified under a given category (cf. Hevner et al. (2004)) is included in this summary when the reviewer deems necessary.

B.7 DATA SYNTHESIS

The adopted method of data synthesis is descriptive (KITCHENHAM; CHARTERS, 2007). In line with the RQ, the main output of synthesis is a categorization of techniques for support heterogeneity in service composition. Each study may present one or more solution proposals to this end, which may belong to one or more categories. Categories are attributed using the summary written for each study during data extraction, with access to studies full text to clear doubts. The process of defining the categories is as follows.

1. Let $C = \emptyset$.
2. Evaluate the next study, choose a category c from C that is adequate to describe its proposal;
    a) If there is no suitable c, let c be a new category;
    b) Else, if the study presents a distinct characteristic that could be present in other studies (e.g., use of ontologies), set c to a new subcategory of c;
    c) Else, if there are several previously evaluated studies that share some characteristic, set c to a new subcategory of c and update those studies to the new c;
    d) Set $C \leftarrow C \cup \{c\}$

3. After all studies have been evaluated:
    a) If a category c has only one subcategory c_s, replace c with c_s in C and on forms;
    b) If a category c has more than one subcategory and there are works classified directly as c, evaluate if those works should not originate a new subcategory.

Another subject of categorization are the reasons why the HATEOAS constraint is not supported in works that claim to support RESTful services despite not supporting HATEOAS. The categorization method is the same.

For each identified category of heterogeneity support method, the highest quality studies, under the subjective evaluation of the reviewer should be discussed. Additional studies should also have aspects discussed until a complete description and summary of that category is achieved. Piloting revealed a large quantity of studies, and discussing each particular study could be counterproductive.

The relation between the three service types in Section 2.3 and the identified method categories should be investigated, in order to determine what are the theoretically possible applications of each method and which applications were observed in the selected studies. Another set of relations to be investigated is the one between sources of heterogeneities in the protocol model (Subsection 2.5.1) and the method categories.

Data synthesis driven by information previously extracted from the full-texts of studies (cf. Section B.6). As part of ensuring quality of extracted data, the information read from the spreadsheet is double checked against the full-text. Any mistakes identified are immediately fixed and already performed synthesis is redone, if needed.

APPENDIX C – SLR SUPPLEMENT

This appendix contains additional results from the SLR. These results were predicted by the SLR protocol but are not necessary for understanding the remainder of the dissertation.

C.1 QUALITY OF SELECTION PROCESS

As discussed in the end of Section B.4, exclusion decisions based on abstract are subject to human error. There are two possibilities of error, false exclusion and false inclusion. In the case of false inclusion, the error will be detected during data extraction. In the case of false exclusion, the sampling procedure described in the end of Section B.4 was adopted.

Out of the 386 studies not automatically excluded, 97 were randomly selected and re-evaluated based on abstract alone. 24 studies (24.74%) were selected for inclusion. Assuming this selection to be correct, 5 *candidate false exclusions* (5.15%) were identified. These 5 studies had selection criteria evaluated against their full-text. Table 17 shows the result of this re-evaluation, where all candidate false exclusions were themselves found to be false. The rationales, explaning why the studies should not be included, are also shown.

The absence of *definitive false positives* in the sample, as revealed in Table 17 does not imply its absence among the population of 386 studies. The sample results can be generalized to the population through confidence intervals. Instead of the textbook approach that uses a normal approximation to the binomial distribution, the Clopper-Pearson (CLOPPER; PEARSON, 1934) method is used. This method is more conservative in the sense that confidence intervals may not be shortest possible, but they satisfy the 95% confidence level even when the number of success trials nears zero (which is the case for definitive false exclusion). Table 18 shows such confidence intervals, also generalized to the whole population of 386 studies.

Despite the apparently large statistical upper bound on the number of definitive false exclusions, there are some observations that which hint that the actual number of errors is smaller. First, the only clearly wrongly selected abstract is that of (PAUTASSO, 2008). However, it most likely that the error was not in reading the abstract, but not marking the “has extended version” flag during selection. Second, the other four abstracts all contained evidence that matched exclusion criteria. Therefore, the upper bound of 28 candidate false exclusion is likely made up of mostly ambiguous abstracts. Third, Studies that tar-

Table 17 – Full-text selection for false exclusions

Study	**Exclusion Criteria**
Yu et al. (2009)	Describes a system that employs SOAP and event-oriented services for different roles. Heterogeneity, by system design, is not an issue
Bagnasco et al. (2006)	Describes an architecture that combines SOAP and RESTful services for different roles. Heterogeneity, by architecture design, is not an issue
Van Hoecke et al. (2014)	As per Abstract and Introduction, the study targets only EOS implemented as RESTful services. What prompted inclusion was the use of an automatic composition algorithm for RESTful services. The paper contents do not deliver any integration between EOSs and non-EOS. In fact, contents cover only RESTful service composition, where the fact that these services expose sensor data is merely context.
Taher et al. (2013)	Events are employed for monitoring Service Level Agreements (SLAs). Composite services are homogeneous
Pautasso (2008)	Has extended version: (PAUTASSO, 2009)

Source: the author.

get all three service types are less likely to be wrongly excluded, since selection looks for one heterogeneity case and such works contain two. Fourth, studies that target heterogeneity with EOS are more likely to be wrongly excluded.

Table 18 – Confidence intervals for proportions of errors in study selection

Error	**Proportion CI**	**Bounds in number of studies**
Candidate false exclusion	0.251% – 7.25%	$0 < n < 28$
Definitive false exclusion	0% – 3.732%	$0 \leq n < 15$

Source: the author.

C.2 SLR SELECTED STUDIES

Table 19 – Selected studies and main categorical variables. **T** is short for "Service Types", **HAT.** for HATEOAS and, respectively, S,R,E stand for REST, SOAP and EOS.

Study	**T**	**RMM**	**HAT.**	**Categories.**
(ZHANG et al., 2014)	S, R	prot.	ign.	process_language/-extension(BPEL)
(HAUPT et al., 2014)	S, R	res.	static	process_language/-extension(BPEL) middleware
(BERTOLI et al., 2009)	S, E	NA	NA	process_language/-existing(BPEL)
(LALIWALA; CHAUDHARY, 2006)	S, E	NA	NA	event_processor/ECA
(IM; JEONG, 2016)	S, R	res.	ign.	event_processor middleware (for soap-rest-Android) process_language/new
(LIU et al., 2014)	S, R, E	POX	ign.	event_processor middleware
(HANG; ZHAO, 2013)	S, R	prot.	uns.	middleware process_language/new
(SUPAVETCH; CHUNITHIPAISAN, 2011)	S, R	POX	ign.	process_language/new

Table 19 – (continued)

Study	T	RMM	HAT.	Categories
(PAUTASSO, 2009)	S, R	prot.	static	process_language/extension(BPEL) middleware
(GEORGANTAS et al., 2013)	S, E	NA	NA	middleware process_language/new
(THAKAR; TIWARI; VARMA, 2016)	S, R	POX	ign.	process_language/existing(BPEL)
(WU; LIN; CHEN, 2013)	S, R	POX	ign.	process_language/existing(BPEL)
(LOMOTEY; DETERS, 2013)	S, R	unclear	ign./no_eff.	proxy(REST)
(BO et al., 2009)	S, E	NA	NA	middleware(embedded in BPEL engine) process_language/existing(BPEL)
(LIN; LIN, 2007)	S, E	NA	NA	process_language/existing(BPEL) middleware(embedded in BPEL engine)
(BAGLIETTO et al., 2010)	S, E	NA	NA	process_language/existing(BPEL) direct_implementation
(PENG; MA; LEE, 2009)	S, R	POX	ign.	proxy(SOAP)
(MINGUEZ; ZOR; REIMANN, 2011)	S, E	NA	NA	process_language/existing(BPEL)
(CHENG et al., 2017)	S, R, E	prot.-unclear	ign.	event_processor middleware

(MAJDA; AHMED, 2016)	S, R	unclear	ign.	proxy(SOAP)
				proxy(REST)

(TAO et al., 2015)	S, R	prot.	ign.	proxy(SOAP)
(LIECHTI et al., 2015)	R, E	pox	ign.	event_processor/ECA
(LEE; LEE; WANG, 2015b)	S, R	prot.-unclear	ign.	process_language/extension(BPEL) middleware

Table 19 – (continued)

Study	T	RMM	HAT.	Categories
(ROMAN et al., 2015)	S, R	prot.	static	common_description/-semantic
(FOKAEFS; STROULIA, 2013)	S, R	res.-unclear	ign.	common_description/-metamodel
(WU et al., 2012)	S, R	prot.	static-unclear	proxy(SOAP)
(MENG et al., 2012)	S, R	unclear	ign.	middleware
(YOO; PARK; BAE, 2011)	S, R	POX	ign.	common_description/-semantic
(MANNAVA; RAMESH, 2011)	S, R	prot.	ign.	proxy(other)

(GIORGIO; RIPA; ZUCCALà, 2010)	S, R	prot.	ign.	common_description/-semantic
				proxy(SOAP)
				proxy(REST)

(LUO; XU; SUN, 2010)	S, R	POX	ign.	common_description/-syntactic
(JURIC, 2010)	S, E	NA	NA	process_language/-extension(BPEL)
(HE, 2009)	S, R	prot.-unclear	ign.	middleware
(CIPOLLA et al., 2009)	S, E	NA	NA	event_processor/ECA
(SIBIRTSEVA et al., 2008)	S, R	POX	ign.	proxy(SOAP)

(GAEDKE; HäRTZER; HEIL, 2008)	S, R	res.	ign.	proxy(SOAP)
				proxy(REST)
				common_description/-syntactic

(LEE; LEE; LEE, 2007)	S, E	NA	NA	event_processor/ECA

(CHOU; LI; LIU, 2005)	S, E	NA	NA	process_language/-existing(BPEL)
				proxy(SOAP)
				proxy(EOS)

Table 19 – (continued)

Study	T	RMM	HAT.	Categories
(UPADHYAYA; ZOU, 2012)	S, R	prot.	static	common_description/syntactic
(BROMBERG et al., 2011a)	S, R	prot.	ign.	middleware common_description/syntactic
(BRAUBACH; POKAHR, 2013)	S, R	prot.	ign.	proxy(other)
(LI; CHOU, 2009)	S, R, E	prot.	ign.	common_description/syntactic middleware
(DREWNIOK et al., 2009)	S, E	NA	NA	process_language/existing(BPEL) direct_implementation (Web Service, Distributed Orchestrator) middleware (JSLEE/Mobicents)
(PICO-VALENCIA; HOLGADO-TERRIZA, 2016)	S, R	prot.	ign.	middleware
(BLUMBERGS; KRAVCEVS, 2011)	S, E	NA	NA	process_language/existing(BPEL proxy(EOS)
(ARDISSONO et al., 2009)	S, R, E	prot.-unclear	ign.	proxy(EOS) event_processor
(SRBLJIĆ; ŠKVORC; SKROBO, 2010)	S, E	NA	NA	process_language/extension(BPEL); process_language/new
(RAMTOHUL; OGUNLEYE, 2017)	S, R	prot.	static	process_language/extension(BPEL)
(RAJARAM; BABU; GANESAN, 2015)	S, R	unclear	ign.	middleware
(BERGWEILER, 2015)	S, R	prot.-unclear	ign.	process_language/existing(BPMN) middleware

Table 19 – (continued)

Study	**T**	**RMM**	**HAT.**	**Categories**
(DOMINGOS; MARTINS; ĈANDIDO, 2013)	S, E	NA	NA	process_language/-extension(BPEL) event_processor/ECA
(VASKO; DUSTDAR, 2007)	S, R	prot.	ign.	middleware common_description/-syntactic process_language/new
(MAXIMILIEN et al., 2007a)	S, R	prot.-unclear	ign.	process_language/new middleware
(AMNUAYKANJANASIN; NUPAIROJ, 2005)	S, E	NA	NA	process_language/-existing(BPEL)

APPENDIX D – INTERMEDIARY DESCRIPTION MODULES

In addition to the core module of the intermediary descriptions, there are other modules that cover specific aspects that are not common to every service or that only serve to support the Unserved architecture. These modules are:

- `unserved-c`: Describes components stored in the component repository.
  Main classes: `uc:ActionRunner` (runs an workflow action), `uc:Parser` (transforms message data into RDF), `uc:Renderer` (transforms RDF data into specific message payloads);
- `unserved-j`: Describes properties that relate a component to its implementing code.
  Main property: `uj:factoryClassName` (fully qualified name of a factory for the component);
- `unserved-n`: Describes NFPs of services.
  Main classes: `un:NFP` (the superclass of all NFPs), `un:ResponseTime` and `un:Throughput`.
  Main properties: `un:value` (the numeric value of the NFP) and `un:nfp` (associates a reaction message to a NFP);
- `unserved-o`: Describes specific aspects of surrogate outputs added by translators to represent actions.
  Main class: `uo:OperationRepresentation` (a representation for use with surrogate outputs).
  Main property: `uo:operatesOn` (connects the surrogate output to inputs);
- `unserved-p`: Describes activities of the *ad-hoc* workflow language used in the architecture.
  Main classes: `up:Assign` `up:Copy` `up:Send`, `up:Receive`, `up:Sequence`.

In addition to the modules listed above, the unserved-x module is a collection of sub-modules that deal with low-level details. Most of such details pertain to specific protocols, service types and formats.

- `atom.ttl`: Models aspects of interaction with Atom EOSs.
  Main classes: `ua:Atom` (the media type), `ua:Entry` (a representation for Atom entries), `ua:AtomPollRequest` (a continuous poll request for atom feeds) and `ua:AtomPollEntry` a reaction to `ua:AtomPollRequest` with many-valued cardinality);

- `media-type.ttl`: Describes media types.
  Main class: `umt:MediaType`.
  Main property: `umt:mediaTypeValue` (defines the media type instance by its string representation);
- `http.ttl`: Helpers that connect the Unserved core to the HTTP ontology (KOCH; ACKERMANN; VELASCO, 2017).
  Main classes: `uh:HttpMessage`, `uh:AsHttpProperty` (binds variables to specific parts of the HTTP message);
- `rdf.ttl`: Helpers that allow describing messages of services that serve RDF.
  Main classes: `ur:RDF` (a built-in `ur:MediaType` for RDF documents) and `ur:AsProperty` (binds variables to properties of the RDF contained in a message);
- `sawsdl.ttl`: Describes lifting and lowering mappings.
  Main classes: `us:SchemaMapping` (describes one such mapping) , `us:LiftingPartModifier` (describes how to transform XML into RDF that will be bound to a variable), `us:LoweringPartModifier` (describes how to transform RDF value of a variable into XML that can be sent to SOAP services);
- `wsdl20.ttl`: Describes request and response messages of SOAP services, bound to WSDL 2.0 documents.
  Main classes: `uw:WsdlRequest`, `uw:WsdlResponse`.
  Main properties: `uw:wsdlOperation`, `uw:wsdlPort`, `uw:wsdlBinding`;
- `wsdl11.ttl`: Same as `wsdl20.ttl`, but uses WSDL 1.1.

APPENDIX E – SCENARIO SUPPLEMENT

This appendix presents fragments of the original service descriptions and their translated intermediary descriptions. Here only lower-level technical details are discussed. These details were deemed as too specific for discussion in Chapter 4 or Chapter 5, and were not included to prevent cluttering. Nevertheless, these details have play roles in enabling the base scenario in Section 5.2.

All listings in this appendix are incomplete. Only parts more relevant to the scenario are shown. The indentation was manipulated to ensure listings fit in a single page.

The Presence feed, whose descriptions are shown in Listing 15, is implemented as an Atom service. Atom service documents (GREGORIO; HORA, 2007) allow specifying which categories of entries can be in a feed, and optionally allows to link categories with terms in external vocabularies. No vocabulary format is dictated and OWL not cited. Nevertheless, the translator builds IRIs for these terms (gluing `atom:schema` to `atom:term`, if necessary) and treats the term as an OWL class. In line with Section 1.4, all atom entries have RDF as their `atom:content`. Nottingham & Sayre (2005) enforces no relation between categories and content, but the translator assumes this.

The Atom publishing protocol in its entirety is treated as a lower-level protocol (Subsection 2.5.1). The results is an initial request message and a reaction message that contains a single entry. The receiver component internally polls the feed, parses it and emits a reaction message for the content of each entry. The `u:variable` that receives this content is marked with the `ua:theContent` part modifier.

The lighting control SOAP services is described in WSDL 2.0 (WEERAWARANA et al., 2007) with SAWSDL annotations (KOPECKÝ et al., 2007). Its only described operation, `TurnOn`, receives an XML representation of a `saref:BuildingSpace`. The mapping between the XML and RDF is done with lifting/lowering schema mappings linked by SAWSDL annotations to the `tRoom` type. The *modelReference* annotation from SAWSDL is applied to the `tRoom` type and to the `<operaetion>`. In the first case it determines the `u:type` of the operation I/Os; in the second, it motivates the generation of a surrogate output representing the operation itself.

The air conditioner control RESTful service is described using the Hydra (LANTHALER; GÜTL, 2013) ontology. An URI template (GREGORIO et al., 2012) defines how to obtain an air conditioner resource. The only variable in the template is mapped to `ex:roomName`, but Hydra provides no mean to specify the relation between this value

Listing 15 – Description of presence sensors.

(a) Atom Service description.

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://www.w3.org/2007/app"
         xmlns:atom="http://www.w3.org/2005/Atom"
         xmlns:exs="http://example.org/">
  <workspace>
    <atom:title>Presence Events</atom:title>
    <collection href="http://eos.example.org/presence">
      <accept>application/atom+xml;type=entry</accept>
      <categories fixed="yes">
        <category
          scheme="http://example.org/events"
          term="http://example.org/ns#Entry"/>
      </categories>
    </collection>
  </workspace>
</service>
```

(b) Intermediary description.

```
_:GETPresenceEvents1 a u:Message, ua:AtomPollRequest;
  http:mthd httm:GET;
  http:absoluteURI "http://eos.example.org/presence"^^xsd:anyURI;
  u:when u:spontaneous.

_:GETPresenceEvents1Reply a u:Message, ua:AtomPollEntry;
  u:when [ u:reactionTo _:req; u:reactionCardinality u:many ];
  u:part [ u:partModifier ua:theContent;
    u:variable _:entry ].

_:entry u:type ex:Entry;
  u:representation ur:RDF;
  u:part [ u:variable _:room ; u:partModifier [ ur:property ex:room; ur:propertyOf _:entry ] ].
_:room  u:type saref:BuildingSpace; u:representation ur:RDF;
  u:part [ ur:propertyOf _:room; ur:property ex:roomName ].
```

Source: the author.

and the resource identified by the resulting URI. In the corresponding intermediary description message, an `u:Variable` instance representing the room name is mapped to `http:absoluteURI` using the URI template as a modifier. For other cases, the translation of the Hydra description occurs as described in Chapter 4.

Listing 18 shows the specification of the composition problem given to the Unserved architecture. Two input variables are given: a goal temperature of 23°C as a `saref:Temperature` and the description of a room as a instance of `saref:BuildingSpace`. Three outputs are expected: the `ex:Entry` event, the surrogate `saref:OnCommand` output of the lighting service and the surrogate `http:POST` output of the air conditioner control service. The surrogate outputs are used in a SPIN post-condition stating that the `saref:OnCommand` was applied to the room, and that the `http:POST` action was applied to the temperature of the air conditioner.

Listing 16 – Description of lights control service.

(a) WSDL of lights service.

```
<wsdl:description
    targetNamespace="http://soap.example.org/service.wsdl"
    xmlns:tns="http://soap.example.org/svc.wsdl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ts="http://soap.example.org/types.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/wsdl/wsdl20.xsd" >
  <wsdl:types>
    <xs:schema
        targetNamespace="http://soap.example.org/types.xsd"
        xmlns="http://soap.example.org/types.xsd"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:complexType name="tRoom"
                      sawsdl:modelReference="http://uri.etsi.org/m2m/saref#BuildingSpace"
                      sawsdl:liftingSchemaMapping="http://soap.example.org/mappings/tRoomLifting.xlst"
                      sawsdl:loweringSchemaMapping="http://soap.example.org/mappings/tRoomLowering.xlst
        <xs:attribute name="name" type="xs:string" />
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:interface name="LightControl">
    <wsdl:operation name="tns:TurnOn"
                    pattern="http://www.w3.org/ns/wsdl/in-out"
                    sawsdl:modelReference="http://uri.etsi.org/m2m/saref#OnCommand">
      <wsdl:input element="ts:tRoom">
      <wsdl:output element="ts:tRoom">
    </wsdl:operation>
  </wsdl:interface>
  <!--- <wsdl:service> omitted --->
</wsdl:description>
```

(b) Intermediary description.

```
_:TurnOnRequest a u:Message, uw2:WsdlRequest;
  uw2:wsdlOperation "http://soap.example.org/svc.wsdl#TurnOn"^^xsd:anyURI;
  uw2:wsdlInterface "http://soap.example.org/svc.wsdl#LightControl"^^xsd:anyURI;
  uw2:wsdlEndpoint  "http://soap.example.org/svc.wsdl#Ep"^^xsd:anyURI;
  u:when u:spontaneous;
  u:part [ u:variable _:room;
    u:partModifier [ us:lowering _:theLoweringForRoom ] ].
_:room u:type saref:BuildingSpace; u:representation ur:RDF;
  u:part [ u:variable [ u:representation xsd:string ];
    u:partModifier [ ur:propertyOf _:room;
      ur:property ex:roomName ] ].

_:TurnOnResponse a u:Message, uw2:WsdlReply;
  u:when [ u:reactionTo _:TurnOnRequest ; u:cardinality u:one ];
  u:part [ u:variable _:onCommand;
    u:partModifier uo:theOperation ];
  u:part [ u:partModifier [ us:lifting _:theLiftingForRoom ];
    u:variable [ u:type ex:Room; u:representation ur:RDF ] ];
  u:condition [ sp:subject _:onCommand;
    sp:predicate uo:operatesOn;
    sp:object _:room ].
_:onCommand u:type saref:OnCommand ; u:representation uo:OperationRepresentation.
```

Source: the author.

Listing 17 – Description of Air Conditioner control RESTful service.

(a) .

```
[ { "@type": "hydra:ApiDocumentation",
    "hydra:supportedClass": [
      { "@id": "ex:AirConditioner", "@type": "hydra:Class",
        "rdfs:subClassOf": ["saref:Device", "saref:HVAC"],
        "hydra:supportedOperation": [
          { "hydra:method": "GET",
            "hydra:returns": "ex:AirConditioner" } ]
        "hydra:supportedProperty": [
          { "@id": "ex:on", "@type": "hydra:Link",
            "rdfs:range": "saref:OnOffState"
            "hydra:supportedOperation": [
              { "hydra:method": "POST",
                "hydra:returns": "ex:AirConditioner" } ] }
          { "@id": "ex:temperature", "@type": "hydra:Link",
            "rdfs:range": "saref:Temperature",
            "hydra:supportedOperation": [
              {"hydra:method": "POST",
               "hydra:expects": "saref:Temperature",
               "hydra:returns": "ex:AirConditioner" } ] }
        ] } ] },
  { "@type": "hydra:IriTemplate",
    "hydra:template": "http://rest.example.org/{room}/ac/",
    "rdfs:range": "ex:AirConditioner"
    "hydra:variableRepresentation": "hydra:BasicRepresentation",
    "hydra:mapping": [
      {"hydra:variable": "room",
       "hydra:property": "ex:roomName"} ] } ]
```

(b) Intermediary description.

```
_:AirConditionerTplRequest a u:Message, http:Request;
  http:mthd hm:GET;
  http:absoluteURI "http://rest.example.org/{room}/ac";
  u:when u:spontaneous;
  u:part [ u:variable _:roomName;
    u:partModifier [ uh:httpProperty http:absoluteURI;
      uh:httpResource _:AirConditionerTplRequest;
      uh:modifier [ uh:uriTemplateVariable "room";
        uh:uriTemplate "http://rest.example.org/{room}/ac" ] ] ].
_:AirConditionerTplResponse a u:Message, http:Response;
  u:when [ u:reactionTo _:AirConditionerTplRequest; u:reactionCardinality u:one ];
  u:part [ u:variable _:AirConditioner1;
    u:partModifier [ uh:httpProperty http:body;
     uh:httpResource _:AirConditionerTplResponse ];
  ].
_:AirConditioner1 u:type ex:AirConditioner; u:representation ur:RDF;
  u:part [ u:variable [ u:type ex:on; u:representation xsd:anyURI ];
    u:partModifier [ ur:propertyOf _:AirConditioner1; ur:property ex:on ] ];
  u:part [ u:variable [ u:type ex:temperature; u:representation xsd:anyURI ];
    u:partModifier [ ur:propertyOf _:AirConditioner1; ur:property ex:temperature ] ].
# _:AirConditioner2 same as above

_:POSTTemperatureRequest a u:Message, http:Request;
  http:mthd hm:POST;
  u:part [ u:variable [ u:type ex:temperature; u:representation xsd:anyURI ];
    u:partModifier [ uh:httpResource _:POSTTemperatureRequest; uh:httpProperty http:absoluteURI ] ];
  u:when u:spontaneous .
_:POSTTemperatureResponse a u:Message, http:Response;
  u:when [ u:reactionTo _:POSTTemperatureRequest; u:reactionCardinality u:one ] ;
  u:part [ u:variable _:AirConditioner2 ;
    u:partModifier [ uh:httpProperty http:body; uh:httpResource _:POSTTemperatureResponse ] ];
  u:part [ u:variable [ u:type ]].
```

Source: the author.

Listing 18 – Input for the composition problem.

```
_:goalTemp a u:Variable; u:type saref:Temperature;
  u:representation ur:RDF; u:valueRepresentation ur:RDF;
  u:hasResourceValue [ a saref:Temperature;
    saref:isMeasuredIn om:degree_Celsius;
    saref:hasValue "23"^^xsd:float ].

_:room a u:Variable; u:type saref:BuildingSpace;
  u:representation ur:RDF; u:valueRepresentation ur:RDF;
  u:hasResourceValue [ a saref:BuildingSpace;
    ex:roomName "living room" ].

_:wantedOnCommand a u:WantedVariable; u:type saref:OnCommand; u:representation ur:RDF.

_:entry a u:WantedVariable; u:type ex:Entry.

_:wantedPOST a u:WantedVariable; u:type http:POST; u:representation ur:RDF.

[ a u:Wanted;
  u:condition (
    [ sp:subject [sp:varName "ac"];
      sp:predicate ex:temperature;
      sp:object _:temp ]
    [ sp:subject _:wantedPOST;
      sp:predicate uo:operatesOn;
      sp:object _:temp; ]
    [ sp:subject _:wantedOnCommand;
      sp:predicate uo:operatesOn;
      sp:object _:room ]
  )
].

_:temp sp:varName "temp".
```

Source: the author.

GLOSSARY

application protocol The sequence of operations allowed within a particular application. For exemple, the add to cart, set shipping address, pay with credit cardin a e-commerce application.. 39, 41, 58, 81, 90, 92, 131

architectural style Describes a set of software architectures as a set of constraints on architecture elements and their relations. Style and architecture differ mainly in their purposes, being similar in construction (PERRY; WOLF, 1992). 39, 42, 56, 58, 59, 77, 245, 246

basic graph pattern (BGP) A list of triple patterns that must all match for the *basic graph pattern* to match (SEABORNE; HARRIS, 2013).. 25, 116

cache [**architectural style**] Architectural style characterized by the presence of a cache which stores the response of a request in order to quickly respond that request in the future without the need of sending the request accross the network (FIELDING, 2000).. 39, 58, 78

Client-Server [**paradigm,architectural style**] Architectural style with two processing elements: client and server. The server passively awaits for comunication intiated by the client and then performs some processing (e.g., printing) or action as requested by the client. This was initially described as a paradigm (SINHA, 1992; ANDREWS, 1991). However, the processing elements and constraints are clearly visible, which has led Fielding (2000) to consider this an architectural style.. 40, 41, 58, 79, 80, 130, 245

Client-Stateless-Server [**architectural style**] Architectural style derived from the Client-Server style with the additional constraint that the server must not mantain session state accross messages (FIELDING, 2000). 39, 58, 78, 82, 130

Code on Demand [**architectural style**] An architectural style where executable code is tranfered between components in different locations and dinamically transformed in a software component which can interact with other components in the same location (or host) with negligible communication cost (FUGGETTA; PICCO; VIGNA, 1998, p.352).. 39, 58

first-order logic A logic is said to be of first order when it includes relations, quantifiers and does not allow application of quantifiers to objects outside the domain of discourse. $\forall x(Man(x) \implies$

$\exists y(Loves(y,x)))$ is a sentence in classical first-order logic, while $\forall x,y(\forall X((X(x) \iff X(y)) \implies x=y))$ is a sentence in second-order logic, due universal quantification of the X relation. (HODGES, 1983). 48

gateway A gateway is similar in functionality to a proxy. However, the client is not aware of its existence (as a gateway), since the gateway presents itself to the client as a normal server (FIELDING, 2000, 46,97). Due to this nature, gateways are also known as *reverse proxies*.. 58, 77

hyperlink An hypermedia control that simply relates one resource to another.. 45, 60, 73, 110

hypermedia Hypertext, possibly including other medias. Hypertext was coined by Nelson (1965) as text not constrained to be linear, and containing links. Currently, both terms are sometimes used interchangeably (BERNERS-LEE; FISCHETTI, 2000). 60, 62, 73, 81, 246

hypermedia control An element contained within representations (e.g., a link or a form in the case of HTML) through which the client of an hypermedia system determines possible interactions (FIELDING; TAYLOR, 2002). 17, 39, 41, 42, 58, 60, 62, 81, 90, 91, 93, 94, 95, 96, 102, 108, 112, 122, 128, 131, 139, 141

interaction paradigm An examplary pattern of interaction between software components. Examples of such paradigms can be found in (EUGSTER et al., 2003; ANDREWS, 1991; FUGGETTA; PICCO; VIGNA, 1998). 41, 62, 63

Layered System [architectural style] An architectural style where processing elements are fragmented into layers. A layer can only communicate with the adjacent upper layer, to whom it provides functionality, and lower layer, which it uses to implement the provided functionality (GARLAN; SHAW, 1993, p. 9).. 39, 58, 77

load balancing Load balancing is the problem and associated task of distributing work among available processors in such way that no processor is overloaded and while some processors are idle. Ideally all available processors should have the same load. (CYBENKO, 1989). 70

logical model The precise definition of a model varies according to the specific logic under study. In general, a function from the set of sentence symbols to the set of truth values (i.e., $\{T,F\}$), is a model of a theory if, and only if, all sentences of the theory are truth if symbols are replaced according to the function. The function is often called an interpretation. (HODGES, 1983, p. 12-13).. 47

[software] maturity model A classical definition by Klimko (2001) is that maturity models describe the development of an entity over time. Wendler (2012) reviews existing definitions and concludes that the purpose of maturity models is to outline the conditions under which the object of study reaches the perfect (or most mature) state for its intended application.. 61

media type A string that labels content transported by protocols such as HTTP, allowing the content to be processed by adequate components (FREED; KLENSIN, 2005). 73, 140

middleware A middleware is a software layer, present in all nodes of a distributed system that provides common abstracttions and services, thus hiding heterogeneity among different nodes (PUDER; RÖMER; PILHOFER, 2006; JINGYONG et al., 2009; PIETZUCH, 2004). The precise origin of the term is unclear, but Pietzuch (2004) points to the 1980s.. 17

modal logic A modal logic introduces has its semantics defined in terms of possible worlds and introduces the notions of necessity ($\Box$) and possibility ($\Diamond$). The necessity operator implies a sentence is true in all possible worlds, while the possibility operator implies the sentence is true in at lesat one of the possible worlds. (CHELLAS, 1980). 48

N3 logic N3Logic is a logic language based on the Notation3 langauge, which extends the RDF abstract syntax with the concept of formulas. Formulas are nodes which internally contain another N3 graph. N3Logic includes existential (blank nodes) and universal quantification, conjunction, implication and scoped negation (to test if a formula is present in another). (BERNERS-LEE et al., 2008). 73, 102, 112, 135, 136, 161

paradigm When used in computer science and specially software engineering literature (EUGSTER et al., 2003; SINHA, 1992; ANDREWS, 1991; FUGGETTA; PICCO; VIGNA, 1998), the word paradigm is not defined and its usage is consistent with the dictionary definition of “an example serving as a model” (Dictionary.com, 2017). 17, 39, 40, 41, 54, 58, 62, 92, 95, 132, 167

proxy A proxy (SHAPIRO, 1986, p. 3) encapsulates remote servers and provides a single view to their services. The proxy can then intercept communication and provide additional functionality, such as message translation and performance enhancement. The client must take the initiative of selecting and using a proxy (FIELDING, 2000, p. 46,97).. 58, 77, 246

Safe [HTTP method] A safe method is one which causes no side effects for which the user can be held accountable (FIELDING et al., 1999, p. 51). 213

software architecture A software architecture describes a set of software systems in terms of their implementation elements. An architecture is composed by elements (processing, data and connectors), relations between elements, and justifications for both (PERRY; WOLF, 1992).. 17, 40, 42, 43, 44, 45, 70, 105, 245

triple pattern A triple pattern is specified by a subject, a predicate and a object. Any of these three elements may be bound to specific RDF nodes or may be variables (SEABORNE; HARRIS, 2013).. 124, 245

INDEX