Journal of Network and Computer Applications 60 (2016) 32–53

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

RESTful service composition at a glance: A survey

Martin Garriga [a,c,*], Cristian Mateos [b,c], Andres Flores [a,c], Alejandra Cechich [a], Alejandro Zunino [b,c]

[a] GIISCo Research Group, University of Comahue, Buenos Aires 1400, Neuquen 8300, Argentina
[b] ISISTAN Research Institute, UNICEN University, Campus Universitario, Tandil B7001BBO, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

ARTICLE INFO

Article history:
Received 28 October 2014
Received in revised form
12 June 2015
Accepted 25 November 2015
Available online 13 December 2015

Keywords:
Web Services
Restful services
Service composition
Service mashups

ABSTRACT

In the last years, Web Service composition has undoubtedly become the most promising way to integrate business-to-business applications. However, the industry and the academia often disagree on materializing current solutions, which are based on either SOAP Web Services or semantic Web Services. Besides, any service composition mechanism entails multiple and complex factors such as adaptability, scalability and lightweightness. Recently, RESTful services have shown their potential to compose reliable and visible Web-scale applications based on the so-called mashups. In this paper, we survey a comprehensive set of RESTful composition approaches, i.e., the most promising in their area, totaling 29 approaches. Then, we propose two sets of features to analyze, characterize and compare such approaches: features inherent to SOAP services composition approaches and RESTful services composition features. Lastly, we discuss research challenges and open research problems in the area.

Contents

* Corresponding author at: GIISCo Research Group, University of Comahue, Buenos Aires 1400, Neuquen 8300, Argentina. Tel.: +54 299 4490300x638.
E-mail addresses: martin.garriga@fi.uncoma.edu.ar (M. Garriga), cmateos@conicet.gov.ar (C. Mateos), andres.flores@fi.uncoma.edu.ar (A. Flores), acechich@fi.uncoma.edu.ar (A. Cechich), azunino@conicet.gov.ar (A. Zunino).

http://dx.doi.org/10.1016/j.jnca.2015.11.020
1084-8045/

1. Introduction

As time passed, Web Service technologies matured and organizations started to embrace services to outsource common parts of their application functions. Moreover, Web Service composition has become the most promising way to support business-to-business application integration. Web Service composition refers to combining outsourced Web Services to offer new, value-added services (Singh, 2001). Composition differs from traditional system integration – i.e., via Enterprise Application Integration technologies – in which applications tightly engage each other in a white/gray box fashion – while Web Services promote integration in a black-box fashion. Several initiatives have been conducted to provide platforms, frameworks and languages to enable loose-coupled integration of heterogeneous systems, mostly for SOAP-based Web Services. This encompasses a wide set of standards, such as WSDL (syntactic service description), UDDI (Zimmerman et al., 2003) (service discovery), OWL-S (Martin et al., 2007) (semantic service specification), and BPEL for workflow-based representation of service compositions where bindings between services are known beforehand (Weerawarana et al., 2005).

However, stakeholders disagree in materializing these solutions, mainly due to a lack of widely accepted usage standards (Daniel et al., 2008; Bozkurt et al., 2013; Rauf et al., 2008). Particularly, organizations often develop and/or describe Web Services using different interface specification practices and concept models. From the specification practices point of view, unless appropriately specified by providers, service meta-data can be counterproductive and obscure the purpose of a service, thus hindering its adoption. These phenomena are known as *anti-patterns*, or indicators of poor-quality service interfaces (Rodriguez et al., 2012) that entangle definition and analysis of Web Services and compositions. From the concept models point of view, different approaches model Web Service compositions using domain specific languages and new modeling notations. Such ad hoc modeling methods are hard to learn and use, and often have a very limited tool support available (Rauf et al., 2008).

In the last few years, RESTful (REpresentational State Transfer) (Fielding, 2000) services appeared as a lightweight and cost-effective alternative for SOAP-based services. Lightweight RESTful services are designed to ease consumption, composition and building of community-driven services (mashups). Motivated by these facts, in this paper, we analyze and compare existing proposals in the RESTful service composition area. This area is fairly new and remains somewhat unexplored, evidenced by the lower number of mature proposals compared to SOAP-based service composition, but the area is growing at a rapid pace and has interesting practical implications. We have defined two sets of features to guide this analysis, covering functional and non-functional characteristics found in both SOAP-based and RESTful composition approaches. Briefly, the main contributions of this paper are

- A common ground to taxonomically analyze service composition approaches, which comprises two sets of features – features inherent to SOAP-based services and RESTful-specific features.
- A survey of RESTful service composition approaches, according to the aforementioned features.
- An outline of research challenges in the area of RESTful service composition and open future research opportunities.

The rest of this paper is organized as follows. Section 2 steps into some of the concepts mentioned along this section and presents related surveys. Section 3 presents an overview of the features identified in existing service composition approaches that guide this analysis. Section 4 discusses the reviewed RESTful service composition approaches, analyzed according to the previously identified features. Section 5 highlights the results of the analysis. Section 6 attempts to foresee the evolution of RESTful service composition by tracing a parallel with SOAP-based service composition, pointing out research challenges in the area. Conclusions and future work are presented afterwards.

2. Background

From the SOAP side, two cornerstone terms are commonplace to classify specifications and technologies for Web Service composition: (1) *Orchestration*, an executable business process built with Web Services seen from a single-party perspective; and (2) *Choreography*, the message sequences between multiple Web Services seen from the perspective of multiple parties (Peltz, 2003). These terms were initially coined in the context of SOAP-based service composition, but they applied to RESTful composition approaches as well.

From the REST side, RESTful services provide a simple, lightweight and scalable alternative to SOAP-based services. REST uses

the basic built-in HTTP remote interaction methods (`PUT`, `POST`, `GET`, and `DELETE`) applying their intended semantics to access any URI-referenceable *resource* (Fielding, 2000). A resource could be any piece of data in the Web such as a document, an image, a tweet or a weather forecast. Moreover, RESTful services exhibit four properties:

1. Resources represent an abstraction for server-side application state and entities, i.e., any element that may be the target of a hypertext reference is a resource.
2. Each resource is addressable using an unique worldwide identifier (URI).
3. All resources share a uniform interface – HTTP methods – to interact with client applications.
4. Interaction with a resource is stateless.

RESTful Web Services are perceived to be *simple* because REST leverages existing well-known Web standards (HTTP, XML, URI) and the only necessary infrastructure is the Web, which has already become pervasive. HTTP clients and servers are available for all major programming languages and operating system/hardware platforms. This leads to the second main strength of RESTful services, *lightweightness*, where services can be built with minimal tooling, which is inexpensive to acquire and thus has a very low barrier for adoption (Pautasso et al., 2008). Lightweight services are easier to consume, are more often used to provide services across organizational borders and are interesting for community-driven services (Lanthaler and Gutl, 2010). This is particularly true for the growing pervasive computing environment, where mobile devices with different capabilities can act as clients and even servers or hosts for Web Services. The notions of simplicity and lightweightness bring support to *scalability*, since a RESTful Web Service can scale to serve a very large number of clients, thanks to the built-in support for caching, clustering and load balancing of REST. Not having to store state between requests allows the server to quickly release computational resources, and further simplifies implementation because the server does not have to deal with "conversational" resource usage across requests (Fielding, 2000). Additionally, stateless servers allows the service consumer (human or machine) to directly manipulate the state of the application through hyperlink navigation, which is known as the HATEOAS (Hypermedia As The Engine Of the Application State) principle. The Web as the RESTful architecture by excellence empowers the scalability potential of REST.

Resource composition in REST is handled via *mashups*. A mashup often takes the form of a Web page or website that combines information and services from multiple sources on the Web, offering user-oriented value-added services (Rosenberg et al., 2008). From the point of view of their purpose, mashups can be broadly classified into the following categories (Peenikal, 2009):

- *Data-oriented* mashups involve converting, transforming and combining similar data elements represented by resources into a single value-added output.
- *Process-oriented* mashups imply a step forward from data-oriented mashups, allowing the composition of business services and the integration of these latter within existing business processes.
- *Consumer-oriented* mashups are aimed at the general public as an effective means for customer personalization of data/viewing, where users employ any Web browser to combine and reformat the data according to their needs. An example is Yahoo Pipes,[1] which uses the idea of Unix-like pipes for mashing up services. Then, unlike data-oriented mashups, in consumer-oriented mashups composition is mostly GUI-driven and can be seen as the simplest case of data-oriented mashups.

In this paper we will focus on data-oriented and process-oriented mashups, since they present the most challenging issues from a service composition perspective.

2.1. Related Work

Recently, RESTful architectures gained momentum mainly because their scalability, their usage in Web 2.0 mashups and their simplicity to be published and consumed (Pautasso, 2009). The *mashup* concept emerged in 2008, and since then a large number of Web 2.0 applications in the form of mashups have been developed (Benslimane et al., 2008). In (Yu et al., 2008) these approaches were surveyed, showing that they were mostly data-oriented mashups. Surveyed approaches were either manual, requiring programming skills and knowledge about schemes and semantics of data sources, or tool-assisted, enabling end users to quickly prototype their own Web applications via mashups and speeding the development process. More recently, the work in de Vrieze et al. (2011) analyzed recent mashup tools, enterprise mashups and service-oriented workflows. An enterprise (or business) mashup is an application that combines data from multiple internal and public sources and publishes the results to enterprise portals, application development tools, or as a value-added service. Enterprise mashups must also interoperate with enterprise application technologies for security, governance, monitoring, and availability. The survey showed that, in general, available mashup tools are mainly data-driven Web applications providing simple and seamless end-user programming. The work asserts that enterprise mashups still focus on providing aggregation at a business data level, with less emphasis on the composition of business functions. However, enterprise mashups represent a step towards providing process-oriented mashup development. Lastly, service-oriented workflows are mostly heavyweight approaches for collaborative applications.

The survey in Lanthaler and Gutl (2010) compares "heavyweight" (i.e., SOAP-based) and "lightweight" (i.e., RESTful) approaches based on different aspects of services lifecycle. In particular, Lanthaler and Gutl (2010) focuses on service discovery and composition, since in practice a client (human or machine) has to find a desired Web Service prior to composing it with other services. Since no UDDI-like technology exists for REST, the usual way to find a RESTful service implies a human manually inspecting a Web site such as ProgrammableWeb[2] that collects and categorizes published services. If service descriptions were augmented via semantic annotations such as SA-REST (Semantic Annotations of Web Resources) or MicroWSMO (a microformat based on Web Service Modeling Ontology), it would be even possible to use that semantic data for partially automating service discovery and composition. Fully automation of the composition process, however, needs semantic annotations using concepts from ontologies but it is not always possible to define a complete mapping from the native data format of RESTful services to the data structure of an ontology (Lanthaler and Gutl, 2010).

After finding suitable RESTful services, mashups are the weapon of choice to combine them. Typically, a developer implements a special mediation layer to translate data formats between different services. Figure 1 illustrates different stacks of languages and protocols for both SOAP-based and RESTful services.

[1] http://pipes.yahoo.com/pipes/.

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

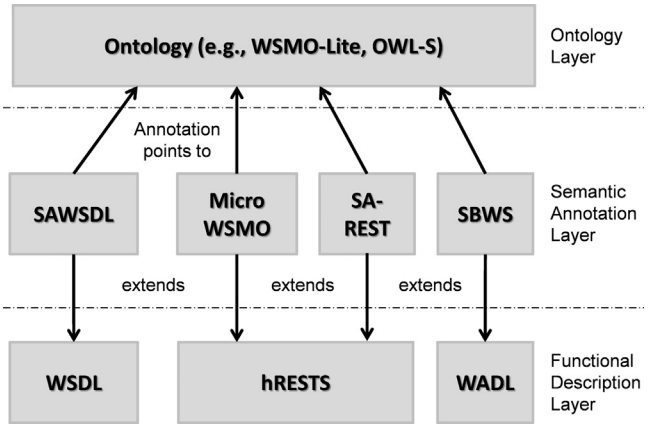

Fig. 1. Comparison of SOAP-based and RESTful languages stack (adapted from Lanthaler and Gutl, 2010).

SOAP-based services have witnessed the creation of more languages and standards.

Even though mashups bring innovative and value-added applications for Web users, they are not wide-spread in enterprises due to the lack of a standardized process or specification to embody mashups (Peng et al., 2009). As long as current approaches mature, stakeholders will think of mashup applications as compositions of loosely-coupled and reusable services, and then business modeling languages will connect and manage these services together with SOAP-based services in a standard way.

Finally, Zur Muehlen et al. (2005) illustrates the technical and social debate "REST vs. SOAP" from the standardization process point of view, and discusses the implications of these design philosophies. REST and SOAP are not necessarily opposites: REST is an architectural style, and SOAP is a general protocol that can be used as an element of many different architectures. Authors suggest thinking REST as a navigational style of design, and thinking SOAP as a procedural style. Despite these terms are easy and often-used, they lead to active debates. In terms of architectural principles, conceptual decisions and technological decisions, RESTful services emerged as an ad hoc integration alternative, while SOAP-based services are used for enterprise level application integration where transactions, reliability and message-level security are critical (Pautasso et al., 2008).

However, many efforts have demonstrated the potential of RESTful services for enterprise integration scenarios (de Vrieze et al., 2011; Vinoski, 2007; Richardson and Ruby, 2008). Hence, motivated by the protagonic role RESTful services have acquired recently, specially for service composition tasks, in this paper we survey existing RESTful service composition approaches to provide a comprehensive analysis of the developments in the area. To the best of our knowledge, this is the first survey focused on offering an overview of existing composition approaches, and their strengths and weaknesses.

3. Features to characterize RESTful service composition

RESTful service composition is a new challenging research area lacking standardized solutions, and without a clear context to characterize current efforts. According to the surveys presented in Section 2.1, and others that provide common terminology in the area (i.e., (Peltz, 2003)), overview current SOAP-related standards (i.e., (Daniel et al., 2008)), and establish guidelines for automatic service composition (i.e., (Rao and Su, 2004; Zhao and Doshi, 2009)), we have defined two sets of features to characterize service composition approaches:

- *General features* (Section 3.1): Eight features that cover how the approaches relate to up-to-date service-oriented technologies, languages, practices and standards. In this case, standards refer to SOAP-based and Web-based standards that could be also applied to RESTful services and compositions.
- *REST-specific features* (Section 3.2): Four features that represent REST-specific properties that do not apply to traditional SOAP-based service composition.

The motivation behind the first set of features to characterize both kinds of composition approaches is that, despite their differences, SOAP-based and RESTful approaches share architectural, conceptual and technological underpinnings (Pautasso et al., 2008). Then, we considered that those features could frame RESTful approaches as well, and therefore we regarded them during this analysis as general features.

3.1. General features

Table 1 presents a summary of each general feature along with a list of their possible values. Then, Section 4 outlines the analysis of current composition approaches based on this set of general features.

3.1.1. Composition view

The Composition View feature characterizes current approaches mainly considering: orchestration, choreography, workflow, data-oriented or process-oriented mashup. Orchestration and choreography (as defined in Section 2) seem to encompass all SOAP-based composition alternatives and often both are slightly overlapped. They also apply to many RESTful composition approaches, as we will show in Section 4. Furthermore, the workflow composition view arises from the fact that, in many ways, a composite service is similar to a workflow. The current interest in Web Services is directing attention to issues that have a longer history in the workflow community. Viewing a service composition as a workflow allows addressing service interoperability with well-known and broadly-tested tools, techniques and standards (Zur Muehlen et al., 2005; Rao and Su, 2004). The data-oriented mashup and process-oriented views are strictly inherent to RESTful approaches, as discussed in Section 2.

Possible values for this feature are "Orchestration", "Choreography", "Workflow", "Data-oriented Mashup", "Process-oriented Mashup", or "Ad hoc", syndicating in the last case the name of the proposed composition view.

3.1.2. Automation level

According to this feature, service composition approaches can be broadly classified into three categories: manual, semi-automated and automated (Majithia et al., 2004). Manual composition frameworks expect the user to generate composition scripts (which sketch the composition in an abstract way) either graphically or through a text editor, and then submit the scripts to an execution engine (which instantiates and eventually executes the composition). However, these systems have several drawbacks such as programming effort, manual re-binding – which implies manually changing atomic services of a composition upon detecting failures – and the need of users with good programming skills. Semi-automated composition techniques make suggestions for service selection during the composition process. Nevertheless, the user still needs to select services and link them up in the desired order. These systems are not scalable yet – as the filtering process may provide numerous services for the user to select from – and lack self adaptation and autonomy. Moreover, automated techniques use AI planning or similar techniques to automate the entire composition process. Fully automated composition requires

Table 1
Summary of general features to characterize composition approaches.

Category	Feature	Description	Possible values
Technologies	Composition view	Point of view (one partner, multiple partners) and focus (process, data) of the composition addressed by the approach	Orchestration / Choreography / Workflow / Data oriented Mashup / Process oriented Mashup / Ad hoc
	Automation level	The proposed solution automates to some extent the composition problem through the use of ontologies and/or AI planning	Manual / Semi-Auto / Auto
	Definition and binding	Applications can be built by composing services either at design time (static), or at runtime and on the fly (dynamic). In hybrid approaches, the composition is delineated statically and concrete services are bound dynamically	Static / Dynamic / Hybrid
Standards	Standards conformance	The proposed solution adheres to some technical standards, languages or specifications related to service composition	Yes*(name of the standard)*/ No
	Service and composition specification	Types of (composite) service specifications accepted by the composition proposal	(*Name of the input language*)/ Ad hoc
Practices	Adaptation perspective	Composition logic can be adapted and hence altered from an abstract perspective only, and/or adapted from a concrete perspective, or adaptation support is not provided at all. In case of adaptations performed at the concrete level, functional incompatibilities between services can be managed by modifying service interfaces or message ordering, while QoS infringements or component failures can be managed by re-binding the composition workflow	Abstract / Concrete (Interface, Protocol, Workflow re-binding) / Both/ Not Applicable
	Verification & validation (V&V)	The proposed solution supports verification, validation or testing of the service compositions built	Yes (*provided aspect*) / No
	QoS-awareness	Service composition considers QoS or non-functional properties	Yes / No

understanding the context, semantics, and problem space of the general composition domain. In fact, proposed approaches in this line have dealt with small, restricted parts of fully automated composition (Paik et al., 2012).

Possible values for this feature are “Manual’, “Semi-automatic” and “Automatic”.

3.1.3. Definition and binding

Services can be composed either manually and a priori, or autonomously and on the fly. The first approach is called *static* service composition, and the latter is *dynamic* service composition (Chakraborty and Joshi, December 2001). Compared with static service composition, dynamic service composition enables for more flexible and adaptive applications through the composition of services according to user contexts (e.g., location, time, and profile) and user preferences. Dynamic service composition also reduces application development costs because new applications are created by saving service discovery time and simply deploying a small number of new components which autonomously locate required services. However, static service composition is better suited for designing complex interaction patterns, such as branch or iteration, often present in B2B applications, which are complex but easy to foresee at design time (Fujii and Suda, 2004). All in all, each composition time is better suited for certain contexts.

Possible values for this feature are “Static”, “Dynamic”, or “Hybrid”, since a service composition approach could support both static and dynamic definition and binding.

3.1.4. Standards conformance

Without robust and common agreement on standards, researchers and practitioners cannot address real interoperation and composition problems adequately (Daniel et al., 2008). Standards are, by definition, well-founded and documented. Intuitively, standards-conforming composition approaches may be easier to implement and adopt than ad hoc solutions. Therefore, approaches whose composition schemes conform to current standards belong to this category. In most proposals, the language used to specify either services or compositions is standardized. Such standardized languages are encompassed in the corresponding feature *Service Composition Specification*. It is worth noticing that, although all RESTful composition proposals conform to the HTTP standard, this feature looks for other standards and particularly REST-specific standards.

Possible values for this category are binary (“Yes”, “No”), along with the names of the standards adopted by each approach when applicable.

3.1.5. Service and service composition specification

This feature identifies which specifications a composition system supports for sketching either required services or compositions. Functional descriptions communicate what an atomic or a composite service does. Non-functional descriptions discern between two (or more) services designed for performing the same task. When such descriptions involve semantic information, they require to place extra effort on building domain ontologies and annotations (Crasso et al., 2011). Other approaches focus on categorizing as much QoS aspects as possible without requiring any extra semantic markup (Al-Masri and Mahmoud, 2007).

Possible values for this feature are the specifications/languages supported by the system under review. A service composition approach may also accept as an input a specification or language not strictly designed for Web Services – e.g., UML. For example, possible values in this respect are “WSDL”, “WADL”, “BPEL” and “UML”.

3.1.6. Adaptation perspective

The mediation or adaption approach is interesting as an economic and effort-saving approach to address partial compatibility in real-life Web Service compositions. Ideally, when detecting undesirable changes in certain threshold values – typically defined over QoS attributes such as reliability or response time – the service system should be able to adapt itself due to such changes (Immonen and Pakkala, 2014).

Adaptation can be performed at different stages of the composition lifetime. In general, compositions can be built in terms of an abstract composition, where only the required functionalities (tasks) and their composition logic are specified, and a concrete composition, where the tasks of an abstract composition are bound to actual implementations – i.e., concrete services. Based on this distinction, adaptation in the SOA domain may be circumscribed to concrete composition perspective, acting on the implementation each task is bound to and leaving the composition

logic unchanged (i.e., the overall abstract composition), or adaptation can affect both the concrete and abstract composition perspective, meaning that the composition logic can be altered (Cardellini et al., 2012).

The first case (adapting only the concrete composition) is the most common, and can involve different levels (Eslamichalandar et al., 2012; Canfora et al., 2006): interface level addresses incompatibilities in service operation signatures; protocol level addresses behavioural mismatches; and workflow re-binding level implies changing atomic services of a composition upon detecting context changes, QoS infringements or component failures. These levels are in concordance with the adaptation actions proposed in (Cardellini et al., 2012) that involve only the services of the composition (i.e., concrete adaptation): interface level and protocol level are analogous to service tuning actions (changing properties or behaviour of services respectively), while workflow re-binding encompasses service selection and coordination pattern selection.

Possible values for this feature are “Abstract”, “Concrete”, “Both” or “Not Applicable (N/A), where “Concrete” can be further decomposed in “Interface Level”, “Protocol Level”, “Workflow re-binding”.

3.1.7. Verification and Validation (V&V)

Validation ascertains the correct behaviour of a service composition – i.e., the service composition behaves as expected. Verification checks the maintenance of certain desirable properties of the composite service (Narayanan and McIlraith, 2002) with respect to the individual composed services. Verification of a composite Web Service flow prior to its execution is mandatory (Nakajima, 2002). Then, during execution of composite services, non-functional aspects such as QoS can also be controlled through runtime monitoring (Yu et al., 2008; Immonen and Pakkala, 2014).

Possible values for this feature are binary (“Yes”, “No”), plus the name of the supported V&V technique, such as “Model-Checking”, “Formal Validation/Verification” or “Runtime Monitoring”, when applicable.

3.1.8. QoS awareness

Considering non-functional properties, particularly Quality of Service (QoS), is crucial for companies to meet the requirements of their customers (Berbner et al., 2006; Daniel et al., 2008). *Quality of Service* (QoS) is a broad measure for how well a service serves its customer. Particularly, for the scope of this work, services are Web Services, and customers are client applications (Strunk, 2010). The end-to-end QoS of a service composition depends on the QoS of the constituting services (Cardoso et al., 2004).

The composite service must fulfill both the quality goals from the consumer's viewpoint and the business goals of the composite service provider. In addition to the traditional definition of quality in software engineering (Kitchenham and Pfleeger, 1996), several new challenges arise in the new ecosystem of Web Service composition (Immonen and Pakkala, 2014). Service selection must consider quality of atomic services, as the QoS of composite service is dependent on qualities of its atomic services. In addition to selecting reliable services, composite service providers must be able to verify QoS of the selected services and the composite service during run-time. This requires atomic/composite services execution to be monitored. Then, if detecting undesirable changes in QoS thresholds, the service system should ideally be able to self-adapt using proper adaptation techniques. Thus, it is noticeable that QoS-awareness is a crosscutting aspect of Web Service composition, concerning not only non-functional properties but also runtime monitoring and self-adaptation of atomic and composite services to ensure required QoS levels.

Besides, with the increasing number of Web Services with similar or identical functionality, the non-functional properties of a Web Service are becoming more and more important. Since several Web Services may embody the same functionality, the key in service composition is to find the services that better fulfill customer's QoS requirements (Immonen and Pakkala, 2014; Berbner et al., 2006). QoS is measured in terms of attributes or properties, such as response time, reliability, availability or reputation.

Possible values for this feature are binary (“Yes”, “No”), plus the particular QoS attributes when QoS is considered.

3.2. REST-specific features

RESTful services provide some advantages over SOAP-based services (Zhao and Doshi, 2009), including lightweightness, easy accessibility and flexibility. Although the research community has put significant effort on SOAP-based Web Service composition so far, it has put much less attention in the RESTful service composition problem. Table 2 presents the set of specific features – comprising four properties adapted from (Zhao and Doshi, 2009) – with which RESTful approaches should be compliant. Each feature is described below, along with their possible values that are binary (Yes/No).

Those four characteristics make RESTful Web Service composition fundamentally different than the composition of SOAP-based services. However, as we will see later, the resource-centric perspective is relatively new for Web Services, and most of the claimed RESTful approaches do not fully adhere to REST principles (Zhao and Doshi, 2009).

3.2.1. Lightweight

RESTful services use HTTP as the invocation protocol, avoiding tunneling higher-level protocols through HTTP (Vinoski, 2007). A resource *representation* itself becomes the response of a RESTful service invocation without any extra encapsulation or envelopes. Additionally, RESTful services depend less on vendor software and mechanisms that implement additional SOAP layers on top of HTTP.

These mechanisms are materialized in the WS-* set of standards and protocols. They include, but are not restricted to, WS-Notification, WS-Security, WSDL, and SOAP, and address the design of process-oriented, brokered distributed services. This granularity of application tends to be more prevalent in businesses and government applications, and less prevalent in technical and academic areas, although recent RESTful approaches are bridging this gap (de Vrieze et al., 2011; Vinoski, 2007).

This set of protocols is not considered lightweight because it takes HTTP to a transport protocol for big XML payloads that wrap the real content of the message/operation. The resulting service is far more complex than its RESTful counterpart, in the sense that the service is harder to debug (because of the multiple-protocol stack), and forces clients to remain tied to a particular configuration (Richardson and Ruby, 2008).

3.2.2. Understandable

Client applications can share and pass around URIs that identify resources. The URIs and the representation of resources are – if specified precisely – self-descriptive and thus make RESTful services easily understandable and accessible. On the other hand, WSDL specifications of SOAP-based Web Services bury these advantages under multiple abstraction layers (Richardson and Ruby, 2008). Paradoxically, developers of SOAP-based services tend to create WSDL descriptions that hinder services understandability and discoverability (Crasso et al., 2010; Mateos et al., 2013).

3.2.3. Scalable

RESTful services support server-side caching of and parallelization on URIs access, just like Web pages are currently cached in and retrieved from network proxies and gateways for improving request performance. The responses of GET (a side-effect free

operation) can be cached as well. Additionally, RESTful services provide an effective way to support server-side load balancing based on URI partitioning. Moreover, a RESTful composition is considered scalable or extensible if new services can be integrated without decreasing the overall performance, availability or other QoS attributes of the composition.

3.2.4. Declarative

In contrast to (SOAP-based) imperative services – operations perspective – RESTful services adopt a declarative approach, namely resources perspective. They focus on *what* resources are modelled and the information that can be exchanged with them, rather than describing *how* those resources perform their functions. The declarative style brings the fundamental differences between RESTful and SOAP-based services to the forefront.

Table 2
Summary of specific features of RESTful approaches.

Feature	Description	Possible values
Lightweight	Uses HTTP as the invocation protocol avoiding improperly tunneling other protocols	Yes/No
Understandable	Service clients can share self-descriptive URIs that identify resources	Yes/No
Scalable	Supports URI server-side caching, parallelization and partitioning just like regular Web pages	Yes/No
Declarative	Services focus on the description of resources, rather than describing how their functions are performed.	Yes/No

4. Restful service composition approaches

Service stakeholders started to embrace REST to ease service publication and consumption. Composition could exploit this wealth of RESTful services to offer value-added services (Pautasso, 2009). Besides, RESTful service composition is often associated with Web 2.0 mashups, which combine and reuse services and data sources in novel and creative ways. In such direction, researchers have proposed some emergent lightweight RESTful composition approaches. Indeed, after searching in online sources, we found several works addressing the RESTful service composition problem. We searched for articles indexed in Scopus, Science Direct, IEEE Xplore, ACM Digital Library, SpringerLink, Google Scholar and WileyOnline. Then, we looked for relevant references included in the works previously found, in order to identify other potential works. As the RESTful service composition is a very recent topic, we included both journal and conference articles. We found twenty nine relevant approaches in total. Tables 3 and 4 summarize the key aspects of the analyzed approaches. Then, for the sake of presentation, we grouped the twenty nine surveyed works into three categories, namely Heterogeneous composition, Formalization of RESTful composition and Pure RESTful composition.

Apart from organizing the discussed works, this categorization draws a parallel with the evolution of SOAP-based composition approaches (Rao and Su, 2004; Srivastava and Koehler, 2003; Sheng et al., 2014; Issarny et al., 2011), as we will discuss in Section 6. On the one hand, industry and business approaches, mainly workflow-based, embraced RESTful services by integrating them into existing SOAP-based business processes as quickly and seamlessly as possible. Hence, heterogeneous service composition approaches (Section 4.1), which bridge the gap between RESTful and SOAP-based services integrating them into a single composition framework. In this sense,

Table 3
RESTful service composition approaches (part I).

Id	Approach	Key points
01	Pautasso (2009)	BPEL4REST extension, enabling composition of both RESTful and SOAP-based services and publication of BPEL processes with REST primitives.
02	De Giorgio et al. (2010)	SOA4All extension, with dynamic replacement of SOAP and REST services through MicroWSMO semantic annotations. Supported by LPML (Lightweight Process Modeling Language), a language based in BPMN and BPEL
03	Peng et al. (2009)	REST2SOAP framework to wrap RESTful services into SOAP services, creating a composite BPEL service
04	Rodrigues et al. (2013)	Data event-based approach for Web Service composition in which the execution of business processes is driven by changes in data states
05	Maximilien et al. (2008)	Composition of RESTful and SOAP-based via interpreted programming languages. Provides a comprehensive solution to creating, reusing, deploying, and managing Web API mashups
06	Liu et al. (2012)	Dynamic mashup building using RESTful and SOAP-based Web Services with their interfaces unified through lightweight semantics
07	Farokhi et al. (2012)	MDCHeS, a model-driven framework to support dynamic composition of SOAP-based and RESTful services
08	Haupt et al. (2014)	Meta model for REST service composition built based on extension capabilities of BPEL, namely *extension activities*
09	Wu et al. (2013)	Exploits Apache ODE (Orchestration Director Engine) extensions to generate WSDL documents for RESTful services to invoke them from BPEL processes using the standard HTTP methods
10	Zhao et al. (2011)	Linear logic-based formal approach to automatic RESTful Web Service composition, executable at the business management level in a search-efficient, correct, complete, and formalized way through process calculus
11	Zhao and Doshi (2009)	Situation calculus-based formal model for describing individual RESTful Web Services and automating their composition. Formal description of services through SWRL (Semantic Web Rule Language)
12	Riabov et al. (2008)	Iterative refinement of data-oriented mashups definition, supported through a planning-oriented formalism
13	Zuzak et al. (2011)	A formal model for describing RESTful systems that comprehensively follow REST principles
14	Zhang et al. (2014)	Context-aware discovery and composition of generic (electronic and non-electronic) services. A soft constraint-based context model can give the user several "good enough" solutions according to context information
15	Li and Chou (2014)	Approach based on Category Theory. RESTful services are defined through two categories: resources and links (functions). Then, simple algebraic rules define composition upon link transitivity
16	Xie et al. (2013)	A semantic resource service model (SRSM) that combines structural and operational semantics for resource-oriented service composition through ontology reasoning
17	Pautasso (2009)	Visual composition language that uses an iterative methodology for composition development with interactive debugging and testing tools
18	Alarcon et al. (2011)	ReLL, a dynamic and lightweight RESTful service composition approach driven by the hypermedia network, using Petri-Nets to dynamically describe machine-client navigation
19	de Vrieze et al. (2011)	A lightweight Business Process Modeling (BPM) approach that eases requirements specification of process-oriented enterprise mashups
20	Li et al. (2013)	A lightweight composition mechanism in Web browsers which allows users to interactively chain distributed atomic Web Services similar to using Unix pipelines
21	Rosenberg et al. (2008)	Bite, a composition language and lightweight framework to create Web-scale workflows based on RESTful services.

Table 4
RESTful service composition approaches (part II).

Id	Approach	Key points
22	Pautasso and Wilde (2013)	Business processes modelled and observed using RESTful push services, Atom feeds and notifications through push protocols, instead of traditional pull-oriented solutions
23	Lanthaler and Gutl (2011)	A machine-readable semantic description language for RESTful services based on JSON, to enable automatic discovery and composition
24	Choi (2012)	Transforming results of atomic RESTful services into objects, to bring service composition to lay object-oriented programmers
25	Sepulveda et al. (2014)	Identifies key QoS elements, particularly for the security domain, captured as an ontology. An extension to ReLL that considers security constraints (ReLL-S) allows a machine-client to interact with secured resources
26	Bennara et al. (2014)	A composition engine that allows to discover the interaction patterns a resource offers and its orchestration possibilities with other resources, using the REST principles and linked data
27	Bellido et al. (2013)	Analysis and implementation of control-flow patterns through the extension of the HTTP status codes. Allows composed resources to delegate control flow to different services according to status codes
28	John and Rajasree (2013)	A combination of Microformat markup over HTML, RDF and RDFS to support semantic RESTful services description, discovery and composition
29	Lu et al. (2015)	An architecture to design, implement and execute RESTful business processes with maximal adaptability described through BPMN. An automatic decision-making module can adapt the business process at runtime.

approaches in this category in principle have an increased practical value.

On the other hand, academic solutions are mainly based upon the formalization of domain concepts through ontologies or other semantic constructs. Then, some underlying theory (such as Finite State Machines, Semantic Reasoning or Constraint Satisfaction) allows composing services while formally verifying some desirable properties. Thereby, formalization of RESTul compositions approaches (Section 4.2), which formalize description and composition of RESTful services. Their final goal is either full automation of the composition process or satisfaction of certain properties throughout the composition lifecycle. As such, these kind of approaches could experience a stepwise adoption in the industry.

Finally, acknowledging the maturity of RESTful service composition as a research field itself, certain approaches propose novel composition methods particularly for RESTful services, i.e., with little or even no ties to SOAP-based service concepts. Hence, Pure RESTful service composition approaches (Section 4.3), which are only intended to cope with RESTful services, composing them as an evolution of data-oriented or process-oriented mashups.

We analyzed the works in these categories according to the two sets of features defined in Section 3. Tables 5 and 6 summarize the results regarding the general features. Table 7 summarizes the results in regards to the specific features.

4.1. Heterogeneous service composition

One of the most mature proposals in the field of RESTful Web Service composition is (Pautasso, 2009) (Id. 01). The author proposes an extension for BPEL to enable composition of both RESTful Web Services and "traditional" SOAP-based Web Services. The contribution is twofold: it shows how the HTTP binding of the 2.0 version of WSDL can be used to describe RESTful services, and how a BPEL process can be published as a RESTful Web Service by exposing certain parts of its execution state using the REST primitives `GET`, `PUT`, `POST` and `DELETE`. Three BPEL extensions map the *resource* abstraction and support the corresponding interaction mechanisms and invocation patterns: (a) a BPEL activity for each native REST primitive to invoke RESTful services, (b) a dynamic resource representation to expose the execution state of a BPEL process to clients, and (c) a revision of BPEL constructs to fit with REST design principles.

Another work that brings together SOAP-based and RESTful services is (De Giorgio et al., 2010) (Id. 02). It addresses the dynamic replacement of both kind of services from within a service composition. MicroWSMO (Kopecky et al., 2008) and LPML (Un et al., 2010) (Lightweight Process Modeling Language) support the approach. MicroWSMO is a semantic annotation mechanism for RESTful Web Services based on a microformat called hRESTS (HTML for RESTful Services). This leads to machine-readable descriptions of Web APIs, and serves as a rough equivalent to WSDL enabling the substitution of a SOAP-based service with a RESTful one (and viceversa) to implement activities of a business process. LPML borrows concepts from BPMN and BPEL in a lightweight manner, i.e., by introducing little extensions to support dynamic binding and service substitution at runtime. The idea of adding an annotation level to wrap RESTful services into more descriptive specifications counterposes that of directly invoking RESTful services as proposed in (Id. 01).

Another effort to integrate SOAP-based services and RESTful services is the REST2SOAP framework (Peng et al., 2009) (Id. 03). Due to the similarity between SOAP-based compositions and Web 2.0 mashups, the authors handle REST-based mashup applications as compositions of loosely-coupled and reusable services, with BPEL as the standard way to connect and manage these services. However, according to the authors, BPEL cannot fully integrate and invoke RESTful services, due to the inherent differences between RESTful and WSDL/SOAP based services, such as operation styles, supported data types and representation media. REST2SOAP leverages WADL (Hadley, 2006) (Web Application Description Language) and XUL (Bojanic, 2003) (XML User Interface Language) specifications to wrap RESTful services into SOAP services semi-automatically, fostering the creation of BPEL composite services combining SOAP services, RESTful services and GUIs. This approach is consistent with the wrapping idea proposed in (Id.02).

The work in (Rodrigues et al., 2013) (Id. 04) proposes the WED-flow (Workflow Events Data) approach, in which the execution of business processes is driven by changes in data states. According to the authors, the control flow is not a requirement but a consequence of Web Services execution, while a data event-based approach for Web Service composition provides greater flexibility for the development and maintenance of applications. The work evaluates possible scenarios for data event-based orchestration and choreography. Then, the WED-flow executes Web Services which interact in the context of those possible scenarios. It uses a polling-like approach, monitoring relational databases in order to catch and handle data events. For instance, an e-shopping Web Service can change a database by writing a payment order. Then, WED-flow monitors that change in the database and triggers the execution of a Web Service for payment validation. Accepted services can be either RESTful or SOAP-based as long as their data is available for usage by WED-flow monitors. The idea of a data-event driven composition is different to the aforementioned approaches, since these latter focus on modeling the control flow of compositions (e.g., through BPEL) rather than the data flow.

Table 5
Characterization of RESTful approaches according to general features (part I).

Id	Composition view	Automation level	Definition and binding	Standards conformance	Service and composition specification	Adaption perspective	Verification & Validation	QoS-awareness
01	Orchestration	Manual	Dynamic	No	Ad hoc (Extended BPEL)	N/A	N/A	No
02	Workflow	Semi-auto	Dynamic	No	LPML, BPML, REST	Concrete (Protocol)	N/A	Yes (not specified)
03	Workflow, process-oriented mashup	Semi-auto	Static	No	WADL, WSDL, BPEL, XUL	Concrete (Interface)	WADL validation	No
04	Workflow	Manual	Dynamic	No	Ad hoc (WED-flow)	Concrete (Workflow re-binding)	Monitoring	No
05	Process-/data-oriented mashup	Manual	Static	No	Ruby on Rails / Ad hoc (DSL)	N/A	No	No
06	Data-oriented mashup	Semi-auto	Hybrid	No	MSM (Minimal Service Model)	Concrete (Workflow re-binding)	No	No
07	Workflow	Semi-auto	Dynamic	No	MD ad hoc specifications	Concrete (Workflow re-binding)	No	Yes (Availability, response time)
08	Workflow	Manual	Hybrid	No	Ad hoc (BPEL Extension)	N/A	No	No
09	Workflow	Manual	Static	Yes (BPEL)	BPEL, WSDL	N/A	No	No
10	Workflow	Auto	Static	No	Linear Logic	N/A	Correctness	No
11	Workflow	Auto	Static	Yes (SWRL)	SWRL	N/A	Formal verification	No
12	Data-oriented mashup	Semi-auto	Dynamic	No	Ad hoc (SPPL)	Concrete (Workflow re-binding)	Formalization of composition flow	No
13	Ad hoc (FSM)	Manual	Static	No	FSM with ϵ-transitions	N/A	FSM Formalization	No
14	Ad hoc (Planning)	Semi-auto	Static	No	Ad hoc (planning-like)	N/A	No	Yes (context information)
15	Ad hoc (Categorial link composition)	Manual	Static	No	Ad hoc (categorial links)	N/A	Category Theory formalization	No
16	Ad hoc (Semantic reasoning)	Semi-auto	Static	No	Ad hoc (SRSM), OWL	Concrete (Interface, Workflow re-binding)	Semantic reasoning	No

Table 6
Characterization of RESTful approaches according to general features (part II).

Id	Composition view	Automation level	Definition and binding	Standards conformance	Service and composition specification	Adaption perspective	Verification & Validation	QoS-awareness
17	Ad hoc (control flow)	Manual	Hybrid	No	Ad hoc (JOpera)	Concrete (Interface)	Testing	No
18	Ad hoc (petri-nets)	Manual	Static	No	Ad hoc (ReLL)	Concrete (Interface)	N/A	No
19	Process-oriented mashup	Manual	Hybrid	No	Ad hoc (unnamed)	N/A	N/A	No
20	Data-oriented mashup	Manual	Static	Yes (HTML5)	HTML5 Microdata	N/A	N/A	No
21	Process-oriented mashup	Manual	Dynamic	No	Ad hoc (Bite)	N/A	N/A	No
22	Process-oriented mashup	Manual	Static	Yes (Atom)	Atom feeds	N/A	N/A	No
23	Data-oriented mashup	Semi-auto	Static	No	Ad hoc (SEREDASj)	N/A	N/A	No
24	Data-oriented mashup	Semi-auto	Static	No	Java	N/A	N/A	No
25	Choreography	Manual	Dynamic	Yes (OAuth)	Ad hoc (ReLL)	N/A	No	Yes (security)
26	Orchestration	Semi-auto	Dynamic	Yes (JSON LD, POWDER)	POWDER, JSON-LD	Concrete (Workflow re-binding)	No	No
27	Choreography	Manual	Dynamic	No	Ad hoc (ReLL)	N/A	No	Yes (availability, throughput, response time)
28	Orchestration	Semi-auto	Hybrid	No	Ad hoc (HTML Microformat + RDF)	N/A	No	No
29	Workflow	Semi-auto	Hybrid	No	BMNL, ad hoc annotations	Both (Abstract, Workflow re-binding)	Runtime monitoring	Yes (availability, response time)

Table 7
Characterization of RESTful approaches according to specific features.

Id	Lightweight	Understandable	Scalable	Declarative	Satisfied properties
01		Yes		Yes	2/4
02	Yes				1/4
03		Yes			1/4
04					0/4
05					0/4
06	Yes		Yes		2/4
07					0/4
08		Yes		Yes	2/4
09					0/4
10				Yes	2/4
11				Yes	1/4
12	Yes		Yes		2/4
13	Yes	Yes	Yes	Yes	4/4
14					0/4
15	Yes		Yes		2/4
16		Yes			1/4
17	Yes	Yes		Yes	3/4
18		Yes		Yes	2/4
19	Yes				1/4
20	Yes			Yes	2/4
21	Yes	Yes			2/4
22		Yes	Yes	Yes	3/4
23	Yes	Yes	Yes	Yes	4/4
24			Yes		1/4
25		Yes		Yes	2/4
26	Yes	Yes		Yes	3/4
27		Yes	Yes	Yes	3/4
28	Yes			Yes	2/4
29		Yes	Yes	Yes	3/4
Total	12	14	9	15	2/4 (average)

Moreover, the work in Maximilien et al. (2008) (Id. 05) presents an online mashup platform (Swashup) that supports mashup design, sharing and management. The final goal is to solve Web integration and service composition problems through mashups via interpreted programming languages. Both RESTful and SOAP-based services are represented in an ad hoc DSL (Domain Specific Language), developed to specify services and mashups. Then, the platform executes these DSL-specified services through a collection of Ruby on Rails (RoR) Web applications and APIs, all backed by a relational database. In this approach, unlike all the previous approaches, both RESTful and SOAP-based services are mapped to a more abstract representation through the DSL.

Another approach to ease dynamic mashup development is presented in Liu et al. (2012) (Id. 06). Dynamic mashups are built using Web Services described with lightweight semantics. The user interface components interact with unified interfaces of both SOAP and RESTful services, rather than invoking those services directly. These unified interfaces are described using the Minimal Service Model (MSM), which captures the semantics of services in a simple RDF ontology, where each service consists of a set of operations (HTTP methods applied to resources in the case of RESTful services) plus links to functional classifications and non-functional properties. Thus, the most important step is the Semantic Services Authoring, where service descriptions are annotated with MSM and domain ontologies, and then published via the iServe platform (Pedrinaci et al., 2010). With these descriptions, the platform invokes services by instantiating requests with parameters received into URIs, and then post-processes service results either to call another service in the mashup, or to transform them into RDF format to merge them with other results before sending back the complete answer to the consumer. The MSM abstraction for both RESTful and SOAP-based services is similar to the DSL-based representation in (Id. 05).

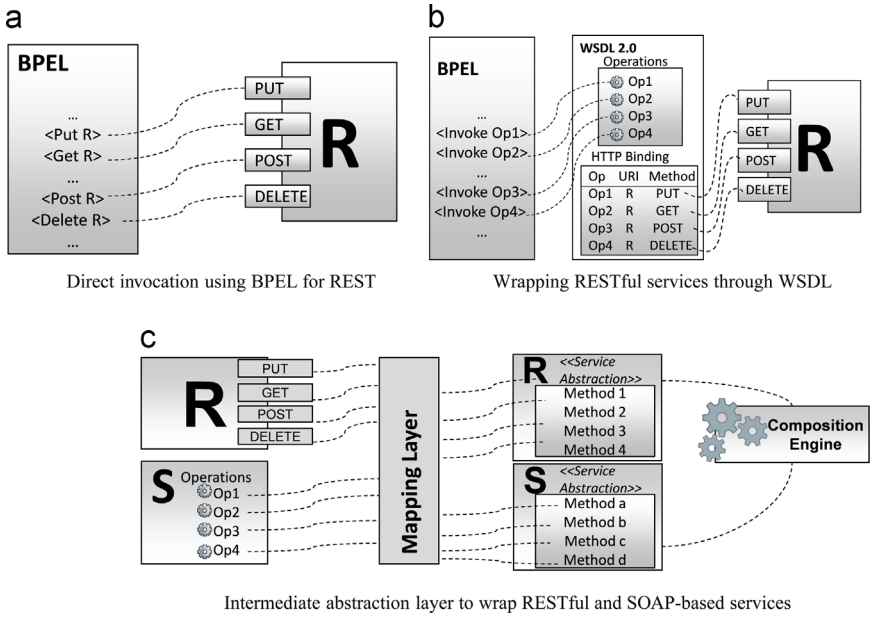

Fig. 2. Three alternatives to compose RESTful and SOAP-based services (adapted from (Pautasso, 2009)). (a) Direct invocation using BPEL for REST. (b) Wrapping RESTful services through WSDL. (c) Intermediate abstraction layer to wrap RESTful and SOAP-based services.

However, in this case all services are invoked in a RESTful way (through the iServe platform).

A model-driven framework to support dynamic composition of both SOAP-based and RESTful Web Services is presented in (Farokhi et al., 2012) (Id. 07). It comprises three different views. The Data view represents in a metamodel the composition elements and their relationships, to construct a composite service. The Process view defines the process of designing a composite service and derives models to increase the abstraction, dynamicity and simplicity level of such process. Finally, the Component view presents components and their interactions to fulfill the required tasks. As in (Id. 05) and (Id. 06), the key point of the approach to compose RESTful and SOAP-based services is their common representation through a higher level of abstraction. In this case, the metamodel of the Data view provides the *Service* element, which contains all the common attributes for a Web Service, such as name, type (RESTful/SOAP-based), method name (HTTP methods or operations respectively), function (the main intent of the service), URI, parameters, and non-functional requirements (when provided).

The work in Haupt et al. (2014) (Id. 08) proposes a meta model for RESTful service composition as an extension of BPEL. The resulting BPEL processes are also BPEL-compliant processes. To achieve this goal, authors rely on extension capabilities already defined in the BPEL language, namely *extension activities*. For each HTTP method, a corresponding REST extension activity is defined in BPEL, also containing its URI and related resources (according to the HATEOAS principle). This approach is similar to (Id. 01) as it leverages BPEL extension capabilities to support RESTful services.

Finally, the approach in (Wu et al., 2013) (Id. 09) uses Apache ODE extensions[3] to generate WSDL documents for RESTful services in order to allow BPEL processes to invoke RESTful services the same as SOAP-based services. RESTful services are invoked by sending HTTP requests instead of SOAP messages. Then, a typical BPEL process can transparently invoke both types of services through the `invoke` primitive and the corresponding partner link pointing to the corresponding service. Service description and composition into BPEL processes is semi-automatically supported by an ad hoc platform named Service Generation System (SGS), which is implemented as an Eclipse plugin. As in (Id. 02) and (Id. 03), this work wraps RESTful services into WSDL specifications – in this case WSDL 1.1 – to transparently call them from a BPEL process without modifying this latter.

Figure 2 graphically shows the three alternatives to compose RESTful and SOAP-based services presented along this section. Figure 2(a) represents the proposal in (Id. 01) and (Id. 08), which directly invokes RESTful services extending the BPEL language. In contrast, Fig. 2(b) wraps RESTful services leveraging the HTTP binding of WSDL specifications. This allows using both RESTful and SOAP-based services without modifying the BPEL workflow that manages the composition – as proposed in (Id. 02), (Id. 03) and (Id. 09). Figure 2(c) shows the third alternative for heterogeneous service composition. In this case, both RESTful and SOAP-based services are mapped to a common representation, which allows composition through an ad hoc composition engine – as proposed in (Id. 05), (Id. 06) and (Id. 07). Finally, (Id. 04) does not fit in these models since it is a data-events based approach, where the composition flow is determined by monitoring the Web Services impact on shared data.

4.2. Formalization of RESTful compositions.

An approach to formalize RESTful composition (Zhao et al., 2011) (Id. 10) encompasses a two-level Linear Logic based program synthesis procedure. Linear Logic is a resource-sensitive logic in which any resource can be used only once, and two copies of the same resource are treated as distinct. Firstly, Linear Logic is applied at the abstract-resource level, in which business constraints and resources are expressed as axioms and composition requirements are expressed as theorems of Linear Logic. A theorem proof is then performed to determine the possibility of composition, with Web resources as variables of the axioms. Then, Linear Logic is applied at service-method level with process calculus (π-calculus), where RESTful services are also translated to axioms. π-calculus empowers Linear Logic with concurrent process modeling. The theorem proof is then performed to determine whether the composition requirements can be fulfilled by composing existing RESTful services within the available business constraints. The whole process guarantees that the composition is both correct and

[3] http://ode.apache.org/extensions.

complete. The process model derives automatically from the complete Linear Logic proof, and can then be mapped to a business process modeling language to execute.

Another work proposes a formal model for describing individual RESTful services and automating their composition (Zhao and Doshi, 2009) (Id. 11). A State Transition System (STS) supports automatic composition of such services, based on situation calculus, a logic formalism for representing dynamic domains. Authors classify RESTful services in three types in order to represent them as STS. Resource Set Services represent a set of domain resources and are mapped to a set of ontology instances of the same concept. Individual Resource Services represent the individual resources and are mapped to an ontology instance. Finally, Transitional Services are a special case of transition-oriented RESTful Web Services, i.e., they consume, create, update or transform states from related resources. Transitional services are formally described as transition rules (between resource states) using SWRL (Horrocks et al., May 2004), a formal language based on OWL, and the UML profile RuleML (Boley et al., 2012) (Rule Markup Language). Unlike (Id. 10), which is based on theorem proving, (Id. 11) uses a formal model (STS) which intrinsically represents the HATEOAS principle introduced in Section 2. In this case, the transitions between states of the STS are analogous to hyperlink navigation.

The work in (Riabov et al., 2008) (Id. 12) presents MARIO (Mashup Automation with Runtime Orchestration and Invocation), a tool to simplify data mashup composition. The composition engine allows users to explore a space of potentially composable data mashups and preview composition results as they iteratively refine their "wishes", i.e., mashup composition goals. It generates new flows on demand to match a user' s request. The tool makes automatic composition approachable by maintaining an abstraction of taxonomy-extended tag-based search in the space of existing and generated flows. A formal model based on the Stream Processing Planning Language (Riabov and Liu, 2005) (SPPL) allows the formal definition of the compositional semantics of a flow, capturing inputs/outputs and functional dependencies as planning primitives. SPPL extends the classical planning formalism by adding constructs to represent stream processing problems and automatically build workflows or mashups to solve these problems. Interestingly, planning is a common alternative to automate and formalize composition in SOAP-based systems, but only two approaches (in the scope of this survey) use planning in the context of RESTful composition.

A formal model for RESTful systems is presented in (Zuzak et al., 2011) (Id. 13). Authors identified that, although formal models of hypermedia systems in general do exist, neither of such models cover the fundamentals of REST. Moreover, the models often include unRESTful properties, such as improperly handling application state by overloading the meaning of *state* (application state, resource state) or by updating the state without user interaction. Another concern addressed by the authors is to develop machine-driven clients for RESTful services, rather than the typical human clients that interact with RESTful services through a Web browser. Then, machine-driven clients can lead to machine-to-machine RESTful interactions and compositions. The work uses a Finite State Machine (FSM) formalism for modeling and expressing constraints of RESTful systems. FSMs are a mathematical formalism for describing processes with a finite number of possible states and sequential state transitions. The feasibility of using FSMs follows from one of the core principles of the REST style: resource representation interchange is used for transitioning agents from one state to another, which suggest the suitability of a state transition system formalism. (Id. 13) shows that a non-deterministic FSM with epsilon transitions enables formalization of the application state following the HATEOAS principle. This is similar to (Id. 11) which uses STSs rather than FSMs, where the latter are more restrictive.

A method to formalize context-aware composition of "generic" services is presented in Zhang et al. (2014) (Id. 14). As the authors affirm, the scope of Service Computing should cover all kinds of UDDI-listable services, including both electronic and non-electronic services, e.g., entertainment services such as restaurants, cinemas, etc. In practice, non-electronic services are discovered and composed through RESTful APIs such as Google Places[4] and Foursquare[5]. Context information such as location, budget and time is included as soft constraints to provide good enough solutions when the optimal solution is not feasible. Soft constraints impose a penalty on certain service assignments rather than prohibiting them. Then, the composition problem is treated as a planning and constraint satisfaction problem (Bartak and Salido, 2011), and solved through a Branch and Bound algorithm (Leenen et al., 2008). A Branch and Bound algorithm splits the composition problem into subproblems (e.g., one per context variable) and calculates bounds for the objective function (i.e., the composition goal) that are then combined into various good enough solutions. As in (Id. 12), this approach uses planning to model and solve the composition problem, but including in this case context information as soft constraints.

Based on Category Theory (Awodey, 2010), the work in (Li and Chou, 2014) (Id. 15) presents a language called Categorial Links to define and compose RESTful services for resource-resource compositions. Category Theory is a mathematical theory that studies relations between categories. Authors affirm that current (non-standard) languages developed for REST are designed for client-service interactions, and are not adequate for automatic resource-resource communications. Thus, to allow composition, a uniform surface is needed to describe resources. In this sense, a RESTful system is decomposed into two layers: link category L and resource category R, and the cohesion between the layers is maintained by a functor $F : L \rightarrow R$. Then, composition is mathematically defined as a transitivity law between links that connect resources. Categorial links provide a uniform and extensible tool to model basic, nested and concurrent interactions among RESTful resources driven by a few algebraic operations. This approach extends the hyperlink pipeline concept – which will be discussed in the context of (Id. 20) by formalizing them as categorial links, through Category Theory.

Finally, the approach in Xie et al. (2013) (Id. 16) presents a semantic resource service model (SRSM) to combine structural and transitional semantics for resource-oriented service composition. Entity-oriented resources represent the object which is manipulated by a task or process. Other resources are defined as transition-oriented resources that consume and manipulate other entity resources. The composition is accomplished by reasoning over structural and operational semantics defined for entity-oriented and transition-oriented resources respectively. Structural semantics are used to match the input and output of services (i.e., input/output reasoning) as operational semantics are used to express the functions of services (i.e., pre-condition and effects reasoning). Thus, the composition is solved as a regression problem by finding the state transition sequence (by means of transition-oriented resources) that transfers from the initial state to the desired end state of the entities. The derived resource classification is similar to the one presented in (Id. 11): entity-oriented resources are analogous to individual resources and resource set services, while transition-oriented resources are analogous to transitional services.

[4] http://www.developers.google.com/places.
[5] http://www.developer.foursquare.com/.

4.3. Pure RESTful service composition

The approach in Pautasso (2009) (Id. 17) establishes a set of features that should be accomplished by composition languages for RESTful services: dynamic late binding, dynamic typing, content-type negotiation, state inspection, and compliance with the uniform interface principle. The JOpera visual composition language (formerly presented by the author in Pautasso and Alonso (2005)) provides a composition environment that supports those features. This is achieved by two core parts of the language. An iterative methodology for compositions development with interactive debugging and testing tools, and feedback information to gain knowledge about failures during the execution of composed services. Such information allows inferring an improved composition model.

An approach fully driven by RESTful design principles is presented in Alarcon et al. (2011) (Id. 18). The authors propose a hypermedia-driven approach based on the ad hoc language ReLL (Resource Linking Language) and Petri-Nets. ReLL focuses on hypermedia characteristics and serves as a description language for RESTful services, while Petri-Nets allow modeling and early assessing compositions. RESTful clients discover and decide which link/controls to follow/execute at runtime satisfying the HATEOAS principle. According to the authors, RESTful services consider humans as their principal consumer, explaining the lack of machine-readable descriptions that forces providers to describe their APIs in natural language. In contrast, the authors explore the impact of ReLL descriptions in supporting machines/clients to automatically navigate, retrieve and compose Web resources. As in (Id. 17), this approach proposes an ad hoc language to meet RESTful principles as the authors argue that are not strictly followed by available languages. Besides, the ReLL language provides an automatic support for service discovery and composition, which is out of the scope of JOpera.

The work in (de Vrieze et al., 2011) (Id. 19) presents a lightweight Business Process Modeling (BPM) approach that eases the specification of requirements for process-oriented enterprise mashups. Firstly, the authors identify features that are inherent to process-oriented mashups (excluding data-oriented mashups). Process-oriented mashups fit within the enterprise information system context, providing support for “upgrading” the mashup to usage by colleagues with similar issues. They are simple, flexible applications that solve daily problems; can be created in minutes, hours, or days, and often by end-users. Also, process-oriented mashups can be “situational” in nature, easily customised to meet the unique needs of an individual or situation, by mashing up functionality from different sources to support new insights. Finally, process-oriented mashups often support self-service application development, and they can help make SOA more business-relevant and visible, increasing reuse of services and widgets.

Furthermore, the paper propose a preliminary tool that provides those characteristics. As it is user-oriented, a key part of the business process mashup tool is the control panel that allows users to instantiate new processes, view their current processes, and edit the available process models. After specifying the process parameters, the user can review, and if necessary edit the process model through the process viewer. The current state of implementing process-oriented enterprise mashups is on-going. Unlike previous approaches in this category, (Id. 19) is focused in conceptual modeling of process-oriented mashups rather than implementing the solution in a particular language.

Another lightweight service composition mechanism in Web browsers, called hyperlink pipeline (Li et al., 2013) (Id. 20), allows users to interactively chain distributed atomic Web Services into composite services similar to using Unix pipelines. Although it is designed for end-user service composition, hyperlink pipeline provides advanced features. Unlike consumer-oriented mashup tools that separate design and execution phases, applications deployed through hyperlink pipeline can be changed on the fly, which requires automatic data conversion and pipelining mechanisms. Also, applications designed using a certain tool cannot be ported to another tool without changes, while in this case applications are fully portable as the approach leverages standardized formats. The functional composition framework facilitates recursive hyperlink pipeline construction and execution, based on RESTful services and the HTML5[6] Microdata model. Microdata annotates HTML elements with scoped name/value pairs to customize the available vocabulary, allowing to extend the entities that are representable (through tags) in the HTML hierarchy. The proposed mechanism is implemented based on the Web Real-Time Communication[7] (WebRTC) API, which provides constructs to implement realtime multimedia applications within a Web browser. Unlike (Id. 17) and (Id. 18), this approach relies in existent and standardized models to comply with RESTful principles, namely HTML5 Microdata model and WebRTC.

The *Bite* composition language (Rosenberg et al., 2008) (Id. 21) addresses RESTful service composition into Web-scale workflows. Bite is a lightweight and extensible composition language that uses RESTful services as its main composable entities. The Bite language enables an agile, iterative and community-extensible development approach, by means of an atomic life-cycle model (processes are collections whose members are all running process instances), a lightweight process model (a basic set of predefined language constructs for specifying the flow), a script-like approach (by adopting dynamic data types and allowing extensions in any scripting language), language extensibility (new activity types) and Web-human integration. The basic Bite process model comprises a flat graph containing atomic actions (activities) and links between them. The set of core activities consist of basic HTTP communication primitives for HTTP requests, utility activities for waiting, calling local code, or terminating the flow, and control helpers, such as external choice and loops. A Bite flow both uses external services in its flow logic and exposes itself as a service. The flow approach is similar to the idea of process-oriented mashup in (Id. 19). Moreover, the Bite language could be an interesting alternative to implement the business process mashup tool proposed in (Id. 19).

Business Process Modeling and RESTful *push* services are combined in Pautasso and Wilde (2013) (Id. 22) so that business processes can be modelled and observed in a RESTful way, by publishing process instances and tasks as resources. Traditional Web-style interactions are often described as *pull* (the clients pulling resource state from the server), and thus the complementary functionality is often described as *push*, where resource state changes are pushed to a client. Based on the push notion, clients can subscribe to be notified via Atom feeds when certain states in a business process are reached. Different alternatives can ensure a timely propagation of notifications, such as PubSubHubbub[8] (PuSH) and WebSocket (Fette and Melnikov, 2011) protocols. Traditionally, feeds are a RESTful pull-oriented architecture, and clients need to repeatedly pull feeds to become aware of updates. However, PuSH is a push-oriented protocol based on Atom feeds that allows clients to be notified of feed updates immediately, which makes this approach different to all other approaches within this category. Moreover, the idea of using

[6] http://www.w3.org/TR/html5/.
[7] http://www.w3.org/TR/webrtc/.
[8] https://code.google.com/p/pubsubhubbub/.

Table 8
Summary of advantages and drawbacks of the analyzed proposals (part I).

Group	Id	Advantages	Drawbacks
Heterogeneous service composition	01	Process-based composition language tailored to support the specific properties and constraints of the REST architectural style	Switching to RESTful services is not transparent, contrary to WSDL documents wrapping RESTful services, which can cleanly maintain the existing business process (Peng et al., 2009)
	02	Fully automatic RESTful composition through machine-readable specifications of services	Semantic annotations are needed to specify RESTful services. This requires extra effort from developers, potentially preventing its adoption
	03	Conceives RESTful mashups and SOAP-based compositions as being the same thing from the end users' perspective	Several languages involved may discourage its adoption: WADL for describing RESTful services, WSDL for atomic services, XUL for user interfaces, BPEL for composite services
	04	No need to create and maintain neither a complete BPM model nor a workflow specified with an executable language	Spurious data events can led to inconsistent WED-states, thus a mechanism to handle them properly is needed
	05	Facilitates sharing and reusing of mashups through a highly extensible platform	Limitations inherent to a DSL-approach (non-generic solution, debugging issues, absence of primitives for GUI description)
	06	Accessible, configurable and robust dynamic mashups through re-binding	Requires semantically annotating atomic services to then compose them, augmenting the composition burden
	07	Combines both RESTful and SOAP based services to design composite services in a semi-automatic fashion	MD approaches are very exhaustive and costly. The dynamic nature of RESTful mashups encourages using lightweight approaches as well as lightweight services
	08	The main requirements of REST are explicitly mapped to BPEL constructs that satisfy them as part of the composition language	Handling long running processes is difficult and inefficient as REST lack support to asynchrony. *Apush* mechanism rather than a polling mechanism would be more suitable
	09	Can be transparently used in current standard BPEL 2.0 execution environments over WSDL 1.1 without modification	Generates WSDL descriptions for RESTful services, burying the uniform interface and its various advantages under a RPC-like envelope
Formalization of RESTful composition	10	Automatic composition process. Process calculus enables a straightforward connection to executable business process languages	Requires formal descriptions and semantic annotations for services, often unavailable
	11	Formal description of the RESTful service composition problem. Situation calculus-based state transition system for automatic composition	The proposed typification introduces the overhead of classifying RESTful services prior to their composition. Adequate and meaningful URIs could replace this classification
	12	Iterative definition and refinement of the composition goals, allowing users to explore potentially compatible services on the fly	The problem of finding optimal plans is time consuming. SPPL planners can not guarantee termination in polynomial time
	13	A natural formal model for representing RESTful principles	Models for complex systems are very large due to the number of representations of application states and links. Discontinuation in hyperlinks may introduce failures
	14	Context-awareness, support for generic services and soft constraint satisfaction guarantee "good enough" solutions (i.e., compositions) for most scenarios	The approach is not intended particularly for RESTful services but for 'generic services'. Thus, it is not compliant with the RESTful principles listed in Table 2
	15	A description language focused on resource-resource composition, rather than client-service interaction, which fosters automation	Links between resources are not part of resource representations as in conventional REST. This idea goes against REST HATEOAS principle
	16	Models both entity-oriented resources and transition-oriented resources capturing particular semantics of data and operations	The overhead of the ad hoc semantic resource representation over OWL and XML metadata is not justified by the little reasoning applied in the approach

BPM together with RESTful services is also fostered in (Id. 19), but in a pull-oriented fashion.

The work in Lanthaler and Gutl (2011) (Id. 23) features a Semantic Description Language for RESTful Data Services based on JSON (SEREDASj). JSON (JavaScript Object Notation) is a lightweight data-interchange format based on a subset of the JavaScript[9] standard. According to the authors, the major problem of RESTful services is that no agreed machine-readable description format exists to document them, which hinders their automatic discovery and composition. SEREDASj semantic descriptions allow annotating JSON elements to link them to concepts in an ontology and to further validate them according flexible rules. Semantically annotated data (in the form of JSON elements) is the first step for (semi-)automatic integration with other data sources, i.e., SEREDASj could automate data-oriented mashup development. Semantic annotation of resources is a key aspect of the languages proposed in this approach (SEREDASj) and in (Id. 18) (ReLL).

The approach proposed in Choi (2012) (Id. 24) transforms results of atomic RESTful services into object instances in the Java language. Then, these object instances can interact with one another in a way prescribed by the programmer to solve the composition problem. However, the way in which object instances can interact and then convert the results into a RESTful-compatible representation is not addressed in (Id. 24). Easing the adoption of the composition approach is also a key aspect in (Id. 19), (Id. 20) and (Id. 21), but addressed in these cases through lightweight ad hoc composition languages rather than a well-known paradigm as in (Id. 24).

The work in Sepulveda et al. (2014) (Id. 25) identifies key QoS elements in a RESTful composition as an ontology, particularly for the security domain. These elements serve to model hypermedia-based, decentralized security descriptions. The security ontology is inspired in the WS-Security model for SOAP-based services, but adapted to a REST ecosystem. The three core concepts of the ontology are security goals (confidentiality, integrity and identity authentication), security tokens (hold by service consumers) and protocols (such as OAuth). The security ontology is put on top of ReLL service and composition descriptions. ReLL (resource linking language) presented in (Id. 18) provides automatic discovery and composition capabilities for RESTful services.

A unified approach to discover the interaction patterns offered by a resource and its orchestration possibilities is presented in

[9] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

Bennara et al. (2014) (Id. 26). To accomplish Linked Data principles (Bizer et al., May 2009), authors extend a resource with a RDF descriptor that contains meta-data about the resource together with information about related resources. Then, from the URI of a resource it is possible to get its descriptor too, following a generic interaction pattern. The machine client (a browser plugin) takes the URI of the first resource as input, then it follows links found in the metadata and calls HTTP operations on the right resources with the right data in order to fulfill the user needs. The navigation process is manually done, but a reasoner could automate the task. Web browser based discovery and composition was also discussed in (Id. 20).

The work in Bellido et al. (2013) (Id. 27) analyses and implements fundamental control-flow patterns of RESTful compositions through callbacks and redirections. Particularly, this is done through the extension of the HTTP status codes (3XX). The approach allows composed resources to delegate control flow to different services in a decentralized way through status codes, including sequence, unordered sequence, alternative, iteration, parallel, discriminator, and selection control flows. For example, the sequence code (311) indicates the invocation of a service without any guard condition (sequence control flow), while the alternative code (312) indicates that the client must choose one service from a list of available services to be executed next. As in (Id. 25), services are described by means of ReLL, the ad hoc description and composition language presented in (Id. 18).

The RESTdoc framework (John and Rajasree, 2013) (Id. 28) features a combination of microformat markup and RDF Schemas to describe, discover and compose RESTful services by adding semantics. Services are described using annotations in the HTML code of Web pages, while RDF/RDFS is used to describe relationships (links) between services. This enables either a Web browser driven discovery or an automatic discovery approach, and finally compose services as a graph of resources. However, this last capability is only superficially described. This approach is comparable to other semantic description solutions such as SEREDASj (Id. 23) and ReLL (Id. 18). However, RESTdoc provides a lightweight semantic description embedded in the HTML code of Web pages that contain resources, while the other approaches provide separate representations of semantics as JSON (SEREDASj) or XML (ReLL) documents.

Finally, the last work under analysis regarding pure RESTful composition approaches is presented in (Lu et al., 2015) (Id. 29). RESTfulBP is an architecture that uses REST in business process design, implementation and execution to improve adaptability. RESTfulBP business processes are modelled in a declarative way and reusable process fragments are linked dynamically. Models are described through BPMN plus ad hoc annotations that describe both process fragment constraints (e.g., pre-/post-conditions) and decision-making points where dynamic adaptation may take place. An automatic decision-making module implements business rules and policies as constraints. Such module performs dynamic workflow adaptation and reshaping to maximize the overall business value of the process, by dynamically linking process fragments. Pure RESTful business processes are also discussed in (Id. 19), (Id. 21) and (Id. 22), but in this approach the abstract workflow shape can be dynamically modified by invoking different process fragments.

Table 9
Summary of advantages and drawbacks of the analyzed proposals (part II).

Group	Id	Advantages	Drawbacks
Pure RESTful services composition	17	Recursive composition where composing a set of RESTful services results in another RESTful service, transitively holding the architectural principles	Poor support for content type negotiation, state managent, security and scalability concerns
	18	Fully complaint with the HATEOAS (Hypermedia As The Engine Of Application State) property, a core idea of REST	Ad hoc metamodel. Introduces coupling between clients and servers (as both need to know the mediating metamodel). Generates *static*composite services as a result
	19	Clearly defines process-oriented enterprise applications, with their requirements and the features that they should provide	The process-oriented enterprise mashup system is at a preliminary stage, which is insufficient to demonstrate the suitability of the approach
	20	Allows dynamic, lightweight and user-guided composition supported by well-known and standardized models and APIs	Client-side service description and hyperlink compatibility tests imply extra burden which is not always acceptable for low-end clients
	21	Flexible, agile developing language for Web applications. Provides collaborative facilities to build enterprise-scale mashups	The Bite language does not give a direct representation for some of the REST interaction primitives (e.g., the PUT verb is not supported)
	22	Push support for RESTful interactions, avoiding HTTP long polling through lightweight Atom feeds and the PuSH protocol	Push interactions hinders understandability of the approach, having different macro and micro roles in the interaction at the same time
	23	Allows loosely coupled machine-to-machine communication based on RESTful services automating interactions	The suitability of the semantic description language is not empirically demonstrated or deeply discussed in the context of RESTful service composition
	24	Allows regular object-oriented programmers to compose services	Supports only XML-formatted output for RESTful services, does not address the re-conversion from objects execution results to the output of a composite service
	25	Provides a Security Model upon REST that supports the most widely used security approaches for the Web, inspired in the WS-Security standard for SOAP	The ad hoc metamodel introduces coupling between clients and servers. QoS is considered but only directed towards security
	26	The resource header is imbued with a lightweight semantic description that eases automatic service discovery and composition with little overhead	The descriptors only provide naïve information about resources, which is easily deductible. The composition task is still manually performed
	27	Decentralized implementation of control-flow patterns holds most desirable RESTful properties for compositions	The use of an ad hoc language goes against standardization making the approach less interoperable
	28	Reuses existing documentation of services making it machine-readable through annotations	Composition is merely mentioned as an extension of discovery. Even though RESTful properties are claimed to be satisfied, discussion about this is very shallow
	29	Runtime, semi-automatic decision-making support for concurrent adaptation of multiple RESTful business process instances	The adaptation handler is fairly complex. This complexity is only justified with a high number of similar users running workflow instances in parallel

5. Discussion

When analyzing the RESTful service composition approaches presented in Section 4, we identified different key aspects, which are discussed in this section. As a summary, Tables 8 and 9 collect the advantages and disadvantages of each analyzed approach.

5.1. Agreement to standards

Standards have been key enablers of SOAP-based Web Services adoption and deployment in the past (Curbera et al., 2002). However, when it comes to RESTful service composition, there is no robust and common agreement on standards yet. Thus, it becomes difficult to engage composition solutions, while standards remain narrowly adopted: only 20% of the analyzed approaches (6/29) embrace consolidated standards. In certain cases, some standards may be referred but the implementation is based on stand-alone solutions.

Several initiatives have been conducted to provide standards that enable adoption of RESTful services. For instance, WADL and hRESTs have been incepted for creating functional descriptions, which eases discovery and composition; or MicroWSMO and SA-REST for semantically annotating services, which allows machine-to-machine automatic interoperation. However, none of them gained broad support so far to become an agreed standard (Lanthaler and Gutl, 2010; Peng et al., 2009; Pautasso, 2009; Nacer and Aissani, September 2014). This is due to the fact that RESTful approaches are less mature than SOAP-based approaches. Also, the proposed standards for machine-readable RESTful services descriptions have not been widely adopted so far. Machine-readable descriptions allow to interpret the service documentation to then automatically build powerful discovery and composition methods. In contrast, SOAP-based services are experiencing an increasing adoption of semantic descriptions, such as OWL-S, to automate discovery and composition activities. In consequence, RESTful services are almost exclusively oriented to human-readable documentation describing the URIs and the data expected as service inputs and outputs (Lanthaler and Gutl, 2011; Immonen and Pakkala, 2014).

The standardization process itself reflects the different decision criteria that academia, vendors and users have. According to different authors, the standardization process also involves a social dimension (Zur Muehlen et al., 2005; Vinoski, 2007). Academic organizations run and support non-profit standardization ventures such as W3C, which grants standards adopters with royalty-free licenses of use. The economic interest, the sense of influence and the ownership of software vendors as standards-submitting parties may be affected by these policies. Thus, software vendors tend to discuss their proposals in other organizations that they control, such as OASIS. Moreover, industry users of standards join together and run a different standardization process, which is centered on the business rather than the technology. For industry users, the agreement is more important than the technical details, and can overcome any implementation inefficiencies. That is the case of certain domain-oriented standards such as SWIFT (for the financial industry), HL7 (for the healthcare industry) and ACORD (for the insurance industry) (Zur Muehlen et al., 2005). In the case of RESTful services, there is no prevailing organization leading the standardization process yet, neither from the industry nor the academia. However, these RESTful services are still maturing, and following a natural process of practice dictating the upcoming standards (Adamczyk et al., 2011).

5.2. Impact of legacy composition views

The current interest in Web Services is focused on issues that have a long history in the workflow community (Zur Muehlen et al., 2005). Several RESTful composition approaches have actually adopted the workflow view alone or in combination with the mashup view, totaling 35% of the analyzed approaches. Notably, the majority of these approaches belong to the Heterogeneous Service Composition category (presented in Section 4.1). The most common solution for integration and interoperability between RESTful and SOAP-based services has been bringing RESTful services to the well-known field of workflow development. Likewise, it is not a coincidence that a business-oriented language such as BPEL had been selected in most cases for representing the workflow view in practice.

Another interesting finding is the low presence of the choreography view among RESTful service composition approaches – only 2 recent approaches presented choreography view. Choreographies track the message sequences between multiple Web Services seen from the perspective of multiple parties. However, RESTful composition is still seen from a one-party perspective: a consumer (human or machine) navigating hyperlinks instead of a composition of collaborating services without a centralized control. This evidences the preliminary evolution stage at which RESTful service composition currently is w.r.t. global integration. The developer has to manually generate integration layers between services to build up the composition, which forces a centralized control. The efforts to automate this process with, for instance, semantic annotations (as in Id. 25, Id. 26), contribute to decentralization of RESTful compositions, to eventually lead to RESTful choreographies in a multiple-vendors service environment (Lanthaler and Gutl, 2010).

5.3. Takeoff from data-oriented mashups

In the early years of RESTful mashups they were mainly data-oriented (Sheth et al., 2007). This means that mashups were composed by piping one service output into the next service input while filtering content and making slight format changes. This approach limits the number of atomic services that can interact into a mashup application. Typically, they deal with in-house services of an organization, or services that have common outputs (such as RSS or Atom). Nowadays, the transition to RESTful process-oriented mashups implies a takeoff from traditional mashups, overcoming the data-level integration problem where a 50% of the mashup approaches proposed a process-oriented mashup solution for the RESTful composition problem. Process-oriented mashups bring RESTful services to the enterprise, solving both business and IT challenges. This is particularly useful for small, medium and virtual enterprises that have less resources to create heavyweight BPM solutions – e.g., with SOAP-based services and BPEL or other workflow approaches (de Vrieze et al., 2011). However, these scenarios present challenges that mashups are still immature to deal with, such as security, authorisation and QoS. In contrast, these challenges are addressed by the different WS-* protocols for SOAP-based services (Vinoski, 2007).

5.4. Tradeoffs between complexity of materialization and interoperability

Figure 3 plots the tradeoff between complexity of materialization (*x*-axis) and interoperability (*y*-axis) of composition approaches. Heavyweight approaches are rigorous, formal and standard-oriented, fostering interoperability. However, they tend to be costly to deploy and use, which hinders their adoption. On the other hand, lightweight approaches can be built with minimal

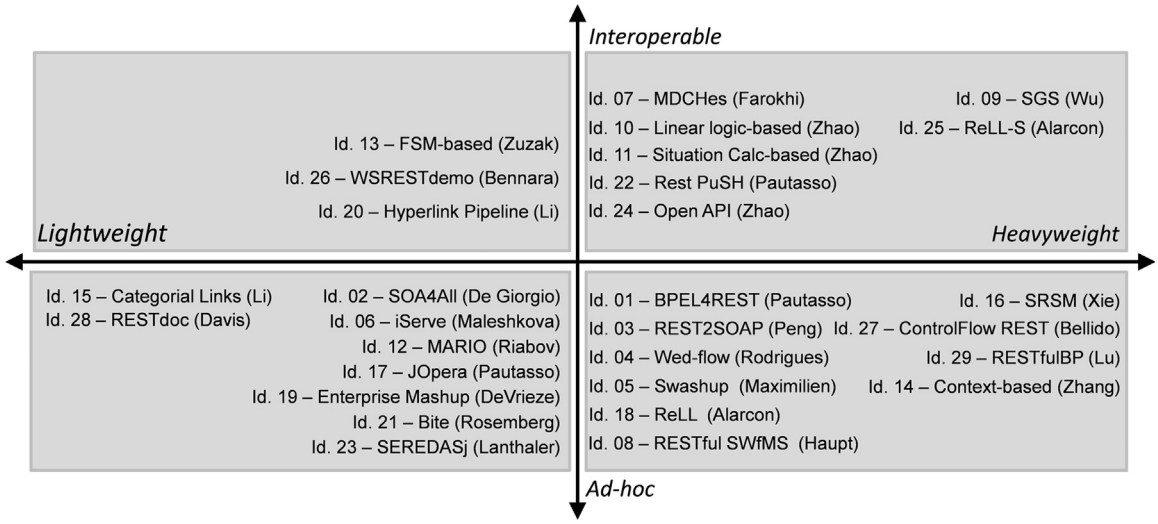

Fig. 3. Plot of analyzed approaches according to two key dimensions: complexity of materialization and interoperability.

tooling, at low cost and with a low adoption barrier, but tend to be ad hoc in terms of materializing technology, less maintainable, small-scaled and restricted (Immonen and Pakkala, 2014). Although most of the analyzed approaches are ad hoc, the trend shows that composition solutions that pursuit large-scale interoperability are heavyweight. In terms of architectural decisions (Pautasso et al., 2008), this tradeoff could be decomposed and assessed at different technological dimensions of a RESTful system:

- *Payload format*: The developer may choose among a variety of MIME types for representing resources. This can hinder the interoperability of a RESTful service, as – for example – clients expecting data in a JSON format will not be able to parse a XML payload. Also, a RESTful service capable of retrieving resources in multiple representation formats requires more maintenance effort. However, the benefit of preferring a lightweight format such as JSON over XML can outweigh the extra effort and the lack of interoperability with a significant reduction in network overhead and parsing effort. Meanwhile, the agreement over a payload format can foster interoperabiltiy in the future.
- *Service description*: RESTful services have adopted a human-oriented approach based on informal, textual descriptions, giving developers extensive documentation of the API of the provided service. This is neither lightweight nor interoperable. However, developers can use a variety of languages to describe services, ranging from small-sized interfaces described in LPML and microformats (lightweight), to complex interface descriptions in BPEL4REST and WADL (heavyweight).
- *Service composition*: RESTful Web services can be composed using the WSDL-based invocation abstractions provided by, for instance, WS-BPEL (Overdick, 2008). Although this enables REST/SOAP interoperability, the lack of formally described interfaces for RESTful services and the possibility of not always using XML messages can make the composition process cumbersome. The lightweight alternative for composition of RESTful services are mashups, which are transitioning from their usage in ad hoc data-level integration to enterprise-level integration, as discussed in Section 5.3. However, the current maturity stage of mashups still fits better for ad hoc integration over the Web.

Other technological dimensions such as the transport protocol (HTTP), security (HTTPS over SSL) and service identification (URIs) do not provide alternatives so far to balance the tradeoff between complexity of materialization and interoperability.

5.5. REST principles adherence

As shown in Table 7, many of the analyzed solutions do not fully adhere to REST principles: only two approaches satisfy all the four properties. The average number of satisfied properties is 2 out of 4.

Lightweightness was discussed in Section 5.4, as a property which has to be balanced with the interoperability of RESTful composition approaches. This property is fulfilled by 41% of the analyzed approaches.

Some of the analyzed approaches heavily rely on a RPC (Remote Procedure Call) operation-based model ignoring the *understandable* and *declarative* fundamental architectural properties of REST. Instead of focusing on describing resource representations they fall into an RPC-like model describing the inputs and outputs for operations, which results in tight coupled, unclear and procedural descriptions of services. Consequently, these approaches do not align well with the RESTful services design principles (Lanthaler and Gutl, 2011). From the analyzed approaches, a 48% and 52% are understandable and declarative, respectively.

With regard to *scalability*, RESTful services are, by definition, highly scalable. The Web itself is the best example, since it is a worldwide REST-based architecture (Fielding, 2000). RESTful composition approaches built upon this architecture should scale in the same way. However, many of these solutions do not even mention this issue. Some authors barely acknowledge scalability as a RESTful property, and thus the proposed composition approaches lack it: only 31% of them address scalability. It can be expected that in the following years, RESTful approaches will put more emphasis on this issue while they exploit lightweight, understandable and declarative interfaces as well.

5.6. Support for full automation

Only 7% of the analyzed approaches support full automation of the composition process, while 41% provide semi-automatic mechanisms. Although automation can be seen as a desirable feature, many proposals provide facilities for manual or semi-automatic composition by end users, thus full automation is out of their scope. Manual composition frameworks assist the user to

sketch the composition either graphically or through a text editor, such as (Id. 17) and (Id. 20). Semi-automated composition techniques make suggestions for service selection during the composition process, such as (Id. 06) and (Id. 12).

5.7. Dynamic definition and binding for flexible composition

Dynamic service composition enables for more flexible and adaptive applications. This seems to be in line with the REST principles that pursuit simplicity and flexibility, with 52% of the analyzed approaches providing dynamic or hybrid service composition. On the other hand, static service composition suits better for designing complex interaction patterns often present in B2B applications which are easy to foresee at design time. Approaches that bring RESTful services to business scenarios in the form of process-oriented mashups are mainly static (or hybrid). However, recent approaches for REST-enabled business processes show a trend in dynamic binding and runtime adaptation.

5.8. Ad hoc service and service composition specifications

This feature identifies which specifications a composition system supports for sketching either required services or compositions. Interestingly, a high number of the analyzed approaches (62%) developed their own ad hoc specifications for either atomic services or compositions. This suggests that researchers do not consider suitable any of the existing languages to describe services and compositions. Probably, existing languages do not capture the requirements of RESTful service composition, which is another indicator that there is room for improving such languages.

5.9. Adaptation support

According to the literature (Benatallah et al., 2005; Eslamichalandar et al., 2012; Motahari Nezhad et al., 2007; Cardellini et al., 2012), adaptation can be performed from a concrete composition perspective and/or an abstract composition perspective. In the former case, in turn, interface adaptation addresses incompatibilities in service operation signatures, protocol adaptation addresses behavioural mismatches, and workflow re-binding implies changing atomic services of a composition upon detecting context changes, QoS infringements or component failures. In the present survey, 35% of the approaches addressed adaptation at some extent from the concrete perspective. Incompatibilities at interface level do not affect RESTful compositions as long as the uniform interface constraint is satisfied: their interaction is prescribed, at syntactic level, to the HTTP operations. Adaptation from an abstract composition perspective (i.e., changing the composition shape at runtime) is only addressed by one approach, particularly through runtime process fragment swapping in the context of RESTful business processes (Lu et al., 2015), which also comprises adaptation from a concrete perspective (workflow re-binding). The majority of the approaches are not concerned with adaptation as a key issue yet. Nevertheless, as RESTful services are fine-grained, the required adapters may be simple and easy to generate automatically.

5.10. Support for static/dynamic Verification and Validation

The Verification and Validation feature is addressed by a 35% of the analyzed approaches. Verification checks the maintenance of certain desirable properties of the composite service with respect to the individual composed services. Five of the V&V-centered approaches provide verification upon a formal basis such as Finite State Machines, State Transition Systems or Planning. Other approaches use runtime monitoring and testing to ascertain properties of the composition. Recently, dynamic quality verification have been seen as more and more crucial in development of service-oriented systems (Immonen and Pakkala, 2014). However, Verification and Validation remain unexplored in RESTful composition. These topics could be addressed by extending the aforementioned techniques and applying them considering the dynamic approach to composition fostered by REST: it may become difficult to statically verify properties of a composition at design-time. Thereby, testing at run-time becomes critical to check properties of the composite service while managing the independent evolution of the underlying RESTful services (Pautasso, 2009).

5.11. QoS-awareness

Notably, only 17% of the analyzed approaches mentioned Quality of Service or Non-functional properties, albeit superficially. RESTful services today ignore QoS requirements; their main concern is providing functional interfaces (Adamczyk et al., 2011). Machine-readable specifications for external services describing functionality or QoS are still unrealistic. A mature Web Service composition platform should support the specification, monitoring and dynamic agreement of QoS (Garriga et al., 2015). As a first step, a commonly agreed language for describing the QoS parameters and a mechanism to incorporate the description in the HTTP payload is needed. A standard QoS description language, at least for atomic RESTful services, might come from ontologies and semantic descriptions proposed by the Semantic Web community, such as SA-REST (Sheth et al., 2007) or SWEET (Maleshkova et al., 2009). These descriptions define common ways of interpreting information, such as QoS parameters, enabling all clients to interpret them the same way. QoS ontologies have been suggested recently on the basis of the specification benefits of ontologies in general. However, the solutions are diverse, which reveals the lack of standardization (Immonen and Pakkala, 2014). Despite ontologies, quality policies (ISO, 2008) can be used to generate quality objectives, and they also serve as a general framework for QoS-awareness. Moreover, model-driven approaches can define QoS in any of the Platform Independent Models (PIMs), as outlined in the Input Model of (Id. 07), where QoS is represented as constraints, e.g., $availability > 70\%$ or $responseTime < 0.5$ s. These QoS notions for atomic RESTful services could be extended in the future to support QoS-aware RESTful service composition.

In contrast, SOAP-based approaches provide QoS-aware discovery and composition mechanisms, either by leveraging typical service descriptions (Al-Masri and Mahmoud, 2007) (represented by the WSDL, UDDI and SOAP triad), by adding a semantic QoS description in the form of ontologies, or by representing the composition as an optimization problem (Strunk, 2010). QoS-aware RESTful service composition might benefit from the work in SOAP-based composition as well.

6. Future research possibilities

As we pointed out in the previous section, the analysis leads to different research challenges in RESTful service composition. However, it is still unclear how and in which order researchers could address these challenges. In this section, we draw a parallelism with the evolution of SOAP-based service composition, in an attempt to foresee the evolution of RESTful service composition. We hypothesize that the latter could evolve in a similar way, but the evolution of RESTful service composition has been taking place with a difference of 5–10 years w.r.t. its close relative. Section 6.1 looks back at the challenges in early years of SOAP-based service composition, and their current state in RESTful service composition. Section 6.2 points out current, open challenges in SOAP-based

service composition and their correlation with RESTful service composition as well.

6.1. Early days of SOAP service compositions – how to define and automate a service composition

In the early years of SOAP-based service composition, according to different surveys (Rao and Su, 2004; Srivastava and Koehler, 2003) the literature mainly focused on two aspects, discussed below.

Definition of clear/standard steps for Web Service composition: Although sometimes named differently, the commonly agreed steps for Web Service composition mainly consisted in:

- modeling the abstract composition, where a process generator (either machine or human) tries to solve a certain complex functional requirement by composing simpler, abstract functionalities;
- composing (binding) abstract functionality from this composition model to concrete services, where the process generator takes the functionalities of services as input, and outputs a set of selected atomic services and the control/data flow among these services in the process model;
- executing the composition, which basically consists in sending a sequence of messages between concrete services according to the process model. The data flow of the composite service is defined as the transfer of output data from an executed atomic service to the input of an atomic service to be executed; and finally
- verifying the composition flow according to different properties or by their overall utilities based on the non-functional attributes provided by the process generator.

These aspects are still under discussion for RESTful composition, where some efforts consider humans as the principal consumer/composer for RESTful services (Alarcon et al., 2011; Rosenberg et al., 2008), which explains the lack of machine-readable descriptions, and the massification of user-driven composition approaches. However, other authors claim that the absence of an agreed machine-readable description format is a major problem, which severely hinders the definition of techniques for automatic discovery and composition (Lanthaler and Gutl, 2010; Nacer and Aissani, September 2014).

Classification and characterization of Web Service composition solutions: On one side, industry and business solutions are mainly workflow-based, where the interface description of Web Services is typically defined in WSDL. Service interactions and message exchange are described in a standard business protocol specification language (such as BPEL4WS), which specifies the allowed roles and message exchange schemes. The industry approach looks at composite services mainly from the perspective of runtime functions, data and control flow. On the other side, academic solutions are mainly based upon the Semantic Web and planning notions from the AI field. Service developers annotate service capabilities, inputs (pre-conditions) and outputs (post-conditions) in some semantic language such as OWL-S, relying on ontologies to formalize the domain concepts shared among services. Then, given a goal description, a planner outlines the appropriate plan for composing published Web Services.

This classification seems to hold for RESTful services as well. Process-oriented mashups (de Vrieze et al., 2011), extended business composition languages (such as BPEL4REST (Pautasso, 2009)) and RESTful compositions with an alike workflow view align with the notions found in SOAP-based compositions used in the industry and business solutions. In contrast, RESTful composition approaches based on semantic annotations (Lanthaler and Gutl, 2011), planning (Riabov and Liu, 2005) and formalization (Zhao et al., 2011) align with academic solutions from the SOAP side.

6.2. Present days of SOAP service compositions – how to achieve adaptive, scalable, trustworthy service composition approaches

Nowadays, the issues related to defining, classifying and characterizing composition solutions mentioned in the previous section are assumed as solved. However, according to the literature (Sheng et al., 2014; Issarny et al., 2011) new open issues have arisen in SOAP-based service compositions because of the challenges posed by new usage scenarios/research areas which were born in the last years, such as the Internet of Services, Mobile Computing and Pervasive Computing. Some of them are partially solved by current approaches, but need further improvement. Also, some advantages of RESTful services such as lightweightness and scalability can be exploited in this context.

Correctness: In recent years, model checking and verification of SOAP-based service compositions have become active research topics. However, more research is needed for developing novel solutions and tool sets for correct services composition. As seen in Section 5.10, this aspect also remains rather unexplored for RESTful compositions.

Malleability: Nowadays, the environment in which composite services are developed and executed has become more open, dynamic, and ever changing. This raises several malleability issues including self-configuring, self-optimizing, self-healing, and self-adapting. This may involve devices with limited resources and computational capabilities. Thus, the algorithms for designing and dynamically adapting the compositions need to be efficient. In this sense, RESTful services could be the way ahead because of their lightweightness and simplicity.

Pervasiveness: Composing services across multiple mobile devices presents new challenges from that of traditional services composition settings. In particular, composition in pervasive environments must address context awareness, heterogeneity, contingencies of devices, and personalization. In this context, further research in RESTful services could provide interesting solutions due to their inherent technological portability. A pervasive environment also raises the need of adopting appropriate semantic technologies, shared standards and mediation, which are required to assure semantic interoperability (Guizzardi, 2005; Nacer and Aissani, September 2014) of heterogeneous entities such as mobile devices, sensors, and networks.

Security support Apart from functional aspects, non-functional composition properties concerning security and trust are crucial for the adoption of composition technologies. Security issues must be considered to adequately handle critical data from users. Several specifications for SOAP-based Web Service security have been proposed as a part of the WS-* stack, such as WS-Security, WS-Trust and WS-Federation, although none of them has been broadly adopted. In contrast, RESTful composition only supports HTTPS (HTTP over SSL) as the default security mechanism so far.

Scalability: In a new environment which empowers users who are now becoming “prosumers” (Issarny et al., 2011) (i.e., both producers and consumers), it is still unclear how to combine the need to aggregate several services, maintain their QoS, and keep the composition coupling level as low as possible. In the context of REST, notions of enterprise mashups could be exploited to compose RESTful services at large, while QoS have not been addressed yet. Additionally, the coupling level increases as different participant RESTful services have different URIs which are not directly interchangeable.

6.3. Conclusions

RESTful service composition and mashups are in an earlier stage of evolution compared to SOAP-based composition. They are transitioning from the mere description and classification of services and compositions to more advanced features such as those already addressed for SOAP-based services. This is evidenced by the challenges of RESTful services composition discussed so far, which still include:

- an explicit, machine-readable description format to represent RESTful services, ranging from a Microformat or RDF tagging, to more complex ontologies such as OWL-S, which will allow machine reasoning upon RESTful services and compositions. However, dealing with ontologies is already hard, as the heterogeneity problem has now moved to the ontology abstraction level itself – i.e., it is also a problem about ontological foundations (Guizzardi, 2005). Meanwhile, those semantic descriptions for RESTful services must also cover REST principles. This can led to tradeoffs, as discussed for the lightweight principle in Section 5.4;
- a composition language tailored for RESTful services, which allows users to construct a composite RESTful service and effectively deal with the dynamic, heterogeneous nature of RESTful services; and
- a state management method, intended to handle the state of the application which is not attributable to RESTful services since they are stateless by definition.

We believe that open standard agreement is the basic prerequisite for achieving high interoperability and compatibility, being a key issue to be addressed. A standardized service composition approach can assure compatibility with any third-party service and achieve greater re-usability. Finally, standardized composite services can collaborate with partners for better data portability.

7. Final remarks

In this paper, we have presented a survey of recent proposals in the newborn field of RESTful service composition. We employed two sets of features to analyze, compare and contrast current composition approaches: eight general features applicable to both RESTful and SOAP-based composition approaches, and four specific features inherent to REST properties.

RESTful composition approaches are fairly new but in our view a significant maturation is expected in the following years. To the best of our knowledge, the first approaches appeared in 2008 and their number is gradually increasing year by year. Meanwhile, RESTful service composition is experiencing a transition to a robust and holistic framework. In addition, recent efforts have demonstrated the potential of integrating both SOAP-based and RESTful Web Services in a proper new service ecosystem or "Internet of Services", in which machine readable descriptions allow automatic discovery, composition and communication of collaborative services (Lanthaler and Gutl, 2010). The adoption of SOA principles also allows for decomposing complex and monolithic systems into ecosystems of simpler and well-defined services (Schroth and Janner, 2007). The use of principles such as common interfaces and standard protocols gives a horizontal view of an enterprise system (Atzori et al., 2010).

In addition, while analyzing the proposals we found different perspectives in the "REST vs. SOAP" debate. However, according to different authors, the alleged debate seems meaningless, since RESTful and SOAP-based services have different objectives, are better suited for certain contexts, and can even be used in conjunction (Vinoski, 2007; Pautasso et al., 2008; Farokhi et al., 2012). Individually, RESTful services can be more scalable, reliable and visible, and better fitting for ad hoc integration at Internet-scale. However, composition approaches built upon RESTful services may lack some of these properties – e.g., scalability. SOAP-based services can be used for enterprise level application integration where QoS, reliability and message-level security are critical (Pautasso et al., 2008; Lanthaler and Gutl, 2010). Recent efforts in large-scale legacy system migration to services have demonstrated the suitability of SOAP-based technologies and standards (Rodriguez et al., 2013). In this sense, both perspectives could benefit each other: incorporating concepts from SOAP-based composition will endow the Web 2.0 platform with a powerful and highly usable integration paradigm; while REST architectural principles – and the Web platform as a whole – may enrich the enterprise community by integrating RESTful services with traditional back-end systems (Rosenberg et al., 2008).

However, RESTful services still suffer from shortcomings on semantically describing, finding and composing services as well as the absence of a holistic framework covering the entire service lifecycle. The main reason for these issues is the lack of an agreed standard to materialize RESTful services and compositions. None of the proposed approaches or their satellite languages have gained broad support so far (Lanthaler and Gutl, 2010). Consequently, it is expected that research efforts in the area will focus on addressing these issues, which are indeed critical for the success of RESTful services and compositions.

Acknowledgements

We acknowledge the financial support provided by grants PICT 2012-0045 (ANPCyT), and 04/F001 (UNComa). We would like to thank the anonymous reviewers for their helpful comments to improve the quality and scope of this work.

References

Adamczyk P, Smith PH, Johnson RE, Hafiz M. REST and Web Services: in theory and in practice. REST: from research to practice. New York, USA: Springer; 35–57.

Alarcon R, Wilde E, Bellido J. Hypermedia-driven RESTful service composition. In: Maximilien E, Rossi G, Yuan S-T, Ludwig H, Fantinato M, editors, Service-oriented computing. Lecture notes in computer science, vol. 6568. Berlin, Heidelberg: Springer; 2011. p. 111–20. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-3-642-19394-1_12⟩.

Al-Masri E, Mahmoud QH. Qos-based discovery and ranking of web services. In: Proceedings of 16th international conference on computer communications and networks (ICCCN). IEEE; 2007. p. 529–34. Honolulu, Hawaii.

Atzori L, Iera A, Morabito G. The internet of things: a survey. Comput Netw 2010;54 (15):2787–805.

Awodey S. Category theory. New York, USA: Oxford University Press; 2010.

Bartak R, Salido M. Constraint satisfaction for planning and scheduling problems. Constraints 2011; 16(3): 223–7. [Online]. Available from: ⟨http://dx.doi.org/10.1007/s10601-011-9109-4⟩.

Bellido J, Alarcón R, Pautasso C. Control-flow patterns for decentralized RESTful service composition. In: ACM Transactions on the Web vol. 8(December (1)), 2013. p. 5:1–30. [Online]. Available from: ⟨http://doi.acm.org/10.1145/2535911⟩.

Benatallah B, Casati F, Grigori D, Nezhad H, Toumani F. Developing adapters for web services integration. In: Pastor O, Falcao e Cunha J, editors, Advanced information systems engineering. Lecture notes in computer science, vol. 3520. Berlin, Heidelberg: Springer; 2005. p. 415–29. [Online]. Available from: ⟨http://dx.doi.org/10.1007/11431855_29⟩.

Bennara M, Mrissa M, Amghar Y. An approach for composing RESTful linked services on the web. In: Proceedings of the 23rd international conference on World Wide Web Companion (WWW Companion); 2014. p. 977–82. [Online]. Available from: ⟨http://dx.doi.org/10.1145/2567948.2579222⟩.

Benslimane D, Dudstar S, Sheth A. Service mashups: the new generation of web applications. IEEE Internet Comput 2008;12(5):13–5.

Berbner R, Spahn M, Repp N, Heckmann O, Steinmetz R. Heuristics for QoS-aware web service composition. In: Proceedings of the international conference on web services (ICWS). IEEE; 2006. p. 72–82, Chicago, USA.

Bizer C, Heath T, Berners-Lee T. Linked data – the story so far. Int J Semant Web Inf Syst May 2009;5(3):1–22.

Bojanic P. The joy of XUL ⟨https://developer.mozilla.org/en-US/docs/The_Joy_of_XUL⟩; 2003.

Boley H, Athan T, Paschke A, Tabet S, Grosoft B, Bassiliades N, et al. RuleML version 1.0; April 2012. ⟨http://ruleml.org/1.0/⟩.

Bozkurt M, Harman M, Hassoun Y. Testing and verification in service-oriented architecture: a survey software testing. Verif Reliab 2013;23(June(4)):261–313.

Canfora G, Di Penta M, Esposito R, Perfetto F, Villani M. Service Composition (re) binding driven by application-specific qos in service-oriented computing (ICSOC). In: Dan A, Lamersdorf W, editors, Lecture notes in computer science, vol. 4294. Berlin, Heidelberg: Springer; 2006. p. 141–52. [Online]. Available from: ⟨http://dx.doi.org/10.1007/11948148_12.

Cardellini V, Casalicchio E, Grassi V, Iannucci S, Lo Presti F, Mirandola R. MOSES: a framework for QoS-driven runtime adaptation of service-oriented systems. IEEE Trans Softw Eng 2012;38(September–October (5)):1138–59.

Cardoso J, Sheth A, Miller J, Arnold J, Kochut K. Quality of service for workflows and web service processes. Web Semant: Sci Serv Agents World Wide Web 2004;1(3):281–308.

Chakraborty D, Joshi A. Dynamic service composition: state-of-the-art and research directions. Technical Report TR-CS-01-19, University Of Maryland, Baltimore; December 2001.

Choi M. RESTful web service composition. In: Park JJJH, Leung VC, Wang C-L, Shon T, editors, Future information technology, application, and service. Lecture notes in electrical engineering, vol. 164. Netherlands: Springer; 2012. p. 569–76. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-94-007-4516-2_58⟩.

Crasso M, Rodriguez JM, Zunino A, Campo M. Revising WSDL documents: why and how. IEEE Internet Comput 2010;14(5):48–56.

Crasso M, Zunino A, Campo M. A survey of approaches to web service discovery in service-oriented architectures. J Database Manag 2011;22(January–March (1)):102–32.

Curbera F, Duftler M, Khalaf R, Nagy W, Mukhi N, Weerawarana S. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Comput 2002;6(2):86–93.

Daniel F, Pernici B. Web service orchestration and choreography: enabling business processes on the web in e-business models, services and communications. In: Lee I, editor. IGI global; 2008. p. 250–73 chapter 12.

De Giorgio T, Ripa G, Zuccala M. An approach to enable replacement of SOAP services and rest services in lightweight processes. In: Current trends in web engineering. Daniel F, Facca FM, editors, Lecture notes in computer science, vol. 6385. Berlin, Heidelberg: Springer; 2010. p. 338–46. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-3-642-16985-4_30⟩.

de Vrieze P, Xu L, Bouguettaya A, Yang J, Chen J. Building enterprise mashups. Future Gener Comput Syst 2011;27(5):637–42.

Eslamichalandar M, Barkaoui K, Motahari Nezhad HR. Service composition adaptation: an overview. In: Proceedings of the international workshop on advanced information systems for enterprises (IWAISE). IEEE; 2012. p. 20–7. Constantine, Algeria.

Farokhi S, Ghaffari A, Haghighi H, Shams F. MDCHeS: model-driven dynamic composition of heterogeneous services. Int J Commun Netw Syst Sci 2012;5(September (9A)):644–60.

Fette I, Melnikov A. The websocket protocol; 2011. [Online]. Available from: ⟨http://www.hjp.at/doc/rfc/rfc6455.html⟩.

Fielding R. Architectural styles and the design of network-based software architectures [Ph.D. dissertation]. University of California, CA, USA; 2000.

Fujii K, Suda T. Dynamic service composition using semantic information. In: Proceedings of the 2nd international conference on service-oriented computing (ICSOC). ACM; 2004. p. 39–48. New York, USA.

Garriga M, Flores A, Cechich A, Zunino A. Web services composition mechanisms: a review. IETE Tech Rev 32 (5), 2015, 376-383.

Guizzardi G, Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology; Enschede, The Netherlands; 2005.

Hadley MJ. Web application description language (WADL). Technical Report SMLI TR-2006-153, Sun Microsystems, Inc., Mountain View, CA, USA; 2006.

Haupt F, Fischer M, Karastoyanova D, Leymann F, Vukojevic-Haupt K, Service composition for REST. In: Proceedings of the 18th international enterprise distributed object computing conference (EDOC). IEEE; September 2014. p. 110–19. Ulm, Germany.

Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M, et al. SWRL: a semantic web rule language combining OWL and RuleML W3C consortium. W3C Members Submission SWRL-20040521 May 2004.

Immonen A, Pakkala D. A survey of methods and approaches for reliable dynamic service compositions. Serv Oriented Comput Appl 2014;8(2):129–58.

I.O. for Standardization Commitee, ISO 9001:2008 quality management systems – requirements; 2008. ⟨https://www.iso.org/obp/ui/#iso:std:iso:9001:ed-4:v2:en⟩.

Issarny V, Georgantas N, Hachem S, Zarras A, Vassiliadist P, Autili M, et al. Service-oriented middleware for the future internet: state-of-the-art and research directions. J Internet Serv Appl 2011;2(1):23–45.

John D, Rajasree M. RESTdoc: describe, discover and compose RESTful semantic web services using annotated documentations. Int J Web Semant Technol 2013;4(January (1)):37–49.

Kitchenham B, Pfleeger SL. Software quality: the elusive target. IEEE Softw 1996;13(1):12–21.

Kopecky J, Gomadam K, Vitvar T. hRESTs: an HTML microformat for describing RESTful web services. In: Proceedings of the international conference on web intelligence and intelligent agent technology (WI-IAT), vol. 1. IEEE/WIC/ACM; 2008. p. 619–25. Sidney, Australia.

Lanthaler M, Gutl C. Towards a RESTful service ecosystem. In: Proceedings of the 4th international conference on digital ecosystems and technologies (DEST). IEEE; 2010. p. 209–14. Dubai, UAE.

Lanthaler M, Gutl C, A semantic description language for RESTful data services to combat semaphobia. In: Proceedings of the 5th international conference on digital ecosystems and technologies conference (DEST). IEEE; 2011. p. 47–53. Daejeon, South Korea.

Leenen L, Ghose A. branch and bound algorithms to solve semiring constraint satisfaction problems. In: Ho TB, Zhou ZH, editors, Proceedings of the trends in artificial intelligence conference (PRICAI). Lecture notes in computer science, vol. 5351. Springer: Berlin, Heidelberg; 2008. p. 991–7.

Li L, Chou W, Categorial link: REST service composition based on category theory. In: International conference on web services (ICWS). IEEE; June 2014. p. 431–8. Alaska, USA.

Li L, Chou W, Cai T, Wang Z. Hyperlink pipeline: lightweight service composition for users. In: International joint conferences on web intelligence (WI) and intelligent agent technologies (IAT). IEEE; Atlanta, USA; 2013. p. 509–14.

Liu D, Li N, Pedrinaci C, Kopecky` J, Maleshkova M, Domingue J. An approach to construct dynamic service mashups using lightweight semantics. In: Harth A, Koch N, editors, Current trends in web engineering. Lecture notes in computer science, vol. 7059. Springer, 2012; p. 13–24. Berlin, Germany.

Lu Q, Xu X, Zhang W, Zhu L, Li S. Business-driven process fragment selections in RESTful business processes. Int J u- e- Serv Sci Technol 2015; 8(1): 173–188. [Online]. Available from: ⟨http://dx.doi.org/10.14257/ijunesst.2015.8.1.16⟩.

Majithia S, Walker D, Gray W. A framework for automated service composition in service-oriented architectures. In: Bussler C, Davies J, Fensel D, Studer R, editors, The semantic web: research and applications. Lecture notes in computer science, vol. 3053. Berlin, Heidelberg: Springer; 2004. p. 269–83. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-3-540-25956-5_19⟩.

Maleshkova M, Pedrinaci C, Domingue J. Semantically annotating RESTful services with sweet. In: Proceedings of the 8th international semantic web conference (ISWC); 2009. p. 25–9. [Online]. Available from: ⟨http://oro.open.ac.uk/23102/ ⟩.

Martin D, Burstein M, McDermott D, McIlraith S, Paolucci M, Sycara K, et al. Bringing semantics to web services with owl-s. World Wide Web 2007;10(3):243–77.

Mateos C, Crasso M, Zunino A, Ordiales Coscia J. Revising WSDL documents: why and how-part 2. IEEE Internet Comput 2013;17(5):46–53.

Maximilien M, Ranabahu A, Gomadam K. An online platform for web APIs and service mashups. IEEE Internet Comput 2008;12(5):32–43.

Motahari Nezhad HR, Benatallah B, Martens A, Curbera F, Casati F, Semi-automated adaptation of service interactions. In: Proceedings of the 16th international conference on World Wide Web (WWW). ACM; 2007. p. 993–1002. Seoul, Korea.

Nacer H, Aissani D. Semantic web services: standards, applications, challenges and solutions. J Netw Comput Appl 2014;44(September):134–51.

Nakajima S. Model-checking Verification for reliable web service. In: Proceedings of the workshop in object-oriented web services (OOWS). ACM; 2002. Seattle, USA.

Narayanan S, McIlraith S. Simulation, Verification and automated composition of web services. In: Proceedings of the 11th international conference on World Wide Web (WWW). ACM; 2002. p. 77–88. New York, USA.

Overdick H. Towards resource-oriented bpel. In: Emerging web services technology, vol. 2. Springer; 2008. p. 129–40. Basel, Switzerland.

Paik I, Chen W, Huhns M. A scalable architecture for automatic service composition. IEEE Trans Serv Comput 2012;7(January (1)):82–95.

Pautasso C, Wilde E. Push-enabling RESTful business processes. In: Kappel G, Maamar Z, Motahari-Nezhad H, editors, Service-oriented computing. Lecture notes in computer science, vol. 7084. Berlin, Heidelberg: Springer; 2011. p. 32–46. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-3-642-25535-9_3⟩.

Pautasso C, Alonso G. The JOpera visual composition language. J Vis Lang Comput 2005;16(1):119–52.

Pautasso C, Zimmermann O, Leymann F. RESTful web services vs. "big" web services: making the right architectural decision. In: Proceedings of the 17th international conference on World Wide Web (WWW). ACM Press; 2008. p. 805–14. New York, USA.

Pautasso C. RESTful Web service composition with BPEL for rest. Data Knowl Eng 2009;68(9):851–66.

Pautasso C. Composing RESTful services with JOpera. In: Bergel A, Fabry J, editors, Software composition. Lecture notes in computer science, vol. 5634. Berlin Heidelberg: Springer; 2009. p. 142–59. [Online]. Available from: ⟨http://dx.doi.org/10.1007/978-3-642-02655-3_11⟩.

Pautasso C, On composing RESTful services. In: Leymann F, Shan T, van den Heuvel WJ, Zimmermann O, editors, Proceedings of the dagstuhl software service engineering seminar; 2009. [Online]. Available from: ⟨http://drops.dagstuhl.de/opus/volltexte/2009/2043⟩.

Pedrinaci C, Liu D, Maleshkova M, Lambert D, Kopecky J, Domingue J. iServe: a linked services publishing platform. In: Ontology repositories and editors for the semantic web workshop at the 7th extended semantic web conference (ESWC), vol. 596. 2010.

Peenikal S. Mashups and the enterprise mphasis. HP White Paper; September 2009.

Peltz C. Web services orchestration and choreography. IEEE Comput 2003;36(10):46–52.

Peng Y, Ma S, Lee J. REST2SOAP: a framework to integrate SOAP services and RESTful services. In: Proceedings of the international conference on service-oriented computing and applications (SOCA). IEEE; 2009. p. 1–4.

Rao J, Su X. A survey of automated web service composition methods. In: Proceedings of the international workshop on semantic web services and web process composition (SWSWPC); 2004. p. 43–54.
Rauf I, Iqbal M, Malik Z. UML based modeling of web service composition – a survey. In: Proceedings of the international conference on software engineering research (SERA). IEEE; 2008. p. 301–7. Prague, Czech Republic.
Riabov A, Liu Z. Planning for stream processing systems. In: Proceedings of the national conference on artificial intelligence. AAAI; 2005. p. 1205–10. Pittsburgh, USA.
Riabov AV, Boillet E, Feblowitz MD, Liu Z, Ranganathan A. Wishful search: interactive composition of data mashups. In: Proceedings of the 17th international conference on World Wide Web (WWW). ACM; 2008. p. 775–84. New York, USA.
Richardson L, Ruby S.RESTful web services. O'Reilly Media, Inc.; 2008. Sebastopol, USA.
Rodrigues MC, Ferreira JE, Pu C, Web services composition through data events approach. In: Proceedings of the international conference on services computing (SCC). IEEE; 2013. p. 320–327. Santa Clara, USA.
Rodriguez JM, Crasso M, Mateos C, Zunino A. Best practices for describing, consuming, and discovering web services: a comprehensive toolset. Softw: Pract Exp 2012;43(6):613–39.
Rodriguez JM, Crasso M, Mateos C, Zunino A, Campo M. Bottom-up and top-down COBOL system migration to web services: an experience report. IEEE Internet Comput 2013;17(2):44–51.
Rosenberg F, Curbera F, Duftler M, Khalaf R. Composing RESTful services and collaborative workflows: a lightweight approach. IEEE Internet Comput 2008;12 (5):24–31.
Schroth C, Janner T. Web 2.0 and SOA: converging concepts enabling the internet of services. IT Prof 2007;9(3):36–41.
Sepulveda C, Alarcon R, Bellido J. QoS aware descriptions for RESTful service composition: security domain World Wide Web 18 (4), 2015, 767-794.
Sheng QZ, Qiao X, Vasilakos AV, Szabo C, Bourne S, Xu X. Web services composition: a decade's overview. Inf Sci 2014;280(0):218–38.
Sheth A, Gomadam K, Lathem J. SA-REST: semantically interoperable and easier-to-use services and mashups. IEEE Internet Comput 2007;11(6):91–4.
Singh M. Being interactive: physics of service composition. IEEE Internet Comput 2001;5(3):6–7.
Srivastava B, Koehler J, Web service composition – current solutions and open problems. In: Proceedings of the workshop on planning for web services (ICAPS), vol. 35; 2003. pp. 28–35.
Strunk A. QoS-aware service composition: a survey. In: Proceedings of the 8th European conference on web services (ECOWS). IEEE; 2010. p. 67–74. Aiya Napa, Cyprus.
Un P, Genevski P, Gorroñogoitia Y, Radzimski M, Ripa G, Mos A, Norton B, et al. SOA4All project deliverable: D6. 3.3 evaluation and final design of the lightweight context-aware process modeling language European commission. Technical Report FP7- 215219, SAP; 2010.
Vinoski S. REST eye for the SOA guy. IEEE Internet Comput 2007;11(1):82–4.
Weerawarana S, Curbera F, Leymann F, Storey T, Ferguson DF. Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable messaging and more. Upper Saddle River, NJ, USA: Prentice Hall PTR; 2005.
Wu B, Lin R, Chen J, Integrating RESTful services into BPEL business process on service generation system. In: Proceedings of the international conference on services computing (SCC). IEEE; June 2013. p. 527–34. Santa Clara, USA.
Xie C, Cai H, Jiang L. Ontology combined structural and operational semantics for resource-oriented service composition. J Univers Comput Sci 2013;19(July (13)):1963–85.
Yu Q, Liu X, Bouguettaya A, Medjahed B. Deploying and managing web services: issues, solutions, and directions. VLDB J 2008;17(3):537–72.
Zhang Y, Wang J, Yan Y, Context-aware generic service discovery and service composition. In: International conference on mobile services (MobServ). IEEE; June 2014. p. 132–139. Alaska, USA.
Zhao H, Doshi P. Towards automated RESTful web service composition. In: Proceedings of the international conference on web services (ICWS). IEEE; 2009. p. 189–96. Los Angeles, USA.
Zhao X, Liu E, Clapworthy G, Ye N, Lu Y. RESTful web service composition: extracting a process model from linear logic theorem proving. In: Proceedings of the 7th international conference on next generation web services practices (NWeSP). IEEE; 2011. p. 398–403. Salamanca, Spain.
Zimmerman O, Tomlinson M, Peuser S. Perspectives on web services – applying SOAP, WSDL and UDDI to real-world projects. 1st ed. Berlin, Germany: Springer-Verlag; 2003.
Zur Muehlen M, Nickerson J, Swenson K. Developing web services choreography standards – the case of REST vs. SOAP. Decis Support Syst 2005;40(1):9–29.
Zuzak I, Budiselic I, Delac G. Formal modeling of RESTful systems using finite-state machines. J Web Eng 2011;10(4):346–60.