Information Sciences 280 (2014) 218–238

Contents lists available at ScienceDirect

Information Sciences

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

Web services composition: A decade's overview

Quan Z. Sheng [a,*], Xiaoqiang Qiao [b,*], Athanasios V. Vasilakos [c], Claudia Szabo [a], Scott Bourne [a], Xiaofei Xu [d]

[a] School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia
[b] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[c] Department of Electrical and Computer Engineering, National Technical University of Athens, Greece
[d] School of Computer Science and Technology, Harbin Institute of Technology, China

ARTICLE INFO

Article history:
Received 12 July 2013
Received in revised form 9 February 2014
Accepted 10 April 2014
Available online 20 May 2014

Keywords:
Web services composition
Composition life cycle
Composition requirement
Service composition challenge

ABSTRACT

Service-oriented computing (SOC) represents a paradigm for building distributed computing applications over the Internet. In the past decade, Web services composition has been an active area of research and development endeavors for application integration and interoperation. Although Web services composition has been heavily investigated, several issues related to dependability, ubiquity, personalization, among others, still need to be addressed, especially giving the recent rise of several new computing paradigms such as Cloud computing, social computing, and Web of Things. This article overviews the life cycle of Web services composition and surveys the main standards, research prototypes, and platforms. These standards, research prototypes, and platforms are assessed using a set of assessment criteria identified in the article. The paper also outlines several research opportunities and challenges for Web services composition.

1. Introduction

Since the late 1990s, service-oriented computing (SOC) has emerged as an important computing paradigm and changed the way software applications are designed, delivered and consumed. In SOC, services are used as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments [13,60,90]. SOC relies on service-oriented architecture (SOA) to organize software applications and infrastructures into a set of interacting services. SOA establishes an architectural model that allows services to be published, discovered, and consumed by applications or other services, the goal of which is to realize loosely coupled, standard-based and platform-independent distributed computing [60].

Web services technology is the most promising choice to implement service oriented architecture and its strategic objectives. A Web service is essentially a semantically well-defined abstraction of a set of computational or physical activities involving a number of resources, intended to fulfill a customer need or a business requirement. A Web service could be described, advertised and discovered using standard-based languages, and interacted through Internet-based protocols. Today, two types of Web services are most popular and widely used: SOAP-based Web services and RESTful Web services

* Corresponding authors.
E-mail addresses: qsheng@cs.adelaide.edu.au (Q.Z. Sheng), qxq@otcaix.iscas.ac.cn (X. Qiao), vasilako@ath.forthnet.gr (A.V. Vasilakos), claudia.szabo@adelaide.edu.au (C. Szabo), scott.bourne@adelaide.edu.au (S. Bourne), xiaofei@hit.edu.cn (X. Xu).

http://dx.doi.org/10.1016/j.ins.2014.04.054
0020-0255/

[62]. The former is based on Web Services Description Language (WSDL)[1] and Simple Object Access Protocol (SOAP),[2] while the latter conforms to the REST architectural principles [24].

With the technology of Web services, enterprises are able to expose their internal business processes as services and make them accessible via the Internet. Nowadays, companies such as Google, Amazon, Twitter, and Facebook have offered Web services to provide simple access to some of their resources, enabling third-parties to combine and reuse their services. According to several Web services publication websites, such as `ProgrammableWeb`,[3] `WebServiceList`,[4] and `WSIndex`,[5] Web services have experienced an exponential increase in popularity and usage in the past few years. According to a recent statistics from `seekda.com`, there are 28,606 Web services available on the Web, offered by 7739 different providers. Moreover, the rapid adoption of Cloud computing, social networks, and Web of Things further accelerate the increase of available Web services on the Internet [12,42,21]. Obviously, Web services will continue to play a central role for modern software development.

One key challenge for SOA and Web services technology is services composition.[6] For service oriented distributed applications, services are the basic blocks and applications are realized by the interoperations among services. The true capacity of SOA can only be achieved through composing multiple services into more capable and powerful applications. Web services composition allows organizations to form alliances, to outsource functionalities, and to provide one-stop shops for their customers. From a business perspective, services composition dramatically reduces the cost and risks of building new business applications in the sense that existing business logics are represented as Web services and could be reused.

Over the last decade, Web services composition has become a thriving area of research and academic efforts, with a substantial body of research work produced. The aim of this article is to survey these existing major techniques, research prototypes, and standards on Web services composition and identify open research challenges in this area. Several surveys that focus on Web services composition have been published such as [22,67,83,51,6]. In [22], the authors overview a number of available Web services composition approaches based on simple classification criteria such as manual vs. automated and static vs. dynamic. In [67], the authors discuss Web services composition methods according to the level of automation, and [83,51,6] focuses on various Web services composition standards. Almost all of these surveys were conducted at the earlier stage of services composition (before 2007) and recent advancements in this technology were not covered. Additionally, these surveys overview and analyze services composition technologies only from specific perspectives and lack a holistic view of Web services composition. The aim of this work is to provide a better understanding of the current research issues and activities in this area.

In this paper, we first present the life cycle of Web services composition. This life cycle consists of four phases and for each phase, we identify a set of assessment criteria, which are used as a benchmark to study related research prototypes. 30 research prototypes that cover the most representative efforts in Web services composition in the last decade are compared using these dimensions. Major Web services composition standards and platforms are also discussed and compared. Based on our analysis, several future research challenges and possible directions on Web services composition are discussed.

The remainder of the article is organized as follows. In Section 2, we introduce some basic concepts and terminologies related to Web services composition. In Section 3, we present the life cycle of Web services composition and identify a set of requirements for each phase. In Section 4, we overview and compare representative standards for Web services composition. Then in Section 5, we examine representative research prototypes that support Web services composition and evaluate them using the proposed requirements. In Section 6, we also discuss and assess some major Web services composition platforms. In Section 7, we highlight some future directions for Web services composition research and development. Finally, we provide some concluding remarks in Section 8.

2. Overview of Web services composition

Web services composition is the process of aggregating multiple services into a single service in order to perform more complex functions. In this section, we introduce some basic concepts and terminologies related to Web services composition.

2.1. Web services

The term *Web service* is used very often nowadays, although not always with the same meaning. Existing definitions of Web services range from the very generic and all-inclusive to the very specific and restrictive. A Web service is often seen as an application accessible to other applications over the Web. This is a very open definition, under which almost everything that has a Universal Resource Locator (URL) is a Web service.

A more formal definition provided by IBM is that Web services are "a new breed of Web application, and they are self-contained, self-describing, modular applications that can be published, located and invoked across the Web". This definition

[1] http://www.w3.org/TR/wsdl.
[2] http://www.w3.org/TR/SOAP.
[3] http://www.programmableweb.com/.
[4] http://www.webservicelist.com/.
[5] http://www.wsindex.org/.
[6] In the rest of this paper, we will use the terms Web services composition and services composition interchangeably.

is more detailed, placing emphasis on the need to be open, which essentially means that a Web service has to be published and can be located and invoked over the Web.

The W3C[7] (World Wide Web Consortium) defines a Web service as "a software system identified by a Universal Resource Identifier (URI), whose public interfaces and bindings are defined and described using eXtensible Markup Language (XML). Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols". The W3C definition stresses that Web services should be able to be *defined*, *described*, and *discovered* and clearly mentions the requirements of *Internet-oriented*, *standards-based* interfaces. In other words, Web services are components that can be integrated into more complex distributed applications.

There are generally three roles involved in Web service applications: *service provider*, *service registry*, and *service requester*. The interactions between these three roles are *publish*, *find* and *invoke/bind*. Web services are implemented and published by service providers. They are discovered and invoked by service requesters. Information about a Web service (i.e., service descriptions) is kept within a service registry.

2.1.1. SOAP-based vs. RESTful Web services

There are two main ways for developing Web services: the traditional *SOAP-based* Web services and conceptually simpler, *RESTful* Web services [62].

"SOAP-based" Web services, also called WS∗ Web services, depend on three important standardization initiatives, i.e., WSDL, SOAP, and the Universal Description, Discovery, and Integration (UDDI).[8] Service registration, discovery and invocation are implemented by SOAP calls. SOAP-based Web services are protocol independent and stateful, but demand more computation resources, especially when handling SOAP messages. SOAP-based Web services are typically used to integrate complex enterprise applications.

By contrast, "RESTful" Web services utilize the REST model. REST stands for Representational State Transfer, which was introduced as an architectural style for building large-scale distributed hypermedia systems [24]. RESTful Web services are identified by URIs, which offer a global addressing space for resource and service discovery. RESTful Web services interact through a uniform interface, which comprises a fixed set of operations in the context of the Web and the Hypertext Transfer Protocol (HTTP): GET, PUT, DELETE and POST. Services interact by exchanging request and response messages, each of which includes enough information to describe how to process the message. In contrast to SOAP-based Web services, RESTful Web services are lightweight and stateless, which are well suited for tactical, ad hoc integration over the Web. A popular technique is *mashup* that enables users to create situational applications based on existing application components [57].

2.1.2. Atomic vs. composite Web services

We distinguish between *atomic* and *composite* Web services. An atomic service (also called *elementary service* [72]) is an access point to an application that does not rely on another Web service to fulfill user requests. Each atomic service provides a programmatic interface based on SOAP and WSDL. For legacy applications such as those written in CORBA, appropriate adapters can be developed so that they can be invoked as Web services.

A composite service [7,16,1] is an umbrella structure that brings together other composite and atomic services that collaborate to implement a set of operations. The services brought together by a composite service are referred to as its *component services*. An example of a composite service would be a travel preparation service, integrating services for booking flights, booking hotels, searching for attractions, etc.

Whether atomic or composite, a Web service is specified by an *identifier* (e.g., URL), a set of *attributes*, and a set of *operations*. The attributes of a service provide information, which is useful for the service's potential consumers.

2.2. Web services composition

In this subsection, we identify several concepts and issues related to Web services composition. Firstly, we introduce the concepts of Web service orchestration and choreography. Secondly, we distinguish between static and dynamic composition. Finally, we discuss automated services composition and manual services composition.

2.2.1. Orchestration vs. choreography

The standard set of Web service technologies (XML, SOAP and WSDL) provides the means to describe, locate, and invoke a Web service as an entity in its own right. Although a Web service may expose many operations, each WSDL file describes fairly atomic, low-level functions. What the basic technologies do not give us is the *rich* behavioral detail that describes the role the service plays as part of a larger, more complex collaboration [39,75].

There are two ways to describe the sequence of activities that make up a business process: service *orchestration* and service *choreography* [13,63]. Service orchestration represents a single executable business process that coordinates the interaction among the different services, by describing a flow from the perspective and under control of a single endpoint.

[7] http://www.w3.org/.
[8] http://www.uddi.org.

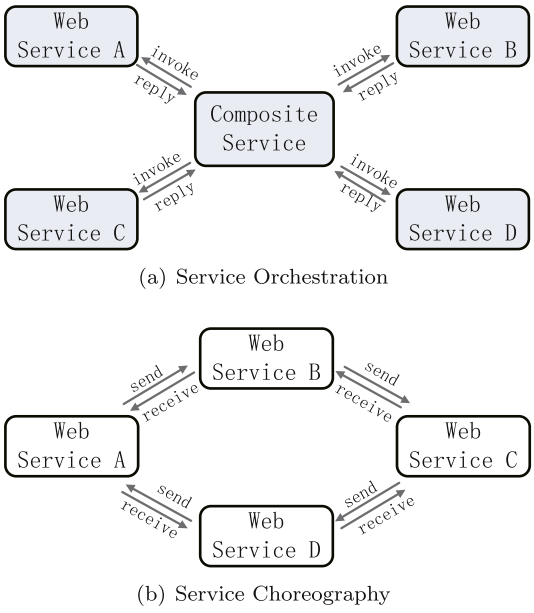

Fig. 1. Service orchestration and service choreography.

Orchestration can therefore be considered as a construct between an automated process and the individual services that enact the steps in the process. Orchestration includes the management of the transactions between the individual services, including any necessary error handling, as well as describing the overall process. The standard for Web services orchestration is WS-BPEL[9] (or BPEL in short), which is largely supported by the industry.

Service orchestration always represents control from one party's perspective (see Fig. 1(a)). This differs from service choreography, which is more collaborative and allows each involved party to describe its part in the interaction (see Fig. 1(b)). Choreography represents a global description of the observable behavior of each of the services participating in the interaction, which is defined by public exchange of messages, rules of interaction and agreements between two or more business process endpoints. Choreography is typically associated with the interactions that occur between multiple Web services rather than a specific business process that a single party executes. The choreography mechanism is supported by the standard WS-CDL[10] (Web Services Choreography Description Language).

2.2.2. Static vs. dynamic composition

Based on the time when Web services are composed, services composition could be categorized into *static* or *dynamic*. In a static composition, the aggregation of services takes place at design time. Service components required for composition are chosen, bound together, and then deployed. Static Web services composition works well if the business partners involved in the process are relatively fixed, and service functionalities or composition requirements do not, or rarely, change. Static composition is not flexible and adaptable in cases when there are frequent runtime changes of requirements or services that cannot be predicted at design time. Once an early binding service becomes unavailable, or if there is a better alternative service, static composition will not be able to provide better support for the execution of composite service in real-time.

In contrast, a dynamic Web services composition allows determining and replacing service components during runtime. Dynamic Web services composition requires the execution system to support automatic discovery, selection, and binding of service components. Undoubtedly, dynamic composition is ideal as the Web services environment is highly dynamic in nature. However, dynamic services composition is a very challenging task and a number of important issues need to be considered such as composition correctness, time limits, transactional support and so on [31,71].

2.2.3. Manual, semi-automated, and automated composition

Composite services typically involve complex business processes, which may include multiple tasks as well as interactions among these tasks (e.g., control and data flow, transaction dependencies). Ad-hoc development of composite services is time-consuming, inflexible, cumbersome, and error prone, and is therefore hardly applicable because of the volatility and size of the Web. Current efforts in Web services composition can be generally grouped into three categories: *manual*, *automated*, and *semi-automated* [51,71]. By manual composition, a human designer (i.e., service provider) should create an

[9] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[10] http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217.

abstract composite process, utilizing some Web service standard language, like BPEL or OWL-S. Then, the designer binds the Web services to the abstract process manually, which is a non-trivial work. This kind of composition is a time-consuming and error-prone procedure with no guarantee that the execution result will indeed satisfy the user's requirements.

In contrast, automated services composition is a sort of "service semantic integration system". Automated services composition approaches typically exploit the semantic Web [11] and artificial intelligence (AI) planning techniques. By giving a set of component services and a specified requirement (e.g., user's request), a composite service specification can be generated automatically [10]. However, realizing a fully automated services composition is still very difficult and presents several open issues [9,48]. The basic weakness of most of research efforts proposed so far is that Web services do not share a full understanding of their semantics, which largely affects the automatic selection of services.

There exist some research efforts that leverage manual and automated compositions. These methods aim to assist the user at each step of the services composition procedure. For example, in [80,4], the authors propose model-driven approaches for Web services composition. A completed executable service specification (e.g., BPEL) can be generated from the composite service specification (e.g., UML activity model, protocol specifications and interface). In [73], composite services are specified as *process schema* and the component services are selected at runtime based on the non-functional properties (e.g., QoS parameters) and constraints specified by the user.

3. Requirements on Web services composition

In this section, we discuss the life cycle of Web services composition, which is divided into four phases. For each phase, a set of requirements is identified. These requirements will be used as a benchmark to evaluate and analyze existing research prototypes on Web services composition in Section 5.

3.1. The life cycle of Web services composition

As illustrated in Fig. 2, the life cycle of Web services composition includes four phases: the *Definition* phase, the *Service Selection* phase, the *Deployment* phase, and the *Execution* phase.

- *Definition phase.* During this phase, the service requester specifies the services composition requirement, which should provide enough information about the user requirements and preferences for the composite service. The requirement is then decomposed, either semi-automatically or automatically, into an abstract process model (i.e., the *abstract composite service*), which specifies a set of activities, the control and data flow among them, the Quality of Service (QoS) requirements, and the exceptional behaviors.
- *Service selection phase.* In this phase, for each activity in the composite service, suitable Web services that match the activity's requirements are located by searching the service registry, based on information contained in the published service description documents. It is likely that more than one candidate service will meet the requirements. Therefore, the best matched service needs to be selected. After all the required Web services are identified and bound to the corresponding activities, the *constructed composite service* is produced.

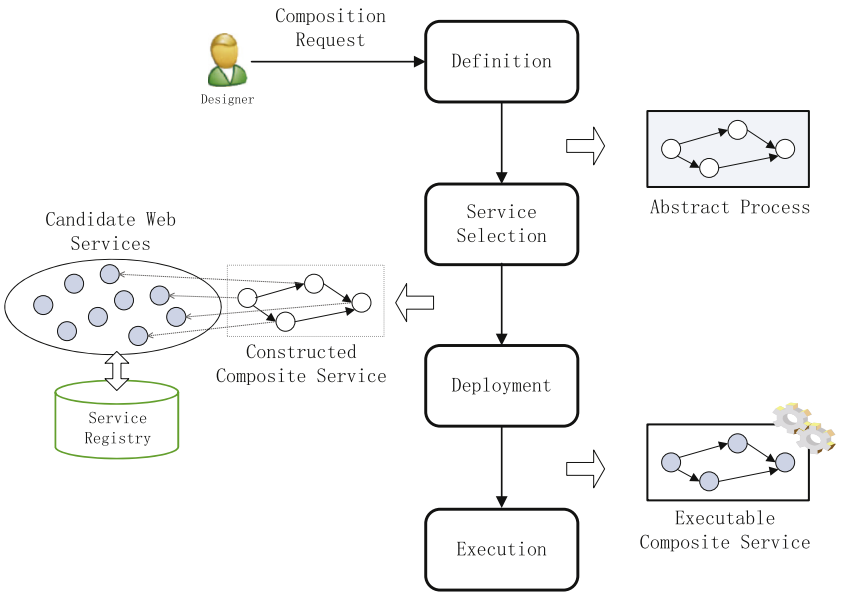

Fig. 2. The life cycle of Web services composition.

- *Deployment phase.* In this phase, the *constructed composite service* is deployed to allow its instantiation and invocation by end users. The result of this phase is the *executable composite service*.
- *Execution Phase.* In this phase, the composite service instance will be created and executed by the execution engine, which is also responsible for invoking the individual service components. During the execution of the composite service instance, the monitoring tasks, including logging, execution tracking, performance measuring and exception handling, should be performed.

It should be noted that for some automated composition methods, such as [46,79], the first two phases are merged where the *constructed composite service* is directly generated according to the composition requirements without creating the *abstract composite service*.

3.2. Requirements

Considering the highly dynamic and distributed nature of Web service environments, some requirements should be met in order to achieve successful Web services composition. In this section, we identify several requirements for each phase of the Web services composition life cycle.

3.2.1. Definition phase

- *Expressibility.* Expressibility refers to the expressive power of a process modeling language. A highly expressive process model should have the following six capabilities: (1) modeling the interactions between the composite service and participated service components, (2) modeling complex structures such as sequence, choice, concurrency and iteration, (3) allocating activities to respective roles, (4) representing the data and specify the data flow among activities, (5) specifying temporal constraints, including startup time, deadline, waiting time, etc., and (6) supporting exception handling and transactional features.
- *Correctness.* The design of Web services composition is an error-prone task. Like other distributed systems, it is hard to guarantee that the composite services will behave as intended. Because runtime errors will lead to unanticipated behaviors and cause significant loss, it is important to detect and fix design errors before the actual deployment of composite services [70]. Correctness refers to the ability to ensure that the composite service conforms to its functional requirements. There are two levels of correctness issues. At the first level, it allows verifying whether the behavior of the composite service is correct during execution (e.g., there is no deadlock). At the second level, it allows verifying the composite service have certain requested properties (safety and liveness properties) [55].

3.2.2. Service selection phase

- *Automation.* Service discovery is the process of finding suitable service(s) for a given activity, which is the prerequisite of service selection. Service discovery is usually based on matchmaking between a request query and available Web services' descriptions. Automation support is regarded as the cornerstone to provide effective service discovery in large and dynamic environments. There are two ways to implement service discovery: *syntactic matching* and *semantic matching*. The syntactic matching methods only offer search facilities based on names and identifiers, in which case the matchmaking accuracy is low and limited automation support is provided. On the other hand, in semantic based methods, Web services are supplied with semantic descriptions of many aspects (e.g., operations, inputs, outputs, and even capabilities and behavior), to increase the level of automation in Web services discovery.
- *Selectability.* The result of service discovery could be a set of services with similar (or identical) functionality but different non-functional characteristics. Web services composition systems should have the ability to select the service with the best QoS among the candidate services. This comprises two fundamental questions: how to describe non-functional attributes and how to select the most appropriate services. The W3C working group defines various QoS attributes for Web services, which include *performance*, *reliability*, *scalability*, *capacity*, etc. There are two types of methods in the service selection literature, namely *Optimization-based* and *Negotiation-based*. Optimization-based method assumes a predetermined set of QoS attributes (optimization could be performed at local and global level respectively) [92], while negotiation-based method permits QoS attributes to be flexible [87].

3.2.3. Execution phase

- *Adaptability.* As mentioned above, Web service environments are highly dynamic where new Web services may come online at any time, existing services could be removed or become temporarily unavailable, and the content and capabilities (e.g., QoS attributes) of services could be changed. On the one hand, a Web services composition system should support dynamic service binding or even automatic replacement of selected component services with new services offering better QoS at runtime. On the other hand, business environments are also highly dynamic where changes may occur rapidly. Therefore, the services composition system should provide composite services that can dynamically adjust their behaviors to meet the changes of customer requirements and to incorporate new business policies.

- *Scalability.* Composing two Web services is not the same as composing 50 or 100 services. In practice, complex composite services (e.g., enterprise applications) may involve many Web services, possibly several hundred [51]. Therefore, one critical issue is how the proposed composition approaches *scale* with the number of component services. Scalability can be evaluated from the execution of composite services. There are generally two different kinds of approaches: *centralized* and *decentralized* [7]. In the centralized mode, a single execution engine is responsible for the invocation of composite services, including the enforcement of control flows and message exchanges among component services. In the distributed mode, the component services participating in a composite service interact with each other to ensure that the control and data flow dependencies expressed in the specification of composite service are respected.
- *Monitoring.* Runtime monitoring of Web services composition has been widely acknowledged as a significant problem [5]. Once the composite service is executed, its progress needs to be managed and monitored to gain a clear view on how the composite service and its component services perform within the execution environment. Moreover, properties and requirements that are verified at design time can be violated at runtime. Web services composition monitoring involves several distinct activities including: (1) logging and viewing execution details of the composite service and component service instances, (2) obtaining QoS statistics by analyzing execution data of the composite service and component services (e.g., response time, throughput, and availability), and (3) verifying the functional requirements as well as evaluating the non-functional properties of the composite service.
- *Reliability.* Reliability refers to the degree of robustness of the services composition system against exceptional behaviors during the execution of the composite service. Runtime exceptions for the invocation of composite services could be vital and are not acceptable for many important Web applications – for example, mission-critical applications – and therefore, should be avoided [71]. Reliability entails handling and recovering from exceptions generated within the composite service as well as faults generated by the interactions with other component services. During the execution of a composite service, the selected service that executes an activity of the composite service may become unavailable because it is overloaded or its respective server is unavailable. In addition, the execution of a service might take longer than the estimated time and might even fail. The main exception recovery techniques in Web services composition include retrying the failed service, compensating the failed service, and executing an alternative service. Web services composition should also have transactional support, which guarantees that composite services have consistent outcomes. Traditional ACID (atomicity, consistency, integrity, and durability) properties are too restrictive for the loosely coupled and distributed services composition environment, and usually a compensation mechanism is adopted to achieve transaction management for Web services composition [40].

3.2.4. Overall requirements

There are some requirements that should be considered during the whole composition procedure, including *Tool support* and *Personalization*, which we elaborate below.

- *Personalization.* Personalization is an important requirement for providing composite services in highly dynamic and distributed environments. The pervasiveness of the Internet offers the possibility to interact with services anytime and anywhere. For example, business travelers now expect to be able to access their corporate servers, enterprise portals, e-mail, and other collaboration services while on the move. Since the requirements and preferences of users are varied, it is essential that services composition approaches enable a *user-centric* and *context-aware* service provisioning [76]. Users should be able to express contextual preferences and constraints regarding the specification and enactment of services compositions (e.g., where and when they wish to interact with the services, how to select component services).
- *Tool support.* Modeling abstract composite services, finding suitable services from a set of candidate services, and managing and monitoring the execution of the composite service, could be made easier by using appropriate tools. Tool support is therefore a significant aspect to access the usability of the services composition system.

4. Standardization efforts

Over the years, efforts have been underway to define standards for composing Web services. In this section, we overview and compare these standardization efforts. Due to the large number of existing standards, we are not going to give an exhaustive survey, but focus on several representative efforts.

4.1. Overview of services composition standards

- **WS-BPEL (Web Services Business Process Execution Language).** WS-BPEL, or BPEL in short, is an XML language for Web services composition. In BPEL, the composition result is called a *process*, participating services are called *partners*, and message exchange or intermediate result transformation is called an *activity*. BPEL introduces several types of primitive activities to: (1) allow for interaction with the applications being composed (`invoke`, `reply`, and `receive`), (2) wait for some time (`wait`), (3) copy data from one place to another (`assign`), (4) indicate error conditions (`throw`), (5) terminate the entire composition instance (`exit`), or (6) do nothing (`empty`).
  These primitive activities can be combined into more complex ones using *structured* activities (also called constructs) provided by BPEL such as `sequence`, `while`, and `flow`. One particular construct offered by BPEL is *scope*, which provides a

way to divide a complex business process into hierarchically organized parts. In a scope, there is a collection of activities that can have their own *variables*, *fault handlers*, and *compensation handlers*. A fault handler gets executed when an exception arises, for example through the execution of the `throw` activity, while compensation handlers are triggered due to faults or through compensate activities that force compensation of a scope. An interesting feature of BPEL is its support for two distinct styles of process modeling: the graph–oriented style, involving definition of a composition using graph primitives (nodes and edges), and the "algebraic" style derived from process calculi, in which complex control constructs (such as the compound activities above) result in implicit control flow. Each of these alone provides sufficient expressibility. Supporting both styles gives the designer maximum flexibility to develop a model in the most intuitive way.

- **WS-CDL (Web Services Choreography Description Language).** WS-CDL is an XML-based language that describes peer-to-peer collaborations of participants by defining their observable behavior. WS-CDL captures service interactions from a global perspective, meaning that all participating services are treated equally, which is different from BPEL where service interactions are described from one single participant perspective. WS-CDL supports exception handling through a special kind of workunit, namely `Exception`, which is associated with a choreography and is enabled when an exception occurs. Another special kind of workunit is `Finalizer`, which is enabled when its associated choreography completes successfully. A finalizer can confirm, cancel or modify the effects of completed actions of the choreography, which can be used to provide a compensation mechanism, as well as a range of coordination models.
- **BPML (Business Process Modeling Language).** BPML[11] is an XML-based language that is proposed by the Business Process Management Initiative (BPMI.org). Initially developed to describe business processes that can be executed by a business process management system, the later version of BPML incorporates many concepts of Web Service Choreography Interface (WSCI),[12] which focuses on the choreography of Web services. BPML provides basic and structural activities that are similar to those in BPEL. The basic activities are used to invoke the available services (`action`), assign a new value to a message (`assign`) and instantiate a process (`call`). Structured activities are used to manage the branch selection (`choice` and `switch`), repetition (`until` and `while`), sequential (`sequence`) and concurrent activities (`all`). BPML allows developers to schedule tasks to determine the time to perform the task by using `schedule`. `Context` is an important element in BPML. A context contains local definitions that specify common behavior for all activities within that context, which can be used to exchange information and coordinate execution. `Signal` is used to synchronize between parallel activities executing within the same context. Designed for long-running business processes, BPML also supports the feature of persistency. Nested processes could be established by aggregating several subprocesses. Two transaction mechanisms, atomic and open nested transactions, are provided in BPML. The former is for short-lived transactions while the latter is for long-running transactions. BPML also provides the exception handling mechanism (*exception process*) to deal with exceptional events, and the compensation mechanism (*compensation process*) to reverse the effects of a completed activity.
- **ebXML (Electronic Business Using XML).** ebXML[13] aims at defining a set of specifications for enabling business-to-business (B2B) interactions among companies of any size. ebXML consists of the following major components:
  - *Messaging service* provides a standard way to exchange business messages between organizations. One important feature of the messaging service is that it does not rely on any particular file transport mechanism (such as SMTP, HTTP, or FTP) or network for exchanging data.
  - *Registry* is a database of items that support doing business electronically. It stores important information about businesses such as XML schemas of business documents, definitions of library components for business process modeling, and trading partner agreements.
  - *Trading partner information.* The Collaboration Protocol Profile (CPP) provides the definition of an XML document that specifies the details of how an organization is able to conduct business electronically. The Collaboration Protocol Agreement (CPA) specifies the details of how two organizations have agreed to conduct business electronically.
  - *Business Process Specification Schema (BPSS)* provides the definition of an XML document modeling business processes (i.e., composite services). It identifies such things as the overall business process, the roles, transactions, identification of the business documents used (the DTDs or schemas), document flow, legal aspects, security aspects, business level acknowledgments, and status.
- **OWL-S.** OWL-S[14] previously DAML-S (DARPA Agent Markup Language for Web Services), provides the ability for describing and reasoning over services semantically. OWL-S consists of three upper ontologies: *service profile*, *process model*, and *grounding*. The service profile is used to describe services for the purposes of discovery. Service descriptions and queries are constructed from a description of functional properties (e.g., inputs, outputs, and preconditions) and non-functional properties (e.g., QoS parameters). In addition, the service profile class can be subclassed and specialized to create *profile taxonomies* that subsequently describe different classes of services. OWL-S process models describe the composition and execution of Web services. The process model is used both for reasoning about possible compositions (e.g., validation) and for controlling the enactment and invocation of a service. OWL-S defines three process classes: *composite*, *simple*, and *atomic*. Atomic processes are directly invocable and have no subprocesses. Simple processes are not invocable and provide a means of describing service or process abstractions. A simple process does not have any specific binding to a physical service and

[11] http://www.ebpml.org/bpml.htm.
[12] http://www.w3.org/TR/wsci.
[13] http://www.ebxml.org.
[14] http://www.w3.org/Submission/OWL-S.

thus has to be realized either by an atomic process, or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. Finally, the grounding of a service specifies the details of how to access the service. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (e.g., the IP address of the machine hosting the service, the ports used to expose the service).

- **WSMF.** The Web Service Modeling Framework (WSMF) [23] is an European initiative to provide a fully fledged modeling framework for describing various aspects related to Web services. Its main goal is to fully enable e-Commerce by applying semantic Web technology to Web services. WSMF is centered on two complementary principles: (1) a strong de-coupling of the various components that realize an e-Commerce application, and (2) a strong mediation service enabling Web services to communicate in a scalable manner. WSMF consists of four main elements: *ontologies* that provide the terminology used by other elements; *capabilities repositories* that define the problems that should be solved by Web services; *Web services descriptions* that define various aspects of a Web service; and *mediators* that bypass interoperability problems. There are two main projects in WSMF: the semantic Web enabled Web Services (SWWS) and the Web Service Modeling Ontology (WSMO). SWWS provides a description framework, a discovery framework, and a mediation framework for Web services, while WSMO service ontology includes definitions for goals, mediators, and Web services.

4.2. Comparison of Web services composition standards

We compare the composition standards described above according to the following requirements:

- *Composability* indicates the ability to assemble participating services into a composition process and model the interactions between them.
- *Role representation* indicates the ability to reflect the behavior that a participant has to exhibit in order to interact in the composition process.
- *Complex structure support* is the ability to model the complex structures that reflect the execution logic and ordering rules of actions performed within the composition process.
- *Adaptability* is the ability to deal with business exceptions and process faults during the execution of the composition process.
- *Compensability* represents the ability to reverse the effect of some unsuccessful work in cases where exceptions occur.
- *Semantic support* is the ability to represent semantics of participating services to facilitate service dynamic discovery and automated composition.

Table 1 gives a detailed language capacity comparison of BPEL, WS-CDL, BPML, ebXML, OWL-S, and WSMF. From Table 1, we can see that all languages except WSMF support the modeling of message exchanging and complex structure. Only BPEL and WS-CDL have the comprehensive expressibility to define interacting roles, complex structures, exception handling and compensation management. BPML does not support role representation and ebXML does not provide a compensation mechanism. We also can observe that only OWL-S and WSMF provide semantic support for services composition.

5. Research prototypes

The past decade has witnessed prosperous research and development activities on Web services composition. Since there are a huge number of services composition prototypes in the literature, it is impossible for this survey to present an exhaustive overview. Instead, we focus on 30 representative prototypes. These research prototypes are analyzed and compared by using the requirements identified in Section 3.

5.1. Overview of major research prototypes

As shown in Fig. 3, the selected research prototypes are categorized according to the automation level of the services composition modeling.

Table 1
Comparison of Web services composition standards.

Standards	WS-BPEL	WS-CDL	BPML	ebXML	OWL-S	WSMF
Composability	+	+	+	+	+	–
Role representation	+	+	–	+	–	–
Complex structure support	+	+	+	+	+	–
Adaptability	+	+	+	+	–	–
Compensability	+	+	+	–	–	–
Semantic support	–	–	–	–	+	+

(+) Support, (–) No support.

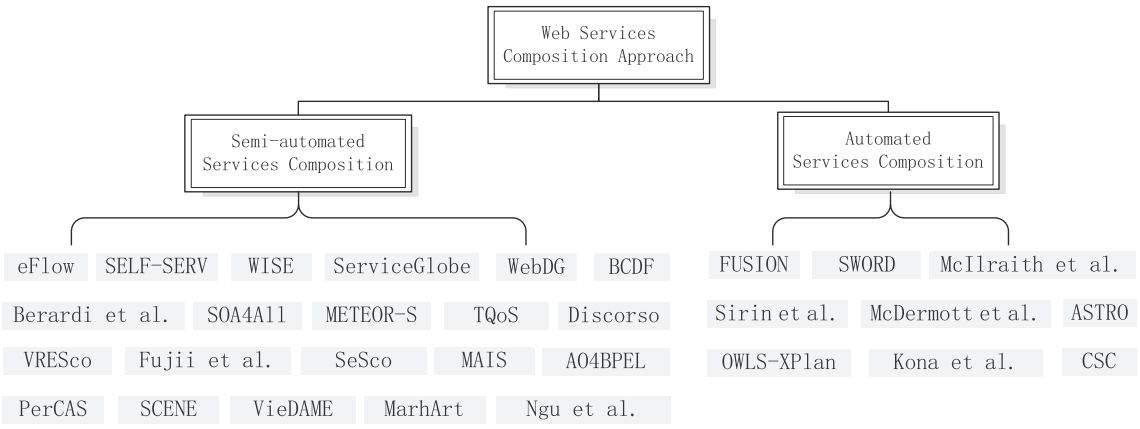

Fig. 3. The classification of selected research prototypes.

5.1.1. Semi-automated services composition

- **eFlow.** eFlow [16] is one of the first few platforms for specifying, enacting, and monitoring composite services. Composite services are specified as business processes that combine basic and composite services and are enacted by a service process engine. A composite service is modeled by a graph that defines the execution sequence among nodes in the process. eFlow provides a number of features that support adaptive service provisioning in dynamic environments, including *dynamic service selection*, *dynamic conversation selection*, and *generic nodes*. eFlow service nodes include the specification of a service selection rule that is represented in a query language. When a service node is started, the service selection rule is first executed to dynamically select an appropriate service. Dynamic conversation selection allows eFlow to select a corresponding conversation for the selected service at runtime, which makes the dynamic service selection practical, especially when service interfaces are unknown at the design time. Generic service nodes are introduced to support personalized services composition, which include a configuration parameter that can be set with a list of actual service nodes either at runtime or at process instantiation time.
- **Self-Serv (compoSing wEb accessibLe inFormation and buSiness sERVices).** Self-Serv [72] is a framework for dynamic and peer-to-peer provisioning of composite Web services. In Self-Serv, Web services are declaratively composed and the composite services are executed in a decentralized way. Self-Serv distinguishes three types of services: *elementary services*, *composite services*, and *service communities*. A composite service is specified by statecharts, data conversion rules, and provider selection policies. A service community is a container of a set of alternative services, and when a community receives a request, it dynamically selects one of its current members and delegates the request to it. In Self-Serv, a service is associated with a coordinator which controls and monitors the execution of the corresponding service. Once a service execution terminates, the control is passed to the next coordinator identified using a routing table. Self-Serv introduces a service quality model to specify non-functional properties of Web services [91]. The platform supports replanning procedures at runtime to ensure that the QoS of the composite services execution remains optimal.
- **WISE (Workflow based Internet SErvices).** WISE [35] project aims at providing a software platform for process based business-to-business electronic commerce. WISE has both a development environment and a runtime component associate with it. WISE is organized into three service layers: *database services*, *process services*, and *interface services*. The database service layer acts as the storage manager of all kinds of system data such as templates, instances, history, and configuration. The process service layer contains all the components required for coordinating and monitoring the execution of processes. The interface service layer contains interfaces with which users interact the system. Users specify processes via a process definition tool named StructWare that is based on Petri-nets.
- **ServiceGlobe.** The ServiceGlobe [30] is a lightweight, distributed, and extensible service platform that aims at providing new techniques for Web service execution and deployment in dynamic environments. To support the development, execution, and deployment of flexible and reliable services, ServiceGlobe proposes two approaches: *dynamic service selection* and the *dispatcher service*. Dynamic service selection offers Web services the possibility of selecting and invoking services at runtime based on a technical specification of the desired service. The dispatcher service addresses load balancing and high availability of Web services. The dispatcher forwards requests to different service instances and therefore reduces the risk of a service being unavailable and speeds up request processing because of load sharing. The dispatcher service implements a feature called *automated service replication*. Using this feature new services can be installed on idle hosts on behalf of the dispatcher. ServiceGlobe also implements a *context framework* that facilitates the development and deployment of context-aware adaptable Web services.
- **WebDG (Web Digital Government).** The WebDG project [47] enables semi-automated services composition by dealing with three major challenges. Firstly, WebDG presents an ontology-based framework for organizing and describing semantic Web services. Each service community is defined as an instance of an ontology called community ontology.

Then WebDG proposes a composability model to check whether semantic Web services can be combined together, hence avoiding unexpected failures at runtime. The notions of composability degree and τ-composability are introduced to cater for partial and total composability. Finally, based on the composability model, WebDG provides a set of algorithms that automatically generate detailed descriptions of composite services from high-level specifications of composition requests. The composite services are specified using a language called *Composite Service Specification Language* (CSSL), which is provided by WebDB. A Quality of Composition (QoC) model is also introduced to assess the quality of the generated composite services, which can be used to select an optimal composite service in case there exist multiple composition plans.

- **BCDF (Business Collaboration Development Framework).** Orriëns et al. [59] presents BCDF, a business rule driven services composition framework in which the process of services composition is divided into five phases: abstract definition, scheduling, construction, and execution. The framework consists of a Service Composition Manager (SCM) and Service Composition Repository (SCR). SCR facilitates maintaining of composition elements and rules that are used for composition. SCM consists of *Definer*, *Scheduler*, *Constructor* and *Executor*, which interact with SCR to assist developing, executing and managing services composition. When receiving the request from the application developer, SCM passes it on to *Definer* to construct the composition structure, add message-exchanging behavior, assign roles and define event-handling behavior. Then, *Scheduler* selects the service providers and *Constructor* generates executable composite service. Finally, *Executor* executes the composite service and presents the result. The rules used in the definition and scheduling phases include structure, data, constraint, resource and exception rules.
- **Berardi et al.** In Berardi et al. [8], the behaviors of Web services are described by Roman Model in which services export their behavioral features by using finite transition systems. For verifying the existence of a Web services composition, the existence of a simulation relation between the target and the component services is checked, which is optimal from the computational complexity point. Because the maximal simulation contains enough information for extracting every possible composition through a choice function, this technique could be used to implement services composition that could deal with events that may occur at run-time. Therefore, the actual services composition could be synthesized in a "just-in-time" fashion at run-time.
- **SOA4All (Service Oriented Architectures for All).** SOA4All [36] is a large scale integrating project funded by the European Seventh Framework Programme, which aims to provide the power, flexibility and simplicity that is necessary for a wider uptake of service-oriented technologies. SOA, semantic Web, Web of data and context adaptation are adopted as the core principles of SOA4All. In SOA4All, available Web services, including RESTful services and SOAP-based services, are annotated with semantic information, intending to implement automated service discovery, mediation and composition. Contextual information, including local environmental constraints, organizational policies and personal preferences are considered in services composition. SOA4All uses a mashup-like way to assist end-users to compose Web services.
- **METEOR-S (Managing End To End OpeRation for semantic Web).** METEOR [52] is a workflow management system, which focuses on formal modeling, central and distributed scheduling and execution of workflow. After adopting service oriented architecture and semantic Web services, METEOR evolved into METEOR-S [78], incorporating Data, Functional and QoS semantics to support the complete life cycle of semantic Web processes. The architecture of METEOR-S is divided into three main parts: The *Specification and Annotation*, the *Dynamic Configuration* and the *Process Adaptation*. The Specification and Annotation part adds semantic met-data to WSDL elements by using SAWSDL.[15] The Dynamic Configuration part is in charge of service discovery, constraint analysis, and data mediation. The Process Adaptation part deals with the problem of adapting the composite services to runtime events and faults.
- **TQoS (Transactional QoS-Driven Web Services Composition).** El Haddad et al. [28] presents a transactional and QoS-aware selection method for Web services composition. Service selection is realized based on transactional and QoS requirements. Transactional requirement is expressed by a risk notion that denotes if the results could be compensated or not. Quality requirement is described as a set of weights over QoS criteria. In [28], five QoS criteria (execution price, execution duration, reputation, successful execution rate and availability) are used and a local optimization method for service selection is proposed. The authors also present and formally analyze a service selection algorithm based on the workflow patterns and the transactional properties of the component services.
- **Discorso (Distributed Information System for Coordinated Service-Oriented Interoperability).** Discorso [3] is a framework that aims to provide a comprehensive service-based solution for specifying and managing flexible and adaptive composite services. Discorso provides a set of tools to support specification of all required information for runtime adaption of the composite services. The service selection method implemented in Discorso is based on mixed-integer linear programming model, which uses negotiation technique to bargain QoS parameters with service providers when an end user has severe QoS constraints and thus available solutions cannot be found. Discorso provides mediation support if the selected service's interface differs from the interface that the corresponding task definition requires. Discorso also provides supervision rules to monitor service execution and trigger corrective actions if needed.
- **VRESCo (Vienna Runtime Environment for Service-Oriented Computing).** Michlmayr et al. [50] present a runtime environment, namely VRESCo, for Web services composition. The VRESCo *Runtime Environment* is provided as a server application, which consists of a *Query Engine*, a *Notification Engine*, a *Publishing/Metadata Service*, a *Management Service*

[15] http://www.w3.org/TR/sawsdl/.

and a *Composition Engine*. Dynamic and Asynchronous Invocation of Services (Daios), a client framework for dynamic service invocation, is also provided to conduct the message exchange with the involved Web services. Daios supports both SOAP-based and RESTful services. VRESCo allows QoS based service selection and composition. A *QoS Monitor* is deployed as an independent module to measure the availability, performance and accuracy of the target Web services. Service versioning, dynamic service binding and invocation, as well as service mediation are supported in this system.

- **Fujii et al.** Fujii et al. [26] implement a semantic-based, context-aware services composition framework which allows composing applications requested in a natural language. This framework is made up of component service model with semantics (CosMoS), component runtime environment (CoRE), and semantic graph-based services composition (SeGSeC). CoSMoS is a component model supporting function, semantic, and context representation of services. CoRE is a middleware that supports CoSMoS on different distributed computing technologies. SeGSeC provides a mechanism to construct a composite service by synthesizing its process based on the semantics and contexts of component services. SeGSeC implements context-aware services composition based on specified rules or user preferences obtained through learning.
- **SeSCo (Seamless Services Composition).** SeSCo [29] provides a services composition mechanism in pervasive environments. SeSCo utilizes an event-oriented middleware platform, called Pervasive Information Communities Organization (PICO), to realize service interactions in an efficient manner. The modeling constructs in PICO offer techniques to extract the capabilities of resources and allow services to be built around the capabilities. The services composition mechanism in SeSCo models services as directed attributed graphs, maintains a repository of service graphs, and dynamically combines multiple services into complex services. A hierarchical services composition mechanism based on a device overlay formed through the latch protocol is also presented, which provides essential service-related support to resource-poor devices.
- **MAIS (Multichannel Adaptive Information Systems).** The MAIS [43] project aims at creating a platform, a methodology, and a set of design tools to develop distributed information system based on e-Services. MAIS fulfills QoS-aware dynamic services composition using contextual information. In MAIS, a service is described by a name, a short description, a service category, and an aggregation of three types of elements: a *Channel* (containing contextual information), one or more *ServiceProviders* and a *FunctionalDecription*. When requesting a composite service, a user specifies the composition requirement and desired QoS constraints, while services are selected based on QoS constraints and their contexts.
- **AO4BPEL (Aspect-Oriented for BPEL).** Charfi et al. [18] design and implement an aspect-oriented workflow language, namely AO4BPEL, which extends BPEL to support aspect features and provides a framework to define workflow aspects for cross-cutting concerns such as logging, auditing, security, and dynamic adaptation of Web services composition at runtime. By extending the BPEL engine with an aspect runtime component, AO4BPEL can dynamically change the deployed process through activating or deactivating the defined aspects. In [17], AOBPEL is further extended to support programmatic activation and deactivation of the adaptation aspects. The extended AO4BPEL is used to implement a plug-in architecture for self-adaptive Web services composition, including automatic replacement of services, self-adaptation to policy changes, and service-level agreements (SLAs) monitoring.
- **PerCAS (Personalized Context-aware Services).** Yu et al. [89] presents a model-driven approach, named PerCAS, to implement personalized services composition that could adapt to user-specific requirements. The Personalized Context-awareness Logic Model (PCLM), specified by using a rule language, is separated from the base functionality logic of a composite service. Personalized rules could be dynamically adjusted at runtime. The runtime environment integrates a BPEL engine and a Drools rule engine to support the execution of such services.
- **SCENE (Services Composition ExecutioN Environment).** SCENE [19] is part of *Service Centric System Engineering* (SeCSE) project, which aims at providing methods, tools and a platform for service engineering. In SCENE, BPEL language is extended with rules, which enables binding and re-binding self-reconfiguration operations at runtime. The SCENE platform offers an execution environment for composite services described by the SCENE language, in which the BPEL engine is extended with a set of *Proxies* to support reconfiguration. A rule engine, *Drools*, is integrated to execute Event–Condition-Action (ECA) rules at runtime, and the *Binder* module is created to execute dynamic binding of candidate services. A monitoring system is also integrated to provide SCENE with the required monitoring feedbacks.
- **VieDAME (Vienna Dynamic Adaptation and Monitoring Execution).** Moser et al. [53] propose an execution environment called VieDAME that focuses on QoS attributes monitoring and dynamic adaptation of BPEL processes. VieDAME allows replacement of component services based on various service selection strategies, such as availability and response time. The dynamic adaptation mechanism is implemented with an aspect-oriented approach by intercepting messages between the composite service and participant services. The VieDAME system is split into the VieDAME *core* and the VieDAME *engine adapters*. The VieDAME core is responsible for monitoring, service selection and message transformation (to compensate service interface mismatches). The engine adapters offer the aspect-oriented interfaces to integrate different BPEL engines.
- **Ngu et al.** Mashup is a Web based services composition method, which enables the service consumers to create new applications just by simple actions. Ngu et al. [57] propose an approach to implement progressive composition of Web components including portlets, Web applications, native widgets, legacy systems and so on. Semantic annotation for components and the semantic matching algorithm for searching components are offered. The implementation system includes a Composite Application Infrastructure (CAI) and an associated Composite Application Editor (CAE). CAI is the runtime environment for the composite applications, and CAE is utilized to compose application components at development time. End-users can drag and drop the components into composition canvas, and connect them by data-flow arcs to realize a composite application without any low-level programming efforts.

- **MashArt.** A universal composition approach for UI components as well as data and application logic services is the base of the MashArt system [20], which aims at enabling end-users to perform complex UI and services composition. The universal component model is specified by means of the *MashArt Description Language* (MDL) that consists of four abstractions: *State*, *Events*, *Operations*, and *Properties*. *Events* communicate *State* changes and other information to the composition environment, and *Operations* are the method invoked and often represent state change. *Properties* include arbitrary setup information of the component. The universal composition model is defined using the proposed Universal Composition Language (UCL), which operates on MDL descriptors only. MashArt uses event-based mechanisms and data flow between various types of components to allow the construction of complex applications.

5.1.2. Automated composition

- **FUSION.** FUSION [86] is a framework for service portals. By giving a user service specification, a correct and optimized service execution plan can be automatically generated and executed, and the results are verified. FUSION contains six parts: *User Specification*, *Web Services Dynamic Plan Generator*, *Plan Execution*, *Verification*, *Recovery*, and *User Response Generation*. The user specification part is a graphical form-based interface that allows users to specify their abstract requirements. The Web services dynamic plan generator takes input the user specifications and generates correct and optimized execution plan, in a language called *Web Services Execution Specification Language* (WSESL). The plan execution part maps the execution plan into executable code and invokes the method instances described in the plan. The execution part interacts with (1) the verification part to make sure that the service result meets the users satisfied criteria, and (2) the recovery part to initiate an appropriate recovery process in case of the failure of the verification. Finally, the user response generation part is responsible for preparing and delivering the final response to the user.
- **SWORD.** SWORD [65] provides a set of tools that allows developers to quickly compose existing Web services to realize new composite Web services. It is interesting to note that SWORD does not exploit emerging service standards like WSDL and OWL-S, instead, it uses the Entity-Relationship (ER) model [85] to specify Web services. In SWORD, each service is modeled by its inputs and outputs, which are specified in a "world model" that consists of entities (e.g., movies) and relationships among entities (e.g., a theater shows a movie). To create a composite service, a service developer only needs to specify the initial and final states of the composite service. A rule-based expert system is then used to automatically determine whether a desired composite service can be realized using existing services.
- **McIlraith et al.** McIlraith et al. [46] present an approach to address automated Web services composition and its execution for the semantic Web. The authors adapt and extend GOLOG [37], a logic programming language built on top of the situation calculus, to enable processes *generic*, *customizable* and *usable* in the context of the Web. Web services are conceived as primitive actions (information-gathering or world-altering) and complex actions with precondition and effects. Web services composition is conceived as a planning problem, and a composite service is a set of component services which are connected by GOLOG constructs (sequence, choice and so on). Sohrabi et al. [82] extend the method so that GOLOG generic procedures could be customized not only with "hard" constraints but with "soft" user constraints (user preferences).
- **Sirin et al.** Sirin et al. [79] adopt Hierarchical Task Network (HTN) planning to address the problem of automated Web services composition. The authors believe that HTN planning is especially promising for automated services composition, because the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in the OWL-S process ontology. SHOP2 is a domain-independent HTN planning system [56], which is applied in this approach to perform the services composition planning. In this approach, OWL-S specification are translated into HTN planning and submitted to SHOP2 planner to construct the composite service, which is a sequence of Web services calls that can be subsequently executed. A plan converter is also built to convert SHOP2 plans into OWL-S processes which can be directly executed by the OWL-S executor.
- **McDermott et al.** The Planning Domain Definition Language (PDDL) [44], a language for the specification of planning domains, is a standard input for some planners. In McDermott et al. [45], PDDL is extended to formalize Web services, and Estimated-regression (a planning technique) planners are applied to implement automated Web services composition. In this approach, DAML-S (former version of OWL-S) descriptions are translated into PDDL format and thus the Web services composition problem is seen as a PDDL planning problem.
- **ASTRO.** In Trainotti et al. [84], a services composition framework, based on the concept of "Planning as Model Checking", is presented. Given a description of the external protocol (e.g., expressed as an abstract BPEL specification) and composition requirements (i.e., the business goal, expressed with the EaGLe goal language), the planner synthesizes automatically the composite service that implements the internal process. A monitor of the composite service is also automatically generated and able to detect whether the component services behave consistently with the specified protocols. Later they develop the ASTRO toolset, consisting of WS-gen, WS-mon, WS-console and WS-animator, to support automated services composition, monitoring and execution. WS-gen and WS-mon are responsible for generating the composite service (BPEL process) and the monitors respectively. WS-animator executes the composite services and WS-console presents the status of the monitors associated with each composite service instance.
- **OWLS-Xplan.** OWLS-Xplan [41] is a composition tool that supports automated and flexible composition of OWL-S specified services in the semantic Web. In OWLS-Xplan, an artificial intelligence planner called Xplan is applied to generate the services composition plan, which takes as input the PDDL descriptions of OWL-S services and a planning query, and returns a plan sequence of component services that satisfies the query goal. Xplan is a hybrid planner which extends

Table 2
Comparison: prototypes vs. services composition requirements.

Prototypes	Definition phase		Selection phase		Execution phase				Overall	
	Expressibility	Correctness	Automation	Selectability	Adaptability	Scalability	Monitoring	Reliability	Personalization	Tool support
eFlow	SFS (∼)	NA (↓)	SY (∼)	NA (↓)	PA/DB (↑)	CE (∼)	EL (∼)	EH/TS (↑)	UP (∼)	ST (∼)
Self-Serv	FM (↑)	NA (↓)	SY (∼)	LO/GO (↑)	PA/DB (↑)	DE (↑)	EL (∼)	EH (∼)	UP/CS (↑)	ST (∼)
WISE	FM (↑)	CV (↑)	MS (↓)	NA (↓)	DB (∼)	DE (↑)	EL/PA (↑)	EH (∼)	NA (↓)	MT (↑)
ServiceGlobe	NA (↓)	NA (↓)	SY (∼)	NA (↓)	DB (∼)	NA (↓)	NA (↓)	NA (↓)	CS (∼)	FT (↓)
WebDG	SFS (∼)	NA (↓)	SE (↑)	LO (∼)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	ST (∼)
BCDF	SD (∼)	NA (↓)	SY (∼)	LO (∼)	NA (↓)	CE (∼)	NA (↓)	EH (∼)	NA (↓)	FT (↓)
Berardi et al.	FM (↑)	CV (↑)	SY (∼)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	FT (↓)
SOA4All	SL (↑)	NA (↓)	SE (↑)	GO (↑)	NA (↓)	CE (∼)	EL (∼)	EH/CS (∼)	UP/CS (↑)	MT (↑)
METEOR-S	SL (↑)	NA (↓)	SE (↑)	LO (∼)	PA/DB (↑)	CE (∼)	EL (∼)	EH/CS (∼)	NA (↓)	ST (∼)
TQoS	FM (↑)	NA (↓)	SY (∼)	LO (∼)	NA (↓)	CE (∼)	NA (↓)	TS (↑)	UP (∼)	FT (↓)
Discorso	SL (↑)	NA (↓)	SE (↑)	LO/GO (↑)	PA/DB (↑)	CE (∼)	EL (∼)	EH/CS (∼)	UP/CS (↑)	ST (∼)
VRESCo	SFS (∼)	NA (↓)	SY (∼)	LO (∼)	DB (∼)	CE (∼)	QS (↑)	NA (↓)	NA (↓)	MT (↑)
Fujii et al.	SFS (∼)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	NA (↓)	UP/CS (↑)	ST (∼)
SeSCo	SFS (∼)	NA (↓)	SE (↑)	LO (∼)	DB (∼)	DE (↑)	NA (↓)	NA (↓)	UP/CS (↑)	FT (↓)
MAIS	FM (↑)	NA (↓)	SY (∼)	LO (∼)	DB (∼)	DE (↑)	NA (↓)	NA (↓)	UP/CS (↑)	FT (↓)
AO4BPEL	SL (↑)	NA (↓)	MS (↓)	NA (↓)	PA/DB (↑)	CE (∼)	EL/QS (↑)	EH/CS (∼)	NA (↓)	ST (∼)
PerCAS	SL (↑)	NA (↓)	MS (↓)	NA (↓)	PA/DB (↑)	CE (∼)	EL (∼)	EH/CS (∼)	UP/CS (↑)	ST (∼)
SCENE	SL (↑)	NA (↓)	SY (∼)	LO (∼)	DB (∼)	CE (∼)	EL (∼)	EH/CS (∼)	NA (↓)	ST (∼)
VieDAME	SL (↑)	NA (↓)	SY (∼)	LO (∼)	DB (∼)	CE (∼)	QS (↑)	EH/CS (∼)	NA (↓)	MT (↑)
Ngu et al.	NA (↓)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	NA (↓)	NA (↓)	ST (∼)
MashArt	SFS (∼)	NA (↓)	MS (↓)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	NA (↓)	NA (↓)	ST (∼)
FUSION	PL (↓)	CV (↑)	SY (∼)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	EH (∼)	NA (↓)	ST (∼)
SWORD	PL (↓)	NA (↓)	SY (∼)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	EH (∼)	NA (↓)	ST (∼)
McIlraith et al.	PL (↓)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	CE (∼)	NA (↓)	NA (↓)	UP (∼)	FT (↓)
Sirin et al.	PL (↓)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	CE (∼)	EL (∼)	NA (↓)	UP (∼)	ST (∼)
McDermott et al.	PL (↓)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	FT (↓)
ASTRO	SL (↑)	NA (↓)	SY (∼)	NA (↓)	NA (↓)	CE (∼)	EL (∼)	EH/CS (∼)	NA (↓)	MT (↑)
OWLS-Xplan	PL (↓)	NA (↓)	SE (↑)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	FT (↓)
Kona et al.	SFS (∼)	CV (↑)	SE (↑)	NA (↓)	DB (∼)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	FT (↓)
CSC	NA (↓)	NA (↓)	SE (↑)	GO (↑)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	NA (↓)	FT (↓)

Definition phase

Expressibility		Correctness	
SFS	Self-defined Flow Structure	CV	Correctness Verification
FM	Formal Method	NA	Not Addressed
SL	Standard Language		
PL	Planning Language		
NA	Not Addressed		

Selection phase

Automation		Usability	
SY	Syntactic Support	LO	Local Optimization
SE	Semantic Support	O	Global Optimization
MS	Manual Selection	NA	Not addressed

Execution phase

Adaptability		Scalability		Monitoring		Reliability	
PA	Process Adaptation	DE	Decentralized Execution	EL	Execution Logging	EH	Exception Handling
DB	Dynamic Service Binding	CE	Centralized Execution	QS	QoS Monitoring	TS	Transactional Support
NA	Not Addressed	NA	Not Addressed	PA	Properties Analysis	CS	Compensation Support
				NA	Not Addressed	NA	Not Addressed

Overall

Personalization		Tool support	
UP	User Preferences Support	MT	Provide Many Tools
CS	Context-Aware Support	SM	Provide Some Tools
NA	Not Addressed	FT	Provide Few Tools

Legend: (↑) High (∼) Medium (↓) Low

an action based FastForward-planner (graph based planning) with HTN planning and re-planning component. OWLS-Xplan consists of several modules for preprocessing and planning. The preprocessing module is responsible for creating the required data structures, generating the initial connectivity graph and goal agenda. The planning module supports the heuristically relaxed graph-plan generation and enforced hill-climbing search.

– **Kona et al.** Kona et al. [32] use the Universal Service-Semantics Description Language (USDL) to specify the formal semantics of services. USDL uses WordNet as a common basis for understanding the meaning of services. A service is represented by pre-condition, inputs, side-effect, affected object, outputs and post-conditions. Given a repository of available Web services and a query of the requirements on the request service, a directed acyclic graph of Web services to achieve the desired service is automatically synthesized and translated into OWL-S specification. The composition flow includes sequential, non-sequential and conditional structures.

– **CSC (Cooperative Service Composition).** Mehandjiev et al. [49] propose a cooperative services composition method to support automated synthesis of composite services. Service providers are proactive in the synthesis of the composite services. The service requestor announces the need for a composite service by specifying its start and end states on a specialist noticeboard. Each service provider agent will bid to contribute a partial solution for the required composite service. When a full solution is obtained, it is presented to the service requestor agent. All possible solutions are evaluated using the composition quality metrics, which include semantic quality and non-functional QoS of the solutions.

5.2. Research prototype comparison

The evaluation of Web services composition prototypes covers 30 representative research prototypes. Some are classical research prototypes (e.g., eFlow, WISE and Self-Serv) that have made fundamental contributions and have influenced the field of services composition, while the rest are recently emerged, representative research work. In this section, these research prototypes are compared using the requirements based on the Web services composition life cycle, presented in Section 3. Table 2 summarizes our results.

For each research prototype, we summarize the main features according to the identified requirements. For example, for the two requirements of the definition phase, eFlow features a *self-defined structure* in terms of expressibility and provides no support for correctness checking (i.e., *not addressed*). For each requirement, we also provide an indicative assessment using *high*, *medium*, or *low*. For instance, for Automation in the selection phase, if a prototype supports manual selection of services (i.e., MS), we give a ranking of *low*. If it supports semantic-based selection (i.e., ME), we give a ranking of *high*, and similarly, a *medium* for syntactic based service selection (i.e., SY).

From the table we can see that eFlow uses self-defined graph structure to describe the composite service process. The expressibility of the modeling method is comparatively low and the correctness verification is not supported. eFlow utilizes selection rules to provide syntactic support for automated service selection. However, non-functional properties are not considered in service selection. eFlow is adaptable because it supports dynamic modification of composite service processes and dynamic service binding. eFlow is not scalable because it employs a centralized execution engine. The monitoring mechanism of eFlow is only limited to execution logging of composite services. eFlow supports exception handling and transaction management and thus the reliability of eFlow is high. eFlow supports personalization in the sense that users can customize the composite service processes. Finally, some tools are available in eFlow to facilitate the modeling of composite service processes and the monitoring of running instances.

From the *expressibility* column in Table 2, we can see that some of the semi-automated composition prototypes (e.g., Self-Serv, WISE, and MAIS) adopt formal methods, including statecharts, Petri-nets and state machine, to model the composite service processes. There are about one-third of semi-automated composition prototypes (e.g., SOA4All, METEOR-S, and AO4BPEL) whose composite services are designed using the BPEL language, the de facto standard for defining Web service based processes. Other semi-automated composition prototypes (e.g., eFlow, WebDG, VRESCo and MashArt) use self-defined flow structures as modeling languages. In comparison, the use of formal methods and BPEL allows for more expressive power in defining complex structures, data flow, temporal constraints, exception handling, transactional features, etc. Most of the automated composition prototypes (e.g., FUSION, SWORD and OWLS-Xplan) adopt planning languages as their modeling methods. The expressibility of these planning languages is relatively low and most of the automated composition methods are limited to sequential composition of atomic services. For the *correctness* requirement, we note that only a few of the research prototypes support behavioral analysis for composite services. Among these prototypes, FUSION, the work by Berardi et al., and the work by Kona et al. provide correctness guarantees, while WISE provides what-if and bottleneck analysis.

For the *automation* requirement, a few of research prototypes (WISE, AO4BPEL, PerCAS and MashArt) only support manual selection of the component services. Half of the rest (e.g., eFlow, Self-Serv, and ServiceGlobe) support automated service selection based on syntactic matchmaking while the other half (e.g., WebDG, SOA4All, and METEOR-S) provide semantic support for automated service selection. For the *selectability* requirement, we note that more than half (57%) of the prototypes do not consider any QoS constraints for service selection. Only Self-Serv, SOA4All, Discorso, and CSC implement QoS driven service selection based on global planning, while the other prototypes (e.g., WebDG, METEOR-S, and VRESCo) perform optimal service selection for each individual task without considering the overall QoS constraints of a composite service.

For the *adaptability* requirement, a high proportion of research prototypes (53%) do not provide adaptation mechanism at runtime. 27% of the research prototypes (e.g., WISE, SeSCo, and VieDAME) only support dynamic service binding and the rest 20% (i.e., eFlow, Self-Serv, METEOR-S, Discorso, and AO4BPEL) support both service Process Adaptation and dynamic service binding. For the *scalability* requirement, Self-Serv, WISE, SeSCo, and MAIS support decentralized execution mode. These four prototypes have higher scalability than the rest of the research prototypes (87%) that only provide centralized engine, or do not provide support for the execution of the composite services. Moreover, we note that the majority of research prototypes (87%) do not, or only partially, provide support for the *monitoring* requirement. Only a few research types (i.e., WISE, VRESCo, AO4BPEL, and VieDAME) provide abilities to implement QoS statistics and functional verification. For the *reliability* requirement, 33% of research types support exceptional handling and transactional management for the composite services, most of which (e.g., SOA4All, and AO4BPEL) depend on the reliable mechanism (*fault handler* and *compensation handler*) of the BPEL language and only eFlow and TQoS provide specific functions to support transactional properties. 17% of the research prototypes (e.g., Self-Serv, WISE, and FUSION) only support exception handling and the rest 50% (e.g., ServiceGlobe, WebDG, and ASTRO) do not have any particular mechanism for the reliability requirement.

Finally, it is worth mentioning that more than half of the approaches (60%) neglect the *personalization* issues of composite services. Self-Serv, SOA4All, Discorso, MAIS, PerCAS and the work of Fujii et al. support both user preferences and contextual constraints feature for personalized service provisioning. For the *tool support* requirement, we note that only WISE, SOA4All, VRESCo, VieDAME, and ASTRO (17%) provide comprehensive software tools to facilitate process modeling, service discovery, execution monitoring and management. 47% of the research prototypes (e.g., eFlow, Self-Serv, and WebDG) offer some tools to implement basic functions (e.g., process modeling). As to the remaining research prototypes (36%), little is presented and discussed on tool support for services composition.

6. Services composition platforms

Major software vendors (e.g., IBM, Oracle, Microsoft) and open source organizations (e.g., Apache,[16] JBoss[17]) have been working on implementing services composition platforms in the last decade. In this section, we first present a set of features required for services composition platforms and then compare several major platforms based on these features. Similarly, we focus on the major players in this arena. Our study is based on vendor white papers since there are few or no published technical papers detailing these products. Additionally, these platforms operate at different levels of disclosure.

6.1. Features of services composition platforms

- *Open standard support.* Open standard support is the basic prerequisite for achieving high interoperability and compatibility, with which a services composition platform can choose services provided by third parties to achieve greater re-usability, and collaborate with partners to achieve better data portability. Information can be exchanged and interchanged in a straightforward manner.
- *Ease-of-use.* Services composition platforms should be easy to use, meaning that necessary tools are required to reduce the complexity and improve the efficiency of services composition development. For example, the composite service process could be modeled more easily using a drag-and-drop editor.
- *Simulation.* Simulation is a means of creating a virtual composite service in order to evaluate its performance before executing it in a real environment. This allows developers to dynamically verify process flows, validate the model and collect timing and resource information on the composite service with a view to business process improvement.
- *Administration and monitoring.* The runtime administration, including starting, suspending, resuming and terminating composite service instances, should be supported by a services composition platform. Monitoring includes tracking the execution states of instances and analyzing the performance characteristics for process improvement.
- *Adaptability.* To capture the dynamics of service oriented environment, a services composition platform should be able to provide adaptability to composite services, such as dynamic service re-binding, process version management, process definition updating and instance updating. The change of the composite service process requires dynamic updating without suspending or terminating related running instances.
- *Optimization.* Optimization is an important way to reduce costs and create greater business value. The intent of optimization is to retrieve process performance information from modeling or monitoring phase, to promote full-capacity use of services by reducing the potential or actual bottlenecks, to enhance efficiency by arranging the activities in the best order, and to identify the potential opportunities for cost savings or other improvements.
- *Security.* It is important to ensure security in Web service applications. The security techniques of services composition include providing message protection through message integrity, encryption and authentication, and preventing unauthorized access of services through access control policies.

6.2. Comparison of services composition platforms

We compare several representative services composition platforms including five commercial platforms IBM *Business Process Manager*,[18] Oracle *BPEL Process Manager*,[19] Microsoft *BizTalk Server*,[20] SAP *NetWeaver Process Integration*,[21] Active Endpoints *ActiveVOS*,[22] and two open source software products Apache *Orchestration Director Engine (ODE)*[23] and JBoss *jBPM*[24] The comparison result is shown in Table 3.

From Table 3, we note that all services composition platforms adopt BPEL as the services composition language. Some of the platforms (e.g., IBM Business Process Server, Oracle BPEL Process Manager, and Apache ODE) provide BPEL-based

[16] http://www.apache.org/.
[17] http://www.jboss.org/.
[18] http://www-03.ibm.com/software/products/us/en/business-process-manager-family.
[19] http://www.oracle.com/technetwork/middleware/bpel/overview.
[20] http://www.microsoft.com/en-us/biztalk/default.aspx.
[21] http://help.sap.com/nwpi.
[22] http://www.activevos.com.
[23] http://ode.apache.org.
[24] http://www.jboss.org/jbpm.

Table 3
Comparison of services composition platforms.

Platforms	Open Standard support	Ease-of-use	Simulation	Administration and monitoring	Adaptability	Optimization	Security
IBM Business Process Manager	WS-BPEL	Provide eclipse-based tools for BPEL design	Support process simulation and replay	Support administration of process state and performance	Provide recovery facilities; support authoring and editing of business rules	Integrate process optimizer for bottlenecks identification and historical analyses	Support WS-security for lower-level integration
Oracle BPEL Process Manager	WS-BPEL	Provide process designer for graphical modeling and service browser	Support transit simulation	Support deploying, managing, administration, auditing and debugging BPEL processes with BPEL Console	Integrate rules engine for dynamic decision making at runtime	Use Business Process Analysis (BPA) for optimization	Support security policies including password protection and message encryption
Microsoft BizTalk Server	WS-BPEL & XLANG	Integrate visual studio for graphical development	Support Web services simulation	Support real-time monitoring by Business Activity Monitoring (BAM)	Support business rules engine	Support performance monitor and analysis	Support access control and data security
SAP NetWeaver Process Integration	WS-BPEL	Provide graphical modeling tool	No support	Support monitor and administrate individual and end-to-end processes	Provide rules composer and rules engine	Support measurement and analysis of business processes	Support data protection and communication security
ActiveVOS	WS-BPEL; BPEL4People; WS-HumanTask	Provide ActiveVOS Designer for development, test and deployment of processes	Provide business process model simulator	Support engine properties monitoring and abnormalities handling by ActiveVOS Monitoring Alert Service	Allow process to automate changes to the configuration documents; support process instances migration	Provide performance dashboard for identifying resources and bottlenecks	Support WS-Security
Apache ODE	WS-BPEL	eclipse BPEL editor	No support	Support process support for instance-state administration and monitoring	No support	No support	Support WS-Security
JBoss jBPM	WS-BPEL; WS-HumanTask	Provide eclipse-based and Web- based editor for graphical creation of processes	No support	Provide management console supporting process instance management and task management; history logging	Support process instance migration	No support	security features are still in alpha stage

graphical modeling tools, while other platforms adopt graphical models (such as BPMN) and support automated transformation from those models to BPEL.

For most of the platform providers, the services composition platform is a major part of their Business Process Management Suite and human tasks support is an important part of a BPM solution. Therefore, some of the platforms (e.g., ActiveVOS and jBPM) support WS-HumanTask or BPEL4People, describing how human interaction in BPEL processes can be performed. In addition, IBM Business Process Server, Oracle BPEL Process Server, and BizTalk Server also provide various mechanisms to implement human interactions. For example, Human Workflow Services (HWS), a standard part of BizTalk Server, is used for workflow activities that involve human intervention. The advantage of the commercial software is that they provide a large set of tools to implement requirements of Process Adaptation, optimization, security, etc. On the other hand, the advantage of the open source software (Apache and JBoss) is that they are more flexible and open, which makes it easy to add new features.

7. Open issues

Although Web services composition has been extensively studied in the past decade, techniques are still not fully mature yet with several open issues remaining. Moreover, the rapid rise and adoption of new computing paradigms such as Cloud computing, social computing, and Web of Things in recent years also presents compounded challenges in this area [12]. In this section, we identify several directions for future research on services composition.

- *Dependable services composition.* The components of a composite service are normally distributed and autonomously provided by different organizations. As indicated from our analysis in Section 5, providing reliable and dependable services composition still remains a significant challenge [22,94,75]. When composing services, particularly for mission-critical applications (e.g., health care, stock trading, and air traffic control), service developers should be able to check the soundness and completeness of compositions so that the design problems can be identified and addressed at early stages. This typically involves modeling, verifying, and checking service behaviors in terms of transaction and conversation support. In recent years, model checking and verification of fault tolerant Web services compositions has become an active research topic [80,94,75,25,68,14,39]. We believe more research is needed for developing novel solutions and tool sets on dependable services composition. Furthermore, Web services composition needs to consider application domains. Different domains may have very different constraints and features, which present significant challenges in dependable services composition. Domain knowledge, constraints, preferences, and policies need to be specified and enforced [88].
- *Adaptable and autonomous services composition.* Nowadays the environment in which composite services are developed and executed has become more open, dynamic, and ever changing. Accordingly, there is a need for a more adaptable and flexible approach for services composition. Autonomous services composition is a promising research effort to increase the adaptability of services composition, which includes several fundamental properties: self-configuring, self-optimizing, self-healing, and self-adapting [90,17,61,74]. Self-configuring composition means that the composite service can discover and select new component services automatically. Self-optimizing composition indicates that component services will be chosen according to the constraints of QoS. Self-healing composition can automatically detect the violations of requirement and react to these violations correctly. Self-adapting compositions means the composite service could adjust its behavior in case of changes of external services with only a little or none of human intervention. Typically, Web services are developed independently by different organizations. These services are often incompatible and cannot interact directly. Service mediation provides an effective way in dealing this challenge and several research efforts have been devoted on this topic in the last few years [39,54,33,38]. These approaches mostly focus on how to synthesize service mediators semi-automatically or automatically in the case when services could be mediated. In practice, interactions among services might not be fully mediated due to irreconcilable mismatches. This calls for further research on Web services mediation.
- *Pervasive services composition.* The proliferation of ubiquitous, interconnected computing devices (e.g., PDAs, 3G mobile phones), as well as recent advances in radio-frequency identification (RFID) technology and sensor networks, are fostering the emergence of environments where Internet applications and services made available to mobile users are a commodity [76,29,15,66,64,2]. Composing services across multiple mobile devices in such an environment presents new challenges that do not occur in traditional services composition settings [15]. In particular, composition mechanisms in pervasive environments need to address context awareness, heterogeneity and contingencies of devices (e.g., unpredictable availability of services and mobile devices), and personalization (e.g., service provisioning based on user preferences). Since the devices where services are running are usually resource constraint (e.g., limited memory and battery life), special considerations are necessary for the efficiency and performance of composite services. From our analysis of existing services composition prototypes (see Section 5), it is clear that relatively few research focuses on services composition in pervasive environments. Extensive research efforts are therefore needed in this direction.
- *Support of RESTful services and mashups.* RESTful services are software services that are published on the Web, emphasizing the correct and complete use of the HTTP protocol. Due to its lightweight, simplicity, and performance, REST model has been widely accepted as an alternative to SOAP-based Web services in services provisioning [62,77]. Companies such as Google, Amazon, Twitter and Facebook are increasingly exposing their services as RESTful Web services. In the emerging Web of Things, RESTful Web services are the first choice to implement smart things [42,27]. Some initial efforts towards RESTful services composition have been presented recently [93]. Mashup is another approach to aggregate

multiple services and resources [57]. Challenges of automated RESTful services composition include: (1) an explicit, machine-readable description format to represent RESTful services such as Microformat [34], (2) a composition language specific for RESTful services, that can be used to construct a composite RESTful service and effectively deals with the dynamic, flexible nature of RESTful services, and (3) a state management method, that is able to maintain the state of the application because RESTful services are stateless.

– *Security support of services composition.* Services composition technologies promise cheap and effective means for application integration over the Internet. Beyond the functional aspects, non-functional composition properties such as security and trust are of utmost importance for the adoption of composition technologies [12,69,81]. This is especially the case when compositions happen in open environments such as cloud [58]. There are several security issues (e.g., confidentiality, integrity, privacy, authentication, and authorization) that must be considered to give users the confidence that their data are safely handled. Several specifications for Web service security have been proposed such as WS-Security,[25] WS-Trust,[26] and WS-Federation.[27] However, these specifications have not yet fully found their way to composite Web services. We believe that intensive research and development are needed to support secure and trustworthy services composition.

8. Conclusion

For more than a decade, Web services composition has been an active area of research and development. A wealth of exciting activities including standardization, research, and system developments have been conducted. In this article, we present a comprehensive survey on the state-of-the-art of Web services composition. We abstract a generic model for the life cycle of Web services composition, which is used to compare different research prototypes based a set of assessment criteria. We overview and compare 30 representative research prototypes on Web services composition, from pioneering works conducted at its early stage, to the most recent efforts in this research topic. Additionally, a number of Web services composition standards and services composition platforms are compared. Along with the current research efforts, we identify several open research questions that we hope to stimulate further research in this important area.

Acknowledgments

Quan Z. Sheng's work has been partially supported by Australian Research Council (ARC) Discovery Grants DP0878367 and DP140100104. The authors would like to thank the anonymous reviewers for their valuable feedback on this work.

References

[1] Semantic Web Enabled Composition of Web Services, PhD thesis, Virginia Polytechnic Institute and State University, Falls Church, Virginia, USA, 2004.
[2] Giovanni Acampora, Matteo Gaeta, Vincenzo Loia, Athanasios V. Vasilakos, Interoperable and adaptive fuzzy services for ambient intelligence applications, ACM Trans. Auton. Adapt. Syst. (TAAS) 5 (2) (2010) 8.
[3] Danilo Ardagna, Luciano Baresi, Sara Comai, Marco Comuzzi, Barbara Pernici, A service-based framework for flexible business processes, IEEE Softw. 28 (2) (2011) 61–67.
[4] Karim Baina, Boualem Benatallah, Fabio Casati, Farouk Toumani, Model-driven Web service development, in: Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04), Lecture Notes in Computer Science, vol. 3084, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 42–45.
[5] Fabio Barbon, Paolo Traverso, Marco Pistore, Michele Trainotti, Run-time monitoring of instances and classes of Web service compositions, in: Proceedings of the 4th IEEE International Conference on Web Services (ICWS'06), IEEE Computer Society, Washington, DC, USA, 2006, pp. 63–71.
[6] Maurice Beek, Antonio Bucchiarone, Stefania Gnesi, A survey on service composition approaches: from industrial standards to formal methods, in: Proceedings of the 2nd International Conference on Internet and Web Applications and Services (ICIW '07), IEEE Computer Society, Washington, DC, USA, 2007, pp. 15–20.
[7] Boualem Benatallah, Marlon Dumas, Quan Z. Sheng, Facilitating the rapid development and scalable orchestration of composite Web services, Distrib. Parallel Dat. 17 (1) (2005) 5–37.
[8] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, Fabio Patrizi, Automatic service composition via simulation, Int. J. Found. Comput. Sci. 19 (2) (2008) 429–451.
[9] Daniela Berardi, Giuseppe De Giacomo, Diego Calvanese, Automatic composition of process-based Web services: a challenge, in: Proceedings of the WWW 2005 Workshop on Web Service Semantics: Towards Dynamic Business Integration (WSS 2005), ACM Press, New York, NY, USA, 2005.
[10] Daniela Berardi, Giuseppe De Giacomo, Massimo Mecella, Basis for automatic service composition, in: Tutorial at the 14th International World Wide Web Conference (WWW'05), May 2005.
[11] Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, May 2001, pp. 29–37.
[12] Athman Bouguettaya, Quan Z. Sheng, Florian Daniel (Eds.), Advanced Web Services, Springer, 2013.
[13] Athman Bouguettaya, Quan Z. Sheng, Florian Daniel (Eds.), Web Services Foundations, Springer, 2013.
[14] S. Bourne, C. Szabo, Q.Z. Sheng, Ensuring well-formed conversations between control and operational behaviors of Web services, in: Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC 2012), Springer-Verlag, Berlin, Heidelberg, 2012, pp. 507–515.
[15] J. Bronsted, K.M. Hansen, M. Ingstrup, Service composition issues in pervasive computing, IEEE Pervasive Comput. 9 (1) (2010) 62–70.
[16] Fabio Casati, Ming-Chien Shan, Dynamic and adaptive composition of e-services, Inform. Syst. 26 (3) (2001) 143–162.
[17] Anis Charfi, Tom Dinkelaker, Mira Mezini, A plug-in architecture for self-adaptive Web service compositions, in: Proceedings of the 7th IEEE International Conference on Web Services (ICWS'09), IEEE Computer Society, Washington, DC, USA, 2009, pp. 35–42.
[18] Anis Charfi, Mira Mezini, AO4BPEL: an aspect-oriented extension to BPEL, World Wide Web 10 (3) (2007) 309–344.

[25] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
[26] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html.
[27] http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP.

[19] Massimiliano Colombo, Elisabetta Di Nitto, Marco Mauri, SCENE: a service composition execution environment supporting dynamic changes disciplined through rules, in: Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC'06), Lecture Notes in Computer Science, vol. 4294, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 191–202.
[20] Florian Daniel, Fabio Casati, Boualem Benatallah, Ming-Chien Shan, Hosted universal composition: models, languages and infrastructure in MashArt, in: Proceedings of the 28th International Conference on Conceptual Modeling (ER'09), Lecture Notes in Computer Science, vol. 5829, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 428–443.
[21] Qiang Duan, Yuhong Yan, Athanasios V. Vasilakos, A survey on service-oriented network virtualization toward convergence of networking and cloud computing, IEEE Trans. Network Serv. Manage. 9 (4) (2012) 373–392.
[22] Schahram Dustdar, Wolfgang Schreiner, A survey on Web services composition, Int. J. Web Grid Serv. 1 (1) (2005) 1–30.
[23] D. Fensel, C. Bussler, The Web service modeling framework WSMF, Electron. Comm. Res. Appl. 1 (2) (2002) 113–137.
[24] Roy Thomas Fielding, Architectural styles and the design of network-based software architectures, PhD thesis, 2000.
[25] X. Fu, T. Bultan, J. Su, Synchronizability of conversations among Web services, IEEE Trans. Software Eng. 31 (12) (2005) 1042–1055.
[26] Keita Fujii, Tatsuya Suda, Semantics-based context-aware dynamic service composition, ACM Trans. Auton. Adapt. Syst. 4 (2) (2009) 12:1–12:31.
[27] D. Guinard, V. Trifa, Towards the Web of things: Web mashups for embedded devices, in: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Conjunction with WWW'09, Madrid, Spain, April 2009.
[28] Joyce El Haddad, Maude Manouvrier, Marta Rukoz, TQoS: transactional and QoS-aware selection algorithm for automatic Web service composition, IEEE Trans. Serv. Comput. 3 (1) (2010) 73–85.
[29] Swaroop Kalasapur, Mohan Kumar, Behrooz A. Shirazi, Dynamic service composition in pervasive computing, IEEE Trans. Parallel Distr. Syst. 18 (7) (2007) 907–918.
[30] Markus Keidl, Alfons Kemper, Towards context-aware adaptable Web services, in: Proceedings of the 13th International World Wide Web Conference (WWW'04), ACM Press, New York, NY, USA, 2004, pp. 55–65.
[31] Ravi Khadka, Brahmananda Sapkota, An evaluation of dynamic Web service composition approaches, in: Proceedings of the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2010), SciTePress, Athens, Greece, 2010, pp. 67–79.
[32] Srividya Kona, Ajay Bansal, Luke Simon, Ajay Mallya, Gopal Gupta, Thomas D. Hite, USDL: a service-semantics description language for automatic service discovery and composition, Int. J. Web Serv. Res. 6 (1) (2009) 20–48.
[33] W. Kongdenfha, H.R. Motahari-Nezhad, B. Benatallah, F. Casati, R. Saint-Paul, Mismatch patterns and adaptation aspects: a foundation for rapid development of Web service adapters, IEEE Trans. Serv. Comput. 2 (2) (2009) 94–107.
[34] Jacek Kopecký, Karthik Gomadam, Tomas Vitvar, hRESTS: an HTML microformat for describing RESTful Web services, in: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'08), IEEE Computer Society, Washington, DC, USA, 2008, pp. 619–625.
[35] Amaia Lazcano, Gustavo Alonso, Heiko Schuldt, C. Schuler, The WISE approach to electronic commerce, J. Comput. Syst. Sci. Eng. 15 (5) (2000) 345–364.
[36] F. Lécué, Y. Gorronogoitia, R. Gonzalez, M. Radzimski, M. Villa, SOA4All: an innovative integrated approach to services composition, in: Proceedings of the 8th IEEE International Conference on Web Services (ICWS'10), IEEE Computer Society, Washington, DC, USA, 2010, pp. 58–67.
[37] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, Richard B. Scherl, GOLOG: a logic programming language for dynamic domains, J. Logic Program. 31 (1997) 59–83.
[38] Xitong Li, Yushun Fan, Stuart Madnick, Quan Z. Sheng, A pattern-based approach to protocol mediation for Web services composition, Inform. Software Technol. 52 (3) (2010) 304–323.
[39] Xitong Li, Yushun Fan, Quan Z. Sheng, Zakaria Maamar, Hongwei Zhu, A petri-net approach to analyzing behavioral compatibility and similarity of Web services, IEEE Trans. Syst. Man Cybernet. Syst. 41 (3) (2011) 510–521.
[40] Benchaphon Limthanmaphon, Yanchun Zhang, Web service composition transaction management, in: Proceedings of the 15th Australasian database conference (ADC'04), Australian Computer Society, Inc., Darlinghurst, Australia, 2004, pp. 171–179.
[41] M. Schmidt, M. Klusch, A. Gerber, Semantic Web service composition planning with OWLS-XPlan, in: Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web, AAAI Press, 2005, pp. 55–62.
[42] Sujith S. Matthew, Yacine Atif, Quan Z. Sheng, Zakaria Maamar, The Web of things: challenges and enabling technologies, in: Nik Bessis, Fatos Xhafa, Dora Vaarvaigou, Richard Hill, Maozhen Li (Eds.), Internet of Things and Inter-Cooperative Computational Technologies for Collective Intelligence, Springer Verlag, 2013.
[43] Andrea Maurino, Enrico Mussi, Stefano Modafferi, Barbara Pernici, The MAIS framework for composite Web services, Int. J. Interoperab. Bus. Inform. Syst. 6 (2007) 32–64.
[44] D.V. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – The Planning Domain Definition Language, Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.
[45] Drew V. McDermott, Estimated-regression planning for interactions with Web services, in: Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002), AAAI Press, 2002.
[46] Sheila Mcilraith, Adapting golog for composition of semantic Web services, in: Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR'02), Morgan Kaufmann, 2002, pp. 482–493.
[47] Brahim Medjahed, Athman Bouguettaya, A multilevel composability model for semantic Web services, IEEE Trans. Knowl. Data Eng. 17 (7) (2005) 954–968.
[48] Brahim Medjahed, Athman Bouguettaya, Ahmed K. Elmagarmid, Composing Web services on the semantic Web, VLDB J. 12 (4) (2003) 333–351.
[49] Nikolay Mehandjiev, Freddy Lécué, Martin Carpenter, Fethi A. Rabhi, Cooperative service composition, in: Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012), Springer-Verlag, Berlin, Heidelberg, 2012, pp. 111–126.
[50] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, Schahram Dustdar, End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo, IEEE Trans. Serv. Comput. 3 (3) (2010) 193–205.
[51] Nikola Milanovic, Miroslaw Malek, Current solutions for Web service composition, IEEE Int. Comput. 8 (6) (2004) 51–59.
[52] John A. Miller, Devanand Palaniswami, Amit P. Sheth, Krys J. Kochut, Harvinder Singh, WebWork: METEOR's Web-based workflow management system, J. Intell. Inform. Manage. Syst. 10 (2) (1998) 185–215.
[53] Oliver Moser, Florian Rosenberg, Schahram Dustdar, Non-intrusive monitoring and service adaptation for WS-BPEL, in: Proceedings of the 17th International Conference on World Wide Web (WWW'08), ACM Press, New York, NY, USA, 2008, pp. 815–824.
[54] Hamid Reza Motahari Nezhad, Guang Yuan Xu, Boualem Benatallah, Protocol-aware matching of Web service interfaces for adapter development, in: Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, North Carolina, USA, 2010.
[55] Srini Narayanan, Sheila A. McIlraith, Simulation, verification and automated composition of Web services, in: Proceedings of the 11th International Conference on World Wide Web (WWW'02), ACM Press, New York, NY, USA, 2002, pp. 77–88.
[56] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, Fusun Yaman, SHOP2: an HTN planning system, J. Artif. Intell. Res. 20 (1) (2003) 379–404.
[57] Anne H.H. Ngu, Michael P. Carlson, Quan Z. Sheng, Hye-young Paik, Semantic-based mashup of composite applications, IEEE Trans. Serv. Comput. 3 (1) (2010) 2–15.
[58] Talal H. Noor, Quan Z. Sheng, Sherali Zeadally, Jian Yu, Trust management of services in cloud environments: obstacles and solutions, ACM Computing Surveys (CSUR), 46 (1) (2013) 12.
[59] Bart Orriëns, Jian Yang, Mike P. Papazoglou, A rule driven approach for developing adaptive service oriented business collaboration, in: Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC'05), Amsterdam, The Netherlands, December 2005, pp. 61–72.
[60] Mike P. Papazoglou, Willem-Jan Heuvel, Service oriented architectures: approaches, technologies and research issues, VLDB J. 16 (3) (2007) 389–415.

[61] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, Service-oriented computing: a research roadmap, Int. J. Coop. Inform. Syst. 17 (2) (2008) 223–255.
[62] C. Pautasso, O. Zimmermann, F. Leymann, Restful Web services vs. "Big" Web services: making the right architectural decision, in: Proceedings of the 17th International World Wide Web Conference (WWW 2008), Beijing, China, 2008.
[63] C. Peltz, Web services orchestration and choreography, Computer 36 (10) (2003) 46–52.
[64] Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Barbara Pernici, Sandeep Yadav, MicroMAIS: executing and orchestrating Web services on constrained mobile devices, Software – Pract. Exp. 42 (9) (2012) 1075–1094.
[65] Shankar R. Ponnekanti, Armando Fox, SWORD: a developer toolkit for Web service composition, in: Proceedings of the 11th International World Wide Web Conference (WWW'02), May 2002, pp. 83–107.
[66] M.R. Rahimi, N. Venkatasubramanian, S. Mehrotra, A.V. Vasilakos, MAPCloud: mobile applications on an elastic and scalable 2-tier cloud architecture, in: IEEE Fifth International Conference on Utility and Cloud Computing (UCC 2012), 2012, pp. 83–90.
[67] Jinghai Rao, Xiaomeng Su, A survey of automated Web service composition methods, in: Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), Springer-Verlag, Berlin, Heidelberg, 2004, pp. 43–54.
[68] Anders Ravn, Jiri Srba, Saleem Vighio, A formal analysis of the Web services atomic transaction protocol with UPPAAL, in: Proceedings of the 4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation, 2010, pp. 579–593.
[69] Fumiko Satoh, Takehiro Tokuda, Security policy composition for composite Web services, IEEE Trans. Serv. Comput. 4 (4) (2011) 314–327.
[70] Q.Z. Sheng, Z. Maamar, H. Yahyaoui, J. Bentahar, K. Boukadi, Separating operational and control behaviors: a new approach to Web services modeling, IEEE Int. Comput. 14 (3) (2010) 68–76.
[71] Quan Z. Sheng, Composite Web Services Provisioning in Dynamic Environments, PhD thesis, The University of New South Wales, Sydney, NSW, Australia, 2006.
[72] Quan Z. Sheng, Boualem Benatallah, Marlon Dumas, Eileen Mak, SELF-SERV: a platform for rapid composition of Web services in a peer-to-peer environment, in: Proceedings of the 28th International Conference on Very Large Databases (VLDB'02), Morgan Kaufmann, 2002, pp. 1051–1054.
[73] Quan Z. Sheng, Boualem Benatallah, Zakaria Maamar, Marlon Dumas, Anne H.H. Ngu, Enabling personalized composition and adaptive provisioning of Web services, in: Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04), Lecture Notes in Computer Science, vol. 3084, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 322–337.
[74] Quan Z. Sheng, Boualem Benatallah, Zakaria Maamar, Anne H.H. Ngu, Configurable composition and adaptive provisioning of Web services, IEEE Trans. Serv. Comput. 2 (1) (2009) 34–49.
[75] Quan Z. Sheng, Zakaria Maamar, Lina Yao, Claudia Szabo, Scott Bourne, Behavior modeling and automated verification of Web services, Inform. Sci. 258 (2014) 416–433.
[76] Quan Z. Sheng, Jian Yu, Schahram Dustdar (Eds.), Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, CRC Press, 2010.
[77] Amit P. Sheth, Karthik Gomadam, Jon Lathem, SA-REST: semantically interoperable and easier-to-use services and mashups, IEEE Int. Comput. 11 (6) (2007) 91–94.
[78] Amit P. Sheth, Karthik Gomadam, Ajith Ranabahu, Semantics enhanced services: METEOR-S, SAWSDL and SA-REST, IEEE Data Eng. Bull. 31 (3) (2008) 8–12.
[79] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, Dana Nau, HTN planning for Web service composition using SHOP2, Web Seman.: Sci. Serv. Agents World Wide Web 1 (4) (2004) 377–396.
[80] David Skogan, Roy Gronmo, Ida Solheim, Web service composition in UML, in: Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC'04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 47–57.
[81] Halvard Skogsrud, Boualem Benatallah, Fabio Casati, Model-driven trust negotiation for Web services, IEEE Int. Comput. 7 (6) (2003) 45–52.
[82] Shirin Sohrabi, Nataliya Prokoshyna, Sheila A. Mcilraith, Web service composition via the customization of Golog programs with user preferences, in: Conceptual Modeling: Foundations and Applications, Lecture Notes in Computer Science, vol. 5600, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 319–334.
[83] Biplav Srivastava, Jana Koehler, Web service composition – current solutions and open problems, in: Proceedings of the ICAPS 2003 Workshop on Planning for Web Services, AAAI Press, 2003, pp. 28–35.
[84] Michele Trainotti, Marco Pistore, Gaetano Calabrese, Gabriele Zacco, Gigi Lucchese, Fabio Barbon, Piergiorgio Bertoli, Paolo Traverso, ASTRO: supporting composition and execution of Web services, in: Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC'05), Lecture Notes in Computer Science, vol. 3826, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 495–501.
[85] Jeffrey D. Ullman, Jennifer Widom, A First Course in Database Systems, Prentice-Hall, 1997.
[86] Debra VanderMeer, Anindya Datta, Kaushik Dutta, Helen Thomas, Krithi Ramamritham, Shamkant B. Navathe, FUSION: a system allowing dynamic Web service composition and automatic execution, in: Proceedings of the IEEE International Conference on E-Commerce (CEC'03), IEEE Computer Society, Washington, DC, USA, 2003, pp. 399–404.
[87] Yao Wang, Julita Vassileva, A review on trust and reputation for Web service selection, in: Proceedings of ICDCS 2007 Workshop on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2007, pp. 25–32.
[88] Ingo Weber, Hye-Young Paik, Boualem Benatallah, Form-based Web service composition for domain experts, ACM Trans. Web 8 (1) (2013) 1–40.
[89] Jian Yu, Jun Han, Quan Z. Sheng, Steven O. Gunarso, PerCAS: an approach to enabling dynamic and personalized adaptation for context-aware services, in: Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC 2012), Springer-Verlag, Berlin, Heidelberg, 2012, pp. 173–190.
[90] Qi Yu, Xumin Liu, Athman Bouguettaya, Brahim Medjahed, Deploying and managing Web services: issues, solutions, and directions, VLDB J. 17 (3) (2008) 537–572.
[91] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng, Quality driven Web services composition, in: Proc. of the 12th International World Wide Web Conference (WWW'03), Budapest, Hungary, May 2003.
[92] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, Henry Chang, QoS-aware middleware for Web service composition, IEEE Trans. Software Eng. 30 (5) (2004) 311–327.
[93] Haibo Zhao, Prashant Doshi, Towards automated restful Web service composition, in: Proceedings of the 7th IEEE International Conference on Web Services (ICWS'09), IEEE Computer Society, Washington, DC, USA, 2009, pp. 189–196.
[94] Zibin Zheng, Michael R. Lyu, Personalized reliability prediction of Web services, ACM Trans. Software Eng. Methodol. 22 (2) (2013) 12.