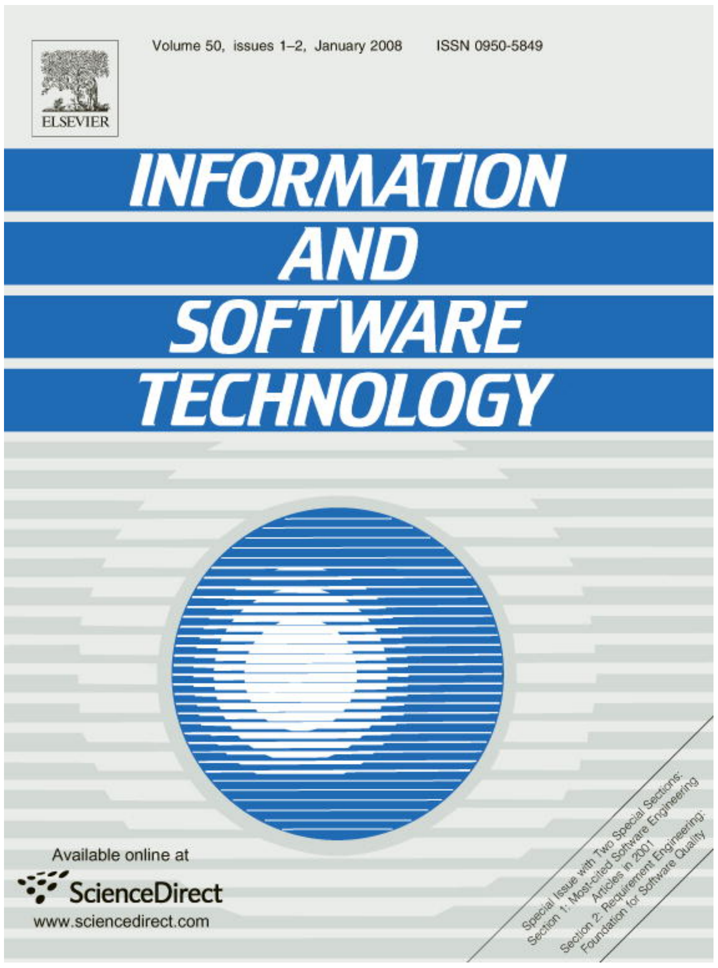
Volume 50, issues 1–2, January 2008
ISSN 0950-5849
ELSEVIER
INFORMATION AND SOFTWARE TECHNOLOGY
Available online at
ScienceDirect
www.sciencedirect.com
Special Issue with Two Special Sections:
Section 1: Most-cited Software Engineering Articles in 2001
Section 2: Requirement Engineering: Foundation for Software Quality

Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 22–35
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

On adopting Content-Based Routing in service-oriented architectures

Gianpaolo Cugola, Elisabetta Di Nitto [*]

Politecnico di Milano, Dip. di Elettronica e Informazione, via Golgi 40, 20133 Milano, Italy

Available online 16 October 2007

Abstract

Two requirements of SOAs are the need for a global discovery agency, which assists requesters in finding their required services, and the need for new interaction paradigms, which overcome the limitations of the usual request/reply style. Content-Based Routing (CBR) holds the promise of addressing both these aspects with a single technology and a single routing infrastructure. To provide arguments for our hypothesis, we review the on going efforts for service retrieval and asynchronous communication in SOAs, identify their limitations and the advantages, and discuss how incorporating CBR into SOAs allows to solve most limitations, but also poses some interesting challenges.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Service-oriented architectures; Content-Based Routing; Publish–subscribe; Query–advertise

1. Introduction

In our previous work [22] we discussed about the usage of publish–subscribe to develop a complex application (i.e., a workflow management system). In this paper we study how a generalization of publish–subscribe, Content-Based Routing, can help in the development of Service-Oriented Architectures (SOAs).

SOAs are attracting more and more interest by researchers and practitioners thanks to their ability to dramatically increase the interoperability among different software systems. The main roles of SOAs are *services* that provide operations to the external world and *requesters* that exploit these services. The two are usually decoupled from each other and can even be played by components owned by different organizations. Thanks to the technological stack behind SOAs, in fact, service requests are designed to pass across the boundaries of single organizations, thus enabling the realization of open software systems [8], where the various actors can cooperate in a large scale, loosely coupled environment. In this scenario, a third important actor is the *discovery agency*. It supports services and requesters in meeting each other: services publish their *description*, thus making available to requesters the information needed to access them, while, in turn, requesters exploit discovery agencies to find those services that satisfy their needs. Such a discovery of services and the subsequent *binding* to some of them does not necessarily happen at design time. It can be performed even at run-time on the basis of the specific needs the requester has at the time it performs its request.

In principle, all the aforementioned aspects enable the realization of a global and open market of services where each *service provider* can offer its services and each requester can find what it needs at design but also at run-time. In this paper we focus on two key aspects for the realization of this vision:

- the need for a global discovery agency to allow requesters to find proper services on an Internet scale, and
- the need for new styles of interaction among services that could: (a) extend the adoption of SOAs to application domains requiring not only a purely proactive, request/reply interaction style, but also more asynchronous, reactive approaches, and (b) reduce the coupling among services, enabling the realization of applications capable of adapting to the changes that frequently happen in a global, open world.

[*] Corresponding author. Tel.: +39 0223991; fax: +39 0223993574.
E-mail addresses: cugola@elet.polimi.it (G. Cugola), dinitto@elet.polimi.it (E. Di Nitto).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.10.004

In particular, we argue that *Content-Based Routing (CBR)* has the potential of becoming the technology to address both aspects, enabling global service retrieval and large-scale asynchronous interaction in SOAs, *with a single technology and a single routing infrastructure.*

CBR differs from classical routing in that messages are addressed based on their content instead of their destination. In conventional systems the sender explicitly specifies the intended message recipients using a unicast or multicast address, while in CBR the sender simply injects the message in the network, which determines how to route it according to its content. CBR is at the core of many classes of systems and two of them are relevant here: *query–advertise* and *publish–subscribe*. The former allows large bodies of data to be searched and retrieved. This is usually obtained by exploiting the actual content of queries to route them only toward those stores potentially holding matching data. Similarly, the components of a publish–subscribe application publish messages that are routed, based on their content, only toward those that subscribed to receive them.

To support our claim about the potential of CBR, we first describes the main aspects of SOAs and CBR (in Section 2), then we carefully review the current state of the art in supporting asynchronous, publish–subscribe interactions among services and in large-scale service discovery (in Sections 3 and 4, respectively), identifying the strengths and limitations of current approaches. In Section 5 we describe our approach on adopting CBR as the enabling technology to support both large-scale service discovery and publish–subscribe interactions in SOAs, and discuss the opportunities and challenges that we derived from this experience. Finally, in Section 6 we draw some conclusions and present an agenda for future development.

2. Background in SOA and CBR

In this section we briefly describe service-oriented architectures and Content-Based Routing approaches, identifying the main peculiarities of such technologies.

2.1. Service-oriented architectures

Service-Oriented Architectures have various incarnations focusing on different application domains. Common examples are OSGI (Open Grid Services Infrastructure) [82], a platform supporting the development of grid services; JXTA [79], a peer-to-peer middleware that allows all connected devices on the network to collaborate and communicate as peers; and Jini [77], a framework supporting the development of services in a pervasive environment. However, the most known implementation of SOAs is based on *web services*.[1]

[1] From now on we will use the term SOA to refer mainly to its web service-based implementation.

W3C defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [12].

The main element that characterizes a service is the explicit definition of its interface in WSDL (Web Service Description Language) [19]. It allows designers to define the syntactical interface of a web service in terms of the set of *operations* it exports. Each operation can be invoked by sending the corresponding message to the web service. Standard operations (defined as `in-out` in [18]) produce a result for the caller and should be invoked synchronously, in a request/reply style of interaction, while others, which do not produce any result, can be defined as `in-only` and invoked asynchronously.

SOAP [38] is a protocol, based on XML, that defines the way messages (operation calls) are actually exchanged among web services. It is usually based on HTTP but other transport protocols can be used (e.g., SMTP). It also defines several types of message exchange patterns (like one-way and request-response) that can be used to implement the different kind of interactions defined at the level of the WSDL interface (e.g., `in-out` and `in-only`).

Besides WSDL and SOAP, the third standard that appears to be essential for building applications based on web services is the Universal Description Discovery & Integration (UDDI) [20]. It focuses on the characteristics that a discovery agency (called *registry* in the web services jargon) should have and the operations it should offer to support publication and discovery of services. Fig. 1 shows the main concepts defined by UDDI. A *businessEntity* owns or requires certain kinds of *businessServices* that, in turn, are implemented by some concrete services identified by one or more *bindingTemplates* that, finally, can refer to some *tModels* that encapsulate any kind of specific information about these services. A UDDI registry holds information about such entities allowing them to be efficiently searched by clients. As for replication and distribution, UDDI defines the basic mechanisms to replicate a single logical registry on different nodes, mainly to increase fault tolerance. Moreover, the last UDDI specification introduces the concept of *promotion*, as the mechanism to explicitly copy entities from

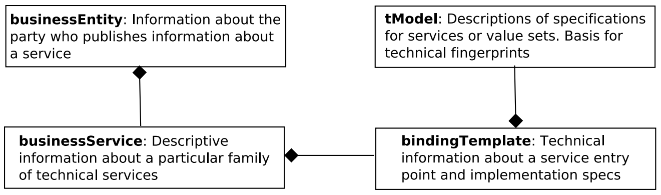
businessEntity: Information about the party who publishes information about a service
tModel: Descriptions of specifications for services or value sets. Basis for technical fingerprints
businessService: Descriptive information about a particular family of technical services
bindingTemplate: Technical information about a service entry point and implementation specs

Fig. 1. UDDI registries data structure (derived from [20]).

a registry to another. The specification suggests that to make this happen in a safe way, registries should be organized in a hierarchy, where a root registry has the control on key generation and the other registries, named *affiliates*, exploit such control. If this allows to publish the description of a service on multiple registries, it does not support a truly distributed registry. In particular, UDDI does not define any mechanism to propagate discovery queries toward the site where the corresponding information is stored. Thus, the efficacy of queries depends on the completeness of the information stored on the registry that is being contacted (which can be increased by a careful use of the promotion mechanism), on the precision of the query, and also on the ability of the requester to explicitly query different registries.

The real challenging aspect of SOAs in general and web services in particular is the possibility of composing services (that could have been identified through a registry) to create complex applications, possibly exposing themselves, again, as services. These compositions are often developed by exploiting workflow languages allowing the flow between calls to various services to be precisely defined. The main standard in this area is the Business Process Execution Language (BPEL) [3]. It offers all main constructs typical of workflow languages and supports two main interaction paradigms with the component services: request/reply and notifications. While request/reply allows a service to invoke another under the circumstances defined by the BPEL model, the notification mechanism allows a branch of the workflow (starting with a specific construct called `pick`) to be executed on the receipt of a notification message from a component service.

Fig. 2 summarizes the main actors of a SOA and identifies the most standards we have presented so far. Other interesting standard proposals for web services are so called WS* standards. These are being developed to address a wide number of issues ranging from security [51] to the definition of Service Level Agreements (SLAs) [43].

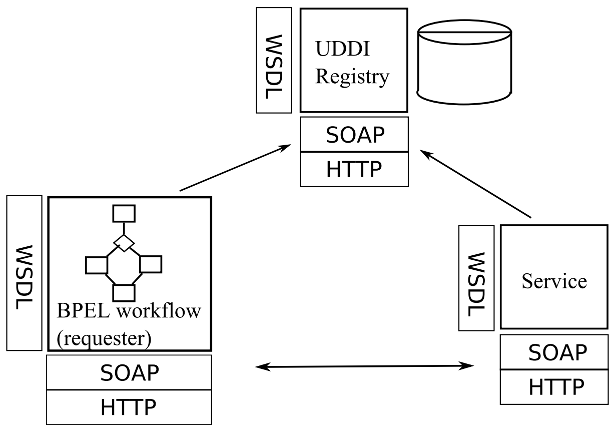

Fig. 2. The main elements that characterize SOAs built using web services.

2.2. Publish–subscribe, query–advertise, and Content-Based Routing

Publish–subscribe is a popular style of interaction among the components of a distributed application, which are allowed to *publish* messages and to *subscribe* to the messages they are interested in. A special component of the architecture, the *dispatcher*, is in charge of routing messages from publishers to the interested subscribers. The popularity of such style is motivated by the strong decoupling among components that it provides, especially when compared to more traditional approaches like client-server. The publishers do not need to know the identity and number of the subscribers they interact with and vice versa. This allows to easily change the architecture of the application at run-time, by adding new components, removing them, or even moving them from host to host. This flexibility justifies the popularity of the publish–subscribe model and the recent introduction of a wealth of publish–subscribe middleware, each interpreting the publish–subscribe paradigm in a different way [14,29,49].

The first successful publish–subscribe infrastructures adopted a *centralized*, *topic-based* dispatching approach. The topic-based approach is quite simple: subscriptions define the category —usually called topic, subject, or channel—of messages interesting for receivers. Messages have any structure (or no structure at all) and are declared as belonging to a certain category. They are dispatched to those subscribers interested to the corresponding category. These publish–subscribe systems demonstrated to be suitable for supporting enterprise-level development and integration of distributed applications. They, however, were not suited to large scale scenarios due to the use of a centralized dispatcher and the lack of expressiveness of the topic-based subscription language. To address these aspects, commercial and academic efforts have focused on *distributed*, *content-based* dispatchers.

A distributed dispatcher is implemented as a set of *brokers* (see Fig. 3) interconnected in an overlay network, which cooperatively route subscriptions and messages sent by the application components connected to them. A content-based subscription language provides an increased expressiveness by allowing subscribers to use expressions—usually called *filters*—that permit sophisticated matching on the entire message content. The format of

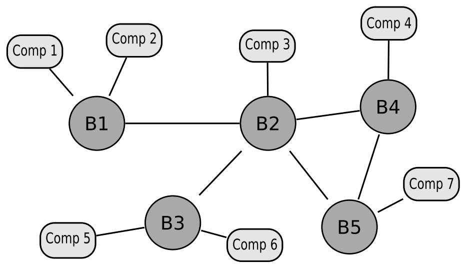

Fig. 3. A publish–subscribe application adopting a distributed dispatcher.

such filters depends on the format of messages. As an example, in case of messages organized as a set of typed fields, it is usually possible to express filters as predicates on such fields, using the standard comparison operators provided by programming languages. So, the filter: `[stockName="Acme'' and price>100]` would match a message including a field `stockName` whose value equals "`Acme`" and a field `price` with a value greater than 100.

Query–advertise is another popular style of interaction among the components of a distributed application, which may *advertise* the data (or, more generically, the resources) they want to share and *query* for the resources they need. The routing infrastructure supporting this paradigm is in charge of transporting queries only toward those components potentially holding the relevant resources, forwarding their replies back. Query–advertise is at the basis of every distributed data-sharing application and several middleware systems support this style of interaction. They range from those, like the system presented in [40], that adapt a publish–subscribe infrastructure to perform query–advertise routing, to the most advanced Distributed Hash Tables (DHTs), i.e., those offering not only the standard (for a DHT) identifier-based lookup mechanism, but also more sophisticated filtering mechanisms to search for data on the basis of its content.

An analysis of publish–subscribe and query–advertise reveals that they represent the two sides of the same coin: publications are routed based on subscriptions, while queries are routed based on advertisements. Such routing is multicast and it is not based on an explicit indication of the recipients of a certain message, but, instead, it depends on the content of such message and on some previously defined association between this content and the interested receivers. It is a *Content-Based Routing* (CBR). What is interesting for us about CBR is that it holds the promise of addressing, *at the same time and with a single routing infrastructure*, the two issues we identified in SOAs: providing support to an asynchronous, publish–subscribe interaction style, and enabling complex searches to be executed in a completely distributed environment.

3. Asynchronous interactions in SOAs: State of the art

The form of interaction typically adopted in SOAs, particularly using web services, is synchronous, point-to-point request/reply. This is a simple, natural, and effective model to build distributed applications. It is well known, however, that modern distributed applications also require other interaction styles. In particular, an asynchronous, multicast interaction model, like that enabled by publish–subscribe, is often required. For instance, in a VoIP application several components (e.g., the call control and the accounting services) must be notified when a phone call is terminated. Similarly, in location-based applications there are often several components interested in knowing when some user has entered a certain area. In similar situations, the polling approach suggested by a request/reply interaction style may become very inefficient.

This is something that has recently become clear among SOA experts. As a result, starting from 2003 several companies focused on developing a specification to introduce asynchronous notifications among web services. The result were two competing proposals: WS-Eventing [13] and WS-Notification [37]. While quite different at the beginning, currently they have a lot in common. In particular, WS-Eventing is very similar to the portion of WS-Notification that defines the interfaces between producers and consumers of events, i.e., WS-Base Notification [35]. The remaining parts of WS-Notification are WS-Brokered Notification [17] and WS-Topics [71].

WS-Base Notification defines the core roles of *notification producer*, *subscription manager*, *subscriber*, and *notification consumer*. To create a subscription a subscriber sends specific information to a notification producer including:

- the consumer's reference to which the producer must send matching notifications;
- the topic expression that identifies the topics the consumer is interested in (see below);
- an optional *selector*, which allows filtering on the content of notifications.

In response, the notification producer returns the identifier of the newly created *subscription* and a reference to the subscription manager in charge of managing it. Through the subscription manager, the subscriber may temporarily pause and resume subscriptions or definitely cancel them.

WS-Topics describes the format of topics, which are organized in trees. This allows components to subscribe to a "super-topic" to match several topics at once. Different languages can be adopted to define topic expressions, including a simple one that allows only root topics to be used and a full one that uses XPath [10] expressions to precisely define the topics of interest. The same happens for the selector. The result is a specification that can support both topic-based and content-based interactions, offering various levels of expressive power for the description of topics and selectors.

The architecture defined by WS-Base Notification in which producers send notification messages directly to consumers can be acceptable in a small, closed world, in which subscribers know publishers and vice versa, but it does not scale to a global, open-world of services. This weakness is removed by the WS-Brokered Notification, which introduces the concept of *notification broker*. It implements both the notification producer and notification consumer interfaces, thus acting as an intermediary that decouples publishers from subscribers taking care of routing notifications. Moreover, a broker can subscribe to another. This way it is possible to create a network of brokers similar to those that compose the dispatcher of a distributed, publish–subscribe middleware, as defined in Section 2.2. Currently, several systems exist that implement the

WS-Notification specification. These include the WebSphere Application Server V6.1 [84], WSRF.NET V3.0 [42], the Globus Toolkit V4 [32], and Apache Muse V2.2 [81]. While WebSphere implements the specification entirely and provides a full fledged broker for WS-Notification, WSRF.NET focuses on the WS-Resource Framework [36] and implements WS-Notification as a way to notify changes in resources. Similarly, the Globus Toolkit and Apache Muse only support WS-Base Notification and WS-Topics.

WS-Messenger [41] is a research prototype, which supports both WS-Notification and WS-Eventing, thus allowing interoperability between the two. On the other hand, what is interesting for us about WS-Messenger, is the fact that it is implemented as a thin software layer using an existing publish–subscribe middleware for message routing. In particular, it can integrate every JMS [78] compliant middleware, thus providing a first solution to integrate existing publish–subscribe solutions in a SOA infrastructure. A similar, while less general, approach is presented in [60], which describes an implementation of WS-Notification based on Meteor [52], a pre-existing publish–subscribe middleware. Similarly, NaradaBrokering [33] is a distributed publish–subscribe middleware whose recent versions support web service interactions by implementing the WS-Eventing specification.

Another sign of the importance of asynchronous interactions among web services was the recent introduction of a new technology to ease service integration and composition: *Event Service Buses* (ESBs) [16]. These aim at becoming the main backbone for SOAs by offering support to the life cycle of services, their deployment, and their composition. In terms of the service interaction paradigm, they usually support the coexistence of both synchronous and asynchronous mechanisms. Asynchronous mechanisms, in particular, are thought as the way to support loosely coupled integration in a business-to-business context that crosses the boundaries of a single organization. Messages in the ESB can follow *itineraries*, i.e., paths between the sender and the recipients, that are computed by so called *CBR services*. These services act as intermediaries. They receive the messages and perform a limited form of Content-Based Routing in the sense that they can inspect a message and route it depending on some predefined *rules* they encapsulate. Such rules can, for instance, dictate that the message cannot be routed until a different message is not received, or they can modify the message before forwarding it to the receiver. A message can pass across various CBR Services before reaching its final destination. A Message-Oriented Middleware (MOM) is usually adopted to move messages through their itineraries (and the corresponding CBR Services). Such middleware either works as a topic-based publish–subscribe middleware or as a message queue, where messages are stored until receivers extract them. Examples of existing ESB are the open source Mule [80] and the well known IBM WebSphere Message Broker [76].

This large body of initiatives shows that the issue of asynchronous interaction in SOAs is recognized as being very important. The proposals based on some form of distributed, publish–subscribe routing infrastructure, seem to address the main concern about scalability that reduces the applicability of centralized solutions in the global scenario we focus. On the other hand, as it has been recently discussed in [47], currently available standards, products, and prototypes still lack the ability of addressing various QoS concerns. As we discuss in the concluding remarks, this will be one of the challenges of the next few years for publish–subscribe and CBR routing in particular.

4. Large-scale service discovery in SOAs: State of the art

The challenge of SOAs and web service technology is the possibility to create systems that exploit services residing far away from each other and run under the control of different organizations. To make this vision possible, proper mechanisms to support discovery of services on a global, Internet scale have to be provided. The UDDI approach that we introduced in Section 2.1 does not meet this objective. As a matter of fact, in [4,5] authors surveyed three approaches for web service discovery: UBR, the global, root UDDI registry available on the Internet, some web services portals, and some standard search engines. From their analysis it emerged that a search engine, namely Google, by allowing the indexing even of those web services that have not explicitly published, provided the best coverage in terms of number of services found, even if with limited precision. On one end this shows the failure of UDDI on a global scale,[2] on the other hand it shows that much has still to be done in the area.

In particular, several aspects need to be improved. The first concerns the need for defining precise service descriptions that enable the usage of sophisticated querying approaches that would enhance the precision of results. The second concerns the need for adopting appropriate querying algorithms. The third concerns the possibility for requesters to find services distributed on a large scale, independently of where and when they have been published.

As for the first and the second issues, various approaches have been defined, both in the semantic web and in the service engineering communities. WSMO [85] and OWL-S [48] leverage the semantic web to allow providers to describe their services by exploiting the concepts defined in proper ontologies. Such enhanced descriptions can be searched through specific discovery mechanisms that provide good performance for small to medium scale scenarios [1]. The interest for this kind of approaches is highlighted by the establishment of a standardization working group called SAWSDL (Semantic Annotations for WSDL) which, at the beginning of 2007, has released

[2] The UBR registry has been shut down at the beginning of 2006.

a first recommendation on a language that extends WSDL to include also semantic information [30].

The SeCSE project [83] advocates an approach based on *facets*, which describe various aspects of a service, ranging from its behavior (expressed through state machines or BPEL fragments), to its QoS characteristics, to the use cases it is able to fulfill, to the test cases it has passed [73]. Thanks to this rich information, various coexisting discovery approaches have been identified that are applicable in the different phases of the life cycle of a service-based system. In particular, it is possible to match service descriptions with use cases defining the high level requirements of users [28]. Similarly, by exploiting the behavioral information about services, it is possible to allow a designer to find those that best suit the architecture she is defining [45]. Finally, at runtime, it is possible to find services that are fully compliant to the structure of the BPEL process in which they have to be exploited [66].

As for the third issue (large scale distribution) we already mentioned how the UDDI approach, at least in its centralized incarnation, is inadequate. A first attempt to overcome this limitations is represented by the WS-Discovery proposal [9]. It is based on the idea of using a protocol based on IP-multicast for service discovery. Clients may send multicast messages to find services by type, scope, or name. While not centralized, this approach is not suited to large scale scenarios and works only on a LAN due to the inherent limitations of IP-multicast.

Large scale scenarios are instead addressed in [62], where a flooding approach is exploited to manage propagation of queries to a network of UDDI registries. This approach can be seen as complementary to the one advocated by the UDDI standard, where the service descriptions can be replicated (i.e., promoted) on different registries. However, even this approach has some limitations coming from the use of flooding and from the need of connecting registries in an explicit overlay network, which has to be configured and maintained by some administrator.

The work presented in [63] describes a platform, part of the WSMO research effort, which try to overcome the limitations above by using a tuple space to share information among registries. While the idea is interesting, we feel that the centralized implementation of the tuple space will quickly become a bottleneck for the system.

A really distributed approach is taken by the Meteor-S project. In this case a peer-to-peer network of UDDI registries, called MWSDI [72], is built by exploiting JXTA as a middleware [79]. From the logical viewpoint, registries are grouped through a *registries ontology*. Each time a new registry connects to the network, it has to identify in such ontology a domain it wants to connect to or it has to create a new domain. Whenever a new service description has to be published, the publishing client identifies a domain and declares its intention to publish the description on that domain. The description is then published on one of the registries belonging to that domain. Discovery happens in a similar way, starting from the selection of a domain and distributing the query to all the registries that belong to that domain.

DIRE [7], the publication infrastructure developed within SeCSE, follows a different philosophy. In this case the basic assumption is that registries are operated by different organizations, each one desiring to publish its own services on the registry it operates. Thus, differently from the Meteor-S case, all registries can host any kind of service descriptions, regardless of their content. To allow queries spanning multiple organizations, registries are connected by REDS (see Section 5.2.1) and exploit the subscription and publication mechanisms made available by this middleware to replicate service descriptions on the network of registries. More specifically, registries can decide to join a *topic* (by subscribing to it). All registries joining the same topic define, implicitly, a federation sharing the same service descriptions. Indeed, when a service description is published on a registry, it is propagated by REDS to the other registries that have subscribed to the same topic. These, in turn, store the service description for a certain time and then discard it (unless it is explicitly renewed). Orthogonally to federations, it is always possible for registries, and for clients, to issue declarations of interest for some content in a service description (this is performed by exploiting REDS content-based subscriptions). In this case, they will receive all newly published descriptions fulfilling their requirements.

The approach presented in [46] can be seen as complementary to both METEOR-S and DIRE. Indeed, it allows service consumers to be notified when a service that respect their expectations is published or modified. This is obtained by leveraging the routing mechanisms provided by Siena, a distributed publish–subscribe middleware, whose subscription language is extended with semantic-enabling concepts. This approach does not explicitly support the execution of specific queries proactively defined by the user, while it can be used as a mechanism to refresh some existing knowledge that the consumer already owns about some services.

An approach that could complement all the others is the one realized in WSRB [2]. It follows the Google philosophy by offering a crawler that collects information from multiple registries, indexing and storing them on a centralized repository. Such an approach, even if promising to ensure that the demand properly meet the offer of services, raises critical concerns in terms of scalability. Google has proven that this limitation can be removed but this has a cost that must be carefully evaluated.

Fig. 4 summarizes the main characteristics of the approaches and infrastructures in terms of the three aspects that we claim are important: languages for service descriptions, approaches for querying, and large scale distribution. As for this last aspect, we notice that the various approaches address it in various ways. Some of them assume that service descriptions are more or less explicitly replicated on various registries, some others that queries are propagated within the network of registries. Again,

Approaches	Service description language	Query approach	Distribution
UDDI	XML-based	Based on XPath	Allowed for service descriptions, not for queries. No middleware support to setup and maintain the registry network.
SeCSE DIRE	Facet-based	Three approaches suitable in different phases of service engineering	Allowed for service descriptions, not for queries. Middleware support to setup and maintain the registry network
WSMO	Semantic-based	Semantic-based	Potentially allowed through a tuple space
WS-Discovery	XML-based. Services have type, name, and scope	Simple matching by type, name, or scope	Allowed for query via multicast. Not applicable to large scale
Query federation	As UDDI	As UDDI	As UDDI for replication of service descriptions. Offers support for query propagation. No middleware support to setup and maintain the registry network
METEOR-S MWSDI	Requires the definition of a shared ontology	Semantic-based	Distribution of both service descriptions and queries decided based on the ontology concepts owned by each registry. Service providers and consumers cannot control the location where service descriptions are stored
Lynch et al.	Requires the definition of a shared ontology	Semantic-based	Supports propagation of service descriptions through a distributed publish-subscribe middleware. It does not offer mechanisms for users to proactively query for services. It allows users to express their interest in the occurrence of some service specification changes
WSRB	No details available	No details available	Supports crawling of service descriptions stored in distributed UDDI and ebXML registries

Fig. 4. Summary of the presented discovery approaches.

such propagation can either we performed explicitly by the user or executed by the platform depending on some criteria. Also, one of the approaches allows users to subscribe for the publication of new descriptions (or updated descriptions) that follow a certain structure. This enables users that have already explored the domain to be informed about updates. Finally, the last approach advocates the use of crawlers “a la Google” to concentrate all the relevant pieces of information in a single place.

Of course all the mentioned approaches offer interesting solutions to the problem. Given the potential high distribution of registries and of users, however, most of them require proper middleware support to ensure that propagation of information (service descriptions, queries, notifications of changes) occur in a scalable and dependable way. As we will discuss in the next section, a CBR infrastructure may enable all the aforementioned propagation models still holding the promise of being scalable if not dependable.

5. The role of CBR in SOA: Opportunities, first experiences, challenges, and open issues

5.1. Opportunities

As mentioned in the introduction, SOAs propose a vision of distributed applications and related markets that is both “global” and “open” [8]. Service providers are expected to offer their products in a global marketplace, accessed by developers at design-time to search for the components they need. If this was not enough, the SOA’s vision also promotes dynamism at run-time, with applications capable of self-organizing by accessing the global marketplace of services at run-time to choose those that better fit the situations they encounter, thus dynamically adapting to changes in the external environment. We argued that the vision above requires an efficient and scalable service discovery infrastructure to allow different organizations to offer and access services globally, complemented by a publish–subscribe infrastructure to suit the needs of those systems that have a inherently asynchronous behavior and to enable the ability of monitoring the environment and reacting to its changes by reorganizing the structure of the application at run-time.

Many application domains could benefit from the aforementioned aspects, including:

- Grid computing, where the need for efficient discovery of services and resources is of paramount importance.
- Business-to-business interaction, which usually concerns the execution of complex workflows, whose natural style is asynchronous and reactive.

- Pervasive systems, where different living environments encapsulate services that could range from controllers of the temperature in a room to locators of objects and people, etc. In many of these cases there is a need for flexible discovery approaches, which not only allow for distributing queries on a large scale, but also for distributing queries depending on the context. For example, in some application there might be no interest in knowing the temperature of a room where the user is not located at the moment, while there might be an interest in accessing all the location services worldwide to look for some person who has lost the contact with his/her relatives.

This is exactly where CBR may enter the picture by holding the promise of addressing, with a single technology and a single routing infrastructure, the scalability and expressiveness requirements of service discovery and publish–subscribe interaction among services.

5.2. First experiences

To provide an initial assessment of the claim above, we adopted the layered approach depicted in Fig. 5, by using REDS [25], a CBR middleware developed at Politecnico di Milano, to implement both a WS-Notification compliant publish–subscribe infrastructure and a distributed UDDI registry.

5.2.1. REDS

Born from the ashes of Jedi (the publish–subscribe middleware presented in our previous paper [22]) as a distributed, content-based, publish–subscribe middleware, REDS is more than that. It is a framework (in the object-oriented sense) of Java classes to easily build a modular CBR infrastructure.

The first peculiarity of REDS is the possibility it offers to system programmers of defining their own filters and messages by implementing the `Filter` and `Message` interfaces, respectively. This way REDS can operate as a publish–subscribe middleware, using filters to encode the interests of components (subscriptions) and messages to notify relevant events; but also as a query–advertise middleware, using filters to specify the resources held by each component (advertisements) and messages to encode queries. Moreover, REDS, differently from the other similar middleware, natively offers the possibility of replying to messages [34]. This feature can be very useful to transfer the result of a query from the resource owner to the requester, thus completely addressing the requirements of query–advertise.

Another key feature of REDS is the internal structure of its brokers, which are organized as a set of modules that implement well-defined interfaces and encapsulate the major aspects of CBR (see Fig. 6). By choosing the right implementation for each module, developers may adapt the REDS behavior to their needs. In particular, the `RoutingStrategy` can be changed to adapt to different scenarios, from small to large scale. Similarly, the `SubscriptionTable` can be changed to implement different algorithms to store filters and match them against incoming messages, e.g., to optimize routing of different type of messages and filters, from the simplest to the most complex ones (like those based on XML and XPath, which are required in the web services world).

The last feature of REDS is its ability of reconfiguring the topology of the routing network at run-time. In particular, the `TopologyManager` encapsulates the strategy and protocol used to maintain the overlay network of REDS brokers. It answers the requests coming from the application layer to add or remove brokers and reacts to changes in the networking environment (e.g., failing links) and to failures occurring in the REDS network itself (e.g., failing brokers). Analogously, the `Reconfigurator` is the component in charge of restoring subscription information and recovering lost messages when the topology of the

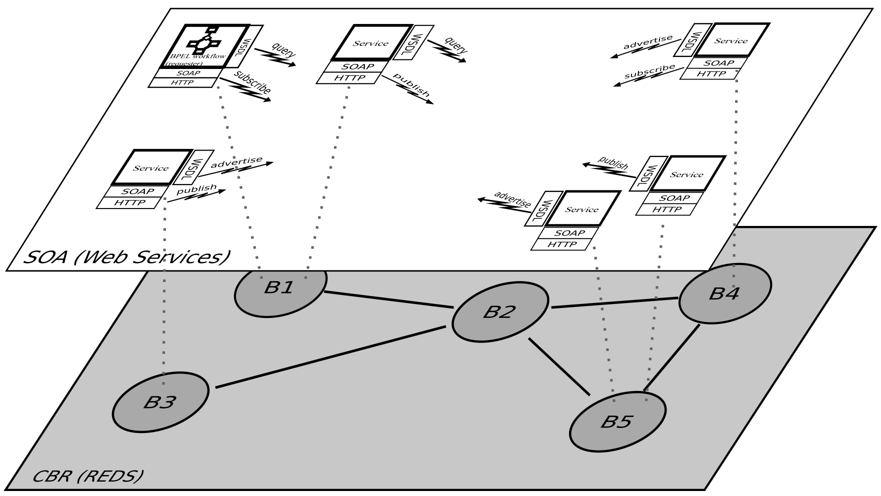

Fig. 5. Using a CBR infrastructure for discovery and publish–subscribe interaction.

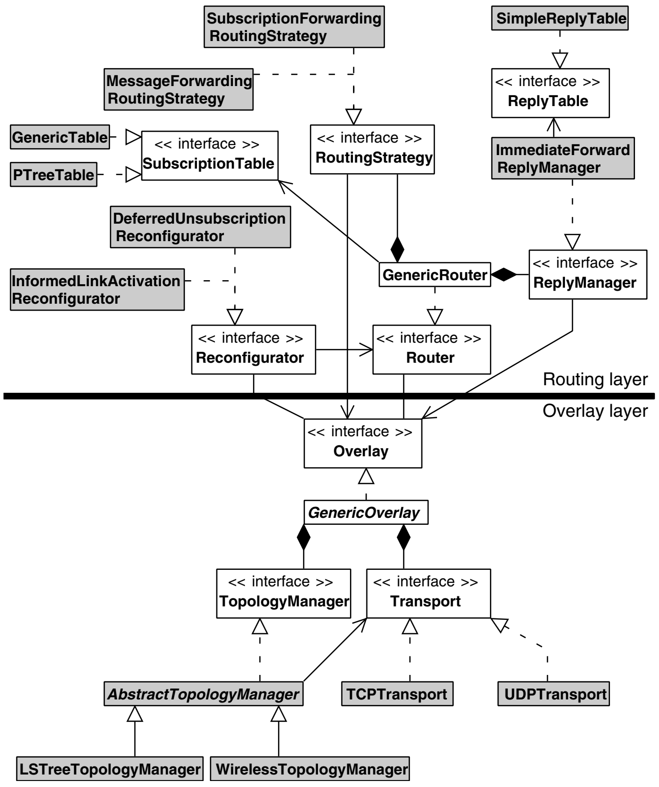
SubscriptionForwarding RoutingStrategy
MessageForwarding RoutingStrategy
SimpleReplyTable
<< interface >> ReplyTable
GenericTable
PTreeTable
<< interface >> SubscriptionTable
<< interface >> RoutingStrategy
ImmediateForward ReplyManager
DeferredUnsubscription Reconfigurator
InformedLinkActivation Reconfigurator
GenericRouter
<< interface >> ReplyManager
<< interface >> Reconfigurator
<< interface >> Router
Routing layer
Overlay layer
<< interface >> Overlay
GenericOverlay
<< interface >> TopologyManager
<< interface >> Transport
AbstractTopologyManager
TCPTransport
UDPTransport
LSTreeTopologyManager
WirelessTopologyManager

Fig. 6. The architecture of a REDS broker.

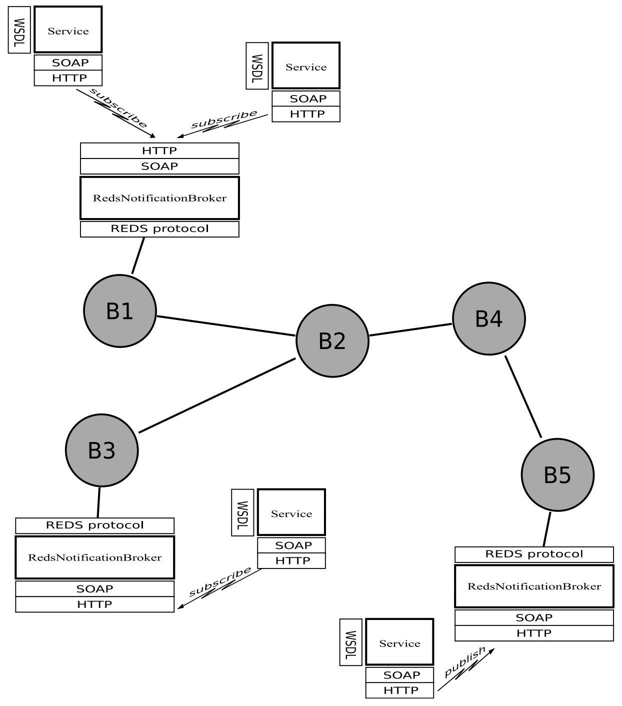

Fig. 7. Using REDS to implement a distributed WS-Notification broker.

REDS network changes, thus ensuring correct routing of messages during reconfiguration. By choosing the right implementation for these two modules, system developers may adapt REDS to different environments, from the very static to the most dynamic ones, including those involving mobile nodes and wireless connections [24].

5.2.2. WS-notification with REDS

Implementing the WS-Notification specification in REDS was a relatively simple task. We started by realizing a `XMLMessage` and a `XPathFilter` implementing the REDS `Message` and `Filter` interfaces, respectively. They allow REDS to correctly manage the data structures used by WS-Notification.

To translate between the WS-Notification and the REDS interface, we built a `RedsNotificationBroker`. It implements the WS-Brokered Notification specification, while at the same time it acts as a REDS client, connected to a broker to access the CBR services provided by REDS. Indeed, as shown in Fig. 7, to provide a fully distributed and scalable WS-Notification service, we instantiate several `RedsNotificationBroker` instances, one for each REDS broker that wants also to act as a WS-Notification entry point.

When a web service acting as a subscriber connects to a `RedsNotificationBroker` and issues a subscription, the `RedsNotificationBroker` subscribes to the REDS dispatching infrastructure by specifying a `XPathFilter` that encodes the interests of the subscriber. Similarly, a notification producer (in the WS-Notification jargon) may access a `RedsNotificationBroker` connected to the other side of the REDS network to notify events. When this happens a `XMLMessage` is created encoding the notification and it is routed by REDS toward the subscribed web services. This provides an elegant and efficient implementation of the WS-Notification specification, which builds upon the characteristics of REDS to support large-scale, dynamic scenarios, including those involving mobile nodes.

Finally, the resulting system allows designers to build hybrid systems where services following the WS-* standards may coexist with others that are compatible with the REDS default protocol.

5.2.3. Dynamic service discovery with REDS

To implement a scalable and flexible service discovery infrastructure for web services we followed a very similar route, but this time we used REDS as a query–advertise infrastructure. More specifically, we extended the work described in [7], leaving behind the idea of federating a set of existing UDDI registries, to implement a single, fully distributed, UDDI registry, potentially capable of supporting global service discovery.

As before, we started by building a `XMLFilter` that implements the REDS `Filter` interface, plus a `UDDIInquiryMessage` and a `XMLMessage`, both implementing the REDS `Message` interface. They are used, respectively, to encode in REDS the advertisements about new services (expressed in XML using the UDDI schemas), the queries issued by service browsers (expressed as UDDI "inquiries" [20]), and the replies to such queries (expressed in XML).

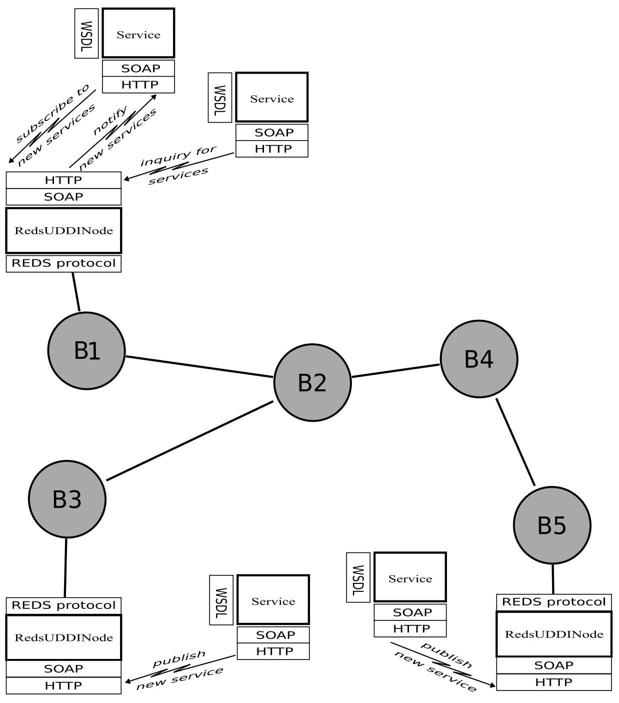

Fig. 8. Using REDS to implement a distributed UDDI registry.

As a gateway between the REDS and the web services worlds, we built a `RedsUDDINode`. As shown in Fig. 8, several instances of such component, connected by REDS, build a distributed UDDI registry [20].

In particular, when an organization wants to publish some information about the services it provides, it connects to a `RedsUDDINode`, which translates the provided information as a REDS `XMLFilter`. Similarly, when an organization needs to find a specific service, it connects to the closest `RedsUDDINode` and issues an "inquiry". This is translated by the `RedsUDDINode` in a `UDDIInquiryMessage`, which is dispatched by REDS toward the `RedsUDDINode` instances possibly holding some relevant information (i.e., those nodes that previously subscribed with a `XMLFilter` that matches the `UDDIInquiryMessage`). Finally, replies are encoded as `XMLMessages` and transported by REDS toward the inquiring node. As mentioned, this realizes a fully distributed UDDI registry,[3] which inherits scalability and flexibility from REDS.

To better show the advantages of using a CBR infrastructures as the underlying routing facility, we added an additional feature to our `RedsUDDINode`, which may act as a notification producer to allow organizations to be promptly notified about new services. In fact, when a new service description is published, not only the `RedsUDDINode` subscribes to the REDS infrastructure with an appropriate `XMLFilter`, as described above, but it also publishes a `XMLMessage` encoding the same information. This message is routed by REDS toward the subscribing `RedsUDDINode` instances, where it is used to notify the interested entities (e.g., the topmost service in Fig. 8) about the publication of the new service. This second mechanism is similar to the one presented in [46].

[3] Actually, since providing a full implementation of the UDDI specification was not our main goal, our current prototype focuses on the core publishing and searching services, only.

5.3. Discussion

While the research described above is still in progress, with more and more features added to our prototypes, the preliminary experience we collected so far allows us to formulate some considerations that are worth sharing.

First of all we confirmed our intuition that the content-based nature of CBR (and REDS in particular) nicely fits the requirements of both publish–subscribe interaction and service discovery. Fully content-based WS-Notification subscriptions and messages can be easily managed by our `RedsNotificationBroker` (i.e., routed by REDS), while precise queries for exactly the services required can be formulated via our `RedsUDDINode`, which routes them toward the right service providers. While we have not yet used our prototypes on real, large scale scenarios, the good performance of most advanced CBR solutions (including REDS) suggests that a WS-Notification broker and a distributed registry built on top of them will perform well, providing good scalability. Moreover, by building our prototypes on top of REDS we could also take advantage of its features of reconfigurability and adaptability to changes coming from the networking environment[24]. As a result, our prototypes are able to operate in dynamic environments, like those involving mobile nodes and wireless links, and, in some sense, this is something that we obtained for free, thanks to the layered approach we adopted.

If these are the positive aspects, we also learned that there are negative ones. In particular, in designing and implementing our prototypes, we experienced how the web services world poses several challenges in front of the CBR experts. Among them three are major and are worth discussing here: one has to do with matching, the other with security, the last with reconfigurability.

The matching challenge originates from a simple consideration: today the web services world is strongly XML-based and in the future it might rely on rich ontologies to define the meaning of all the published items, including web services. At the same time, the research in the area of CBR has usually considered simpler messages and filters, with the former typically organized as a set of typed fields and the latter expressed as predicates on such fields. While several works (e.g., see [69,27,15,55,65,57]) already addressed the problem of efficiently filtering a stream of XML documents using XPath [10] and XQuery [11] expressions, more remains to do to provide the level of scalability usually offered by the best CBR systems. This includes defining a suitable mechanism to avoid duplicating the effort of XML parsing and matching at each step along the CBR network, and determining how XPath and

XQuery expressions can be merged to reduce the need of routing subscriptions among brokers. This issue would get even more complex if content-based matching between subscriptions and messages should be performed by taking into account their semantical, not just syntactical aspects. A problem still largely ignored by current research in CBR.

Even more complex is the security challenge. Being content-based, CBR inherently requires brokers to access the content of messages, something that potentially breaks message confidentiality. Some initial proposals exists to overcome this limitation [31,74,61,58,67,44,54,53] but a fully scalable and efficient solution for secure communication in CBR networks has still to emerge. Moreover, the web services world has developed its own standards about security (i.e., WS-Security [51]), which should be integrated into the chosen CBR solution.

Last challenge we consider is the ability of the CBR infrastructure to self-adapt to changes in the networking environment. An Internet-wide deployment like that envisioned in some of the scenarios depicted above, requires the ability of adapting to changes that may happen in the networking environment, with new nodes and links appearing, existing nodes and links vanishing or failing, temporary network partitions, and so on. The same happens if we consider scenarios involving mobile nodes and wireless networks. In both cases the CBR network must be able to adapt to such situations. Again, some proposals already exist to solve this issue, like those on self-stabilizing CBR [50,64,75], those that suggest using a peer-to-peer overlay substrate to exploit its self-organizing capabilities [6,39,70,68,59], and our own works in the area [26,23,21,24]. Unfortunately, all these approaches still present some limitations in terms of performance. Moreover, they often reduce the guarantees offered by the CBR infrastructure (e.g., most of the approaches above only provide a best effort service). Therefore, a final solution is still to come.

5.4. Open issues

If the points presented in the previous section are somehow being addressed and likely will be solved in a reasonably short time, several other issues remain completely open and require further studies and experiences before they could find an answer. Here is a list of those we find most relevant.

5.4.1. Quality of service

Current CBR infrastructures provide very different qualities of service. Some provide a best effort service, while others guarantee the delivery of messages. Some deliver messages in an arbitrary order, while others provide FIFO ordering (for each single source), causal ordering, or even total ordering. Usually, the more guarantees are offered the less scalability is provided. Unless a one-fit-all solution comes from the CBR research, the best CBR infrastructure to choose depends on the application domain considered.

5.4.2. Dependability vs. scalability

This is a generalization of some of the issues above. Dependability, which includes availability, reliability, safety, and security, is a desirable (but generic) property of a communication infrastructure. Dependability is often achieved at the expense of scalability. Given the importance of both aspects for SOAs, the community should figure out some kind of solution for this issue. A partial solution we propose is to offer to the system designer the possibility of adapting the CBR middleware in order to take into account the property, among the two, that is more important for each specific case. Of course, if this can be considered a basic mechanism to allow flexibility, still a complete methodological and technical solution has to be defined.

5.4.3. Expressiveness vs. scalability

Considerations similar to those above can be done about the expressiveness of the filtering language used by the CBR infrastructure (i.e., to express the interest of components or the content of queries). Indeed, usually the most expressive languages involve complex matching algorithms that may limit scalability and reduce the performance of the CBR infrastructure. Even if XML and XPath are becoming the de-facto standard in the web services community, the CBR infrastructure could use simpler languages at the price of delivering more messages than those strictly required. Finding the right compromise between expressiveness and scalability is an open issue.

5.4.4. Reflectiveness

In [56] the authors advocate the need for the services accessing a WS-Notification infrastructure of a mechanism to query the system to know the topics defined and active at a certain instant. This is reasonable in a global scenario in which services are implemented and managed by different organizations, each unaware of the others. Moreover, it is in line with the philosophy of web services that are assumed to expose in their descriptions all characteristics that are relevant to the interaction with the other parties in the system. This could be addressed by introducing some reflective mechanisms in the CBR infrastructure. The question is to determine the level of reflectiveness that must be provided by the CBR infrastructure and how, i.e., which mechanisms using to avoid breaking the security and dependability of the infrastructure.

5.4.5. Context-awareness

Some domains, like pervasive computing, require service interactions to adapt to the context in which they happen. Context information can be encoded as part of the notifications, advertisements, and queries routed by the CBR infrastructure, but this approach could be inefficient. A CBR infrastructure that natively supports a concept of context, and routes subscriptions and notifications based on it, could offer better performance. We are currently investigating this issue in REDS, but the work is still at a preliminary stage and we cannot draw any conclusions apart from the increased expressiveness that this approach provides.

6. Conclusions

SOAs and web services seem to represent the most promising approaches to deal with the development of complex distributed systems. Their attractiveness stems from the fact that the researchers and practitioners community is working hard at the development and use of proper standards that aim at guaranteeing interoperability and at providing a common view and some solutions to the problems typical of the area. Starting from this point, in this paper we have argued that the usage of CBR techniques could greatly improve discovery of services by overcoming the current limitations due to the centralized nature of registries. Moreover, it could support large scale interaction among services through a decentralized, anonymous, and asynchronous interaction style. There are, however, various aspects that still need to be addressed. We highlighted those we feel most relevant in the previous section.

As for the future, our plan is to work at developing a real and complex case study to apply the prototypes we described in this paper. This should allow us to better reason on how to tackle the dichotomy between scalability and the other desirable properties of the resulting system as well as perform a detailed evaluation of the advantages of our approach.

Acknowledgements

This work has been partially funded by the European Community under the IST-511680 SeCSE and IST-034963 WASP projects, by the italian National Research Council (CNR) under the IS-MANET Project, and by the Italian ministry of research and education under the Artdeco FIRB project.

We warmly thank our colleagues Carlo Ghezzi and Matteo Migliavacca for their useful comments and suggestions, Francesco Leone who is working at our implementation of WS-Notification on top of REDS, and the reviewers of the paper who helped us dramatically improve the quality of the paper.

References

[1] D. Aiken, D10.1v0.1 Focused crawler for web service discovery, Tech. Rep., WSMO, 2005, <http://www.wsmo.org/TR/d10/d10.1/v0.1/>.

[2] E. Al-Masri, Q. Mahmoud, A framework for efficiet discovery of web services, in: Proceedings of IEEE Consumer Communications and Networking Conference (CCNC), 2007.

[3] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C.K. Liu, D. Knig, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu, eds., Web Services Business Process Execution language version 2.0, Tech. Rep., OASIS, 2006, <http://www.oasis-open.org/apps/org/workgroup/wsbpel/>.

[4] D. Bachlechner, K. Siorpaes, D. Fensel, I. Toma, Web service discovery – a reality check, in: Proceedings of the First Workshop: SemWiki2006 – From Wiki to Semantics, co-located with the Third Annual European Semantic Web Conference (ESWC'06), 2006.

[5] D. Bachlechner, K. Siorpaes, D. Fensel, I. Toma, Web service discovery – a reality check, Tech. Rep. DERI, 2006.

[6] R. Baldoni, C. Marchetti, A. Virgillito, R. Vitenberg, Content-based publish-subscribe over structured overlay networks, in: Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05), IEEE Computer Society Press, Columbus, OH, USA, 2005.

[7] L. Baresi, M. Miraz, A Distributed Approach for the Federation of Heterogeneous Registries, in: Proc. of the Fourth International Conference on Service-Oriented Computing, 2006.

[8] L. Baresi, E. Di Nitto, C. Ghezzi, Toward open-world software: Issue and challenges, IEEE Computer 39 (10) (2006) 36–43.

[9] J. Beatty, G. Kakivaya, D. Kemp, T. Kuehnel, B. Lovering, B. Roe, C.S. John, J.S. (Eds.), G. Simonnet, D. Walter, J. Weast, Y. Yarmosh, P. Yendluri, Web Services Dynamic Discovery (WS-Discovery), Tech. Rep., Microsoft Corporation, 2005.

[10] A. Berglund, S. Boag, D. Chamberlin, M. Fernandez, M. Kay, J. Robie, J. Simeon, Xml path language 2.0, Tech. Rep., W3C, January 2007, <http://www.w3.org/TR/xpath20/>.

[11] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simeon, Xquery 1.0: An xml query language, Tech. Rep., W3C, Jan 2007, <http://www.w3.org/TR/xquery/>.

[12] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, eds., Web Services Architecture, 2004, <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.

[13] D. Box, L. Cabrera, C. Critchley, F. Curbera, D. Ferguson, S. Graham, D. Hull, G. Kakivaya, A. Lewis, B. Lovering, P. Niblett, D. Orchard, S. Samdarshi, J. Schlimmer, I. Sedukhin, J. Shewchuk, S. Weerawarana, D. Wortendyke, Web services eventing, Tech. Rep., W3C (Mar 2006), <http://www.w3.org/Submission/WS-Eventing/>.

[14] A. Carzaniga, D. Rosenblum, A. Wolf, Design and evaluation of a wide-area event notification service, ACM Trans. on Computer Systems 19 (3) (2001) 332–383.

[15] R. Chand, P. Felber, Xnet: a reliable content-based publish/subscribe system, in: Proc. of the 23rd IEEE Int. Symp. on Reliable Distributed Systems (SRDS'04), IEEE Computer Society Press, Florianpolis, Brazil, 2004.

[16] D. Chappell, Enterprise Service Bus, O'Reilly, 2004.

[17] D. Chappell, L. Liu, Web services brokered notification 1.3, Tech. Rep., OASIS, 2006, <http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf>.

[18] R. Chinnici, H. Haas, A.A. Lewis, J.-J. Moreau, D. Orchard, S. Weerawarana, Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, Tech. Rep., W3C, 2007, <http://www.w3.org/TR/2007/PR-wsdl20-adjuncts-20070523/>.

[19] R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, Tech. Rep., W3C, 2007, <http://www.w3.org/TR/2007/PR-wsdl20-20070523/>.

[20] L. Clement, A. Hately, C. von Riegen, T. Rogers, eds., UDDI Version 3.0.2, Tech. Rep., OASIS, 2004, <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>.

[21] P. Costa, M. Migliavacca, G.P. Picco, G. Cugola, Epidemic algorithms for reliable content-based publish-subscribe: an evaluation, in: Proc. of the 24th Int. Conf. on Distributed Computing Systems (ICDCS'04), IEEE Computer Society Press, Tokyo, Japan, 2004.

[22] G. Cugola, E. Di Nitto, A. Fuggetta, The jedi event-based infrastructure and its application to the development of the opss wfms, IEEE Trans. on Software Engineering 27 (9) (2001) 827–850.

[23] G. Cugola, D. Frey, A.L. Murphy, G.P. Picco, Minimizing the reconfiguration overhead in content-based publish-subscribe, in:

Proc. of the 19th ACM Symp. on Applied Computing (SAC'04), ACM Press, Nicosia, Cyprus, 2004.

[24] G. Cugola, A. Murphy, G. Picco, Content-based publish-subscribe in a mobile environment, in: P. Bellavista, A. Corradi (Eds.), The Handbook of Mobile Middleware, Chap. 11, Auerbach Publications Taylor & Francis Group, Boca Raton, US, 2006, pp. 257–285.

[25] G. Cugola, G. Picco, REDS: a reconfigurable dispatching system, in: Proc. of the Sixth Int. Workshop on Software Engineering and Middleware (SEM'06), ACM Press, Portland, Oregon, USA, 2006.

[26] G. Cugola, G.P. Picco, A.L. Murphy, Towards dynamic reconfiguration of distributed publish-subscribe systems, in: A. Coen-Porisini, A. van Der Hoek (eds.), Proc. of the 3rd Int. Workshop on Soft. Eng. and Middleware (SEM'02), co-located with the 24th Int. Conf. on Soft. Eng. (ICSE'03), vol. 2596 of Lecture Notes on Computer Science (LNCS), Springer, Orlando (FL, USA), 2002.

[27] Y. Diao, S. Rizvi, M. Franklin, Towards an internet-scale xml dissemination service, in: Proc. of the 30th Int. Conf. on Very Large Data Bases (VLDB'04), Morgan Kaufmann, Toronto, Canada, 2004.

[28] N. Dourdas, X. Zhu, N. Maiden, S. Jones, K. Zachos, Discovering Remote Software Services that Satisfy Requirements: Patterns for Query Reformulation, in: Proc of the 18th Conference on Advanced Information Systems Engineering (CAiSE'06), 2006.

[29] P. Eugster, P. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys 2 (35).

[30] J. Farrell, H. Lausen, Semantic Annotations for WSDL and XML Schema, Tech. Rep., W3C (2007), <http://www.w3.org/TR/sawsdl/>.

[31] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, G. Muhl, Security aspects in publish/subscribe systems, in: A. Carzaniga, P. Fenkam (Eds.), Proc. of the Third Int. Workshop on Distributed Event-Based Systems (DEBS'04), IEEE Computer Society Press, Edinburgh, Scotland, UK, 2004.

[32] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in: Proc. of the IFIP International Conference on Network and Parallel Computing, vol. 3779 of Lecture Notes in Computer Science (LNCS), Springer, Beijing, China, 2005.

[33] G. Fox, S. Pallickara, Deploying the naradabrokering substrate in aiding efficient web and grid service interactions, Special Issue of the Proceedings of the IEEE on Grid Computing 93 (3) (2005) 564–577.

[34] G. Cugola, M. Migliavacca, On adding replies to publish-subscribe, in: Proc. of the Inaugural Int. Conf. on Distributed Event-Based Systems (DEBS'07), ACM Press, Las Vegas, Nevada, USA, 2007.

[35] S. Graham, D. Hull, B. Murray, Web services base notification 1.3, Tech. Rep., OASIS, 2006, <http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf>.

[36] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, I. Sedukhin, Web Services Resource 1.2 (WS-Resource), Tech. Rep., OASIS, 2006, <http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf>.

[37] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl, Publish-subscribe notification for web services, Tech. Rep., OASIS, 2004.

[38] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H.F. Nielsen, A. Karmarkar, Y. Lafon, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Tech. Rep., W3C, 2007, <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.

[39] A. Gupta, O. Sahin, D. Agrawal, A. El Abbadi, Meghdoot: Content-based publish/subscribe over p2p networks, in: Proc. of the Fifth ACM/IFIP/USENIX Int. Conf. on Middleware, vol. 3231 of Lecture Notes in Computer Science (LNCS), Springer, Toronto, Canada, 2004.

[40] D. Heimbigner, Adapting publish/subscribe middleware to achieve gnutella-like functionality, in: Proc. of the 2001 ACM Symp. on Applied Computing (SAC'01), ACM Press, Las Vegas, Nevada, USA, 2001.

[41] Y. Huang, A. Slominski, C. Herath, D. Gannon, Ws-messenger: A web services-based messaging system for service-oriented grid computing, in: Proc. of the Sixth IEEE Int. Symp. on Cluster Computing and the Grid (CCGRID'06), IEEE Computer Society Press, Singapore, 2006.

[42] M. Humphrey, G. Wasson, Architectural foundations of wsrf.net, International Journal of Web Services Research 2 (2) (2005) 83–97.

[43] A. Keller, H. Ludwig, The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management 11 (1) (2003).

[44] H. Khurana, Scalable security and accounting services for content-based publish/subscribe systems, in: Proc. of the 20th ACM Symp. on Applied Computing (SAC'05), ACM Press, Santa Fe, New Mexico, USA, 2005.

[45] A. Kozlenkov, G. Spanoudakis, A. Zisman, V. Fasoulas, F. Sanchez Cid, Architecture-driven Service Discovery for Service Centric Systems, International Journal of Web Services Research 4 (2) (2007) 81–112.

[46] D. Lynch, J. Keeney, D. Lewis, D. O'Sullivan, A proactive approach to semantically oriented service discovery, in: Proceedings of the Second Workshop on Innovations in Web Infrastructure (IWI 2006), 2006.

[47] S.P. Mahambre, M.S.D. Kumar, U. Bellur, A taxonomy of qos-aware, adaptive event-dissemination middleware, IEEE Internet Computing 11 (4) (2007) 35–44.

[48] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, Owl-s: Semantic markup for web services, Tech. rep. W3C (2004).

[49] G. Mühl, L. Fiege, P. Pietzuch, Distributed Event-Based Systems, Springer, Berlin, 2006.

[50] G. Mühl, M. Jaeger, K. Herrmann, T. Weis, L. Fiege, A. Ullbrich, Self-stabilizing publish/subscribe systems: Algorithms and evaluation, in: Proc. of the European Conf. on Parallel Computing (EuroPar'05), vol. 3648 of Lecture Notes in Computer Science (LNCS), Springer, Lisboa, Portugal, 2005.

[51] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker, Web services security: Soap message security 1.1 (ws-security 2004), Tech. Rep., OASIS, 2006, <http://docs.oasis-open.org/wss/v1.1/>.

[52] J. Nanyan, C. Schmidt, M. Parashar, A decentralized content-based aggregation service for pervasive environments, in: Proc. of the IEEE Int. Conf. on Pervasive Services 2006 (ICPS'06), IEEE Computer Society Press, Lyon, France, 2006.

[53] L. Opyrchal, A. Prakash, Secure distribution of events in content-based publish/subscribe systems, in: Proc. of the 10th USENIX Security Symp., USENIX, Washington, USA, 2001.

[54] L. Opyrchal, A. Prakash, A. Agrawal, Designing a publish-subscribe substrate for privacy/security in pervasive environments, in: Proc. of the Int. Conf. on Pervasive Services (ICPS'06), IEEE Computer Society Press, Lyon, France, 2006.

[55] S. Pallickara, G. Fox, Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids., in: Proc. of the Fourth ACM/IFIP/USENIX Int. Middleware Conf., Rio de Janeiro, Brazil, 2003.

[56] S. Pallickara, G. Fox, H. Gadgil, On the creation and discovery of topics in distributed publish/subscribe, in: Proc. of the Sixth IEEE/ACM Int. Workshop on Grid Computing (GRID'05), IEEE Computer Society Press, Seattle, Washington, USA, 2005.

[57] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Llirbat, D. Shasha, Webfilter: A high-throughput xml-based publish and subscribe system, in: Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB'01), IEEE Computer Society Press, Roma, Italy, 2001.

[58] L. Pesonen, D. Eyers, J. Bacon, Access control in decentralised publish/subscribe systems, Journal of Networks 2 (2) (2007) 57–67.

[59] P. Pietzuch, J. Bacon, Peer-to-peer overlay broker networks in an event-based middleware, in: Proc. of the Second Int. Workshop on Distributed Event-Based Systems, San Diego, CA, 2003.

[60] A. Quiroz, M. Parashar, Design and implementation of a distributed content-based notification broker for ws-notification, in: Proc. of the

Seventh IEEE/ACM Int. Conf. on Grid Computing (GRID'06), Barcelona, Spain, 2006.

[61] C. Raiciu, D. Rosenblum, Enabling confidentiality in content-based publish/subscribe infrastructures, in: Proc. of the 2nd Int. Conf. on Security and Privacy in Communication Networks, Batlimore, MD, USA, 2006.

[62] P. Rompothong, T. Senivongse, A query federation of UDDI registries, in: Proc. of the 1st international symposium on Information and communication technologies, Trinity College Dublin, 2003.

[63] B. Sapkota, D. Roman, S.R. Kruk, D. Fensel, Distributed Web Service Discovery Architecture, in: Proc. of the International Conference on Internet and Web Applications and Services (ICIW'06), 2006.

[64] Z. Shen, S. Tirthapura, Self-stabilizing routing in publish-subscribe systems, in: A. Carzaniga, P. Fenkam (Eds.), Proc. of the 3rd Int. Workshop on Distributed Event-Based Systems (DEBS'04), IEEE Computer Society Press, Edinburgh, Scotland, UK, 2004.

[65] A. Snoeren, K. Conley, D. Gifford, Mesh-based content routing using xml, in: Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP'01), ACM Press, Chateau Lake Louise, Banff, Canada, 2001.

[66] G. Spanoudakis, K. Mahbub, Z. A, A Platform for Context Aware Runtime Web Service Discovery, in: Proc. of IEEE 2007 International Conference on Web Services, 2007.

[67] M. Srivatsa, L. Liu, Securing publish-subscribe overlay services with eventguard, in: Proc. of the 12th ACM Conf. on Computer and Communications Security, ACM Press, Alexandria, VA, USA, 2005.

[68] W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, A. Buchmann, A peer-to-peer approach to content-based publish/subscribe, in: Proc. of the Second Int. Workshop on Distributed Event-Based Systems, San Diego, CA, 2003.

[69] F. Tian, B. Reinwald, H. Pirahesh, T. Mayr, J. Myllymaki, Implementing a scalable xml publish/subscribe system using relational database systems, in: Proc. of the 23rd ACM SIGMOD Int. Conf. on the Management of Data, ACM Press, Paris, France, 2004.

[70] P. Triantafillou, I. Aekaterinidis, Content-based publish-subscribe over structured p2p networks, in: A. Carzaniga, P. Fenkam (Eds.), Proc. of the 3rd Int. Workshop on Distributed Event-Based Systems (DEBS'04), IEEE Computer Society Press, Edinburgh, Scotland, UK, 2004.

[71] W. Vambenepe, S. Graham, P. Niblett, Web services topics 1.3, Tech. Rep., OASIS, 2006, <http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf>.

[72] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, J. Miller, Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services, Inf. Tech. and Management 6 (1) (2005) 17–39.

[73] J. Walkerdine, J. Hutchinson, P. Sawyer, G. Dobson, V. Onditi, A Faceted Approach to Service Specification, in: Proc of the Second International Conference on Internet and Web Applications and Services (ICIW07), 2007.

[74] C. Wang, A. Carzaniga, D. Evans, A. Wolf, Security issues and requirements for Internet-scale publish-subscribe systems, in: Proc. of the 35th Annual Hawaii Int. Conf. on System Sciences (HICSS-35), Big Island, Hawaii, 2002.

[75] Z. Xu, P. Srimani, Self-stabilizing publish/subscribe protocol for p2p networks, in: Proc. of the Seventh Int. Workshop on Distributed Computing (IWDC'05), vol. 3741 of Lecture Notes in Computer Science (LNCS), Springer, Kharagpore, India, 2005.

[76] Ibm websphere message broker, <http://www-306.ibm.com/software/integration/wbimessagebroker/>.

[77] Jini, <http://www.jini.org/>.

[78] JMS, <http://java.sun.com/products/jms>.

[79] JXTA, <http://www.jxta.org/>.

[80] Mule ESB, <http://mule.codehaus.org/display/MULE/Home>.

[81] Apache muse, <http://ws.apache.org/muse/>.

[82] OSGI alliance, <http://www.osgi.org/>.

[83] SeCSE project, <http://secse.eng.it>.

[84] WebSphere, <http://www.ibm.com/software/websphere>.

[85] Web Service Modeling Ontology (WSMO), <http://www.wsmo.org/>.