The BPool System: a Scalable Abstract WS-BPEL instantiation mechanism using Semantic Constraints

Aemro B. Amare and Ronny Siebes

Faculty of Sciences, VU University Amsterdam, The Netherlands

Abstract. Most workflow languages such as WS-BPEL assume that parties playing the roles within the workflows, ie. the Web Services are known at design time. However in an open environment like the Web, it is often desirable to have a more dynamic mechanism where parties can subscribe to a predefined workflow at any time and where a subscription and matchmaking system manages the subscriptions and notifies the parties when a workflow is fully instantiated. In this paper we describe a system where parties can upload 'abstract' WS-BPEL workflows or subscribe to the roles contained within them. The subscriptions can be semantically annotated with constraints, which allows more fine-grained automated matchmaking. In order to make our approach scalable, we implemented our prototype using peer-to-peer technology.

1 Introduction

Business integration and re-use of software are the leading motivation for shifting to the Service Oriented Architecture (SOA) paradigm and to Web Services. Although SOA currently is used mainly intra-organizational they are the first big step towards an inter-organizational Web Service architecture. This is because the software already is written down as a service making it easier to connect them with services made by other organizations. According to a recent Forrester Research survey in 2008[1], IT organizations with small, medium, and large budgets are adopting service-oriented architecture (SOA) in large numbers; this is true whether looking at overall IT budget, overall software budget, or the percentage of software budget spent on new initiatives. Whatever way one cuts the data, at least 44% of enterprises are currently using SOA, and at least 63% will use it by the end of 2008.

The aforementioned means that more and more software from small companies could become accessible as Web Services on the Internet. Unfortunately, not many services are currently available to be used. We think that the main reason is that companies like to be in control with whom they interact. To solve this

[1] Randy Heffner, Gene Leganza, Tim Sheedy, Kahini Ranade, *SOA Adoption Data*, April 9, 2008, Forrester Research

problem, workflow languages like the Web Services Business Process Execution Language (WS-BPEL 2.0) are introduced to describe the way the services interact and with whom they interact. The problem with the current approaches is that most languages assume that the parties know eachother at the time the workflows are designed. Especially in an open environment like the Web, the parties know which roles they can play but do not know always beforehand with whom they can or want to interact. For example, a credit card company may have a Web Service able to play the role as payment agent in a trading workflow.

The above example shows that it is not needed that the company knows all other potential parties beforehand. Actually, it only needs to put an advert on a kind of public repository, like UDDI[2], stating that it can play the role in a certain workflow and that it is looking for other parties to fully instantiate the subscribed workflow(s). A client system can poll this repository and notify the subscribing user if all roles are instantiated.

Subscribers should also be able to add some extra constraints in order to exclude workflow instantiations it cannot participate in. For example, a European shipment company could offer a service that fulfills the role of shipper in a workflow, however it could add the constraint that only shipments within Europe are possible. Initiatives to do add semantic constraints like in OWL-S[11] makes this possible however has its limitations as we describe in the next section.

For a such a system to be Web scalable, we focus on technology that distributes the load of the matchmaking and storage processes over more computers. Peer-to-Peer technology is an interesting candidate as a basis for such a large scale billboard system, but more on this in the next sections.

The contribution of this paper is as follows: First we describe the functionality of the BPool portal, where parties can subscribe themselves to roles described in abstract WS-BPEL workflows. The system monitors who subscribed to which workflows and if all roles are instantiated with potential players, they get a notification containing the instantiated WS-BPEL workflow. Second, in order to allow even more finegrained matchmaking, semantic constrains can be added which is written down in a formal language, to improve automatic filtering and matchmaking of the parties. Third, our system is implemented by allowing many (stateless) clients to a scalable Peer-to-Peer based storage overlay.

In the next section, we describe the scope and limitations of existing Web technologies. In the third section, we describe the design of BPool and then briefly describe its basic features. In the fourth section, we present the architecture of BPool. In section five, we review and examine related work. Finally, we give summarize our work and propose future research directions.

[2] http://www.uddi.org

2 From BPEL to Abstract BPEL with Semantic Constraints

WS-BPEL is one of the most popular workflow languages and an industry standard for modeling business processes. WS-BPEL (in short BPEL) is used for implementing executable business processes and describing non-executable abstract processes. BPEL is also the de-facto standard for the composition and orchestration[3] of Web services. It integrates services by treating them as partners playing roles in a process model and allows most basic patterns like sequences, parallel splits, choices, merge etc[4]. Since BPEL is closely linked to Web Services, WSDL[9] plays a major role for the specification of interfaces when partner services are concretely defined within a BPEL document.

BPEL documents that are directly invocable for specific tasks are said to have *concrete assignments*. All interface descriptions of partner Web Services are known at design time. Hence, integrity and compatibility problems will not occur at run time.

A BPEL document that has a control flow specification without actual service bindings is called an *abstract BPEL* document. An abstract BPEL document is not executable until all the abstracted specifications are concretely defined. When all the binding descriptions of Web Services assigned for roles of a workflow are included in the BPEL document, we say the document is an *instantiated* BPEL document.

The dynamic selection of Web Services to populate abstract workflows depends on the task at hand. The question rises how to ease this matchmaking process. The Semantic Web[10] supports defining meanings of Web contents which helps to automate this process. It enables to create common understanding by matching role constraints written in terms from a shared vocabulary, ie. an ontology. The Web Ontology Language(OWL) describes classes, properties, and relations among these conceptual objects in a way that facilitates machine interpretability of Web content[7]. OWL-S is an OWL-based ontology designed for describing semantic Web Services. It facilitates to automatic discovery, invocation and composition of Web Services under specified constraints. The problem with OWL-S is that it is difficult to integrate with popular workflow languages like BPEL.

[3] An orchestration defines the sequence and conditions in which one Web service invokes other Web Services in order to realize some useful function. i.e., an orchestration is the pattern of interactions that a Web Service agent must follow in order to achieve its goal

[4] For more information about the patterns of BPEL and other workflow languages, we would like to refer to
http://www.workflowpatterns.com/evaluations/standard/index.php

Constructs of WS-BPEL are not declarative for semantic Web Service specifications. With the current standard WS-BPEL we can not instantiate roles of an abstract WS-BPEL document with semantically annotated service specifications. Some researchers have tried to convert WS-BPEL descriptions to OWL-S in order to use OWL-S's matchmaker[11,2]. This approach only partially reach the goal, because it is not possible to map all WS-BPEL's complex language constructs to OWL-S constructs one-to-one. Also some other features of the WS-BPEL language, like runtime error handling, can not be translated.

In addition, there should be the possibility of automated binding of Services to WS-BPEL documents. Current design of abstract WS-BPEL description is limited to step by step implementation of workflow at design time. It does not deal about run time compositions of Web services.

We focus on finding solutions for the limitation of Web technologies described above. On the following sections we briefly describe our approach and found results. At the end of this paper we compare our work with other similar approaches.

3 BPool

In this section we describe BPool. The main functionality of BPool is to facilitate the process of instantiating roles from abstract BPEL workflows by Web Services added by users. BPool can be seen as a portal for uploading, matchmaking and instantiating workflows. When a user subscribes a service to a role in a workflow, (s)he can also limit its usage to a certain domain formulating a number of constraints by enumerating a set of concepts from an ontology, for example written in a Semantic Web language like RDF or OWL. The scalable database behind the BPool portal keeps track of the empty or partly instantiated workflows. And everytime a user subscribes a Web Service to a role, a matchmaking process starts to figure out which other services (subscribed to the same workflow) match to the constraints. The user can then check the list of recommended partners and possibly selects to ones it likes.

3.1 Abstract BPEL Publishing and Role Subscription

As described previously, abstract BPEL documents describes a process without having the roles instantiated by endpoints, being Web Services. We replace the values of the elements from regular BPEL document that contains information about the endpoints with *opaque* tokens. The following example shows how we define these unknown values of attributes in *receive* element.

Listing 1.1. Example Abstract receive activity

```
<receive partnerLink="shippingRequester"
    operation="##opaque"
    variable="##opaque" />
```

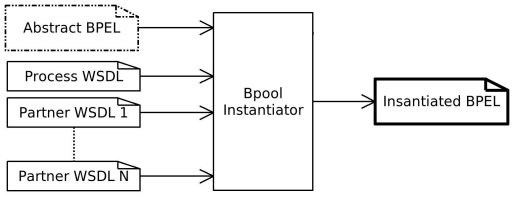

Fig. 1. Instantiating an abstract BPEL

The idea of abstract BPEL description is to enable process developers to complete execution details at these opaque tokens at a later point in time[10]. We implement this idea by our BPool workflow management system to use these abstract BPEL documents as a template where users logged into our portal can subscribe services. The empty or partially defined(abstract) workflows can be uploaded to the BPool portal where after that the users can subscribe their services. When a user subscribes a service, time the abstract part of the role of the BPEL document will be instantiated by the information extracted from the WSDL document belonging to the service, which the user submits as part of the subscription process, as depicted in Figure 1. BPool uses the WSDL files to replace undefined element and attributes of an abstract BPEL documents with concrete values. The information linked by the ***import*** elements provides the matchmaking process and the users the information to map the interface definitions. For example, the undefined messageType elements from the abstract BPEL documents would be imported from via many WSDL services descriptions. Figure 2 shows a simple abstract *hello world* BPEL document which has only one role. The WSDL file next to it, is a WSDL service description uploaded during subscription request. During the instantiation process BPool reads the bindings and messagestype information and replaces with the *opaque* valued attributes and elements of an abstract BPEL with the information in the WSDL file. BPool's interactive interface allows the subscriber to choose the proper definition when the WSDL document has multiple number of possible definitions.

3.2 Semantic Matchmaking

Matchmaking in our context is the process of finding candidate service providers that satisfies requester's requests[14]. BPool makes such matchmaking processes during the interactive subscription moments. After subscription, it recommends subscribed services for new subscription requesters. The match between requests and advertisements is made based on a constraint subscribers subscribed.

During role subscription request BPool makes matchmaking process to list out the possible partners from the subscription database. First, we use a standard SQL Query to list subscribed partners for the same BPEL with the same ontology reference but different role name. This ontology reference points to shared online vocabularies containing the concepts expressed in the enumeration of constraints.

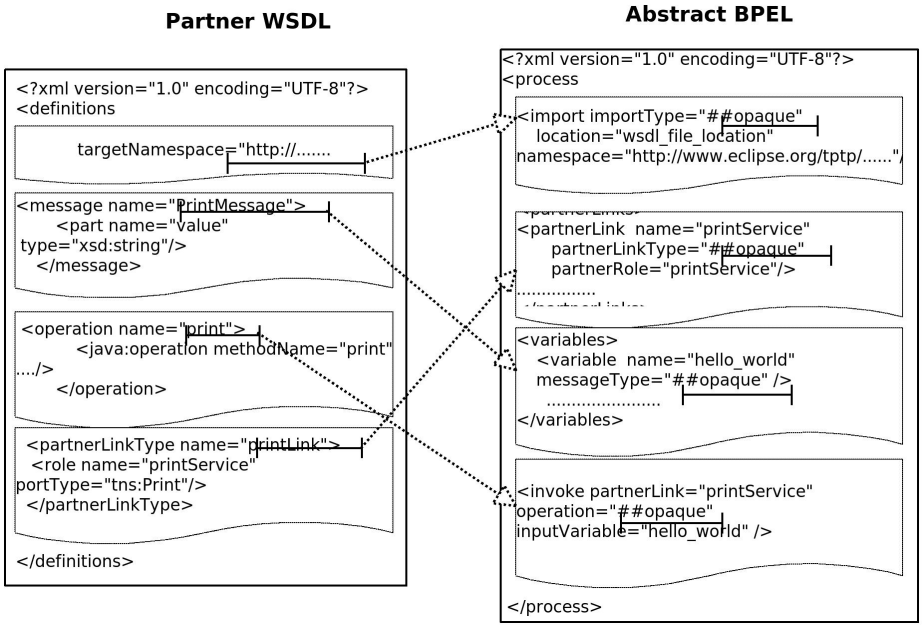

Fig. 2. Instantiating abstract BPEL using Partners WSDL

Listing 1.2. SQL Query to filtered candidate partners

```
SELECT * FROM RoleSubscription WHERE BPELId= ? AND
    RoleName!= ? AND OntologyAddress = ?
```

In this way, subscribers will get a list of all possible partners which have similar or compatible constraints within the workflow of interest. They can also get more details about the subscribers themselves etc. Before going to the next step of subscription, the subscriber should choose a partner or a group of partners they would like to play with. If the subscriber does not find any partner who matches with their constraint an empty set of partners will be created for the subscriber.

4 BPool Architecture

In this section we describe the architecture of the BPool clients and BPool storage. We try to show that we have designed a highly scalable workflow management system. We assume that users will publish abstract BPEL documents on the BPool storage, for example to increase the chance that the usage of their Web Services (playing roles in these workflows) is increased. We assume that the ontologies used to describe the role constraints are somewhere available on the

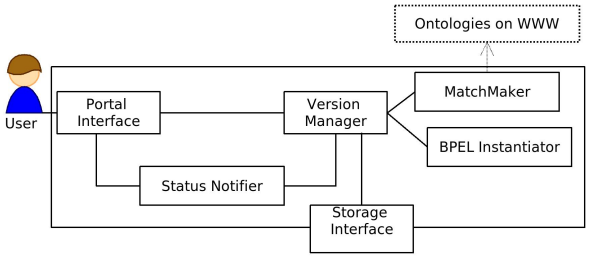

Fig. 3. Overview of the architecture of the BPool system

World Wide Web(WWW) and logically we assume that the Web Services that are subscribed to are actually deployed on a server connected to the Internet.

The system architecture contains six components(Figure 3) which we describe now. Later we show the interaction between the components for the different use cases of our system.

1. ***Portal Interface:*** is the communication interface with the users. It receives the user data, validates it and sends it to the appropriate component.
2. ***Version Manager:*** is the monitor of the system and updates the versions of subscriptions. It interacts with all the other components and receives user request from Portal Interface.
3. ***Storage Interface:*** is the interface for storing and retrieving the objects in BPool, like subscriptions, BPEL files etc. This interface can be implemented by a simple database, but also more scalable as we present in the next section..
4. ***Matchmaker:*** is responsible for matchmaking and validating ontology references. During a subscription, it copies the referred ontologies in memory and tries to identify which subscriptions to a workflow are compatible with the constraints.
5. ***Status Notifier:*** keeps subscribers up-to-date about the status of the workflows. Currently, we implemented a simple email notifier that sends a notification containing the instantiated BPEL file together with the WSDL files to all participants.
6. ***BPEL Instantiator:*** updates abstract and semi-instantiated BPEL documents using data sent by the Version Manager. If a new instantiation process is required then the original abstract BPEL document will be duplicated and the instantiation process will be started on the new copy of the document. For every new copy of a semi-instantiated document, a new Id will be generated by the Version Manager. Generation of identifiers is described in Section 5.

BPool receives, from the user, three types of requests: searching BPEL documents and subscribers, publishing abstract BPEL documents and subscribing a role in an abstract BPEL document.

Users can **search** for published abstract BPEL documents by using keywords submitted via the BPool portal, or any BPool client. BPEL and WSDL documents will be stored in the BPool storage. Hence, the documents can be discovered using the same keywords. Figure 4 shows a sequence diagram depicting the interaction of components for search requests.

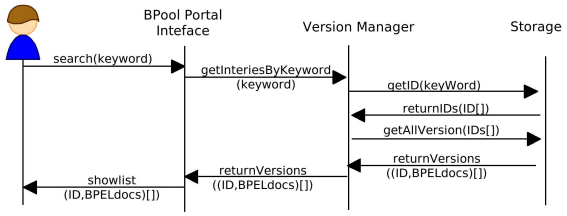

Fig. 4. Sequence diagram showing the interaction of components for a **search** request

Users can also **publish** abstract BPEL documents including some basic parameters, as shown in Figure 5. Informative keywords and descriptions should be included, so that the documents can easily be retrieved by the consumers. Published abstract BPEL documents will be stored in the BPool storage(see Section 5).

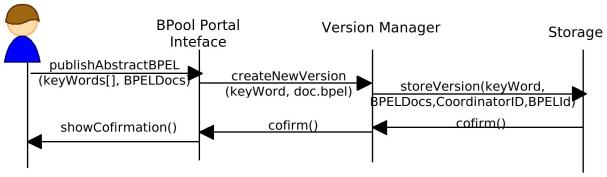

Fig. 5. Sequence diagram showing the interaction of components for **publish** request

Users can **subscribe** a Web Service to any abstract BPEL role available in the BPool system. During a subscription, the user can provide some constraints by listing a set of URI's pointing to concepts from an ontology somewhere hosted on the WWW. In addition, users are required to submit valid WSDL descriptions for each role. Subscribers can also specify the number of times their service could be accessed. If the user does not specify this number, a default number will be assumed which is a parameter of the system. All the other basic parameters are shown in Figure 6. When a user sends the first subscription request, the system will return a list of possible partner groups (s)he can join. This will be done by matchmaking the user's constraints against the constraints from the

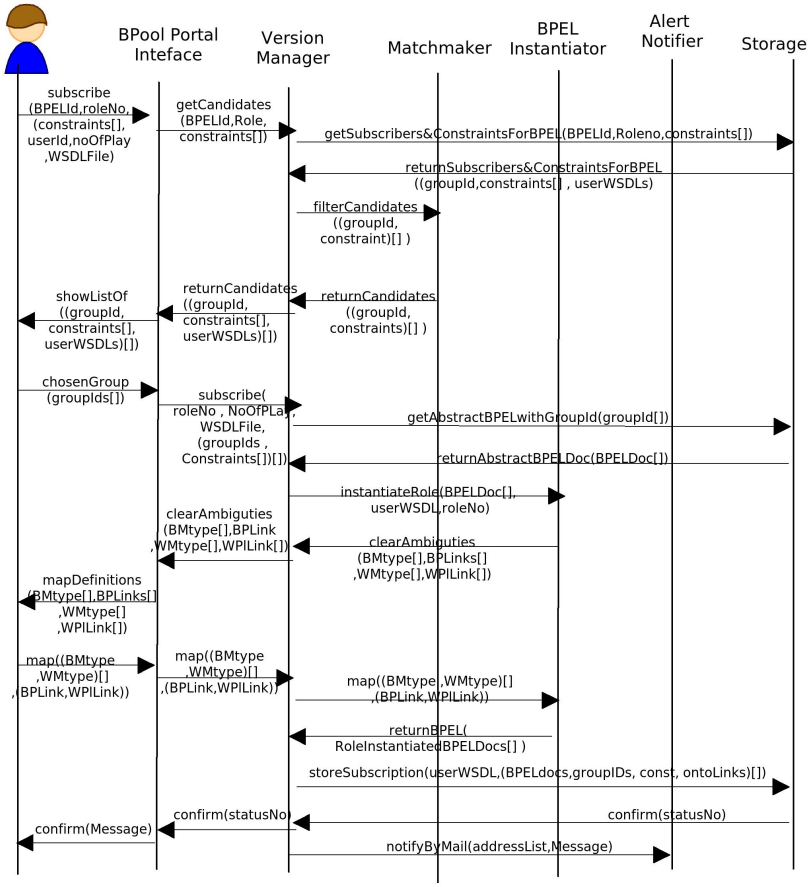

Fig. 6. Sequence diagram showing the interaction of components for **subscribe** request

other subscriptions for other roles in the same workflow. In the second step, the subscriber chooses one or more groups[5] (s)he would like to join.

It is also possible to start a new group and wait for other users to subscribe to services for the rest of the roles. In the third step, the BPEL Instantiator parses the subscriber's WSDL document and the abstract(semi-instantiated) BPEL document, and it tries to instantiate the service for the role. If there are multiple definitions for the same *messageType* and *partnerLink(operation name)* in the submitted WSDL documents, it will inform the user and asks to select one (see Section 2). In other words, the user is expected to map the definitions of *messageType* and *operation* from the WSDL to that of the BPEL document.

[5] a group is a team of subscriber who subscribe different of for the same instance of an abstract BPEL. Every group Id which is the version name of the abstract BPEL(see section 5)

In the last step, the *BPool Instantiator* completes the instantiation process by submitting the new version together with the WSDL file of the subscriber and other parameters to the storage. The *Status Notifier* sends an email to the subscribers involved with the workflow being updated.

Note that each BPool client can read and write data to the BPool storage. To maintain data consistency, each implementation of such a storage needs to have a kind of locking mechanism to prevent clashes of different users working on the same workflow.

In the next section we describe our implementation for the BPool storage interface.

5 Scalable implementation of the BPool storage

In this section we describe a scalable implementation of the BPool storage and the clients connecting to it. The storage of the WSDL documents, the abstract BPEL documents, the subscriptions and their constraints and the user details require a scalable storage space and the matchmaking of the subscriptions and the interaction with the user require CPU cycles. The former is implemented by using an implementation of a Distributed Hash Table (DHT) overlay, originating from research on peer-to-peer systems for storing and retrieving data. The latter is implemented as client software doing the matchmaking etc and connecting to any computer in the DHT when needed. We first start describing this distributed storage and then how we implemented the portal, which is actually a BPool client to the storage overlay.

5.1 Peer-to-peer

Peer-to-peer (p2p) overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. Peers form self-organising overlay networks that are overlayed on the Internet Protocol networks, allowing a mix of features such as a robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance.

Peer-to-peer overlay systems go beyond services offered by client-server systems by having symmetry in roles, whereby a client may also be a server. The peer-to-peer way of pooling and organising resources allows us to build systems that support fault-tolerance, self-organization and massive scalability.

5.2 DHTs

DHTs [15,16,1,13,3] are a particular subclass of p2p systems aimed at storing content in a completely decentralized way [6]. Nodes function autonomously and

collectively form a complete and efficient system without any central coordination. In DHT overlays, each object is associated with a key, chosen from a large space. This space is partitioned in bins, and each peer is responsible for the keys and corresponding objects in the respective bin. Peers need to maintain connections only to a limited number of other peers and the overlay has the ability to self-organize, with respect to peer connections and object distribution, to handle network churn. In principle, all DHT-based systems provide the following functionality: *store(key, object)* storing an object identified by its key, and *search(key)* which returns the object (when it exists) from the peer responsible for the key. Current systems need approximately O(log(N)) messages to search or store and each peer needs to keep O(1) to O(log(M)) pointers to other peers, where N is the number of peers in the network and M is the maximum number of peers.

5.3 Pastry and FreePastry

Pastry[16] is a generic, scalable and efficient substrate for peer-to-peer applications. Pastry nodes form a decentralized, self-organizing and fault-tolerant DHT overlay network within the Internet. Pastry provides efficient request routing, deterministic object location, and load balancing in an application-independent manner. Furthermore, Pastry provides mechanisms that support and facilitate application-specific object replication, caching, and fault recovery.

FreePastry[6] is an open-source implementation of Pastry intended for deployment in the Internet. We use FreePastry as the underlying storage infrastructure for the store and retrieve functionality of BPool. More precisely, it implements the API of the *storage* component described in the previous section. FreePastry is implemented in Java5.0 and is easy to deploy. Via some simple batch files, nodes can be started (each node has its own Java Virtual Machine accessible by its own IP-port). Every time a new node is started, we have to tell it the IP-address and port of another node that already is in the DHT overlay.

For BPool we define the following keys/object mappings in the DHT:

- **Keyword → BPEL-IDs.** As previously mentioned, each abstract BPEL document, is annotated with a set of keywords (which are manually added or automatically extracted by scanning the content from the documents). The **version-manager** component (described in the previous section), creates a unique identifier, the *BPEL-ID* for the object, for example by hashing the content via the SHA-1 algorithm. The keywords will be the *keys* mapping to these identifiers. This means that a lookup on a keyword in the DHT will result in a set of BPEL-IDs. When more than one keyword is used, the intersection of the returned identifiers need to be taken.
- **BPEL-ID → BPEL-versions.** Every time a user subscribes a service to a role, a new version of the initial or partially instantiated Abstract BPEL

[6] http://freepastry.rice.edu/

document is generated by the **BPEL-instantiator** component. The identifier of the original abstract BPEL combined with **groupId** [7] document is used to store and retrieve the different (partially) instantiated versions of that document in the DHT.
- **WSDL-ID → WSDL-doc.** Every time a user subscribes a service to a role, a new WSDL document will be added in the DHT. Like BPEL-IDs documents **version-manager** creates a unique identifier by by hashing the content using the SHA-1 algorithm. The identifier is used to store and retrieve the WSDL document.

The BPool *portal* is a Web application which serves as a P2P client to the BPool storage and retrieval overlay[8]. The portal has a mechanism that maintains consistency of objects during instantiation process. When a client retrieves an object(for example an abstract BPEL) for an updated it removes the object from the DHT and stores it local storage. When the update process is finished it restores the object back in the DHT.

6 Review of Related work

The University of Maryland developed a planning system which can be used with DAML-S for dynamic Web Service composition[5]. SHOP2 planner is applied for automatic composition of Web Services, which are provided with DAML-S descriptions. It plans for tasks in order that they will later be executed. Apart from Web Services it is also possible to call external programs. This feature helps to process intermediate data specially when some input check up is required for some Web Services. The main problem of SHOP2 is that it is highly dependable on DAML-S. It does not have its own struct controls. DAML-S is not a standard workflow specification languages. Most of DAML-S supporting researchers moved to the development of other new semantic based workflow specification languages.

The LSDIS Lab of University of Georgia designed new semi-dynamic Web services composition system by extending the existing Web Service technology standards with Semantic Web technologies to achieve greater dynamism. They developed a framework called METEOR-S. METEOR-S Web Service Composition Framework is designed as a template-based approach for capturing the Semantic requirements of processes. The templates can be used to dynamically discover suitable services and generate executable BPEL4WS documents[2]. The framework is designed by extending the features of UDDI and using the extended OWL(OWL-S) to satisfy the semitic requirements. This makes it more complicated and Web technology dependant. METEOR-S is designed for pre-known

[7] groupId is an Id generated by BPool for every initial instantiation of abstract BPEL. This Id uniquely identifies the group of subscribers for an instance of an abstract BPEL

[8] the prototype implementation of BPool cab be downloaded from http://student.vu.nl/ aae900/BPool/

combinations of Web Services for a workflow which are listed out as a template. It needs a central point to feed up these templates for newly registering Web Services. Generally, it lacks the possibility of collaboration between anonymous Web services.

An approach for Dynamically discovering Web Services using an extended UDDI which can manage semantic matching had been done by IBM researchers, Paolucci and Akkiraju[8]. Mandel and McIlraith came up with user defined constraints for (semi) automatic service selection by combining DAML-S and BPEL4WS to achieve dynamic binding. This work is based on a centralized client-server architecture. It is not scalable to implement on the current broad and complex Business processes. Budak, Ruoyan, Boanerges, and Angela[4] improved the concept of ontology-driven Web Services composition taking quality of service in to account. Ranking and filtering of decomposable services in a registry using user's profile and other criterion are included. For this case specially designed or modified UDDI is required. This approach looks costly in processing time. So they have proposed a decentralized peer-to-peer computing technique. This method suffers the technology dependability problem; for example they may need to modify UDDI for every new version releases.

Shen, Yan and Yang from University of Wollongong, Australia proposed an adaptive composite service orchestration system on a peer-to-peer network named SwinDeW-S[11]. SwinDeW-S is an extended version of SwinDeW(Swinburne Decentralized Workflow) for supporting incomplete processes with on-the-fly task decompositions of Services. Usually matchmaking is process consuming. Most of the implementations of dynamic Web composition systems have a problem of openness[9] and scalability. Scalability and openness principles of SwinDeW-S makes it more advanced. Durring abstract BPEL instantiation SwinDeW-S converts BPEL4WS document to an ontology-based OWL-S profile. As we explained previously BPEL4WS has more features and functions than the not matured OWL-S; hence during conversion of format from BPEL4WS to OWL-S some specifications of BPEL4WS which cannot be represented on BPEL4WS will be missed. Therefore, SwinDeW-S does not always work with abstracts BPELs with their full functionality and behavior.

Wei-Chun, Ching-Seh and Chun Chang proposed an optimizer for Web Service component composition using evolutionary algorithms[12]. The idea is: the algorithm can learn the best combination of Web components from historical interactions for a certain constrains. They did their research in a specific problem domain. The issue of integration within partner services is left for further research.

[9] Here when we say openness we are referring the ability of a system to work with any Web Service deployed on the Internet

7 Summary and Future Work

Even though there is a trend towards Service Oriented Architectures, Web Services are not yet massively used, perhaps because the owners like to keep control over their usage. Current workflow languages like WS-BPEL assume that the participants that implement the endpoints of the roles, are known at design time. This means that even though these languages give more control about participating partners, it is not really suitable for an open setting like the Internet, where participants may not know each other at design time.

We have developed a simple scalable subscription and matchmaking system for Web Services and (abstract) WS-BPEL workflows. In order improve and ease the process of finding matching services to play in a workflow, we adopt Semantic Web technology to express additional constraints.

In order to have a scalable solution we combined stateless clients together with a storage mechanism on top of a peer-to-peer overlay.

Current limitations of our system are firstly that for the automatic matchmaker to work, the constraints for the same workflow need to be described via one ontology. Future work lies in using mapping techniques between ontologies found in the literature to match constraints expressed in concepts from different ontologies. Another limitation is that we do not have an elaborate access-control mechanism. Authentication and service agreement protocols could be added to facilitate the trust of using the BPool system in an open setting where users may not know each other.

8 Acknowledgements

Many thanks to Prof. Maarten van Steen for his useful comments. This work is supported by the EU OpenKnowledge project (FP6-027253).

References

1. Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.
2. Rohit Aggarwal. Masters thesis with a title of constraint driven web service composition in meteor-s. February 2002.
3. Marc Sánchez Artigas, Pedro García López, Jordi Pujol Ahulló, and Antonio F. Gómez-Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *Peer-to-Peer Computing*, pages 49–56, 2005.
4. Ruoyan Zhang Angela Maduko Budak Arpinar, Boanerges Aleman-Meza. Ontology-driven web services composition platform. pages 146–152. IEEE, 2004.
5. E. Sirin J. Hendler D. Wu, B. Parsia and D. Nau. Automating daml-s web services composition using shop2. ICAPS, 2003.
6. F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *IPTPS '03*, February 2003.

7. Connolly D. van Harmelen F. Hendler J. Horrocks I. McGuinness D. L. Patel-Schneider P. F. Dean, M. and Stein. Web ontology language (owl) reference version 1.0. w3c working draft. November 2002.
8. R. Akkiraju Doshi, P. Goodwin and R. Verma. Dynamic workflow composition using markov decision processes. pages 1–17. ACM Press, March 2005.
9. Greg Meredith Erik Christensen, Francisco Curbera and Sanjiva Weerawarana. W3c technical documents for wsdl. March 2001.
10. Charlton Barreto et al. Oasis web services business process execution language (wsbpel) v2.0. May 2007.
11. C. Zhu J. Shen, Y. Yang and C. Wan. From bpel4ws to owl-s: Integrating e-business process descriptions. pages 181–188. IEEE Computer Society Washington, DC, USA, July 2005.
12. C. Zhu J. Shen, Y. Yang and C. Wan. Optimizing dynamic web service component composition by using evolutionary algorithms. pages 708– 711, September 2005.
13. G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, pages 329–350, Heidelberg, Germany, November 2001.
14. Sangyong Han Okkyung Choi and Ajith Abraham. Semantic matchmaking services model for the intelligent web services. pages 146–148, 2006.
15. Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.
16. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.