Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 480 – 487

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

An Application Suite for Service Enabled Workflow

Mostafijur Rahman[a], Wendy MacCaull[a]

[a]*St. Francis Xavier University, Antigonish, Nova Scotia, Canada*

Abstract

Recently our research group introduced the notion of Service Enabled Workflow (SEW) with the integration of Semantic Web Service and Workflow. SEW considers workflow as a collection of tasks with specific control flow where tasks are carried out as services. In this paper, we present a Service Oriented Architecture (SOA)-based integrated application suite to design and develop ontology-based SEWs. Quality-aware semantic web services for tasks in a workflow, more closely aligned to the needs of the consumer through the use of ontology matching, are discovered, selected, composed and executed automatically by a smart phone-based software agent. We evaluate the effectiveness of the application suite using a case study where we introduced service enabled functionality for tasks in a simplified workflow.

Keywords: Semantic Web Service Discovery; Service Composition; Workflow; Service Enabled Workflow; Quality of Service; OWL-S

1. Introduction

Service Enabled Workflow (SEW)[2] is a relatively new concept in the area of Semantic Web (SW)-based research. The literature shows that the underlying concepts are about 10 years old arising from Business Process Model and Notation, Business Process Execution Language, Business Process Execution Language for Web Services and Web Service composition. While SEW has a lot of potential, it still requires a great deal of maturity and support of tools to become an industry standard. In this paper, we present a SOA-based application suite called MOSEW that provides functionalities to design and develop ontology-based SEWs. With this application suite one can graphically define workflow task specifications using ontology instances and execute the workflows automatically through the consumer agent, developed on a mobile platform where QoS-aware semantic web services are discovered, selected, composed and executed automatically for the tasks in a workflow to complete the overall execution.

A workflow is a collection of interconnected tasks with a specific control flow. In SEW, tasks are carried out as services. A service is a self-contained unit of functionality that is placed in a location or is performed to provide a repeatable and consistent set of outcomes to systems. Workflow tasks can be classified into service-configurable tasks

[*] Corresponding author. Tel.: +1-902-872-1113 ; fax: +1-902-867-2329.
E-mail address: x2013ici@stfx.ca

doi:10.1016/j.procs.2016.04.212

which have specifications representing the action needed to be carried out and service-non-configurable tasks which have only task execution rules. Typically, a task specification (TS) is described by several elements including a name, a list of input and output parameters and a task execution rule. To extend the specifications and include the semantic aspects into it, we followed Grossmann et al.[6] work to describe TS using domain specific ontology instances.

Web Services (WSs) are platform independent computational elements that can be described, published, discovered and programmed using XML for the purpose of developing massively distributed applications[15]. SOA is an approach to build distributed systems that deliver application functionality as services which are language and platform independent. It consists of three main building blocks, namely, a Service Repository, a Service Consumer and a Service Provider. Semantic Web Service (SWS) is a standalone piece of functionality that is self-descriptive, machine-readable and can be automatically discovered and executed from the web[16]. WSs does not support automated service discovery and execution from web while SWS does. Consumers can utilize this advantage to accomplish their desired tasks automatically by using SWS based discovery, selection and execution approach. With an increasing number of WSs providing similar functionalities, QoS properties become an important criterion for the selection of the most suited WS. In general, QoS properties are divided into two sub-categories: Measurable (response time, execution price, throughput and reliability etc.) and Non-measurable (reputation and security etc). In our research, we worked with measurable QoS properties.

Over the last few years, smart phones have become very common and convenient devices with high usability. Advances in Information and Communication Technology have influenced service providers to improve their infrastructures so that consumers can consume their services through the smart phone driven application or agent. The agent can act as a mediator to discover services automatically in a dynamic workflow execution environment. Nowadays, an increasing number of companies and organizations implement their core business functionalities as services and publish them over the Internet to provide access to others. Thus, the ability to efficiently and effectively select and integrate cross-organizational and heterogeneous services on the Web at runtime is an important step towards the development of WS-based applications. In particular, if no single WS can satisfy the functionality required by the user, the services need to be combined or composed to achieve the desired goals of the consumer[12]. For example, setting up an appointment requires discovery and selection of both patient and physician. These are provided by two different services: Discover Patient and Discover Physician. In this example, the Discover Patient service would have to be invoked first, because this will select the patient. Only after the invocation of the Discover Patient service is completed and the patient is selected, will the Discover Physician service be invoked. So, the services need to be combined or composed to achieve the desired goals of the consumer.

The rest of the paper is organized as follows: Section 2 describes a Motivating Example, and Section 3 presents our approach for QoS-aware SWS discovery. Section 4 provides a brief description on the MOSEW architecture. In Section 5, we present experimental results. Section 6 provides a brief description on related works and in Section 7, the conclusion and future works are given. While we use the example described in Section 2 to evaluate the effectiveness of the application suite, our work is designed to be domain independent and is not restricted to only this example. We kept the ability to integrate any domain specific ontology into our application suite to manage any complex scenarios and to execute related workflows automatically to achieve their desired goals.

2. Motivating Example

We[19] have been working with the clinical guidelines related process models for last few years. The example we describe here is based on the simplified Hospice Palliative Care[20] process model. Suppose Emma performs most of the tasks involved in this process model and the use-cases are as follows: 1. Patient referrals are received by the Palliative-Care program. 2. Appointments are set with the physicians based on their availability for each of the patients. 3. A Physician consults with the patient. 4. A Physician decides if the patient is appropriate for the program or not. 5. If the patient is not appropriate she is refused with an explanation. 6. Otherwise the patient is registered into the program. 7. A team is formed with formal and informal caregivers for each of the registered patients. 8. The teams continue to provide care to each of the patients until they are released.

In this process model, depending on the eligibility status of each patient, a part of the process model is executed. Consider a scenario, where Emma can use SWSs to perform all these tasks. It would be difficult for her to execute all these tasks by consulting the SWSs manually because:

- For each patient, she will have to search for an appropriate physician and set an appointment between the physician and the selected patient.
- Based on patient's eligibility status, either she will have to refuse the patient with an explanation or form a team consisting of formal and informal caregivers to provide care to the patient.

Instead, suppose the service providers use the MOSEW domain and QoSMetric ontologies[1] to design QoS-aware SWSs using a semantic web service discovery framework, called OWL-S and publish those services in the repository so that consumers can discover and execute their services. Then as a service consumer, Emma uses a mobile agent to initiate a service discovery request for each of the service-configurable tasks in the workflow. SWSs are discovered and selected for each of these tasks automatically, this results in a semi-automatic WS composition for the process model or workflow. The composed WSs are executed dynamically to finish the overall execution of the workflow.

3. QoS driven SWS Discovery

To incorporate QoS information in the dynamic WS discovery process, the major problems are specifications and storage of QoS information[9]. To deal with these two problems, we proposed an ontology-based QoS conceptual model (shown in Fig 1) which was integrated into OWL-S 1.2 framework. We address the problem of consumers' QoS specifications and preferences by allowing them to specify the quality of each QoS properties. We provide flexibility to the service providers by allowing them to define QoS information for the services they offer. We classify each QoS property using a software metric; here we use High, Medium, Low and Fail, each of which depends on a range of values for the specific property. For example, from pricing point of view, we classified the Execution Price (EP) into the four grades. Many users do not understand these grades. So, we provided the mapping from user's point of view to a pricing point of view. The grades are: Low - EP ≤ \$25/per request, Medium - EP > \$25 and ≤ \$50/per request, High - EP > \$50 and ≤ \$100/per request and Fail - EP > \$100 /per request.

We map each quality to an appropriate numerical weight, e.g., Low Response Time gets the highest weight while Low Reliability gets the Lowest value for the weight. These QoS quality metrics are reflected in the QoSMetric ontology we designed. Service providers use these metrics to describe the quality of QoS properties of the services they provide. From a pricing or network point of view, we can classify each QoS properties, such as Execution Price, Response Time, etc., into several grades. Due to network, security, maintenance and other provider specific aspects, values of the grades for these QoS properties may vary from provider to provider. Providing detailed analysis of this is not in the scope of our work and for simplicity, we will move forward with some constraints and assumptions.

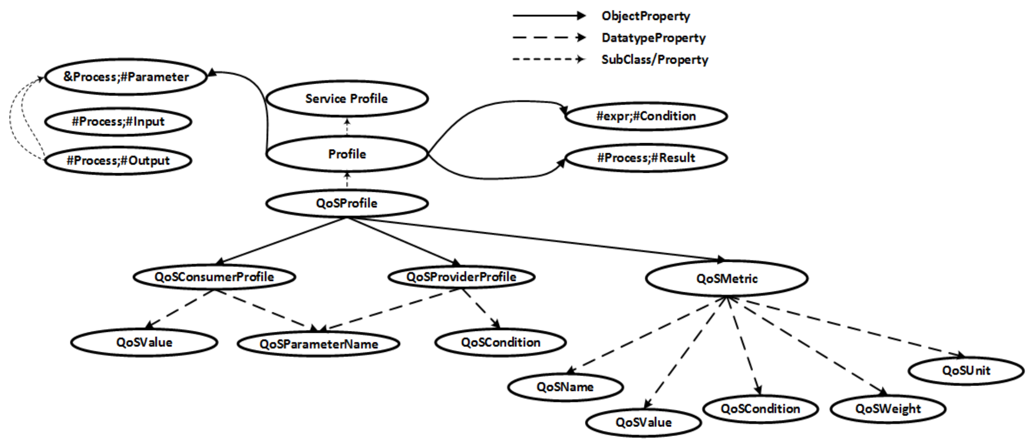

Fig. 1. QoS Conceptual Model

For the MOSEW application suite, we could not use the APIs[13] or[22] to read and execute OWL-S 1.2 service specifications because both[13] and[22] were developed based on OWL-S 1.1 service descriptions (readers of these APIs do not support OWL-S 1.2 specifications) and[13] has problems with WSDL service grounding[21]. Motivated by[13] and[22], we designed and developed the OWL API based OWL-S API that provides a Java API to read OWL-S 1.2 service descriptions and execute the WSDL service operations. The data models are designed to reflect the structure of the OWL-S service model. We created a set of readers to read the descriptions of Service, ServiceProfile, ServiceModel and ServiceGrounding in the OWL-S 1.2 service model. The Pellet reasoner is integrated into the OWL-S 1.2 API to provide the required reasoning support.

The core matching algorithm, which extends algorithm[14], consists of two parts: basic functional (I/O) property-based matching and non-functional property (QoS)-based matching. Based on the steps involved in the core match-making algorithm, we divided the scoring process into two parts: Output/input property-based scoring and QoS property-based scoring. If the total score is same for both the output and input matching of the advertised WSs, QoS matching can be used to break the tie between them.

The degree of matching between two outputs or two inputs depends on the subsumption relation between the concepts associated with the outputs or inputs. We divided the degree of matching into five grades and gave weights to each of these grades: Exact (4), Approximate (3), Plugin (2), Subsumes (1) and Fail (0). Suppose C and D are two concepts. If C and D are equivalent concepts, this is considered as an Exact match. If C is an immediate sub class of D, then the match is considered as an Approximate match. If D subsumes C, this is considered as a Plugin match. If C subsumes D, this is a Subsumes match. If no subsumption related is found, this is declared as a Fail match.

```
CalculateOutputMatchingScore(outputRequests, outputAdvertisements){
    forall outputRequest in outputRequests do
        forall outputAdvertisement in outputAdvertisements do
            match = SewMatchMaker
                    .MatchConcept(outputRequest,outputAdvertisement)
            if(match >0){
                outputConceptRank = SewMatchMaker
                    .GetOutputRankingScore(outputRequest)

                if(outputConceptRank >0){
                    totalOutputScore += outputConceptRank * match
                }
                else{
                    totalOutputScore += match
                }
            }
    return totalOutputScore
}
```

Fig. 2. Algorithm to Calculate Total Output Matching Score

The main control loop of the semantic matchmaking algorithm (SMA) iterates over the advertised services (AS). On each iteration, the matching algorithm passes one of the ASs and the request as parameters to the algorithm. This algorithm sequentially calls the methods to check the output, input and QoS input parameters of an AS. An AS matches a request when all the outputs of the request are matched by the outputs of the advertisement and all the inputs of the advertisement are matched by the inputs of the request. The matching between the input concepts is computed using the same approach for the output concepts, except the order of the request and the advertisement are reversed[14]. We calculated the total output matching score using the procedure outlined in Fig 2.

```
CalculateQoSMatchingScore(reqQoSList, advQoSList, qosMetricList){
    forall reqQoSVal, reqQoSType in reqQoSList do
        forall advQoSCond, advQoSType in advQoSList do
            if(reqQoSType.equals(advQoSType)){
                qosTypeList = qosMetricList.get(reqQoSType or advQoSType)
                reqQoS = qosTypeList.get(reqQoSVal)
                advQoS = qosTypeList.get(advQoSCond)
                if(reqQoS !=null && advQoS !=null){
                    match = SewQoSMatchMaker.MatchQoSConcept(reqQoS,advQoS)
                    if(match >0){
                        totalQoSMatchingScore += match
                    }
                }
            }
    return totalQoSMatchingScore
}
```

Fig. 3. Algorithm to Calculate Total QoS Matching Score

The matching degree of QoS concepts is similar to the degree of input/output concepts except we do not use the Approximate grade. We calculated the total QoS matching score for the QoS properties using the procedure outlined in Fig 3. Suppose there are N services in the Service Repository. If V and E refer, respectively to the number of concepts in the MOSEW domain ontology and the number of edges to connect the concepts, then the total time required to perform the operations on N services, T_N, *is O(N * (V+E))*. To rank the SWSs, we used the input/output and QoS matching scores of each service. We developed the ranking algorithm and placed this on top of the Service Discovery Engine that executes the SMA that orders the matched services in ascending order.

4. MOSEW Architecture

Service Oriented Architecture (SOA) is a very popular architectural paradigm for designing and developing distributed systems. To develop the MOSEW integrated application suite[1] based on a SOA paradigm, we designed and developed five components. Fig 4 shows the architecture of the MOSEW application suite. Details about the application suite will be found in chapter 5 of the MSc thesis relating to this research work[18].

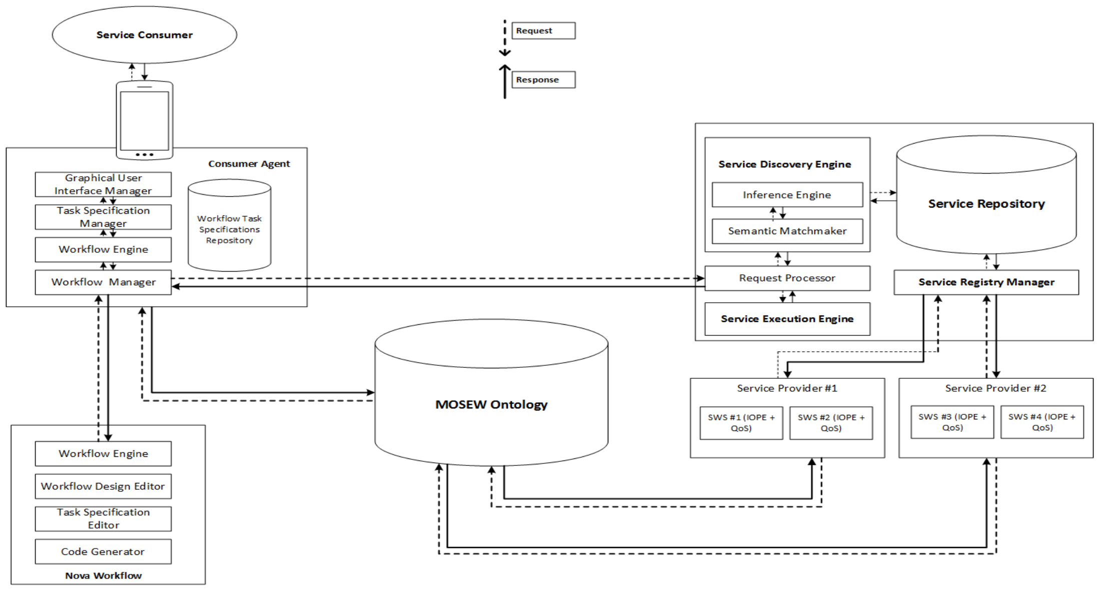

Fig. 4. MOSEW Architecture

We find the Service Discovery Engine (SDE) on the upper right hand side of Fig 4. The SDE is developed on the Jersey, a Restful WS framework that can be deployed in various application servers. The Service Execution Engine (SEE) is also developed on the Jersey framework. On the upper left hand side, we find the Consumer Agent (CA). The CA is developed on the Android mobile platform to make the development of this system easier by seamlessly integrating the loosely coupled components into the overall development process. The Service Repository (SR) is developed on the .Net platform using the Windows Communication Foundation framework and the Service Registry Manager (SRM) is developed in the Asp.Net Model View Controller framework.

The SDE component is built on top of several layers such as Jersey Restful API Layer, Business Layer (BL) and Data Access Layer (DAL). The DAL contains two underlying components: Semantic Matchmaker and Inference Engine. The DAL communicates with the Service Repository to get the list of advertised services. The Semantic Matchmaker applies the semantic matchmaking algorithm and the Inference Engine performs the reasoning. If any service satisfies the request, it is added to the matched service list. After the service list is discovered, the SDE sorts the list, transforms the WS related information into the JSON format and returns the response result to the CA. The SEE executes the services requested by the CA using the ServiceGrounding specifications of the OWL-S service to execute the service that maps the processes to WSDL operations.

The CA is a visual tool to configure specifications of the workflow tasks and execute the workflows dynamically. This component consists of four sub components: Workflow Manager (WM), Worklfow Execution Engine, Task Specification Manager and Graphical User Interface (GUI) Manager. The WM communicates with the Nova Workflow Engine[4] to get the available workflow data models. The GUI Manager provides user interfaces to define specifications (Inputs, Outputs, QoS Inputs and Task Rule) of each workflow task using ontology instances. The Workflow Execution Engine uses the control flow of the workflow data model and task specifications of the service-configurable tasks to execute the workflows dynamically. The SRM component provides user friendly interfaces to service providers for publishing their designed OWL-S services. The SRM also keeps logs of service providers and their published services in a relational database system and stores OWL-S service files in a tree based folder structure and displays published services to them. The SR is the repository where all the SWSs are stored. Service providers describe services functionalities using the extended OWL-S framework and publish their services to the repository through the SRM.

5. Experimental Results

We used the example described in Section 2 to evaluate the performance of the MOSEW application suite. We used the NOVA Workflow Design Editor to design the workflow. The editor produces structured workflows and stores the workflow models in XML format. We utilized the XML formatted workflow data model and the MOSEW components to introduce the service enabled functionality in the Hospice Palliative Care workflow. To demonstrate how SEW works, we addressed each of the service-configurable tasks from SOA point of view. For each task, as a service provider, we advertised a list of SWSs to the Service Repository. Then, as a service consumer, we initiated each service discovery request to the Service Discovery Engine for each of the tasks in a workflow. Suppose we advertised a list of OWL-S services in the Service Repository including these two services for the Discover_Select_Patient task:

- **Community Care Service:** A service with Low ResponseTime and Low Throughput, that returns the Patient list referred to the Community Care Program.
- **Palliative Care Service:** A service with Low ResponseTime, High Reliability and Medium Execution price, that returns the Patient list referred to the Palliative Care Program.

At the Discover_Select_Patient task, the consumer provides inputs into the mobile device asking for a service with Low ResponseTime and Low ExecutionPrice, that returns the Patient list referred to the Palliative Care Program. For the above service discovery request, the SDE returns a list of matched services with *Palliative Care Service* as the top ranked service. The Workflow Execution Engine selects the top ranked service, places it to the selected service queue and moves to the next task in the workflow. With the discovery of SWSs for each of the tasks in the workflow, service composition is performed at runtime. During the execution of the services, a sequence of messages are passed among the services to finish the overall execution of the workflow.

We deployed the MOSEW server side components and the Nova Workflow Engine on a high performance Amazon EC2 cloud server (64 bit Windows 2008 R2 and 4GB memory). Also, we deployed the CA in four next generation Android devices with different system configurations. There are eight atomic tasks in a workflow. To evaluate the performance of this application suite, for each of the atomic tasks, we designed four WSs. We used the same specifications for the atomic tasks and published the additional number of services with different inputs, outputs and QoS inputs in different runs. In the first run, we published one service per atomic task. In Samsung Galaxy TAB 4 device, it took 5, 8, 6, 6, 7, 6, 6 and 5 seconds consecutively for the tasks Discover_Select_Patient, Discover_Select_Physician, Setup_Appointment, Consult, Discover_Select_Caregiver, Deliver_Care, Explanation and Prescribe_Drugs. Then we published two, three and four services in the second, third and fourth runs, respectively.

We measured the execution time of the workflow in seconds (unit). In addition to the total number of tasks in a workflow and total number of services in the service repository, total execution time may vary depending on the network parameters such as bandwidth at both consumer and server sides. For this reason, we measured several readings and averaged the execution times at each run for each device. Fig 5 shows the performance graph of the MOSEW application suite based on the experimental results on four Android devices. Details explanations on performance graph will be found in chapter 6 of[18].

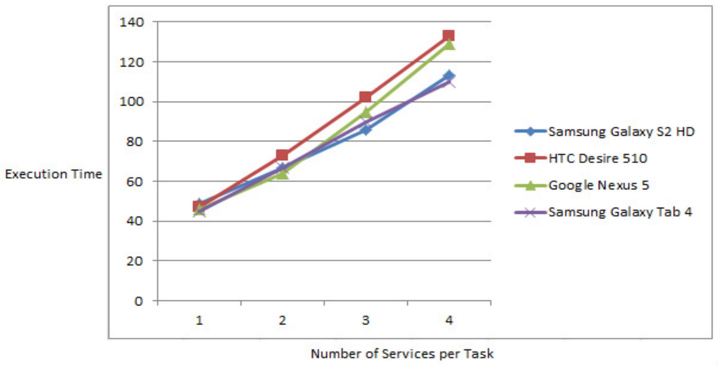

Fig. 5. MOSEW Performance Graph

6. Related Works

For the related works, we focus on the QoS property-based efforts integrated into SWS Discovery framework. There are several QoS ontology proposals which provide a shared vocabulary for the QoS concepts. They also facilitate the intercommunication in homogeneous and heterogeneous environment[10]. We consider the QoS models based on the OWL-S 1.1 framework as we could not find any work based on the OWL-S 1.2 framework. DAML-QoS[11] is an ontology that works with the OWL-S 1.1 framework . The authors defined QoS metrics for the QoS properties and developed external ontologies for these metrics. The main problem is that the QoS property constraints are described by cardinality constraints, which restricts the number of values a property can take. Moreover, this approach does not consider the WS consumer's requirements and preferences. The QoSOnt[10] ontology is a modular ontology, which features several QoS attributes that can be measured by several metrics. This allows unit conversion and requirement specifications but the requirement specification does not allow the description of consumer preferences. A three layer-based matching strategy is discussed in[7] for SWS discovery based on user preferences and QoS, while in[3], the authors presented a four layer-based SWS discovery based on user preferences, reputation and QoS information. In both cases, they provided options to the users to set monotonically increasing or decreasing weights in order to specify their personal choices. Both these strategies have been accomplished on the OWL-S 1.1 framework. In our work, we provided options to choose QoS values (high, medium or low) and used the most recent OWL-S 1.2 framework and integrated our proposed QoS conceptual model into it.

The API[13] provides access to read, write and execute OWL-S (1.1 and 1.0) service descriptions. The API[13] also provides an Execution Engine that can invoke atomic processes which has WSDL grounding. The work[22] converts OWL-S descriptions into Java objects. Neither these works support OWL-S 1.2 service descriptions nor have support for OWL models. These are built on top of Jena so the function returns the corresponding resource in the Jena model.

In[5] and[8], the authors proposed a WS discovery method based on the domain ontologies. The proposed method calculates semantic similarity based on relative distance between the concepts in both service request and service advertisement. They did not consider the matching based on subsumption relationship among the ontology concepts. The work presented in[14] proposed an OWL-S based matchmaking algorithm which is used to match a requested service with a set of advertised ones. This matching algorithm provides functional property-based matching where it compares the input and output concepts of the consumer request to the service description in the repository and defines four levels of matching: Exact, Plug in, Subsumes and Fail. It does not provide QoS property-based matching and can not differentiate between the immediate subclass of an ancestor class and the subclasses which have distance more than one.

7. Conclusion and Future Work

The service discovery approach is sequential and the performance graphs appear to be linear (see Fig 5). The more advertised services we have in the repository, the more time will be required to discover the matched services to process a single service discovery request that we observed in our experimental results. Parallelizing the service discovery approach is an obvious solution to improve the performance of the application suite. Currently we are working on parallelizing the Semantic Matchmaking algorithm to apply it to the service discovery approach. Next we will work with the full version of the Hospice Palliative Care workflow[20] to observe the scalability, validity and practically of the application suite depending on the number of inputs, outputs, outputs preferences and size of ontology. Then we will compare the performance reports that we will find by applying both sequential and parallelized service discovery approach on the application suite.

In our research, we worked with the basic subsumption relationship based matching. We plan to work with richer domain specific ontologies using the full expressiveness of the ontology language OWL. Also research work[2] is in progress in our lab for the development a SWS Discovery framework where we use rules and rule based reasoning. This work will overcome the limitations of existing WSMO and OWL-S frameworks. Our OWL-S framework based work may be replaced by this newly developed SWS Discovery framework in future too.

In this paper, we present MOSEW, a SW-based integrated application suite that is used to design and develop SEWs running on mobile devices. With this one can define workflow task specifications using domain specific ontology instances and introduce the service enabled functionality in real life workflows. We achieved this through SWS discovery, selection, semi-automatic run time WS composition and execution. This type of WS composition is time consuming and as a result, is less flexible. The automatic WS composition method generates the process model automatically or locates the correct services if an abstract process model is presented. In future, we will extend the MOSEW application suite to support automatic WS composition.

References

1. Mostafijur Rahman and Wendy MacCaull, Design and Development of a Tool Suite for Service Enabled Workflow. Saint Francis Xavier University, Antigonish, Nova Scotia, Canada, 2015.
2. Altaf Hussain and Wendy MacCaull, Context Aware Service Discovery and Service Enabled Workflow, pp. 45-48. Canadian Semantic Web Symposium, 2013.
3. Rohallah Benaboud, Ramdane Maamri, and Zadi Sahnoun, Semantic Web Service Discovery Based on Agents and Ontologies, pp. 467-478. International Journal of Innovation, Management and Technology, 2012.
4. Wendy MacCaull and Fazle Rabbi, NOVA Workflow: A Workflow Management Tool Targeting Health Services Delivery, Springer Lecture Notes in Computer Science, pp. 75-92, volume 7151. FHIES, 2011.
5. G. Lu, G. Zhang and S.Li, Semantic Web Service Discovery Based on Domain Ontology, pp. 1-4. World Automation Congress, 2012.
6. Georg Grossmann, Michael Schere and Markus Stumptner, A Conceptual Modeling Approach for Web Service Composition Supporting Service Re-Configuration, pp. 43-52. in: 7th Asia-Pacific Conference on Conceptual Modeling, 2010.
7. Zheng K. and Xiong H., Semantic Web Service Discovery method based on user preference and QoS, pp. 3502-3506. In: 2nd International Conference on Consumer Electronics, Communication and Networks, Yichang, 2010.
8. Ying Zhang, Houkuan Huang, Dong Yang, Hongke Zhang, Han-Chieh Chao and Yueh-Min Huang, Bring QoS to P2P-based Semantic Service Discovery for the Universal Network, pp. 471-477. Personal and Ubiquitous Computing, 2009.
9. Ziqiang Xu, Patrick Martin, Wendy Powley and Farhana Zulkernine, Reputation-Enhanced QoS-based Web Services Discovery, pp. 249-256, In Proc. the International Conference of Web Services, 2007.
10. G. Dobson, R. Lock and I. Sommerville, QoSOnt: an Ontology for QoS in Service-Centric Systems, pp. 80-87. Euromicro Conference on Software Engineering and Advanced Applications, 2005.
11. C. Zhou, L.T. Chia and B.S. Lee, DAML-QoS Ontology for Web Services, pp. 472-479. International Conference on Web Services, 2004.
12. Jinghai Rao and Xiaomeng Su, A Survey of Automated Web ServiceComposition Methods, pp. 43-54. International Conference on Web Services, 2004.
13. Evren Sirin and Bijan Parisa, The OWL-S Java API. Alternate Paper Tracks, 2003.
14. Massimo Paolucci and Takahiro Kawamura and Terry R. Payne and Katia P. Sycara, Semantic Matching of Web Services Capabilities, pp. 333-347. International Semantic Web Conference, 2002.
15. Dragon Gasevic, Dragon Djruic and Vladan Devedzic, The Semantic Web in: Model Driven Architecture and Ontology Development, pp. 79-107. Springer, 2006.
16. Jos De Brujin and Dieter Fensel and Mick Kerrigan and Uwe Keller and Holger Lausen and James. Sciecluna, Semantic Web Service in: Modeling Semantic Web Services pp. 9-20. Springer, 2008.
17. Dragon Gasevic, Dragon Djruic and Vladan Devedzic, Ontologies in: Model Driven Architecture and Ontology Development, pp. 79-107. Springer, 2006.
18. Design and Development of a Tool Suite for Service Enabled Workflow, `http:///services.biocomalert.com/thesis/Sew01.27.2016.3.30.pdf`
19. Center for Logic and Information, `http://logic.stfx.ca/home/`
20. A model to guide hospice palliative care, `http://www.chpca.net/media/319547/norms-of-practice-eng-web.pdf`
21. OWL-S, `https://code.google.com/p/owl-s/`
22. OWL-S API, `http://projects.semwebcentral.org/projects/owl-s-api/`