CONTEXT AWARE SERVICE DISCOVERY AND SERVICE ENABLED WORKFLOW

Altaf Hussain, Wendy MacCaull
Centre for Logic and Information, Dept. of Mathematics, Statistics, and Computer Science
St. Francis Xavier University
Antigonish, Nova Scotia, Canada
ahussain@stfx.ca, wmaccaul@stfx.ca

***Abstract*—We provide a conceptual model for context aware Semantic Web Service (SWS) discovery, which can utilize real-time legacy data from external systems and support user context-based service discovery and selection. This model offers advantages over current SWS technology which cannot be easily applied to different domains or be integrated with legacy systems. Using this conceptualization we propose an intelligent decision support system, which offers Service Enabled Workflow.**

Keywords—Semantic Web Service, Context Aware Service Discovery, Service Enabled Workflow, Service Metadata, Ontology

I. INTRODUCTION

A service is an entity that offers an intended value to its consumer; in today's society, people are dependent on service paradigms. A service consumer may need to pay an exchange value to consume a service but does not have to be concerned with how the service is developed or delivered. The service model design, development, and delivery are the concern of, and are handled by, the service providers: e.g., *the Postal Service*. *Web Service (WS)* is the technology that makes services available as consumable entities accessed and consumed through computers, via the Web: e.g., *the Email Service*. WS technology, backed by *Service Oriented Computing* and *Service Oriented Architecture (SOA)* has gained attention and popularity in the commercial computing sector as an enabling technology for the most enduring service planning, development, delivery and management methodology. As a result, a new spectrum of web applications has emerged supporting Business-to-Business integration, e-commerce, and industry wide collaboration. These applications are empowered by the WS technology, which provides a platform supporting independent communication and machine-to-machine interaction framework. However, WS technologies need extensive human involvement for service discovery, composition, invocation, etc.

In the recent years, a new paradigm has evolved, called the *Semantic Web (SW)*, supporting machine-readability, and automated trusted interaction between computers with minimal human intervention. The markup language of the SW is based on the Web Ontology Language (OWL), which can be used to express logical relations among entities on the web, and leads to a new class of WS called *Semantic Web Service (SWS)*. A Semantic Webservice is a standalone piece of functionality that is self-descriptive, machine-readable, and can be automatically discovered and executed via the web. The SWS, inheriting the properties of the SW and the WS has achieved many desirable properties, namely: a) machine independent communication and machine readability b) easy and widely acceptable collaboration methodologies c) exploitation of SW and reasoning techniques. Effort has been made in the areas of SWS, for example: semantic description of WS, semantic reasoning based WS discovery and SWS delivery thorough ontology based concepts and frameworks e.g., Web Ontology Language–Services (OWL-S) [1], and Web Service Modeling Ontology (WSMO) [8].

As SWS becomes more popular, users expect it to be easier to integrate with different domains and legacy systems. Existing SWS approaches do not provide any easy methodology to integrate domain data (often housed in traditional databases) in the service discovery process to

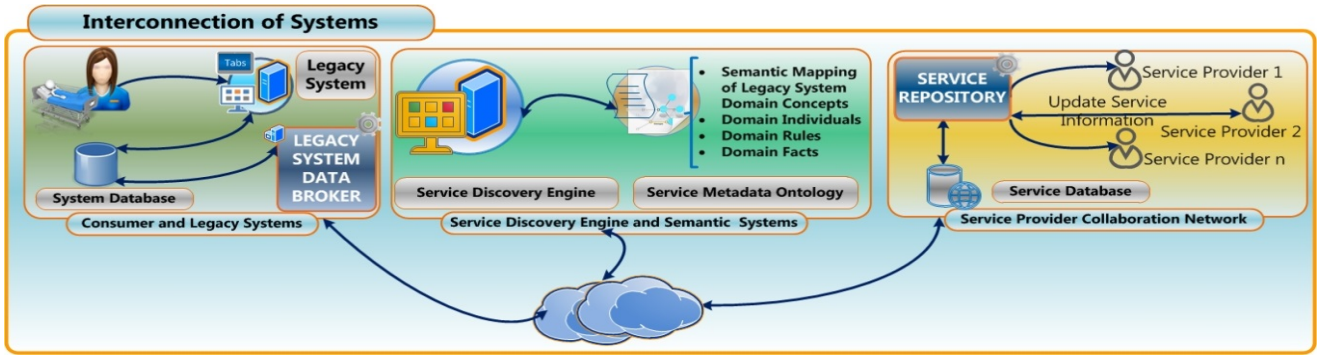

Figure 1: Interconnection of legacy systems and SWS system

support *context aware* service discovery. However, users frequently need to select services based on domain situations. To support automatic interoperation of the SWS discovery process with traditional systems, SWS discovery should be able to utilize real-time data from external systems and domains, providing automatic discovery and selection services based on domain situations and conditions. See figure 1.

stating the real-time patient data properties and values, *rules* stating the action required to be taken by the user based on the facts and services, and the queries. We can model S*election* S*trategy* 2 by the listed *Fact 1, Rule 1* and *Query 1*.

In addition, the patient's condition may also force the user to select several other services that should accompany the selected service (the primary service). In such a case, the user must know which services can be provided to the patient along with a primary service. To support such features, the user has to consider the services enabled by one service and with regard to patient's medical service consumption history and current condition. For example in Table 1, an Oxygen Supply Service is enabled by the Helicopter Service which means, if a user chooses Helicopter Service, he can also choose Oxygen Supply Service. However, for the Bus Service, he cannot choose the Oxygen Supply Service. The user has to manually interface different system components, namely: the patient data system, the service dependability knowledge and SWS discovery engine. Hence, the user faces a great deal of difficulties while trying to provide more than one service at a time to the patient.

TABLE I. RELATED SERVICES AND THEIR DESCRIPTION

Service Name	Service Quality Property: Cost, Relocation Duration	Dependent Services	Related Domain Data
Helicopter Service	$2000, 1 hour	Paramedic Service, Oxygen Supply Service, …	Patient Condition, Patient Respiratory Status, …….
Ambulance Service	$1000, 3 hours	Paramedic Service, Oxygen Supply Service, …	
Bus Service	$100, 4 hours	Paramedic Service …	
……	……..	……..	

We present a small example from healthcare describing problems users face to discover a service that depends on domain context, and motivating features to be supported. Suppose a patient is in a hospital in Antigonish and a medical professional determines that he should be relocated to Halifax for care that is more specialized. The user submits *Query 1* (see Textbox 1 below) to a SWS discovery engine, which will match the query with a service repository and provide a list of relocation services. However, this query does not incorporate other inputs such as patient condition, or patient disease history and the user later may need to select a service depending on such patient properties (examples of such selection strategies are *Selection Strategy 1* and *Selection Strategy 2)*. If none of the discovered services fits patient properties, the user must initiate another discovery request and lose precious time.

Query 1 can be answered by state-of-the-art SWS approaches like OWL-S or WSMO. However, *Selection Strategies 1* and *2* show how a user's decision may change based on patient properties. To support strategies representing domain awareness, the user must inspect the patient medical record and then make a decision based on the quality properties of the list of services discovered. The selection strategies can be articulated using domain object properties called *facts*

Query 1:"Get a Relocation Service that can relocate Patient P from Antigonish to Halifax."
Selection Strategy 1: If the Patient's Condition is Normal, Select the Low Cost Service for relocating the Patient from Antigonish to Halifax.
Selection Strategy 2: If the Patient's Condition is Critical, select the Fastest service for relocating a Patient from Antigonish to Halifax.
Fact 1: The condition of the Patient P is Critical.
Rule 1: If the patient's condition is Critical, use fastest mode of Relocation Service.
Fact 2: The Patient P has a Respiratory Problem.
Rule 2: If the patient has a Respiratory Problems, there should be an Oxygen Sservice supplied while relocating.
Rule 3: If the Patient's Condition is Critical, a Paramedic should accompany the Patient while relocating.
IQ 1: "Get the fastest Relocation Service to relocate Patient P from Antigonish to Halifax (uses Query 1, Fact 1, and Rule 1)."
IQ 2:"Get the fastest Relocation Service that supports Oxygen Supply Service while relocating Patient P from Antigonish to Halifax (uses Query 1, Fact 1, Fact 2, Rule 1, and 2)."
IQ 3: "Get the fastest Relocation Service that can support Oxygen Supply Service and Paramedic Sservice while relocating Patient P from Antigonish to Halifax (uses Query 1, Fact 1, Fact 2, Rule 1, 2 and 3)."

Textbox 1: Examples of Queries, Facts and Rules

Step 1: Determine if the Patient's Condition isCritical or not. If yes, then
Step 2: Select the fastest service manually from the list of services returned by the service discovery engine for Query 1.
Step 3: Find out if the Patient has a Respiratory Problem. If yes, then
Query 2: "Get an Oxygen SupplySservice that can be provided while Patient is transferring using fastest Relocation Service selected by Query1."
Step 4: If there is an Oxygen Supply Service that can be provided with the selected Relocation Service then continue to the next fact. If there is no such Oxygen Supply Service selected from Query 1, go back, reissue Query 1, and select the next fastest service. Repeat until an Oxygen Supply Service is found.
Step 5: If the Patient's Condition is Critical then,
Query 3: "Get Paramedic Service that can be provided while the Patient is relocating with the service selected by Query 1."
Step 6: If there is a Paramedic Service returned by the service discovery engine, the user could select that one. If there is no such service, the user has to select next fastest service from Query 1.

Textbox 2: A scenerio of user interfacting different systems manually

In addition, while the user tries to select services for a patient the user might need facts and rules in relation to selection strategies (e.g., facts and rules are *Facts 1 and 2, Rules 1*, *2* and *3 in* Textbox 1*)*. This situation requires the user to check the database, and do additional steps. Also, based on the service dependencies, the user may have to restart the process from the beginning if the selected service cannot provide all of the required services. A typical scenario is given below.

The domain facts and rules lead the user to do several more queries (Query 2 and Query 3) (see Textbox 2) and manually select services that are returned by traditional SWS discovery processes. However, one can see that from Query 1, Facts 1 and 2 and Rules 1, 2, and 3, we are really interested in getting

the results of the possible *inferred queries* IQ1, IQ2 or IQ3 (see Textbox1), where IQ3 is the optimal query. For time critical applications, taking such service dependencies into the discovery process makes it more efficient and user friendly.

We describe a framework for intelligent SWS description, discovery, and delivery that extends existing frameworks to: improve service discovery performance, facilitate integration of domain-based information, and interface with legacy systems such as workflow management systems. A workflow is a collection of interconnected Tasks with a specific control flow. Each Task has a specification representing the action needed to be carried out. We propose the notion of Service Enabled Workflow (SEW) which will allow us to discover services using the task specification as a query to the SWS discovery engine which will determine services that can carry out the action required by the task. SEW can provide desirable features such as: a) distributed workflow execution utilizing the standalone nature of the services; b) service collaboration among various service providers as SEW can support the choice and execution of services from different providers, using them in a single workflow; c) decentralization of the workflow design, execution, and low coupling among workflow design and execution environment.

II. PROPOSED MODEL AND ARCHITECTURE

Our framework focuses on the easy integration of SWS with a domain context and facilitates the interfacing with systems developed using traditional approaches. The basic approach of service discovery traditionally contains a Service Discovery Engine, a Service Repository, and a Domain Service Ontology; we add a data and context integration component and a service metadata ontology. We can incorporate the data and context of legacy system by facts and rules that can be utilized by SWS discovery for context and real-time data service discovery and selection. The model supports context dependent service discovery using two ontologies which provide the logic for a given service selection: 1) Service Metadata Ontology which contains the service relationships with legacy system data and context; 2) Domain Facts and Rules Ontology. The Service Metadata Ontology consists of a) Service Domain Data Dependencies and b) Inter-Service Dependencies. These ontologies allow us to do reasoning over service metadata, can be specified using OWL-2, and, can be accommodated in both the OWL-S and in WSML-DL versions of WSMO approaches. We now discuss desirable features of a hybrid SWS based decision support systems.

Domain Integration and Context Aware Service Discovery: The "Relevant Domain Data" model articulates the association of a service with the relevant domain data; in Table 1 it includes column 1 and column 4. Based on the relevant domain data stated, we fetch data from the legacy system and assert them as facts in the Domain Facts and Rule Ontology. We can then use these facts asserted based on real-time data in the SWS discovery process. Asserting a fact about a domain at runtime, such as *Fact 1,* depends on the availability of the Patient P's property "*Patient Condition*" and the availability of property value "*Critical*" which is gathered in real-time from a database. Rules depending on the system's situation and data context that express the decision strategy related to a fact are also asserted in the domain ontology. At runtime, these rules will change the result of the discovery query to that of an inferred query due to a more refined search and discovery of services. Applying the facts and rules during discovery, the answer to an inferred query can will obtained by applying reasoning. This will reduce the need of user inspection and interaction to get a service that best suits the user's need.

Service Metadata Based Reasoning and Discovery: The Inter-Service Dependency Relationships model can enable us to do on the fly service orchestration which can also save the number of queries required. The model expresses the relationships between services in the service spectrum. A list of interdependent services are provided in the service description which then can be used in the discovery process and reasoning. E.g., in Table 1, if the user selected a *Relocation Service* like *BusService*, the user cannot select *OxygenSupplyService* because it is not supported but can select *ParamedicService*. So, depending on the need of the patient and service relationship, a service selection decision can be made.

Aggregation Query Support during Discovery: It is hard to support some special queries like *"Get the fastest relocation service"* using existing SWS discovery techniques. This requires that we incorporate procedural programming capability in a service discovery query. Procedural programming operation will be used along with DL based ontology query languages e.g. SPARQL Protocol and RDF Query Language (SPARQL) [4] and Semantic Query-Enhanced Web Rule Language (SQWRL) [14]. This will allow users to express complex aggregation and procedural operations easily and intuitively in a discovery query.

Service Enabled Workflow: SEW imagines workflow as a collection of tasks with control flows where tasks are carried out as services. A workflow task has defined specifications, which can be imagined as a user query for the discovery of a service and the workflow engine can ask the service discovery engine to discover services according to these task specifications. The workflow user may select a service to execute from the discovered list of services. Continuing in this fashion, we can provide dynamic composition of services: the overall result is SEW. SEW is a desirable feature that can easily provide workflow collaboration support with minimum efforts thorough service discovery and runtime composition.

We propose a SOA based architecture shown in figure 2, which supports integration of different domains; it consists of the following components:

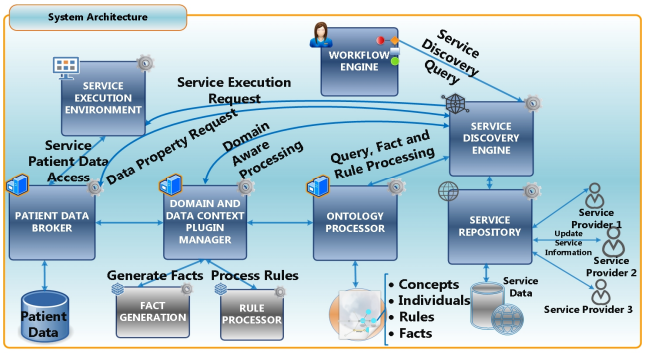

Figure 2: System Architecture

Workflow Engine: works as a user query generator and execution engine that enables Service Enabled Workflow.

Service Discovery Engine: serves as a central communication hub. It also carries out several decision-making tasks about service dependency reasoning, and carries out rules resulting in procedural steps.

Service Execution Environment: a server environment providing service runtime requirements and run services.

Patient Data Broker (Object Data Broker): works as a broker to get data from external systems.

Ontology Processor: is responsible for managing the ontologies and querying the ontologies.

Service Repository: is responsible for holding information about services provided by the service providers.

Domain and Data Context Plugin Manager: is responsible for the facts and rules related processing and domain based plugin management.

III. Related Work

The prominent conceptualizations of the SWS are OWL-S [1][11] and WSMO [3][15]. OWL-S helps software agents to discover web services that satisfy some specified quality constraints also provide a minimal set of composition templates. However, these abstract definitions can only be applied in a static service composition and can only be arranged as a predefined combination of services in the ontology. In [6], several types of inter-process dependencies are modeled using UML including Enabling, Cancelling, Triggering, and Disabling dependencies. WSMO also provides a concept vocabulary to express service description in terms of IOPEs but it currently only supports syntactical matching of a user's goal against service descriptions. OWLS-MX [9] and WSMX [7] are the SWS execution and testing environments for the SWS developed using OWL-S and WSMO approaches, respectively. OWLS-MX implemented the hybrid service discovery matchmaking using the OWL-2 reasoner Pellet. OWLS-MX and WSMX both support SW query languages SQWRL or SPARQL to perform semantic discovery of services but do not use domain data dependent facts and rules to discover services. SADI [16] provides a design pattern for publication of services, interoperability with traditional WS, and, semantic discovery and workflow generation based on service input/output transition metadata. SADI does not support domain data and context based service discovery and selction via integration and interoperation with legacy systems and data. Presently, there are a variety of approaches to improve the accuracy of a service discovery process, including collecting and integrating user feedback [2] and the addition of contextual information by defining design time semantic based user context [12]. In our approach, the service description enables us to foresee the services dependencies and reason about them to discover services that best suit the system and context conditions based on described facts and rules. In [5] a conceptual model of task-based workflow is provided that motivates our proposed Service Enabled Workflow. We extend the approach of [5] to support closer relationships with systems and contexts, and improve the state-of-the-art of such workflow systems. The Nova Workflow Workbench [10] is a task based workflow engine equipped with a high-level language, T□, [13] which is used to write task specification which include integration of data from a domain ontology. We plan to integrate our service discovery process to accept the task specification. The discovery process then can provide the selected service to the Nova Workflow engine, which executes the service to accomplish the task.

IV. References

[1] Ankolekar, A. et al., 2002. "DAML-S: Web service description for the semantic web." In The Semantic Web—ISWC 2002, Springer, p. 348–363.

[2] Averbakh, A., et al., 2009. "Exploiting user feedback to improve semantic web service discovery." In The Semantic Web-ISWC 2009, Springer, p. 33–48.

[3] Davies, J. et al., 2006. Semantic Web technologies: trends and research in ontology-based systems. Wiley.

[4] Garc'ia, J. et al., 2012. "Improving semantic web services discovery using SPARQL-based repository filtering." Web Semantics: Science, Services and Agents on the WWW.

[5] Grossmann, Georg et al., 2011. "Conceptual modelling approaches for dynamic web service composition." In The evolution of conceptual modelling, Springer, p. 180–204.

[6] Grossmann, G. et al., 2008. "Modelling inter-process dependencies with high-level business process modelling languages." In Proceedings of the fifth Asia-Pacific conference on Conceptual Modelling-Volume 79, p. 89–102. Austrian Computer Society, Inc.

[7] Harold, M. 2008. "WSMX documentation." http://maczar.deri.ie/papers/wsmx-documentation.pdf.

[8] Keller, U. et al., 2004. "Wsmo web service discovery." WSML Draft, http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112.

[9] Klusch, M. et al., 2009. "OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services." Web Semantics: Science, Services and Agents on the World Wide Web 7(2): 121–133.

[10] W. MacCaull and F. Rabbi, "NOVA Workflow: A Workflow Management Tool Targeting Health Services Delivery," in FHIES'11, ser. LNCS, vol. 7151. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 75–92.

[11] Martin, D. et al., 2004. "OWL-S: Semantic mark-up for web services." W3C member submission 22: 2007–04.

[12] Merla, C. 2010. "Context-Aware Match-Making in Semantic Web Service Discovery." Int'l Journal of Advanced Engineering Sciences and Technologies 9(2): 243–247.

[13] F. Rabbi and W. MacCaull: "T-Square: A Domain Specific Language for Rapid Workflow Development," in ACM/IEEE 15th Conf. on Model Driven Engineering Languages & Systems (MODELS 2012), Innsbruck, Austria (September, 2012). Proc, Lecture Notes in Computer Science, Volume 7590. pp. 36-52.

[14] Rodriguez, J. et al., 2010. "Improving Web Service descriptions for effective service discovery." Science of Computer Programming 75(11): 1001–1021.

[15] Wang, H. et al., 2012. "A formal model of the Semantic Web Service Ontology (WSMO)." Information Systems 37(1): 33–60.

[16] M. D. Wilkinson, et al., 2011. The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation. *Journal of biomedical semantics*, *2*(1), 8.