CASD: A Framework for Context Aware Service Discovery and Selection

Altaf Hussain[1(✉)], Wendy MacCaull[1], and Yngve Lamo[2]

[1] St. Francis Xavier University, Antigonish, Nova Scotia, Canada
altaf_sust_82@yahoo.com, wmaccaul@stfx.ca
[2] Bergen University College, Bergen, Norway
yngve.lamo@hib.no

Abstract. We present the architecture for a framework for semantic web-based service discovery, suitable for integration with relational database systems. Existing discovery algorithms often lead to poor results due to limitations of the service description used and lack of domain data. Our framework incorporates context aware service discovery vis a vis dynamic (run-time) update using relational (domain) data from a legacy system. A template for service ontologies is provided, so the user may represent their services and the interrelationships between them. The domain data is represented as an OWL-ontology. The systems takes information in the service and domain ontologies and performs rule based reasoning using rules articulating inter-service dependencies as well as dependencies between services and domain data. An aggregation over service quality properties allows the aggregated selection of the best-suited services. The framework is implemented as a web application following the Service Oriented Architecture; extensive testing shows that the system is robust. Features of the framework are illustrated using a detailed case study for the health-care domain.

Keywords: Ontology · Data integration · Dynamic context representation · Context aware service discovery · Domain and Service Integration

1 Introduction

Nowadays, people are dependent on services which are entities that offer value to consumers. Web Service (WS) [10] is the technology that makes various services available as consumable entities, accessed and consumed through computers, such as the Email Service. WS technology, backed by Service Oriented Architecture (SOA) [16] has gained a lot of focus and popularity in the commercial computing sector as an enabling technology for service planning, development, delivery and management methodology. In addition, with the advancement of Relational Database technology, businesses have invested in and developed data

Acknowledgment: The second author was supported by grant by the Natural Sciences and Engineering Research Council of Canada.

I. Ciuciu et al. (Eds.): OTM 2016 Workshops, LNCS 10034, pp. 252–264, 2017.
DOI: 10.1007/978-3-319-55961-2_26

driven dynamic applications resulting in a new spectrum of web applications supporting business-to-business integration, e-commerce, and industry wide collaboration. These applications are empowered by the WS technology, which provides a platform of independent communication and machine-to-machine interaction. However, WS technologies need extensive human involvement for optimizing service discovery, selection and composition [16].

In recent years, a new paradigm has evolved, called the Semantic Web (SW) [18], which supports machine-readability through an Artificial Intelligence inspired content markup language based on the Web Ontology Language (OWL) [17]. Its ability to express logical relations among entities on the web has led to a new kind of WS technology called the Semantic Web Service (SWS) [18]. Due to the lack of adequate service description, many discovery approaches often lead to poor discovery results. Most current approaches for service discovery perform syntactic matching and semantic Input/Output matching which retrieve services using a description that contains particular keywords from the user's service query. Context aware service discovery approaches provide a promising way to more accurately discover services based on the domain or user's context. Contextual information, by nature, is dynamic and can reflect the current state and condition of the domain. However, existing context aware service discovery approaches provide limited support for the mechanisms required to dynamically represent and update the context.

Here, we present a framework called CASD for context aware, domain data dependent and inter-service relationship based service discovery, which automatically selects the best suited services via aggregation over service quality properties. This paper is organized as follows: first we provide a motivating example. Section 2 provides an overview of the CASD framework. Section 3 provides a brief description of the knowledge base of the framework. Sections 4 and 5 provide the system architecture and performance analysis of the framework, respectively. Related and future work are provided in Sect. 6.

1.1 Motivating Example

Suppose a person gets injured in an accident. Depending on the status of the patient (i.e., context), paramedics need to decide how to relocate the patient to a medical facility. For example, if the patient's condition is critical, the fastest mode of relocation should be used, otherwise the cheapest relocation service may be used. However if the patient has respiratory problems, oxygen supply should be ensured. Suppose the paramedics have the option to select from an *Ambulance-Service*, a *HelicopterService*, and a *BusService*. Among these services suppose the *HelicopterService* is the fastest service, the *BusService* is the cheapest, but only the *Ambulance Service* has oxygen supply. Now, if the paramedics decide that the patient's condition is normal and does not require oxygen, the *BusService*, i.e., the cheapest service is the best option for the context. On the other hand, if patient's condition is critical and he is not suffering from respiratory problems, paramedics can select the fastest service for relocation, that is, the *HelicopterService* service.

However, if the patient's condition is critical and he is also suffering from respiratory problems the *Ambulance Service* should be selected as that is the only service that provides oxygen. Here we have provided a simplified imaginary emergency care guideline. Real-world guidelines for emergency care, response and life support are obviously much more complex [7,8].

In the above example, the patient status after the accident such as: *Patient Condition is critical, Patient shows Respiratory Problems* is referred as the *Context*. The 'if-else' rules on which the paramedics decide what to do are called *Context Based Rules*. The quality the paramedics seek in a specific service such as the fastest service, or the cheapest service, is found by performing quantitative reasoning over some of the service properties say: the cost or speed of the relocation. We refer to *Cost and Speed* as Quality of Service (QoS) properties. This type of discovery and selection of services is referred to as *Context Aware Service Discovery*. The selection of *Ambulance Service* which enables *Oxygen Supply Service* is based on the relationships between these two services which we refer as an *Inter-Service relationship*. In addition, selection of a service may differ based on patient clinical history or age, which may restrict consumption of a service. Eg., humidified oxygen supply may be required if the patient is a child. Hence, the service discovery process should also take the domain data (in this case, patient data) into consideration. We refer to this type of service discovery as *Domain Data Dependency based Discovery*.

2 CASD Framework: Features and Approach Overview

In this section we briefly discuss the features and provide an overview of the service discovery approach of our framework. The features include:

1. Interoperability Between Legacy Systems and SW Systems: The CASD framework provides automatic integration with a legacy system's relational database (RDB) or domain database and SW system. The CASD framework provides and uses a tool called "RDB2O"to convert a domain database to semantic web accessible data or ontology (called Domain ontology).
2. Dynamic Context Representation and Update: Domain database schema can be used as a source of vocabulary for concept and relationships to create the Domain ontology, which eventually represents the context. As we create a Domain ontology from a domain database automatically, the CASD framework can represent the context dynamically for a domain.
3. Service Ontology Template: In the CASD framework, we provide a Service ontology template, which provides the concepts and relationships necessary to include services that can be accessible in a domain, the service quality properties (QoS) and the inter-service dependency properties. The inter-service relationships express whether a consumer can consume a service depending on the consumption history of another service by the consumer.
4. Context Aware Service Discovery and Aggregated Selection: By combining the domain and the service ontologies we can define service discovery and selection rules using relations and concepts from both ontologies. The CASD

framework utilizes the context present at runtime in the domain ontology to discover and automatically select the best-suited services. The context aware discovery process uses rule based reasoning through the Pellet reasoner [19] for discovery of services.

(a) Domain Data Dependent Discovery and Selection: The CASD framework can discover and select services not only based on the QoS properties, but also on the domain data. For example, in the health-care domain, depending on whether a patient's condition is *critical* or not, the CASD framework can select the *fastest* or the *cheapest* service.

(b) Inter-service Relationship based Service Discovery: Based on service relationships, there might be services which may not allow consumption of other services or there might be services that have to be consumed with another service. In the motivating example, we have to select the second fastest service, the *Ambulance Servie*, which is the only service that allows the consumption of the required *OxygenSupplyService*.

5. Accessibility and Extendability of the CASD framework: The CASD framework is designed and developed as a web application, which is easily accessible through browsers.

3 Knowledge Base and Context Aware Service Discovery

The KB used in the CASD framework consists of the following:

1. Domain Ontology: We create the Domain ontology from a domain database automatically using our tool, RDB2O. RDB2O creates the ontology $\mathsf{T}_{\mathsf{Box}}$ by converting the domain database schema. The $\mathsf{T}_{\mathsf{Box}}$ contains the concepts and relationships which are created based on a database-ontology conversion mapping algorithm in RDB2O. The Domain ontology $\mathsf{A}_{\mathsf{Box}}$ consisting of facts, is populated with domain data at the runtime.
2. Service Ontology: The Service Ontology $\mathsf{T}_{\mathsf{Box}}$ provides concepts and properties for domain services. The user can instantiate related $\mathsf{A}_{\mathsf{Box}}$ or instances of services and their relationships based on the services offered by the domain. The Service ontology also contains another type of service called the RuntimeQoSFlagService, which is not consumable and is created at runtime by CASD framework based on QoS properties of individual domain services.
3. Context Based Rules: We express context based rules using the Semantic Web Rule Language (SWRL) [12], which is an expressive OWL-based rule language. SWRL includes a high-level abstract syntax for Horn-like rules. We can write context based rules using concepts and properties from both the Domain and Service ontologies. The following is a rule expressed in SWRL: *If the the patient's condition is critical, select the fastest relocation service*: Patient(?p) $\wedge$ hasCondition(?p,"Critical") $\wedge$ MaxSpeedQoSFlagService (?maxSpeedServ) $\rightarrow$ SelectedService(?maxSpeedServ).

Figure 1 shows the items in the CASD process for discovery and selection of services, which are briefly described below:

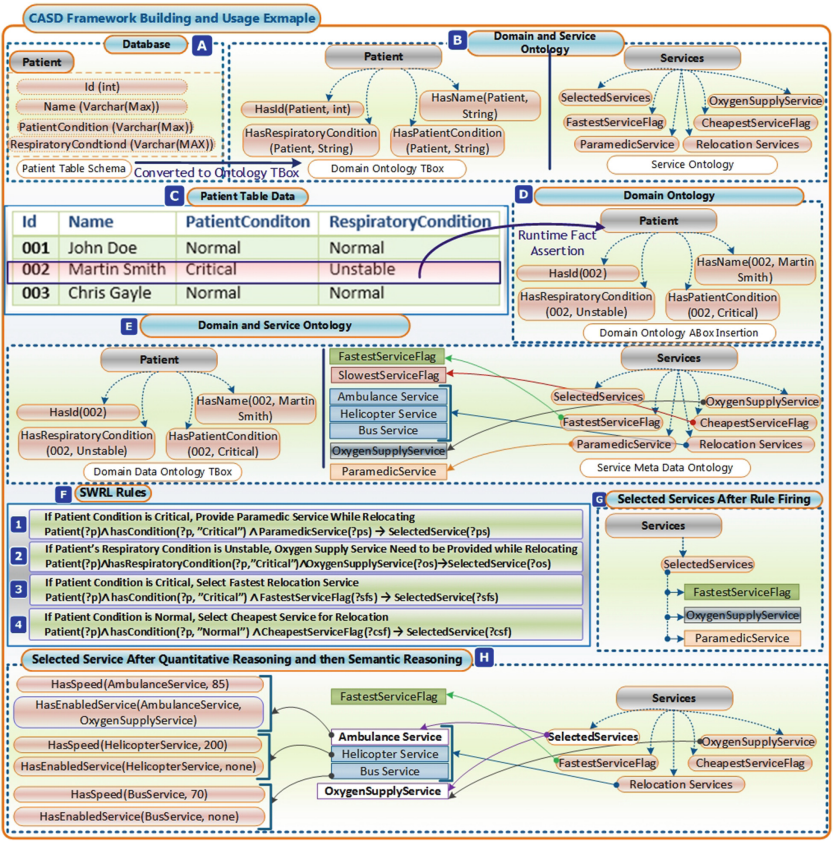

Fig. 1. Example of setup steps and usage of the CASD framework

A. Here we see an example legacy system relational database schema that is to be used as domain data provider. In the figure, the schema of a *Patient* table is shown, which is part of a *health-care* application, eg., the *EHR* [2]. We convert the schema of the database into an ontology to get the Domain ontology.

B. In this item, we combine the Domain ontology, the Service ontology and the Context Based Rules to create the KB.

C. Shows example data in the *Patient* table.

D. Based on the user query and rules in the domain, we gather data from the legacy system database and assert them into the Domain ontology i.e., into the KB. Item D shows how the data for the requested patient, whose id is 002, is collected from the domain database and asserted in the Domain ontology.

E. Shows examples of populated Domain and Service ontologies with required patient data and service instances, respectively.

F. Shows rules the reasoner applies to discover services for the current context.

G. In this item, the services that are required to be selected are shown which resulted from executing the rules in the reasoner. From rule 1, the reasoner concluded that the patient needs OxygenSupplyService. From rule 2, it is reasoned that the patient requires the fastest service for relocation. Combining these results we can conclude that the user needed to be relocated using the fastest service that enables OxygenSupplyService.

H. In item H, the CASD applies aggregation over the QoS property *Speed*, to find the fastest service that enables OxygenSupplyService. Here the AmbulanceService is the fastest service that supports, i.e., enables the OxygenSupplyService.

4 Architecture

We design the CASD framework following the SOA and N-tier architecture which makes integration of different components easier while providing better separation among user interface, logic layers and data layers. We have developed the framework using *C#.NET, ASP.NET* and *Java*. For ontology and rule processing we use DotNetRDF API [20], OWLAPI and Jena API [4]. The CASD frame-

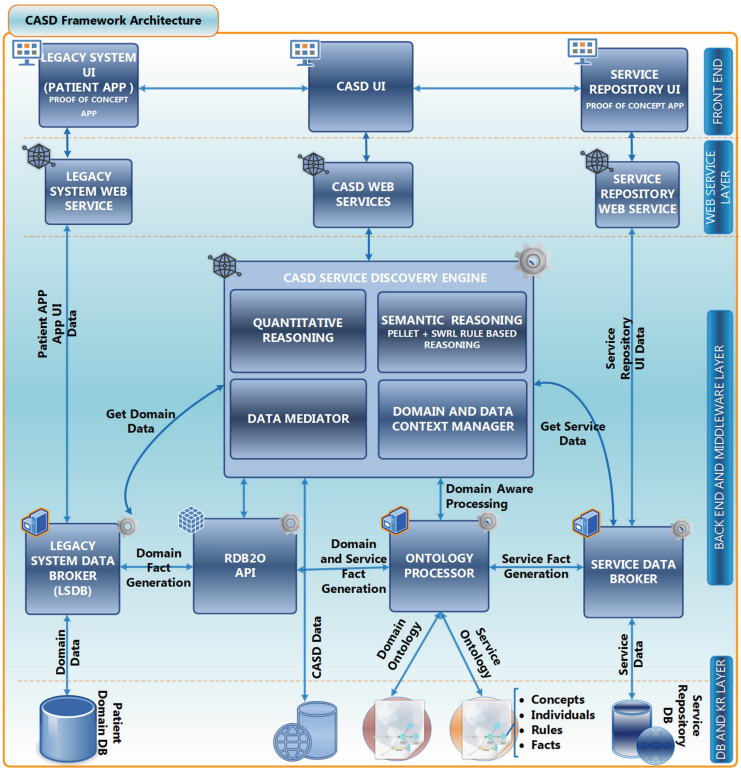

Fig. 2. Full architecture of the CASD framework

work provides the RDB2O tool [13] and uses it as an API to easily convert and integrate legacy system data and can leverage real-time data to generate ontology facts and represent context using an ontology. The architecture of the CASD framework is given in Fig. 2 and has the following tiers:

1. CASD Front End: The CASD front end is developed as a web application which can be accessed from browsers. We have also provided integration of the CASD framework with two other applications, namely, a simple healthcare system and a service repository.
2. Web Service Layer: We provide service discovery functionalities of the CASD framework as web services. This layer also provides web services for a healthcare system and service repository, which are used for integration with the CASD framework.
3. The Back End and Middle-ware Layer: The back end tier and the middle-ware layer components of the CASD framework are responsible for service discovery and selection, data conversion, accessing data and updating ontologies at runtime.
   (a) Legacy System Data Broker (LSDB): This collects data from a domain database depending on the rules using algorithms specified based on types of atoms in the rule. For each atom in the rule, the LSDB creates SQL queries at runtime to select data from the domain database.
   (b) Service Discovery Engine (SDM): The SDE accepts user queries from the UI module and determines what data need to be collected, after consulting the Data Mediator, LSDB and KB modules.
   (c) Reasoning: The Reasoner is responsible to carry out the rule based reasoning using Pellet. A SDM request starts the reasoning process and gathers the result.
   (d) Quantitative Reasoner: This performs procedural processing and provides the aggregation functionality and, indeed, any post processing required.
   (e) RDB2O API: The RDB2O API allows the CASD framework to access the database to ontology conversion mapping used for Domain ontology creation at runtime and thus allows the Ontology Processor (see below) to assert facts in the Service and Domain ontologies according to the mapping used in conversion.
   (f) Data Mediator: The Data Mediator is responsible for communicating with the LSDB. It passes the request from the SDM to the LSDB for the data required for the SDM and also collects data and relays the data to the SDM.
   (g) Domain Data and Context Manager: The Manager keeps records and relationships of domains, associated ontologies and runtime context for a particular domain.
   (h) Service Data Broker: The Service Data Broker allows the CASD framework to access a service repository at runtime to generate service facts and passes these facts to Data Mediator for assertion into the Service ontology by the Ontology Processor.
   (i) Ontology Processor: This updates and modifies both the Domain and Service ontologies as facts become available at runtime.

4. Data and Knowledge Representation (KR) Layer: The Data and KR layer contains the domain database (patient database), the service repository, the database for the CASD application and the Domain and Service ontologies.

5 Robustness and Performance of the CASD Framework

In this section we provide a performance analysis of the framework with basic functional testing and load based testing. The CASD framework is deployed as a web application on IIS 7.0 on a Lenovo ThinkPad running on Windows 7 64 bit. The computer has 12 GB ram, Intel(R) Core (TM) i5 @ 2.50 GHz.

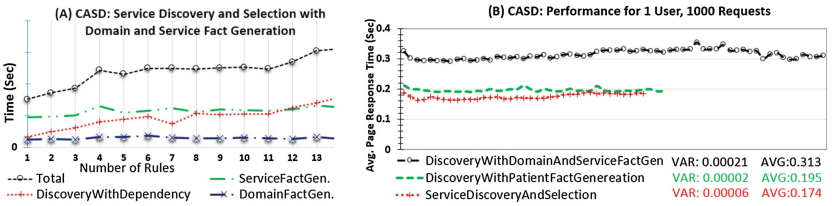

Fig. 3. Testing and performance analysis of the CASD framework

There are settings in the CASD framework called "GenerateServiceFacts" and "GenerateDomainFacts", which, respectively, determine whether to generate service facts to insert into the Service ontology with updated QoS properties from the service repository and domain facts to update the domain ontology or context with up-to-date domain data at runtime. Otherwise, it reuses the facts from the previous run and can save time for service and/or domain facts generation.

Figure 3(A), shows the time required for service discovery and selection (line: ..+..), for domain fact generation (line:–X–X–), for service fact generation (line: –..–..) and total time for discovery and selection when both domain facts and service facts generation are enabled (line:- -o- -o- -), as we gradually increase the number of rules from 1 to 15. Effectively, the total time for discovery is the sum of time required for service discovery and selection, domain fact generation, and service fact generation. As we can see from the graph, the time for domain fact generation (line:–X–X–) remains flat as we apply services for only one patient and for each request, the CASD framework generates the domain ontology for the same patient (or domain object) regardless the number of services to discover or number of patients in the database. The time required for discovery and services (line:..+..+..) increases as the number of rules (for this domain, executing 15 rules, discover 13 services) increases, as more QoS properties and more inter-service dependencies by quantitative reasoning have to be satisfied. The time required for runtime service facts generation (line: –..–..) increases in a linear fashion as the number of services in the repository increases.

We measured the performance of the CASD framework for 1000 requests to execute 10 rules by a single simulated user using the Visual Studio 2013 load test tool. The graph in Fig. 3(B) shows the average page response time for the different settings discussed above. The page response time constitutes the time required for posting the request to the server for a particular patient, receiving response or rules execution result, rendering discovery and selection of services. The time required for service discovery and selection without service and patient facts generation (line: .+.+.+.) remains flat over the time of tests resulting in a fixed page response time. The time for required for discovery and selection of services with only domain (patient) facts generation (line:.....) is somewhat higher, than the time required with no facts generation (line .+...+..+.). The time required for service discovery and selection with both service facts and patient facts generation enabled is shown using line:-o-o-o-o-, which is somewhat higher than other two test criteria.

We have also tested the framework with more scenarios (due to space limitations these results are not included) such as: (i) Load test for 10 vs 15 user executing 10 rules, (ii) Executing 10 rules for a fixed patient vs a different patient for each run, and (iii) Load test for 1 vs 10 vs 15 users using the system to execute 1000 requests with 15 rules. For each of these test cases, we found the result is satisfactory and consistent (see the first author's Masters thesis, [13]).

6 Related and Future Work

The desirability for automation in service discovery and collaboration backed the escalation of the SWS with the maturity of the SW. The most widely used conceptualizations of the SWS are the OWL-S [18], the WSMO [11] and the SADI framework [21]. OWL-S helps software agents discover web services that satisfy some specified quality constraints (QoS) in terms of Input, Output, Preconditions, Post-condition and Effects (IOPEs). OWL-S also helps the service composition and service interaction by providing a minimal set of composition templates including: Sequence, Split, Unordered, Split+Join, etc. OWL-S does not provide any methodology for domain data integration with service model, thus does not support service discovery based on domain data, which is supported by the CASD framework. On the other hand, WSMO provides a concept vocabulary to express service description in terms of IOPEs and currently only supports syntactical matching of a user's goal against service descriptions. The WSMO supports selecting a service based on one or the other of two criteria, namely, "always the first" or multi-criteria selections, which depends on nonfunctional properties like reliability, and security. The SADI [21] framework discovers services based on IO matching and can make dynamic composition of services to match service IO requirements. The CASD framework discovers services based on dynamic QoS properties, inter-service dependency relationships, and domain context and automatically selects best-suited services.

OWLS-MX [15] and WSMX [11] are the SWS execution and testing environments for the OWL-S and WSMO approaches, respectively. OWLS-MX implemented the hybrid service discovery matchmaking (semantic matching of service

description and user query provided) using the OWL-2 reasoner, Pellet. The OWL-S API uses the Jena Semantic Web Framework under the hood to modify the OWLS-MX matchmaker ontology. WSMX can work with Pellet. We are using Pellet and Jena and the OWL-API for reasoning, ontology manipulation, fact writing, and running SWRL rules.

In [9], several types of inter-process dependencies are modeled using UML including Enabling, Canceling, Triggering, and Disabling dependencies. However, no implementation was provided. We have integrated two of these service dependencies in the CASD framework by implementing the Enabling and Disabling dependencies.

In Table 1 we summarize features of the different frameworks for service discovery and selection (Based on literature provided). Due to space limitations we are not providing comparison with OWLS-MX.

Table 1. Comparisons among different frameworks for service discovery and selection

Features	CASD	WSMX	SADI
Service ontology	Service QoS and Inter-service rel	WSMO ontology	Service IO
Domain data integration	Domain ont. (run-time) d	Not supported	Not supported
Context rep	Service and domain ont.(dynamic)	Not supported	Not supported
Context data source	RDB (run-time data)	Not supported	Not supported
Reasoning support	Yes (rule based)	Semantic IO matching	Yes (SPARQL Query)
Domain setup by user	Yes	No	Limited
User SW knowledge req	Not needed for basic user	Required	Required
Context aware discovery	Yes	No	No
Aggregated selection	Yes	No	No
Inter-service rel. discovery	Yes	No	No
Discovery focus	Which service, when to use	Which service	Which service
Legacy system integration	Yes	Not supported	Not supported
Security support	Implemented	No	Implemented
Accessibility and extendibility	Web based, Web APIs, SOA	Desktop application	Web based, Web APIs

S. Cuddy et al. proposed a context aware service discovery technique base on static and dynamic service properties in [5]. This approach provides dynamic context representation with XML, using weighted service properties based on the user preferences, which must be provided for each discovery request. Also, if multiple services are discovered from a request, the approach randomly selects a service without taking the most suitable service into account. The CASD framework discovers services based on context represented from real-time domain data as well as dynamic service properties and inter-service dependencies. The CASD framework also features automatic service selection using aggregation over service properties. T. Broens et al. proposed another context aware service discovery approach which represents the context using an ontology [1]. This approach proposed using a pre-defined vocabulary for a domain built by consensus from a related community. The approach proposed by Xiao et al. [22] also represents context using an ontology, but uses some static features of the services such as location, keywords, etc. In our framework, we use a dynamic context representation using an ontology, that uses vocabulary directly from the Domain

ontology which we gather by converting the domain database. Moreover, our framework supports most up-to-date context values as the framework can access the domain data and update the Domain ontology to reflect the changes in the context. Table 2 provides comparisons among different context aware service discovery and selection approaches with the CASD framework (comparisons are provided based on the literature provided). A more comprehensive analysis of related work can be found in [13].

Table 2. Comparisons among different approaches for context aware service discovery and selection

Features	CASD framework	S. Cuddy et al.	Xiao et al.
Context representation	Service and domain ontologies	XML	Ontology
Dynamic context vocabulary	Yes (service and domain onto.)	Partially dynamic (QoS)	Pre-defined
Runtime context update	Converted from RDB (at run-time)	User defined	User defined
Discovery criteria	QoS and domain context	QoS (IO matching)	QoS
Domain data integration	Yes	No	No
Data dependent discovery	Yes	No	No
Inter-service rel. discovery	Yes	No	No
QoS based discovery	Yes (dynamic QoS, aggregation)	Yes (partially dynamic)	Yes (static QoS)
Automatic service selection	Yes (fully automatic)	Random	Yes (static QoS)

To provide integration of legacy systems with SW systems, we need a tool that can convert a database to an ontology and can be used as an API for runtime data conversion. There are tools like [3,6] that can provide mappings of domain databases to ontologies. However, there are not many implemented tools available but some require intermediate manual mapping via bridge programming and do not provide run-time data conversion as an API. DB2OWL [6] and RDB2OWL [3] each provides an easy mapping process but the first can not handle self-reference of Tables and the second requires a manual mapping process. In our RDB2O tool, the mapping is based on the relationships and constraints specification among tables which can provide a database to ontology conversion automatically (without any manual mapping) and can be used as an API for run-time data mapping. We implement RDB2O using *C#.NET* and it can convert MS-SQL Server databases.

Our approach is still preliminary and some improvements can be made. The generation of ontological facts at runtime from a domain database is based on a serial algorithm. For larger domain databases, we may gain significant performance benefits if the fact generation process can be parallelized. Also when running multiple rules, which results in the discovery and selection of multiple secondary services, the discovery process is dependent on the sequence of the specific rule execution. This situation arises when we try to satisfy inter-service dependencies. A promising way go to froward is by defining discovery priorities based on service types for secondary services based on domain data.

Earlier [14] we discussed providing Service Enabled Workflow (SEW) leveraging the CASD framework. 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 services to the CASD framework. The workflow user may select a service to execute from the discovered list of services against a task specification. Continuing in this fashion, we can provide dynamic composition of services: the overall result is SEW. In order to achieve SEW support using the CASD framework, we need to implement a workflow engine to provide control flow and develop an interconnection with the workflow engine and CASD framework for service discovery and selection for discovering services based on workflow task specification.

References

1. Broens, T., Pokraev, S., Sinderen, M., Koolwaaij, J., Dockhorn Costa, P.: Context-aware, ontology-based service discovery. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 72–83. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30473-9_7
2. Canada Health Infoway: EHRS Blueprint Version 2. https://www.infoway-inforoute.ca/en/component/edocman/391-ehrs-blueprint-v2-full/view-document. Accessed Mar 2016
3. Čerāns, K., Būmans, G.: RDB2OWL: a language and tool for database to ontology mapping. In: 27th International Conference on Advanced Information Systems Engineering (2015). http://ceur-ws.org/Vol-1367/paper-11.pdf
4. Clark, P.: The OWL API. http://owlapi.sourceforge.net/. Accessed Mar 2016
5. Cuddy, S., et al.: Context-aware service selection based on dynamic and static service attributes. In: Wireless and Mobile Computing, Networking and Communications, vol. 4, pp. 13–20. IEEE (2005)
6. Cullot, N., Ghawi, R., Yétongnon, K.: DB2OWL: a tool for automatic database-to-ontology mapping. In: 15th Italian Symposium on Advanced Database Systems (SEBD 2007), pp. 491–494 (2007)
7. Emergency Health Service Branch, Ministry of Health, Long-Term Care: Basic life support patient care standard. http://www.health.gov.on.ca/. Accessed Mar 2016
8. Emergency Medical Services, Manitoba Health, Province of Manitoba: Emergency treatment guidelines. http://www.gov.mb.ca/health/ems/guidelines/etg.html. Accessed Mar 2016
9. Grossmann, G., et al.: Modeling inter-process dependencies with high-level business process modeling languages. In: Hinze, A., Kirchberg, M. (eds.) 5th Asia-Pacific Conference on Conceptual Modelling, vol. 79, pp. 89–102 (2008)
10. Haas, H., et al.: Web services glossary. W3C Working Group Note (2004)
11. Herold, M.: WSMX documentation. Digital Enterprise Research Institute Galway, Ireland 3 (2008). http://www.wsmx.org:8080/wsmxsite/papers/documentation/WSMXDocumentation.pdf
12. Horrocks, I., Boley, H., et al.: SWRL: a semantic web rule language combining OWL and RuleML. W3C Member Submission **21**, 79 (2004)
13. Hussain, A.: A framework for context aware service discovery and selection, MSc. thesis. St. Francis Xavier University (2016)
14. Hussain, A., MacCaull, W.: Context aware service discovery and service enabled workflow. In: 4th Canadian Semantic Web Symposium, CEUR-WS pp. 45–48 (2013). http://ceur-ws.org/Vol-1054/paper-11.pdf

15. Klusch, M., et al.: OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services. Web Semant. Sci. Serv. Agents World Wide Web **7**(2), 121–133 (2009)
16. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall Professional, Upper Saddle River (2005)
17. McGuinness, D.L., Others: OWL web ontology language overview. http://static.twoday.net/71desa1bif/files/W3C-OWL-Overview.pdf. Accessed Mar 2016
18. McIlraith, S.A., et al.: Semantic web services. Intell. Syst. IEEE **16**(2), 46–53 (2001)
19. Sirin, E., et al.: Pellet: a practical OWL-DL reasoner. Web Semant. Sci. Serv. Agents World Wide Web **5**(2), 51–53 (2007)
20. Vesse, R., Team: dotNetRDF - semantic web, RDF and SPARQL library for C-sharp/.Net, http://www.dotnetrdf.org/. Accessed Mar 2016
21. Wilkinson, M.D., et al.: The semantic automated discovery and integration (SADI) web service design-pattern, API and reference implementation. J. Biomed. Semant. **2**(1), 1–23 (2011)
22. Xiao, H., Zou, Y., et al.: An approach for context-aware service discovery and recommendation. In: 2010 IEEE International Conference on Web Services, pp. 163–170. IEEE (2010)