DISCO: A Dynamic Self-configuring Discovery Service for Semantic Web Services

Islam Elgedawy
Computer Engineering Department, Middle East Technical University, Northern Cyprus Campus, Kalkanli, Guzelyurt, Mersin 10, Turkey

Keywords: Service Discovery, Self-configuring Services, DISCO, JAMEJAM, Service Knowledge Management.

Abstract: The service discovery process involves many complex tasks such as service identification, composition, selection, and adaptation. Currently, there exist many discovery schemes that separately handle such discovery tasks. When a company needs to build a discovery service, it manually selects the suitable discovery schemes, encapsulates them as services, then invokes them as a composite web service. However, when different discovery tasks/schemes are needed, such composite discovery service needs to be manually reconfigured, and different versions of the discovery service are created and managed. To overcome such problems, we propose to build a dynamic self-configuring discovery service (i.e., DISCO), that takes the required discovery policy from users, then automatically finds the suitable discovery schemes in a context-sensitive manner, and finally arranges them as a collection of executable BPEL processes. This is done by adopting different types of knowledge regarding the services' aspects, discovery schemes, and the adopted software ontologies. Such different knowledge types are captured and managed by the previously proposed JAMEJAM framework. Experimental results show that DISCO successfully managed to reconfigure itself for different discovery policies.

1 INTRODUCTION

One way to develop modern business applications in an agile, and efficient manner is by adopting the service computing paradigm, in which business processes are realized by invoking different internal/external services. Such services are not necessary known to businesses during the construction of the business processes. Hence, businesses use discovery services to help them to find the suitable business services. A discovery service is a web service that implements and executes the service discovery process, which is the process for finding services without a priori knowledge of their existences. The output of a discovery service is a list of atomic and/or composite services that fulfill users' goals.

The service discovery process is not trivial, as it requires solutions for many complex problems such as service semantic description, service identification, service composition, automated SLA verification (a.k.a., service selection), service evaluation, service adaptation and presentation. Hence, the discovery process could be seen as a multi-stage sequential process, where the output of a given stage is the input for the following stage, as shown in Figure 1. The figure shows that the service discovery process starts from the service description task, in which service providers describe their services in a machine-understandable format. Such services' descriptions are later published in different service registries (i.e., external and/or internal) for advertising

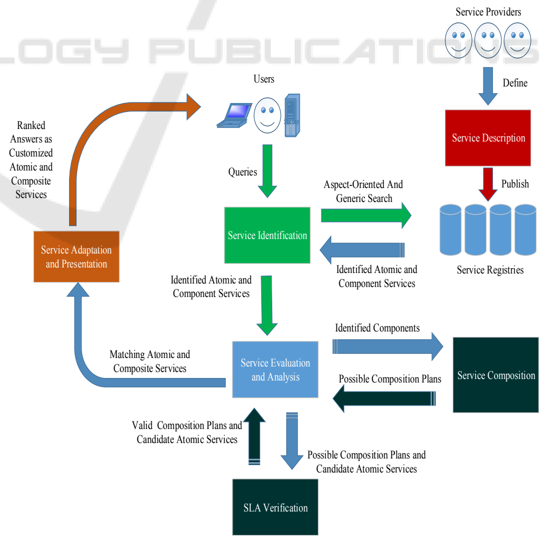

Service Providers
Define
Service Description
Publish
Service Registries
Users
Ranked Answers as Customized Atomic and Composite Services
Queries
Aspect-Oriented And Generic Search
Service Identification
Identified Atomic and Component Services
Service Adaptation and Presentation
Identified Atomic and Component Services
Identified Components
Matching Atomic and Composite Services
Service Evaluation and Analysis
Possible Composition Plans
Service Composition
Valid Composition Plans and Candidate Atomic Services
Possible Composition Plans and Candidate Atomic Services
SLA Verification

Figure 1: The Service Discovery Process.

Elgedawy, I.
DISCO: A Dynamic Self-configuring Discovery Service for Semantic Web Services.
DOI: 10.5220/0006234703350342
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 307-314
ISBN: 978-989-758-243-1

purposes. When users submit their queries to the chosen discovery service, the service identification task starts by examining different service registries to find suitable matching candidates. Once candidate services are identified, they are analysed and evaluated to select the best fitting services. Such service analysis task involves more complex tasks such as service composition and automated SLA verification. Finally, winning service candidates should go through adaptation and customisation step to ensure the satisfaction of users' interfacing and presentation requirements.

The service discovery problem attracted many researchers for a long time (see Section 3). However, due to its complexity, researchers focused their work to address few discovery stages at a time, but there is no holistic approach that addresses all the stages of the discovery process at once. Currently, one approach to overcome such limitation, is to realize the discovery process as a composite web service, where its components are platform-services used to handle the required discovery tasks. Such platform-services are statically chosen at design time. This is done by choosing one of the existing discovery approaches for a given stage, implement it, then encapsulate it as a platform-service. For example, a company could choose an approach for service identification, another one for service composition, and another one for adaptation, then encapsulate these approaches as three platform-services, then compose the discovery service from them. We denote such approach as the **static discovery process**. As we can see, in the static discovery process approach, the composite discovery service is tightly-coupled to the platform-services chosen at design time. That any business requirements' change will lead to the reconfiguration of the created composite discovery service, which could be a time consuming process if it is done manually. Furthermore, this could be limiting to the company users, as users might be interested to try different discovery approaches, or just need to have one or two stages of the discovery process. In such cases, different versions of the composite discovery service will be created, which adds more service management headache. We believe the current static discovery process is limiting and hinders business agility and ignores users and services' diversity.

To overcome such problems, we propose DISCO, a dynamic self-configuring service for semantic web services. DISCO can automatically change the discovery process stages and their realizing platform-services based on the users' queries and contexts. It takes the required discovery policy from users, then automatically finds the suitable discovery schemes (i.e., matching, evaluation, and adaptation approaches) in a context-sensitive manner, and finally arranges them as a collection of executable BPEL processes. This is done by adopting different types of knowledge regarding the services' aspects, discovery schemes, and the adopted software ontologies. Such different knowledge types are captured and managed by the previously proposed JAMEJAM framework (Elgedawy, 2016)). Experimental results show that DISCO can dynamically reconfigure itself according to the given discovery policies. It is important to note that DISCO is not a new discovery scheme, but it is a self-configuring service that realizes the discovery process in context-based manner using "existing" discovery schemes , details are given in Section 4.

The rest of the paper is organized as follows. Section 2 provides an overview over the JAMEJAM framework. Section 3 discusses related work. Section 4 provides the required design assumptions to create DISCO. Section 5 discusses the required types of knowledge and proposes the required meta model for describing discovery schemes. Section 6 discusses the required format for DISCO query and shows how discovery schemes are matched in a context-based manner. Section 7 provides the adopted verification experiments, and finally Section 8 concludes the paper and discusses directions for future work.

2 BACKGROUND: JAMEJAM FRAMEWORK OVERVIEW

JAMEJAM framework depicted in Figure 2 enables users/ companies to manage and incrementally build different types of knowledge regarding the services, the application domains, and the matching schemes required for the discovery process automation. Figure 2 shows that JAMEJAM mainly consists of four main subsystems: the aspects knowledge management subsystem, the services knowledge management subsystem, the matching schemes knowledge management subsystem, and the service discovery subsystem. JAMEJAM subsystems also need a vertical layer of auxiliary services that help them to accomplish their tasks. JAMEJAM subsystems could be summarized as follows: **1) Aspects Knowledge Management Subsystem:** It is the subsystem responsible for managing aspects knowledge, and its corresponding repository. An aspect knowledge is the facts, information, and skills acquired through experience, education, theory and practice regarding such aspect. JAMEJAM captures such knowledge via aspects' ontologies. Every aspect registered with JAMEJAM must have a descriptor that provides some meta-data

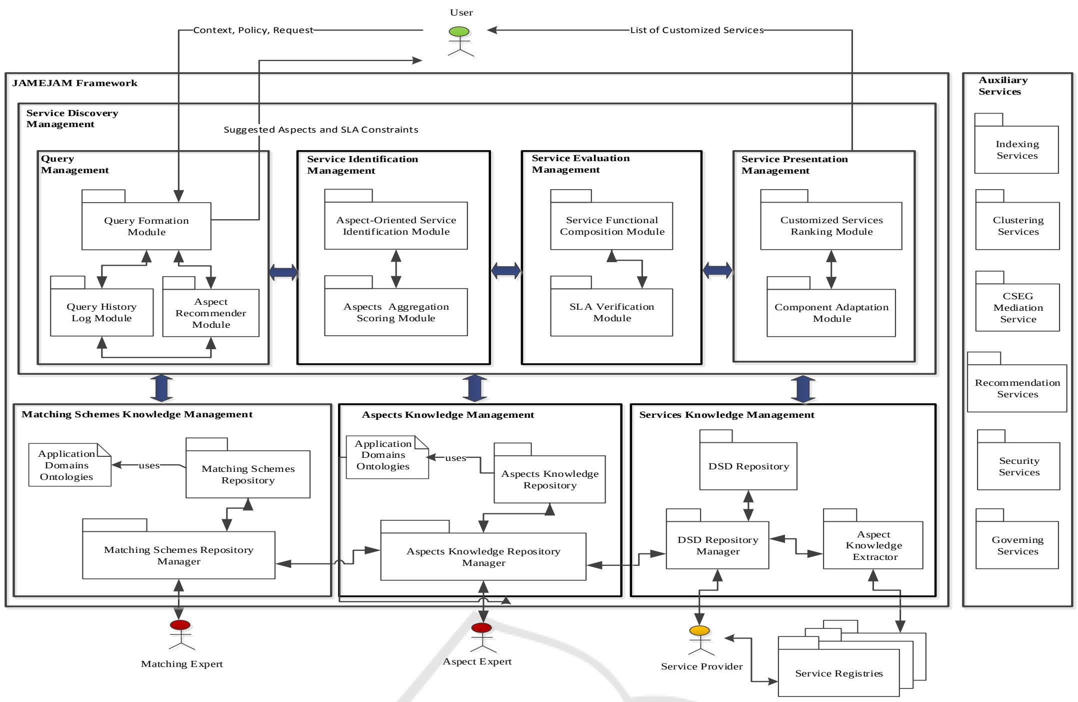

User
Context, Policy, Request
List of Customized Services
JAMEJAM Framework
Service Discovery Management
Suggested Aspects and SLA Constraints
Query Management
Query Formation Module
Query History Log Module
Aspect Recommender Module
Service Identification Management
Aspect-Oriented Service Identification Module
Aspects Aggregation Scoring Module
Service Evaluation Management
Service Functional Composition Module
SLA Verification Module
Service Presentation Management
Customized Services Ranking Module
Component Adaptation Module
Matching Schemes Knowledge Management
Application Domains Ontologies
uses
Matching Schemes Repository
Matching Schemes Repository Manager
Aspects Knowledge Management
Application Domains Ontologies
uses
Aspects Knowledge Repository
Aspects Knowledge Repository Manager
Services Knowledge Management
DSD Repository
DSD Repository Manager
Aspect Knowledge Extractor
Auxiliary Services
Indexing Services
Clustering Services
CSEG Mediation Service
Recommendation Services
Security Services
Governing Services
Matching Expert
Aspect Expert
Service Provider
Service Registries

Figure 2: JAMEJAM Framework (Elgedawy, 2016).

about the aspect such as the aspect name, the adopted aspect ontology (if any), the adopted application domain ontology (if any), and the aspect specification category. An aspect could be described using different ontologies, where every ontology-based description is separately stored in the aspect descriptor. **2) Service Knowledge Management Subsystem:** It is the subsystem responsible for managing services knowledge, and its corresponding repository. Every service created by a service provider should be described according to the adopted software ontology following the adopted aspects ontologies, forming what is known by the service knowledge. Service creators should register their services with JAMEJAM by defining a dynamic service descriptor (DSD) for each service, which contains the aspects values defined according to the adopted aspects ontologies. JAMEJAM enables service creators to enter the DSDs manually or automatically via extractors that retrieve the required knowledge from the service package. **3) Matching Schemes Knowledge Management Subsystem :** It is the subsystem responsible for managing the matching schemes knowledge, and its corresponding repository. JAMEJAM aims to capture different meta-data about the matching schemes in a matching scheme descriptor. A matching scheme is any existing discovery approach that realizes a given discovery task or subtask. For example, we could have matching schemes for service identification, other schemes for behavior matching, other schemes for adaptation. and so on. Such matching schemes should be encapsulated as platform-services. **4) Service Discovery Management Subsystem :** It is the subsystem responsible for managing the service discovery process. Once users defined their adopted software ontology, matching schemes, and its services DSDs, they will be ready for service discovery. Users can define their queries, along with their contexts, goals, and their desired matching policies, which will be used to construct a customized discovery process for every query. Such discovery management subsystem consists of other four subsystems: the query management subsystem, the service identification management subsystem, the service evaluation management subsystem, and service presentation management subsystem. **5) Auxiliary Services:** These are the services that JAMEJAM subsystems use to accomplish their tasks such as indexing, clustering, recommendation, and security services.

3 RELATED WORK

As we indicated before, the web services discovery process consists of four main stages. Hence, our study for the existing works basically focuses on identifying the gaps not covered by existing works. Table 1 provides a comparison between some of the existing approaches (both old and new). The existence of these gaps is what motivates us to create DISCO to have a holistic solution for all the discovery stages.

We organized the works in a chronological order from top to bottom. In the table, we compare between the approaches in terms of their coverage and completeness for the discovery process stages. An approach is considered complete for a given stage when it provides the semantic models and algorithms to address the stage problems. An approach is considered approximate for a given stage when it provides partial models and algorithms to address the stage problems. An approach is considered generic for a given stage when it uses generic models (e.g., free service description) and generic algorithms (e.g., frequency based keyword matching algorithms) to address the stage problems. However, we listed the service evaluation stage as two separate stages (analysis and selection), in order to facilitate the comparison with the existing works that accomplish only the analysis stage. The symbol ✓ means a specialized semantic approach has been proposed, the symbol $\approx$ means a specialized approximate approach has been proposed, while the symbol $\sim$ means a generic approach has been proposed, while a blank entry means function is not supported.

Table 1: A Comparison between some of the Existing Discovery Approaches.

Existing Discovery Approaches	Service Identification	Service Evaluation	Service Selection	Service Adaptation
(Papazoglou et al., 2002)	$\approx$	✓		
(Berardi et al., 2003)	$\approx$	✓		
(Medjahed et al., 2003)	$\approx$	$\sim$	$\approx$	
(Keller et al., 2004)	✓	✓	$\approx$	$\approx$
(Thakkar et al., 2004)	$\approx$	✓	$\approx$	$\approx$
(Benatallah et al., 2005)	$\approx$			
(Kokash et al., 2006)	$\approx$			
(Elgedawy et al., 2008)	✓	✓	$\approx$	
(Brogi et al., 2008)	$\approx$	✓	$\approx$	
(Plebani and Pernici, 2009)	$\approx$			
(Paliwal et al., 2012)	$\approx$			
(Sangers et al., 2013)	$\approx$			
(Elgazzar et al., 2013)	$\sim$			
(Zisman et al., 2013)	$\approx$	✓	$\approx$	
(Kritikos et al., 2014)	$\approx$	✓	$\approx$	
(Bislimovska et al., 2014)	$\approx$	$\sim$		
(Bianchini et al., 2014)	$\approx$	$\approx$		

Table 1 shows none of the existing approaches completely handles all stages. To cover such gap, DISCO is proposed as a holistic solution for the discovery problem, as it will enable users to customize their discovery processes, and choose the required stages to be included in the discovery process, also it decouples users from the headache of choosing the suitable matching, evaluation, and adaptation schemes, as it selects the suitable schemes on the fly without users intervention. This is done with the help of the JAMEJAM framework that provides different types of knowledge needed to have a self-configuring discovery service.

4 DISCO ARCHITECTURE AND ASSUMPTIONS

Reconfigurability of the service discovery process involves two separate issues: First, the choice of the involved discovery stages. Second, the choice of the suitable discovery scheme(s) to realize a given discovery stage. The discovery management system of JAMEJAM handles the second issue, but it was not flexible for the first issue, as the JAMEJAM query must go through all the four discovery subsystems. To overcome such problem, we propose DISCO to become the coordinator and the orchestrator between JAMEJAM discovery sub-modules, as it will interact only with the required sub-modules corresponding to the defined discovery policy. This interactions will be done automatically without the user involvement. Hence, to be able to build DISCO we require the following:

- Each JAMEJAM subsystem should be encapsulated as a platform-service that can be invoked independently.
- Business services and existing discovery approaches must be semantically described and registered with JAMEJAM. Details about description and registration processes are in (Elgedawy, 2016).
- Users' queries should contain their preferred matching/discovery aspects, their preferred discovery policies, and their contexts described as a set of satisfiable constraints. Also the query should contain the required discovery stages, as shown in Section 6.

Figure 3 shows the main components of the DISCO service, which are : the query formulator, the discovery process formulator, and the discovery process executer. **The query formulator** interacts with the user as well as the corresponding JAMEJAM service to be able to form the required DISCO query. **The process formulator** takes the DISCO query as an input, then interact with the JAMEJAM service to get list of candidate discovery schemes, then finally creates BPEL processes corresponding to the identification, evaluation, and adaptation stages, where every stage is realized as a BPEL process. Such BPEL processes' partners are the platform-services corresponding to the discovery schemes retrieved from JAMEJAM's schemes repository. If a suitable discovery scheme cannot be found for a given aspect, DISCO

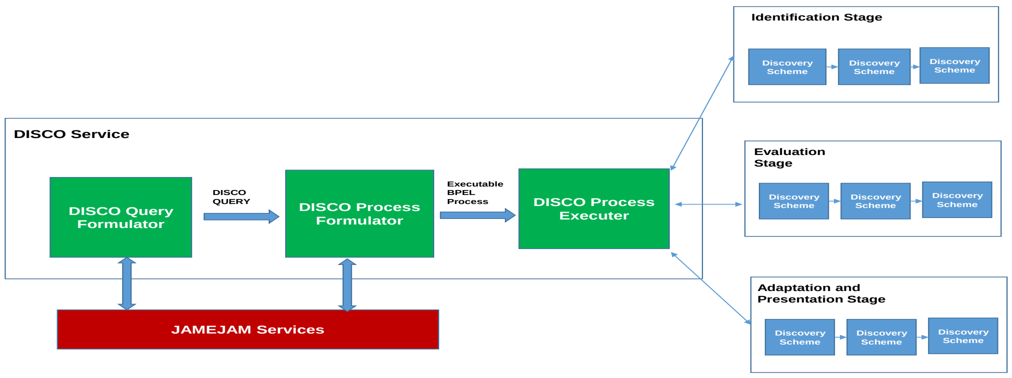


Figure 3: DISCO Architecture.

assumes a generic matching scheme will be used such as keyword-based matching schemes. While, generic evaluation schemes adopt constraints satisfiability schemes for the services' static attributes. Once the BPEL process is constructed, it is given to **the process executor** to invoke the platform-services corresponding to every stage, and finally DISCO passes the results to the users.

5 DISCOVERY KNOWLEDGE MANAGEMENT

DISCO needs different types of knowledge regarding services, and discovery schemes in order to be able to customize the discovery process for users' queries. The JAMEJAM framework captures and manages such knowledge. In what follows, we will summarize the required knowledge types:

- **Discovery Schemes Knowledge Management**: Discovery schemes are introduced to provide the know-how discovery information (i.e., simply the algorithms). To register a discovery scheme with JAMEJAM, discovery experts are required to enter the corresponding discovery scheme descriptor. We define a discovery scheme descriptor as a tuple $\langle SchemeRef, AspectName, StageRef, SWORef, SchemeService, PreConditions, PostConditions \rangle$, where $SchemeRef$ is a unique reference for the matching scheme, $AspectName$ is the name of the involved comparison aspect, $StageRef$ is the reference to the discovery stage the scheme corresponds to, $SWORef$ is the reference to the adopted software ontology, $SchemeService$ is the reference to the web service encapsulating the discovery approach, $PreConditions$ and $PostConditions$ are the sets of required preconditions, and post-conditions, respectively.
- **Service Knowledge Management**: Service knowledge is the collective knowledge regarding the service various aspects. That for every aspect defined in the software ontology, a corresponding value should appear in the service description. Hence, every service registered with JAMEJAM should have a dynamic descriptor known as a DSD (i.e. Dynamic Service Descriptor). DISCO defines the service DSD as set of aspect value descriptor, where an aspect value descriptor is defined as the tuple $\langle AspectName, AspectValue, SWORef \rangle$, where $AspectName$ is the name of the aspect, $AspectValue$ is the aspect value, and $SWORef$ is a reference to the adopted software ontology. Such DSDs should be defined in any machine-understandable format. Hence, for simplicity we propose to use standard XML format to describe the aspects and their attributes' values. However, any semantic web representation language could be used to describe the DSDs (such as RDF, OWL, or WSMO).
- **Software Ontology Management**: Once the company experts defined their preferred aspects and discovery schemes, they can group these definitions into software ontologies. A software ontology is simply a collection of aspects and discovery schemes descriptors defined before. We formally define the software ontology as the tuple $\langle SWORef, AspectsList, DiscoverySchemesList \rangle$, where $SWORef$ is a reference to the software ontology, $AspectsList$ is a list of required aspect descriptors defined as per JAMEJAM model, and $DiscoverySchemesList$ is a list of the required discovery schemes descriptors. Use of software ontologies will make things easier for the users, as they just need to reference the required software ontology in their queries, and DISCO will simply know the involved aspects and discovery scheme

definitions. DISCO does not force all companies to use the same software ontology, however it enables every company to define its own software ontology based on the aspects they see important, as DISCO supports multi-tenancy, where every tenant can have its own services, schemes , and software ontologies.

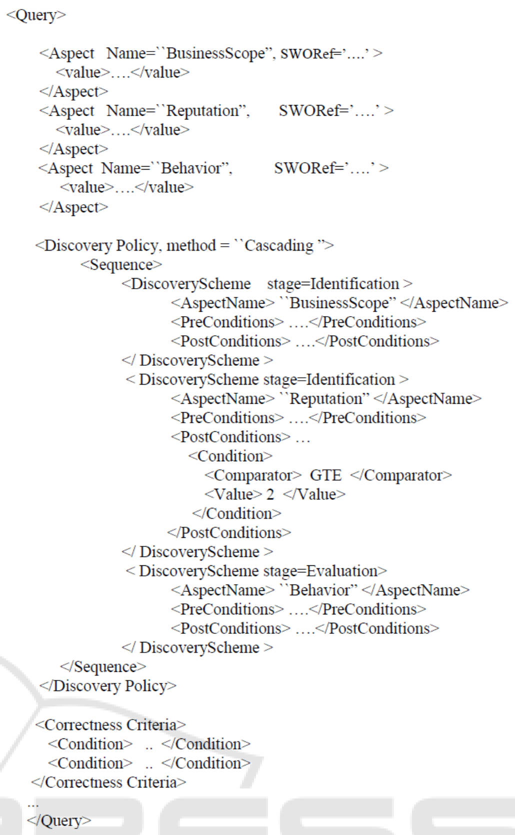
6 DISCO QUERY

DISCO requires an explicit query that should contain the following information:

- The required aspects, their values and the adopted software ontology. Users could also define for every identified aspect, the sources that could be used to generate the aspect value via the identified aspect extractors, as per the JAMEJAM framework.
- The required discovery policy that defines users' preferences and logic regarding the service discovery process. Users can arrange the discovery schemes in any way they find suitable, and can choose only the discovery stages they are interested in.
- User's context and SLA obligations. User's context should be defined in terms of the defined aspects if possible. However, if users could not find the suitable aspects to define their contexts, they can specify extra conditions (known as the correctness criteria) to be satisfied by the matching results, where generic condition matching approaches are used to check their satisfiability. For example, if users need to specify a provider geographical scope aspect in their context, and such aspect is not supported by the adopted software ontology, hence they can define such aspect as a generic condition.

All this information should be defined in a machine-understandable format, as shown in Figure 4, which shows an example for an explicit DISCO query skeleton represented in XML format. The figure shows different aspects to be used in the search process, also it shows the required matching policy and the discovery schemes' preferences (i.e., any meta-data attributes of the discovery schemes' descriptors to be examined). The query contains the targets business scope, external behavior, and reputation aspects. The discovery process for a given query is performed in the following manner:

1. For every aspect appeared in the query, retrieve from the discovery schemes repository all the discovery schemes having the same aspects' appearing in the query for *AspectName* and *SWORef* attributes.
2. For every identified discovery scheme, check if its corresponding preconditions are satisfied based on users' contexts.
3. The obtained answers are checked against the correctness criteria provided by users (if any).



```
<Query>

   <Aspect  Name=``BusinessScope", SWORef='....' >
     <value>....</value>
   </Aspect>
   <Aspect  Name=``Reputation",     SWORef='....' >
     <value>....</value>
   </Aspect>
   <Aspect Name=``Behavior",       SWORef='....' >
      <value>....</value>
   </Aspect>

   <Discovery Policy, method = ``Cascading ">
        <Sequence>
             <DiscoveryScheme   stage=Identification >
                  <AspectName> ``BusinessScope" </AspectName>
                  <PreConditions> ....</PreConditions>
                  <PostConditions> ....</PostConditions>
             </ DiscoveryScheme >
              < DiscoveryScheme stage=Identification >
                  <AspectName> ``Reputation" </AspectName>
                  <PreConditions> ....</PreConditions>
                  <PostConditions> ...
                     <Condition>
                        <Comparator>  GTE  </Comparator>
                        <Value> 2  </Value>
                      </Condition>
                  </PostConditions>
             </ DiscoveryScheme >
              < DiscoveryScheme stage=Evaluation>
                  <AspectName> ``Behavior" </AspectName>
                  <PreConditions> ....</PreConditions>
                  <PostConditions> ....</PostConditions>
             </ DiscoveryScheme >
      </Sequence>
   </Discovery Policy>

   <Correctness Criteria>
     <Condition>  ..  </Condition>
     <Condition>  ..  </Condition>
  </Correctness Criteria>
  ...
  </Query>
```

Figure 4: An Example for a DISCO Query Skeleton.

Based on the stages of the defined discovery policy, DISCO will contact the suitable JAMEJAM services to get the results, then constructs the corresponding BPEL processes, where the flow of the processes is the same as the flow defined in the discovery policy, and the partners are the platform-services of the discovery schemes matching the query aspects.

7 DISCO EVALUATION

This section provides the DISCO verification experiments. Our goal in these experiments to show DISCO adaptability and how it reconfigures itself to different discovery policies. Our goal is not to compare between different discovery policies to come up with the best performing policy, as this will require a completely different set of experiments.

7.1 Discovery Processes Configuration Verification Experiments

In this section, we show how DISCO can reconfigure itself against different discovery policies. This is done by invoking DISCO with different well-known discovery policies (i.e., weighted parallel matching, cascade matching, and generic matching), then plot the precision/recall graph for every policy. Hence, we expect the curve to change from one policy to another, as an evidence for DISCO adaptability. This will be done automatically by submitting different DISCO queries. However, lack of real life data that contains semantic descriptions for services still a big challenge for researchers till today. However, existing datasets such (WS-challenge, OWL-TC, SAWSDL-TC, and WSMO-TC) are quite limited, and there is no ready discovery schemes for them. Hence, researchers opt to use artificial data for their experiments. Such approach has been widely adopted by many works such as the works in (Zisman et al., 2013) (Sangers et al., 2013) (Bislimovska et al., 2014) (Kritikos et al., 2014). Hence, we will follow the same approach and generate the artificial data suitable for our experiments, as our goal is just to show adaptability, and not to compare between policies' accuracy.

We adopt the same steps used to generate artificial data mentioned in (Elgedawy, 2016). However, due to space limitation we will not mention the steps here, and interested readers should refer to the paper for more details. In this dataset, we generate DSD for a number of services (i.e. arbitrary chosen as 10,000 service), such DSDs contain the business scope aspect, the behavior aspect, and the reputation aspect. The corresponding matching approaches are encapsulated as platform-services. To generate the query set. We select a random 100 distinct service DSD from the generated dataset. For each DSD in the query set, we generate a random number of DSD replicas. Such number is chosen from the arbitrary range of (0-50) to ensure having different number of services for each service description. Finally, such generated replicas are added to the dataset and randomly distributed among the DSDs. By doing so, we can automatically identify the correct answer for each query, which is the corresponding service and its replicas. Hence, recall and precision could be automatically computed. Once the dataset and query sets are generated, we generate different DISCO queries that adopt the mentioned aspects, however different discovery policies will be used. For simplicity, we will make the discovery policies target the identification stage only, as if DISCO managed to reconfigure itself for the identification stage, it will be able to reconfigure itself for other stages, as similar steps will be carried out. Hence, the discovery policy is reduced to a matching policy.

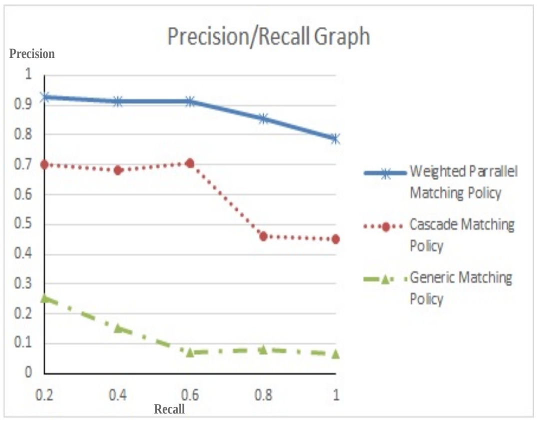


Figure 5: Different Discovery Configurations Comparison With DISCO.

The first used matching policy is the weighted matching policy, in which all aspects' matching schemes are applied independently over the whole DSD repository, and the answers of all schemes are aggregated by computing a total score for every answer. In our experiments, we used equal weights. The second used matching policy is the cascade matching policy, in which aspects are matched in a cascade order starting by the reputation aspect, followed by behavior aspect, followed by the behavior aspect, then we computed the corresponding precision and recall. The last matching policy is the generic one that uses keyword matching approach. We submitted the three queries to DISCO, and computed the precision and recall for every case; results are depicted in Figure 5. The figure shows every matching policy provided a different result, which means DISCO managed to reconfigure itself according to the given matching policies. Also the figure shows the worst performing policy is the generic policy, as it is known of providing the biggest number of false positives (Elgedawy et al., 2008). However, we cannot say the weighted matching policy outperforms the cascading policy in general, as the order of the aspects in the cascade policy affects the final accuracy. Hence, we can only say for the given cascade order, and for the given dataset, the weighted method is the best. However, if we need a more through comparison between policies different experiments are required, which is not in the scope of this paper. However, interested reader could refer to (Elgedawy, 2015) for more details about the effect of the cascading order on the discovery process accuracy.

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed DISCO, a dynamic self-configuring discovery service for semantic web services. DISCO creates for every query a collection of executable BPEL processes, that can identify, evaluate, and adapt the obtained matching results on the fly. Such BPEL processes are realized on the fly by choosing the suitable discovery schemes; adopting different types of knowledge regarding the business services and discovery schemes, which are captured and managed by the JAMEJAM framework. Experimental results show that DISCO could be automatically re-configured according to the provided discovery policies.

The main limitation of DISCO is lack of real-life consolidated discovery knowledge repository. From our experience industry is reluctant to build such knowledge repository. Hence, our future work will focus on filling JAMEJAM with the required knowledge types. Hoping that will make DISCO more practical, and appealing to industry.

REFERENCES

Benatallah, B., Casati, F., Grigori, D., Nezhad, H. R., and Toumani, F. (2005). Developing adapters for web services integration. In *Proceedings of CAiSE, LNCS vol. 3520,*, pages 415–429.

Berardi, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., and Mecella, M. (2003). Automatic composition of e-services that export their behavior. In *Proceedings of the first InternationalConference on Service Oriented Computing (ICSOC)*, pages 43–58, Trento, Italy.

Bianchini, D., Cappiello, C., De Antonellis, V., and Pernici, B. (2014). Service identification in interorganizational process design. *Services Computing, IEEE Transactions on*, 7(2):265–278.

Bislimovska, B., Bozzon, A., Brambilla, M., and Fraternali, P. (2014). Textual and content-based search in repositories of web application models. *ACM Trans. Web*, 8(2):11:1–11:47.

Brogi, A., Corfini, S., and Popescu, R. (2008). Semantics-based composition-oriented discovery of web services. *ACM Trans. Internet Technol.*, 8(4):19:1–19:39.

Elgazzar, K., Hassanein, H. S., and Martin, P. (2013). Daas: Cloud-based mobile web service discovery. *Pervasive and Mobile Computing*.

Elgedawy, I. (2015). USTA: An aspect-oriented knowledge management framework for reusable assets discovery. *The Arabian Journal for Science and Engineering*, 40(2).

Elgedawy, I. (2016). JAMEJAM: A framework for automating the service discovery process. *Journal of Software (JSW)*, 11(7).

Elgedawy, I., Tari, Z., and Thom, J. A. (2008). Correctness-aware high-level functional matching approaches for semantic web services. *ACM Transactions on Web, Special Issue on SOC*, 2(2).

Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., and Fensel, D. (2004). WSMO web service discovery. http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112.

Kokash, N., van den Heuvel, W.-J., and D'Andrea, V. (2006). Leveraging web services discovery with customizable hybrid matching. In *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 522–528. Springer.

Kritikos, K., Plexousakis, D., and Paternò, F. (2014). Task model-driven realization of interactive application functionality through services. *ACM Trans. Interact. Intell. Syst.*, 3(4):25:1–25:31.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. (2003). Composing web services on the semantic web. *Very Large Data Base Journal*, 12(4):333–351.

Paliwal, A. V., Shafiq, B., Vaidya, J., Xiong, H., and Adam, N. (2012). Semantics-based automated service discovery. *IEEE Transactions on Services Computing*, 5(2):260–275.

Papazoglou, M., Aiello, M., Pistore, M., and Yang, J. (2002). Planning for requests against web services. *IEEE Data Engineering Bulletin*, 25(4):41–46.

Plebani, P. and Pernici, B. (2009). Urbe: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642.

Sangers, J., Frasincar, F., Hogenboom, F., and Chepegin, V. (2013). Semantic web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11):4660–4671.

Thakkar, S., Ambite, J., and Knoblock, C. (2004). A data integration approach to automatically composing and optimizing web services. In *Proceedings of the second ICAPS International Workshop on Planning and Scheduling for Web and Grid Services*, British Columbia, Canada.

Zisman, A., Spanoudakis, G., Dooley, J., and Siveroni, I. (2013). Proactive and reactive runtime service discovery: a framework and its evaluation. *IEEE Transactions on Software Engineering*, 39(7):954–974.