Revelation of Consolidated Web Services and Architecture Framework

Vijay Kumar Prodhan ¹, B. Shankar², Bhavani Sankar Panda³
¹, ²Department of Computer Science & Engineering, Centurion University of Technology and Management, Odisha, INDIA
³Assistant Professor, Department of Computer Science & Engineering, Centurion University of Technology and Management, Odisha, INDIA

ABSTRACT

Service architecture is gaining popularity because the composite service presents the features that an individual service cannot present. There are multiple web services available over the web for different tasks. Semantic web is the advance form of the current web, where all the contents have well defined meanings, due to this nature; semantic web enables the automated processing of web contents by machines. At run time, the architecture of these services based on the requester’s functional and non-functional requirements is a difficult task due to the heterogeneous nature of results of the services. In this paper we present a theoretical analysis of service architecture in terms of its dependency with service discovery. Driven by this analysis we define an architecture framework by means of integration with fine-grained I/O service discovery that enables the generation of a graph-based architecture which contains the set of services that are semantically relevant for an input-output request. The proposed framework also includes an optimal architecture search algorithm to extract the best architecture from the graph minimizing the length and the number of services, and different graph optimizations to improve the scalability of the system. A practical implementation used for the empirical analysis is also provided. This analysis proves the scalability and flexibility of our proposal and provides insights on how integrated architecture systems can be designed in order to achieve good performance in real scenarios for the Web.

Keywords— Web Services, Semantic Web Services, Architecture Approaches, Service revelation , Service Architecture Framework; Service Architecture Performance.

I. INTRODUCTION

Web services are well defined, self-described and reusable software components that can be used over the web using the most silent and stable technologies such as Simple Object Access Protocol (SOAP) as a communication framework, Web Service Definition Language (WSDL) and Universal Description, Discovery and Integration (UDDI) that provides a mechanism to clients to find services. A web service is a set of related functions that can be accessed through programming over the web. The key feature of the web services is that they are loosely coupled, allows ad hoc and dynamic binding and are reusable software components. Web services can be divided into three categories and three entities. The categories are publish, find and bind, while the entities are service requester, service provider and the registry. The role of facilitator of service outsourcing is one of the most significant aspect of the web that can reduce the overhead of companies and flourish the business. WSDL is the emerging language for describing the present web service technology and presents the syntactic description of the web services. It only present the structure of the data sent and received through the web, but is unable to present the meaning of the data. This makes the automated web service architecture difficult as architecture, semantic description and execution of web services is necessary for automatic discovery. Existing techniques for web services provides only the syntactic description which as a result, makes it difficult for requester and provider to interpret the meaning of the input and output. Semantic web services are the combination of web services and the semantic web. In the domain of semantic web, Web Ontology Language for Services (OWL-S) and Web Service Modeling Ontology (WSMO) are two prominent techniques used for service architecture. Semantic web services are the extension of the existing web services where the information is represented in a well-defined way. Large amount of data over the web is understandable only by the humans and the custom software. The target goal of semantic web is the medium where the data could be shared easily and processed automatically. The key technology for such concept is the web services. Normal web uses HTML for presenting information, issues, images
and active links which makes it is easy to understand for human beings but difficult for the machines to understand the presented information. But, semantic web which is the advance form of the normal web and refers to the ontology languages, development frameworks and development tools; it uses semantic annotation (web pages with structured data to facilitate the software / intelligent agents to process the data) for describing some of the parts of the web and the meaning of the message of the web services. With the help of annotations semantic web services infer inherent properties to identify services that meet to the requesters demand during the discovery process.

Semantic web services are used for combining data and services from different sources without losing their meaning. Through the discovery and assembly of web services, semantic web services provide the value-added services to complete the domain tasks. Web services can be combined to provide the unified service with some additional extra values. Services discovery and architecture are in general complex tasks that require considerable effort, especially when vast amounts of services are available. Service discovery solutions range from the initial UDDI proposal that relied on the syntactic description of services and a prefixed categorization, to more advanced generic solutions able to discover Web APIs and Web services across domains exploiting rich user-provided semantic service descriptions. Similarly, a plethora of service architecture solutions have been produced spanning from mere graphical support to completely automated solutions. Both discovery and architecture engines essentially rely on the processing of service descriptions, which increasingly go beyond syntactic representations to include the semantics of the service(s) to enable more advanced computations. An analysis of the service architecture literature highlights that, regardless of the approach, a central task that needs to be frequently performed throughout the architecture activity, is the discovery of suitable services to use. Whether one looks at fully automated architecture engines based on Artificial Intelligence (AI) planning techniques, or at more constrained solutions that rely on pre-defined skeletal plans or at graph based approaches focused on semantic input output parameter matching, service discovery is a central activity that needs to be carried out at every main step during the generation of the architecture. Yet, despite the strong dependency between both activities, research and development in both areas has evolved for the most part independently. On the one hand, service discovery has traditionally been approached as a one-of-activity to be sporadically carried out by humans when looking for services. As a consequence the interface exposed by discovery engines assumes that requests are fully specified in terms of a well-defined interface and categorization. Moreover, response times of discovery engines are orders of magnitude above what would be acceptable for a architecture engine that should it delegate the thousands discovery requests it needs to issue at architecture time. These limitations hamper the development of fast architecture systems where discovery and architecture are two fundamental, interrelated activities. On the other hand, partly due to the particularly demanding computational needs of architecture algorithms, most architecture engines re-implement locally their own discovery methods instead of integrating existing components providing state of the art discovery algorithms. Additionally, this approach relies on the unnecessary and often unrealistic assumption that the entire set of services should be locally available to the architecture engine. This assumption requires pre-importing all services locally which is only viable for those registries providing entire public dumps of the service descriptions they hold. Furthermore, most architecture engines do not introduce optimization techniques to improve the scalability by identifying equivalent or dominant functionality that could appear when many different service registries are involved in the architecture. This prevents the use of optimal search strategies since the complexity usually grows exponentially with the number of services.

II. CONSOLIDATED WEB SERVICE ARCHITECTURE APPROACHES

In semantic web services architecture, machines can automatically select, integrate and invoke different web services in-order to achieve the user specified task according to the user constraints. Automatic architecture of web services involves both routine and complex tasks to be performed on the web without user involvement and hence saving time for composing and integration of information. Semantic web service architecture is a widely-studied field since the last decade and different architecture techniques have been identified. Most of the work done on semantic web service architecture approaches can be classified into two categories; Semantic web architecture approaches with QoS support and Semantic web architecture approaches without QoS support. Short descriptions of these approaches are given below.

2.1 Terms and Definitions

In this section we have presented the definitions for common acronyms used in this paper.

2.2 Universal Description Discovery and Integration (UDDI)

UDDI contain description of web services and mechanism for requester to get access to the services publication and description. It contains data with combination of white, green and yellow pages. White pages contain information like company name, address, contact information etc. yellow pages contain information like business category, business type etc. while green pages presents that which kind of services the business offer.
2.3 Simple Object Access Protocol (SOAP)

SOAP is XML based communication protocol to access the web services over the network using transport protocol like HTTP. SOAP is a language and platform independent.

2.4 Web Service Description Language (WSDL)

WSDL specifies the mechanisms to access a Web service over the network. WSDL files are stored in UDDI as a registry so that a web service requester locates them. It is an XML based language which provides syntactic description of web services.

2.5 Web Ontology Language for Services (OWL-S)

OWL-S is a high level language (XML based) used for describing web services properties. It consists of three parts, service profile, process model and grounding. Service profile includes general information and is used to describe what the service will do; process model describes how the service will perform it’s functionally while grounding describe links with industry standards. Its main goal is to enable users to automatically discover, invoke, compose and execute web services under certain conditions.

2.6 Semantic Web Services

Semantic Web services consist of two terms, the semantic web and the web services. Semantic web means adding machine processable semantics to data. With the help of well-defined semantics, machines can understand the information and process it on behalf of human user whereas web services aim at global infrastructure for distributed computation and for integration of various applications and automation of business processes. Semantic web services promises for providing the most suitable service to the user.

2.7 Web Service Architecture

Composing existing Web services to deliver new functionality is a requirement in many business domains. Service architecture extends the notion of service discovery by enabling automatic architecture of services to meet the requirements of a given a high level task description. With semantic markup of services, the information necessary to select and compose services is available via the semantic descriptions of the requirements and capabilities of services. This enables automatic architecture of web services.

III. SEMANTIC WEB SERVICE DESCRIPTION

The convergence of semantic Web with service oriented computing is manifested by Semantic Web Services (SWS) technology. It addresses the major challenge of automated, interoperable and meaningful coordination of Web Services to be carried out by intelligent software agents. In this chapter, we briefly discuss prominent SWS description framework, that are the standard SAWSDL, OWL-S and WSML1. This is complemented by main critics of Semantic Web Services, and selected references to further readings on the subject.

3.1 Issues of Semantic Service Description

Each semantic service description framework can be characterized with respect to
(a) what kind of service semantics are described.
(b) in what language or formalism.
(c) allowing for what kind of reasoning upon the abstract service descriptions?

Further, we distinguish between an abstract Web Service, that is the description of the computational entity of the service, and a concrete service as one of its instances or invocations that provide the actual value to the user. In this sense, abstract service descriptions are considered complete but not necessarily correct: There might be concrete service instances that are models of the capability description of the abstract service but can actually not be delivered by the provider.

3.2 Functional and Non-Functional Service Semantics

In general, the functionality of a service can be described in terms of what it does, and how it actually works. Both aspects of its functional semantics (or capability) are captured by a service profile, respectively, service process model. The profile describes the signature of the service in terms of its input (I) and output (O) parameters, and its preconditions (P) and effects (E) that are supposed to hold before or after executing the service in a given world state, and some additional provenance information such as the service name, its business domain and provider. The process model of atomic or composite services describes how the service works in terms of the interplay between data and control flow based on a common set of workflow or control constructs like sequence, split+join , choice, and others. This general distinction between profile and process model semantics is common to structured Web Service description frameworks, while differences are in the naming and formal representation of what part of service semantics. We can further differentiate between stateless (IO), respectively, state-based (PE) abstract service descriptions representing the set of its instances, that are concrete services providing value to the user. The non-functional service semantics are usually described with respect to a quality of service (QoS) model including delivery constraints, cost model with rules for pricing, repudiation, availability, and privacy policy.

3.3 Structured Representation of Service Semantics

A domain-independent and structured representation of service semantics is offered by upper (top-level) service ontologies and languages such as OWL-S and WSML with formal logic groundings, or SAWSDL which comes, in essence, without any formal semantics. Neither OWL-S nor WSML provide any agreed formal but intuitive, standard workflow-based semantics of the service process model (orchestration and choreography). Alternatively, for abstract service descriptions grounded in...
WSDL, the process model can be intuitively mapped to BPEL orchestrations with certain formal semantics. 

3.4 Monolithic Representation of Service Semantics

The formal specification of service semantics agnostic to any structured service description format can be achieved, for example, by means of a specific set of concept and role axioms in an appropriate logic. Since the service capability is described by means of a single service concept, this representation of service semantics is called monolithic and allows to determine the semantic relations between service descriptions fully within the underlying logical formalism based on concept satisfaction, subsumption and entailment. However, it does not provide any further information on how the service actually works in terms of the process model nor any description of non-functional semantics.

3.5 Data Semantics

The domain-dependent semantics of service profile parameters (also called data semantics) are described in terms of concepts, roles (and rules) taken from shared domain, task, or application ontologies. These ontologies are defined in a formal semantic Web language like OWL, WSML or SWRL. If different ontologies are used, agents are supposed to automatically resolve the structural and semantic heterogeneities for interoperation to facilitate better Web Service discovery and architecture. This process of ontology matching is usually restricted to ontologies specified in the same language, otherwise appropriate inter-ontology mappings have to be provided to the agents.

In subsequent sections, we briefly introduce prominent approaches to both types of service representation. For structured semantic service descriptions, we focus on OWL-S, WSML, and SAWSDL, and omit to discuss alternatives like DSD (DIANE service description format) and SWSL (Semantic Web Service Language).

3.6 Reasoning about Semantic Service Descriptions

The basic idea of formally grounded descriptions of Web Services is to allow agents to better understand the functional and non-functional semantics through appropriate logic-based reasoning. For this purpose, it is commonly assumed that the applied type of logic reasoning complies with the underlying semantic service description framework.

IV. WEB SERVICE ARCHITECTURE PROBLEM

Service architecture aims to help construct composite services that could fulfill a user request, e.g., booking an entire holiday, when no known service can achieve such a request on its own. A core activity for creating service architectures is, indeed, the discovery of relevant services. In this context, relevant services are those that could be invoked and contribute to obtaining an executable architecture that would fulfil the needs set out by the client. We herein formalize the architecture problem in close relationship with discovery as a means to better study and approach the integration of discovery and architecture engines. The formalization of the problem is data-flow centric, focused on the semantic input-output parameter matching of services’ interfaces.

4.1 Semantic Web Service Relevance

The semantic Web service discovery problem consists of locating appropriate services from one or more service registries that are relevant to an input-output request.

V. ARCHITECTURE FRAMEWORK

On the basis of the formal definition of the problem, in this section we present a graph-based framework for automatic semantic Web service architecture. The process is triggered by an architecture request that specifies the user requirements in terms of inputs and the expected outputs. This information is used in the architecture graph generation phase to build a graph with all the relevant services and the semantic relations between their inputs and outputs. In order to find the relevant services, the architecture graph phase is interleaved with the discovery phase. The discovery phase is responsible for retrieving the relevant services given the data available at different stages during the architecture graph generation phase. The relationships between the inputs and outputs of services are computed in the matchmaking phase, where the semantic matching degree between inputs and outputs is computed using a semantic reasoned. The service architecture graph is eventually generated on the basis of the relevant services and the I/O matching information. This graph contains all possible service architectures that satisfy the architecture request, in addition to a few others that, although invokable, do not manage to entirely fulfill the request. The service architecture graph is then optimized applying different techniques to group and reduce the number of services and relations. Next, an optimal search is performed over the graph to find the optimal architecture. This phase is interleaved with a search optimization phase that analyses and reduces the search space.

5.1 Semantic Service Discovery

Semantic service discovery is the process of locating existing Web Services based on the description of their functional and non-functional semantics. Discovery scenarios typically occur when one is trying to reuse an existing piece of functionality (represented as a Web Service) in building new or enhanced business processes. Both service oriented computing and the semantic Web envision intelligent agents to proactively pursue this task on behalf of their clients. Service discovery can be performed in different ways depending on the service description framework, on means of service selection, and
on its coordination through assisted mediation or in a peer-to-peer fashion. In general, any semantic service discovery framework needs to have the following components.

- **Service description**: Formal means to describe the functional and non-functional semantics of Web Services.
- **Service selection**: Reasoning mechanisms for service matching, that is the pairwise comparison of service descriptions in terms of their semantic relevance to the query, and ranking of the results based on partially or totally ordered degrees of matching and preferences.
- **Discovery architecture**: Environmental assumptions on (centralized, decentralized) network topology, service information storage (e.g. distribution of services, ontologies, registries) and location mechanisms, and functionality of agents involved (e.g. service requestor, provider, middle agents). In the following, we survey existing approaches to semantic service selection and discovery architectures.

### 5.2 Classification of Semantic Web Service Matchmakers

Semantic service matching determines whether the semantics of a desired service (or goal) conform to that of an advertised service. This is at the very core of any semantic service discovery framework. Current approaches to semantic service matching can be classified according to:

- what kinds and parts of service semantics are considered for matching, and
- how matching is actually be performed in terms of non-logic-based or logic-based reasoning on given service semantics or a hybrid combination of both, within or partly outside the respective service description framework.

### 5.3 Non-Logic, Logic, and Hybrid Semantic Service Matching

The majority of Semantic Web Service matchmakers performs logic-based semantic service matching. That is, they are keeping with the original idea of the semantic Web to determine semantic relations between resources including services based on logical inferencing on their annotations grounded in description logics (DL) and/or rules. In fact, the set of logic-based Semantic Web Service matchmakers for OWL-S and WSML still outnumbers the complete set of non-logic-based or hybrid semantic matchmakers available for any Semantic Web Service description format. Non-logic-based semantic matchmaker do not perform any logic-based reasoning but compute the degree of semantic matching of given pairs of abstract service descriptions based on, for example, syntactic similarity measurement, structured graph matching, or numeric concept distance computations over given ontologies.

### 5.4 Service Profile and Process Model Matching

Most Semantic Web Service matchmakers perform service profile rather than service process model matching.

---

**Figure 1: Categories of existing Semantic Web Service matchmakers.**

Services profile matching (so called “black-box” service matching) determines the semantic correspondence between services based on the description of their profiles. The profile of a service describes what it actually does in terms of its signature, that is its input and output (IO), as well as preconditions (P) and effects or post conditions (E), and non-functional aspects such as the relevant business category, name, quality, privacy and pricing rules of the service. We classify additional context information for service matching such as the organizational (social or domain) roles, or geographic location of service requesters and providers in their interaction as non-functional.

Service process-oriented matching (so-called “glass-box” service matching) determines the extent to which the desired operational behavior of a given service in terms of its process control and data flow matches with that of another service. Like with service profile matching, we can distinguish between non-logic based, logic-based and hybrid semantic process matching approaches depending on whether automated reasoning on operational semantics specified in some certain logic or process algebraic language (e.g. CCS, π-calculus) is performed, or not. An overview of relevant approaches to process mining for process discovery is given in.

### VI. REFERENCE IMPLEMENTATION

We developed a reference implementation of the integrated graph-based architecture framework that is based on two main components: iServe, a service warehouse with advanced discovery support which provides the service registry and takes care of the matchmaking and service discovery activities, and
ComposIT, which is in charge of the graph-based architecture part. In a nutshell the architecture process is carried out as follows. When a architecture request is sent to the system through the Web UI, ComposIT starts computing the architecture graph with all the relevant services for the request. To this end, all the relevant services are discovered layer by layer using the fine-grained I/O logic-based discovery support provided by the Semantic Discovery Engine of iServe. This engine relies on the Service Manager and the KB Manager to retrieve the relevant services using semantic reasoning capabilities. During the architecture graph generation, ComposIT also makes intensive use of the KB Manager in order to carry out concept level matching and consequently figure out how the inputs and outputs of the services obtained can be connected. Once the architecture graph is generated, ComposIT applies the backward pruning and the interface dominance optimizations to reduce the graph size. These optimizations are applicable using only the information contained in the graph, and thus there is no need to interact with the discovery component. Finally, an optimal search is performed over the graph using a backward A* algorithm that extracts the optimal architecture from the graph. In the next sections we shall cover in more detail the inner workings of iServe and ComposIT respectively.

6.1 iServe

iServe, is a service warehouse whose functionality includes the core service registry anchored on Linked Data principles, semantic reasoning support, advanced discovery functionality, and further analysis components able to assist in automatically locating and generating semantic service descriptions out of Web resources. For the purposes of this work we have essentially exploited the registry and discovery functionality. The service discovery functionality builds on top of the Storage Access Layer, which is in charge of managing the registry’s data that includes Service descriptions, related documents and the corresponding Ontologies. This layer essentially provides a RDF/S and OWL storage and reasoning support, document storage, as well as basic crawling facilities to automatically obtain referenced Ontologies. RDF/S and OWL storage and reasoning support is delegated to dedicated engines which are accessed by means of the SPARQL 1.1 standard. Therefore, the reasoning capabilities depend largely on the actual configuration of the store. Concretely, the discovery infrastructure contacts the Service Manager to list services given basic criteria such as the input and output types provided, and the KB Manager to obtain concepts, properties, and their sub or super concepts. Depending on their implementation Service and KB Managers combine internal indexes with SPARQL queries issued to the triple store by means of Jena. Services are imported to iServe using a range of transformation engines able to import service descriptions in a variety of formalisms including SAWSDL, WSMO-Lite, OWL-S, and Micro WSMO. These plugins generate descriptions expressed in terms of a simple RDF/S model, Minimal Service Model (MSM) [2], which essentially captures the intersection of existing service description formalisms. By means of these transformations iServe provides an homogeneous description for services that were originally annotated using heterogeneous means. Given that, the response time of the overall architecture is highly dependent on the performance of the service discovery and concept matchmaking tasks, we extended iServe with various implementations of the Service and Knowledge Base Managers. We tested different configurations to study their individual performance and the overall impact on architecture response times. In particular, we used the following configurations:

1) SPARQL D/M: pure SPARQL Discovery / Matchmaking where all interactions with the Service and Knowledge Base managers are directly implemented as SPARQL queries. This is the typical approach of discovery engines and was the original implementation of iServe.

2) Index. D/SPARQL+Cache M: I/O service discovery is based on an index. We additionally used herein an intermediate cache at the level of the concept matcher in order to avoid issuing recurrent SPARQL queries.

3) Full Indexed D/M: both service discovery and concept matchmaking relied on local indexes prepopulated at load time (and updated with writes). In this configuration, service discovery and concept matchmaking do not need to issue any SPARQL query to the backed.

6.2 ComposIT

ComposIT, is the semantic Web service architecture engine we rely on. It implements all the different graph-based architecture phases of the framework described in Sec. 4. The semantic service discovery and matchmaking mechanisms, which originally were directly implemented internally, are delegated to iServe by means of integration adapters implemented for the purposes of this work. ComposIT nonetheless uses an internal cache and an index to efficiently recover the information of the generated architecture graph. It is worth to note that the architecture supports the deployment of multiple, distributed iServe instances to provide different endpoints that can be used by ComposIT in the architecture phase by aggregating the results of the registries at the ComposIT API level. Indeed, since the services to contemplate at architecture time are identified by the remote registry and we just use them directly, composing this set of services out of just one API call or several calls in parallel (one per registry) is a trivial change. The overall response time analysis would still remain unchanged, and would have an upper-bound determined by the slowest registry. This also applies to other third-party discovery engines as long as they support fine-grained I/O discovery queries. The integration of these third party registries could be achieved by developing interface adapters (with capabilities to
retrieve input and output relevant services) which could be plugged in to the system, keeping the generation of the architecture graph isolated from the concrete registries used. The generated architecture graph can contain different architectures with the same or different length (number of layers) and with different number of services depending on the services that have been selected to generate the needed data. Among the different combinations that can be obtained, the goal of ComposIT is to find the shortest service architecture with the minimum number of services. For this purpose, ComposIT searches for the optimal architecture by carrying out a heuristic search based on the A* algorithm. This search was implemented using Hipster4j to identify a minimal subset of the services from the graph that satisfy the request (in terms of inputs and outputs). Note that multiple architectures can be extracted from the architecture graph since there may be different services that generate outputs of the same concept.

VII. CONCLUSIONS

In this paper we have presented a theoretical analysis of service architecture in terms of its dependency with service revelation. Driven by this analysis we have defined architecture framework anchored on the integration of service discovery and matchmaking within the architecture process. We have devised a reference implementation of this framework on the basis of two pre-existing separate components, namely iServe and ComposIT. This reference implementation has been used to empirically study the impact of discovery and matchmaking on service architecture, and we have provided three different configurations with varying performance. Our empirical analysis shows that, indeed, typical approaches followed by discovery engines cannot serve as a suitable basis to support efficient service architecture as they lead to prohibitive execution times. We have also shown, though, that with the adequate interface granularity and indexing, discovery engines can support highly efficient architecture akin to that obtained by the fastest architecture engines without having to assume to local availability and in-memory preloading of service registries. This work proves the scalability and flexibility of our proposal and provides insights on how integrated architecture systems can be designed in order to achieve good performance in real scenarios, where service registries and architecture frameworks are likely to be distributed and controlled by diverse organizations.

REFERENCES

[13] Sandro Hawke, I.H., Prud’hommeaux, E., Swick,