OWL-S Grounding Parameters Matching by Means of LLM: Preliminary Investigation

Domenico Redavid[1][a], Eleonora Bernasconi[2][b] and Stefano Ferilli[2][c]
[1]*Economics and Finance Department, University of Bari, Largo Abbazia S. Scolastica, Bari, 70124, Italy*
[2]*Computer Science Department, University of Bari, Via E. Orabona 4, Bari, 70125, Italy*

Keywords: Semantic Web Service, OWL-S Composition, LLM.

Abstract: SOA architecture was created to systematise issues relating to the interoperability of M2M services, focusing on issues such as security and privacy. With the advent of generative AI, there is a different way to perform the operations for which Semantic Web Services were created, in a much simpler way, but losing control over the level of security and privacy. In this paper, we seek to propose a combined vision of the two approaches, identifying how generative AI can be used to solve specific, rather than general, problems. To this end, we attempt to analyse how an LLM could be used by a software agent to align different types of XML parameter data in WSDL descriptions.

1 INTRODUCTION

Semantic Web Services (SWS) represent an evolution of traditional web services, enriched with semantic capabilities that enhance their interoperability, discovery, and automatic integration.

Unlike classic Web services (SOAP, REST), which rely on syntactic descriptions (such as WSDL), SWSs incorporate semantic metadata that enable:

- A deeper offered functionalities understanding
- precise matching between requests and services,
- An automatic composition of multiple services.

Key Components for the SWS concrete implementation are Semantic Web (SW) Ontologies (i.e., formal structures that describe concepts, relationships, and logic in a specific domain), semantic annotation languages (such as OWL for Services (OWL-S) (Martin et al., 2005), Semantic Annotations for WSDL and XML Schema (SAWSDL) (Kopecký et al., 2007), Web Service Modeling Ontology (WSMO) (Fensel et al., 2008)), and SW reasoning engines (Khamparia and Pandey, 2017) for inferring relationships and compatibility between services.

The main advantages that these modes of representation offer can be summarized as:

- Automatic discovery: Services can be found based on their semantics, not just keywords;
- Enhanced interoperability: Shared understanding of domains facilitates integration;
- Dynamic composition: Ability to create complex workflows by combining services automatically;
- Adaptability: Greater resilience to changes in the services ecosystem.

SWS represent an important step toward realizing the vision of the Semantic Web, where machines can understand and use information on the Web. In addition to the application of ML approaches (Ekie et al., 2021), new scenarios and challenges have arisen with the advent of generative AI for automating operations, particularly composition, related to Web services (Pesl et al., 2023; Aiello and Georgievski, 2023). Recent trends aim to simplify discovery and composition operations by abandoning the Service Oriented Architecture towards a simplified definition of services, moving from structured to unstructured descriptions (Pesl et al., 2025).

However, this vision does not take into account the fact that Web services in the narrow sense were created to enable automatic communication between machines and were designed with a number of issues related to security and monitoring of service level agreements in mind. In this paper we ask whether it is possible to use deep learning approaches to simplify practical problems related to SWSs instead of

[a] https://orcid.org/0000-0003-2196-7598
[b] https://orcid.org/0000-0003-3142-3084
[c] https://orcid.org/0000-0003-1118-0601

Redavid, D., Bernasconi, E. and Ferilli, S.
OWL-S Grounding Parameters Matching by Means of LLM: Preliminary Investigation.
DOI: 10.5220/0013854200004000

In *Proceedings of the 17th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2025) - Volume 2: KEOD and KMIS*, pages 192-199
ISBN: 978-989-758-769-6; ISSN: 2184-3228

replacing them. To do this we will use OWL-S as a description language since it is natively related to the web service syntactic description technologies.

2 BACKGROUND

2.1 Web Ontology Language for Services (OWL-S)

Semantic Web Services(McIlraith et al., 2001) provide an ontological framework for describing services, messages, and concepts in a machine-readable format, enabling logical reasoning on service descriptions. The Web Ontology Language for Services (OWL-S) provides a Semantic Web Services framework on which an abstract description of a service can be formalised. It is an upper ontology described with OWL[1] whose root class is *Service*, therefore, every described service maps onto an instance of this concept. The upper level Service class is associated with three other classes:

- **Service Profile.** The service profile specifies the functionality of a service. This concept is the top-level starting point for the customizations of the OWL-S model that supports the retrieval of suitable services based on their semantic description. It describes the service by providing several types of information, in particular: Human readable information, Functionalities, Service parameters, Service categories.
- **Service Model.** The service model exposes to clients how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. In other words, it describes how to ask for (invoke) the service and what happens when the service is carried out.
- **Service Grounding.** A grounding is a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. In general, a grounding indicates a communication protocol, a message format and other service-specific details (e.g., port numbers, the serialization techniques of inputs and outputs, etc.). From the point of view of processes, a service grounding enables the transformation from inputs and outputs of an atomic process into a concrete atomic process grounding constructs.

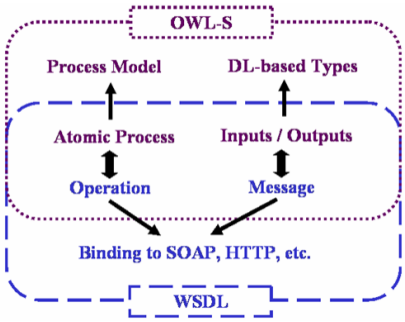

Figure 1: Schema Mapping WSDL-OWL-S.

As we can see from Figure 1, OWL-S grounding map the semantic description of the service with the corresponding Web Services Description Language (WSDL)[2], i.e., it maps directly with the description allowing the concrete invocation of the service.

2.1.1 OWL-S Grounding

The service grounding specifies the details of how to access the service by different kinds of information: protocol and message formats, serialization, transport, and addressing. The role of grounding is mainly to bridge the gap between semantic description of web services and the existing service description models (i.e., syntactic). In general, service grounding maps from the more abstract semantic notions to the concrete elements that are necessary for interacting with the service. The service profile and service model present the abstract side of a service description that doesn't deal with the messages exchanged during service execution. The only part of a message that is abstractly described is the content, through the description of the input and output properties of the Process class in the Service Model ontology. The service grounding ontology is based on and expands these primitive communication parameters. The key role of a service grounding in OWL-S is to realize process inputs and outputs as messages that are sent and received.

OWL-S makes use of the Web Services Description Language (WSDL) to describe a practical grounding mechanism, but other mapping can be used. To describe REST services, Web Application Description Language (WADL) (Filho and Ferreira, 2009) has been proposed.

[1]OWL Web Ontology Language, W3C Recommendation 10 February 2004 - http://www.w3.org/TR/owl-features/

[2]W3C Recommendation, Web Service Describe Language (WSDL) Version 2.0 Part 1: Core Language, 2007, http://www.w3.org/TR/wsdl

The work behind service grounding aims to benefit from the advantages of both WSDL and OWL-S. As described in previous sections, the OWL-S process model is an expressive way of describing the inner workings of a service and OWL's typing mechanisms which are based on XML Schema provide the developer with a set of design advantages. On the other hand, the existing description mechanism of WSDL and message declaration and software support of SOAP have been standardized and used extensively, thus they constitute the best available option for the declaration of message exchanges. In the Figure 1, the overlap between the two languages is illustrated. While WSDL defines abstract types specified using XML Schema in order to characterize the inputs and outputs of services, OWL-S allows for the definition of abstract types as OWL classes, based on description logic. Both languages, however, lack something. On the one hand, WSDL is unable to express the semantics of an OWL class as it is not a semantic language and lacks many required features. On the other hand, OWL-S has no means, as currently defined, to express the binding information that WSDL captures. As a result, both languages are indispensable in a grounding declaration.

Once the high-level association has been established[3], it is possible to specify the concrete association with the WSDL core elements which involves operations, ports, and messages. Specifically, *wsdlOperation* is the URI of the WSDL operation corresponding to the given atomic process. In turn, *wsdlService* is an optional property containing the URI of the WSDL Service that offers the given operation. If we are aware of the port that offers the service and not the service itself, the equivalent *wsdlPort* property can be used. Both *wsdlService* and *wsdlPort* are optional since a *wsdlOperation* property sometimes is enough to uniquely identify a specified operation. However, if multiple ports and/or multiple services offer the specified operation, then the *wsdlPort* and *wsdlService* properties are used to uniquely identify the operation.

2.2 WSDL Structure

WSDL is an XML standard developed by the W3C to describe web services based on protocols such as SOAP (Simple Object Access Protocol). It defines: 1) what a web service does (available operations), 2) how to access it (protocols and message formats), and 3) where to find it (endpoint URL).

[3]The complete OWL-S code is public available at: https://www.daml.org/services/owl-s/1.1/Grounding.owl

A WSDL document is organized into the following six main sections:

1. ⟨*types*⟩: Defines the data types used (using XML Schema/XSD). The types defined by WSDL are data structures, even complex ones, that are used as basic elements to build the input and output messages defined in the "messages" section.
2. ⟨*message*⟩: Describes the messages exchanged (input/output). Messages are the elements that constitute the inputs and outputs of services. Individual messages can contain complex data types defined in the "types" section, or simple primitive data.
3. ⟨*portType*⟩: Defines the service operations (similar to an interface). The operations section defines the operations provided by the web service.
4. ⟨*binding*⟩: Specifies the protocol (SOAP, HTTP GET/POST) and message format. The binding section maps the abstract service, defined in the previous sections, to the concrete communication protocol. The content of this section is highly dependent on the protocol used and therefore on the related WSDL extensions.
5. ⟨*service*⟩: Physical address (endpoint) of the service. The last element of a WSDL file is the service definition: this section allows all operations to be collected under a single name. The service is identified by a name and may have a description.
6. ⟨*documentation*⟩ (optional): For each element described, it is possible to add an element indicating its functionality containing arbitrary human-readable text. It is present at the operation level (i.e., functionality) and at the service level (i.e., overall service).

For each operation, WSDL defines input and output messages. The presence and order of these elements determines the type of service, which can be one of four different types:

- One-way. This is a configuration where the endpoint simply receives the message sent by the client (i.e., there is there is only one input element).
- Request-response. The endpoint receives a request message, performs the necessary processing, and returns a response message to the client.
- Solicit-response. This is the opposite of the previous one. The endpoint initiates communication by sending a message to the client, which in turn must respond accordingly.
- Notification. This is the opposite of the one-way type. The endpoint sends a message to the client

without the client having to send a response. Only the output message is present.

The interaction model to be used depends on the nature of the service.

After providing a broad overview of the structure of a WSDL document, it is important to consider other elements, even if not mandatory, that are useful for describing Web Services. The use of WSDL can also become very complex when connecting to protocols other than SOAP and HTTP. For each protocol and transport layer, it is necessary to define a specific WSDL extension and build WSDL documents according to this grammar. The advantage of WSDL's great extensibility comes at the cost of complexity. Another disadvantage of the WSDL standard is that WSDL only provides a snapshot of the service, thus giving a static view. What is missing is the dynamism of the service, also known as behavior. Through behavior, it is possible to know how a service works and what operations are allowed according to its internal state.

2.3 OWL-S Composition Approach

In this section we briefly report the characteristics of the considered OWL-S composer and some required notions about OWL-S composite services. The work presented in (Redavid et al., 2008) specify how to encode an OWL-S atomic process as a SWRL rule (Horrocks et al., 2005) (i.e., *inCondition* $\wedge$ *Precondition* is the body, *output* $\wedge$ *effect* is the head). After obtaining a set of SWRL rules, the following algorithm was applied: it takes as input a knowledge base containing SWRL rules set and a goal specified as a SWRL atom, and returns every possible path built combining the available SWRL rules in order to achieve such a goal. The set of paths can be considered as a SWRL rules plan representing all possible combinable OWL-S Atomic processes that lead to the intended result (the goal). Subsequently, this connected SWRL rule set is used to produce a composite OWL-S service as described in (Redavid et al., 2011; Redavid et al., 2013). One crucial feature of a composite process is the specification of how its inputs are accepted by particular sub-processes, and how its various outputs are produced by particular sub-processes. Structures to specify the *Data Flow* and the *Variable Bindings* are needed. When defining processes using OWL-S, there are many places where the input to one process component is obtained as one of the outputs of a preceding step, short-circuiting the normal transmission of data from service to client and back. For every different type of *Data Flow* a particular *Variable Bindings* is given. Formally, two complementary conventions to specify *Data Flow* have been identified:**consumer-pull** (the source of a datum is specified at the point where it is used) and **producer-push** (the source of a datum is managed by a pseudo-step called *Produce*). Finally, we remark that a composite process can be considered as an atomic one using the OWL-S Simple process declaration. This allows to treat Composite services during the application of the SWRL Composer.

Two different OWL-S atomic service parameters identified from two different OWL Classes declared equivalent with the axiom *OWL:sameAs* will be treated as if they were the same meaning during the composition process using *Data Flow* and *Variable Bindings* to connect service at Process Model level. But this is not sufficient for the invocation of the concrete WSDL services and therefore for the execution of the obtained composite service.

2.4 Problem Specification

As reported in (Sycara and Vaculin, 2008), there can be three different types of incompatibility that arise in order to ensure interoperability between a (automatic or non-automatic) requester and a service provider.

1. Data level mismatches:
   (a) Syntactic / lexical mismatches: data are represented as different lexical elements (numbers, dates format, local specifics, naming conflicts, etc.).
   (b) Ontology mismatches: the same information is represented as different concepts in the same ontology (subclass, superclass, siblings, no direct relationship) or in different ontologies.
2. Service level mismatches:
   (a) A requester's service call is realized by several providers' services or a sequence of requester's calls is realized by one provider's call.
   (b) Requester's request can be realized in different ways which may or may not be equivalent (e.g., different services can be used to to satisfy requester's requirements).
   (c) Reuse of information: information provided by the requester is used in different place in the provider's process model (similar to message reordering).
   (d) Missing information: some information required by the provider is not provided by the requester.
   (e) Redundant information: information provided by one party is not needed by the other one.

3. Protocol / structural level mismatches: control flow in the requester's process model can be realized in very different ways in the provider's model (e.g., sequence can be realized as an unordered list of steps, etc.).

At the invocation level, it is necessary to resolve issues related to *Data level mismatches* (1.(a)). In the same paper (Sycara and Vaculin, 2008) this is seen as a secondary issue, framing it as natively resolved by the fact that they can be handled directly in the OWL-S grounding description through the definition of transformations between syntactic representation of web service messages and data structures. In particular, it is possible to specify mechanisms (e.g., XSLT transformations) to map various syntactic and lexical representations to the shared semantic representation when the grounding is specified manually. Although this is correct for maintaining the right level of abstraction (i.e., syntactic and semantic) (Redavid et al., 2014a; Redavid et al., 2014b), in order to apply the automatic composition method described in section 2.3, this aspect of grounding must also be handled automatically. In practice, to invoke the execution of a service, the types of the input parameters must be those declared in the WSDL, i.e. if a parameter is declared as type *xsd:string*, it is not possible to invoke the service by passing a parameter of type *xsd:int*.

We therefore question whether it is possible to use deep learning methods to automatically generate a grounding specification that would make directly executable the composite service automatically created using the methods described in the previous sections.

3 PROPOSED SOLUTION

To verify whether the idea of using AI approaches to generate an OWL-S Grounding of the composite OWL-S service, including harmonisation between the input and output parameter types of the various atomic services that are part of the composition, we used the LLM DeepSeek[4]. This allowed us to check the results produced and refine the queries to improve them. In general, the process can be summarised as follows:

1. Application of OWL-S Composer to a set of OWL-S atomic service descriptions that have effective grounding, i.e., WSDL exists (Redavid et al., 2008).
2. Generation of composite OWL-S services (Redavid et al., 2011; Redavid et al., 2013).
3. For each composite OWL-S service, harmonisation of the binding through the insertion of a Transformer capable of changing, where necessary, the data types of the WSDL output of an atomic service into the data types of the WSDL input of the service invoked subsequently, following the process model defined for the OWL-S service in question.

In accordance with the initial vision of SWS (McIlraith et al., 2001), it is a software agent that governs operations related to composition and invocation, for which the agent itself may use LLM or Deep Learning approaches. In the first case, the following approaches may be used:

- Prompt Engineering for XML (Tam et al., 2024): use LLMs with structured prompts and fine-tune LLMs on XML-formatted datasets (e.g., DTD or XSD-guided examples).
- XML as Text with Delimiters: Flatten XML into a string format (e.g., $\langle tag \rangle value \langle /tag \rangle$) and fine-tune LLMs to predict sequences.
- Schema-Guided Generation (Zhang et al., 2023): Use an XML schema (XSD/DTD) to constrain LLM outputs via Constrained decoding (e.g., grammar-based sampling) or Post-processing validation (e.g., XML validators like lxml[5]).

In the case of Deep Learning (Non-LLM), the following other approaches currently exist:

- Sequence-to-Sequence (Seq2Seq) Models (Vinyals et al., 2015; Aharoni and Goldberg, 2017): Train encoder-decoder models (e.g., T5, BART (Setiyarini et al., 2024) to map XML with text/JSON and use attention mechanisms to handle nested tags.
- Graph Neural Networks (GNNs) (Bastings et al., 2017): Model XML as a tree/graph and process with GNNs (e.g., GAT, GraphSAGE).
- Hybrid Tokenization: Custom tokenizers (e.g., SpaCy[6] + XML tags) for embeddings.
- XML-Specific Architectures (Tai et al., 2015): Tree-LSTMs: Process XML trees recursively or Transformer-XH: Extend transformers for hierarchical data.

3.0.1 Experiment

Given that the aim of this paper is to verify the applicability of advanced AI approaches, we con-

[4]DeepSeek V3 - https://www.deepseek.com/en

[5]lxml: Python XML parsing/validation - https://github.com/lxml/lxml

[6]SpaCy: Industrial-Strength Natural Language Processing - https://spacy.io/

Listing 1: BookFinder WSDL.

```
<definitions name="BookFinder"
  targetNamespace="http://ms/owlsmx/wsdl/bookfinder"
  xmlns:tns="http://ms/owlsmx/wsdl/bookfinder"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <service name="BookFinderService">
    <documentation>This is a book search engine.
    </documentation>
    <port name="BookFinderPort"
                binding="tns:BookFinderSoapBinding">
      <soap:address
           location="http://ms/owlsmx/wsdl/bookfinder" />
    </port>
  </service>
  <types>
    <schema
      targetNamespace="http://ms/owlsmx/wsdl/bookfinder"
      xmlns="http://www.w3.org/2001/XMLSchema">
    <complexType name="Request">
      <sequence>
        <element name="RequestInfo" type="xsd:string" />
      </sequence>
    </complexType>
    <complexType name="Book">
      <sequence>
        <element name="Name" type="xsd:string" />
        <element name="ISBN" type="xsd:string" />
      </sequence>
      <attribute name="id" type="xsd:short" />
    </complexType>
   </schema>
  </types>
  <message name="BookFinderInput">
    <part name="body" type="tns:Request" />
  </message>
  <message name="BookFinderOutput">
    <part name="body" type="tns:Book" />
  </message>
...
</definitions>
```

ducted a specific experiment on atomic OWL-S services equipped with WSDL. The WSDLs considered are those listed in 1, 2 and 3[7].

Applying the OWL-S composer, we obtain the following possible combination of services: BookFinderService → BookInfo2 → EBookOrder1

At this point, we tried to ask DeepSeek the following query:

'Given the following WSDL files and supposing that we have a PLAN A: BookFinderService, BookInfo2, EBookOrder1, obtain two new Web services that match WSDL outputs with WSDL inputs of subsequent service where the means of the parameter name match'.

Considering that the mismatch problem concerned the ISBN parameter, which in BookFinderService is of type *xsd:string*, while in BookInfo2 it is of type *xsd:int*, DeepSeek generated the WSDL shown in listing 4, which takes the string-type ISBN as input and returns the corresponding integer as output. This demonstrates that the proposed approach is feasible.

[7]ms means my site domain

Listing 2: BookInfo2 WSDL.

```
<definitions name="BookInfo2"
    targetNamespace="http://ms/owlsmx/wsdl/bookinfo2"
    xmlns:tns="http://ms/owlsmx/wsdl/bookinfo2"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
   <service name="BookInfoService">
     <documentation>This is a book information service
     </documentation>
     <port name="BookInfoPort"
                binding="tns:BookInfoSoapBinding">
       <soap:address
                location="http://ms/owlsmx/wsdl/bookinfo2" />
     </port>
   </service>
   <types>
   <schema targetNamespace="http://ms/owlsmx/wsdl/bookinfo2"
           xmlns="http://www.w3.org/2001/XMLSchema">
     <element name="Request">
       <complexType>
         <sequence>
           <element name="Name" type="xsd:token" />
           <element name="Author" type="xsd:token" />
           <element name="ISBN" type="xsd:int" />
         </sequence>
       </complexType>
     </element>
     <element name="Info">
       <complexType>
         <sequence>
           <element name="Price" type="xsd:float" />
           <element name="NumOfPages" type="xsd:int" />
           <element name="Availability" type="xsd:string" />
         </sequence>
         <attribute use="required" name="ID" type="xsd:int" />
       </complexType>
     </element>
   </schema>
  </types>
  <message name="BookInfoInputMsg">
    <part name="body" element="tns:Request" />
  </message>
  <message name="BookInfoOutputMsg">
    <part name="body" element="tns:Info" />
  </message>
...
</definitions>
```

Listing 3: EBookOrderService WSDL.

```
<definitions name="EBookOrder1"
    targetNamespace="http://ms/owlsmx/wsdl/ebookorder1"
    xmlns:tns="http://ms/owlsmx/wsdl/ebookorder1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <service name="EBookOrderService">
    <documentation>This is a e-book order service
    </documentation>
    <port name="EBookOrderPort"
                binding="tns:EBookOrderSoapBinding">
      <soap:address
        location="http://ms/owlsmx/wsdl/ebookorder1"/>
    </port>
  </service>
  <types>
    <schema
        targetNamespace="http://ms/owlsmx/wsdl/ebookorder1"
            xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="Request">
        <complexType>
          <sequence>
            <element name="Name" type="xsd:token" />
            <element name="Author" type="xsd:token" />
            <element name="ISBN" type="xsd:int" />
          </sequence>
        </complexType>
      </element>
      <element name="Account">
        <complexType>
          <sequence>
            <element name="Username" type="xsd:token" />
            <element name="Password" type="xsd:token" />
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="EBookOrderInputMsg">
    <part name="booktitle" element="tns:Request" />
    <part name="account" element="tns:Account" />
  </message>
  <message name="EBookOrderOutputMsg">
    <part name="body" element="xsd:string" />
  </message>
...
</definitions>
```

Listing 4: BookFinderToBookInfo2 WSDL.

```
<definitions name="BookFinderToBookInfo2"
  targetNamespace="http://ms/mediation/BookFinderToBookInfo2"
  xmlns:tns="http://ms/mediation/BookFinderToBookInfo2"
  xmlns:bf="http://ms/owlsmx/wsdl/bookfinder"
  xmlns:bi="http://ms/owlsmx/wsdl/bookinfo2"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <service name="BookFinderToBookInfo2Service">
    <port name="BookFinderToBookInfo2Port"
          binding="tns:BookFinderToBookInfo2SoapBinding">
      <soap:address
     location="http://ms/mediation/BookFinderToBookInfo2"/>
    </port>
  </service>
  <types>
    <schema targetNamespace=
        "http://ms/mediation/BookFinderToBookInfo2"
          xmlns="http://www.w3.org/2001/XMLSchema">
      <!-- Input matches BookFinder's Book type -->
      <element name="MediationInput">
        <complexType>
          <sequence>
            <element name="Name" type="xsd:string" />
            <element name="ISBN" type="xsd:string" />
          </sequence>
          <attribute name="id" type="xsd:short" />
        </complexType>
      </element>
      <!-- Output matches BookInfo2's Request type -->
      <element name="MediationOutput">
        <complexType>
          <sequence>
            <element name="Name" type="xsd:token" />
            <element name="Author" type="xsd:token" />
            <element name="ISBN" type="xsd:int" />
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="MediationInputMsg">
    <part name="body" element="tns:MediationInput" />
  </message>
  <message name="MediationOutputMsg">
    <part name="body" element="tns:MediationOutput" />
  </message>
  <portType name="BookFinderToBookInfo2PortType">
    <operation name="mediateBookInfoRequest">
      <input message="tns:MediationInputMsg" />
      <output message="tns:MediationOutputMsg" />
    </operation>
  </portType>
...
</definitions>
```

4 CONCLUSIONS

Generative artificial intelligence for automating web service operations has given rise to new scenarios and new challenges. Recent trends aim to simplify discovery and composition operations by abandoning service-oriented architecture in favour of a simplified definition of services, moving from structured descriptions to unstructured descriptions. The composition of SOA web services through the application of methods and techniques for SWS is still important, as there are issues that require the correct identification of responsibilities, especially in critical contexts. For this reason, we sought to investigate the applicability of these new methods to less complex but equally fundamental issues in the field of SWS. This paper represents only a starting point for further investigation of the following aspects: 1) Can a software agent intelligently use LLMs to find the best solution to syntactic matching problems in services? 2) Can deep learning methods also be used in combination with LLM methods for the problem examined? 3) Is it possible to extend the approach to solve problems related to the OWL-S Service Model? We are currently working on implementing the proposed idea by leveraging the ontologies for describing cultural heritage, and specifically digital libraries and archives developed as part of the CHANGES project. Trying to answer these questions is not easy, but it stimulates our interest.

ACKNOWLEDGEMENTS

This research was partially supported by project Cultural Heritage Active innovation for Next-GEn Sustainable society (CHANGES) (PE00000020), Spoke 3 (Digital Libraries, Archives and Philology), under the NRRP MUR program funded by the NextGenerationEU.

REFERENCES

Aharoni, R. and Goldberg, Y. (2017). Towards string-to-tree neural machine translation. In Barzilay, R. and Kan, M.-Y., editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 132–140, Vancouver, Canada. Association for Computational Linguistics.

Aiello, M. and Georgievski, I. (2023). Service composition in the chatgpt era. *Serv. Oriented Comput. Appl.*, 17(4):233–238.

Bastings, J., Titov, I., Aziz, W., Marcheggiani, D., and Sima'an, K. (2017). Graph convolutional encoders for syntax-aware neural machine translation. In Palmer, M., Hwa, R., and Riedel, S., editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, Copenhagen, Denmark. Association for Computational Linguistics.

Ekie, Y. J., Gueye, B., Niang, I., and Ekie, A. M. T. (2021). Web based composition using machine learning approaches: A literature review. In Ahmed, M. B., Boudhir, A. A., and Mazri, T., editors, *NISS2021: The 4th International Conference on Networking, Information Systems & Security, KENITRA, Morocco, April 1 - 2, 2021*, pages 48:1–48:7. ACM.

Fensel, D., Kerrigan, M., and Zaremba, M., editors (2008). *WSMO and WSML*, pages 43–65. Springer Berlin Heidelberg, Berlin, Heidelberg.

Filho, O. F. F. and Ferreira, M. A. G. V. (2009). Semantic Web Services: A RESTful Approach. In *IADIS International Conference WWWInternet 2009*, pages 169–180. IADIS.

Horrocks, I., Patel-Schneider, P. F., Bechhofer, S., and Tsarkov, D. (2005). OWL rules: A proposal and prototype implementation. *J. of Web Semantics*, 3(1):23–40.

Khamparia, A. and Pandey, B. (2017). Comprehensive analysis of semantic web reasoners and tools: a survey. *Education and Information Technologies*, 22(6):3121–3145.

Kopecký, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. (2005). Bringing semantics to web services: The owl-s approach. In Cardoso, J. and Sheth, A., editors, *Semantic Web Services and Web Process Composition*, pages 26–42, Berlin, Heidelberg. Springer Berlin Heidelberg.

McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53.

Pesl, R. D., Mombrey, C., Klein, K., Zyberaj, D., Georgievski, I., Becker, S., Herzwurm, G., and Aiello, M. (2025). Compositio prompto: An architecture to employ large language models in automated service computing. In Gaaloul, W., Sheng, M., Yu, Q., and Yangui, S., editors, *Service-Oriented Computing*, pages 276–286, Singapore. Springer Nature Singapore.

Pesl, R. D., Stötzner, M., Georgievski, I., and Aiello, M. (2023). Uncovering llms for service-composition: Challenges and opportunities. In Monti, F., Plebani, P., Moha, N., Paik, H., Barzen, J., Ramachandran, G. S., Bianchini, D., Tamburri, D. A., and Mecella, M., editors, *Service-Oriented Computing - ICSOC 2023 Workshops - AI-PA, ASOCA, SAPD, SQS, SSCOPE, WESOACS and Satellite Events, Rome, Italy, November 28 - December 1, 2023, Revised Selected Papers*, volume 14518 of *Lecture Notes in Computer Science*, pages 39–48. Springer.

Redavid, D., Ferilli, S., Carolis, B. D., and Esposito, F. (2014a). Guidelines and tool for meaningful OWL-S services annotations. In Filipe, J., Dietz, J. L. G., and Aveiro, D., editors, *KEOD 2014 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Rome, Italy, 21-24 October, 2014*, pages 130–137. SciTePress.

Redavid, D., Ferilli, S., Carolis, B. D., and Esposito, F. (2014b). A tool for complete OWL-S services annotation by means of concept networks. In Fred, A. L. N., Dietz, J. L. G., Aveiro, D., Liu, K., and Filipe, J., editors, *Knowledge Discovery, Knowledge Engineering and Knowledge Management - 6th International Joint Conference, IC3K 2014, Rome, Italy, October 21-24, 2014, Revised Selected Papers*, volume 553 of *Communications in Computer and Information Science*, pages 405–420. Springer.

Redavid, D., Ferilli, S., and Esposito, F. (2011). SWRL Rules Plan Encoding with OWL-S Composite Services. In Kryszkiewicz, M., Rybinski, H., Skowron, A., and Ras, Z. W., editors, *ISMIS*, volume 6804 of *Lecture Notes in Computer Science*, pages 476–482. Springer.

Redavid, D., Ferilli, S., and Esposito, F. (2013). Towards dynamic orchestration of semantic web services. *Trans. Comput. Collect. Intell.*, 10:16–30.

Redavid, D., Iannone, L., Payne, T. R., and Semeraro, G. (2008). OWL-S Atomic Services Composition with SWRL Rules. In An, A., Matwin, S., Ras, Z. W., and Slezak, D., editors, *ISMIS*, volume 4994 of *Lecture Notes in Computer Science*, pages 605–611. Springer.

Setiyarini, D., Adji, T. B., and Hidayah, I. (2024). Evaluating performance of transformer models for dialogue summarization: A comparison of t5-base, t5-small, and bart-base. In *2024 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, pages 184–190.

Sycara, K. and Vaculin, R. (2008). Process mediation of owl-s web services. In et al., T. S. D., editor, *Advances in Web Semantics I*, pages 324 – 345. Springer.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566. The Association for Computer Linguistics.

Tam, Z. R., Wu, C., Tsai, Y., Lin, C., Lee, H., and Chen, Y. (2024). Let me speak freely? A study on the impact of format restrictions on large language model performance. In Dernoncourt, F., Preotiuc-Pietro, D., and Shimorina, A., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - Industry Track, Miami, Florida, USA, November 12-16, 2024*, pages 1218–1236. Association for Computational Linguistics.

Vinyals, O., Kaiser, L. u., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a foreign language. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Zhang, Y., Floratou, A., Cahoon, J., Krishnan, S., Müller, A. C., Banda, D., Psallidas, F., and Patel, J. M. (2023). Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1558–1571.