SPARQL Endpoints as Front-end for Multimedia Processing Algorithms

Ruben Verborgh[1], Davy Van Deursen[1], Jos De Roo[2],
Erik Mannens[1], and Rik Van de Walle[1]

[1]Ghent University – IBBT, ELIS – Multimedia Lab
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
{ruben.verborgh,davy.vandeursen,erik.mannens,rik.vandewalle}@ugent.be
http://multimedialab.elis.ugent.be/
[2]Agfa Healthcare – ACA
Moutstraat 100, B-9000 Ghent, Belgium
jos.deroo@agfa.be

Abstract. Multimedia processing algorithms in various domains often communicate with different proprietary protocols and representation formats, lacking a rigorous description. Furthermore, their capabilities and requirements are usually described by an informal textual description. While sufficient for manual and batch execution, these descriptions lack the expressiveness to enable automated invocation. The discovery of relevant algorithms and automated information exchange between them is virtually impossible. This paper presents a mechanism for accessing algorithms as SPARQL endpoints, which provides a formal protocol and representation format. Additionally, we describe algorithms using OWL-S, enabling automated discovery and information exchange. As a result, these algorithms can be applied autonomously in varying contexts. We illustrate our approach by a use case in which algorithms are employed automatically to solve a complex multimedia annotation problem.

Keywords: multimedia processing, N3Logic, OWL-S, Semantic Web, SPARQL

1 Introduction

In the last decade, the world has witnessed an unprecedented growth of multimedia data production and consumption. In this context, metadata, which are generally defined as 'data about data', play a crucial role. Metadata enable the effective organization, access, and interpretation of multimedia content. Therefore, they play an increasingly important role in bringing order to the growing amount of available multimedia content. However, the lack of availability of many kinds of metadata forms the main obstacle in many multimedia applications. For professionals, metadata generation adds to the production cost because annotating requires tedious manual work. Amateur producers do not possess the necessary skills to provide metadata formally. Clearly, we need automated tools to assist with this cumbersome task.

While automated feature extraction algorithms exist, they are prone to errors and lack an intelligent view on the object under annotation. Currently, people select the appropriate algorithms and parameters for a specific problem and initiate the process. As a result, manual intervention is required when a proposed solution does not meet the requirements. This approach lacks actual cooperation between different algorithms, which are unaware of their own abilities and limitations.

Suppose we have a series of photographs that need to be provided with a textual description automatically. We dispose of the following multimedia processing algorithm implementations:

- *description generation:* creates a description based on image elements;
- *face detection:* detects face regions in an image;
- *face recognition:* recognizes a face in a region;
- *text recognition:* translates bitmap text into a string.

The above list encompasses all required algorithms to handle the task: we should detect faces, recognize them, translate text, and finally generate a description based on the found faces and text. While humans can find and execute the necessary steps for this task, an automated platform cannot, because:

(1) it does not know how to **select algorithms**;
(2) it cannot combine these algorithms into a **solution plan**;
(3) each algorithm has an **informally specified format** for input and output.

Clearly, the first problem arises because the algorithms lack a formal description of their capabilities and requirements. A textual explanation of what the algorithm performs, is insufficient to decide automatically whether it fits a certain purpose. Problems 1 and 2 are closely related, since the creation of a plan also involves a formal description of the desired solution. Based on this description and that of the processing algorithms, an automated planner can devise a solution plan.

The third problem occurs because algorithms usually have a proprietary interaction scheme. More specifically, different algorithms use different ways to specify input and output parameters. Further, these algorithms implement different (standardized or proprietary) multimedia content description schemes to provide their results. This makes it impossible to interact with these algorithms automatically. In addition, the required parameters and the effect of these parameters are often described informally.

In this paper, we address these shortcomings and aim to enable automated algorithm discovery and execution. We propose RDF as input and output representation format. We describe how to transform multimedia processing algorithms into SPARQL endpoints to formalize their interaction. Therefore, we introduce a query prototype suitable for algorithm invocations. We describe algorithm capabilities and requirements using OWL-S, adding support for SPARQL groundings. We zoom in on the expression of various relations between input and output, introducing N3 expressions into OWL-S. Finally, we discuss how to create concrete

algorithm implementations as SPARQL endpoints, providing a number of implementation details. These contributions pave the way for complex multimedia processing scenarios.

2 Algorithm Interaction

2.1 Representation format

The tasks performed by processing algorithms span a very wide range, so finding a comprehensive interaction model poses a challenge. Input and output parameters must be precisely specified, at the same time leaving room for new, previously unused parameters. The algorithm interaction should be:

- **interoperable:** enabling communication with various components, regardless of low-level details such as operating system and programming language;
- **flexible:** handling a variety of inputs and outputs;
- **formal:** able to communicate in a formal way with well-defined semantics so machines are able to interpret the information.

The *Resource Description Framework* (RDF, [13]) is a data format fitting the above requirements well. More specifically, RDF provides a means to represent knowledge in a formal, machine-understandable way [4]. Further, vocabularies and ontologies can be expressed with *RDF Schema* [5] and the *Web Ontology Language* (OWL, [17]), which are both built on top of RDF. We can define or reuse ontologies for the input and output vocabulary of a specific algorithm. For instance, formalized versions of multimedia content description standards (e.g., *COMM* representing a formal way to express *MPEG-7* [1]) constitute one possibility. Integrated semantics facilitate the automated interpretation and exchangeability of results, countering the issues with proprietary formats.

2.2 Communication protocol

An algorithm communication protocol should meet these design requirements:

- **flexible:** able to specify variations on input and outputs;
- **distributed:** provide access to algorithms located on different machines;
- **transparent:** exhibit identical behavior, regardless of varying properties such as physical location and technological differences.

The above requirements hint at a *Service-Oriented Architecture* (SOA, [19]). Therefore, we consider algorithms as Web services. A classic Web service communication protocol choice is the *Simple Object Access Protocol* (SOAP, [11]). However, this protocol is rather verbose and does not account for sufficient flexibility: input and output parameters are passed in a rigid structure that does not allow variations. Given the use of RDF and the definition of algorithms as information-generating entities, our approach is to implement algorithms as *SPARQL* endpoints.

The *SPARQL Protocol and RDF Query Language* [7, 20] is used to retrieve information from semantic data sources, traditionally static databases of RDF content. However, an algorithm is essentially a data source, which does not necessarily offer predefined information, but rather creates new information in a demand-driven way. By using SPARQL as Web service communication protocol, we obtain the following features:

- Web services can be invoked using formally defined input and output parameters;
- the input and output is represented directly in RDF;
- Web services wrapping multimedia algorithms can be seamlessly integrated with other endpoints in the Semantic Web.

There are clear advantages over using SPARQL endpoints instead of passing RDF literals through classic technologies such as SOAP:

- the overhead and verbosity of SOAP is absent;
- the endpoint can check the presence of necessary parameters, as it is able to parse and understand RDF. RDF literals, on the other hand, would be treated as plain text and syntactic or semantic errors would go unnoticed.

Additionally, the cost of maintaining an endpoint is outweighed by the possibility to do post-processing of the returned results directly in SPARQL:

- selection of relevant output values;
- structuring the output values to fit the application format.

In the next subsections, we present a number of querying techniques for on-demand data sources such as multimedia processing algorithms.

Classic SPARQL queries The concept behind SPARQL is similar to that of other query languages: to retrieve a specific view on a larger data collection. The query mechanism validates the data against the constraints in the `WHERE` clause, conditioning the output. This aligns with the way we think about the underlying RDF database: the `WHERE` clause is a *filter* that retains all triples matching the specified template. A first approach to query algorithms would be the exact same way we query data sources.

Suppose we have an algorithm mixing two colors. We begin by defining formal characteristics for the input and output parameters:

- **Input:** a number of `Color` entities with a `hasColorName` property;
- **Output:** a `Color` that has a `hasColorName` property, and a `isMixOf` property with the list of input colors as value.

If we want to ask the result of mixing red and yellow, we could invoke the algorithm using the query in Listing 1. Note that we can choose `SELECT` queries (to retrieve individual values) or `CONSTRUCT` queries (to retrieve an entire RDF graph).

```
PREFIX c: <http://example.org/ontologies/Colors#>
CONSTRUCT { ?color c:hasColorCode ?colorCode. }
WHERE {
  c:Red a c:Color;
         c:hasColorCode "#FF0000".
  c:Yellow a c:Color;
          c:hasColorCode "#FFFF00".
  ?color a c:Color;
        c:hasColorCode ?colorCode;
        c:isMixOf (c:Red c:Yellow).
}
```

Listing 1. Color mixer invocation with classic SPARQL

As opposed to querying data sources, invoking an algorithm is not as such about filtering, but about retrieving the outputs of a process on the inputs. Although it would be possible to build an algorithm this way, there are several drawbacks to this approach:

- **missing indication** that the query is an algorithm invocation;
- **unclear distinction between input and output** in the `WHERE` clause;
- **conceptual mismatch** by forcing the inputs into output conditions.

Clearly, we need a more advanced query which makes the invocation explicit, while strictly adhering to the SPARQL standard.

SPARQL queries with explicit invocation We consider the algorithm as a virtual data source and refer to each invocation by a dedicated `Request` entity. The input and output parameters of the invocation are are values of the `sr:input` and `sr:output` properties. The inputs are generally threated as known values; the outputs are usually unbound variables. Inside the SPARQL query, we can refer to this entity and its associated properties. This enables us to invoke an algorithm while keeping all the benefits of a SPARQL query and its declarativeness.

The query in Listing 2 shows how this technique overcomes previous weaknesses, clearly indicating the invocation, its parameters, and their direction.[1]

The algorithm maintains an entity corresponding to the `Request` in the `WHERE` clause, containing the same information as the entity in the query. The algorithm executes its task on the entity `input` values; its computed output gets bound to the entity `output` values. The resulting graph containing the `Request` entity is subsequently queried in its entirety. We highlight the fact that this `Request` entity is purely virtual: it is only accessible during the execution of the query and remains invisible to other clients accessing the endpoint at the same instant.

[1] The ontology can be found at `http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#`

```
PREFIX c: <http://example.org/ontologies/Colors#>
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
CONSTRUCT { :mixedColor c:hasColorCode ?colorCode. }
WHERE {
  [a sr:Request;
   sr:method "MixColor";
   sr:input [a c:Color;
             c:hasColorCode "#FF0000"],
            [a c:Color;
             c:hasColorCode "#FFFF00"];
   sr:output [a c:Color;
              c:hasColorCode ?colorCode]]
}
```

Listing 2. Algorithm invocation with a `Request` entity

SPARQL queries with named parameters To generalize the query prototype to a broader class of algorithms, it is necessary to provide parameter identification for input and output. This is done by setting the value of the `input` and `output` to a `ParameterBinding` entity, instead of solely the actual value. An example is applying a mask to an image (Listing 3).

For simplicity, the query can be abbreviated by removing parts of the `WHERE` clause that are known in advance. A request will always have type `Request`, and a parameter will always be of type `ParameterBinding`, so these statements can be omitted. The method name is mostly unnecessary, because algorithms usually have a single task that is consequently identified the moment they receives a query. However, if a query is viewed out of its usual context, these items clarify its intended meaning.

SPARQL queries with complex parameters In case the input or output parameters are complex, it is possible to specify either of them as *Notation3* (N3, [2]) strings, which are serialized representations of RDF graphs. They may be required when the structure of the output RDF graph is unknown in advance, making it impossible to reserve sufficient variables for retrieval. Furthermore, complex graphs – such as those containing reified statements – are also better represented by N3 strings. The N3 specification has been extended with this functionality by means of the `log` namespace.

The query in Listing 4, segmenting an image, illustrates complex output. We do not know beforehand how deeply the segments and thus the result graph will be nested. Since SPARQL does not provide facilities to capture *all* descending nodes, we `SELECT` an N3 string to store the result, which must be parsed afterwards to obtain the corresponding RDF graph. For the sake of example, we also supply the input parameter as an N3 string. N3 strings also offer the freedom to return more expressive values. Consider an algorithm that recognizes words

```
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
CONSTRUCT { <source.jpg> :hasMaskedImage ?maskedImage }
WHERE {
  [a sr:Request;
   sr:method "ApplyMask";
   sr:input [a sr:ParameterBinding;
             sr:bindsParameter "source";
             sr:boundTo <source.jpg>],
            [a sr:ParameterBinding;
             sr:bindsParameter "mask";
             sr:boundTo <mask.jpg>];
   sr:output [a sr:ParameterBinding;
              sr:bindsParameter "masked";
              sr:boundTo ?maskedImage]]
}
```

Listing 3. Algorithm invocation with named parameters

in an audio fragment. In the strict case, the return value is ill-defined when uncertainty between two alternatives arises. RDF enables us to express this uncertainty semantically as shown in Listing 5, where is stated that a certain audio fragment contains the word "summer" *or* the word "sombre".

2.3 Algorithm conversion

Having defined a formal model for invocations, we now direct our attention to a conversion method from algorithm to SPARQL endpoint. When designing an algorithm, an author arrives at a point where the output format must be decided. Often, a proprietary format is created, accompanied by an informal description. In this case, it would be convenient to choose RDF as underlying data model since this gives access to an existing formal model, compliant with other information in the Semantic Web. Should the author proceed with another model – proprietary or standardized – then an adapter needs to be written to convert the input and output into RDF.

As such, algorithms communicate entirely using RDF. A SPARQL query engine, surrounding the program, processes the virtual `Request` entity, turning the algorithm into a SPARQL endpoint. This process is visualized in Fig. 1. The RDF inputs are extracted from the query (1) and passed to the algorithm (2), which generates RDF output (3). Input and output form the completed `Request` entity (4). The query is executed on this entity (5), yielding the final result. The latter implies that SPARQL is not only used to simply pass input and output parameters, but can also be used to apply formalized restrictions on input and output parameters and the final result.

```
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
PREFIX log: <http://www.w3.org/2000/10/swap/log#>
SELECT ?segmentsN3
WHERE {
  [a sr:Request;
   sr:method "SegmentImage";
   sr:input [a log:Formula;
             log:n3String "<image.jpg> a :Image."];
   sr:output [a log:Formula;
              log:n3String ?segmentsN3]]
}
```

Listing 4. Algorithm invocation with complex parameters

```
@prefix e: <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>.
({<speech.wav#t=5.38,5.71> :contains "summer".}
 {<speech.wav#t=5.38,5.71> :contains "sombre".}) e:disjunction [a e:T].
```

Listing 5. Expressing uncertainty in algorithm output

3 Capability and Requirements Description

3.1 Description method

Since SPARQL endpoints are in fact Web services, we can describe them as such. A common RDF-compliant specification for semantic Web service descriptions is *OWL-S* [16], which consists of a three-part paradigm:

- **service profile:** a limited description of the service's capabilities and requirements;
- **service model:** service usage modalities and a more in-depth description of its capabilities and requirements, suitable for service composition;
- **service grounding:** technical details regarding communication with the service.

Other possibilities for service descriptions include WSMO [15]. We chose OWL-S because its definition of the process model is more mature [14], but the concepts are transferable to WSMO. In the next subsections, we elucidate these different parts through an example algorithm that recognizes a face in an image region:

- *input:* a region which depicts a face;
- *output:* the recognized face and the depicted person.

This informal description leaves room for interpretation and should be formalized. If the algorithm author already formalized the input and output parameter model in RDF, we can copy this into the service description. However, it will

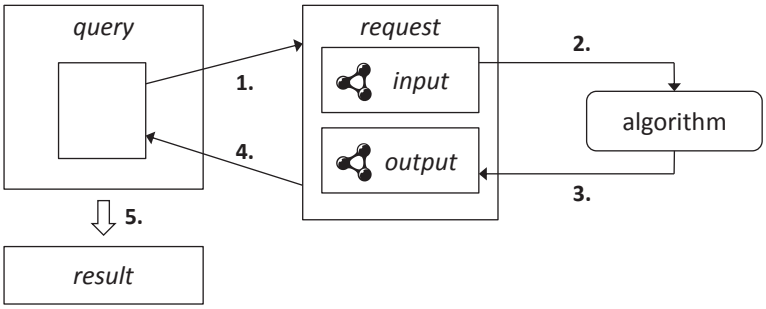

Fig. 1. Algorithm invocation process

```
@prefix Profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>.
:FaceRecognitionProfile a Profile:Profile;
                        Profile:hasInput :RegionInput;
                        Profile:hasOutput :FaceOutput;
                        Profile:hasOutput :PersonOutput;
                        Profile:has_process :FaceRecognitionProcess.
```

Listing 6. Algorithm profile description

often be more convenient to create the formal service description first, deciding which RDF classes to use for input and output, and then use this description to implement the actual RDF parameters of the algorithm. This corresponds to the *design by contract* methodology in software engineering [18].

3.2 Service profile

The *profile* part is primarily meant for human reading or rendering purposes. Therefore, it contains some basic information (useful for human interpretation), reused from our model that we will define later on (Listing 6).

3.3 Service model

The model can be described as a process, containing detailed information about all parameters. We will start with a basic description (Listing 7), extending it as inadequacies come to light.

Although this profile seems correct on the surface, it does not convey all intended semantics for a reliable description of a face recognition activity. Consider an algorithm that, regardless of the input it receives, always returns the exact same predefined person in the `PersonOutput` parameter. This algorithm does not recognize faces, yet it fully complies with the description of the example above. Also, there is no guarantee whatsoever that the region in `RegionInput`

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
:FaceRecognitionProcess a Process:AtomicProcess;
                        Process:hasInput :RegionInput;
                        Process:hasOutput :FaceOutput;
                        Process:hasOutput :PersonOutput.
:RegionInput a Process:Input;
             Process:parameterType "http://example.org/Images#Region".
:FaceOutput a Process:Output;
            Process:parameterType "http://example.org/Faces#Face".
:PersonOutput a Process:Output;
              Process:parameterType "http://example.org/Persons#Person".
```

Listing 7. Basic algorithm process description

actually contains a face; it could depict anything or nothing at all. This means that even an actual face detection algorithm could fail to return a correct result. To obtain a process description that fits the algorithm, we need to correct the following problems:

- the input is not guaranteed to contain a face
  ⇒ *the* ***input constraints*** *must be specified rigorously;*
- the face in the input is not necessarily that of the person in the output
  ⇒ *we require a* ***semantic relation*** *between input and output parameters.*

We can capture these semantics by using preconditions and postconditions.

Preconditions OWL-S supports the use of preconditions to enforce input constraints that go beyond RDF types. These conditions are typically expressed in languages such as *KIF* [9] or *SWRL* [12]. We have opted to use N3 expressions, which are very powerful due to the possibility of more sophisticated built-in functions [3] and the existence of advanced reasoners such as Euler [8].

Therefore, we needed to extend the `Expression` ontology to include support for N3 expressions and conditions, resulting in the `N3Expression` ontology.[2] Our example description is supplemented with preconditions in Listing 8.

Input and output parameters are referred to by *parameter variables* whose name is the last segment of the parameter URI, lowercasing the first letter. As a result, the parameter variable `?regionInput` refers to the parameter named `http://example.org/FaceDetection#RegionInput`. In case of ambiguity, the description document can provide parameter name aliases. This technique is similar to that of SWRL variables in OWL-S. However, a rigorous mechanism that links parameters to variable names should be created. This is left as future work.

Also note the introduction of custom variables (`face`) in the precondition. They are ordinary variables that remain bound in the postconditions, where they can be used together with input and output variables.

[2] `http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#`

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
@prefix Expression:
      <http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#>.
@prefix N3Expression:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#>.
:FaceRecognitionProcess Process:hasPrecondition :RegionPrecondition.
:RegionPrecondition a N3Expression:N3-Expression;
                    Expression:expressionBody
  "?regionInput <http://example.org/Images#regionDepicts> ?face.
   ?face a <http://example.org/Faces#Face>.".
```

Listing 8. Algorithm process description preconditions

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
@prefix Expression:
      <http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#>.
@prefix N3Expression:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#>.
:FaceRecognitionProcess Process:hasResult :FaceRecognitionResult.
:FaceRecognitionResult a Process:Result;
                       Process:hasEffect :FaceRecognitionEffect.
:FaceRecognitionEffect a N3Expression:N3-Expression;
                       Expression:expressionBody
  "?regionInput <http://example.org/Images#regionDepicts> ?faceOutput.
   ?faceOutput <http://example.org/Faces#isFaceOf> ?personOutput.".
```

Listing 9. Algorithm process description postconditions

Postconditions In a similar fashion, relations between input and output parameters can be expressed in the N3 format. OWL-S terminology defines postconditions as effects of a service result. It is possible to specify multiple results that account for different cases, such as normal and erroneous execution. For simplicity, we will limit the face recognition example to a single result and effect. In Listing 9, we express that the region of the input contains the face of the person in the output.

3.4 Service grounding

The remaining part of the algorithm description details its SPARQL endpoint properties. To access a SPARQL endpoint, we need at least the following details:

- the **URL** of the endpoint;
- the **SPARQL versions** supported;
- the supported **query forms** (e.g., `CONSTRUCT`, `SELECT`, `ASK`, or `DESCRIBE`).

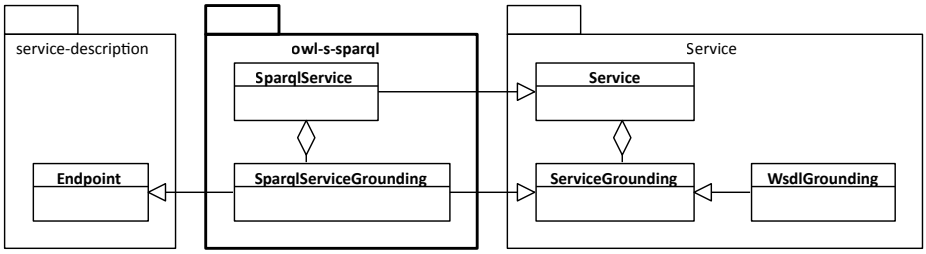

Fig. 2. `owl-s-sparql` ontology overview

```
@prefix sd: <http://www.w3.org/2009/sparql/service-description#>.
@prefix owl-s-sparql:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/owlssparql#>.
:FaceRecognitionGrounding a owl-s-sparql:SparqlServiceGrounding;
  owl-s-sparql:supportsQueryForm owl-s-sparql:SparqlQueryFormConstruct,
                                 owl-s-sparql:SparqlQueryFormSelect;
  owl-s-sparql:supportsSparqlVersion owl-s-sparql:SparqlVersionQuery1_0;
  sd:url <http://example.org/FaceRecognition/sparql>.
```

Listing 10. Algorithm SPARQL grounding description

Unfortunately, OWL-S only provides built-in support for *Web Services Description Language* (WSDL, [6]) service groundings. At the moment of writing, a draft by the W3C SPARQL Working Group on endpoint descriptions exists, but it is not linked to the OWL-S ontologies [23, 24].

An interesting approach is to create a service grounding, compatible with OWL-S, linked to the SPARQL endpoint description. Our `owl-s-sparql` ontology[3] offers this functionality by uniting the definitions of service grounding and SPARQL endpoint. Common grounding properties are provided by OWL-S. SPARQL endpoint specific properties are imported from the service description ontology of the W3C draft. Properties that are currently missing, such as supported SPARQL versions and query forms, are described in `owl-s-sparql`. As depicted in Fig. 2, `SparqlServiceGrounding` is both a `ServiceGrounding` (OWL-S) and an `Endpoint` (service description). A `SparqlService` is a `Service` that has at least one `SparqlServiceGrounding`. Listing 10 shows a possible grounding for the face detection algorithm.

3.5 Service

Finally, the three service parts need to be stitched together in an OWL-S service construct (Listing 11). The user can choose to complement this description with properties that facilitate human interpretation, such as textual descriptions.

[3] `http://ninsuna.elis.ugent.be/ontologies/arseco/owlssparql#`

```
@prefix Service: <http://www.daml.org/services/owl-s/1.1/Service.owl#>.
:FaceRecognitionService a Service:Service;
                        Service:describedBy :FaceRecognitionProcess;
                        Service:presents :FaceRecognitionProfile;
                        Service:supports :FaceRecognitionGrounding.
```

Listing 11. Algorithm service description

4 Implementation

While algorithms can be developed independently, it is convenient to have a software library to abstract common tasks. Two approaches are possible:

- the *wrapper* approach, in which the algorithm is a standalone program, invoked by a configurable wrapper;
- the *toolkit* approach, in which the algorithm directly uses the library for tasks such as RDF parsing and SPARQL querying.

The advantages of the former are that the algorithm does not need to be altered, at the cost of customizing the wrapper sufficiently. More importantly, creating a configuration mechanism that accounts for all possible algorithm interaction schemes is impossible, which is of course the main reason why we introduced SPARQL endpoints. The toolkit approach requires adaptations to the algorithm, improving performance but depending on the existance of an interface between the employed programming language and the toolkit. In general, the wrapper approach – where possible – proves best for existing algorithms and the toolkit approach for algorithms in development, eliminating the need to provide an intermediary communication format.

We implemented toolkits in C++ and C#/.Net, as well as a standalone command line version, to enable interaction with a great variety of programming languages. As an example, we transformed two multimedia processing algorithms into SPARQL endpoints: a face detection and a face recognition algorithm. The face detection algorithm was built from scratch and is based on an implementation of the Viola-Jones face detection algorithm [22]. Hence, in this case, we used the toolkit approach to enable SPARQL communication. For the face recognition algorithm, we used an existing implementation developed by Verstockt et al. [21], which recognizes a face in a well-delineated region. Therefore, we applied the wrapper approach for the face recognition algorithm in order to make the algorithm accessible through SPARQL. Note that the face recognition algorithm adheres to the example description of Section 3. Also note that both algorithms can be deployed for the use case lined out in the introduction of this paper.

Example output for the face detection and recognition algorithms is shown in Listing 12 and Listing 13 respectively. Invoking the face detection algorithm results in one found region depicting a face. The latter is used as input for the face recognition algorithm, which subsequently results in the recognition of the

```
@prefix sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>.
@prefix img: <http://example.org/Images#>.
@prefix face: <http://example.org/Faces#>.
_:regionInput sr:bindsParameter "regionInput";
              sr:boundTo <pict.jpg#xywh=45,121,51,51>.
<pict.jpg#xywh=45,121,51,51> img:regionDepicts [a face:Face>].
```

Listing 12. Face detection algorithm example output

```
@prefix sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix face: <http://example.org/Faces#>.
_:faceOutput sr:bindsParameter "faceOutput";
             sr:boundTo [face:isFaceOf dbpedia:Johnny_Depp].
_:personOutput sr:bindsParameter "personOutput";
               sr:boundTo dbpedia:Johnny_Depp.
```

Listing 13. Face recognition example output

face of the actor Johnny Depp. As one can see, multimedia processing algorithms are not only accessed in a transparent and formalized way, they can also contain pointers to other information available in the Semantic Web. This way, software agents using these kind of endpoints can transparently query both static linked open data sets and multimedia processing algorithms.

5 Conclusions and Future Work

The use of RDF as input and output representation format for algorithms adds expressiveness with well-defined semantics. By approaching algorithms as SPARQL endpoints, we have a standardized protocol to access them, combined with a flexible query language to demand specific information. This way, we can employ algorithms transparently, regardless of low-level system properties. Algorithm queries consist of classic SPARQL, yet indicating the fact that they are invocations. OWL-S enables us to describe the capabilities and requirements of algorithms rigorously, N3 expressions therein can relate input and output parameters in various ways. An additional ontology lets us describe the SPARQL grounding in OWL-S. We illustrated our approach with two example algorithms.

An interesting direction for future research, is the composition of a plan to solve complex solutions using algorithms. We also require a framework which executes the plan and maintains state between invocations. Additionally, we must elaborate some details such as variable identification in OWL-S descriptions.

It is important that we will apply our approach to several larger multimedia use cases. As outlined in the introduction, information-generating tasks such as metadata annotation prove interesting. Applications such as multimedia adaptation could benefit from service-based algorithms, as suggested in [10]. Eventually, we could extend the approach to general problem solving.

Acknowledgments

The research activities as described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO-Flanders), and the European Union.

References

1. Arndt, R., Troncy, R., Staab, S., Hardman, L., Vacura, M.: COMM: Designing a Well-Founded Multimedia Ontology for the Web. In: 6th International Semantic Web Conference (ISWC 2007). Busan, Korea (November 2007)
2. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Recommendation (Jan 2009), http://www.w3.org/TeamSubmission/n3/
3. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34 (2001)
5. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF Schema. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
6. Christensen, E., Curbera, F., Greg, M., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Member Submission (Mar 2001), http://www.w3.org/TR/wsdl
7. Clark, K.G., Feigenbaum, L., Torres, E.: SPARQL protocol for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-protocol/
8. De Roo, J.: Euler proof mechanism, http://eulersharp.sourceforge.net/
9. Genereth, M.R.: Knowledge Interchange Format. Draft Proposed American National Standard, http://logic.stanford.edu/kif/dpans.html
10. Geyter, M.D., Soetens, P.: A planning approach to media adaptation within the Semantic Web. In: Distributed Multimedia Systems. pp. 129–134 (2005)
11. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J.: SOAP version 1.2 part 1: Messaging framework (second edition). W3C Recommendation (Apr 2007), http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
12. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S.: SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3C Member Submission (May 2004), http://www.w3.org/Submission/SWRL/
13. Klyne, G., Carrol, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

14. Lara, R., Roman, D., Polleres, A., Fensel, D.: A conceptual comparison of WSMO and OWL-S. In: ECOWS 2004. LNCS, vol. 3250, pp. 254–269. Springer (2004), http://dx.doi.org/10.1007/978-3-540-30209-4_19
15. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO), howpublished = W3C Member Submission, note = http://www.w3.org/ Submission/WSMO/, day = 3, month = jun, year = 2005
16. Martin, D., Burstein, M., Hobbs, J., Lassila, O.: OWL-S: Semantic markup for web services. W3C Member Submission (Nov 2004), http://www.w3.org/ Submission/OWL-S/
17. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-owl-features-20040210/
18. Meyer, B.: Applying "Design by Contract". IEEE Computer 25(10), 40–51 (1992)
19. Perrey, R., Lycett, M.: Service-Oriented Architecture. IEEE/IPSJ International Symposium on Applications and the Internet Workshops p. 116 (2003)
20. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-query/
21. Verstockt, S., Van Leuven, S., Van de Walle, R., Dermaut, E., Torelle, S., Gevaert, W.: Actor recognition for interactive querying and automatic annotation in digital video. In: IASTED International conference on Internet and Multimedia Systems and Applications, 13th, Proceedings. pp. 149–155. ACTA Press, Honolulu, HI, USA (2009)
22. Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision 57(2), 137–154 (May 2004)
23. Williams, T.G.: SPARQL 1.1 service description. SPARQL Working Draft (Oct 2009), http://www.w3.org/TR/2009/WD-sparql11-service-description-20091022/
24. Williams, T.G., Mikhailov, I.: SPARQL service description. SPARQL Working Group (2009), http://www.w3.org/2009/sparql/wiki/Feature:ServiceDescriptions