Application of Semantic Web Technologies for Automatic Multimedia Annotation

Ruben Verborgh, Davy Van Deursen, Erik Mannens, Chris Poppe, and Rik Van de Walle

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/

Abstract

The generation of metadata, facilitating the retrieval of multimedia items, requires a large amount of manual work. Several processing algorithms that automate parts of this task exist, but they lack a global vision on the object under annotation. In this paper, we investigate how we can apply Semantic Web knowledge to integrate and enhance current processing algorithms in order to answer more advanced metadata queries. We propose a generic problem-solving platform that uses Web services and various knowledge sources to find solutions to complex requests. The platform employs a reasoner-based composition algorithm, generating an execution plan that combines several algorithms as services. It then supervises the execution of this plan, intervening in case of errors or unexpected behavior. We illustrate our approach by a use case in which we annotate the names of people depicted in a photograph.

Keywords: annotation, metadata generation, Semantic Web, service composition, Web services

1 Introduction

1.1 Background

The ever increasing multimedia production rate on the Internet cannot be harnessed unless we have an efficient means of retrieving relevant information. There are many algorithms for searching textual data; searching data types such as image and video however, is more difficult. Metadata annotations [25] facilitate retrieval by describing each item. Unfortunately, metadata generation is a tedious task that involves a significant amount of manual work and knowledge about the annotation domain. For example, a person annotating press photographs needs to recognize depicted people and situations. Algorithms for detecting and recognizing human faces exist, but they are prone to errors and lack an understanding of the photograph in its entirety. Furthermore, none of them are designed to handle composite problems; instead, they are specialized for a specific task.

On the one hand, we can consider these algorithms as services on the World Wide Web. In fact, the Web has evolved from a static document-oriented information source to a dynamic service-oriented platform providing loosely coupled applications [10]. The main focus of Web services is to achieve interoperability between heterogeneous, decentralized, and distributed applications. Furthermore, there is a growing need for composing Web services into more complex services due to increasing user demands and inability of single Web services to achieve a user's goal by itself.

On the other hand, there is the Semantic Web [6] which contains a vast amount of information about diverse domains in extensive knowledge bases such as *DBpedia* [1] and *Freebase* [3]. This formalized knowledge enables advanced reasoning about multimedia item contents, if we connect it to feature extraction algorithms.

1.2 Goal

This paper describes how to apply Semantic Web knowledge and technologies to multimedia annotation. We present a generic semantic problem-solving platform, which combines Web services to achieve a specific task and uses the Semantic Web as knowledge source. The platform is responsible for composing an execution plan that answers a certain request using services. Furthermore, it supervises the execution of this plan, handling the information collection and the interaction between services. When errors occur, it is able to find alternative paths that lead towards an equivalent solution.

1.3 Use case

During this paper, we will demonstrate the introduced concepts by means of an image annotation use case. Take the case of a publisher of a current affairs magazine who has a digital photo archive which needs to be annotated. Apart from the image bitmap data, no additional information is available. As a first step, we would like to identify the people on the photographs, which – given the context of the magazine – will mostly be celebrities. We dispose of the following algorithms (among others):

- a face detection algorithm;
- a face recognition algorithm.

Furthermore, we have access to the following knowledge:

- image, region, and face ontologies and rules;
- Semantic Web knowledge, particularly about celebrities, through DBpedia.

In the next section, we will introduce the platform.

2 Architecture

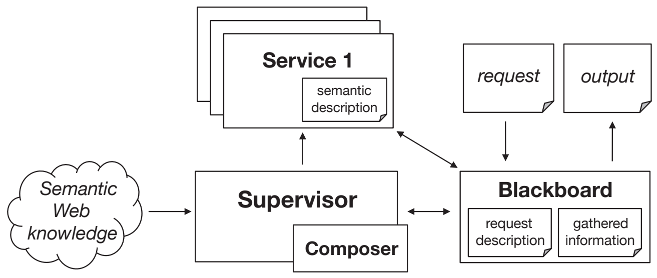


Fig. 1: Blackboard-based architecture of the platform

The architecture of the problem-solving platform, depicted in Fig. 1, implements the blackboard architectural pattern [8] widely used in artificial intelligence applications. It consists of the following components:

- a *blackboard* that contains the currently requested and the gathered information;
- a collection of *services*, accompanied by a description, that perform a variety of specific tasks;
- a *supervisor*, which invokes the services that contribute to the solution of the request.

The supervisor accepts a SPARQL [21] query and the blackboard uses RDF [16] to store supplied and gathered information while retaining all semantics. In our use case, the following query could start the process on the image `Loft.jpg`:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { <Loft.jpg> foaf:depicts ?person. }
WHERE {
  <Loft.jpg> a foaf:Image;
             foaf:depicts ?person.
}
```

Listing 1: SPARQL request for image annotation

The supervisor does not naively try different services, but follows an execution plan created by a *service composer*. Both are assisted by formally described knowledge to relate the services to the request and each other. Note that such knowledge can either be application-specific or knowledge available in the Semantic Web, as detailed in Section 6.
An iterative process progresses towards a solution:

1. the supervisor invokes a service with the current blackboard contents;
2. the service produces a result and sends this to the blackboard;
3. the supervisor supplements the blackboard with derived knowledge, inferred from available knowledge.

For our use case, the supervisor could invoke a face detection algorithm on the image `Loft.jpg`. The algorithm would then return the coordinates of the detected regions, upon which the supervisor could infer that none of these regions overlap.

```
PREFIX sr:
  <http://ninsuna.elis.ugent.be/ontologies/
                                arseco/sparqlrequest#>
PREFIX imreg:
  <http://www.w3.org/2004/02/image-regions>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { <img.jpg> foaf:depicts ?person }
WHERE {
  [a sr:Request;
   sr:input [sr:bindsParameter "region";
             sr:boundTo <img.jpg#xywh=5,7,42,43>];
   sr:output [sr:bindsParameter "person";
              sr:boundTo ?person]]
}
```

Listing 2: Face recognition SPARQL query

3 Services

Our platform requires a flexible interaction model for services, as a great variety of different services needs to be plugged in. It is of utmost importance that the semantics of the concepts of the blackboard are preserved when communicating with a service. Furthermore, we need a formal description of the capabilities and requirements of each service.

We access multimedia algorithms by invoking them as SPARQL endpoints. Benefits include interoperability, flexibility in terms of inputs and outputs, and formal communication with well-defined semantics. For our platform, it is specifically interesting that input can be sent in RDF as part of the `WHERE` clause of the query, and output can be retrieved as RDF by using a `CONSTRUCT` query. An example of a face recognition service query is shown in Listing 2.

The algorithms can be described formally as Web services in OWL-S [18], complemented with formal input and output relationships described in an expression language. These descriptions should not only cover input and output types, but should also determine the effect of the former on the latter. The description of the use case's face recognition service with inputs, outputs, preconditions, and postconditions is shown in Listing 3.

4 Composition

4.1 Formal definitions

When discussing the composer, it is convenient to dispose of a formal definition of a service composition. Firstly, we specify sets that appear in the definitions.

- **The set of parameter names** Π which is the union of all possible input and output parameter names of services. (e.g., *image*, *language*)
- **The set of parameter values** Ω which is the union of all possible input and output values of services. (e.g., `<file.jpg>`, `"en-US"`)
- **The set of variable references** Ψ, containing composer generated identifiers, used as placeholders for unknowns. (e.g., `?image1`, `?language7`)

```
@prefix : <http://example.org/facerecognition#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@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 a Process:AtomicProcess;
                        Process:hasInput :Region;
                        Process:hasOutput :Face, :Person;
                        Process:hasPrecondition :RegionContainsFaceCondition;
                        Process:hasResult [a Process:Result; Process:hasEffect :DepictionEffect].
:Region a Process:Input;❶
        Process:parameterType "http://www.w3.org/2004/02/image-regions#Region"^^xsd:anyURI.
:Face a Process:Output;❷
      Process:parameterType "http://example.org/ontologies/Face.owl#Face"^^xsd:anyURI.
:Person a Process:Output;❸
        Process:parameterType "http://xmlns.com/foaf/0.1/Person"^^xsd:anyURI.
:RegionContainsFaceCondition a N3Expression:N3-Expression;❹
                             Expression:expressionBody
                                 """@prefix imreg: <http://www.w3.org/2004/02/image-regions#>.
                                    @prefix face: <http://example.org/ontologies/Face.owl#Face>.
                                    ?region imreg:regionDepicts [a face:Face].""".
:PersonDepictionEffect a N3Expression:N3-Expression;❺
                       Expression:expressionBody
                           """@prefix imreg: <http://www.w3.org/2004/02/image-regions#>.
                              @prefix face: <http://example.org/ontologies/Face.owl#Face>.
                              ?region imreg:regionDepicts ?face.
                              ?face face:isFaceOf ?person.""".
```

Listing 3: Input and output conditions of the face recognition service in OWL-S

Definition 1. *A **parameter mapping** β is a function $\beta\colon \Pi \to \Omega \cup \Psi$ which assigns parameter names to either a value or a variable reference. The set of all parameter mappings is B. An element (p, v) of B is written as $p \mapsto v$ and called a **parameter assignment** of p to v.*

Definition 2. *A **service invocation** I is a triple $(S, \beta_{in}, \beta_{out})$, written as $\beta_{in} \Lleftarrow \mathbf{S}\ \beta_{out}$, that represents an execution of a service S with input mappings β_{in} and output mappings β_{out}. The domains of β_{in} and β_{out} are the service input and output parameter names, respectively. The parameter value for each parameter name must be an element of the corresponding service parameter domain. The set of all invocations is Φ.*

Definition 3. *An **invocation execution** I is a process step that executes the service S of an invocation $(S, \beta_{in}, \beta_{out})$, passing the actual values of the parameters in accordance with β_{in}. The output values returned by the service are stored in accordance with β_{out}.*

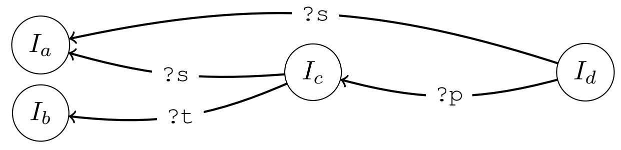
Definition 4. *A **service composition** C is a directed, labeled, acyclic multigraph with*
- *a subset Φ_Δ of the invocation set Φ as vertex set;*
- *a subset Ψ_Δ of the variable reference set Ψ as edge label set.*

An edge with label ψ from a vertex $(S_1, \beta^1_{in}, \beta^1_{out})$ to a vertex $(S_2, \beta^2_{in}, \beta^2_{out})$ is created if and only if $\psi \in \Psi_\Delta \cap \mathcal{R}(\beta^1_{in}) \cap \mathcal{R}(\beta^2_{out})$. That is: if an input value of the first invocation is a variable reference produced by the second invocation as an output value. An edge between two invocations signifies a dependency of the first on the second.

*In a **complete** composition, dependencies are satisfied by values or other invocation outputs: $\forall I_S(\beta_{in}, \beta_{out}) \in \Phi_\Delta : \forall \psi \in \mathcal{R}(\beta_{in}) \cap \Psi_\Delta : \exists I_{S'}(\beta'_{in}, \beta'_{out}) \in \Phi_\Delta : \psi \in \mathcal{R}(\beta'_{out})$. This means that there exists at least one invocation execution order in which all parameter values are known at the start of each execution. A composition is **partial** if it does not satisfy this requirement.*

Example 1. Consider the following complete composition C_0 of calculus service invocations, which computes the value of the calculation $(1 + 2)^{(1+2)\cdot(-1+3)}$.

$$\begin{cases} I_a := \{\mathit{sum} \mapsto \texttt{?s}\} \Lleftarrow \mathbf{Add}\,\{\mathit{termA} \mapsto \texttt{1}, \mathit{termB} \mapsto \texttt{2}\} \\ I_b := \{\mathit{sum} \mapsto \texttt{?t}\} \Lleftarrow \mathbf{Add}\,\{\mathit{termA} \mapsto \texttt{-1}, \mathit{termB} \mapsto \texttt{3}\} \\ I_c := \{\mathit{product} \mapsto \texttt{?p}\} \Lleftarrow \mathbf{Multiply}\,\{\mathit{factorA} \mapsto \texttt{?s}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{factorB} \mapsto \texttt{?t}\} \\ I_d := \{\mathit{result} \mapsto \texttt{?r}\} \Lleftarrow \mathbf{Exp}\,\{\mathit{base} \mapsto \texttt{?s}, \mathit{exp} \mapsto \texttt{?p}\} \end{cases}$$



One possible execution of all invocations of C_0 is:

1. I_a: execute **Add**, using `1` for *termA* and `2` for *termB*, storing the value of *sum (=3)* as `?s`;
2. I_b: execute **Add**, using `-1` for *termA* and `3` for *termB*, storing the value of *sum (=2)* as `?t`;
3. I_c: execute **Multiply**, using `?s` for *factorA* and `?t` for *factorB*, storing the value of *product (=6)* as `?p`;
4. I_c: execute **Exp**, using `?s` for *base* and `?p` for *exp*, storing the value of *result (=729)* as `?r`.

4.2 Service matching

The first obstacle in composition creation is to determine whether two services match. A *start service* S_σ matches an *end service* S_ϵ if an invocation $I_{S_\sigma}(\beta^\sigma_{in}, \beta^\sigma_{out})$ of S_σ exists that enables an invocation $I_{S_\epsilon}(\beta^\epsilon_{in}, \beta^\epsilon_{out})$ of S_ϵ. The first invocation implies fulfillment of both the *input conditions* (necessary to allow the invocation) and the *output conditions* (as a result of the invocation) of S_σ. This signifies that a match is guaranteed when the union of the start service's input and output conditions implies the end service's input conditions.

Listing 3 shows the description of a service that recognizes a face in an image region. It shows the input conditions, consisting of the *input type declarations* ① and the *preconditions* ④. Similarly, the output conditions consist of the *output type declarations* ②③ and the *postconditions* ⑤. The additional conditions ④⑤ are expressed in the Notation3 format (N3, [4]). Input and output parameters are referred to by variables in these expressions. Here, the preconditions state that `Region` should depict the face of a person; the postconditions state that the `Person`'s `Face` is depicted in `Region`. These complex expressions prevent that service matchers and composers only focus on data type matching. For example, there is no point in passing a region of a chart to the recognition algorithm. Therefore, semantic matching is required.

4.3 Inadequacy of point-to-point matching

Service composition comprises more than simple point-to-point matching. Consider the following services:

1. *face detection service* (input: an image, output: the list of detected image regions that contain a face);
2. *face recognition service* (input: a region that contains a face, output: the depicted person's name);

Upon seeing these, we humans know that, in order to annotate people in an image, we need to 1) detect face regions in the image and 2) recognize the faces in each region. That is because we realize that the person names returned by service 2 are connected to the image of service 1, even though service 2 is completely unaware of the existence of such an image. We intuitively construct a *holistic* vision on the problem by combining effects of different services on a concrete problem instance.

Composers that function by matching services point-to-point are unable to transcend the individual service capabilities and, as a consequence, cannot create similar complex compositions. Although they understand the complete functionality of the above services and are even able to match both services, they cannot devise that this composition recognizes faces in an image. This occurs because they do not "remember" the semantics across different junctions, interpreting the result of service 2 as *a* person in *some* region, not *the* person in *that* region of *the* image. This example illustrates that we require a composer with a holistic vision on the problem, understanding that the combination of services embodies more than a simple sum of their individual capabilities.

4.4 Reasoner-based composing process

To create a holistic composition, we employ a goal-driven reasoner on the problem as a whole instead of solely on the junctions. The composing process consists of three steps:

1. each service is **translated** into an N3 Logic rule [5] that simulates its functionality;
2. a reasoner determines whether the request can be **deduced** from the input;
3. the compositions are **reconstructed** from the rules used for deduction.

Obviously, the first action needs to execute only once. The number of deductions found in step 2 indicates how many possible compositions exist. The individual actions are discussed below.

Translation into rules Based on the OWL-S description, an N3 Logic rule is created, simulating the execution of an actual service. Instead of producing actual content, the rule creates placeholders. The conversion process translates input conditions into antecedents and output conditions into consequents. Input parameters become unbound variables; outputs parameters become placeholder variables that will be instantiated with a dummy value upon execution of the rule.

We complemented the rule with tracking information necessary to reconstruct the composition later on, including the service name and the parameters it was invoked with. This was achieved by adding to the consequence of the rule a `boundBy` statement, with the output mapping as subject and the service name and input mapping as object. The parameter assignments of the input and output mapping are formatted as a list of `mappedTo` statements. The automatic translation of the face recognition service is displayed in Listing 4.

Note that some reasoners, such as Eye [9], have an option to display a proof of the deducted knowledge, eliminating the need of tracking. However, such a proof contains a lot of unnecessary details and is more difficult to interpret than our custom tracking statements.

Reasoner deduction Now that we dispose of N3 Logic rules for all services, we need one more rule representing the request. Again, information to track the binding is added, using a `hasBinding` statement. Listing 5 shows the request rule representing the query of the use case.

A backward-chaining reasoner is called with the service rules, request rule, and possibly input statements reflecting the current state of the blackboard. We ask to deduce all possible `boundBy`, `hasBinding` and `mappedTo` statements, which are then stored for composition reconstruction. The reasoner will try to use the request rule, as this is the only way to generate `hasBinding` statements. This requires the fulfillment of the rule's antecedents, each of which can be satisfied either directly by the inputs or by a service rule. In the latter case, the

```
@prefix : <http://example.org/facerecognition#>.
@prefix c: <http://example.org/composer#>.
@prefix imreg:
  <http://www.w3.org/2004/02/image-regions#>.
@prefix face:
  <http://example.org/ontologies/Face.owl#Face>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
{
  ?region a imreg:Region;
          imreg:regionDepicts [a face:Face].
}
=>
{
  ?person a foaf:Person.
  ?face a face:Face;
        face:isFaceOf ?person.
  ?region imreg:regionDepicts ?face.
  ({:Face c:mappedTo ?face.}
   {:Person c:mappedTo ?person.}) c:boundBy
     [a c:Invocation;
      c:ofService :FaceRecognitionService;
      c:withInput ({:Person c:mappedTo ?person.})].
}.
```

Listing 4: Automatic N3 Logic rule translation of the face recognition service description of Listing 3

```
@prefix c: <http://example.org/composer#>.
@prefix var: <http://temp/variables#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
{
  <Loft.jpg> a foaf:Image;
             foaf:depicts ?person.
}
=>
{
  _:solution c:hasBinding
    ({var:Person c:mappedTo ?person.}).
}.
```

Listing 5: N3 Logic rule translation of the use case request

fulfillment of the rule's antecedents is necessary, again by inputs or a rule. The output is built up recursively using this principle.

It is important to notice that the reasoner's knowledge is not limited to the N3 rules deduced from the service descriptions. Indeed, application-specific ontologies and rules, and knowledge available on the Semantic Web, can also be part of the reasoner's knowledge, resulting in advanced capabilities of the rule-based composer. Moreover, we should strive to create knowledge on the highest possible level of abstraction, so that it can be reused across many problem domains.

Composition reconstruction We then transform the generated statements using a three-step process:

1. find all **solution bindings**, indicated by `hasBinding` statements;
2. find all **variable mappings** that are unresolved, they will lead to new invocations;
3. **recursively repeat** step 2 to generate the entire composition graph.

The algorithm produces the correct result, because of the following reasons:

– Each `hasBinding` statement corresponds to exactly one possible composition. The only rule that creates such a statement is the request rule, which can solely be triggered if the solution bindings were successful.
– Each `boundBy` statement uniquely identifies the invocation that executed the binding, because those statements are created only by service rules, which can solely be triggered if their input conditions are satisfied.
– *availableInvocations* will not become empty until the composition is finished: because the composition exists, a path that respects dependencies must exist as well.

5 Supervision

The supervisor is a component responsible for solving a problem using services and an execution plan composer. Its tasks include:

1. *selecting* the appropriate execution plan;
2. *executing* this plan;
3. *recovering* from unanticipated output or errors;
4. *displaying* the solution process progress (optional);
5. *formulating* a response to the request.

Note that we will not consider displaying the progress in this paper. To formulate a response to the request, we have two options. Only the requested output could be returned: the request parameters are bound using the variable binding and returned, all other obtained information is discarded. Alternatively, since often certain intermediary results are of interest as well, the supervisor could also return all the statements available on the blackboard in addition to the response output.

In the next subsections, we elaborate on three tasks of the supervisor: selection, execution, and recovery.

5.1 Composition selection

The supervisor firstly demands the composer to search for complete compositions. If none were found, partial compositions can also be considered. The compositions are then evaluated by criteria such as the following, which should be balanced against each other.

– *Cost:* the expected cost associated with the execution of the services. This cost is at least the sum of the individual execution costs, but it can increase in case of failure. It should be expressed as a mixture of different quantities, such as processor time and amount of money, as external services and employees can be involved.
– *Accuracy:* some services have a higher success rate than others, usually at the expense of a higher cost.
– *Performance:* faster compositions should be allocated to urgent tasks.

- *Availability:* some services are not always available, which can be due to server outage or working schedules if the service task involves people.
- *Completeness:* if the request cannot be solved entirely or if the proposed solution is too expensive, other solutions that only solve part of the problem can be included.

5.2 Composition execution

When the supervisor receives a new request, it initializes the blackboard and adds the input. The composer transforms the blackboard and the request into in a number of possible executions, the best of which is selected. We have to keep track of these additional items:

- the *current information* kept on the blackboard;
- the *variable binding*, a mapping of variable identifiers and values;
- the *current composition*, *current invocation* and *past invocations* with results.

Since a composition consists of an invocation list, its execution – in most basic form – comes down to the execution of these invocations, as detailed in Algorithm 1.

```
Require: $invocations$, $binding$, $blackboard$
  for all $invocation \in invocations$ do
    $service \leftarrow invocation.service$
    $inputs \leftarrow invocation.inputMapping$, replacing variables using $binding$
    $output \leftarrow service.execute(inputs)$
    $blackboard \leftarrow blackboard \cup output.statements$
    $binding \leftarrow binding \cup invocation.outputMapping$, replacing variables using $output$
  end for
  return $blackboard$
```

Algorithm 1: Execution of an invocation list

The variable binding clearly plays a crucial part in the contiguity of the execution and deserves some explanation. Its concept is similar to that of a *single-assignment store* [23] in programming languages, meaning that once a variable is assigned to, its value cannot change. The composition is in fact a *declarative program* whose execution order is solely governed by *data dependencies.* This declarativeness follows naturally from the fact that a composer constructs a plan that indicates *how* to solve a certain problem. In contrast, the supervisor *interprets* the declarative program, determining *what* steps should be performed. We can take advantage of this high level of freedom to exploit parallel or batch execution capabilities. The following definition is analogous to Definition 1 of a parameter mapping.

Definition 5. *A* **variable binding** β_v *is a function* $\beta_v\colon \Pi_v \to \Omega_v$*, which assigns variable names to a simple value (*$\in \Omega \cap \Omega_v$*) or complex value (*$\in \Omega_v$*). The set of all bindings is* B_v*. An element* (n, v) *of* B_v *is called a* **variable assignment***, assigning* v *to* n*.*

5.3 Failure recovery

The composer optimistically assumes correct and successful behavior of all involved services. If we were to withdraw this assumption, the construction of viable compositions would be virtually impossible since every service can be subject to failure. The supervisor therefore handles error recovery, a process consisting of:

1. **failure detection:** catching runtime errors and incomplete service output;
2. **impact determination:** defining the consequences of the failure;
3. **plan adaptation:** changing the plan to reach (possibly adjusted) goals in a different way.

We now examine these different steps thoroughly.

Failure detection We distinguish two kinds of failures: *errors* during service execution and normal execution with *incomplete output.* Since the surrounding programming environment usually detects errors by an exception mechanism, we assume that this task is trivial.

To detect incomplete or empty input, we make use of the invocation's output mapping. If certain parameters of the output mapping do not appear in the output, or if the postconditions specified in the service's description are not met, the output is incomplete and we should initiate the failure recovery process.

For example, the face recognition algorithm could block because of server downtime (error) or could fail to recognize the face (incomplete).

Impact determination Once the *point of failure* is identified, we can determine the failure impact by searching for the invocations that – directly or indirectly – depend on its output. At least one invocation will be affected, since only outputs necessary for future invocations are mapped. The failure repeatedly propagates through these invocations, eventually reaching one or several of the solution generating invocations. The *affected part* of the composition consists of all these invocations, starting at the point of failure. The affected part in Fig. 2 spans the failed invocation and the two rightmost invocations.

The severity of the impact indicates whether the composition should be adapted locally or recreated as a whole. We designate a *resumption point* where normal execution is continued. Fig. 2 shows the same failure with different resumption points. In Fig. 2(a), this is the second invocation from the right; in Fig. 2(b), it is the rightmost invocation. The selection of the resumption point is influenced by the availability of an alternative plan and the history of attempted invocations.

For example, if face detection fails, then face recognition is also affected.

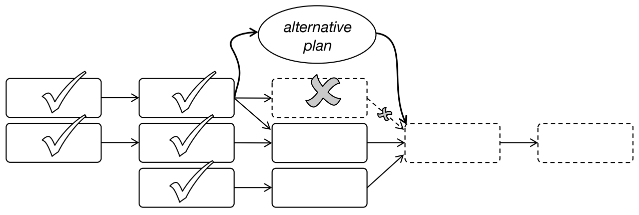
Plan adaptation To recover from failure, the supervisor asks the composer to generate compositions for the affected part of the plan. New compositions start with the current state of the blackboard and end in the resumption point.

Prior to the generation, the supervisor deduces as many additional facts as possible from the blackboard using application-specific knowledge and/or knowledge available in the Semantic Web. The amount of available information is generally larger than that at the time of the initial composition, since the partial execution may have yielded intermediary results. As a result, new compositions that make use of this increased knowledge are possible. This practice can be seen as a forward-chaining reasoning approach that, together with the backward-chaining approach used for composition, constitutes a hybrid mechanism. This brings the advantages of forward-chaining to the execution of compositions, that were created in a goal-driven way.

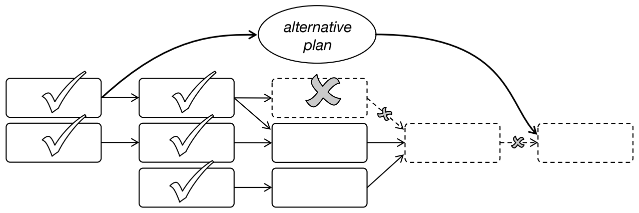
We only consider compositions without previously executed invocations – failed or successful – to avoid infinite failure recovery loops and the overhead of duplicate invocations, whose results are already known.

Fig. 2 shows two different strategies to handle the same failure. Fig. 2(a) depicts an adaptation which tries to correct the failure locally, substituting one service invocation with an alternative plan. Fig. 2(b) uses an entirely new composition instead of the old one, restarting at the inputs while retaining the blackboard contents. Of course, for more complex compositions, various intermediate degrees exist. The supervisor decides what strategy to use, based on the problem parameters, the last failure point, and the possible history of failed recovery attempts.

For example, in case face recognition fails, we could try another algorithm (local) or request human assistance for the entire task (global).



(a) Plan adaptation with local recovery



(b) Plan adaptation with global recovery

Fig. 2: Different plan adaptation strategies. Dotted lines indicate the affected part.

6 Use case

The framework developed so far is a general-purpose semantic problem-solver. The employed knowledge and available services determine the problem domain in which a framework instance operates. This section discusses a metadata generation use case, illustrating the added value of Semantic Web technologies in metadata problem solving.

We return to the image annotation use case introduced in Subsection 1.3. We start by plugging in services relevant to the problem domain. Therefore, we transformed two algorithms into SPARQL endpoints and described them using OWL-S, being:

– an implementation of the Viola-Jones **face detection** algorithm [29], which finds regions in an image that contain a human face;
– an implementation of the **face recognition** algorithm by Verstockt et al. [28], which recognizes a face in a well-delineated region, using a training set.

We add links to relevant ontologies and rulesets describing common facts about images, people and faces. These include both simple and complex facts relating different concepts, such as:

– a person has exactly one face;
– a region belongs to exactly one image;
– regions can depicts faces;
– the depiction of a face of a person implies the depiction of that person;
– . . .

For this use case, we direct our attention to the photograph `Loft.jpg`, shown in Fig. 3. It is a still of the movie *Loft*, depicting the four main actors. Automated face recognition is hampered by lens blur (person 1), occlusion with the actor's hand (person 1), and shadows (person 3). We investigate how our semantic problem solver handles this image and how it overcomes the aforementioned difficulties.

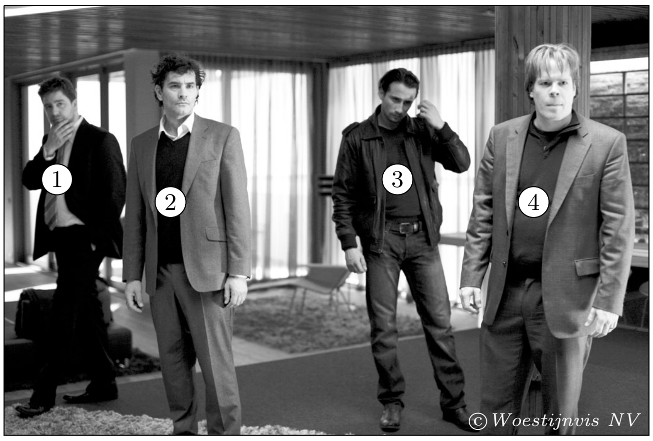


Fig. 3: Movie still depicting four persons

6.1 Execution plan

The user expresses the result as SPARQL (Listing 1) and starts the platform. The supervisor creates a start service from the `WHERE` clause and an end service from the `CONSTRUCT` clause, which are sent to the active composer. Although the composing process seems trivial because of the limited number of services and parameters, Subsection 4.3 has shown the contrary. We should also consider the presence of several other services in addition to the ones mentioned, which hinders a prima facie composition.

First, the composer tries to find a path from the request towards the input using backwards-chaining. In this process, it transforms the request by using the facts in the ontologies and rules, which relate the different concepts. From this, it can deduce that an image with a region containing the person's face is sufficient to meet the request. Our composer is able to deduce that a composition of the face detection and face recognition services (Fig. 4) fulfills the request.

$$\begin{cases} I_\sigma \;:= \{\mathit{param1} \mapsto \texttt{<Loft.jpg>}\} \Lleftarrow \textbf{Input}\{\} \\ I_D \;:= \{\mathit{regionList} \mapsto \texttt{?list1}\} \\ \qquad\quad \Lleftarrow \textbf{FaceDetection}\{\mathit{image} \mapsto \texttt{?image}\} \\ I_R \;:= \{\mathit{face} \mapsto \texttt{?face1},\ \mathit{person} \mapsto \texttt{?person1}\} \\ \qquad\quad \Lleftarrow \textbf{FaceRecognition}\{\mathit{region} \mapsto \texttt{?list1.item}\} \\ I_\epsilon \;:= \{\} \Lleftarrow \textbf{Request}\{\mathit{person} \mapsto \texttt{?person1}\} \end{cases}$$

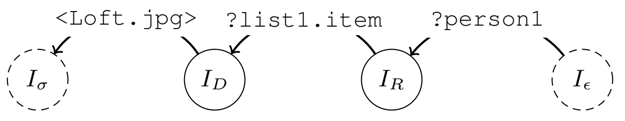


Fig. 4: Composition from image input to person request

Note the appearance of advanced variable bindings, as the face recognition service is executed for *each* of the regions returned by the face recognition algorithm.

6.2 Supervision process

We now step through an actual execution of the request.

Face detection The invocation of the face detection service with parameter value `<Loft.jpg>` succeeds and returns the coordinates of four regions, which are identified by media fragment URIs [27]:

1. `<Loft.jpg#xywh=45,121,51,51>`;
2. `<Loft.jpg#xywh=221,91,56,56>`;
3. `<Loft.jpg#xywh=535,118,43,43>`;
4. `<Loft.jpg#xywh=734,83,69,69>`.

The four resource URIs are assigned to the `?list1` variable, as instructed by the composition. Visual inspection of this output reveals that the detector finds the faces correctly.

Face recognition We now proceed with the face recognition service invocation. The composition demands that this is executed for every item assigned to the `?list1` variable. This results in respectively:

1. *(no output)*;
2. `dbpedia:Koen_De_Bouw`;
3. *(no output)*;
4. `dbpedia:Bruno_Vanden_Broucke`.

The service was able to find two of the four assignments for `?person1`, but the others failed. Looking at Fig. 3, we can understand why: the correctly recognized faces of person 2 and 4 were relatively easy because of their orientation, illumination and contrast. The face of person 1 is harder to recognize because it appears slightly out of focus and the persons's right hand rests on his chin. The left hand of person 3 casts a shadow on his face, decreasing the image contrast locally, interfering with feature extraction. We now show how Semantic Web technologies can help recognizing the two remaining persons.

6.3 Failure recovery

Failure detection and impact determination Two face detection invocations do not return an answer, which the supervisor classifies as a failure. The impacted part spans the face detection invocation and the end service. Peculiarly, this impact is only partial in that half of the needed values are available. Consequently, the adaptation only needs to find alternatives for the two failed invocations.

Blackboard enrichment Prior to the generation of a new plan, the supervisor tries to enrich the blackboard by deriving new semantic knowledge. This enrichment is a combination of semantic inference and a technique known as *sponging*: looking up related information using semantic data sources. This follow-you-nose concept works thanks to the principles of Linked Data [7]. For our use case, the sponging process on the two found actor names on DBpedia reveals facts such as:

– their personal details;
– movies they starred in;
– their co-stars;
– . . .

Plan adaptation The question now is how this additional information can help us in repairing the composition. The relationship between the people in the photograph can assist us. Our knowledge source is aware that the statistical probability to appear in the same photograph is significantly higher with acquaintances compared to random people. Furthermore, it assumes that people know each other if a working relationship exists between them. Also, two actors co-starring in the same movie implies a working relationship.

The supervisor can now employ this knowledge to guide the face recognition service. The latter has an optional `candidates` parameter, by which we can suggest faces for the recognition process. In response, the service can temporarily boost the probability of those faces in its internal training set, enabling a more pronounced recognition result. The supervisor adapts the composition by adding a new invocation, adding the derived acquaintances to the `candidates` parameter. Note that the actual process is slightly more complex, but some details were omitted for brevity.

6.4 Face recognition (bis)

The execution of the second face recognition invocation returns the following values:

1. `dbpedia:Koen_De_Graeve` *OR* `dbpedia:Bruno_Vanden_Broucke`;
3. `dbpedia:Matthias_Schoenaerts`.

The information acquired through sponging has proven useful: person 3 is recognized correctly as `dbpedia:Matthias_Schoenaerts`. However, the algorithm still doubts between two alternatives for person 1 and returns both options, indicating its uncertainty, which can be expressed straightforwardly in RDF. Semantic knowledge comes to the rescue again: it indicates that a single person can only appear once in the same photograph. Since `dbpedia:Bruno_Vanden_Broucke` already appears on `<Loft.jpg>`, the supervisor deduces that person 1 must necessarily be `dbpedia:Koen_De_Graeve`.

All people in the photograph are now identified and the process terminates.

7 Related work

A number of approaches to match formally described Web services exist; most of them are based on OWL-S. For instance, OWL-S Matchmaker [26] and OWL-S MX [15] use both the input and output parameters of OWL-S service descriptions to find proper matches. In [17], a more advanced matching method is presented based on description logic reasoning. Junghans *et al.* propose a formal model for Web services and requests in [14], where service matching is enhanced by using preconditions and effects described in first order logic rules, which is similar to our approach.

Next to service matching approaches, there also exist a number of algorithms for composing formally described services. For instance, in [19], an ontology-based framework is proposed for the automatic composition of Web services. Dynamic compositions based on OWL-S service descriptions using an HTN planning algorithm are presented in [13]. Shin *et al.* [24] describe a method which uses path finding from an initial state to a desired state. However, they only apply this intelligence on a point-to-point basis, so that propagations of the effects – and thus holistic composition – are impossible. They also determine the usefulness of a certain composition using precision and recall. This does not apply to our method, since a reasoner will only return compositions that are logically sound (completeness requirement of a composition) and thus satisfy the initial request. Redavid *et al.* [22] have suggested the use of an SWRL [12] reasoner for composing services and follow a similar approach as our work: composition using translated service rules. However, our approach has a number of advantages over theirs:

- Their approach is limited to parameters which are characterized by an OWL class, which is a limitation in terms of expressivity. Our approach does not have such restrictions: all kinds of relationships between parameters are expressible, even if the parameter has a primitive datatype.
- Our approach does not suffer from unbound variables in the generated rules.
- Although N3Logic and SWRL are both able to represent rules, N3Logic has a number of advantages compared to SWRL: more builtins are supported, an efficient reasoner (i.e., Eye) is available, and last but not least, N3Logic integrates with existing RDF knowledge in a very natural and transparent way.

All of these systems focus either on the matching or the composition of services, while we use a combined approach of service matching and composition resulting in a holistic vision on the given request. As pointed out in Section 4, to create a holistic composition, we employ a reasoner on the problem as a whole instead of solely on the junctions.

Classic planning literature [11] has solved many difficult problems using forward-chaining. Fortunately, the services required for metadata generation have no side-effects that “change the world”; they only generate an answer based on a request. This means that the platform did not have to provide a notion of time. Also, full forward-chaining reasoning would have proven difficult, considering the massive amount of information available on the Semantic Web. The simultaneous use of composer and supervisor, which uses a limited form of forward-chaining, creates a hybrid system that profits from forward-chaining when errors occur. A drawback of several existing planners is that their functionality is inherently limited by the amount of intelligence they contain. Our approach is able to take into account domain-specific knowledge provided by the user or available on the Semantic Web.

As for efficiency and scalability, we use the Eye reasoning tool, which outperforms several other generic reasoners [2,20]. Furthermore, we do not try to find rigorous solutions up front, but start with a basic execution plan which the supervisor adapts as complications arise. The reasoner and the blackboard can also cache intermediary results (e.g., additional knowledge derived from existing facts and rules).

8 Concluding remarks

Semantic Web technologies can play a crucial role in image annotation. We list three advantages over traditional image annotation approaches:

– the use of semantically described feature extraction algorithms, which can be plugged in as needed;
– an automatic matching and composing system, combined with a supervisor that is capable of advanced error recovery;
– algorithms benefit from domain-specific and situation-specific knowledge, enhancing their functionality beyond what was possible before.

The use case clearly indicates the possibilities and opportunities of this platform.

References

1. DBpedia, http://dbpedia.org/
2. Eye deep taxonomy benchmark results, http://eulersharp.sourceforge.net/2003/03swap/dtb-2010.txt
3. Freebase, http://www.freebase.com/
4. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Recommendation (Jan 2009), http://www.w3.org/TeamSubmission/n3/
5. 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)
6. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34 (2001)
7. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) 5(3) (2009), http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf
8. Corkill, D.D.: Blackboard systems. AI Expert 6(9), 40–47 (Sep 1991)
9. De Roo, J.: Euler proof mechanism, http://eulersharp.sourceforge.net/
10. Fensel, D., Bussler, C.: Semantic Web Enabled Web Services. In: 2nd Annual Diffuse Conference (January 2002)
11. Finzi, A., Pirri, F., Reiter, R.: Open world planning in the situation calculus. Proceedings of the National Conference on Artificial Intelligence pp. 754–760 (2000)
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. Hristoskova, A., Volckaert, B., Turck, F.D.: Dynamic composition of semantically annotated web services through QoS-aware HTN planning algorithms. In: ICIW '09: Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services. pp. 377–382. IEEE Computer Society, Washington, DC, USA (2009)
14. Junghans, M., Agarwal, S., Studer, R.: Towards Practical Semantic Web Service Discovery. In: The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30-June 3, 2010, Proceedings. Springer (June 2010)
15. Klusch, M., Fries, B., Khalid, M.: OWLS-MX: Hybrid OWL-S service matchmaking. In: In Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web (2005)
16. 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/
17. Li, L., Horrocks, I.: A software framework for matchmaking based on Semantic Web technology. In: WWW '03: Proceedings of the 12th international conference on World Wide Web. pp. 331–339. ACM Press, New York, NY, USA (2003)
18. 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/
19. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the Semantic Web. The VLDB Journal 12(4), 333–351 (2003)
20. Osmun, T.: Euler Eye installation, demo, and deep taxonomy benchmark, http://ruleml.org/WellnessRules/files/WellnessRulesN3-2009-11-10.pdf
21. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-query/
22. Redavid, D., Iannone, L., Payne, T.: OWL-S atomic services composition with SWRL rules. In: Proceedings of the 4th Italian Semantic Web Workshop (Dec 2007), http://eprints.ecs.soton.ac.uk/15658/
23. Roy, P.V., Haridi, S.: Concepts, Techniques, and Models of Computer Programming. MIT Press, Cambridge, MA, USA (2004)
24. Shin, D., Lee, K., Suda, T.: Automated generation of composite Web services based on functional semantics. Web Semantics: Science (Jan 2009)
25. Smith, J.R., Schirling, P.: Metadata standards roundup. IEEE MultiMedia 13, 84–88 (2006)
26. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web services. Journal of Web Semantics 1(1), 27–46 (December 2003)
27. Troncy, R., Mannens, E., Pfeiffer, S., Van Deursen, D.: Media Fragments URI 1.0. W3C Working Draft, http://www.w3.org/TR/media-frags/
28. 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)
29. Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision 57(2), 137–154 (May 2004)