

# Functional Description of Geoprocessing Services as Conjunctive Queries<sup>1</sup>

Daniel Fitzner<sup>1</sup>, Jörg Hoffmann<sup>2</sup>

<sup>1</sup>Institut für Geoinformatik, Münster  
fitzner@uni-muenster.de

<sup>2</sup>Universität Innsbruck, DERI  
joerg.hoffmann@deri.at

**Abstract.** Discovery of suitable web services is a crucial task in spatial data infrastructures. We present a novel approach to the semantic discovery of Geoprocessing Services (WPS). Requests and WPS are annotated as conjunctive queries in a logic programming (LP) language and the discovery process is based on LP query containment checking between these descriptions. Besides the types of input and output, we explicitly formalise the relation between and hence are able to capture the functionality of WPS more precisely. The use of LP query containment allows for effective and decidable reasoning tasks during service discovery.

## 1 INTRODUCTION

Different types of geospatial web services exist: those that provide (e.g. WFS), display (e.g. WMS) or process (e.g. WPS) spatial data. One of the most challenging problems in the area of (geospatial) web services is discovery, the process of locating web services that potentially fulfill some user's needs. Recent works e.g. from Lutz et al. have shown that geospatial service discovery is more precise when not only relying on syntactical comparison (keyword matching) of discovery request and service advertisement, but also taking into account well-defined and formal semantics, specified in ontologies.

Most efforts that use a formal language to describe Geospatial discovery-requests and service-advertisements concentrate on data providing services such as the OGC's Web Feature Service, and only provide formal descriptions of the service's output. Geoprocessing services are different since they offer functionality (usually found in Geographic Information Systems) instead of data (some WFS with filter encoding combine both). In order to be executed, they require a specific input to be delivered by the user, e.g. a

---

<sup>1</sup> This work is funded by the European Commission under the SWING project (FP6-26514)

service offering the geospatial overlay-operation requires as input some objects to be overlaid. Therefore, the semantic description has to consider the input that is accepted by a specific service, and a service should only be discovered if the requester can provide this input. For the user, the question is:

"How can I discover a service that can compute on the input I can provide, the output I want?"

Hence, approaches to the functional description of geoprocessing services should at least be able to consider the input/output signature. Moreover, there are geospatial operations that have the same signature but behave differently. There are e.g. difference distance calculations that all have equal type signatures, but compute different operations such as the euclidean or spherical distance. Therefore, the functional descriptions should additionally consider *the relation between input and output*. Another crucial aspect is the computational cost of the discovery process. The current approaches either use a formalism that cannot capture the necessary level of detail (Lemmens, 2006), or they are highly expressive at the expense of costly, even undecidable, reasoning tasks during discovery (Lutz, 2007).

We show in this paper how the *logic programming (LP)* paradigm can potentially be a solution to all the above.

## 2 WPS DISCOVERY IN LP

We describe requests and services as *conjunctive LP-queries*, and base discovery on *query containment checking*; Figure 1 gives an overview. Each request or service-description consists, following the WSMO approach (Fensel, 2006), of a *precondition*, a *postcondition*, and a set of *shared variables* appearing in both. The formal terminology relies on domain ontologies that are derived from current standards for geographic datatypes, like e.g. the Spatial Schema (ISO, 2002) or GML (OGC, 2003).

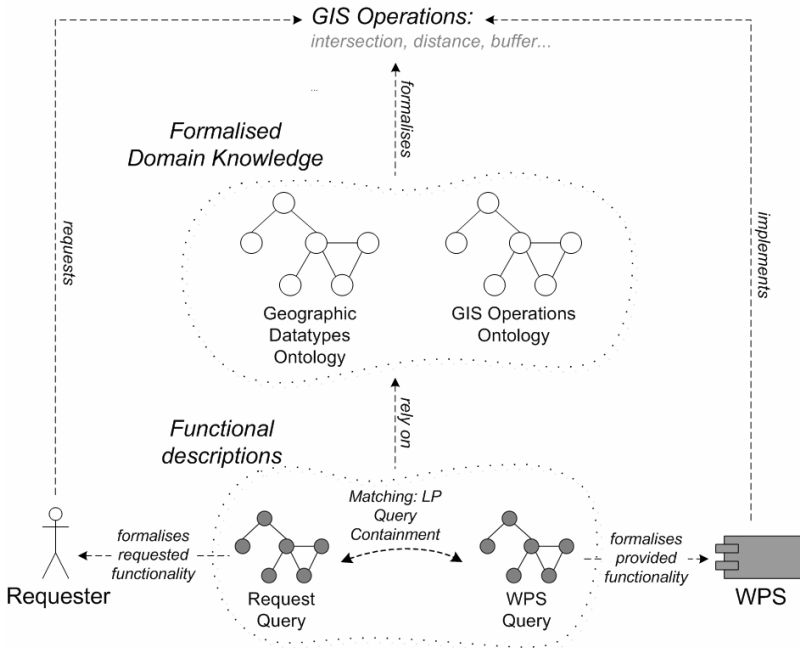


Figure 1: Overview

Computationally, LP query containment is a good choice because (in the variant we use) it is decidable, and a well-researched task for which powerful engines exist. In terms of expressivity, it is a good choice because the shared variables allow us to naturally formulate dependencies between inputs and outputs; note that this is not possible in Description Logics. Our descriptions formulate: 1. *The in/out signature*, modelled as unary predicates in pre/postcondition. 2. *The in/out dependencies*, modelled via shared variables. 3. *The performed operation*, modelled in an ontology of geospatial operations; this is a light-weight approach offering a good trade-off between accuracy for matching, and feasibility of annotation.

### 2.1 Example

Consider a service offering the geospatial difference operation on polygons (Figure 2) and a user request for  $difference(A,B)$  between polygons  $A$  and  $B$ .

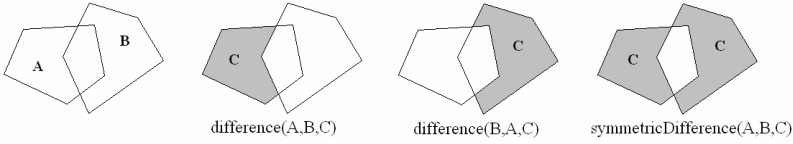


Figure 2: The difference operations

We formalize operations on polygons, and their relations, in our geospatial operations ontology, stating e.g. that symmetric-difference can be computed via difference and union operations. This is done via logic programming rules like e.g.:  $\text{symmetricDifference}(A,B,C) :- \text{difference}(A,B,E) \wedge \text{difference}(B,A,F) \wedge \text{union}(E,F,C)$ .

Request and service are annotated as follows:

*Request R :*

$$R_{pre} : \text{polygon}(A) \wedge \text{polygon}(B)$$

$$R_{post} : \text{polygon}(C) \wedge \text{difference}(A,B,C)$$

$$R_{share} : A, B$$

*Service S :*

$$S_{pre} : \text{polygon}(X) \wedge \text{polygon}(Y)$$

$$S_{post} : \text{polygon}(Z) \wedge \text{difference}(X,Y,Z)$$

$$S_{share} : X, Y$$

The  $\text{polygon}(\cdot)$  statements provide the in/out signature. Note that, when considering only these signatures, one could choose to map  $X$  to  $B$  and  $Y$  to  $A$ , obtaining a wrong result. This is precluded by the in/out dependencies given in the shared variables.

## 2.2 Matching by Query Containment

We define the following *Plug-In-Match*:

1. Step:

Test whether  $R_{pre} \subseteq S_{pre}$ .

If this test succeeds, we get a non-empty set  $IM$  of *input mappings*  $im_k : \text{var}(S_{pre}) \rightarrow \text{var}(R_{pre})$  from the variables in  $S_{pre}$  to the variables in  $R_{pre}$ : under those mappings, the containment check succeeds.

2. Step:

Denoting with  $y_1 \dots y_n (x_1 \dots x_m)$  the shared variables of  $S(R)$ , test for all  $im \in IM$  whether:

$$S_{post} \wedge shared(y_1) \wedge \dots \wedge shared(y_n) \wedge order(y_1 \dots y_n) \subseteq \\ R_{post} \wedge shared(x_1) \wedge \dots \wedge shared(x_m) \wedge order(im(y_1), \dots, im(y_n))$$

**Step 1** ensures that the service can execute on the request's input. The different input mappings are the different possible instantiations of the inputs. We are only interested in those mappings that have a correspondent output mapping adhering to the shared variables; this is ensured in **Step 2** by the inclusion of the *shared(.)* and *order(...)* predicates.

Note that, with the ontology of operations, when matching by query intersection rather than query containment, we can discover relevant services (difference, union) for a symmetric-difference request even if there is no symmetric-difference service.

### 2.3 Example Match

1. Step:

$$polygon(A) \wedge polygon(B) \subseteq polygon(X) \wedge polygon(Y)$$

This delivers two input mappings  $im_1, im_2$ :

$$im_1(X) = A, im_1(Y) = B \text{ and } im_2(X) = B, im_2(Y) = A$$

2. Step:

Input mapping  $im_1$ :

$$polygon(Z) \wedge difference(X, Y, Z) \wedge shared(X) \wedge shared(Y) \wedge order(X, Y) \subseteq \\ polygon(C) \wedge difference(A, B, C) \wedge shared(A) \wedge shared(B) \wedge order(A, B)$$

Obviously, a suitable output mapping  $om(A$  to  $X$  and  $B$  to  $Y)$  exists, and the match succeeds.

Input mapping  $im_2$ :

$$polygon(Z) \wedge difference(X, Y, Z) \wedge shared(X) \wedge shared(Y) \wedge order(X, Y) \subseteq \\ polygon(C) \wedge difference(A, B, C) \wedge shared(A) \wedge shared(B) \wedge order(B, A)$$

This containment check fails due to the conflict between the use of  $A$  and  $B$  in *difference(...)* respectively *order(..)*.

Input mapping  $im_1$  (and output mapping  $om$ ) are then returned to the requester in order to ensure a valid instantiation of input (output) variables before (after) web service execution.