Integration of Qualitative Spatial Reasoning into GIS-An Example with SparQ

Sahib Jan, Malumbo Chipofya
Institute for Geoinformatics, University of Munster
{s_jan001 | mchipofya}@uni-muenster.de

1. Introduction

Qualitative reasoning is an approach for dealing with commonsense knowledge without using numerical computation. Instead, one tries to represent knowledge using a limited vocabulary such as qualitative relationships and qualitative categories for representing real values (J. Renz, B. Nebel, 2007). Qualitative approach is use to deal with incomplete knowledge and it is considered to be close to how humans represent and reason about commonsense knowledge. This point, among others, motivates the integration of QSR into Geographic Information Systems (GIS). During the last two decades a multitude of formal calculi for spatial relations has been proposed focusing on different aspects of space like topology, orientation, and distance (Freksa and Röhrig, 1993). However, the application of these calculi in GIS remains sparse. We approach this problem by building an appropriate Application Programming Interface (API) that encapsulates the functionalities of the qualitative spatial reasoner SparQ1 to make them available to GIS applications. Our API which is written in Java provides a set of Extensible Markup Language (XML) data structure for specifying the query to SparQ and returning the results. The API itself resides on the client-side and accepts XML structured queries which it then passes on to SparQ in the latter’s own syntax. Results from SparQ are converted back into XML and returned to the user application.

In this paper we will first describe the API we developed for SparQ and how it has been tested with the open source java-based GIS software OpenJUMP2. Then we will note some shortcomings of our work especially with respect to the applicability of the API in a broader context, for example considering restrictions on the types of entities on which qualitative reasoning can be applied. Finally, we will discuss some future directions for our work and the challenges we envisage.

2. Connecting GIS to SparQ

2.1 Qualitative Spatial Reasoning Using SparQ

---

1 SparQ was developed and is maintained at the University of Bremen under the project (R3-[Q-Shape], etc.). Go to [http://www.sfbtr8.uni-bremen.de/project/r3/sparq/](http://www.sfbtr8.uni-bremen.de/project/r3/sparq/) for more details.

2 OpenJUMP is an open source GIS application developed and maintained by the Canadian Companies Vivid Solutions and Refractions Research. Go to [http://www.openjump.org/](http://www.openjump.org/) for more details.
SparQ (Spatial reasoning done Qualitatively) is a toolbox that makes available a set of binary and ternary calculi and reasoning techniques developed in the QSR community (Wallgrün, 2009). As a toolbox, SparQ was designed to be used directly in other applications over a TCP/IP connection or as a standalone console application. It is a modular software program with four main modules. The Compute-relation module allows computation of operations defined in the specific calculi. It takes as parameters the name of an operation together with a set of variables representing entities from the appropriate domain and constraints between the given set of entities, each labeled by its corresponding relation. The Qualify module implements a single operation (qualify) which takes a quantitative scene description (coordinates of entities in the scene) and returns a qualitative description in terms of the possible constraints between the given entities for a particular calculus. Constraint-reasoning reads a description of a constraint network and performs operations to identify network consistence. Finally, the Algebraic-reasoning module is used for reasoning about real-valued domains using techniques of algebraic geometry.

2.2. QSR API Design

Our API comprises a set of Java classes and XML files. It contains a set of rules and specifications that a software program can follow to access services provided by the API. As previously stated, the purpose of designing an API in this study is to integrate qualitative spatial reasoning engines with GIS applications, particularly to integrate the reasoning engine SparQ with the GIS application openJUMP. The API will serve as an interface between these two applications. Initially, the API establishes a TCP/IP connection with the reasoning engine. Figure 1 shows the global architecture of the GIS-API-SparQ configuration. The API allows a user to send a query in XML format from the client (GIS) application and retrieve results in XML format as well. It parses the given XML file, transforms the query into the syntax and encoding format of the reasoner, forwards the query to the reasoner, and waits for the results. Upon receiving the results, the API transforms them back into defined XML structure and returns them to the GIS application.
XML files are defined based on an analysis of SparQ’s syntax. Qualitative calculi were analyzed using all modules and their specific operations in SparQ to identify their syntax commonalities. Each module in SparQ takes a command with a sequence of module specific parameters. The general syntax of a SparQ query as it would be given at the command prompt ($>$) of a terminal is as follows

```bash
$> ./sparq <module-name><calculus-name><module specifies parameters>
```

The module specific parameter must be consistent with the module and calculus specified at the beginning of the command. They include the set of operations, relations, and constraint-networks to be used for the reasoning task. We categorized the possible input queries and designed the XML structure for each module and module specific operation. Every XML query structure is set of tags that include module-name, calculus-name, operation, relations, and other modules specific parameters as shown, for example, in Figure 2 below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module name="qualify">
    <calculus type="binary" name="dra-24">
        <controlMode>all</controlMode>
        <entity type="-3 0 8 0">A</entity>
        <entity type="8 0 4 8">B</entity>
    </calculus>
</module>
```

**Figure 1:** API workflow overview

**Figure 2:** XML structure for input queries
The above given XML structure for input query is specific for the Qualify module. The XML query above contains module and calculus names and calculus type (binary or ternary) as attribute values. The controlMode tag in the query takes two value (ALL/First2all) used to return all possible constraints between given entities or to return possible constraints between first-two entities. Each XML structure for input query varies with respect to the module used.

2.2.1. From XML to SparQ Syntax

Queries in XML format are converted into SparQ syntax by reorganizing the data in the query and augmenting it with required non-functional syntactic elements like parentheses, blank-spaces, etc. We use the DOM parser to generate the document tree for a given XML file which is then mapped into a SparQ-syntax formatted query. The resulting query is then forwarded to SparQ as a simple text string. Figure 3 shows an XML query with the Constraint-reasoning module’s algebraic-closure operation.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module name="constraint-reasoning">
<calculus type= "binary" name="cardir">
<operation type= "constraint-reasoning">algebraic-closure</operation>
<entity>A</entity>
<relation>N</relation>
<entity>B</entity>
<entity>B</entity>
<relation>N</relation>
<entity>C</entity>
</calculus>
</module>
```

**Figure 3:** XML query for constraint-reasoning on given constraint-network

After processing the query above, the resulting string will look as follows.

```xml
<constraint-reasoning> “ ”<CARDIR> “ ”<algebraic-closure> “ ”((<A> “ ”<N> “ ”<B> “ ”)<B> “ ”<N> “ ”<C>)) “\n”.
```

2.2.2. SparQ Result Analysis

SparQ results are analyzed to identify the possible output patterns in the results for given input queries. We used all possible modules specific queries to find out commonalities between the results given by the reasoner and the type of errors generated. The purpose of result analysis is to design a suitable mapping into common XML data structures for given results.
Based on the analysis, we generalized SparQ outputs into the five categories of simple-text, simple-text and constraint-network, constraint-network, syntax errors and simple-relations. The standard XML structure for requesting a result from constraint reasoning is shown in the example presented in Figure 4.

```xml
<result type=“constraint-reasoning”>
  <operation name=“algebraic-closure” type=“constraint-reasoning”>
    <comments> Modified Network</comments>
    <entity>A</entity>
    <relation>rrlr</relation>
    <entity>B</entity>
    <entity>A</entity>
    <relation>ells</relation>
    <entity>C</entity>
  </operation>
</result>
```

**Figure 4: SparQ result in XML format**

The extracted results from SparQ are generalized into five categories. API coverts received result into defined output XML structure. The above mentioned result in XML format contains composition of text and constraint-network. Result-tag that represent used module for reasoning. Operation-tag contains operation-type and name to represent module and module specific operation used for reasoning. Comment-tag is used to pass a received comment to the end-user and the received constraint-network is split into entity-tags and relation-tags that represent possible relationship between given entities.

### 2.2.3. Converting Results From SparQ Syntax into XML

The developed API extracts results from SparQ and stores them as a string array. During conversion these results are split into sub results based on type of result received from the reasoner. The API contains set of module specific methods and conditions to extract and process the received results from the reasoner. For example, queries using algebraic-reasoning and scenario-consistency generate results as compositions of simple-text and constraint-networks. The results are extracted, stored in a string array, and then split into two string arrays based on predefined numeric values between 0 and 9 and the ‘.’ character. The array that contains simple text is forwarded as comment in the comment tags. The second array that contains the constraint-network is further processed and split based on SparQ syntax (punctuation) rules. During this process the data elements of the substring
are mapped back into XML as attribute values or text-data in defined standard tags resulting in a structure similar to the one shown in Figure 4.

2.3. Connecting OpenJUMP with API – A Specific Example

OpenJUMP supports reading and writing shape files and simple GML file format as other GIS applications. It supports different data format including GML, SHP, DXF, JML, MIF, TIFF and postGIS etc (Dalluege, 2006). OpenJUMP provides functionality to extend application by writing own plug-ins, cursor tools, renderers, and other such facilities with the help of the extension class. To test our API we implemented a Java based OpenJUMP plug-in and used it to pass qualitative reasoning tasks to SparQ. Presently, the extension consists of an input screen for selecting an XML file (using the Input text field in Figure 5) containing the query, displaying XML data for the input query as well as its results (the Output text area in Figure 5), connecting to and disconnecting from SparQ via the API, and sending the specified query to the reasoner.

![Figure 5: OpenJUMP plug-in provides interface for reasoning](image)

The OpenJUMP extensions API provides a broad array of functions that allow developers to not only write code that can access data loaded by the GIS but also to modify and enhance the behavior of standard functionality such as rendering, processing, and editing of the data. The simplicity of the extension model and the above stated advantages are what led us to select OpenJUMP as a testing platform for our study.
3. Concluding Remarks

3.1. Shortcomings and Future Challenges

The developed framework (API) is limited in several respects like selection of qualitative reasoning engines, automating spatial queries, and supporting visual representation of results on the client side. We developed the API based on a particular reasoner, SparQ, in mind. It is possible to improve the developed API to interact with other reasoners but this would involve substantial rewriting of the XML-to-reasoner-syntax translation code. The approach we intend to take here is to improve the current design by making our XML schemas more generic and integrate support for extensible style sheets (XSLT) on which query and result transformations can be based.

Task chaining in which a composite reasoning task is computed in a single query request is an essential feature for GIS applications. One of our targets is to support such chaining tasks by allowing the output of one or more tasks to be used as input for other tasks possibly with additional information supplied by the GIS between subtasks. To achieve this level of functionality we believe it is necessary to understand the sorts of GIS tasks that may require QSR methods. As such, one aspect that needs to be clarified is the utility of QSR in GIS. This leads us to analyze the scenarios in which employing qualitative as opposed to quantitative processing would be most beneficial or at least desirable for GISs and pin point those calculi that are most useful in those situations.

A final shortcoming that is worth noting here is related to the restrictions that formal definitions of qualitative spatial calculi place on the admissible primitive entities that can be reasoned over. For example, SparQ does not directly support reasoning over entities of type polygon (in Euclidean 2-Space) which would be useful for applying RCC reasoning on vector data with geometries of that type. Integrating multiple reasoning engines with different types of reasoning capabilities maybe constitute part of the solution to this problem.

3.2. Summary

The developed a platform independent API allows GIS users to integrate the reasoning engine with spatial applications like ArcGIS and OpenJUMP. The API provides a set of functions for establishing a connection with the reasoner through a GIS application and sending and receiving queries it. The main advantage of the API is that it supports a machine and human understandable format (XML) and can easily integrate with any Java based application or can be accessed over a network or can be converted into a web-based API. Further reasoning tasks can be performed on results from
any previous reasoning tasks enabling complex tasks to performed without requiring the reasoning mechanisms to be implemented in the GIS. The qualify facility provided by SparQ will allow developers of GIS extensions that use QSR to focus on the core functionality they intend to provide instead of the details of deriving qualitative descriptions from quantitative data or qualitative reasoning algorithms.

Acknowledgements

This work has been supported in part by an Erasmus Mundus Program scholarship granted by North-Rhine-Westphalian Ministry for Innovation, Science, Research and Technology (MIWFT) under object-number 1640052 and by the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant GRK 1498.

References


Uwe Dalluege (2006). OpenJUMP tutorial, Department of Geomatik, Hafen City University, Hamburg.