FORMAL LANGUAGES FOR EXPRESSING DATA CONSISTENCY RULES AND IMPLICATIONS FOR REPORTING OF QUALITY METADATA

Paul Watson

1Spatial, Cavendish House, Cambridge Business Park, Cambridge, CB4 0WZ, UK

KEY WORDS: Re-use, rules, metrics, mining, languages, metadata

ABSTRACT:

Over the past thirty years, in excess of 36 billion dollars has been invested in the collection of spatial information across the public sector in Europe (PIRA, 2000). There is a clear imperative now to maximise the use of that data captured at such great expense. The information society in which we live makes it ever easier to discover and exploit that data, in principle. However, data collected in the past for a specific application is not necessarily directly re-useable in a new and different context. The original collection process will have imposed many restrictions and brought many assumptions which need to be re-evaluated against the new usage scenarios. More difficult still is the fact that many of the underlying assumptions and restrictions were not explicitly stated, for example in data specifications, at the time but are simply inherent in the data itself. In practice, this means that data often has to be qualified and tested for the specific criteria important to a new pattern of use.

This paper discusses the requirements for a rules language which is capable of expressing such consistency criteria for spatial and other linked data which is initially found in scattered locations in an internet or intranet environment. The process of representing the knowledge inherent in scattered data and the rules which constrain it in a portable and platform independent way is illustrated through a series of examples. Knowledge persisted in this way can be used within real execution environments to generate automatically important data consistency and quality metrics. The storage of such quality metrics within a database and provision of registry enquiry services opens up the possibility of searching for data on explicit quality and consistency grounds to meet well-defined application needs. The implications of this for spatial data re-use and advanced data mining applications are reviewed.

1. INTRODUCTION

Since the advent of the digital age, data on a wide variety of subjects has been collected and stored in computer databases. Some of the earliest technology developed for the digital computer was database software. This data management function has been a key element in our developing use of the computer for decision support activities. However, to remain useful beyond its immediate application, data must be clearly classified and organised by type and purpose. This kind of metadata allows users or potential users of the data to understand the nature and context of the data and, when correctly organised, to discover data might be relevant to their current application via database registries.

Data is typically collected for one or a small number of specific uses by a particular individual or group. By the very nature of the collection process, many aspects of the data are specific to the originating application and these constraints of the data acquisition process act to limit the useful range of application of the data in future. In the most severe cases of mismatch, the required data elements for a new application are simply not present and the inappropriateness of the data for reuse is discovered immediately. However, the mere presence of the sought after data elements does not guarantee that they are fit for purpose in new applications. Both silent statistical and other selection biases inherent in the data acquisition and semantic differences in terminology between application domains can lead to unintentional misuse of data. These more subtle errors are not uncovered directly and may readily lead to false conclusions being drawn from the data.

These considerations apply equally well to spatial information as they do to any other data. Interoperability of services which exchange and reuse spatial information is dependent on interoperability of the underlying spatial information at two quite separate levels, syntactic and semantic. The syntax or encoding rules of the information must be well understood by the exchanging parties. XML encoding languages such as Geographic Markup Language (GML) provide a good foundation for ensuring syntactic interoperability based as they are on XML and XMLSchema with their own well-defined syntactic structure. GML provides additional encoding rules for constructing valid application schemata which further assist in the marshalling and unmarshalling of geographic features and their attributes and the representation of associations between them.

However, while these syntactic constraints are necessary for reliable data transport, they are not sufficient to ensure that the exchanging parties can correctly interpret the features meaning and exploit the features correctly for decision support purposes. To guarantee that the features are consistent with a particular domain interpretation (for example land management or contour data), it is necessary to describe the logical consistency constraints within that domain in a formal way and test the features against these constraints or rules. We know that, in general, land parcels should not overlap and contours should not intersect. However, simply building an application schema containing these terms (syntactic structure) does not guarantee that feature instances encoded in GML satisfy the logical domain constraints (semantic structure). The ability of the features to support reasoning and decision support tasks within the problem domain then depends on the degree of conformance with the domain consistency rules. These formal semantic rules must therefore be expressed in addition to the application schema. Given the particular role that spatial information plays in decision support, it has received much attention from governmental and other agencies and this has given rise to a classification of dedicated geographic metadata by the International Standards Organisation (ISO 19115). In addition to the conventional role of metadata in identifying the classification, source and authority for the data, ISO 19115 develops a sophisticated taxonomy of

metadata elements relating to data quality which relate directly to the range of applicability of the data. These elements address areas such as completeness (errors of commission or omission), logical consistency (conformance with a particular data model and set of constraints e.g. topology rules), thematic accuracy (whether data elements are correctly classified), positional accuracy and temporal accuracy. This metadata taxonomy is an invaluable way of conceptualising and categorising spatial metadata. However, the standard does not address in detail the precise contents of the metadata. Many metadata elements are specified simply as free text. This unstructured data model clearly facilitates simple uses such as browsing by a human being or free text searching. However, as the range of sources and variety of spatial information available on line grows, it will become increasingly difficult reliably to locate accurate and relevant information and interpret its meaning without a more formal content specification for metadata. This content specification should allow the logical and semantic constraints which apply to spatial information to be represented as a set of logical rules which the data obey and for these rules to be queried across the Web.

The aim of this paper is to define a set of requirements for such content and outline an implementation of that logical rules language which leads to rigorous and semantically meaningful and queriable metadata.

2. METHODS

2.1 Requirements

We begin by setting out the requirements of the rules language which will allow the logical constraints satisfied by a particular data source to be specified.

1. Unambiguous. It is important that the domain constraints are expressed in a mathematically rigorous way. This allows the rules to be used as the basis of fair testing and means that the results of such testing are fully objective and are not open to interpretation.
2. Logical & Portable. Given the distributed and diverse nature of geographic data, it is advantageous to keep a logical separation between the terms and definitions (ontology) of the feature application schema and the terms and definitions of the domain model to which the rules apply. In other words, the rules are genuinely abstract knowledge and should be decoupled from any particular physical implementation of the instance data to allow reuse of the logical rules with any number of logically compliant feature sources. A feature source may be compliant with the rules either directly, because it is expressed using the same ontology as the rules, or alternatively, a feature source may be compliant with the logical model of the domain even though the syntactic structure of the data differs. In this case, various model transformations, whether simple styling rules or otherwise, may be used to covert feature instance data into a form which can be verified against the rules.
3. Compact. For reasons of manageability and ease of comprehension, it is important that the rules language has a concise grammar. The number of distinct concepts in the language should be kept as small as reasonably possible.
4. Intuitive. It is important that the language be naturalistic and easy to learn. The transformation between constraints expressed using natural, spoken language and the formal rules language should be kept as simple as possible. In some cases, this may conflict with the requirement for simplest terms, above. A balanced view is required in these cases.
5. Quantitative. The language should support quantitative reasoning about the feature data and the development of formal metrics which summarise the level of compliance to data quality measures.
6. Web compatible. The language should be compatible with feature data which is scattered across multiple physical and organisational barriers. Constraints which apply between feature data held under separate authority and control is often equally important to constraints which apply within data under the same authority. The principal implication of this requirement is that the rules language support a naming authority and disambiguation scheme or namespace support.
7. Declarative & Refinable. The nature of collection and management of feature data is such that the entire rules base which constrains the data is very rarely known completely at the start. Both within and between sources of feature data, new constraints are constantly be discovered and added to the rules base. It is important, therefore, that the rules language makes it simple to add and refine rules without disrupting the overall structure of the rules base unduly

2.2 Choice of Rules Language

The field of knowledge management or the authoring and exploitation of abstract knowledge representations has received much attention recently through initiatives such as the Semantic Web community (W3C, 2004b). A key objective of this initiative is to make the exchange of mathematically rigorous models of knowledge such as conceptual graphs possible. This has led to the development of Web Ontology Language (OWL) (W3C, 2004a) as the basis for conceptual representation. This language has its foundations in a field of mathematical logic called Description Logic and this has been used to formally classify the different complexity classes of different sorts of logical expression. OWL divides the complexity of expressions into three kinds:

1. OWL Lite. This is a simple dialect suitable for expressing simple concepts and relationships.
2. OWL DL (Description Logic). This sub-language represents only concepts which are formally decidable (there exists a decision procedure whether a logic expression is true or false.)
3. OWL Full. This language permits a much richer range of expression (e.g. concepts which may represent both instances and classes) which make the language formally undecideable (there exists no decision procedure).

Development to date has concentrated on OWL Lite and OWL DL precisely because these sub-languages are mathematically tractable and therefore completely general tools support is feasible. These languages support various reasoning tasks such as deriving complex logical classification schemes (as entailments) from a simpler set of declared relationships.

Some kinds of constraints, especially reasoning over relationships are not supported using the concepts defined in OWL but can be expressed using a rules language layered on top of OWL. An early candidate draft of Semantic Web

Rules Language (W3C, 2004c)) has been proposed within the Semantic Web initiative for this purpose. SWRL extends the conceptual model of OWL to include rules expressions. However, SWRL contains a fixed set of built-in operators which address only basic XMLschema datatypes and therefore have no support for derived geometric types. This makes SWRL unsuitable for the current purpose. Instead, a dedicated XML grammar based on first-order logic (predicate logic) was developed and used. It should be recognised that the approach here is conceptually similar to SWRL with the addition of support for spatial operators.

The XML rules have a very simple vocabulary:

1. Predicate, an operator which returns either true or false
2. Constant
3. Variable –free or bound
4. Built-in Function
5. Logical Connective - NOT, AND, OR
6. Quantifier – universal, existential

2.3 Predicate Types

Predicate Type
Relational Predicate
Exists Predicate
ForAll Predicate
Conditional Predicate
Referential Predicate
Range Predicate
And Predicate
Or Predicate
Not Predicate

Table 1. Predicate Types

The simplest predicate type is the RelationalPredicate. It is used to check whether two Values (see below) have a defined relation. It consists of two Values, an LeftValue (Lvalue), a RightValue (Rvalue) and a comparison operator (Relation).

The ExistsPredicate is an existential quantifier. It contains a feature type, a numerical quantifier, a relation and a child predicate. It allows expressions of the form, "There exist greater than 3 features of type B for which the following condition holds –> {child predicate}". This may be used to test for the existence or absence of features of a particular type, as in "For Lake features: There exist exactly zero forest features for which the forest geometry is contained within the lake geometry."

The ForAllPredicate is a universal quantifier. It contains a feature type and two child predicates. It allows expressions of the form, "For all features of type X which satisfy {first child condition} verify that {second child condition} also holds true.

The ConditionalPredicate permits conditional evaluation of parts of a rule. It contains two child predicates. It allows expressions of the form, "If {first child condition} holds then check that {second child condition} also holds."

The ReferentialPredicate tests whether a particular named association exists between two features. It contains two target feature types and an association name. It allows expressions of the form, "Check if there exists a relationship from {feature instance A} to {feature instance B} via the association {reference name}".

The RangePredicate tests whether a value lies in a range. It contains three Values and tests the first supplied Value to find whether in lies between the second and third supplied Values. It allows expression of the form, "Check whether {First Value} lies between {Second Value} and {Third Value}."

The logical predicates AndPredicate, OrPredicate and NotPredicate allow for Boolean logic to be applied to any of the results returned by other predicate types. AndPredicate and OrPredicate take two child predicates and return the standard Boolean result. The NotPredicate logically inverts the sense of the child predicate result.

2.4 Value Types

Value Type
Static Value
Dynamic Value
Temporary Value
Conditional Value
Aggregate Value
Built-in Function Value
Class Value
Summed Value
Difference Value
Product Value
Quotient Value
Modulus Value
Negated Value

Table 2. Value Types

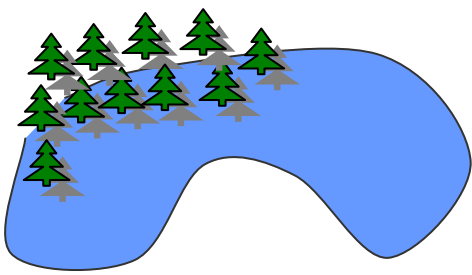
A StaticValue is a typed constant. Its value is assigned explicitly within the rule expression and this value can then be used within other comparisons such as RelationalPredicates. The only datatypes currently supported are simple scalar datatypes such as integers, reals and strings.

An AssignableValue represents a variable in a rule expression is one of two types – a DynamicValue is a typed attribute fetched from a feature instance, a TemporaryValue is used to hold a derived result within a rule for comparison in a later and possibly unrelated clause.

A ConditionalValue is a value which may take one of two values depending upon the truth of a child predicate. It contains two values and a predicate. If the predicate evaluates to true the first value is returned else the second is returned.

An AggregateValue is used to return some Aggregated result (sum, average, concatenation, geometric union etc.) from a number of features. It contains a feature type, a feature attribute name, an aggregation function and a child predicate which holds true for the features to be aggregated. It allows expressions of the form, " For features of type {Type} which satisfy {Child Predicate}, compute and return the {Aggregation Function} from the attributes {Attribute Name}."

A BuiltinFnValue is used to derive one Value from another using a specified algorithm. It contains a Value of any type and an algorithm name. A variety of algorithms are supported varying by the datatype of the Value supplied, including simple mathematical and string manipulation functions as well as geometric algorithms such as convex hull, buffer or Douglas Peucker simplification. This functionality can be used, for example, to test whether a feature lies within a specified buffer of the geometry of another feature. (The set of supported algorithms can be augmented by



implementing an algorithm according to a particular rules system interface. The algorithm then becomes available as another Built-in function within the rules language.)
A ClassValue returns the class name or feature type of a feature.
The final set of Value types are simple arithmetic convenience types, SummedValue, DifferenceValue, ProductValue, DivisionValue, ModulusValue, NegatedValue, having the conventional meanings.

2.5 Relation Types

Relation Types
Scalar
Equals Relation
NotEquals Relation
Less Relation
LessEquals Relation
Greater Relation
GreaterEquals Relation
Begins Relation
Ends Relation
RegExp Relation
Spatial
Spatial Equals Relation
Spatial Disjoint Relation
Spatial Intersects Relation
Spatial Touches Relation
Spatial Overlaps Relation
Spatial Crosses Relation
Spatial Within Relation
Spatial Contains Relation
Spatial Within Distance Relation

Table 3. Relation Types

Relation types are gathered into two groups, ScalarRelation and SpatialRelation. ScalarRelation specifies a relationship test between two scalar values of an appropriate type. Numerical relationships supported are EqualsRelation, NotEqualsRelation, LessRelation, LessEqualsRelation, GreaterRelation, GreaterEqualsRelation, with the conventional meanings. Character String relationships are BeginsRelation and EndsRelation which test whether a character string value begins or ends with the supplied fragment or RegExpRelation which tests whether a character string value matches a supplied fragment according to a PERL-compatible regular expression.
SpatialRelation types correspond to the ISO/OGC Simple Feature specification spatial interaction types (ISO 19125-2:2004) and take those meanings. In addition to the topological interaction types, SpatialWithinDistanceRelation can be used to test whether to geometries approach within a user specified distance.

2.6 Rule Examples

This example represents the simplest spatial consistency test possible. It states the constraint that, in most cases, the presence of forest within water areas is inconsistent. Therefore, forest features should be tested to ensure that their geometry does not intersect the geometry of any water body features. The illegal forest features can be depicted graphically:

Figure 1: Illegal Coniferous Forest-Water Area Relationship

The constraint might be expressed in prose as follows:

Check for **Coniferous Forest** objects that there are no **Water Area** objects for which **Coniferous Forest**.geometry overlaps **Water Area**.geometry

The rule can be visualised using a predicate tree structure as follows:

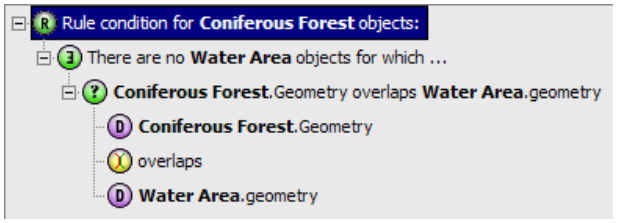
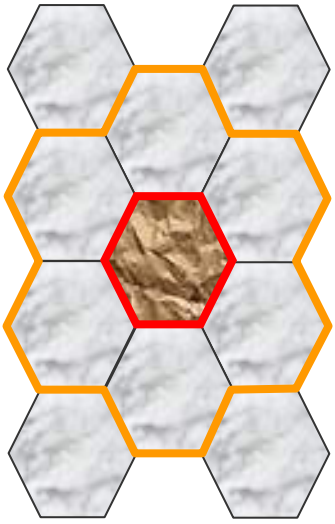


Figure 2: Predicate Tree Structure for Coniferous Forest – Water Area Consistency Rule

This tree shows that the main rule structure is an ExistentialPredicate testing for the existence (or non-existence in this case) of Water Area features which meet a particular RelationalPredicate. The RelationalPredicate tests candidate Water Area features to see whether their geometries overlap the Coniferous Forest feature currently under test.
This predicate tree corresponds very closely with the XML serialisation of this rule:

```
<?xml version="1.0"?>
<Rule>
 <RootPredicate classLabel="Coniferous Forest">
  <ExistsPredicate qualifier="exactly" n="0" classLabel="Water Area">
   <RelationalPredicate>
    <DynamicValueclassRef="Coniferous Forest" propName="geometry"/>
    <SpatialOverlapsRelation/>
    <DynamicValue classRef="Water Area" propName="geometry"/>
   </RelationalPredicate>
  </ExistsPredicate>
 </RootPredicate>
</Rule>
```

The target feature types appear as the classLabel and classRef attributes of the appropriate Predicates and Values and the feature property names for each DynamicValue are given in the Value propName attribute.
In this second example, we show that some complex and powerful expressions may be constructed from the relatively simple building blocks of Predicates, Values and Functions. Some slightly more advanced features of the rules language such as BuiltinFunctionValues and AggregateValues are used to demonstrate the use of derived results internal to the logic of the rule.
This rule tests that the shoreline of Island features matches the corresponding limits of all of the Water Areas which border the Island.
We can portray the correct relationship between Island and Water Area:
The Island is the brown hexagon at the centre of the picture. It is surrounded by a number of Water Area features (blue hexagons). The derived shoreline of the Island is drawn in red. The derived set of Water Area features which abut the Island are outlined in orange.



The rule can be expressed in something approaching prose as:
Check for **Island** objects that outer_ring(**Island**.geometry) equals intersection(**Island**.geometry,union(**Water Area**.geometry) over all **Water Area** objects for which (**Water Area**.geometry touches **Island**.geometry))

Figure 3: Island Water Area Consistency Rule

The corresponding predicate tree looks like this:

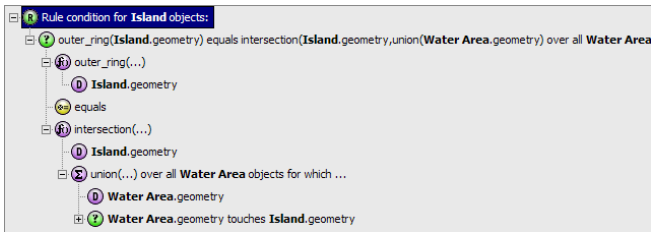


Figure 4: Predicate Tree for Island Water Area Consistency Rule

This tree is a simple RelationalPredicate which compares a BuiltinFunctionValue (outer_ring) with another BuiltinFunctionValue (geometric intersection) which in turn nests an AggregateValue (geometric union over Water Areas touching the Island) and tests them for (geometric) equality. The resulting tree is very compact for such a sophisticated expression.
Once again the XML encoding closely mirrors the predicate tree structure.

```
<?xml version="1.0"?>
<Rule>
 <RootPredicate classLabel="Island">
  <RelationalPredicate>
   <BuiltinFnValue fnName="outer_ring">
    <DynamicValueclassRef="Island" propName="geometry"/>
   </BuiltinFnValue>
   <SpatialEqualsRelation/>
   <BuiltinFnValue fnName="intersection">
    <DynamicValue classRef="Island" propName="geometry"/>
    <AggregateValue fnName="union" classLabel="Water Area">
     <DynamicValue classRef="Water Area" propName="geometry"/>
     <RelationalPredicate>
      <DynamicValue classRef="Water Area" propName="geometry"/>
      <SpatialTouchesRelation/>
      <DynamicValue classRef="Island" propName="geometry"/>
     </RelationalPredicate>
    </AggregateValue>
   </BuiltinFnValue>
  </RelationalPredicate>
 </RootPredicate>
</Rule>
```

An XSLT stylesheet also caters for rendering this rule into pseudo-prose, although as rules incorporate many BuiltinFnValues and AggregateValues the result becomes less clear and less like spoken English. It remains a good sanity check, however, as reading the styled rule through helps to confirm that the meaning has been captured correctly.
A further advantage of the strict hierarchical structure is that it is simple to parse the rule to determine its validity and feedback any syntactic inconsistencies (e.g. values out of scope) in the rule to the user.

2.7 Metadata Publication

The final results of the conformance tests are obtained in the form of metadata which is compliant to the conceptual model of ISO 19115 Metadata and encoded in the form recommended in ISO 19139. The results are supplied within the DQ_DataQuality metadata element as DQ_Element descriptors. The nameofMeasure and measureIdentification are taken from the corresponding rule or ruleset identifier. The dateTime is taken from the completion time of the conformance check and the results (DQ_Result) are compiled from the appropriate summary statistics within the conformance checking session. The metadata can be published automatically to a compliant OGC Catalogue (OGC, 2005) for long-term archiving and to facilitate discovery of data with appropriate quality characteristics.

2.8 Implementation

We have made an on line rules engine, Radius Studio, capable of evaluating logical rules expressed using the language on multiple sources of vector feature data from disparate locations.
The server has been implemented as a number of stateful web services each of which manage a distinct set of entities within the system, such as datastores, rules or sessions. In each case, the service has been exposed using a standardised SOAP binding (W3C, 2003) with request/response messages in RPC/literal form.
In addition, for the benefit of clearer demonstration, a thin, javascript, browser-based client was written to facilitate interaction with each of the service components.
A datastore is an external repository for data, usually including a geographic component. A datastore acts as an abstraction over feature services including OGC Web Feature Service (OGC, 2004) . The user selects some or all of the data in the store by specifying the feature types and attributes of interest and an extent for spatial selection. The service checks it for conformance to a defined set of rules and optionally applies automated corrections. Any corrected data may be returned to the same data store or a different data store. An optional schema mapping defines how data should be converted between the schema of a data store and the internal rules schema used by the server. It translates between feature and attribute types, and classes and properties in the server workspace. The user selects which features and attributes to import and defines the names of the corresponding classes and properties for the rules environment. This permits data from different stores to be compared more flexibly using a common set of terminology for any given domain such as transport or hydrography.
It is possible to define several data stores that access the same data through different schema mappings. It is possible to read data from several data stores into the service for

processing against a set of rules that analyse relationships between datasets as well as within a dataset. Each data store has two schema mappings associated with it – one input and one output. Output mapping is not needed if the service is being used only for checking rules without changing data. It is also possible to input from one store and output to a different store. The service also provides integrated support for externally defined ontologies which describe the structure of the data in a specific data store. This is achieved by interfacing with the open source Jena ontology library (see http://jena.sourceforge.net/), allowing ontologies in various formats such as RDF and OWL to be read into the service and used for rules authoring and rules-based reasoning.

A rule is a logical expression which can be used to test the logical consistency of a feature. The rule is expressed using the rules language which is an XML encoding of first order logic as outlined above. Rules are managed within the server by a dedicated web service – the RuleManager Service. This service allows rule expressions, along with suitable metadata, to be stored and their definitions retrieved and used within conformance checking and data reconciliation tasks.

The SessionManager service allows the definition of an ordered sequence of tasks to process data. The service also manages the execution of these sequences against feature instance data and storage and retrieval of the resultant metadata.Task types are:

• *Open Data*, which enables access to data from a defined datastore. A session may choose to open data from a number of data sources and then check rules based on relationships between features stored in different locations.

• *Discover Rules*, which analyses data based on a defined discovery specification to identify candidate rules.

• *Check Rules*, which checks a defined set of rules on the data and reports non-conformances. It is also possible to publish conformance checking results to a catalogue server. Published metadata consists of data quality items (qualitative and quantative metrics) and is encoded in a standard form (ISO 19139).

• *Apply Action*, which applies one or more actions to the data which are encoded similarly to rules.

• *Apply Action Map*, which checks a set of rules defined in an action map and applies the associated action to each non-conforming object.

• *Commit Data*, which will incrementally commit any data changes back to the data store it came from. Typically this occurs after a correcting action has been applied by an action or action map.

• *Copy to...* which will copy data to a different data store.

• *Pause*, which requests the service to suspend processing to allow results to be examined before processing the next task in the session sequence.

A session can be viewed as a specialised workflow template for data quality testing and reconciliation. The template can be stored, retrieved and modified as necessary to work against different sources of feature data or to incorporate new rule definitions or actions.

The Rule Builder allows the definition of potentially complex rules with an easy to use, tree structrured browser interface. The rule is expressed as a series of clauses built up using pulldown menus from the bar immediately above the graphical illustration of the rule.

The description at the bottom provides English text representing the currently selected clause; in this case the complete rule. The element details are used to specify the parameters associated with the currently selected rule. This part of the form always includes a description of the information required. In this case, the top-level rule specifies the class to be checked. An optional name label is used when the rule needs to distinguish between two different features of the same class. While editing a rule, it may temporarily be incomplete until a new clause is added or parameters are defined. These problems are highlighted clearly in red and a description of what is required displayed. Multi-level undo/redo is available to recover from mistakes while editing. Drag and drop can be used to reorder clauses of a rule. Cut and paste can be used to transfer all or part of a rule into another rule.

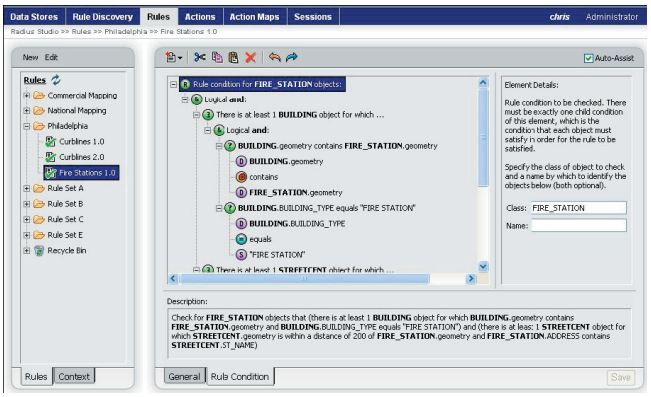


Figure 5: Rules Browser Form

3. RESULTS

This section describes the use of the SOAP Web Service interfaces for the purposes of validating features remotely. The features in question are a pair of data layers (MOORLAND and LFA) from different feature services with a geometric containment constraint between the feature instances. The scenario will use just one of the service endpoints – the SessionManager. All other objects, Datastores and Rules, are assumed to exist before the session commences. Note that Datastores and Rules may be created dynamically through the relevant web service endpoint just like the Session, but this usage will not be illustrated directly.

The first task is to create a work session definition in which the validation rules will be checked. This session definition consists of a number of tasks: a connection must be established to the relevant data source(s); a set of rules will be indicated for checking. The SOAP message to create the session object within the system is as follows:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://laserscan.com/RadiusStudio">
                <soap:Body>
                    <ns:create>
                        <DTO_1>
                            <id
xmlns:ns1="http://www.w3.org/2001/XMLSchema-instance"
ns1:nil="true"/>

                <metadataXml><![CDATA[<Metadata>
<Name value="Test Session"/>
<Description value="Created by TQAS Web Service"/>
<Comments value="Should be deleted after use."/>
</Metadata>]]></metadataXml>
                            <name>Test Session</name>

                <parentId>0ac8dd7dc0a85abd01bd967519e607
48</parentId>
```

```
                        <referencesXml
xmlns:ns2="http://www.w3.org/2001/XMLSchema-instance"
ns2:nil="true"/>
                        <sequence>1</sequence>
                        <xml><![CDATA[<Session>
<Sequence>
<Task label="1" type="Open Data">
<DataStoreRef ref_id="0ac3a94ac0a85abd0181f53b7fc2e033"/>
</Task>
<Task label="2" type="Open Data">
<DataStoreRef ref_id="0ac98ebdc0a85abd019732afc38ace6e"/>
</Task>
<Task label="3" type="Check Rules">
<RuleRef ref_id="1a714a8ec0a85abd01dafa55d327e7be"/>
</Task>
</Sequence>
</Session>]]></xml>
                    </DTO_1>
                </ns:create>
            </soap:Body>
</soap:Envelope>
```

There are two main sections to the request: the metadataXML and definition of the sequence of tasks itself xml. This example states that the system should open and read data from two data sources (each known by a unique identifier) and that is followed by conformance checking a single rule, again known by its unique identifier. The check rules task can also refer to a folder (logical collection of rules) but the syntax is identical.The Datastore and Rule objects which are referenced within the body of the session definition are defined in a similar way to the Session object, by the create() method, and may be queried and retrieved by invoking the get() and getByName() methods on the appropriate DatastoreManager or RuleManager service endpoint.

The response is in effect a copy of the input embellished with further metadata and internal references and crucially an identifier in the id element which can be used to refer back to this session definition. Once the session has been defined, it can be run using the run() method. The request is given below:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://laserscan.com/RadiusStudio">
    <soap:Body>
        <ns:run>

    <String_1>1a7331fbc0a85abd006c7fcd01bc0b08</String_1
>
        </ns:run>
    </soap:Body>
</soap:Envelope>
```

The progress of any session can be monitored using the getSessionProgress() method by passing in the persistent identifier of the Session. The request is given below:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://laserscan.com/RadiusStudio">
    <soap:Body>
        <ns:getProgress>

    <String_1>1a7154bfc0a85abd006c7fcd67afd715</String_1
>
        </ns:getProgress>
    </soap:Body>
</soap:Envelope>
```

The session passes through a number of states while executing and the sequence number of the current working task and % complete are reported while the session continues to execute. Once the session has run to completion, it is said to be in the “finished” state as given in the status attribute of the Progress element. To obtain the Session results, use the getSessionResults() method. An example request is given below:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://laserscan.com/RadiusStudio">
    <soap:Body>
        <ns:getResults>
<String_1>1a7154bfc0a85abd006c7fcd67afd715</String_1>
            <String_2>3</String_2>
            <int_3>1</int_3>
            <int_4>10</int_4>
        </ns:getResults>
    </soap:Body>
</soap:Envelope>
```

This method takes four parameters: the identifier of the session, the task within the session for which the results are required (3 was the Check Rules task), the index of the first result required starting at 1, the index of the final result required (0 for all results).

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://laserscan.com/RadiusStudio"
xmlns:ns1="http://dto.studio.radius.laserscan.com/jaws">
    <env:Body>
        <ns0:getResultsResponse>
            <result>
                <Results                                count="10"
finished="1164379794439"          first="1"          last="10"
started="1164379791002">
                    <Summary count="228" error="0" label="3"
total="1355" type="Check Rules">
                        <Object              class="MOORLAND"
count="228" error="0" total="1355"/>
                        <RuleRef     count="228"     error="0"
ref_id="1a714a8ec0a85abd01dafa55d327e7be" total="1355"/>
                    </Summary>
                    <Object class="MOORLAND">
                        <RuleRef
ref_id="1a714a8ec0a85abd01dafa55d327e7be">&amp;lt;font
color=red&amp;gt;&amp;lt;u&amp;gt;there is exactly 1 LFA
object for which MOORLAND.geometry is contained within
LFA.geometry&amp;lt;/u&amp;gt;&amp;lt;/font&amp;gt;</Rule
Ref>
                        <Attribute name="ID">62.0</Attribute>
                        <Attribute name="geometry">
                            <MBR        x0="304193.000139739"
x1="361489.001589227"             y0="482708.995242653"
y1="527026.999345939"/>
                        </Attribute>
                    </Object>
…… truncated ……
```

The Results element indicates the number of individual results contained and the start and end timestamps as milliseconds since UTC origin. The response has two major sections: the summary which gives overall conformance levels for the rules checked; the detailed per-feature metadata (i.e. which features failed which rule checks). The Rule identifier and text is included with any unique key attributes

and the bounding box (envelope) of the checked feature geometry attribute. Detailed feature level metadata can be used in manual reconciliation.

This information can be used to make a range of decisions about data quality in the overall context of a workflow. In the example above, the service indicates that 288 out of 1355 (16.8%) failed the single rule check giving an overall conformance level of 83.2%.

4. CONCLUSIONS

We have shown that distributed geospatial data validation and data quality reporting is feasible within an open Web Services environment. In addition, the utility of encoding quality rules and constraints in an open XML-based form has been shown and the approach followed, that of using a first-order logic formalism, appears to yield rules which are both compact and to a large degree generic and implementation independent. The choice to implement the conformance service using standard SOAP RPC/literal bindings and the success of the resulting scenario also shows that the methodology is suitable for integration into much richer and potentially dynamic workflow environments such as are enabled by enterprise workflow technologies like BPEL (OASIS, 2007).

The implications of collecting quantitative metadata relating to semantically rigorous constraints for spatial information are potentially far-reaching. As each of the logical rules or rulesets can be uniquely identified, this technology opens up the possibility of establishing standardized spatial semantic models within specific application domains, such as link-node in transportation networks. Once such rulesets exist, data providers may use them to certify their data against the standard model and publish quantitative quality metrics within a geospatial metadata portal. As semantic metadata resources grow, conformance to a particular domain model, at a given level of quality, becomes yet another input field in a metadata search for data consumers to hone in on relevant data sources for reuse in novel applications.

Further work in this area is needed to examine the relationship between GML application schemas and the underlying, abstract conceptual models (ontologies) against which the rules are specified. Eliminating or automating model transformation steps will be key to improving the usability of an on-line quality assessment service and limiting the proliferation of terms that would otherwise ensue. The number and value of standard rules domains (e.g. link-node transportation networks, cadastral land management), the mechanisms for industry standardization and provision of a set of canonical validation rulesets for enhancing semantic interoperability between datasets (with distinct application schemas) should be explored.

5. REFERENCES

Baader, F., 2002 (ed.). Description Logic Handbook, Cambridge University Press, Cambridge

ISO 19115:2003. Geographic Information – Metadata

ISO 19125-2:2004. Geographic information -- Simple feature access -- Part 2: SQL option

ISO 19139:2007. Geographic Information – Metadata – XML schema implementation

OASIS, 2007. WS-BPEL 2.0 - Web Services Business Process Execution Language. http://www.oasis-open.org/committees/download.php/22036/wsbpel-specification-draft%20candidate%20CD%20Jan%2025%2007.pdf.

OGC, 2004. Web Feature Service (WFS) Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=8339

OGC, 2005. Catalogue Service. http://portal.opengeospatial.org/files/?artifact_id=5929&version=2

PIRA, 2000. Commercial Exploitation of Public Sector Information

W3C, 2004a. Web Ontology Language (OWL). http://www.w3.org/TR/owl-features/

W3C, 2004b. Semantic Web. http://www.w3.org/2001/sw/

W3C, 2003. SOAP – Simple Object Access Protocol. http://www.w3.org/TR/soap/

W3C, 2004c. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/