Open Geospatial Consortium, Inc.

Date: 2008-09-12

Reference number of this document: OGC 07-160r1

Version: 0.1.0

Category: Discussion Paper

Editor: Pete Brennan

OGC® OWS-5 Conflation Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Preface

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Forward

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Contents

Page

Figures

OGC® OWS-5 Conflation Engineering Report

1 Introduction

1.1 Scope

This OGC Engineering Report describes the process of conflation, outlines a framework for conflation and conflation rules services within a service oriented architecture, and describes the implementation of conflation services during the OGC OWS-5 testbed.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Pete Brennan	Northrop Grumman Corporation
Chris Evans	1Spatial Ltd

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
Feb 26 2008	OGC 07-160r1 v. 0.0.2	Stefan Falke	9, 10	Added conflation rules service and OWS-5 descriptions.
Mar 12 2008	OGC 07 160 r1 v.0.0.3	Jim Ressler	7, 9 (10)	Updated business rule definition and web service interface description to align with NGA conflation working group.
Mar 26 2008	OGC 07 160 r1 v.0.0.4	Chris Evans	9	Inserted new section 9 with sample implementation of Conflation service
Apr 18 2008	OGC 07 160 r1 v.0.0.5	Chris Evans	9	Proper merging of changes in v0.0.4 & some minor editorial changes following review
Aug 13 2008	Same	Carl Reed	Various	Make ready for posting as a DP

1.4 Future work

Improvements in this document are desirable to continue to evaluate and provide guidance for the implementation of conflation and conflation rules as web services.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 06-121r3, *OpenGIS® Web Services Common Specification*

OGC 05-007r6, *OpenGIS® Web Processing Service*

OGC 07-138r1, *OWS-5 GeoProcessing Workflow Architecture Engineering Report*

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

4.1
Conflation
The process of unifying multiple separate sources of data into one integrated all-encompassing result.

4 Conventions

4.1 Abbreviated terms

BPEL	Business Process Execution Language
GPW	GeoProcessing Workflow
WFS	Web Feature Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XML	eXtensible Markup Language

5 Conflation Overview

Conflation is the process of unifying multiple separate sources of data into one integrated all-encompassing result. While applicable to raster sources and vector sources, the scope of this document is limited to geospatial vector features, that is, features represented by one or more coordinates and accompanying metadata and attribution.

Digital representations of geospatial features (such as roads, rivers, and forests) vary between databases, and while conflation processing is akin to forming a union between databases, differences in how features are represented in each database makes forming an integrated result challenging. When features from different sources are superimposed on-screen, they will typically differ in alignment, precision, location, completeness, and potentially in geometric representation as well. Not initially visible are differences in attribution and topology. The core of the conflation process is identifying and associating the common features across multiple data sources in spite of all these challenges, reconciling the differences between them, and constructing one integrated result. The integrated result should contain all the unique features and all the unique attributes from the sources being processed, the "best" (see below) geospatial representation of features deemed to be common, the combined attribution for features in common, and where values differ for attributes in common, the "best" value likewise must prevail as well.

Given the variability of sources that may be conflated, the reasons why they are integrated, and the particular origins of the feature data, the definition of what "best" means may vary, sometimes on a case-by-case basis. Conflation systems must permit users to configure a rule system and define what "best", "highest priority", or "winner" means based on their own needs and domain expertise.

The next section illustrates a sample conflation scenario that sets the initial context for conflation systems. As this document continues, the subsequent sections describe, respectively, the desirable functionality of a conflation solution, considerations and hurdles that complicate the process, and foremost, the desirable capabilities for rule systems and processing options. These topics are approached from the perspective of defining an operating framework for what conflation processing should produce, and intentionally does not define approaches, algorithms, methodologies or otherwise unnecessarily impose constraints on current and future solutions to automated conflation and data integration systems in general.

6 Conflation Framework

This section illustrates a sample conflation as a means to set the context of discussion and defining an operating framework.

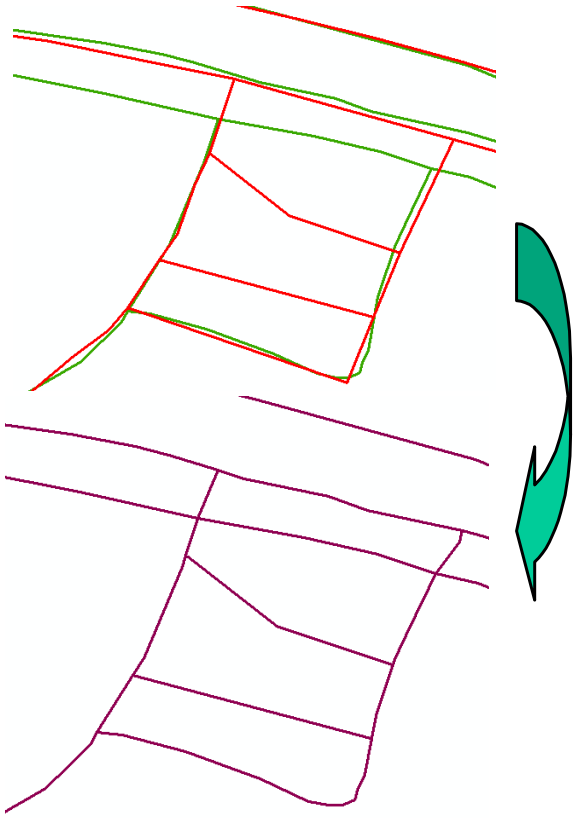

Figure 1 - An example showing conflation of two sets of line features (top) into a unified set of features (bottom)

Figure 1 illustrates the contents of two data sources superimposed over each other, from an orthogonal view. One source is drawn in red and one in green. These are both differing representations of road features, where features in each source are represented using line geometry. As shown, there are some red features without a corresponding green and vice versa. Many features, however, are represented in both databases, albeit a little differently in each. The integrated result, shown in purple, is the combination of unique red and green features blended with the "best" representation of the features in common. In this example, the green features were denoted as being the best in all instances. The determination of best is one of the many functions of the business rules.

The integrated result also shows topology handling issues, such as preserving connectivity (vertical red to green, upper center right) and resolving potential overshoot issues (unique red feature, middle center right).

There are many aspects of conflation processing not addressed in the above illustration, such as attribute handling, vicinity, and many other challenges. Those that pertain to the overall conflation framework are discussed below; business rules and options are discussed in the subsequent sections.

6.1 Metadata

6.1.1 Lineage

The origins of each feature, sometimes referred to as the lineage or pedigree, must be retained. For example, although the integrated result is shown in a single color in the above diagram, the notion that a feature originated from a red or green source should be stored with each feature. If the data sources that are input to conflation lack this information for some or all features, the user should be given the opportunity to specify such information at the start of the conflation process. At a minimum, the origin requires two pieces of metadata, a date and a name. The name may be a recognized product name (e.g., FFD, MSDS, ADRG, TIGER), collection (e.g., IFSAR, Quickbird, GPS), or other identifiable name. The date should refer to the date the feature data was originally collected. In the case of formally produced products, such as legacy NGA products, the formal production date or source date may be used. Together, this metadata uniquely tag the feature, allowing the lineage to be preserved, and should be associated with each feature in the integrated result.

6.1.2 Absolute Horizontal Accuracy

Geospatial feature is collected via numerous mechanisms of varying fidelity, both in the resolution and also the preciseness of given coordinates. Absolute horizontal accuracy (AHA), measured in meters, is the spatial accuracy of the vector representation, relative to its true location (with an associated 90% circular error probable (CEP) region confidence). For example, a GPS representation of a feature would have a very low number and less accurate products would have higher numbers. AHA is generally provided by the data producers and is typically constant for all features of a given product, although it need not be. If the data sources that are input to conflation lack this information for some or all features, the user should be given the opportunity to specify such information at the start of the conflation process for the sources being processed. This metadata should be associated with each feature in the integrated result. For example, the green features in the conflation example will retain their original AHA value.

6.2 Feature Type

The conflation example illustrated two sources where all features were stated as being roads. In the most general case, any given geospatial feature source should not be assumed to contain only one feature type. For single feature sources, the user should be given the opportunity to specify such information at the start of the conflation process for the sources being processed. Feature type information, be it road, lake, park, pylon, or any of over a thousand others, is best stored as a code with each feature, much like product source name and date would be. Data providers may choose to use any one of a number of standards to code their feature data, and conflation systems need to be cognizant of and handle the differences. This will be covered in greater detail in the business rules section of this document.

6.3 Alignment

The conflation example illustrated features that were in close alignment, where the red and green features were nearly overlapping. Such close alignment represents the optimal conflation situation, and is aided by processing sources with absolute horizontal accuracy values that are either small numbers or that are reasonably close to each other. The more common case of misaligned data represents a challenge, more so in the absence of attribute data, that conflation systems must address.

6.4 Competition

The conflation example illustrated a desirable processing scenario in that it lacked competition. Competition refers to the choice of alternatives considered when determining the features in common across data sources. For example, each green feature corresponded to either zero or one red feature, as opposed to the more challenging case of considering multiple features.

6.5 Geometry

The conflation example illustrated two sources where all features were of line geometry. In the most general case, geospatial features could conceivably be any geometry type, where point, line, and area (polygon) geometries are most common. Geometry may potentially vary with a single source, but more importantly, may differ between sources, representing situations where various sources represent the same feature differently. For example, a certain bridge may be represented in one product using point geometry and in another as a line, and in a third as a thin polygon. In addition to like-geometry processing scenarios (e.g., point-to-point, line-to-line, area-to-area), processing mixed-geometry scenarios (e.g., point-to-line) may be required.

Conflation systems either need to account for these differences, or publish their inability to handle certain formats. Additionally, conflation systems should provide an option that would restrict processing to specific geometries, such as “like-geometry only”.

6.6 Coordinate Specification

Coordinate specification refers to issues associated with potential differences among geospatial data sources regarding variations in coordinate system and, separately, dimensionality. The WGS-84 standard is common, where a coordinate’s X and Y components are longitude and latitude values expressed in positive and negative decimal degree numbers. However, hundreds of possibilities exist, including other decimal degree-based standards, but also UTM, and State Plane. Conflation systems either need to account for these differences, or publish their inability to handle certain formats.

Dimensionality refers to the richness of each coordinate. Coordinates may be strictly two-dimensional where they have an X and Y component only, have a measured value (M) component and also an elevation (Z) value. Conflation systems must be cognizant of the differences and should preserve the maximum richness when generating the integrated result.

6.7 Duplicate Features

Individual data sources may contain duplicate features within the identical feature type. Two types of duplication are possible. First, features may be exact duplicates where they have the identical coordinates and identical values for all attributes. Second, features may have identical coordinates, but have differing attribute values. The first case represents truly superfluous features, but both cases should be accounted for.

6.8 Schemas

In addition to geospatial coordinates, features often have associated attribution, although this is not always the case. An important issue that impacts conflation processing is the potentially differing schemas of each source (in whole or in part). The term "schema" refers to the collection of attributes, attribute data type, units of measure, and domain (valid values). Schemas may or may not follow a given standard. With the intent on generating a truly best-of-breed result, the schemas from all input sources should be preserved when forming the output schema. However, there are issues in forming such a union. For example, attributes in common to both schemas may have different data types, common string-valued attributes may have different lengths, common attributes may not be representing the same quantity (semantic difference), and common attributes may have different units of measure. Also, attributes not in common (by name association) may nonetheless be holding the same information. These attributes could additionally have the same issues as attributes named in common.

One approach for handling the issues with different input source schemas is to designate one of them as the "target" schema, where such designation implies that it is the desired schema because, for example, it is a gold standard of an underlying database or uses preferred attribute types. The target schema thus becomes the baseline for the integrated conflation result and other schemas are fitted to it. In the case of differing data types for a commonly-named attribute for example, values would be converted to that of the target schema.

6.9 Feature Association

Associating feature across multiple input sources can take many forms, including combinations of full and partial feature matching coupled with either single feature matching or multiple feature matching. Full feature matching is where the association between sources is made on complete features only. Partial feature matching is where only a segment, section, or subset of a feature in one source is associated to a feature in another source. Single feature matching means that features are matched exclusively on a

one-to-one basis, in whole or in part. Multiple feature matching means that any given feature in one source matches to several (typically adjacent) features in another source.

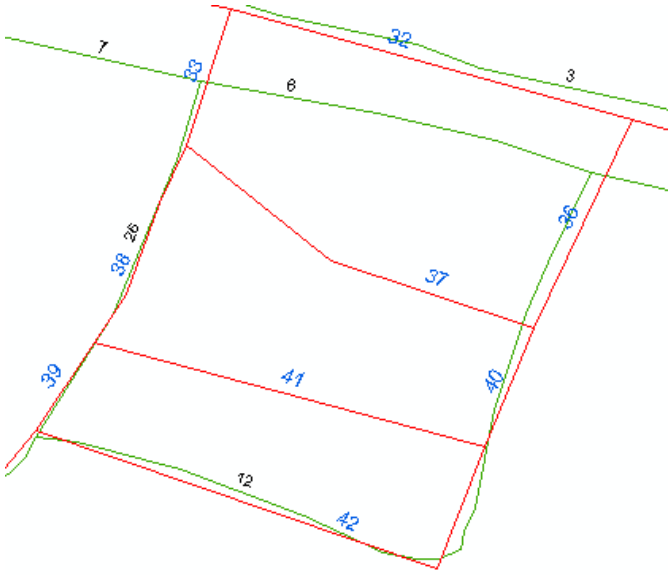

Figure 2 - A more detailed view of the source features for conflation

Figure 2 illustrates a multiple feature partial and partial match between a green features and three red sources features. Each source contains features unique to themselves, such as green features 6 and 7 and red features 37 and 41. An example of corresponding representations of the same feature is red features (numbered in light blue) 36, 40, and 42 and green feature 12. Red features 42 and 40 are fully matched, but only about 75% of feature 36 is matched by green feature 12.

6.10 Logs

Conflation systems should generate a log file that details the processing scenario, suitable for display in an Internet browser (e.g., HTML).

6.11 Errors

The complexities involved in performing an automated conflation may result in errors, both that the system was able to self-detect and report in a log file, and those it does not know it made. This latter collection of errors may be errors of omission, where truly common features were not associated at all or only partially associated, and where an association was made that should not have been (erroneous association). These types of errors may only be detected by post-conflation processing, including visual inspection and/or by using automated anomaly finders or other automated geospatial reasoning tools.

7 Conflation Business Rules

Selecting the "best" of all sources is not a one-size-fits-all problem, subject to the actual feature data being processed, user knowledge and expertise, and other criterion. As such, a comprehensive set of business rules is necessary to control aspects of the process such as feature prioritization, attribute handling, coding standard conversions, and more.

The collection of conflation business rules are functionally tied to the conflation process. To retrieve and update rules within a rules template, a specification of specific business rules is needed that addresses the capabilities described in this section. To maintain independence from a specific implementation of the business rules by a tool or service, the following families of rules are defined.

- Similarity – *feature similarity* deals with the many names for the same or similar types of geospatial features and *schema similarity* is used to define equivalent attributes between features, to include units, unknown and null values.
- Matching - determines how close the proximity, type, geometric representation and significant attributes of a feature need to be in order to potentially be the same feature.
- Selection - used to select matched features and merge them in terms of attribution and geometry.

The following figure illustrates the composition of the families of business rules:

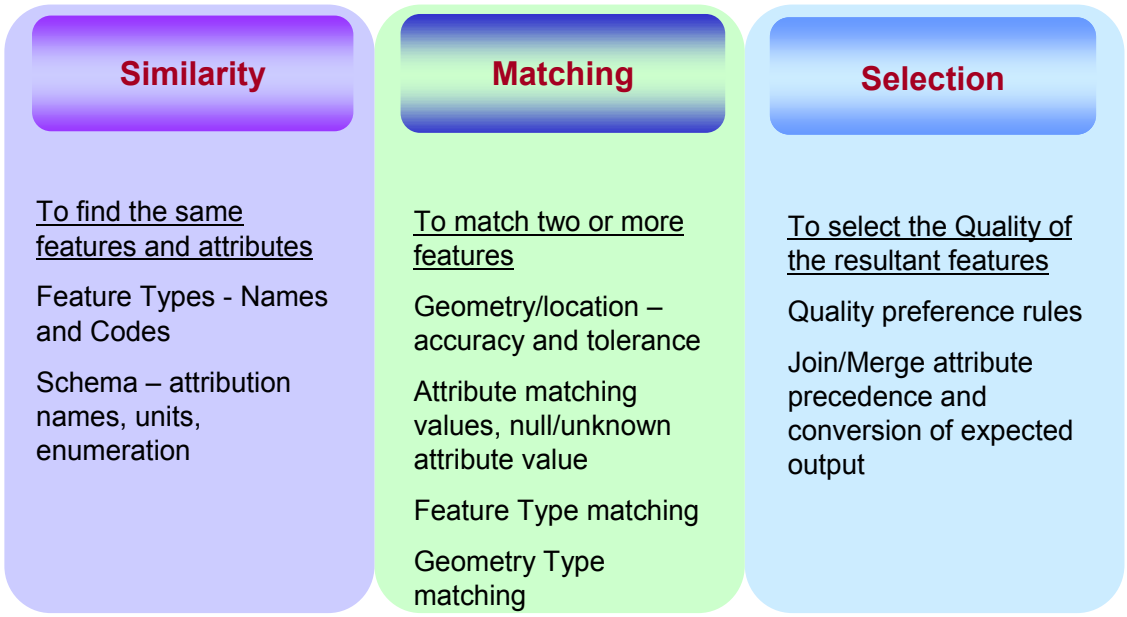

Figure 3: Business Rule Families

7.1 Similarity Rules

7.1.1 Feature Similarity

Feature similarity rules establish which feature types (e.g., road, lake, pylon) should be considered "similar" or "related" to one another. These settings allow non-exact feature code matching across multiple data sources, such as "road" and "interchange". The assumption is that these settings apply to a common coding standard, and features coded in a different standard would be mapped.

Many data products, even those adhering to the same standard often use different feature codes to label what turns out to be the same feature. These differences may exist because of particular feature code subsets used by a product, from cataloging features in varying degrees of specificity, such as "river" versus "inland water", and from normal evolution of the coding standard.

Not allowing feature type flexibility can cause erroneous results by not enabling features to be considered for association that really need to be. Users with domain expertise of the input sources being processed may elect to follow a strict "exact-match only".

7.1.2 Schema Similarity

Schema from different feature products use various names to describe the same attribution. The schema of each feature type is often defined in the data itself, though it may be described in a formal model (e.g. XML schema, DDL, entity relationship, UML model). In addition to the feature type similarity, to accurately conflate two feature types, the similarity of feature schema is defined in the business rules. These rules can be expressed in terms of mapping between attribute names. For example, a schema mapping between could be defined between a point feature HEIGHTAGL expressed in feet and a feature database HGT expressed in meters. The schema mapping could apply to specific feature types or all feature types within a source.

A complete similarity mapping includes the attribute type (such as float, integer, string, date), a textual description, the unit of measure (e.g. feet | meters, pounds | kilograms), the range of minimum/maximum values, enumerated values, and null values. The schema similarity rules are used in the attribute constraints and Selection rules for specific portions of the conflation processing.

7.2 Matching Rules

7.2.1 Attribute Constraints

In addition to considering (or excluding) potential cross-source feature association based on location and feature code similarity, attribute constraint rules apply additional "differential" criteria to include or exclude features from potential cross-source feature association, based on one more or common criterion. These rules would permit uses with domain expertise on the sources being conflated to further guide the feature association/matching process. For example, one criterion may be that communication

towers being associated across sources cannot represent the same tower if their heights differ by more than 30 meters. Similarly, for buildings, the building function or type may usefully factor into the matching process. Such rules are envisioned as being most helpful in densely populated regions, and for features of point geometry.

Similar to the source product priority rules, attribute constraint rules may potentially be quite complex and be based on the specific sources being conflated, the specific feature types of the features that were matched, the geometries of those features, or a combination of any of these.

The same as for priority rules, an important issue regarding building such rules is (a) determining which attributes in each schema refer to the information that is required, and (b) performing the comparison on attribute values that are legitimate. Beginning with the former case, the rule system must be cognizant of the schema to which it refers, regardless if rules are written generically (independent of any schema), or written using specific attribute names, in either case domain expertise is required. For example, the FACC/DFDD standard attribute ZV2 refers to elevation while HGT is the height above surface level. A rule written using a generic "height" must be internally mapped to a ZV2 in some instances, and perhaps HGT in others. Conflation systems need to specify if they are able to handle comparison of attributes that are differently named.

Regarding legitimate attribute values, there is a possibility that attributes written into rules do not exist in the particular schemas from the sources being processed, that the value for the attribute on a particular feature is empty or null, or that a value is not a legitimate one, such as a negative height. Rule systems need to safeguard against performing calculations with missing or illogical attribute values.

7.2.2 Unknown Attribute Value Settings

Normally, when differing values are found for common attributes between matched features, a singular value is selected based on the attribute priority rules. However, when this singular value is "unknown" in nature, it is often not desirable to use it over a "known" value even if the known value is from the lower priority feature. Also, the evaluation of priority rules and attribute constraint rules require legitimate attribute values. The unknown attribute value settings may be encapsulated in the schema similarity rules, or distinctly defined for matching purposes.

The unknown value rules enables users to set which value(s) for a given attribute should be treated as "unknown". The meaning of unknown varies on an attribute-by-attribute basis, depending on the conventions of the data producer. The unknown values rule may consist simply of a list of unknown values for each attribute.

7.3 Selection Rules

Rule systems must provide the ability to create and maintain a custom set of conflation rules for prioritizing which source product's geometric representation is "best" and which attribute values are "best" when features are deemed common across the input sources. Geometry is independently controllable from attribution. Having separate rules removes

preconceived notions that geometry must come from one source and the attributes must be from the other.

While priority rules should allow for an unequivocal (static) specification of one source over another, the true power is in creating rules that allow dynamic comparison based on attributes or metadata information associated with a feature. One default rule might be that the best or winning feature for geometry is the one that is the most spatially-accurate, for which the AHA values (or equivalent) can be compared. When features deemed in common having differing values for the same-named attribute, the default rule might be to use the most recent value. Here, the product date metadata would be useful. What is common to both cases is the need for performing a comparison of attribute values, and the need to either resolve a tie, such as having a date comparison evaluates to "equal".

Prioritization rules may potentially be quite complex and be based on the specific sources being conflated, the specific feature types of the features that were matched, the geometries of those features, or a combination of any of these.

7.3.1 Quality Rules

When comparing two or more matched features, the selection of attributes and geometry for merge is dependent upon several factors at the direction of the user. These factors are generally called Quality rules because they specify the relative quality of one matched feature vs. another feature. For example, the quality selection can be dependent upon the currency (source date), the type of source from which the feature was derived, the horizontal accuracy of the feature and the type of geometry. Using a comparison of matched features, the preference for a logical combination of quality factors in determining the feature is used to select the "winner" for merging attribute and geometry.

7.3.2 Join Attribute Value Rules

Rule systems can provide the ability to override the usual prioritization rules employed on matched features and specify attribute names for which differing values from associated features across databases should have both values retained when differing values are found for common attributes between associated features instead of using a singular value based on the attribute priority rules. Normally, when differing values are found for common attributes between matched features, a singular value is selected based on the attribute priority rules. The attribute value join rules specify those attribute names where both values should be retained. Such is envisioned useful for feature names, textual descriptions, or any attribute where multiple values are legitimately possible.

8 Conflation Options

This section discusses two processing controls that are defined as options as opposed to rules.

8.1 Match Like-Geometry Only

Specifying this option instructs the conflation system to only consider associating (matching) features if they have the same geometry (e.g., point, line, area). This additional constraint is useful in cases where the input sources consist of multiple geometries.

8.2 Vicinity Tolerance Override

The notion of feature vicinity, or the proximity of one feature to another, such as the red and green features in the conflation example, naturally factors into the conflation process. In those instances where the expert user has analyzed the scenario and performed some distances measurements between the products, this option can be used to limit the cross-source vicinity region, and override default algorithms inside conflation systems. The caution is that if the value is set too small, the correct feature match may not be possible.

9 Sample implementation

As part of the OWS-5 test bed a sample set of conflation rules were to be implemented to prove the concept of rule-based conflation. This was implemented using 1Spatial's Radius Studio product as used to implement the TQAS in OWS-4. Details of how Radius Studio is used to implement quality rules is detailed in [2], and it is recommended that this is reviewed before continuing with this section, as the basic principles are not repeated here.

The sample data for the conflation process consisted of a 'baseline' Mission Specific Dataset (MSD3) with an Aeronautical profile, as well as a DVOF Vertical Obstruction (VO) dataset to represent a set of updates to the baseline. The intention was to process these directly from GML3.2.1 , but owing to limitations of the software (see below), the examples were run against a shapefile profile.

This section will discuss the implementation of the above business rules and process in this rules-based environment.

9.1 Outline of a rule-based system

In OWS4, Radius Studio was used to execute quality assessment rules and report on exceptions to these rules.

The outline mechanism for Radius Studio is that data is read form any number of data sources into an internal cache. Rules are executed on this cache, and any exception reports generated. Finally the data may be written back out to external data targets.

Another feature of Radius Studio is the ability to perform *Actions* based on business rules. The rules are used to define features that match certain business criteria, and the actions modify the data in the cache before potentially writing out to the external data targets. This is the process used to perform conflation, where the actions are used to

copy data from source datasets into target datasets, before these target datasets are finally exported.

9.2 Process Outline

This process considers the conflation of two datasets with dissimilar but known schemas. One is considered to be a baseline dataset, consisting of the majority of the features, and generally made up of static data, that generally does not change over time. The other is considered as an 'update' dataset consisting of either data that has changed in the baseline, or data that is to be added to the baseline.

Despite the term 'update' it is not necessarily the case that data is always replaced in the baseline by update features, for example in the case where the baseline is newer or contains more accurate data than the update.

The output of the process is a third dataset, in the schema of the baseline dataset, where all features 'duplicated' in the two datasets are removed. The definition of 'duplicated' depends on the business rules described in the previous section and implemented below. The fact that the update dataset is in a different schema from the baseline and resultant dataset adds to the complexity of the process in that a schema mapping also needs to be performed.

To clarify terminology, when a dataset is read into the Radius Studio system, that internal cached storage is known as a datastore.

The overall process can be described as follows:

1. Read the baseline dataset into the system and store in a results datastore
2. Read the update dataset into the system and store in the same results datastore (note at this point the results datastore will contain the schema for both baseline and updates dataset.)
3. Process all features from the update dataset in the results datastore
    a. Apply business rules to find matching features in the baseline data
    b. Apply business rules to determine action to be performed on the update feature (delete, merge with baseline, add etc)
    c. Apply appropriate action
4. Export the results datastore to external dataset.

The principle of the process is that a dastore is able to temporarily load both baseline and update datasets in a single internal structure with a mixed schema. The system provides the ability to apply a schema mapping during the import process (providing a mapping between dataset namespace and datastore namespace) to ensure that schema clashes are avoided (e.g. where you have classes with the same names).

The example examined in the testbed had no name clashes, but in the use-case where baseline and update datasets have the same schema, a mapping would have to be set up for one of the datasets to ensure that the data for the datasets would still be separated out to separate classes. The reason for this is that the next step is to perform a scan over the update data items only.

The export process from the internal datastore is the converse of the import process, and again a mapping can be applied. Generally though, this will not be needed as the export format will map directly to one of the input datasets (in this case the baseline dataset). The export process is set up to only allow the export of classes that correspond to the baseline classes, and ignore those in the update dataset.

9.3 Feature mapping

One of the requirements of the process is to map the features in the VO dataset in a DVOF profile into corresponding MSD3 features. This mapping was provided as a spreadsheet [3] providing a 1:1 mapping between DVOF feature codes to MSD Feature, and defining attributes. This mapping is not in itself a complex task, but can be an intensive process to set up. There are a couple of mechanisms that could be used to perform this mapping:

9.3.1 Requirements

In the VO data schema, the data is held as 3 feature classes, representing points, lines and areas, with a single attribute defining a feature type. These features need to be separated out using this feature code to one of a number of MSD feature classes based on this feature type attribute.

Also there is the issue of name mapping and value translation of attribute values. As noted above, the FACC/DFDD elevation attribute ZV2 needs to map to the MSD3 Height AboveSurfaceLevel, but as was noticed, there is also a conversion between imperial and metric measurements.

This mapping of feature and attributes, complicates both the rule writing and the data transfer aspects of a conflation solution. There are two basic methods of handling this situation.

9.3.2 On-demand mapping

In this mechanism we are accepting the fact that there are two separate schemas in the conflation process and use these schemas in defining the rules and actions. That means that there assumptions are made on the schemas that are conflated. i.e a ruleset is created

that will handle the conflation of an MSD baseline with a FACC/DFDD update. This ruleset will not handle any other configuration.

There will therefore be specific rules for example that try to compare a VO_POINT (Point Vertical Obstruction) features containing a ZV2 height attribute with a MSD Pylon feature with a HeightAboveSurfaceLevel attribute.

You also have specific actions that map attributes between VO_POINT and all the possible classes that points can be mapped to (pylons, aerials etc, as well as actions to update the height attributes of building areas based on point heights).

This was generally the approach implemented during this testbed, but does have obvious limitations.

The permutations of mapping VO points to MSD point features and VO lines to Line features is not insignificant. There are 175 separate feature codes in the DVOF schema, but many of these can be grouped into similar feature types (e.g. there are 28 separate 'building' feature types which end up in the same feature class in MSD) but again each of these themselves may use different attributes to define a sub type of that feature class.

This complex mapping of VO feature code to MSD class/MSD featureFunction is difficult to implement in a rule-based system such as Radius Studio, without resorting to a series of if/then statements, and specific assignment statements for each attribute, and this is detailed below.

Radius Studio has the ability to use custom developed 'built-in' functions that are extensions to the standard functions available. These built-in functions are developed as separate Java classes, but may be used in rules and actions in the same way as normal functions. A built-in could be developed to read in this mapping of VO feature codes to MSD classnames, attributes and values from say an external xml mapping file, and return these to the rules. This development was not done as part of the test bed as the desire was to try and investigate the mechanism with an off-the-shelf product, but could be considered in future work. Similar builtin functions could be developed to provide a generic attribute mapping capability, greatly simplifying the rulebase, but at a potential cost of losing some flexibility in the rule development.

9.3.3 Ontology

An alternate approach to the mapping issue would be to translate both incoming schemas to a common conceptual schema before the conflation process is attempted. This is most easily accomplished by the use of ontologies to map both the VO and MSD data to a common schema. With identical conceptual schemas, the issue of mapping is no longer a major issue, and the rules can be more easily expressed in terms common to the models. The inverse mapping from the conceptual model to the physical MSD schema can then be performed during the export process again using the ontology. Radius Studio is capable of handling ontologies, but no suitable ontology definition for the schemas were available during the testbed. Again this could be a suitable area for future research.

9.4 Business rules

As described above, the process of applying the business rules to produce a conflated dataset involves iterating over the update features. The rules are therefore written for the feature classes in the update dataset only, and they attempt to match features in the baseline dataset only.

It is very difficult (unless ontologies are utilized) to make a conflation ruleset schema independent. This is especially true if the conflation takes place over datasets with different complex schemas.

A typical business rule for conflation therefore has the following structure:

Rule for update feature class:

If there is at least one feature in a specified baseline feature class that:

- Passes a geometrical 'similarity' match
- Passes an attribute based feature match

→ Report the update feature is a duplicate of one in the baseline

else for all features in the specified baseline feature class that:

- Just pass a geometrical match with the update feature

If the update feature data is 'better' than baseline feature

→ Copy attributes from update feature to baseline feature in results

→ Report the update feature has been merged

else

→ Create a new feature in results based on update feature

→ Report a new feature has been created

There was a question as to whether the process above should delete the update features from the datastore as they are processes successfully (i.e. wherever there is a report above). As the update features are never exported, it was decided for the testbed not to remove these features. It can be foreseen though that deleting those items that have been successfully conflated could be useful at a later date when full reporting of conflation issues is introduced as the data from the update dataset causing the issues will still exist in the results datastore for analysis.

9.4.1 Feature class matching – Rules

As detailed in section 7.1 the first stage of conflation is to identify similar feature classes in both baseline and update datasets. This identification of similar feature classes occurs on two levels:

1. Identification of feature classes that are considered to be ‘similar’ in the update dataset

2. Identification of matching ‘similar’ feature classes in the baseline dataset

The DVOF dataset used an attribute to identify a feature type, therefore it is fairly easy to express similar update features by matching this feature type attribute as shown in the following partial rule:

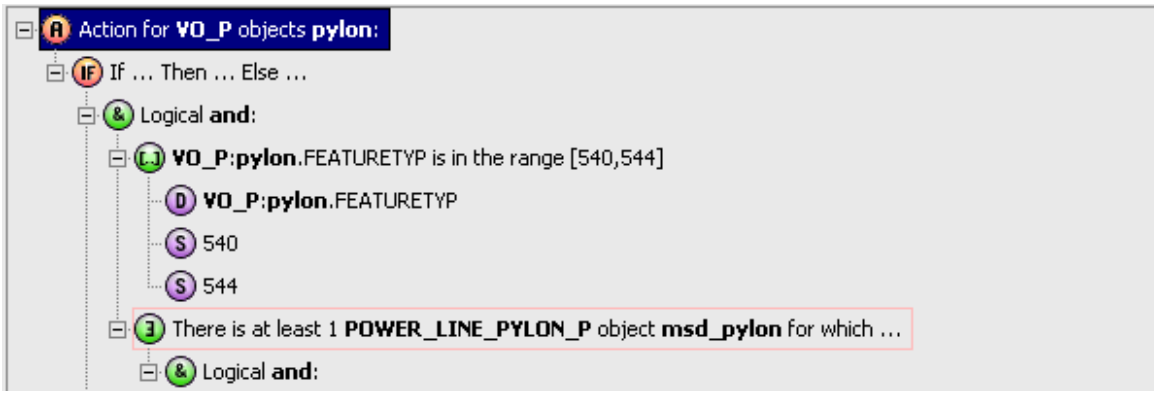

Figure 4 – Feature matching rule segment

Here VO_P is the feature class for update points. The rule identifies them as pylons by them having a FEATURETYP attribute in the range 540->544. This could also be expressed as a sequence of if..then constructs, if the feature codes were not consecutive. For each pylon in the update datasets, the rule then goes on to identify the matching feature class in the baseline dataset.

There is no reason in the above not to match multiple feature classes in the baseline dataset. This was not implemented though in the testbed as a formal specification of matching feature classes between the VO and MSD data was not available. It is also possible, if the business rule required it, to extend the match in the baseline dataset by adding additional attribute matches.

9.4.2 Geometric matching

The matching geometrically of features is fairly simple for point data in that it is a direct within-distance comparison. An issue does rise though if the underlying rules engine lacks the ability to dynamically convert units of measure when specifying and executing rules. Business rules will normally be defined in terms of distances in meters, but as the data is defined in terms of WGS84, the default units of measure is degrees. Radius Studio for example currently only supports the natural units of measure for the datasets, although the support of other units of measure is in development. The conversion between meters and degrees is obviously dependant on the location of the dataset, so it is therefore difficult to define a ruleset containing geometric distance comparisons that can be used across worldwide data. For the testbed a value in degrees for meters distance

was calculated and used throughout, although this value would not be suitable for datasets at a significantly different spatial location.

There is also an open question as to what is meant by 'close' or 'similar' linear or area geometries? Several definitions of close may be used, for example –

- all points on one line are within a distance of the corresponding point on the other line
- all points on one line are within a distance of the other line.

Out of the box Radius Studio does not have such comparisons, although the second could be expressed in terms of one of the lines being wholly within a buffered version of the second line. Radius Studio as mentioned above does have the ability to support user-defined built-in functions, and a suitable built in function could be developed to support either of the above (or any other considered), which could then be used in the ruleset.

The rule used in this document considers the conflation of VO points into the MSD PowerLinePylon class, which is specified as a point, where a simple point distance defines matching features. Point data though may be used to update the height attribution of area features (for example Buildings), and an appropriate geometry match needs to be used for such matches. For example if the matching rule were that if a 'hotel' VO feature were within a MSD building, then the rule would be expressed as:

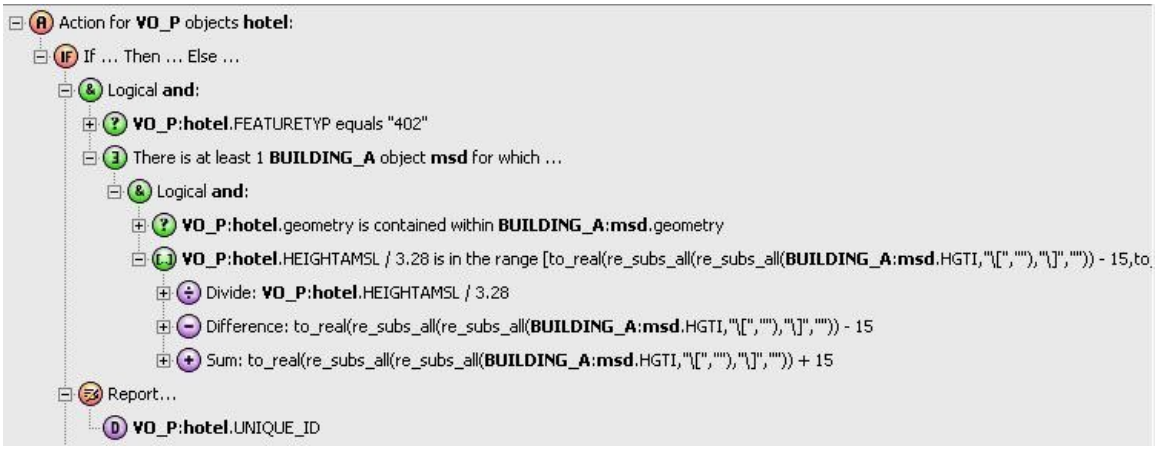

Figure 5 – Geometrical matching rule segment (containment)

whereas if the rule were that the 'hotel' feature had to only be within 25m of a Building, then it could be expressed as:

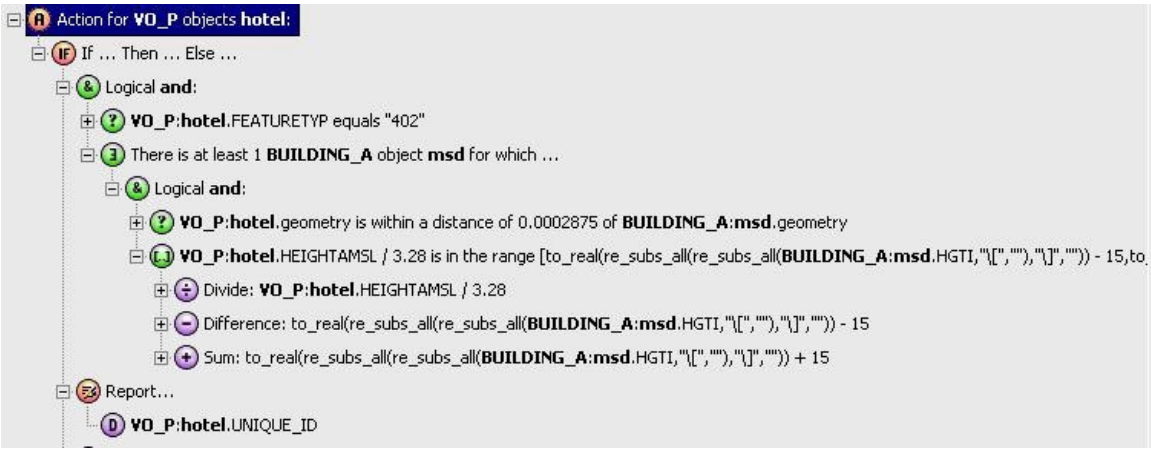

Figure 6 – Geometrical matching rule segment (proximity)

9.4.3 Attribute matching

As mentioned above, attribute matching requires domain knowledge to determine which feature classes, attributes, and tolerance of values need to match to determine identical features. A knowledge of the units of measure for each dataset is also needed.

Another thing to be aware of is the need to handle 'unknown' values and ranges. This is especially true if , as in case of MSD, a complex schema is mapped onto a simple format. Height values in the shapefile profile are passed as a string containing lower & higher values surrounded by brackets e.g. "[20][30]" to represent a feature with a minimum height of 20 & a maximum height of 30. The shape data used in the testbed though only had one height value surrounded with brackets to add to the complexity. This is actually much simplified if the GML schema had been available as this complex string mapping is replaced by the simpler to handle complex data types. It still means that all rules need to be aware that they may get a true value or an 'unknown' value, and this adds considerably to the complexity of the rules, as a clause needs to be added whenever an attribute that can be unknown is accessed, to define the action in the case of an unknown value. For simplicity in the rest of this document, this case is ignored.

If we combine the spatial matching with the attribute matching, we get rules such as the following to determine if two points are the same.

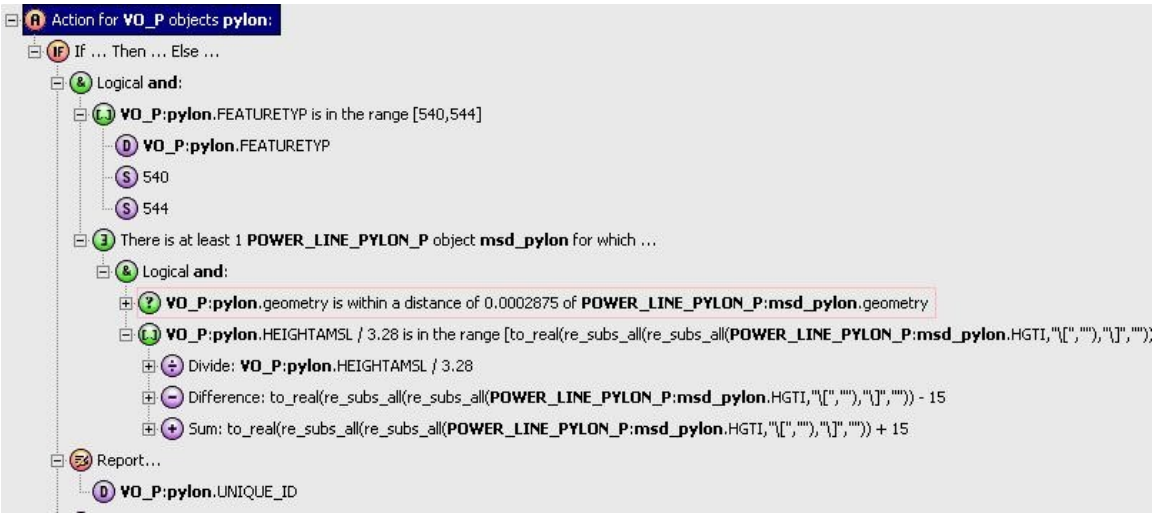

Figure 7 – Attribute matching rule segment

The rule is matching VO pylons with PowerLinePylons. If it finds any within a distance of 25m ground (0.0002875 degrees), and the height (converted from feet to meters) is in the range +/-15 m of the PowerLinePylon, then delete the VO version of the Pylon as it should no longer be considered, as it is the same as an existing MSD PowerLinePylon. The 'Report' line adds an entry to the log that the VO Pylon is the same as an existing feature in the baseline dataset and is therefore not conflated. This logging is discussed later.

Obviously further attribute matching rules could be added to the above by adding further comparisons, and here the ability for domain expert to express their own rulesets for others to use for conflation comes to the fore.

9.4.4 Prioritisation

If two features are identified as probably being the same, but with different key attributes (for the testbed this was taken as being the height above surface level), then a merging operation needs to take place. What data is retained depends on a decision as to what is the 'best' data. This could be based on any number of parameters, accuracy, quality, capture date, update date etc. For the testbed the capture date was used as an example to produce the following rule fragment:

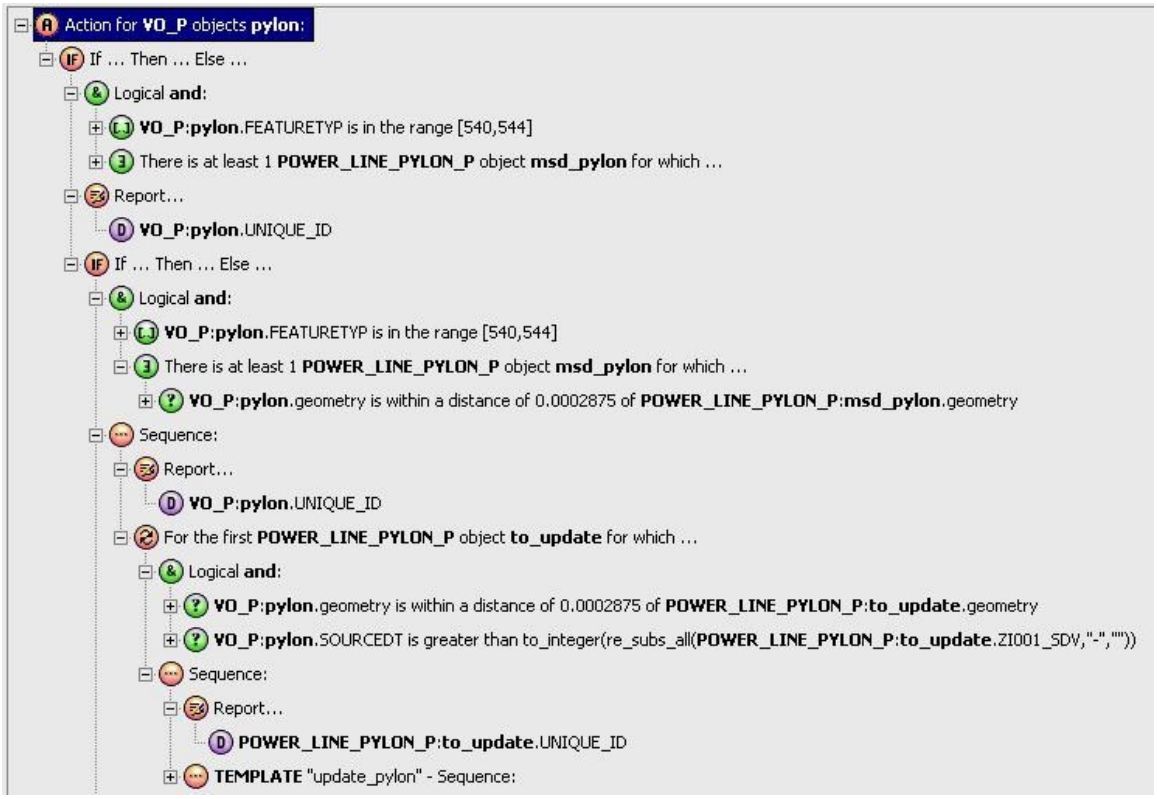

Figure 8 – rule segment showing prioritisation

The rule repeats the check for featuretype and distance to a PowerLinePylon, to identify there are MSD PowerLinePylons within 25m of the VO Pylon being considered, and then update the first MSD PowerLinePylon within 25m of the VO pylon where the source date is earlier than that of the VO pylon source date. The effect of this is that if the MSD source date is more recent than the VO pylon date, then no updates are performed and the MSD pylons are considered to be the most accurate. As mentioned the check on the source date could be replaced/added to as desired by the domain expert.

This updating of the MSD data is handled by a 'template' (in effect a sub-routine) in the action. Although not obvious from the above screen capture, the template is passed 2 parameters, being the source VO Point and a target MSD PowerLinePylon. A series of templates would need to be set up to handle each potential mapping from the three VO classes to the MSD classes, although there is scope to reduce the number of required templates if the naming of the attributes for target classes were common. A mapping template would have the following form:

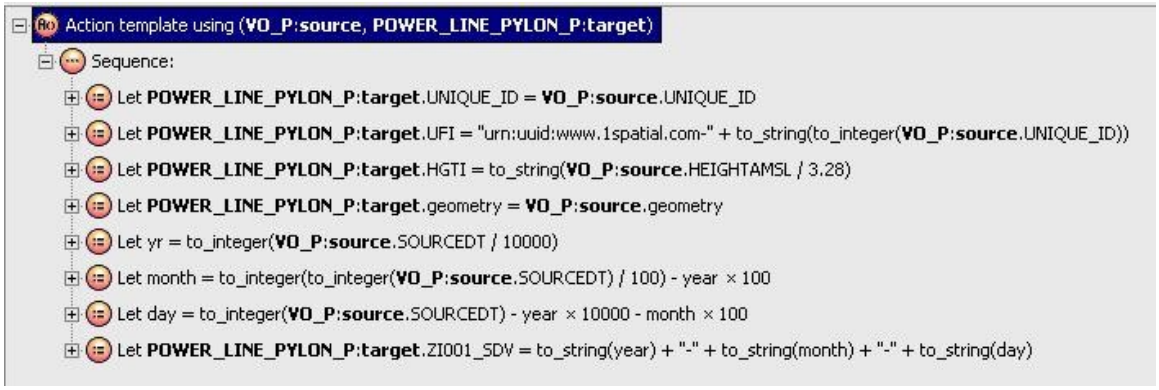

Figure 9 – Sample rule template

As can be seen it is a simple sequence of assignments from the source feature (in this case VO points) to MSD PowerLinePylons. This also demonstrates the added complexity of conversion between the two different date formats (yyyymmdd to yyyy-mm-dd)

9.5 Conflation Options

Sections 6, 7 & 8 express a desire for considerable user control and user parameterization of the conflation process. Some parameterization is obviously necessary, such as potentially modifying the tolerance values of geometry matches and certain attribute matches, such as height. There does need though to be careful consideration of how much user control is allowed for a conflation process, as this does significantly complicate the development of the rules. One of the advantages shown by Radius Studio over the last few years has been the ability to keep rulesets simple by maintaining only a core set of functionality, and keeping rules comparatively simple. Extending the complexity and customization of the conflation rules may involve the domain expert having to implement a significant quantity of custom code to extend this base functionality .

As Radius Studio is geared towards executing a set of pre-defined rules, which themselves contain domain-expert specified tolerance values, there is no simple mechanism for passing a significant number of end-user parameters to the rule execution engine. The mechanism explored during the test-bed was to generate a series of template rulesets, with a series of pre-defined tolerances that the domain expert considers to be applicable to the datasets in question. This though does not satisfy the desire mentioned above to include significant parameters, and this may need to be considered in the future.

Simple parameterization was performed during the testbed for defining the datasets that were passed into the process. The ruleset used for conflation can be held as an xml file and passed to the Radius Studio web service. A mechanism was set up to store the rulesets as a template xml file with suitable replaceable strings instead of physical filenames. The Conflation web service was passed the name of a template as well as the URIs for the baseline and update data. This performed a simple string replacement for

the parameters, and then passed the resultant xml to the Studio web service for processing.

9.6 Logs and Errors

Sections 6.10 and 6.11 discuss the ability to handle logging of what occurs during the conflation process and any errors that occur, and the passing of this log back to a client. In the rules shown above there a are lines of the form 'Report VO_P:to_update.UNIQUE_ID' . Not shown in the screen grabs is the ability to add a suitable textual message to each of these Report lines. These reports are returned from the execution of the action as an XML document of the form:

```
<?xml version="1.0" encoding="UTF8" ?>

  <Results finished="1205938728216" started="1205938727404">

  <Summary count="0" error="0" label="14" processed="425" total="425" type="Apply Actions">

     <Object class="VO_P" error="0" processed="425" total="425" />
      <ActionRef error="0" processed="425" ref_id="7f2ed4bec0a85ad9005efe79c70889b5" />
</Summary>
<Object class="VO_P" gid="14">
  <Attribute name="UNIQUE_ID">
              <Value>14.0</Value>
  </Attribute>
  <Attribute name="geometry">
     <MBR x0="74.88727904" x1="74.88727904" y0="39.11025425" y1="39.11025425" />
   </Attribute>
   <ReportedValues label="create new">
       <ReportedValue datatype="real" description="VO_P:pylon.UNIQUE_ID"
                                  value="14.0" />
      </ReportedValues>
    <ReportedValues label="date">
         <ReportedValue datatype="integer"
                                 description="to_integer(VO_P:pylon.SOURCEDT)"
                                 value="19940301" />
    </ReportedValues>
  </Object>

  <Object class="VO_P" gid="33">
     <Attribute name="UNIQUE_ID">
              <Value>33.0</Value>
     </Attribute>
     <Attribute name="geometry">
         <MBR x0="74.89009757" x1="74.89009757" y0="39.11308627" y1="39.11308627" />
    </Attribute>
    <ReportedValues label="create new">
         <ReportedValue datatype="real" description="VO_P:pylon.UNIQUE_ID"
                                 value="33.0" />
    </ReportedValues>
    <ReportedValues label="date">
              <ReportedValue datatype="integer"
                                 description="to_integer(VO_P:pylon.SOURCEDT)"
                                 value="19940301" />
```

```
        </ReportedValues>
        </Object>
etc
```

This retuned XML will also contain any errors found during processing.

9.7 Conclusion

Most of the business rules discussed above can be successfully implemented in a rules based system, assuming the rule author has a suitable domain knowledge of both datasets.

There is a significant risk that the sheer number of feature classes and the inherent complexity of feature class mapping would make maintenance of such a rule set difficult unless great care were taken to structure such rulesets.

Significant simplification could be obtained by the development of ontologies to produce a neutral format for both datasets and all rules and actions are implemented in terms of this neutral model. This significantly reduces the impact of feature mapping, and class specific mechanisms to handle attribute transfer.

The MSD format gives a highly flexible mechanism of defining data, but this flexibility does make defining class independent rules very difficult. Defining conflation rules on attributes and geometric aspects that are common across classes would significantly simplify the rule authoring/maintenance exercise.

As mentioned in the introduction, Radius Studio does not support the reading of GML3 data, and the complexity of the MSD schema challenged the GML3 reader that is in development as well as existing tools used by other vendors. This implies that the MSD schema is pushing the capabilities of systems being developed to handle GML3, and this may cause limitations in the number of systems that may be used to produce WPSs based on GML3.

The business rules for a conflation service details many cases where limitations and parameterization of the service should be published in the description of the service. This description for a typical conflation service could potentially be huge, and it is not obvious how such information could be usefully communicated to a client, and what form would be most applicable for potentially such a huge description. Returning it directly from the GetServiceDescription endpoint may not be sensible owing to the size and potentially complex nature of the documentation.

The most significant limitation at the moment seems to be the mechanism for parameterization of the conflation process. When the testbed was first scoped, a limited parameterization was proposed (allowing the changing of the baseline and update datasets). As the testbed progressed, analysis of the desired business rules indicated a much higher level of parameterization than originally expected, and the mechanisms are not really in place to allow this as rulesets are basically fixed by the domain expert. This increased customization could be the target of a future testbed.

10 Conflation Rules as Distributed Services

Conflation for selecting the “best” of all sources is not a one-size-fits-all problem, subject to the actual feature data being processed, user knowledge and expertise, and other criterion. As such, a comprehensive set of business rules is necessary to control aspects of the process such as feature prioritization, attribute handling, coding standard conversions, and more, and be applicable to the core conflation processing as well as pre-conflation data setup and preparation steps.

A service oriented architecture is well suited to support distributed conflation rules services. A rule “provider” could define custom rules and make those rules available to conflation services and applications. We envision two use cases for handling conflation rules services, 1) where the client only supports predefined sets of rules and 2) where the client allows user customization of the conflation rules.

In the predefined rules use case, the rules service will serve rules “templates” that consist of fixed settings for the rule categories. Each template will be described with its intended purpose (e.g., specific settings for conflating two particular datasets or all default rule category settings). The simpler client would then read these fixed rule settings into its conflation, fusion, or other service. The community would need to define appropriate rules templates.

For the more advanced client with the capability of customizing conflation rules, the rules service will serve rules query results, e.g., lists of feature codes from one dataset that are similar to a particular feature in another dataset (see feature similarity description above). Rules customization is useful for refining conflation processes. For example, in using the feature similarity rule category, a user may need to account for changes to a coding standard (i.e., DFDD) over time, the introduction of more specificity in the codes versus those more general in the standard, tolerance to account for more general feature code assignment that results from automated feature extraction programs, or any conversions that might be performed from alternate coding standards. For users knowledgeable about the data products they are conflating, they may elect to create or use a template with less flexibility (perhaps even "exact code only") - with the benefits of potentially faster execution time and accuracy since erroneous matches would be reduced.. Users may also desire to delete existing rules, add new rules, augment or delete attribution that is part of the rules, change how attributes are compared, etc. The rule schema itself would not be changeable by the user, but the contents could be, where additional rules are added, existing contents changed or deleted.

The customization process would result in new templates that others could leverage after they are uploaded to the rule server and properly cataloged (so others understand the customization). For constructing templates, each of the nine categories would require a query method to retrieve the complete existing rule set for a specified template.

Figure 4 depicts a conceptual representation of the template and customized rule interactions with the conflation rules service.

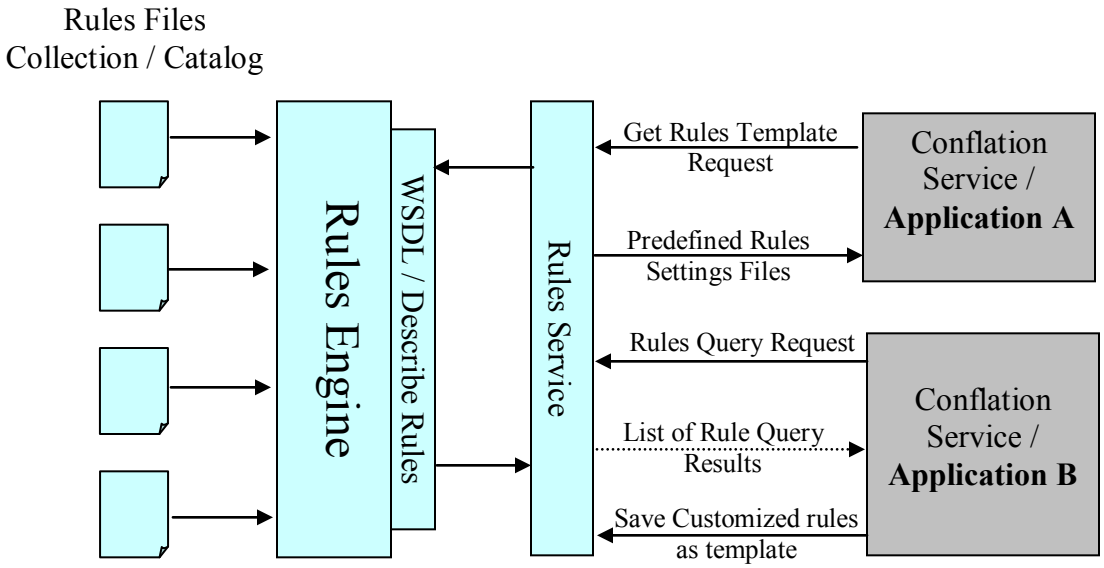

Figure 10: Rules service for template access and customized rule access.

In supporting rule customization in the SOA and web service chaining framework, methods for authenticating, tracking and maintaining user changes are needed.

One possible approach to implementing rules as services would be to use the OGC WPS standard with the various conflation rules processes defined, for example, as GetRuleX, SetRuleY, etc. The DescribeProcess would describe the inputs needed to retrieve the Rule as well as a description of the rule and its assumptions.

The architecture would resemble a distributed catalog of rules with multiple servers each serving their own rules or registering their rules into mediated rules catalogs.

10.1 Rules Web Service Interface

The business rule web service interface utilizes a schema with get/set operations for each family of business rule defined in section 7. The schema and operations will be defined in a WSDL document that is the subject of a future revision to this report. An XML rule language schema such as RuleML is a candidate basis for the rule schema.[1]

11 OWS-5 Conflation Services

This section describes the conflation services implemented conflation rules as part of the workflow during OWS-5. The description presented here is for purposes of understanding

[1] http://www.ruleml.org/

the conflation algorithm and the conflation data inputs (both feature data and rules) within the context of a broader conflation definitions and service framework. The OWS-5 GPW conflation workflow is described in OGC 07-138r1, *OWS-5 GeoProcessing Workflow Architecture Engineering Report*.

11.1 Extending the OWS-5 Workflow with Rules Services

While time and resources did not allow implementing rules services in the OWS-5 testbed, it is useful to describe how rules service could be used in a conflation workflow like the one implemented for OWS-5. Figure 11 depicts the conflation service flow in OWS-5 and suggests how distributed Rules Services could, in the future, be queried by conflation services to provide the attributes and characteristics of rules for conducting a particular conflation process.

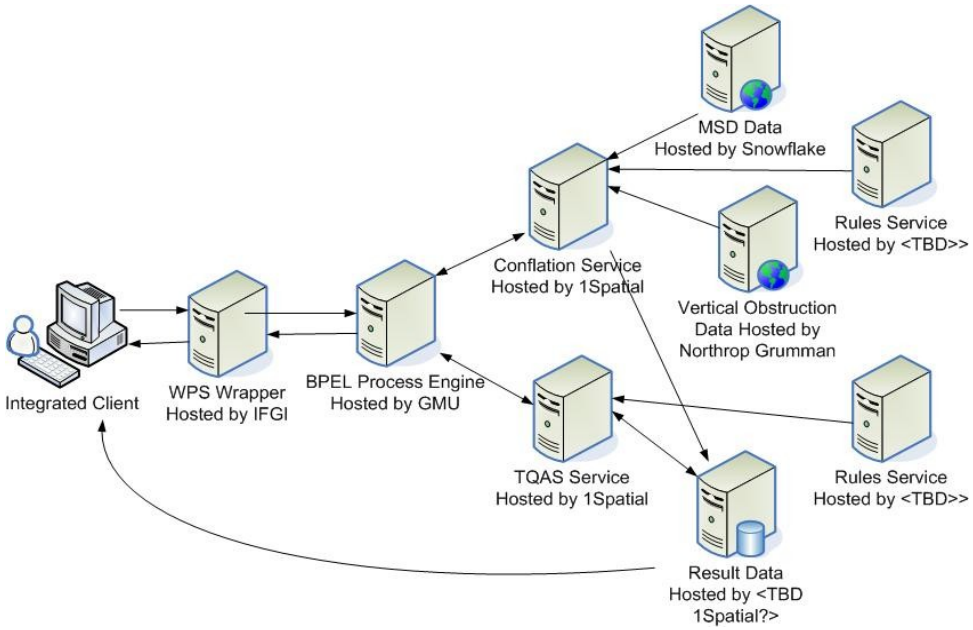

Figure 11: Example Conflation Workflow with Distributed Rules Services

Bibliography

[1] Guidelines for Successful OGC Interface Standards, OGC document 00-014r1

[2] OWS4 – Topology Quality Assessment IPR, OGC document 07-007r1

[3] DVOF Mappings.xls – Provided by the NGA (available from the OGC portal - https://portal.opengeospatial.org/files?artifact_id=24665)