DISTRIBUTED AGENTS FOR ONLINE SPATIAL SEARCHES

Elizabeth-Kate Gulland[1], and Simon Moncrieff[2], and Geoff West[3]

Department of Spatial Sciences, Curtin University
Kent Street, Bentley WA 6102, Australia
and Cooperative Research Centre for Spatial Information (CRCSI)
Melbourne VIC, Australia
[1]Email: e.gulland@curtin.edu.au
[2]Email: s.moncrieff@curtin.edu.au
[3]Email: g.west@curtin.edu.au

ABSTRACT

As the availability and utilisation of online data blossoms, automated online searches - whether to answer a simple question, seek specific sensor readings, or investigate research in a particular domain - have raised a number of issues. Simple search tools do not access the deep web of services and online forms, and cannot handle knowledge domain-specific search problems, but specialist search tools can have a narrow domain and applicability. Some online tools circumvent these problems by putting more filter controls into the hands of users, but this leads to more complex interfaces which can raise usability barriers. A distributed approach, where specialised search agents act autonomously to find contextualised information, can provide a useful compromise between a simple, general search interface and specialist searches. This paper outlines work in progress on design and use of specialist search agents, with a case study to find public transportation bus stops within a spatial region. The approach is demonstrated with a proof of concept web interface, developed to interpret a text query to find and show bus stop locations within a named boundary by coordinating multiple online search agents. Search agents were designed to follow a common model to allow for future development of agent types, including specialist agents used in the case study to search standard open web services and extract spatial features.

Keywords: agents, online services, spatial search

1. INTRODUCTION

Users searching for data are accustomed to simple text inputs, such as implemented by web search engines like Google, Bing and Yahoo, which are straight-forward to use but raise processing complications for flexible searches including: 1) how to interpret the user's context, aims and expectations; 2) what filters are appropriate and applicable; 3) how and where the data itself can be accessed; 4) how to format results; and 5) how to rank the relevance of results.

In this paper, we outline an approach for using online search agents, each capable of applying specialist operations and/or accessing specific data sources, to manage complex problems such as these. We demonstrate the approach with a case study web application to find and display bus stop locations within regions found by name.

The case study example shows how agents can be used to access the deep web by extracting individual spatial features from online services, rather than documents such as web pages or metadata descriptions that are indexed by regular search engines. It is tested on data

conforming to the Open Geospatial Consortium (OGC) Web Feature Service (WFS) standard[1], although the common design and interoperability of agents mean that this can be expanded to allow for other data formats and search operations.

The example implements and coordinates agents, based on a common model, to find: 1) spatial features in a WFS by name and/or location; 2) regions by name; and 3) bus stop point locations relative to region(s). From the user's perspective, the workflow leading from a natural language text input to the display of results is hidden; they enter a text query into the web application, which then makes use of the agents to find and display a solution.

In Section 2 we provide background to the search problem and the use of online agents to solve complex tasks. We discuss the approach and methodology for our case study example in Section 3 and results in Section 4. Discussion and conclusions about our findings are in Sections 5 and 6.

2. BACKGROUND

Current online information retrieval (IR) tools typically incorporate flexibility into user interfaces via specialist input forms, that can add specific filters to a search operation, such as choice of spatial location, temporal range, codes or identifiers, or problem-specific categories like author name or government department. This can provide more sophisticated search tools, but may limit applicability to a narrower purpose or group of users. Simplifying the interface to a single generic text input requires more complex processing to interpret natural language text queries and manage potential disparity between what the user asks for and what the data or metadata provides.

Our framework for contextual online searches makes use of multiple software agents which can be accessed as web services, and hence distributed across different machines. Agents can be designed to carry out specialised tasks, such as spatial searches for geographic features within datasets defined using differing formats. Individual agents can also take advantage of other resources such as the semantic web, with its definitions of relationships between resources and terminology.

2.1. Agents

Software agents are computer programs that can communicate with other agents, machines and/or users to solve a task. Web services are one way to facilitate communication between distributed agents which may be housed on different machines. An agent can use communication standards such as the RESTful framework (Representational state transfer) as an access point to a web service; for example, a *DescribeFeatureType* request can be sent to a WFS service to describe a particular spatial dataset ("feature type").

Individual agents can be designed to solve problems such as translating between a user's text query and messages to send to web service(s) to answer that query, accounting for implementation details such as syntax differences between service versions [1, 2]. Complex tasks typically require the interaction of multiple agents and, as a result, systems allowing interaction between agents and services have been developed in a number of contexts, including geospatial [3-5].

[1] http://www.opengeospatial.org/standards/wfs

Coordination of agents is complex in a multi-agent system, particularly as agents can be developed independently and hosted separately. One approach is to design agents by focusing on how to interact with them, rather than their internal mechanisms [6]. The specific problem of service discovery – finding relevant agents to access – is outside the scope of this paper, but discovery agents making use of the framework described in this paper could be incorporated into future search systems.

2.2. Natural language queries

Searches for data based on a natural language text query are complicated by factors including multiple interpretations of terms within the query, depending upon context. For example, a query such as “stations in Perth” could mean police stations or train stations, and could refer to Perth in Western Australia or Scotland, amongst other possibilities.

For a flexible search tool to be applicable across a range of contexts and user types, it should be able to take advantage of multiple strategies. It would need to distinguish between contexts, search strategies, data sources, and types of results. Using multiple search agents, where each can apply a more limited range of strategies and search a specific subset of data, is an approach to include this flexibility whilst managing the complexity involved.

The semantic web facilitates machine-to-machine communication by defining relationships between entities, which may be defined differently across different systems. Entities can describe anything including people, online agents, named locations, or terms in a dictionary and are typically semantically linked using ontologies, which define triples such as ["bus stop", "*is-a-type-of*", "station"]. Related terminology can be directly relevant to spatial queries - a semantic triple like ["Perth", "*is-in-the-state-of*", "Western Australia"] could inform an agent processing the “stations in Perth” query such that a dataset described as “public transport in Western Australia” could be marked as potentially relevant, though the dataset and query have no terms in common, even if the agent has no spatial capabilities to check within boundary polygons.

Semantic search tools have been developed to make use of a wide variety of resources, including web pages, predefined ontologies, Wikipedia [7], Google [8], and context-specific resources such as in the biomedical field [9]. Some search tools combine semantic web and offline search techniques [10]. Recent research has investigated the combination of the semantic web with geospatial search tools [4, 11].

2.3. Spatial searches

Traditional web searches are applied to indexed documents and cannot delve into the deep web, which includes data accessible via online forms or web services such as WFS [12, 13]. Geographic search engines are designed to search for online data that are relevant to a text query and spatial location.

WFS data was selected for the case-study as it is a common format for providing web access to spatial features and their data attributes [5], and WFS data from multiple providers has been used across different applications such as disaster management [14], health [15], and general-purpose Spatial Data Infrastructures (SDI) [16]. The WFS standard format allows for data use across different technical systems and, although its queries are not easily designed by a lay-person without external help, it can be part of a larger system, for instance by connecting with online semantic tools [17].

Standard spatial and service formats facilitate automated discovery of new online data sources to search, using a variety of automation techniques including ontologies [11] and web crawlers [18]. Although service discovery is beyond the scope of this paper, the agents being developed and tested are provided as online services such that they can theoretically be found by tools like these, and service discovery agents could also be designed to take advantage of the proposed framework.

WFS and other spatial service standards define query parameters such as bounding regions, often making use of other standards such as Geographic Markup Language (GML). Each standard has numerous options and syntax can differ across versions and implementations. A level of expert knowledge is necessary to manipulate these settings directly, so it is more feasible to produce requests to a data service programmatically via a user interface or software agent. To enable reuse of standard services without coupling them to specific use-cases, extra details about the services are required. One strategy to manage this approach is to record semantic information about the services themselves [5].

2.3.1. Spatial Data Service Example: WFS

Standard web services for accessing spatial data include OGC standards such as WFS. A simple, attribute-matching query for a WFS data source is shown in Table 1, with an extract from its associated output in Table 2. The request syntax and available options of a WFS depend upon the service itself and its version. Note in Table 2 that the bounding box syntax for individual features differs from the overall dataset reference system. These variations complicate the manual or, more commonly, machine to machine communication required for flexible access to the data service.

```
http://.../geoserver/region/ows?service=WFS
&request=GetFeature&version=1.1.0&maxFeatures=50
&outputFormat=application/json&typeName=region:PerthSuburbs
&filter=<Filter>
 <PropertyIsLike wildCard="*"
    singleChar="." escape="!">
 <PropertyName>SSC_NAME</PropertyName>
 <Literal>*Perth*</Literal>
 </PropertyIsLike></Filter>
```

Table 1. WFS (version 1.1) request to find features by name

```
{"type":"FeatureCollection","totalFeatures":6,
"features":[
   {"type":"Feature", "id":"PerthSuburbs.59",
    "geometry":{"type":"MultiPolygon",
          "coordinates":[[[[115.867820512,-31.9533984945,0],
              ..., [115.867820512,-31.9533984945,0]]]]},
    "geometry_name":"the_geom",
    "properties":{"SSC_CODE":"50240",
              "SSC_NAME":"East Perth",
              "CONF_VALUE":"Very good",
              "SQKM":3.31589225707466,
              "bbox":[115.866387744,-31.969096022000002,
                      115.887805664,-31.938366116]}},
   {"type":"Feature", "id":"PerthSuburbs.136",
    "geometry":{"type":"MultiPolygon","coordinates":...},
    "geometry_name":"the_geom",
    "properties":{"SSC_CODE":"50590",
```

```
              "SSC_NAME":"North Perth",
              "CONF_VALUE":"Very good",
              "SQKM":3.08997567818837,
              "bbox":[115.844222624,-31.936526365,
                      115.863977152,-31.9128422395]...}},
    ...
 ],
 "crs":{"type":"name",
        "properties":{"name":"urn:ogc:def:crs:EPSG::4326"}},
 "bbox":[-31.9885831265,115.817992544,
         -31.9128422395,115.99540371200001]
}
```

Table 2. Extract of results from WFS request (Table 1)

Adding a spatial filter to a WFS request increases its complexity and requires details such as geometry attribute name, and spatial reference system. A flexible WFS agent needs to be able to discover details such as these without direction from a user. Table 3 shows an example of a more complex spatial operation using GML (Geographic Markup Language) that is not available in all WFS.

```
http://.../geoserver/transport/ows?service=WFS
&request=GetFeature&version=1.0.0
&maxFeatures=200&outputFormat=application/json
&typeName=transport:Stops
&filter=<Filter
   xmlns:gml="http://www.opengis.org/gml">
 <Within><PropertyName>the_geom</PropertyName>
 <gml:MultiPolygon srsName="EPSG:4283">
 <gml:polygonMember><gml:Polygon>
 <gml:outerBoundaryIs><gml:LinearRing>
 <gml:coordinates>115.867820512,-31.9533984945
   115.867814048,-31.95334971
   115.867764352,-31.9529839465
   ... 115.867820512,-31.9533984945
 </gml:coordinates>
 </gml:LinearRing></gml:outerBoundaryIs>
 </gml:Polygon></gml:polygonMember>
 </gml:MultiPolygon></Within></Filter>
```

Table 3. WFS (version 1.0) request to find point features within a region

2.4. Orchestration of Data Search Agents

Manual coordination of multiple services to answer a specific question is a complex process. An example manual workflow to find bus stops within a suburb via WFS data sources is:

1. Find a data service that contains information about suburbs.
2. Enter a WFS query (Table 1) to find suburbs with names that match the target name.
3. Extract polygon geometry information from returned records (Table 2).
   3.1. If necessary, transform the polygon feature(s) to the second source's spatial reference system.
   3.2. If necessary, create a buffer around the polygon feature for a "nearby" search.
4. Convert polygon(s) into a filter format (e.g. GML) that the bus stop WFS can interpret.
5. Find a second WFS data source holding bus stop information.
6. Enter a WFS query (Table 3) with the new filter to retrieve bus stop features.

7. Extract desired property value(s) and geometries from any returned records and display them.

The case study described in this paper focusses on linking results from known agents rather than data service discovery (steps 1 and 5 in the manual workflow above). However, the design is extensible to allow for future enhancements such as service discovery and parallel processing. With the data services pre-set, the case study automates the workflow above, hiding processing detail from the user so that they need only enter an initial text query.

Extracting records from within a web service is an example of a deep search operation, and search agents have been designed to automate searches for geometry and textual features from spatial data services. Encapsulating format-specific requirements such as WFS syntax into agents allows for communication between agents that use more general parameters.

A benefit of multiple search agents is that the same request parameters can be sent to specialist agents that can process data in alternative formats, such as OGC standards, spreadsheets, or databases. They can also be designed to interpret queries based on terminology specific to a particular knowledge domain, allowing for parallel searches across different contexts.

3. APPROACH

The aim of the case study was to reduce a multi-step process to a single user action: entering a text query. Compare this with the manual, multi-step workflow described in Section 2.4.

The proposed framework was tested with a case study using agents to search for public transportation sites in Western Australian suburbs. All models and the coordinating web application were developed in the Django web application framework. Figure 1 outlines the overall process followed by the case study, with details of individual search agents covered in Section 3.2.

The search coordinator is a web application that extracts a text query from an input box and parses this parameter to extract a region name, feature type, and spatial operator. Two text patterns are catered for:

A. `<feature type> <operator> <region name>` "*stations near Mount Lawley*"
B. `<region name> <feature type>` "*Mount Lawley stations*"

Feature types are compared against a list of known types – in the case study, these include terms such as "bus stop" and "station". Unknown types cause a warning to be displayed, although the default feature types are still searched.

Currently recognised operators are *near* (any of [near, nearby, near to, close to]) and *within* (any of [in, within, inside]). The second pattern is turned into a sequence of possible names, for example ["Mount", "Mount Lawley", "Mount Lawley stations"]. Queries of this type are assumed to be *within* searches. Each possible region name is passed to a boundary search agent (item 1 in Figure 1) which searches for one or more matching geometric regions that can be used in later stages of the search process. If no regions are found, for instance if

the query input is invalid or the boundary agent service fails, the search may proceed without this spatial filter, as described in later sections.

This approach of extracting labels for different options lends itself to ontologies, where operator names extracted from the query could be linked to the relevant agents to use. Names with similar meanings such as “nearby” and “close to” could also be connected to each other with ontologies.

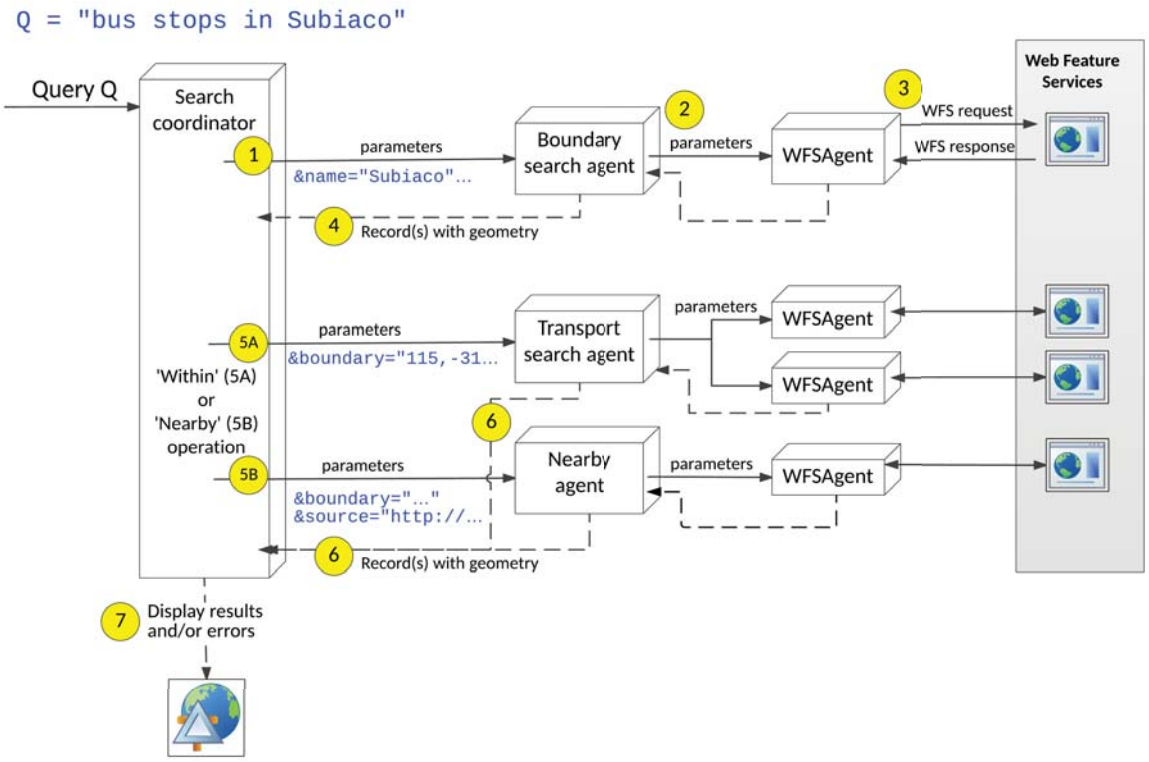

Figure 1. Processing steps between search agents in the case study implementation

3.1. Orchestration

A web application was developed to provide a user interface, coordinate individual agents’ actions, and display the final results. In essence, it acts as a mediator agent orchestrating queries to, and results from, other agents. As it cannot be assumed that all agent types return geometries or even individual records, any results returned to the coordinating web application were ignored if they contained no records or no geometries.

After initial parsing of the user’s textual query, the region name(s) are sent as a *name* parameter to the first type of agent, specific to boundary regions (Figure 1-1). This agent looks at information from its own source(s) to find attributes that are likely to hold a region name, for example any (case-insensitive) attribute name partially matching “name”, “ID” or “label”. The boundary agent passes relevant parameters on to its source, a WFS agent (Figure 1-2), which builds a WFS request to find features that match (or partially match) the region name, and formats the WFS response into its own list of results (Figure 1-3)

Provided that at least one valid region was returned by the boundary agent (Figure 1-4), it is included as a *boundary* parameter to be sent to another agent for finding features relative to a region. For a "within" operation, a public transportation agent is used (Figure 1-5A). This defines its own WFS agent sources to build requests, this time with a spatial filter, similarly to the previous stage. For a "nearby" operation, a nearby agent is sent the boundary region as a target parameter, and a URL is sent to define the source of features to be searched (Figure 1-5B). Any records returned by step 5 are returned to the coordinator (Figure 1-6).

Where spatial features are retrieved, a map is produced using the Leaflet[2] JavaScript library with marker clusters[3], and a scrollable text area added to list selected attributes from the returned features, as seen in Figure 3. Attributes are also displayed on the map when the mouse rolls over a point feature, as shown in the zoomed-in area in Figure 3.

In the demonstration web application, all results are displayed. However, the format of agents' query results allows for alternative visualisations, as discussed in Section 5. If the target feature type (such as bus stop or station) is not recognised or no region is found, a warning is shown (Figure 4).

3.2. Search Agents

The request format for an agent type is consistent, irrespective of internal variations such as the version and capabilities of its data source. Similarly, agent responses are returned in a consistent format. Each response always includes the request parameters, the date it was invoked, the service type, and a list of results, which may be empty. If errors occur during the agent's search, error messages are also listed. Each entry in the result list contains the source URL and where available, a list of individual records. Each record, where available, is labelled and can also contain other data such as location ("geometry") and attributes listed by name.

An agent can be accessed programmatically or via a RESTful web interface. Request parameters include *request*, *query* (the initial query text) and other parameters depending on the agent's purpose and context, such as *bbox*, *boundary*, and *name* (Table 4). New agent types can add additional input parameters. The implemented agents return JSON (JavaScript Object Notation) output although the design is extendable to allow more formats in the future.

A *DataAgent* model was designed as a generic search agent, a template for all other search agents to build upon (Figure 2). As a minimal requirement, every agent model defines its type (such as "WFS" or "boundary"), name, service web address, and read-only list of data sources. Each agent defines at least three actions: *process* (accepting a dictionary of query parameters), *getCapabilities*, and *addSource*.

[2] http://leafletjs.com/
[3] https://github.com/Leaflet/Leaflet.markercluster

RESTful query parameter	Purpose	Example value
request	Action	getCapabilities, search
query	User query string	bus stops in Perth
outputFormat	Format of results from agent	JSON (default)
name	Record (partial) name	Perth
bbox	Bounding box to search within	-43.65,113.15,-10.68,153.64
boundary	Text definition of a polygon	A GeoJSON or GML string

Table 4. A selection of agent input parameters

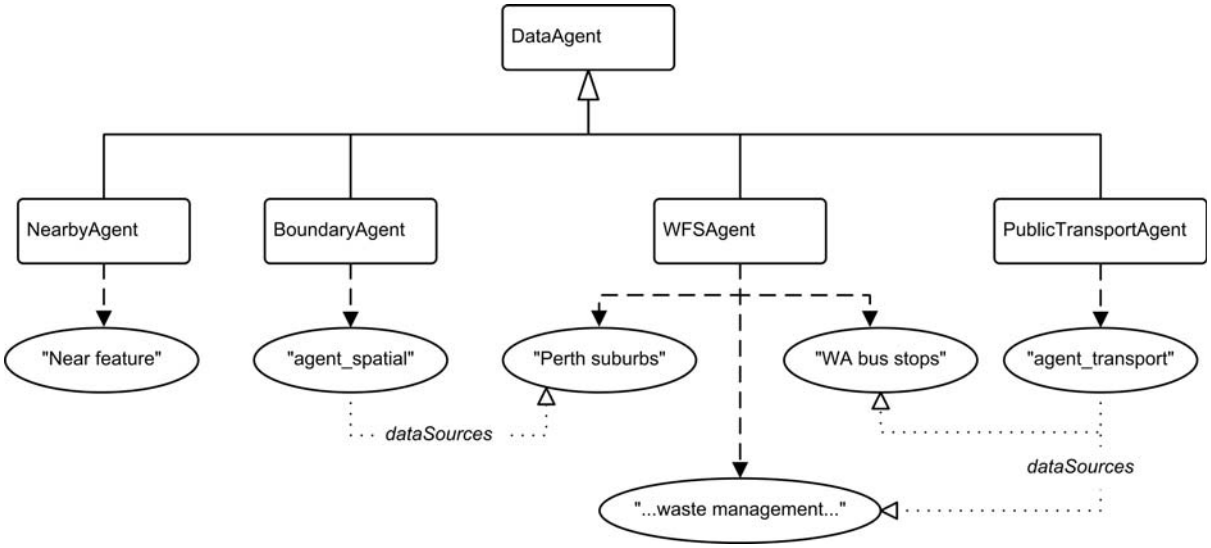

Figure 2. Data agents: classes (rectangles) and instances (ellipses)

A *DataAgentSource* model was designed to allow any agent to record links to one or more other agents as data sources. Each agent can define 0, 1 or more sources, and can itself be the source of one or more other agents.

As shown in Figure 2, four specialised search agents were designed as Django models, subclassed from DataAgent:

- *WFSAgent*: to handle a Web Feature Service layer. This agent builds a WFS query based upon input parameters and its source service's capabilities.
- *BoundaryAgent*: to retrieve boundary region records from one or more sources.
- *PublicTransportAgent*: to extract a subset of bus stop features from one or more sources within a boundary region.
- *NearbyAgent*: to extract spatial features near a provided geometry.

An agent will respond to a query with output including date, request and list of results. Optional information in the response can include likelihood weights of results and/or records, with or without geometry detail. An example response is shown in Table 5.

3.2.1. WFSAgent

Internally, the WFSAgent uses WFS requests such as *GetCapabilities* and *DescribeFeatureType* to discover its source's version, capabilities, and details about its feature type (dataset) and attributes. This allows it to check for capabilities including spatial operators before attempting to apply them. The agent also looks for names of attributes likely to contain a geometry field or a label. In the latter case, partial matches were sought to any of ["name", "id", "label"] - in the suburb data source, for instance, a match was found to an attribute called "SSC_NAME". It uses this information to build a valid WFS request for records.

A typical process within a WFSAgent will follow a sequence such as:

1. Check that the WFS is currently active and available online.
2. Get the WFS's capabilities for the preferred version via a GetCapabilities request.
3. Get the FeatureType (layer) name to use from the WFS.
4. If the query parameters include "name":
    a. Get the most likely attribute name in the WFS with a DescribeFeatureType request – the first partial match to any of ["label", "name", "id"].
    b. If the WFS capabilities include partial matches, create a filter with wildcards, otherwise create an exact match filter.
5. If the query parameters include "boundary":
    a. Get the most likely attribute name for geometry information in the WFS with a DescribeFeatureType request.
    b. Create a spatial filter for the preferred WFS version.
6. If the query parameters include "bbox":
    a. Create a bounding box filter.
    b. As WFS will not allow both a bbox and spatial filter in the same query, remove the bbox filter (or create a combined spatial filter) if a spatial filter is also present.
7. Combine filters 4-6 as appropriate into a single GetFeature request and send it to the WFS.
8. Get any records (and/or error messages) and add into the expected DataAgent response format.

Three instances of WFSAgent were created (Figure 2), one for each of three online services: a) Perth suburbs; b) Perth bus stops; and c) public waste management sites. A GeoServer instance was set up to host the first two WFS layers, which were created from public spatial data layers: Western Australian bus stops from Transperth[4], and 2011 state suburbs from the Australian Bureau of Statistics[5]. A public online WFS for Australian waste management point sites[6] was also selected as a proxy for bus stop locations, as it contained point data in the same location as the suburb data.

[4] http://www.transperth.wa.gov.au/About/Spatial-Data-Access
[5] http://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/1270.0.55.003July\%202011?OpenDocument
[6] http://www.ga.gov.au/gis/services/topography/National_Waste_Management_Facilities/MapServer/WFSServer

An example RESTful request to a WFSAgent for feature(s) with names including "Perth" would produce a WFS request such as shown in Table 1, with results as shown in Table 2. The syntax produced could differ depending upon its WFS's version and capabilities.

3.2.2. BoundaryAgent

The BoundaryAgent model defines one or more source agents using the DataAgentSource model. In the case study, a WFSAgent linking to the suburb service was defined as a single source (Figure 2).

An example RESTful request to a boundary agent for region(s) with names including "Perth" is **`http://.../boundary?request=search&query=bus stops in Perth&name=Perth`**, which would send the name parameter to its source for processing, returning a response as shown in Table 5.


```
{"date": "12/12/2015",
 "request": {"query": "bus stops in Perth", "name": "Perth",
             "request": "search"},
 "results": [
    {"url": "http://.../geoserver/region/ows",
     "records": [
        {"SSC_CODE":"50240",       "SSC_NAME":"East Perth",
         "CONF_VALUE":"Very good", "SQKM":3.31589225707466,
         "bbox":[115.866387744,-31.969096022000002,
                 115.887805664,-31.938366116],
         "geometry": {"type":"MultiPolygon",
           "coordinates":[[[[115.867820512,-31.9533984945,0],
                    ..., [115.867820512,-31.9533984945,0]]]]},
         "type": "region:PerthSuburbs",
         "label": "East Perth"}, ...
      ],
    "name": "region:PerthSuburbs"}
 ],
 "service": "boundary"
}
```


Table 5. Response extract from a BoundaryAgent

Although the data source was pre-set in this case, consistent request parameters and output formats allow for new agents to be added at a later stage as alternative data sources.

3.2.3. PublicTransportAgent

Like BoundaryAgent, this agent records the specific agents it uses as sources of data. In this case, it uses two WFSAgents, linked to the Transperth and waste management services (Figure 2).

This agent passes queries on to its WFS source(s), including a "boundary" or "bbox" parameter containing a region described as text, such as GeoJSON format. The agent searches for features purely within spatial filters rather than attribute values.

3.2.4. NearbyAgent

The NearbyAgent does not internally record the source it uses to find data. Instead, it uses parameters for a comparison polygon as text, and the web address of a source agent. In the case study, the Transperth service is used as the source.

This agent accepts a "target" parameter of a spatial feature in text format, and a "source" parameter containing the URL of a spatial data service. It also optionally accepts "distance" and "units" inputs to determine how near to search to the target. A default distance (100 metres) is used if these inputs are not specified.

This agent uses the "source" parameter to look for a previously saved spatial search agent, or creates one if none is found. In this proof-of-concept, a WFSAgent is used as a source. The NearbyAgent then creates a buffered region from the input target and passes this information on to its source.

Each NearbyAgent can define its own default distance and units so that agents can be set up for use in different scenarios, such as a larger distance in a rural setting than in a dense housing area. This would facilitate the use of user feedback and context in the future.

4. RESULTS

Manual workflows were tested to extract spatial features from the feature services described in Section 3 for WFS versions 1.0.0, 1.1.0 and 2.0.0. Text attribute filters were tested for matches to exact, partial, and non-existent suburb and bus stop names. The case study tool was tested with queries that included exact, partial, case-insensitive or misspelled suburb names, or that missed a suburb name entirely. It was also tested with known feature types (such as bus stops and stations) and unrecognised feature types.

Testing of the case study web application showed that it could find suburbs after a partial or case-insensitive name match. In contrast, a manual WFS request for "subiaco" found no matching polygon features, even when the *PropertyIsLike* filter was used instead of a direct equality comparison, because the name attribute expected a capital letter: "Subiaco".

All features returned from a spatial search agent included a *label* attribute, even if there was no attribute called "label" in the original WFS data records.

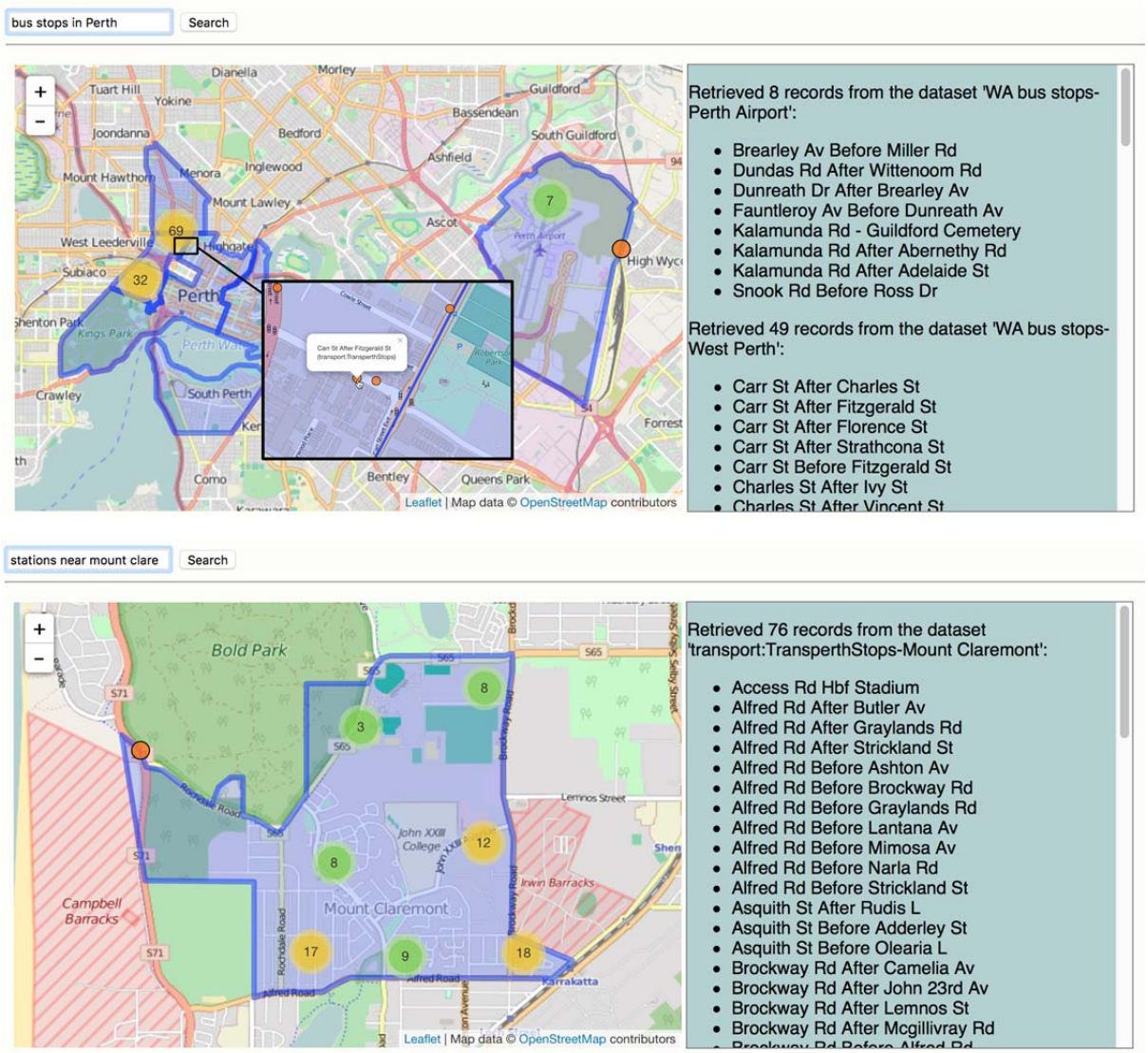

Figure 3. Search results, shown as clustered map features and list of names.

Where an agent was unable to interpret a request or found no matching results, it returned an empty list, which was ignored by the coordinator displaying results. As well as invalid region names or feature types as discussed in Section 0, this can be caused by a problem or limitation within the data service itself, such as a temporary loss of access. The waste management service WFS, which can only output in XML format and has limited spatial filter capabilities, did not return any points to the PublicTransportAgent that utilised it. However, the coordinating agent still returned features from its other source - the Transperth WFS. This demonstrated robustness in the overall design.

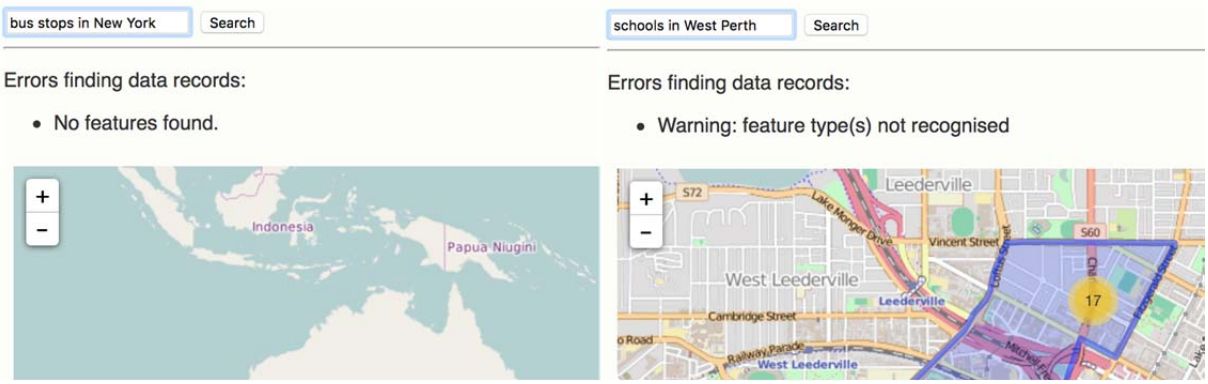

Figure 4. Search results for unknown region (*left*) or target type (*right*)

5. DISCUSSION

Automating the creation of WFS parameters from generic query parameters within WFSAgent was complicated by differences between versions and implementations, but returning empty datasets where problems arose allowed agents to continue searching alternative sources. Encapsulating syntax requirements into a WFSAgent allowed for specialised search agents like the BoundaryAgent and PublicTransportAgent to focus on relevant actions without needing to consider quirks of different data sources. These agents pass a "process" signal to their sources, so extra agents and more diverse agent types could be added to their source lists without needing further alteration.

The case study application could be extended to add interaction by taking advantage of optional features defined in the data agent response format. For instance, the format allows agents to specify the likelihood of a result set and/or individual record with optional "weight" values. An orchestrator could take advantage of this and other metadata within agents' result sets to provide feedback to users such as ranking and provenance of records. Any application or agent using this information would need to check for the existence of optional values before utilising them.

In the next stage, additional agent(s) will be implemented to further test coordination of agents. The coordinating agent would need to be extended to cater for distributed agents, by allowing it to await responses from agents concurrently processing search parameters. As agents are designed to return an empty set upon failure or lack of results, a coordinator can send out responses to all known agents, rather than pre-determining which ones to access. An exception, as demonstrated by BoundaryAgent in the case study, is where a sequence of actions is necessary. In this case, the BoundaryAgent had to be accessed first in order to find boundary parameters to send to other search agents.

An agent currently under development focusses upon expanding a query with semantically related terms, which may include related terms from different knowledge domains. This will allow for future expansion into ranking of results and contextual display, such as showing results within facets [19] defined by the results' or agents' domains. In combination with ranking of results within and between facets, this would assist users to focus in on datasets from topics they are particularly interested in.

The common framework for search agents facilitate extension and creation of coordinators to interact with future agents. These agents may include features beyond those shown in the case study demonstration, such as service discovery of data sources; weighted results that can be ranked by the strength of their relationship to a query; or temporal filters.

6. CONCLUSIONS

This paper has described a design for search agents that can be coordinated to solve complex search problems based on a textual query. A case study was developed to test its use with a spatial query problem requiring multiple processing stages and data sources. This initial test showed promise for simplifying a user's workflow for finding spatial data by using a combination of search agents. Embedding specialised syntax and requirements within agents reduced the required level of expert knowledge for users of a simple query interface, who would otherwise need to manually solve a multi-stage problem based on online spatial data and service formats.

7. ACKNOWLEDGEMENTS

The research reported in this paper was supported by the Australian Primary Health Care Research Institute (APHCRI), which was supported by a grant from the Australian Government Department of Health. The information and opinions contained in it do not necessarily reflect the views or policy of the Australian Primary Health Care Research Institute or the Australian Government Department of Health. The Cooperative Research Centre for Spatial Information, whose activities were funded by the Australian Commonwealth Cooperative Research Centres Programme, has supported this work.

8. REFERENCES

1. Huang, W. and D. Webster. *Enabling Context-Aware Agents to Understand Semantic Resources on The WWW and The Semantic Web*. in *2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI '04)*. 2004. IEEE Computer Society.

2. Zhao, T., C. Zhang, M. Wei, and Z.-R. Peng, *Ontology-Based Geospatial Data Query and Integration* in *Geographic Information Science: 5th International Conference, GIScience 2008, Park City, UT, USA, September 23-26, 2008, Proceedings*, T.J. Cova, et al., Editors. 2008, Springer: Berlin Heidelberg. p. 370-392.

3. Yue, P., L. Di, W. Yang, G. Yu, and P. Zhao, *Semantics-based automatic composition of geospatial Web service chains.* Computers & Geosciences, 2007. **33**(5): p. 649-665.

4. Zhao, P., T. Foerster, and P. Yue, *The Geoprocessing Web.* Computers & Geosciences, 2012. **47**: p. 3-12.

5. Tian, Y. and M. Huang, *Enhance discovery and retrieval of geospatial data using SOA and Semantic Web technologies.* Expert Systems with Applications, 2012. **39**(16): p. 12522-12535.

6. Viroli, M., A. Ricci, and A. Omicini, *Operating instructions for intelligent agent coordination.* Knowledge Engineering Review, 2006. **21**(1): p. 49-69.

7. Gabrilovich, E. and S. Markovitch, *Wikipedia-based Semantic Interpretation for Natural Language Processing.* Journal of Artificial Intelligence Research, 2009. **34**: p. 443-498.

8. Cilibrasi, R.L. and P.M.B. Vitanyi, *The Google Similarity Distance.* IEEE Transactions on Knowledge and Data Engineering, 2007. **19**(3): p. 370-383.

9. Rybinski, M. and J.F. Aldana-montes, *Calculating semantic relatedness for biomedical use in a knowledge-poor environment.* BMC Bioinformatics, 2014. **15**(Suppl 14): S2.

10. Dong, H., F.K. Hussain, and E. Chang. *A Survey in Semantic Search Technologies*. in *Second IEEE International Conference on Digital Ecosystems and Technologies*. 2008. Phitsanulok Thailand: IEEE.

11. Bogdanović, M., A. Stanimirović, and L. Stoimenov, *Methodology for geospatial data source discovery in ontology-driven geo-information integration architectures.* Web Semantics: Science, Services and Agents on the World Wide Web, 2015. **32**: p. 1-15.

12. Chun, S.A. and J. Warner, *Semantic Annotation and Search for Deep Web Services*, in *Tenth IEEE Conference on E-Commerce Technology and the 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services*2008, IEEE: Washington DC, USA. p. 389-395.

13. Madhavan, J., L. Afanasiev, L. Antova, and A. Halevy, *Harnessing the Deep Web: Present and Future.* ArXiv, 2009.

14. Zhang, C., T. Zhao, and W. Li, *Towards Improving Query Performance of Web Feature Services (WFS) for Disaster Response.* ISPRS International Journal of Geo-Information, 2013. **2**(1): p. 67-81.

15. Moncrieff, S., G.A.W. West, J. Cosford, N. Mullan, and A. Jardine, *An open source, server-side framework for analytical web mapping and its application to health.* International Journal of Digital Earth, 2014. **7**(4): p. 294-315.

16. Rautenbach, V., S. Coetzee, and A. Iwaniak, *Orchestrating OGC web services to produce thematic maps in a spatial information infrastructure.* Computers, Environment and Urban Systems, 2013. **37**: p. 107-120.

17. Zhang, C., T. Zhao, W. Li, and J.P. Osleeb, *Towards logic-based geospatial feature discovery and integration using web feature service and geospatial semantic web.* International Journal of Geographical Information Science, 2010. **24**(6): p. 903-923.

18. Bone, C., A. Ager, K. Bunzel, and L. Tierney, *A Geospatial Search Engine for Discovering Multiformat Geospatial Data Across the Web.* International Journal of Digital Earth, 2014. **9**(1): p. 47-62.

19. Adams, B. and G. McKenzie, *Inferring Thematic Places from Spatially Referenced Natural Language Descriptions*, in *Crowdsourcing Geographic Knowledge: Volunteered Geographic Information (VGI) in Theory and Practice*, D.Z. Sui, S. Elwood, and M.F. Goodchild, Editors. 2013, Springer: Dordrecht. p. 201-221.