Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings

Volume 15 *Seoul, South Korea* Article 9

2015

Distributed Agents For Contextual Online Searches

Elizabeth-Kate Gulland
Cooperative Research Centre for Spatial Information (CRCSI)

Simon Moncrieff
Department of Spatial Sciences, Curtin University

Geoff West
Department of Spatial Sciences, Curtin University

PO-14

DISTRIBUTED AGENTS FOR CONTEXTUAL ONLINE 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)
[1]Email: e.gulland@curtin.edu.au
[2]Email: s.moncrieff@curtin.edu.au
[3]Email: g.west@curtin.edu.au

ABSTRACT

As semantic web use and research blossoms, automated online searches - whether to answer a simple question, seek specific sensor readings, or investigate research in a particular domain - has raised a number of issues. Simple search tools cannot handle context-specific search problems, but specialist search tools 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 case-study 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.

1. INTRODUCTION

Users searching for data are accustomed to simple text inputs, such as implemented by Google search, 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 distributed 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 by suburb name.

The case study example shows how agents can be used to extract individual spatial features rather than complete datasets or documents, such as shapefiles or web pages. 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 in the future to allow for other data formats and search operations. This

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

example coordinates agents, based on a common model, to find: WFS features by name and/or location; regions by name; and bus stop point locations. From the user's perspective, the process from a natural language text input to an output map and list of results is hidden.

In section 2 we introduce some background to the search problem. We discuss the approach 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 by adding specific filters to the interface, such as choice of spatial location, temporal range, codes or identifiers, or problem-specific categories like author name or government department. This can limit applicability to a narrower purpose or group of users and lead to more complex search tools, but simplifying to a single 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 distributed across different machines. Agents can be designed to take advantage of the *semantic web*, with its definitions of relationships between resources and terminology. They can also be designed to carry out specialised tasks, such as *spatial searches* for geographic features within datasets with differing formats.

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 can be housed on different computers. An agent can use communication standards such as the RESTful framework (Representational state transfer) as an access point to a web service; for example, by sending a *DescribeFeatureType* request to a WFS service with parameters specifying the feature type (i.e. spatial dataset) to be described.

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 (Huang and Webster 2004, Zhao *et al.* 2008). 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 (Yue *et al.* 2007, Zhao *et al.* 2012, Tian and Huang 2012).

2.2 Semantic Web

The semantic web also facilitates machine-to-machine communication, by defining relationships between entities, examples of which can include related terminology or a sequence of web services to use in a knowledge discovery process. Entities and relationships are commonly defined in ontologies, using triples such as ["highway", "*is-a-type-of*", "road"]. Linking resources and terms together facilitates the expansion of simple text-based search interfaces by allowing agents to take advantage of more complex processes, without requiring the user to manually enter extra search parameters.

Relating terms together can assist with the processing of natural language queries such as “bus stops in North Perth”, and this can be extended to cater for cases where context affects the interpretation of a query: consider the term “K12”, which could refer to a geographic feature (mountain in South Asia); the ICD10[2] code for “stomatitis and related lesions” in health, or “kindergarten to year 12” in education. The most appropriate data sources to search would depend on the intended context and, in some cases, the type of data returned would also depend upon context - for instance, a polygon or point object could be more appropriate for a geographic feature, where a textual description or statistical plot could be more useful in the health context. Related terminology is also relevant to spatial queries - a semantic triple like ["North Perth", "*is-in-the-state-of*", "Western Australia"] could inform the search example without requiring spatial operators or features. Where an agent has no ability to check spatial boundaries, a dataset described as “stations in Western Australia” could be marked as potentially relevant, even though it shares no terms with the original query.

Methods for automatic extraction of semantic links have been investigated for a number of information sources, including Wikipedia (Gabrilovich and Markovitch 2009), Google (Cilibrasi and Vitanyi 2007), and context-specific resources such as in the biomedical field (Rybinski and Aldana-Montes 2014). Semantic search tools have been developed to make use of different resources, including ontologies, XML, and images, and some search tools combine semantic web and offline search techniques (Dong *et al*. 2008).

For a flexible search tool to be applicable to 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.

2.3 Spatial searches

Spatial service standards such as WFS describe syntax for defining 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 requirements 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 (Tian and Huang 2012).

Geographic search engines are designed to search for web pages that are relevant to a text query, both in text content and spatial location. Tools such as the IR-Tree are used to rank the likely relevance of web pages for both aspects of content (Li *et al.* 2011).

The case study described in this paper combines aspects of both approaches, by including textual information with spatial searches and also returning features rather than entire datasets or documents. Encapsulating technology-specific requirements such as syntax into agents allows for communication between agents that uses more general parameters.

[2] International Classification of Diseases and Related Health Problems, http://apps.who.int/classifications/icd10/

A benefit of using multiple agents for different data sources and contexts is that it provides the opportunity to process and search data defined in alternative formats by using standard request parameters. For example, SIRF[3] is an online infrastructure developed by the Australian CSIRO (Commonwealth Scientific and Industrial Research Organisation) for linking data that includes spatial content. It aims to use consistent identifiers for locations that may have multiple IDs and names over different government departments. It has its own unique APIs that could be accessed by SIRF-aware agents. SIRF aims to be an intersection point between government data in various formats, including OGC standards.

3. APPROACH

A *DataAgent* model was designed as a generic search agent, with each specialised agent inheriting from this model. As a minimal requirement, every agent model defines its type (such as "WFS" or "boundary"), name, and optionally service URI and read-only list of data sources. Each agent defines actions: *process* (accepting a dictionary of query parameters), *getCapabilities*, and *addSource*. The process action can be passed on to any data source agents, with or without alteration of the initial parameters.

Subclasses of DataAgent were created to search for WFS features, boundary regions, and public transportation locations respectively. Each implemented agent can be accessed programmatically or via a RESTful web interface for access. Request parameters include at least *query* (the initial query text) and other parameters depending on the agent's purpose and context, for example *bbox*, *boundary*, and *name*. An example RESTful request to an agent searching for boundaries, in this case for regions with a name that includes the word "Perth", is `http://localhost:8000/spatial/boundary?query=bus stops in Perth&name=Perth`. Compare this to a WFS request for the same data, as produced by the agent, shown in **Error! Reference source not found.**. The WFS syntax produced could differ from that shown, depending upon implemented WFS capabilities or versions, whilst the request to the agent would remain the same.

```
http://localhost:8080/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>
```

Listing 1. WFS (version 1.1) request to find Perth suburbs by name

Adding a spatial filter to a WFS request increases its complexity and requires details that an agent can discover, such as geometry attribute name, and spatial reference system. An extract from an agent-produced WFS request extract is shown in Listing 2.

```
http://localhost:8080/geoserver/transport/ows?service=WFS
&request=GetFeature&version=1.0.0&maxFeatures=200
&outputFormat=application/json&typeName=transport:Stops
&filter=<Filter
```

[3] Spatial Identifier Reference Framework, http://portal.sirf.net/about-sirf/

```
   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>>
```

Listing 2. WFS (version 1.0) request to find point features within a region

All agents return information about the service itself - its type and the date it was invoked - and results, if any are found. Results are returned in a list, with each entry containing metadata about the source and, where available, a list of individual records. Each result is named and each record (where available) is labelled: see the example in Listing 3 for results from a *PublicTransportAgent*, which also shows how geometry information for each record, where available, is returned.

```
{"date":" 30/06/2015 ",
 "request": {"query":"bus stops in Perth"},
 "results": [
   {"name":"transport:Stops",
    "url":"http://...",
    "records": [
      {"label": "Charles St After Gill St",
       "geometry": {"type":"Point",
       "coordinates" :[...]} , ...
      }, ...]
   }, ... ]
}
```

Listing 3. Response (extract) from a PublicTransportAgent

3.1 Case Study Design

The aim of the case study was to reduce a multi-step manual workflow to a single user action: entering a text query. An example manual workflow, assuming spatial data is accessed via WFS, is:

1. Find a data service that contains information about suburbs.
2. Enter a WFS query to find suburbs with names that match the target name.
3. Extract polygon geometry information from returned records.
4. Find a second data source holding bus stop information.
5. If necessary, transform the polygon feature(s) to the second source's a compatible spatial reference system.
6. Convert polygon(s) into a GML filter that the bus stop service can interpret.
7. Enter a WFS query with the new filter to retrieve bus stop features.
8. Extract desired property value(s) and geometries from any returned records and display them.

A local GeoServer instance was set up to host WFS layers 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 project was developed using the Django web application framework connected to a PostgreSQL database and tested locally with development settings. Specialised search agents were designed as Django models:

- *DataAgent*: the template for all other search agents to build upon.
- *DataAgentSource*: a link between an agent and a data source agent.
- *WFSAgent*: to handle a Web Feature Service layer.
- *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.

Three instances of WFSAgent were created, whose sources were the local GeoServer WFS layers and a public online WFS for Australian waste management point sites[6] respectively. The latter was selected as a proxy for bus stop locations as it contained point data in the same location as the suburb data. An instance of BoundaryAgent was created with the suburb WFSAgent as a source, and an instance of the PublicTransportAgent was created with the other two WFSAgents as source. Although the data sources were manually pre-set in this case, consistent parameter and output formats mean that agents with service discovery capabilities could be designed at a later stage and used as alternative data sources.

Internally, the WFSAgent adapts the query parameters it creates to send to its WFS source by interrogating its WFS version, capabilities, and details about its feature type (dataset) via WFS requests such as *GetCapabilities* and *DescribeFeatureType*. This allows it to check for capabilities such as 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 - in the suburb data, for instance, a match was found to an attribute called “SSC_NAME”.

A web application was designed with a textbox for the user’s initial text query. Python code was added to extract region names and target feature types (e.g. bus stop or station) from simple text patterns such as “bus stops in Subiaco” or “Subiaco bus stops” in a user’s text query. If the region name was ambiguous, a selection of possible names was extracted - for example, “Mount Lawley stations” becomes ["Mount", "Mount Lawley", "Mount Lawley stations"]. The web application coordinated the use of the specialised agents, as shown in Figure 1: requested suburb name(s) were passed as a parameter to the BoundaryAgent and, if a polygon feature was retrieved by the agent, this was sent as a boundary parameter to the PublicTransportAgent which used it to filter out point features. As it cannot be assumed that all types of sources return geometries or even individual records, any results it returned to the coordinating web application were ignored if they contained no records or no geometries. All other results were combined and processed for display on the user interface.

[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

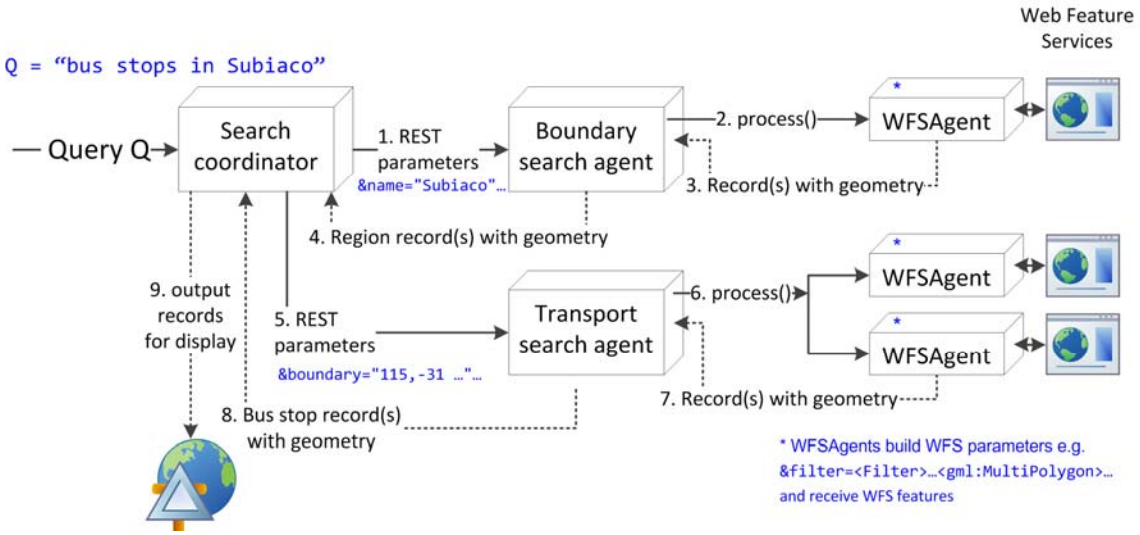

Figure 1. Processing steps between search agents in the case study implementation

If no region is found, or the query's target feature type (such as bus stop or station) is not recognised, an error message is shown (Figure 2) and a plain map with no added features is displayed.

Figure 2. Search results for unknown target (left) or unknown region (right)

Where point features were retrieved, a map was produced using the Leaflet[7] JavaScript library with marker clusters[8], and a scrollable text area was added to list selected attributes from the returned features, as seen on Figure 3. Attributes are also displayed on the map when the mouse rolls over a point feature, as shown in the zoomed-in area on the left of Figure 3.

[7] http://leafletjs.com/
[8] https://github.com/Leaflet/Leaflet.markercluster

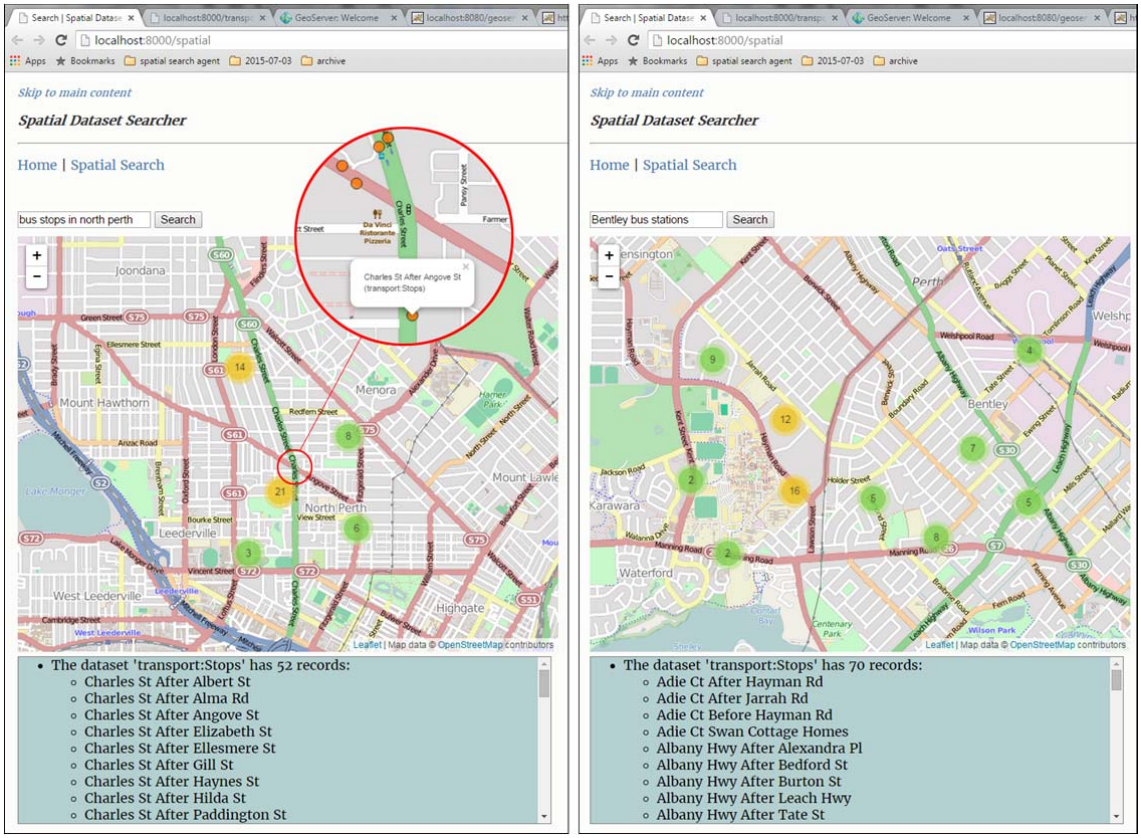

Figure 3. Screen captures of search window showing clustered results.

4. RESULTS

Manual workflows were tested to extract spatial features from the local GeoServer host using queries for WFS versions 1.0.0, 1.1.0 and 2.0.0. Filters were tested for matches to exact, partial, and non-existent suburb and bus stop names. The public waste management service, described with WFSAgent instances in section 3.1 was also tested manually. 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 points within suburbs after a partial or case-insensitive name match. In contrast, a WFS request for “subiaco” found no matching polygon features, even when the *PropertyIsLike* filter was used, because the name attribute expected a capital letter: “Subiaco”.

Where an agent was not flexible enough to interpret a request, it returned no results. For instance, when the WFSAgent requested JSON output from the waste management service WFS, which can only output in XML format, no points were returned. However, the coordinating PublicTransportAgent still returned features from its other source - the local Transperth WFS. This demonstrated the robustness of the overall design. All the returned points had labels that could be used for display, although there was no attribute called “label” in the original WFS data records.

Comparing results from the case study tool to the Google browser showed that both can interpret similar, simple semantic queries ("bus stops in North Perth") but with some differences. Google can interrogate a data source to find a geocoded suburb bounding box to zoom in to and another source to show bus stop point features. However, zooming out shows that point features are not filtered to the suburb polygon, or even the bounding box (Figure 4). Also, the global geocoding can give unexpected results - searching for "Albany" from a computer located in Western Australia found the city in New York, USA instead of Australia.

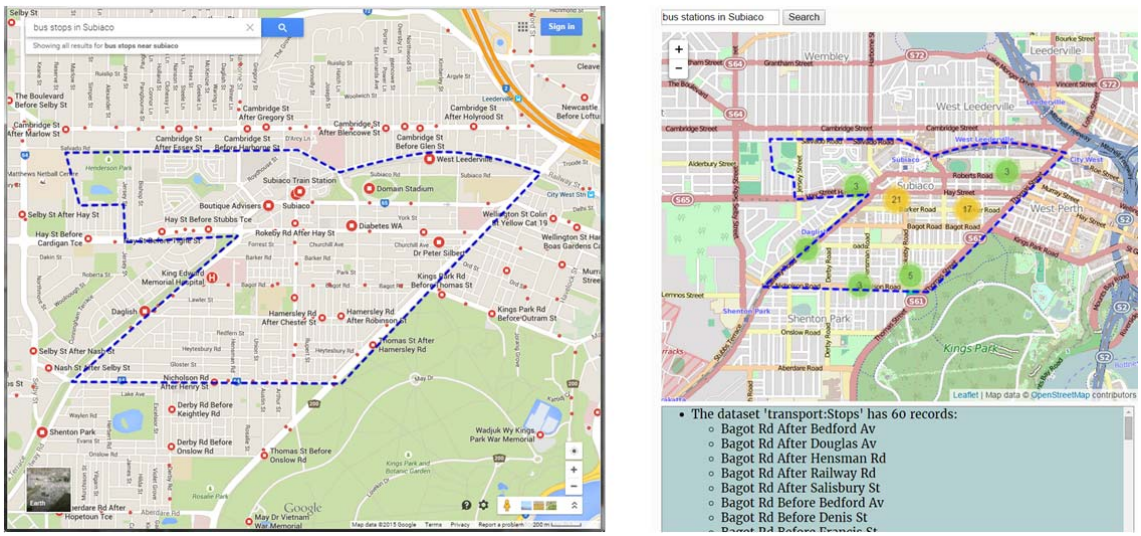

Figure 4. Search results from Google Maps (left) and the case study tool (right)

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 other agents to continue searching any other sources they had available. Encapsulating syntax requirements into a WFSAgent allowed for specialised search agents like the BoundaryAgent and PublicTransportAgent to focus on relevant, contextual actions without needing to consider quirks of different data sources. More agents could be added as sources to these agents to allow access to additional data sources without needing to alter the agents, even if the sources have different usage requirements (such as SIRF).

In the next stage, additional contextualised agent(s) will be implemented to further test coordination of distributed agents. The coordinating agent will need to be extended to allow it to await responses from distributed agents that are concurrently processing initial search parameters. As agents are designed to return an empty set upon failure or lack of results, the coordinator can send out responses to all known agents, rather than pre-determining which ones to access. An exception, as demonstrated in the case study, is the BoundaryAgent, which could be called first in order to find boundary parameters to send to any other agents that can make use of it.

Another agent currently under development focusses upon expanding a query with semantically related terms, which may include related terms from different contexts. This will allow for future expansion into ranking of results and contextual display, such as showing results within facets (Adams and McKenzie 2013) 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.

6. CONCLUSIONS

This paper has described a design for distributed agents that can be coordinated to solve complex, contextual 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 the task of finding spatial data from text queries by using a combination of search agents. Embedding specialised syntax and contextual requirements within agents was shown to reduce the amount of expert knowledge required by 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 is supported by the Australian Primary Health Care Research Institute (APHCRI), which is 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 has supported this work, whose activities were funded by the Australian Commonwealth Cooperative Research Centres Programme.

8. REFERENCES

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

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

Dong, H., Hussain, F.K., and Chang, E., 2008. A Survey in Semantic Search Technologies. Second IEEE International Conference on Digital Ecosystems and Technologies, Phitsanulok Thailand, 26-29 February.

Gabrilovich, E., and Markovitch, S., 2009. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*. 34:443-498.

Huang, W., and Webster, D., 2004. Enabling context-aware agents to understand semantic resources on the WWW and the semantic web. *IEEE/WIC/ACM International Conference on Web Intelligence*, 138-144.

Li, Z., Lee, K. C. K., Zheng, B., Lee, W.-C., Lee, D. L., and Wang, X., 2011. IR-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering*, 23(4): 585-599.

Medelyan, O., Milne, D., Legg, C., and Witten, I.H., 2009. Mining meaning from Wikipedia. *International Journal of Human-Computer Studies*. 67(9): 716-754.

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

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

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

Zhao, P., Foerster, T., and Peng, Y., 2012. The Geoprocessing Web. *Computers and Geosciences*, 47:3-12.

Zhao, T., Zhang, C., Wei, M., and Peng, Z.-R., 2008. Ontology-Based Geospatial Data Query and Integration. *Geographic Information Science: 5th International Conference, Park City UT USA, September 23-26*. 370-392.

Zhang, Z., Gentile, A.L., and Ciravegna, F., 2012. Recent advances in methods of lexical semantic relatedness - a survey. *Natural Language Engineering*, 19(4):411-479.