Semantic Representation and Scale-up of Integrated Air Traffic Management Data

Richard M. Keller*✦, Shubha Ranjan°■, Mei Y. Wei*✦, and Michelle M. Eshow*■
NASA Ames Research Center
Moffett Field, California, USA
{rich.keller, shubha.ranjan, mei.y.wei, michelle.eshow}@nasa.gov

ABSTRACT
Each day, the global air transportation industry generates a vast amount of heterogeneous data from air carriers, air traffic control providers, and secondary aviation entities handling baggage, ticketing, catering, fuel delivery, and other services. Generally, these data are stored in isolated data systems, separated from each other by significant political, regulatory, economic, and technological divides. These realities aside, integrating aviation data into a single, queryable, big data store could enable insights leading to major efficiency, safety, and cost advantages.

In this paper, we describe an implemented system for combining heterogeneous air traffic management data using semantic integration techniques. The system transforms data from its original disparate source formats into a unified semantic representation within an ontology-based triple store. Our initial prototype stores only a small sliver of air traffic data covering one day of operations at a major airport. The paper also describes our analysis of difficulties ahead as we prepare to scale up data storage to accommodate successively larger quantities of data -- eventually covering all US commercial domestic flights over an extended multi-year timeframe. We review several approaches to mitigating scale-up related query performance concerns.

CCS Concepts
• **Information systems~Information integration** • *Information systems~Network data models* • Applied computing~Computers in other domains

Keywords
big data; semantic data

1. INTRODUCTION
With over 30 million flights flown yearly [1], the global air transportation industry generates a vast amount of heterogeneous data daily, including real time data from aircraft sensors, flight radar, air traffic control systems, airline reservation systems, baggage systems, and a myriad of other sources. Although the content of the data from these different systems is interrelated, the systems and their data are only loosely connected. Aviation data is big data by any measure: volume, variety, or velocity. If this data could be integrated, queried, and analyzed by aviation operations personnel and policy makers, we might enable discoveries that could potentially impact the efficiency and safety of our air transportation system. In particular, such a data resource might be used to monitor and analyze air traffic efficiency and safety, to facilitate strategic decision-making by air traffic managers and airlines, and to assist in the design, research, and analysis of air traffic and aviation safety systems.

We are exploring an approach to integrating data across aviation systems using ontologies, linked data, and semantic integration techniques. In particular, we are focusing on a subset of government flight, weather, and airspace management data systems that are part of the US air traffic management (ATM) system. These systems track aircraft as they make their way along their flight path and maintain data about aircraft routing and weather conditions that can impact air traffic.

In the balance of this paper, we describe our motivation for this work, our prototype data integration system, and our analysis of barriers ahead as we scale up our data store to make it useful.

2. BACKGROUND AND MOTIVATION
Our work is motivated by a desire to improve the data management services available to aeronautics researchers within the National Aeronautics and Space Administration (NASA). To perform their research and analysis, the researchers access data stored in the Sherlock aviation data repository [2]. Sherlock replicates aeronautics data from a variety of different sources generated by agencies in the US government, including the Federal Aviation Administration (FAA), the National Oceanographic and Atmospheric Administration (NOAA), and the Department of Transportation (DOT). These data include flight track data, flight plans, weather data, airport delay data, and airspace navigation data, among others. Data refreshed on a fixed periodic update cycle, such as information on airports and air routes, are incorporated into Sherlock within 72 hours of release; real-time data, such as aircraft flight tracks or weather data, are streamed from FAA or NOAA systems, and recorded live to our filesystem and uploaded nightly to Sherlock, making it available for users to search and download. The repository includes data back to 2009.

* National Aeronautics and Space Administration
° Moffett Technologies, Inc.
◆ Intelligent Systems Division ■ Aviation Systems Division

SBD'16, July 01 2016, San Francisco, CA, USA
DOI: http://dx.doi.org/10.1145/2928294.2928296

The majority of data stored in Sherlock consists of raw ASCII or binary data files in their original source format. Some types of frequently requested raw data files are routinely processed, parsed, and inserted into an Oracle database for easier query and access. However, the sheer volume of data in Sherlock (30TB of raw files) – coupled with relatively infrequent access – makes it economically unjustifiable to house all the data within a conventional database[1]. Aeronautics researchers generally use an Oracle web application (APEX [3]) to download archived files and generate reports from the database. This application also supports direct SQL querying of the Oracle database by users.

Although various types of interrelated data are all stored on Sherlock's file system or within its database, they are not integrated in any true sense. The various data sources use differing data formats, inconsistent terminology (e.g., different field names for the same values across data sources), varying spatial and temporal resolutions, different numerical units, and often-incompatible conceptual underpinnings. In short, there is no unifying data model underlying the data stored in Sherlock – no holistic view of how all of this interrelated ATM data connects together. Within Sherlock, each data source is like an isolated island of information that cannot be bridged and connected without considerable effort.

To illustrate, consider how flight data is stored within the data systems replicated in Sherlock. Although a 'flight' is a key entity to track and manage within the ATM domain, none of the government data systems we examined had any explicit record structures corresponding to a flight. Instead, all of the systems contained records that made reference to a flight's 'callsign' (its flight identifier, e.g. UA237) as a proxy for the flight. Formulating a true flight record would require discovering and assembling various bits and pieces of flight metadata from the different data tables or records in which the callsign appears. To complicate matters further, the callsign by itself is not a unique identifier and must be paired with temporal and spatial data to disambiguate among multiple flights with the same identifier. If one wishes to go further and integrate flight data with other types of data, there are additional problems. For example, although weather data measures the meteorological properties of the airspace through which a given flight flies, integrating weather data with flight data is a real challenge due to the different spatial and temporal representations used by weather data sources.

Sherlock's lack of ability to integrate replicated data using a common model has two principal impacts on aeronautics customers. First, users who need to access data across two or more data sources must expend whatever programming effort is necessary to integrate the data themselves, on an as-needed basis. This process can be error-prone, and requires considerable knowledge of the original data – something users do not generally possess. It also leads to duplicative effort because multiple users may end up doing similar integration work. Second, because there is no common view over all the data, it is not possible to formulate queries across the different sources. There would be significant benefits if users were able to execute cross-source queries using a comprehensive store of current and historical air traffic management data. In particular, this capability could assist in diagnosis of bottlenecks in the air traffic control system and in the design and development of new processes and procedures to improve the system.

[1] Efforts are underway to deploy a Hadoop-based Big Data implementation to cover a subset of Sherlock's data.

3. SEMANTIC DATA INTEGRATION

Our approach to addressing these impacts was to establish a common description of ATM data using an ontology model, and then apply semantic integration techniques [4,5] to populate a triple store with data from Sherlock. The integrated triple store can be queried using SPARQL to answer cross-cutting questions about ATM operations. We developed a prototype system to demonstrate this capability with a limited subset of Sherlock's data. The system includes four components (see Figure 1), described in the following subsections.

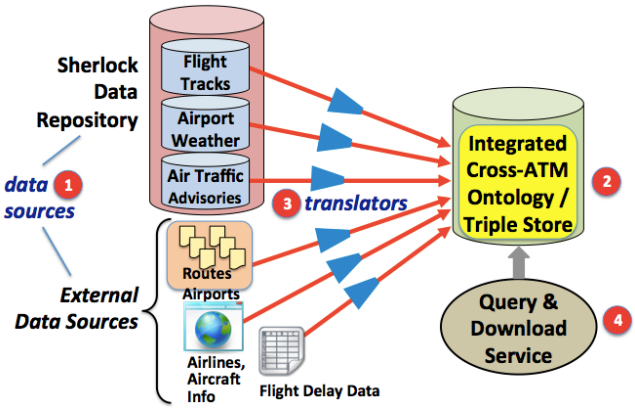

Figure 1: ATM Data Integration Architecture Components

3.1 Data sources

We selected three data sources from the Sherlock repository for integration: flight track data (from FAA's ASDI Aircraft Situation Display to Industry XML feed [6], routinely used by commercial flight tracking software), airport weather data (from NOAA's METAR [7] meteorological observation station data feed), and information on air traffic advisories[2] (scraped routinely from the FAA Air Traffic Control System Command Center website [8]). Given limited resources, we did not attempt to integrate all of the different data sources archived in Sherlock as part of our proof-of-concept prototype. However, we did find it necessary to incorporate some ATM infrastructure data that was not explicitly stored in Sherlock, including information from FAA and other sources on air traffic routes, traffic management facilities, airports, runways, airlines, and aircraft. These data were essential 'glue' required for the integration process.

3.2 Ontology & Triple Store

We constructed an ATM ontology to provide a unified data model for integrating data from the various sources. The RDF-based ontology includes various classes and properties required to describe the state of the ATM system. Although a complete specification of the ontology is beyond the scope of this paper, a sampling of some of its representative classes and properties are included below:

Sample Classes: Flight, FlightPath, FlightConstraint, Airport, AirspaceInfrastructureElement (ArrivalRoute, DepartureRoute, NavigationFix, ATCsector, Airway), MeteorologicalCondition, AirCarrier, AircraftModel, AircraftSystem, AircraftManufacturer, AirportInfrastructureElement (Runway, Taxiway, DeicingPad,

[2] Air traffic advisories consist of modifications issued by the Air Traffic Control Command Center to manage the flow of aircraft under congested operations or in adverse weather conditions.

Gate, Terminal), AirspaceControlFacility (AirTrafficCmdCtr, TeminalRadarCtrlCtr, Tower), TrafficManagementInitiative, etc.

Sample Datatype properties: aircraftHeading, groundspeed, actualDepartureTime, maxRunwayVisibility, airportArrivalRate, manufactureYear, ICAOcarrierCode, aircraftEquipmentCode, etc.

Sample Object properties: arrivalRunway, adjacentSector, operatedBy, manufacturedBy, aircraftFlown, departureConstraint, surfaceWindCondition, hasAircraftFix, hasAgreementWith, etc.

Figure 2 illustrates the ontology representation for a Flight, a key entity in the domain. The flight itself has connections to its departure and arrival airports, the air carrier operating the flight, the aircraft being flown, and its flight route, which is demarcated by a sequence of radar track positions (elaborated further in Figure 3). The color-coding in Figure 2 shows how data from different sources (aeronautical, flight, weather, equipment, and industry) are merged together by the ontology.

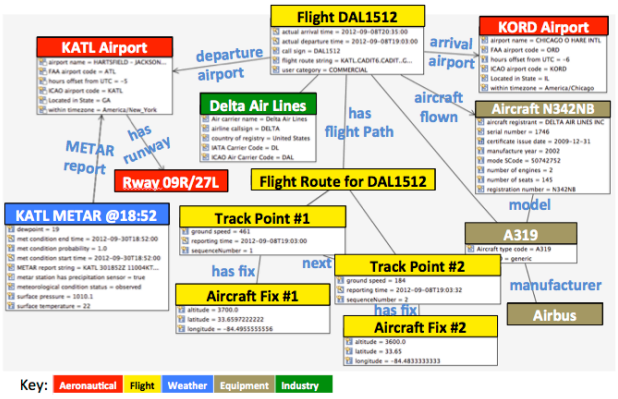

Figure 2: Ontology Representation for a Flight

At the time of this writing, the ontology features more than 150 classes, 150 datatype properties, 100 object properties, and 450K instances. These instances cover one day's worth of integrated ATM data for a single airport. The instances reside on a dedicated Linux server running the AllegroGraph [9] triple store.

3.3 Translators

We wrote data source specific, custom Python and Java code to translate ATM data from its original source format and convert it into ATM ontology triples. For data stored within Sherlock, the sources were either raw, record-oriented files or SQL tables; for ATM infrastructure data gathered elsewhere, the sources were csv files, html files, web service outputs, etc. This data translation step corresponds to a conventional ETL (extract-transform-load) process in a relational data warehouse, except that the target format is RDF triples and that the mappings from the original sources to the triples are often more complex than can be handled with simple ETL mapping tools.

3.4 Query & Download Service

The last component of our prototype has yet to be implemented. Even though it is possible to query data in our triple store using its native SPARQL capabilities, and to download data using vendor-supplied functionality, we feel this is insufficient for our needs. Requiring aeronautics users to understand the ATM ontology and learn SPARQL syntax is not realistic. Our vision is to develop a specialized graphical user interface that allows for entry of queries in a simpler, more intuitive and domain-relevant fashion. In addition, we plan to supply a download capability tailored to the needs of aeronautics and aviation researchers.

3.5 Alternative Approach

One alternative to our semantic approach would have been to utilize conventional relational database warehousing technology and build a common database schema to be populated by an ETL process. However, we believe that ontology-based approaches offer key advantages over conventional relational approaches, principally based on their modifiability, ease of maintenance, built-in inference capabilities, and potential for reuse [10]. Modifiability is particularly important because additional data sources are regularly added to Sherlock, and existing data sources are periodically revised by their producers. Our common data model must be flexible and easily altered without requiring us to rebuild the entire repository. Wholesale rebuilding of tables is often necessary with relational schema changes; due to the finer granularity of RDF, ontology changes have less impact on preexisting triples in a repository.

Although the semantic integration approach we pursued was not particularly novel, it is the first application of these integration techniques to a complex aviation data domain – based on our survey of the open literature.

4. SCALING ISSUES

For our prototype, we processed only one day's worth of commercial US domestic flights to and from Atlanta airport (1342 flights), and transformed associated flight, weather, air traffic, and advisory data into triples within our semantic store. The result was ~2.4M triples. Estimating conservatively at 25,000 US domestic flights per day, processing data from all US domestic flights would generate approximately 29M triples daily, or over 10B triples yearly. Researchers often require multiple years of flight data for their analytical studies, so the potential volume of data to be integrated and queried is significant, and quickly enters the realm of big semantic data. Although our prototype has significant potential, that potential cannot be realized if scaling cannot be achieved successfully.

4.1 Benchmarking

Prior to purchasing a commercial triple store to support this work, we conducted some experiments to evaluate the performance and scalability of two triple store products under consideration: AllegroGraph v5.1 [9] and GraphDB v6.1 [11]. Part of our goal was to examine the feasibility of scaling up our prototype from a single day of Atlanta operations to a month's worth of data. We developed a set of 17 benchmark SPARQL queries to investigate the impact of scale-up. These sample queries were developed in response to feedback provided by domain experts, and address topics of potential interest to NASA aeronautics researchers. In addition – to illustrate the benefits of data integration – these queries require data from multiple Sherlock sources, and cannot be answered using one data source alone. Among these benchmark SPARQL queries are the following (translated into English):

- **F1**: Find Delta Airlines flights using Airbus A319 equipment departing any Atlanta region airport;
- **F3S**: Find flights with rainy departures from Atlanta-Hartsfield airport;
- **S4**: Find which US airspace sector was traversed by the most flights during a given hour;
- **S1S**: Find flights that passed through a given airspace sector within 30 minutes of each other;
- **S6**: Find the busiest airspace sectors in the US on a given day, aggregating hourly;
- **T1**: Find flights that were subject to airport ground delays;

- **W1**: Calculate the hourly Weather-Impacted Traffic Index at Atlanta-Hartsfield airport

(See Appendix for actual SPARQL queries corresponding to F1, F3S, S4, and S1S.) After formulating the benchmark queries, we loaded one day of flights into each of the two candidate triple stores, and ran the queries. Query execution times for the benchmark queries ranged anywhere from 11 milliseconds to 9.6 seconds (average = 1.19 seconds), depending on the query. Next, we synthetically generated and processed one month of flight data and reloaded the triple stores with the resulting 36M triples (an amount 15 times greater than the 2.4M triples generated for a single day). This time, query execution times ranged from 8 milliseconds to 27 minutes (average = 96.65 seconds). (See Appendix for execution timing statistics.)

Depending on the computational complexity of the SPARQL graph-matching problem and the efficiency of the implementation, the results for scaling up from one day to one month varied. Execution time for most of the queries (~60%) scaled up roughly in proportion to the increase in number of triples. In a smaller number of queries (~30%), the increase in execution time was negligible – virtually the same in both cases. However, for at least one query, the time increase appeared to be exponential: 350 or 700 times longer, depending on which triple store product was used. Benchmarking for another query surfaced significant performance discrepancies between the triple stores; one implementation reported a modest 4x increase for one month of data, while the other reported a 1400x increase on the same query.

Overall, the results of our benchmarking exercise do not bode well for an effortless scale-up experience. Clearly, if we wish to scale up to analyze multi-year data, new strategies will be required to answer queries in reasonable response timeframes; many of these queries would require hours or even days to produce responses over multi-year data.

4.2 Possible Approaches to Dealing with Scale

Given that efficiency considerations can undermine the potential value of adopting semantic data representations, we must examine various approaches to improving the query performance of triple stores – especially in the world of big data. These approaches fall into several categories: hardware approaches (e.g., construction of specialized hardware appliances to store and process triples [12]); algorithm improvements (e.g., new graph-matching algorithms [13,14]); software approaches (e.g., SPARQL query planner improvements, triple storage and access improvements [15, 16]); query reformulation approaches (e.g., rewriting and/or reconceptualizing queries [17]); and triple reduction approaches (e.g., representation change, use of auxiliary non-triple storage). For an application builder using a commercial triple store (vs. a technology developer or a researcher), the last two categories represent the sole means available to improve efficiency. As such, we will discuss these two in more detail.

4.2.1 Query reformulation

With any large data store, it is necessary to consider efficiency when formulating queries. When using conventional relational database systems, specific tools are available to analyze and debug query performance issues by examining the query plan, calculating the size of the table joins, formulating indices, creating table partitions, etc. But when using semantic data stores, the current tools are not as mature, and it is more difficult to analyze and optimize queries. We found that although some query planning statistics could be generated using the triple stores we benchmarked, the support for query analysis was primitive and poorly documented. Furthermore, these systems provided little transparency as to how SPARQL queries were translated into code and executed, so it was difficult to determine how to go about rewriting queries to be more efficient.

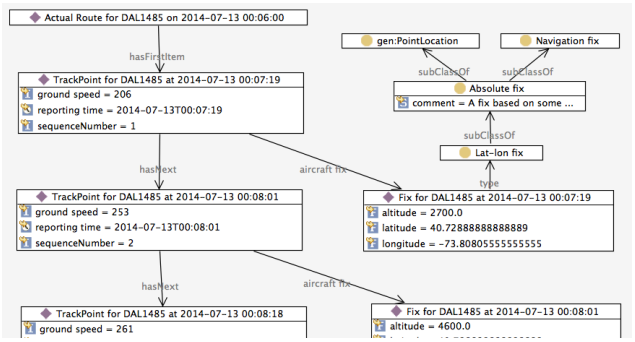

Figure 3: Ontology Representation of Aircraft Track Points

4.2.2 Triple reduction

When it is not possible to improve query performance due to the sheer size of the graph search space, it becomes necessary to rethink the underlying semantic representation and consider ways to reduce the size of the search space. In these cases, an unfortunate tradeoff between representational fidelity and efficiency must be considered. Consider an example from our ATM application domain, where much of the data volume is due to geospatial flight track information. Each flight is tracked by radar, and data is recorded about the position, altitude, and speed of each airborne aircraft every 30 seconds.[3,4,5] Within the triple store, each tracking point is stored as a `TrackPoint` instance. The `TrackPoint` records the airspeed, heading, and time as datatype properties, and location of the aircraft as an object property (`aircraftFix`) whose range is a `LatLonFix` instance (see Figure 3). The `LatLonFix` is a subclass of `PointLocation` and `NavigationFix`, and captures the aircraft's latitude, longitude, and altitude as datatype properties. Conceptually, this representation separates the temporally-dependent properties of the aircraft (e.g., speed, heading) at the track point from the static location in three-space through which the aircraft is passing (the `LatLonFix`).

With this representation, it would be possible to record multiple aircraft passing through the same `LatLonFix` at different times, or to record the weather conditions at that location at different times. However this representational advantage comes at a penalty in terms of query efficiency, because for each radar track point, two linked instances are instantiated and stored in the triple store (one `TrackPoint` instance and one `LatLonFix` instance). These instances represent a majority of all instances in our triple store (~70%).

An alternative and less costly representational approach – in terms of storage and query efficiency – would be to merge the location-specific datatype properties and the aircraft-specific datatype properties into one class. This approach would reduce the size of

[3] FAA operational systems sample radar signals at a much higher frequency: once per second.

[4] The volume of aircraft data is significantly greater than just the geospatial tracking data available to the FAA; operational aircraft generate volumes of high frequency systems health data.

[5] The average number of tracking points per flight in our Atlanta dataset is 156, which corresponds to a 78-minute duration flight.

the triple store by ~35%. However, this representation compromises fidelity and is less flexible and extensible.

Yet a different path to triple reduction would be to develop a hybrid approach, where the fidelity of the semantic representation is not compromised, but a subset of the data is removed and accessed outside the triple store – in secondary storage. For example, the sequence of `LatLonFix` points constituting a flight route and the boundaries of airspace sectors and regions could be stored in a geographic database optimized for spatial representations and computations. (Indeed, several graph database vendors offer specialized functionality to support geospatial data [18].) A similar argument could be made for storing certain types of multidimensional weather data external to the triple store.

5. RELATED WORK

Although there is little indication that "bleeding edge" semantic data management approaches have been adopted as yet by the inherently conservative aviation sector, the technology has begun to emerge in a small number of prototypes and experimental efforts. A recent workshop on Semantic Web for Air Transportation [19] uncovered ten active projects sponsored by government agencies, standards bodies, and private companies. The air transportation industry is acutely aware of data interoperability problems and has been actively pursuing more mainstream approaches for some time, publishing UML-based XML exchange model standards for weather, flight, and aeronautical information [20]. However, these models lack the formal logic underpinnings of ontologies and therefore cannot support data inference – a much-vaunted key capability of semantic approaches. In addition, the focus of these models is on system interoperability through the provision of a standardized, cross-industry exchange language. The more challenging problem of assembling heterogeneous data from multiple sources to create a single integrated data view is not directly addressed. Similarly, in [21] a system for integrating aircraft surveillance data from geographically-distributed signal receivers using RDF as a uniform data layer is proposed. But this work uses a triple store as a means of aggregating homogenous standardized data from multiple sources; there is no complex integration involved. There is little published work on ontologies in the air traffic management domain. van Putten [22] describes an ontology developed to model people, processes, and products associated with air traffic management. Although there is some conceptual overlap with our ontology, the scoping and scaling of this work is quite restricted, and is a mismatch to our specific requirements.

6. CONCLUSIONS

In this paper, we described an approach to constructing an integrated, semantics-based air traffic management data resource. Our prototype provides a proof-of-concept implementation, scaled down to a manageable scope covering one day's worth of commercial domestic flights arriving or departing a major US airport. If realized on a larger-scale basis, this data resource has significant potential to serve a broad community of aviation researchers, operational flight management personnel, policy makers, and beyond. There are many challenging tasks that must be addressed to realize our goals, but data scale-up is one of the most critical to our success. The burning question is whether semantic tools are up to the task, and whether big semantic data is practical for real-world, big data domains where complex queries must be issued and answered in an acceptable timeframe. Our experience thus far indicates that this question remains very much open; without further work to develop tools that support serious performance tuning, future prospects are in doubt.

7. ACKNOWLEDGMENTS

This work was funded by the Airspace Operations and Safety Program from within NASA's Aeronautics Research Mission Directorate. Eric Wang provided systems support for this effort.

8. REFERENCES

[1] IATA Safety Report 2014, Retrieved February 11, 2016 from International Air Transport Association web site: http://www.iata.org/publications/Pages/safety_report.aspx

[2] Eshow, M. M., Lui, M., and Ranjan, S. 2014. Architecture and capabilities of a data warehouse for ATM research. In *Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd*. IEEE.

[3] Oracle Application Express (APEX). From Oracle web site: https://apex.oracle.com.

[4] Noy, N. F. 2004. Semantic integration: a survey of ontology-based approaches. ACM *SIGMOD Rec.* 33, 4 (Dec. 2004), 65-70. DOI=http://dx.doi.org/10.1145/1041410.1041421.

[5] Doan, A. and Halevy, A. Y. 2005. Semantic integration research in the database community: A brief survey. *AI magazine* 26.1 (2005): 83. DOI= http://dx.doi.org/10.1609/aimag.v26i1.1801.

[6] ASDI: Aircraft Situation Display to Industry. From FAA web site: http://www.fly.faa.gov/ASDI/asdi.html.

[7] METAR: Meteorological Terminal Aviation Routine Weather Report. From NOAA web site: https://www.aviationweather.gov/adds/metars.

[8] FAA Advisories. From FAA Command Center web site: http://www.fly.faa.gov/adv/advADB.jsp.

[9] AllegroGraph, Retrieved February 16, 2016 from Franz Inc. web site: http://franz.com/agraph/allegrograph/.

[10] Uschold, M. and Gruninger, M. 2004. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.* 33, 4 (December 2004), 58-64. DOI=http://dx.doi.org/10.1145/1041410.1041420

[11] GraphDB, Retrieved February 16, 2016 from Ontotext web site: http://ontotext.com/products/graphdb/.

[12] Cray Urika-GD Graph Discovery Appliance, Retrieved February 11, 2016, from Cray corporate web site: http://www.cray.com/products/analytics/urika-gd.

[13] Zou, L., Mo, J., Chen, L, Özsu, M.T., and Zhao, D. 2011. gStore: answering SPARQL queries via subgraph matching. *Proc. VLDB Endow.* 4, 8 (May 2011), 482-493. DOI=http://dx.doi.org/10.14778/2002974.2002976

[14] Neumann, T. and Weikum, G. 2009. Scalable join processing on very large RDF graphs. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (SIGMOD '09), Carsten Binnig and Benoit Dageville (Eds.). ACM, New York, NY, USA, 627-640. DOI=http://dx.doi.org/10.1145/1559845.1559911

[15] Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., and Boncz, P. 2012. Heuristics-based query optimisation for SPARQL. In *Proceedings of the 15th International Conference on Extending Database Technology* (EDBT '12), Elke Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari (Eds.). ACM, New York, NY, USA, 324-335. DOI=http://dx.doi.org/10.1145/2247596.2247635

[16] Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. 2013. A distributed graph engine for web scale RDF data. *Proc. VLDB Endow.* vol. 6, no. 4 (February 2013), 265-276. DOI=http://dx.doi.org/10.14778/2535570.2488333

[17] Giese, M., Soylu, A., Vega-Gorgojo, G., Waaler, A., Haase, P., Jimenez-Ruiz, E., Lanti, D., Rezk, M., Xiao, G., Ozcep, O., Rosati, R. 2015. Optique: Zooming in on Big Data, *Computer*, vol.48, no. 3, pp. 60-67, Mar. 2015, DOI=10.1109/MC.2015.82

[18] Patroumpas, K. et al. 2013. Market and Research Overview, GeoKnow EU/FP7 project deliverable 2.1.1. Retrieved February 16, 2016 from GeoKnow web site: http://svn.aksw.org/projects/GeoKnow/Public/D2.1.1_Market_and_Research_Overview.pdf

[19] Keller, R and Kaplun, M (eds.), Semantic Web for Air Transportation Meeting, electronic proceedings, Retrieved February 11, 2016 from FAA Service Semantics web site: https://www.faa.gov/nextgen/programs/swim/governance/servicesemantics/#SWAT-Special-Interest-Group

[20] Bradford, S., Humbertson, R., Perper, A. 2014. Global harmonisation through information. *Journal of Airport Management*, vol. 8, no. 2, pp. 119-128.

[21] Klauza, M., Czekalski, P., Tokarz, K. 2015. Air Traffic Data Integration using the Semantic Web Approach. Athens Journal of Technology & Engr., vol. 2, no. 2, pp. 115-128

[22] van Putten, B-J, Wolfe, S, Dignum, V. 2008. An Ontology for Traffic Flow Management, in *Proc. 8th Aviation Technology, Integration, and Operations Conference*, AIAA.

9. APPENDIX

9.1 Sample SPARQL Queries

F1: Find Delta A319s departing ZTL airports on 9/8/12

```
SELECT ?flight ?departYear ?departMonth ?acModel
WHERE {BIND (nas:ZTLcenter AS ?center) . BIND
(2012 AS ?departYear) . BIND (9 AS ?departMonth) .
BIND(eqp:A319AircraftType AS ?modelType) .
BIND (nas:DALairline AS ?operator) .
 ?flight a atm:Flight .
 ?flight atm:actualDepartureDay ?departDay .
 ?departDay nas:calendarYear ?departYear .
 ?departDay nas:calendarMonth ?departMonth .
 ?flight atm:operatedBy ?operator .
 ?flight atm:aircraftTypeFlown ?modelType .
 OPTIONAL {?flight atm:aircraftFlown ?ac .
         ?ac eqp:hasAircraftModel ?acModel .} .
 ?flight atm:departureAirport ?departAirport .
  {{?tracon nas:hasLOAwith ?departAirport .
    ?tracon nas:hasLOAwith ?center .} UNION
  {?departAirport nas:hasLOAwith ?center .} . } .}
```

F3S: Find rainy departures from ATL on 9/8/12

```
SELECT DISTINCT ?flight
WHERE {BIND (nas:KATLairport AS ?departAirport) .
 ?flight a atm:Flight .
 ?flight atm:departureAirport ?departAirport .
 ?flight atm:actualDepartureDay ?departDay .
 ?flight atm:actualDepartureTime ?departTime .
 ?departAirport data:hasMETARreport ?metarRpt .
 ?metarRpt data:metConditionStartDay ?departDay .
 ?metarRpt data:metConditionStartTime ?rptTime .
 ?metarRpt data:hasWeatherCondition ?wxcond .
 ?wxcond data:weatherPhenomenon "rain" .
 BIND (hours(?departTime) AS ?dptHour) .
 BIND (minutes(?departTime) AS ?dptMins) .
 BIND (hours(?rptTime) AS ?rptHour) .
 BIND (minutes(?rptTime) AS ?rptMins) .
 BIND (((?rptHour * 60)+ ?rptMins)AS ?totRptMins).
 BIND (((?dptHour * 60)+ ?dptMins)AS ?totDptMins).
 BIND (abs((?totRptMins - ?totDptMins))
    AS ?reportingLagMins) .
 FILTER (?reportingLagMins < 30) .}
ORDER BY (?flight)
```

S4: Find which sector controlled the most
flights during a given hour

```
SELECT DISTINCT ((COUNT(?flight)) AS ?totalFlight)
       ?sector
WHERE {BIND (2 AS ?hourExamined) .
    ?flight atm:hasActualRoute ?route .
    ?route gen:hasSequencedItem ?posn .
    ?posn atm:reportingTime ?time .
    ?posn atm:aircraftFix ?fix .
    ?fix atm:locatedInSector ?sector .
    FILTER (hours(?time) = ?hourExamined) .}
GROUP BY ?sector
ORDER BY DESC (?totalFlight)
```

S1S: Find flights passing through a given sector
within 30 minutes of each other

```
SELECT DISTINCT ?flight1 ?flight2
WHERE {BIND (nas:ZTLsector040 AS ?sector1) .
    ?flight1 atm:hasActualRoute ?route1 .
    ?flight1 atm:callSign ?callSign1 .
    ?flight1 atm:actualDepartureDay ?departDay .
    ?route1 gen:hasSequencedItem ?posn1 .
    ?posn1 atm:reportingTime ?time1 .
    ?posn1 atm:aircraftFix ?fix1 .
    ?fix1 atm:locatedInSector ?sector1 .
    ?flight2 atm:hasActualRoute ?route2 .
    ?flight2 atm:callSign ?callSign2 .
    ?flight2 atm:actualDepartureDay ?departDay .
    ?route2 gen:hasSequencedItem ?posn2 .
    ?posn2 atm:aircraftFix ?fix2 .
    ?posn2 atm:reportingTime ?time2 .
    ?fix2 atm:locatedInSector ?sector1 .
BIND (hours(?time1) AS ?rptHour1) . BIND (minutes
(?time1) AS ?rptMins1) . BIND (hours(?time2) AS
?rptHour2) . BIND (minutes(?time2) AS ?rptMins2) .
BIND (((?rptHour1 * 60) + ?rptMins1) AS
?totRptMins1) . BIND (((?rptHour2 * 60) +
?rptMins2) AS ?totRptMins2) . BIND (abs
((?totRptMins1 - ?totRptMins2)) AS ?deltaTime) .
FILTER ((?deltaTime < 30) &&
           (?callSign1 < ?callSign2)) .}
ORDER BY (?flight1) (?flight2)
```

9.2 Benchmark Timing Statistics

Query #	Execution Time in Milliseconds				Scale Factor	
	2.4M triples		36M triples		36M/2.4M ratio	
	Store #1	Store #2	Store #1	Store #2	Store #1	Store #2
A1	49	197	53	210	1.08	1.07
A2	36	176	37	147	1.03	0.84
A3	12	37	8	31	0.67	0.84
F1	98	64	2584	324	26.37	5.06
F2	36	28	298	96	8.28	3.43
F3S	466	482	12462	5070	26.74	10.52
S1S	1033	4749	726565	1651215	703.35	347.70
S2	11	858	59	19363	5.36	22.57
S3	1844	6060	35500	115389	19.25	19.04
S4	1786	4991	34985	108882	19.59	21.82
S5	1096	1412	11170	31199	10.19	22.10
S6	4825	9640	96846	163205	20.07	16.93
T1	32	43	269	171	8.41	3.98
T2	11	28	8	42	0.73	1.50
T3	193	68	268898	259	1393.25	3.81
W1	11	33	426	130	38.73	3.94
W2	11	37	11	39	1.00	1.05