24th International Command and Control Research and Technology Symposium

29-31 October 2019
Laurel, Maryland, USA

Track 8: Information and Knowledge Systems

Microservice API design to support C2 semantic integration

Andrew Zschorn
Joint and Operations Analysis Division
Defence Science and Technology Group
West Ave
Edinburgh, South Australia 5111

Hing-Wah Kwok
Joint and Operations Analysis Division
Defence Science and Technology Group
West Ave
Edinburgh, South Australia 5111

Dr Wolfgang Mayer
School of Information Technology and Mathematical Sciences
University of South Australia
Mawson Lakes Campus
Mawson Lakes, South Australia 5095

Microservice API design to support C2 semantic integration

Andrew Zschorn[1], Hing-Wah Kwok[1], Wolfgang Mayer[2]

[1] Joint and Operations Analysis Division,
Defence Science and Technology Group Australia
[2] University of South Australia, Adelaide
{andrew.zschorn,hing-wah.kwok}@dst.defence.gov.au, wolfgang.mayer@unisa.edu.au

Operational Headquarters (HQ) need to maintain situation awareness and command and control (C2) in an increasing variety of operations. As the operational focus of HQ changes over time, so too do the information requirements of HQ staff. C2 and decision support systems need to be highly flexible to support this level of information variety. Achieving 5th Generation HQ capabilities will place higher still demands for organizational agility, which must be supported by agile information systems. We use the term agile semantic integration for the process of timely deployment of new information sources into operational HQ, such that it is amenable to human and machine reasoning, and integrated with existing HQ information sources. We propose a microservices architecture, and review technology implementation choices for information access and integration, focusing in particular on Application Programming Interface design, and the agility and maintainability of the system.

Introduction

Joint operations headquarters (HQ) staff are charged with maintaining situation awareness of, and conducting operations in, a variety of locations and contexts, including: military operations; peacekeeping; national security operations; humanitarian assistance and disaster relief efforts; and assistance to the civilian community during crises, for example, extreme weather events. The information requirements of Defence operational staff are thus many, varied and changing, and sometimes unpredictable.

These various operations require at times communication and coordination with coalition military partners, federal and state police forces, other government agencies, and non-government agencies. Open-source information could also inform operational decision making at different times and in different contexts. HQ information systems should then be capable of absorbing information from systems owned by these agencies and open-source information. Moreover the information needs to be searchable and presented in an integrated way with HQ-owned or other highly trusted information.

In the absence of software tools that can abstract these complexities, staff must access and search siloed systems with different user interfaces and manually correlate data from separate search results. Automating some of these information integration gaps can potentially relieve staff of some of this burden, freeing up more time for tasks that benefit from human creativity and insight.

Information would be more easily integrated if it followed a common schema or ontology. However, while operational HQ staff are consumers of a broad range of datasets, they ought not to be data owners or managers for all such datasets because of the extra burden this would entail. Mandating a common schema or ontology for all datasets across the enterprise in support of information integration is a considerable information modelling challenge. And such a schema would not address integration of datasets developed outside the organisation, and may even promote overly rigid adherence rather than flexibility and agility in information systems.

5th Generation Operational Level Military Headquarters

Yue, Kalloniatis et al. (2016) present a concept for a 5th Generation Operational Level Military Headquarters. The authors lay out a vision for what capabilities must be realised inside future operational headquarters, especially for middle powers such as Australia, if they are to effectively serve and command militaries given foreseeable stressors of diverse types of operations, increased operational tempo, and increasing amounts and variety of required information.

Two advanced capabilities of 5th generation HQ are: the automated smart push and smart pull of information; and a robust and flexible distribution of situation awareness across the human and synthetic agents that constitute the HQ. Smart push and smart pull are capabilities that would allow staff to be notified of, or efficiently retrieve or share any piece of information currently relevant to them or their role, in the face of the deluge of all available information. To realise these capabilities, we need machines that can reason sensibly and contextually with information, with minimal oversight from HQ staff. A quality of 5th generation HQ is organisational agility; different operations and adversaries may require different internal HQ organisational structures, workflows, and information flows. To achieve this organisational agility the information systems supporting the organisation must be equally agile.

We believe that fielding information integration capabilities into HQ information systems would be transformative for the speed and quality of staff situation awareness and decision making. The Defence Science and Technology (DST) Group of Australia is investigating software architectures and development processes that will support the rapid prototyping of information access and information integration datasets and tools.

Saki architecture

DST is developing the Saki system as a prototype microservice platform to demonstrate technologies and elicit the needs that operations staff have for efficient, flexible decision support systems. The prototype system is aimed at improving situation awareness, course of action development and plan development activities where staff need to search for or visualise a variety of information. For example, searching for populated places, civilian and military infrastructure, or military platforms, and then visualising this information on a geospatial display. It is designed to save staff time by presenting a single, web-based interface to search multiple data sets. By speeding-up these frequent information retrieval tasks and widening the data coverage that is readily available, we also anticipate that staff will be able to explore more planning options.

Saki's underlying information system architecture has been designed to be flexible and support the rapid addition of new data sets. An important feature of Saki is that it is designed to flexibly ingest datasets in those formats and schema chosen by the data set owners, rather than trying to impose the use of a single, consistent schema. Saki is currently focussed on serving structured and semi-structured datasets, such as relational databases, spreadsheets, and single tables in CSV, XML or JSON format.

Microservice approaches are an evolution of Service-Oriented Architectures, where the engineering emphasis has shifted from few, multi-function services to having many, single-function services. As the services are less complex, the time and resources required to maintain and upgrade any individual service should be reduced, and this can be done independently of upgrade cycles of other microservices. There are more services to deploy and monitor in this case, but there are now many deployment, orchestration and monitoring tools developed for

cloud native computing[3] to support these common tasks. A smart networking layer such as a service mesh can perform service registration and discovery, authentication and authorization, and potentially circuit breaking and load balancing, to further reduce the complexity of microservices and separate these cloud native, resilience functions from the microservice business logic. As these tools grow in maturity and levels of automation, microservice architectures can potentially deliver highly scalable and resilient software deployment that should simplify the work of system developers and maintainers.

We anticipate that a heterogeneous mix of technologies is needed to address a task as multi-faceted as situation awareness for military operations. For example, relational databases, document stores, triple stores and are all likely to play different data storage roles across the system. As microservice architectures achieve integration at the level of web APIs (Application Programming Interfaces), they have good support for deploying different technologies as each microservice's task dictates.

While the Saki architecture supports accessing a range of information sources from a single user interface, to date the system cannot automatically integrate information from different data sources. Information duplicated across sources can create confusion. For example, if an open source and an internal airport database are both available on the system, then searches for airport names will return two, separate results, and the same location on the map will be highlighted twice. In addition, if both an airport database containing runway properties and an aircraft database containing runway requirements are available, staff must examine data from both sources to reconcile which aircraft can land at which airports.

A proposed extension to the Saki microservice architecture that can address these information integration challenges is shown in Figure 1. Briefly, in the data services layer, each data source has a microservice interface wrapper or proxy to enable point-to-point, web-based access to the data, with optional access control. Such datasets include sea-port and airport names and locations, feeds of emergency and weather events, and names and locations of population centres. Data service APIs support record-wise access and data is locally indexed for keyword, facet and geospatial search. Data services may be conceptually clustered within information domains – example datasets are shown for the Airport domain. For each domain there is a data integration layer that consults all the data services in that domain, integrated that data, and again makes the integrated product available for record-wise access and keyword search via an API. Services at the information integration layer consult and interlink data from across domains. The integrated information is again available via an API for record-wise and search access. Additionally, the integrated information can be pushed into a knowledge graph for long-term storage and retrieval by user-interface components. Access to the knowledge graph is mediated by adapter services to allow a level of schema/ontology encapsulation and enforcement of business rules such as metadata retention, to support information retraction or update. These steps happen in batch mode, or when new datasets become available or are updated.

[3] https://landscape.cncf.io/

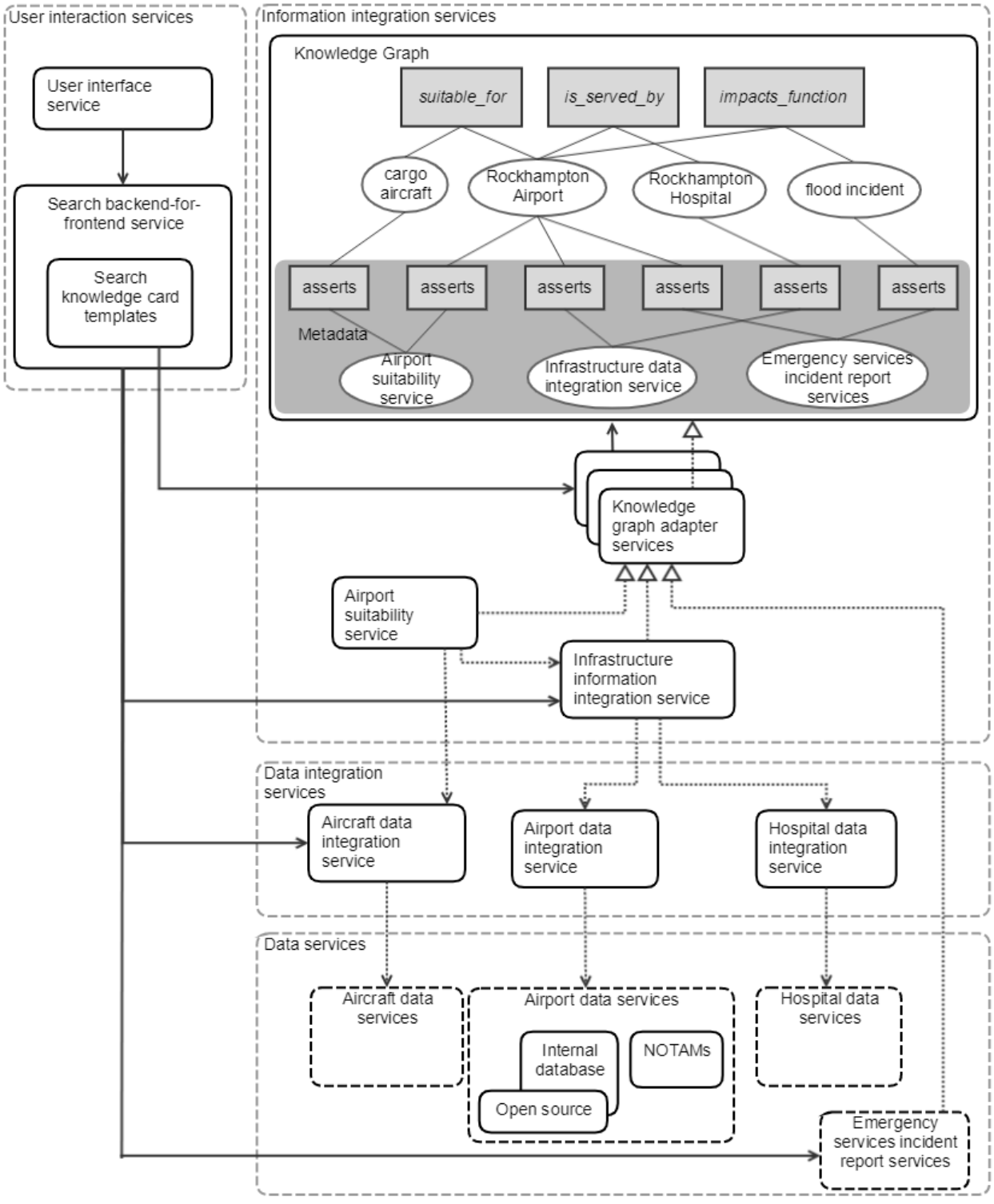
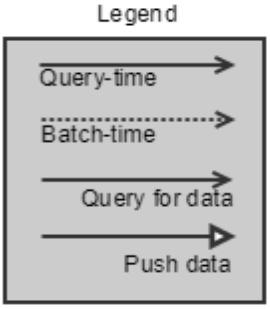


Figure 1 The proposed system architecture.



When a user query comes in through the user interface service, a search backend-for-frontend service sends that search on to a set of data integration services and data services that covers all the data in the system. In this example the infrastructure data service is searched for both airport and hospital information, so there is no need to also consult the airport and hospital data integration services. After ranking the results, the Search backend-for-frontend service uses a template corresponding to the top-ranked result and queries the knowledge graph for the parameters that template requires to build a knowledge card to accompany the display of search results.

Web search engines have reached their current high level of speed and utility by using algorithms such as PageRank and machine learning techniques. These approaches rely on the interlinked, crawlable nature of the open Web, and require huge amounts of data about user search terms and click behaviour. In the corporate context, including operational HQ, data almost never arrives interlinked, and the amount of data we could collect about staff search behaviour is relatively limited. On the other hand, in a corporate context we can exploit a much greater understanding of staff and enterprise roles and intent to improve search utility.

Agile semantic integration

Semantic integration is the process of taking information from multiple isolated sources, and information about staff roles and intent, and semi-automatically reproducing the information in a way that makes it easier for staff to form a more relevant, coherent picture of a situation. In general, isolated information sources can contain identical, complementary or inconsistent information about the same real-world entities. And relationships that hold between entities within and across information sources that may be relevant to staff intent can be missing or tacit. Manually navigating and reconciling this information is time consuming and error prone. We seek to prototype data integration and information integration capabilities that make semantic integration more agile, in terms of aiding the addition and removal of datasets as operations require.

Data integration

We wish to present users with a single, integrated view across multiple datasets, and in general such datasets may cover the same domain and thus contain overlapping information. With this prototype we want to be able integrate multiple datasets so staff can use a single interface to search across them all. This is sometimes known as data integration or level one data fusion.

In certain situations open source and crowd-sourced datasets may become relevant to HQ information requirements. In this case we would like to take advantage of such information sources by integrating it with internal data sources. Of course, open source information is in general of lower quality than internally developed and trusted information, thus the quality of integrated data will be uneven, and so provenance information of each record and field needs to be available to staff so they may judge the trustworthiness of the integrated information.

Information integration and knowledge graph

The system should support staff to work directly with the military operations concepts they prefer, and the system should offer information expressed in those same terms. In 5th generation headquarters we expect that machines must be capable of flexibly performing parts of the work of maintaining situation awareness task (Yue, Kalloniatis et al. 2016). This requires that the system is able to sensibly map information from low-level databases into higher level military operations concepts. Such mappings should results in a path of logical semantic relations holding between entities represented within and across databases. Asserting semantic

relations between entities is sometimes known as information integration, level two data fusion, or situation assessment. Following Lambert (2001), this level of data fusion is analogous to Endsley's (1995) second component of human situation awareness, comprehension.

For example, an airport database may contain runway length and geo-location information, but it would be more useful for HQ staff to know which type of aircraft can use which airports. Another example is when searching for an airfield, HQ staff should be immediately prompted if any current NOTAMs (notices to airmen) are associated with that airfield, as they may contain information that would rule airfields out of planning consideration, such as if runways are closed. In another example, staff may have access to an airport location database and a hospital location database and want to know which hospitals are served by, or close to, which airports.

This entails embedding such concepts into information integration services, and using them to describe relationships between entities, or records, found in the system across the data integration services. In these examples the data sets would be airport databases, aircraft capability databases NOTAM repositories and hospital databases. The relations may be *is_suitable*, *is_served_by* and *impacts_function*.

A knowledge graph is a way of conceptualising, and potentially encoding, semantic relations that hold between entities represented in an information system. Knowledge graphs are an appropriate approach for tackling semantic integration because of the ease with which they can span different domains of knowledge, and they can be incrementally extended as needs evolve. Knowledge graphs can provide a central, long- lived repository where data and metadata such as inferences, quality, provenance, and other expensive calculations can be stored in context. Additionally, it can also store human validations, corrections and contributions.

While the term is more widely used, in the Semantic Web community knowledge graphs necessarily adhere to formal logic systems, which means they are appropriate for automated reasoning tools to detect inconsistencies or calculate reachability and inferences. The term knowledge graph has also been applied to informal structures, for instance as repositories of products from data mining and natural language-based techniques for entity and relationship extraction. These types of knowledge graphs do not follow formal systems, and thus cannot be applied to automated reasoners. However, they can still be useful representations for human users.

Linked Data (Bizer, Heath et al. 2011) is a Semantic Web movement that promotes semantic integration between open data sets on the web. Such data sets can be considered a web-based distributed knowledge graph. This approach is thus instructive for semantic integration in our context, and such open datasets may be more easily integrated into corporate systems, depending on our technology choices.

It should be noted that knowledge graphs are unlikely to be a suitable technology for all types of information or queries. For example, numeric, temporal and spatial relations are not easily represented in current Semantic Web knowledge graph techniques.

Agility and maintainability

In a microservice architecture, many services may need to access any one dataset. However, as operational needs evolve, changes may be required to the way the information in a dataset was initially modelled. Thus, a key challenge for agile semantic integration in this context is how to retain the benefits of the divide-and-conquer approach of microservice architectures, while still making new and remodelled information available to staff in a timely fashion. We need to avoid as much as possible triggering time-consuming cascades of re-engineering effort each time changes are made to information models.

One way of tackling this problem would be to commit to a model, schema or ontology that covers all possible areas of operations and have all microservice components exchange information using that model. But we contend that:

- this can work for core C2 concepts that are common across operations, but that also unanticipated domains of knowledge will be relevant for some operations;
- any such model will still have to undergo revisions after it is released, and those changes introduced into running C2 support systems, thus these systems should still be built with sufficient flexibility to accommodate such changes with minimal effort and down time;
- and building and extending such a model requires modelling specialists that have detailed understandings of C2 concepts and processes, and the small numbers of suitable skilled staff means that their rate of output is unlikely to ever match the tempo of changes required by an operational HQ.

We don't believe that we can achieve decision support system agility through early commitment to a common model.

The Network-Centric Warfare community adopted the approach of Communities of Interests (COIs) to divide-up problems of data exchange and interoperability. The C2 domain has thus far proved to be too broad for any one COI, and COIs have been considered unsuccessful in achieving information interoperability across COI boundaries (Meyerriecks, Davis et al. 2008, Winters and Tolk 2009). As we believe it impractical to create a priori a model that covers all of the concerns of an operational HQ, we also believe it is impractical to pursue a data warehousing approach to address information integration problems in an operational HQ.

Architectures using data lakes have also been proposed to address big data variety problems (Auer, Scerri et al. 2017). Data lakes are, like data warehouses, central repositories for all an enterprise's data to be managed and updated, but unlike data warehouses, data lakes accept denormalised data that can adhere to a range of different schema. Thus, queries must be written to account for the different schema where information from different sources needs to be brought together at query time. In contrast to both these approaches, a microservices approach such as ours has more in common with federated query architectures. In this approach the data isn't moved to a central repository, instead data can be managed closer to their respective sources and made available for querying via standard interfaces. As noted by Stumptner, Mayer et al. (2015), moving or copying all data into centralised data lakes can create complex data governance issues.

Meyerriecks et al. (2008) encourage collecting operational data-sharing metrics to assess and improve utility and performance of net-centric C2 information exchange. One way of tracking information usage throughout a system would be to build and transmit provenance metadata along with the information itself at each information processing step. Having this complete information would give system architects and developers awareness of the impact to the system of any improvements to information representation or system behaviour. However, collecting provenance metadata in this way can lead to increased volumes of information needing to be transmitted, and it is not a straightforward problem to automate the accurate generation of this metadata (Dimou, De Nies et al. 2016). Taking a microservice approach, we would want to abstract away most of the complexities of the whole system from the range of concerns for a developer seeking to make decisions to improve a single microservice. Another, potentially complementary, way of collecting metrics is be for each microservice to keep track of which records and which fields in those records are in use by other microservices. We expect that collecting metrics can potentially be useful for supporting evolving microservice architectures by allowing engineers to prioritise those parts of the models and workflows with the greatest impact on HQ staff functions.

Related work

The Integrated Law Enforcement project proposes a related architecture for federated querying and integration of information contained in data silos in the law enforcement domain (Stumptner, Mayer et al. 2015, Mayer, Stumptner et al. 2017). This architecture features a Curated Linked Data Store to hold confirmed facts and metadata about entities, where these entities may be external to the system. In this case there is adoption of a common ontology to support information integration. The authors anticipate handling small changes to the ontology, which may necessitate changes to many components of the system, by using ontology matching techniques or machine learning methods for information extraction and linking.

The DIG (Domain Insights Graph) system (Knoblock and Szekely 2015, Szekely, Knoblock et al. 2015) is used by investigators of human trafficking. It uses information scraped from websites to build a knowledge graph of images, phone numbers, locations and names. The websites DIG targets use obfuscated and frequently changing syntax and language, in order to avoid traditional policing efforts. So DIG supports rapid deployment of new parses and information extractors to keep up with these changes. The lightweight semantic annotation JSON-LD[4] (JavaScript Object Notation for Linked Data, discussed below) is used throughout to assist record linkage and other semantic integration tasks. Maintaining a chain of attribution from the knowledge graph back to source websites is important for law enforcement. DIG uses the PROV-O W3 standard (Lebo, Sahoo et al. 2013) to encode this information; the same mechanism is used to store the pipeline of parses and information extractions that produces each knowledge graph assertion. The knowledge graph can then be visualised and queried directly by investigators, or it can be transformed into input for analytics tools internal or external to the DIG system, to better support a wider range of investigators' preferred workflows.

Google builds a knowledge graph as it is crawling web and social media pages to update its search engine index. There is little information available in the open literature about the technical details of how the Google knowledge graph is constructed (Paulheim 2017), but one method seems to be by collating JSON-LD annotations that web page authors embed in web page code. These annotations re-state the information in the page in a machine-readable format using the well-known Semantic Web classes and properties published by schema.org[5]. It also seems that this information is linked to Wikipedia pages and multimedia content, perhaps by using the wikidata.org and/or dbpedia.org Semantic Web counterparts of those pages. The resulting knowledge graph can be queried via a RPC-style web API[6], and returns information in JSON-LD format also. Google uses the knowledge graph itself to augment certain search results pages with a knowledge card, which is displayed as a box in the right-hand side of the page and contains images, links, and text summaries connected to knowledge graph entities relevant to the search terms[7]. For some web searches displaying semantically connected information is arguably closer to meeting a web user's search intent than a presenting long list of pages containing that search string.

Ontology-Based Data Access (OBDA) is a well-researched architectural style for decision support systems requiring information integration (Heyvaert, Dimou et al. 2017). In this approach a common Semantic Web ontology is constructed to serve as a lingua franca to integrate information from a set of disparate, structured online information systems, most commonly relational databases. For each of these information systems a translation is defined

[4] https://json-ld.org/

[5] https://schema.org/

[6] https://developers.google.com/knowledge-graph/

[7] As shown in this search: https://www.google.com/search?q=RAAF+Base+Amberley

between its schema and the common ontology. Queries expressed in the common ontology are re-written as queries for execution in one or across multiple information systems, and the results re-written back into the common ontology, collated and unified. As most OBDA approaches have been implemented in the context of current Semantic Web technologies, i.e. RDF and OWL, the expressiveness of existing systems is limited to description logics. In particular, in the C2 domain, reasoning about spatial, temporal and uncertainty information and meta-data is important (Matheus, Kokar et al. 2003, Winters and Tolk 2009, Deitz, Michaelis et al. 2016). However, the question of how to best represent and reason about this kind of information in ways compatible with Semantic Web standards is still an open question (W3C 2004, Nguyen, Bodenreider et al. 2014, Moreau, Groth et al. 2015, Giménez-García, Zimmermann et al. 2017).

Work continues on extending and improving the Multilateral Interoperability Project (MIP) Information Model (MIM) (Schmitz and Gerz 2016). MIM has been developed as a standard for coalition systems to be able to exchange information in the C2 domain to reduce ambiguity and information loss between systems. It consists of a UML model and also rules expressed in Object Constraint Language. It is capable of representing metadata such as source and uncertainty. However, as it is expressed in UML it has no formal semantics. And as it is does not follow Semantic Web standards it is not natively compatible with Linked Data.

API and metadata requirements

Above we have discussed a case for building systems for semantic integration in microservice architectures to support C2 situational awareness in an operational HQ context. In this section we discuss the features required of microservice APIs to support semantic integration.

In our architecture data services have three responsibilities: data provision, in a manner that supports data integration, discussed below; on-request updating of the data from the originating source; and search functions to support runtime queries from staff.

Data integration is often implemented in three phases: schema or ontology alignment; record linkage; and record fusion. Thus, data services' Web APIs need to support retrieval of their data schema, ontology and other meta-data - in a format suitable for consumption by human developers and machine data integration services - to assist hard-coded, semi-automatic or fully automatic schema and/or ontology alignment. The result of schema or ontology alignment is a mediated schema and mappings from each dataset schema into the mediated schema. Record linkage is the process of identifying which records in two or more datasets refer to the same real-world entity. It is often implemented using the so-called blocking approach, or more recently using metric space indexing (Akgün, Dearle et al. 2018). In either case access to entire datasets is required and the simplest design choice is to have the data integration services read in its own copy of datasets from data services, so that it is free to impose implementation-specific blocking or index metric functions. Record fusion is the process of mapping the data fields from linked records into a single record expressed in terms of the mediated schema. Where datasets conflict in field values, the data integration service can choose the value from the more accurate dataset, which in our context we interpret as the most trusted, up-to-date dataset. To make this decision, the data services need to serve metadata that covers at a minimum the provenance or authorship of the data, and timestamps for when the dataset was updated from the source and when the information is expected to expire.

To avoid expensive re-evaluations of data integration, data services should preserve record identifiers when they update their records from the originating source and also track when their records have changed, or been added or deleted. To re-evaluate incrementally only changed records, data integration services can then use as a parameter the timestamp of when

```
/airport
    description: Airport records
    get:
        description: Page through summaries of all records
        query_parameters:
            changed_since:
                description: Only records that have changed since this time
        responses:
            200:
                body:
                    application/json:
                        example:
                            {
                                "metadata":
                                    "source": {...},
                                    "updated": "20190403T...",
                                    "changed": "20190403T...",
                                    "expires": "20191003T...",
                                "airports":
                                    [
                                        {
                                            "id": 48,
                                            "name": "Rockhampton Airport",
                                            "url": "https://.../airport/48",
                                        },
                                        ...
                                    ],
                                "next_page": "https://.../airport/?..."
                            }
    /{airport_id}:
        description: Get the full record of a particular airport
        get:
            responses:
                200:
                    body:
                        application/json:
                            example:
                                {
                                    "metadata":...
                                    "airport":
                                        {
                                            "id": 48,
                                            "name": "Rockhampton Airport",
                                            "url": "https://.../airport/48",
                                            "location": "POINT(138.5 -34.9)",
                                            ...
                                        }
                                }
/facets:
    description: Get information about search facets
/search:
    description: Perform a search across all records
    get:
        query_parameters:
            keyword_expr
            facet_expr
            geo_expr
```

Code Listing 1: Example data service API for an airport data set.

they last queried the data service and receive only the records that have changed since that time.

To support on-request updating of their data, data services simply need either an API entry point to trigger this function and implement their own logic for downloading or otherwise requesting the original data set, or alternatively to accept as a parameter the dataset itself.

To support runtime queries from staff, data services should have a search entry point in their API that accepts any combination of Boolean keyword expressions, facet expressions and geospatial constraints. At this stage the geospatial constraints we consider are only: that results lie within a given great-circle distance from a given longitude, latitude point, or within a given bounding box, expressed as a pair of longitude, latitude points; or ordering of results by distance from a given longitude, latitude point. In order for the search backend-for-frontend (BFF) service to inform the construction of a user interface that supports input of appropriate facet constraints, data services will also need an entry point that describes the facets and their possible values.

Data and integration services are layered, with each layer increasing the domain span of the data they integrate. Thus data and information integration services should have at least the same API entry points as data services. To perform searches the BFF should consult with the highest level integration services that covers the whole dataset, or directly with a data service if no such data integration service yet exists. This approach means that the responsibility for ranking search results within data domains is with the relevant data integration service. This API is sketched out in Code Listing 1.

As the knowledge graph deals with a different shape of data representation, with more expressive capability, than data services, a tree-shaped REST API isn't a convenient way to interact with this service. Rather, query languages such as SPARQL or Cypher can be used to both query and assert information into the knowledge graph. Alternatively, GraphQL, or JSON-LD Framing can be used to specify an explicit graph-to-tree mapping layout. These options are discussed further below.

Whenever a service asserts new information into the knowledge graph, it must also add metadata including the provenance, creation time and expected expiration data. This information will make it possible for staff to flag incorrect data, and for system developers to trace back to the underlying incorrect integration algorithms or dataset.

API technology choices for Semantic Integration

In this section we discuss four modern technology choices for implementing API and query interfaces.

Web APIs & RESTful Services

Web Application Programming Interfaces (APIs) and its specialisation to Representational State Transfer (RESTful) Services are two standards widely adopted in the construction of modern, lightweight Service Oriented Architectures (SOAs). These standards are small extensions to the HTTP standard, to support point-to-point Remote Procedure Call (RPC) interfaces. HTTP is of course stable, widely understood, and widely implemented in many tools including web browsers, which lowers barriers to adopting these techniques. RESTful services are Web APIs with further constraints on their behaviour, which render their behaviour more predictable and widely understood. One such constraint is the strict use of URIs to represent every entity and action known to or controlled by the service. This can be exploited to realise Hypermedia As The Engine Of Application State (HATEOAS) – a feature that enables the service itself to offer guidance towards sensible interactions, in the form of the set of possible or likely subsequent URLs given the current state. Both Web APIs and RESTful services typically exchange data in JSON format, and typically also don't publish a data schema or API behaviour contract. This means that, while the HATEOAS mechanism is useful for developers to manually explore APIs, there is little or no automated compile-time error checking for development of consuming services, and this also makes the task of schema alignment more difficult. Despite this apparent

```
1.  aqa:SufficientCandidateAirports
2.      a sh:NodeShape ;
3.      sh:targetNode dbo:Airport ;
4.      sh:property [
5.          sh:path [ sh:inversePath rdf:type ] ;
6.          sh:qualifiedValueShape aqa:CandidateAirportShape ;
7.          sh:qualifiedMinCount 400 ;
8.          ] ;
9.      sh:message "Too few candidate airports to proceed." ;
10.     .
```

Code Listing 2. A SHACL rule, expressed in Turtle syntax, to verify that an input data graph contains at least 400 candidate airports. Note in particular the idiomatic targeting of the dbo:Airport superclass node (line 3) and then tracing back all rdf:type links (line 5).

shortcoming, these two simple API styles are currently proving more popular in the software development community compared to older and more sophisticated, but XML-heavy, SOAP and WSDL standards (Pautasso, Zimmermann et al. 2008).

Automatically maintaining awareness of data service usage via Web APIs or RESTful services is imperfect. Because of the HATEOAS, it is straightforward to maintain metrics of the frequency of access of URIs. However, responses are resource descriptions defined by the data service developers at compile time, so it isn't possible to track exactly which fields included in the response are important to consumers.

API versioning allows a Web Service to support system evolution by helping to maintain backwards compatibility. There are various ways of achieving this, for example the method proposed in support of Australia's future Consumer Data Right legislation (Data61/CSIRO 2018). In this way, data services can automatically maintain metrics about which versions of the API are in use; but again, not to the level of response fields.

Another benefit of Web APIs is the existence of mature caching approaches and tools, which can be used to reduce network traffic and increase system responsiveness.

These standards do not address the semantics of the data they provide. It is up to the human consumer, or human developer of the consuming microservice, to read semantics into the purely syntactic responses, and this work needs to be done individually for each data service available. This leads to highly complex, fragile consumer services, and does not scale well as the number of microservices increases.

Semantic Web, Constraints & Rules

Semantic Web ontologies and vocabularies, in particular Linked Data approaches, directly address the concern of using common, semantic representations for information exchange. This aids the task of semantic integration as: ambiguities are removed; developers need become familiar with only the ontologies in use, rather than deal with a plethora of idiosyncratic information representation choices; and automatic reasoning can be used to verify desired characteristics of the input data and resulting knowledge graph.

Semantic Web Protocol and RDF Query Language (SPARQL) endpoints are often used as interfaces to Semantic Web data services. SPARQL provides a standardised, richly featured interface for querying and inserting knowledge into RDF graphs. In this case the SPARQL endpoint can keep local, accurate metrics on which of its data records are in active use, which can be used to plan changes and hence support an evolutionary architecture. The exception to this is the *DESCRIBE* SPARQL operation, for which the data service determine the response. This

```
1.  evac:HospitalIsServedByAirportRule
2.      a sh:NodeShape ;
3.      sh:targetClass dbo:Hospital ;
4.      sh:rule [
5.          a sh:SPARQLRule ;
6.          sh:order 110 ;
7.          sh:construct """
8.  PREFIX evac: <http://localhost/evac-planning-rules.ttl#>
9.  PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
10. PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
11. PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
12. PREFIX dc: <http://purl.org/dc/elements/1.1/>
13.
14. CONSTRUCT { ?hospital evac:isServedBy ?airport . }
15. WHERE {
16.     $this rdf:type evac:SuitableHospital ;
17.           geo:lat ?lat1 ;
18.           geo:long ?lng1 ;
19.           dc:identifier ?hospital .
20.
21.     ?airport rdf:type evac:SuitableAirport ;
22.              geo:lat ?lat2 ;
23.              geo:long ?lng2 .
24.
25.     BIND ( evac:geoDistance(
26.              xsd:double($lat1),
27.              xsd:double($lng1),
28.              xsd:double(?lat2),
29.              xsd:double($lng2) ) AS ?d ) .
30.     FILTER (?d <= "20.0"^^xsd:double) .
31. }"""
32.         ] .
```

Code Listing 3. A SHACL rule, expressed in Turtle syntax, to assert 'isServedBy' relations between suitable hospitals and airports that are no further than 20km apart. As hospitals and airports are initially unconnected, and SHACL cannot do pair-wise node targeting, a SPARQL CONSTRUCT query is used. geoDistance (line 25) is an imported JavaScript function.

is similar to the normal operations of Web APIs and RESTful services where the data service loses awareness of which fields are actually used by consumers and which are ignored.

To take advantage of standard tools for evaluating SPARQL queries, the complete data set must be expressed in RDF format, and moreover the format of the transmitted information will be identical to the internal RDF representation. This means any change to the internal representation of information necessarily also changes the representation as transmitted to consumers. This lack of encapsulation can make difficult to support evolutionary changes to the architecture, in that changes to one component force changes to downstream components.

One approach to this problem in Semantic Web engineering is to represent knowledge using a layering of ontologies, where the inner ontology is intended to be the most stable. But even using this approach it is not straightforward to, for example, provide simultaneously SPARQL access to two or more different, versioned representation of the same information, without duplicating data and having to handle data updates with extra care. An alternative approach is supporting ontology evolution, which is the imposition of a change management process that helps ensure certain properties of the ontology and instance data across modifications (Noy and Klein 2004).

The Shapes Constraint Language (SHACL) is a more recent addition to the stable of Semantic Web standards. SHACL is used to express constraints as shapes or sub-graphs, against which

RDF data graphs can be validated. While this can be read as overlapping with the concerns of Semantic Web ontologies, constraints can be expressed more tightly in SHACL than in Web Ontology Language (OWL) and more conveniently than SPARQL. In particular, you can express cardinals in SHACL, and optionally close a shape so that unexpected extensions of a resource can be detected. Thus, SHACL can be used by consumers of RDF information to perform quality-assurance or assumption checking, e.g. before introducing that data into their reasoning systems. It is also capable of conditionally asserting new statements into an RDF graph. This makes SHACL a potential tool for performing integration across RDF data sets. However, when using SHACL to conditionally assert new relations between two unconnected graphs, it is necessary to embed a SPARQL CONSTRUCT query in the SHACL rule, because you cannot do pair-wise rule targeting in SHACL (Code Listing 3). This is unintuitive and gains no advantage over using SPARQL directly.

SHACL is a relatively recent standard, and as yet there is limited software development tool support for it. Without such support it is easy to introduce typographical errors into SHACL rule expressions and receive no feedback as to whether even the rule is being evaluated as intended. SHACL shape graphs are expressed as RDF graphs, which can be unintuitive. See an example SHACL rule in Code Listing 2. In addition, at the time of writing there is not yet an implemented SHACL engine that has been optimised for speed of processing.

One of the advantages of adopting or interoperating with Semantic Web standards is the opportunity to easily add open source Linked Data (Bizer, Heath et al. 2011). Bringing such datasets into Saki in such a way as to retain their semantics means that the ontology alignment task is already done. Similarly, work continues in the ISR (Boury-Brisset, Kolodny et al. 2016, Michaelis, Tortonesi et al. 2016) and IoT (Jara, Olivieri et al. 2014, Svetashova, Schmid et al. 2017, Karim, Naameh et al. 2018) communities to apply Semantic Web techniques to help organise, discover and process streams of sensor data. Once again, it would be highly advantageous to support Semantic Web and Linked Data standards so that we can support sharing and integrating with such data streams across the enterprise.

JSON-LD Framing

JSON-LD Framing[8] is a relatively recent extension to JSON-LD that may better support an evolutionary architecture. JSON is a purely syntactic serialisation standard that is now very commonly used to represent information in Web services and RESTful web services. Much like XML, JSON can be used to transmit tree data structures, but unlike XML, JSON has only nascent standards for specifying schema, node selections and transformations. However, it is proving more popular than XML due to its lightweight syntax. JSON-LD is a standard that Web services can use to provide a parallel set of lightweight RDF semantic annotations on their JSON representations. This means that consumers of that information that can exploit Semantic Web semantic labels can absorb that information without ambiguities of interpretation. The purely syntactic JSON representation can still be accessed, so simpler clients can still consume the data.

Semantic Web RDF data represents graph data structures directly, while JSON usually presents data as a tree. Thus, in JSON-LD Web services it is the developer of the data service who decides how to render the graph of resources as a tree. JSON-LD Framing is a standard that allows web service consumers to specify the tree layout. Hence, JSON-LD Framing data services can maintain metrics about which resources and properties are being requested, and consumers can dictate the layout and which fields they require. The JSON-LD Framing standard does

[8] https://w3c.github.io/json-ld-framing/

```
type Hospital {
  id: ID!
  hasEmeregencyDepartment: Boolean!
  name: String!
  lat: Float!
  long: Float!
}

type Airport {
  id: ID!
  name: String
  lat: Float
  long: Float
  runwayLengths: [Float]!
}

type FlightToAirportAndHospital {
  hospital: Hospital!
  airport: Airport!
  distanceBetween: Float!
  flight: Float!
}

type Query {
  flightsToSuitableAirportsWithNearbyEmergencyDepartments(
    departureLat: Float!
    departureLong: Float!
    maxFlightDistance: Float!
    minRunwayLength: Float
  ): [FlightToAirportAndHospital]!
}
```

Code Listing 4. A GraphQL schema that supports a query to find airports, with a minimum runway length, in proximity of hospitals, with emergency departments, that are within a given distance of the given latitude and longitude.

however allow default property inclusions, similar to SPARQL *DESCRIBE* and Web API and RESTful service descriptions, which would void accuracy of the field-level metrics.

In contrast to SPARQL, JSON-LD and JSON-LD Framing do not mandate following a particular query or internal representation standard. This means that implementers are free to choose internal and data transfer information representations that can differ. This can be exploited to help make evolutionary changes to the system, e.g. through API versioning, where updated interfaces can be hosted on the one service side-by-side with legacy interfaces, removing the need to update every consuming service simultaneously. The cost of this approach is that all querying behaviour is specialised to each microservice and must be separately implemented.

As it leverages Semantic Web standards, integrating open source Linked Data datasets should be made much easier by removing ambiguities in interpretation of syntactic data structures.

```
query FindPotentialDestinations {
  flightsToSuitableAirportsWithNearbyEmergencyDepartments(
    departureLat: -17
    departureLong: 177
    maxFlightDistance: 3000
    minRunwayLength: 900
  ) {
    flight
    airport {
      name
      lat
      long
    }
    hospital {
      name
    }
    distanceBetween
  }
}
```

Code Listing 5. An instantiated query for the GraphQL interface defined above. Note that although both Hospital and Airport types have latitude and longitude fields, for this query they are only requested for airports.

GraphQL

GraphQL[9] is a standard for querying microservices. It was originated by Facebook, but now has independent communities of interest and support. GraphQL is a query-by-example, Remote Procedure Call technique. It defines a schema by which a data service can publish the query methods it has and the record types each query returns. These are published by the data service for human and machine consumption, which provides manual and automated software development support. Consumers send a standard request format to run one of the queries; a distinguishing feature of GraphQL is these requests are strictly query-by-example – they must include record types and all the fields that are required in the response. This query style has benefits for supporting evolutionary architectures. For example, when modifying a data service to add new fields, existing consumers will be unaffected by the change. It also means data services can easily automatically record accurate metrics on which records and which properties are in active use, which can be used to more safely introduce evolutionary improvements and deprecate components of the system. GraphQL can accept query arguments at any nested level in a request, which Web and RESTful services cannot do. This increased expressivity means that you can be more specific in a single GraphQL query than a single standard Web request, where you may have to perform multiple queries and do processing on the client side for the same effect. However, where Web API and RESTful service results can be easily cached using standard HTTP tools, which is a simple strategy to avoid some types of unnecessary network traffic, it is less obvious to implement in the GraphQL case.

The GraphQL standard does not address or specify internal data representation structures nor how information stores are to be queried. As with Web API, RESTful services, and JSON-LD services this means developers have the freedom and the burden of choosing different internal implementations and specialised query behaviours.

[9] https://graphql.org/

```
{
    "data": {
        "flightsToSuitableAirportsWithNearbyEmergencyDepartments": [
            {
                "flight": 2669.51,
                "airport": {
                    "name": "Hervey Bay Airport",
                    "lat": -25.31,
                    "long": 152.88
                },
                "hospital": {
                    "name": "Hervey Bay Hospital"
                },
                "distanceBetween": 6.40
            },
            {
                "flight": 2686.77,
                "airport": {
                    "name": "Sunshine Coast Airport",
                    "lat": -26.60,
                    "long": 153.09
                },
                "hospital": {
                    "name": "Nambour General Hospital"
                },
                "distanceBetween": 14.01
```

Code Listing 6. A fragment of the JSON response to the GraphQL query above.

Of the technologies discussed here GraphQL has arguably the best feature set for supporting information consumption in an evolvable microservice architecture. The most important of these is: the ability to separate internal from external information representations, which can better support gradual deprecation of outdated APIs rather than forcing system-wide changes; and as it enforces strict query-by-example each microservice can maintain accurate awareness of its immediate data dependents at the field level; it automatically produces API documentation; and it has very good software engineering support, in terms of implementation languages, documentation and community involvement.

However, unlike Semantic Web approaches, GraphQL does not address the challenge of conserving semantic interpretation of information from multiple microservices. Rather, all responsibility for semantic interpretation falls to the consumer. This approach leads to complex, fragile consumer code, and does not scale well.

As GraphQL uses JSON responses, it is compatible with JSON-LD annotations, which could be used to better share the burden of semantic integration between developers of data services and consumers. There are so far at least two notable approaches for enabling JSON-LD semantic annotations via GraphQL interfaces, namely GraphQL-LD (Taelman, Sande et al. 2018) and HyperGraphQL (Semantic Integration 2018). However, both rely on internal and external information representation being the same to automate translation between GraphQL and RDF. As discussed above, this leaves no room for encapsulation, which can hinder evolution. Thus, we think there is benefit in designing data services with GraphQL interfaces that support JSON-LD annotations on queries and responses, while also having potentially different internal representations and query implementations.

Knowledge graph interfaces

As described above, knowledge graphs have more expressive power than the data services and data integration services we are considering here, and not all API technology choices discussed above support that level of expressivity. The traditional approach from the Semantic Web community is to use SPARQL endpoints to query and assert information into knowledge graphs that are represented in RDF triple stores. If using a graph database, query languages such as Cypher or Gremlin can be used.

Allowing services to directly query the knowledge graph would result in two notable design constraints. Firstly, external services will need to be aware of the ontology that the knowledge graph uses, meaning that any changes to the ontology would have to also be reflected in all the external services that query it. Secondly, when asserting new information, all external services will need to be well behaved in order to label all new statements with appropriate provenance metadata, all following the same standards.

We will employ adapter microservices to add a flexible layer of control around the implementation details of the knowledge graph. A downside for using adapter microservices is that any knowledge graph API endpoint must be implemented in both the knowledge graph and then an adapter microservice, and this can slow down development. But this is outweighed by the benefits: looser coupling, which more easily can support making improvements to the knowledge graph ontology; and the option of enforcing business rules, such as mandatory provenance metadata. The interfaces of the adapter microservices can be more tailored, constrained and less expressive than the internal knowledge graph, and can thus be implemented in a wider range of technologies. Optionally, JSON-LD Framing or GraphQL can be used to perform arbitrary graph-to-tree mappings specified by consuming services at query time, which keeps the onus and control of which layout and fields are needed on the consumer.

Future work

We will continue to test these ideas in the Saki prototype architecture. The next steps are to develop and deploy data services that use more capable APIs as described here, and from there to build up semantic integration of data services by fielding a knowledge graph capability. In the first instance we plan to exploit the knowledge graph to augment search results.

Future situation awareness tools should be simpler to operate, maintain and adapt than the systems they replace. Thus, we are interested in exploring ways to automate the process of dynamically adding data services and other capabilities into Saki, and making that data available to search and geospatial UIs and semantic integration services, without overloading staff with information or configuration options. We plan to pursue re-use of existing semantically annotated data sets, and using smart information integration tools such as Karma (Knoblock and Szekely 2015, Szekely, Knoblock et al. 2015) and automatic schema alignment methods (Mathur, O'Sullivan et al. 2018) as ways to minimise the effort required for semantic integration of new datasets.

Summary

The information requirements of operation HQ staff are many, varied and changing, and sometimes unpredictable. We are investigating cloud-native, microservice architectures for information systems that can achieve the resilience, flexibility and composability that will be required in future HQ, without the need for a common information schema.

Microservice architectures bring with them increased numbers of services and dependencies and interactions between services. It is important that future software architectures and information management tools minimise maintenance burden for such systems, given that HQ staff time is a finite resource. Appropriate API design and technology choices can be made to minimise the effort required to maintain and improve these systems.

Of the current set of API technologies, GraphQL has the strongest design choices for supporting an evolvable microservice architecture. Different to traditional REST API approaches, with GraphQL services can maintain accurate, local metrics of the usage of their data, down to the level of fields. This gives system developers and maintainers good awareness for planning changes to or retiring data services. Different from other query interfaces, GraphQL supports: different internal information representations from data transfer information representations; and implementation-specific internal query mechanisms. This can be exploited to shield downstream services from requiring updates every time an upstream service is updated.

While GraphQL supports local data usage awareness by a pull mechanism, it is still important in HQ information systems to push and aggregate provenance metadata through each information processing step. This is so HQ staff can quickly judge the accuracy and trustworthiness of information integration products. Provenance metadata can also be used to support faster, partial information update processing and retraction of erroneous information. Additionally it can be exploited to make metrics that prioritise maintainers' efforts on the highest value information and services in the system, by tracing through those services that lie on the information processing path to the most frequently accessed information, as revealed by HQ staff usage metrics.

Information integration has the potential to present HQ staff with interfaces that draw from increasing numbers of sources, while also maintaining or improving information quality and avoiding clutter. Semantic Web technologies are aimed at solving integration problems, but impose design and development approaches that make them difficult to adopt in agile development teams. GraphQL is separate from Semantic Web standards, and does not provide an equivalent mechanism that imposes or encourages the development of precise, shared interpretation of information. There is an opportunity to extend GraphQL interfaces at least lightweight semantic annotations, such as JSON-LD, in order to better support information integration.

The 5th Generation Operational HQ concept shows that future information systems can be expected to take on more of the burden of sense-making and even prediction in HQ in order to lower cognitive load on staff. Knowledge graphs have been successfully applied to build towards these goals in comparable systems and contexts. Semantic cards that augment bag-of-words search results is an initial use case for knowledge graph in our system. Knowledge graphs have different API requirements than data services as they are conceived here, and hence should use query language interfaces, such as SPARQL. However, direct query interfaces expose internal knowledge representation details, so we favour the use of adapter microservices to decouple where possible changes to the knowledge graph from changes to those data and information services with which it interacts.

Acknowledgements

The authors wish to thank the Chief of Joint and Operations Analysis Division and Research Leader of Joint Warfare and Operations branch of DST Group and the Data to Decisions Cooperative Research Centre (D2D CRC) leadership for their support of this work.

References

Akgün, O., A. Dearle, G. Kirby and P. Christen (2018). Using Metric Space Indexing for Complete and Efficient Record Linkage. 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining.

Auer, S., S. Scerri, A. Versteden, E. Pauwels, A. Charalambidis, S. Konstantopoulos, J. Lehmann, H. Jabeen, I. Ermilov, G. Sejdiu, A. Ikonomopoulos, S. Andronopoulos, M. Vlachogiannis, C. Pappas, A. Davettas, I. A. Klampanos, E. Grigoropoulos, V. Karkaletsis, V. de Boer, R. Siebes, M. N. Mami, S. Albani, M. Lazzarini, P. Nunes, E. Angiuli, N. Pittaras, G. Giannakopoulos, G. Argyriou, G. Stamoulis, G. Papadakis, M. Koubarakis, P. Karampiperis, A.-C. N. Ngomo and M.-E. Vidal (2017). The BigDataEurope Platform – Supporting the Variety Dimension of Big Data, Cham, Springer International Publishing.

Bizer, C., T. Heath and T. Berners-Lee (2011). Linked data: The story so far. Semantic services, interoperability and web applications: emerging concepts, IGI Global**:** 205–227.

Boury-Brisset, A.-C., M. A. Kolodny and T. Pham (2016). ISR asset visibility and collection management optimization through knowledge models and automated reasoning. Knowledge Systems for Coalition Operations KSCO, collocated with the 21st International Command and Control Research and Technology Symposium: C2 in a Complex Connected Battlespace.

Data61/CSIRO. (2018). "Versioning – Consumer Data Standards." Consumer Data Standards, from https://consumerdatastandardsaustralia.github.io/standards/?shell#versioning.

Deitz, P. H., J. R. Michaelis, B. E. Bray and M. A. Kolodny (2016). The Missions & Means Framework (MMF) Ontology: Matching Military Assets to Mission Objectives. 21st International Command and Control Research and Technology Symposium: C2 in a Complex Connected Battlespace.

Dimou, A., T. De Nies, R. Verborgh, E. Mannens, P. Mechant and R. Van de Walle (2016). Automated metadata generation for Linked Data generation and publishing workflows. LDOW2016 - Proceedings of the 9th Workshop on Linked Data on the Web, CEUR-WS.org**:** 1–10.

Endsley, M. R. (1995). "Toward a Theory of Situation Awareness in Dynamic Systems." Human Factors **37**(1): 32-64.

Giménez-García, J. M., A. Zimmermann and P. Maret (2017). NdFluents: an ontology for annotated statements with inference preservation. European Semantic Web Conference, Springer**:** 638–654.

Heyvaert, P., A. Dimou, R. Verborgh and E. Mannens (2017). Ontology-based data access mapping generation using data, schema, query, and mapping knowledge. European Semantic Web Conference, Springer**:** 205–215.

Jara, A. J., A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner and A. F. Skarmeta (2014). "Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence." International Journal of Web and Grid Services **10**(2-3): 244–272.

Karim, F., O. A. Naameh, I. Lytra, C. Mader, M. Vidal and S. Auer (2018). Semantic Enrichment of IoT Stream Data On-demand. 2018 IEEE 12th International Conference on Semantic Computing (ICSC)**:** 33-40.

Knoblock, C. A. and P. Szekely (2015). A scalable architecture for extracting, aligning, linking, and visualizing multi-int data. Next-Generation Analyst III, International Society for Optics and Photonics. **9499:** 949907.

Lambert, D. A. (2001). "Situations for situation awareness." Proc. of Fusion 2001.

Lebo, T., S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik and J. Zhao (2013). "PROV-O: The PROV ontology." W3C recommendation.

Matheus, C. J., M. M. Kokar and K. Baclawski (2003). A core ontology for situation awareness. Proceedings of the Sixth International Conference on Information Fusion. **1:** 545–552.

Mathur, S. N., D. O'Sullivan and R. Brennan (2018). Milan: Automatic Generation of R2RML Mappings. European Semantic Web Conference.

Mayer, W., M. Stumptner, P. Casanovas and L. De Koker (2017). Towards a linked information architecture for integrated law enforcement. Proceedings of the workshop on linked democracy: artificial intelligence for democratic innovation (LINKDEM 2017). **1897**.
Meyerriecks, D., S. Davis, J. Pipher and P. Guthrie (2008). Independent Assessment Team Report on C2 Data, Institute for Defense Analyses, Alexandria VA.
Michaelis, J. R., M. Tortonesi, M. Baker and N. Suri (2016). Applying Semantics-aware Services for Military IoT Infrastructures. 21st International Command and Control Research and Technology Symposium: C2 in a Complex Connected Battlespace.
Moreau, L., P. Groth, J. Cheney, T. Lebo and S. Miles (2015). "The rationale of PROV." Web Semantics: Science, Services and Agents on the World Wide Web **35**: 235-257.
Nguyen, V., O. Bodenreider and A. Sheth (2014). Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property. Proceedings of the 23rd International Conference on World Wide Web. New York, NY, USA, ACM**:** 759–770.
Noy, N. F. and M. Klein (2004). "Ontology evolution: Not the same as schema evolution." Knowledge and information systems **6**(4): 428-440.
Paulheim, H. (2017). "Knowledge graph refinement: A survey of approaches and evaluation methods." Semantic web **8**(3): 489-508.
Pautasso, C., O. Zimmermann and F. Leymann (2008). Restful web services vs. big'web services: making the right architectural decision. Proceedings of the 17th international conference on World Wide Web, ACM**:** 805–814.
Schmitz, H.-C. and M. Gerz (2016). Applying OntoClean for the Evaluation of the MIP Information Model. Knowledge Systems for Coalition Operations KSCO, collocated with the 21st International Command and Control Research and Technology Symposium: C2 in a Complex Connected Battlespace.
Semantic Integration, L. (2018). "HyperGraphQL." from http://hypergraphql.org/.
Stumptner, M., W. Mayer, G. Grossmann, J. Liu, W. Li, P. Casanovas, L. De Koker, D. Mendelson, D. Watts and B. Bainbridge (2015). An architecture for establishing legal semantic workflows in the context of Integrated Law Enforcement. AI Approaches to the Complexity of Legal Systems, Springer**:** 124–139.
Svetashova, Y., S. Schmid and Y. Sure-Vetter (2017). New Facets of Semantic Interoperability: Adding JSON-JSON-LD Transformation Functionality to the BIG IoT API. International Semantic Web Conference (Posters, Demos & Industry Tracks).
Szekely, P., C. A. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. Yin, D. Kapoor, P. Natarajan, D. Marcu, K. Knight and others (2015). Building and Using a Knowledge Graph to Combat Human Trafficking. The Semantic Web - ISWC 2015, Springer**:** 205–221.
Taelman, R., M. V. Sande and R. Verborgh (2018). GraphQL-LD: Linked Data Querying with GraphQL. Proceedings of the 17th International Semantic Web Conference.
W3C. (2004). "RDF Primer." RDF Primer, from https://www.w3.org/TR/rdf-primer.
Winters, L. and A. Tolk (2009). C2 domain ontology within our lifetime, United States Joint Forces Command Norfolk VA.
Yue, Y., A. Kalloniatis and E. Kohn (2016). A concept for 5th generation operational level military headquarters. 21st International Command and Control Research and Technology Symposium: C2 in a Complex Connected Battlespace.