Using Semantic Web Technologies to Enable Interoperability of Disparate Information Systems

Dr. Marwan SABBOUH, Dr. Joseph K. DEROSA, Ms. Susan A. POWERS, Mr. Scott R. BENNETT

Abstract **- This paper addresses the problem of integrating various data sources and web services in an enterprise that uses a service-oriented architecture. We take advantage of ubiquitous enterprise concepts like** ***Types of Things*****,** ***Time*** **and** ***Position*** **(What, When and Where) and build a context ontology for each that relates all the various representations across the enterprise. Then, we use Information System data models, context ontologies and a small number of simple OWL/RDF mappings to enable information originating in one part of the enterprise to be used in another in a way that is highly (if not fully) automated.**

1.0 INTRODUCTION

Typically, a community of interest (COI) creates an information model with shared semantics and consistent representation of the associated data and web services. Large enterprises are moving towards a service-oriented architecture (SOA) strategy to make data and services from one part of the enterprise available for use in another part. However, SOAs do not address semantics of data. For example, regarding *Position* information, often the Datum is missing, or in the case of *Time*, the time zone designation is missing. Not only can this lead to execution errors, but also to lengthy testing and integration cycles to find and correct the errors [1].

Building on the semantic framework presented in [2], in this paper we show how to integrate disparate Information Systems (IS) in a SOA using ontologies (expressed in OWL/RDF [3-4]) as well as OWL/RDF mapping relations. Our approach is no longer dependent on building domain ontologies in the Gruber sense [5]. Instead we limit ourselves to ontological characterizations of the individual COI data and web service (WS) models. Thus the IS ontologies are data models expressed in OWL/RDF ontologies. Further, we take advantage of ubiquitous enterprise concepts like *Types of Things*, *Time* and *Position* (What, When and Where) and build a context ontology for each that relates all the various representations across the enterprise. Note that although these concepts are ubiquitous across the enterprise, each may have different representations within different COIs. By using OWL/RDF mappings to relate IS ontologies to context ontologies we are able to resolve representational differences. We use additional OWL/RDF mappings between IS ontologies to resolve structural and syntactic mismatches. Other OWL/RDF mappings associate web services with ontologies. We then automate the interoperability of disparate ISs by reasoning over this set of IS ontologies, the context ontologies and OWL/RDF mappings. The reasoning results in workflow discovery, automatic web service invocation, and reconciliation of mismatches.

The framework presented in [2] is implemented using a commercially available ontology management system (OMS) [6] along with a collection of developed tools that provide synergistic services as shown in Fig. 1.

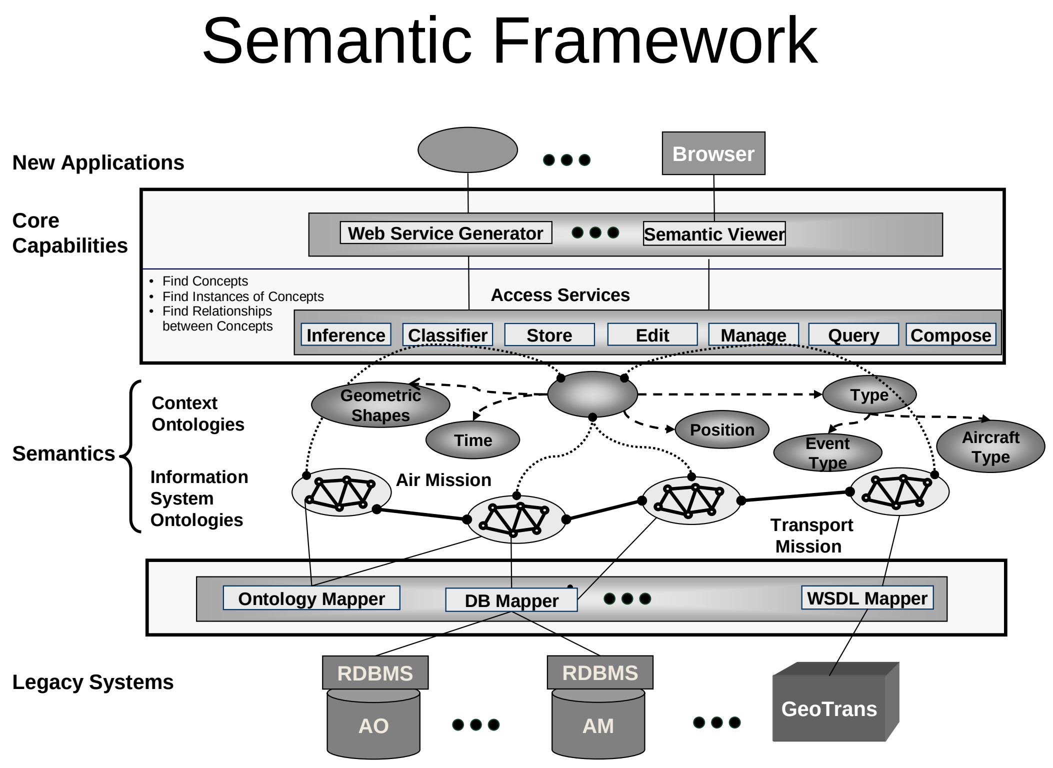


Fig. 1. Semantic Framework

In addition to standard services like *Store, Edit, Manage, Query,* the OMS offers a *DB Mapper*, *Web Service's WSDL Mapper*, *Ontology Mapper*, *Semantic Viewer*, *Inference Engine*, and a *Classifier*. The *DB Mapper* is used to map databases into the ontology. As a result, we are able to query the databases by simply querying the ontologies. The *Inference Engine* supports standard OWL DL inferences. The *Classifier* takes the XML document returned by a web service and translates it into RDF and classifies it into the ontology. The *Web Service's WSDL Mapper* reads in a WSDL and generates an ontological description of the WSDL. The *Ontology Mapper* is described in details in section 2.5. It is invoked by the Semantic Viewer. The *Semantic Viewer* implements code that, for a given request, discovers workflow, invokes and process web services, and invokes the classifier in order to enable the exchange of information between systems.

The outline for this paper is as follows: In section 2, we present the use case, build the IS ontologies, context ontologies, and their mappings. Then, we reason over the mapped ontologies to identify workflows whose execution achieves the flow of information. In section 3, we generalize the solution. In section 4, we discuss our findings and relate our work to the published literature.

2.0 ENGINEERING OF ONTOLOGIES

In our use case, we have two flight scheduling systems: Air Mobility (AM) for Military Airlift and Air Operations (AO) for Military Air Superiority. The systems were designed and developed independently. Each system includes its own flight scheduling service that creates flights and stores them in the system's own custom data store. A new requirement has emerged for flight information to be exchanged between these two systems. Specifically, flight information created and stored locally by the AM system must be provided to the AO system. Our goal is to use the ontologies and mapping techniques outlined above to discover and execute workflows that result in AM instances being retrieved from the AM database and transformed into AO instances. An overview of the approach is shown in Fig. 2.

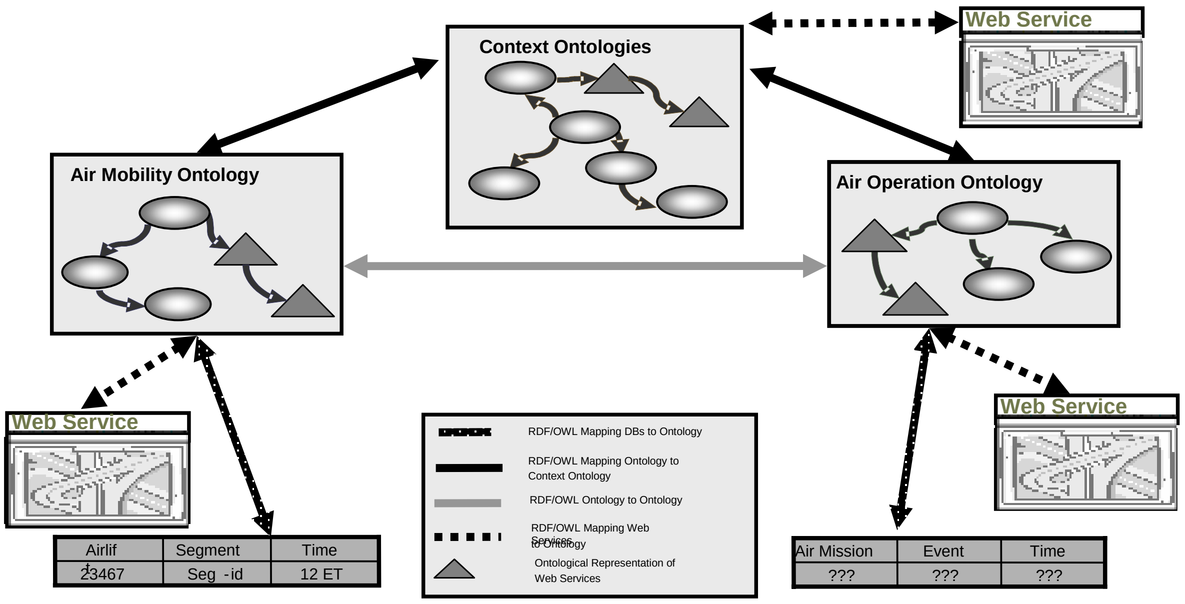

Context Ontologies
Web Service
Air Mobility Ontology
Air Operation Ontology
Web Service
Web Service
Airlif
Segment
Time
23467
Seg -id
12 ET
RDF/OWL Mapping DBs to Ontology
RDF/OWL Mapping Ontology to Context Ontology
RDF/OWL Ontology to Ontology
RDF/OWL Mapping Web Services to Ontology
Ontological Representation of Web Services
Air Mission
Event
Time
???
???
???

Fig. 2. Overall Approach

2.1 Build the IS Ontologies

We use OWL/RDF to build the AO and AM IS ontologies. These two ontologies encompass flight scheduling concepts for two disparate flight scheduling systems and were designed by different people working on our project. Relevant portions of the two ontologies are shown in Figs. 3 and 4. While both ontologies include concepts representing aircraft and events, significant distinctions exist in structure, representation and terminology. In particular, looking at Figs. 3 and 4, note that:

- AM represents position with Universal Transverse Mercator (UTM) coordinates while AO represents position with Geodetic coordinates

- AM uses the terminology ‘ARR’ and ‘DEP’ to denote arrival and departure events while AO denotes corresponding events as ‘LANDING’ and ‘TAKEOFF’
- AM and AO use different terminology to denote equivalent Aircraft Types
- AO provides both starting and stopping times for events while AM uses a single event time
- AM and AO use different overall structures to represent a flight

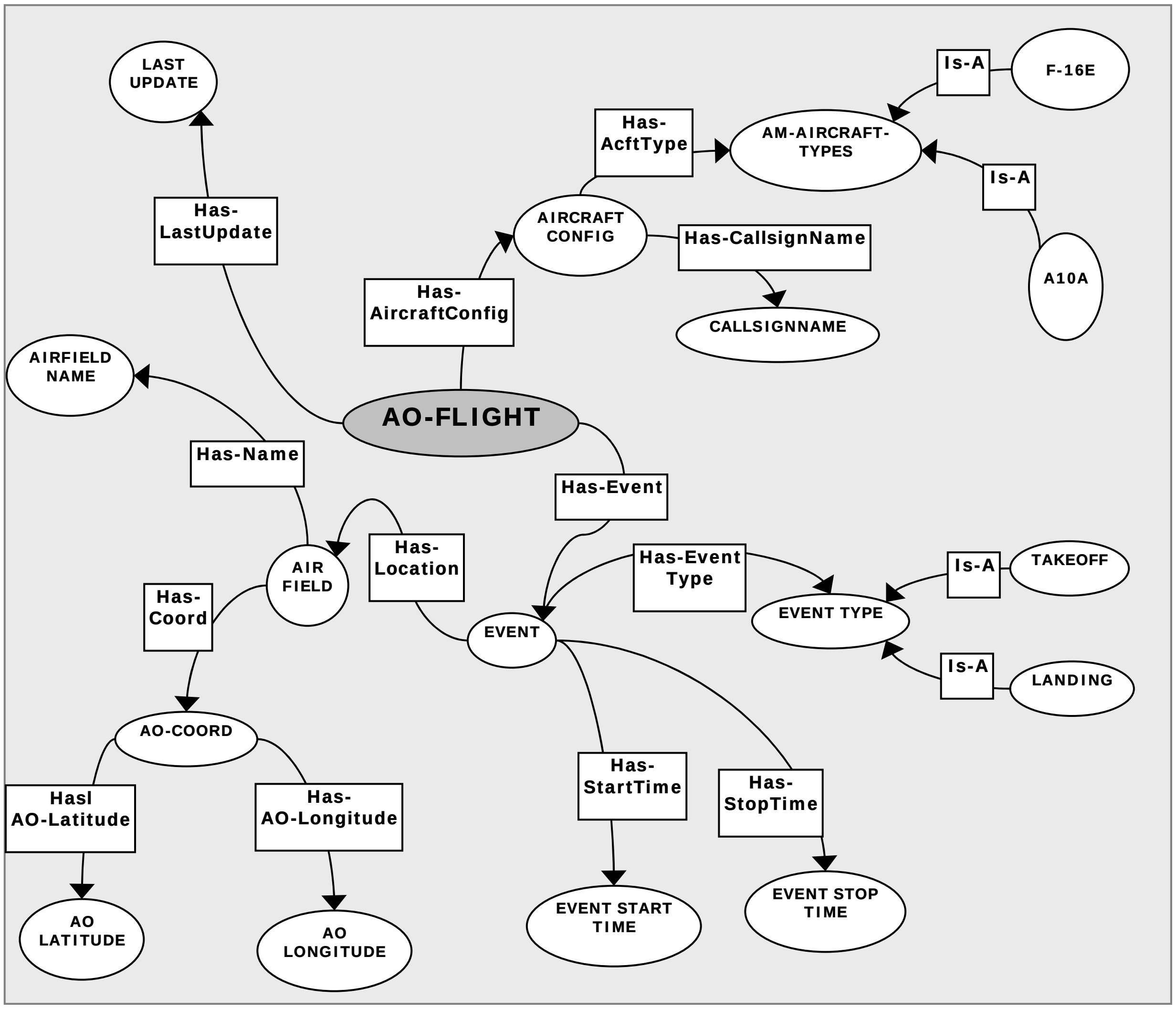


Fig. 3. AO System Ontology

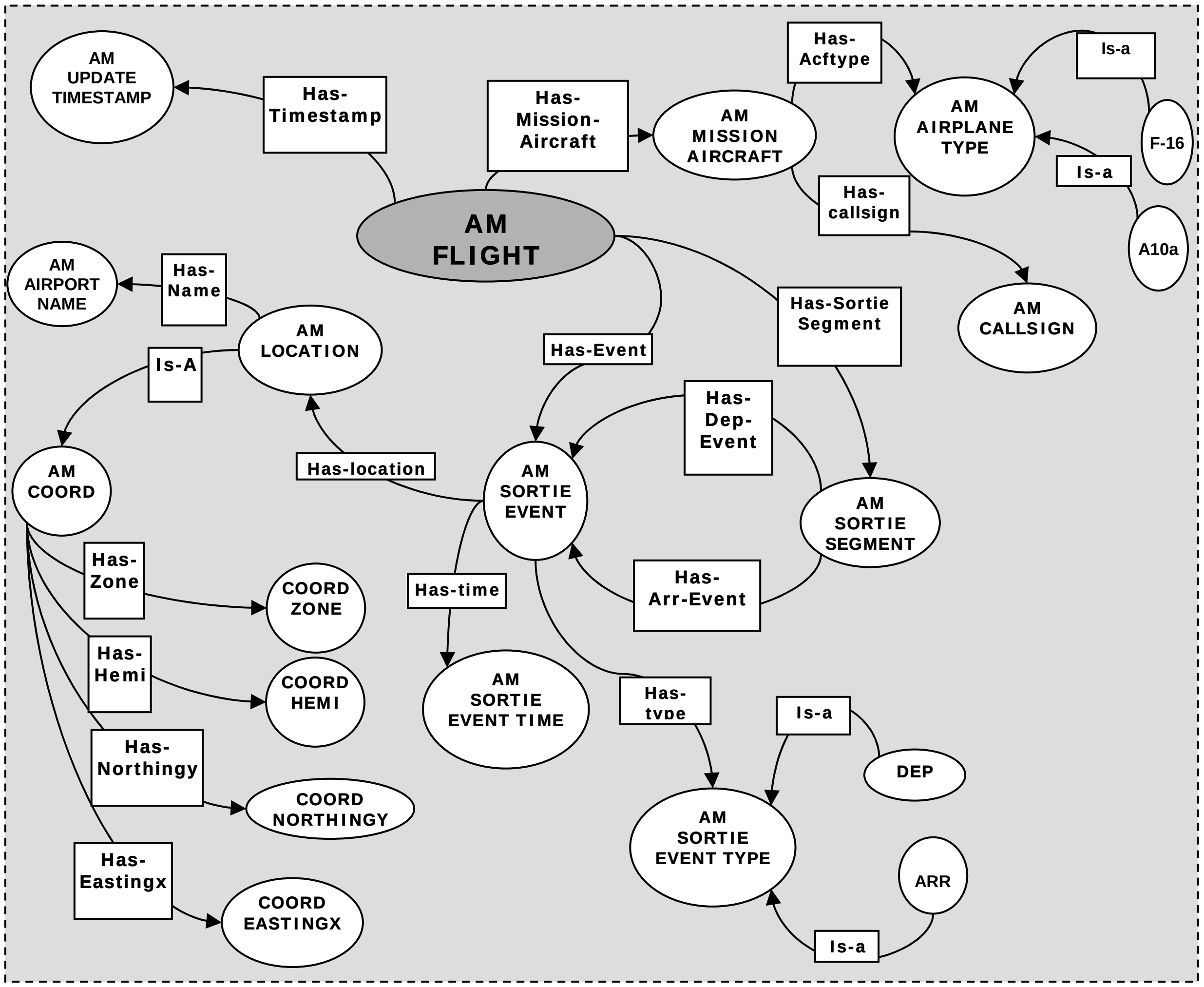


Fig. 4. AM System Ontology

2.2 Use Relevant Context Ontologies

To provide interoperability between information systems, we develop ontologies that capture common concepts across the enterprise, while taking into account each information system's representation for a particular concept. We refer to such ontologies as *context ontologies*. In this paper, we address three such context ontologies: *Position*, *Time*, and *Types of Things*, i.e., Where, When and What.

The use of context mediation in database integration is nothing new [7]. Our use of the term *context* refers to the various ways data can be represented and used in an enterprise, or even across enterprises [8]. At the heart of this paper is the assertion that there are a small number of context ontologies needed to achieve interoperability across the Department of Defense (DoD) enterprise [9], and that the various representations within each category are finite and can thus be captured in an ontology. So far, these assumptions have held true. The categories of *Position*, *Time*, and *Types of Things* have already proven very useful in the DoD enterprise as articulated in the Cursor on Target (CoT) initiative [10]. Upcoming work is under way to uncover remaining context ontologies in the DoD enterprise and we will leave the details of uncovering those context ontologies to a future paper.

2.2.1 Position Context Ontology

Our DoD Position context ontology contains comprehensive specifications of all the different representations of a Geo-Coordinate point in the DoD, i.e. the genus of Coordinate Systems, Datums, Coordinate Reference Frames and formats. We built the Position context ontology based on the set of coordinate systems used by National Geospatial Agency [11]. Fig. 5 shows part of the Position context ontology. The full listing of this ontology is given in [12].

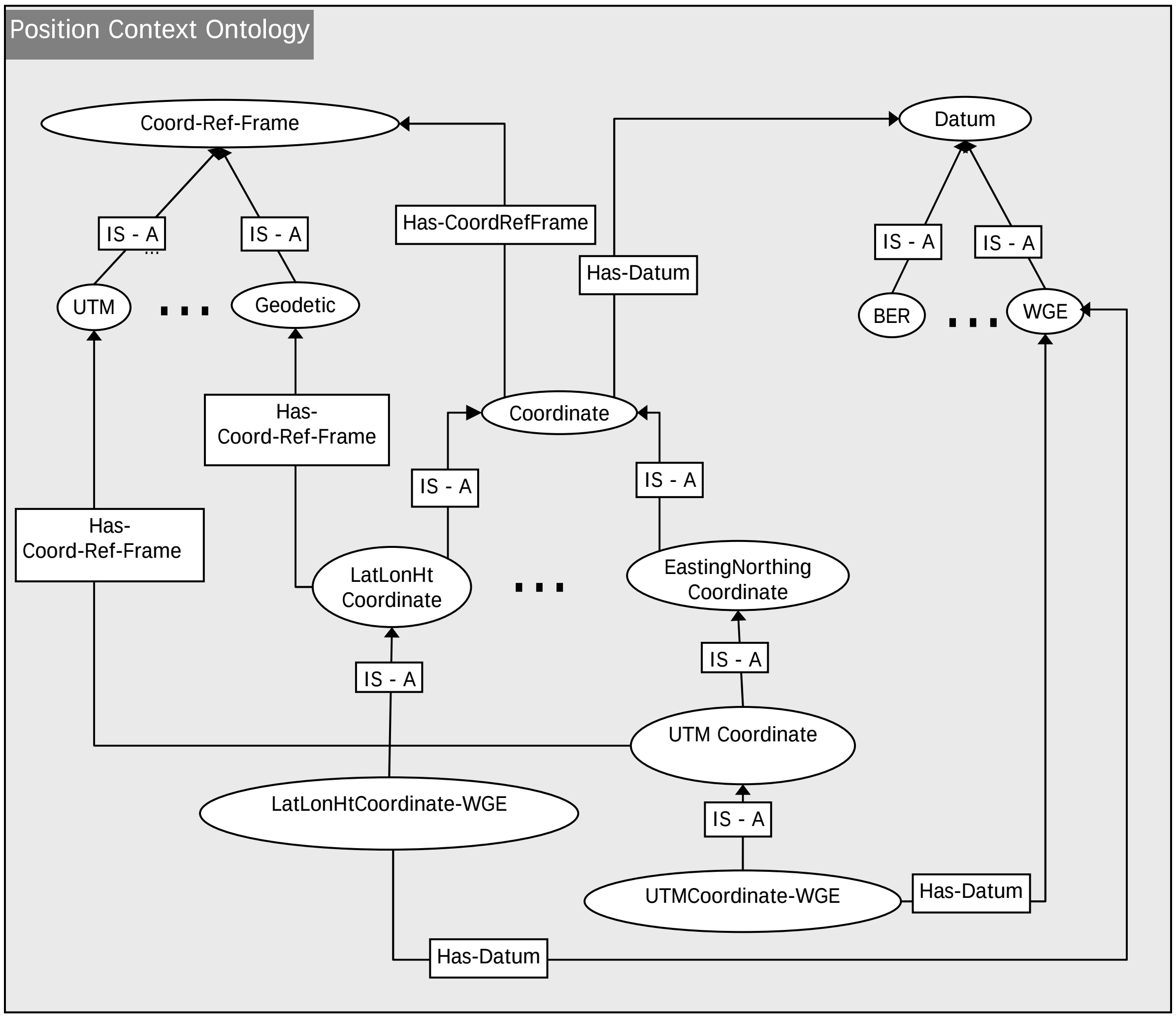


Fig. 5. Portion of Position Context Ontology

Using this context ontology, we can disambiguate any Geo-Coordinate position by specifying its Coordinate Systems, its Coordinate Reference Frame, and its Datum. These classes and the relationships between them are defined in the ontology by the following OWL classes and OWL object properties:

```
<owl:Class rdf:id=”COORDINATE”/>

<owl:Class rdf:id=”DATUM”/>

<owl:Class rdf:id=”COORD-REF-FRAME”/>
```

```
<owl:ObjectProperty rdf:ID=" Has-Datum">
    <rdfs:domain rdf:resource="#COORDINATE "/>
    <rdfs:range  rdf:resource="#DATUM "/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID=" Has-Coord-Ref-Frame">
    <rdfs:domain rdf:resource="#COORDINATE "/>
    <rdfs:range  rdf:resource="#COORD-REF-FRAME"/>
</owl:ObjectProperty>
```

To address the various Coordinate Reference Frames in use across the DoD, we assert for example:

```
<owl:Class rdf:id="GEODETIC">
   `<rdfs:subClassOf rdf:resource="#COORD-REF-FRAME"/>
</owl:Class>
<owl:Class rdf:id="UTM">
   <rdfs:subClassOf rdf:resource="#COORD-REF-FRAME"/>
</owl:Class>
```

To address the various Datums defined across the DoD, we add for example:

```
<owl:Class rdf:id="WGE">
   <rdfs:subClassOf rdf:resource="#DATUM"/>
</owl:Class>
```

A particular type of Coordinate, such as LATLONHTCOORDINATE or UTMCOORDINATE, is then specified as follows:

```
<owl:Class rdf:id="LATLONHTCOORDINATE">
```

```
    <rdfs:subClassOf rdf:resource="#COORDINATE"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Has-Coord-Ref-Frame ">
     <rdfs:domain rdf:resource="# LATLONHTCOORDINATE"/>
     <rdfs:range  rdf:resource="#GEODETIC"/>
</owl:ObjectProperty>
<owl:Class rdf:id=”UTMCOORDINATE”>
    <rdfs:subClassOf rdf:resource="#COORDINATE"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Has-Coord-Ref-Frame ">
     <rdfs:domain rdf:resource="#UTMCOORDINATE "/>
     <rdfs:range  rdf:resource="#UTM"/>
</owl:ObjectProperty>
```

To assert the DATUM for a Coordinate, we add these statements:

```
<owl:Class rdf:id=”LATLONHTCOORDINATE-WGE”>
<rdfs:subClassOf rdf:resource="# LATLONHTCOORDINATE"/>
</owl:Class>
<owl:ObjectProperty rdf:ID=" Has-Datum">
     <rdfs:domain rdf:resource="# LATLONHTCOORDINATE-WGE"/>
     <rdfs:range  rdf:resource="#WGE"/>
</owl:ObjectProperty>
<owl:Class rdf:id=”UTMCOORDINATE-WGE”>
    <rdfs:subClassOf rdf:resource="# UTMCOORDINATE"/>
    </owl:Class>
  <owl:ObjectProperty rdf:ID=" Has-Datum">
     <rdfs:domain rdf:resource="# UTMCOORDINATE-WGE"/>
     <rdfs:range  rdf:resource="#WGE"/>
  </owl:ObjectProperty>
```

For the remainder of this paper, we drop the OWL notation and substitute the equivalent graph or triple representation.

2.2.2 Time and Type Context Ontologies

We use similar specifications for Time and Types of Things. Several Time ontologies are available [13] and are generally based on the many representation the military uses to denote date/time (e.g., mm/dd/yyyy, Zulu, EST, etc.). Therefore the treatment of the Time context ontology is similar to the Position context ontology. For the Type context ontologies, we discerned two different types: Aircraft-Types, and Event-Types. These cover the different representations of aircrafts and events used by both systems. As a result, we developed two sub-types for Aircraft-Types, and Event-Types as shown below:

AM-AIRCRAFT-TYPES **subClassOf** *AIRCRAFT-TYPES*

AO-AIRCRAFT-TYPES **subClassOf** AIRCRAFT-TYPES

F-16 **instanceOf** (OWL Individual) *AM-AIRCRAFT-TYPES*

F-16E **instanceOf** (OWL Individual) *AO-AIRCRAFT-TYPES*

AM-EVENT-TYPES **subClassOf** *EVENT- TYPES*

AO-EVENT-TYPES **subClassOf** *EVENT- TYPES*

DEP **instanceOf** (OWL Individual) *AM-EVENT- TYPES*

TAKEOFF **instanceOf** (OWL Individual) *AO-EVENT- TYPES*

2.3 Integrate Web Services Using Ontologies

When translation between representations is needed, we associate a translator web service with the context ontology. Our approach for integrating a web service into a context ontology is discussed in the next section. Note that this same approach would be used to integrate a web service into an IS ontology.

2.3.1 Integrate GeoTrans Web Service into the Position Context Ontology

For our use case, we attach a GeoTrans web service based on the Geographic Translator [11] to the Position context ontology. The GeoTrans Web Service Description Language (WSDL) file is given at [14].

Integrating GeoTrans involves the creation of a web service upper ontology that is similar to an OWL-S Service Profile [15], and the recreation of its WSDL file in an ontology. This step is further articulated in [2]. We then use OWL/RDF mappings to attach both the GeoTrans upper ontology and the GeoTrans WSDL ontology to the Position context ontology as illustrated in Fig. 6.

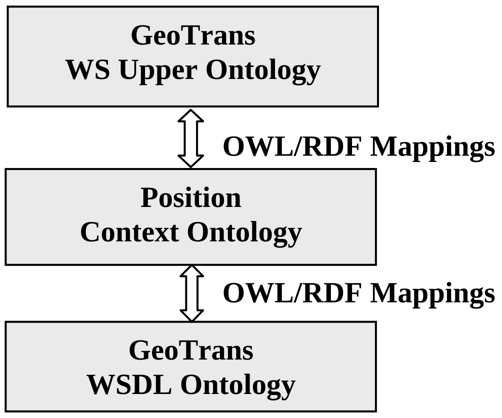


Fig. 6. Integrating WS and Context Ontologies

The GeoTrans upper and WSDL ontologies and the mappings that connect them to the Position context ontology are shown in Fig. 7. Note that only a few concepts of the Geotrans WSDL ontology are shown.

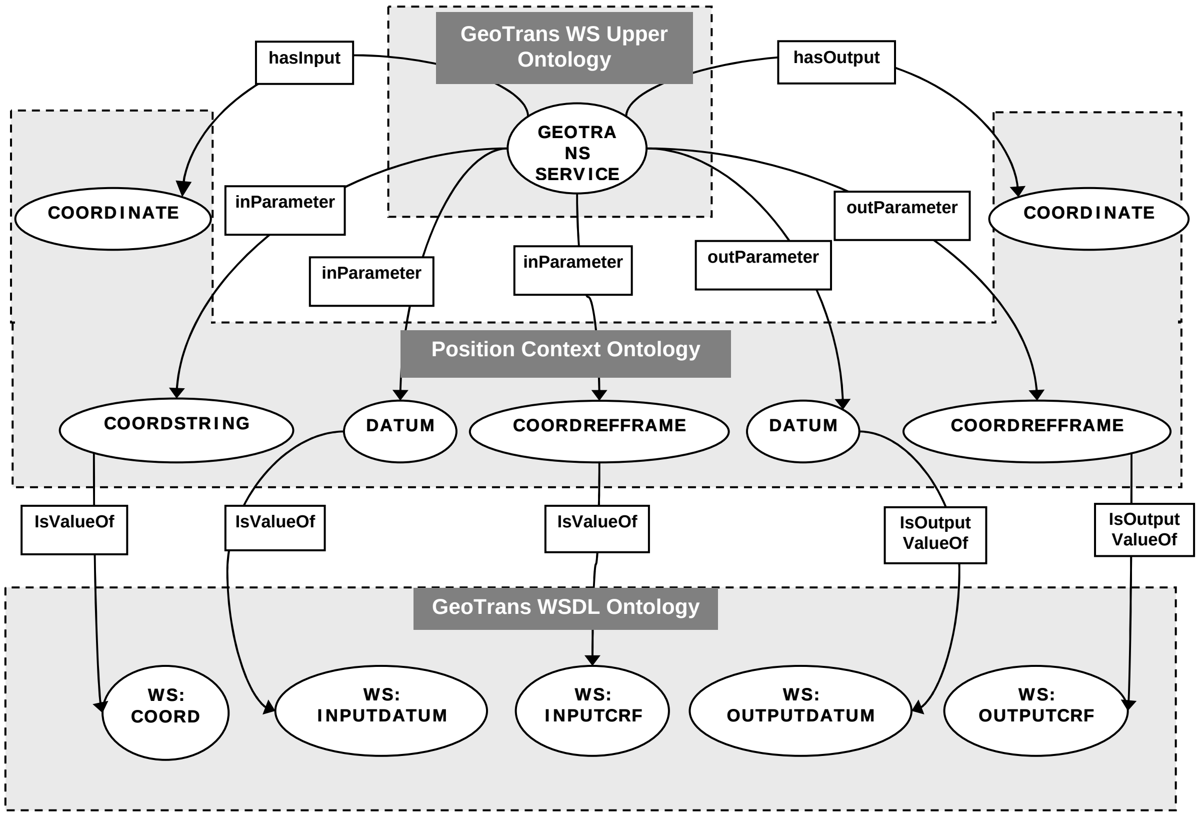


Fig. 7. Geo-Trans WSDL Ontology

The OWL/RDF mappings shown in Fig. 7 and their domains and ranges are summarized in Table 1. Note that while our use case employs these mappings to connect a web service to a context ontology, these same mappings are employed to connect a web service to an IS ontology.

Domain	OWL/RDF Mappings	Range	Purpose
rdfs:Class *in Context Ontology)*	isInputOf	Webservice: Class *(in WS upper ontology)*	Workflow discovery
Webservice: Class *(in WS upper ontology)*	hasInput (inverseOf isInputOf)	rdfs:Class *(in Context Ontology)*	Workflow discovery
rdfs:Class *(in Context Ontology)*	isOutputOf	Webservice: Class *(in WS upper ontology)*	Workflow discovery
Webservice: Class *(in WS upper ontology)*	hasOutput	rdfs:Class *(in Context Ontology)*	Workflow discovery
Webservice: Class *(in WS upper ontology)*	inParameter	rdfs:Class *(in Context Ontology)*	URL generation
Webservice: Class *(in WS upper ontology)*	outParameter	rdfs:Class *(in Context Ontology)*	URL generation
Webservice: Class *(in WS upper ontology)*	hasEffect	rdfs:Class *(in Context Ontology)*	Effect
Webservice: Class *(in WS upper ontology)*	hasClassification Condition	rdfs:Class *(in Context Ontology)*	Pre-condition
rdfs:Class *(in Context Ontology)*	isValueOf	rdfs:Class *(in WS WSDL Ontology)*	URL generation
rdfs:Class *(in Context Ontology)*	isOutputValueOf	rdfs:Class *(in WS WSDL Ontology)*	URL generation
rdfs:Class *(in Context Ontology)*	hasResult	rdfs:Class *(in WS WSDL Ontology)*	Import/ Classification of return web service result
rdfs:Class *(in Ontology)*	isCorrelatedWith	rdfs:Class *(in Ontology)*	Correlation between instances of data

Table 1. Mapping Web Service Ontologies to Context Ontology

We will show in Section 2.6 that an instance of a coordinate in any reference system can automatically be translated into an instance of a coordinate in any other reference system through the automated invocation of GeoTrans.

2.4 Mapping of IS and Context Ontologies

To enable the exchange of instances between the AO and the AM systems, we map the relevant AO concepts to the corresponding concepts in AM. This establishes the semantic matches – but not the representational matches - between corresponding concepts. Additional mappings of AO and AM concepts to the context ontologies are required to resolve representational mismatches. Note that concept matching requires agreement between the AO and AM users, whereas mapping to Context Ontologies is done independently for each system. We introduce the mappings in Table 2.

Domain	**OWL Object Property**	**Range**	**When to Use**
rdfs:Class *(in IS Ontology)*	hasContext	rdfs:Class *(in Context Ontology)*	Representational change
rdfs:Class *(in Context Ontology)*	isTheContextOf *(inverse of hasContext)*	rdfs:Class *(in IS Ontology)*	Representational change
rdfs:Class *(in IS Ontology)*	hasMatch *(symmetric)*	rdfs:Class *(in IS Ontology)*	Representational change
rdfs:Class *(in IS Ontology)*	hasMatchingValue (*symmetric*)	rdfs:Class *(in IS Ontology)*	No representational change

Table 2. OWL/RDF Mappings

In our use case, to reconcile a representational mismatch between coordinates, we assert the following:

- *AO-COORD* **hasContext** *LATLONHTCOORDINATE_WGE*
- *AM-COORD* **hasContext** *UTMCOORDINATE_WGE*
- *AO-COORD* **hasMatch** *AM-COORD*

To reconcile terminology mismatches between various event types or various aircraft types, we assert, for example:

- *AM-AIRCRAFT-TYPES* **hasMatch** *AO-AIRCRAFT-TYPES*
- *AM-EVENT-TYPES* **hasMatch** *AO-EVENT- TYPES*
- *F-16E* **OWL:sameAs** *F-16*
- *DEP* **OWL:sameAs** *TAKEOFF*
- *ARR* **OWL:sameAs** *LANDING*

When instance values can be copied without transformation, we assert, for example:

- *AIR-FIELD-NAME* **hasMatchingValue** *AM-AIRPORT-NAME*

Fig. 8 shows the AO ontology fully mapped to the AM and context ontologies.

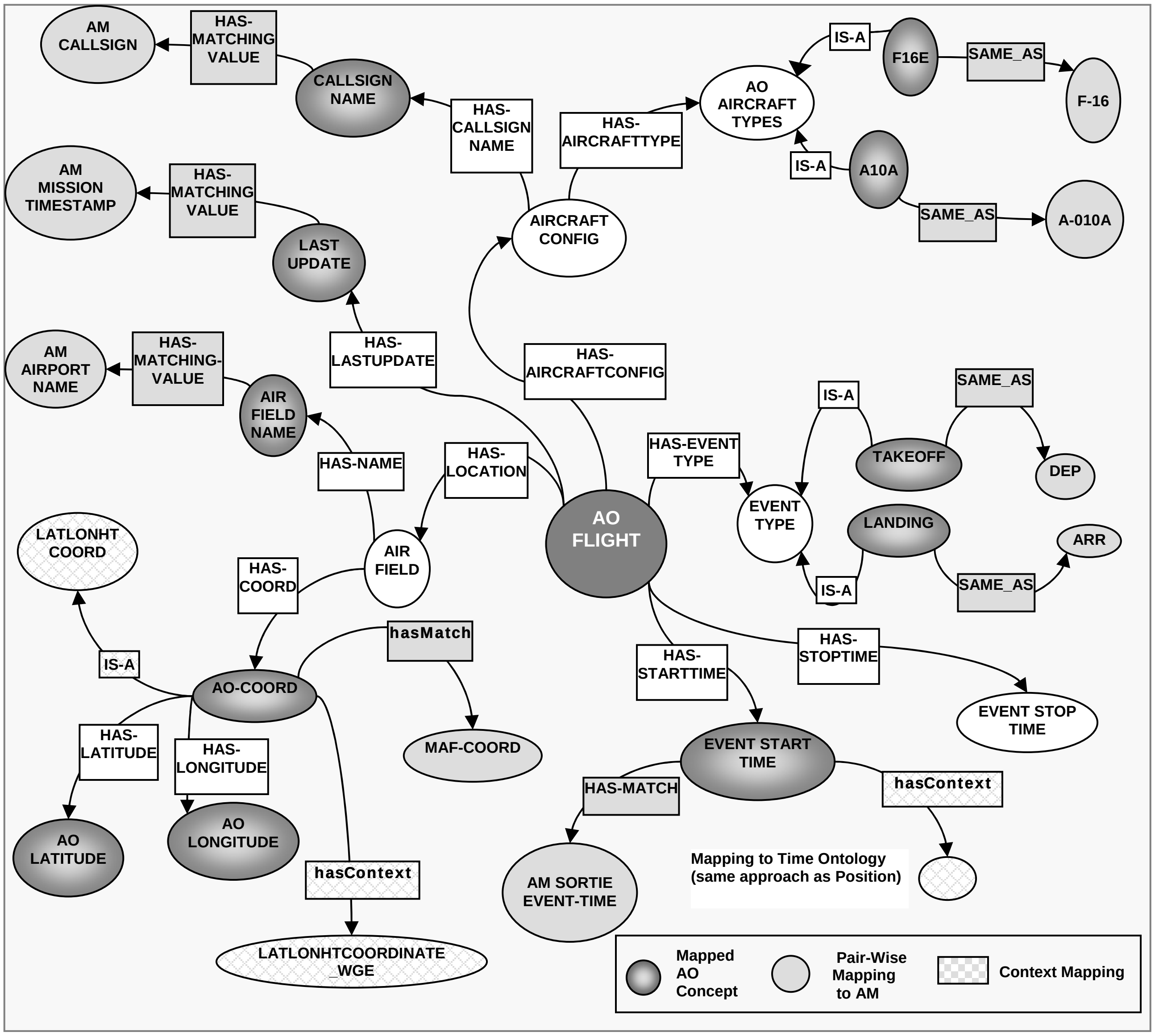


Fig. 8. Mapped Ontologies

2.5 Purpose of Onto-Mapper: Interpreting the OWL/RDF Links

In order for AM data to be translated to the AO system, the AM instances must be translated to match the structure, terminology, and representations of the AO system. To accomplish this end, we built a mapping interpreter, Onto-Mapper, that processes the OWL/RDF mappings presented in Table 2. This results in the creation of new AO instance data from the AM system. In the semantic framework, we represent Onto-Mapper as a specialized web service that gets invoked when data is being exchanged. We implement this Mapping Interpreter by building a specialized web service that acts as a Service Agent. Rather than have arbitrary inputs and outputs, this specialized WS only interprets the rdf/owl links to create instance data in the target system from data that originated in the source system. The formal definition of Onto-Mapper is shown below:

Service-Agent **subClassOf** Web-Service

Onto-Mapper **subClassOf** Service-Agent

hasAgentInput **subProperty** hasInput

hasAgentOutput **subProperty** hasOutput

isAgentInputOf **subProperty** isInputOf

isAgentOutputOf **subProperty** isOutputOf

isAgentInputOf **InverseOf** hasAgentInput

isAgentOutputOf **InverseOf** hasAgentOutput

To trigger the invocation of Onto-Mapper by the Semantic Viewer in our use case, we simply assert the following:

Onto-Mapper **hasAgentInput** AM-FLIGHT
Onto-Mapper **hasAgentOutput** AO-FLIGHT

In the next section we detail how Onto-Mapper then creates a new AO instance data.

2.6 Reasoning with the Mapped Ontologies / Creating Instances of Source Data in the Destination IS

Having accomplished the mappings of web services, as well as the mappings of IS ontologies to context ontologies, we map the AM database into the ontology, as described in [2]. In this fashion, we treat the AM data as instances of the AM ontology as indicated in Fig. 9. Then, to translate an AM instance into an AO instance, the Semantic Viewer reasons with the mapped ontologies to discover workflows, to invoke\execute\process web services, and to create the AO instance.

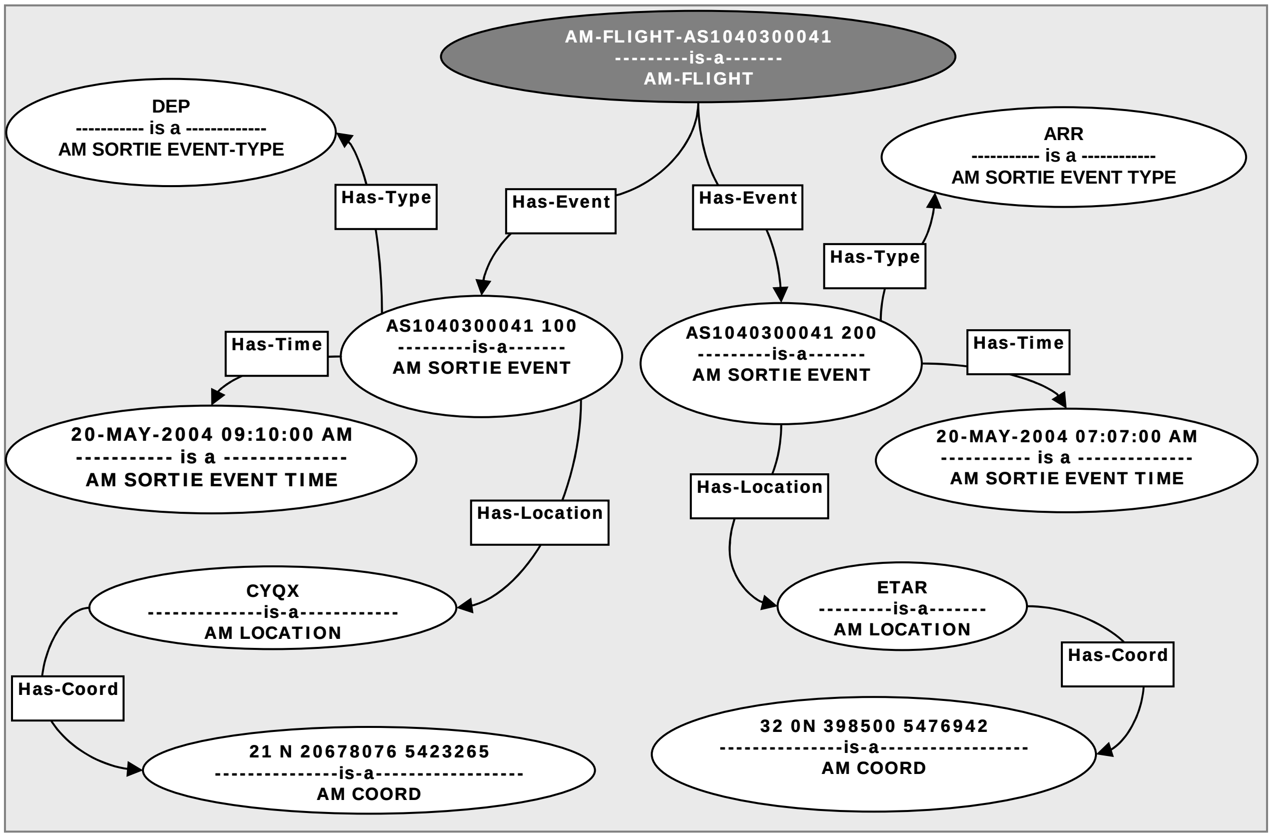


Fig. 9. AM Ontology incorporating AM Instance Data

2.6.1 Reasoning Algorithms

In addition to using RDF/OWL inferences, we make extensive use of two graph traversal algorithms: Direct Path Query (DPQ), and Incoming Intersection Query (IIQ). Both algorithms make use of RDF/OWL inferences and are described as follows:

Direct Path Query (DPQ)

Given a list of input values and a desired output, the DPQ creates the set of all the direct paths that lead to the desired output concept from the input concepts, i.e.

- For input list i_n, output v
- Find the direct paths P_k $\{p_1,p_2,..\}$ ending with v, and starting with each i in i_n, where a direct path is the sequence of nodes (i.e., concepts in the ontology graph) and relations or links that connects them. The system can be configured to exclude nodes connected by specific links or to only return paths containing certain links.

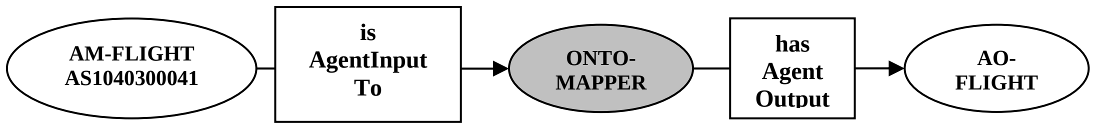
Incoming Intersection Query (IIQ)

First, the algorithm creates the set of all the direct paths that lead to the desired output concept using a DPQ. Second, for each input value, the algorithm creates the set of direct paths that lead to the given input. Third, the algorithm calculates and returns the intersection of these sets. This algorithm may be defined more formally as follows:

- For input list i_n, output v
- Find the list of nodes $x_i\{x_1, x_{2,..}\}$ that has direct paths P_k $\{p_1,p_2,..\}$ with v
- For each i in i_n, find the list of nodes $y_j\{y_1, y_{2,..}\}$ that has direct paths Q_m $\{q_1,q_2,..\}$ with i
- Return $\{x_i, P_k, Q_m\}$ where $x_i = y_j$

2.6.2 Discovering Initial Workflows

To discover the workflow for this use case, we specify the AM-FLIGHT instance AM-FLIGHT-AS1040300041, shown in Fig. 9, as input in the Semantic Viewer and AO-FLIGHT as output. The Semantic Viewer runs a DPQ and, if no workflow is found, an IIQ. We exclude the following relationships from the above search: inParameter, outParameter, hasMatchingValue, hasMatch, hasContext, isTheContextOf. The paths returned are interpreted as a workflow by the Semantic Viewer when they contain the relationship isInputOf or its sub-properties. In this use case, the following path is returned:



The presence of the RDF/OWL link isAgentInputOf indicates that Onto-Mapper must execute to create an instance of AO-Flight, which is the object of the hasAgentOutput RDF/OWL link.

2.6.3 Processing Onto-Mapper

Onto-Mapper searches for representational and terminology mismatches by interpreting the links hasContext, isTheContextOf, and hasMatch. The result of the search is a set of workflows (paths containing isInputOf or its sub-properties) that need to be processed for reconciliations of mismatches between the AM and AO domains. Each workflow consists of a sequence of web services. When all workflows and their component web services have been executed, a new AO instance is created from the AM instance AM-FLIGHT-AS1040300041. The algorithm shown below discovers the mismatches between the two domains and return workflows.

```
Paths discover-mismatches(input, output)
{
  X= Get Full Def of (output); C(i)= All-Triples (output);
  if (C(i) != NULL)
  {
     For each i in C(i)
```

```
        {
          o(i)= Object-value ( C(i) );  r(i)= Relation (C(i));
          if (r(i) == hasMatch and (o(i) ismemberof(input)) )
                    Answer = DPQ (o(i), input ) Exclude links:
                     hasMatch, sameAs, and hasMatchingValue
          else                  discover-mismatches ( o(i) );
          i = i+1;
          }
          Return Answer
        }
    }
```

An example of workflows returned by the algorithm is shown in Fig. 10. A quick scan of this workflow reveals that the GeoTrans web service must execute to derive the instance value of AO-COORD from *COORD: 21 N 20678076 5423265* (an instance of *AM-COORD*).

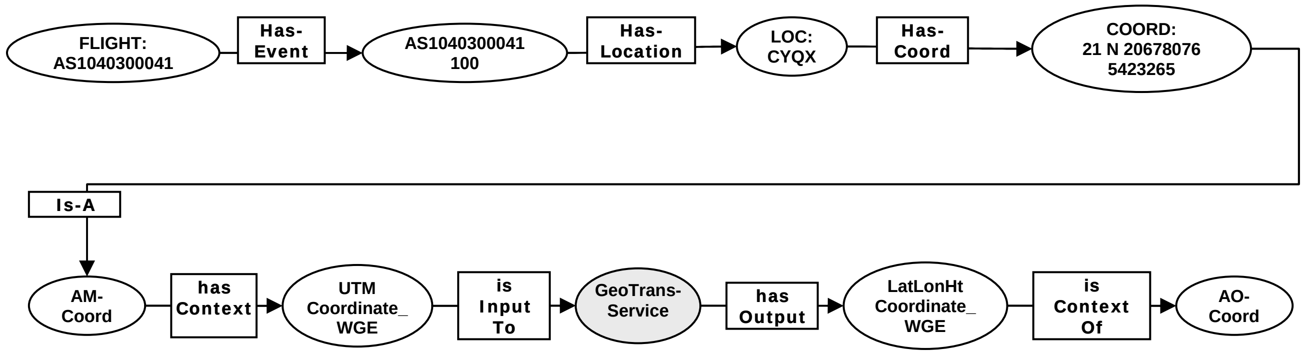


Fig. 10. Example Workflow

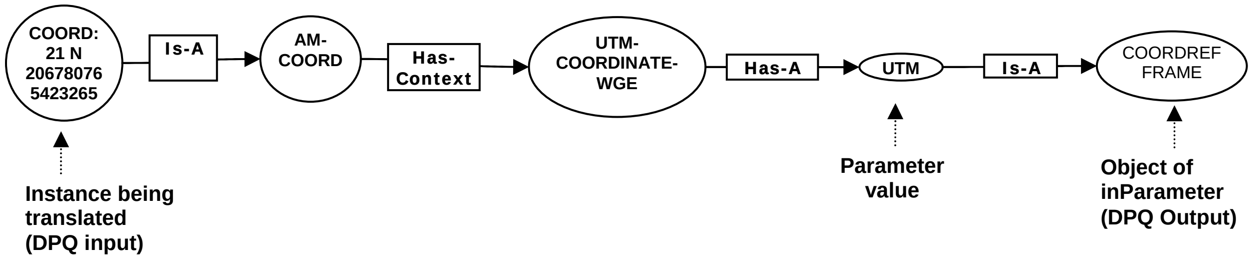
The Semantic Viewer finds additional workflows for other instances of AM-COORD that need translation. Similarly, workflows are discovered for Time representational mismatches.

2.6.4 Processing the Workflows Returned by Onto-Mapper

The Semantic Viewer processes the workflow node by node. In the example above, the first node is AM-FLIGHT-AS1040300041 followed by AS1040300041 100, LOC:CYQX, AM COORD 21 N 20678076 5423265, and UTMCoordinate-WGE. Since AM-COORD 21 N 20678076 5423265 inherits the hasContext link to UTMCoordinate-WGE from AM-COORD, the Semantic Viewer makes AM-COORD 21 N 20678076 5423265 an instance of UTMCOORDINATE-WGE which links it to GeoTrans using isInputOf. The latter signifies that GeoTrans must execute. Since LATLONHTCOORDINATE-WGE is linked to GeoTrans using hasOutput, this signifies that the result returned by the web service is an instance of LATLONHTCOORDINATE-WGE. The processing of isTheContextOf makes the instance of LATLONHTCOORDINATE-WGE an instance of AO-COORD.

2.6.5 Executing GeoTrans

The first step is to build the URL required to invoke the GeoTrans web service. The base URL and parameter names are read from the WSDL ontology. The parameter values are inferred from the mapped ontology. Specifically, when execution of GeoTrans is requested, its full definition (shown in Fig. 7 in section 2.3.1) is retrieved from the OMS, and the base URL, http://base.mitre.org/Geotrans/, is retrieved from the GeoTran's WSDL ontology. Then for each object of inParameter (e.g. COORDREFFRAME), the Semantic Viewer runs the DPQ with input "COORD: 21 N 20678076 5423265", and output being that object of inParameter (e.g. COORDREFFRAME). The returned path contains the parameter value to be used in the URL for that object of inParameter (e.g. COORDREFFRAME). This parameter value is identified in the returned path as the instance of the object of inParameter (e.g., COORDREFFRAME). For example, in the following path returned by the DPQ, the COORDREFFRAME's value is UTM.


COORD: 21 N 20678076 5423265
Is-A
AM-COORD
Has-Context
UTM-COORDINATE-WGE
Has-A
UTM
Is-A
COORDREF FRAME
Instance being translated (DPQ input)
Parameter value
Object of inParameter (DPQ Output)

Then to determine the parameter name we simply follow the link isValueOf which reveals the label of the parameter name (inputCRF). When the Semantic Viewer has processed all inParameter links, the URL will be of the form:

http://base.mitre.org/Geotrans/inputCRF=UTM&inputDatum=WGE&CoordString=21 N 20678076 5423265

Once all of the objects of each inParameter are processed, the Semantic Viewer turns it attention to outParameter. Similar to the processing for inParameter, the Semantic Viewer repeatedly runs the DPQ with input "COORD: 21 N 20678076 5423265 and output being each object of outParameter (e.g., COORDREFFRAME) . It then finds the matching parameter label using isValueOf. Thus the complete URL is of the form:

http://base.mitre.org/Geotrans/inputCRF=UTM&inputDatum=WGE&CoordString=21 N 20678076 5423265&
outputCRF=Geodetic&outputDatum=WGE

When the invocation of GeoTrans returns the XML document, the Semantic Viewer translates it into an RDF instance of AO-COORD, AO-COORD: 48.936668,-54.568333. The translation occurs as follows. Elements in the XML document are matched with concepts in the WSDL ontology. The latter are linked to the mapped ontology using **isValueOf** and **iisOutputValueOf**. This effects the creation of an *LATLONHTCOORDINATE-WGE* instance from the XML document. The Semantic Viewer completes the workflow processing by reclassifying the *LATLONHTCOORDINATE-WGE* instance as an *AO-COORD* due to the **isTheContextOf** link between *LATLONHTCOORDINATE-WGE* and *AO-COORD.* It also creates an **isCorrelatedWith** link is between *AM-COORD: 21 N 20678076 5423265* and *AO-COORD: 48.936668,-54.568333*.

2.6.6 Creating a New AO Instance

Once these workflows are executed, then the following algorithm accomplishes the creation of the new instance in AO domain, which is linked to the AM instance AM-FLIGHT-AS1040300041 using the link isCorrelatedWith and imported to the OMS.

```
Input=Instance in AM domain; Output=Class in AO domain
Child-Concepts= method-process-instance ("COORD: 21 N 20678076 5423265", Output, 1);


method-process-instance (Input-Instance, Output, FLAG)
{
Child-Parent-Concept= NULL;
 X= Get Full Def of (Output);     C(i)= All-Triples (X);


   if (C(i) != NULL)
   {
      For each i in C(i)
      {
         o(i)= Object-value ( C(i) );
         r(i)= Relation (C(i));
         if (r(i) == hasMatchingValue)
         {
                 Value-instance(n)= DPQ( Input-instance, o(i) )
                   Return Value-instance( );
         }
```

```
else if (r(i) == hasMatch || sameAs )
 {
        Value-instance(n)= DPQ(Input-instance, o(i) ) Exclude links:
         hasMatch, hasInput , hasOutput, hasContext, and isTheContextOf
        Child(n)= IIQ(value-instance(n), Output)
        Return Child ();
 }
  else if {r(i) == hasContext || isTheContextOf || isAgentOutputOf || isOutputOf)          { do
  nothing;}
  else {
        Child()=method-process-instance ( o(i), 0 );
        If ( FLAG== 0 )
        {
              For each j in Child(j){
                    if( Child(j) != Null){
                          if (Child-Parent-concept(j)==NULL){
                                Child-Parent-concept(j)= New KN
                                      Assert Child-Parent-concept(j) subClassOf
                                      Concept
                          }
                          Assert Child (j)   subClassOf     o(i)
                          Assert Child-Parent-concept(j) r(i)   Child(j);


                    }
              }
              Else
              {
                    For each j in Child(j){
```

```
                                    if( Child(j) != Null){
                                            if (Child-Parent-concept(0)==NULL){
                                            Child-Parent-concept(0)= New KN
                                            Assert Child-Parent-concept(0) subClassOf   Concept
                                            }
                                    Assert Child (j)   subClassOf   o(i)
                                    Assert Child-Parent-concept(0) r(i)   Child(j);
                                    }
                            }

                        }
                }
        i = i+1;
        }// end for each i
  }// end c(i)!= Null
Return Child-Parent-concept ();
}// end method
```

3.0 Generalization

Our solution is easily generalized and extended beyond the use case presented above. In the more general case, the algorithms presented above return multiple initial workflows (see section 2.6.2), as shown in the example in Fig. 11.

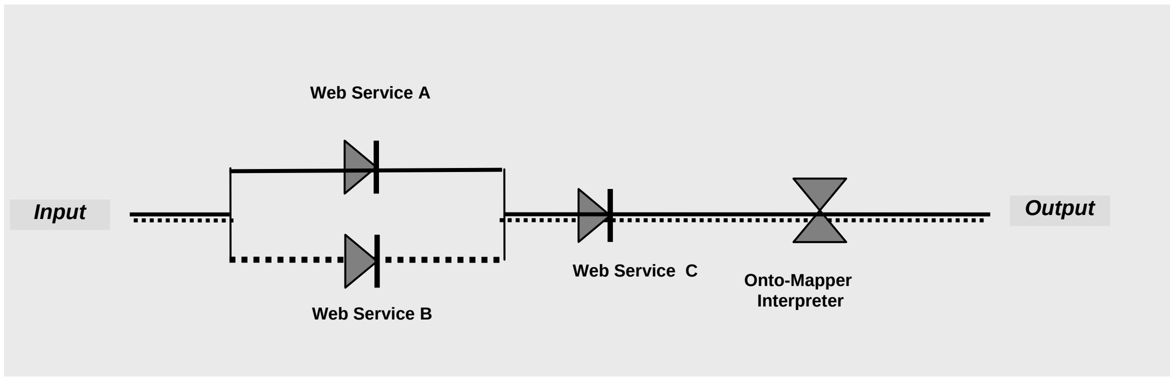


Fig. 11. Multiple Workflows

The system attempts to execute each workflow to the extent that it has the necessary data input. Each workflow stores whatever output it produces in the OMS, making it available to subsequent workflows. In the example above, the first workflow is composed of the sequenced web service A → web service C → Onto-Mapper. The execution of A succeeds and its result is stored in the OMS. The execution of C, however, fails since the output of web service B is needed as input to C. The system proceeds to execute the second workflow: web service B → web service C → Onto-Mapper. This workflow successfully completes since the necessary output from A is now available in the OMS.

Beyond this generalization to multiple workflows, the solution is also extensible. We can incorporate new context ontologies and extend existing context ontologies at will without having to change existing mappings.

4.0 DISCUSSION

We see several significant advantages to our context ontology-based approach to semantic integration. Foremost, this is a general solution that can be applied to interoperate any two ISs using the set of context ontologies and same RDF/OWL mappings. The workflow required for reconciliation of mismatches is derived by reasoning with the mapped ontologies and is not preprogrammed into the system.

Moreover, the effort to implement the mappings is minimized by the polymorphic nature of this approach. Mappings are defined at the concept level not the instance level, but they enable semantic based transformation of instance data. For example, once an objects of inParameter (e.g., Datum), is linked to GeoTrans, then instances of the object (e.g., WGE) are passed as input arguments. This is because the DPQ and IIQ make use of OWL inferences: i.e., **subclassOf**, **inverseOf**.

We have also found that with our methodology, IS and context ontologies are sufficient. This eliminates the need to develop domain ontologies that are typically expensive to build and maintain. In this fashion, we can also leverage existing XML schemas and web services as the IS ontologies are nothing more that the IS data models expressed in OWL.

Another key characteristic of this approach is the loose coupling between the IS ontologies and context ontologies. Mappings to context ontologies can be added to an existing IS ontology without any change to the IS data model and the same context ontologies can be reused to annotate multiple IS ontologies. More importantly, context ontologies can be used to augment data models with missing information. Note that in our case, Datum was missing from the AM/AO ontologies. However, due to the mapping to the Position context ontology, the system is able to pass an appropriate value for Datum to the GeoTrans webservice.

Further benefit derives from our method of using an ontological approach to connect translation web services to context ontologies. Once an IS ontology is mapped to a context ontology, no further relations are required between the IS ontology and the web service for translations to take place. For example, when translation is needed between coordinates, Geotrans is invoked without linking it explicitly to any IS concept.

Finally, it is evident that hasMatchingValue can be replaced with owl:equivalentClass, and in place of hasContext, we can have a rdfs:subClassOf, or even owl:equivalentClass. However, hasMatch is needed and cannot be replaced with an OWL language construct.

4.1 Relation to Literature

This approach leverages existing initiatives and builds on them. For example, the set of RDF/owl mappings augment the ontology of mapping relations work done at Stanford [16], with some newer mappings. Further, we repurpose their use. We also have use of the Web Service Profile ontology in common with OWL-S [15], although we take the view that web services are extensions of data, with the net result of broader applicability of this solution. For example, this approach can be viewed as a candidate solution for the harmonization of multiple Community of Interest (COI) problem [17]. We also share common aspects with Web Service Modeling Ontology (WSMO) [18] approach from the perspective of the use of mediators. For example, ONTO-MAPPER mediates between the various representations. However, in our solution, workflows are derived and not programmed. We think this is important, as enterprises are concerned with emergent behavior.

4.2 Future Work

We are in the process of building context ontologies for the DoD categories Geometric Shapes, and Unit of Measures. Additionally, we are investigating approaches to automating the creation or generation of some of links between IS ontologies. To that end, we are building a shared ontology across Position and Time, and investigating how rules [19] might help. We are also adding the handling of conditions when traversing graphs and the implementation of causality [20] in the system. The former enhances our capabilities to specify pre-conditions and constraints. Causality allows for reasoning with state changes in the system over time.

REFERENCES

[1] CW3 Xavier Herrera, USMC, "The bottom line for accurate massed fires: common grid", Field Artillery Journal, Jan-Feb, 2003

[2] Sabbouh, M and DeRosa, J.K., "Using semantic web technologies to integrate software components", Proceedings of the Third International Semantic Web Conference (ISWC 2004), November, 2004

[3] Web Ontology Language (OWL), World Wide Web Consortium, http://www.w3.org/2004/OWL/

[4] Resource Description Framework (RDF), World Wide Web Consortium, http://www.w3.org/rdf/

[5] T.R. Gruber. "A translation approach to portable ontologies", J on Knowledge Acquisition, Vol 5(2), p199-220, (1993)

[6] Language and Computing. http://www.landc.be

[7] Goh, C.H., Bressan, S., Madnick, S., Siegel, M., "Context interchange: new features and formalisms for the intelligent integration of information", ACM Trans. on Information Systems, 13(3), pp. 270-293, 1999

[8] Gannon, T., Madnick , S., Moulton , A., Siegel , M., Sabbouh , M., Zhu, H. "Semantic information integration in the large: adaptability, extensibility, and scalability of the context mediation approach", MITRE Corporation , Massachusetts Institute of Technology (MIT) - Sloan School of Management, MIT Sloan Working Paper No. 4541-05; CISL Working Paper No. 2005-04 , May 2005

[9] Byrne, R.J., "Cursor on Target: A case study deploying what, where and when in the battlefield", MITRE Technical Report MP04B0000056, MITRE Corp., December 2004

[10] Byrne, R.J., "Getting DOD linked – how to build netcentric operations", MITRE Technical Report MP04B0000058, MITRE Corp., December 2004

[11] National Imagery and Mapping Agency (NIMA) USA, GEOTRANS 2.2.4-Geographic Translator, http://earth-info.nima.mil/GandG/geotrans/

[12] Kazura A., Sabbouh M., Position ontology, The MITRE Corporation, pending public release

[13] Hobbs, J., "A DAML ontology of time", 2002, http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt

[14] Schroeder, B. and Sabbouh, M., Geotrans WSDL, 2005, The MITRE Corporation, pending public release

[15] OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/, November, 2004

[16] Crubrzy,M.,Pincus,Z.,Musen.M.A., "Mediating knowledge between application components", Semantic Integration Workshop of the Second International Semantic Web Conference (ISWC-03), Sanibel Island, Florida, CEUR, 82. 2003.

[17] Renner, S., "A Community of Interest approach to data interoperability", Federal Database Colloquium '01, San Diego, August 2001.

[18] Web Service Modeling Ontology (WSMO), http://www.w3.org/Submission/WSMO/, June 2005

[19] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", W3C Member Submission 21 May 2004, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[20] Lehmann, J., Borgo, S., Masolo, C., Gangemi, A., "Causality and causation in DOLCE", to appear in A.C. Varzi, L. Vieu (eds.), Formal Ontology in Information Systems, Proceedings of the International Conference FOIS 2004, Torino, November 4-6, 2004, IOS Press Amsterdam, 2004, pp. 273-284