A Semantic Infosphere

Michael Uschold[1], Peter Clark[1], Fred Dickey[1], Casey Fung[1], Sonia Smith[1], Stephen Uczekaj[1], Michael Wilke[1], Sean Bechhofer[2], and Ian Horrocks[2]

[1] Boeing, Phantom Works , P.O. Box 3707,m/s 7L-40
Seattle, WA USA 98124-2207,
michael.f.uschold@boeing.com

[2] Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK,
seanb@cs.man.ac.uk

Abstract. We describe a prototype implementation of a semantic filtering capability added to an existing XML-based publish and subscribe infrastructure. An ontology is used to provide vocabulary for expressing both 1) the semantic annotations that characterize the published documents and 2) the subscriptions specifying the class of documents to be routed to a given client. A description logic (DL) classifier is used to determine which subscribers an incoming document is routed to. We outline the key elements of the ontology for the battlefield domain and give some sample annotations and subscriptions. This is the basis for describing a number of scenarios showing how this filtering capability could be used practice. We critically analyze the suitability of a DL language and reasoner in general, and the particular implementation choices (DAML+OIL, FaCT and OilEd) for performing this task. A key result of the work is to demonstrate the importance of testing semantics-based technologies on practical problems. We discovered a number of new and interesting areas for future work, which in turn can direct the focus of the research community.

1 Introduction

An infosphere is a platform of protocols, processes and common core services that permit stand-alone or web-based applications to submit, discover and share information over a network. An XML-based infosphere architecture called the Xinfosphere was developed at Boeing Phantom Works Seattle in 1999-2001. It was made available for download on the Boeing intranet.

The Xinfosphere supports two types of users and two types of core services. First, there are client users & applications that utilize data dissemination services such as publish, subscribe and query. Second, there are client administrators that utilize administration and security services such as configuration, monitoring and access control. Published items that can be subscribed to are called Information Data Objects (IDOs). They are in XML format. Each may have corresponding metadata, also in XML. Subscriber clients specify IDOs of interest by creating a filter. When an IDO is published, all the filters are checked to see which subscriber clients the IDO gets routed to. A given subscriber client only receives the IDOs that pass through their specified [subscription] filters.

The original mechanism for implementing the publish/subscribe component of the Xinfosphere is purely XML-based. All IDOs and metadata are in XML, filters are expressed in XPath, XSL or XQL [2,3,6]. As such, the approach is purely syntactic. The goal of this research is to explore the hypothesis that there are advantages to augmenting the Xinfosphere publish/subscribe component with a semantic filtering capability. Some potential advantages are as follows:

1. the subscriber will no longer be required to know the structure and syntax of the documents in order to create a filter, they focus instead on the semantics of the content;
2. [related to 1] a rich and powerful semantics-based mechanism can be used to express filters allowing greater selectivity and finer control;
3. the collection of filters from all the subscriber clients can be automatically classified, *a priori*, potentially resulting in faster routing;
4. a set of filters can be checked for internal consistency. This could be done for a single client, or for a group of clients;
5. the types of IDOs that can be subscribed to no longer need to be just a flat list; instead they may be arranged in a rich hierarchy that arises from an ontology in the domain of the IDOs.

We recognize that semantic approaches are not a panacea. In some circumstances, a syntactic approach may be adequate. The system has been architected to allow additional filtering mechanisms to be added. We have added one, leaving the original one in place. Other semantic filtering technologies can be added, for example, different description logic reasoners [1], or other ontology languages and inference engines. This gives clients the ability to choose the filtering technology most suitable to their needs.

Semantic Filtering: Subscriptions and Queries The Xinfosphere supports both publish/subscribe and answering queries from a static repository. Subscriptions and queries are closely related concepts, for example one can think of a subscription as a query to be applied in the future. In both cases, there is a description that an Information Data Object must match before it is passed along to the requester. The descriptions are the filters. We mainly focus on the publish/subscribe aspect of the Xinfosphere in this paper. We will usually not distinguish between queries and subscriptions, speaking only of semantic filters that could be used for either.

1.1 Approach

In this section we describe our approach for implementing a Semantic Infosphere. The existing Xinfosphere is architected to enable alternative filtering technologies. We added one that is semantics-based, rather than purely XML-based. We identify the main elements and comment briefly on our choice of which semantics technologies to use. Our approach includes the following key elements:

Ontology. Create an ontology that represents the key concepts and relationships in the domain of interest. Represent it in a formal language that has a corresponding inference engine.

Semantic Annotations. Augment documents with semantic annotations (i.e. metadata) using terms from the ontology, encoded in the ontology language. Documents are published along with their semantic annotations.

Semantic Filtering. This involves two steps. First, create semantic filters specifying which IDOs are of interest. Second, use an inference engine to determine which IDOs pass through the filters, thus ensuring that subscriptions route (or queries return) the correct documents to clients requesting them.

Our choice of technologies to use for these steps is summarized below.

- Use the description logic SHIQ [4] in DAML+OIL [11] syntax to represent the ontology, the semantic annotations and the filters;
- Use OilEd [8] to create the ontology;
- Use FaCT DL reasoner [10] both to 1) check the consistency of the ontology during its development, and 2) determine whether there is a match between the semantic annotations of a document and a semantic filter (subscription).

DAML+OIL was chosen because it was an emerging standard. OilEd was the only GUI ontology editor available that provided [nearly] full support for DAML+OIL, while FaCT was the natural choice for an inference engine as it was linked to OilEd. We discuss tools further in Section 5. For the remainder of this paper, we elaborate on the above steps, giving examples and scenarios illustrating the ideas. Finally, we consider some related and future work.

2 Ontology

The basis for a semantic infosphere is semantic publishing and filtering. In turn, the basis for semantic publishing and filtering is a formal ontology and associated inference engine. The ontology is represented in a formal language and captures the key concepts and relationships in the domain of interest. For example, in the battlefield ontology, we have concepts like ‘units’, ‘movement’ and ‘place’.

2.1 Domain: Battlefield Reports

We used a scenario that was generated as a framework to support Natural Language Processing (NLP) techniques as part of the Boeing Real Time Information System (RTIMS) development [7]. The scenario is played on terrain which is Ft Lewis WA and vicinity. The US force was a standard J series Mechanized Battalion Task Force and the adversary was a defending Russian class motorized rifle battalion. A movement to contact framework was used to generate standard US Army spot report format (Date/Time,Location, Description, Reporting Unity, Info Evaluation) as the fictional situation developed. Outcomes, movements, incidents, casualties and losses were strictly arbitrary and scripted to generate a diversity of reports. Here is a simple spot report:

Spot Report 1
1. 220555Z Jun 92
2. 46 53’ 25" N, 122 41’ 40" W
3. Scout platoon completed recon of Rainier.
   No enemy contact.
   Sct platoon moving to battalion left flank.
4. 2-48 Inf

There are two main things being reported: 1) a reconnaissance event by a particular unit and 2) a movement by that same unit. In addition, there is information about time and location as well as the name of the unit. In general, a report can contain many different items of information. For example, a single report may have information about a unit engaging the enemy and destroying some of its equipment, the enemy abandoning a vehicle, the movement of a unit, and the fact that the unit is on watch.

This scenario data was ideal for our purposes. First, there were hundreds of messages available. Second, the ontology already existed. Finally, there was a set of semantic annotations for each message that had been *automatically generated* using natural language processing techniques, as noted above (see example below). We entered a portion of the ontology (initially represented in Prolog) into OilEd and exported it in DAML+OIL syntax.

There are four main kinds of information being reported on:
1. *Move*: for movement activities
2. *Event*: for other activities
3. *Situation Report*: to describe the status of a friendly force
4. *Enemy Order of Battle*: to describe enemy status and activities

Each has a corresponding template, for representing the semantic annotations. A single report can have one or more of each of the four kinds of template. For spot report 1, two templates were generated, an *Event* template and a *Move* template. For other spot reports, several could be generated.

2.2 The Domain Ontology

The ontology is based on the information that goes into each of the above four templates. As can been seen in the example above, each template consists of a fixed number of slot-filler pairs which represents the information contained in the message. Perhaps the most central concept is that of a unit. A unit has attributes such as Id, Size, Location, Force type (friendly/enemy), Function (e.g. recon, cavalry), Purpose (e.g. make contact, relieve), and Vehicles and Losses (e.g. equipment or personnel).

There are two roles for units: agent and object. The agent unit performs an activity. The object unit is the object of the agent activity. The agent does, and the object is done to. For example an agent unit may engage an enemy [object] unit. Note that an object role need not be filled by a unit, other things can be ‘done to’, e.g. a place might be the object of a recon activity. An agent unit has an additional attribute: instrument-of, indicating what is used to perform the action. For a unit in the object role, there is an attribute for response of object (e.g. abandon vehicle, hold position).

There is a wide variety of other things that are reported in these messages. When movement is being reported, the direction may be indicated. Reports on the status of friendly or enemy units will include the time of the report and various weapons and other equipment. For friendly units, a commander's assessment is included. Figure 2 depicts a number of these concepts in OilEd.

As part of the infrastructure for representing semantic annotations, we have a container for holding all the semantic annotations for a given spot report. This is a class called Message. As noted above, message has a number of templates associated with it. These are represented using a class called Template. It has four subclasses for the four template types (Event, Move, Situation Report, and Enemy order of Battle).

We use OilEd 'properties' (i.e. relations) to represent two main things:

1. The relation between Messages and Templates indicating which Templates are associated with a given Message.
2. The template slots. Every template slot has as its domain, a kind of Template. Some slots apply to all templates, others to two or more, and some to just one.

The relation for 1. is called 'hasTemplate'. Its domain is Message, and its range is Template. It has four sub-relations: hasMove, hasEvent, hasEoB and hasSitRep. All properties representing template slots have as their domain some kind of Template. If the slot applies to all templates, then the domain is exactly: Template. For those that apply to only some of the templates, the domain is a subclass of Template (e.g. Template-Event or Template-Move, or perhaps their union.). For example:

agentForce: Template → ForceType
objectUnitFunction: Template → UnitFunction
destination: Template-Move → Location

We use OilEd individuals to represent possible values for slots. For example, ForceType has two instances: friendly and enemy. UnitFunction has many instances such as: armor, cavalry and recon. This way of modeling the domain mirrored the way the metadata was represented and was chosen for expedience. There may be better modeling choices.

3 Semantic Annotations

Semantic annotations of the IDOs provide the foundation for semantic publishing. They are published along with the IDOs themselves. It is these annotations that are used to determine which IDOs pass through a given filter. The annotations may contain metadata about the document independent from its content, e.g. Dublin Core metadata such as date published and author. They may also describe the actual content of the document. In this paper, we focus mainly on the latter.

These annotations are like metadata in that they contain information augmenting the core IDO. However, they are different from metadata concerning the document as a whole (such as date published, author, number of words). Rather, the semantic annotations *formally represent portions of the content of the IDO in the syntax of the ontology language.* Each template is represented formally and linked to the IDO when it is published. As an example, templates generated for message 1 above are shown below. Slots with no data have been removed. This is the actual output of the metadata extraction program (RTIMS), generated from an internal Prolog representation.

```
sp-001

 1.  Template Type:                   EVENT
 2.  Message Nr:                      SP-001
 3.  Time of Event:                   220555Z JUN 92
 4.  Force Initiating:                FRIENDLY
 5.  Agent Activity:                  RECON
 6.  Agent Location:                  046 53 25 N 122 41 40 W
 7.  Object Location:                 046 53 25 N 122 41 40 W
 8.  Agent:                           SCT PLT/2-48 INF
 9.  Agent Unit ID:                   SCT PLT/2-48 INF
10.  Agent Unit Function:             RECON
11.  Agent Unit Size:                 1:PLT
13.  Object:                          RAINIER
27.  General Result:                  NO ENEMY SIGHTED

 1.  Template Type:                   MOVE
 2.  Message Nr:                      SP-001
 3.  Time of Event:                   220555Z JUN 92
 4.  Maneuver Type:                   MOVE
 5.  Agent:                           SCT PLT/2-48 INF
 6.  Agent Force:                     FRIENDLY
 7.  Agent Unit ID:                   SCT PLT/2-48 INF
 8.  Agent Unit Function:             RECON
 9.  Agent Unit Size:                 1:PLT
11.  Agent Location:                  046 53 25 N 122 41 40 W
14.  Destination:                     2-48 INF LEFT FLANK
```

For the proof of concept demonstrator, we manually translated these informal annotations into DAML+OIL using OilEd. A better solution is to adapt the automatic metadata generator to output DAML+OIL instead of formatted templates in English.

As noted above, we use a class in the ontology called Message as a container for the semantic annotations. Logically, we wish to create instances of this concept for each spot report. In a publish/subscribe context, the semantic filtering inference will only be applied to one message at a time, as each message is published. After a message is routed to the right subscribers, it is either discarded, or placed in a data store for possible future query. For testing purposes, it is convenient to create a suite of messages and to determine all at once which messages get routed to which subscriptions. This is accomplished in OilEd by doing a complete classification (computing the class hierarchy) and realization (computing the most specific classes that individuals are instances of).

For our testing, we tried creating explicit instances in OilEd. This, however, resulted in unacceptable response times when we ran FaCT to do the classification and realization. As a workaround, we represented messages as concept descriptions instead of instances. Given the fact that we do not use the full expressive power of the logic w.r.t. individuals – in particular we never assert relationships between pairs of individuals – the realization computed using this workaround is indistinguishable from that which would be computed using "real" individuals: it can be viewed as an optimization rather than a workaround. To simplify this discussion, we will ignore such details for now – in Section 6 we discuss a proper solution using an *instance store* [9].

We created a new class called Message-Individual whose instances are particular messages. By convention, we prefix the name of all individuals with 'I' followed by a dash followed by an abbreviation of the class that it is an instance of. For example, 'I-

Aty-Recon' is an instance of the class 'Activity', and 'I-Ft-Friendly' is an instance of the class: 'ForceType'. An English rendition of the formal representation of some key portions of message 1 is given below. These annotations are created using OilEd and exported in DAML+OIL syntax. They are then added to the IDOs that are published in the Infosphere.

Message-001 is a subclass of Message-Individual with the following restrictions:

1. It has an Event template, with a filler restriction requiring that the Event template:
    a. has some agentForce that has value 'I-Ft-Friendly';
    b. has some eventActivity that has value 'I-Aty-Recon';
    c. has some agentUnitId that has value 'UID-`SCT PLT/2-48 INF`';
    d. has some agentUnitSize that is has value 'I-Sz-Platoon';
    e. has some generalResultOf that has value 'I-Rslt-NoEnembySighted';
    f. has some objectOf that has value 'I-Cty-Ranier'; *etc.*
2. It has a Move Template with a filler restriction requiring that the Move Template :
    a. has some agentForce that has value 'I-Ft-Friendly'
    b. has some eventActivity that has value 'I-Aty-Recon'.
    c. has some agentUnitId that has value 'UID-`SCT PLT/2-48 INF`'
    d. has some agentUnitSize that has value 'I-Sz-Platoon'
    e. has some agentLocation that has value 'I-Grd-`046 53 25 N 122 41 40 W`'
    f. has some destination that has value '`2-48 INF LEFT FLANK`'

Next we discuss how subscribers create filters indicating which IDOs they are interested in. From here on in, we will use the term 'subscription' instead of 'filter', since that is our main emphasis.

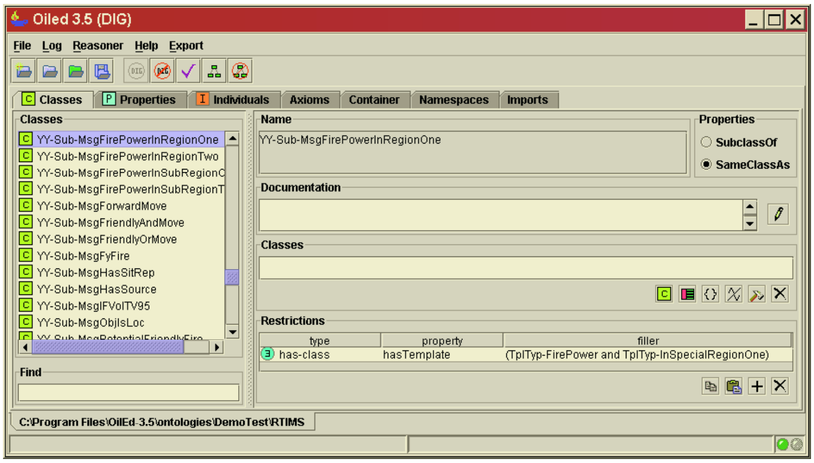
4 Semantic Filtering

Recall that semantic filtering involves two steps: 1) creating the subscriptions and 2) performing the inference to determine which messages get routed to which subscribers. We express both the annotations and the subscriptions in DAML+OIL using the vocabulary from the ontology. The FaCT reasoner[1] is used to perform the semantic filtering inference. We have seen that each IDO is published along with a semantic annotation that logically represents an instance of the concept: Message-Individual. As a workaround, this is currently represented not as an instance, but as a [pseudo]-concept in DAML+OIL (see above). A subscription is a DAML+OIL concept representing a class of IDOs that a client is interested in. FaCT is used to answer the question: "*Is the DAML+OIL individual representing the message a member of the DAML+OIL class representing the subscription*"?

4.1 Creating Subscriptions

A subscription characterizes a particular subset of messages that a user is interested in seeing. For example, a client may subscribe to all messages where:

[1] http://www.cs.man.ac.uk/fact



Classes
YY-Sub-MsgFirePowerInRegionOne
YY-Sub-MsgFirePowerInRegionTwo
YY-Sub-MsgFirePowerInSubRegionC
YY-Sub-MsgFirePowerInSubRegionT
YY-Sub-MsgForwardMove
YY-Sub-MsgFriendlyAndMove
YY-Sub-MsgFriendlyOrMove
YY-Sub-MsgFyFire
YY-Sub-MsgHasSitRep
YY-Sub-MsgHasSource
YY-Sub-MsgIFVolTV95
YY-Sub-MsgObjIsLoc
YY-Sub-MsgPotentialFriendlyFire

Name: YY-Sub-MsgFirePowerInRegionOne

Properties: ( ) SubclassOf [● SameClassAs]

Restrictions

type	property	filler
has-class	hasTemplate	(TplTyp-FirePower and TplTyp-InSpecialRegionOne)

C:\Program Files\OilEd-3.5\ontologies\DemoTest\RTIMS

Fig. 1. Firepower in Region One

1. there is some firepower activity going on in a particular region;
2. a recon unit is engaging in a move activity;
3. an enemy unit is engaging in a move activity.

Here is an English rendition of the formal concept representing the first subscription:

– The class of all messages that have at least one template with a filler restriction requiring that the template type to be both TplTyp-FirePower Template and TplTyp-InSpecialRegionOne (see Figure 1).

In turn these template types are defined as:

– *TplType-FirePower*: the class of all templates that have at least one eventActivity with a filler restriction requiring that the activity be of type: FirePowerActivity. This concept has a number of member individuals including I-Aty-Attack, I-Aty-HitMine, and I-Aty-ReturnFire.
– *InSpecialRegionOne:* the class of all templates that have at least one agentLocation with a filler restriction requiring it to be of type: InRegionOne.

All of the subscriptions are defined in a similar manner using OilEd. To keep track of which concepts are subscriptions and which are part of the main ontology we create a special class called: YY-Subscription[2]. All subscriptions are defined to be subclasses of this.

Intermediate Vocabulary

Note the use of the concept: TplType-FirePower which was used to define the overall subscription. We anticipate that in practice, a client will create a variety of such concepts. They need not be part of the overall ontology, but are specific to a subscriber or group of subscribers' needs. They are used as an intermediate

[2] This name is used to force the subscriptions to be shown in OilEd after the main ontology concepts, all of which are in alphabetical order.

vocabulary for defining subscriptions. For concepts that are not going to be used again, it could be easier to just embed the concept definition in the subscription and not create it as an explicitly named concept.

4.2 Performing Semantic Filtering

The final step is to do the semantic filtering to determine which subscribers each newly published document should be routed to. Each document has a semantic annotation associated with it that is represented as a DAML+OIL individual. Formally, we need to find all the subscription concepts that a given semantic annotation individual is a member of. In Figure 2, all and only the nodes in italics are classes that the message for Spot Report 1 is a direct instance of.

In the testing phase, to make sure the messages would be routed properly, we represented all the messages and subscriptions in OilEd. By invoking the classifier, we automatically see which messages will get routed to which subscriber, as shown in Figure 2. Note that the message will be routed to all the subscription classes that are superclasses of the subscriptions of which the messages are direct instances. This is an advantage of semantic filtering.

4.3 A Set of Related Filters

Note that some of the subsuming subscriptions in Figure 1 are listed in the bottom window. These are the classes that directly subsume ZZ-Msg-001. In addition, there are all the subscriptions that subsume these. For example each of the following subsumes the prior subscription.

- ReconInSubRegionOne: any template with recon activity in SubRegionOne
- ReconInRegionOne: any template with recon activity in RegionOne
- ReconAty: any template related to Recon activity. which is in turn subsumed by Recon: any template having anything to do with Recon units or activities.

This illustrates one potential advantage of using classification to perform filtering. All the subscriptions themselves can be pre-classified. Figure 2 illustrates a set of related subscriptions that could be defined. These are pre-classified as indicated in the figure. This means that it can be faster to find all the matching subscriptions, than if each had to be tested one by one. We have not determined under what conditions this theoretical advantage would have practical import.

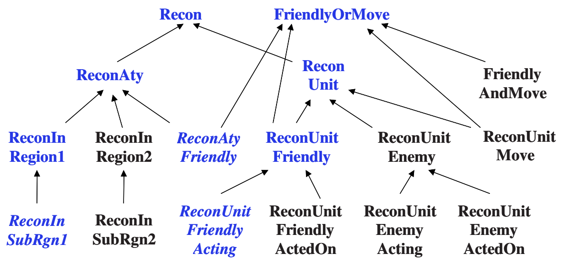


Fig. 2. Related Subscriptions

4.4 Usage Scenarios

For some brief scenarios whereby this semantic publish and subscribe system could be useful, consider a battle situation, where messages coming in from troops in the field. Different people have different jobs, and thus require different information. There would be a general in charge who needs to decide about moving troops to the best position. There will be other specialists, planners, recon experts etc. The recon specialist may need detailed information on a variety of recon units and activities, but possibly only in some regions. A pub/sub system running live would route the information to the right people at the right time. Different people could add or remove subscriptions dynamically.

5 Architecture and Tools

The XInfosphere is a platform of protocols, processes and common core services that permit applications and users to submit, discover and share information over a network. A core service of the XInfosphere is to allow for the exchange of XML information using a publish-subscribe-query model of data. Data producers publish XML data to an XInfosphere Information Data Object (IDO), which can be subscribed to by data consumers. Subscribers have the ability to filter information using the structural elements of the message.

The filtering mechanism used by the XInfosphere is extensible and allows additional message filtering techniques to be added. In order to adequately match information to user's needs based on content rather than structure, we extended the XInfosphere to support the semantic filtering of information. In this section, we discuss our selection of tools, and present the overall architecture.

5.1 System Architecture

The overall architecture is shown in Figure 3 and involves the following steps.

Ontology Creation. We loosely integrated OilEd, a graphical ontology editor developed at the University of Manchester, into XInfosphere. OilEd is bundled with the XInfosphere distribution software and can be launched from within XInfosphere. We used OilEd to develop a demonstration ontology in the domain of Army battlefield messages, exported in DAML+OIL syntax.

Semantic Publishing. We augmented the Army messages with semantic annotations using concepts and terms from the ontology. The messages, along with the semantic annotations are published to an IDO in an XInfosphere server.

Semantic Filtering. The Xfiltering mechanism was extended to allow for semantics based filtering, in addition to the existing XPath-based filtering. The messages containing semantic annotations contain a pre-defined metadata slot, which is recognized by the Xfiltering mechanism. We developed a FaCT client specifically for Xinfosphere. There are two steps: creating the filters, and performing the inference to determine which subscribers get which messages.

The filters (i.e. subscriptions) are created using OilEd. The subscriptions are included in the ontology that is loaded into FaCT. When a message is published, the FaCT client is asked to return all the subscription classes that the corresponding semantic annotation is a member individual of. Messages that successfully match the filter are forwarded to the subscriber.

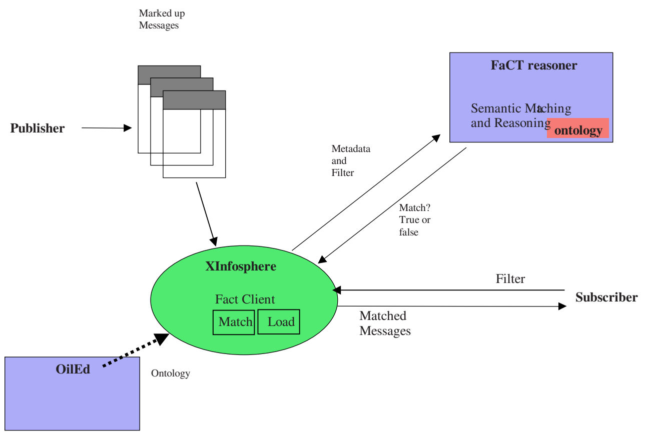


Fig. 3. System Architecture

5.2 Tools

We selected DAML+OIL as the language for representing the ontology because 1) it was highly expressive, 2) there was a fast inference engine for doing the semantic filtering (FaCT) and 3) it was an emerging W3C standard with the likely good tool support in the near future. OilEd was a natural choice for an ontology editor; the decision to use it followed from the decision to use a description logic reasoner to perform semantic filtering. OilEd was the only DAML+OIL tool ontology editor available at that time. Since then, a DAML+OIL plug-in for Protégé was developed [13]. However, it is very limited in the range of DL descriptions that you can create with it. To our knowledge, OilEd remains the only fully functional description logic ontology editor available.

We expressed both the annotations and the subscriptions in DAML+OIL using the vocabulary from the ontology. However, this was not strictly necessary, alternatives are possible. Nor did the decision to use DAML+OIL dictate what inference engine we would use. Prior to selecting OilEd to be the editor, we considered Rule-based matching in Jess as a way to do semantic filtering. This approach is used on a DAML-based publish/subscribe prototype called Ontology-Driven Knowledge Dissemination (ODKD)[3] developed as part of the Components for Ontology Driven Information

[3] http://www.daml.org/2002/02/iow/grci

Push (CODIP) program[4]. A set of rules is processed to find matches using the Rete algorithm. Logically, a matching rule is of the following general form:

If <Semantic Filter Pattern>
Then <Enable the Fetching and Routing of the appropriate document content>.

6 Discussion, Limitations, and Future Work

Using semantic rather than syntactic subscriptions provides a much more powerful mechanism for recognizing relevant messages. E.g., it might be possible to deduce from the ontology that if a unit is being engaged by a friendly unit, then it must be an enemy unit (even if this is not explicitly stated), and therefore that messages concerning such units should be routed to subscribers asking for information about enemy units.

A key message of this paper is that there is much to be learned by applying existing semantics-based technologies to solve challenging practical problems. The shortcomings of the current state of the art are highlighted. Some issues are readily resolved by good software engineering, others point to deeper issues and serve as useful feedback to the community in setting research goals. Here we discuss a variety of limitations with existing technologies and other ideas for future work.

Many limitations arose due to the fact that we were using research-ware, rather than commercial products. OilEd is a case in point; while being valuable for our proof of concept demo, we identified a variety of shortcomings that would need to be addressed in order to deploy a production system.

Individuals and Efficiency. As discussed above, performance problems arose when we tried to reason about messages represented as individuals in OilEd. This is because OilEd treats such individuals as *nominals* – singleton classes that can be used in forming ontology descriptions [14]. Reasoning with nominals is known to be hard. In this application, however, individuals are only used in a very restricted way, and it is possible to optimize reasoning (without any loss of inferential power) by treating individuals as classes.

It would obviously be convenient to have a reasoning tool that automatically provided optimized reasoning by exploiting our limited usage of individuals. This requirement has been recognized by the developers of FaCT, who have now developed a so called *instance store* for use in this and other similar applications [9]. As well as providing improved reasoning performance, the instance store uses a database to store individuals, and so is able to deal efficiently with very large numbers of individuals. This would also be of importance in a realistic application scenario where it might be necessary to deal with very large numbers of messages. Moreover, storing message individuals in a database will support not only subscriptions, but also ad hoc queries against all stored messages.

[4] http://grcinet.grci.com/maria/www/codipsite/codip.html

Scale. Whenever complex reasoning systems are used the question of scalability always arises. In this case, the number of different subscriptions is the crucial factor, as each subscription is represented as a class in the ontology. It seems reasonable, however, to assume that the number of different subscriptions would not grow without bound, and ontologies containing tens of thousands of subscription classes should not be a problem for the reasoner. In the current architecture, messages are not persistent, and so would not have a cumulative effect on performance. With the proposed new instance store solution, messages are persistent, but they are stored in a database which is able to cope with very large numbers of message instances.

Modularity. A limitation of OilEd is its lack of support for modularity. Formally, a subscription is just another class description which can be classified in the subsumption lattice. It is necessary to keep subscription [classes] separate from the lattice of classes in the ontology proper because the inference returns all classes in the lattice that a message individual belongs to. Only the subscription classes are of interest. There are many simple ways to address this, for example by creating a special class for subscriptions, and creating all subscriptions as subclasses of it. One could also use namespaces. More importantly, a user may need to save out to a separate file the subscriptions that they have created, with different users creating different sets of subscriptions using the same ontology. OilEd does not support this.

There is also a need to save out sets of messages (instances) associated with a given ontology. In OilEd, there is no way to create and export a single individual, or more generally, a specified portion of an ontology. Currently, we have to save out the whole ontology including all subscriptions and messages and extract the messages and subscriptions of interest in DAML+OIL format. This places a serious limitation on the ability to develop the ontology independently from a particular set of subscriptions. Other questions arise such as: "Do the subscriptions from different users get classified together"? and "can one user see the subscriptions created by another user"?

On-the-Fly Subscriptions. In a production setting a user will require the ability to create subscriptions and pass them along to the Xinfosphere on the fly. This requires 1) a special purpose GUI for creating subscriptions that exposes only those parts of an ontology editing tool that is of interest to an end user and 2) the ability to save out one or more individuals and/or subscriptions in a modeler manner. In general, there may be many different specialized GUIs required, one for each domain, or class of user. This will require customizable ontology editing functionality which is a layer above the GUI widget layer.

Semantic Bingo. There is a need for keeping track of multiple related messages that may come in over a period of time and which, when taken together, should result in their being routed to relevant subscribers. E.g., a risk of friendly fire might occur when: 1) friendly forces are engaging an enemy at a particular location; 2) reinforcements are moving to that location; 3) the enemy retreats. A set of messages indicating that all of these conditions have been met should be routed to battlefield commanders who can then take steps to avoid a friendly fire incident. This introduces some interesting technical challenges. One is reasoning over time – the above messages should indicate roughly contemporaneous actions; another is variable co-reference – a key element is that different messages refer to the same location. A possible solution might be to express information about time using a DAML+OIL datatype, and to use a more powerful query language such as the DAML Query Langue (see http://www.daml.org/dql/) to express complex subscriptions. Investigating this approach will be part of future work.

Limited Expressive Power. We also faced questions of expressivity requirements for the ontology language and inferencing. Two issues were particularly prominent:

Variables to enforce co-reference. As mentioned above, it is often useful for subscriptions to express interest in multiple occurrences of the same component (such as a location) in a message, without specifying any particular value. A possible solution would be the use of a more powerful query language.

Datatypes and values. Our domain requires the ability to incorporate computation with data values into the classification reasoning. A subscriber may be interested in certain regions as denoted by ranges of latitude and longitude. If a message is published with these coordinates, then numerical reasoning will be incorporated into the classification reasoning to conclude that this document should be passed along to that subscriber only if the actual coordinates fit into the specified region. Similar reasoning can be used to infer which dates fell into a certain range that a subscriber was interested in. Such information can be expressed in DAML+OIL using datatypes and values, but they cannot be reasoned with using FaCT. Full support for DAML+OIL datatypes will, however, be included in a future version of FaCT.

Plug and Play Semantic Filtering. We were successful in integrating a description logic based approach for publish and subscribe. The use of a DL in general, and of OilEd in particular provided the important advantage of using the classification engine to help ensure the consistency of the ontology. The ontology developer had no prior experience with building DL-based ontologies, nor therefore with this aspect of ontology checking. It was found to be extremely useful in spotting logical errors. The effect of this is to significantly increase confidence that the ontology correctly captures the developer's intuition about the structure of the domain.

In the long term, we do not want to force a particular ontology language, inference engine and tool suite onto a user community wishing to use a semantic filtering capability. Users may already be committed to a particular tool, or they may have special needs with respect to inference or expressive capabilities. Future work will include adding new semantic filtering technologies.

Semantic Heterogeneity. In a large or heterogeneous community, it cannot be assumed that all documents will be annotated using the same ontology. In general there will be more than one ontology that different publishers and/or subscribers use. If clients wished to subscribe to items from different publishers, or if their preferred ontologies were different from those of the publishers, then some form of ontology integration or mapping would be required [Kalfoglou and Schorlemmer 2003].

Acknowledgements. We are grateful to Deborah McGuinness, Volker Haarslev and Ralf Möller for discussions on how description logic might be used in a publish/subscribe environment. Liz Duncan, Heather Holmback, Steve Poteet and Herman Beuttel provided help with RTIMS and the battlefield scenario data.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Application*s. Cambridge University Press, 2002.
[2] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0. W3C Working Draft, May 2003. Available at http://www.w3.org/TR/xpath20/.
[3] James Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, Nov 1999. Available at http://www.w3.org/TR/xslt.
[4] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180. Springer, 1999.
[5] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Web ontology language (OWL) abstract syntax and semantics. W3C Working Draft, 3 February 2003. Available at http://www.w3.org/TR/2003/WD-owl-semantics-20030203/.
[6] Jonathan Robie. XQL (XML Query Language), Aug 1999. Available at http://www.ibiblio.org/xql/xql-proposal.html.
[7] Duncan, L., Holmback, H., Kau, A., Poteet, S., Miller, S., Harrison, P., Powell, J., & Jenkins, T. (1994). *Message processing and data extraction for the Real-Time Information Extraction System (RTIMS) project* (Boeing Technical Report BCSTECH–94–057).
[8] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: A Reasonable ontology editor for the semantic web. In Proc. of KI 2001, number 2174 in LNAI, pages 396–408. Springer, 2001.
[9] Sean Bechhofer, Ian Horrocks, and Daniele Turi. A simple DL instance store. In Submitted, May 2003. Available at http://www.cs.man.ac.uk/~horrocks/Publications/publications.html.
[10] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. KR'98*, pages 636–647, 1998.
[11] Ian Horrocks. DAML+OIL: a description logic for the semantic web. *Bull. of the IEEE Computer Society Technical Committee on Data Engineerin*g, 25(1):4–9, 2002.
[12] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Web ontology language (OWL) abstract syntax and semantics. W3C Working Draft, 3 February 2003. Available at http://www.w3.org/TR/2003/WD-owl-semantics-20030203/.
[13] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modelling at the millenium (The design and evolution of Protégé-2000). In Proc. of KAW'99, 1999.
[14] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the SHOQ(D) description logic. In *Proc. of IJCAI 2001*, pages 199–204, 2001.