Applying Semantic Rules to Achieve Dynamic Service Oriented Architectures

Suzette Stoutenburg[1], Leo Obrst[2], Deborah Nichols [3], Ken Samuel [4], and Paul Franklin [5]
The MITRE Corporation

***Abstract* - As the complexity and tempo of world events increase, Command and Control (C2) systems must move to a new paradigm that supports the ability to dynamically modify system behavior in complex, changing environments. Historically, the behavior of Department of Defense (DoD) C2 systems has been embedded in executable code, providing static functionality that is difficult to change. We propose the use of semantic models to represent system behaviors abstracted from procedural code, and we demonstrate that this provides a well-defined foundation for dynamic Service Oriented Architectures. This paper describes an implementation that models a military convoy traveling through an unsecured area under changing conditions. The W3C standard Web Ontology Language (OWL) was used to describe the battlespace domain, and the proposed W3C Semantic Web Rule Language (SWRL) was used to capture recommended operating procedures for convoys in theater. In our experiment, two sets of rules were used: one set models rules of engagement for favorable visibility conditions on the battlefield, and the other models rules of engagement for poor visibility conditions. Ontologies and rule sets were translated into integrated knowledge bases that could be consulted as a service to derive alerts and recommendations for the convoy commander. Messages injected over an enterprise service bus (ESB) provide the changing conditions that affect the battlespace. We then were able to show that a dynamic event, such as an unexpected sandstorm, causes the appropriate set of rules of engagement grounded in the ontologies to be applied to the service to guide the convoy to safety. This paper describes the overall approach and the challenges we encountered. We outline the architectural options for constructing dynamic services and describe the semantic-based approach selected. We conclude with our findings and recommendations, including a set of requirements for a standard rule language needed to support agile services.**

Index Terms – Dynamic Service Oriented Architectures, Integrated ontology and rules, Knowledge Management, Semantic rules.

[1] Suzette Stoutenburg, suzette@mitre.org
[2] Leo Obrst, lobrst@mitre.org
[3] Deborah Nichols, dlnichols@mitre.org
[4] Kenneth Samuel, samuel@mitre.org
[5] Paul Franklin, pfranklin@mitre.org

INTRODUCTION

This paper presents the findings of the MITRE Sponsored Research (MSR) project, *Toward a Standard Rule Language for Semantic Enterprise Integration.* This is a three-year effort to investigate the interaction between the rule and ontology layers of Semantic Web languages in order to determine how the standards should evolve to best address the needs of our DoD sponsors. Our motivation is to apply semantic technology to advance some of the most critical DoD requirements, including Enterprise Integration, Interoperability across agencies and agile Net-Centric system development. To that end, we focused this year on demonstrating how ontologies and rules can be integrated to achieve *dynamic* Service Oriented Architectures (SOAs), a key part of a Net-Centric Enterprise.

USE CASE

To demonstrate the potential power of agile services, we selected a Convoy Support scenario for our use case. In this scenario, a Convoy moves through enemy territory. As the Convoy approaches potential threats, a web service consults an integrated knowledge base (consisting of ontologies and rules) to generate alerts and recommendations for action. The integrated knowledge bases can be changed dynamically, thus achieving instantaneous change in service behavior. In particular, we implemented an approach in which dynamic events, such as an unexpected sandstorm, automatically trigger the swapping of knowledge bases, thus effecting dynamic services.

With this scenario, we demonstrate agility in a dynamic battlefield, a current real mission need, with application to mission challenges of the Army, Air Force, Joint Forces and others.

IMPLEMENTATION OVERVIEW

The high-level design of the application is shown in Figure 1. The components of the system include the following.

- Enterprise Service Bus (ESB)

- Google Earth Client[6]
- AMZI[7] Prolog Logic Server
- Knowledge Base
- Convoy Service
- Adapter
- Message Injector

We selected Mule[8] as our ESB solution to manage the interactions of the components in our solution. The ESB allows us to detect messages moving between components, including events that cause us to swap knowledge bases. ESB technology also applies translations when appropriate, by using the XSLT capabilities of the Adapter.

We chose Google Earth as the client, since it offers seamless integration of multiple data sources via its Keyhole Markup Language (KML)[9]. We were able to show that disparate message source data can be translated to KML and easily rendered, thus offering the potential for dynamic, user-defined displays.

AMZI's Prolog Logic Server was selected as the platform on which to host the integrated ontologies and rule base. Prolog was selected because it is based on formal logic and therefore supports the notion of proof.

The Knowledge Base consists of integrated ontologies, rules and instances. Ontologies were constructed in the Web Ontology Language (OWL) and the rules in the Semantic Web Rule Language (SWRL). These were then translated to one Knowledge Base in Prolog using techniques described later in this paper.

We constructed the Convoy Service, a software service that detects events (message exchanges over the ESB), consults the knowledge base, and delivers appropriate alerts and recommendations to the Convoy Commander via Google Earth clients.

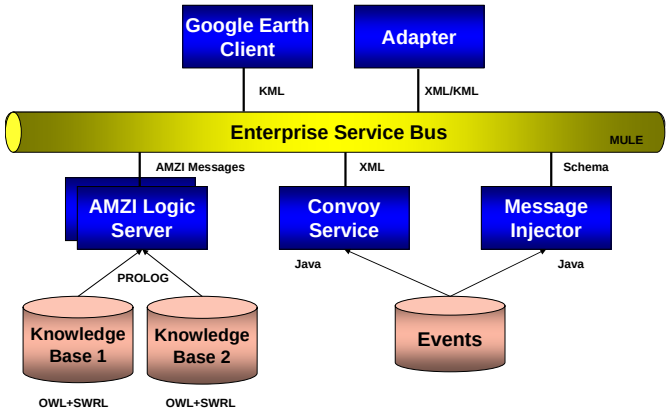

FIGURE 1
HIGH-LEVEL APPLICATION DESIGN

[6] http://earth.google.com/
[7] http://www.amzi.com/
[8] http://mule.codehaus.org/
[9] http://www.keyhole.com/kml/kml_doc.html

The service follows a very basic set of instructions:

"Something moved on the battlefield.
What does that mean for the Convoy?"

To determine the answer, the Convoy Service queries the integrated knowledge base to determine what new events mean to the Convoy. So, the rules for behavior of the Convoy Service are in fact, expressed in data, thus allowing for agile response to real-time events. The Convoy Service also detects when the rules of behavior should change (based on message exchanges over the ESB) and triggers the swapping of rules on the AMZI Logic Server.

The Adapter comprises a set of XSLTs that are invoked by the ESB to translate messages to the appropriate format as they move between components. XSLTs have been developed to convert from OWL, SWRL and RuleML to Prolog. For this particular effort, however, we expressed the rules in SWRL.

We constructed a Message Injector, which injects messages over the ESB to simulate events on the battlefield. For this experiment, we constructed messages using the Ground Moving Target Indicator (GMTI) Format (STANAG 4607) and the United States Message Text Format (USMTF) Intelligence Summary (INTSUM) message format (3/31/2004).

The application works as follows. First, events are injected over the ESB. The events include Convoy movement on the battlefield and simulated weather events. The Adapter detects the events (expressed as messages) and applies the new information to the knowledge base; in our case, both knowledge bases are updated so that they can be ready to be applied in real-time. If the Convoy Service sees an event that could potentially impact alerts and recommendations to the Convoy Commander, a query is sent to the knowledge base, requesting the latest in alerts and recommendations. These are then rendered to the Google Earth client.

Certain events are recognized by the Convoy Service as events that should trigger a change in rule sets. For example, if a weather report is issued indicating reduced visibility on the battlefield, the Convoy Service begins querying the appropriate knowledge base, that is, the knowledge base that implements rules of engagement for reduced visibility on the battlefield.

ARCHITECTURE OPTIONS FOR DYNAMIC SERVICES

There are a number of issues that must be considered when developing an integrated knowledge base to support dynamic service behavior. These include design decisions such as how to structure the rule bases (which impacts how to trigger and implement the swapping of rules), and where to store instances (in an ontology or database).

One approach to structuring knowledge bases is to build fully separate knowledge bases designed to handle different environments or situations. For example, we chose to build two independent rule sets, one to handle favorable visibility on the battlefield and one to handle poor visibility on the

battlefield. If this approach is used, triggers to invoke the applicable knowledge base could be handled by services or through a set of meta-rules in the knowledge base, as shown in figures 2 and 3 respectively.

We selected the former case to simplify the rule set necessary to model the use case. This approach also reduced the risk of unintended interactions of rules. The disadvantage of this approach, however, is that the burden of detection is on the service, which reduces the agility of the approach. Applying meta-rules offers more flexibility, since these are expressed in data and can therefore be changed without change in code.

Our implementation approach was to instantiate multiple logic servers with each rule set. This meant that each knowledge base was ready to be swapped at any time, allowing us to change service behavior instantaneously, once a real-time event was detected.

Another design decision involves whether to store instances in the database or the ontology. We chose to store instances in the ontology to simplify the overall approach. However, since we planned to swap between logic servers in real-time, we found we had to keep both knowledge bases updated with instance information and synchronized at all times. This didn't pose any performance problems and in fact, worked well in the AMZI environment. Finally, if

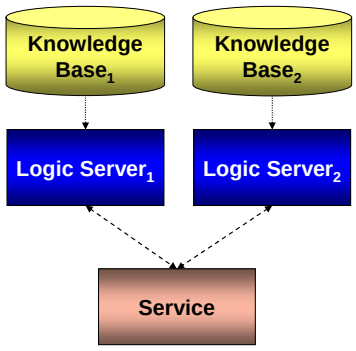

FIGURE 2
OPTION FOR DYNAMIC SERVICES:
SERVICE TRIGGERS KNOWLEDGE BASE SWAP

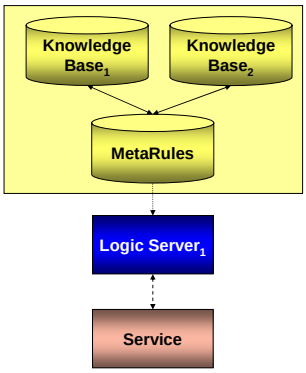

FIGURE 3
OPTION FOR DYNAMIC SERVICES:
META-RULES TRIGGER KNOWLEDGE BASE SWAP

instances are stored in a database, then tools for linking database tables to ontological concepts must be used.

ONTOLOGIES

We modeled the use case in a set of five ontologies, each named for its most important class.

- TheaterObject – to describe objects in the battlefield and reports about them.
- RegionOfInterest – to describe regions of interest on the battlefield.
- Convoy –to describe the convoy, its mission, components, etc.
- ConvoyRoute – to describe routes the convoy might take.
- ConditionsAndAlerts – to model how the knowledge base builds, resulting in conditions and alerts that affect the Convoy.

Figure 4 shows the high-level relationships between the five major ontologies and their major concepts.

The heart of the model is the class *TheaterObject*, representing the class of objects in theater (i.e., on the battlefield.) Subclasses of *TheaterObject* include *MilitaryUnit*, *Sniper*, *RoadObstacle*, and *Facility*. Instances of *TheaterObject* have a location, and they may have a speed, heading, and a combat disposition (*combatIntent*), among other features. The property *combatIntent* is used to represent whether an object in theater is friendly, hostile or has an unknown intention.

To distinguish the objects in theater from reports about them, we specified the class *ObservationArtifact*, which is the class of reports about objects in the theater. Instances of *ObservationArtifact* have properties such as time and location of the observation, the object observed, and the observation source and/or platform.. We found the distinction between theater objects and observations to be very important, as it allows inferencing over multiple reports about the same object in theater. This provided the foundation for using rules to fuse information from multiple sources. The distinction required that we build rules to transfer, or derive, property values from an instance of *ObservationArtifact* to a corresponding instance of *TheaterObject*. We modeled subclasses of *ObservationArtifact*, including *GMTIObservation* and *IntelSummary*, based on the message formats referenced above.

The *RegionOfInterest* (ROI) ontology models the class of geospatial areas of special interest surrounding objects in theater. We modeled that a *TheaterObject* is the focal object of a *DynamicROI*, since most theater objects move on the battlefield. The location of an ROI is centered on the position of its focal object. An ROI has shape, dimensions and area, the dimension and area of which depend on the type of threat or interest. For example, ROIs are used to define a "safety zone" around a convoy which must not be violated by hostile or suspicious objects. Also, ROIs are used to define the area around a reported hostile track that delineates the potential strike area of the threat.

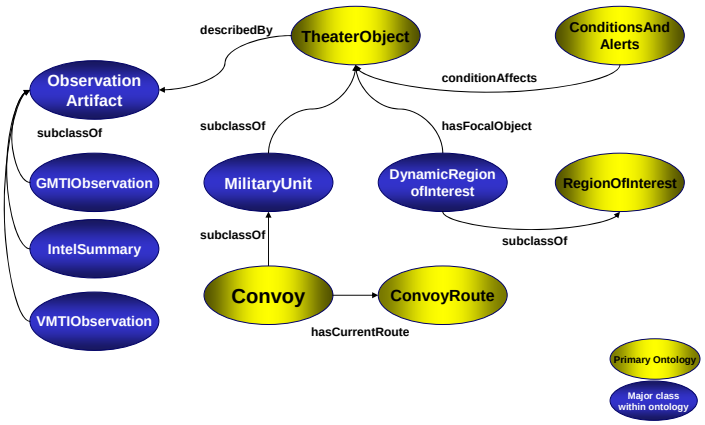

FIGURE 4
ONTOLOGY OVERVIEW

The *Convoy* ontology models the class of organized blue forces moving on the ground. This ontology allows specification of the mission, components and personnel associated with a convoy. *Convoy* is a subclass of *TheaterObject*.

The *ConvoyRoute* ontology provides a representation of paths of a convoy, including critical points on primary and alternate routes. Recommended routes can change based on application of rules.

The *ConditionsAndAlerts* ontology provides a description of situations on the battlefield based on aggregations of events and actions of theater objects. As the knowledge base grows, a set of conditions is constructed based on events on the battlefield, which can result in alerts and recommendations to blue forces. Conditions, alerts and recommendations are generated through the application of rules.

RULES

We used rules in our model for a number of purposes. First, we used rules to construct a conceptualization of the battlespace for enhanced situational awareness. This was done in two major ways. First, rules were used to transfer the characteristics of *ObservationArtifacts* to *TheaterObjects*. If a location, speed, combatIntent, etc., are reported by (or inferable from) an observation, those characteristics must be transferred to the observed object, as shown in Example Rule 1 below. Note that this design positions the model to reason over multiple messages in the future, a potential mechanism for sensor fusion.

Example Rule 1.
If there is a GMTI report about a mover, then the velocity of the mover is assigned the velocity in the GTMI report.

The second way the battlespace was conceptually constructed using rules was to establish regions of interest (ROIs) around each theater object and derive characteristics about those ROIs. For example, if the object is hostile, then we classify its ROI as an *AreaOfRedForceActivity*. See Example Rules 2 and 3 below. This also builds the basis for future enhancements, such as defining the ROI radius size based on the capability of the threat. For example, a dirty bomb would result in a much wider ROI than a sniper.

Example Rule 2.
If there is a TheaterObject, then there exists a RegionOfInterest that surrounds that object.

Example Rule 3.
If the TheaterObject is hostile, then classify its RegionOfInterest as anAreaOfRedForceActivity.

Rules were also used to synthesize information from multiple sources, for greater situational awareness. For example, the convoy's "safety zone," derived from GMTI tracking, is correlated with threat locations reported by human intelligence, allowing convoy commanders to be alerted.

Example Rule 4.
If an AreaOfThreatActivity intersects with the Convoy's RegionOfInterest, then alert the Convoy Commander of the threat.

Rules are also used for logical processing of real-time events. Specifically, as updated information modifies the picture of the battlespace, rules are used to derive new knowledge relevant to the convoy's safety. Example Rule 5 ensures that a new intel report of a threat (such as a mortar emplacement) along the planned convoy route will result in that route being flagged as unsafe.

Example Rule 5.
If a threat has a range that intersects with planned convoy route, then classify that route as unsafe.

Accumulated events result in a build-up of conditions that may lead to alerts and recommendations to the convoy. Examples are provided below.

Example Rule 6.
If a convoy's planned route is unsafe, recommend change of route.

Example Rule 7.
If threat is approaching from behind, recommend that convoy proceed at maximum speed.

INTEGRATION OF ONTOLOGIES AND RULES

Since no single Logic Programming environment existed in which OWL, SWRL and RuleML could be executed simultaneously, we had to develop one to suit the needs of our project. This involved designing a framework for translating OWL, RuleML and SWRL to an executable environment (for which we selected AMZI Prolog). The system we developed is named SWORIER, for Semantic Web Ontologies and Rules for Interoperability with Efficient Reasoning. This framework had to ensure that the features and axioms of OWL were

handled properly, a challenging problem[10]. The rules that capture the semantics of OWL are known as "General Rules" in our system. In particular, integrating the properties isSubclassOf, isMemberOf and equivalentClasses was quite difficult. We have yet to implement all features of OWL and this work will continue into the next phase of this project.

As testing of this framework began, we quickly realized that the "intensional" code (in which reasoning over OWL was performed on the fly) was operating much too slowly, on the order of minutes or worse. This was not totally unexpected, as our initial focus was on establishing a basic capability. So our next step was to devise a knowledge compilation approach to develop an "extensional" version of the code in which all a priori relations were pre-computed and cached. This greatly enhanced the performance of the approach, though the extensionalization process itself takes hours. Optimizing the extensionalization is a work in progress. The high-level view of our approach is shown in Figure 5 below. A developer models a domain in ontologies, databases, and/or rules in the formalism(s) of OWL, RuleML, and/or SWRL. This input is then translated into Prolog code using a set of XSLTs. Finally, a set of General Rules is combined with the XSLT output to provide the full semantics of OWL, RuleML and SWRL. The sets of code form a complete Prolog program that can be queried by users.

In our approach, OWL characteristics are expressed as a set of Prolog facts. For example, we represented class and subclass relations as follows.

isClass(*<Class>*).
isSubClassOf(*<Class1>, <Class2>*).

OWL properties and class membership are expressed as facts as well, as follows.

isMemberOf(*<Instance>,<Class>*).
hasPropertyWith(*<Instance>, <Property>,<Class>*)

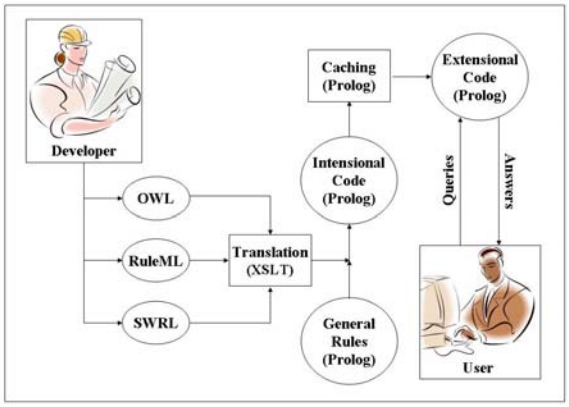

FIGURE 5
HIGH-LEVEL APPROACH TO TRANSLATING OWL, SWRL AND RULEML TO PROLOG

We combine this with a set of specialized Prolog rules designed to implement the semantics of OWL, known as General Rules. These rules operate over OWL facts to classify and derive relationships. For example, there are a number of rules designed to determine if an individual is a member of a particular class, which includes analysis of the transitive nature of subclass, the potential of union, intersection and complement class derivations, inherited properties, and more. Consider the following example.

Example OWL.

```
<motorizedInfantry rdf:ID="motorizedInfantry1">
  <hasCombatIntent rdf:resource="#hostileIntent"/>
</motorizedInfantry>
```

This OWL statement declares the existence of an instance, namely motorizedInfantry1, of the motorizedInfantry class, and states that this instance has the property hasCombatIntent with value hostileIntent. By applying the General Rules, the following translation to Prolog is the result.

Transformed Equivalent in Prolog.

(1) isMemberOf(motorizedInfantry1,motorizedInfantry).
(2) hasPropertyWith(motorizedInfantry1,hasCombatIntent,hostileIntent).
(3) isclass(motorizedInfantry). (Will result if class not previously declared.)
(4) isindividual(motorizedInfantry1).
(5) isindividual(hostileIntent). (Will result if individual not previously declared.)
(6) isMemberOf(motorizedInfantry1,theaterObject).
(7) isMemberOf(motorizedInfantry1,armedForce).
(8) isMemberOf(motorizedInfantry1,thing).

Note that General Rules implements the transitive nature of the subclass relation can be seen in the resulting statements 6, 7, 8. Most of the widely used OWL features are also implemented by the General Rules, though this work will continue into FY06. RuleML and SWRL rules are also transformed to Prolog and operate over the Prolog facts derived from the ontology. RuleML and SWRL were much easier to translate and we had minimal difficulty.

FINDINGS

First, we found that a standard rule language is very useful in constructing dynamic services. We successfully designed a generic set of instructions for the service (i.e., "Something moved on the battlefield, what does that mean for the Convoy?) then expressed the particular behaviors (which alerts and recommendations apply?) in data, that is, in SWRL. We were able to build a demonstration that supported sub second response time, rendering alerts and recommendations to the Google Earth client.

We found that OWL is expressive and meets the majority of identified DoD requirements. However, OWL should be extended to support uncertainty, *n*-ary predicates and

[10] See Ken Samuel, *et al.*, *Applying Prolog to Semantic Web Ontologies and Rules: Moving Toward Description Logic Programs* (2006), submitted to ---..

individual-denoting functions. Extensions to represent uncertainty and probability are needed for modeling real-world information, data sources, and projections in DoD domains. Support for predicates of arity greater than two would enable the representation of important relations (e.g., "*x* is between *y* and *z*") which now require awkward workarounds. Individual-denoting functions offer a convenient way of abbreviating key types of data (e.g., "(Meter 1000)").

We found that neither SWRL nor RuleML supports some of the more important DoD requirements identified in this use case, such as reasoning with uncertainty, non-monotonic reasoning, assertion of the existence of new instances, triggers to execute external code, and *n*-ary predicates. Specific DoD needs and a comparison of which are addressed by SWRL and RuleML are shown in Table 1.

We found that the state of tools for the Semantic Web are immature or non-existent. We believe that an integrated framework of tools and capabilities is needed to support semantic web development, particularly in the DoD. This is particularly true if we are to construct dynamic services using semantic standard languages. First, we need tools to allow expression of semantic rules that support the emerging W3C standards. Further, an integrated approach to specifying ontologies and rules is needed. For example, we would like to see drop-downs of imported OWL classes and properties while rules are being built, similar to the way XML Spy and other tools offer drop down lists of valid XML schema entries. We also would like to see integrated validation of ontologies and rules, when rules refer to ontologies or data sources. The standard framework should include tools for specification, validation, translation, execution and debugging. The GUI should hide complexity of rule constructs. Most importantly, integrated reasoning engines to operate over ontologies and rules, with knowledge compilation for performance, are required.

TABLE 1
COMPARISON OF SWRL VS. RULEML
WITH REGARD TO DOD REQUIREMENTS

Capability	SWRL	RuleML
Expression and reasoning with uncertainty		
N-ary predicates (3 or more) - Properties that relate more than 2 entities		
Dynamic assertion of existence of individuals		
Dynamic class declaration	✓	
Logical negation		✓
Trigger external services		?
Built-in functions	✓	
Unary predicates	✓	✓
Management of "slots" (roles of predicate arguments)		?
Compound consequents	✓	*
Explicit import and reference to ontological classes and properties	✓	
Integrity Constraints	*	✓
Rule precedence	*	?
Supporting tools		
Complex atom representation (nested atoms)	*	*
Rule annotation (i.e. Open vs Closed world treatment of rules)		
Support for global variables		

✔ Capability supported
* Workarounds exist
? Claims to support, not tested

RECOMMENDATIONS AND NEXT STEPS

We believe that OWL should be extended for use in the DoD to include a model for uncertainty and *n*-ary predicates, at a minimum. We also recommend that DoD's top needs for a standard rule language be incorporated into the emerging standard being developed by the W3C Rules Interchange Format Working Group.

We plan to propose extensions to SWRL; perhaps our recommendations could form a DoD extension to the W3C standard when it emerges. We also plan to transition the technology to build and deploy dynamic services to solve our sponsors' needs.