Dynamic Service Oriented Architectures through Semantic Technology

Suzette Stoutenburg[1], Leo Obrst[2], Deborah Nichols[2], Ken Samuel[2], and Paul Franklin[1]

[1] The MITRE Corporation, 1155 Academy Park Loop
Colorado Springs, Colorado 80910
[2] The MITRE Corporation, 7515 Colshire Drive
McLean, Virginia 22102
{suzette, lobrst, dlnichols, samuel, pfranklin}@mitre.org

Abstract. The behavior of Department of Defense (DoD) Command and Control (C2) services is typically embedded in executable code, providing static functionality that is difficult to change. As the complexity and tempo of world events increase, C2 systems must move to a new paradigm that supports the ability to dynamically modify service behavior in complex, changing environments. Separation of service behavior from executable code provides the foundation for dynamic system behavior and agile response to real-time events. In this paper we show how semantic rule technology can be applied to express service behavior in *data*, thus enabling a *dynamic* service oriented architecture.

Keywords: Dynamic Service Oriented Architectures, Integrated ontology and rules, Knowledge Management, Semantic rules, Web services.

1 Introduction

In this paper, we describe an implementation that applies a proposed standard rule language with ontologies to construct dynamic net-centric web services.   We successfully show that rules for service behavior can be expressed in XML-based languages with formal semantics. This greatly simplifies the service code and allows rules for behavior to be changed dynamically with no code modifications, thus achieving an agile service architecture.

We modeled a military convoy traveling through an unsecured area under changing conditions.   The World Wide Web Consortium (W3C) standard Web Ontology Language (OWL)[1] was used to describe the battlespace domain and the proposed W3C Semantic Web Rule Language (SWRL)[2] was used to capture recommended operating procedures for convoys in theater. We translated the ontologies and rules

---

[1] http://www.w3.org/TR/owl-features/
[2] http://www.w3.org/Submission/SWRL/

into a logical programming language to produce an integrated knowledge base that derives alerts and recommendations for the convoy commander. 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. When a dynamic event, such as an unexpected sandstorm, occurs, this causes the latter set of rules of engagement to be applied to the service to guide the convoy to safety. In this paper, we provide a description of our approach and outline the architectural options for constructing dynamic services. We briefly describe the semantic-based approach utilizing the ontologies and rules that we developed. We conclude with our findings and recommendations.

2 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. These alerts and recommendations are provided to the convoy commander for decision support. The integrated knowledge bases can be switched 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.

3. 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[3] Client
- AMZI Prolog Logic Server[4]
- Knowledge Base
- Convoy Service
- Adapter
- Message Injector

We selected Mule[5] as the ESB solution to manage the interactions of the components in our solution. The ESB detects messages moving between components,

---

[3] http://earth.google.com/
[4] http://www.amzi.com/
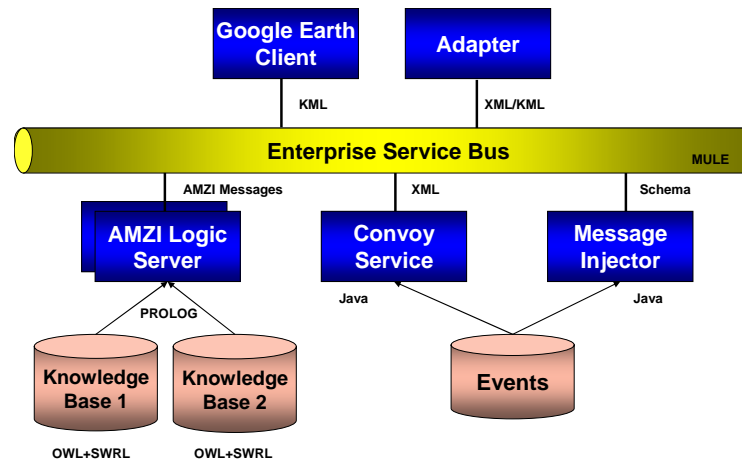[5] http://mule.codehaus.org/

Fig. 1. High-Level Application Design

including events that cause the swapping of 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). We were able to show that structured data from heterogeneous sources 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.

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.

The service operates under 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[6] to Prolog.

We constructed a Message Injector, which sends 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, the Message Injector sends event messages over the ESB, such as convoy movement and weather events. The Adapter detects the event messages and applies the new information to both knowledge bases, so that they are both ready to be applied at any 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 AMZI Prolog Logic Server, requesting the latest alerts and recommendations. The Google Earth client then presents them to the convoy commander.

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 stops querying the high-visibility knowledge base, switching to the low-visibility knowledge base. More details on the separation of rules from software are provided in section 5.


4. 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 whether to store instances in an ontology or a 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.

---

[6] http://ruleml.org/
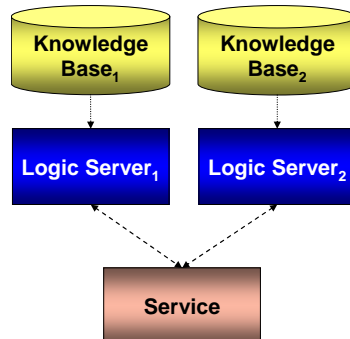
Fig. 2. Option for Dynamic Services: Service Triggers Knowledge Base Swap
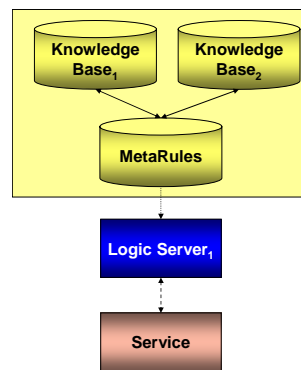
Fig. 3. Option for Dynamic Services: Meta-rules Trigger Knowledge Base Swap

We selected the former case to simplify the rule set necessary to model the use case. This approach also reduces the risk of unintended interactions of rules. By instantiating a different logic server for each rule set, each knowledge base is ready to be swapped at any time, allowing us to change service behavior instantly, whenever a real-time event is detected. The disadvantage of this approach, however, is that the burden of detection is on the service, which reduces agility. Applying meta-rules, on the other hand, offers more flexibility, since these are expressed in data and can therefore be changed without change in code.

Another design decision involves whether to store instances in a database or 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 instances are stored in a database, then tools for linking database tables to ontological concepts must be used.

5. Ontologies

To understand how we separated rules from service behavior, it is necessary to have a basic understanding of the ontologies we built to model the use case. The following ontologies were constructed, 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 conditions and alerts that affect the convoy.

Figure 4 shows the high level relationships between the five major ontologies and their key concepts. Note that in some cases, the name of an ontology is also the name of a class. For example, *Convoy* is the name of the ontology but it is also the name of a class in the ontology. Thus, it is shown as a subclass of *MilitaryUnit*. This was done to show the high level structure of the ontology set.

The heart of the model is the class *TheaterObject*, representing objects in theater (i.e., on the battlefield.) Subclasses of *TheaterObject* include *MilitaryUnit*, *Sniper*, *RoadObstacle*, and *Facility*. An instance of *TheaterObject* has a location, and it may have a speed, heading, and a combat disposition (*combatIntent*), among other features.
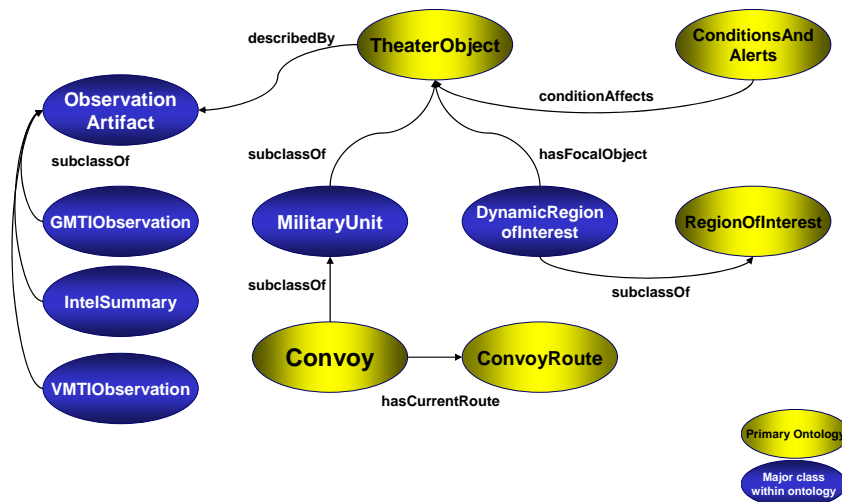

Fig. 4. Ontology Overview

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 created the class *ObservationArtifact*, which is the class of reports about objects in the theater. An instance of *ObservationArtifact* has properties such as the 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 theater objects. Each *TheaterObject* is the focal object of a *DynamicROI*, since most theater objects move on the battlefield. An ROI is centered on the position of its focal object. An ROI has shape, dimensions and area, which may 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.

The *Convoy* ontology models the class of organized friendly 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 possible 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 friendly forces. Conditions, alerts and recommendations are generated through the application of rules.

6. Rules

Rules were used to specify the behavior of the Convoy Service by incrementally constructing a knowledge base of the situation on the battlefield from which alerts and recommendations could be derived and made available to the convoy commander. To that end, rules were applied in numerous ways. 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, then recommend change of route.

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

7. Convoy Service Design

The design of the Convoy Service is very simple. The service basically monitors the ESB for messages that provide information about events on the battlefield. When the service detects that a relevant event has occurred that may impact the convoy, the service queries the knowledge base to determine if new alerts and recommendations may have been generated by the new event. The query is a semantic one, using the ontologies and rules modeled in the integrated knowledge base. For example, consider the case in which a track of unknown intent is moving on the battlefield. The Convoy Service detects the event and queries the knowledge base, basically querying for the concept of "unknown and hostile movers" and "alerts and conditions". The data source of instances of these concepts does not matter, since the query is semantic; that is, over the *concepts* not the tables. The query triggers a set of rules to fire, including rules to apply the new position of the unknown mover, determine the size and location of the region of interest around that mover, and whether or not the mover is now within proximity of the convoy. If the unknown mover is within proximity, additional rules fire to construct conditions that lead to the generation of alerts and conditions.

The Convoy Service also controlled which Logic Server was consulted when battlefield events were detected. Recall that we implemented two sets of rules; one was used to model rules of engagement for favorable visibility conditions and the other set modeled rules of engagement for poor visibility conditions. When the Convoy Service detected particular weather events, in particular, an unexpected sandstorm, the service would switch logic servers. Subsequent queries would be launched to the logic server instantiated with the more conservative (poor visibility) rule set.

8. Findings and Conclusions

First, we found that expressing service behavior in structured data is a feasible solution for constructing dynamic services. By separating the rules from the executable code, and expressing service behavior in *data*, we show that dynamic services can be constructed. So, given a real time event, we can swap rules sets, thus delivering services that can be agile in real time. We were able to design a generic set of instructions for the service (i.e., "Something moved on the battlefield, what does

that mean for the convoy?) then express 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 believe that a standard XML-based language with formal semantics, such as SWRL, is the best choice for expressing service behavior since this allows the rules to be "understandable" by machines, building the foundation for a Machine-to-Machine Enterprise. Also, since the languages are W3C established or proposed standard, they are *inherently extensible,* meaning that as other ontologies and rule sets are developed, they can potentially be linked and reused for a richer knowledge model. We also found that ontologies are useful to bridge the "dialects" of each data source, allowing querying over *concepts*, thus freeing the application from having to understand the many ways of representing an event.

Regarding the richness of the W3C semantic languages, 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 (n > 2) and individual-denoting functions. 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. Extensions to these languages will be proposed to support the DoD.

We found that the state of tools for expressing rules in W3C proposed standards are immature or non-existent. We believe that an integrated framework of tools and capabilities is needed to support dynamic service development, particularly in the DoD. First, we need tools to allow the 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 down lists of imported OWL classes and properties while rules are being built, similar to how XML Spy and other tools offer drop down lists of valid XML schema entries. We would also 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 the complexity of rule constructs.   Most importantly, integrated reasoning engines to operate over ontologies and rules, with knowledge compilation for performance, are required.