10TH INTERNATIONAL COMMAND AND CONTROL RESEARCH AND TECHNOLOGY SYMPOSIUM

THE FUTURE OF C2

Title: **CommandRESPONSE: An Open Source Instantiation of an Event-Driven and Services-Based Architecture for Net-Centric Warfare and Public Safety Applications.**

Topic: Net-Centric Warfare and Public Safety/Homeland Security

Authors:
Dee Quashnock, Space and Naval Systems Command
Pericles Haleftiras, Antin Div. of SYS Technologies, Inc.;
Mike Bennett, Antin Div. of SYS Technologies, Inc.;
Dan Dunaway, Antin Div. of SYS Technologies, Inc.
Jay Ford, Antin Div. of SYS Technologies, Inc.
Khanh Vo, Antin Div. of SYS Technologies, Inc.
Myoki Spencer, Antin Div. of SYS Technologies Inc.

Point of Contact: Pericles Haleftiras
Antin Div. of SYS Technologies, Inc.

5050 Murphy Canyon Road
Suite 200
San Diego, CA 92130
619-851-3210 (w)/858-715-5510(f)
pericles.haleftiras@antin.com
www.systechnologies.com

Abstract

CommandRESPONSE:
An Open Source Instantiation of an Event-Driven and Services-Based Architecture for Net-Centric Warfare and Public Safety Applications.

Today's navy warfighting and domestic public safety professionals must deal with information overload, new types of threats and complex fields of engagement while attempting to achieve a shared situational awareness and make critical decisions uder a high degree of uncertainty . The current information systems supporting these efforts were designed to accomplish singular functions with each system requiring its own information management and processing infrastructure. These systems are difficult to adapt or enhance and limited horizontal integration of information amongst the systems exists. 'Stovepiped' systems result in making the creation of a common operational picture of the 'point-of-problem' difficult at best. Moreover, the organizations procuring such solutions pay for that infrastructure many times over.

The nature of an asymetric threat can change overnight. Warfighting and public safety systems designed to address such threats must themselves be flexible and capable of rapidly adapting to new threat scenarios. This dynamic, combined with the economic drivers to reduce total cost of ownership, operation and manning, has driven the adoption of net-centric technologies and architectures that facilitate the composability of mission capabilities and associated information environments while substituting networks, sensors and intelligence for manpower. New mission capabilities must be constructed 'on-the-fly'.

Joint/coalition/military-civilian/combined operations place further burdens on information sharing, the decision making process and distribution of command intent. First responders and military warfighters both need information management and interchange that supports secure, tailorable and timely access to all required information for real-time planning, resource allocation, control, and execution of the incident response mission.

The Office of Domestic Preparedness (ODP) funded a pilot initiative in the City of Chula Vista to explore the creation of a domestic information sharing and collaborative sense and respond backbone leveraging the most current net-centric warfare technologies, architectures and thinking. The envisioned platform had to be low cost, standards-based, vendor-agnostic, capable of supporting current and future operational requirements of first responder agencies and successfully deliver on the multi-faceted promise of connection-interoperability-and-agility.

Platform success would be measured by the rapidity and robustness with which existing public safety systems such as computer-aided dispatch, incident management and consequence assessment tools could be 'woven' together with new capabilities such as geo-spatial data fusion, vehicle tracking, sensor networks, asset management, event correlation and multi-channel event notification. Once federated, the sense and respond backone needed to provide the capability to handle extensive real-time information and event flows through such systems and to orchestrate new mission capabilities composed of functionality from these formerly 'stovepiped' systems 'composed' in new combinations.

The CommandRESPONSE Platform concept was successfully demonstrated during a large-scale threat exercise involving a staged terrorist attack on a hazardous waste shipment. Documented enhancement of first responder capabilities, enablement of new CONOPS in support of that mission and a dramatic impact on the traditional cost equation for delivery of such capabilities were measured and documented. Maximum agility to leverage current assets, incorporate new capabilities and 'future-proof' the platform were all demonstrated in a cost effective, vendor-agnostic way. The success of this pilot demonstration is being followed by a second project phase designed to incrementally rollout the capability on a limited basis throughout the greater San Diego region.

Chula Vista CommandRESPONSE Project

Today's domestic public safety, emergency management and homeland security professionals must deal with information overload, new types of threats and a complex field of engagement while attempting to achieve a shared situational awareness and make critical decisions uder a high degree of uncertainty . The current monolithic information systems supporting these efforts were designed to accomplish singular functions with each system requiring its own information management and processing infrastructure (stovepipes). These systems are difficult to adapt or enhance and limited horizontal integration of information amongst the systems exists. Creating a common operational picture of the 'point-of-problem' is difficult at best. To make matters worse, local, state and federal government agencies pay for this infrastructure each time they acquire a new capability.

The Department of Defense (DoD) faced a similar state of affairs with its requirement to provide command and control capabilities in support of the warfighter. Integrated command, control, communications, computing, intelligence, surveillance and reconaisance solutions (C4ISR) emerged to address these requirements. Currently, a new generation of C4ISR capabilities is emerging that fully leverage the proliferation of network and Internet-based technologies. Net-centric C4ISR promises dramatic advancements in warfighting capabilities while increasing the agility of the support infrastructure to adapt to new technical capabilities and changes in the threat. Great value and leverage can be gained from the application of DoD Net-Centric C4ISR approaches and technologies to the needs of domestic law enforcement, first responders, emergency preparedness and disaster response teams. This new class of C4ISR application can be considered 'Net-Centric Domestic C4ISR'.

Interoperability, connection and composability of mission capabilities (agility) are the business drivers for Net-Centric Domestic C4ISR solutions. First responders need information management and interchange that supports secure, tailorable and timely access to all required information for real-time planning, resource allocation, control, and execution of the domestic incident response mission. Furthermore, the dynamics of the new threat require the ability to rapidly construct new mission response capabilities 'on-the-fly'.

Department of Homeland Security (DHS) funding has provided the impetus for a staggering number of technology demonstrations and pilot projects focused on Domestic C4ISR challenges. The good news is that significant capabilities have been identified with the potential to enhance first responder capabilities. The bad news is that these projects are like iron filings piled on a sheet of paper without the benefit of an underlying magnetic force to bring coherence and alignment to the initiatives.

The Office of Domestic Preparedness (ODP), in collaboration with the Space and Naval Warfare (SPAWAR) Systems Center San Diego and the City of Chula Vista designed a pilot initiative to establish that 'magnetic force' in the form of a reference enterprise architecture that delivers true interoperability and agility.

A reference architecture defines a standard framework to integrate, align and deliver enterprise capabilities. The pilot initiative, called Chula Vista CommandRESPONSE Pilot, developed the first specific instantiation of an information sharing and command&control backbone based upon ODP/SPAWAR/Chula Vista's vision for a domestic C4ISR reference architecture.

The five core elements of the CommandRESPONSE information sharing and command&control backbone are:

1. The **'Domestic Information Space'** (DIS) that aggregates, integrates, fuses, and intelligently disseminates relevant information to support effective decision-making. The DIS provides individual users with information tailored to their specific functional responsibilities. It integrates data from a wide variety of sources, aggregates this information, aligns data semantics, transforms formats and distributes it in the appropriate form and level of detail required by different users. The DIS employs standards-based ontology-driven semantic integration and a publish-subscribe messaging approach.

2. The **'Domestic Common Operating Picture'** (DCOP) that provides a real-time, tailorable portal interface incorporating all relevant and actionable information that is geo-referenced to a set of computerized maps. The DCOP facilitates collaborative planning and assists all participating agencies to achieve shared situational awareness. To achieve a coordinated response to a given situation, everyone must have access to the visual display of the same relevant information. The DCOP leverages standards-based portal technology with the addition of a geo-spatial intelligence component all of which has been evaluated and assessed from a human systems integration perspective.

3. The **'Domestic Command&Control Backbone'** (DC2B) that provides a services oriented integration framework. Services such as communications, decision support, remote monitoring, alerting-notification-recall, location tracking, computer assisted dispatch, etc., are made available as standards-based web services (XML, SOAP, WSDL, and UDDI) to be 'plugged' into the DC2B and access/feed the DIS and DCOP. Services can publish or subscribe to information and events through the DIS. In addition to web services, the DC2B provides other standards-based integration mechanisms (i.e. java messaging service (JMS), java connection adaptor (JCA), XML messaging & presence protocol (XMPP) and Remote Syndication Service (RSS)) to facilitate alternate approaches to the incorporation of existing and new computational assets as needed.

4. The **'Emergency Services Orchestration Engine'** (ESOE) that provides a standards-based process definition, execution and management environment enabling the creation of public safety, emergency preparedness/response and homeland sercurity processes. These processes are themselves composite services derived from the portfolio of application functionality plugged into the DC2B. The ESOE specifically employs standards-based BPEL and BPEL4WS in its process definition and execution approach.

5. The **'Domestic Extreme Event Notification Engine'** (DEEN) that delivers standards-based alerts, notifications and recalls can be issued to small groups or mass populations across multiple channels of communication including voice, fax, pager, SMS text messaging, email, WAP devices and a desktop client. Messages can be ad-hoc or pre-defined. Supplemental information can be delivered across each of the notification channels. Libraries of pre-defined communication plans, configurable by threat-type, role, message and supplemental information can be created. Receipt acknowledgement is tracked across all channels. GIS-driven outbound voice alerts are also possible. Inbound 'Status Check' functionality allows field personnel to call into the system and identify themselves as 'alive and accounted for' or any mitigating circumstances about their current condition. All alert and notification capabilities can be programmatically triggered by other services plugged into the DC2B.

The CommandRESPONSE Platform provides the information, command&control, compound portal and process environments into which public safety, emergency preparedness & response, and homeland security applications with standards-based interfaces (web services, JMS, etc.) can be incorporated. New first responder mission capabilities can be created by orchestrating services accessed across the incorporated application portfolio. This provides the maximum interoperability and agility to leverage current assets, incorporate new capabilities and 'future-proof' the platform. All of this is accomplished in a vendor-agnostic way.

Services-Oriented Integration Network & Service Container

The five CommandRESPONSE core elements are enabled by a services-oriented integration network (SOIN). The SOIN can be considered a standards-based services container which is similar, in concept, to an application server focused on the specific goal of hosting standards-based integration/interoperability web services. The Service Container combines XML, web services, a message invocation framework, content-based routing & semantic transformation and business process orchestration.

The Service Container is a remote process that can host software components with the specific goal of delivering integration capabilities and executing business process flow logic. The Service Container is simple and lightweight while supporting many discrete functions. The Service Container can manage multiple instances of a service. Several Service Containers may be distributed across multiple machines for the purposes of scaling up to handle increased message volume. In this manner, a highly distributed series of Service Containers could be globally networked together to provide service-oriented integration capabilities on a global scale.

Integration capabilities of the Container are implemented as separate services. Specialized integration services, that can be extended and augmented using a service interface allow applications to easily interoperate. Integration services include:

- Specialized transformation services, with the responsibility of applying XSLT stylesheets to convert in-flight XML messages from one dialect to another.

- Content-based routing services with apply Xpath expressions to look contextually int the nodes of the XML documents as they pass through the service and make determinations on where to send the message next.

- XML logging services that extract a copy of XML messages as they travel through a portion of a process flow and logs them for auditing and tracking purposes.

Because the integration capabilities are themselves implemented as services, hosted in the Service Container, they can be independently deployed anywhere within the network. The result is precise deployment of integration capabilities at specific locations, as required. These deployed integration services can be scaled independently as necessary. Services deployed in this manner can be easily upgraded, moved, replaced or replicated without affecting the applications that they cooperate with.

Service Container Management Interface

The Service Container handles inflow and outflow of management data such as configuration, auditing and fault handling. The Service Container management interface is implemented using the Java Management eXtensions (JMX). Management input data can originate from a remote management console issuing commands such as start, stop, shut down, and refresh the cache. Management output data consists of event tracking, such as notification that a message has successfully exited the service or that a failure has occurred. These inputs and outputs are managed by a JMX management console. They could be directed to some other management tool, using standard protocols such as the simple Network Management Protocol (SNMP).

Container Service Interface

The Service Container provides the care and feeding of the services. It provides the message flow in and out of a service. It handles service lifecycle and itinerary management. Thread pooling allows multiple instances of a service to be attached to multiple listeners within a single container. The Container manages an entry endpoint and an exit endpoint. These endpoints are used by the Container to dispatch a message to and from the service. XML messages are received by the service from a configurable entry endpoint. Upon completion of its task, the service implementation simply places its output message in the exit endpoint to be carried to its next destination using a ‘request/reply’ or ‘reply-forward’ pattern.

The output message may be the same message that it received. The service may modify the message before sending it to the exit endpoint. The service may create a completely new message to serve as a ‘response’ to the incoming message and send the new message in the exit endpoint. What is placed in the exit endpoint depends on the context of the situation and the message being processed. One input message can transform into many output message, each

with its own routing information. The routing service implementation is implemented as a specialized transformation service that applies an XSLT stylesheet to the XML document to produce multiple outputs.

SOA via Abstract Endpoints

Applications and services that need to connect to each other and share data through the container are treated as abstract endpoints. An endpoint may represent a discrete operation, such as a specialized service for correlating events. The underlying implementation of the endpoint could represent a local binding to an application adapter, a callout to an external web service or a single, monolithic application such as a legacy computer-aided dispatch system. All service endpoints supported by the Service Container are equal participants in an event-driven services-oriented architecture (EDSOA). The Service Container is the physical manifestation of the abstract endpoints and provides the implementation of the service interface.

Service endpoints are just logical abstractions of services that are plugged into the Service Container. Higher-level tools enable the assembly of service endpoints into process flows. These process flows can be defined in a standards-based business process definition/execution language such as BPEL4WS. The task of building out a SOIN involves tying together service endpoints and applying choreography, transformation and routing rules to process flows between applications. The actual physical locations of the services can be anywhere that is accessible by the SOIN.

Messaging Backbone

A highly scalable enteprise messaging backbone is at the heart of the Service Container. The messaging backbone is capable of asynchronously transporting data, information and events as messages in a secure and reliable fashion. The messaging backbone supports a variety of asynchronous store-and-forward capabilities including SOAP-over-HTTP/S, WS-Rel*SOAP, MOM/JMS (Java Messaging Service), SOAP-over-JMS-over-HTTP/S, and JMS-over-SSL. The MOM core of the Service Container is a multiprotocol messaging bus that supports asynchronous delivery of messages with configurable QoS options ranging from exactly-once delivery with transactional integrity through high-performance, low-latency best-effort delivery. The MOM core is capable of spanning geographic locations and of traversing firewall security using full certificate-based authentication. The Service Container is capable of delivering messages reliably using message persistence and a store-and-forward capability.

The Service Container messaging layer is also able to cluster message servers into a grid-like infrastructure, where all message traffic can be routed seamlessly between message servers without exposing the underlying routing details to the application level of the Service Container process level.

Diversity in connectivity and message transports is a core capability that is fundamental to making the Service Container a realistic approach across a global public safety, emergency response and homeland security ecosystem. It is not reasonable to expect that all applications across the board resort to the same interface technology, whether it be proprietary, web services-based, Java-based, or .NET-based, to participate in the services-oriented integration network.

Because the Service Container can support multiple ways of connecting into it, applications don't require any drastic changes to participate in the SOIN. Applications should be able to connect into the Service Container with as little modification as possible. It is the Container, not the applications, that provides the flexibility in connection technologies.

Message Invocation Framework

As defined above, there can be a number of protocol stacks for transporting XML messages. SOAP is not a prerequisite for a SOA, although it is fairly common for Service Container channels to carry SOAP messages. In some cases, the Service Container my even strip off the SOAP envelope and just use the remaining XML payload information as the message that gets passed between services. Quality of service (QoS) levels can vary depending on which protocol is chosen to carry a message.

Bridging is one way to attach a legacy protocol such as FTP or SMTP (email) into a Service Container. An FTP or SMTP bridge is implemented as a specialized messaging client, which converts data from one protocol to and from the MOM channel. What you gain by using a protocol bridge is a common backbone for transporting data across the enterprise. This pushes the unreliable FTP out to the outermost edge of the SOIN and in many cases completely eliminates its use by removing the dependency on FTP for transporting bytes across the network, and simply retaining the file-based interface between an application and the Service Container.

The Service Container can also provide a bridge across MOM implementations (WebsphereMQ, TIBCO RV, SonicMQ, etc.). MOM bridging allows the Service Container to extend its reach into an existing MOM-based integration infrastructure without needing to tear it our and replace it. In some instances, the Service Container can treat 'foreign' message channels directly as endpoints.

Another form of integrating the Service Container MOM core with another protocol is through direct support for that protocol within the MOM's message broker implementation. The message broker, itself, provides the bridging, or mapping, between the external protocol and the internal MOM channel. An example would be a mapping between SOAP-over-HTTP and SOAP-over-JMS.

HTTP Protocol Handler

The ability to process HTTP requests, both inbound and outbound, are critical capabilities that enable MOM and web services implementations. Providing the HTTP support directly in a message broker allows the configuration and management of server clustering, fault tolerance, and high availability from a single vantage point without additional external processes. The servers in the cluster may act as one logical server, which can communcate on multiple protocol channels that send and receive MOM messages, SOAP messages, or non-SOAP HTTP requests. This means that there are no additional moving parts or additional processes, such as web servers, servlet engines, ISAPI filters, that can become bottlenecks or points of failure.

The Service Container provides an invocation framework that allows a Container service to be invoked as a result of an external HTTP request. A single HTTP request, whether SOAP-based or otherwise, may even trigger the kickoff of an entire business process. The interface to the Container service or business process is described using WSDL and exposed to the outside world as a first-class web service endpoint. While the Service Container is by and large intended for asynchronous processing, it also can process synchronous request/reply message invocations.

The Service Container introduces an HTTP protocol handler which allows the container MOM core to listen for inbound HTTP or SOAP requests, much like a web server or application server would. The protocol handler performs a mapping of the HTTP content into a JMS message and places the JMS message into a queue or a pub/sub topic destination.

In an asynchronous messaging environment, the request/reply is accomplished by setting up two messaging channels: one for the request and one for the reply. The channels may be either pub/sub-based or point-to-point queue-based. The service handler on the other end of the queue or topic invokes the service, correlates the reply with the request, and places the reply on another 'outbound' queue to be sent back to the HTTP protocol handler using a ReplyTo pattern.

In the event that the service or part of the business process is temporarily unavailable, the HTTP protocol handler generates an error back to the sender if the response time exceeds a configurable timeout period. If the service invocation generates an error, then that error will also be propagated back to the caller.

Interactions between an HTTP request and a Service Container process or service often needs more than the requisite subsecond response time required to satisfy a synchronous request/reply. In these cases, the Service Container utilizes asynchronous request/reply. The inbound request contains a URI indicating where to send the reply, which is mapped to a web service endpoint that the requestor uses to listen for the reply.

SOAP Protocol Handler

The SOAP protocol handler is similar to the HTTP protocol handler excepting the ability to understand a SOAP envelop, validate it and generate SOAP faults under error conditions. Validation is performed to verify that the content of the inbound request is indeed a SOAP envelop. Once validated, the SOAP envelop is processed and mapped onto a JMS message and back again.

XML and Ontologies

XML is the language of integration. XML and higher-order ontological enhancements to XML (XML Schema, RDF, RDF Schema and Web Ontology Language (OWL)) are a means of providing the mediation between diverse data structures, formats and meaning. Service Container-enabled data transformation and content-based routing services support a generic data exchange architecture that insulates individual applications from wholesale changes in data structures.

XML is human-readable, readable by many tools and is writeable. XML is well suited for integration tools that can visually create and manage sophisticated data mappings (ontologies). Any well-formed XML document is parseable by a variety of standard parsers. Data represented by XML is more readable and consumable. It is easily read by humans, and easier to consume using parsing technology because it carries context, or information about the nature of the content in the form of metadata. Individual XML elements can be identified and consumed independently from the rest of a well-formed XML document.

An application and its external data representation can be extended without breaking the links between it and the other applications it shares data with. XML is extensible, in that portions of an existing XML document can be modified or enhanced without affecting other portions of the document. Additional data elements and attributes can be added to a portion of an XML document, and only the application that will be accessing that new element needs to be modified.

Other applications that don't require the extended data can continue to work without modification. With XML, we can extend data without adversely affecting all the producing and consuming applications that need to exchange that data. This allows us to selectively upgrade interfaces and data structures. New business requirements driving the need to extend the original data can now be easily incorporated without disturbing the existing relationships between that application and other consuming applications.

Application data is now represented as an XML document, consistent with some information model (schema) captured as an ontology. The ontology is expressed in RDF-S or OWL statements. Ontologies are capable of expressing rich relationships between entities and attributes beyond those possible in a relational database model. These relationships, as captured in the ontology, are programmatically available, opening up new semantic processing possibilities.

The intrinsic extensibility of XML combined with ontological engineering practices provides the Service Container with a loosely coupled data model. This loosely couple, data model is a key aspect of the SOIN with significant benefits for development, deployment, and maintenance of the integration infrastructure. Because of the extensibility of the ontologies, producers and consumers of XML messages can be upgraded independently of one another to deal with extended data structures.

Business Processes, Content-Based Routing & Transformation

Through a process sequence of inspection, routing and transformation combined with the use of XML documents and associated ontologies, the Service Container can handle a wide array of complex integration tasks. This approach adds a great deal of resilience to changing data in the underlying applications and enables us to think about formats, meaning, routes, and transformations seperately from the underlying applications.

A business process definition represents a series of steps in a flow between applications. These steps are mapped into service endpoints that represent physical applications. The business process definition makes it possible to insert additional steps into that path of control flow, without modifying the participating applications directly. Thus, specialized services (steps) that can alter the path or otherwise affect the content of the message can be inserted. The path of execution, or message flow, between the producers and consumers of the message/data can be augmented by adding a transformation service into the business process.

Content-based routing (CBR) services can be plugged into the message flow between producers and consumers. The CBR service is a lightweight process with the sole purpose of applying an Xpath expression to determine whether the message conforms to a specified format. The message can then be routed automatically to another specialized service or a waiting consumer application.

Transformation services implemented as XSLT can restructure XML documents from one format to another, as well as, transforming or enhancing the content in some manner.

This portfolio of specialized services within the Service Container provides the facility to improve the message data without ever modifying the sending applications. Only consuming applications need to be modified to support the improvements in data. Not all consumers of the data need be upgradeed either. For the ones that do, upgrades can happen independently of the data translation upgrades, allowing greater flexibility in the coordination of development timelines of the applications that are driving the requirement.

A Generic Data Exchange Architecture

To create a generic data exchange architecture supported by the Service Container, a canonical, application-independent domain model (ontology) is created. This canonical ontology provide a ‘pivot’ point representation for expressing data in messages as it flows through the Service Container. In this manner, ontologies representing the information models of each individual application can be mapped into and out of the ‘pivot’ ontology or canonical format representation.

By having independent representation of a ‘pivot’ ontology, all application specific work can proceed in parallel without requiring advanced agreement between them. This is an important characteristic of the Service Container given that most large scale enteprise or government organizations must support multiple disparate data sources and applications, across which, it is impossible to get agreement on a single domain model or ontology.

When extensions to the canonical ontology are required, they can simply be added as extra XML/RDF/OWL attributes and elements, relying on the inherent flexibility of the Service Container/XML approach. As applications change, the impact is limited to the transformation into and out of the canonical ontology. Application or service

providers are free to enhance their applications and take advantage of new information available from the canonical 'pivot' ontology.

To complete the implementation of the generic data exchange architecture witin the Service Container, transformation services are created to convert data to and from the canonical ontology and the target data format of the application being plugged in.

The following steps occur as a message (XML document) flows through the Service Container using the canonical ontology-based data exchange mechanism. Each step is treated as an event-driven service that receives a message asynchronously using reliable messaging, and forwards the resulting message on to the next step after processing:

1. An external service provider sends an XML or SOAP message over the HTTP protocol. Once inside the firewall, the message is assigned to a business process that controls the steps through the Service Container.

2. The message is run through a transformation service which converts the XML content from the data model used by the service provider to the data model represented in the canonical ontology.

3. A splitter service makes a copy of the message and routes it to an auditing service. The audit service may add additional information to the message, such as contextual information about the business process and a timestamp of when the message arrived.

4. The audit service itself is implemented as a native XML persistence engine that allows the direct storage and retrieval of XML documents.

5. Before the original message is forwarded on to a destination application that is part of the defined business process or may have subscribed to receive any new messages of this type via the publish/subscribe capability of JMS, the need for any further transformations, as required by the destination application, is determined. If required, a second transformation service is invoked such that the message, as expressed in the format of the canonical ontology representation, is converted to a format that the destination application recognizes.

6. The destination application receives the message and operates on it; that is, it consumes the message and processes it.

7. As part of the business process, the message now needs to be forwarded to another destination application that only recognizes its own proprietary format. Before proceeding to this new destination application, the message is routed to a third transformation service that converts the message into the proprietary format of the next destination application.

8. The transformation service sends the message on to the next destination application/service.

9. After being consumed and processed by this destination application, the message now needs to be routed to another business process. This other business process can be invoked in a variety of ways from a number of places within the Service Container.

10. A CBR service is inserted between the two business processes to examine the the message content and determine if the message needs to be transformed before being forwarded to the next process.

11. Assuming the message must be transformed yet again, the appropriate transformation service is invoked to convert the message prior to its being sent to the consuming business process.

This generic data exchange architecture directly addresses the *N-squared* problem represented by point-to-point (application-to-application) transformations between each application/service. When using specific point-to-point

transformations between each application, the number of transformation service instances increases exponentially with the number of applications. With the use of a canonical ontology, the number of transformations increases more linearly as new applications are brought into the SOIN.

Further benefits of this approach include:

- Each application/service needs to focus on only one type of transformation to and from a common format. Individual application/service owners are only concerned that you plug into the Service Container, post to the Service Container and receive data from the Service Container. The Container is responsible for getting the data where it needs to go, in the target formats that it needs to be in, and using the protocols, adapters and transformations necessary to get it there.

- New applications/services being written to plug into the Service Container can use the canonical ontology format directly. Multiple applications not anticipated today can tap into the flow of messages through the Service Container to rapidly create heretofore unimagined capabilities.

- Container services such as the CBR service and splitter service can be written to use the canonical ontology format. This allows for a service to be written once and then customized on a per-instance basis by supplying different XSLT stylesheets, endpoint definitions, and routing rules.

- Standard stylesheet templates and libraries can be created and reused throughout the organization.

Chula Vista CommandRESPONSE Services

For the Chula Vista CommandRESPONSE Project, the initial portfolios of applications 'plugged' into the DC2B were as follows:

1. **Multi-protocol secure wireless communication system** – TECHALT

   This system enables the establishment of a secure wireless LAN at the incident location between multiple local, state and federal agencies. The LAN can be established across any of several communciation protocols including 802.11x, 1xRTT, EVDO, Imarsat, CDPD, GSM, iDEN and land mobile radios. Session connectivity is maintained across communication protocols in the event that a participant must withdraw from the LAN and move beyond the incident location.

2. **HAZMAT Shipment Tracking System** – QUALCOMM

   Transponders affixed to transport vehicles emit positional information across a satellite feed every 2 minutes. All commercial carriers of munitions and hazardous waste material are mandated to support this capability.

3. **Protect America: Event Correlation System** – CellExchange

   The Protect America Event Correlator scores and correlates the event flows into the DIS. The scoring system is configured by the user community. The system looks for precursor signals, ambiguities, etc. that rise above defined thresholds. These signals are presented to the user through a dashboard display in the DCOP. Users can drill down to look at specific events contributing to any dashboard signal. The event correlator can drive alerts/notifications as circumstances dictate.

4. **Consequence Assessment Tool Suite** (CATS) – SAIC

   A decision support system that provides a portfolio of simulations including fire, bomb detonation, urban area plume dispersion, etc. taking into account context such as current weather conditions, etc.

5. **Computer-Assisted Dispatch** (Police/Fire CAD) - CrossCurrent

Chula Vista police and fire departments share a commone CAD system to handle all of their 911 traffic. Calls come into the dispatch center, are logged into the CAD system and resources deployed to respond as appropriate.

6. **Geographic Information System** (GIS) – ESRI

The City of Chula Vista has made a significant investment over the past seven years in an ESRI geographic information system. The system consists of software, hardware and data to capture, store, check, integrate, manipulate, analyze and present information that is tied to a spatial location.

7. Border Patrol Incident Management System –

The Border Patrol incident management system provides the facility to capture, store, check, analyze and report incident location, severity, impact, estimated duration. It also provides for post-incident analysis.

An Architecture Aimed at Enriching the Web Semantic Data

CommandResponse [1]is developed and deployed using all open source software. The backbone of CommandResponse is the Keel Meta-Framework. Keel is an open source Java project that integrates platform independent services with enterprise applications. For CommandResponse, Keel integrates the Apache Axis web service implementation of SOAP ("Simple Object Access Protocol"), the Cocoon web development framework, the Jakarta Tomcat servlet container, the OpenJMS implementation of the Java Message Service (JMS) specification, and the Sesame RDF (Resource Description Framework) database.

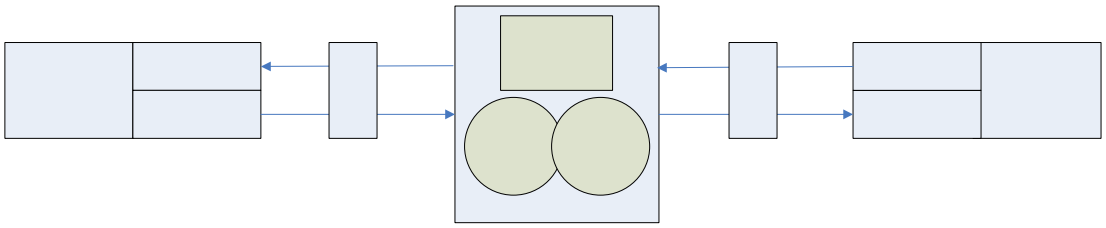

Figure 1: Ontology Mapping

In this scenario, mapping rules are used to define what a term in one ontology means in another ontology. The mediator uses these rules at runtime so that applications can access each other's data. This approach has the advantage of not requiring the application developers to explicitly agree on a shared ontology. The Mapping/Mediator maps operational data expressed in the terminology of on ontology into operational data expressed in the other ontology. Translators are used to reformat data to each subscriber's schemas.

Although the current framework is built for Homeland Security, the possibilities of applications are endless.

[1] See http://xml.antin.com/press/commandResponse/

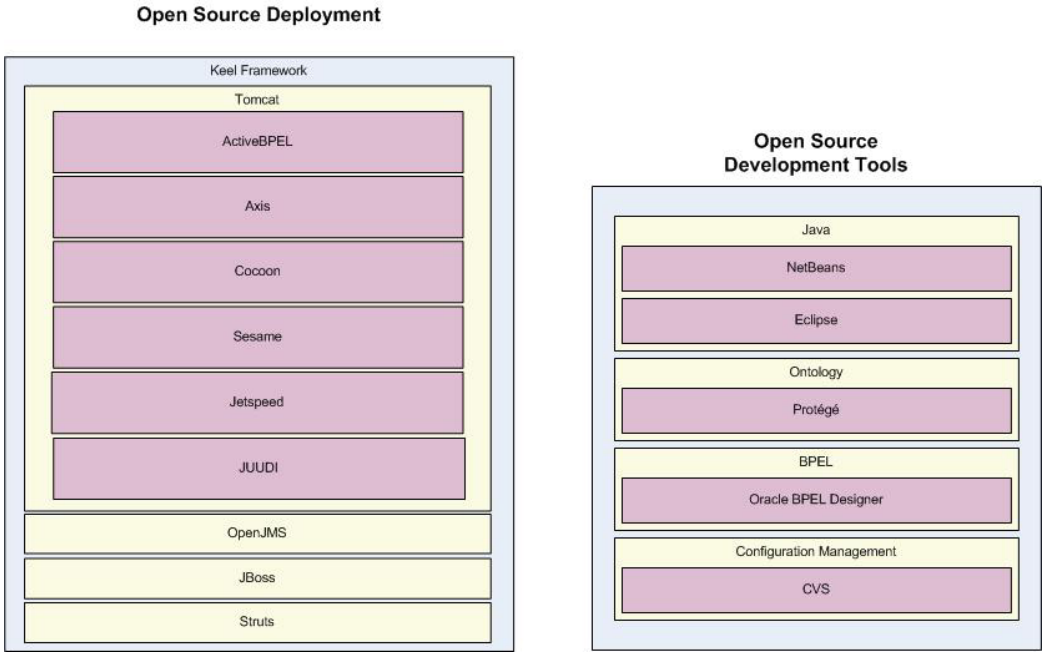

Figure 2: List of Open Source tools

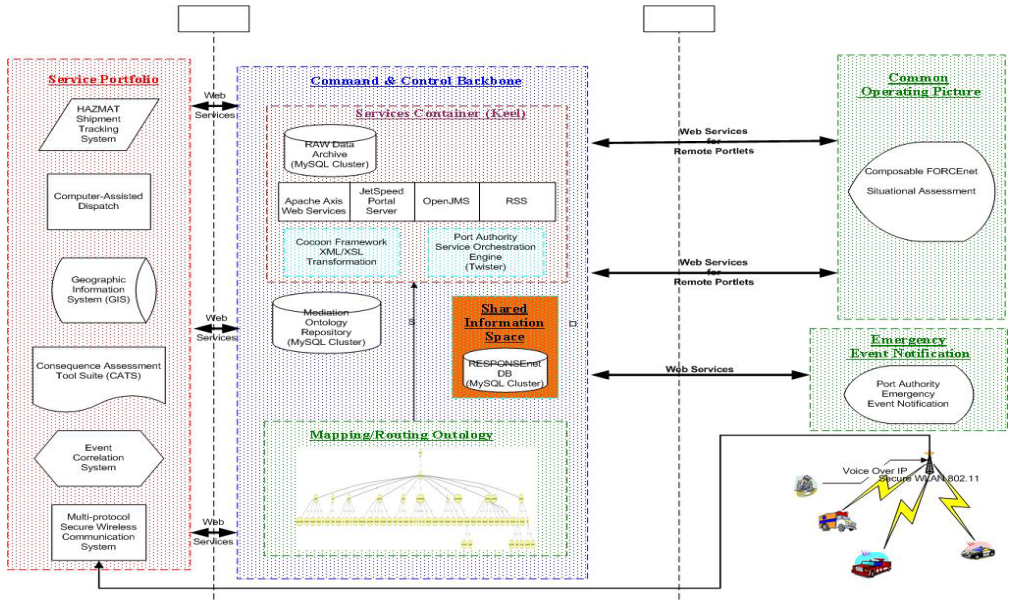

Figure3: Pub/Sub Mediation Framework

1. Axis

Axis[2] is an open source implementation of the W3C (World Wide Web Consortium) specifications for SOAP.

From the draft W3C specification:

> SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses.

For CommandResponse, Axis provides a WSDL (Web Service Definition Language) interface through which vendor clients (e.g., CAD, event correlation, panic button alert) can send their messages. There is no schema change for clients connecting to CommandResponse, as client specific message formats are translated and sent through the system.

Installation and deployment of Axis is much easier using Keel's customizable installation script. After Axis is deployed, developers can update the build by using a build script, created by Keel, which creates a WAR (Web Archive) file. The Axis WAR file can then be copied to the web server of choice (Tomcat in this instance) for instant deployment.

2. Cocoon

Apache Cocoon [3] is a framework built around the concept of separating components of web development. Cocoon uses "pipelines" to connect different components, with each component specializing in a specific operation.

CommandResponse uses Cocoon to transform message elements from one schema to another. When a message is published to CommandResponse, Cocoon creates a message containing the data received into subscriber-specific schemas, thus requiring no schema changes from publishers or subscribers.

Like Axis, Cocoon is installed and deployed through Keel. Cocoon is also built through Keel and a WAR file containing Cocoon's web application can be copied to any web server.

3. Tomcat

Apache Tomcat is a container for Java Servlets and Java ServerPages (JSPs), the presentation layers of the Java platform.

Tomcat allows clients who have been given a vendor identification and password to publish to CommandResponse. The WSDLs for CommandResponse's webservice is also available for view from Tomcat.

Tomcat is installed and deployed through Keel and is highly configurable. Tomcat contains the root directory of the presentation layer, accessible to clients through port 80. Web applications are copied into Tomcat's "webapps" directory and are then deployed automatically.

[2] See http://ws.apache.org/axis/
[3] See http://cocoon.apache.org/

4. Sesame

Sesame[4] is an open-source RDF[5] database which features support for querying and RDF inferencing. Sesame comes with APIs (Application Programming Interfaces) for easy querying of the database.

Sesame is used to store the CommandResponse ontological data. Messages sent to CommandResponse are written to Sesame. Queries return data elements that will be inserted into an outgoing message to a subscriber.

Sesame is deployed and installed through Keel and a WAR file is built through a Keel build script. The WAR file can then be copied as a web application to a web server.

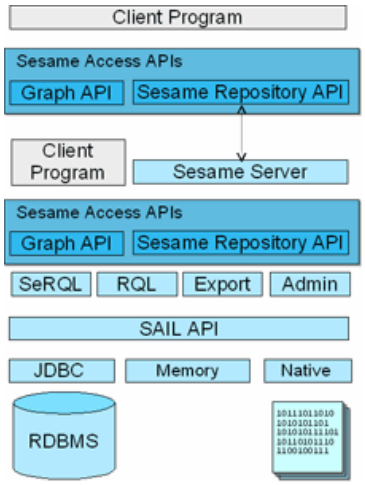

Figure 4: Sesame Architecture

5. ActiveBPEL

The ActiveBPEL[6] engine executes Business Process Execution Language (BPEL) processes. It accepts BPEL process definitions, creates process instances, and executes them. The ActiveBPEL™ engine is a commercial-grade open source implementation of the Business Process Execution Language for Web Services Version 1.1 specification, and is fully compliant with that spec.

BPEL is an XML language for describing business process behavior based on Web services. The BPEL notation includes flow control, variables, concurrent execution, input and output, transaction scoping/compensation, and error handling.
BPEL is layered on top of other Web technologies such as WSDL 1.1, XML Schema 1.0, XPath 1.0, and WS Addressing.

[4] See http://www.openrdf.org
[5] See http://www.w3.org/RDF/
[6] See http://www.activebpel.org

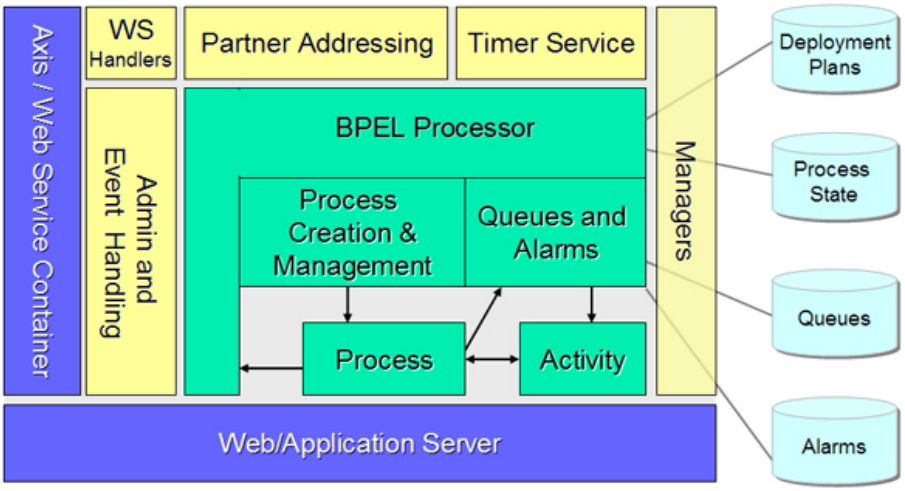

Figure 5: Engine Architecture

6. Framework Summary

Ontologies are used to improve communication between either humans or computers. This consist of assistant in communications between human system integrators (HSI), achieving interoperability and to improve the process and quality of engineering software systems.

The framework is built on:

- Open standards: SOAP, JMS, RDF
- Open source
- Vendor agnostic
- Inter-Operability between computer systems achieved by translating between different modeling methods, paradigms, languages and software tools. The ontology is used as an interchange format.
- Search: An ontology may be used as a standard for indexing into a repository for information.
- Specification: The ontology can assist the process of identifying requirements and defining a specification for an IT system (knowledge based)
- Maintenance: Use of ontologies in system development can improve system maintenance if it is used as a neutral authoring language with multiple target languages. It only has to be maintained in one place.
- Knowledge Acquisition: Speed and reliability may be increased by using an existing ontology as the starting point and basis for building new knowledge based systems.

Conclusions

One of the principal goals of the CommandRESPONSE Platform is to improve the affordability and flexibility of future domestic C4ISR information systems for local, state and federal agencies. The CommandRESPONSE Platform providing core information sharing and command&control backbone capabilities will be offered as open source software and available to government agencies at **low/no cost** and **without vendor lock-in**. Service, maintenance and new releases for the platform will be made available through a professional open source business model. Public safety, emergency preparedness/response and homeland security applications (CommandRESPONSE-compliant) to be plugged into the CommandRESPONSE Platform as services will be unique to the needs of the specific local, state and federal organization(s) and likely be a combination of purchased COTS or GOTS products and services. New technology insertion is enabled without massive integration or restructuring costs.

Other principal goals of the CommandRESPONSE Platform are to (1) foster the interoperation of disparate legacy information systems in ways that could not have been predicted when these systems were first designed, (2) enable Inter-Agency coordinators, first responders and other participants to customize and tune their own information architecture along with their resource structure (personnel deployments) to match any given operational environment, (3) establish interoperable communications among deployed first responders and participating agencies irrespective of existing communication equipment and (4) the distribution of key information to commanders and personnel in the field.