J Intell Robot Syst (2013) 69:41–55
DOI 10.1007/s10846-012-9691-8

A Reference Software Architecture to Support Unmanned Aircraft Integration in the National Airspace System

Curtis W. Heisey · Adam G. Hendrickson · Barbara J. Chludzinski · Rodney E. Cole · Mark Ford · Larry Herbek · Magnus Ljungberg · Zakir Magdum · D. Marquis · Alexander Mezhirov · John L. Pennell · Ted A. Roe · Andrew J. Weinert

Received: 13 June 2012 / Accepted: 4 July 2012 / Published online: 2 August 2012

Abstract This paper outlines an architecture that provides data and software services to enable a set of Unmanned Aircraft (UA) platforms to operate in a wide range of air domains which may include terminal, en route, oceanic and tactical. The architecture allows a collection of command, control, situational awareness, conflict detection and avoidance, and data management elements to be composed in order to meet different requirement sets as defined by specific UA platforms, users, and operating regimes. The architecture discussed is based on a Service Oriented Architecture (SOA) with open standards on the interfaces between elements. Services may include common situational awareness, sense and avoid, weather, data management and flight plan information. Service contracts specify quality of service, interface specifications, service description metadata, security provisions, and governance. Pieces of the architecture have been implemented by MIT Lincoln Laboratory in the form of a Sense and Avoid (SAA) testbed that provides some of the core services. This paper describes the general architecture and a SAA testbed implementation that begins to realize that architecture and quantifies the benefits. The proposed architecture is not directed at a specific program but is intended to provide guidance and offer architectural best practices.

Keywords Unmanned Air vehicle · Sense and Avoid · Service Oriented Architecture · Testbed · Reference architecture

This work is sponsored by U.S. Army, PM-UAS, Product Directorate - Unmanned Systems Airspace Integration Concepts under Air Force Contract FA-8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government. Approved for public release, distribution is unlimited.

C. W. Heisey (✉) · B. J. Chludzinski · R. E. Cole · M. Ford · M. Ljungberg · Z. Magdum · D. Marquis · A. Mezhirov · T. A. Roe · A. J. Weinert
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420, USA
e-mail: heisey@ll.mit.edu

A. G. Hendrickson · L. Herbek · J. L. Pennell
Department of the Army, Project Manager's Office: Unmanned Aircraft Systems (PM UAS), Unmanned Systems Airspace Integration Concepts (USAIC), Redstone Arsenal, AL, USA

A. G. Hendrickson
e-mail: Adam.Hendrickson@PeoAvn.Army.Mil

1 Introduction

The combination of a rapid rise in the number of heterogeneous Unmanned Aircraft (UA) and their use in both military and homeland protection roles has resulted in a need to have UA operate

within the National Airspace System (NAS). Application of Service Oriented Architecture (SOA) to the control of UAs is a risk lowering path to ensuring technical interoperability, manageable governance, acceptable safety and security, and future extensibility. This paper discusses the application of SOAs to Unmanned Aircraft Systems (UAS) and the necessary components/processes to realize that capability.

The principal required UAS support functions provide for (a) command and control of the UAs including their payloads, (b) separation assurance to deconflict airspace with respect to other aircraft, and (c) payload data dissemination. UA separation presents a number of challenges because the operator may not have the same awareness as a pilot in a manned aircraft. Technical solutions require low latency data delivery with high reliability. The SOA discussed herein accommodates different mixes of surveillance sources, multiple UA platform types, multiple user types, and different applications and air domains.

This paper is divided as follows: Section 2 provides background. Section 3 details the reference architecture design. Section 4 describes the services in detail. Section 5 describes the Sense and Avoid (SAA) testbed implementation of core, mediation, and domain-specific services. Section 6 summarizes the evaluation of the SAA prototype testbed performance. Section 7 draws conclusions and describes future work. The contribution of this work is to outline a unique set of technology choices for implementing SOA in a high-performance, low-latency, safety critical problem domain.

2 Background

In order to accommodate UAS in the NAS the foremost challenge is to provide SAA capability. An excellent way to provide SAA in the near-term is by leveraging ground based surveillance. This is called Ground Based Sense and Avoid (GBSAA), and the US Army is using this approach. Airborne Based Sense and Avoid (ABSAA) is a solution in which SAA sensors are located on board the UA. In the long term, the SAA solution will likely involve a hybrid of GBSAA and ABSAA [1].

In addition to SAA, several other components, such as flight plan information, weather, and command and control are needed to integrate UAS in the NAS. These information components can be delivered as SOA services. For example, FAA SWIM has implemented a SOA with a registry and security mechanism to provide weather, flight plan and aeronautical information. SWIM plans to provide surveillance, 4D trajectory and other services [2]. The Net-Centric Enterprise Solutions for Interoperability (NESI) provides guidance and best practices for SOA design [61]. The UAS Control Segment (UCS) architecture [3] is developing standards and architecture for ground stations, which this effort will leverage. At the moment there is a broad collection of radars, an assortment of ground control stations, and few standards exist. The combination of these factors limits interoperability. We are taking the approach of a SOA reference architecture to knit all of this together.

A SOA presents a number of features that ease UA operations and NAS technical integration:

- Providing services to ingest a mix of data sources and to offer a consistent view of that data across multiple disparate consumers via governance by contracts that specify interface, protocol, quality of service and security.
- Allowing service elements to evolve somewhat independently, subject to integration testing and re-certification.
- Facilitating interoperability between organizations in a loosely coupled fashion, subject to mandate and authorization to connect.
- Enabling unanticipated users. Consumers can subscribe to services they are authorized to receive as needed and not have any knowledge of or impact on other users.

The goals of an open SOA are a set of "ilities": modularity and composability, scalability and extendibility, configurability and adaptability, interoperability and reusability, and upgradeability and maintainability [4–6].

Lincoln Laboratory has implemented projects involving pre-SOA [7] and SOA for near real-time applications. These include the Radar Open System Architecture (ROSA) [8], Corridor Integrated Weather System (CIWS) [9], Tower Flight

Data Manager (TDFM) [10], Extended Space Sensors Architecture [11], and Lincoln Distributed Disaster Response System (LDDRS) [12]. Some other non-Lincoln Laboratory real-time SOA efforts are presented in [13–15] that go into detail on middleware and real-time scheduling.

This paper describes an architecture that accommodates the diversity in platforms and missions presented by UA. The architecture supports multiple vendor products, data sources (publishers) and data sinks (subscribers). Open source elements are included, which allow "best of breed" solutions as well. UA diversity needs led us to choose an Open Standards Architecture (OSA) for vendor neutrality and SOA for modular elements. Features also include include Multi-Level Security (MLS) options to partition data horizontally between organizations and vertically within an agency. Because the SOA is software based, flow of data can be tailored on the basis of need, classification and volume restrictions. To vet the design, we have implemented key components in a SOA testbed.

3 Software Architecture Design

The Lincoln Laboratory development team chose to adopt a SOA approach for the reference architecture. This will inform and guide GBSAA architecture options for the Army. A reference architecture provides a template solution that guides specific implementations [16]. A service is defined to have three characteristics [17].

- A service is a self-contained set of functionalities with well defined interfaces.
- Services are loosely coupled, meaning dependencies between subsystems are minimized and interfaces are well-defined and evolve iteratively.
- Services are interoperable. A set of services is associated with a service bus which is an infrastructure that enables interoperability between distributed system elements. The service bus contains a message bus as well as core services.

For this effort, services are standards-based and open. We interpret a "service" to include both request/response services and streaming data-feeds. We do not restrict SOA to Simple Object Access Protocol (SOAP)/Web Services Description Language (WSDL) but incorporate protocols relevant to high-throughput event driven real-time systems.

Service providers register their services in a service registry. This registry includes metadata about the content of the service. A consumer searches the registry to discover the service of interest. A key concept of SOA is the service contract, which is a formal specification of the service interface, quality of service, and governance. The registry returns the service endpoint of the provider. The consumer can then bind to the service through a specified protocol.

Figure 1 shows a notional diagram of the services the reference architecture needs to support. The basic architecture incorporates various sensor data sources publishing data to a bus and consumers subscribing to the sources. A few services such as the SAA service subscribe to data and produce their own output data products. Some services are request response as well.

These services are organized into a layered design according to performance requirement. Layers range from real-time (tactical services), to near real-time (mediators, decision support tools, federated services) to internet time (business, strategic services). The latter are not considered here. At the real-time layer are streaming surveillance data providers and legacy systems. The real-time layer includes a tactical message bus to provide low latency high-throughput surveillance data to a correlator-tracker and decision support tools. The near real-time layer is the enterprise service bus that is exposed for external SOA consumers.

This architecture utilizes two message buses to provide message delivery and quality of service for both tactical (real-time) and enterprise (near real-time) domains, reflecting the different performance requirements of each. Tactical services such as track correlation and camera queuing require real-time latencies of several milliseconds or less. Tactical messages are delivered in binary format over User Datagram Protocol (UDP). The Enterprise Service Bus (ESB) provides messages with latencies of tens of milliseconds. ESB

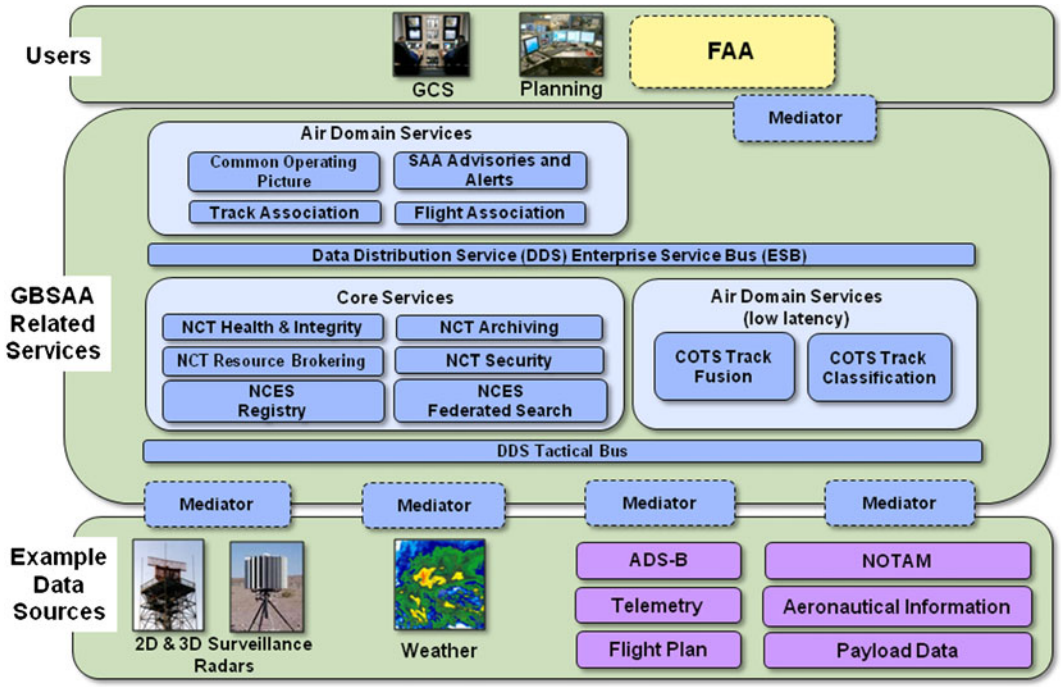


Fig. 1 Notional view of possible services in the reference architecture, subject to interoperablity requirements, authorization, and certification

messages are delivered in Extensible Markup Language (XML) or in a binary format. In the reference architecture, data models are independent of implementation, and bindings to XML, Interface Definition Language (IDL), or JavaScript Object Notation (JSON) [18] are possible, for example. Data Distribution Service (DDS) over Ethernet can meet the requirements of both buses by using different Quality of Service (QoS) settings.

Table 1 shows the open standards used in the reference architecture. Existing open standards are used where possible. Binary formats including IDL are used for tactical messages. IDL and XML are used for ESB messages. XML provides a self-describing, extensible message format, but IDL is a more efficient for data movement. Metadata in the message facilitates ease of integration and promotes interoperability. Open standards can prevent vendor lock in and can lower lifecycle costs [4, 5]. A Navy program has successfully created an open architecture acquisition process [19] and-has created a set of metrics to evaluate openness of a software system [20]. Messages can also be represented in multiple formats simultaneously, for example XML and IDL, as we recognize that no single format is going to meet all customer requirements, although the latter is preferred.

To manage the service implementation lifecycle, we propose that a governance board will be responsible for provisioning and maintenance of services. A process will be established for the full service lifecycle from specification to implementation. New services will be specified by a service specification template. The template captures service name, description, QoS requirements, operation signatures and data models. This service metadata can be placed in a registry. The board can certify services for standards compliance, performance, and security. The proposed board will deploy the services and designate a party responsible for hosting the service and policies for use and maintenance. The board will be responsible for enforcing Service LevelAgreements (SLA)s that guarantee QoS. The governance board could establish standards committees for necessary standards that do not currently exist. The board would contain members of interested stakeholders and would ideally be chaired by a neutral party. National Informa-

Table 1 DODAF TV-1 partial list of open messaging standards used in the reference architecture

Service area	Technical services	Standards	Standard description
Core	Command and control, Resource tasking	UCI, OGS SPS	Unmanned Aerial Systems Command and Control Initiative [21], Open Geospatial Consortium Sensor Planning Service [22]
	Algorithm advisory	New standard	A proposed standard
	Track internal	Army GBSAA	A comprehensive track schema defined in IDL to be replaced by evolving standards
	GCS internal	UCS	UAS Control Segment (UCS) Architecture [23]
	Track consumer	CoT	Cursor on Target [24]
	Track consumer	CAT48 Consumer	ASTERIX CAT48 [25]
	Flight plan	Flight Object XML	Forthcoming standard from FOWG [26] or EUROCONTROL
	Telemetry	STANAG 4586	NATO standard [27]
	Weather	WXXM	Weather Information Exchange Model [28]
	Aeronautical information	AIXM	Aeronautical Information Exchange Model [29]
	Health and integrity	CIM	Common Information Model [30]
Modeling and simulation	Simulated aircraft	DIS	IEEE Std 1278.2-1995IEEE Distributed Interactive System

tion Exchange Model (NIEM) [31] and OpenMPI [32] are examples of standards governed in this manner.

System creators of production deployments may need to be comply to security certification using National Institute of Standards and Technology (NIST), National Information Assurance Certification and Accreditation Process (NIACAP), or Defense Information Systems Certification and Accreditation (DISCAP) methodologies.

SOA systems present a number of safety certification challenges because of the potentially dynamic nature of services, and existing processes will need to evolve. An overview of a process for certification and accreditation of software systems is given in [33]. The FAA has proposed additional software processes for safety critical-systems [34]. Issues and possible ways forward for certification of SOA systems are discussed in [35–37]. Department of Defense Architecture Framework (DODAF) is used to document many DoD software programs [38], and we suggest augmentation with additional SOA artifacts and a service specification template by the Everware-CBDI framework [39–41], which serves as the basis for a template in use by DHS [42]. Architects of safety critical systems are increasingly using Architecture Analysis and Design Language (AADL) to capture the architecture of safety critical systems [43].

Discussions on process modifications for SOA are ongoing [36, 37]. The specification of a service using a service specification template is essential. The service specification template details interface, quality of service contracts, governance policy, priority, and other service details. Service deployments can evolve independently of the architecture once the template is specified because a replacement element could be certified against the conditions in the template.

4 Description of SOA Components

Services fall into several categories. The fabric of the SOA, the general infrastructure, includes security, registries, search, messaging, and archival services. These core services are shared by many Lincoln Laboratory SOA projects. On the other end of the spectrum are domain-specific services, which are shared between projects within a domain. Examples of services unique to air domain include track association, flight plan association

and situation displays. Additionally, there are services that mediate between different formats as well as different protocols. Section 4.1 describes core services. Section 4.2 describes mediation services, and Section 4.3 describes domain services in the reference architecture.

4.1 Core Services and the Net-Centric Toolkit

The Net-Centric Toolkit (NCT) [44] is a collection of interoperable services intended to support net centric systems.

4.1.1 High-Performance Tactical Messaging Bus

The reference architecture employs a high-performance messaging middleware, which utilizes a publish/ subscribe pattern that allows decoupling of producers and consumers. The NCT Data Transfer Service (DTS) provides an abstraction layer separating the messaging implementation from the interface. DTS contains both a data plane (data content) and a control plane (e.g., configuration, management, status, and discovery). The DTS is based on Apache Camel, and the middleware protocol can be switched by changing a text adaption file. The SOA uses Extensible Messaging and Presence Protocol (XMPP), Advanced Message Queuing Protocol (AMQP), Java Messaging Service (JMS), Data Distribution Service (DDS), and Service Mix, as well as UDP, and can support multiple protocols simultaneously.

4.1.2 Registration Service

The reference architecture includes registries where services and datafeeds can be registered and discovered. The registry describes the endpoint where the service resides, data formats, schemas, metadata description, quality of service delivery, and can provide an Interface Control Document (ICD). The Net-Centric Enterprise Services (NCES) [45] service registration is based on standard Universal Description Discovery and Integration (UDDI) with additions for Representational State Transfer (REST)-based services [46].

Datafeeds can also be registered. MIT Lincoln Laboratory is developing a semantics-based registry suitable for datafeeds. The registry provides information on formats of data feeds and an ontology that gives the definitions of the metadata items. The reference architecture provides a semantic ontology for encoding metadata in web service description language such as Web Ontology Language (OWL). The SPARQL Protocol and RDF Query Language (SPARQL) query language [47, 48] is used for matching and lookup of datafeeds.

4.1.3 Resource Brokering Framework

The resource broker is a collection of services that decouple information consumers from information providers. The broker performs a dynamic composition of information resources into processing chains based on a user's request for data [49, 50]. Reasoning over the semantic descriptions of resources enables this composition. These descriptions include annotations on the inputs, outputs, capabilities, and characteristics of a resource. The term "resource" is an abstract concept and may represent a sensor, algorithm, or even a human analyst. Each resource provides a standard management interface for tasking. The tasking interface is based on Open Geospatial Consortium (OGC) Sensor Planning Service (SPS) [22].

4.1.4 Federated Search Service

The federated search service is a part of the NCES model. This service enables system wide queries with good performance characteristics. Archived data needs to be tagged with open standard metadata such as DoD Discovery Metadata Specification (DDMS) with extensions [51]. Each searchable repository registers a search provider that will search and return the metadata of items that match a search request.

4.1.5 Security

The ability to identify individuals is fundamental to a security posture. Certificates in a Public Key Infrastructure (PKI) are widely used for this. Once subjects are authenticated, they can be allowed to access information resources. There are

two dominant methods used today, Role Based Access Control (RBAC) and Attribute-Based Access Control (ABAC.) In RBAC, subjects are allowed access if they belong to a pre-defined group of users. ABAC, on the other hand, allows access if the subject possesses a particular attribute, e.g., clearance level. A security posture needs to be restrictive enough to prevent unauthorized use, yet exhibit some flexibility to allow access to authorized users. In an effective net-centric environment, access is allowed to the unanticipated user for the unanticipated use, which in this case, could be an additional GCS.

Cross-Domain Security (CDS) systems refer to the hardware and software that transfers data between different classification domains. In some cases these CDS systems are one-way in nature, allowing data to only stream in one direction, typically from a lower classification level to a higher level.

Bi-directional CDS systems can also be used when moving data from higher to lower levels of classification where the data needs to be inspected to ensure that higher classified data is not let through. There are a variety of approaches for deep message inspection, including manual inspection of data objects like images, rule-based text matching, and security markup on the message. A commonly used markup scheme for XML data is IC-ISM [31]. Although SOA provides mechanisms for data transfer between domains and organizations, specific implementations and policy would have to be considered on a case by case basis.

4.1.6 Health and Integrity Framework and Auditing

The health and integrity framework provides a set of services and adapters to enable monitoring of the health and status of all components within the architecture. The framework consists of a common status bus on which service components publish their availability, performance metrics, and management endpoint information. The format of these messages is defined by the Common Information Model (CIM) [30]. If services do not support this standard, their existing status messages can be mapped to the CIM schema using an adapter. If a service component does not provide any status messages it is still possible to provide a message to advertise the components availability by hooking into its lifecycle events through its container.

The health and integrity messages published include availability, heartbeats, performance metrics, inventory descriptions, prognostics, and alerts. Different consumers such as an alerting mechanism, archive, or user display can process these messages. In addition to alerts provided by service components, custom agents can be deployed to monitor components or processing chains and provide alerts when service level agreements are in jeopardy. Additionally, a traditional monitoring tool such as Hyperic or OpenNMS can optionally be bundled as an agent on the system to provide a visual monitoring and management interface.

Audit services are provided for security monitoring and troubleshooting. All services can log start-up and shutdown, access, and configuration changes to a central auditing facility. In addition, the audit service consumes health and integrity messages from a status topic on the ESB. Certification and Authorization standards such as NIST 800-53 provide a complete set of auditing requirements.

4.1.7 Archive/Playback Service

The archive service on the tactical bus provides a high data rate store of raw camera and track data. Both buses (tactical and ESB) will store processed data products, such as alerts, decimated track data, video snippets, and snapshots. This will allow event reconstruction and playback.

The archive/playback service can record large volumes of messages and faithfully play them back as they were recorded in terms of content as well as time delay among them. The service allows one to group multiple topics and tag the recording scenario with metadata. The archive/playback service provides functionality to select messages for playback on the basis of a time window and provides filtering by topics. Also, users can playback messages at slower or faster rate than the rate they were recorded. The service provides functionality to transform messages during playback

to alter timestamps and other data as needed. The service can receive and send messages using multiple protocols such as Joint User Messaging (JUM), ActiveMQ, Transmission Control Protocol/Internet Protocol (TCP/IP) and many others. Messages recorded using one protocol can be played back with different protocol. The service provides functionality to schedule recording or playback, and users can create different recording scenarios on a daily basis. One can import and export messages from the service. Also, users can validate messages against a schema. Additionally, there is some functionality to view and analyze message content.

4.1.8 Composable Applications

In addition to composing services into processing chains via the resource broker, services can be explicitly composed using a Graphical User Interface (GUI) tool. Services can be encapsulated by plug-ins that enable them to be discovered and connected dynamically to create an application. In some cases it may be possible to expedite the development process and enable domain specialists to directly create solutions. Plug-ins encapsulate connectivity rules governing how heterogeneous service endpoints can be connected and contain metadata and semantic information to locate services. Compositions in turn can be registered as new services and reused. We have prototyped an initial capability using the Kepler Tool framework [52, 53].

4.2 SOA Sidecars and Mediation Services

MIT Lincoln Laboratory has for many years built small computer systems known as sidecars that can be used to connect sensors (initially) to a network to allow experimental algorithms access to stove-piped sensor data in real time. The idea is to attach a sidecar "online but not inline" to not disturb the host or legacy system.

As time and community acceptance of this technique has advanced, bidirectional sidecars are now being developed, allowing data to flow from the legacy system to the sidecar and tasking to flow from the sidecar to the legacy system.

A sidecar has two principal internal components, the Sensor Adapter and the Network Adapter. The Sensor Adaptor provides the interface to the legacy or proprietary system that encapsulates the proprietary format and contains a format converter. The Network Adapter outputs the common data format and provides the interface to the SOA bus. The sidecar does any necessary protocol conversion as well. Insulating proprietary formats allows independent development of SOA components by different vendors, while allowing a vendor to retain its intellectual property and competitive value.

An example of a sidecar is a service component that wraps a proprietary system element or legacy sensor. The wrapper mediates interactions between the element and the larger system. The purpose of mediation is to facilitate a plug-and-play capability for that element allowing diversity and multiplicity. Internal mediation components that perform information transformations and functionality supplementation will be built, as needed. The sidecar publishes messages on the ESB in XML. The schemas are a combination of DoD and Open Geospatial Consortium OGC based open and open standard extensions if necessary. On the ESB side, the same schemas will be used. The published update rate can be optionally decimated.

The Net-Centric Mediator Framework provides a framework on which new mediators can be quickly deployed. Rapid creation of new mediators is possible since the developer can focus on the actual transformation or processing chain while leveraging the application lifecycle, management, and monitoring capabilities of the framework. The framework provides a container for one or more processing chains that consume and transform message payloads. The framework also provides for the management of mediators through standard Java Management Extensions (JMX) interfaces. A monitoring Application Programming Interface (API) provides for the gathering and reporting of performance related metrics such as number of messages processed, mean transformation times, and other statistical metrics such as standard deviation. The mediation framework also supports more complex behaviors such as

message enrichment or decimation through additional configuration in the processing chain.

4.3 Services Specific to Air Domain

4.3.1 Track Fusion Service and Track Classification Service

The track fusion service uses a correlator-tracker to fuse reports from multiple radars into a single air picture. An adjunct classification service discriminates between aircraft and non-aircraft. Both are placed in the core services level to achieve low latency, which is necessary for queuing of cameras.

4.3.2 Flight Association Service

The flight plan sidecar takes FAA flight plans in Common Message Format (CMS) or other format and converts them into an XML flight message, which will be replaced by a forthcoming schema by the FAA Flight Object Working Group (FOWG) [26] or EUROCONTROL. The sidecar also associates tracks to their respective flight plan via the mode3A code assignment in the filed flight plan, e.g., mode3A 4031 transponder is aircraft with aircraft flight id AAA246.

4.3.3 Track Association Service and Common Operating Picture

The track association service receives tracks from multiple stations, correlates them, assigns a system wide unique identifier and publishes that identifier back to the stations. Messaging occurs via track brokers on the ESB. Some track data in overlapping edge regions may have to be exchanged between stations on the tactical bus. Use of federation enables systems to be scaled through distributed brokers.

A Common Operating Picture provides uniform situation awareness of air traffic for all consumers. It is important for all consumers to have the same view of the air picture, including a universal track number, if the air picture comes from multiple sources.

4.3.4 Sense and Awareness (SAA) Service

The SAA service will be described under a separate forthcoming paper.

5 MIT Lincoln Laboratory SOA Testbed Implementation

Lincoln Laboratory has led a number of SOA open architecture initiatives. This project has leveraged existing services in the reference architecture from these projects through software reuse and has developed some new services. Figure 2 shows the services in the testbed. We describe the implementation and evaluate the performance of these services as a Proof of Concept. We have integrated both Commercial Off the Shelf (COTS) trackers via a mediator to convert proprietary format into open format, and in-house trackers into the testbed.

Several key lessons have been learned in previous SOA initiatives. The NCT is a common set of service components that can be leveraged by many projects. Developing a sidecar Software Development Kit (SDK) and ICD that enables rapid creation of a sidecar in such a way to meet low latency and high-throughput requirements can reduce software development cost. We have learned how to create a high-performance message bus to enable SOA concepts. We have learned several important lessons for creating data models for open interfaces. This background has led to the current implementation.

Without costly flight tests or complex demonstrations, a SOA is being used to test SAA and NAS integration system components quickly. Furthermore, a SOA overcomes the challenge of producing repeatable scenarios by controlling the environment in which SAA and NAS integration systems are tested. For example, two ground surveillance sources can be compared by using the same tracker, UA platform, Concept of Operations (CONOPS), and environment conditions. A SOA is used to help determine requirements for surveillance and weather sources, CONOPS, algorithm performance, communications, operating regions, and more.

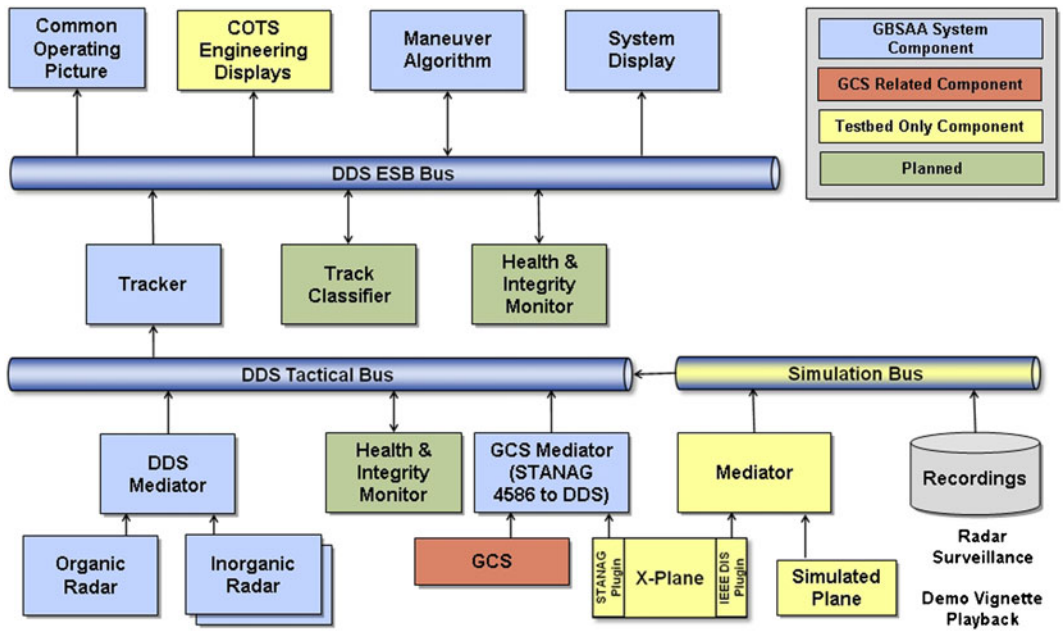


Fig. 2 MIT Lincoln Laboratory SOA Testbed. Organic sensors are existing surveillance radars, and Inorganic Sensors are new sensors dedicated to GBSAA. COTS displays are Commercial Off the Shelf products and may require mediators (not shown) for integration into the testbed

An SAA testbed can also facilitate system design and can be used for the SAA system certification process. It is desirable to test candidate SAA algorithms with surveillance from real air traffic. For repeatability and to reduce mission cost, a testbed enables the injection of simulated air tracks into a UA air surveillance picture in order to develop and test algorithms.

Additionally, the system allows simultaneous evaluation of multiple algorithms against a common scenario. This requires coordinated and time synchronized playback of all data items for repeatability, and it would be desirable to play back in faster than real-time. The system adds the ability to fly simulated UA or intruder aircraft. The system supports the ability to hold an algorithm constant and evaluate the effect of substituting different system components such as correlator-tracker. The testbed allows human in the loop simulations. These use cases established the requirements for a testbed that can operate in real operations or in shadow mode from real operations for human machine interface development.

The target SOA performance regime is for a tactical application. This includes soft real-time and some hard real-time latency goals. This performance regime is not as exacting as messaging in an embedded system or an avionics bus but is more restrictive than traditional SOAP/WSDL web services. The ESB latency must be in tens of milliseconds (mS)s with guaranteed quality of service for delivery and latency jitter. Messages are in the 100 per second to 1000 per second range. Bandwidths are megabits a second.

The SOA testbed implementation includes timing agents that are components that hook into the processing chain in order to measure performance. Each of the processing steps in the chain is implemented by using the mediation service framework described in Section 4.2. In a production scenario, the process chains contain routes that connect a source of data directly to its processor and target endpoint. In a testing mode, the processing chain is modified to include a timing agent step that extracts one or more correlation values and signals a central event processor with the identifier for the step and the correlation

details of the message that it is processing. The central event processor applies a timestamp to each message it receives and records them for future analysis. For example, a step in the chain that consumed Air Track messages and republished them as Cursor On Target would identify itself with a unique label. Each of the timing messages sent from this timing agent would contain the unique Track ID and the Time of Validity fields from the Air Track Message that are collectively unique. This data is sufficient to follow the progress of a single Air Track Message all the way through the message bus. The use of a central event processor to provide the timestamp removes the complexities of keeping multiple clocks in sync.

Standard analytic packages such as Matlab can be used to process the data gathered to provide a detailed look at the performance over a number of dimensions. Additionally, a real-time event processor can be configured to interpret start and end messages from the timing agents and provide a running view on system performance. A simple implementation can include a graphical rendering of the number of messages processed as well as the min, max, and mean latencies for each sample period. Such real-time event processors could be adapted to feed standard monitoring systems such as Hyperic or OpenNMS.

The testbed SOA is deployed on a pair of Dell PowerEdge R610 16 core 2.4 GHz Xeon, with 12 G memory running 64 bit Red Hat Enterprise 5 Linux with 2.6.18 SMP kernel. The testbed has gigabit Ethernet interfaces, and all testing was performed on a private LAN. Code is written in Java 6 release 21 by using the Oracle/Sun standard JVM. JVM garbage collection parameters are set to homogenize collects at a slight increase in overall latency [54].

6 SOA Testbed Performance Evaluation

6.1 SOA Performance Methodology

Performance requirements for a messaging service are determined per mission area and service users. For example, a tracker receiving real-time sensor reports generally puts stringent requirements on message latency and latency distribution but can easily tolerate some dropped messages, while a service distributing operator commands usually needs a very low drop rate but can accommodate a much higher latency.

We selected a set of general purpose metrics to capture sufficient quantitative information about the messaging service performance that they, together with standard network performance metrics (average and peak bandwidth, etc.), can be used to evaluate the suitability of the architecture for specific use cases and mission areas.

For a pair of points A and B in the message processing path we define the following three metrics:

Completeness, a number between 0 and 1, is the fraction of messages passing through **A** that are correctly received at **B**.

Out of order rate, a number between 0 and 1, is the fraction of messages received at **B** out of order.

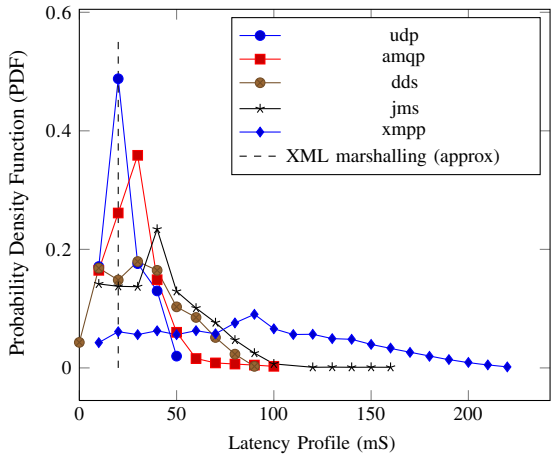
Latency profile, a Probability Density Function (PDF) of message latency, i.e. the difference between time of arrival at **B** and time of departure from **A**. In addition to its use for validation of the end-to-end system the latency profile can provide valuable insight into performance of individual architecture elements. For example, a latency profile can indicate a normal operation of an event-driven, unstressed task or queued transmission bursts.

6.2 MIT Lincoln Laboratory Testbed SOA Performance Results

This section describes benchmarks of the Common Operating Picture service, one of the core services that enables shared situational awareness. The performance metrics in the testbed can be computed on live data or on playback data, and this evaluation uses the latter in order to achieve repeatablilty. The baseline scenario centers on an air picture of the Midatlantic region derived from a 10 minute FAA dataset. The raw radar contacts are tracked with a COTS correlator-tracker that batches the output track data into groups of 2–27 reports (nominally 15), which are the input to

the SOA system. There are nominally between 170 and 220 entities in simultaneous track in the air picture with a total message rate from all surveillance sources on the order of 40 updates per second. The XML track message is on the order of 3900 bytes corresponding to 35 bytes message size in the proprietary vendor binary. If the vendor specific items are removed, the XML size reduces to about 1900 bytes. We have obtained XML compression on the order of 40 % by using Efficient XML protocol [55].

Figure 3 compares the latency characteristics for the SOA by using several transports and shows the corresponding latency Probability Density Function (PDF) profiles. Latency is defined to be the elapsed time data traverses the SOA, which includes time in the tracker sidecar to convert legacy data into XML on the tactical bus, time in the message transport bus, and time through SOA client. This is essentially the latency cost of breaking apart a stove-pipe system and inserting a SOA. A UDP transport in the message bus represents a lower bound to any IP based transport. The latency profiles for AMQP and DDS closely match the straight UDP transport. DDS is implemented in RTI Java DDS using a custom Apache Camel Connector, and XML documents are passed as strings.



Transport	Min latency (mS)	Median latency (mS)	95 % latency (mS)	Max latency (mS)
UDP	4.6	22.6	41.2	55.6
AMQP RabbmitMQ 2.1.1 (tcp)	6.6	26.9	48.3	93.5
DDS RTI 4.5d Java, XML as IDL String, Default QoS (udp)	2.9	32.9	69.4	110.
JMS ActiveMQ 5.2.0 (tcp)	5.7	41.6	82.6	222
XMPP OpenFire 3.6.4 (tcp)	8.2	121	225	344

Fig. 3 Latency Distribution for different SOA middleware bus protocols. The latency includes SOA input mediator (with encoding of vendor binary format into XML), message bus, and SOA client

The study shows that several protocols would be relevant for SOA messaging with latencies in the 10's of mS, if the XML marshalling were replaced with a binary format. DDS would meet this requirement and leverage an open standard as well. Therefore, in the final Army architecture, IDL would be used for more efficient message transfer and to enable content filtering. A DDS "fire and forget" QoS policy is used in which messages are sent without acknowledgment. A DDS reliable UDP QoS setting or any content filtering would likely increase latency. The RTI Java implementation layers on Java JNI, and we expect a native DDS C or C++ application to perform slightly faster. All of the technologies utilize a central broker except for UDP and RTI DDS, which has important implications for a fault-tolerant architecture [56]. These factors would make DDS the leading choice for messaging in a high-performance, low latency SOA.

The latency distribution is important as well, and having a narrow spread is required by some applications. Near real-time applications also require a predictable and known latency jitter. The distributions are narrow with a long thin tail, except for the XMPP, where there are a few outliers due to Java garbage collection. Some benchmark runs, especially the XMPP ones, exhibited pauses on the order of 100 mS. Using one of the real-time Java JVM implementations might be a means to make the garbage collection more deterministic and mitigate the occasional latency spikes. This is under exploration.

We have measured the latency budget of the SOA elements in the processing chains using the techniques in Section 5. The bulk of the latency budget is due to XML marshalling, the bus median latency is a few mS, and the client latency is several mS. We have measured the Apache Camel

overhead to be small, around 0.5 mS. An additional transformation of XML from one schema to another adds on the order of 2 mS to the latency budget. We measured the completeness score to be one, and the out of order rate to be zero. Based on these testbed findings and other techniques, an architecture that uses DDS is very attractive [57]. DDS provides options for QoS settings that are important as one means to implement service specifications. DDS architecture also provides a number of features for redundancy and high availability that make it attractive for mission critical applications, and coupled with its high-performance characteristics, recommended by this work.

We plan to evaluate a number of other techniques that might yield even higher SOA performance. These include use of real-time JVM, network stack parameter tuning, TCP Offload Engine (TOE). DDS will soon enable pluggable transports, and use of a higher performance fabric than IP such as InfiBand [58] is intriguing because these transports provide more deterministic real-time behavior than IP.

The testbed contains the capabilities to simulate and explore network conditions such as lossy or slow links, or the effects of routing anomalies. This framework is called LARIAT [59], and studies related to SOA execution in a challenging network environment will he subject of a future work.

7 Conclusion

We have designed and implemented an open standard-based SOA for UAS integration in the NAS. Key functions were delineated, and simulated operational performance limitations have been estimated. Plans are to extend this work by developing and evaluating additional services. MIT Lincoln Laboratory plans to pursue any necessary standards within relevant standards bodies.

Although SOA presents a number of technically desirable advantages, certifying a SOA system for safety-critical avionics presents a number of challenges, and to the best of the author's knowledge has never been done before. One method is to certify the reference architecture as presented. On the other extreme is dispensing with the architecture entirely and adhering to a traditional avionics software approach. The solution is likely some combination of both, accomplished by lifting some of the key SOA principles and applying them using a traditional avionics software approach. Over time, it would be desirable to incrementally move the boundary toward incorporating more SOA principles. However, substantial work needs to be done for this to happen, and a number of questions need to be asked and answered for this evolution to take place.

One aspect of certification may be more straightforward with a SOA. The use of a formal service contracts, with comprehensive interface specifications, quality of service specifications, and governance process presents an opportunity for quantitatively testing and monitoring a service at the interface level. This is precedented in the satellite and spacecraft community through a component based certification approach [60] in which simulation and formal modeling of elements with well defined interface specifications enable reusable components. There are commercial COTS appliances for monitoring service level agreements of web services, and perhaps these could be adapted to aid in the monitoring and verification of the real-time services described. A testbed provides the opportunity to quantitatively evaluate a service from reliability and quality of service level and other aspects relevant to certification. Future testbed work will include evaluating services from a certification standpoint to order to help establish the current state of maturity. This may help address next steps in incrementally evolving the certification process.

A SOA makes some aspects of certification more challenging. One set of issues lies around the use of object oriented methodology in avionics software. This is addressed in the recent DO-178C standard, which provides some guidance in object oriented avionics software development. Mitigating the deterministic scheduling and garbage collection in Java could be explored by examining some of the real-time Java implementations that are commercially available, or code could be written directly in C++. Java development tends to incorporate third party libraries, which is problematic because of open source code and

possibly complex secondary dependencies. Mitigating this might be explored by in-house library development, and/or use of C++. Additional certification processes and standards may need to be developed for SOA.

We have presented some of the advantages of SOA principles and have shown how they can be implemented in a fashion to meet high-performance requirements. The next step is to address the certification issues. This architecture description serves as a guideline for projects that integrate UAS in the NAS, and we plan to leverage these principles in the Army GBSAA project.

Acknowledgements The authors greatly appreciate the support and assistance provided by Viva Austin, Product Director, Department of the Army, Unmanned Systems Airspace Integration Concepts (USAIC).

References

1. U.S. Army UAS Center of Excellence: Eyes of the Army U.S. Army Roadmap for Unmanned Aircraft Systems 2010–2035. Report no ATZQ-CDI-C 72 (2010)
2. Peña, N., Scarlatti, D., Ollero, A.: UAVs integration in the SWIM based architecture for ATM. J. Intell. Robot. Syst. **54**(1), 39–59 (2009). doi:10.1007/s10846-008-9254-1
3. Batavia, P., Ernst, R., Fisherkeller, K., Gregory, D., Hoffman, R., Jennings, A., Romanski, G., Schechter, B., Hunt, G.: The UAS Control Segment Architecture: An Open, Agile Development and Acquisition Model for UAS Ground Control Stations. AUVSI Unmanned Systems North America (2011)
4. Nelson, E.M.: Open Architecture for the Enterprise, Part 1: Architectural Principles of Open Architecture. IBM (2007)
5. Nelson, E.M.: Open Architecture Technical Principles and Guidelines, version 1.5.8. IBM (2008)
6. Birman, K., Hillman, R., Pleisch, S.: Building net-centric military applications over service oriented architectures. Proc. SPIE **5820**, 255 (2005). doi:10.1117/12.605149
7. Davis, C., Flavin, J., Boisvert, R., Cochran, K., Cohen, K., Hall, T., Hebert, L., Lind, A.-M.: Enhanced regional situation awareness. Linc. Lab. J. **16**(2), 355–380 (2007). http://www.ll.mit.edu/publication/journal/pdf/vol16_no2/16_2_07Davis.pdf. Accessed 23 Sept 2011
8. Rejto, S.: Radar open systems architecture and applications. In: The Record of the IEEE 2000 International Radar Conference, 2000, pp. 654–659 (2000). doi:10.1109/RADAR.2000.851911
9. Evans, J., Ducot, E.: Corridor integrated weather system. Linc. Lab. J. **16**(1), 59–80 (2006). http://www.ll.mit.edu/publications/journal/pdf/vol16.../16_1_4EvansDucot.pdf. Accessed 23 Sept 2011
10. Moser, W.: Design and implementation of the TFDM information management architecture. In: Integrated Communications Navigation and Surveillance Conference (ICNS), 2010, pp. 1–27, 11–13 May 2010. doi:10.1109/ICNSURV.2010.5503304
11. MIT Lincoln Laboratory: Extended Space Sensors Architecture. Lincoln Laboratory Tech Notes (2009). [online] http://www.ll.mit.edu/publications/technotes/TechNote_ESSA.pdf. Accessed Sept 2010
12. Vidan, A.: Lincoln Laboratory distributed disaster response system. Lincoln Laboratory Tech Notes, [online] http://www.ll.mit.edu/publications/technotes/TechNote_LDDRS.pdf. Accessed 23 Sept 2011
13. Garces-Erice, L.: Building an enterprise service bus for real-time SOA: a messaging middleware stack. In: 33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09, vol. 2, pp. 79–84, 20–24 July 2009. doi:10.1109/COMPSAC.2009.119
14. Panahi, M., Nie, W., Lin, K.-J.: A framework for real-time service-oriented architecture. In: IEEE Conference on Commerce and Enterprise Computing, 2009. CEC '09, pp. 460–467, 20–23 July 2009. doi:10.1109/CEC.2009.78
15. Garcia-Valls, M., Rodriguez-Lopez, I., Fernandez-Villar, L., Estevez-Ayres, I., Basanta-Val, P.: Towards a middleware architecture for deterministic reconfiguration of service-based networked applications. In: 2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–4, 13–16 Sept. 2010. doi:10.1109/ETFA.2010.5641071
16. Fattah, A.: Enterprise reference architecture. In: 22nd Enterprise Architecture Practitioners Conference, London, UK. http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf (2009). Accessed 23 Sept 2011
17. Josuttis, N.: SOA in Practice. O'Reilly (2007)
18. [online] http://www.json.org/xml.html. Accessed 23 Sept 2011
19. [online] https://acc.dau.mil/oa. Accessed 23 Sept 2011
20. [online] https://acc.dau.mil/CommunityBrowser.aspx?id=121180&lang=en-US. Accessed 23 Sept 2011
21. [online] http://UAV-c2-initiative.mil. Accessed 23 Sept 2011
22. [online] http://www.opengeospatial.org/standards/sps. Accessed 23 Sept 2011
23. [online] http://www.ucsarchitecture.org/page/home. Accessed 23 Sept 2011
24. [online] http://cot.mitre.org/. Accessed 23 Sept 2011
25. [online] http://www.eurocontrol.int/asterix/public/standard_page/documents.html. Accessed 23 Sept 2011

26. [online] https://faaco.faa.gov/?ref=9088. Accessed 23 Sept 2011
27. [online] http://www.nato.int/docu/standard.htm. Accessed 23 Sept 2011
28. [online] https://wiki.ucar.edu/display/NNEWD/WXXM. Accessed 23 Sept 2011
29. [online] http://www.aixm.aero/public/standard_page/concepts_standards.html. Accessed 23 Sept 2011
30. [online] http://dmtf.org/standards/cim. Accessed 23 Sept 2011
31. [online] http://www.niem.gov/TechnicalDocuments.php. Accessed 23 Sept 2011
32. [online] http://www.open-mpi.org/. Accessed 23 Sept 2011
33. Information Assurance Technology Analysis Center: Software security assurance. In: State-of-the-Art Report (SOAR) (2007)
34. Ibrahim, L., Jarzombek, J., Ashford, M., Bate, R., Croll, P., Horn, M., LaBruyere, L., Wells, C.: Safety and Security Extensions for Integrated Capability Maturity Models. FAA (2004)
35. Scott, A.D., Clay, P., Masone, M.: Certification and accreditation of SOA implementations: programmatic rules for the DoD. CrossTalk: The Journal of Defense Software Engineering **22**(7), 19–24 (2009). http://www.crosstalkonline.org/storage/issue-archives/2009/.../200911-Scott.pdf. Accessed 23 Sept 2011
36. Sharp, D., Bell, A., Gold, J., Gibbar, K., Gvillo, D., Knight, V., Murphy, K., Roll, W., Sampigethaya, R., Santhanam, V., Weismuller, S.: Challenges and solutions for embedded and networked aerospace software systems. Proc. IEEE **98**(4), 621–634 (2010). doi:10.1109/JPROC.2009.2039631
37. Brennan, J.J. (ed.): Information Assurance for SOA. MITRE, Bedford (2010)
38. [online] http://cio-nii.defense.gov/sites/dodaf20/. Accessed 23 Sept 2011
39. [online] http://everware-cbdi.com/cbdi-forum. Accessed 23 Sept 2011
40. Sprott, D.: The CBDI-SAE Reference Framework in 2010. CBDI Journal. http://everware-cbdi.com/cache/downloads/6o5n1t4q4iw4os0ckg8w4ws4c/Journal2010-09.pdf (2010). Accessed 23 Sept 2011
41. CBDI template, pp. 4–17. [online] http://everware-cbdi.com/rss-template. Accessed 23 July 2012
42. DHS Service Oriented Architecture—Technical Framework, version 1 (2007). Section 8 and Appendix D
43. Correa, T., Becker, L.B., Farines, J.-M., Bodeveix, J.-P., Filali, M., Vernadat, F.: Supporting the design of safety critical systems using AADL. In: Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '10), pp. 331–336. IEEE Computer Society, Washington, DC (2010). doi:10.1109/ICECCS.2010.56
44. Gregson, K.: Cross-domain ISR maritime awareness demonstration. Paper presented at HPEC Conference, Lexington, MA (2009)
45. [online] http://www.disa.mil/nces. Accessed 23 Sept 2011
46. [online] http://cio-nii.defense.gov/sites/coi/Training/DT-07-NCES-Capabilities.ppt. Accessed 23 Sept 2011
47. [online] http://www.w3.org/TR/rdf-sparql-query/. Accessed 23 Sept 2011
48. Konieczny, E., Ljungberg, M.: Semantic framework for automatic resource brokering in distributed government systems. In: Semantic Technology Conference (2010)
49. Van Hook, D., Ljungberg, M., Aubin, E., Shaw, R., Ford, M., Konieczny, E., Lee, D.H., Brown, S.T. IV: Resource brokering service: timely and efficient information resource allocation. In: Defense Transformation and Net-Centric Systems, SPIE, vol. 7707 (2010). doi:10.1117/12.850588
50. Ford, M., Ljungberg, M., Van Hook, D., Shaw, R., Aubin, E.: Resource Brokering Service: Automatic Plan Composition and Execution. MILCOM (2010). doi:10.1109/MILCOM.2010.5680336
51. [online] http://metadata.dod.mil/mdr/irs/DDMS/. Accessed 23 Sept 2011
52. Viggh, H., Weed, C., Chan, M.: Composable Applications Using Service Encapsulation (CAUSE). MILCOM (2010). doi:10.1109/MILCOM.2010.5680324
53. [online] https://kepler-project.org. Accessed 23 Sept 2011
54. [online] http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html. Accessed 23 Sept 2011
55. Efficient XML Interchange (EXI) Format 1.0. World Wide Web Consortium, Working Draft (2008)
56. [online] http://www.rti.com/whitepapers/Platform_for_Reconfigurable_UAS.pdf. Accessed 23 Sept 2011
57. Xiong, M., Parsons, J., Edmondson, J., Nguyen, H., Schmidt, D.: Evaluating Technologies for Tactical Information Management in Net-Centric Systems. Vanderbilt University (2007). doi:10.1117/12.719679
58. http://www.infinibandta.org. Accessed 23 Sept 2011
59. Rossey, L.M., Cunningham, R.K., Fried, D.J., Rabek, J.C., Lippmann, R.P., Haines, J.W., Zissman, M.A.: LARIAT: Lincoln adaptable real-time information assurance testbed. In: IEEE Aerospace Conference Proceedings (2002). doi:10.1109/AERO.2002.1036158
60. Weiss, A., Ong, E.C., Levinson, N.G.: Reusable specification components for model-driven development. In: Proceedings of the International Conference on System Engineering (INCOSE '03) (2003)
61. [online] http://nesipublic.spawar.navy.mil/nesix/Frames. Accessed 23 Sept 2011