13th ICCRTS: C2 for Complex Endeavors

DESIGN PATTERNS FOR NET-CENTRIC APPLICATIONS

Topic: C2 Architectures

Seth Landsman, Ph.D.
The MITRE Corporation
202 Burlington Rd, M/S M325
Bedford, MA 01730
781-271-7686 / 781-271-2964
landsman@mitre.org

Sandeep Mulgund, Ph.D.
The MITRE Corporation
202 Burlington Rd. M/S E015
Bedford, MA 01730
Voice 781-271-6207/Fax 781-271-2964
smulgund@mitre.org
Abstract

Numerous efforts are now under way to define techniques for exposing data in a net-centric manner. Such efforts will provide C2 decision-makers with access to unprecedented amounts of real-time operational information, helping to make possible the net-centric vision of improved mission effectiveness through shared situation awareness and self-synchronization. However, relatively little attention is being paid to strategies for consuming and exploit that data effectively and efficiently. Guidance such as the DoD’s Net-Centric Data Strategy (NCDS) suggests how best to expose mission data in a net-centric environment, but what are the right approaches for building user-facing tools and capabilities? Such capabilities will make it possible to transform, visualize, and realize shared sensemaking over net-centric data. As the diversity and number of entities needed to realize effective C2 for complex endeavors grows, establishing agile and practical design guidance for building user-facing capabilities becomes critical.

This paper discusses an effort to establish design patterns – identifiable, repeatable prescriptions for solving commonly occurring software design problems – for realizing agile net-centric consumer applications. The effort described herein centers on the development of a User Defined Operational Picture (UDOP) capability, a collaborative tool that enable agile consumption of net-centric data sources to support time-critical decision-making in crisis situations.

Keywords: UDOP, sensemaking, collaboration, design patterns

1. Introduction

The constant march forward of network-centric warfare (NCW) technologies continue to provide operators in military command centers with access to unprecedented amounts of real-time battlefield information, and offer the potential to enable transformational C2 capabilities that improve mission effectiveness through shared situation awareness and self-synchronization. Effective, intuitive, agile decision support and visualization capabilities will be a key driver of the shared awareness that improves mission effectiveness. It will be insufficient just to get information to people; it must be synthesized, organized, and presented in a way that can improve decision-making and operational effectiveness at all echelons of C2, from theater commanders to national leadership.

Making data available is no longer sufficient for effective C2 operation. Data that cannot be synthesized into a larger C2 picture has limited actionable value and requires significant, costly efforts on the operators part to incorporate it. While there are many efforts under way to define design patterns for exposing data in a net-centric manner, relatively little attention is being paid on strategies to consume and exploit that data effectively and efficiently. More generally, how does one compose user-facing applications in a net-centric environment? Documents such as the DoD’s Net-Centric Data Strategy (NCDS) (DoD, 2003) provide guidance on how best to expose mission data in a net-centric environment, but what are the right approaches for building user-facing tools and capabilities? It is necessary to take the next step after constructing the net-centric data sources and move towards the net-centric consumers of these data sources.

Building agile net-centric consumer applications requires definition of design approaches that represent a departure from the techniques used in legacy “stovepipe” systems. In particular, the

Approved for Public Release; Distribution Unlimited
following key technical questions arise, having broad relevance to the development of net-centric C2 capabilities:

- How do we compose user-facing applications in a net-centric environment?
- How do we realize human-driven workflows for exploiting and collaborating over net-centric data products?
- How do we model composite net-centric data products in a manner that is consistent with the tenets of the NCDS?

The objective of the work described in this paper was to establish design patterns – identifiable, repeatable prescriptions for solving commonly occurring software design problems (Gamma et al., 1995) – for realizing agile net-centric consumer applications. These design patterns provide concrete answers to the following questions:

- How do we partition the overall functionality of an application into a discrete set of communicating components that are consistent with net-centric tenets?
- How do we support multiple interaction styles for retrieving data?
- How do we support the different data formats that the application may need to understand?
- How many data formats should a net-centric application support?
- What are effective strategies for managing information and control flows within a net-centric consumer application?

This paper details design tenets and relevant design patterns for constructing agile consumers of net-centric data. The next section details different methods for interacting with, both accessing and consuming, data that is exposed by net-centric data providers. While a system may expose its data in a net-centric fashion, there is little agreement as to how the data may be accessed or how it can be leveraged in the larger C2 context. Section 3 discusses key design tenets for building consumer of net-centric data. To avoid a tightly coupled, “stovepiped” consumer of data, it is key that a system be constructed so that it is extensible and agile enough to exist in the ever-changing DOD information space, and to be able to adapt to new requirements, new information sources, and new data formats. Section 4 details a prototype User Defined Operational Pictures (UDOP) application that was built as an exemplar of the key net-centric patterns and tenets discussed in sections 2 and 3. This paper concludes with a set of recommendations for future directions for defining net-centric application architectures.

### 2. Data Interaction in a Net-Centric Consumer

Developing robust, interoperable C4ISR systems and solutions for the age of net-centric operations calls for a different design approach than has been used in the past. A typical C4ISR system may have data coming from multiple sources, each of which have a different data format, access method (e.g., representational state transfer (REST), Java messaging service (JMS), or file transfer protocol (FTP)), and security concerns. However, what a system does with data needs to be thought out, it is not enough just to get the data into a C4ISR system.

Within a system, different data may trigger different business logic or be routed to different subsystems. The user, depending on his role and function, may also need to access and use the data through a variety of different visualization tools, all of which may need custom data
representations. Clearly, the complexity of such a system needs to be reduced to be manageable and evolvable.

One approach for managing such complexity is to establish a key set of design patterns, segregating the system into distinct functional blocks that implement identifiable, repeatable designs for executing well-defined behaviors. This section discusses approaches for accessing and exploiting system data, how to obtain it and how to interact with it once it is brought into the system.

### 2.1 Data Access Approaches

While many techniques exist for net-centric data exposure, a consumer must be engineered to support many of them in a scalable way. In reviewing the data sources identified as priorities for net-centric data consumers, two key patterns emerged: simple and multi-part workflows for data access. Simple workflows entail a single exchange between requester and source to retrieve data of interest, although that exchange may entail complex processing on the source’s part. A data source may have to complete a long processing step before sending data to the client system. Such simple workflows may be associated with two different interaction models:

- **Request/reply**, in which a consumer asks for a specific set of data using query parameters, and receives a well-structured response from the producer in reply. Familiar examples include REST-style http interactions and Simple Object Access Protocol (SOAP) web services. Retrieving dynamic data from such sources typically entails polling on a recurring interval to receive updates. Rich Site Syndication (RSS), for example, entails polling against an XML formatted source.

- **Publish/subscribe**, in which a consumer indicates interest in a particular category of data, and then receives that data asynchronously as it becomes available on the network (or receives a notification that it is available, and can then retrieve it through request/reply means). Relatively new web services protocol standards make such behavior possible through SOAP interactions, though these are still somewhat experimental.

Multi-part workflows are a composition of simple workflows, which require multiple steps and supporting data to retrieve data. For example, the first step may be to retrieve a list of available resources or data products available, followed by retrieval of that specific source.

#### 2.1.1 Simple Workflow: Request-Reply

Simple request-reply access forms the basis of web based data retrieval. The client or user sends a request (including query parameters) to an address, and receives a corresponding reply. Figure 1 illustrates a generic request-reply workflow. Note that the client can be configured to make a recurring request on a user or system specified interval. This allows the system to poll for the most up-to-date data; however it does not ensure that the client always has the timeliest data at any given time.
Two common variations of request-reply for net-centric data access are HTTP-GET and SOAP web services. From a high-level interaction perspective they have similar interaction patterns; however the latter provides a greater degree of formalism and structure for web-based XML information exchange, and there are different degrees of complexity in their implementation.

Request-reply interactions over HTTP use the GET method for content retrieval. An HTTP GET request is made via a network address encoded as a uniform resource locator (URL), with any parameters represented as key-value pairs. The URL ideally provides a persistent link to the data source, and provides its most updated data each time it is requested.

SOAP web services use HTTP as the transport mechanism, and they provide more formal specifications for XML-based information exchange. Web Services description Language (WSDL) documents specify the interface for the service. SOAP requests can be transmitted using either HTTP-GET or HTTP-POST, though a formal discussion on the SOAP protocol is beyond the scope of this document (Gudgin, et al, 2007). Creating a client that can interact with a SOAP web service entails the use of software compilation tools that generate the client-side stubs for a given WSDL specification.

### 2.1.2 Simple Workflow: Publish-Subscribe

Publish and subscribe interfaces make it possible to provide information in a more timely fashion than polling via request-reply. Figure 2 illustrates a typical publish-subscribe interaction pattern.
The client sends a request to the data source to establish a subscription. The request may encode filtering parameters that will scope what data the source returns. The data source returns a handle to that subscription to the client. The handle is a unique identifier for the subscription, which the client can use to alter or terminate the subscription in the future. After the subscription is established, the data source will deliver data to the client as it becomes available.

2.1.3 Multi-part Workflow

Multi-part workflows are compositions of simple workflows to retrieve desired data from a providing source. These workflows may entail user interaction to select the desired data from a catalog thereof, or they may be automated so that machine logic chooses what data is appropriate and then requests it. For example, a user may need to retrieve a list of available air missions before choosing which missions to modify or visualize. Two variations of multi-part workflows are those that require user interaction, and those that do not.

Figure 3 illustrates a multi-part workflow that requires user interaction. In this example, the client must process information from the data source and then receive input from the user to make the data request. The client cannot automate the data request until it receives user input to select which data is of interest.
Figure 3, Multi-part workflow with user interaction

First, the client environment will build a list of available data products for selection. The user selects which data product(s) to request. The client environment requires supporting data to help retrieve the required data, so it requests this data first. After retrieving the supporting data, the client environment requests the final data product(s) from the data source. Retrieving data via an RSS advertisement model follow this pattern.

Figure 4 illustrates a multi-part workflow that requires no user interaction. A user may interact with the client environment to initiate the data request, but after that the client environment is responsible for executing the multiple steps required to complete the data request.
The client (or consuming) environment retrieves a list of available data. The data list is processed by the client based on business rules (e.g., select the most recent item, select the item when it impacts country x). The client environment requests the data for the item selected.

Business rules can handle automated polling of a data source based on a specified time period. For example, Air Tasking Order (ATO) data generally changes within a 24-hour cycle. The client environment can contain an automated task that polls for the new ATO every 24 hours. The environment can encapsulate the business logic to execute this process, only requiring minimal configuration during deployment.

2.2 Data Exploitation Approaches

In a previous paper (Mulgund & Landsman, 2007), the authors discussed how data may be exploited in the net-centric consumer. The three major focuses of exploitation were:

- **Visualization and presentation** mechanisms to provide the requisite historical, current, and anticipatory situation awareness
- Domain-specific **business logic** to create derived, added-value information products from the raw data inputs and displayed data and to extract insight based on the content therein
- **Sharing and collaboration** tools to enable shared situation awareness and collaborative decision-making based on decision-focused views and information. In the 21st century net-centric C2 enterprise no user will be operating alone; how to use C4ISR net-centric consumers for collaborative analysis and planning must also be understood.

Once the necessary data products are retrieved into a net-centric consumer environment, it must be visualized to support varying levels of situation awareness. Geospatially oriented data will
naturally be presented using a combination of 2D and 3D map displays. Mission schedules may be shown in timeline views, and it may also be desirable to present tabular data as well as unstructured textual information. The consumer should allow the user to create and display a tailored information environment that stitches together multiple modalities of data (geospatial, temporal, tabular, textual, etc.). Once the picture is built the user may want to annotate features of interest.

Complete situation awareness relies not just on understanding the current circumstances, but also how a particular situation evolved and how it is likely to progress in the future. Thus, the visualization tools should provide “TiVo” style functionality to provide user-controlled animation between past, present, and anticipated future. Animation can be a very effective tool for allowing observers to understand the evolution of a situation. It has long been used as a mechanism for expressing time-dependent information. Animated views should be combined with timeline views that allow the observer to place the current scene in the context of a longer-term timespan such as a mission timeline.

As part of visualization and presentation, metadata regarding the data being exploited should be made apparent. For example, the quality of the data is often key to its value. Depicting information quality requires that the data delivered through the services provide the necessary metrics. The responsibility for describing information quality and pedigree rests with the data providers upstream of a consumer. However, once such measures are available, a consumer should represent them properly.

Closely related to the presentation function is business logic for deriving new information products from the retrieved data, over and above the visual transformations discussed earlier. As the data sets being used and accessed become larger and more complex, pure visualization of the data may not provide the proper interaction with the data. Exploiting data will involve doing analysis and extrapolation on the data.

While discussion of different types of decision-focused data analysis and data mining are beyond the scope of this paper, a net-centric consumer needs to support their operation. One method for doing so is to support the creation of a data artifact, a representation of how data is to be obtained from the net-centric data sources. The approach where all data is locked in the consumer, and cannot be communicated or shared to other clients or services reduces the value of the assembled data. A properly built data artifact can be leveraged by analysis capabilities.

Extrapolation on a data artifact is also an important capability. For example, one straightforward function is extrapolation in space/time, to support the TiVo capability. The ability to collect historical data and created predicted data has utility for, for example, course of action (COA) development by subject matter experts.

Once a data artifact has been created, a mechanism is needed to publish that decision-focused product into the network so that others can consume it. Above, using the data artifact to interact with analysis capabilities was discussed, but just as important is the ability to use it to interact with other subject matter experts and other members of the C2 community. They collaborative functions of a net-centric consumer should include:

- Sharing a data artifact as a specification of information content (i.e., a “recipe”)
- Sharing a data artifact as de-referenced data for the benefit of those users who do not have access to the enabling net-centric data sources
• Annotating a data artifact to provide metadata on its contents, highlight areas of interest, and include subject matter expertise and evaluation
• Incorporating changes in the data artifact from other C2 users

These approaches to data interaction in a net-centric consumer show the types of data and types of interaction over data that are expected of a consumer. In the section, core principles for how a system that matches these capabilities can be constructed are discussed.

3. Core Design Principles

As part of defining how agile net-centric data consumers should consumer and exploit data, and their best practices for doing so, we have defined a set of core design principles that reduce the complexity of developing such a system. This section describes these principles, which are:

• Employ a **loosely coupled architecture** that enforces strict boundaries and clear interfaces between functional components. Such decoupling reduces dependencies between different components and allows each of them to improve without affecting one another.
• Make all data transmissions within the architecture **asynchronous**, regardless of the interaction style of the underlying data source
• **Localize dependencies on specific data formats** and sources, so that the complexity and unique challenges associated with each of them does not propagate all the way to the clients
• **Encapsulate message flows with schemas** and metadata that are understood by all participants
• **Leverage commercial, off the shelf (COTS)** technologies, but identify and **localize any dependencies** for flexibility and portability

Each of these principles is discussed below.

First, a net-centric consumer includes many distinct functional components, whose purposes include data access, information management, and visualization. Technologies to support each of these functions are evolving rapidly. Systems should adhere to the tenet of **loose coupling** between functional components, which decouples them from each other and specifies clear interfaces between them. Such decoupling reduces dependencies between different components and allows each of them to improve without affecting one another.

Visualization technology offers a case in point. There now exist a range of easy-to-use technologies and tools for viewing geospatial data, including Google Earth, Google Maps, Satellite Toolkit, NASA Worldwind, and many others. None of them is “best”, and which solution is optimal depends on the use case and associated constraints. As such, a system designed to support visualization of net-centric data should provide flexibility for integration of different visualization tools as they become available or needed. Doing so requires an architectural approach that promotes such interoperability.

A second key design principle is to make **all data transmissions asynchronous**, for example, by using a **messaging** (Hohpe & Woolf, 2004) architecture. The consumer architecture must accommodate interaction with external data sources that exhibit either a request/reply or publish/subscribe interaction style, as discussed in the previous section. The challenge becomes how to handle both uniformly. In a request/reply interaction, typically the consumer “blocks”
(i.e., waits) for an answer to return from the source. In a publish/subscribe interaction, the consumer establishes an interest in some set of data, and it returns when it is available or changes. The consumer must detect the incoming data and process it accordingly.

A net effect of using an asynchronous model is that consumer applications and user interfaces do not block user interaction while waiting for data to return. While such behavior can be achieved through client-side multi-threading, a core messaging framework unifies data interaction across a range of possible consumers and data sources. Using messaging requires some way of associating replies with requests, so that when data returns the consumer knows why it’s receiving the data. One way to achieve this association is through the use of correlation. Each request contains a unique tag that identifies it. Returning data messages contain that same identifying tag, allowing the client to match the data against known requests.

A net-centric consumer should localize dependencies on specific data formats and sources. The messaging approach discussed above provided a uniform mechanism for managing information flows, but there still exists the question of what the data means. Performing any filtering or data transformation within a consumer requires understanding what that data means. There are many different data formats in use today in the C2 community. While there are efforts under way to unify them into standardized community of interest (COI) vocabularies, there is no single data format that can represent all required data types. Thus as a practical matter, it would be necessary for a consumer to understand at least a few different data formats.

The challenge becomes to define what the supported formats should be, and how to manage their use within the services. Deciding which formats to support internally depends on understanding the spectrum of data that a consumer will be used to visualize. However, having defined them, we can then specify how to deal with those choices within the a consumer architecture. Data source adapters may normalize data into one of a set of supported formats. Underlying data sources must expose their data in those formats, or the adapters must perform a translation to one of the supported formats. This becomes a many-to-few translation problem. In the long run, the few supported formats would likely coalesce around COI standards, and data providers would expose data using those standards. In the meantime, a many-to-few reduction strategy provides a practical path forward. Above the information services layer, we further isolated dependencies on data formats to specific transformation modules within the presentation service, and in the client-side rendering logic. Adding support for new data sources then becomes a matter of writing adapters for interacting with those sources, and defining appropriate transformation modules within a consumer if the data format is not already supported. This approach facilitates expanding the set of supported data formats and interaction mechanisms.

Another key tenet is to encapsulate all message flows with schemas and metadata understood by all participants. A consumer may support many different behaviors, such as connecting to or disconnecting from data source adapters, building, editing, deleting data artifacts, subscribing to data sources or system artifacts, and so on. Defined XML schemas for each key message and specifying web service interfaces in terms of those schemas is one approach to encapsulation. Doing so may simplify interpretation and usage of the various message types.

Finally, while there are many custom developed aspects of a net-centric consumer, a considerable portion of it depends on infrastructure capabilities such as J2EE servers. The development of the reference implementation made extensive use of COTS technologies to accelerate the effort. However, like the localization of data source dependencies, also it is
necessary to identify and isolate any dependencies on COTS products. So for example the architecture may contain web service classes that are deployed to a BEA application server, acting as a proxy to deployment-independent core business logic. Such partitioning will facilitate migrating a consumer between deployment environments, and also not make it dependent on any one particular software vendor.

4. Example: User Defined Operational Picture

The UDOP prototyping effort provides a use case for how to realize the principles and patterns discussed in the previous two sections into practice. This section describes how these patterns were realized and how they accelerated and improved the overall design of the prototype.

4.1 Architecture Overview

Figure 5 illustrates a layered view of the UDOP prototype architecture and components. The architecture shown is expressed in terms of the layered model advocated by the ESC Strategic Technical Plan (ESC, 2005). In STP parlance, the UDOP system resides in the information infrastructure and application layer. The bottom of the diagram shows the various systems of record providing data that will drive a UDOP. These sources are encapsulated by information access services that provide access via request/reply and event-driven means, using a variety of potential transport mechanisms and interaction design patterns (ESB, SOAP, REST, etc.).

Figure 5: Layered UDOP Architecture

The information services layer exposes the data from systems of record for consumption by the UDOP prototype. As discussed earlier, there exists a myriad of mechanisms for exposing data via net-centric means, and there is no single “ideal” approach. Developing interfaces to each type of data source typically entails design-time coding of service clients that are tailored for the
technology in use. The complexity can become overwhelming if the UDOP client applications must mediate between many different styles of data consumption, which will necessitate developing many “one-off” interfaces to different data sources.

To normalize how data is obtained from the varied types of possible data sources, the prototype uses an adapter (Gamma et al., 2005) class, which presents a uniform, service-oriented interface and interaction pattern to each system of record. Each adapter is registered in the data source registry, where its location and metadata, such as what parameters it can filter on what data formats it exposes are recorded.

All data access through the data source adapter is publish/subscribe-based. A connection is established to the data source adapter, which spawns a per-connection Data Source Worker. The worker communicates to the system of record and collects data from it. As new data becomes available it is published to a JMS topic, addressed to the requesting object.

The UDOP services layer corresponds to the STP’s application services layer, and contains three constituent parts, the UDOP Repository, the Presentation Service, and the Presentation Worker, all of which play a role in manipulating the UDOP data artifact, called the UDOP document. The UDOP services are an ensemble of net-centric services that exposes the functionality for building, maintaining, and exposing composite UDOP products. They provide mechanisms for authoring and managing UDOP documents, and presentation logic for dereferencing the contents identified in a UDOP document into data streams that can be rendered in supported clients. It relies on a database that acts as a permanent store of published UDOP documents.

The UDOP Repository provides the mechanisms for creating, modifying and deleting UDOP documents in a database. They also allow clients to retrieve a catalog of existing UDOP documents, search for UDOP documents matching given criteria, and retrieve a particular UDOP document of interest. Their scope is limited to interactions associated with retrieval of UDOP documents only; they perform no direct interaction with the underlying data sources.

The Presentation Service and Presentation Worker jointly handle a client’s exploitation of a UDOP document. When a client first establishes a connection to the UDOP system, a presentation worker object is created for his session. This worker is configured at runtime to support the capabilities of that client, including the data format and access method that client can support. When the client subscribes to a UDOP document, the presentation worker handles the exploitation of the artifact into a stream of data to the client.

The presentation worker contains three modules, which make up the presentation pipeline, the Aggregation Module, the Transformation Module, and the Emitter Module. When new data is received from the data source worker (via the JMS topic), it is sent to the presentation worker’s aggregation module. When a set amount of time elapses or a specified number of messages are received by the presentation worker’s Aggregation Module (configured by the client when it logs in), the messages are passed to the Transformation Module. The transformation module alters the format of the message, changing it, for example, to COT or KML. As the messages are transformed, they are passed to the Emitter Module, which makes the messages available to the client. In the current prototype, the Emitter Module may send the messages to a COT router, a JMS topic, or make them available for collection from an HTTP endpoint.
The top-most layer, the **presentation layer**, contains the applications that allow users to build, visualize, and share user-defined pictures. There are two classes of applications: **authoring** tools that allow users to create new UDOP products and publish them to the UDOP repository, and **viewing** applications that allow users only to subscribe to and visualize existing UDOPs. It is envisioned that there will be two broad classes of users for the UDOP prototype: action-officer level people who have the responsibility to build and manage UDOP products, and senior decision-makers who may prefer a simpler user interface for visualizing available situation awareness products.

### 4.2 Key Design Patterns

In implementing the UDOP prototype, several key design patterns were used. These design patterns provide specific, concrete methods to implement the design tenets and principles discussed above.

The design patterns that were leveraged in the building of the UDOP prototype include:

- **The Observer** (Gamma *et al.*, 1995) pattern, where an observer class subscribes to a subject class, eliminating one-to-one message flows between objects
- **The Messaging** (Hohpe & Woolf, 2004) pattern, which provides a mechanism for communicating, in an asynchronous fashion, packets of information to multiple receivers
- **The Proxy** pattern (Gamma *et al.*, 1995), where a proxy object provides intermediary access for a delegate.
- **The Adapter** pattern (Gamma *et al.*, 1995), where an adapter class is used to transform the interface and output of one class to another type.
- **The Strategy** pattern (Gamma *et al.*, 1995), where a specific set of algorithms or classes is selected at run-time, instead of being selected at design or implementation time.
- **The Aggregator** pattern (Hohpe & Woolf, 2004), where individual messages are collected by an aggregator object and published as a single, integrated message to a subscriber.

The remainder of this subsection details the user of these patterns.

#### 4.2.1 Observer Pattern

To eliminate tight coupling and one-to-one messages within the prototype architecture, the *observer* pattern is used extensively. The observer pattern describes a method of interaction where one or more listening objects are invoked when a publisher sends a notification message.

The advantage of the use of such a pattern is that it eliminates the design-time coupling between publishers and subscribers – subscribers are added dynamically during the system runtime.

The observer pattern is used to inform clients of changes to the UDOP repository. When a new UDOP document is added or an existing UDOP document is modified the UDOP repository will inform all currently active clients of the change. By using this pattern, clients receive fast notification of UDOP document updates, ensuring that they keep a synchronized view of their shared data artifacts.
4.2.2 Messaging Pattern

The messaging pattern provides a fire-and-forget mechanism for sending packets of information to different components. Messages are sent asynchronously, the sender does not wait for messages to be received. The messages are received by all listeners on a messaging bus, with the messaging bus being responsible for all subscription management issues. Within the prototype, this pattern is used for two major information flows.

First, when a data source worker publishes system of record information to the presentation worker, it is done via a messaging interface, implemented using the Java Messaging Service (JMS) capability built into the BEA WebLogic server. Each message from the data source worker is published to a single topic, and contains a correlation identifier specifying which presentation worker should handle the message. When the presentation worker receives the message, it checks the correlation identifier and processes it, if appropriate, or discards it.

Second, as new system of record data comes through the presentation worker and is sent to the clients, a messaging interface is also used, but over different technologies. In the case of some clients, like Google Earth or a web-based client, the client will poll a specified HTTP endpoint for changes. This endpoint is populated by the presentation worker, and is decoupled from the client, thereby still acting as a message interface. One type of consumer may use the User Datagram Protocol (UDP) to receive messages, while another would use a temporary JMS topic, created at run-time, to receive these messages. By incorporating a messaging pattern, instead of wholesale adopting a single technology (e.g., JMS), the system is sufficiently agile to adapt to new technologies as they become available or necessary.

4.2.3 Proxy Pattern

The use of the proxy pattern within the UDOP prototype hides some of the underlying complexity of enabling a net-centric data consumer, and is leveraged in the data source adapter / data source worker pairing. A good deal of machine-to-machine interaction goes into getting data from the system of record to the presentation worker. The use of the proxy pattern encapsulates the underlying complexities of this interaction through a single interface, and allows the underlying machinery to be modified and extended without modifying the consuming client or related interfaces.

The data source adapter provides an interface for the presentation worker to manage a subscription to a system of record. This interface takes a uniform set of parameters for a subscription request and passes them to the data source worker, the object its proxied object, which handles the complexity of interacting with and interpreting the results from the system of record. Similarly, on unsubscribing from the system of record, the data source adapter accepts the unsubscribe request and hands it to the worker, who interacts with the system of record to stop any ongoing processes and ensure that there is a clean shutdown.

4.2.4 Adapter Pattern

To reduce the complexity for external data interfaces into the UDOP prototype, it uses an adapter approach that presents a uniform interaction interface to external data sources. The adapters expose a service oriented interface, and they will be discoverable in the data source registry that is used by the authoring environment. The adapters provide a uniform approach for specification of data filtering parameters, access protocols, and output data formats, and
encapsulate the complexity and diversity associated with individual data sources that are the basis for the actual content that is contained in the UDOP document. Thus from the UDOP prototype’s perspective all data sources will look largely the same, the differences being in supported filter parameters and output data formats. These adapters perform several key functions:

- Handle the source-specific interactions associated with each data source. This requires components that components of the adapter module to be tailored on a per-source basis.
- Emit data in a common vocabulary (or small set of common vocabularies), so that the UDOP prototype is not required to understand all possible data formats. Ideally, upstream data sources would use common vocabularies themselves. However, as a practical matter, if they do not yet support such vocabularies, the adapters can perform necessary format mediation.
- Normalize the interaction interface as seen by the UDOP services, so that they do not have to support the spectrum of possible service-oriented interaction models.

4.2.5 Strategy Pattern

Given the complexity of consuming data from a large number of net-centric sources, none of which agree on a data format or access method, the strategy pattern was used to build the presentation worker’s presentation pipeline when the client first logs into the UDOP system. At login the client supplies a configuration message to the presentation service, which determines how each components of the pipeline is configured and the selected implementation.

As each client may have different refresh desires or capabilities, the aggregation module is configured with an aggregation timeout and threshold when it is instantiated. The aggregation module publishes a collected set of messages to the transformation module when either a number of messages that exceed the threshold has been received or the aggregation timeout expires, whichever happens first.

No UDOP client will handle every possible data format exposed by the different systems of record. The transformation module provides a place in the presentation pipeline for data coming from the systems of record to be alerted to so that the client can interpret them. The transformation module is instantiated based on the type of data formats the client can support.

Finally, since different clients may need to collect data through different mechanisms, i.e., one client would want to get data through an existing cursor-on-target router, the emitter module is selected to handle the egress of data to the client using the client-selected method.

The use of this pattern eliminates the need to create multiple, competing system configurations or to try to fit all the clients into a single interaction style. Having the system handle the processing of system of record data decouples the client from the variety of systems of records.

4.2.6 Aggregator Pattern

As the presentation worker processes each message from the data source worker, it is not always the case that the client or the rest of the presentation worker modules should immediately act on it. A message may need to be processed as part of a larger set of messages. For example:

- A set of messages may be changing so rapidly that it is not advantageous to send each message to the client, such as in the case of a rapidly moving object
• A set of messages may need to be processed together; such as in the case of collapsing a set of air units into a single squadron.

• The client may not be able to process large number of messages in a reasonable amount of time.

The aggregation module, as part of the presentation worker, receives messages from the data source workers and collects them until certain parameters are met – either a timeout occurs, meaning that the client has not received any data recently, or a certain number of messages have been collected. Once one of these events occur, the collected set of messages are sent to the transformation module (and the emitter module, afterwards) for consumption by the client.

The use of aggregation ensures that messages can be processed in unison, if desired, and do not overwhelm the client, which may not be able to handle rapidly changing sets of data in some cases.

5. Summary

As more mission-oriented data becomes available by net-centric means, it becomes increasingly critical to establish how to build systems that can visualize, transform, and support sensemaking over that data. Building agile net centric consumers of this data requires definition of design approaches that represent a departure from the techniques used in legacy “stovepiped” systems. Agile net-centric consumers such as the UDOP prototype discussed in this report allow users to sculpt tailored views of the operational environment that reflect their current needs, roles, and responsibilities. They require flexibility and customizability in the way they access, process, and visualize available data.

6. Recommendations

Based on the insights realized through this effort, we make the following two recommendations for improving the state of the practice in the development of agile net-centric applications:

1. **Employ loosely coupled architectural principles for building net-centric consumer applications.**

The architecture presented in this report provides well-defined and strict boundaries between components with well-defined flows of information and control defining the component interactions. This architectural approach facilitated integration and parallel development of interacting components. Such an approach will be critical in development of large-scale interoperable systems, where individual components can evolve and improve independently of the others. A critical element of such an architectural approach is to agree upon the respective responsibilities of producer, consumer, and mid-tier business logic functionality. The boundaries between them are not always self-evident, and require careful thought and planning.

An effective loosely coupled architecture should give consideration to the following:

- **Decomposition of distinct functional elements at an appropriate level of granularity.** An overall application should be decomposed into key functional elements that operate in cooperative fashion, with clear specification of the boundaries and information exchange protocols between communicating elements.
- **Separation of generic and specific functional elements.** Generic elements define behaviors that may be independent of the particular type of data flowing through them, while specific
elements take into consideration those aspects that cannot be decoupled from the type of data or process. Such a decomposition makes it possible to extend system functionality more readily.

- **Separation of core business logic from implementation-specific details.** Any net-centric application will ultimately be deployed into a specific implementation environment. Such tools should be designed to separate the functionality that depends on the implementation environment (e.g., application server, database) and the core business logic that is independent of that environment. Such separation facilitates migrating the application to different deployment environments as scalability and interoperability requirements evolve.

2. **Define strategies for normalizing interaction with disparate data sources.**

Numerous efforts are underway to define standardized schemas for expressing C2 information. Equally important is the question of interacting with data sources, in terms of specifying filter parameters, subscription models, and workflows for identifying and retrieving data of interest. Even with a standardized data representation, there are many technically valid ways to expose and access that data. Just as convergence on community-of-interest (COI) schemas simplifies the data representation problem, convergence on interaction models to support an ensemble of well-understood use cases can simplify the interaction problem. Incorporating such concerns can help to make exposed data *useful* to consumers. Interaction patterns include consideration of:

- **Workflows for data retrieval.** Simple workflows entail a single exchange between requestor and source to retrieve data of interest, although that exchange may entail complex processing on the source’s part. Multi-part workflows are a composition of simple workflows, which require multiple steps and supporting data to retrieve data. For example, the first step may be to retrieve a list of available resources or data products available, followed by retrieval of that specific source.

- **Interaction style.** Either simple or complex workflows may be associated with request/reply or publish/subscribe interactions. They are fundamentally different from one another and not functionally equivalent. Depending on the use case, there may be very good reason to use one over the other. A variety of mature and reliable technologies exist to support either style, and each has their strengths and weaknesses.

7. **References**


