Design Patterns for Net-Centric Applications

Seth Landsman, Ph.D.
Sandeep Mulgund, Ph.D.
The MITRE Corporation
Bedford, MA 01730
Overview

- Introduction

- Design Patterns
  - Data Interaction Patterns
  - Core System Design Patterns

- Example Implementation: User Defined Operational Pictures

- Summary
Introduction

- While making data available to other services is a key requirement for net-centric applications and services, it is not sufficient
  - Data needs to be synthesized into the larger C2 picture to be valuable
- What is the next step after making data available?
  - How do you move towards net-centric consumption of available net-centric data sources?
- Our work focuses on developing the design patterns that are associated with consuming and exploiting net-centric data
  - How do we partition a net-centric application so that it is not dependent on specific data formats and access methods?
  - How do we support multiple interaction styles for retrieving data?
  - How do we support different data formats that are produced by data sources?
  - What are effective strategies for managing information and control flows within a net-centric application?
Design Patterns

- These design patterns are recurring solutions to software design patterns we see in the construction of net-centric consumer applications

- Data Access Patterns
  - How is data obtained from the system of record into the net-centric application
    - Request/Reply
    - Publish/Subscribe
    - Multi-part workflows

- Core Integrating Design Principles
  - What are the core principles that enable the construction of a net-centric application
Design Patterns: Data Access

- **Publish / Subscribe**
  - Ongoing poll of a data source by the client
  - Client determines parameters of the connection and occasionally requests new data
  - Examples: JMS, RSS feed, WS-Messaging

- **Request / Reply**
  - One-off request from the client service to retrieve data
  - Examples: SOAP, RPC, REST / HTTP
Design Patterns: Data Access

■ Multi-Part Workflow
  – Combination of request / reply and publish / subscribe requests
  
■ Example:
  – Client request list of available data items from a service
  – User chooses one or more data item to subscribe to
  – Client subscribes to selected data items
Core Integrating Design Principles

- Loosely Coupled Architecture
  - Create strict boundaries and clear interfaces between functional components

- Asynchronous Data Transmissions
  - Use messaging to eliminate dependencies on timely component response

- Localized dependencies on Specific Data Formats
  - Choose a handful of key data formats that can be leveraged within the system and the system can transform system of record data into

- Encapsulate Message Flows With Schemas
  - Use well understood data formats for message flows, allowing new components and new clients to be added

- Leverage COTS and Localize Dependencies
  - COTS products may provide useful functionality, but any use of these additional features should be localized
Example Implementation: User Defined Operational Pictures

- User Defined Operational Pictures (UDOP) is a C2 system to create, visualize and share decision-focused views of the operational environment
  - Based on the need to support accurate situational awareness and timely decision-making in a distributed C2 environment
Example Implementation: User Defined Operational Pictures

- Three layer architecture, *information infrastructure, UDOP services*, and *presentation applications*
  - Well defined connections between each layer
# UDOP Key Design Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>Application in UDOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messaging</td>
<td>Asynchronous inter-component communication</td>
<td>Communication between data source adapter to presentation worker and presentation worker to clients</td>
</tr>
<tr>
<td>Observer</td>
<td>Observer subscribes to a subject, eliminating one-to-one messaging flows</td>
<td>UDOP repository changes are sent to clients</td>
</tr>
<tr>
<td>Proxy</td>
<td>Proxy object provides intermediary access for a delegate</td>
<td>Management of data sources is proxied through the data source adapter</td>
</tr>
<tr>
<td>Adapter</td>
<td>Adapter transforms the interface and output of a class</td>
<td>Adapt data sources into a common interface and emit UDOP data in a common vocabulary</td>
</tr>
<tr>
<td>Strategy</td>
<td>Algorithm or class is selected at run-time</td>
<td>Construction of a client-specific presentation pipeline</td>
</tr>
<tr>
<td>Aggregator</td>
<td>Individual messages are collected and published as a single, integrated message</td>
<td>Collect data source adapter messages to avoid overwhelming the client</td>
</tr>
</tbody>
</table>
Summary

- More mission-oriented data is becoming available through net-centric data sources
  - Ensuring that systems can visualize, transform, and support sensemaking over that data is becoming increasingly critical

- A type of design approach is needed that is different than the traditional “stovepiped” system methods

- Our recommendations for developing net-centric consumer applications are:
  - Build a loosely coupled application with clear data and control flows
  - Normalize the interaction with data sources, reducing the impact of new access methods or data formats on the rest of the net-centric application
BACKUPS
Recommendation 1: Employ loosely coupled architectural principles

- Loosely coupled architectures lead cheaper scaling, maintenance, and modification

- A loosely coupled system has two key properties
  - Well defined flows of control and information defining component interaction
  - Strict and enforced boundaries between components

- An effective loosely coupled architecture should
  - Decompose distinct functional elements at an appropriate level of granularity
    - Well defined boundaries and flows between components
  - Separate generic and specific functional elements
    - Decouple and re-use the generic components
  - Separate core business logic from implementation-specific details
    - Separate the “special sauce” of the deployment environment to avoid being tied to a specific application server / database / etc
Recommendation 2: Define Strategies for Normalizing interaction with data sources

- Even with standard data representations, there are numerous, disparate ways to access and interact with data.
- A net-centric consumer needs to limit the impact of supporting an increasing number of data sources with different:
  - Access methods
  - Filter parameters
  - Data formats
- An effective data consumer should consider:
  - Workflows for data retrieval
    - Decomposition of complex, multi-part workflows into a series of simple workflows
  - Appropriate use of different interaction styles
    - Request/Reply and publish/subscribe workflows are not equivalent, and should not be used interchangeably