SERVICE-ORIENTED ARCHITECTURE
ISSUES AND SOLUTIONS FOR MISSION CRITICAL APPLICATIONS

GSAW 2009
23-26 March 2009

Steve Kowalski, Ph.D.
steve.kowalski@ngc.com
SOA Challenges for Mission Critical Applications

- Performance with respect to real-time and data-intensive systems
  - Transport, Data Representation

- Coupling of service providers and requestors
  - Variations in degree of coupling

- Quality of service (QoS) considerations
  - Latency, loss, jitter, fault recovery

- Support for publish/subscribe models of component interaction
  - Beyond request-response message patterns

- Levels of interoperability
  - Technical, semantic, and business process interoperability
In a ground system, information is often exchanged between components (e.g., services, applications) in the form of messages. Latency of message transfer is often a derived ground system requirement. Message transport mechanism and data representation are two architectural aspects that affect latency. Message length is an important consideration in selection of transport mechanism and data representation.

Transport Mechanism Dominates

Shorter Message Length

Data Representation Dominates

Longer Message Length
Performance Considerations - Transport

Latency is an important consideration when selecting a transport mechanism but is not the only consideration.

<table>
<thead>
<tr>
<th>Common Transport Options *</th>
<th>Latency (rough relative order of magnitude) for very short (i.e., ping) messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services (HTTP/SOAP)</td>
<td>50x</td>
</tr>
<tr>
<td>Session EJ B</td>
<td>8x</td>
</tr>
<tr>
<td>Messaging middleware</td>
<td>5x (guaranteed message delivery using persistent queues can increase latency ~50%)</td>
</tr>
<tr>
<td>REST</td>
<td>4x</td>
</tr>
<tr>
<td>Data Distribution Service (DDS)</td>
<td>1x</td>
</tr>
</tbody>
</table>

* Note: Northrop Grumman is developing other transport mechanisms more optimized for the challenging performance and interoperability needs of ground systems.
Performance Considerations – Data Representation/Conversion

Conversion performed during serialization/deserialization

<table>
<thead>
<tr>
<th>Conversion performed during serialization/deserialization</th>
<th>Serialization time for large messages (rough relative order of magnitude)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java → Serialized Object</td>
<td>1x</td>
</tr>
<tr>
<td>Java → XML (SAX)</td>
<td>3x</td>
</tr>
<tr>
<td>SDO → Serialized Object</td>
<td>4x</td>
</tr>
<tr>
<td>SDO → XML</td>
<td>8x</td>
</tr>
<tr>
<td>Java → JSON</td>
<td>17x</td>
</tr>
<tr>
<td>Java → XML (DOM)</td>
<td>41x</td>
</tr>
</tbody>
</table>

Large message conversion can introduce large latencies. XML provides good interoperability but there are alternatives to pure XML that provide lower latency.
Alternatives to XML

• Compressed XML
  - Use a lossless compression algorithm (e.g., gzip, bzip2) to compress XML messages prior to transport and decompress them when they are received
  - Smaller binary messages consume less network resources
  - Extra processing is needed for compression/decompression at the end points

• Pass a reference to data in a shared data store
  - How to maintain integrity between reference and data when data is modified or deleted?
  - Lifetime of shared data

• Native object formats
  - Need to consider platform/language differences between end points
  - OMG Data Distribution Service (DDS) is an open standard

SOAs can use data representations other than XML (sometimes they need to for performance reasons)
Quality of Service (QoS) Considerations

We consider QoS to include

- Latency
  - Commanding, telemetry, mission data processing & distribution, reports/notifications
- Data Loss
  - Dropped packets, best effort delivery
- Jitter
  - Important for certain types of streaming data like voice and video

QoS is important because

- The network is not always reliable
- Network bandwidth is not unlimited
- Processing resources are not unlimited
- Different applications require different levels of QoS

How to manage network resources to deliver the desired QoS?
# QoS Examples and Architecture Patterns

<table>
<thead>
<tr>
<th>Application</th>
<th>QoS Parameters</th>
<th>Architecture Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telemetry updates sent to HMI operator station</td>
<td>Loss, Latency</td>
<td>1. Best effort delivery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Discard aged data</td>
</tr>
<tr>
<td>Streaming data distribution to users (voice, video)</td>
<td>Loss, Latency, Jitter</td>
<td>1. Jitter buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Per-stream QoS control</td>
</tr>
<tr>
<td>External tasking requests sent to ground system</td>
<td>Loss, Latency</td>
<td>1. Gateway for throttling, message screening</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Perimeter guard</td>
</tr>
</tbody>
</table>
Coupling of Service Requesters and Providers

<table>
<thead>
<tr>
<th>More Tightly Coupled</th>
<th>Less Tightly Coupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous interaction (coupled in time)</td>
<td>Asynchronous interaction (decoupled in time)</td>
</tr>
<tr>
<td>RPC-style parameters bound to operation signature of service provider (procedure centric)</td>
<td>Messaging without programming language constructs (data centric)</td>
</tr>
<tr>
<td>Service requester has knowledge of service provider</td>
<td>Service requester and provider have no knowledge of each other</td>
</tr>
<tr>
<td>Request/Response interaction pattern</td>
<td>Publish/Subscribe interaction pattern</td>
</tr>
</tbody>
</table>

The degree of coupling in a SOA can vary widely depending on the design decisions made by the architect.
Publish/Subscribe Pattern is a Good Match for Event Driven Service Interactions

- Events can be externally triggered (e.g., task request received, sensor event) or generated by an internal service (e.g., out-of-bounds state condition, anomaly detection)

- Request-reply pattern is very different from publish-subscribe pattern
  - Publish-subscribe is a good match for event-driven and data-centric models
  - Web services are based on the request-reply pattern

When architecting the infrastructure of mission critical applications, it is important to consider the messaging patterns that must be supported.
Levels of Interoperability

- Technical (Protocol, Syntactic) Interoperability
  - The ability of two or more systems to exchange data and use information
  - Usually concerns protocols and infrastructure needed for protocols to operate as well as data formats, syntax, and encoding

- Semantic (Contextual) Interoperability
  - The ability of two or more systems to exchange information and have the meaning of that information automatically interpreted by the receiving system accurately enough to produce useful results, as defined by the end users of both systems

- Business Process Interoperability
  - The ability of services to be assembled into a workflow to deliver a business function

SOA interoperability considerations should extend past technical interoperability to include semantic and business process interoperability
Acknowledgements

• Thanks to Norman Eaglestone and David Cadmus of Northrop Grumman for reviewing and commenting on this material

• Thanks to Charles P. Brown of IBM for architecture discussions that improved the quality of this presentation