Composing and Decomposing
QoS Attributes for Distributed Systems:
Experience to Date and Hard Problems Going Forward
or How to Handle a 20 year problem in 2-3 year Increments

Dr. Rick Schantz
and a host of talented colleagues, past and present

Monterey Workshop
October 16, 2006
Brief Outline

• Context
• Examples
• Enabling Aspects, Building Blocks and Directions
• Hard Problems and Longer Range Issues, Looking Forward
Historical Context: Software Infrastructure Enables Application Capabilities

1950s 1960s 1970s 1980s 1990s 2000s

Application

Operating System

Network Protocols

App

OS

App

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

App

Middleware

OS

Application

Operating System

Network Protocols

Database Systems

System Development Environments

??

1950s  Fifty Years of Distributed Systems Software Architecture Evolution  2006+

Copyright 2006 BBN Technologies
Background: Underlying Forces at Work

- Everything is a computer
- Everything is a networked computer
- Everything is potentially interdependent
- Things connect to the real physical world
- Increasing heterogeneity, distance and mobility

Mission critical distributed systems will continue to be built, fielded (and vulnerable) with or without proper basis, understanding and tools.
Leading to Current Trends and Directions

• Need for Integrated/Managed End-to-End Behavior
  – Multi-dimensional
  – String & Aggregate
• Multi-Layered Architectures, Network-centric Services & Systems of Systems
• Adaptive Designs Over Widely Varying and Changing Configurations
  – Static → Dynamic
• (More) Advanced Software Engineering (trying to keep pace)
  – Methodologies, Processes, Tools, Complexity Management
Brief Outline

- Context
- Examples
- Enabling Aspects, Building Blocks and Directions
- Hard Problems, Looking Forward
Example DRE Application 1

C2

Collaboration Server

Virtual Target Folder

Quality of Service Management

Object Request Broker

Pluggable Protocol

Link 16 Interface Software

C2 JTIDS Terminal

Collaboration Client

Browser Application

JTIDS Controls & Displays

Quality Object Framework

Adaptive Resource Mgmt

Link 16 Interface Software

F-15 JTIDS Terminal

Link 16
Avionics Dynamic Mission Planning

A Net-meeting like mission replanning collaboration between C2 and fighter aircraft

QoS Techniques
- Tiling
- Compression
- Processor Resource Management
- Network Resource Management

Copyright 2006 BBN Technologies
Multi-UAV Reconnaissance and Situation Monitoring Requires Dynamic End-to-End QoS Management

End-to-End Objective-Driven QoS Management

- **Reconnaissance Mode**
  - Maximize area monitored
  - Sufficient resolution in delivered imagery to determine items of interest

- **Situation Assessment**
  - UAV observing item of interest provides high resolution imagery so that unfolding situation can be monitored, assessed, and acted upon

- **After Action Assessment**
  - UAV providing high resolution imagery until a human operator has determined that it is sufficient
  - UAV over item of interest must continue to provide situation assessment imagery

The challenge is to program the dynamic control and adaptation to manage and enforce end-to-end QoS.

Heterogeneous, shared, and constrained resources

Multi-layer points of view: System-view, goal-view, application-string view, local resource view

Goal-defined requirements and tradeoffs (e.g., rate, image size, fidelity)

Changing modes, participants, and environmental conditions

Radio links are statically allocated

Vehicle platforms are closed with statically scheduled tasks

Ground based CPUs have variable dynamic load

LAN/WAN links are shared with (a priori) unknown load

Example DRE Application 2
Demonstration Imagery Displays (Simulated C2 Receivers)

- Name, role and COI of the asset
- Color of the border reflects the role of the SimUAV
- Image size and rate are a result of QoS information management

Copyright 2006 BBN Technologies
# QoS Policies Used in the Demonstration

<table>
<thead>
<tr>
<th>Mission</th>
<th>Relative Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISR_COI</td>
<td>1</td>
</tr>
<tr>
<td>TST_COI</td>
<td>2</td>
</tr>
</tbody>
</table>

## QoS Constraints and Tradeoffs

<table>
<thead>
<tr>
<th>Roles</th>
<th>Relative Priority</th>
<th>Resource Needed</th>
<th>Quality of Information Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BW Needed (kbps) (Min-Max)</td>
<td>DiffServ Codepoint</td>
</tr>
<tr>
<td>SURVEILLANCE (ISR)</td>
<td>1</td>
<td>50-200</td>
<td>Best Effort</td>
</tr>
<tr>
<td>TARGET TRACKING (TT)</td>
<td>6</td>
<td>150-600</td>
<td>Expedited Forwarding</td>
</tr>
<tr>
<td>BATTLE DAMAGE ASSESSMENT (BDA)</td>
<td>4</td>
<td>300-400</td>
<td>Assured Forwarding</td>
</tr>
</tbody>
</table>
Build an information management system that can survive sustained attacks from nation-state adversary and complete its mission.

*Operate through attacks* by using a layered defense-in-depth concept:

- Accept some degradation
- Protect most valuable assets
- Move faster than the intruder
Architecting Survivability into Large Systems With Realtime Response

Detection and correlation
- Embedded sensors
- Mix of IDS and Policy violation
- Advanced, distributed correlation

Reliability requires architecting in multiple dimensions
Even more so, when the goal is to be resilient not only against errors, but also against attacks....

Layers of protection
- Both HW and SW
- Design Principles, Architectural constraints

Adaptive middleware
- Rapid and coordinated response
- Isolation, recovery, Graceful degradation

Adaptive response

Redundancy: No single point of failure in critical functionality
- Less susceptible to attacker's manipulation of environment

Weak assumptions
- Less susceptible to attacker's manipulation of environment

Diversity: Avoid common mode vulnerabilities

High barrier to intrusion

General principles for survivability
- Protect as best as possible
- Improve chances of detection
- Adapt to manage gaps

Copyright 2006 BBN Technologies
Brief Outline

• Context
• Examples
• Enabling Aspects, Building Blocks and Directions
• Hard Problems, Looking Forward
A Minimum Set of Capabilities for QoS
Adaptive Middleware

- Specify the QoS needs of a mission, application, system
- Measure the QoS available in a system
- Control knobs to achieve QoS
- Adapt to compensate for changes in QoS
- Mediate conflicting QoS requirements
Adaptive Behavior Enablers

CLIENT Delegate

Contract SysCond

SysCond

IDL STUBS

ORB IIOP

in args
operation()

out args + return value

OBJECT (SERVANT) Delegate

IDL SKELETON

OBJECT ADAPTER

Mechanism/Property Manager

Network

ORB IIOP
Use Cases
QuO Components Are Packaged into Reusable Bundles of “Systemic Behavior” Called Qoskets

- The Qosket encapsulates a set of contracts (CDL), system condition objects (IDL), and QoS adaptive behavior (ASL)
- The Qosket exposes interfaces to access QuO controls and information (specified in IDL)
- The Qosket separates the functional adaptive behavior (business logic) from the QoS adaptive behavior and the middleware controls from the QoS mechanisms
**Component-Based Adaptive QoS Frameworks**

- Complementary CCM and Qosket ideas
  - CIAO, Prism support static QoS
  - Qoskets support dynamic, adaptive QoS
- Lifecycle support for dynamic QoS
  - Information available at design, configuration, assembly, and run times
- Crosscutting QoS concerns become many distributed qosket components
  - Better assembly and transparency using CCM tools
  - QoS policies in components, containers, and ORBs working with dynamic contract-based QoS and resource management

**CORBA CCM**

- Component frameworks need static and dynamic QoS management to be suitable for DRE environments
- QoS encapsulation (e.g., qoskets) within a component framework (e.g., CCM) offers the potential for improved composition, transparency, and lifecycle support for QoS
Multi-Layer QoS Management Architecture

- **System and Mission Layer**
  - (High level goals, requirements, trade-offs)

- **Application String Layer**
  - (goals, requirements for an end-to-end application)

- **Resource Layer**
  - (Control and allocation of an individual resource)

- Mapping
- Control
Multi-Layer QoS Management Design

- **System Resource Manager (SRM)**
  - Knows mission goals and tradeoffs
  - Knows number and types of participants, roles and relative importance, and available shared resources
  - Produces policy defined for each participant

- **Local Resource Manager (LRM)**
  - Determines how to utilize allocated resources to meet mission goals
  - Configures and monitors QoS behaviors to enforce QoS policy

- **QoS behaviors**
  - Control and monitor individual resources or mechanisms, or adapt application behavior

- **System Repository**
  - Model of Shared System Resources, Participants

- **System Resource Manager**
  - Policy Status

- **Local Resource Manager**
  - Controller
  - QoS Predictor
  - Constraints QoS levels
  - Configure
  - Feedback

- **System Participant**
  - System Participant
  - Policy Status

- **Mission goals, requirements, trade-offs**

Copyright 2006 BBN Technologies
Resource Allocation Algorithms

- We prototyped a few (variations) allocating based on priority, role, weight, number of assets, and available resources.
- The current algorithm allocates one resource at a time.
- Still some hard problems in allocating for more than one resource at a time:
  - Multi-dimensional resource provisioning.
  - Capturing system dynamics (how changes in one part of the system affects other parts).
  - Measures of utility to drive resource allocation.

QoS Enforcement Algorithms

- We prototyped QoS predictor and mechanism algorithms for a few resources (e.g., bandwidth), datatypes (e.g., images), and techniques (e.g., pacing, compression, scaling).
Patterns of Component Composition

Hierarchical

Parallel

Sequential

Data Shaping QCs

Image Sender

Crop QC  Scale QC  Compress QC  Pacing QC  DePacing QC  Decompress QC  Image Receiver/Display
End-to-end QoS management must
- Manage all the resources that can affect QoS, i.e., anything that could be a bottleneck at any time during the operation of the system (e.g., CPU, bandwidth, memory, power, sensors, …)
- Shape the data and processing to fit the available resources and the mission needs
  - What can be delivered/processed
  - What is important to deliver/process
- Includes capturing mission requirements, monitoring resource usage, controlling resource knobs, and runtime reallocation/adaptation

Control and Monitor Network Bandwidth
- Set DiffServ CodePoints (per ORB, component server, thread, stream, or message)
- Work with DSCP directly or with higher level bandwidth brokers
- Priority-based (DiffServ) or reservation-based (RSVP)

Control and Monitor CPU Processing
- CPU Reservation or CPU priority and scheduling
- Have versions that work with CPU broker, RT CORBA, RTARM

Shape and Monitor Data and Application Behavior
- Shape the data to fit the resources and the requirements
- Insert using components, objects, wrappers, aspect weaving, or interceptors
- Library that includes scaling, compression, fragmentation, tiling, pacing, cropping, format change

Coordinated QoS Management

System resource managers allocate available resources based on mission requirements, participants, roles, and priorities

Local resource managers decide how best to utilize the resource allocation to meet mission requirements

Dynamic QoS realized by
- Assembly of QoS components
- Paths through QoS components
- Parameterization of QoS components
- Adaptive algorithms in QoS components
Integrating the Components

End-to-End Quality of Service

**Local resource manager** configures qoskets to enforce resource management
- Diffserv Code Point
- CPU reservation
- Rate, compression level, amount to scale or crop

Qoskets control resources and shape imagery

**System resource manager** determines allocation of resources to participants and roles
- Assigns a weight to each role based on its relative importance (from the blue controller)
- Divides the total amount of resources (e.g., Mbps or %CPU) by the number of participants in all roles multiplied by their weight to get a resource unit
- Each participant is allocated a resource unit times the weight of its role

**System resource manager** pushes policy to each participant (SimUAVs)
- Role
- Relative importance
- Resource allocation
- Min and Max allowed (from mission requirements)
Dynamic Reconfiguration

Mission Mode Changes and Reconfiguration Dynamic End-to-End QoS Management

UAVs send surveillance imagery

UAV observing situation sends higher quality images

UAV provides after action imagery

Surveillance imagery is displayed at ground display; One UAV is directed to provide persistent coverage of an item of interest

Human operator assesses the situation and determines whether action is required

SRM reallocates resources and pushes new policy to participants; LRM determines new adaptation and control.

Dynamic End-to-End QoS Management

Surveillance imagery is displayed at ground display; One UAV is directed to provide persistent coverage of an item of interest

Human operator assesses the situation and determines whether action is required

SRM reallocates resources and pushes new policy to participants; LRM determines new adaptation and control.

Copyright 2006 BBN Technologies
Brief Outline

• Context
• Examples
• Enabling Aspects, Building Blocks and Directions
• Hard Problems, Looking Forward
Looking Forward Again:
Three Areas of Continued R&D

1. Heterogeneity is your friend, but is still too costly
   – On the one hand we often preach it,
   – But in practice we avoid it
   – Needs fixing, to avoid the risks of innovation slowdown & monoculture

2. Lot’s and lot’s of interacting pieces across platforms with realtime requirements over shared environments
   – Mandates new higher level abstractions for development and tools, tools, tools

3. Many of the distributed, realtime, embedded environments we engage have certifiability requirements
   – Current approach is completely static and exhaustive testing
   – Interconnection drives dynamic behavior which breaks current approaches
Modeling of QoS for DRE adaptive applications that behave appropriately under dynamic conditions

- Simplify the development of end-to-end and system-wide QoS adaptation and control for DRE systems
- Enable a larger class of practitioners, i.e., adaptive DRE system builders with less training in QoS or adaptive programming
- Improve the formalization, robustness, and usability of QoS adaptive concepts and applications.

Want to model and synthesize adaptive behaviors that

- Ensure predictable, controlled behavior
- Satisfy mission requirements (i.e., use requirements to choose suitable tradeoffs)
- Gracefully degrade and recover
### Observable and Controllable Parameters

<table>
<thead>
<tr>
<th>QosParams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
</tr>
<tr>
<td>CPUReservationLevel</td>
</tr>
<tr>
<td>CompressionLevel</td>
</tr>
<tr>
<td>FrameDataSize</td>
</tr>
<tr>
<td>FrameRate</td>
</tr>
<tr>
<td>FrameResolution</td>
</tr>
<tr>
<td>FrameSize</td>
</tr>
<tr>
<td>Importance</td>
</tr>
<tr>
<td>NumOfRSUV</td>
</tr>
<tr>
<td>NumOfSUAV</td>
</tr>
<tr>
<td>NumOfUCAV</td>
</tr>
<tr>
<td>Tiles</td>
</tr>
</tbody>
</table>

### Adaptation Behaviors

We have a set of off the shelf encapsulated behaviors (*qoskets*)

- CPU Broker
- Compression
  - Wavelet, JPEG, PNG
- Cropping
- Setting DiffServ Codepoints
- Pacing (changing frame rate)
- Scaling
- Tiling

### Two levels of resource management controllers

- A *system resource manager* that sets policy for participants
- *Local resource managers* that enforce policy
Adaptive Distributed Systems and Certification

- Adaptive Systems reallocate resources and change strategies at run-time
- provide system agility to tolerate
  - Failures, changing workloads, etc.
- tailors allocations and strategies to current conditions
- cannot deploy adaptive systems unless certified
- static approaches are generally not feasible
  - cannot always know worst-case
  - allocation is not fixed
  - too many choices and possibilities to analyze statically and to largely rely on testing
- need some new approaches to certification
Investigating Ideas Toward the Certification of Dynamic Systems

• To facilitate certification, the process of creating dynamic systems needs to include two related pieces
  – The **ability to assess** the comprehensive quality, reliability, and correctness of system behavior
  – The **ability to** (positively and assuredly) **control** system behavior

• These two go hand-in-hand for certification
  – Ability to drive the system toward assessable “good” behavior
  – Ability to prevent the system from incorrect, unreliable or “bad” behavior

• These two together can potentially provide a **basis for evidence-based certification**
Utility-Based Certification

- Utility measures can capture attributes of system performance and quality
  - Measure user-perceived value derived from control
  - Provide a quantitative measure for certification

- Considering Utility as measured and computed at each level of abstraction

  System utility:
  \[ U = \sum_{i=0}^{M} w_i U_i^m \]

  Mission utility:
  \[ U_i^m = \sum_{j=0}^{S_i} w_i^{S_j} U_i^{S_j} \]

  String utility:
  \[ U_i^{S_j} = \frac{1}{P_j} \sum_{i=1}^{P_j} u_i^{job} \]

  Job utility involves combinations of continuing service and meeting deadlines, ... 

- Feedback control uses utility measurements/estimates to drive toward higher (increasing) utility
  - Allows system to dynamically respond to unforeseen situations