A Real-Time Publish-Subscribe Control Plane for a COTM Node

HPEC
24 September 2008

J. Darby Mitchell
Software Architect
Wideband Tactical Networking
MIT Lincoln Laboratory
Outline

• Introduction
  – Assumptions
  – Requests

• Problem Statement
  – Project Vision and System Context
  – System Architecture
  – Software Architecture Problem

• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View

• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions

• Conclusion
  – Lessons Learned
  – Acknowledgements
Outline

• Introduction
  – Assumptions
  – Requests

• Problem Statement
  – Project Vision and System Context
  – System Architecture
  – Software Architecture Problem

• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View

• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions

• Conclusion
  – Lessons Learned
  – Acknowledgements
Assumptions and Requests

• Assumptions:
  – You know what MIT Lincoln Laboratory does
  – You recognize the value of buying vs. building software
  – You know that there’s no such thing as a “silver bullet”
  – You are familiar with the concepts of call-return middleware
  – Many of you are familiar with real-time publish-subscribe

• Requests
  – If you’d like to discuss any of these assumptions, please talk with me offline
  – Please hold your questions until the end of the talk
Outline

• Introduction
  – Assumptions
  – Requests

• Problem Statement
  – Vision and System Context
  – System Architecture
  – Software Architecture Problem

• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View

• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions

• Conclusion
  – Lessons Learned
  – Acknowledgements
Vision: Evolution of Terminal to Node

- **Milstar On-The-Move (MOTM) Terminal**
  - 3-axis positioner (MITLL)
  - Multi-band antenna and feed (44/30/20 GHz)
  - Blockage mitigation technology for COTM
  - IP over Milstar capability
  - Single link capability

- **COTM Node**
  - Manage multiple links
  - Compose links from modular HW/SW components
  - Facilitate integration of “stovepipe” COTS radios
  - Dynamic routing for cooperative networking
  - Support for insertion of additional radios
System Architecture Concept

- **Network Agent**
- **Physics Package**
- **Black Network**
- **Red Network**
- **HAIPE**
- **Modem**
- **RF**
- **Non-reconfigurable link**
- **Reconfigurable Links**
- **Static Links**

Legend:
- Black: user data (IP/RF)
- Green: node control
- Blue: aperture pointing
Software Architecture Problem

Key Enabling Hardware Decisions:

- Separate control and data planes
- Switched Gigabit Ethernet CompactPCI backplane
- System boards are Intel x86 SBCs running Linux
- Modem boards shall be PPC running VxWorks
Outline

• Introduction
  – Assumptions
  – Requests
• Problem Statement
  – Project Vision
  – System Context
  – System Architecture
  – Software Architecture Problem
• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View
• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions
• Conclusion
  – Lessons Learned
  – Acknowledgements
Quality Attributes and Architectural Styles

• Essential Qualities
  – Predictability: Ability to anticipate task scheduling requirements
  – Timeliness: Ability to meet real-time constraints
  – Reliability: Ensures delivery of critical control data

• Desirable Qualities
  – Modularity: Facilitates decomposition and encapsulation
  – Extensibility: Facilitates addition of components (i.e. functionality)
  – Simplicity: Component development should be straightforward

• Architectural Styles

<table>
<thead>
<tr>
<th>Style</th>
<th>Example Design Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call-return</td>
<td>Client-server, forwarder-receiver</td>
</tr>
<tr>
<td>Implicit Invocation</td>
<td>GoF Observer, Publish-subscribe</td>
</tr>
</tbody>
</table>
Publish-Subscribe

- Subscribers register to collect *issues* to a particular Topic
- Publishers register to distribute *issues* to a particular Topic
- A Topic acts as a GoF Mediator to decouple Publishers and Subscribers

```c
struct Position {
    double latitude;
    double longitude;
    double altitude;
};
```

"VehiclePosition"

```c
struct Position {
    double latitude;
    double longitude;
    double altitude;
};
```

**Issue 1**

- longitude = -71.225
- latitude = 42.447
- altitude = 44.8
Publish-Subscribe

- May be zero or more publishers per topic
- May be zero or more subscribers per topic
Architectural Reasoning

Publish-Subscribe
- Ideal for one-to-many or many-to-many relationships
- Promotes predictability
- Data-centric (data identifier)
- No assumption of existence
- Data source always initiates communication

Result: decoupled interaction

Call-return
- Client
- Forwarder
- Data Flow
- Server
- Receiver
- Data Flow

Result: highly coupled interaction
Quality Attribute Tradeoffs

Publish-Subscribe

- Timeliness
- Predictability
- Modularity
- Extensibility

Call-return

- Reliability
- Simplicity
Real-time Publish-Subscribe with NDDS

- The Network Data Distribution Service is a real-time publish-subscribe middleware developed by Real-Time Innovations, Inc.
- NDDS was designed for distributed real-time systems.
- Provides a number of QoS settings to customize the collection and distribution of issues.
- At the time we selected the product, the OMG DDS specification was still being finalized.
- RTI was a significant contributor to the OMG DDS specification effort.
- RTI had plans to refactor NDDS to conform to the DDS specification.
- RTI had already published their RTPS wire protocol.

### Quality of Service Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>Presentation</td>
</tr>
<tr>
<td>Destination Order</td>
<td>Reliability</td>
</tr>
<tr>
<td>Durability</td>
<td>Resource Limits</td>
</tr>
<tr>
<td>Entity Factory</td>
<td>Time-Based Filter</td>
</tr>
<tr>
<td>History</td>
<td>Transport Priority</td>
</tr>
<tr>
<td>Latency Budget</td>
<td>Group Data</td>
</tr>
<tr>
<td>Lifespan</td>
<td>Topic Data</td>
</tr>
<tr>
<td>Liveliness</td>
<td>User Data</td>
</tr>
<tr>
<td>Ownership</td>
<td>Reader Data Lifecycle</td>
</tr>
<tr>
<td>Ownership Strength</td>
<td>Writer Data Lifecycle</td>
</tr>
<tr>
<td>Partition</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Introduction
  – Assumptions
  – Requests

• Problem Statement
  – Project Vision
  – System Context
  – System Architecture
  – Software Architecture Problem

• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View

• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions

• Conclusion
  – Lessons Learned
  – Acknowledgements
Designing Topics

• **Samples** – periodic, independent measurements of the environment
  Examples:
  – Vehicle position and velocity
  – Link state
  – Modem signal strength
  – Satellite location and velocity
  – UTC Time

- **Events** – sporadic, relative changes in system state
  Examples:
  – Link formation and teardown
  – Status messages
  – Error messages
  – Parameter changes
  – Routing changes

  RELIABILITY: BEST EFFORT
  HISTORY: KEEP LAST

  RELIABILITY: RELIABLE
  HISTORY: KEEP ALL
Topic Mapping

Key

Component
Sample Topic
Event Topic
Publication
Subscription

Position Service
LDR Adapter
Node Controller

AHRS Adapter
Tracker
Antenna Adapter

RF Interface Module
DTR Adapter

S1
S2
S4
S5
S6
S7
S8

E1
E2
E3
E4
E5
E6
E7
S9
**Samples**

S1 – UTCTime  
S2 – AHRSLocation  
S3 – AHRSDisplacement  
S4 – AHRSVelocity  
S5 – AntennaReferenceAngle  
S6 – AcquisitionMetric  
S7 – AntennaAngles  
S8 – DTRSamples  
S9 – LDREnergyMetric

**Key**

- Component
- Sample Topic
- Event Topic
- Publication
- Subscription
**Events**

E1 – DeviceStatus
E2 – TrackCommand
E3 – AntennaCommand
E4 – DTRParams
E5 – DeviceCommand
E6 – LDRCommand
E7 – RIMCommand

---

**Key**

<table>
<thead>
<tr>
<th></th>
<th>Component</th>
<th>Sample Topic</th>
<th>Event Topic</th>
<th>Publication</th>
<th>Subscription</th>
</tr>
</thead>
</table>

---

**Structures**

```c
struct DeviceStatus {
    string deviceId;
    int statusType;
    int code;
    string msg;
};
```

```c
struct DeviceCommand {
    string deviceID;
    int command;
};
```

```c
struct AntennaCommand {
    string deviceID;
    int command;
    double az;
    double el;
};
```
Outline

• Introduction
  – Assumptions
  – Requests
• Problem Statement
  – Project Vision
  – System Context
  – System Architecture
  – Software Architecture Problem
• Software Architecture
  – Quality Attributes and Architectural Styles
  – Architectural Reasoning
  – Quality Attribute Tradeoffs
  – Runtime View
• Design and Implementation
  – Designing Topics
  – Topic Mapping
  – Handling Exceptions
• Conclusion
  – Lessons Learned
  – Acknowledgements
Lessons Learned

• Using publish-subscribe:
  – Made component development slightly more complicated
  – Greatly facilitated software integration
  – Enabled us to successfully defer some components, while still making progress on the project
  – Is not as straightforward when you are marshalling parameters with commands

• Respect the invariants of the architectural style:
  – NodeController could be killed and later restarted with no detrimental impact to system in steady state
  – Debug topics could be published for later use with negligible impact on system performance

• Actively managing consistency of QoS settings was essential

• Having a commercial vendor to delegate middleware support concerns to was very helpful
Acknowledgements

• Sponsor: PM WIN-T, Ft. Monmouth

• Group Leaders: Dr. Marc Zissman and Scott Sharp

• Systems Engineer: Dr. Andrew Worthen

• RF team: Dr. Jim Vian, John Murphy, Ted O’Connell

• Hardware team: Steve Pisuk, John Delisle, Jason Hillger

• Software team: Darby Mitchell, Curran (Nachbar) Schiefelbein, Marc Siegel, Marie Heath

• Testing team: Dr. Mark Smith, Ted O’Connell
Current Work

• TSAT Reference Terminal (TRT)
  – A joint project with Group 64 based on TRUST-T
  – A COTM Node that is based on the Software Communications Architecture (SCA) for software defined radios.
  – The SCA mandates the use of CORBA middleware, so DDS will not be used.

• Network and Link Emulation Testbed (NLET)
  – A distributed network emulation testbed
  – Uses DDS for a distributed real-time context simulation and real-time dynamic control of link emulation.
References

  
  http://www.ll.mit.edu/news/journal/journal.html


Questions?

mitchelljd@ll.mit.edu
Backup Slides
Reasoning About Connectors

- Reasoning about connectors vs. components
- Consider several dimensions:
  - synchronous vs. asynch
  - cardinality (1:1 vs. 1:n)
- Ignore implementation concerns

Publish-subscribe =
Distribution + Implicit Invocation + Collection
System Architecture: Connection View

- **Milstar OTM Terminal Assembly**
  - Space Tracking Processor
  - Serial to Antenna Control Processor
  - Serial to GBS Beacon Receiver
  - Ethernet to GBS DVB Receiver
  - Ethernet to Multiband RF Assembly

- **Antenna Assembly**
  - Serial to Antenna / Positioner

- **GBS Modem Assembly**
  - Ethernet to GBS Beacon Receiver
  - Ethernet to GBS DVB Receiver

- **802.x Wireless Link Assembly**
  - Ethernet to Router
  - Serial to Multiband RF Assembly

- **User Laptop**

**Key**
- Processor
- Notional Assembly
- Control only
- Control & Data
Driver and Adapters

- There is a one to one relationship between Drivers and Adapters
- Node Controller only interacts with an Adapter through its Driver
- A Driver caches Status and Error updates from its Adapter
- Adapters may interact with other Adapters

Test Harness

- Test Case
- Driver
- Adapter
- Logger
- Commands
- Parameters
- Status
- Errors

Device
Exception Handling

- Based on concepts from online article:

- Added mechanism to throw exceptions up to C++ from existing C code
- Added mechanism to translate POSIX signals (e.g. SEGV) to C++ exceptions

--- Caught std::exception ---

--- at [ll/common/obj/x86-linux/TerminateHandler.cpp:44] ---
Type: SIGSEGV
Text: Received signal
Where: LL::Exceptions::SignalTranslator::SignalTranslator(int) [SignalHandler.cpp:122]
Trace:
(0) LL::Exceptions::SignalTranslator::SignalTranslator(int) [SignalHandler.cpp:125]
(1) LL::Exceptions::SignalHandler::handler(int) [SignalHandler.cpp:102]
(2) [libpthread.so.0]
(3) RtiThreadSleep [libutilsip.so]
(4) NddsUtilitySleep [libndds.so]
(5) LL::NDDS::Adapter::enterMainLoop(double) [NDDSAdapter.cpp:92]
(6) main [RimNDDSAdapter_main.cpp:96]
(7) __libc_start_main [libc.so.6]
(8) _start [??:0]

--- Caught LL::Exceptions::RTE ---

--- at ... not readable ---
Where: virtual void LL::SYS::SerialHWInterface::open() [ll/sys/obj/x86-linux/SerialHWInterface.cpp:195]
Trace:
(0) LL::Exceptions::RTE::RTE(LL::Exceptions::RTE::RTETypeEnum, std::string const&, std::string const&) [RTE.cpp:32]
(1) LL::SYS::SerialHWInterface::open() [SerialHWInterface.cpp:195]
(2) LL::SYS::SerialHWInterface::binaryInvoke(std::vector<unsigned char, std::allocator<unsigned char> > const&, int) [SerialHWInterface.cpp:128]
(3) LL::Devices::Rim::isOscillatorLocked() [Rim.cpp:86]
(4) LL::Devices::RimNDDSAdapter::sendDeviceStatus() [RimNDDSAdapter.cpp:99]
(5) LL::NDDS::Adapter::toggleDummy(bool) [NDDSAdapter.cpp:77]
(6) main [RimNDDSAdapter_main.cpp:96]
(7) __libc_start_main [libc.so.6]
(8) _start [??:0]

--- rethrown ---

[DeviceStatus/rim0] rim0 DS_ERROR EC_CRITICALERROR
(exceptionMsg=Type: CriticalError Text: /dev/ttyS2 is not readable Where: virtual void LL::SYS::SerialHWInterface::open() at
ll/sys/obj/x86-linux/SerialHWInterface.cpp:195
(8) [DeviceStatus/rim0] rim0 DS_ERROR EC_CRITICALERROR
(exceptionMsg=Type: CriticalError Text: /dev/ttyS2 is not readable Where: virtual void LL::SYS::SerialHWInterface::open() at
ll/sys/obj/x86-linux/SerialHWInterface.cpp:195
(guid=rim0)

WTN Node Controller
node0 : spirall

---
node0>
Type -> rim0: Type: CriticalError
Text: /dev/ttyS2 is not readable
Where: virtual void LL::SYS::SerialHWInterface::open() at
ll/sys/obj/x86-linux/SerialHWInterface.cpp:195
Trace:
(0) LL::Exception::char, std::allocator<
(1) LL::Exception::char, std::allocator<
(2) [DeviceStatus/rim0] rim0 DS_ERROR EC_CRITICALERROR
(exceptionMsg=Type: CriticalError Text: /dev/ttyS2 is not readable Where: virtual void LL::SYS::SerialHWInterface::open() at
ll/sys/obj/x86-linux/Sequential.cpp:195
(guid=rim0)