Integrity ★ Service ★ Excellence

Unmanned Systems Autonomy Services: Deep Dive

Dr. Derek Kingston
Control Science Center of Excellence
Aerospace Systems Directorate
Air Force Research Laboratory

Design Philosophy

- **Each service:**
  - **Is responsible for subscribing to and storing all data necessary for operation**
  - **Communicates with other services only via ZeroMQ**
    - **No back-channel communication, no shared files**
    - **Attempt to eliminate side-effects and unexpected interactions**
  - **Has well-defined behavior and a single role**
    - **Often acts in request/reply pattern**
    - **Stateless when possible, consistent output**
    - **Operates in a single thread if possible**

- **Q: Why not just a function call from a shared library?**

Design Philosophy

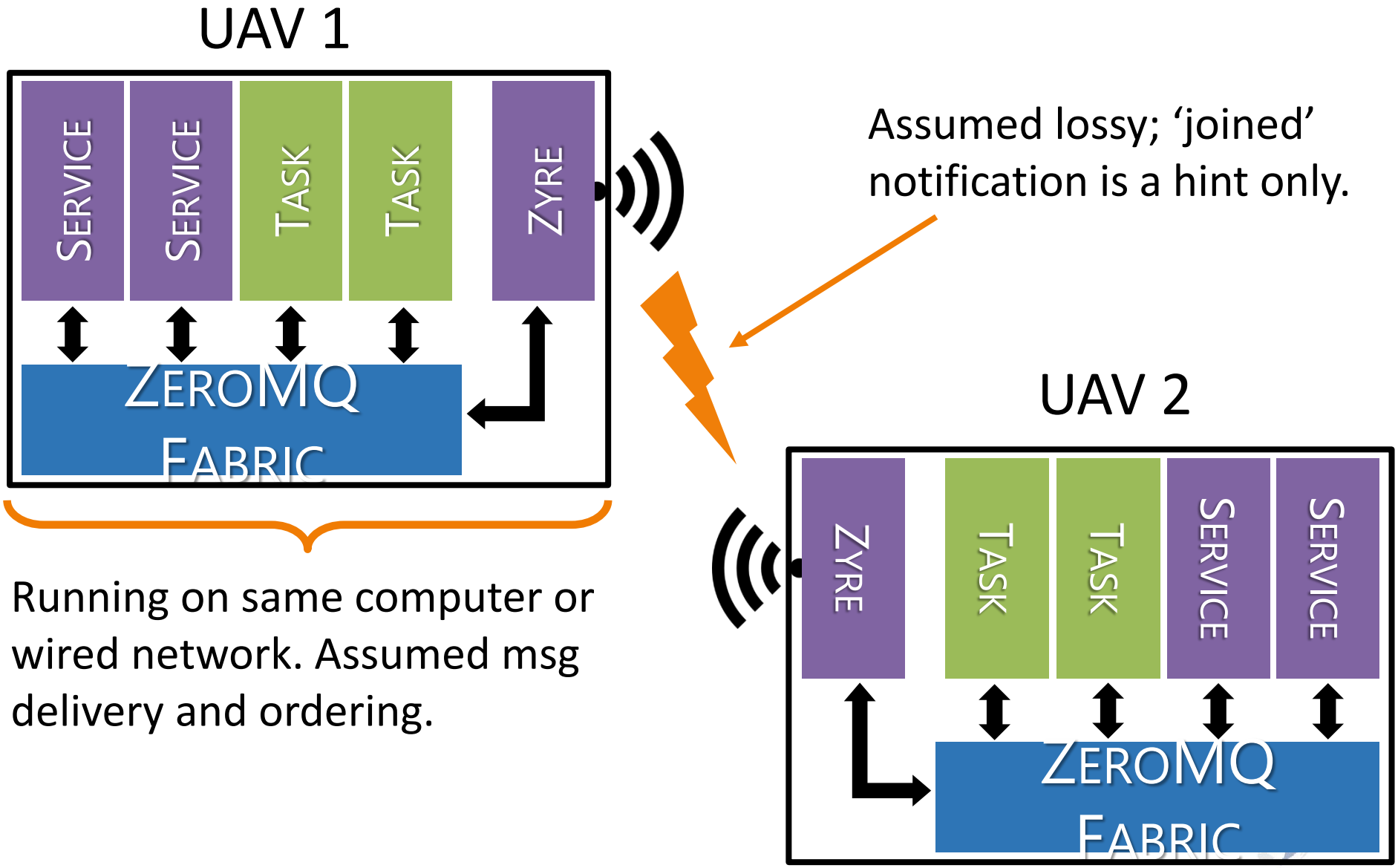
- When in doubt, err on side of separation
  - Allows access to functionality from external software
  - Provides encapsulated ‘block’ of functionality
    - Stated assumptions
    - Documented inputs and outputs
    - Possibility for re-use
    - Unit-testable
  - Easy to reconstruct execution trace from message traffic

- Separation is more important than efficiency
- Pre-mature optimization is discouraged

Design with Multiple Vehicles

- **A “local” UxAS instance can be relied on to be responsive and available**
- **Messages sent between vehicles are always subject to loss and any service relying on such messages must be tolerant/robust to loss**
- **Minimize data required for coordination**
- **Better to establish solid connection and conduct reliable exchange rather than operate at edge of connectivity**
  - **Also known as: be very cautious about exploiting uncertain communication models**
  - **Example: continue to fly towards a neighbor until data requirements are met rather than maneuver early to minimize time to next objective**

Multi-Vehicle Architecture



Middleware Considerations

- **To interconnect separated software components, need:**
  1. **Transport mechanism**
  2. **Content description**
  3. **Serialization (wire transfer format)**

- **Example: Garmin GPS receiver**
  1. **Transport: Serial @ 4800 baud**
  2. **Content description: NMEA 0183**
  3. **Serialization: ASCII characters with defined separators and checksum**

- **Example: Open Mission Systems (reference | vendor CAL)**
  1. **Transport: ActiveMQ | Proprietary to vendor**
  2. **Content description: UCI/UCS | UCI/UCS**
  3. **Serialization: XML strings | Proprietary to vendor**

Middleware Considerations

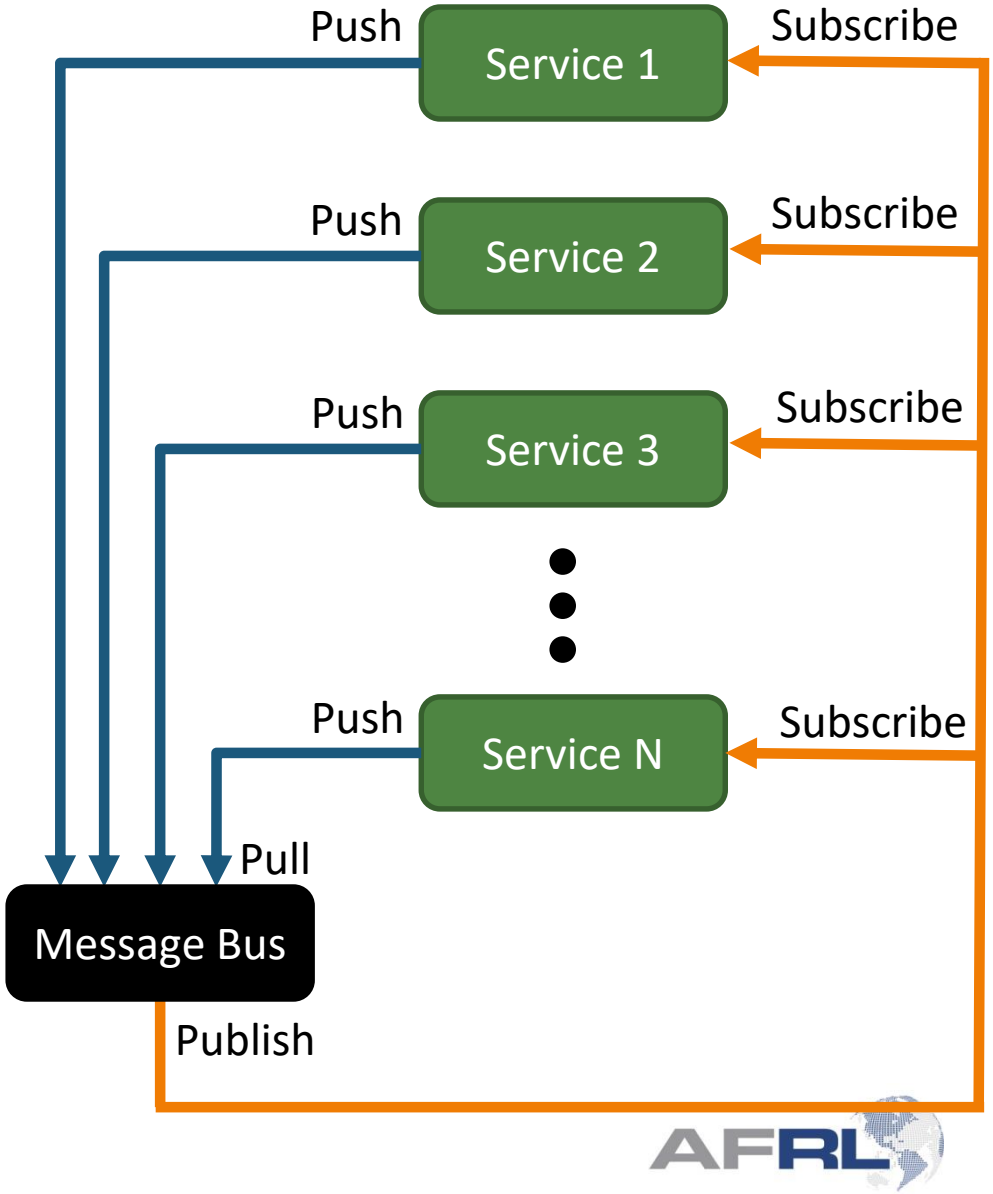
- **Example STANAG 4586:**
  1. **Transport: UDP/IP**
  2. **Content description: NATO STANAG 4586 Edition 4**
  3. **Serialization: byte packing according to STANAG 4586**

- **UxAS**
  1. **Transport: ZeroMQ**
  2. **Content description: LMCP MDM**
  3. **Serialization: LMCP**

- **Note: Lightweight Message Control Protocol (LMCP) is an AFRL custom design that works from an XML message description to produce libraries that have corresponding data structures and serialization procedures**

Pub/Sub Details

- **Uses two complementary ZeroMQ patterns:**
  - **Push/Pull channel for each service to send messages to the "bus"**
  - **Publish/Subscribe (fan-out) for services to be informed**
- **Subscription mechanism is left-to-right string matching**
- **By convention, each message type name is the subscription topic for receiving that message**



Message Envelope

- Each message consists of two major pieces:
  - Envelope
  - Content
- *Envelope* (also called message attributes) consists of:
  - Address [afrl.cmasi.AirVehicleState]
  - Message type [lmcp]
  - Description [afrl.cmasi.AirVehicleState]
  - Source Group [ ]
  - Source Entity ID [eid400]
  - Source Service ID [sid37]
- *Content* consisting of a single serialized LMCP message
- In ZeroMQ, this becomes a 7-part message:
  [address][type][desc][grp][eid][sid][content]
- For serial streams (RS232, TCP), this becomes:
  [address]$[type]|[desc]|[grp]|[eid]|[sid]$[content]

Routing By Addressing

- When ‘message type’ is ‘lmcp’ then the ‘description’ must be the fully qualified message name
- UxAS ignores all messages that are not of type ‘lmcp’

- A *broadcast* is when the ‘description’ field is equivalent to the ‘address’
  - This is the most common way to send messages
  - Request ID should be matched to Response ID (fields of messages using that pattern)
- A *limitedcast* is when a mailbox address is used instead of the fully qualified message name
  - Example: use ‘GroundPlanner’ as address to send only to ground vehicle route planners

Addressing Across Entities

- **Each entity (UAV, UGV, UGS, …) must have a globally unique ID**
- **Each service running on an entity (typically in a single UxAS instance) must have an integer ID unique from other ‘local’ services**
- **The *system* address for a specific services is then:**
  - **eid{entityID}sid{serviceID}**
- **All services that support a particular entity can be addressed by:**
  - **eid{entityID}sidall**

Corner Cases

- *Limitedcast* is used only in the case of route planners
  - Requests are made to classes of planners (e.g. air)
  - Replies from route planners are sent back directly to original requester
- Addressing entities using “eid{entityID}” is used during shutdown and in one internal AFRL experimental project

- Design choice: new message or *limitedcast* ?
  - Rather than address services directly or through shared mailboxes, often new messages are designed
  - Example: TaskAutomationRequest vs UniqueAutomationRequest
  - But now must document appropriate use situations

Examples

- **Subscribe to message**

```
addSubscriptionAddress(afrl::cmasi::KeepOutZone::Subscription);
addSubscriptionAddress(afrl::cmasi::KeepInZone::Subscription);
addSubscriptionAddress(afrl::cmasi::OperatingRegion::Subscription);

addSubscriptionAddress(afrl::cmasi::AirVehicleConfiguration::Subscription);
addSubscriptionAddress(afrl::impact::GroundVehicleConfiguration::Subscription);
addSubscriptionAddress(afrl::impact::SurfaceVehicleConfiguration::Subscription);
```

- **Return to sender**

```
auto routePlanResponse = std::make_shared<uxas::messages::route::RoutePlanResponse>();
if (ProcessRoutePlanRequest(request, routePlanResponse))
{
    auto message = std::static_pointer_cast<avtas::lmcp::Object>(routePlanResponse);
    // always limited-cast route plan responses
    sendSharedLmcpObjectLimitedCastMessage(
            getNetworkClientUnicastAddress(
                receivedLmcpMessage->m_attributes->getSourceEntityId(),
                receivedLmcpMessage->m_attributes->getSourceServiceId()
            ),
            message);
}
```

Questions?