OTF Proof of Concept:

CCSDS Mission Operations Alert Services

Walt Reynolds

February 20, 2009

CCSDS MO Proof of Concept

- Goals:
  - Demonstrate use of CCSDS MO (Mission Operations) standards to implement mission advisory services (alerts).
  - Utilize CCSDS MO MAL (Message Abstraction Layer) messaging.
  - Implement CCSDS AMS (Asynchronous Message Service) concepts for transport and publish/subscribe service.
  - Utilize the CCSDS MO Directory Service concept to register application agents.
  - Investigate CCSDS MO MAL data element definitions and usage
    - Network Zones, Sessions, and Domains

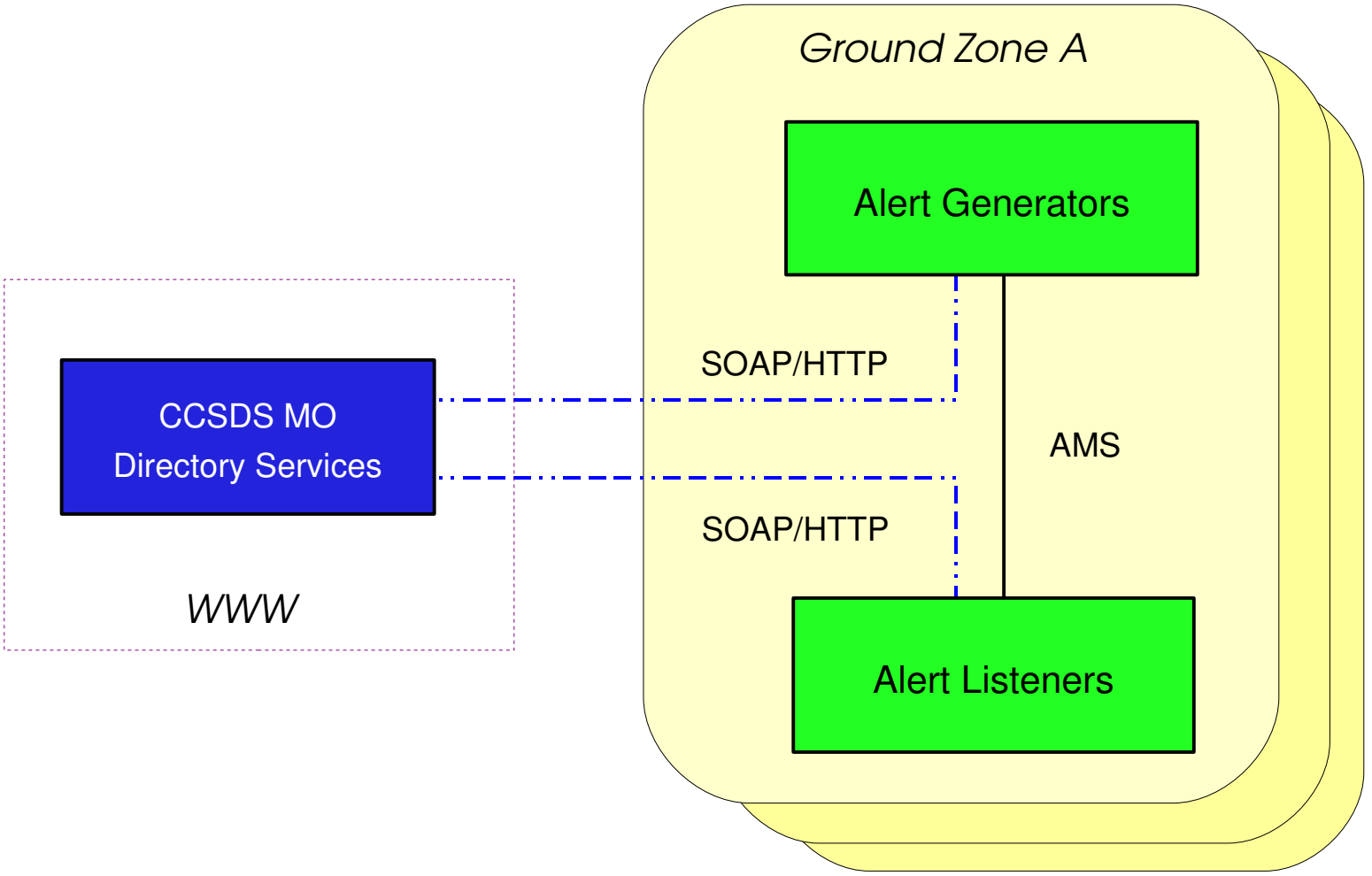
Project Benefits

- Benefits
  - Use of CCSDS standards encourages vendor investment and extends accompanying product life cycles.
  - Provides a growth path for new MCC applications.
    - MO Directory Services for flexible and dynamic application registration.
  - Extend the scope and flexibility of existing console flight data stream interfaces.
    - AMS transport contains its own metadata management.
    - Messaging reaches intra and inter control center applications.

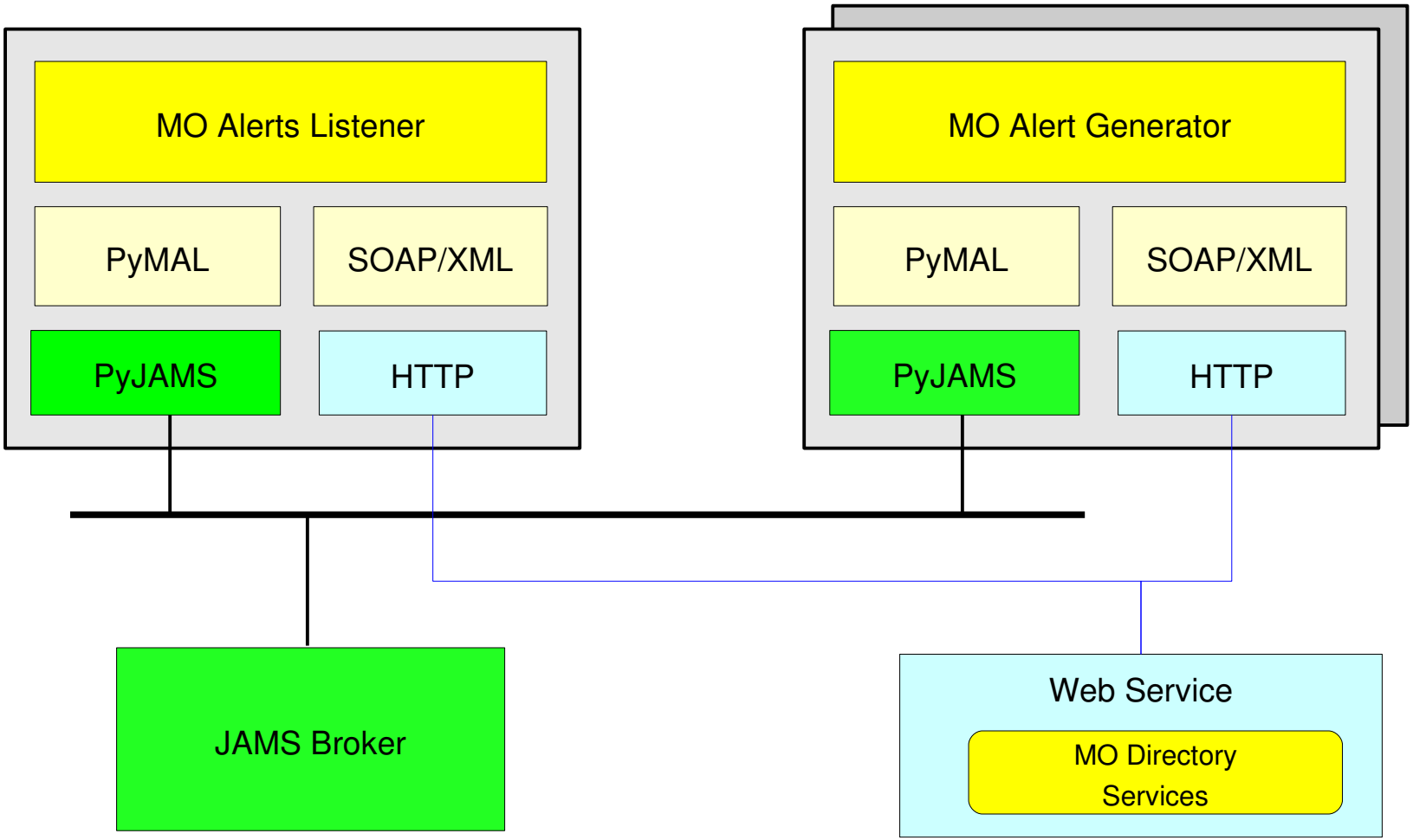
Project Description

- Project Scope
  - Does not address a security architecture.
    - No secure encodings or encryption systems.
    - No authentication (in work) or authorization (future work).
  - Does not implement or test all of Common or Core Services.
  - Performance is not a current requirement but may not preclude an efficient future implementation (some languages allow C/C++ extensions for efficient implementation: Python, Java).
  - Does not utilize OTF test flight data streams and interfaces (in progress or planned).

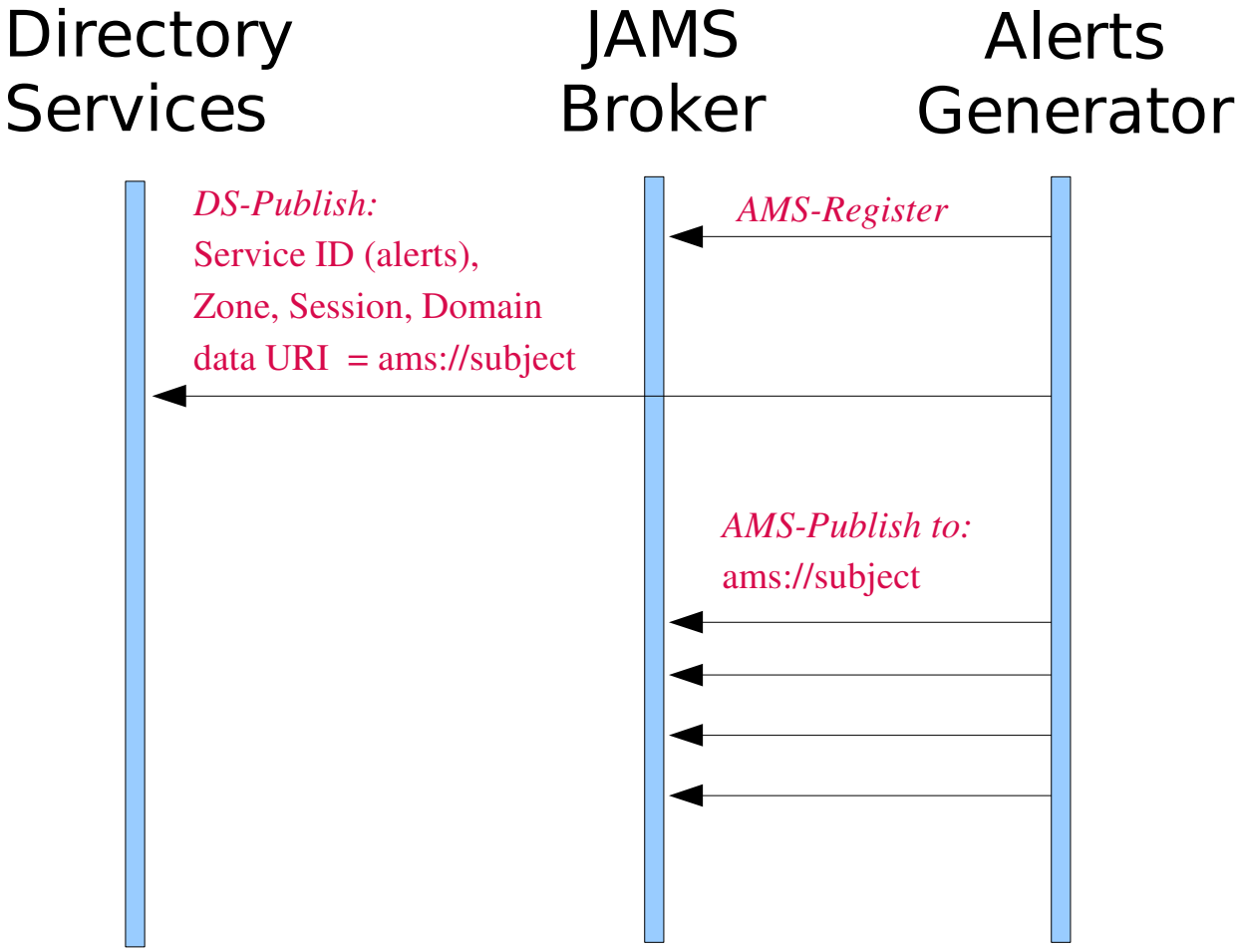
Proof of Concept Components



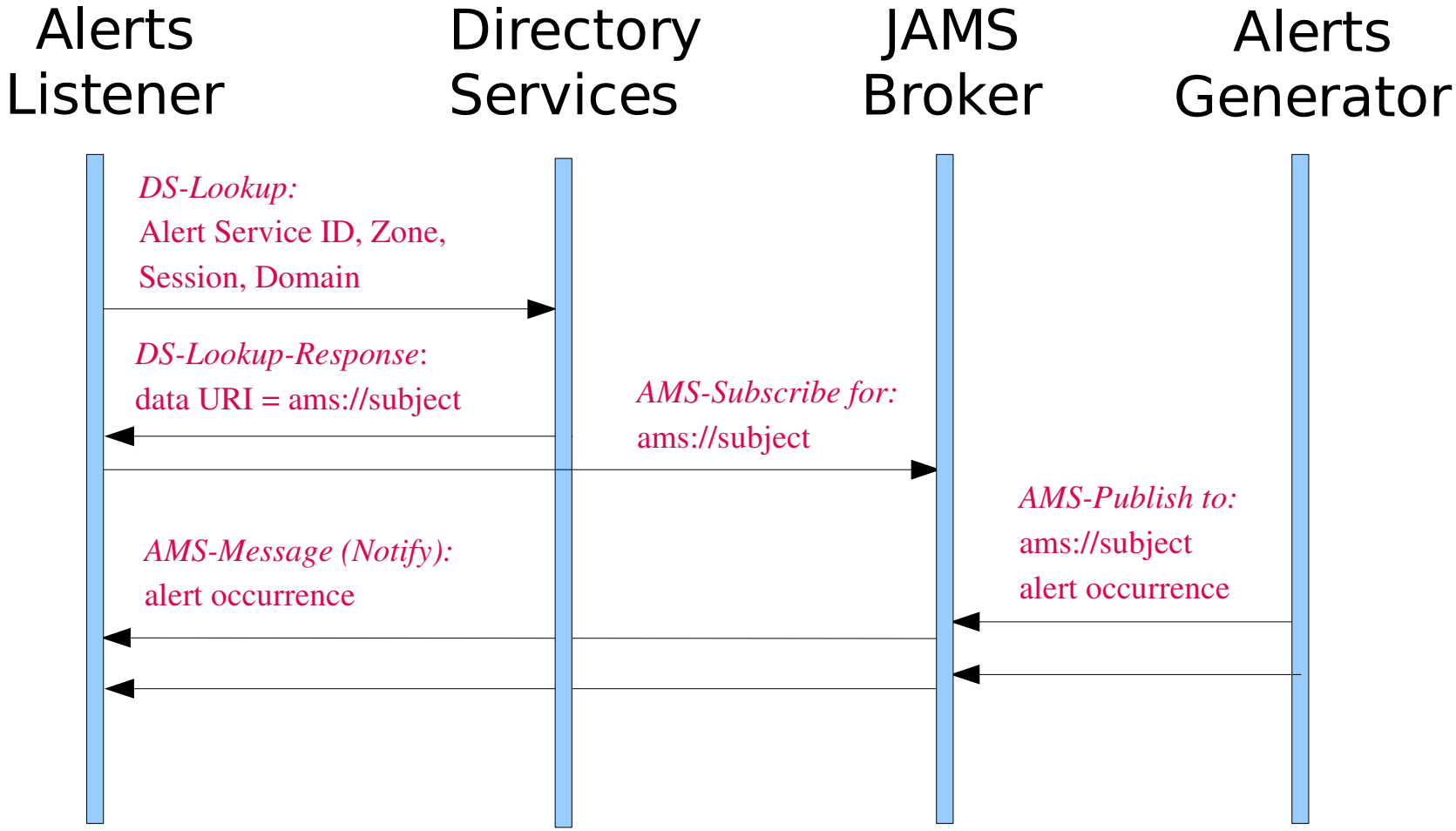
Implementation



Operational Message Flow



Operational Message Flow



Proof of Concept – Conclusions

- Conclusions:
  - Use of multiple vendor frameworks within a component creates thread lock-ups and fragility – conflicts over the control of the main thread.
    - Avoid closed vendor messaging frameworks.
  - The MAL layer *does* isolate the data elements from the messaging framework but application work dispatch is heavily dominated by the framework chosen.
  - API Language is a major factor in implementation.
    - Message APIs with timeouts seem unavoidable in some environments (GUI).
    - Language environment needs to support callbacks (or threads) from the messaging framework to deal with pub/sub management messages and status.
    - Statusing of application communications demand a local broker agent per physical system or else the use of the AMS-style registrar heartbeat.