Modelling of PnP Weapon Systems with AADL – Protocol Behaviour

A. Windisch and H. Schlatt
EADS Military Air Systems, Systems Engineering
81663 Munich, Germany
Contents

- Introduction
- Notational Issues and Modelling Approach
- The PnP Store Control Protocol
- The PnP System Architecture
- Conclusion
Military Air Systems

Introduction – Aircraft/Store

Aircraft

Mission

Store

ASI

MSI
Introduction – Aircraft/Carriage Store/Store

Aircraft
- ASI

Carriage Store
- CSSI

Mission Store
- MSI
- MSCI

Miniature Mission Stores
Contents

- Introduction
- Graphical Notation and Modelling Approach
- The PnP Store Control Protocol
- The PnP System Architecture
- Conclusion
Graphical Notation - Components

Software
- Data
- Thread
- Process

Platform
- Device
- Memory
- Processor
- Bus

Composite
- System
Graphical Notation - Features

Ports

- Data Port
- Event Port
- Event Data Port

Subprograms

<name>

Connections

- Immediate
- Delayed

Client / Server

client → server
Modelling Approach

- The PnP technical architecture defines several protocol layers which provide the means for network transparent control and communication between the aircraft and the weapon systems.
- A concise system model should reflect this generic hierarchical layered communication infrastructure and not put any constraints on the actual physical architecture of the aircraft/weapon system.
- Modelling approach suggested in this presentation is based on abstract buses (protocols) and abstract processors (virtual machines) which are currently developed as part of the AADL.
Contents

• Introduction
• Notational Issues and Modelling Approach
• The PnP Store Control Protocol
• The PnP System Architecture
• Conclusion
• Dynamic connectivity, i.e. air data link used for communication with weapon after release
• Can be modelled in terms of modes (omitted in graphics)
• Communication between system functions is based on abstract store control protocol which provides API
Functional Architecture (AADL)

```plaintext
subprogram group StoreControlAPI
  Initialise: subprogram;
  TransferTargetData: subprogram;
  Release: subprogram;
end StoreControlAPI;

bus StoreControlProtocol
  provides
    API: list of subprogram group StoreControlAPI;
end StoreControlProtocol;

thread WeaponManager
  requires
    storeControl: bus access StoreControlProtocol;
end WeaponManager;

thread implementation WeaponManager.Impl
  calls
    initialise: subprogram storeControl.API.Initialise( .. );
    transfer: subprogram storeControl.API.TransferTargetData( .. );
    ...
end WeaponManager.Impl;
```

Store control API defined as set of subprograms

Provides thread with access to the store control API as provided by the store control protocol
Military Air Systems

Store Control Protocol

Store Control Protocol <abstract> refined to

Store Control Protocol <refined>

Store Communication Protocol <abstract>

Store Communication API

Page 15 AADL Modelling of PnP Weapon, SAE-AS2 Meeting, Ft Walton, January 2006
Military Air Systems

Store Control Protocol (AADL)

```plaintext
bus StoreControlProtocol
provides
    API: list of subprogram group StoreControlAPI;
implementedAs
    StoreControlProtocolSystem.Impl;
end StoreControlProtocol;

system StoreControlProtocolSystem
-- inherited interface
-- provides
-- API: list of subprogram group StoreControlAPI;
requires
    storeComms: bus access StoreCommunicationProtocol;
end StoreControlProtocolSystem;

system implementation StoreControlProtocolSystem.Impl
subcomponents
    fsm: list of thread StoreControlFSM.Impl( comms => storeComms );
properties
    -- binding of API subprograms to individual FSM threads
end StoreControlProtocolSystem.Impl;
```

Refinement of the virtual bus into the specified protocol system

Additional lower-level interface of the refined protocol system to the underlying store comms protocol
Store Control Protocol (AADL)

thread StoreControlFSM
provides
    Initialise: subprogram Initialise;
    TransferTargetData: subprogram TransferTargetData;
    Release: subprogram Release;
requires
    storeComms: bus access StoreCommunicationProtocol;
end StoreControlFSM;

thread implementation StoreControlFSM_Impl
annex Behaviour {**
     states
        init: initial state;
        init_done, ready: state;
        failed, released: return state;
     transitions
        init -> init_done {;
        init -> timeout 150ms -> failed {;
        ...  
        init_done -> ready {;
        ready -> Release? -> released {;
        **}
end StoreControlFSM_Impl;

The behaviour of subprogram Initialise is specified in terms of calls to the store comms protocol API, e.g.

s0 -> PowerOn! -> s1 {;
s1 -> BusyBit! -> s2 {;
s1 -> timeout 150ms -> fail {;
s2 -> StoreDesc! -> s3 {;
s2 -> timeout 500ms -> fail {;
Contents

• Introduction
• Notational Issues and Modelling Approach
• The PnP Store Control Protocol
• The PnP System Architecture
• Conclusion
Military Air Systems

Store Communication Protocol

Store Communication Protocol <abstract>

refined to

Store Communication Protocol <pre-launch>

Store Comms FSM

Store Comms FSM

Store Comms FSM

Store Comms FSM

LL Store Comms Protocol <1553>
Store Communication Protocol (cont’d)

Refined to:

Store Communication Protocol <abstract>
**Abstract Physical Architecture**

**A/C Platform**

- **Comms IF Computer**
- **Mission Computer**
- **Store Mgmt System**

**Avionics Bus**

- **LL Store Comms API (1)**
  - Messages via avionics bus.

**Store Bus**

- **Store #1 (Weapon)**
- **Store #n (Weapon)**

**Air Data Link**

- **Message Router**
  - Transparent routing of all store communication messages between avionics & weapon bus.

**Abstract Processors**

Communication services are provided as subprogram features.
Contents

• Introduction
• Notational Issues and Modelling Approach
• The PnP Store Control Protocol
• The PnP System Architecture
• Conclusion
Conclusion

- AADL provides the capability to specify system architectures that contain several layers of communication protocols (OSI stack model).

- This modelling is based on the notion of abstract buses (protocols) and abstract processors (virtual machines) which can be refined into full-scale system implementations.

- The explicit execution semantics and the behavioural specification formalism inherent to AADL provide an excellent foundation for model validation (e.g. simulation), verification (e.g. model checking), and code-generation.