Applications of XML in Conjunction with DDS for the Development of Distributed Real-Time Software

Streamlining the development process to reduce life-cycle cost and enhance productivity

Dan Schneider
daniel.d.schneider@lmco.com

John Rusnak
john.j.rusnak@lmco.com

Copyright © 2007 Lockheed Martin Corporation
Overview

- Background Information
  - Who we are
  - What we do
  - System Overview

- The Vision
  - Goals for a better system
  - Development for a distributed environment

- The Solution
  - Marrying XML and DDS
  - Observed benefits

- Summary
Who We Are

- Aerospace software developers
- Working simulations and test beds for many years
  - Scalable distributed systems
    - Reflective memory architecture
    - Proprietary data transport software
  - Real-time and non-real-time
  - Highly algorithmic
  - Highly Modular
What we do

Simulations of space hardware components and environments for:
- Qualification of critical space-flight software
- Operational testing and training against critical software

Expected to adhere to a rigorous process
- Requirements
- Code
- Peer reviews
- Test
Where we came from

- Legacy system
  - Heavy maintenance effort
  - Tedious development process (specifically at I/F)
  - Proprietary software (data transport)
    - Not easily scalable
    - Limited capabilities (no QoS)

- Needed a better way
  - Inspired by DDS
Simulation Overview

Legend:
- User Developed Code
- Sim Kernel Code

Simulation Kernel

Event Dispatcher

Data Transport Layer (DDS)

Event

Command Router

Model A
- Action 1

Model B
- Action 1
- Action 2
- Command 1

Test Manager
Model Name : String
Build()
Initialize()
Finalize()

Topic Name : String
Topic 1
Action 1 subscribes
publishes
Topic 2 subscribes
publishes
Topic 3

Event
Event Scheduler dispatches

Action Name : String
execute()

Model A

Event Scheduler

Model developed code
The Vision
Goals – Development

- Independent development of the core kernel and models
  - Abstraction layer for models
  - Clear distinction in roles

- Reduce/eliminate code coupling between models
  - Seek true modularity
  - Adaptability of model algorithms to multiple simulation environments (OpNet, Simulink, …)

- Automate code generation where possible
  - Allow developer to focus on algorithm and not I/F code

- Flexibility in distribution/execution of models
  - Run-time configurable model set
  - Run-time configurable execution parameters

- Seek COTS based on open standards where possible
  - Prevent lock-in to specific toolset
  - Easier integration
Goals – Maintenance/Process

- **Light Maintenance Effort**
  - Model code only loosely coupled to an API, if at all
  - Reduce repetitive/tedious code
  - Allow kernel changes without impact on developed model code
  - Allow migration between multiple systems/platforms

- **Reduction in Manual Processes**
  - Automate model integration
  - Automatically ensure consistent interface definitions
  - Dynamic memory-map construction
  - Generate documentation from single specification
  - Guarantee code and documentation consistency
  - Validate documentation

- **Define a single authoritative source**
  - Resist having to extract data from specifically formatted comments in code (documentation, . . . )
The Solution
The Solution, Part I: DDS

- Open standard for distributed data communications
  - Prevents being locked into specific tools/vendors
  - Clean, simple yet powerful framework
  - Ideal for real-time applications
  - Peer-to-peer with low overhead (vs. CORBA)

- Scalable

- Publish/Subscribe
  - Coupling of models through message definitions
  - No coupling through code
  - QoS constraints capability
The Solution, Part II: XML

- Open, industry accepted standard
- Ideal for specifying data formats and interface specifications
- Widely supported
  - Programming languages (C, C++, Java, Ada, ...)
  - Tools
  - Text-based (human readable)
- Easily transformed
  - Via simple, yet powerful, scripting language (XSLT)
- Supports Validation
  - Schema-based
XML Specification Elements

DDS Interface Specification
- Types
- Topics

Model Behavior Specification
- Initialization Data
- Commands (asynchronous)
- Actions (periodic)

XML Documents

dependencies
Types, Topics, XML and DDS

- User-specified XML type definitions
  - Basic type (float, integer, boolean, …)
  - Scalar, arrays
  - Encapsulates units, value constraints
  - Requires good type management

- Topic definitions built up from basic type definitions
  - DDS provided inspiration for concept of Topics
  - Bit-mapped topics can be specified (i.e., specified at bit-level)
    - Packed messages, e.g. 1553, handled easily

- Able to auto-generate interface code
  - Type and message format (topic) definitions
  - SW to handle publication/subscription to Topics
  - Model skeletons
Model Class Signature

- Commands and Actions map to Operations
- Initialization data can be automatically mapped to attributes
- Implementation to be provided by user
**Fully Automated Glueware**

- **Glueware code is ready to compile**
  - Handles Pub/Sub (DDS, …)
  - Maps kernel actions to operations in model skeleton
  - Bridges Kernel API and Model Interface
  - Glues model to support data

---

XML Documents

*Interface Specification*
- Types
- Topics

*Model Behavior Specification*
- Initialization Data
- Commands
- Actions

---

**Sim Kernel**

Dynamically loaded Library

*Glueware*

- `attr1`
- `attr2`

- +Initialize (init data)
- +Action (inputs, outputs)
- +Command (inputs)
Developer doesn’t have to worry about underlying database implementation or any database API calls.
Reverse of typical development process

- Developers specify model interface in XML
- Auto-coders map user-defined model API to simulation kernel
- Developers have no need to learn underlying simulation kernel API
  - Focus on developing model algorithms
Consistency achieved between code comments, design diagrams and formal documents
Benefits (1)

- **Reduced maintenance effort:**
  - Less code to maintain by hand
  - Hand maintained code fully de-coupled from kernel API
    - Glueware code is now auto-generated
    - Easily accommodate kernel changes by updating the XSLTs (and NOT user-developed model code)

- **Adaptability:**
  - Models can be adapted to new engines
    - OpNet, Simulink, . . .
  - Highly modularized system
    - Simplifies testing
    - Module distribution is database defined
Benefits (2)

- **Lower development/integration costs:**
  - Significantly reduced model development time
  - XML validation reduces integration debug time
  - Single source for interfaces simplifies integration
  - Reviews can focus on critical code – algorithms

- **Consistency:**
  - High level model architecture made consistent
  - Documentation / Code consistency
  - Opens door to browsable documentation and code
Benefits - Testing

- Use schema-based XML test script
- Front end to the sim kernel parses the XML test scripts
  - Set inputs and check assertions
  - Configures data recording
  - Reports run status on completion
- Modularity simplifies execution
  - no surrogate or specific unit test code
  - test modules one by one *in their actual operating environment*
Summary

- Following the DRY principle results in better consistency, less work and easier troubleshooting
- Synergy between XML and DDS can be used to create a simple yet robust architecture with reduced development, integration and maintenance time
  - Modularity can make coding, testing, and distribution easier
  - Using open standards is a way to prevent vendor lock-in