Model-based development of ARINC 653 using UML and SysML

Agenda

- Motivation of Integrated Modular Systems
- Modelling Notation Standards UML and SysML
- Applying UML and SysML to IMS
- ARINC 653
- Using Models to support ARINC 653
- Conclusion
Motivation of Integrated Modular Systems

Integrated Modular Systems (or IMA) aim to

- Minimize Life Cycle Costs
- Enhance Mission and Operational Performance
- Allow greater flexibility and re-use in Development and Maintenance

Existing standards for IMS

- ARINC 653 – for implementing IMS concepts onto RTOSes
- Stanag 4626/EN 4660 – to de-couple Avionics HW and SW
Federated Architectures (shown with UML)

Broad range of possible problems:

- Any failure effects everything
- Scheduling sensitive to any change
- Difficult to maintain:
  - Obsolescence
  - One change effects again everything
- Re-Use?
A closer Look on Federated Architectures

- On Closer inspection, logical ‘wrappers’ appear that perform specific functions.
- These can be interpreted as ‘layers’ in a Federated Architecture.
From Federated to a Layered Architecture

In an IMS layered architecture each layer has a clearly defined responsibility, interface and is as important as each of the other layers.
The resulting IMS Layered Architecture

- The Application Layer is partitioned into a network of (potentially) reusable components.
- Each component has a secure execution environment (processor time and memory).
- Inter-component communication is managed by the OSL.
- All components are loaded into memory and scheduled by the OSL.
- A set of components can be re-configured at run-time
IMS Network Architecture

- A set of components execute on a specific hardware node.
- A complete system will require a number of hardware nodes.
- The OSL (across all the hardware nodes) manages and co-ordinates the system.
IMS Reconfiguration

- The allocation of components to hardware nodes is not static at run-time.

  The failure of a critical component on one hardware node will cause it to be re-deployed to another hardware node.

  The failure of one Hardware Node will cause all components to be re-deployed.

- Changes in Mission may also cause a reconfiguration.
Information required by the OSL

- The OSL must know:
  - *What* component is running
  - *Where* the component is running
  - *How long* the component runs for
  - *What to do if* the component fails
  - *Scheduling policy* for all components
  - *Information required* by all components
  - *Information provided* by all components
  -...

- All this information is held in ‘Blueprints’
  - Software (for each component)
  - Hardware (for each hardware execution environment)
  - Configuration (links Software with Hardware)
  - Run-time (contains multiple configurations)
Modelling Network Architectures
Hierarchical Design Notations

- Keep breaking the problem down until you have solved all the parts...
- Is the whole the sum of all the parts?
- Design notation cannot be extended to capture essential properties (e.g. thread execution parameters).
- The hierarchy becomes almost immutable!
- Complexity becomes hidden in the hierarchy!
Hierarchical Design Notations
Development/Maintenance Issues

- The nodes (data/control transforms) are (relatively) easy to maintain (i.e. changes localised to a node).

- Information flows that link nodes are NOT easy to maintain! The knock-on effects caused by the introduction of the hierarchy can be prohibitive.
Hierarchical Design Notations
Development/Maintenance Issues

- Unless a rigorous maintenance process is adopted, it is often quicker (i.e. cheaper) to implement a change only where the design is significantly affected.

- The higher-levels of the hierarchy become almost obsolete or overlooked.
Hierarchical Design Notations

The *Emerging* Network

- Information-flows at high-levels connect DFD’s at low-levels.

- Design information on High-level DFD’s become an integral part of the network.

- The hierarchy flattens into a network of collaborating nodes.
Resulting Modelling Language considerations

- Hierarchical design notations are deficient in detailing the distributed nature of IMS modules.

- Limited (if any) extensibility to capture specific IMS properties.

- A notation capable to handle networks of generic elements is needed

  => UML (and SysML)!
UML/SysML Modelling for IMS
Using standards and their extensibility
What UML and SysML Provides
Taxonomy of Diagrams (Viewpoints)

Each diagram-type provides a unique perspective or viewpoint on an underlying model.
What UML and SysML Provides
Taxonomy of Diagrams (Viewpoints)
Applying the Basics of UML to IMS

Use Case Model
(Modelling capabilities of each IMS layer)

Scenario Model
(Detailing communication flows between IMS components)

Dynamic Model
(Detailing dynamic behaviour of IMS components)

Class Model
(Detailing static architecture of IMS components)
Extending the SysML & UML Notation

- SysML & UML can be extended by «Stereotypes»
  Stereotypes are applied to standard UML model-elements.
  Additional properties ("Tags Definitions") are added to stereotypes.
- SysML & UML can encompass any language domain:
  Software Engineering
  - Programming Languages (e.g. Ada, Java, C, C++, C#, IDL, VB)
  - Real-time Systems (OMG MARTE Profile)
  - Component-based Development
  - Incremental Development
  Systems Engineering
  - IMS Terminology (Stanag 4626/EN 4660 & ARINC-653)
  - Business Terminology
  - System On Chip (OMG SoC Profile)
  - Architecture Analysis & Design Language (SAE AADL Profile)
  - Goal Structured Notation (GSN)
  - Architectural Frameworks
    - Zachman
    - UPDM (MODAF/DoDAF)
  - AUTOSAR
  - ...
Extensibility – What is a Stereotype?

- A stereotype defines how an existing metaclass (UML model-element type) may be extended, and enables the use of platform or domain specific terminology or notation in addition to the ones used for the extended metaclass.

- A stereotype can be used to change the semantics of a model element in an non-arbitrary way.
Extensibility – What is a Tag Definition?

- Tag Definitions allow additional properties (of the Stereotype) to be documented.

For example, RMS Voltage, Peak Voltage, Current ...

```
<stereotype>
electrical

rmsVoltage : String
peakVoltage : String
current : String
```
Example of Extensibility

ARINC 653

Define the “ARINC 653 Profile”

Create the ARINC 653 Elements (cross referencing with other UML model-elements)

It’s all UML under the cover!
Example of Extensibility
IMS Terminology

- An “IMS Profile” encompasses the terminology of IMS. Properties (WCET, Scheduling policy, memory requirements...) are tags within the IMS Profile.

- These properties are available outside of the model (via XMI/XML for example) for assessment and analysis (because the profile makes these properties a formal part of the modelling language).

Safety assessment, performance analysis etc.
Applying UML, SysML, ARINC 653 + IMS Profiles to IMS

Module Support Layer

Module/OS Interface

Operating System Layer

AP/OS Interface

Application Layer
Modelling IMS Capabilities
UML Use Case Diagram

- Each layer in the architecture provides ‘capabilities’.
- These can be documented as Use Cases.
- Use Cases can be hierarchically organised.

Application Layer
Exploring Use Case
UML Interaction Diagrams

Hazard Analysis: What if something goes wrong?

The Pilot informs the Navigator to initiate a landing sequence.
The Navigator selects ILS.
The VOR ILS (Subsystem) receives the request to enter ILS.
The VOR ILS (Subsystem) informs the VOR ILS (Equipment) to enter ILS mode.
while ILS Not Active loop
  Retrieve the State of the VOR ILS (Equipment)
end loop
VOR ILS informs the Format Manager to Display the ILS Format
Activate the Head Up (Front) (Display) with the ILS Format
Activate the Head Up (Rear) (Display) with the ILS Format
Activate the ILS Manager
while ILS Active loop
  Schedule the ILS Manager
  Retrieve the Raw Localiser Data
  Retrieve the Raw Glideslope Data
  Retrieve the calculated Localiser position
  Retrieve the calculated Glideslope position
  Update the Head Up (Front) (Display)
  Update the Head Up (Rear) (Display)
end loop

Timeout Required?

Land Aircraft (System Only)    Land Aircraft (Software Only)
Modelling IMS Dynamics
UML State Diagram (and Class Diagram)

- A clear understanding of dynamic behaviour is essential.
- UML allows the dynamic properties of the whole IMS, an individual IMS component or an individual thread (within a process) to be linked with the IMS structure.

The behavioural characteristics of an individual thread. Extra properties such as scheduling policy (e.g. priority inheritance, priority ceiling) can be added to the overall dynamic model and analysed.
The (Stanag 4626/EN 4660) Application/OS interface captured as a set of class-partitioned operations.

The run-time characteristics of each operation are captured as IMS properties.
Modelling IMS Hardware
SysML Internal Block Diagram

- An individual Processing Node (execution environment)
- External Connections (Port)
- Boards (Part)
- Data Bus (Part)
- Memory (Part)
Modelling IMS Hardware
SysML Internal Block Diagram

- Topology of all Processing Nodes.

Details (on previous slide) reside in the model, only the top-most part is required here.
Model-based Support of ARINC 653
A Summary on ARINC653

- Standard, supporting IMS
- Aerospace and beyond
- Strong partitioning between applications leads to:
  - System robustness
  - Reuse
    - Application in different configurations
    - Certification where Application changes context
    - Decreased recertification costs (improved change isolation)
- Reduce cost of over-certifying lower SIL applications
  - Isolate them from higher SIL applications using partitioning
- Reduce cost of re-certifying unchanged applications
  - Where applications in other partitions have changed
ARINC 653 Support

- Openly integrated with RTOS and IDE’s
  
  Supporting ARINC 653

- APEX
  
  Applications make calls to the Application Executive defined by ARINC 653 to decouple the applications from vendor ARINC 653 implementations

- Primarily Text Entry
  
  Application behavior and initialization is coded in programming languages (Ada, C, etc)

  System configuration is coded in XML files (ARINC 653 concepts such as partitions, ports, channels etc)
Benefits of using UML and ARINC 653 together

- Synergistic advantages
  - Ease of Adoption
  - Clearer Workflow
  - Communication
  - Productivity
  - Link between configuration and applications clearer in single model

- Possibility to automate and re-use model-based software engineering techniques
Automated Steps using ARINC 653 Profile

- Create all necessary elements based on pre-configured templates
Example Configuration Diagram

- Module, Partition, Application, Process, Port, Channel
Summary of Tool Chain Artifacts

Artisan Studio Model
- Configuration Detail
- Application Detail

Wind River Workbench Workspace
- Configuration Project
  - Configuration XML Files
- Application Project
  - Initialization Source Files
  - Source Files

Wind River Build System
- Binary or Binaries for Target or Simulator

Artisan ACS VxWorks653 Apex Generator
Artisan ACS C VxWorks653 Production Generator

BallGameModule
- Description
  - Player1Partition
  - Player2Partition

loop
seq
«ScheduleWindow» {ReleasePoint, DurationSeconds = 0.25}
seq
«ScheduleWindow» {ReleasePoint, DurationSeconds = 0.25}
end loop
The best possible ARINC 653 Support needs

- (Open) Ergonomic UML Profile containing all ARINC 653 views and elements
  Automation Scripts to ease auto-generation of necessary elements

- Code Generators (and Generator Models) to get ARINC 653 Configuration Files (XML)
  Startup Code for the selected RTOS supporting ARINC 653

- Professional Services to adapt Profile Scripts
  ARINC 653 Generator Models
Questions and Answers

Thanks for your attention!

Speaker