Model-based development of ARINC 653 using UML and SysML

Andreas Korff, Atego – OMG RT Workshop, Paris, 18.04.2012

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

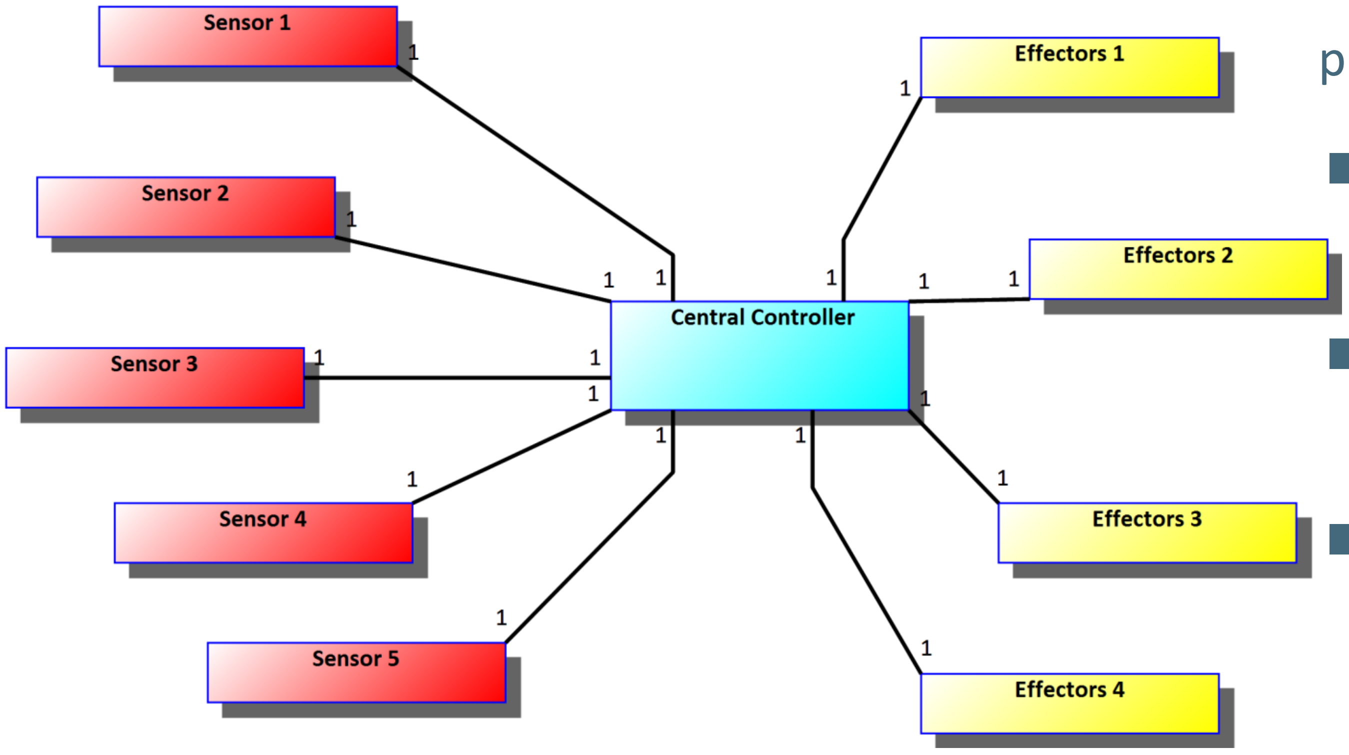
Integrated Modular Systems (or IMA) aim to

- 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:

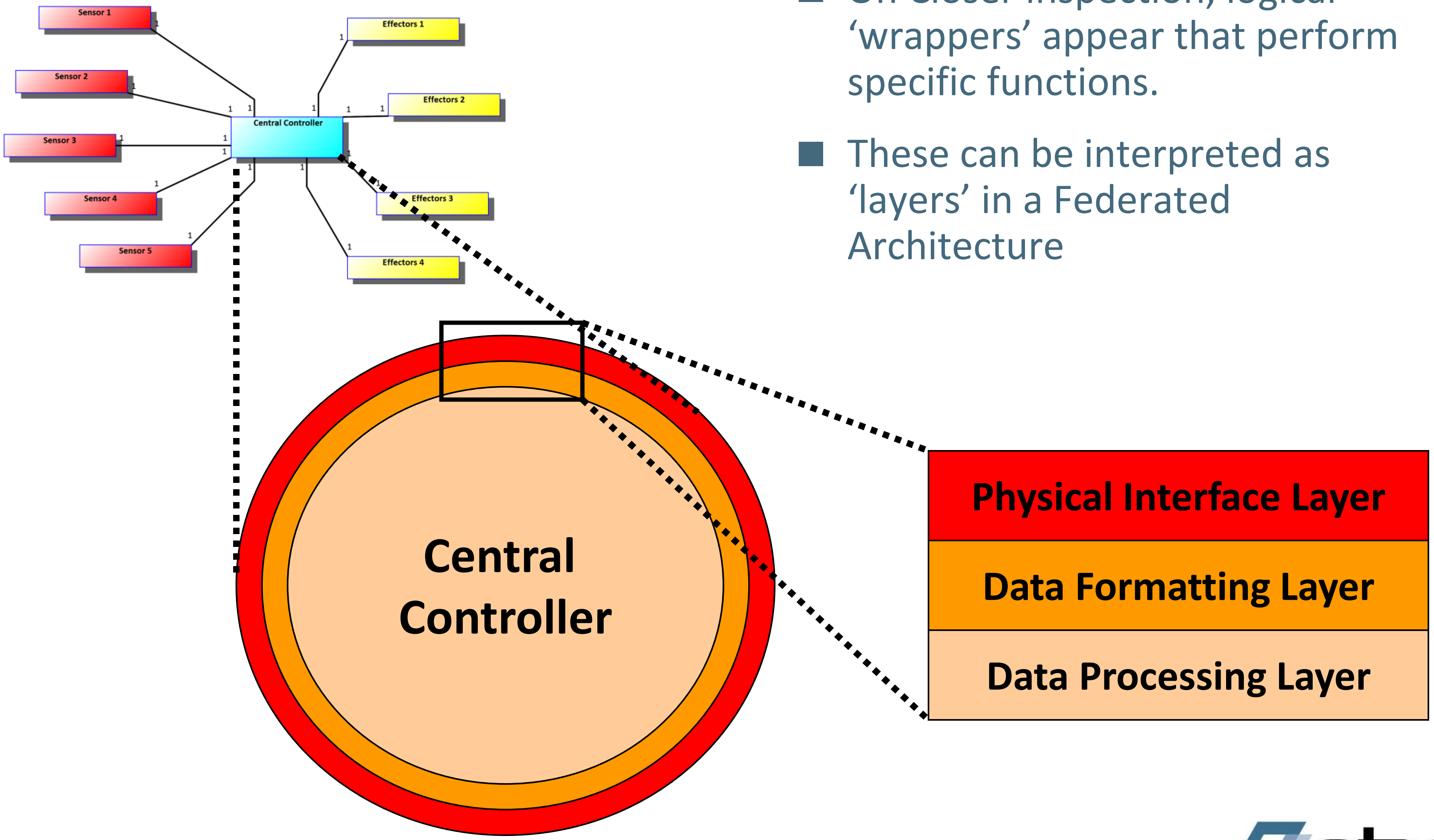
- Any failure effects everything
- Scheduling sensitive to any change
- Difficult to maintain:

  Obsolecence

  One change effects again everything
- Re-Use?

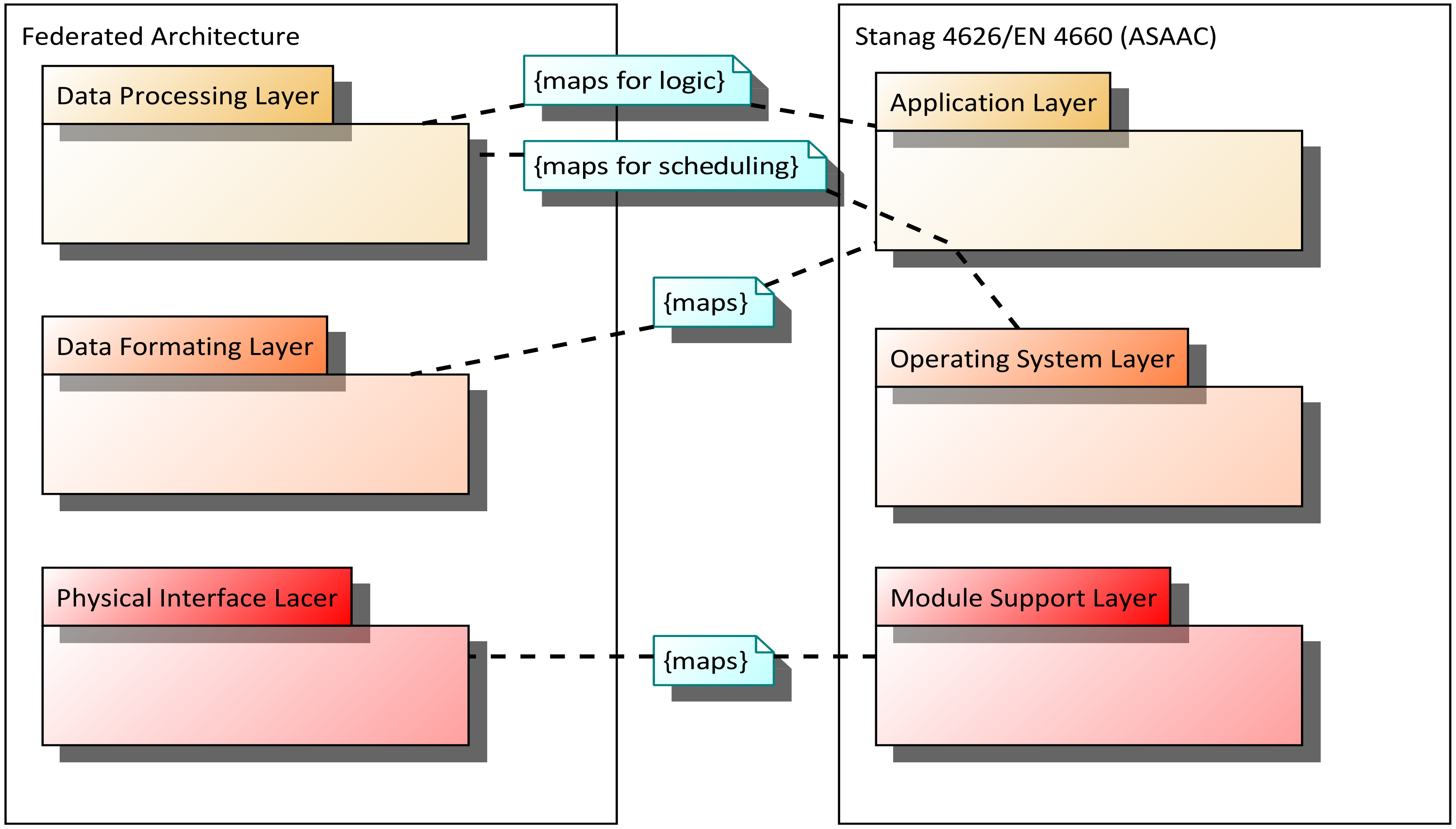
A closer Look on Federated Architectures

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



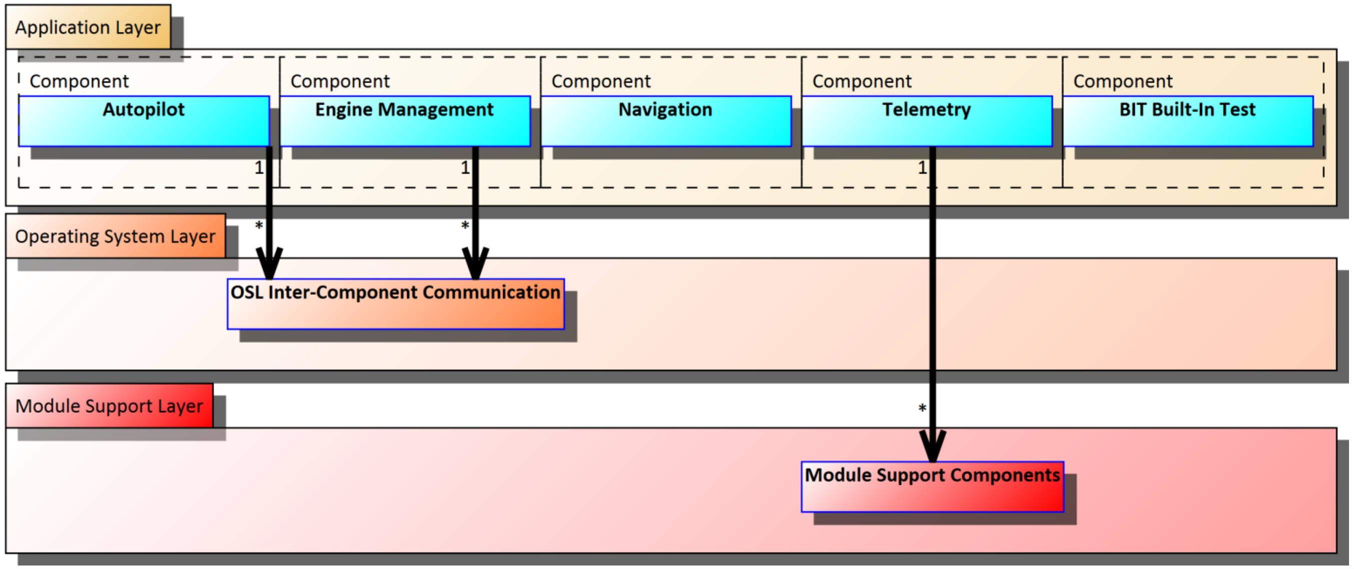
From Federated to a Layered Architecture

- In an IMS layered architecture each layer has a clearly defined responsibility, interface and is as important as each of the other layers



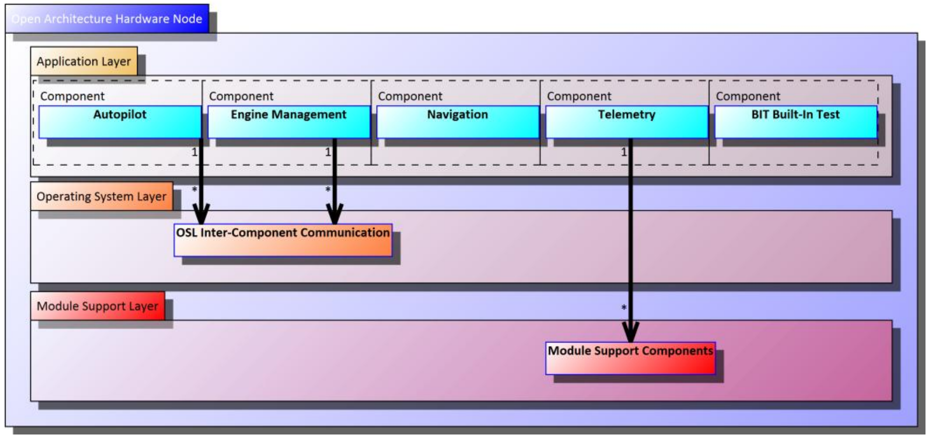
The resulting IMS Layered Architecture

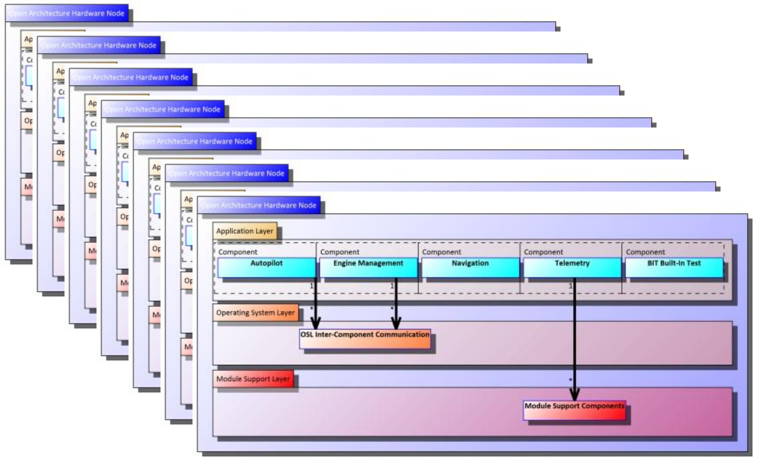
- The Application Layer is partitioned into a network of (potentially) reusable components.
- Each component has a secure execution environment (processor time and memory).
- Inter-component communication is managed by the OSL.
- All components are loaded into memory and scheduled by the OSL.
- A set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

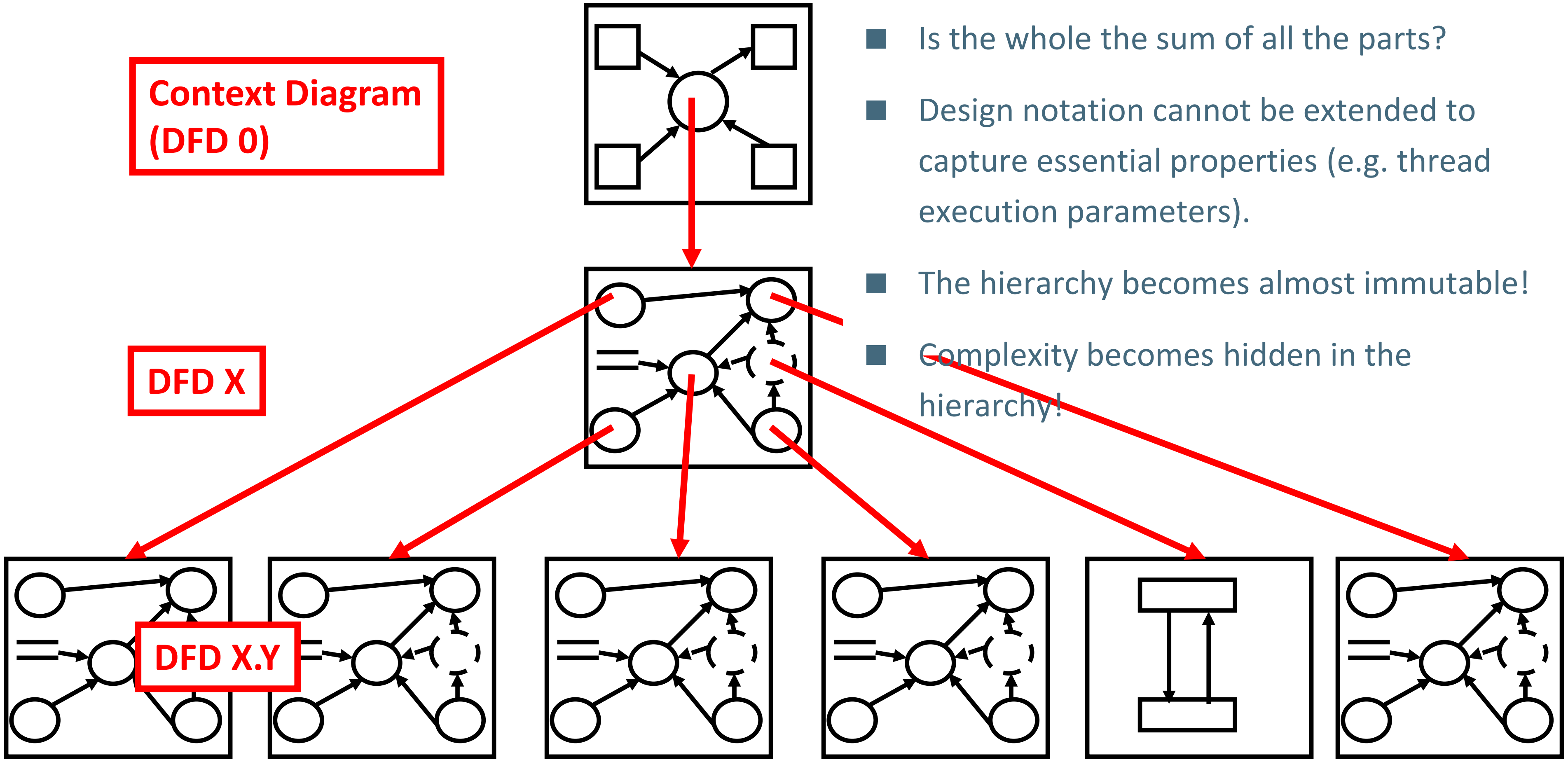
Information required by the OSL

- 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!

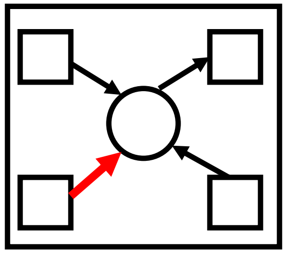


Hierarchical Design Notations

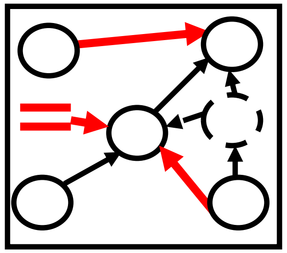
Development/Maintenance Issues

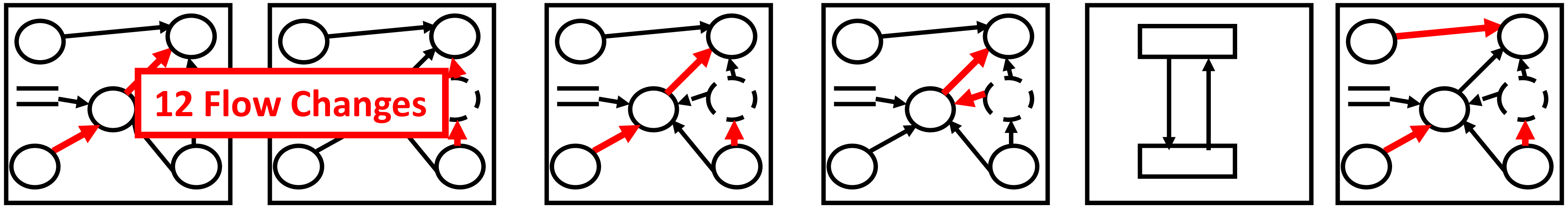
- The nodes (data/control transforms) are (relatively) easy to maintain (i.e. changes localised to a node).
- Information flows that link nodes are NOT easy to maintain! The knock-on effects caused by the introduction of the hierarchy can be prohibitive.

1 Flow Change

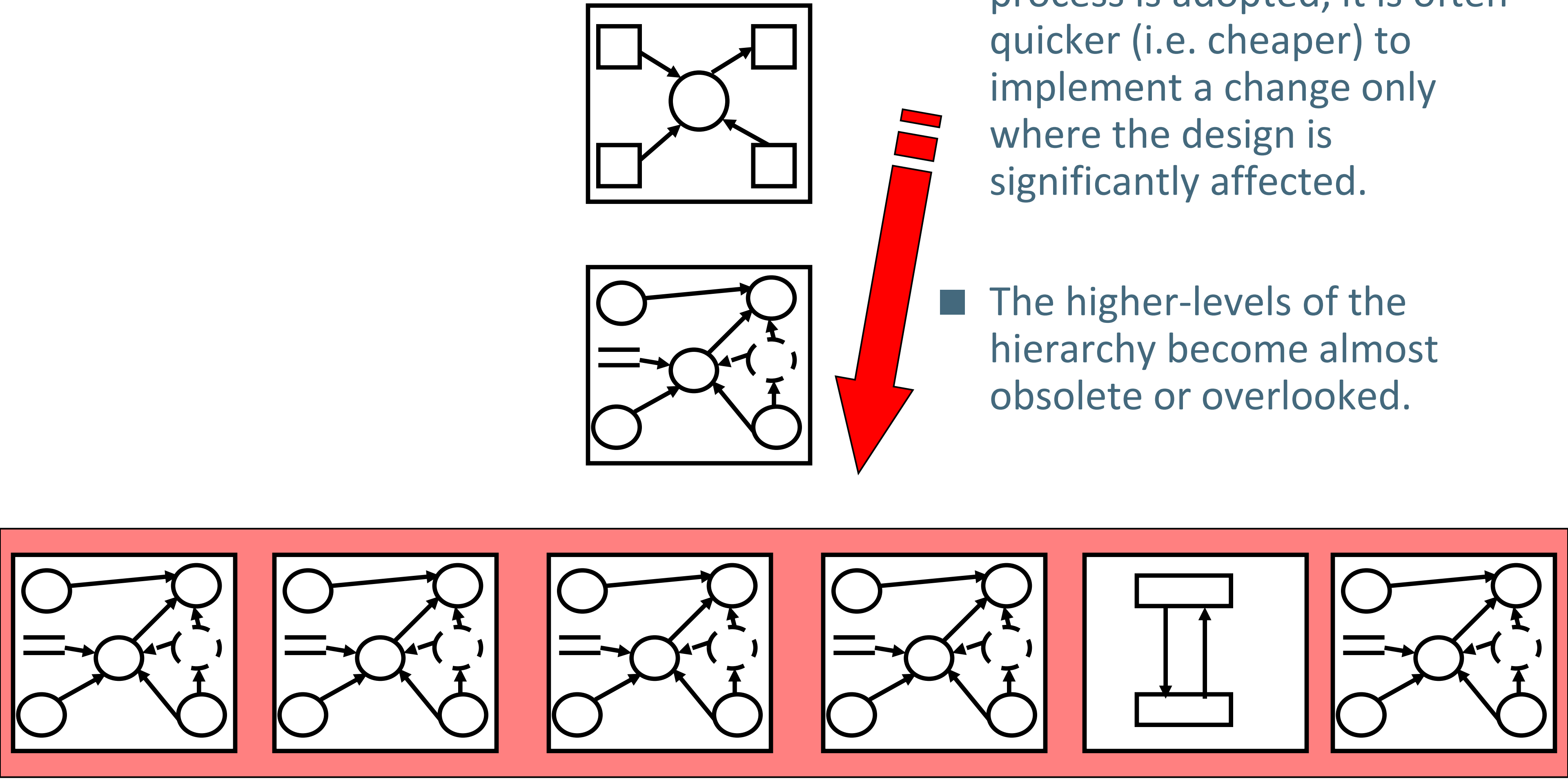


3 Flow Changes





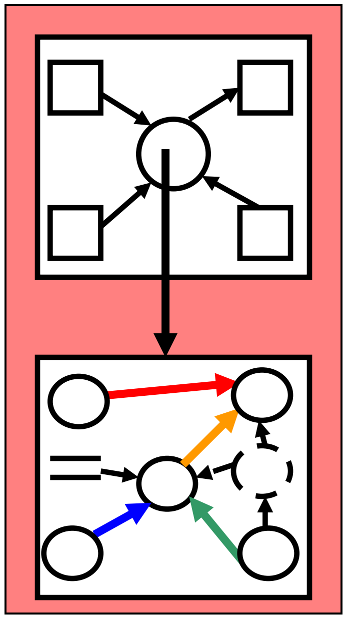
Hierarchical Design Notations
Development/Maintenance Issues



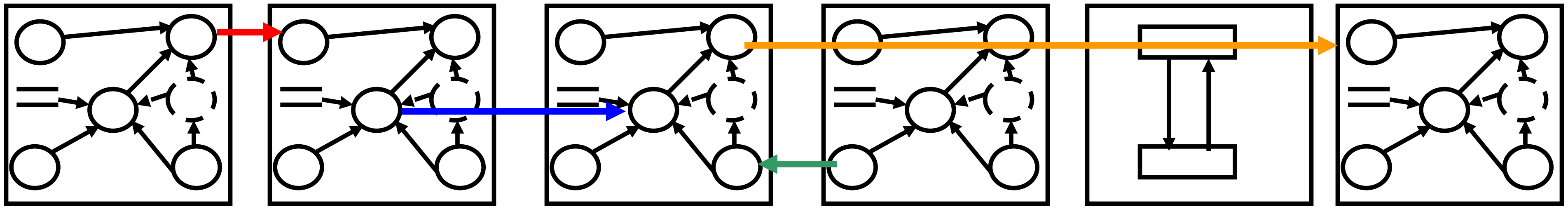
- Unless a rigorous maintenance process is adopted, it is often quicker (i.e. cheaper) to implement a change only where the design is significantly affected.
- The higher-levels of the hierarchy become almost obsolete or overlooked.

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

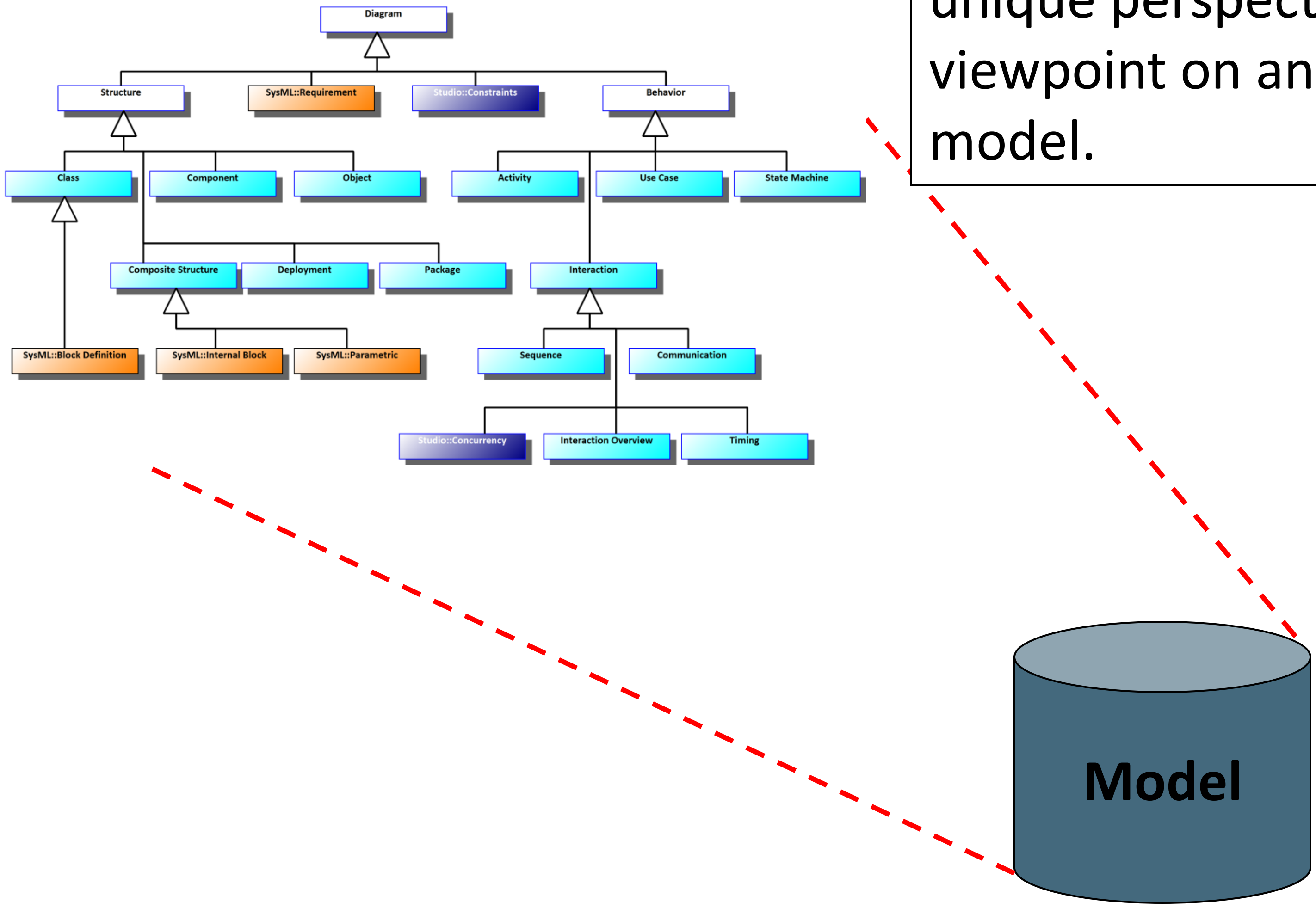
=> UML (and SysML)!

UML/SysML Modelling for IMS

Using standards and their extensibility

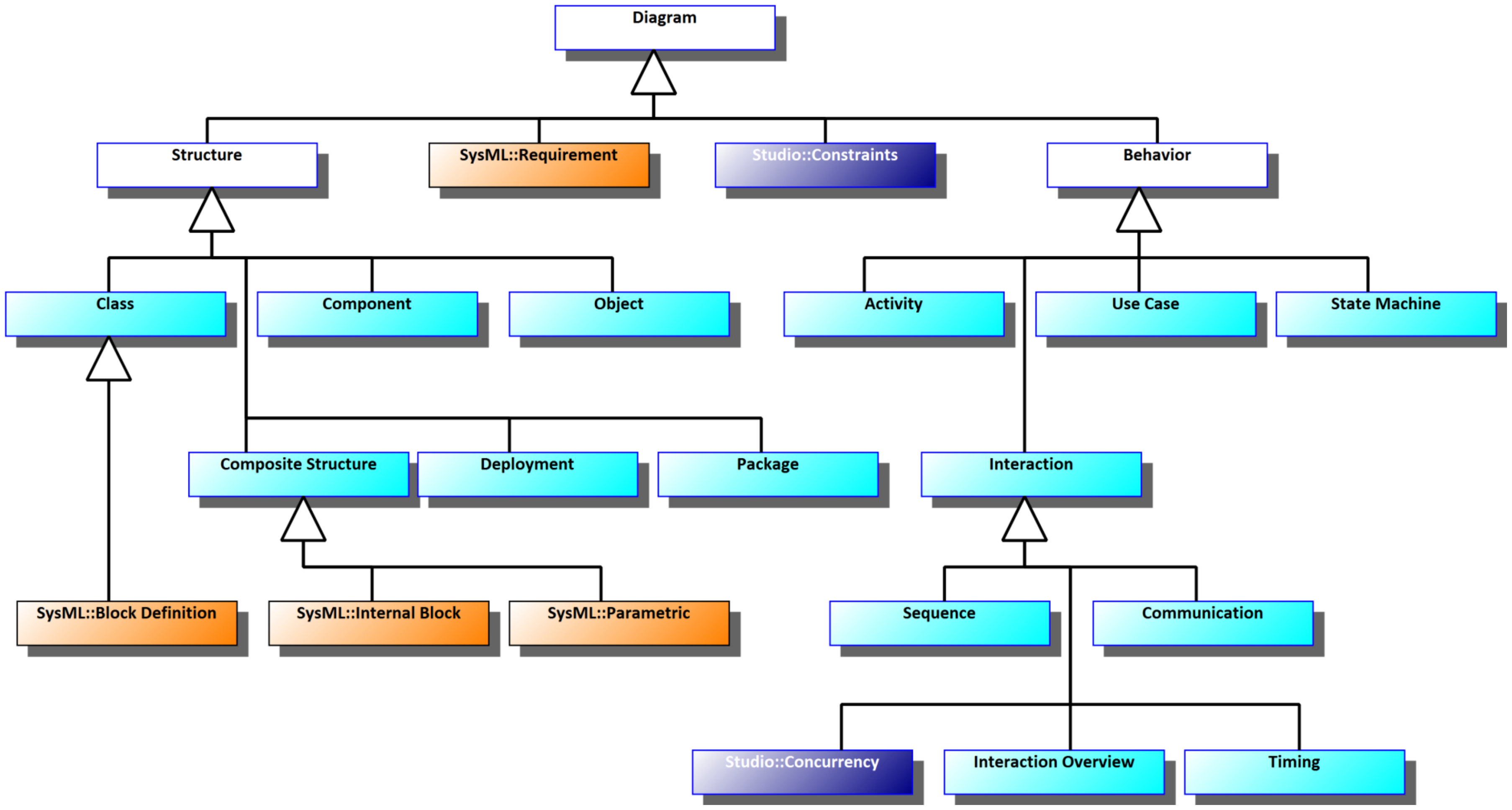
What UML and SysML Provides
Taxonomy of Diagrams (Viewpoints)

Each diagram-type provides a unique perspective or viewpoint on an underlying model.

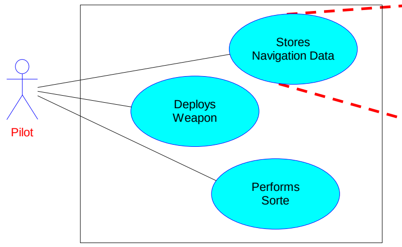


What UML and SysML Provides

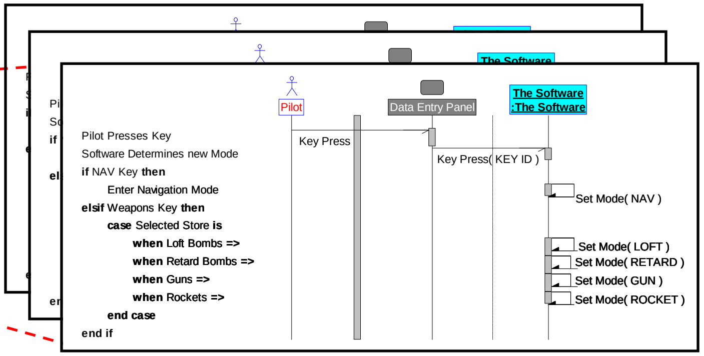
Taxonomy of Diagrams (Viewpoints)



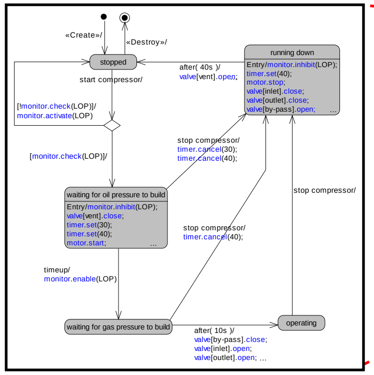
Applying the Basics of UML to IMS



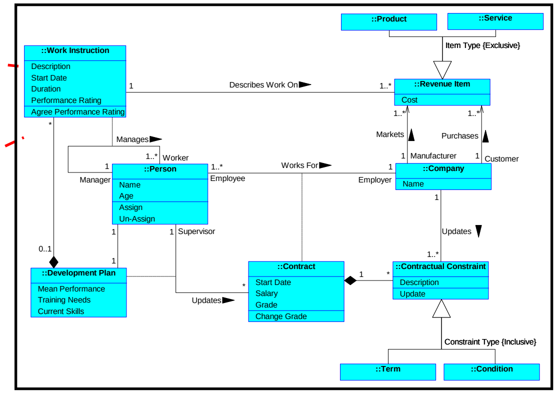
Use Case Model
(Modelling capabilities of each IMS layer)



Scenario Model
(Detailing communication flows between IMS components)


«Create»/
«Destroy»/
stopped
start compressor/
running down
after(40s)/ valve[vent].open;
[!monitor.check(LOP)]/ monitor.activate(LOP)
[monitor.check(LOP)]/
stop compressor/ timer.cancel(30); timer.cancel(40);
stop compressor/
waiting for oil pressure to build
timeup/ monitor.enable(LOP)
stop compressor/ timer.cancel(40);
waiting for gas pressure to build
operating
after(10s)/ valve[by-pass].close; valve[inlet].open; valve[outlet].open; ...

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.

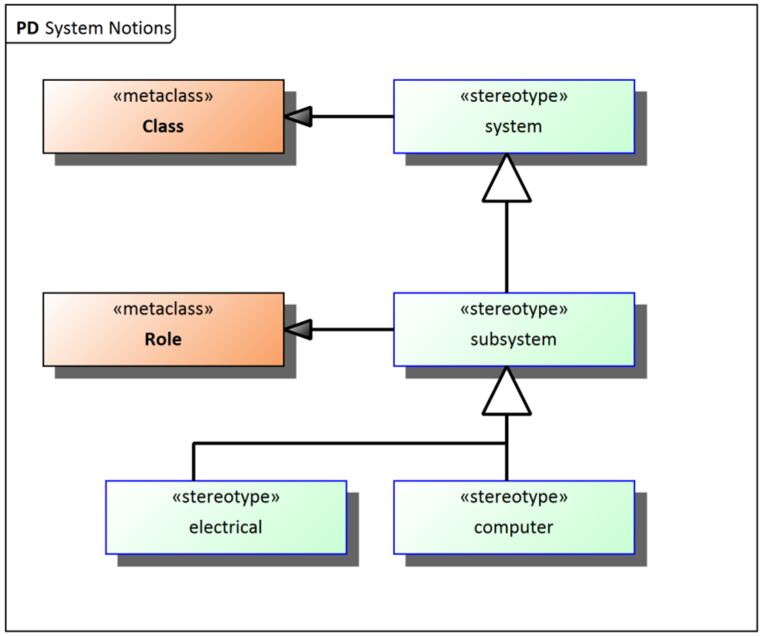
- SysML & UML can encompass any language domain:

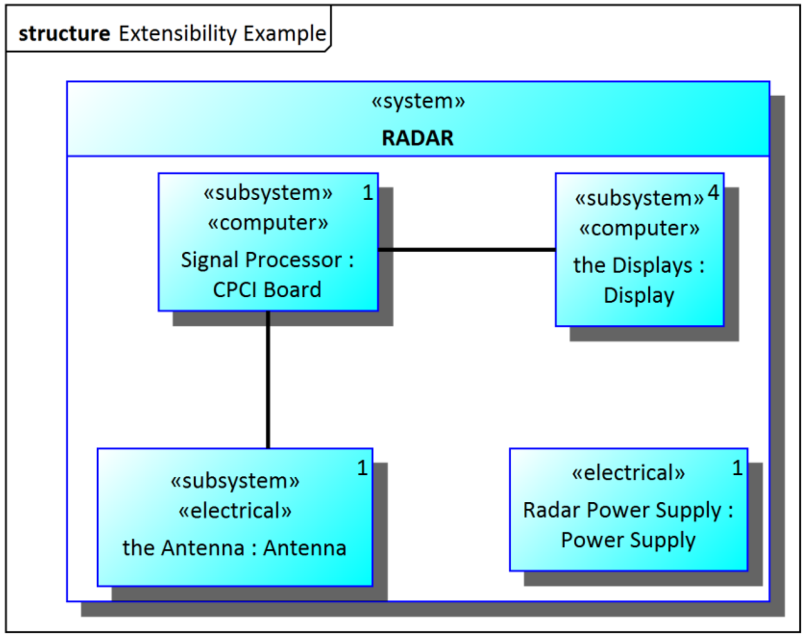
  Software Engineering
    - Programming Languages (e.g. Ada, Java, C, C++, C#, IDL, VB)
    - Real-time Systems (OMG MARTE Profile)
    - Component-based Development
    - Incremental Development

  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.

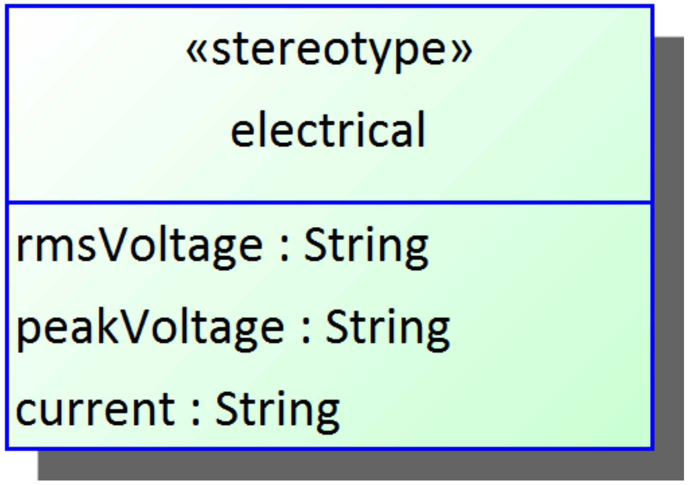


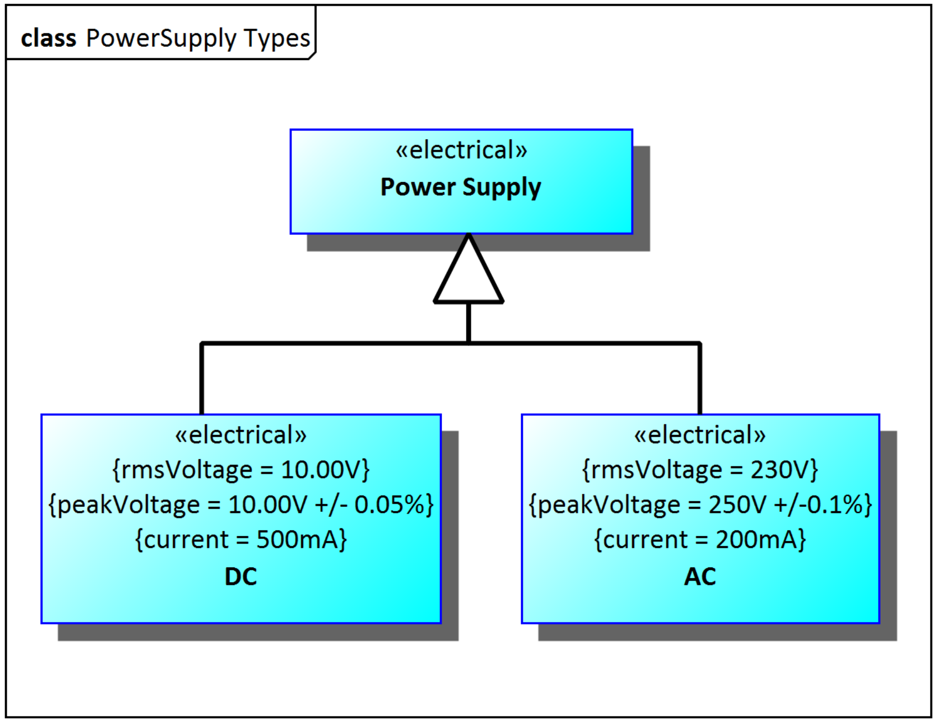


Extensibility – What is a Tag Definition?

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

For example, RMS Voltage, Peak Voltage, Current …



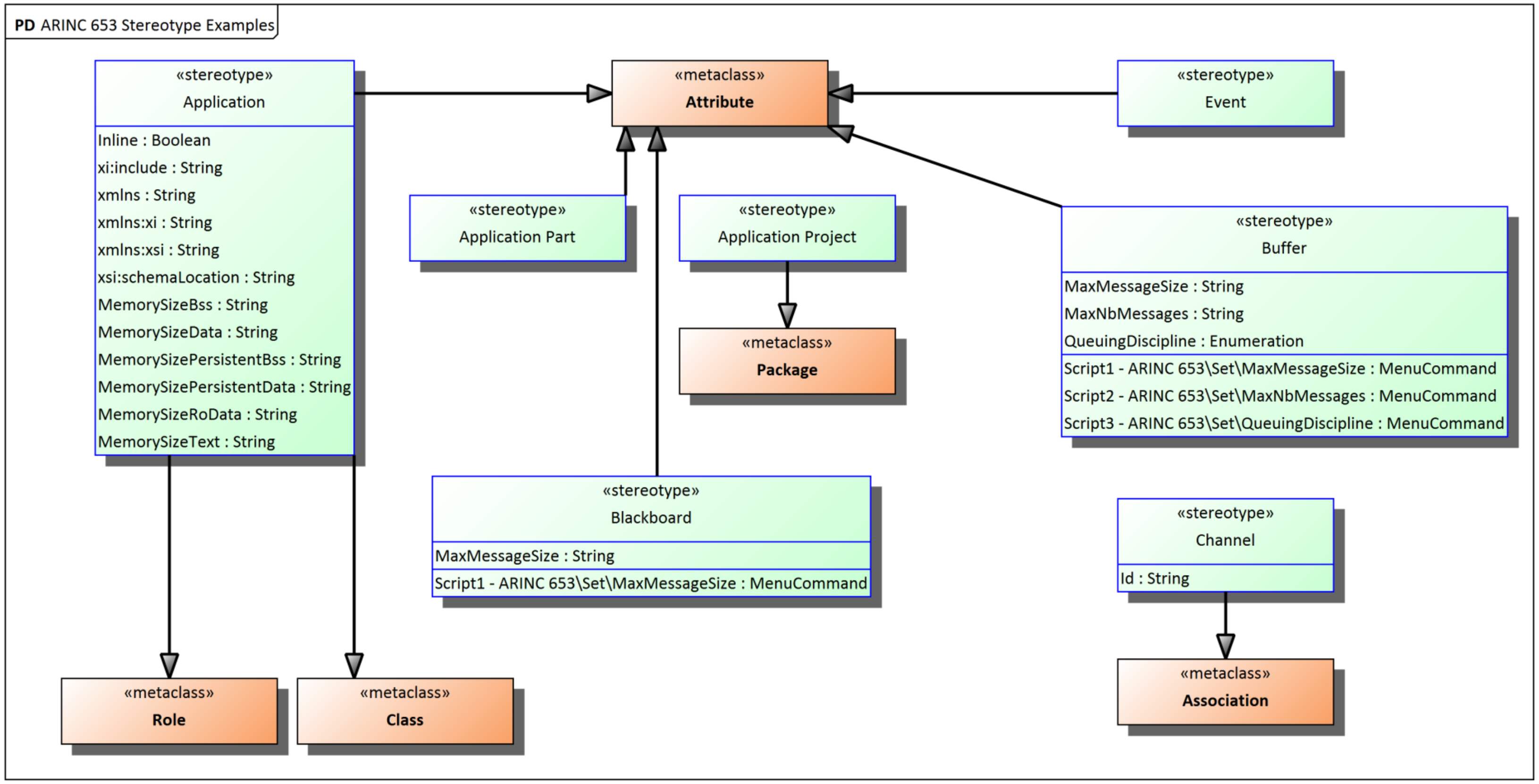


Example of Extensibility
ARINC 653

Define the “ARINC 653 Profile”

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

It’s all UML under the cover!


PD ARINC 653 Stereotype Examples
«stereotype» Application
Inline : Boolean
xi:include : String
xmlns : String
xmlns:xi : String
xmlns:xsi : String
xsi:schemaLocation : String
MemorySizeBss : String
MemorySizeData : String
MemorySizePersistentBss : String
MemorySizePersistentData : String
MemorySizeRoData : String
MemorySizeText : String
«metaclass» Attribute
«stereotype» Event
«stereotype» Application Part
«stereotype» Application Project
«metaclass» Package
«stereotype» Buffer
MaxMessageSize : String
MaxNbMessages : String
QueuingDiscipline : Enumeration
Script1 - ARINC 653\Set\MaxMessageSize : MenuCommand
Script2 - ARINC 653\Set\MaxNbMessages : MenuCommand
Script3 - ARINC 653\Set\QueuingDiscipline : MenuCommand
«stereotype» Blackboard
MaxMessageSize : String
Script1 - ARINC 653\Set\MaxMessageSize : MenuCommand
«stereotype» Channel
Id : String
«metaclass» Association
«metaclass» Role
«metaclass» Class

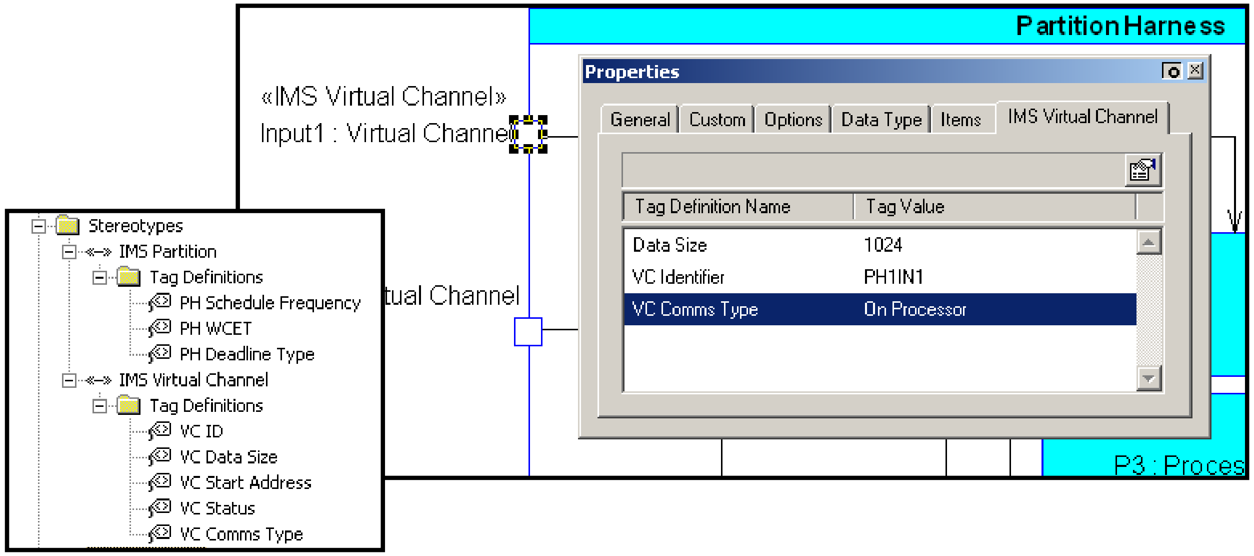
Example of Extensibility
IMS Terminology

- An “IMS Profile” encompasses the terminology of IMS.

  Properties (WCET, Scheduling policy, memory requirements…) are tags within the IMS Profile.

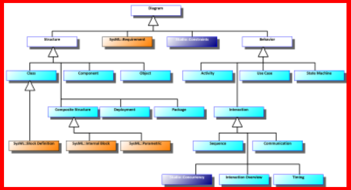
- These properties are available outside of the model (via XMI/XML for example) for assessment and analysis (because the profile makes these properties a formal part of the modelling language).

  Safety assessment, performance analysis etc.

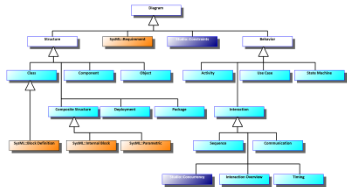


Tag Definition Name	Tag Value
Data Size	1024
VC Identifier	PH1IN1
VC Comms Type	On Processor

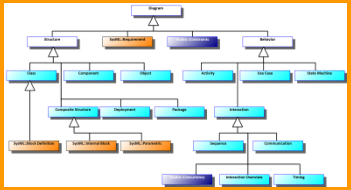
Applying UML, SysML, ARINC 653 + IMS Profiles to IMS



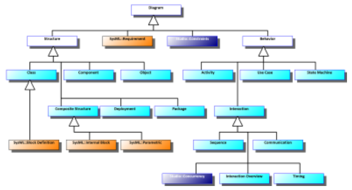
Module Support Layer



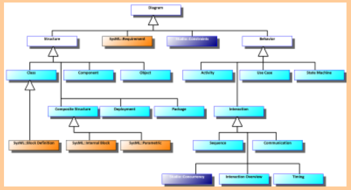
Module/OS Interface



Operating System Layer



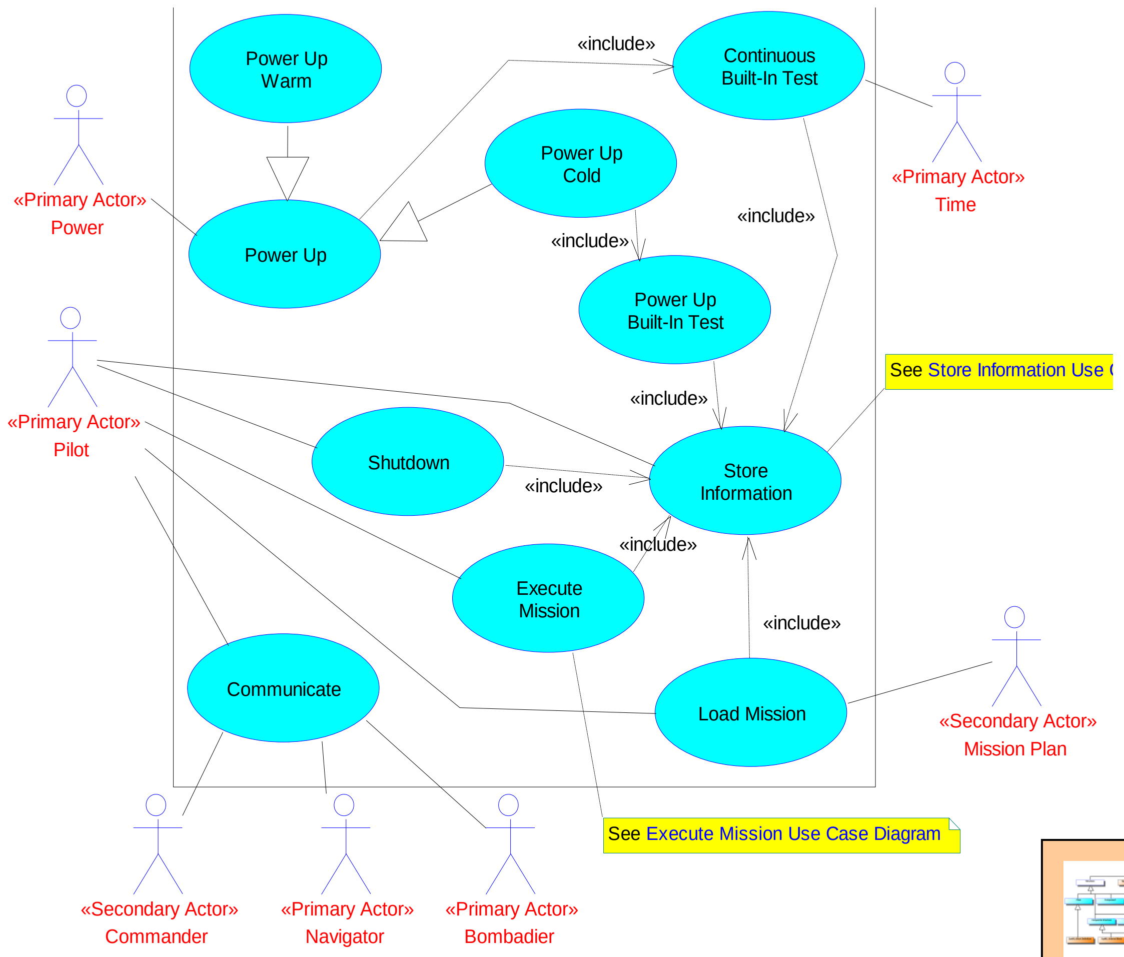
AP/OS Interface



Application Layer

Modelling IMS Capabilities

UML Use Case Diagram



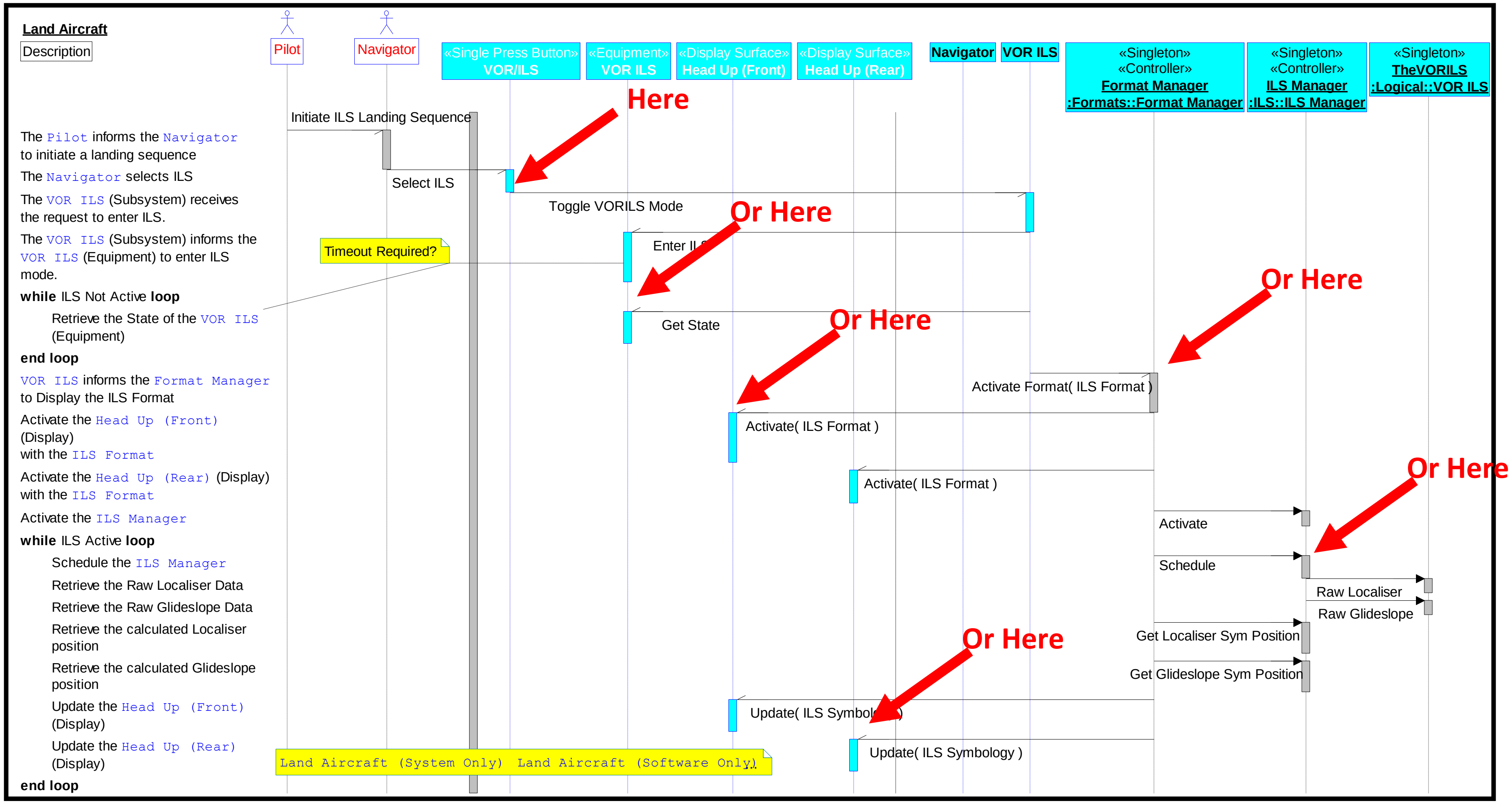
- Each layer in the architecture provides ‘capabilities’.
- These can be documented as Use Cases.
- Use Cases can be hierarchically organised.

Application Layer

Exploring Use Case

UML Interaction Diagrams

Hazard Analysis: What if something goes wrong?



Land Aircraft

Description

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)

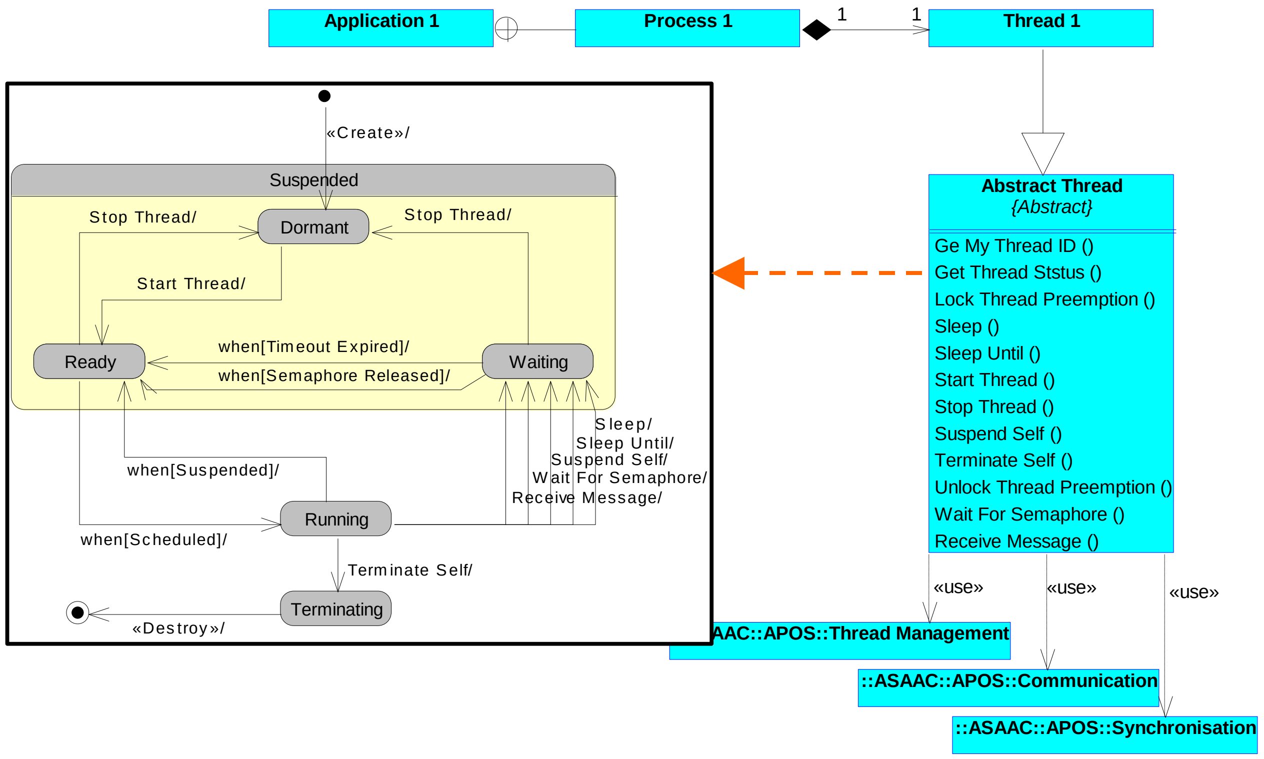
    Update the Head Up (Rear) (Display)

end loop

Pilot
Navigator
«Single Press Button» VOR/ILS
«Equipment» VOR ILS
«Display Surface» Head Up (Front)
«Display Surface» Head Up (Rear)
Navigator
VOR ILS
«Singleton» «Controller» Format Manager :Formats::Format Manager
«Singleton» «Controller» ILS Manager :ILS::ILS Manager
«Singleton» TheVORILS :Logical::VOR ILS
Here
Initiate ILS Landing Sequence
Select ILS
Toggle VORILS Mode
Or Here
Timeout Required?
Enter ILS
Or Here
Get State
Or Here
Activate Format(ILS Format)
Activate(ILS Format)
Activate(ILS Format)
Or Here
Activate
Schedule
Raw Localiser
Raw Glideslope
Or Here
Get Localiser Sym Position
Get Glideslope Sym Position
Update(ILS Symbology)
Update(ILS Symbology)
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.

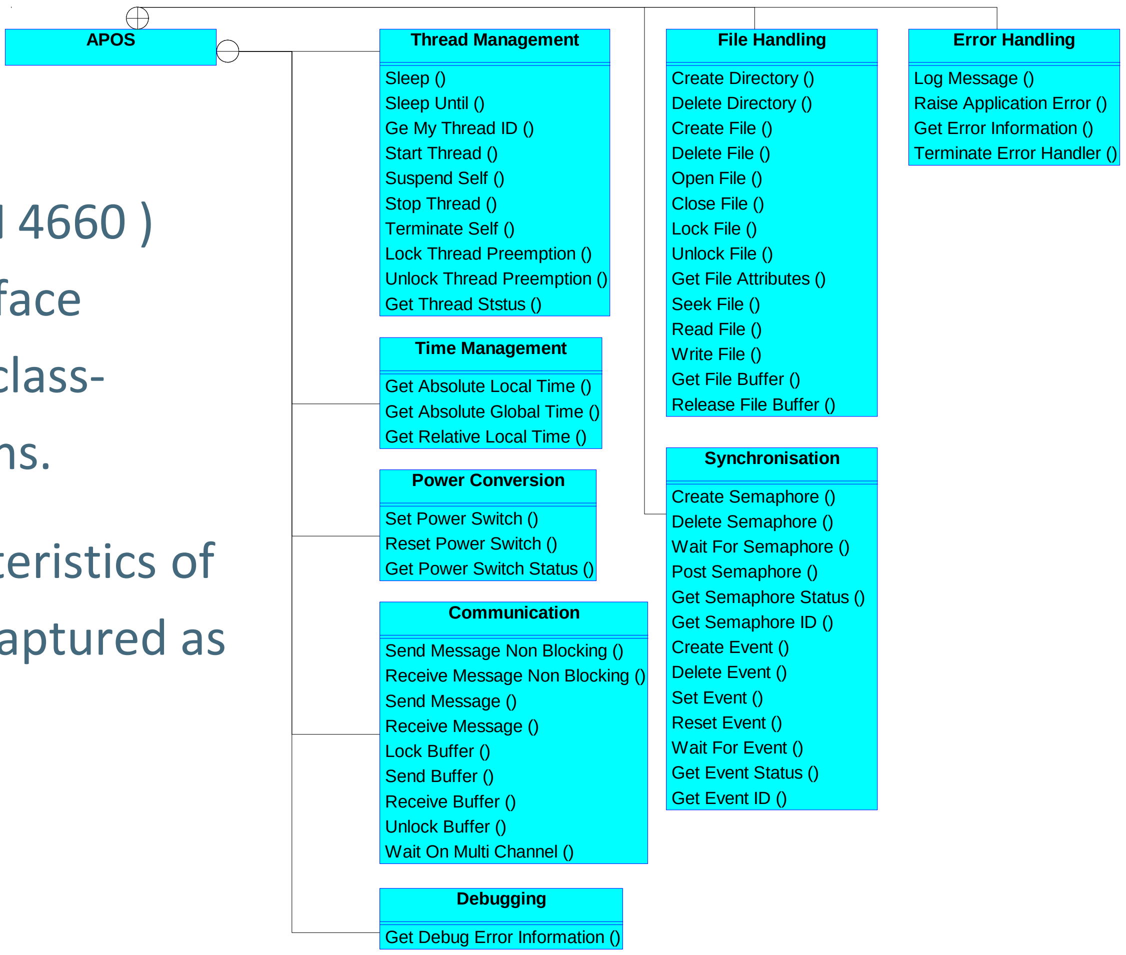


The behavioural characteristics of an individual thread. Extra properties such as scheduling policy (e.g. priority inheritance, priority ceiling) can be added to the overall dynamic model and analysed.

Modelling IMS Interfaces

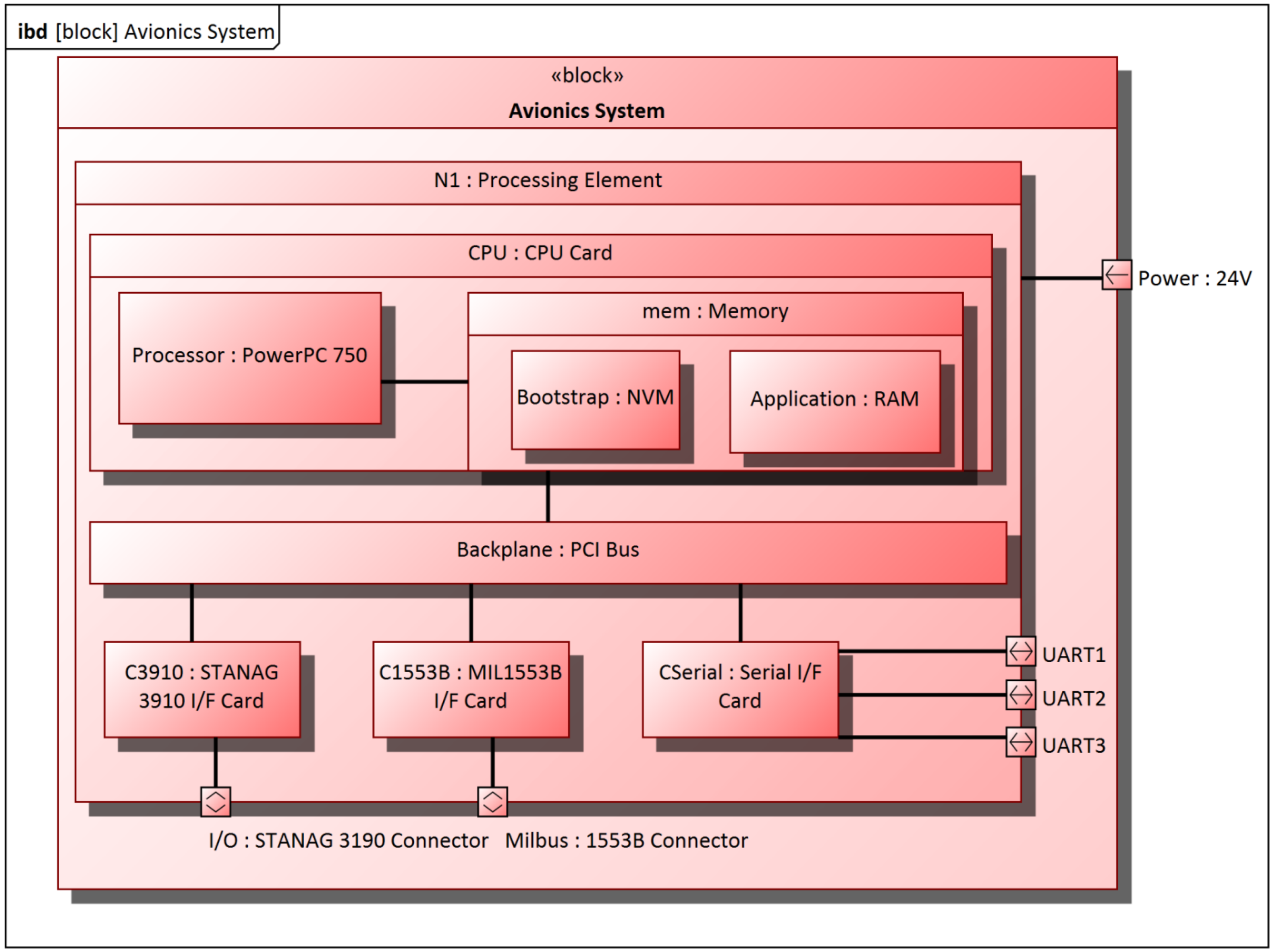
UML Class Diagram

- 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

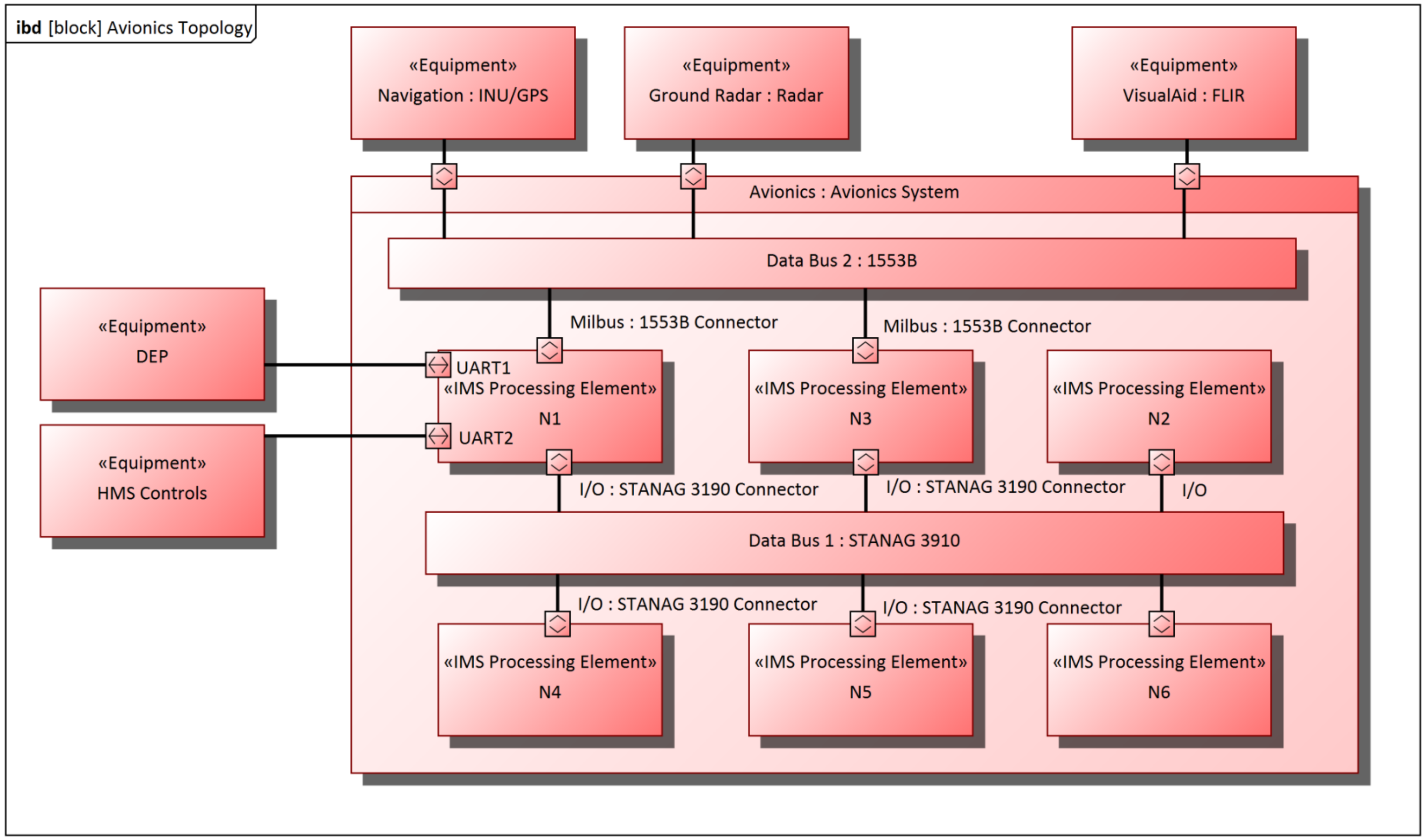
- An individual Processing Node (execution environment)
  - External Connections (Port)
  - Boards (Part)
  - Data Bus (Part)
  - Memory (Part)



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.



ibd [block] Avionics Topology
«Equipment» Navigation : INU/GPS
«Equipment» Ground Radar : Radar
«Equipment» VisualAid : FLIR
Avionics : Avionics System
Data Bus 2 : 1553B
Milbus : 1553B Connector
Milbus : 1553B Connector
«Equipment» DEP
«Equipment» HMS Controls
UART1
«IMS Processing Element» N1
UART2
«IMS Processing Element» N3
«IMS Processing Element» N2
I/O : STANAG 3190 Connector
I/O : STANAG 3190 Connector
I/O
Data Bus 1 : STANAG 3910
I/O : STANAG 3190 Connector
I/O : STANAG 3190 Connector
«IMS Processing Element» N4
«IMS Processing Element» N5
«IMS Processing Element» N6

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 **Ap**plication **Ex**ecutive 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

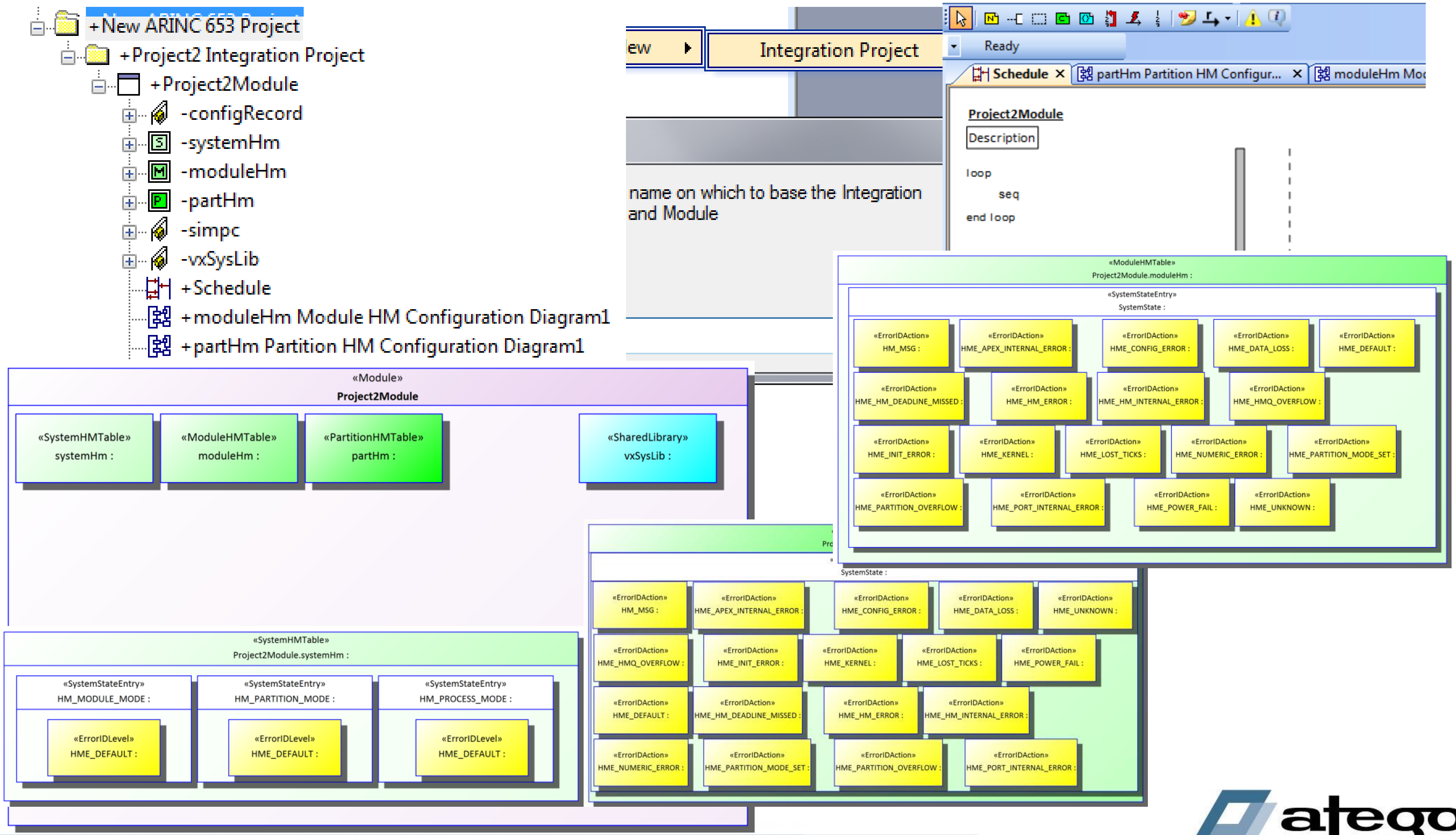
  Productivity

  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


+New ARINC 653 Project
+Project2 Integration Project
+Project2Module
-configRecord
-systemHm
-moduleHm
-partHm
-simpc
-vxSysLib
+Schedule
+moduleHm Module HM Configuration Diagram1
+partHm Partition HM Configuration Diagram1

Integration Project

name on which to base the Integration and Module

Ready

Schedule | partHm Partition HM Configur... | moduleHm Mod

Project2Module

Description

```
loop
    seq
end loop
```

«Module» Project2Module

«SystemHMTable» systemHm : | «ModuleHMTable» moduleHm : | «PartitionHMTable» partHm : | «SharedLibrary» vxSysLib :

«SystemHMTable» Project2Module.systemHm :

«SystemStateEntry» HM_MODULE_MODE : — «ErrorIDLevel» HME_DEFAULT :
«SystemStateEntry» HM_PARTITION_MODE : — «ErrorIDLevel» HME_DEFAULT :
«SystemStateEntry» HM_PROCESS_MODE : — «ErrorIDLevel» HME_DEFAULT :

«ModuleHMTable» Project2Module.moduleHm :

«SystemStateEntry» SystemState :

«ErrorIDAction» HM_MSG : | «ErrorIDAction» HME_APEX_INTERNAL_ERROR : | «ErrorIDAction» HME_CONFIG_ERROR : | «ErrorIDAction» HME_DATA_LOSS : | «ErrorIDAction» HME_DEFAULT :
«ErrorIDAction» HME_HM_DEADLINE_MISSED : | «ErrorIDAction» HME_HM_ERROR : | «ErrorIDAction» HME_HM_INTERNAL_ERROR : | «ErrorIDAction» HME_HMQ_OVERFLOW :
«ErrorIDAction» HME_INIT_ERROR : | «ErrorIDAction» HME_KERNEL : | «ErrorIDAction» HME_LOST_TICKS : | «ErrorIDAction» HME_NUMERIC_ERROR : | «ErrorIDAction» HME_PARTITION_MODE_SET :
«ErrorIDAction» HME_PARTITION_OVERFLOW : | «ErrorIDAction» HME_PORT_INTERNAL_ERROR : | «ErrorIDAction» HME_POWER_FAIL : | «ErrorIDAction» HME_UNKNOWN :

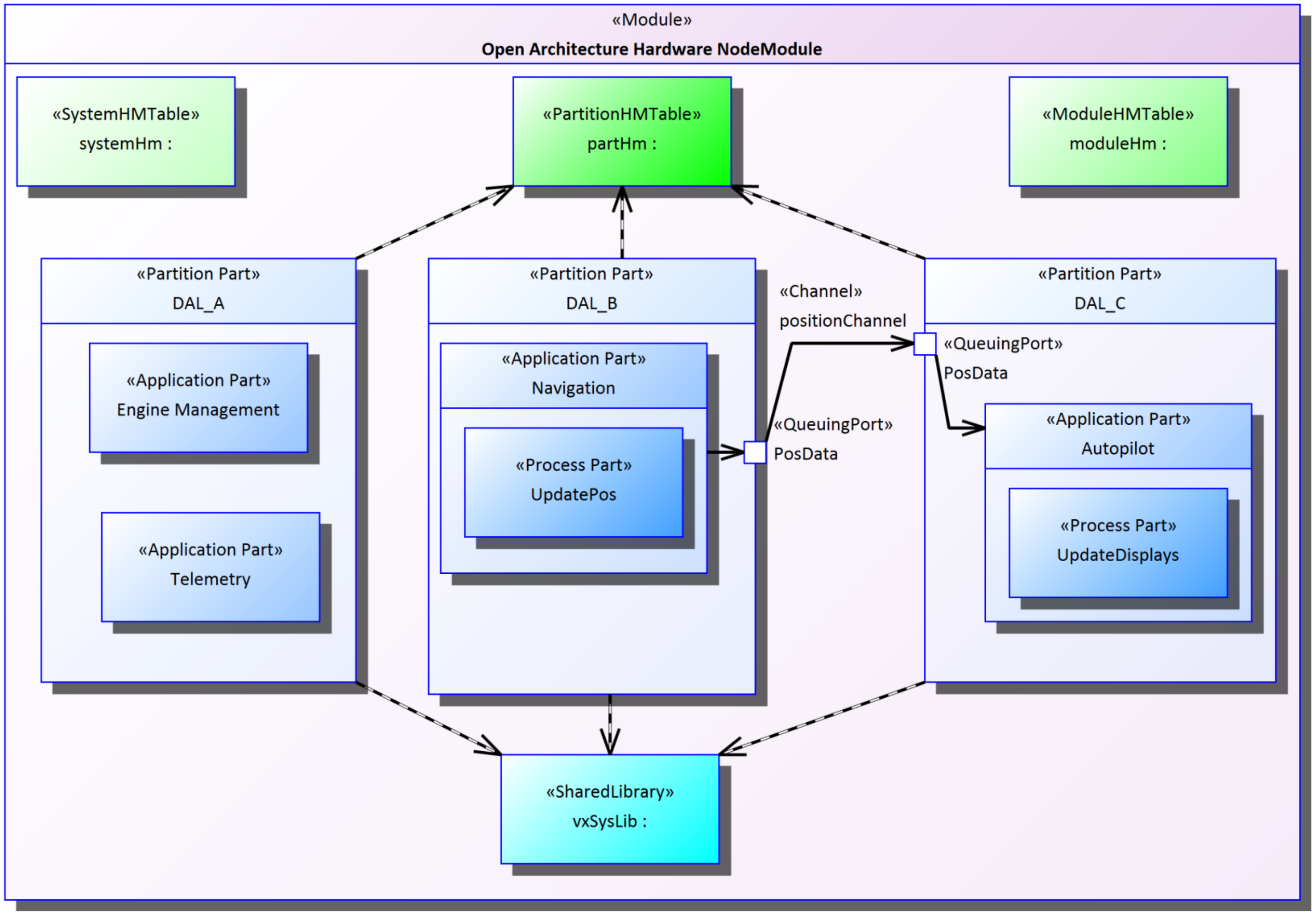
Pro

SystemState :

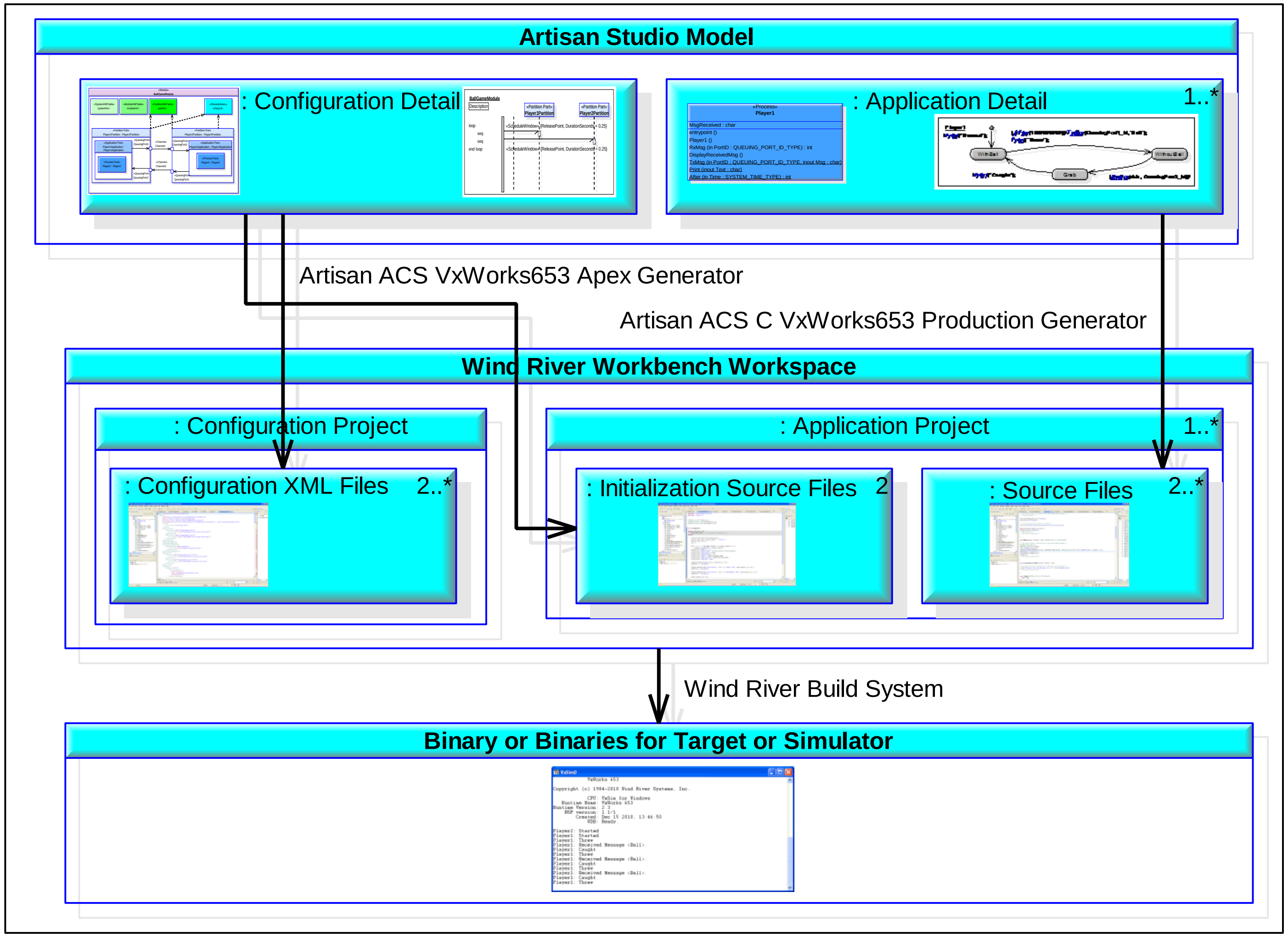
«ErrorIDAction» HM_MSG : | «ErrorIDAction» HME_APEX_INTERNAL_ERROR : | «ErrorIDAction» HME_CONFIG_ERROR : | «ErrorIDAction» HME_DATA_LOSS : | «ErrorIDAction» HME_UNKNOWN :
«ErrorIDAction» HME_HMQ_OVERFLOW : | «ErrorIDAction» HME_INIT_ERROR : | «ErrorIDAction» HME_KERNEL : | «ErrorIDAction» HME_LOST_TICKS : | «ErrorIDAction» HME_POWER_FAIL :
«ErrorIDAction» HME_DEFAULT : | «ErrorIDAction» HME_HM_DEADLINE_MISSED : | «ErrorIDAction» HME_HM_ERROR : | «ErrorIDAction» HME_HM_INTERNAL_ERROR :
«ErrorIDAction» HME_NUMERIC_ERROR : | «ErrorIDAction» HME_PARTITION_MODE_SET : | «ErrorIDAction» HME_PARTITION_OVERFLOW : | «ErrorIDAction» HME_PORT_INTERNAL_ERROR :

Example Configuration Diagram

- Module, Partition, Application, Process, Port, Channel



Summary of Tool Chain Artifacts


Artisan Studio Model
: Configuration Detail
: Application Detail
1..*
Artisan ACS VxWorks653 Apex Generator
Artisan ACS C VxWorks653 Production Generator
Wind River Workbench Workspace
: Configuration Project
: Application Project
1..*
: Configuration XML Files
2..*
: Initialization Source Files
2
: Source Files
2..*
Wind River Build System
Binary or Binaries for Target or Simulator

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