

# Executable Modelling with UML and Ada: The X Factor

John Rowlands, Air Systems

**BAE SYSTEMS**

Copyright BAE SYSTEMS Public Limited Company 2006. The copyright in this document, which contains information of a proprietary nature, is vested in BAE SYSTEMS Public Limited Company, and its contents may not be reproduced, either wholly or in part, in any way whatsoever, without the prior written permission of BAE SYSTEMS Public Limited Company. BAE SYSTEMS Warton Aerodrome, Preston, Lancashire PR4 1AX, England

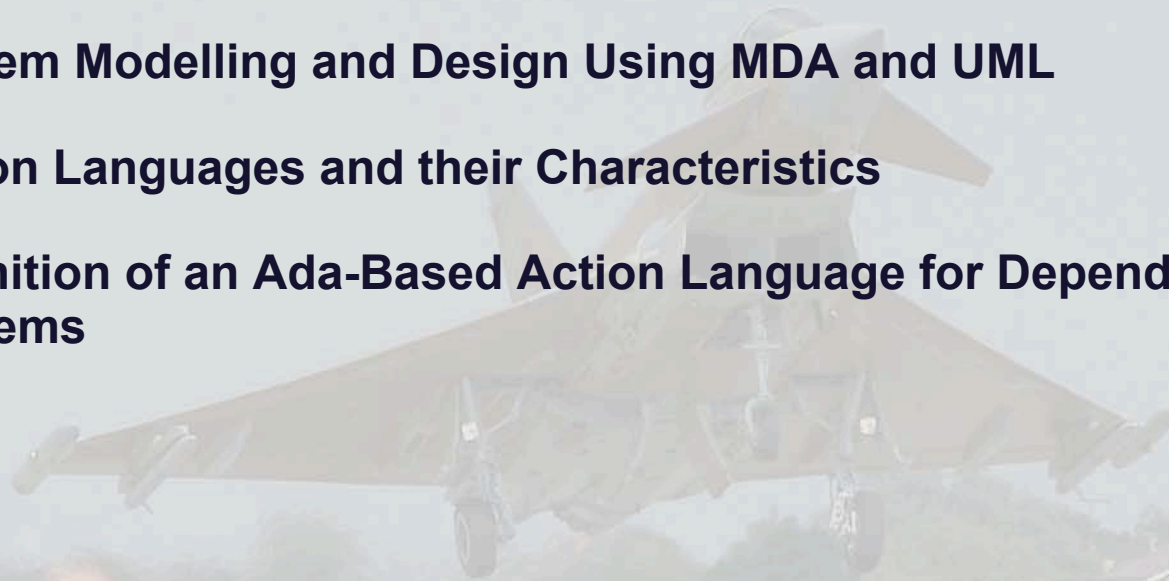


W374B, Warton Aerodrome, Preston, PR4 1AX  
[john.rowlands@baesystems.com](mailto:john.rowlands@baesystems.com)

01772-854630

# begin

- **The Need for Executable Specifications**
- **System Modelling and Design Using MDA and UML**
- **Action Languages and their Characteristics**
- **Definition of an Ada-Based Action Language for Dependable Real-Time Systems**



# eXecutable Specifications – why?

- **The X Factor**
- **Early Validation of the System**
  - Can be subjected to testing
  - Can be combined into an Integrated System and Environment Model
  - Supports Capability-Based Acquisition
- **An Abstract View of the System Being Designed**
  - Focuses on the architecture of the system
  - Easy to navigate
  - Removes unnecessary implementation detail
  - Platform Independent (Hardware, Operating System)

**We use the Principles of MDA to Support Executable Specifications**

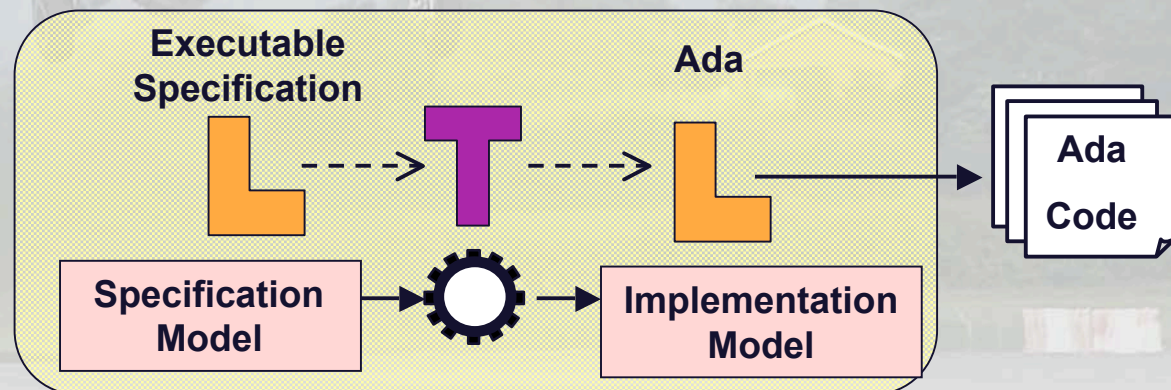
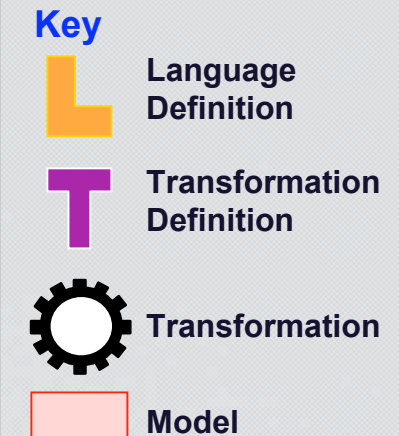
# What is Model Driven Architecture (MDA)?

MDA is a set of techniques that enable us to

- Rapidly construct very different types of models for the system:  
The Language with which the subject matter is captured
  - Specification
  - Implementation
- Capture fundamental aspects of the system (structure/behaviour) and any constraints

This allows us to

- Model in a Platform Independent Way (*hardware type and software language*)
- Model the transformation (e.g. rules for auto-code and migration)
- Standardise the way in which we capture these models



**A Unified Framework for Defining Models, Constraints and Transformations**

# How do we Specify an Executable Model?

## UML Defines the Architecture of a System

## Model Behaviour is Defined Using an **Action Language**

- Bodies of object operations
- Actions on activity diagrams and state diagrams

## It Should Be Possible to Map to any Implementation Language and any Form of Hardware and Firmware

- General Purpose Processor
- FPGA
- COTS Operating System
- Programming Languages other than Ada

**Existing UML tools either use specific programming languages (too low level) or more abstract action languages that do not exploit the strengths of Ada (too informal).**

**Ideally, the strengths of Ada should be preserved in a high level action language**

# Key Characteristics of an Action Language for High Integrity Embedded Systems Development

## Ada



- **Statically checkable**

- Declaration before use
- Strong typing



- **Readable**

- Suited to the implementation of systems that will be developed and maintained for many years (typically 30)
- The person who extends the system is often not the person who developed the initial system



- **Well-defined semantics**

- Reduces misinterpretation
- Eases automated checking



- **Works at the UML Level of Abstraction**

- Knows about Classes, Attributes, Associations, etc.

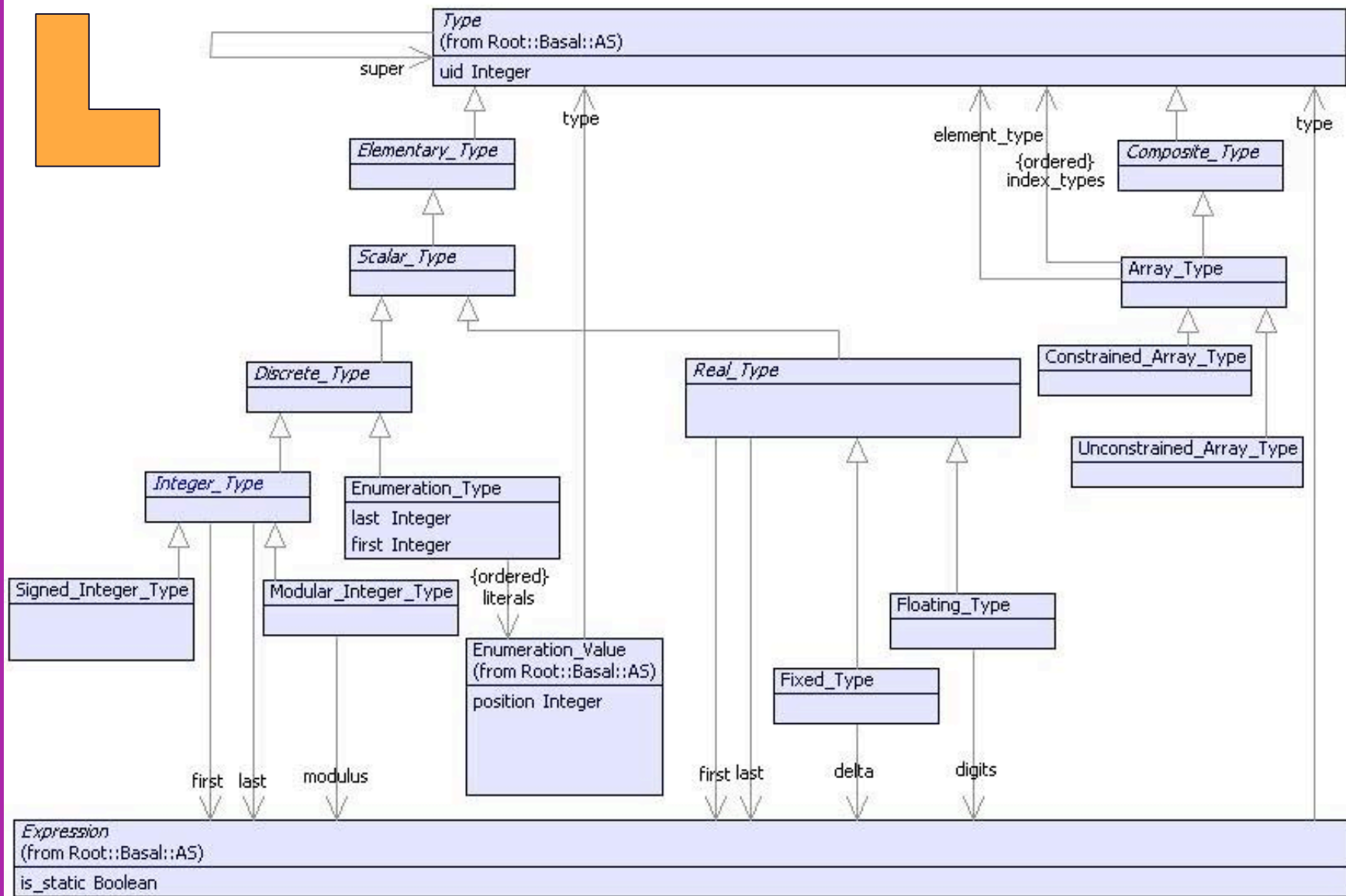
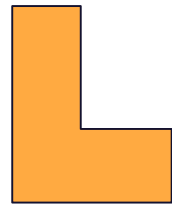


- **Defined using the principles of MDA**

- Amenable to transformation and checking

XMF Mosaic 

# The Language of (SPARK) Ada Types



# Applying a Transformation

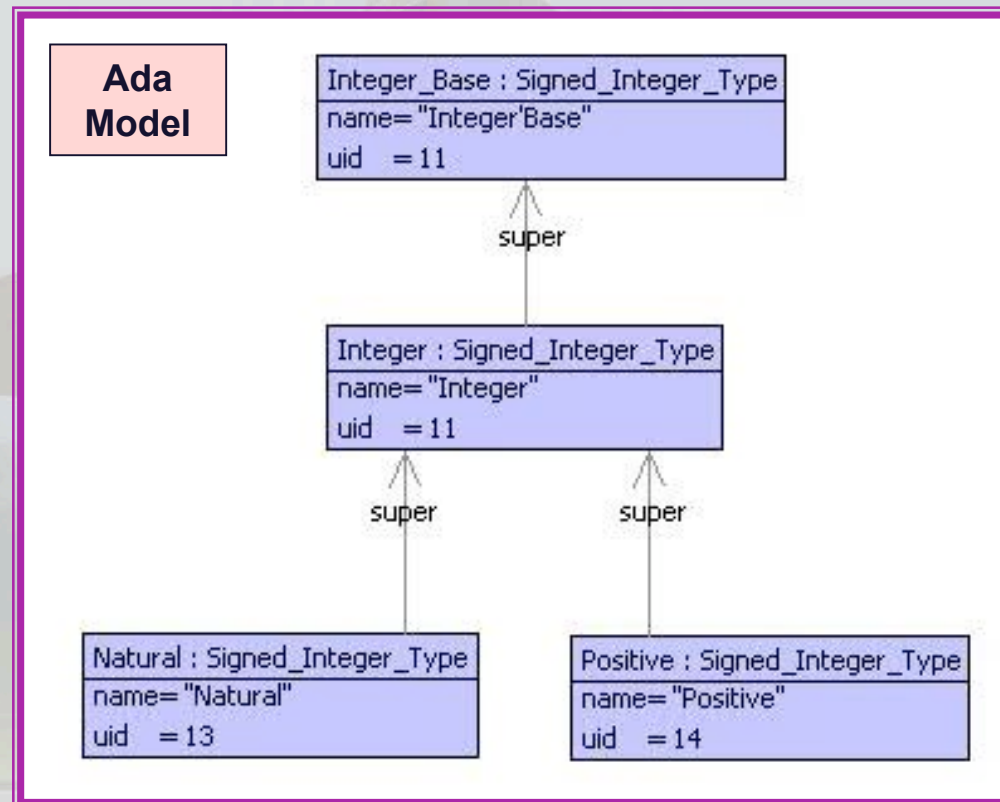
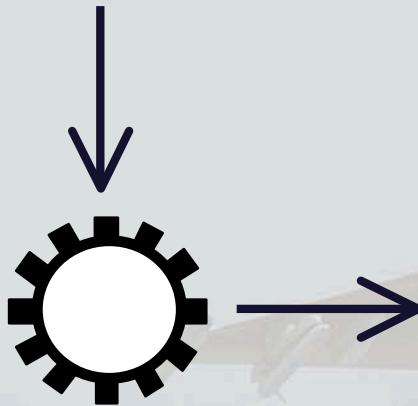
```

type Integer is range -4194304..4194304-1;

subtype Natural is Integer range 0..Integer'Last;

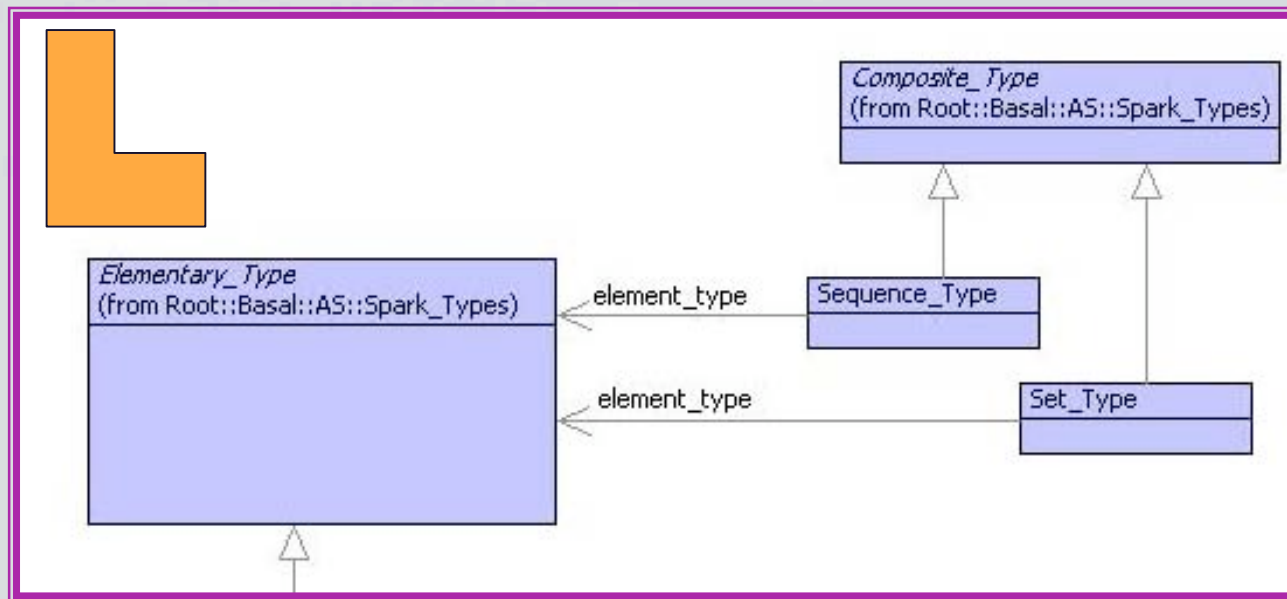
subtype Positive is Integer range 1..Integer'Last;

```



- Statically checkable
- Readable
- Well-defined semantics
- Works at the UML Level of Abstraction
- Defined using the principles of MDA

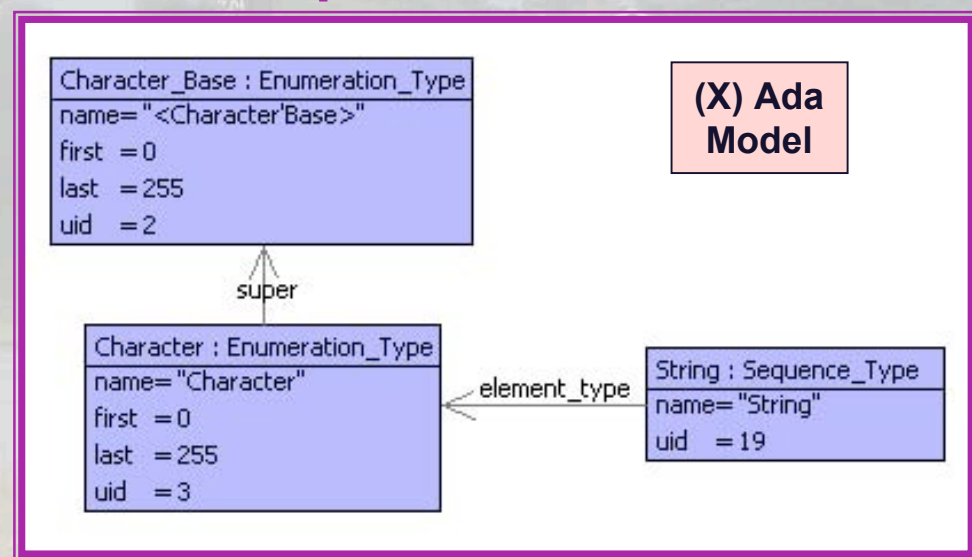
# Extending the Language of Types



↑ Instance of

```
type String is seq of Character;
```

- ✓ Statically checkable
  - ✓ Readable
  - ✓ Well-defined semantics
  - ✓ Works at the UML Level of Abstraction
  - ✓ Defined using the principles of MDA
- 



# Performing Constraint Checks

- Type-Checking is an example of Checking Constraints
- Uses a Rules-Based Language (Similar to Prolog) – Called XRules
- Formalises the Ada Type-Checking Rules

*“If the expected type for a construct is a specific type T, then the type of the construct shall resolve either to T ...”*

```
@Rule Check_Type_Conformance(Source_Type,Target_Type)
  Base_Type(Source_Type,T)
  Base_Type(Target_Type,T)
end
```

*“... or to a universal type that covers T”*

```
@Rule Check_Type_Conformance
  (Signed_Integer_Type[name="<Universal_Integer>"],
   Integer_Type[ ])
```

# Summary

- **Executable Specifications Allow Early Validation of an Abstract View of the System Under Design**
- **UML Needs to be Supported by an Appropriate Action Language**
  - Ada Has Many (not all) of the Qualities Needed
- **MDA and its Supporting Tools Allow**
  - Transformation Between Languages
  - Constraint Checking
  - Formalisation of Languages, Transformations and Checks
  - Extension of Languages
- Investigation of the Implications of Evolving Ada into a Practical UML Action Language

**end;**

- **Questions?**

