

# ***F-16 Modular Mission Computer Application Software***



***Achieving Cross-Platform Compatibility with  
Increased Productivity and Quality using the  
OMG's Model Driven Architecture***

***Lauren E. Clark  
Chief Engineer  
F-16 Modular Mission Computer Software  
Lockheed Martin Aeronautics Company***

***Bary D. Hogan  
Methodology Lead  
F-16 Modular Mission Computer Software  
Lockheed Martin Aeronautics Company***

***Terry Ruthruff  
Staff Specialist  
Software Engineering Core  
Lockheed Martin Aeronautics Company***

***Allan Kennedy  
President  
Kennedy Carter Limited***



- **Basic Software Components**
- **Cross-Platform Compatibility: The Goal**
- **The eXecutable MDA Approach:**
  - *eXecutable UML Modeling*
  - *Platform Specific Mapping (Design Tagging)*
  - *Automatic Code Generation*
- **Advantages of the eXecutable MDA Approach**



## Application Software:

- High-level software that is unique to the application(s) for which the embedded computer (i.e. subsystem) exists
- 80-90% of the total software (in terms of long-term development cost)

## Software Execution Platform:

- Low-level software, the purpose of which is to allow the Application Software to run on the hardware

# Software Execution Platform



**Software Execution Platform:**

- Low-level software, the purpose of which is to allow the Application Software to run on the hardware



## Board Support Package:

- Lowest-level boot software / firmware that allows all other software (including the Operating System) to be loaded into memory and begin executing
- Unique to the hardware; and usually delivered with the hardware (located in some type of ROM)

## Built-In Test (BIT):

- Low-level software that detects and reports hardware errors
- Unique to the hardware; and usually delivered with the hardware



## Operating System:

- Low-level software that, once booted, manages all other software (this management involving such things as multitasking, memory sharing, I/O interrupt handling, error and status reporting, etc.)
- Unique to the hardware (i.e. it must at least be ported to each new hardware platform); and sometimes delivered with the hardware





## Device Drivers:

- Low-level software that manages the input from and output to the various external devices in support of the Application Software
- Unique to the hardware; but usually not delivered with the hardware



## Software Architecture:

- Low-level software providing the framework within which the Application Software executes
- Provides execution control, data / message management, error handling, and various support services to the Application Software
- Assumes a particular Application Software language
- Unique to the hardware; but, since it must support all requirements levied by the Application Software, is not delivered with the hardware





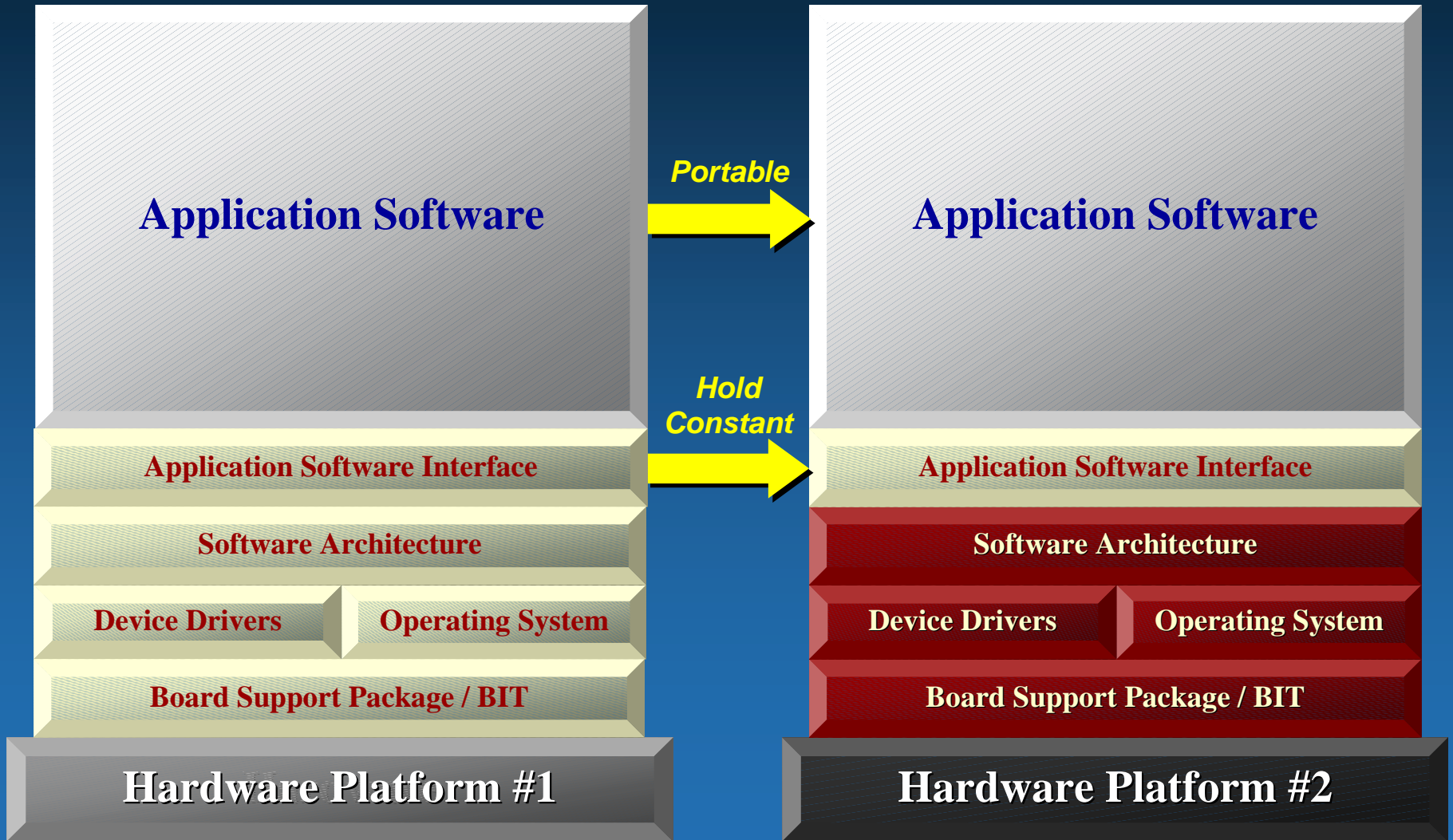
## Application Software Interface:

- The boundary between the Application Software and the Software Execution Platform
- The specified methods by which the Application Software can make requests and use the services of the Software Execution Platform and the Software Execution Platform can provide its services to the Application Software
- This interface is specified by the Software Execution Platform

# Cross-Platform Compatibility: The Usual Approach



**Maintain a constant Application Software Interface**



# Cross-Platform Compatibility Issues



**Can a constant  
Application Software Interface  
always be maintained?**

## Consider...

- What if the language or operating system becomes obsolete?
- What if it is necessary to port even a part of the Application Software to a legacy platform not having the resources to support the newer Software Execution Platforms?

# Cross-Platform Compatibility Issues



***Even if it were possible, would one always want to maintain a constant Application Software Interface?***

## **Consider...**

- What if hardware or Software Execution Platform changes could provide more Application Software capability, but only by means of changing the Application Software Interface?



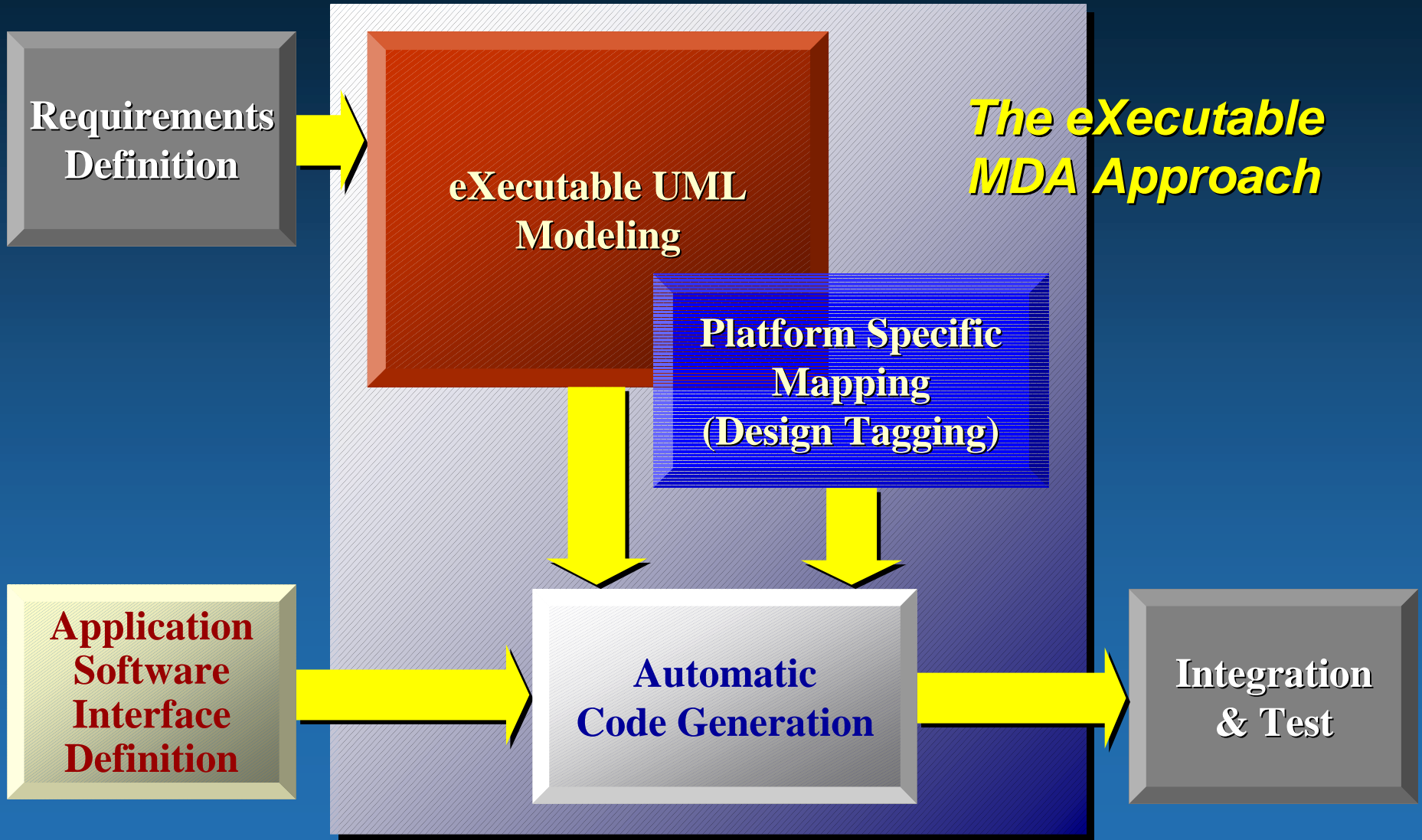
# Cross-Platform Compatibility: The Goal



*The goal should be to provide cross-platform compatibility of Application Software **despite any Implementation, or platform specific, changes:***

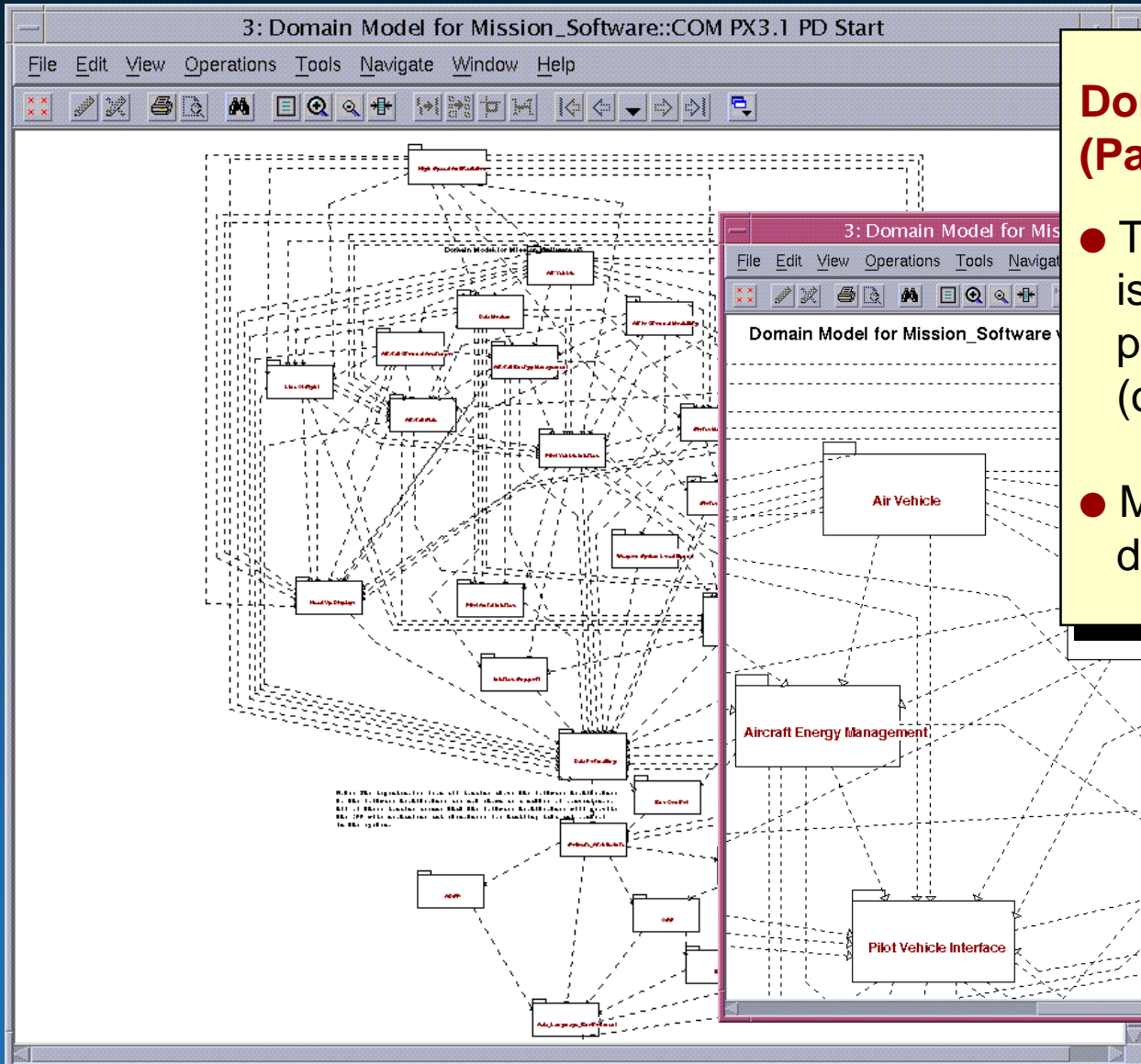
*that is, changes to the Hardware Platform, the Software Execution Platform, **or the Application Software Interface***

# eExecutable MDA: Application Software Development



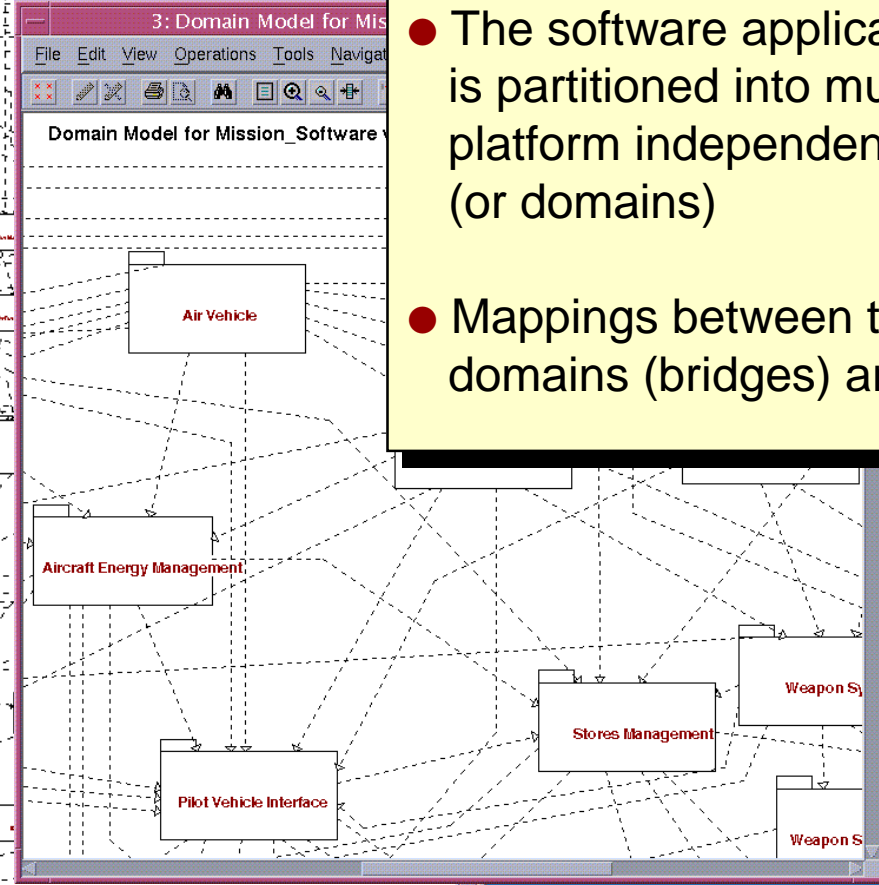


# eExecutable UML Modeling: Domain Model

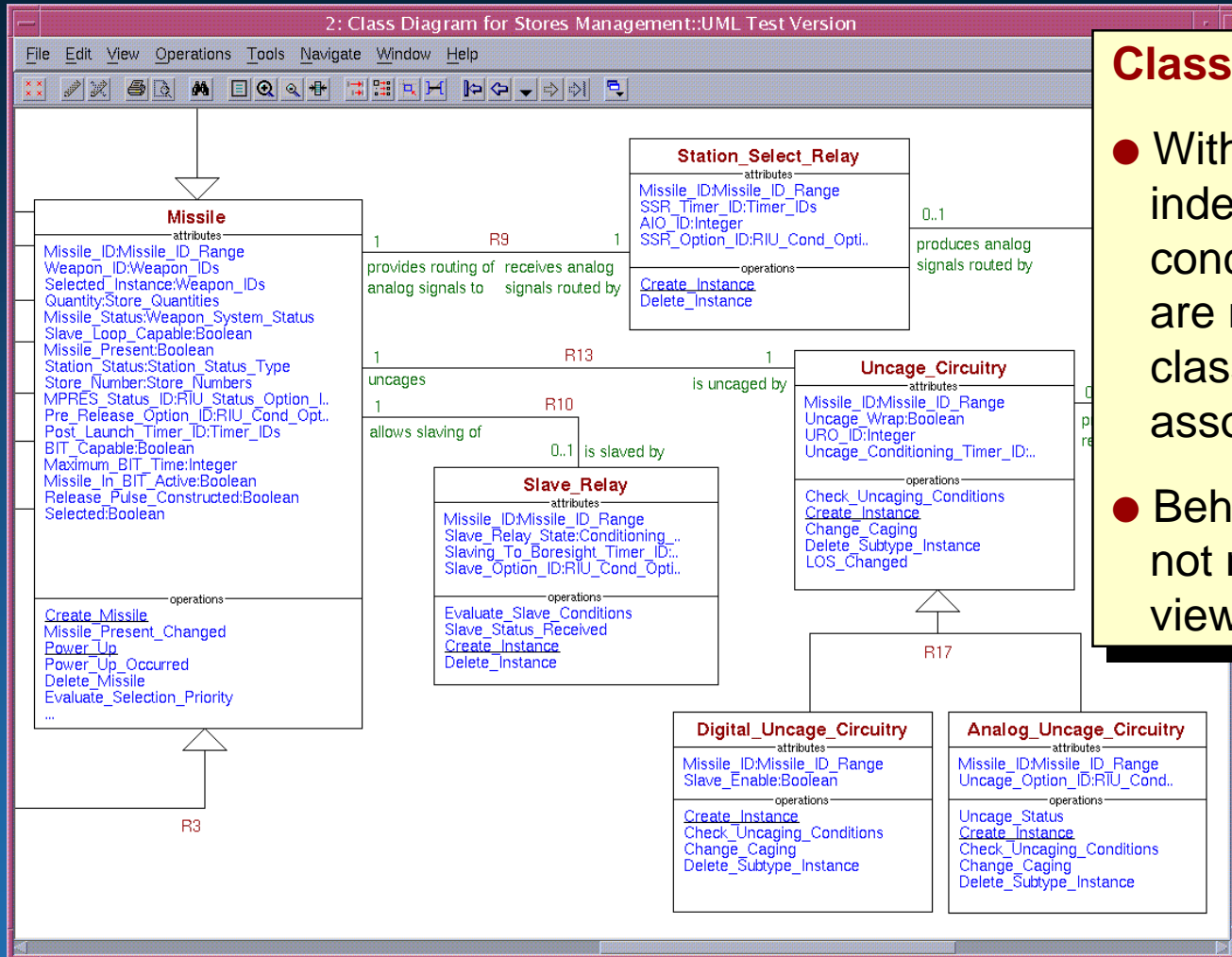


## Domain Model (Package Diagram):

- The software application space is partitioned into multiple platform independent models (or domains)
- Mappings between the domains (bridges) are defined



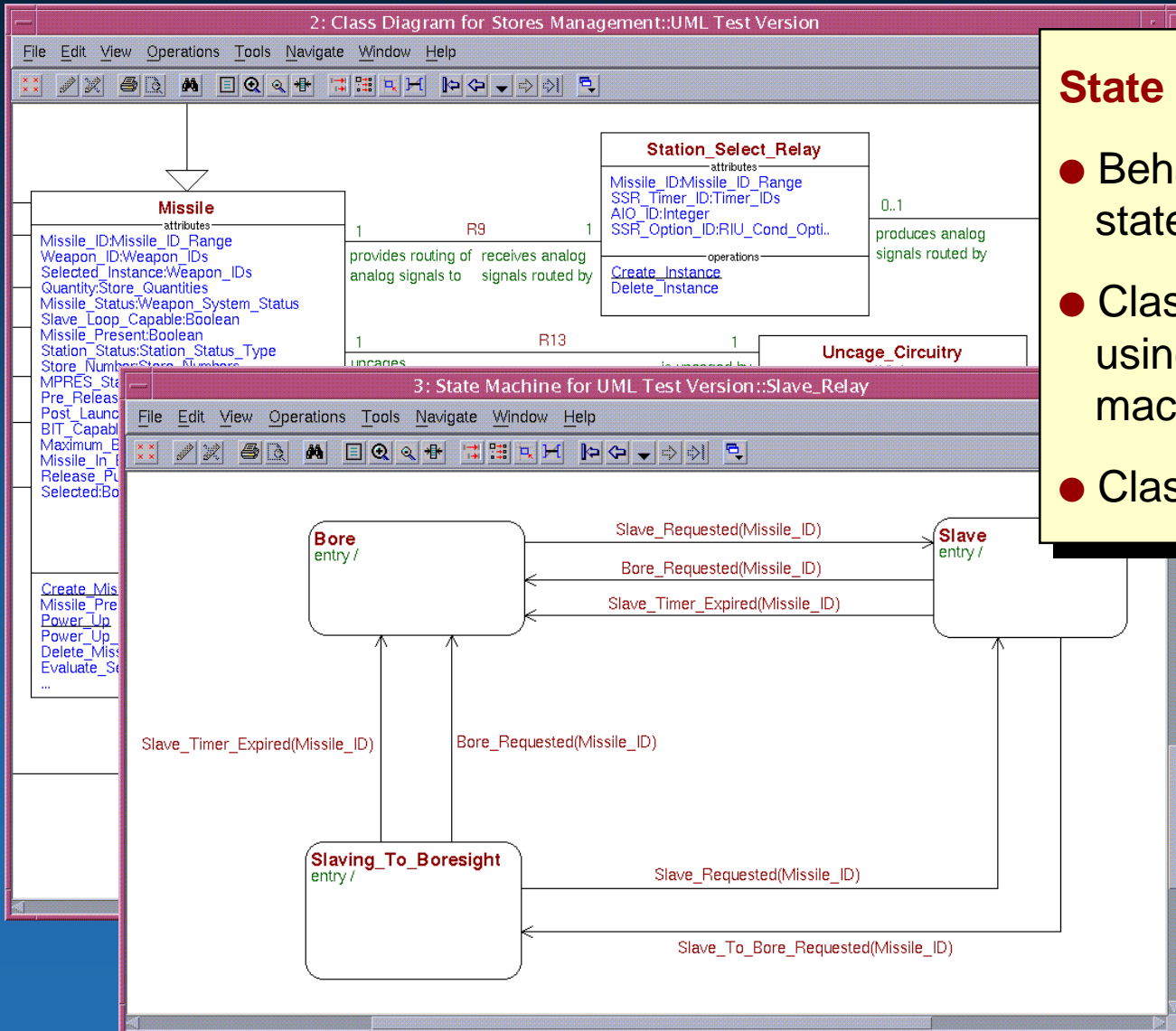
# eExecutable UML Modeling: Class Diagrams



## Class Diagrams:

- Within each platform independent domain model, conceptual entities are modeled first: classes, attributes, and associations are abstracted
- Behavior, though considered, is not modeled explicitly in this view

# eExecutable UML Modeling: State Charts



- ### State Charts:
- Behavior is formalized during state modeling
  - Class lifecycles are modeled using signal-driven state machines
  - Class operations are defined

# eExecutable UML Modeling: ASL



The screenshot displays three overlapping windows from a UML modeling tool:

- 2: Class Diagram for Missile**: Shows a class hierarchy with **Missile** as the base class. Attributes include `Missile_ID:Missile_ID_Range`, `Weapon_ID:Weapon_IDs`, `Selected_Instance:Weapon_IDs`, `Quantity:Store_Quantities`, `Missile_Status:Weapon_System_Status`, `Slave_Loop_Capable:Boolean`, `Missile_Present:Boolean`, `Station_Status:Station_Status_Type`, `Store_Number:Store_Numbers`, `MPRES_State`, `Pre_Release`, `Post_Launch`, `BIT_Capable`, `Maximum_B`, `Missile_In`, `Release_P`, and `Selected:Bo`. A note indicates it provides routing of analog signals to another component.
- 3: State**: Shows a state diagram with states **Bore** and **Slaving\_To\_Boresig**. Transitions include `Slave_Timer_Expired(Missile_ID)` and `Bore`.
- 6: Instance State Details for Missile\_Type\_In\_Launch::Post\_Release**: Contains ASL code for the `Post_Release` state. The code includes metadata (Database, Domain, Version, Class, State), a description, and a detailed action block with conditional logic for handling missile status, ANM instances, and timer events.

## Action Specification Language:

- State actions and class operations are specified using a precise Action Specification Language (ASL)
- ASL is a higher order and much simpler language than a typical high order language (e.g. C++)
- ASL deals with object oriented concepts, not implementation concepts
- ASL conforms to the UML Precise Action Semantics



# eExecutable UML Modeling: Simulation



The main window of the iUML Simulator. It features a menu bar with 'Application', 'View', 'Options', and 'Help'. A text area shows the status: 'Busy.....' followed by 'Stopped : Platform 0 Domain ASM Scenario Setup D5\_setup\_2'. Below this is a control panel with buttons for 'Execution' (Continue, Step Invoke, Step Event, Step Over, Step In), 'Show' (Object, Assigners, Breakpoints, Events, Trace), 'Breakpoints' (Add, Add ASL, Add Event, Clear One), and 'Stimulus' (Timer, External, Schedule).

The 'iUML Simulator - ASL Code' window displays the following code snippet:

```
Platform 0 Domain : ASM
Scenario Setup : D5_setup_2 Line : 23
Show Local Variables

#16     ANOP_Instance = find-one Option_Profile
#17     if ANOP_Instance = UNDEFINED then
#18         New_Option_Profile = create
#19         Profile_ID = 1 \
#20         & Cooling_Option = 'Warn
#21         & TD_Option = 'BP' \
#22         & Current_State = 'Inact
#23     -->endif
#24
#25
#26     ANXP_Instance = find-one Prof
#27     if ANXP_Instance = UNDEFINED
#28         New_Profile = create Profi
#29         Profile_ID = 1 \
#30         & Cooling_Option = 'Cool
```

The 'iUML Simulator - Instance Local Variables' window shows a table of local variables for the current domain and function.

Variable Name	Type	Value
ANSP_Instance	Instance	UNDEFI
New_Slave_Profile	Instance	DEFINE
ANOP_Instance	Instance	UNDEFI
New_Option_Profile	Instance	DEFINE
ANXP_Instance	Instance	UNDEFI
New_Profile	Instance	UNDEFI

The 'iUML Simulator Instance New\_Slave\_Profile' window displays a table for the 'Instance New\_Slave\_Profile'.

Profile_ID	LOS_Option	FOV_Option	Slave_Requested	Auto_Mode	Current_State	R4 Missile Type
1	Slave	Spot	TRUE	FALSE	Inactive	

- Simulation:**
- Since a precise Action Specification Language is used, models are executable and therefore may be simulated
  - Simulation features resemble those of a high order language debugger
  - Models may be validated long before they are implemented

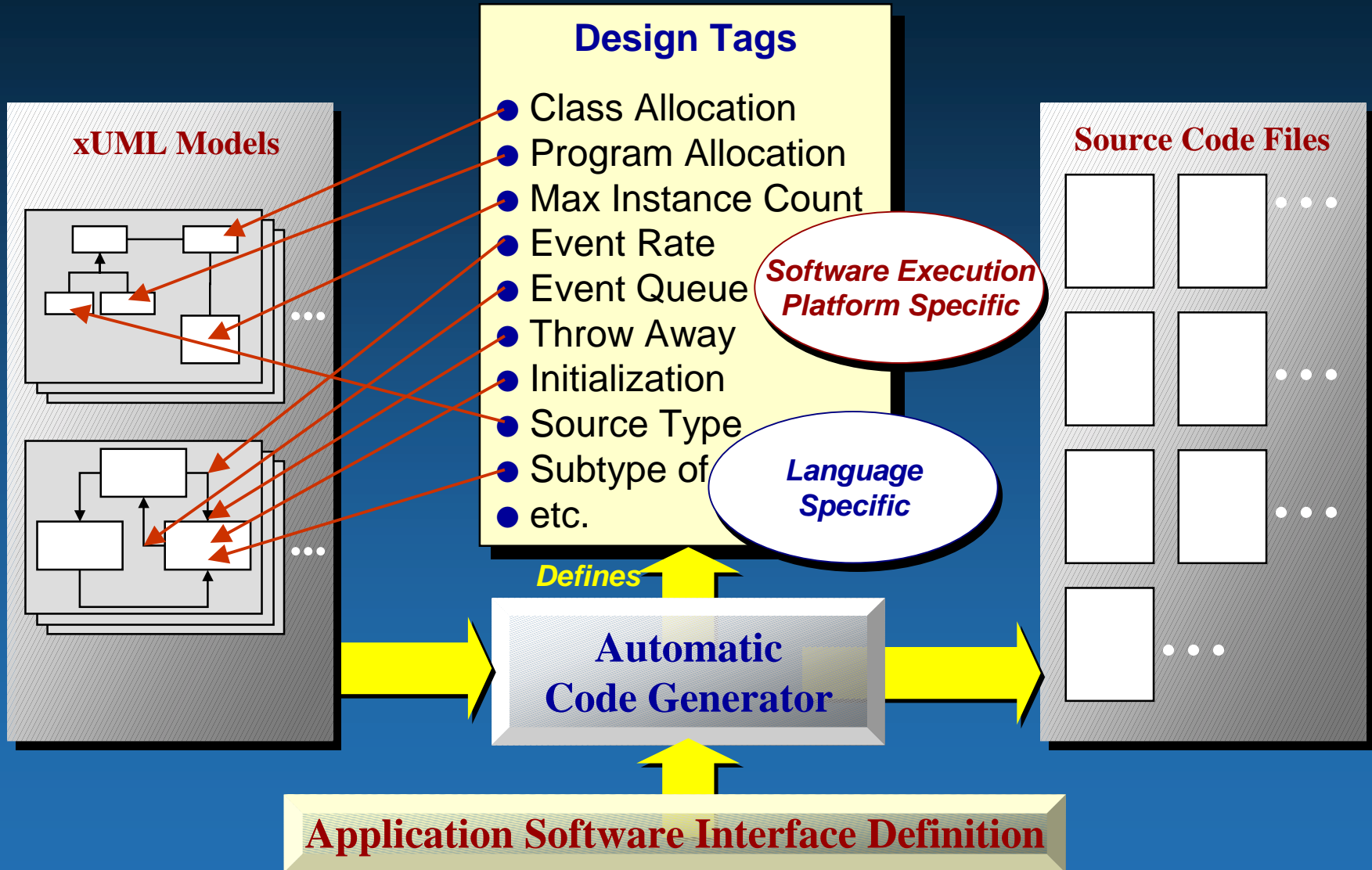


## eXecutable UML Modeling

- xUML models are a complete representation of the application space (not a top-level or preliminary design)
- Modeling is performed using a Unified Modeling Language (UML) representation
- Modeling makes use of a precise Action Specification Language (ASL) and is therefore executable (providing early validation of the models)
- Each xUML model is a Platform Independent Model (PIM), or completely implementation-independent (i.e. independent of the hardware platform, the software execution platform, and the application software interface)



# Design Tagging: Specifying the PIM to PSM Mapping



# Design Tagging: Specifying the PIM to PSM Mapping



**2: Tag Group Details for UML Test Version::MMC**

Database : <Database Name>  
Domain : Stores Management, ASM  
Version : 8: UML Test Version  
Tag Group : MMC Class Descriptions

**Description**  
MMC Code Generator Tags

**Tags**  
Event Queue  
Event Rate  
Initialization  
Maximum Instance Count  
Persistent  
Queued Event Count  
Source Type  
Subtype of  
Throw Away  
Class Allocation

**2: Class Details for UML Test Version::Missile**

Database : <Database Name>  
Domain : Stores Management, ASM  
Version : 8: UML Test Version  
Class : 30 Missile

**Description**  
The Missile object represents a missile that is in inventory.

**Attributes**  
Missile\_ID  
Telemetry Present  
Safe\_To\_Release  
Critical\_HW\_Passed\_BIT  
AUR\_Ready  
Power\_Switch\_ID  
Power\_On\_Timer\_ID  
Communication\_Status  
Digital\_Autopilot\_On  
Current State (Status)

**Identifiers**  
1 (Generalisation R21) (Preferred)  
Missile\_ID

**Exception Handling Code**  
<Exception Code>

**Linked Requirements**

Role	Number	Name
------	--------	------

**Attached Tags**

Name	Value
(MMC Class Des..)Maximum Instance Count	6
(MMC Class Des..)Persistent	True
(Capability/Co..)Include_Missile	True
(MMC Program A..)WM2	Home

## Design Tagging:

- Design tag values represent implementation-specific design decisions
- Design tagging is applied to, but not embedded in, the xUML models (tags and tag values may be included or excluded)
- Code Generator assumes the most standard implementation, such that only exceptions must be tagged



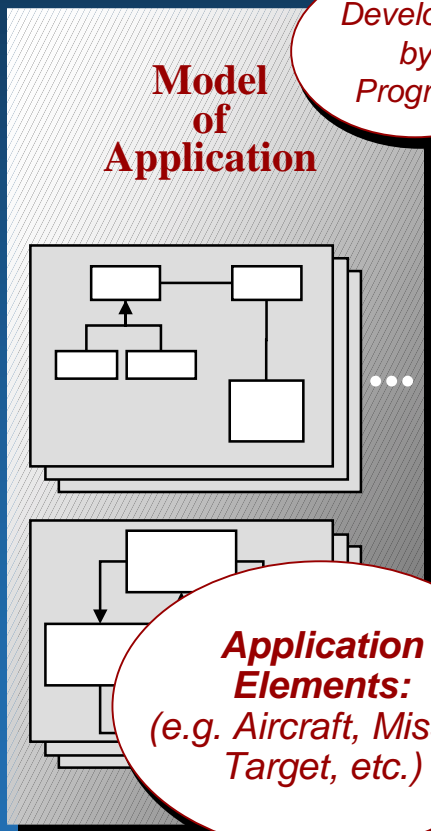
## Platform Specific Mapping (Design Tagging)

- Whereas xUML modeling is implementation-independent, Design Tagging is implementation-dependent (i.e. specific to a particular Application Software Interface)
- Implementation-specific design decisions (only those needed to support code generation) are made during Design Tagging, and are represented with design tag values that are applied to the xUML models
- The most standard implementation is always assumed by the code generator, such that only exceptions must be tagged
- Design Tagging is overlaid on (not embedded in) the xUML models, such that it may be included or excluded

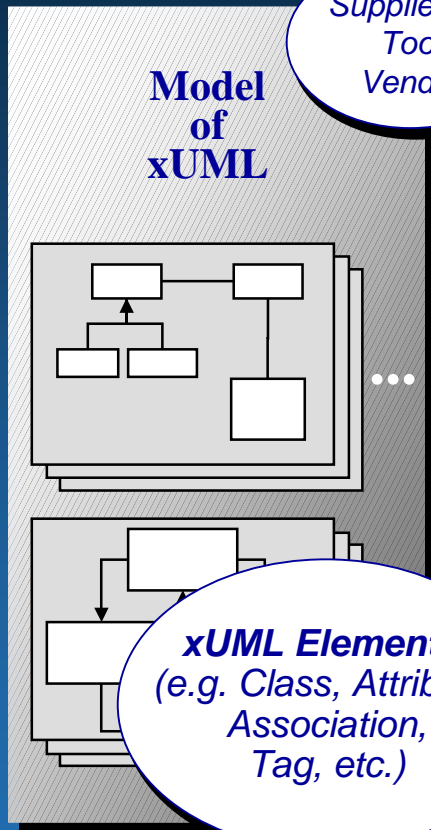
# Automatic Code Generation: 3 Levels of Models



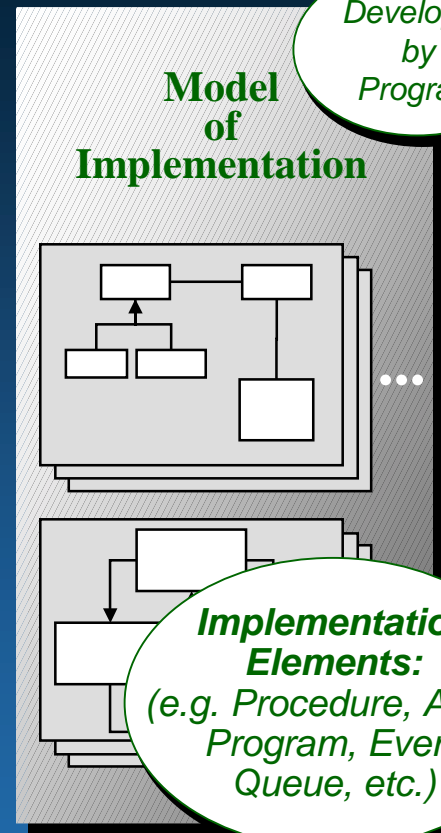
## Level 1



## Level 2

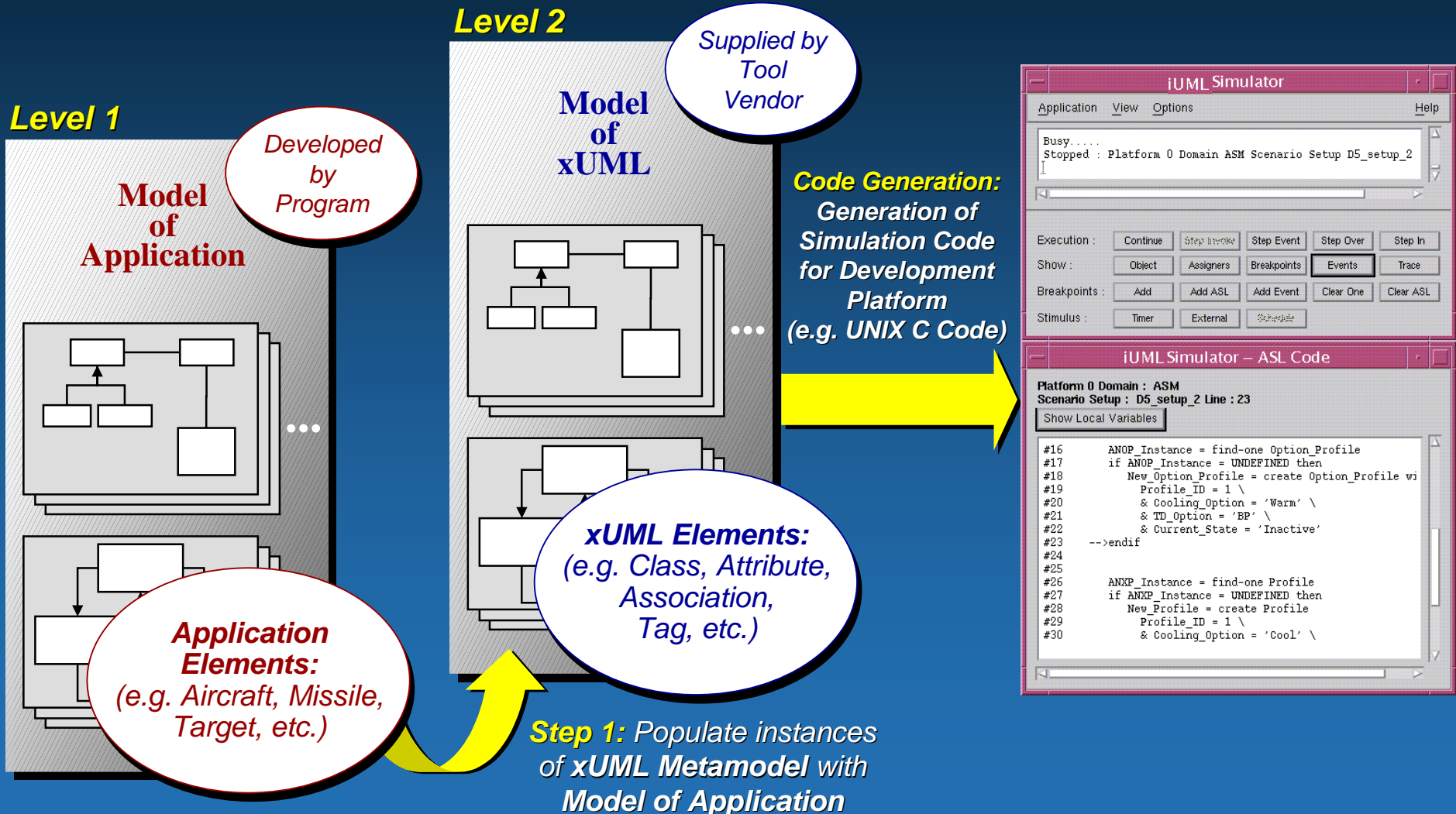


## Level 3



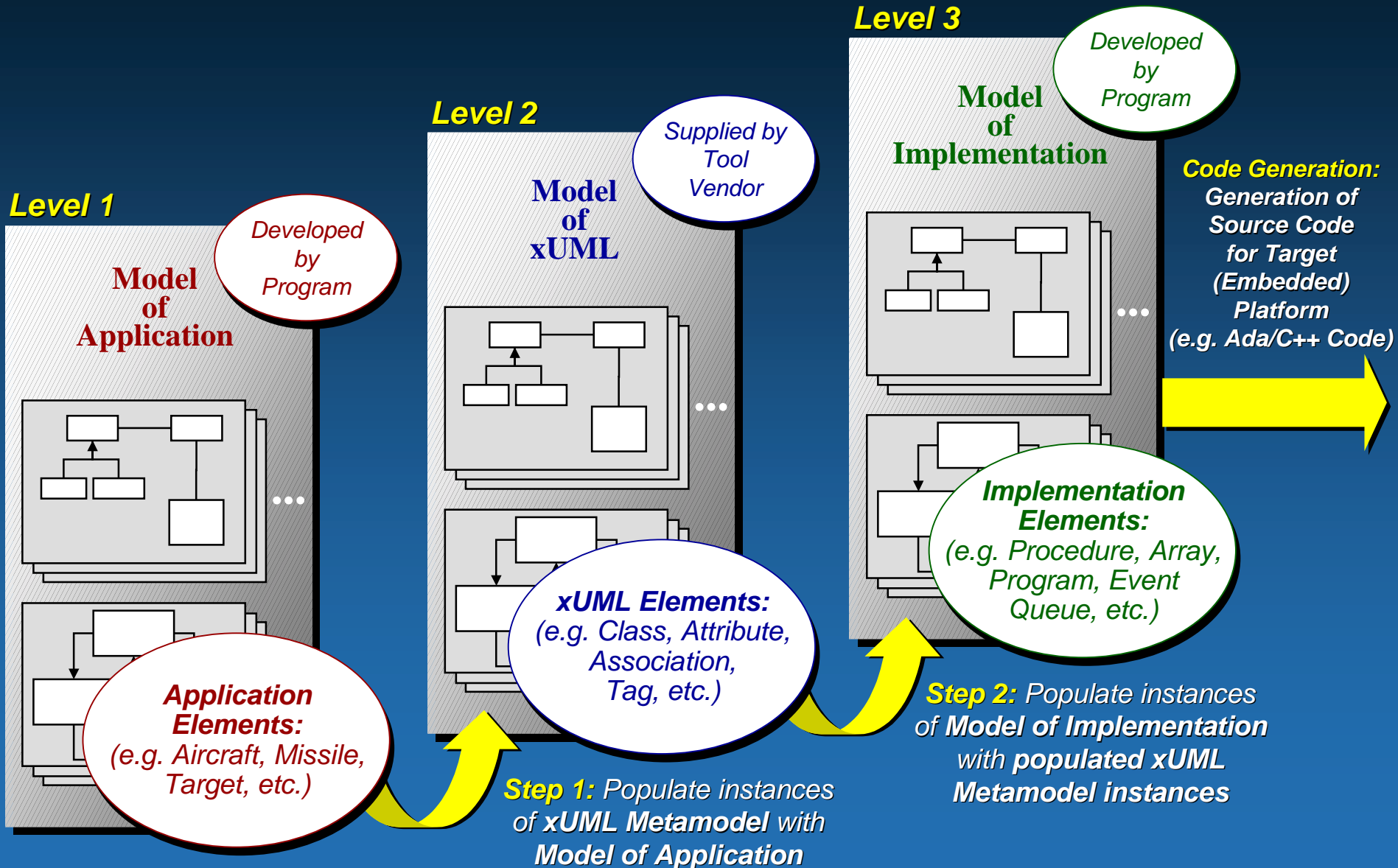
# Automatic Code Generation: Level 2 - Simulation Code

When we say that “xUML models are executable” we mean that “executable code can be automatically generated from them”



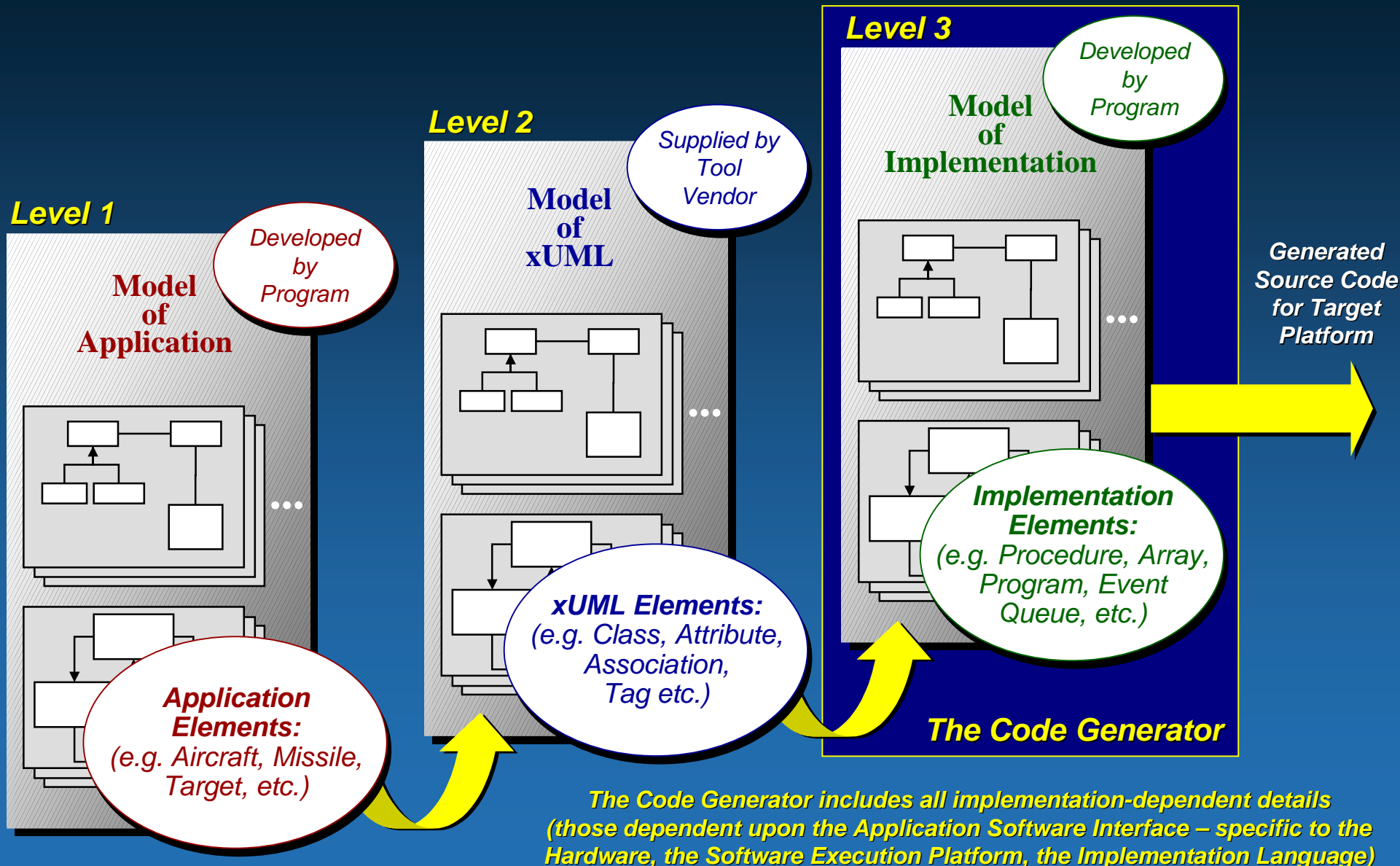


# Automatic Code Generation: Level 3 - Target Code

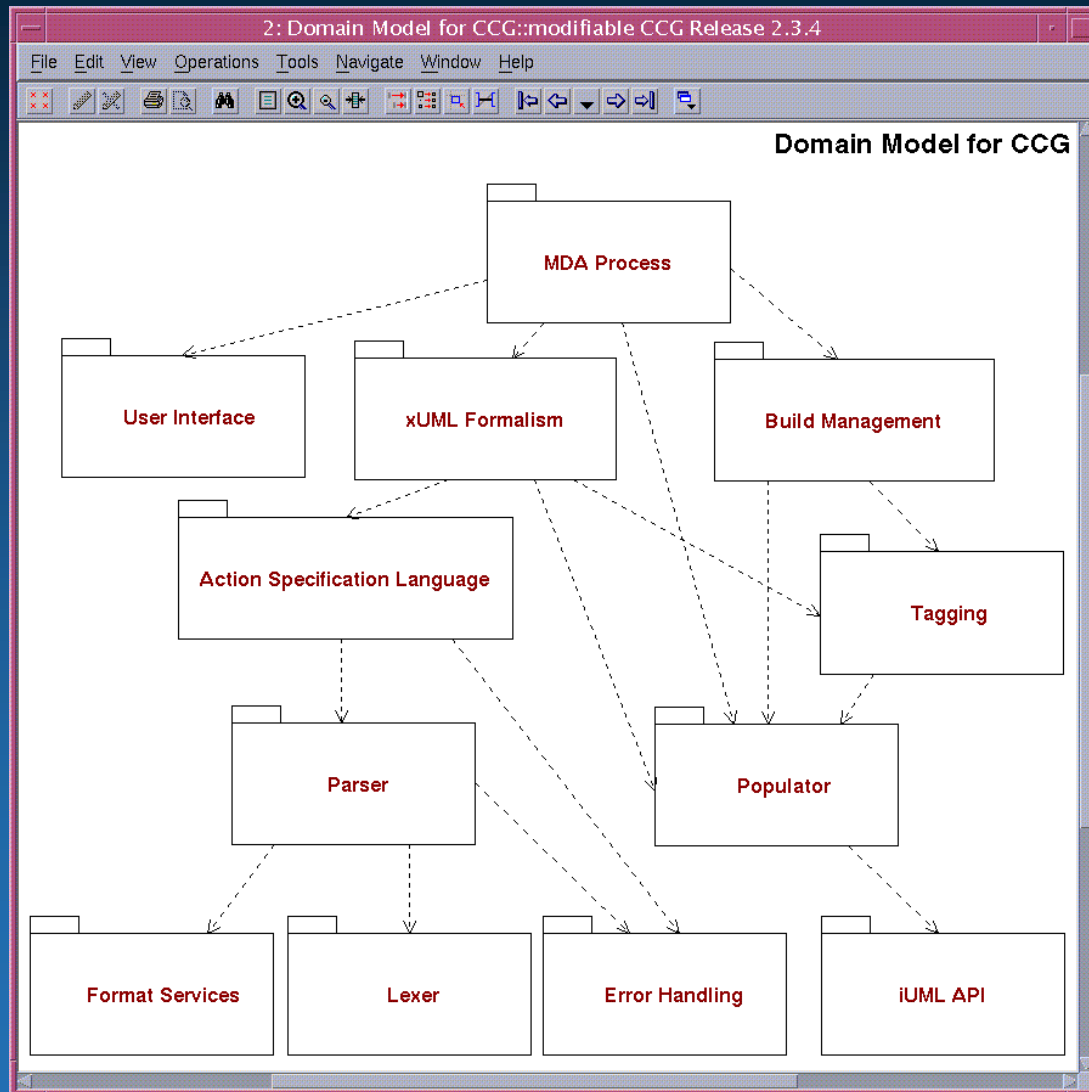




# Automatic Code Generation: The Code Generator



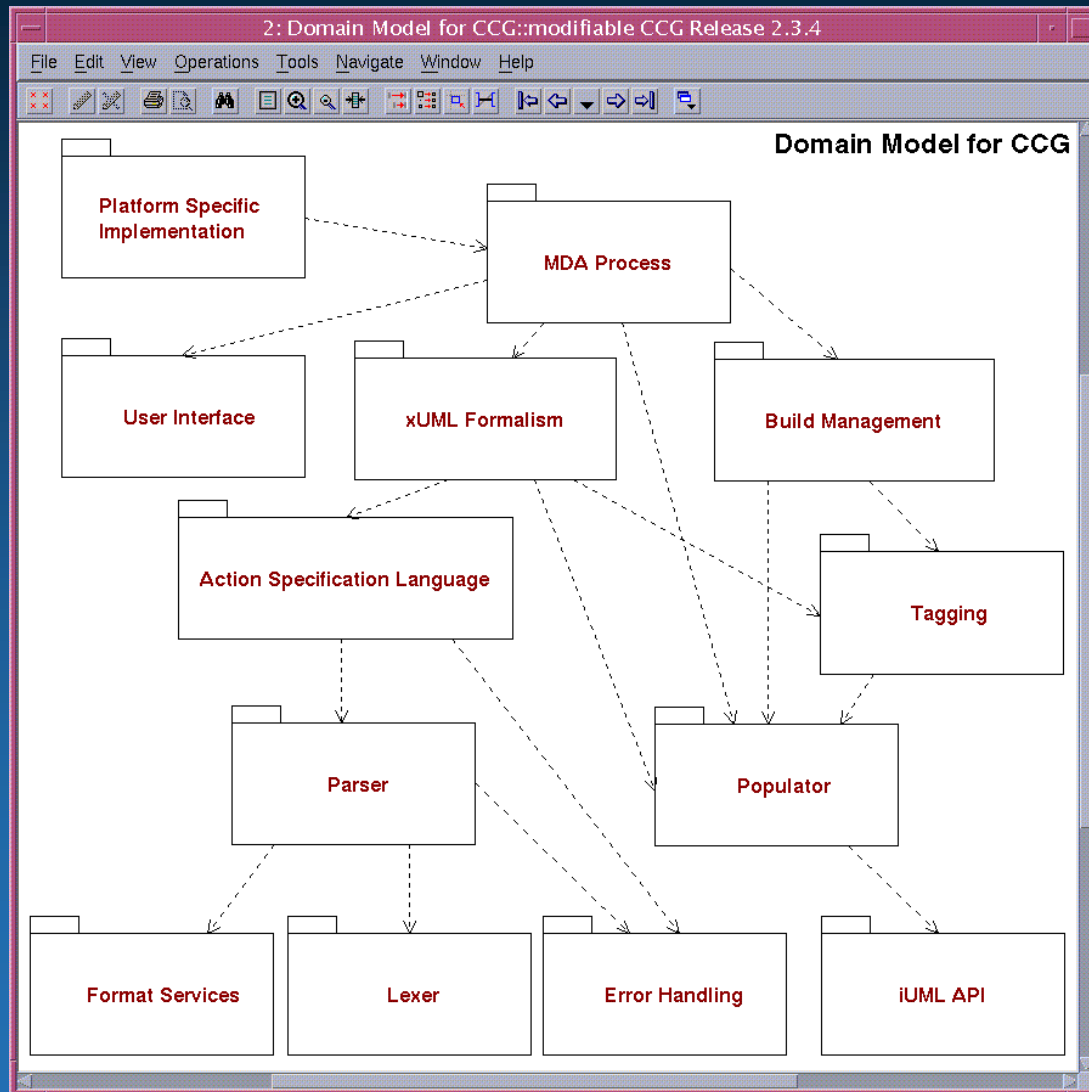
# Automatic Code Generation: Code Generator Development



## Configurable Code Generator:

- Code Generator is developed using the same eXecutable MDA strategy
- The Tool Vendor supplies a set of xUML models (known as the Configurable Code Generator) that serve as a generic translation framework

# Automatic Code Generation: Code Generator Development



## Code Generator Development:

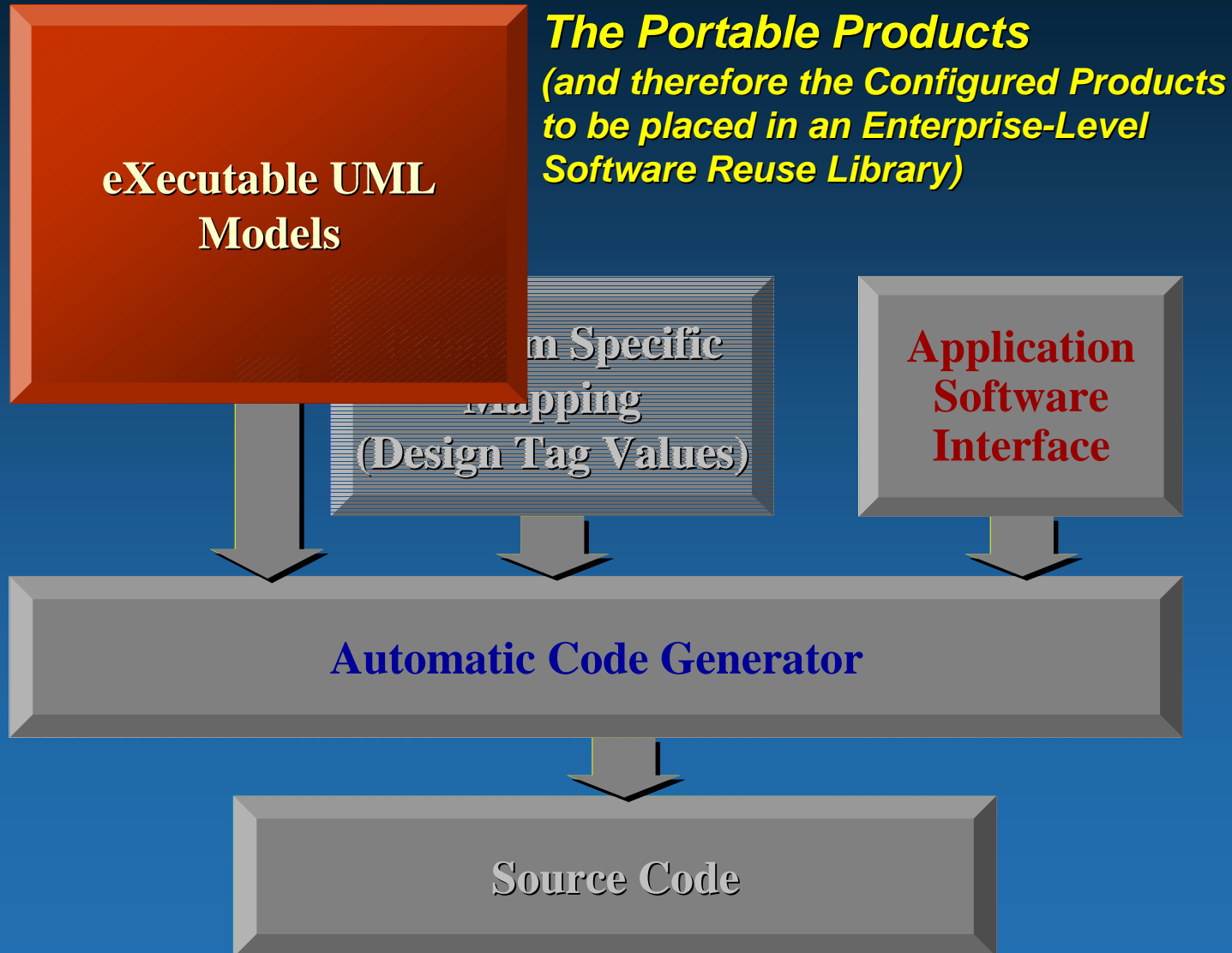
- The Configurable Code Generator may be adapted to meet the requirements of any Platform Specific Implementation (i.e. of any Application Software Interface)
- Code Generator and Application Software development may be performed concurrently



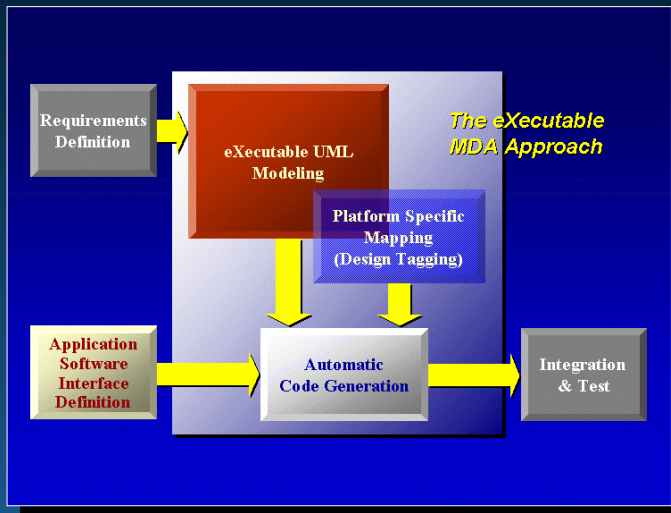
## Automatic Code Generation

- Automatic code generation is simply an extension of the code generation technique used for simulation of the eXecutable UML models on the development platform, this extension being for the target (embedded) platform
- The code generator is developed within the same environment as the application software using the same eXecutable MDA strategy
  - *Development cost: 1-2 architects*
- Nearly all implementation-specific design tasks (all but the design decisions represented by design tag values) are performed by the code generator, not the software developers

# Portable Application Software Products



# Advantages of the eXecutable MDA Approach

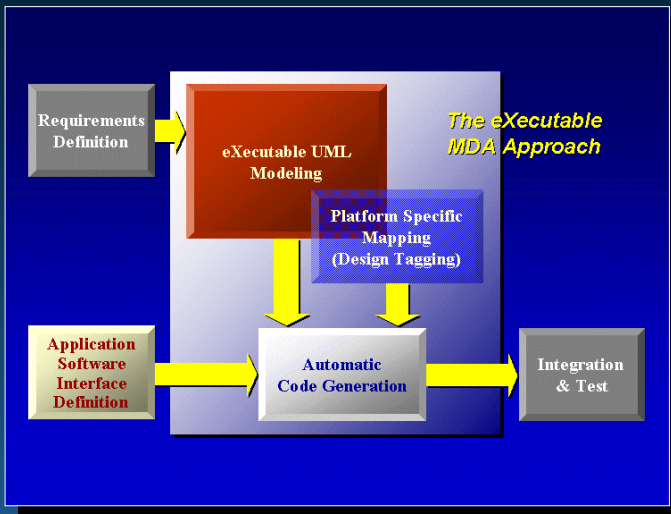


## Increased Quality

- The majority of software developers are isolated from implementation details, allowing them to focus on a thorough analysis of the application space
- Maintenance of the application source code is eliminated, while maintenance of the xUML models is ensured
- Defect injection (and the resulting rework) is reduced by automating the software phase in which most defects are injected
  - *On a typical program, after Requirements Definition approximately 2/3 of the defects are injected during implementation (coding)*



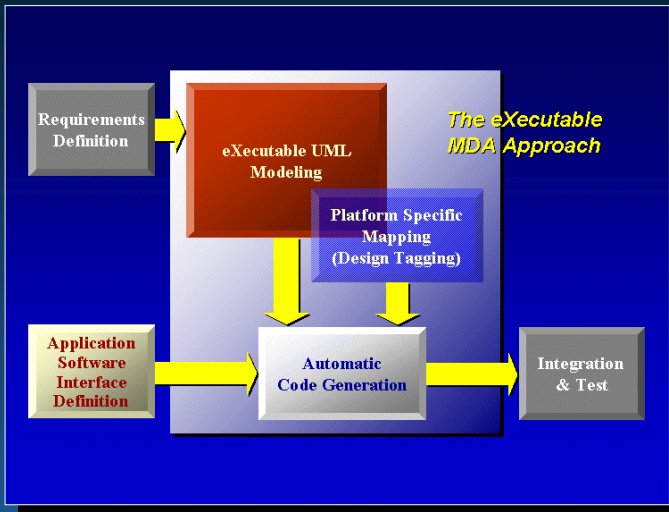
# Advantages of the eXecutable MDA Approach



## Increased Productivity

- **Rework is reduced**
  - *Early validation through simulation reduces rework*
    - Increase in eXecutable UML modeling span time is more than offset by decrease in Integration & Test span time
  - *Higher quality implementation (due to automation) reduces rework*
- **Software development span time is reduced by automating the implementation phase**
  - *Application Software development schedule is reduced by at least 20%*
  - *The code generator, not each software developer, performs the majority of implementation-specific design tasks*
    - 40-60% of physical source code

# Advantages of the eXecutable MDA Approach



## Cross-Platform Compatibility

- One Application Software xUML Model database may be reused (as is) on any platform for which a code generator is developed
  - *xUML models are compatible with any hardware platform, any Software Execution Platform, and any Application Software Interface*
  - *xUML models are compatible with any implementation language*

***The Goal of Cross-Platform Compatibility of Application Software is Attainable with the eXecutable MDA Approach***

# Contact Information



**Lauren E. Clark**  
**Chief Engineer**  
**F-16 Modular Mission Computer Software**  
**Lockheed Martin Aeronautics Company**

**[Lauren.E.Clark@lmco.com](mailto:Lauren.E.Clark@lmco.com)**  
**(817) 763-2748**

**Terry Ruthruff**  
**Staff Specialist**  
**Software Engineering Core**  
**Lockheed Martin Aeronautics Company**

**[Terry.Ruthruff@lmco.com](mailto:Terry.Ruthruff@lmco.com)**  
**(817) 763-3525**

**Bary D. Hogan**  
**Methodology Lead**  
**F-16 Modular Mission Computer Software**  
**Lockheed Martin Aeronautics Company**

**[Bary.D.Hogan@lmco.com](mailto:Bary.D.Hogan@lmco.com)**  
**(817) 763-2620**

**Allan Kennedy**  
**President**  
**Kennedy Carter Limited**

**[allan@kc.com](mailto:allan@kc.com)**  
**(+44) 1483 483 200**