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

Agenda

- **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**

Basic Software Components

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 / Built-In Test

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

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

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

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

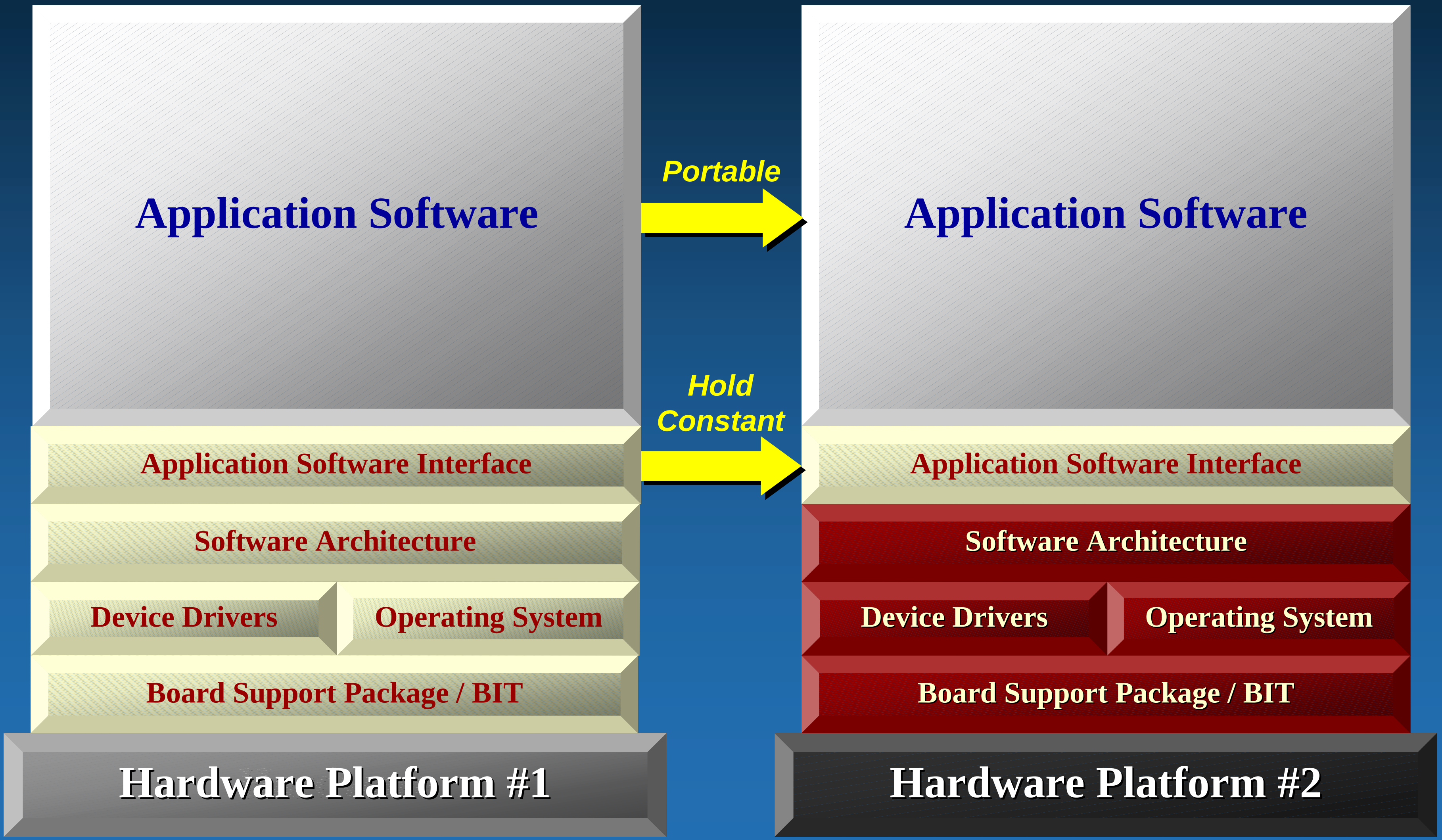
Application Software Interface

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…

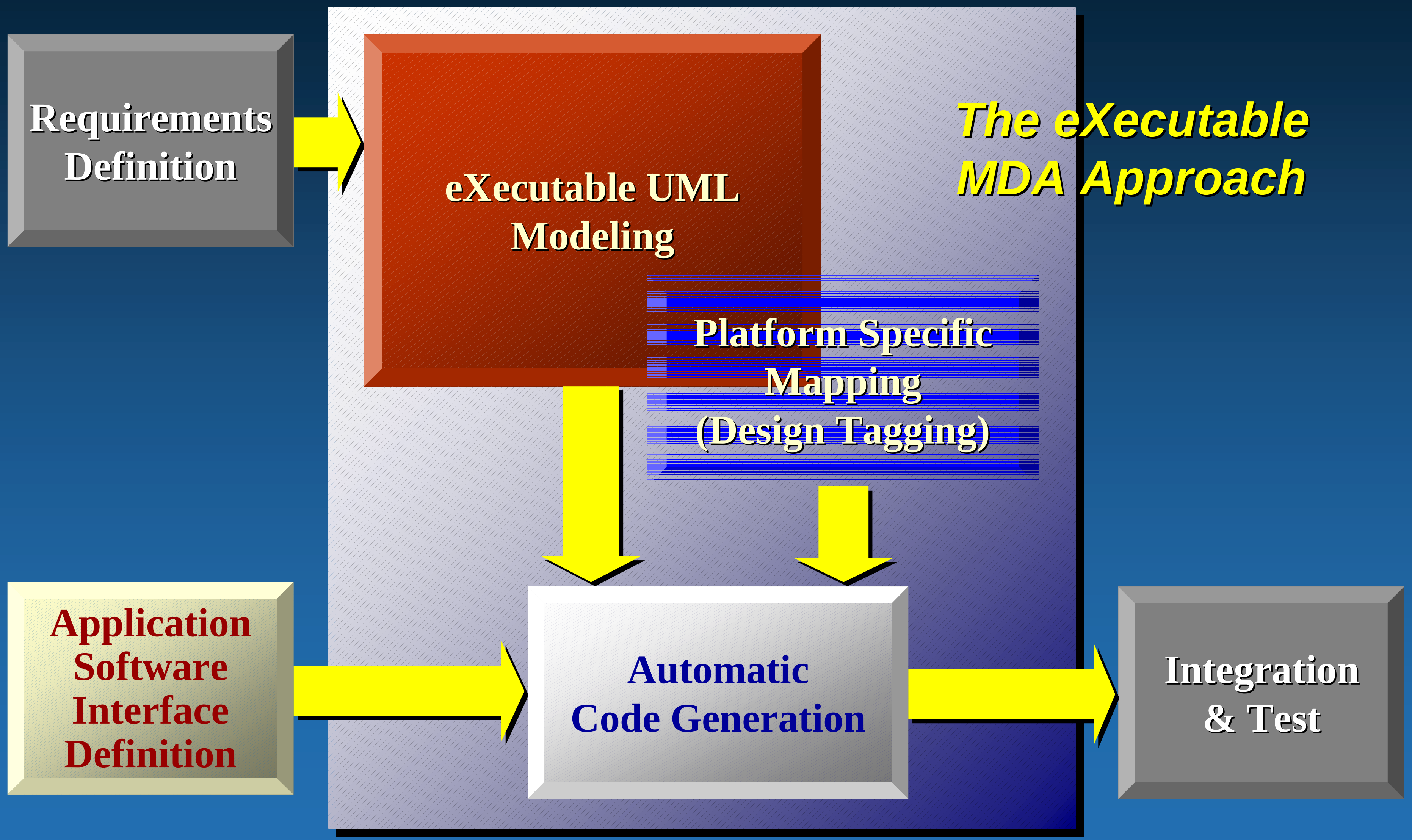
- What if hardware or Software Execution Platform changes could provide more Application Software capability, but only by means of changing the Application Software Interface?

Cross-Platform Compatibility: The Goal

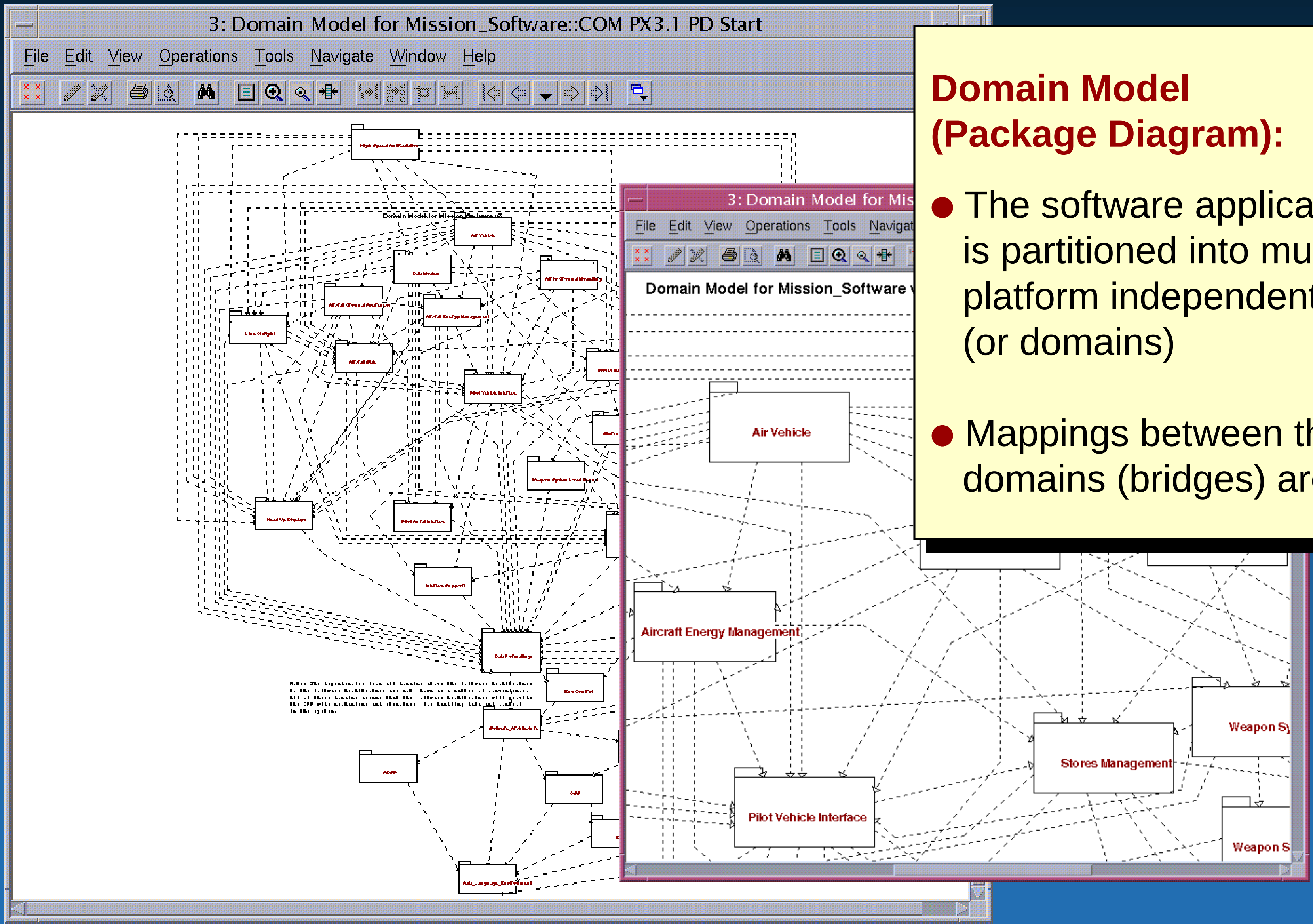
The goal
should be to provide
cross-platform compatibility
of Application Software
despite <u>any</u>
Implementation,
or platform specific, changes:

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

eXecutable MDA: Application Software Development



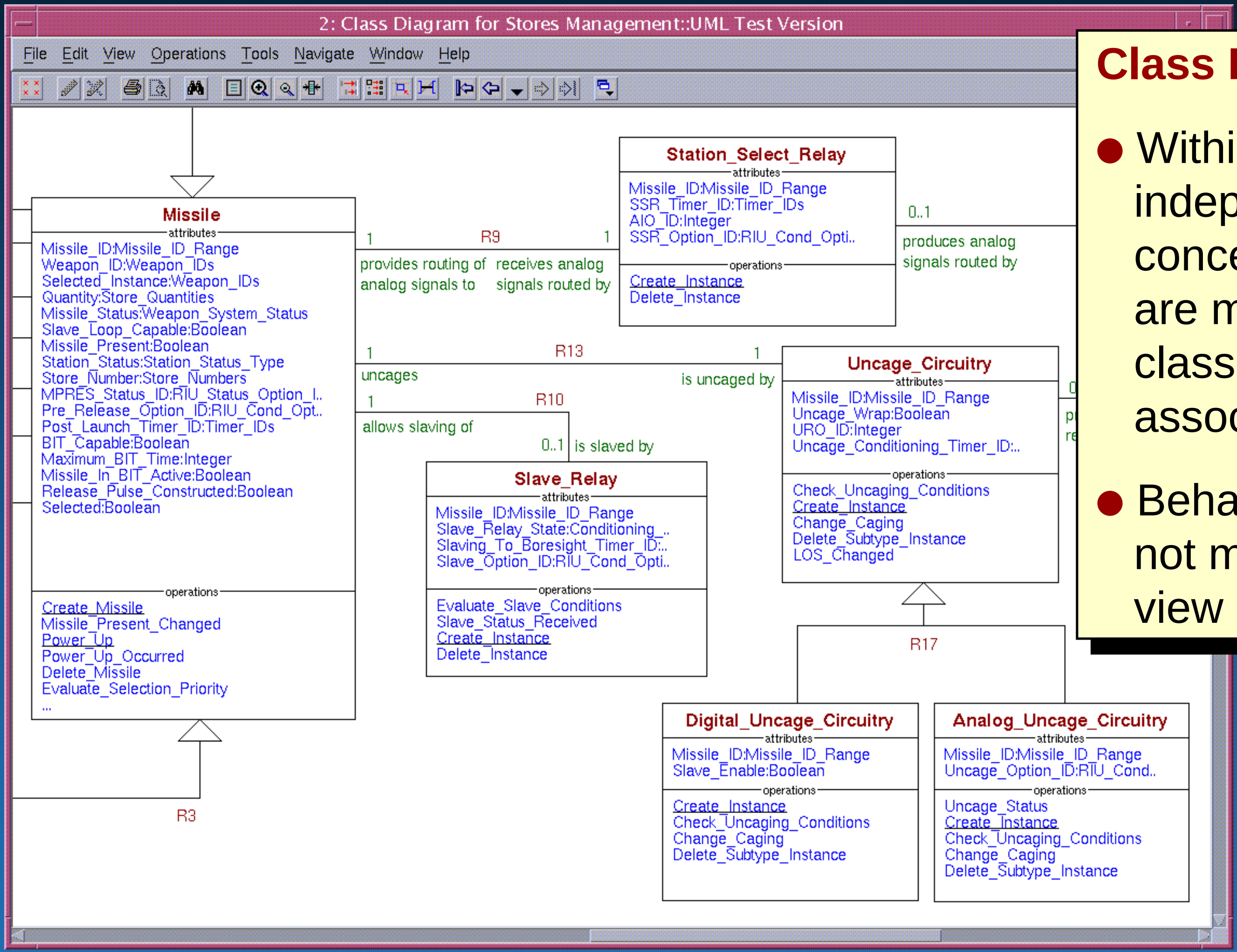
eXecutable UML Modeling: Domain Model



Domain Model (Package Diagram):

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

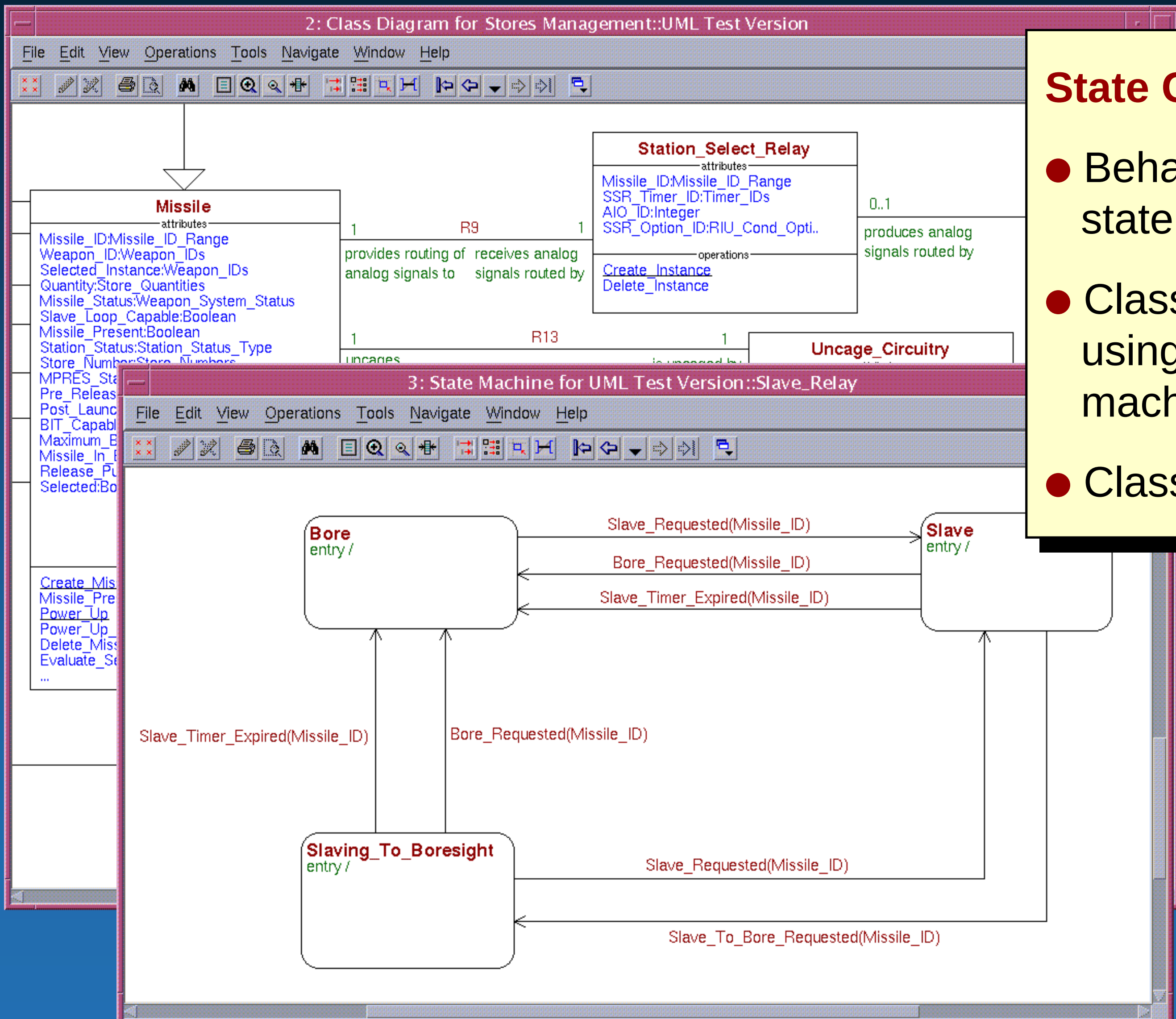
eXecutable UML Modeling: Class Diagrams



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

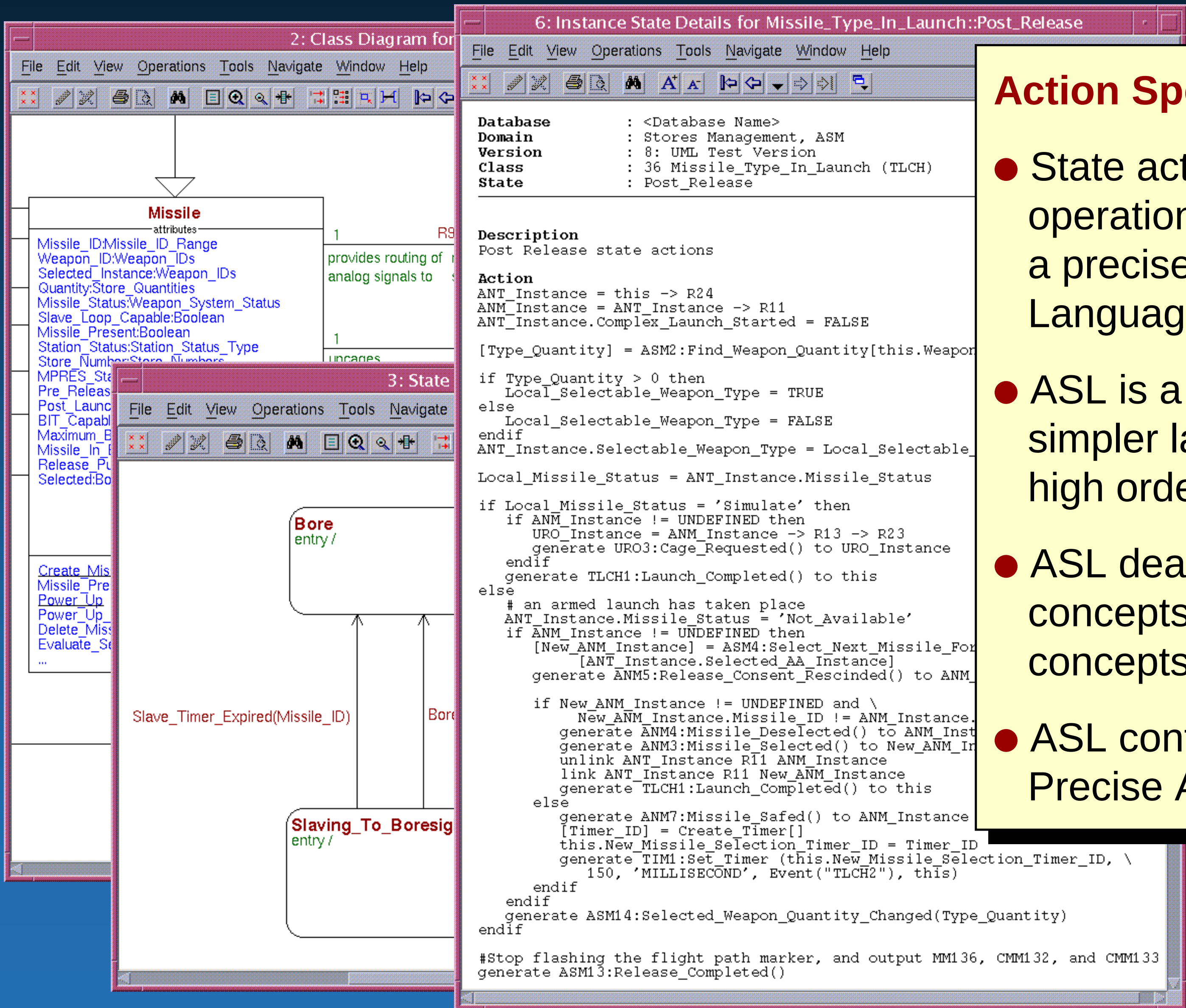
eXecutable UML Modeling: State Charts



State Charts:

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

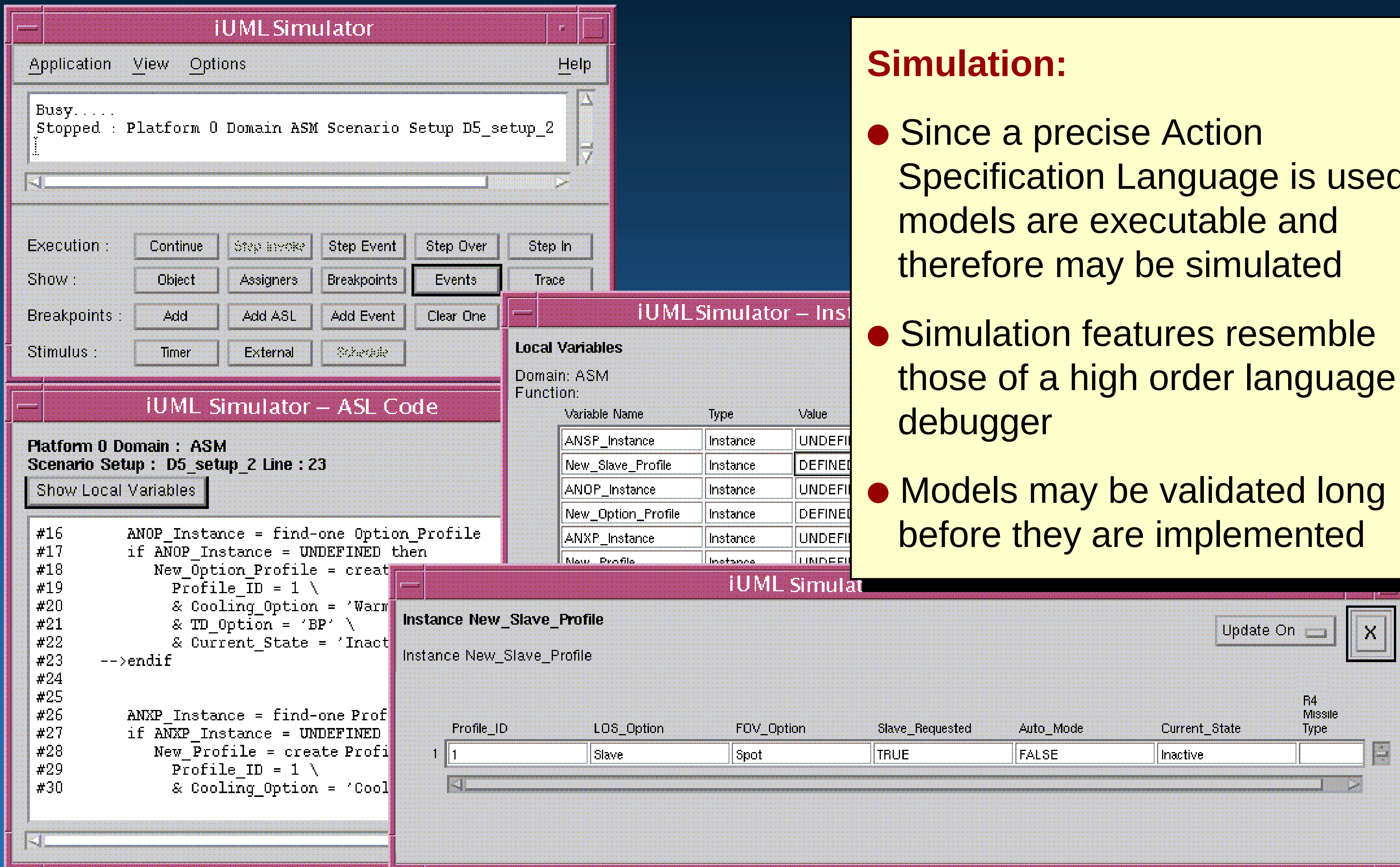
eXecutable UML Modeling: ASL



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

eXecutable UML Modeling: Simulation



Simulation:

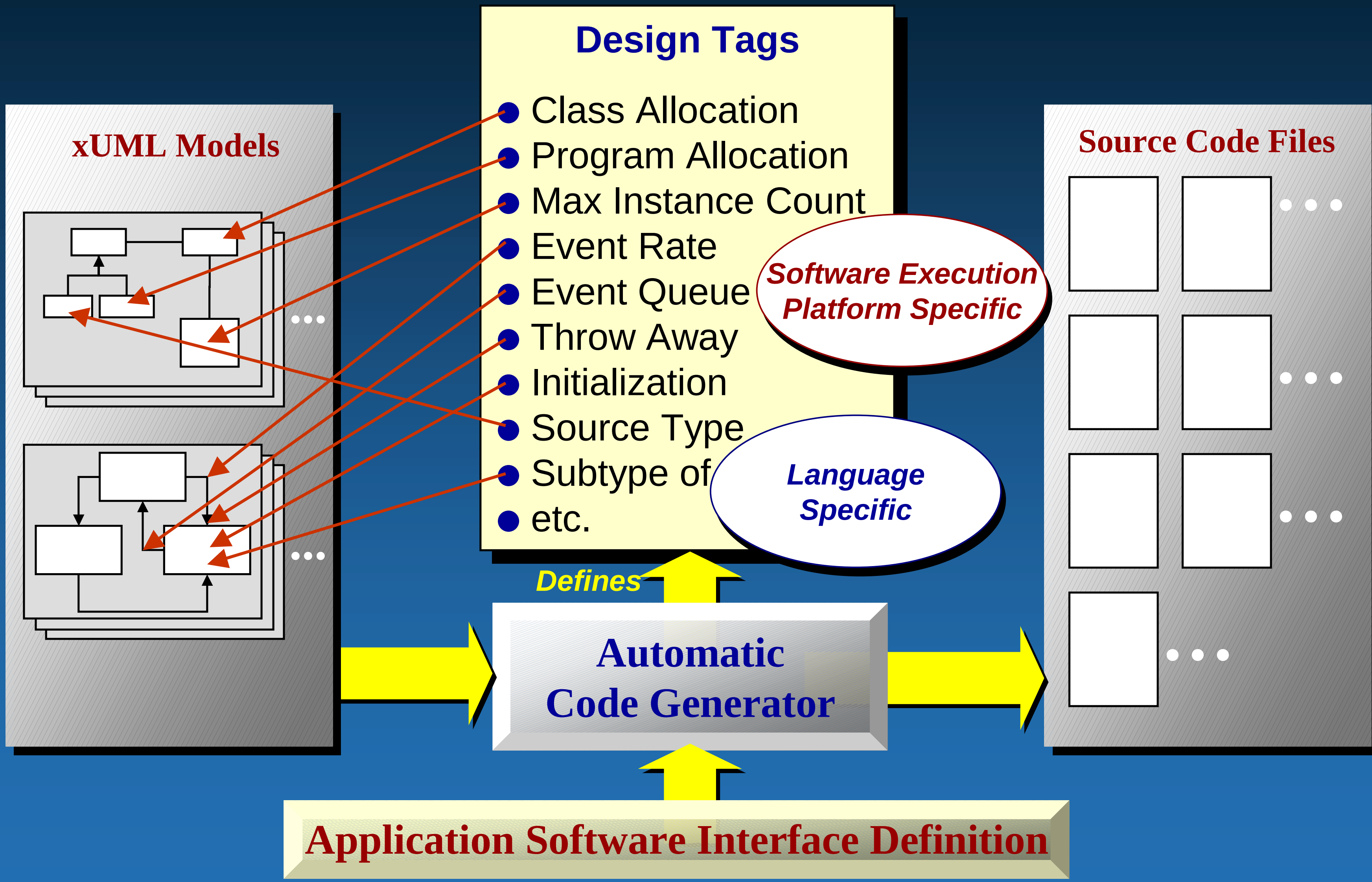
- Since a precise Action Specification Language is used, models are executable and therefore may be simulated
- Simulation features resemble those of a high order language debugger
- Models may be validated long before they are implemented

eXecutable UML Modeling: Summary

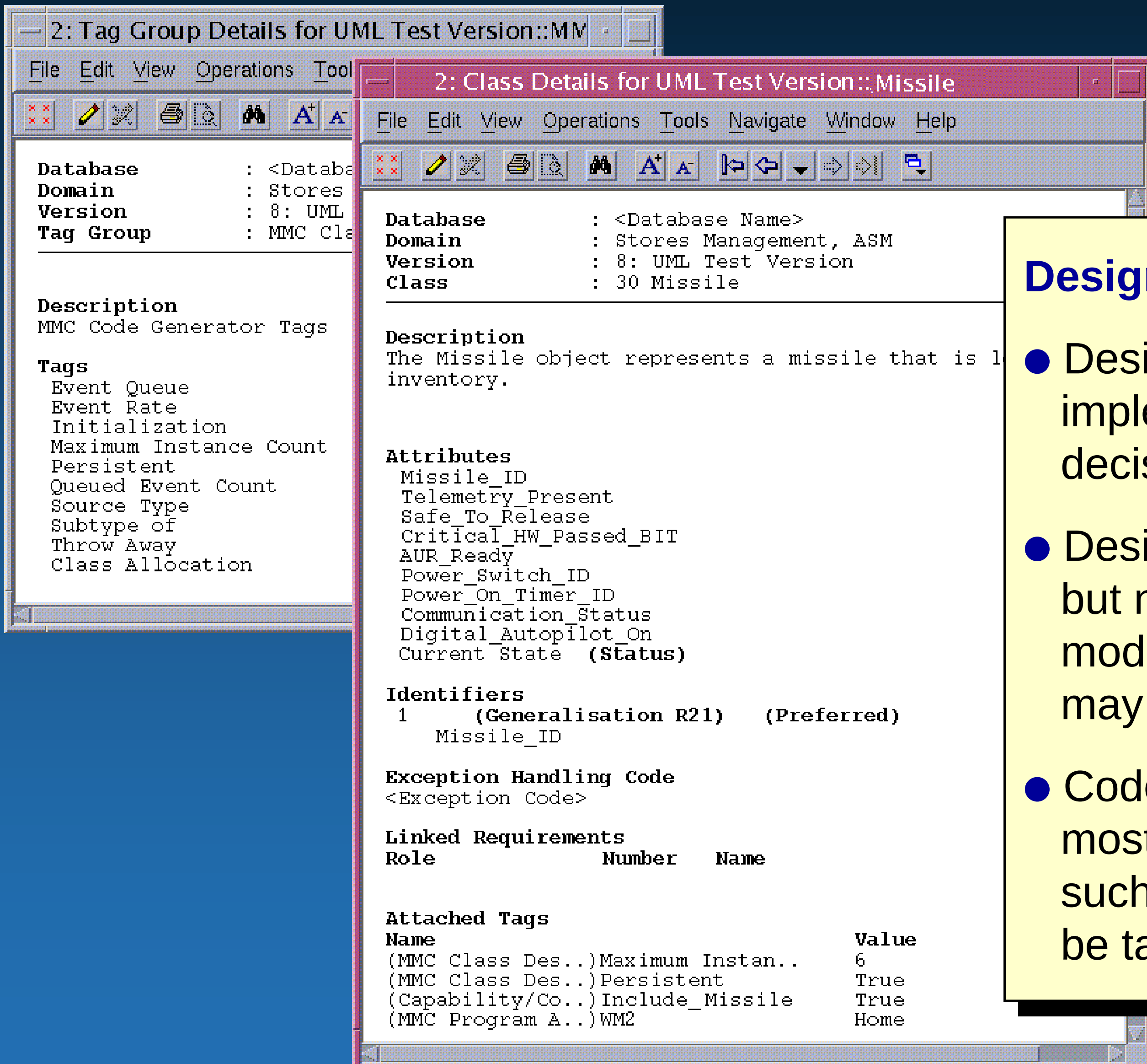
eXecutable UML Modeling

- 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



Design Tagging:

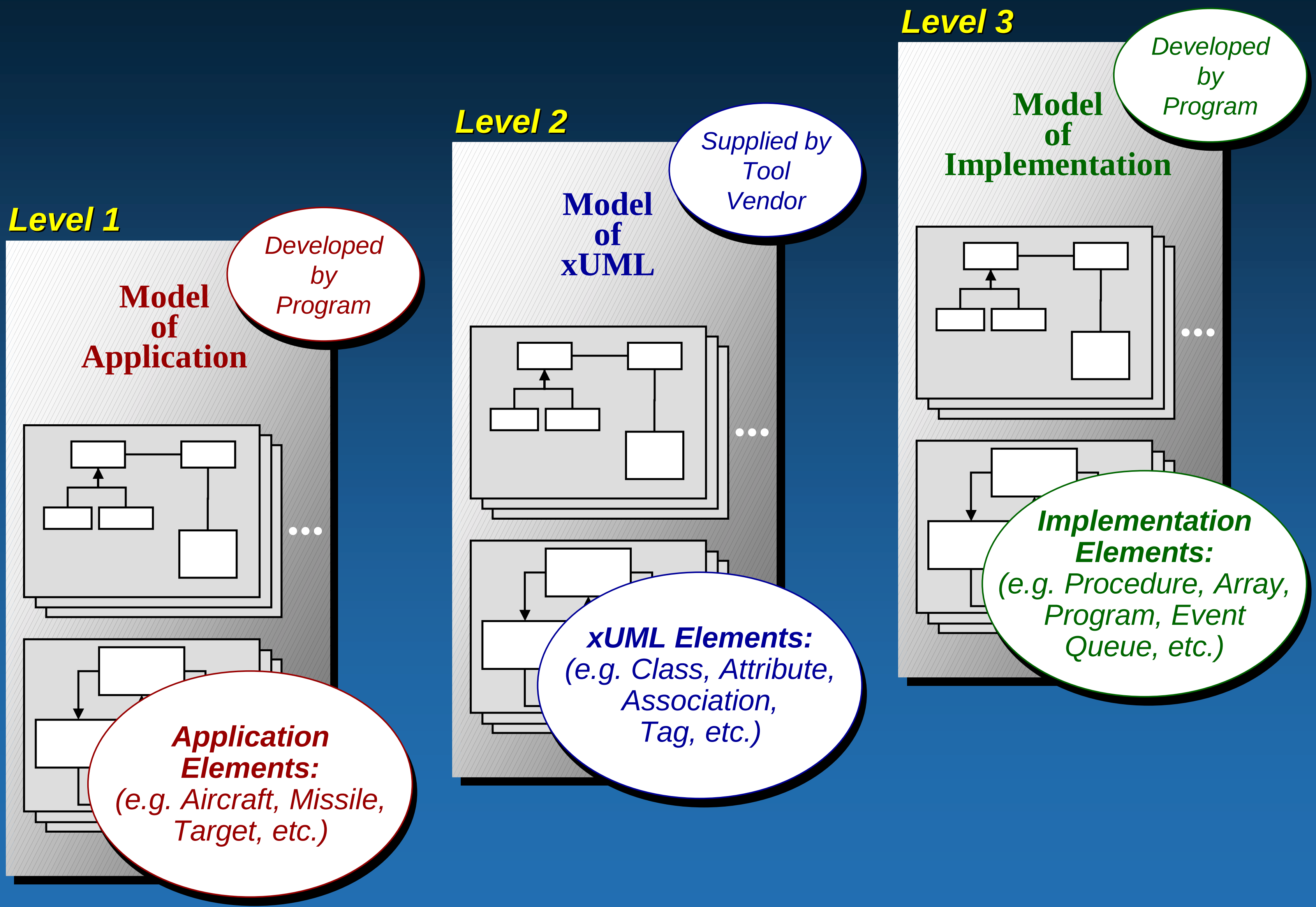
- Design tag values represent implementation-specific design decisions
- Design tagging is applied to, but not embedded in, the xUML models (tags and tag values may be included or excluded)
- Code Generator assumes the most standard implementation, such that only exceptions must be tagged

Design Tagging: Summary

Platform Specific Mapping (Design Tagging)

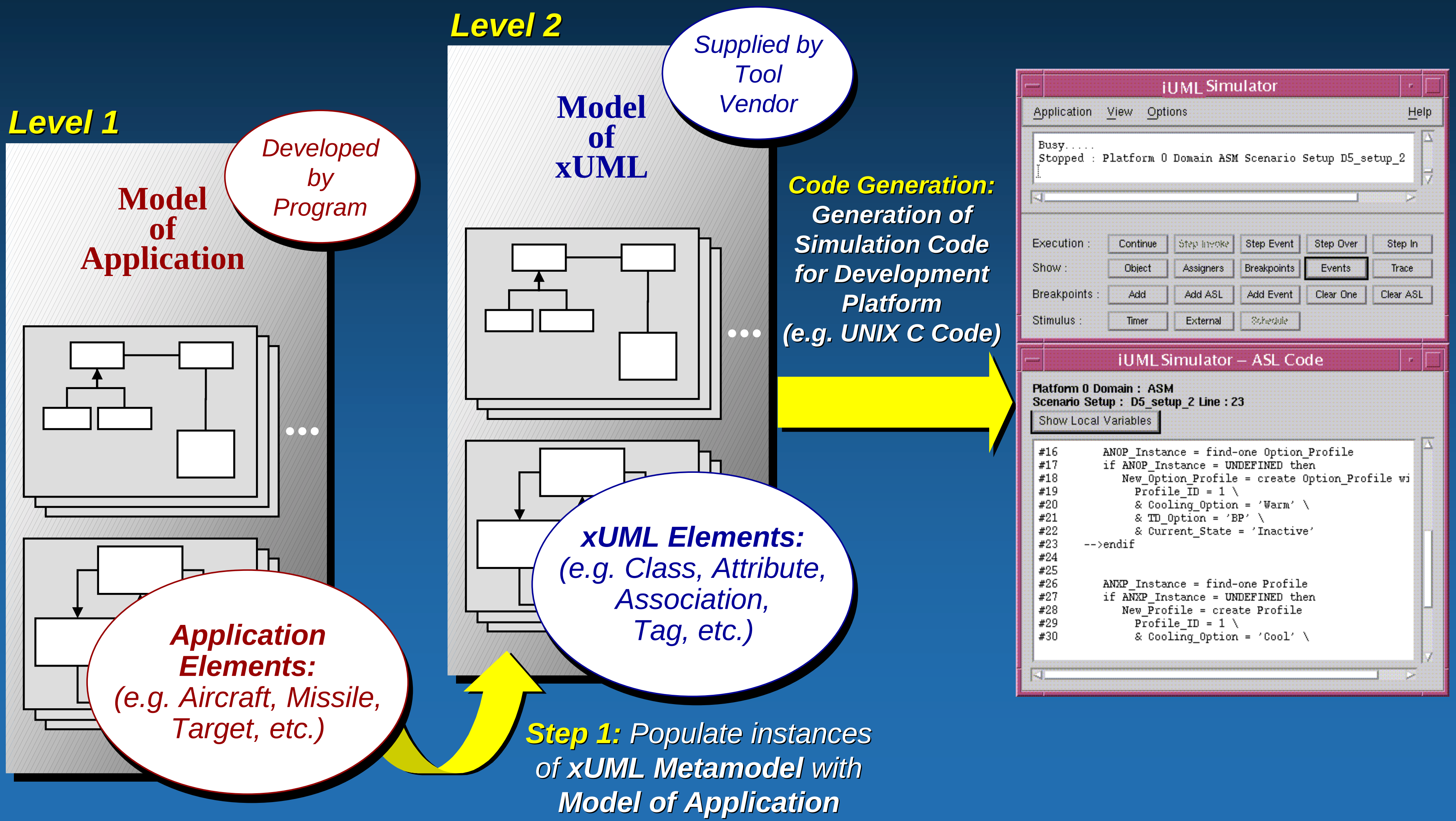
- Whereas xUML modeling is implementation-independent, Design Tagging is implementation-dependent (i.e. specific to a particular Application Software Interface)
- Implementation-specific design decisions (only those needed to support code generation) are made during Design Tagging, and are represented 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

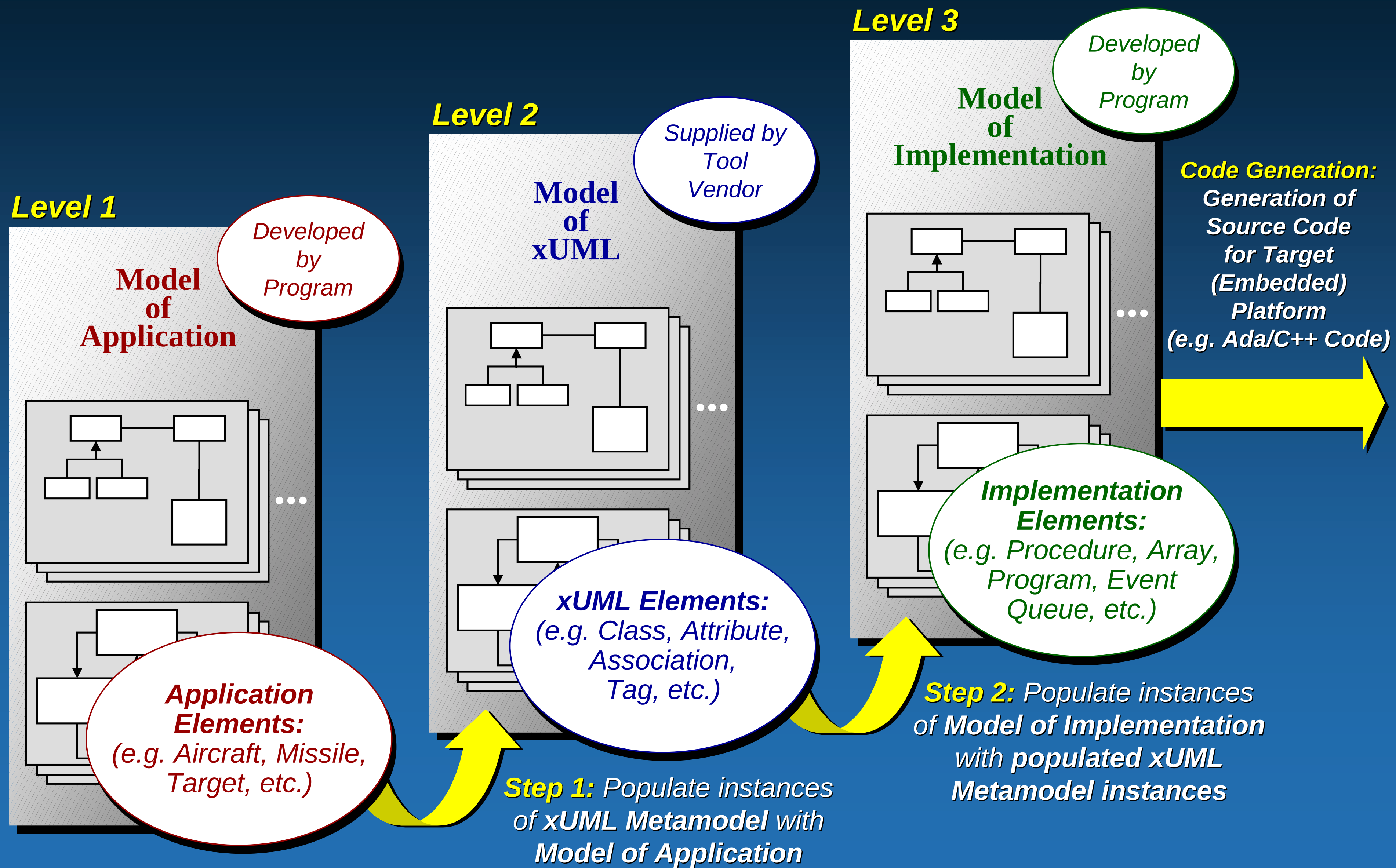


Automatic Code Generation: Level 2 - Simulation Code

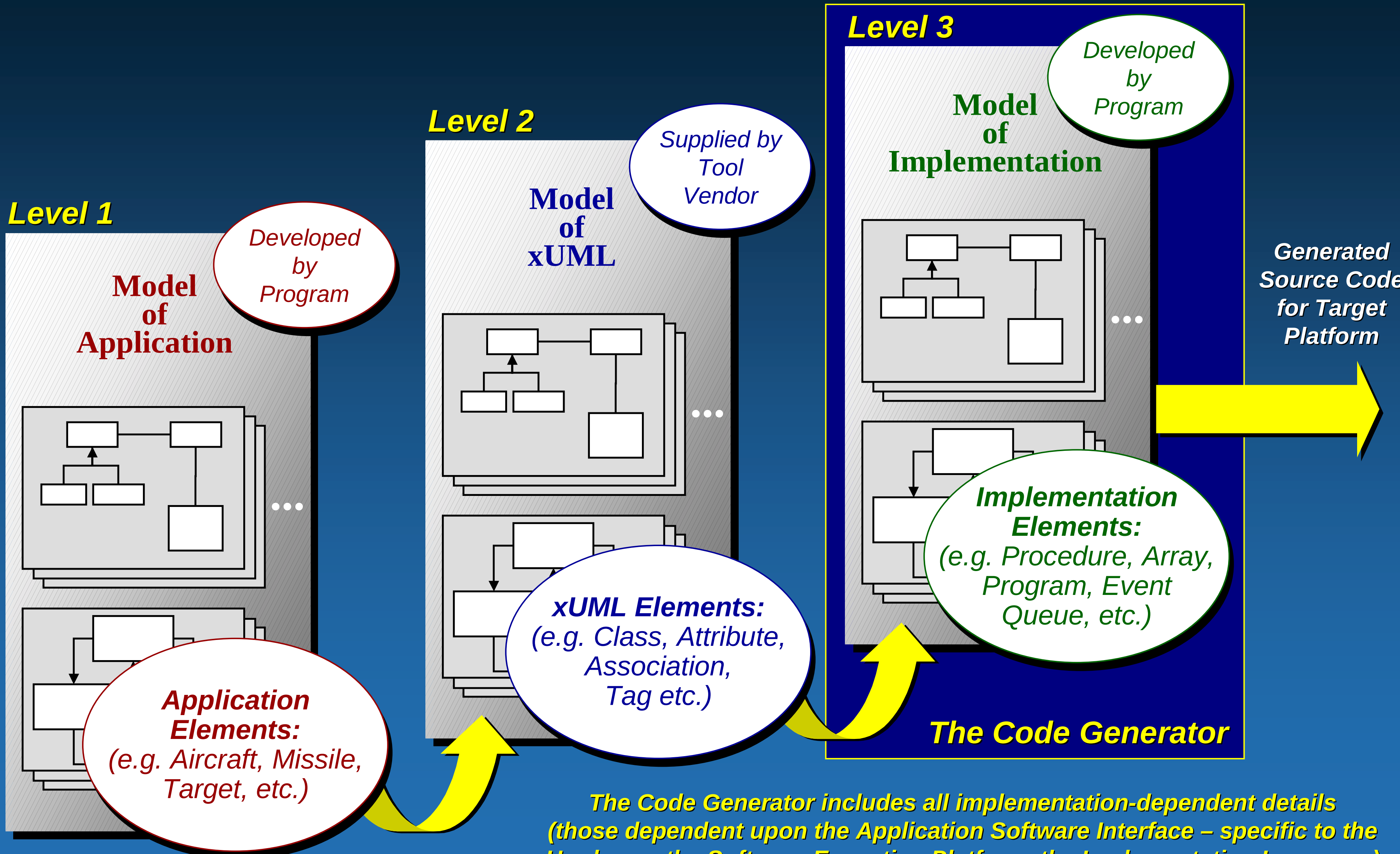
When we say that "xUML models are executable" we mean that "executable code can be automatically generated from them"



Automatic Code Generation: Level 3 - Target Code

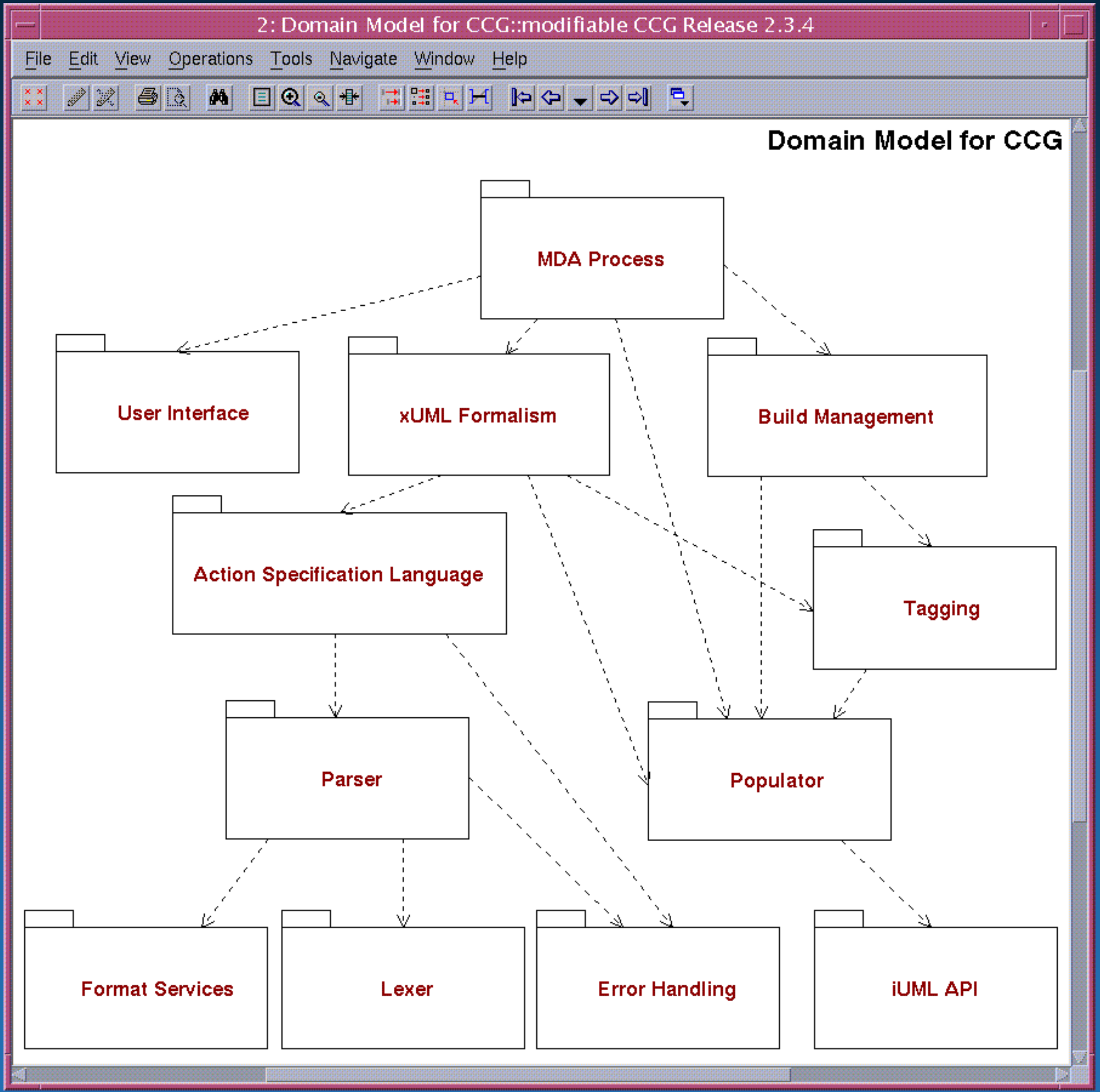


Automatic Code Generation: The Code Generator



The Code Generator includes all implementation-dependent details (those dependent upon the Application Software Interface – specific to the Hardware, the Software Execution Platform, the Implementation Language)

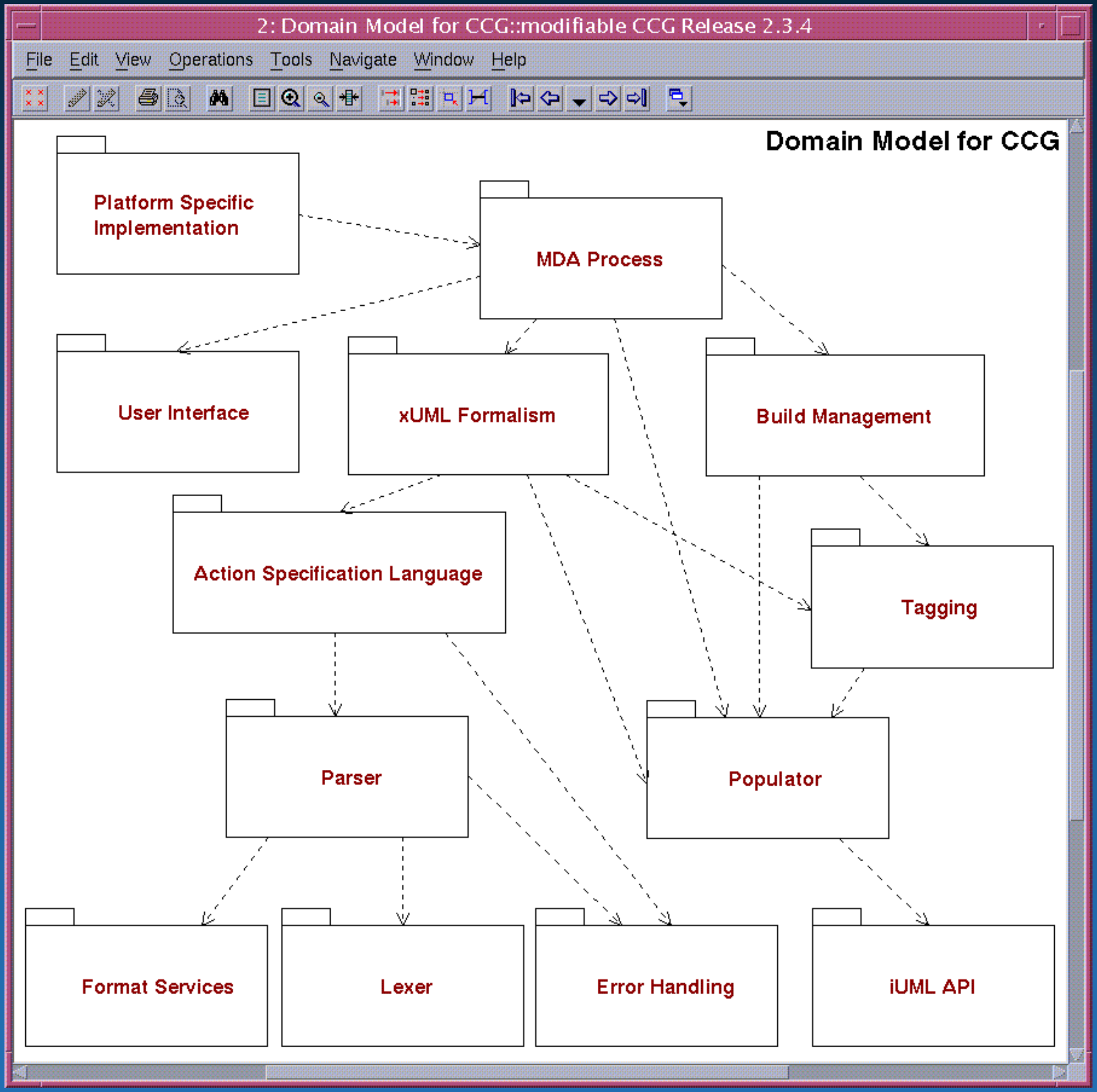
Automatic Code Generation: Code Generator Development



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 the 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: Summary

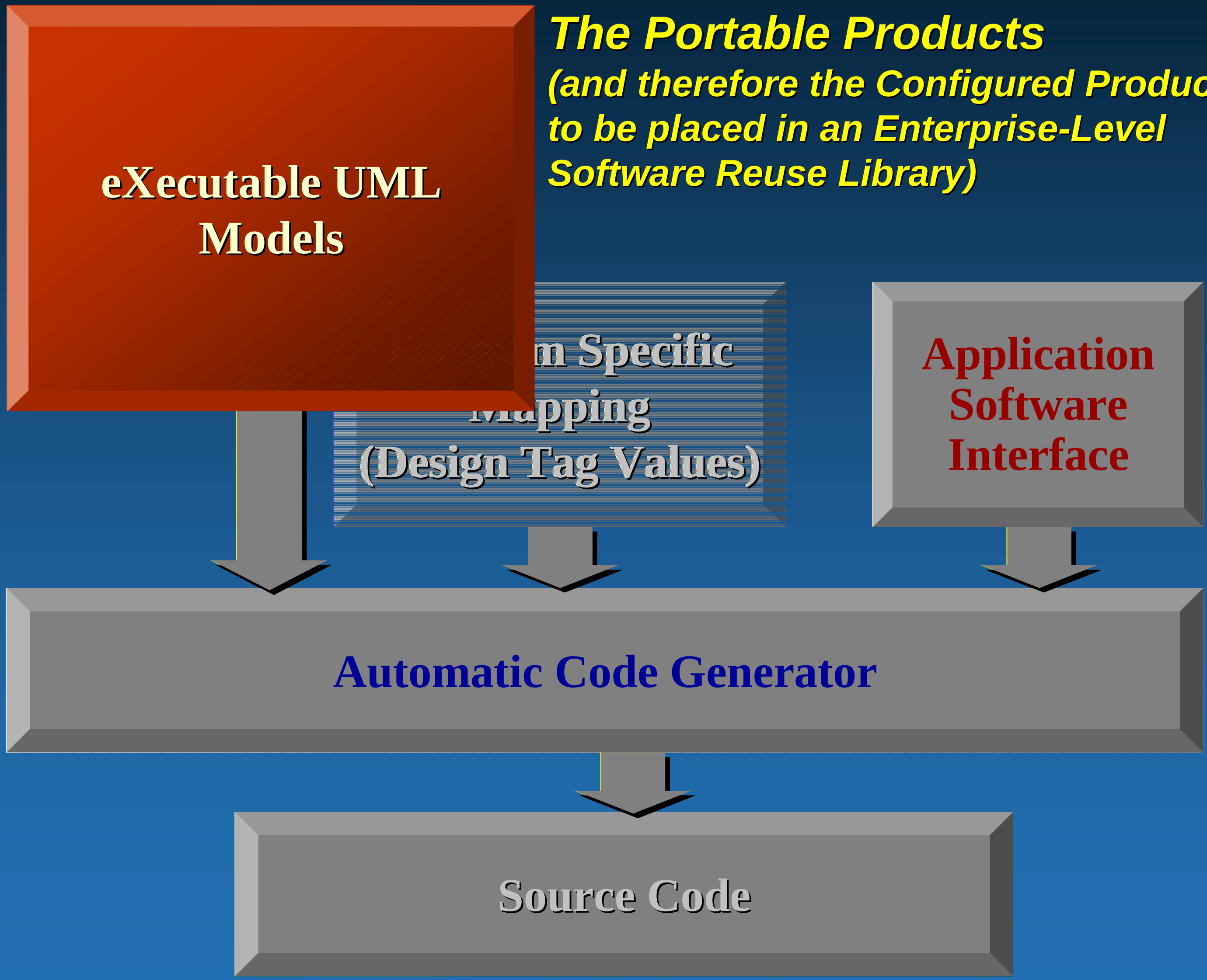
Automatic Code Generation

- 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

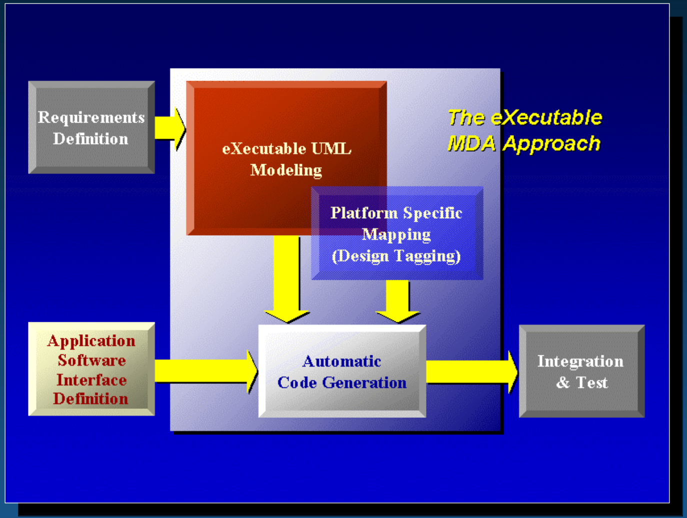
Portable Application Software Products

The Portable Products

(and therefore the Configured Products to be placed in an Enterprise-Level Software Reuse Library)



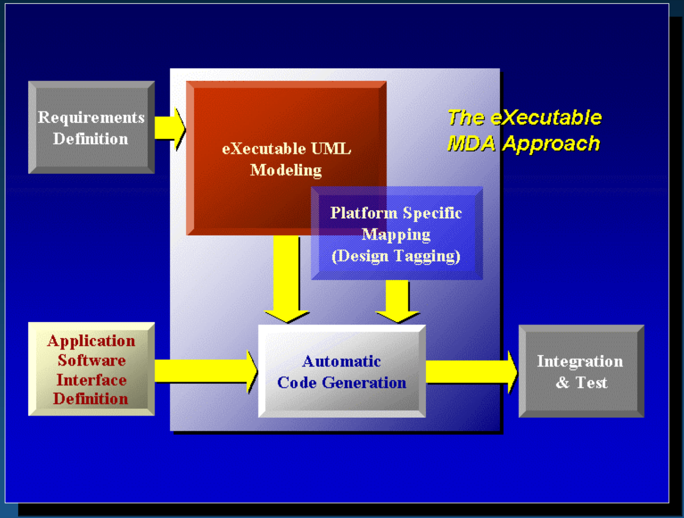
Advantages of the eXecutable MDA Approach



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)***

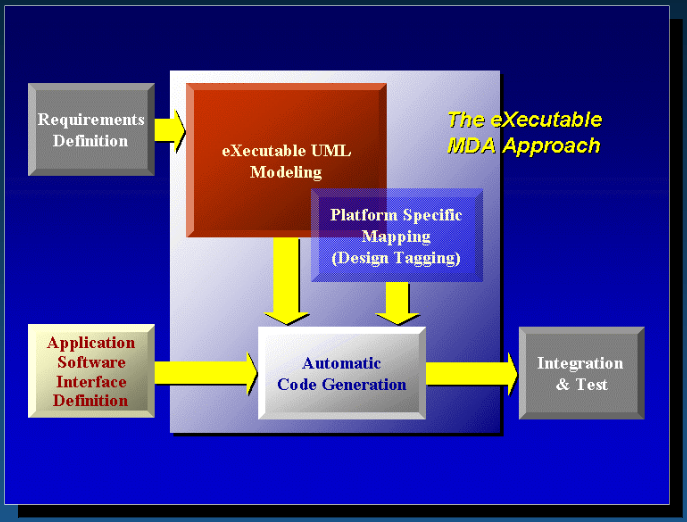
Advantages of the eXecutable MDA Approach



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
(817) 763-2748

Terry Ruthruff
Staff Specialist
Software Engineering Core
Lockheed Martin Aeronautics Company
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
(817) 763-2620

Allan Kennedy
President
Kennedy Carter Limited
allan@kc.com
(+44) 1483 483 200