Computer Systems, Inc.
MERCURY

Enabling Component-based Applications in Embedded Systems

Frank Pilhofer
fp@mc.com

The Ultimate Performance Machine

Contents

- Component-based Development
- Lightweight CCM
- Deployment and Configuration of Component-based Applications
  - Separation of Concerns
  - Four Phases of Deployment
  - Deployment Actors
  - Component Model
- Case Study: SCA Evolution

Computer Systems, Inc.
MERCURY

Component-based Development

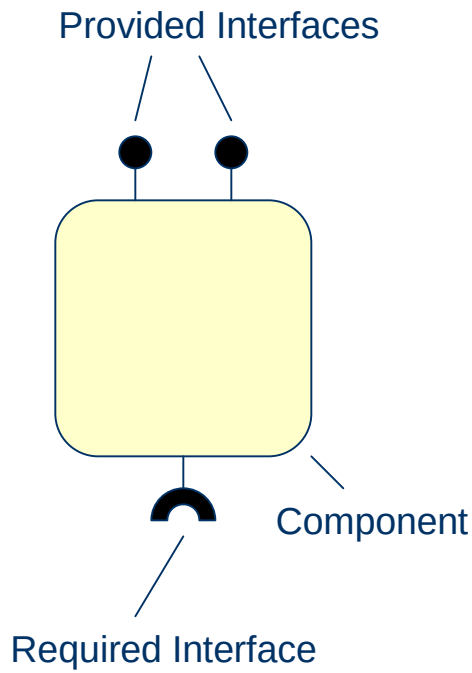
The Ultimate Performance Machine

Components

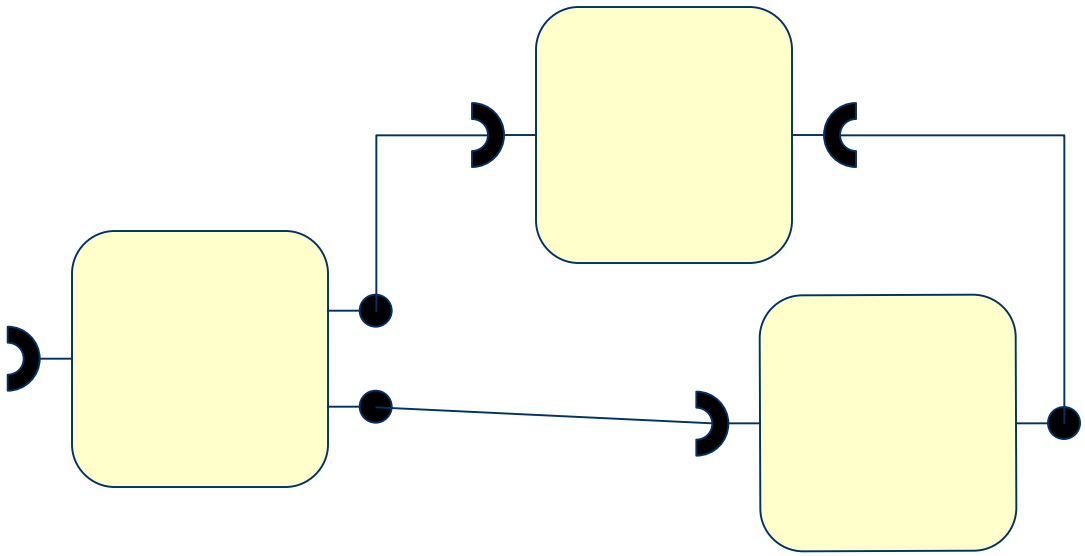
- "A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment" – [UML]
  - "Modular:" building block
  - "Encapsulates contents:" black box
  - "Replaceable:" implementation vs. interface

Components (2)

- A component defines its behavior in terms of provided and required interfaces.
- Conformance is defined by these provided and required interfaces.
- Provided and required interfaces: “Ports”

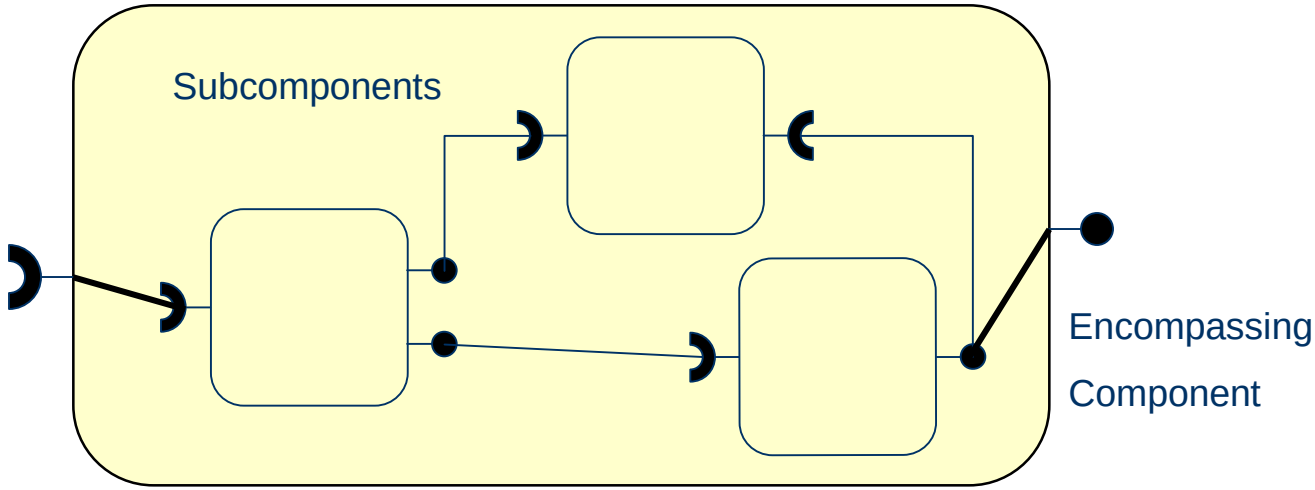


Components (3)



- Larger pieces of a system's functionality may be assembled by reusing components as parts in an assembly, and wiring together their required and provided interfaces.

Hierarchical Assemblies



- **Assemblies are components!**
  - **Assemblies "implement" a specific component interface**
  - **Mapping of component ports and properties to subcomponents**
  - **Assembly can then be reused as a component**

Computer Systems, Inc.
MERCURY

Lightweight CCM

The Ultimate Performance Machine

Lightweight CCM

- **Original CCM**
  - Oriented towards multi-tier architectures
  - Based on Enterprise Java Beans
    - plus ports!
- **Lightweight CCM**
  - Upwards compatible subset of CCM
  - Retains component model
    - Ports, Assemblys
  - Removes “business components” features
    - Persistence, Transactions, Introspection
  - Reduced footprint, more suitable for embedded systems

Lightweight CCM (2)

- Uses “Deployment and Configuration” specification
  - Replaces CCM’s “Packaging and Deployment” chapter
- Supports …
  - Hierarchical assemblies
  - Resource management
  - QoS characteristics
  - Automated deployment
  - Vendor-independent deployment infrastructure

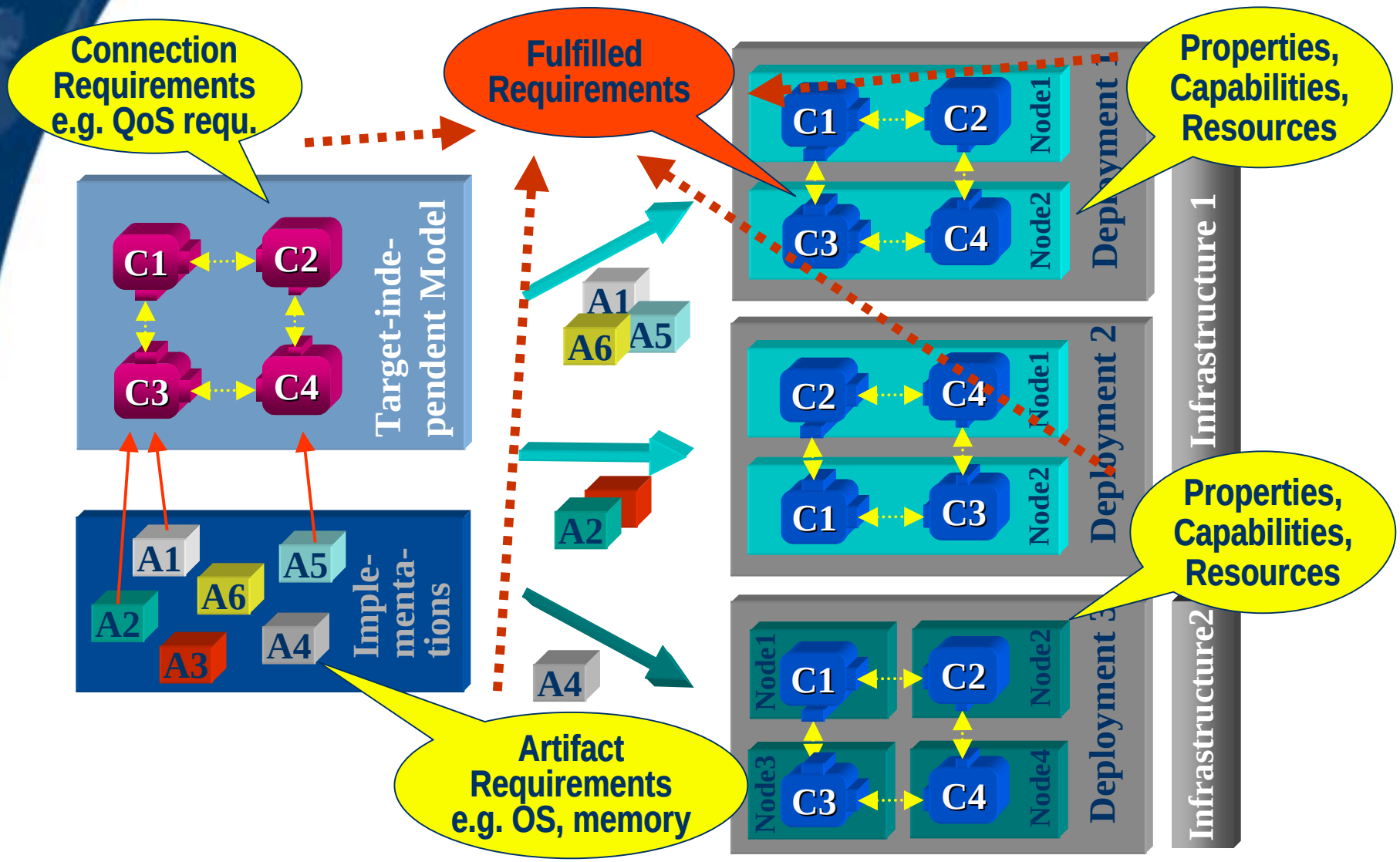
Computer Systems, Inc.
MERCURY

Deployment and Configuration of Component-based Applications

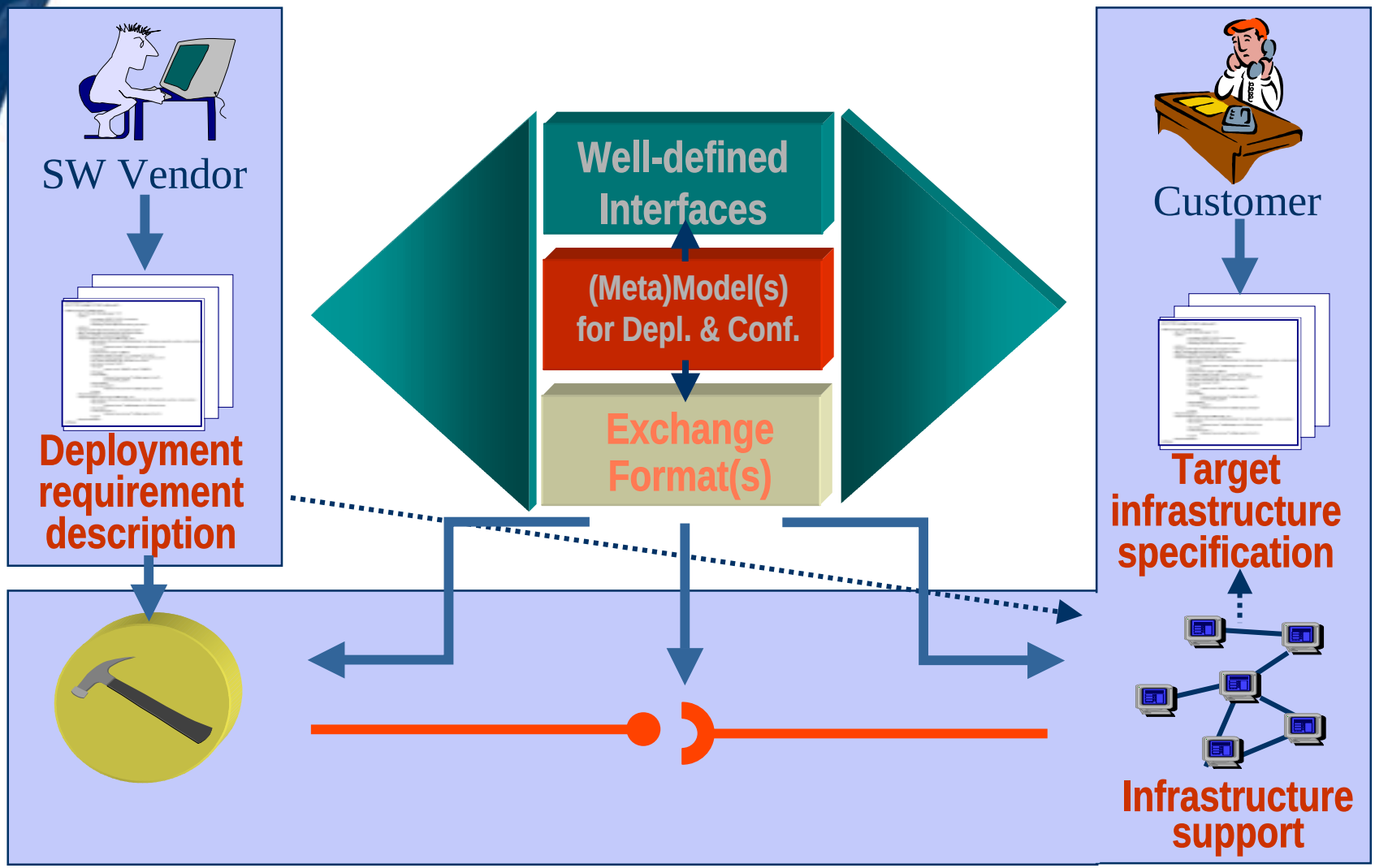
"D+C"

The Ultimate Performance Machine

Deployment Ideas



Deployment Ideas (2)



Deployment Ideas (3)

- Component Packages
  - ZIP files with artifacts and XML metadata
  - May contain alternative implementations
    - For different hardware, and/or
    - With different QoS characteristics
  - Reusable by other packages
- Hierarchical Assemblies
- Resource model
  - Implementations express requirements
    - CPU, OS, Devices, communication bandwidth
  - Target Domain expresses resources
  - Well-defined matching process

Computer Systems, Inc.
MERCURY

D+C
Deployment Model

The Ultimate Performance Machine

Separation of Concerns

- **Three Model Slices**
  - **Component Model**
    - Metadata to describe component-based applications
    - “Repository Manager” interface for installing, maintaining and retrieving Component Packages
  - **Target Model**
    - Metadata to describe available resources
    - “Target Manager” interface for accessing and tracking resources
  - **Execution Model**
    - Metadata to describe “Deployment Plan”
    - “Execution Manager” interface to execute applications according to plan

Compliance Points

- Four independent compliance points
  - Repository Manager, Target Manager, Execution Manager
    - Distinct functionality
    - Expected to be part of a COTS deployment suite
  - Node Manager
    - Responsible for intra-node deployment, as directed by the Execution Manager
    - Hardware, OS, ORB specific
    - Implemented by node vendor with little effort
- Each “Manager” can be replaced separately

Phases of Deployment

- **Four phases of Deployment**
  - **Installation**
    - Package is installed in the Repository
  - **Planning**
    - Component requirements are matched against available resources, resulting in a Deployment Plan
  - **Preparation, Launch**
    - Resources are allocated
    - Artifacts are loaded onto nodes
    - Components are instantiated, interconnected, and started
    - Separation between preparation and launch left open, to allow for a wide range of options (preload, early instantiation vs. late loading)

Planning

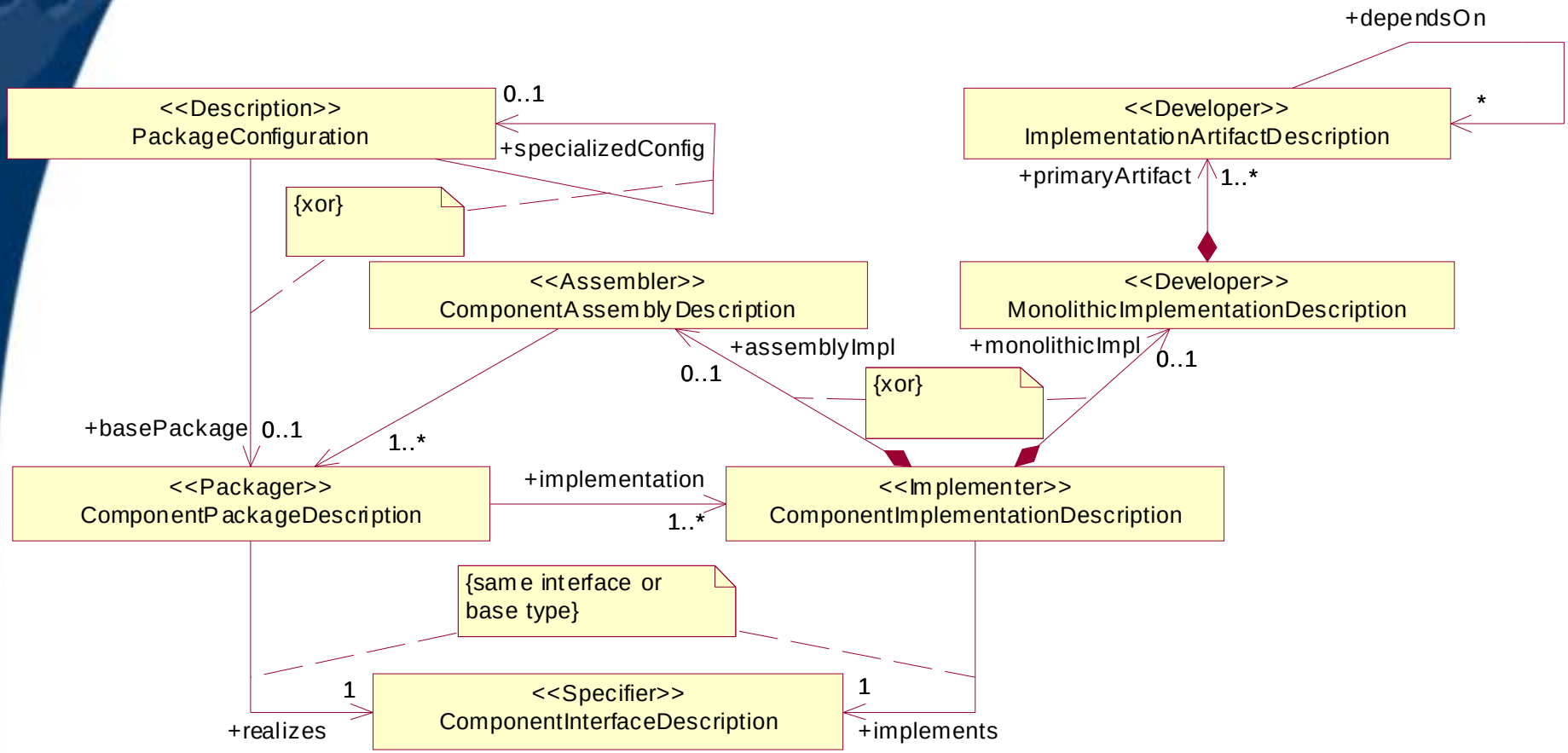
- **Planning involves …**
  - Requirement vs. Resource matching
  - Selecting acceptable implementations
  - Mapping implementations to Nodes
  - Results in Deployment Plan: A “what goes where” script, with configuration
- **Can be done …**
  - Online
    - Based on “live” resource data
    - For immediate preparation and launch
  - Offline
    - Based on a known set of available resources
    - For storage and later use / reuse

Computer Systems, Inc.
MERCURY

D+C Component Model

The Ultimate Performance Machine

Component Data Model



Component Package

- **Package Configuration**
  - Configures an application's properties, independent of implementation choice
  - Selects among alternative implementations
    - E.g., "Latency less than 50ms"
- **Component Package Description**
  - Describes one or more alternative implementations
    - E.g., for Windows, Linux, Java, FPGA …
- **Component Implementation**
  - Assembly-based or Monolithic
  - Describes QoS characteristics

Component Implementation

- **Monolithic implementation**
  - Usually compiled code
  - Describes hardware requirements
- **Assembly-based implementation**
  - Set of interconnected subcomponents
    - Instantiates subcomponents from Packages
    - "Sub-" packages may be included or referenced
  - Describes interconnection requirements (bandwidth)
  - Expresses QoS requirements on subcomponents, to satisfy its own
  - Hardware-independent

Computer Systems, Inc.
MERCURY

D+C and Embedded Systems

The Ultimate Performance Machine

D+C and Embedded Systems

- Limited XML Parsing
  - Only the Repository Manager needs an XML parser, to read “off-line” packages
  - Other information is passed “on-line”, using IDL-defined data structures
- Central Services
  - Repository Manager, Target Manager, Execution Manager are singleton services
  - Node Managers can be small, no “intelligence” necessary
- Planning done locally
  - No interaction with nodes

Round-trip Minimization

- **Round-trips incur latency**
  - **Avoid sequential, synchronous invocations**
- **Minimize round-trips**
  - **Only three round-trips between Execution Manager and Node Managers during application launch:**
    - **First round-trip: node managers return “provided” references for each component**
    - **Second round-trip: execution manager sends references to “uses” ports for each component**
    - **Third round-trip: “start” signal**
  - **These steps can be parallelized**
    - **EM sends invocations in parallel, then waits for all replies (e.g., using AMI)**

Computer Systems, Inc.
MERCURY

Case Study: SCA Evolution

The Ultimate Performance Machine

SCA Introduction

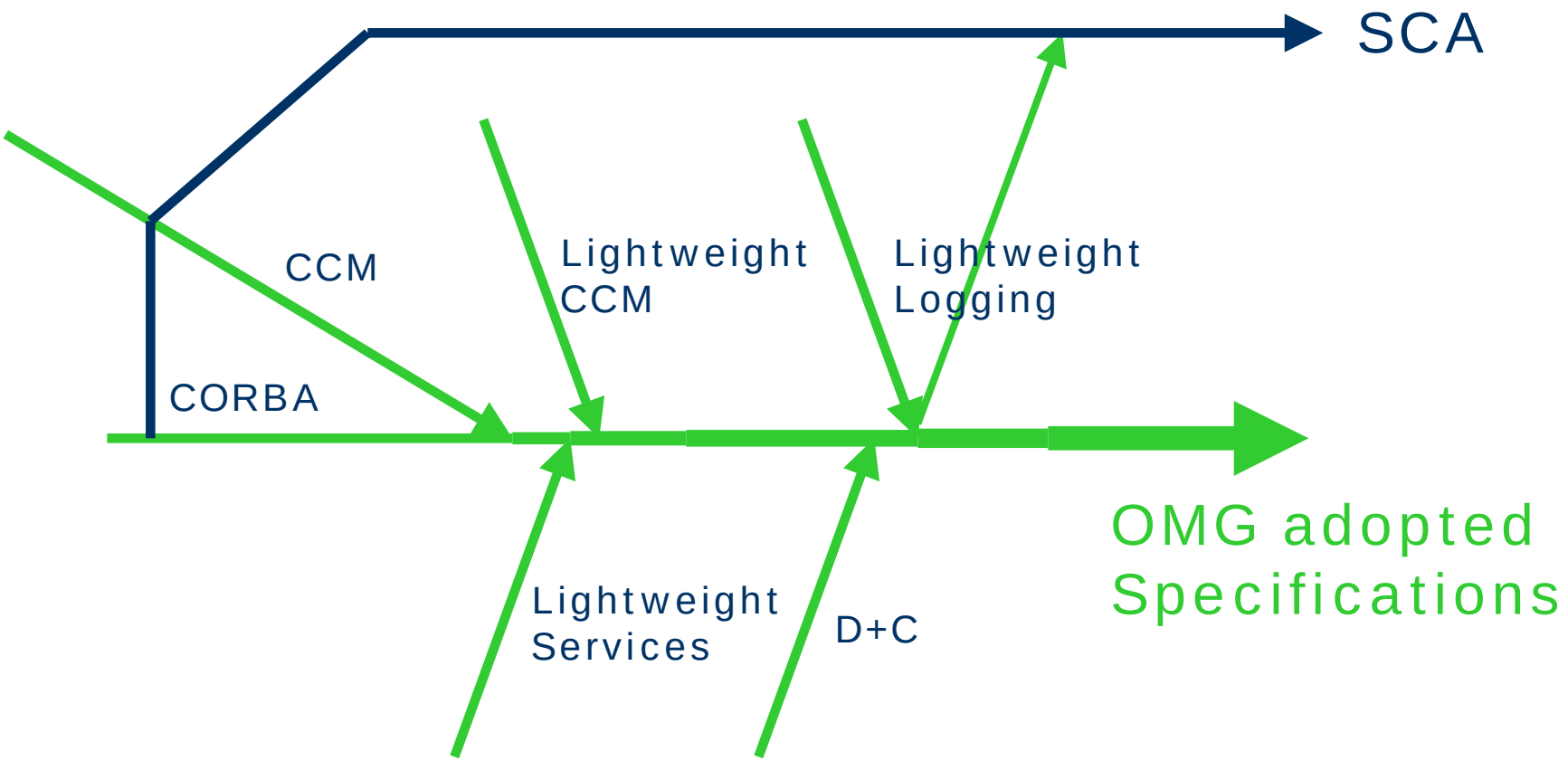
- Software Communications Architecture (current version 2.2.1)
  - CORBA-based component-oriented middleware
    - "Resource" components
    - Assemblies of interconnected resources
  - XML metadata similar to CCM
  - Basic hardware capacity model
  - Automated deployment
    - Inspired D+C effort
- Compliance mandatory for future JTRS Software Radio systems

SCA History

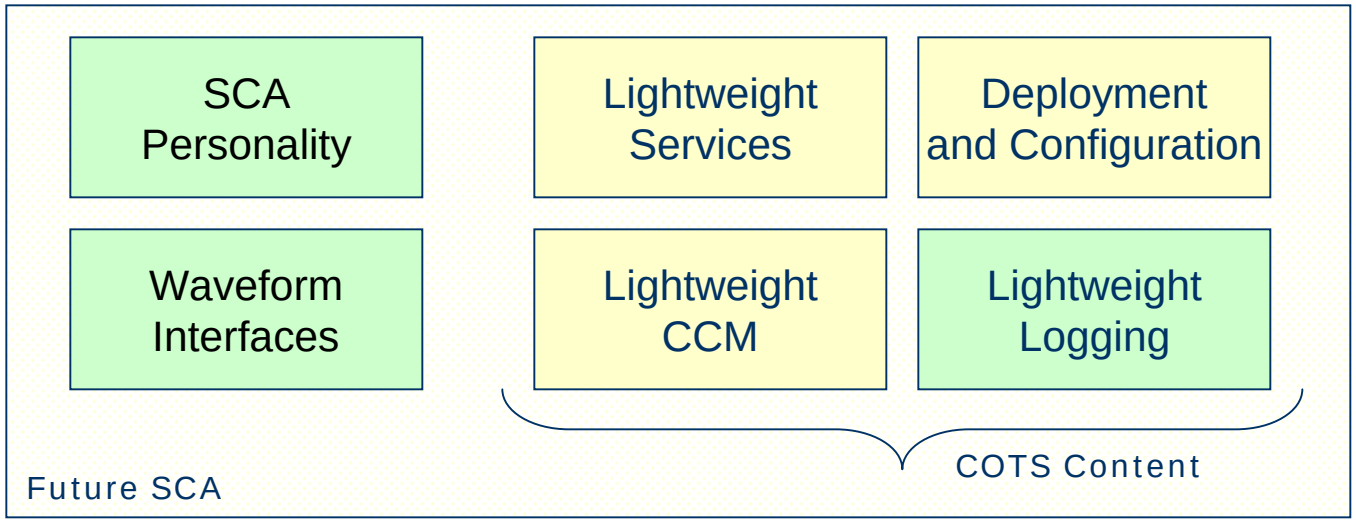
- SCA pioneered component-based development in embedded systems
  - Branched from CCM during finalization
  - Added important concepts of its own
- OMG specifications are catching up, exceeding SCA functionality
  - Lightweight CCM, Streams for CCM, Light-weight Log, Lightweight Services, D+C
  - Combine efforts in component-based embedded system development

SCA, OMG Timeline

- **Leverage OMG standardization efforts**



COTS SCA



- Leverage existing specifications
- Increase COTS Content in SCA
- Focus on Software Radio domain-specific aspects

Computer Systems, Inc.
MERCURY

Summary

The Ultimate Performance Machine

Summary

- **Lightweight CCM and D+C**
  - Evolution of original CCM
  - Enable distributed, component-based Applications — not only in embedded systems
  - E.g., applicable to Software Radio (SCA)
- **Status**
  - Adopted OMG specifications
  - Currently being finalized
  - Implementations expected later this year