Meeting the Challenges of Ultra-Large-Scale Distributed Real-time & Embedded (DRE) Systems

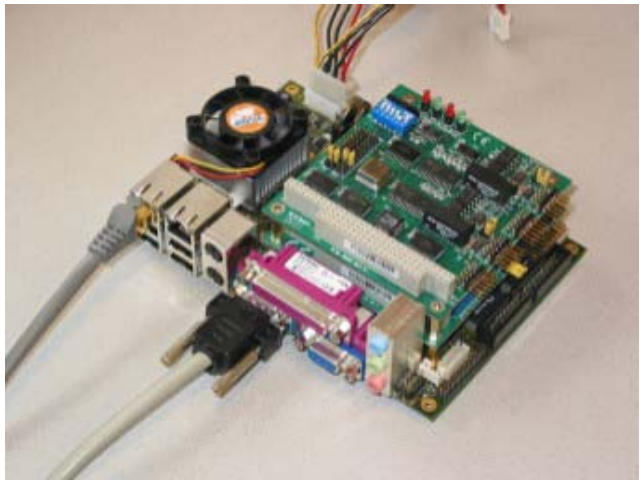
Wednesday, May 30, 2007, WPDRTS, Long Beach, CA

Dr. Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

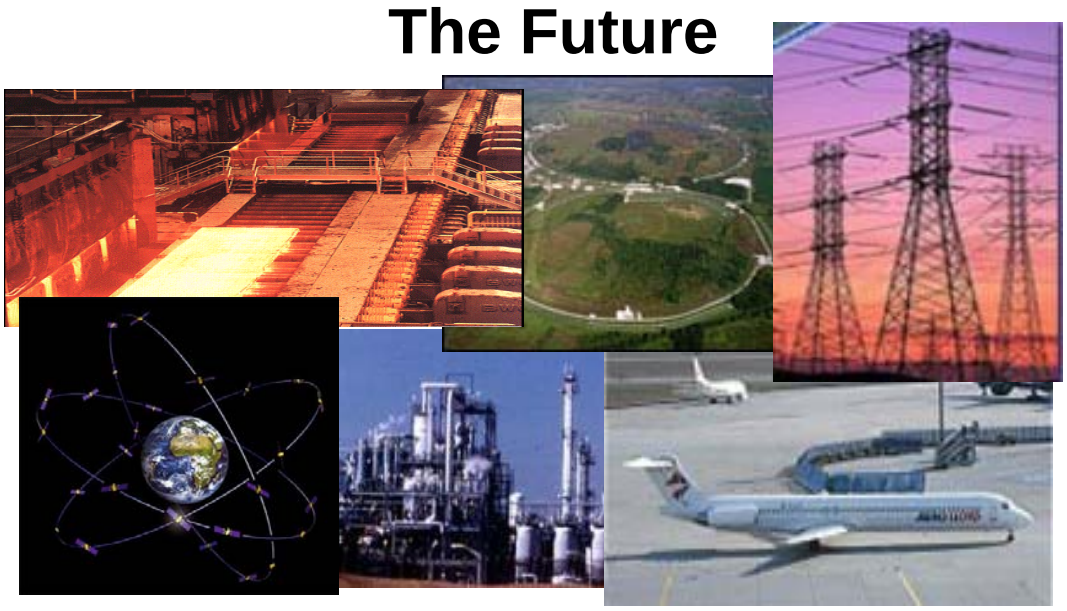
Institute for Software Integrated Systems
Vanderbilt University Nashville, Tennessee

Evolution in Distributed Real-time & Embedded (DRE) Systems

The Past



The Future



Standalone real-time & embedded systems

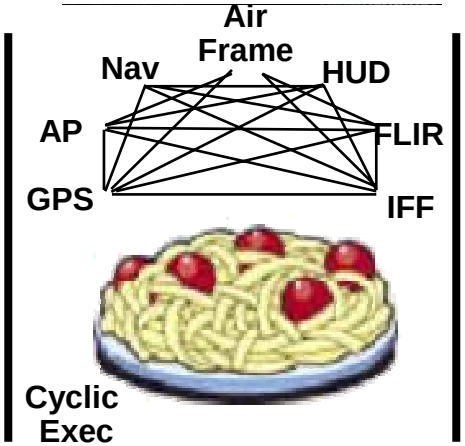
- Stringent quality of service (QoS) demands
  - e.g., latency, jitter, footprint
- Resource constrained

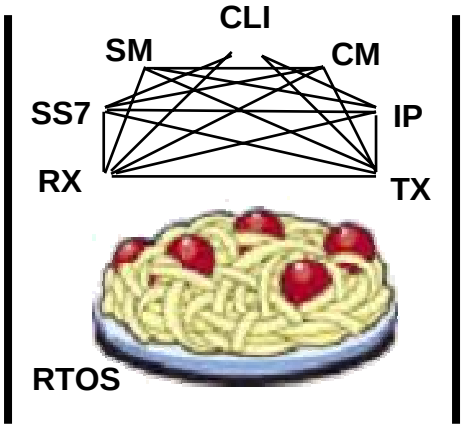
Enterprise distributed real-time & embedded (DRE) systems

- Network-centric "systems of systems"
- Stringent simultaneous QoS demands
  - e.g., dependability, security, scalability, etc.
- Dynamic context

This talk focuses on technologies for enhancing DRE system QoS, productivity, & quality

Evolution of DRE Systems Development





Technology Problems

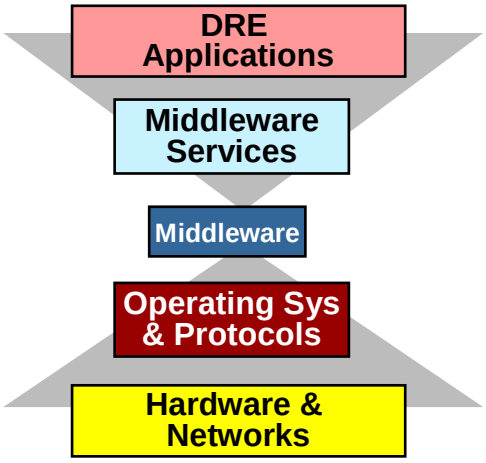
- Legacy DRE systems often tend to be:
  - Stovepiped
  - Proproprietary
  - Brittle & non-adaptive
  - Expensive
  - Vulnerable

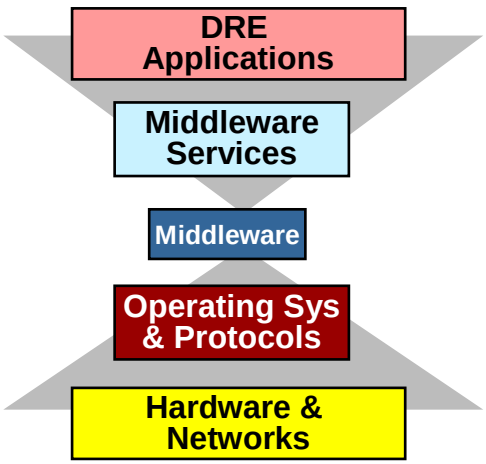
Mission-critical DRE systems have historically been built directly atop hardware

- Tedious
- Error-prone
- Costly over lifecycles

Consequence: Small changes to legacy software often have big (negative) impact on DRE system QoS & maintenance

Evolution of DRE Systems Development





Technology Problems

- Legacy DRE systems often tend to be:
  - Stovepiped
  - Proprietary
  - Brittle & non-adaptive
  - Expensive
  - Vulnerable

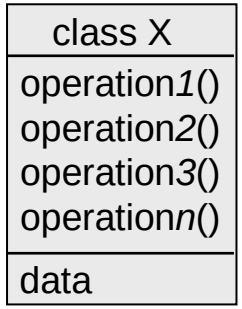
Mission-critical DRE systems historically have been built directly atop hardware

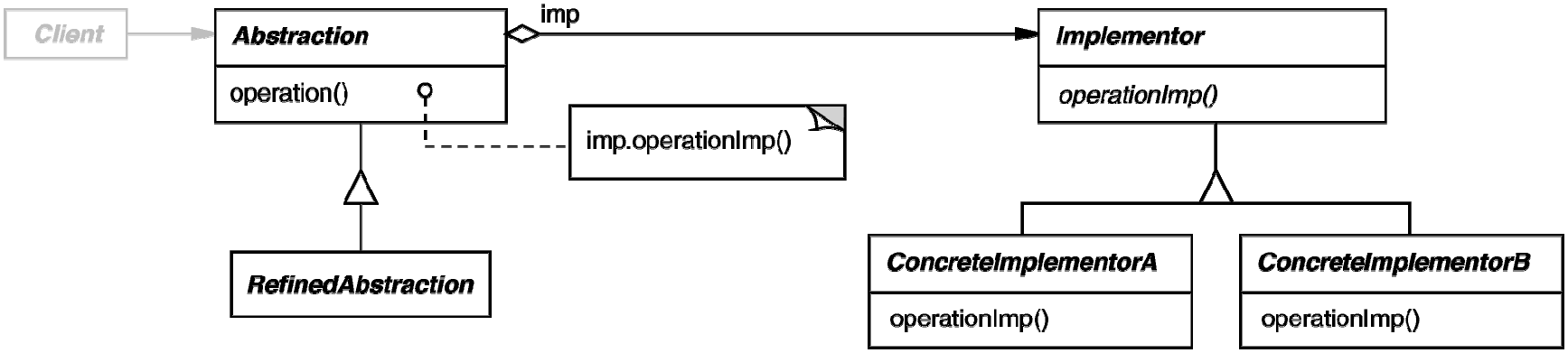
- Tedious
- Error-prone
- Costly over lifecycles

- Middleware has effectively factored out many reusable services from traditional DRE application responsibility
  - Essential for ***product-line architectures***
- Middleware is no longer the primary DRE system performance bottleneck

Where We Started: Object-Oriented Programming

- Object-oriented (OO) programming simplified software development through higher level abstractions & patterns, e.g.,
  - Associating related data & operations
  - Decoupling interfaces & implementations

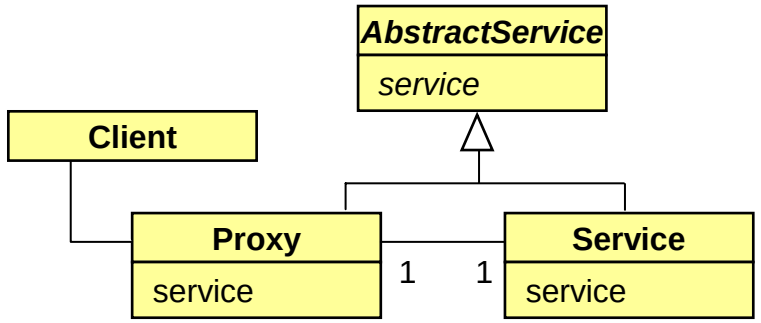


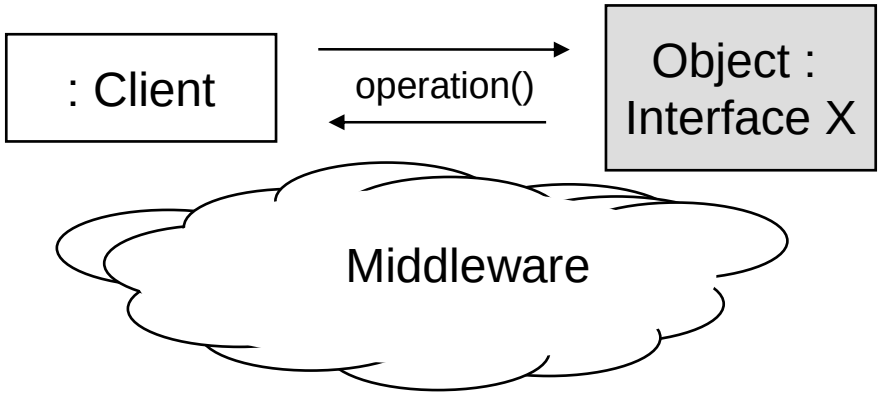


Well-written OO programs exhibit recurring structures that promote abstraction, flexibility, modularity, & elegance

Next Step: Distributed Object Computing (DOC)

- Apply the Broker pattern to abstract away lower-level OS & protocol-specific details for network programming
- Create distributed systems which are easier to model & build using OO techniques
- Result: robust distributed systems built with *distributed object computing (DOC) middleware*
  - e.g., CORBA, Java RMI, etc.

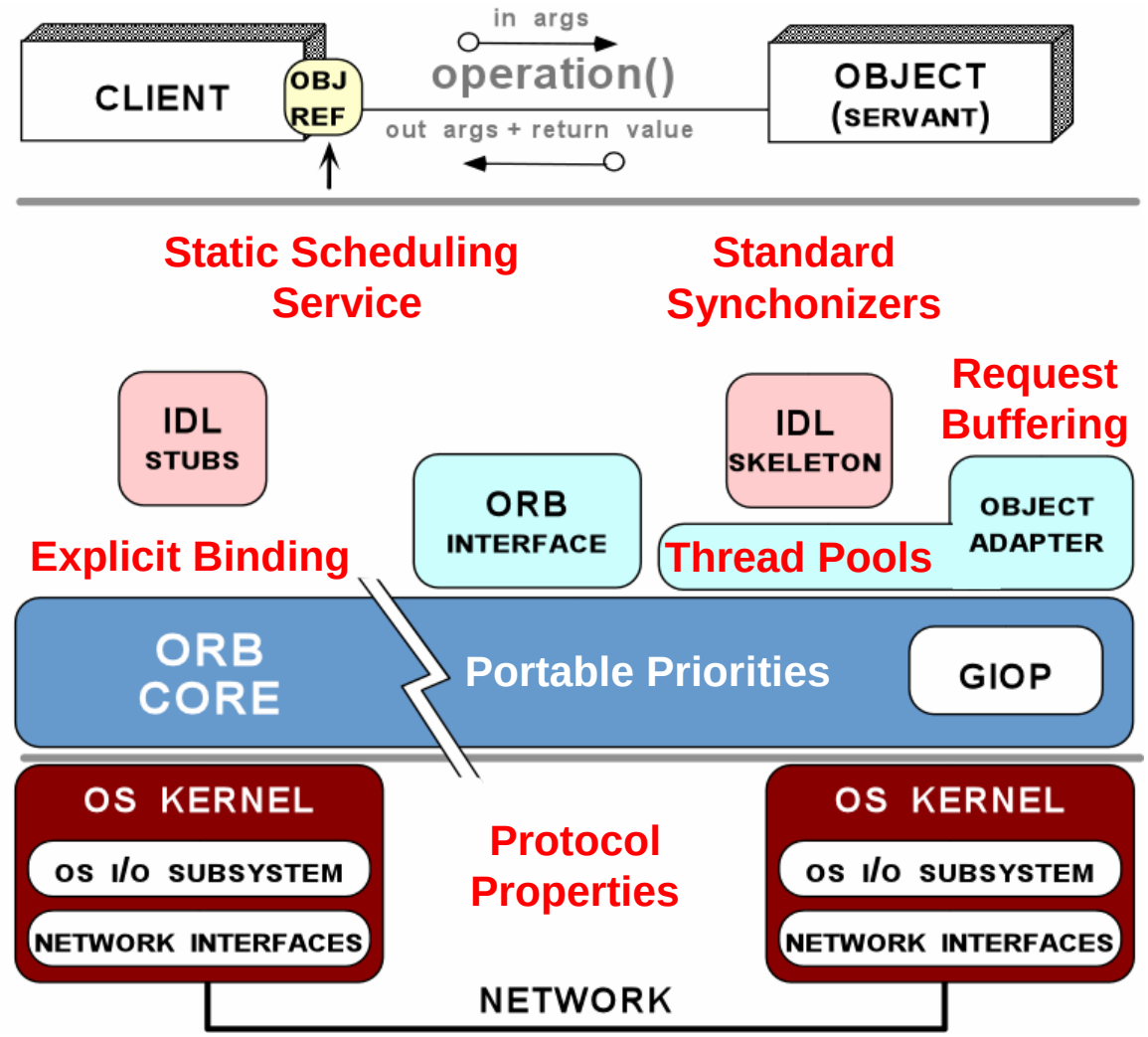




We now have more robust software & more powerful distributed systems

Overview of Real-time CORBA Standard

Client Propagation & Server Declared Priority Models

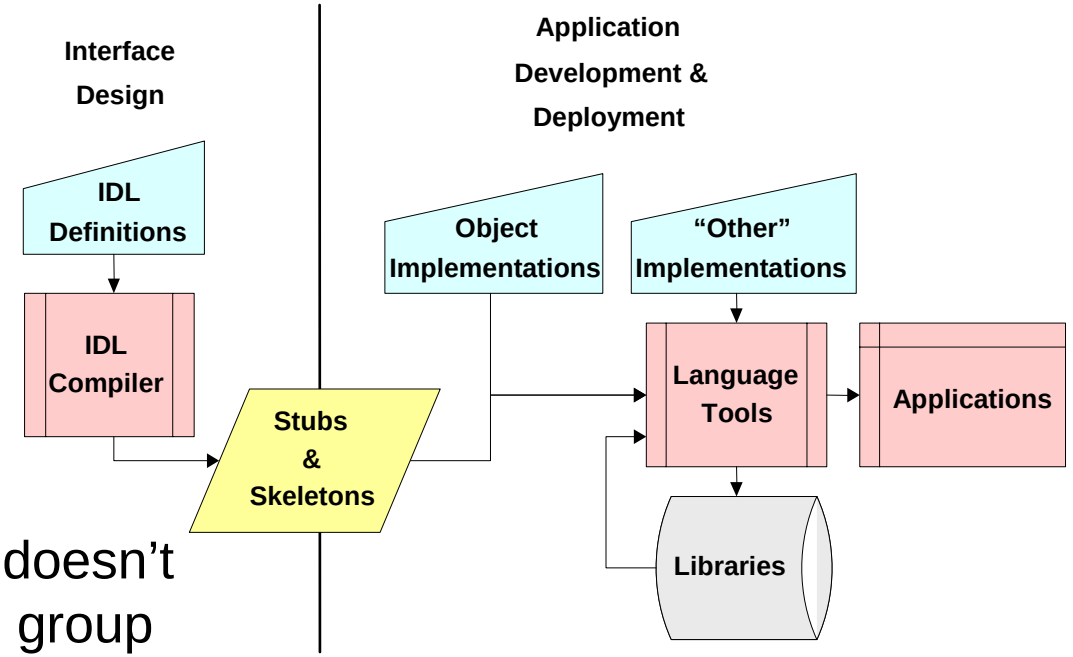


- ***Real-time CORBA*** adds quality of service (QoS) policies to classic CORBA to control:
  1. ***Processor Resources***
     - Thread pools
     - Priority models
     - Portable priorities
     - Standard synchronizers
     - Static scheduling service
  2. ***Network Resources***
     - Protocol policies
     - Explicit binding
  3. ***Memory Resources***
     - Request buffering
- These capabilities address some (but not all) DRE system development & QoS challenges

Real-time CORBA defines interfaces & policies, but *not* implementations

Drawbacks of DOC-based Middleware

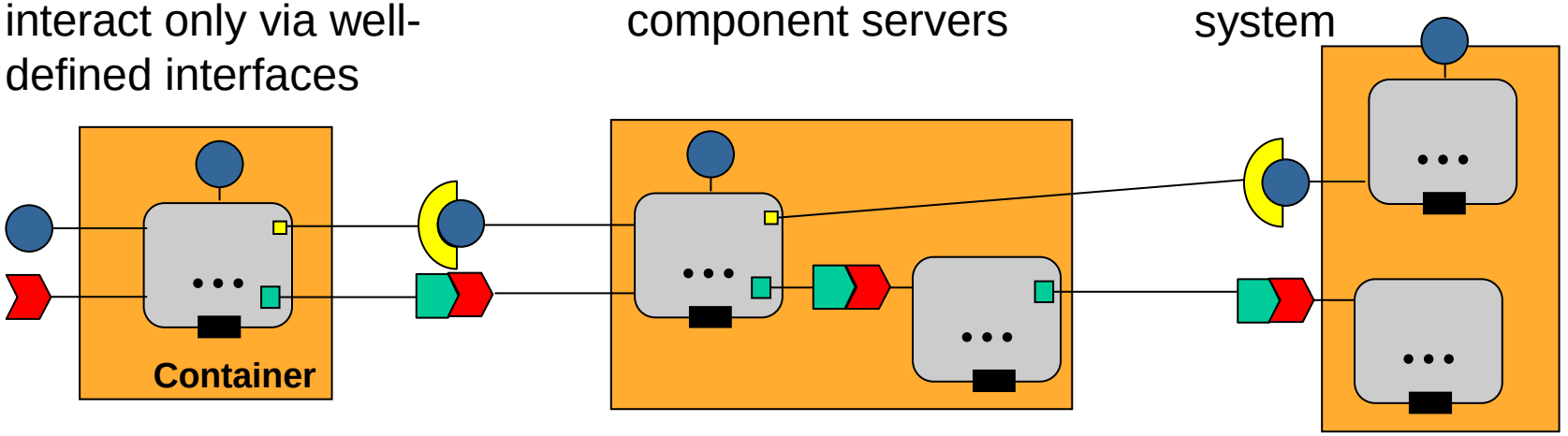
CORBA 2.x application development is unnecessarily tedious & error-prone



- CORBA 2.x IDL doesn't provide a way to group together related interfaces to offer a service family
  - Such "bundling" must be done by developers via CORBA idioms & patterns
- CORBA 2.x doesn't specify how configuration & deployment of objects should be done to create complete applications
  - Proprietary infrastructure & scripts are written by developers to enable this

Solution: Component Middleware

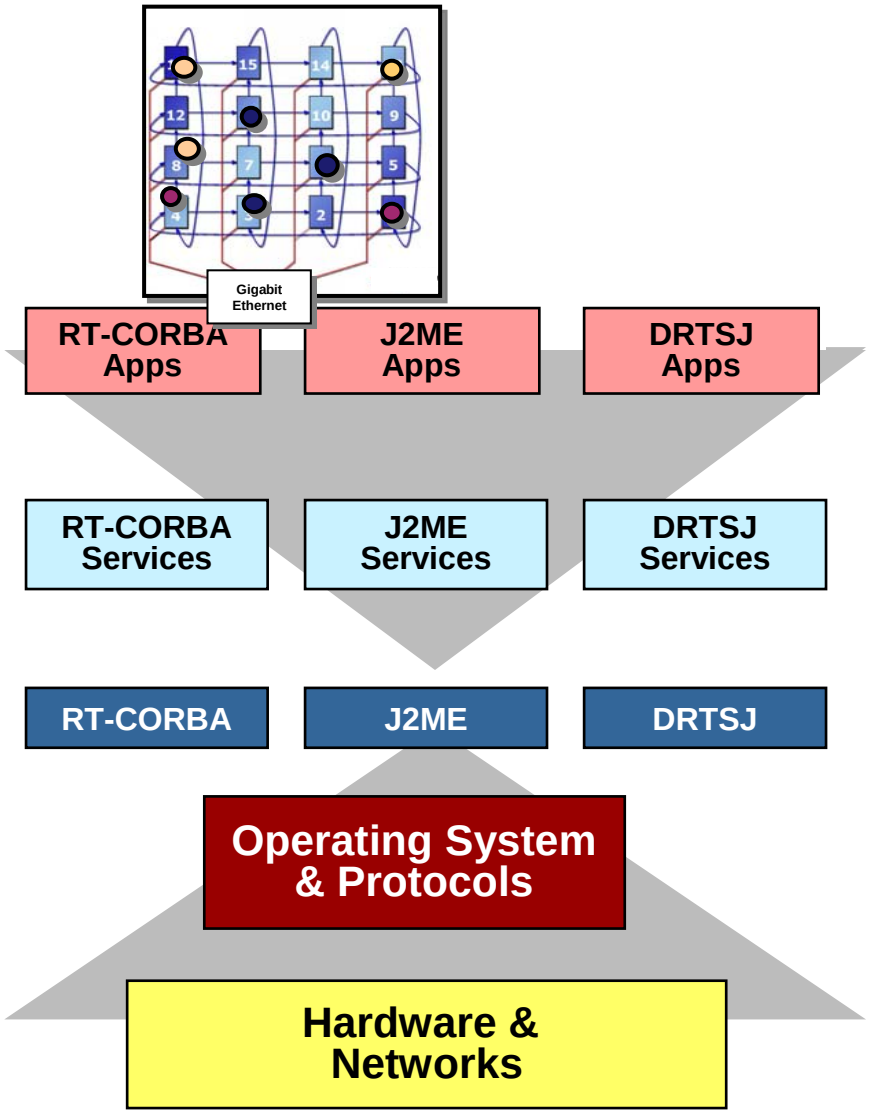
- Creates a standard "virtual boundary" around application **component** implementations that interact only via well-defined interfaces
- Define standard **container** mechanisms needed to execute components in generic component servers
- Specify the infrastructure needed to **configure & deploy** components throughout a distributed system



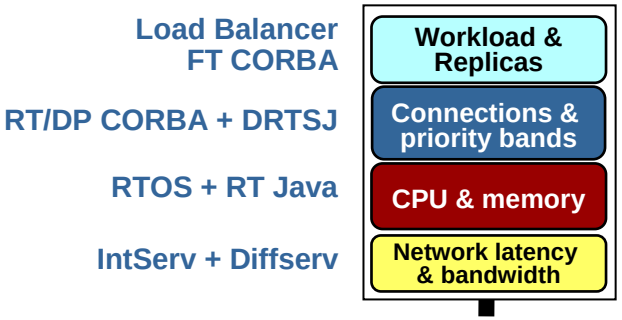

```
<ComponentAssemblyDescription id="a_HUDDisplay"> ...
 <connection>
   <name>GPS-RateGen</name>
   <internalEndPoint><portName>Refresh</portName><instance>a_GPS</instance></internalEndPoint>
   <internalEndPoint><portName>Pulse</portName><instance>a_RateGen</instance></internalEndPoint>
  </connection>
  <connection>
    <name>NavDisplay-GPS</name>
    <internalEndPoint><portName>Refresh</portName><instance>a_NavDisplay</instance></internalEndPoint>
    <internalEndPoint><portName>Ready</portName><instance>a_GPS</instance></internalEndPoint>
  </connection> ...
</ComponentAssemblyDescription>
```

Component middleware defines interfaces, policies, & *some* implementations

DRE Systems: The Challenges Ahead



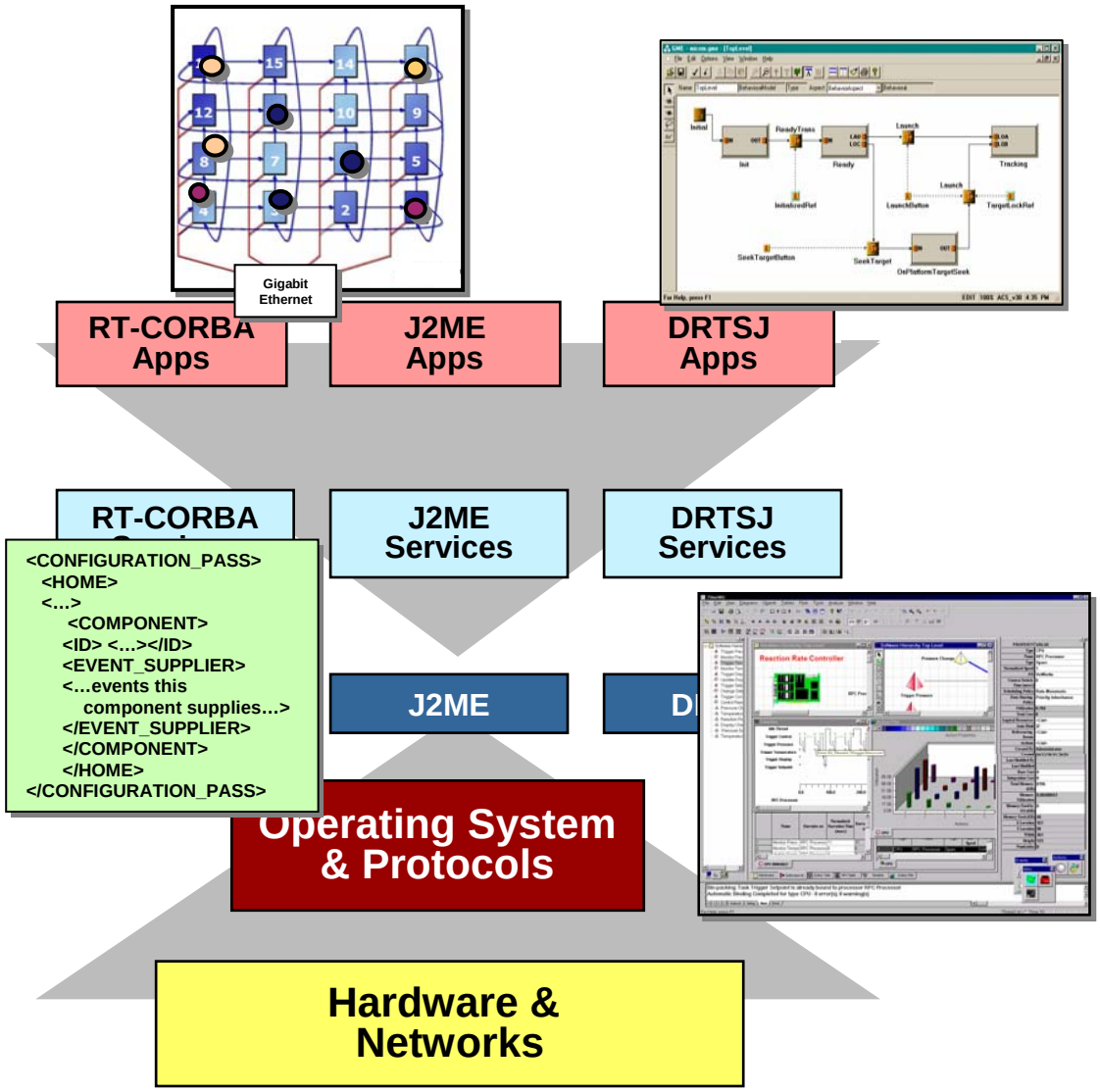
- Limit to how much application functionality can be refactored into reusable COTS middleware
- Middleware itself has become very hard to use & provision statically & dynamically



- Component-based DRE systems are also very hard to deploy & configure
- There are many middleware platform technologies to choose from

Middleware alone cannot solve large-scale DRE system challenges!

Promising Solution: *Model-based Software Development*



- Develop, validate, & standardize generative software technologies that:
  1. ***Model***
  2. ***Analyze***
  3. ***Synthesize &***
  4. ***Provision***

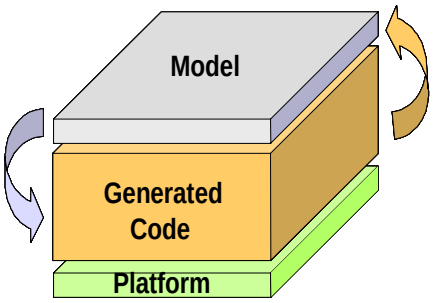
  multiple layers of middleware & application components that require simultaneous control of multiple QoS properties end-to-end
- Partial specialization is essential for inter-/intra-layer optimization & advanced product-line architectures

Goal is to ***enhance developer productivity & software quality*** by providing ***higher-level languages & tools*** for middleware/application developers & users

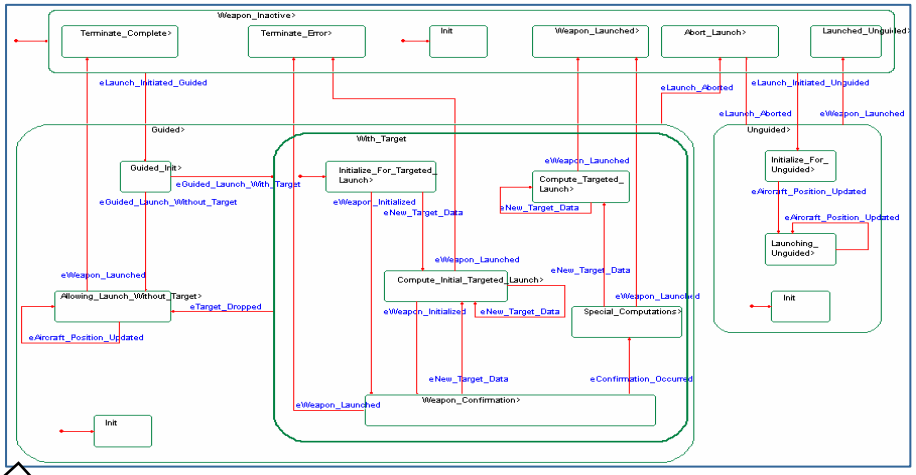
Technology Evolution (1/4)

Level of Abstraction

Programming Languages & Platforms



Model-Driven Engineering (MDE)



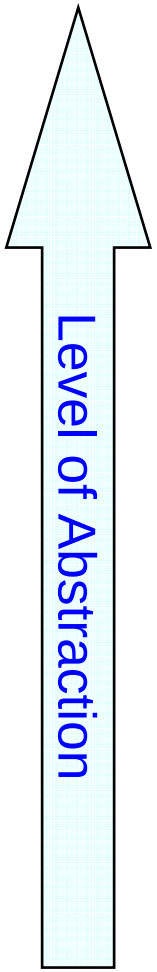
Translation
Translation
Translation

Large Semantic Gap

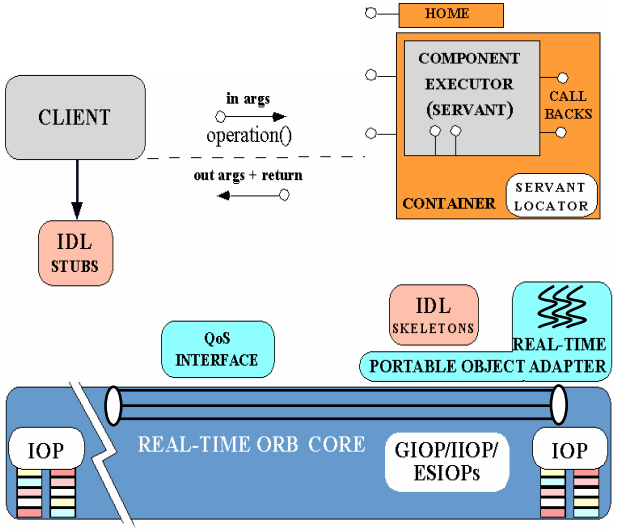
- State chart
- Data & process flow
- Petri Nets

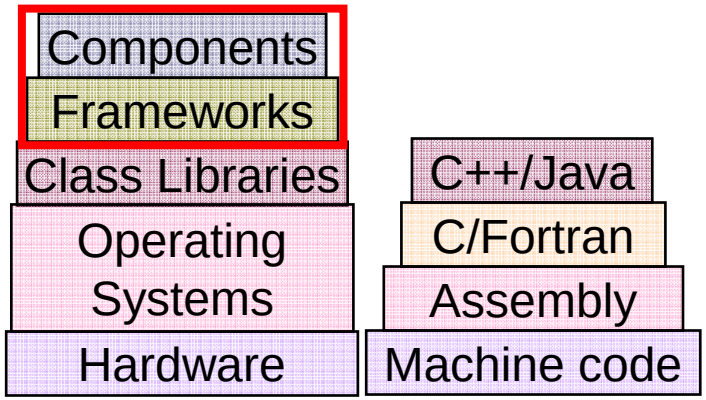
Operating Systems	C/Fortran
	Assembly
Hardware	Machine code

Technology Evolution (2/4)

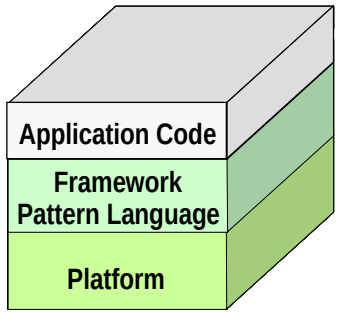


Programming Languages & Platforms





- Newer 3rd-generation languages & platforms have raised abstraction level significantly
  - "Horizontal" platform reuse alleviates the need to redevelop common services



- There are two problems, however:
  - Platform complexity evolved faster than 3rd-generation languages
  - Much application/platform code still (unnecessarily) written manually

Technology Evolution (3/4)

Level of Abstraction

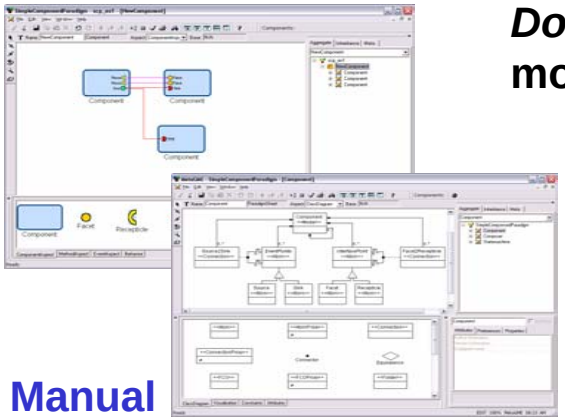
Programming Languages & Platforms

Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

Saturation!!!!

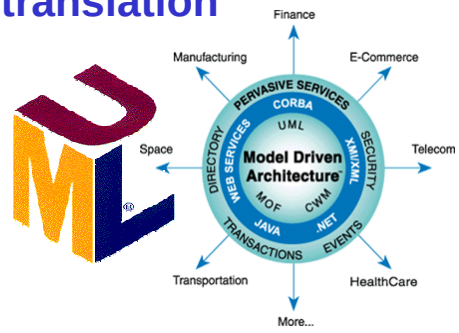
Model-Driven Engineering (MDE)



Domain-specific **modeling languages**

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Manual translation



Domain-independent modeling languages

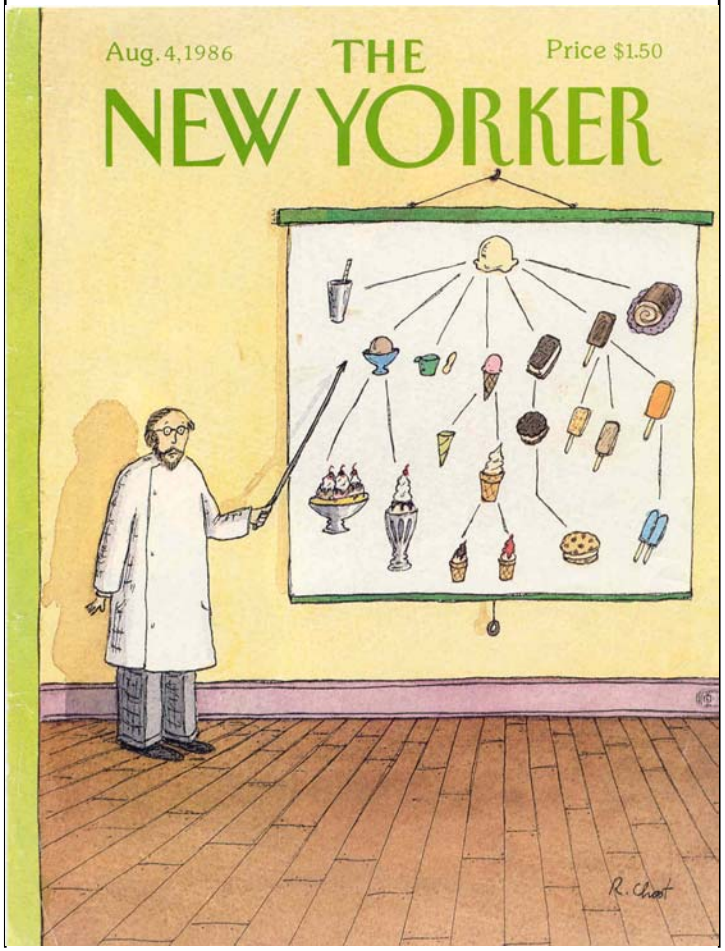
- State Charts
- Interaction Diagrams
- Activity Diagrams

Semi-automated

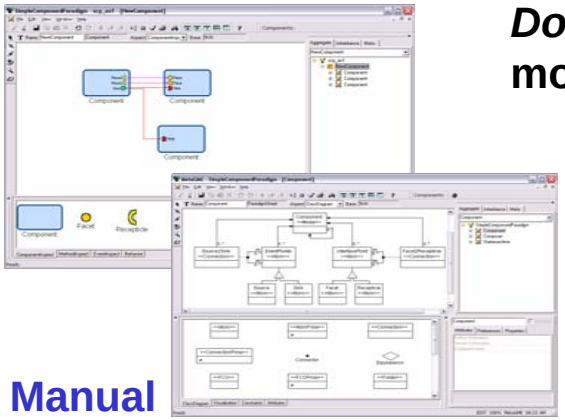
Technology Evolution (3/4)

Level of Abstraction

Programming Languages & Platforms



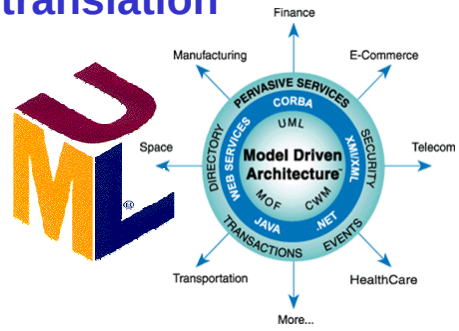
Model-Driven Engineering (MDE)



Domain-specific **modeling languages**

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Manual translation



Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

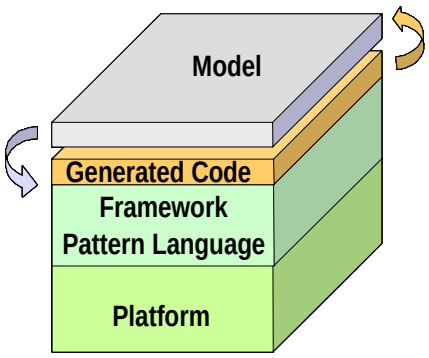
Semi-automated

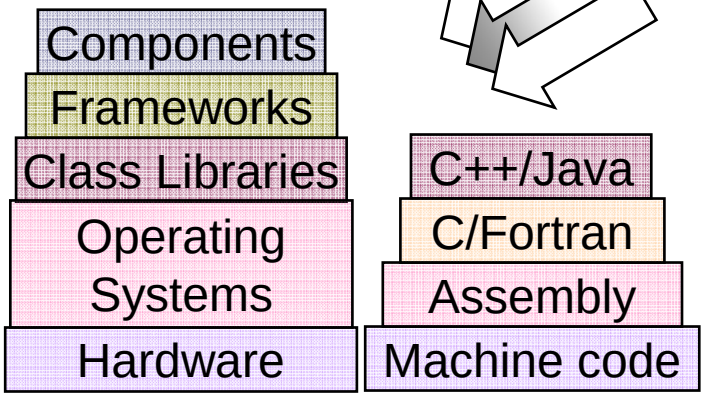
- OMG is standardizing MDE via MIC PSIG
  - mic.omg.org

Technology Evolution (3/4)

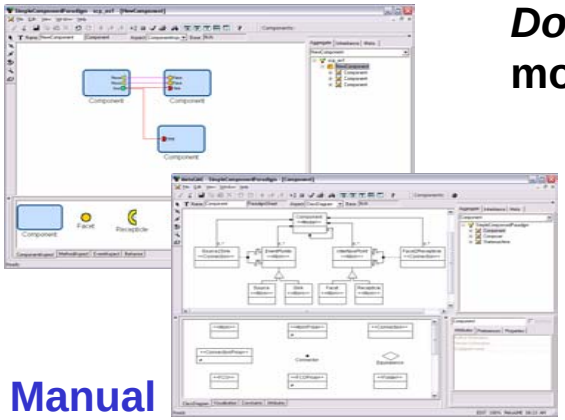
Level of Abstraction

Programming Languages & Platforms





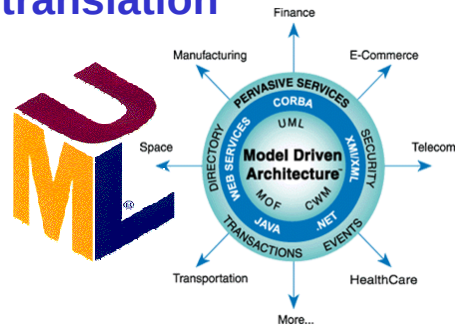
Model-Driven Engineering (MDE)



Domain-specific **modeling languages**

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Manual translation



Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

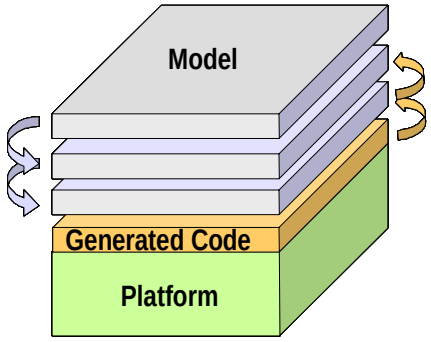
Semi-automated

- OMG is standardizing MDE via MIC PSIG
  - mic.omg.org

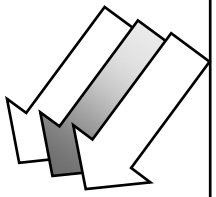
Technology Evolution (4/4)

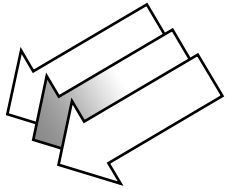
Level of Abstraction

Programming Languages & Platforms



Needs Automation

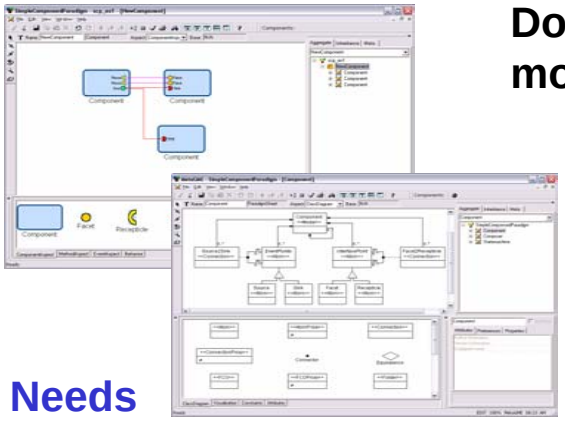




Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

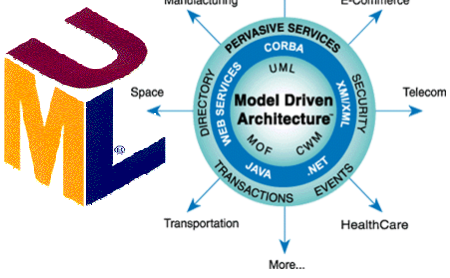
Model-Driven Engineering (MDE)



Domain-specific modeling languages

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Needs Automation



Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

Needs Automation

Research is needed to automate DSMLs & model translators

See February 2006 IEEE Computer special issue on MDE techniques & tools

Relevant Academic Work

- **CADENA**
  - Integrated environment for static analysis using model-checking
- **VEST**
  - DSML developed in GME
  - Pre-defined component Libraries
  - Aspect checks
  - Prescriptive aspect library
- **ESML**
  - DSML developed w/GME
  - Targets PRiSM (Boeing's Bold-stroke component model)
- **Ptolemy II**
  - Modeling, simulation, and design of concurrent systems
  - Allows defining systems based on Models of Computation

K-STATE Kansas State University

cadena.projects.cis.ksu.edu

PCES

Computer Science at the University of Virginia

www.cs.virginia.edu/~stankovic/vest.html

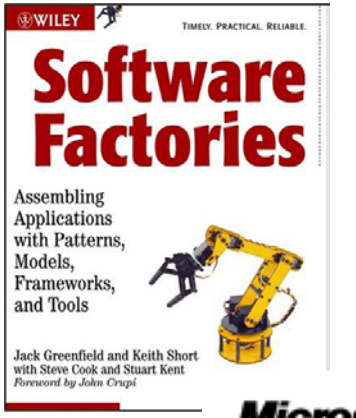
MoBIES

www.isis.vanderbilt.edu/projects/mobies

ptolemy.eecs.berkeley.edu

Ptolemy Project — HETEROGENEOUS MODELING AND DESIGN, UC BERKELEY, EECS

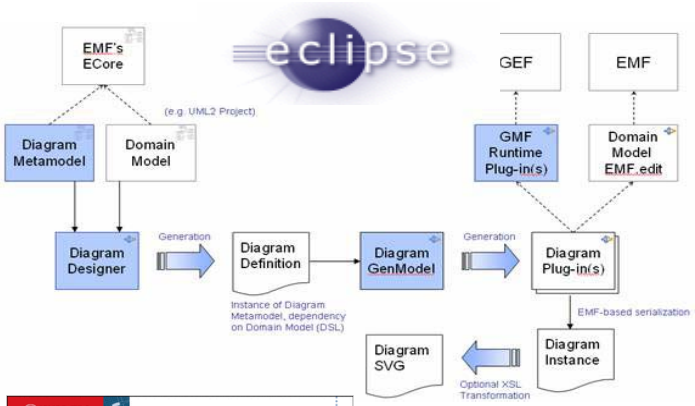
Relevant Commercial Work



Microsoft

- Software Factories go beyond "models as documentation" by
  - Using highly-tuned DSL & XML as source artifacts &
  - Capturing life cycle metadata to support high-fidelity model transformation, code generation & other forms of automation

www.softwarefactories.com



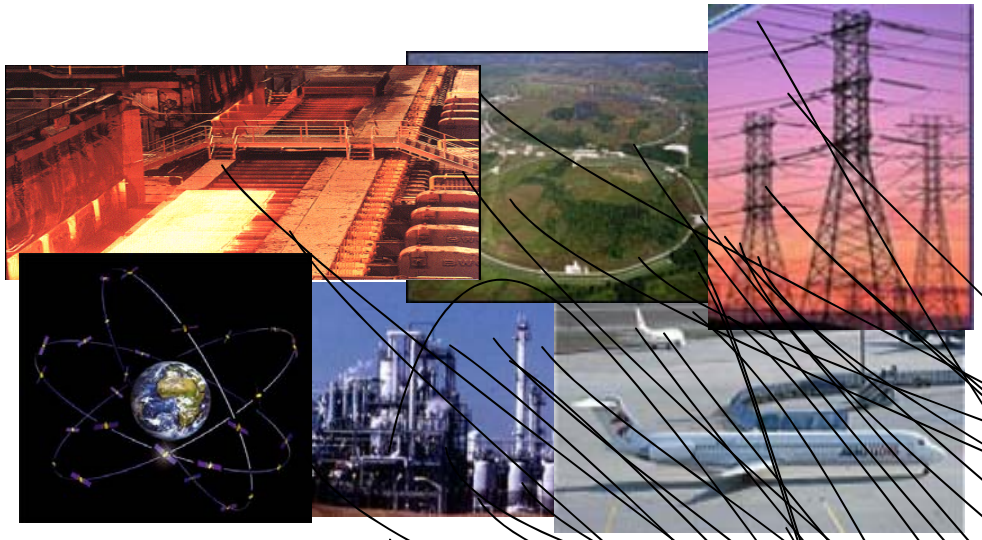
- The Graphical Modeling Framework (GMF) forms a generative bridge between EMF & GEF, which linkes diagram definitions to domain models as input to generation of visual editors
- GMF provides this framework, in addition to tools for select domain models that illustrate its capabilities

www.eclipse.org/gmf/

- openArchitectureWare (oAW) is a modular MDA/MDE generator framework implemented in Java
- It supports parsing of arbitrary models & a language family to check & transform models, as well as generate code based on them

www.openarchitectureware.org

New Challenges: Ultra-Large-Scale (ULS) Systems



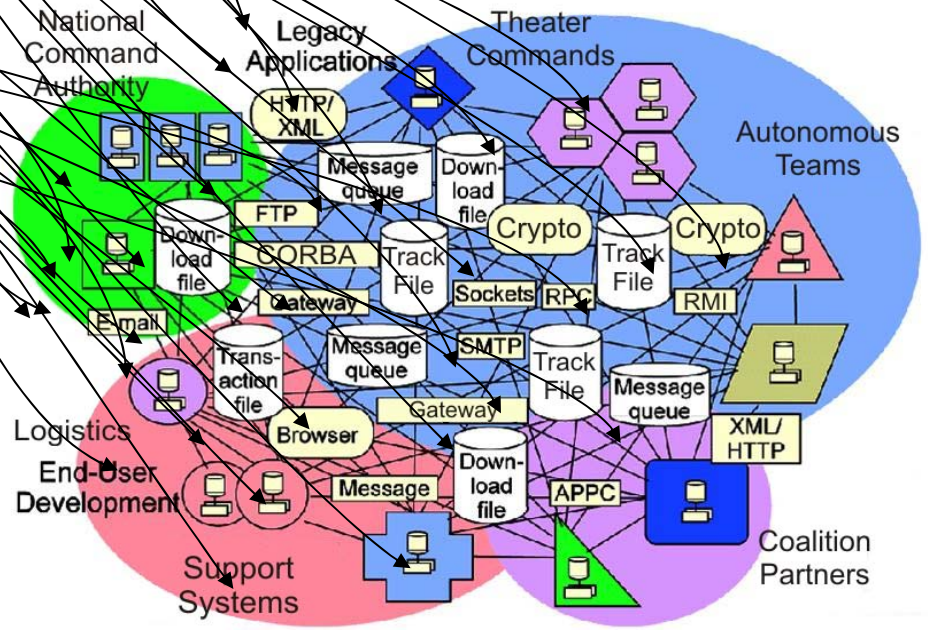
Key ULS ***problem space*** challenges

- Highly dynamic & distributed development & operational environments
- Stringent simultaneous quality of service (QoS) demands
- Very diverse & complex network-centric application domains

Key ULS ***solution space*** challenges

- Enormous accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous platform, language, & tool environments

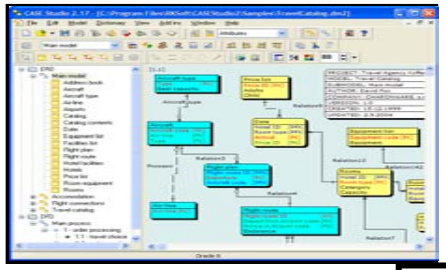
Mapping *problem space requirements* to *solution space artifacts* is very hard



Key R&D Challenges for ULS Systems

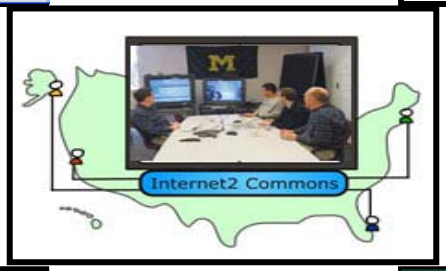
Developers & users of ULS systems face challenges in multiple dimensions

Logical View

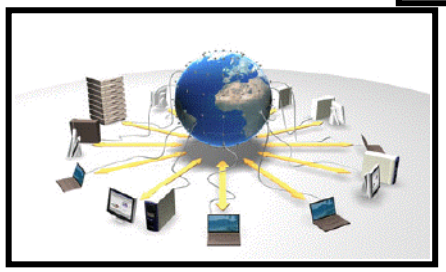


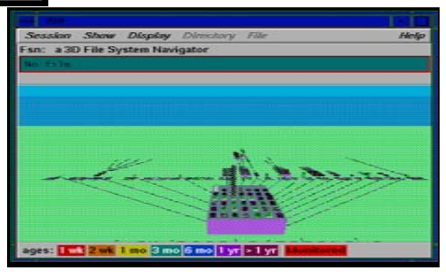
Process View

Use Case View



Physical View





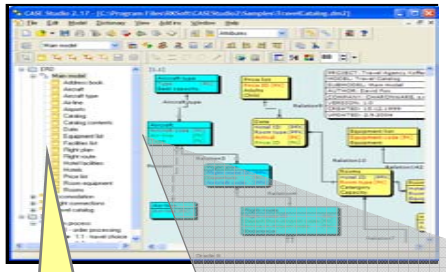
Development View

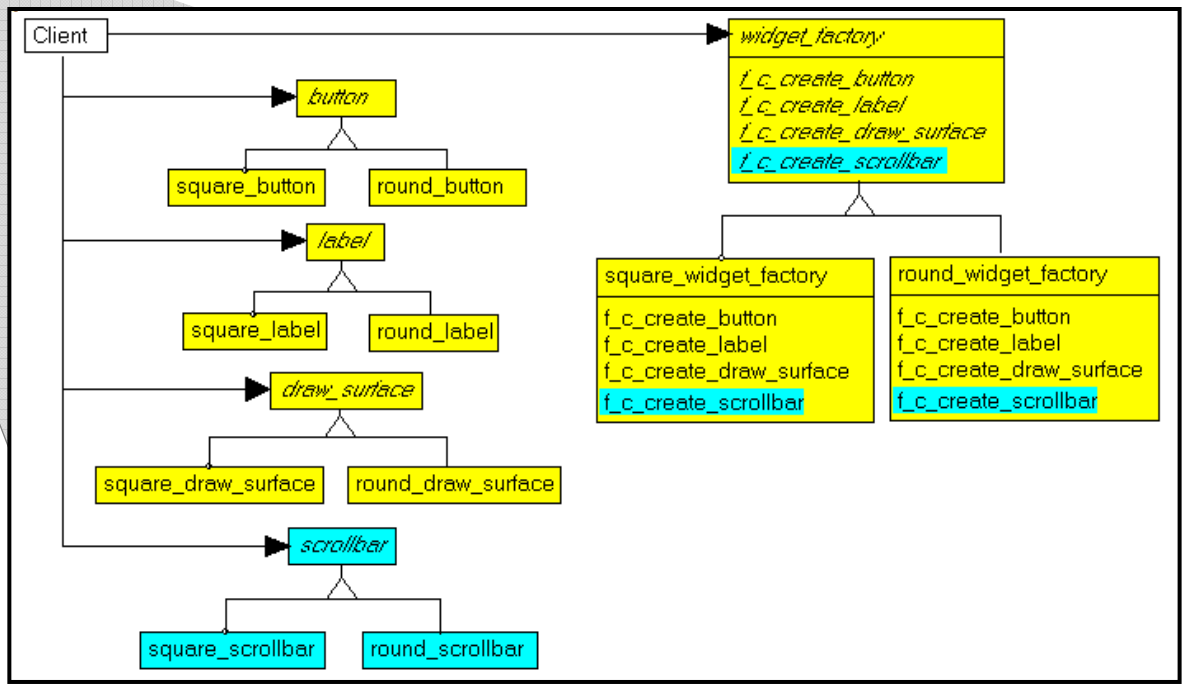
Of course, developers of today's large-scale network-centric systems also face these challenges, but they can often "brute force" solutions…

Key R&D Challenges for ULS Systems

- Popular technologies & tools provide inadequate support for
  - Expressing design intent more clearly using domain concepts
  - Checking pre-/post-conditions & invariants
  - Specifying & analyzing dependencies

Logical View

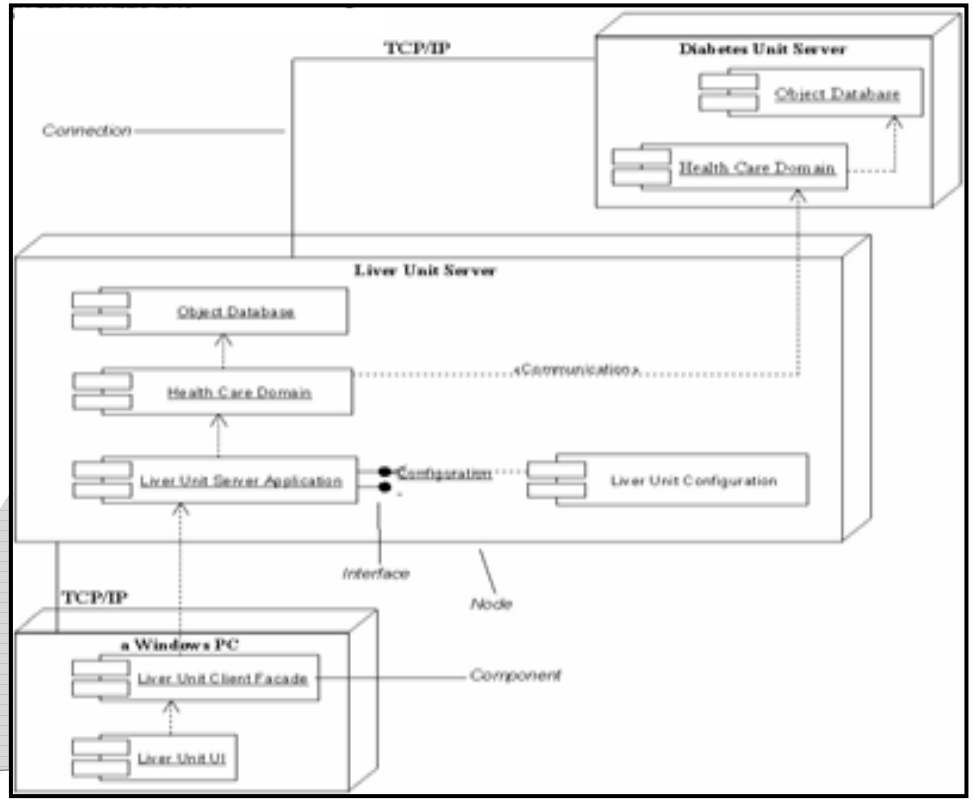




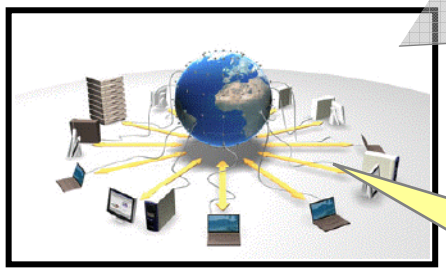
Determining units of abstraction for system (de)composition, reuse, & validation

Key R&D Challenges for ULS Systems

- Popular technologies & tools provide inadequate support for
  - Configuring & customizing components for application requirements & run-time environments
  - Automated deployment, i.e., mapping of components onto nodes in target environments



Physical View



Integrating/deploying diverse new & reusable application components in a networked environment to ensure end-to-end QoS requirements

Key R&D Challenges for ULS Systems

Devising execution architectures, concurrency models, & communication styles that ensure multi-dimensional QoS & correctness of new/reusable components

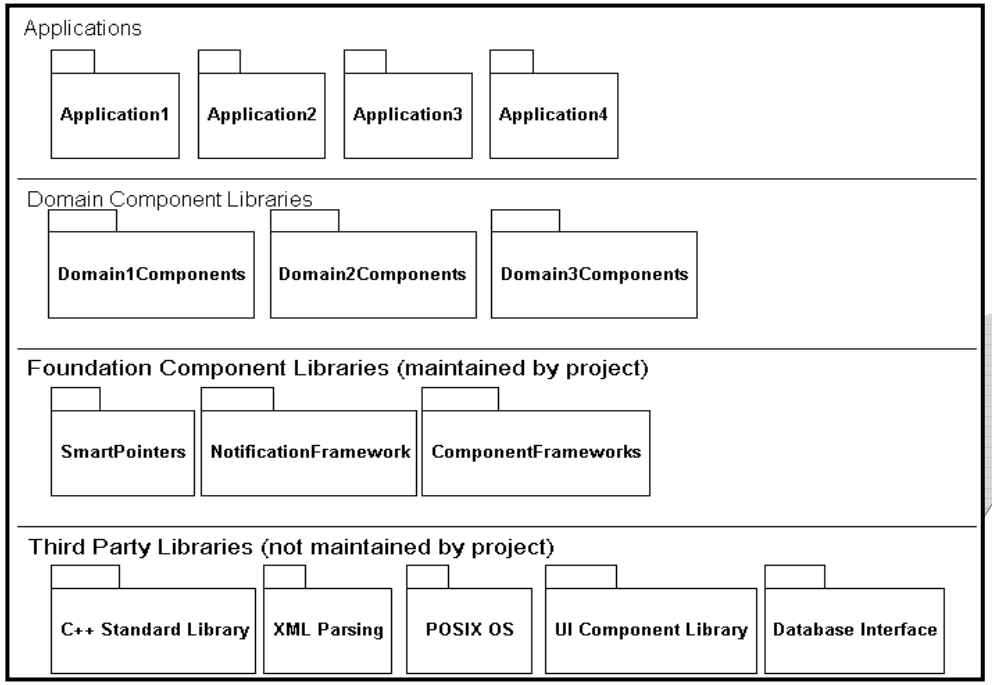
Process View

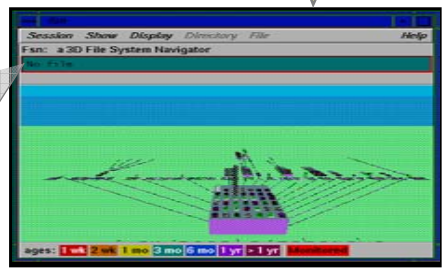
- Popular technologies & tools provide inadequate support for
  - Identifying & reducing performance & robustness risks early in ULS system lifecycles
  - Satisfying multiple (often conflicting) QoS demands
    - e.g., secure, real-time, reliable
  - Satisfying QoS demands in face of fluctuating/insufficient resources
    - e.g., mobile ad hoc networks (MANETs)

Key R&D Challenges for ULS Systems

- Popular technologies & tools provide inadequate support for avoiding
  - Cyclic dependencies, which make unit testing & reuse hard
  - Excessive link-time dependencies, which bloat size of executables
  - Excessive compile-time dependencies, where small changes trigger massive recompiles

(De)composing systems into separate, reusable modules (e.g., packages, subsystems, libraries) that achieve/preserve QoS properties

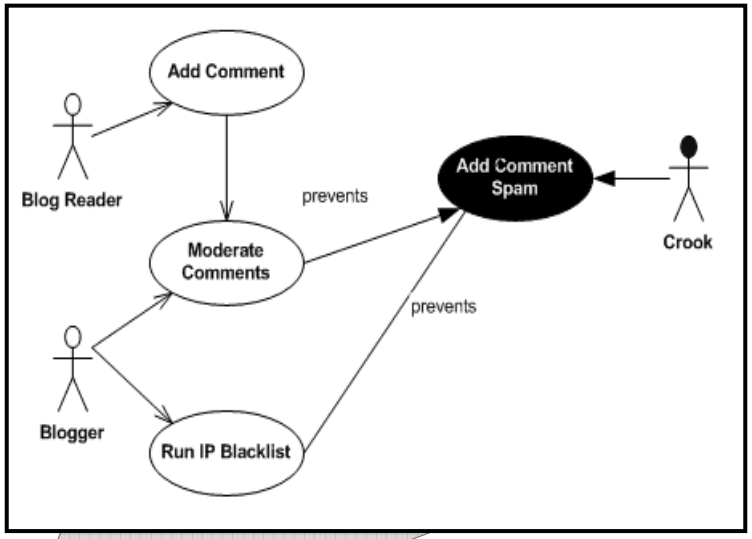




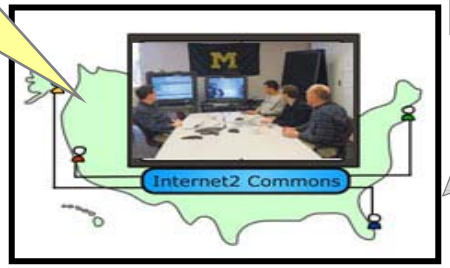
Development View

Key R&D Challenges for ULS Systems

Capturing functional & QoS requirements of systems & reconciling them with other views during evolution



Use Case View

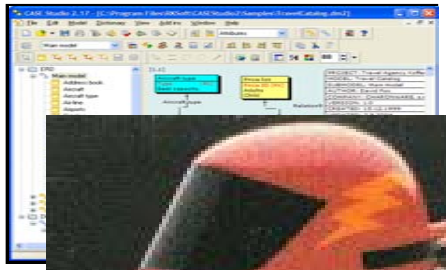


- Popular technologies & tools provide inadequate support for
  - Ensuring semantic consistency & traceability between requirements & software artifacts
  - Visualizing software architectures, designs, & implementations from multiple views
  - Effective collaboration between users & distributed development teams

Key R&D Challenges for ULS Systems

Developers & users of ULS systems face challenges in multiple dimensions

Logical View



Process View

Physical View

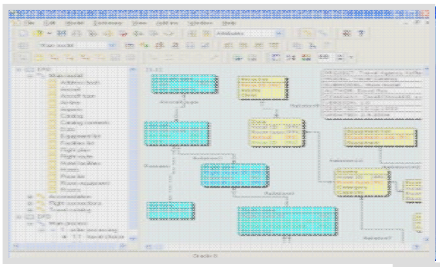
Development View

Solving these challenges requires much more than simply retrofitting our current tools, platforms, & processes!

Key R&D Challenges for ULS Systems

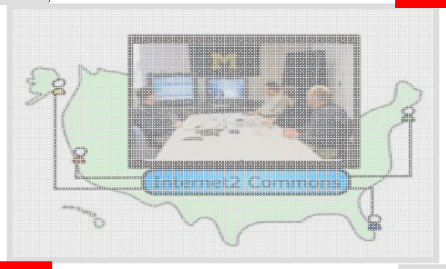
Developers & users of ULS systems face challenges in multiple dimensions

Logical View

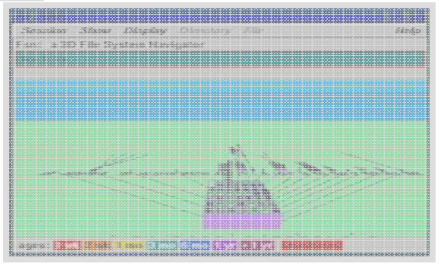


Process View

Use Case View

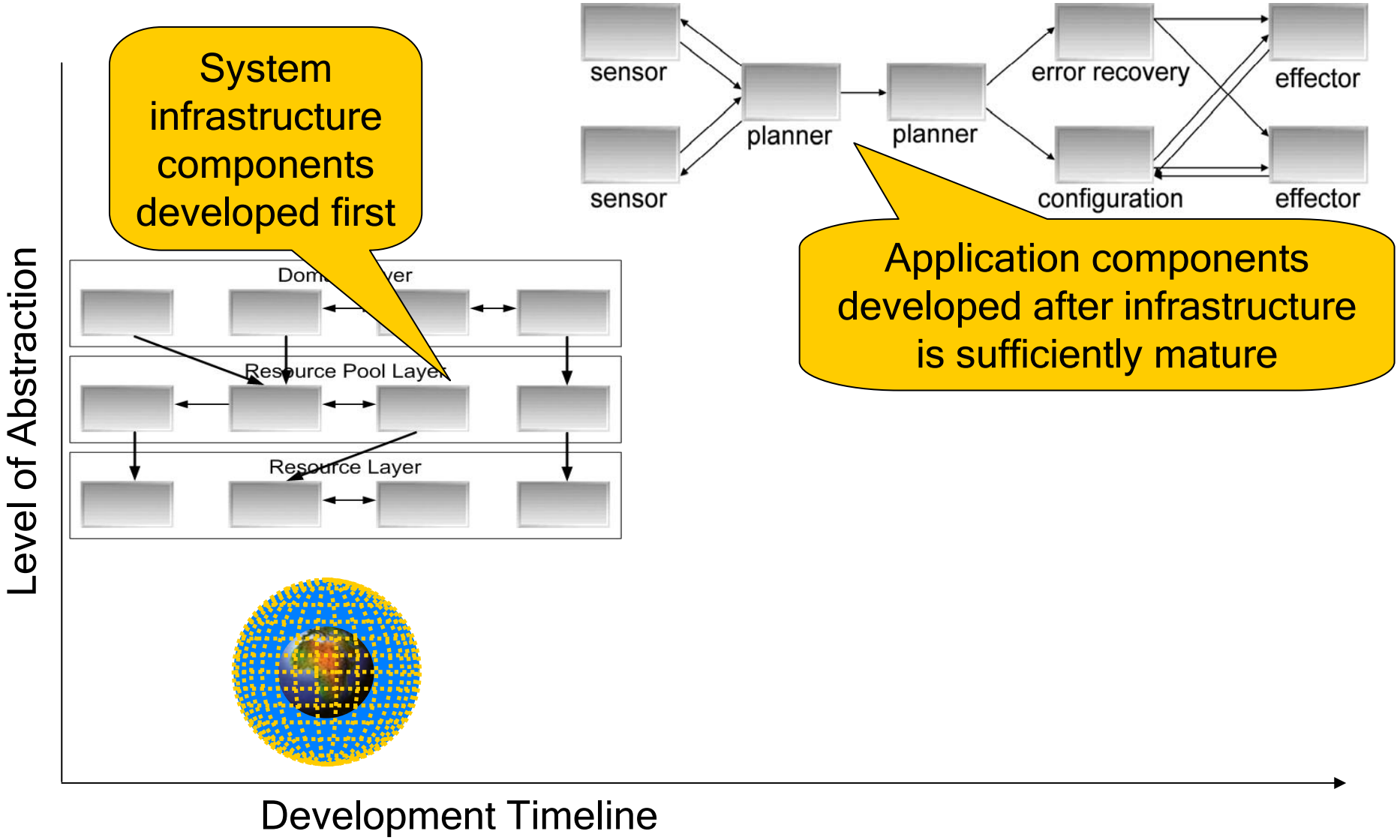


Physical View



Development View

Serialized Phasing is Common in ULS Systems



Serialized Phasing is Common in ULS Systems

System integration & testing is performed after application development is finished

sensor
sensor
planner
planner
error recovery
configuration
effector
effector

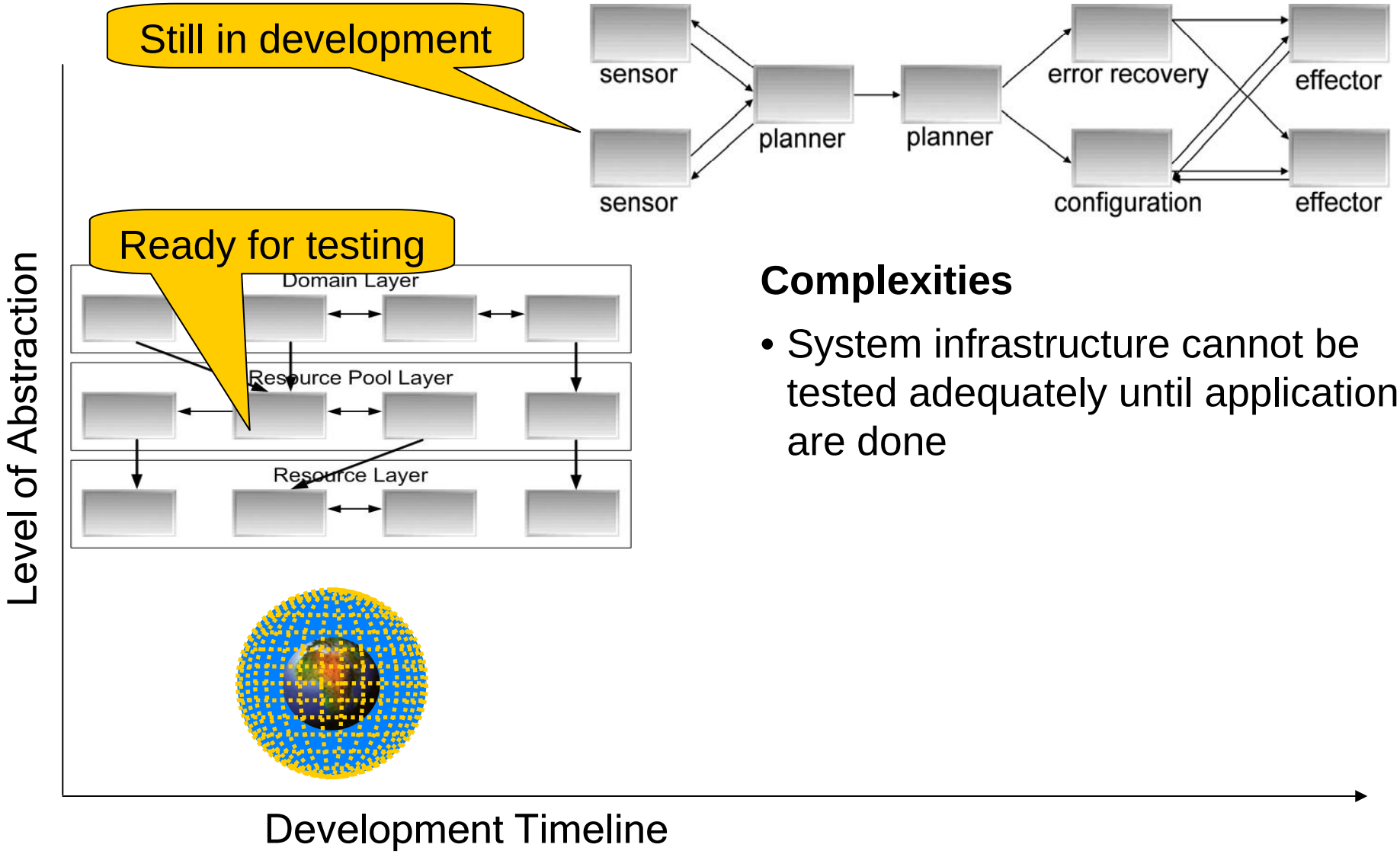
Domain
Resource P er
Resource er

Level of Abstraction

Development Timeline

Integration Surprises!!!

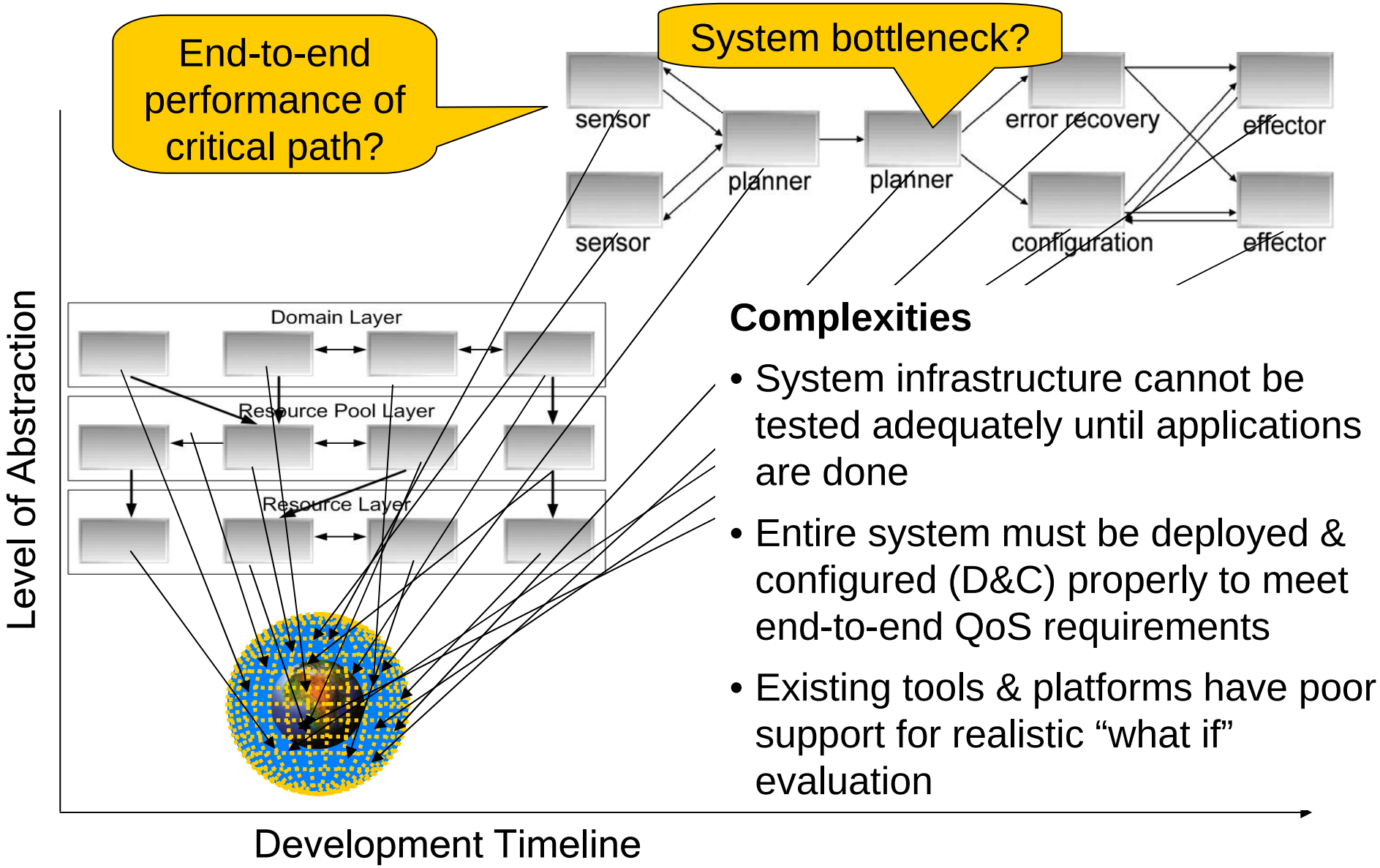
Complexities of Serialized Phasing



Complexities

- System infrastructure cannot be tested adequately until applications are done

Complexities of Serialized Phasing

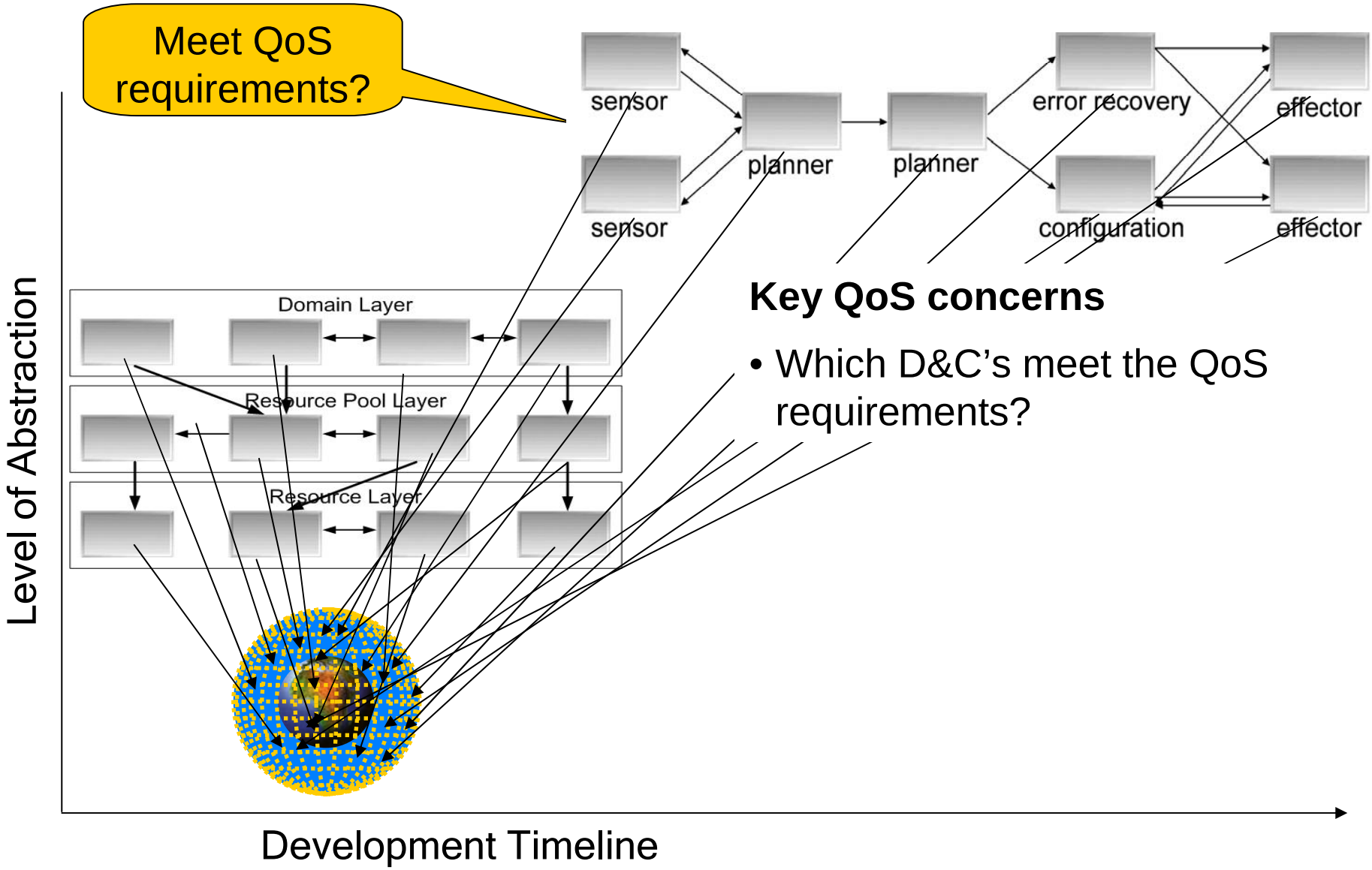


Complexities

- System infrastructure cannot be tested adequately until applications are done
- Entire system must be deployed & configured (D&C) properly to meet end-to-end QoS requirements
- Existing tools & platforms have poor support for realistic “what if” evaluation

QoS needs of components in ULS systems often unknown until late in lifecycle

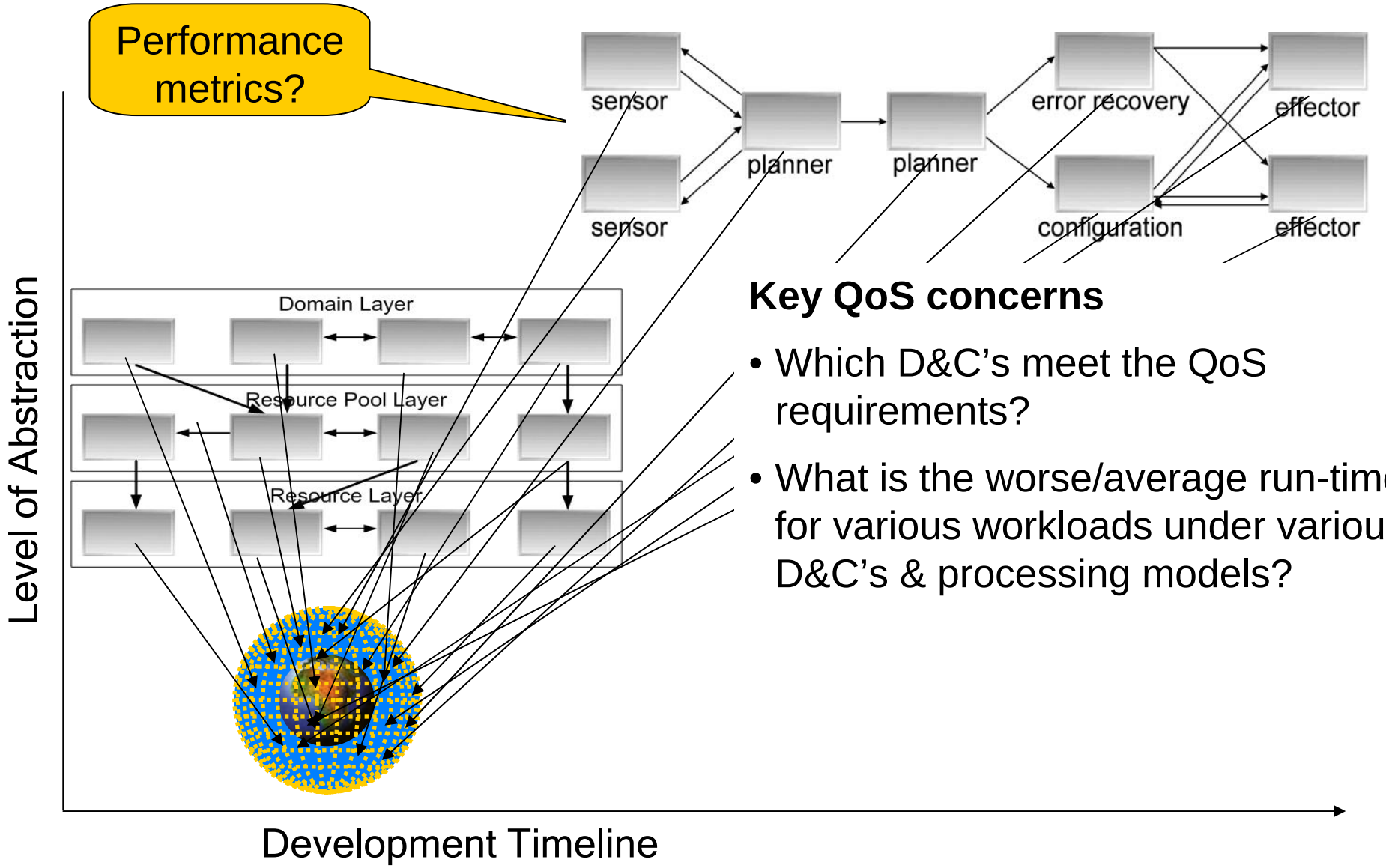
Unresolved QoS Concerns with Serialized Phasing



Key QoS concerns

- Which D&C's meet the QoS requirements?

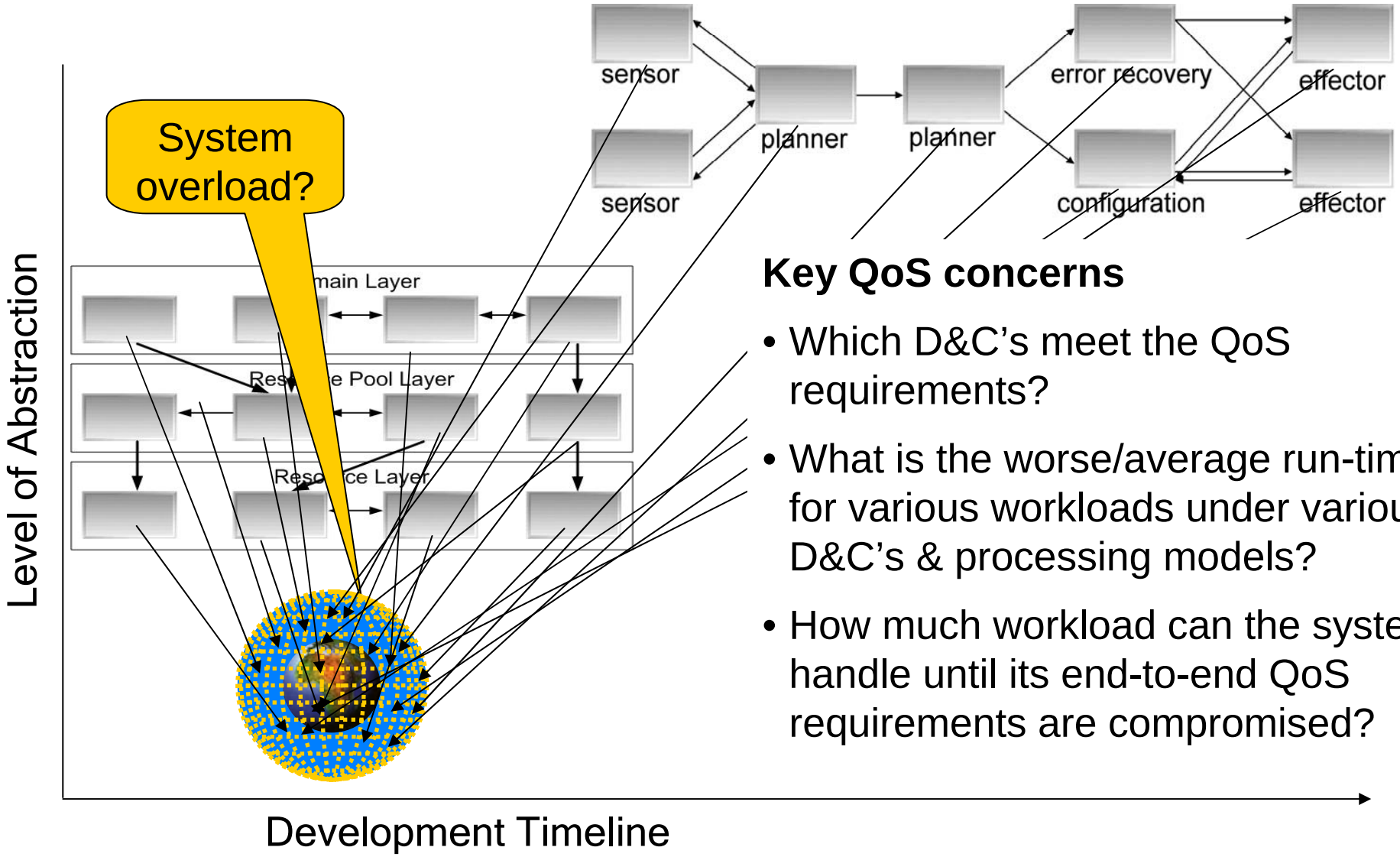
Unresolved QoS Concerns with Serialized Phasing



Key QoS concerns

- Which D&C's meet the QoS requirements?
- What is the worse/average run-time for various workloads under various D&C's & processing models?

Unresolved QoS Concerns with Serialized Phasing

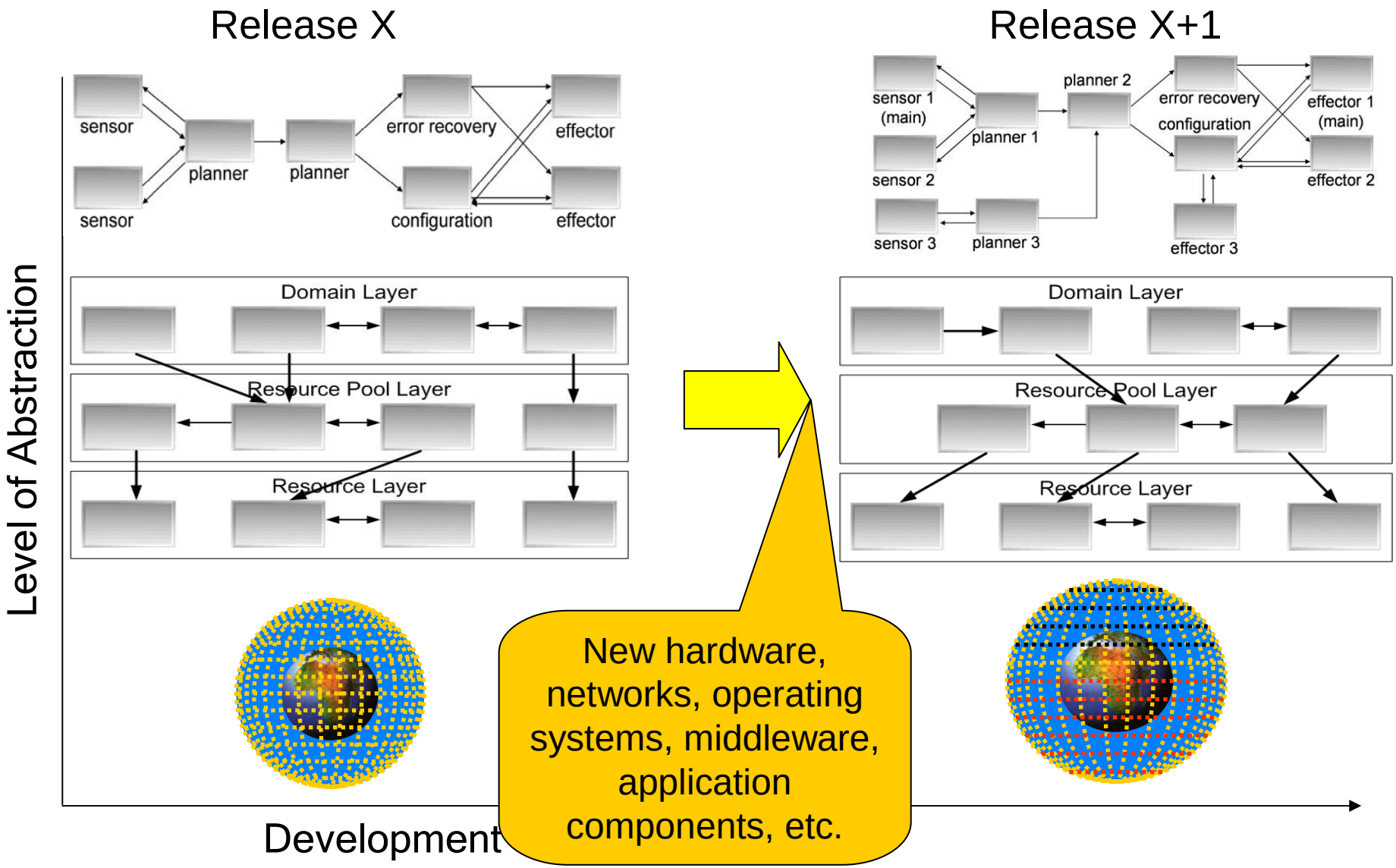


Key QoS concerns

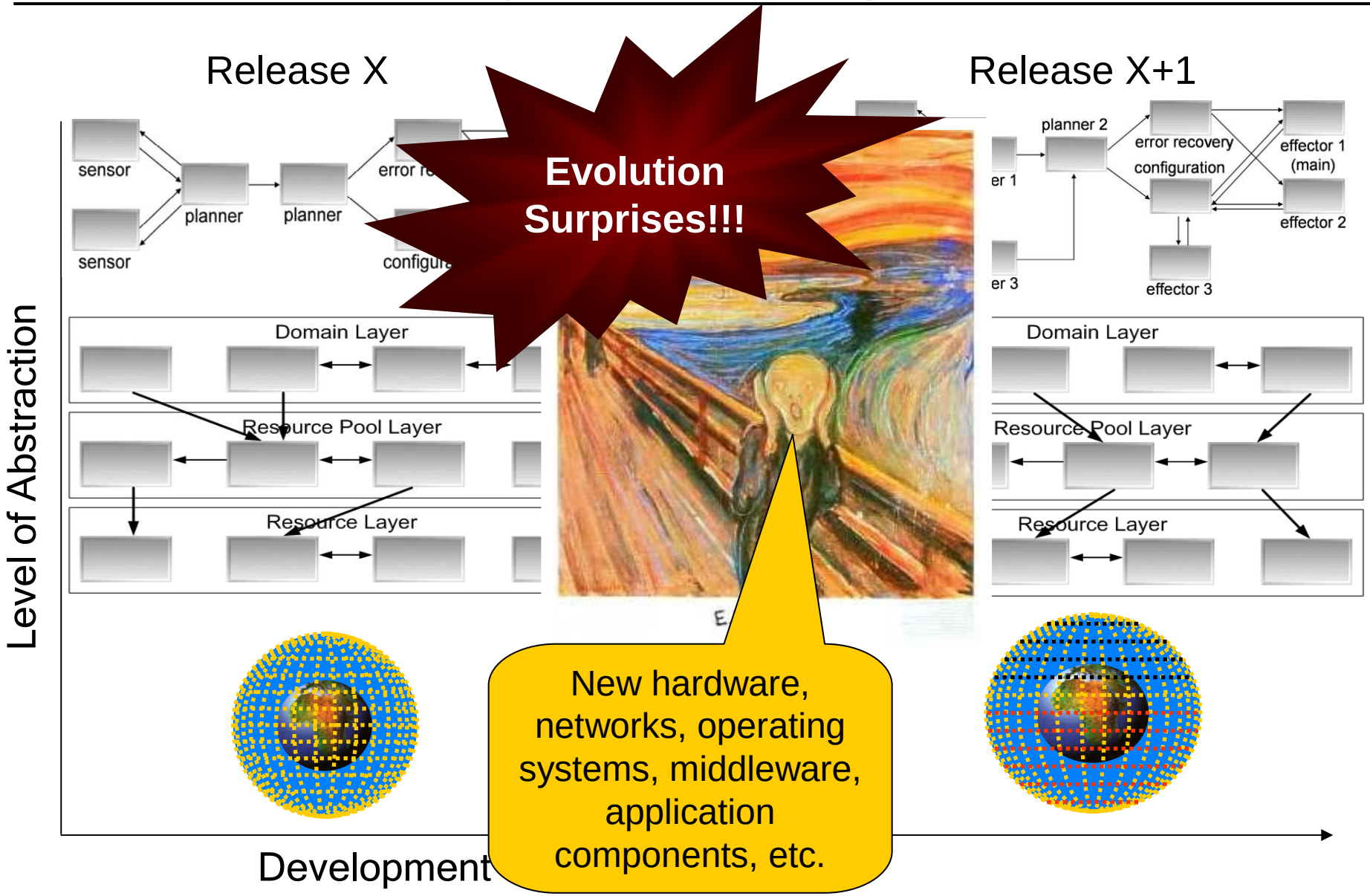
- Which D&C's meet the QoS requirements?
- What is the worse/average run-time for various workloads under various D&C's & processing models?
- How much workload can the system handle until its end-to-end QoS requirements are compromised?

It can take a *long* time (years) to address QoS concerns with serialized phasing

Related ULS System Development Problems



Related ULS System Development Problems



Promising Approach for ULS System Challenges:

System Execution Modeling (SEM) Tools

Tools to express & validate design rules

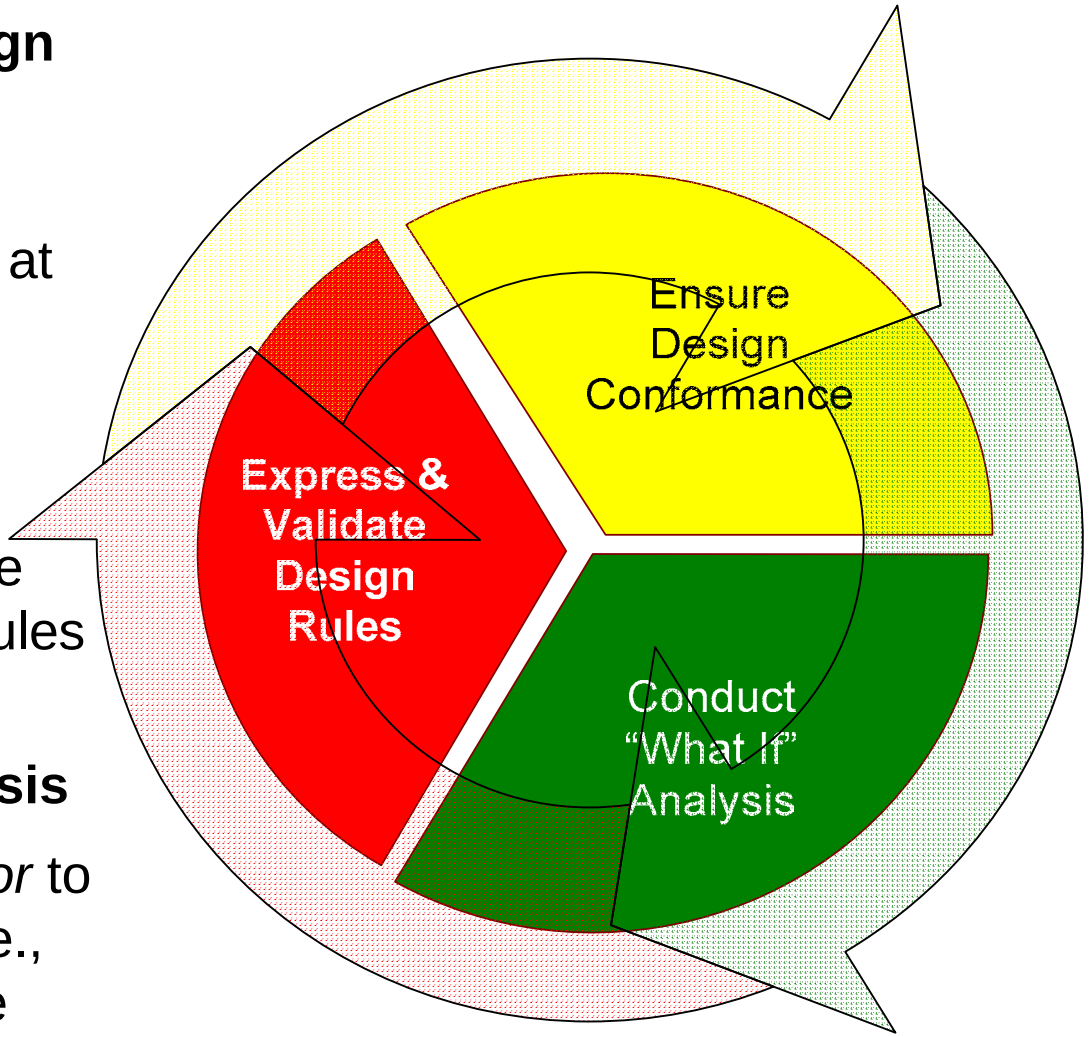
- Help applications & developers adhere to system specifications at design-time

Tools to ensure design rule conformance

- Help properly deploy & configure applications to enforce design rules throughout system lifecycle

Tools to conduct "what if" analysis

- Help analyze QoS concerns *prior* to completing the entire system, i.e., before system integration phase

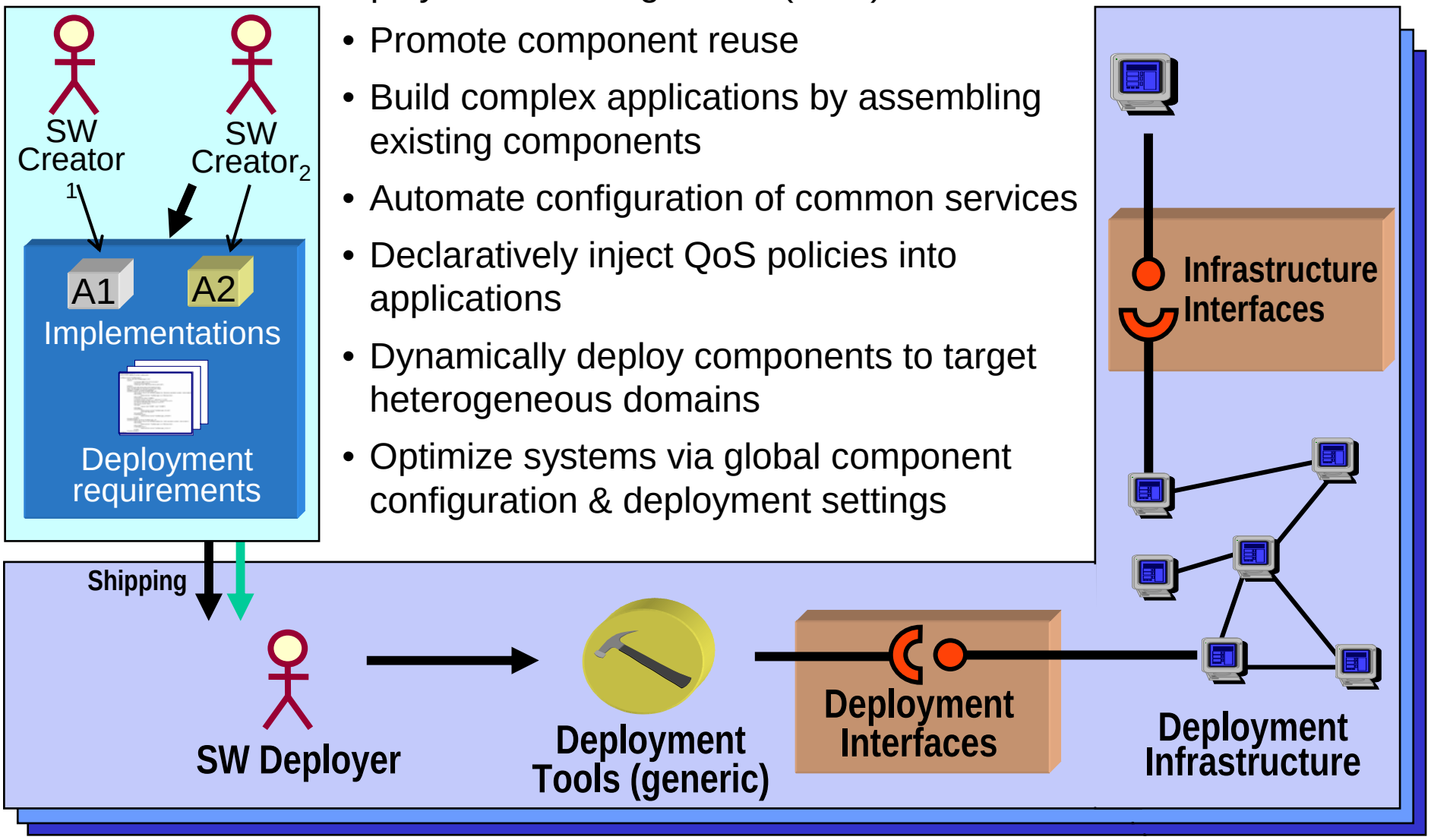


SEM tools should be applied continuously when developing software elements

SEM Tool Example: Component Deployment & Configuration

Deployment & configuration (D&C) Goals

- Promote component reuse
- Build complex applications by assembling existing components
- Automate configuration of common services
- Declaratively inject QoS policies into applications
- Dynamically deploy components to target heterogeneous domains
- Optimize systems via global component configuration & deployment settings



SEM Tool Example: Component Deployment & Configuration

Specification & Implementation

- Defining, partitioning, & implementing app functionality as standalone components

Packaging

- Bundling a suite of software binary modules & metadata representing app components

Installation

- Populating a repository with packages required by app

Configuration

- Configuring packages with appropriate parameters to satisfy functional & systemic requirements of an application without constraining to physical resources

Planning

- Making deployment decisions to identify nodes in target environment where packages will be deployed

Preparation

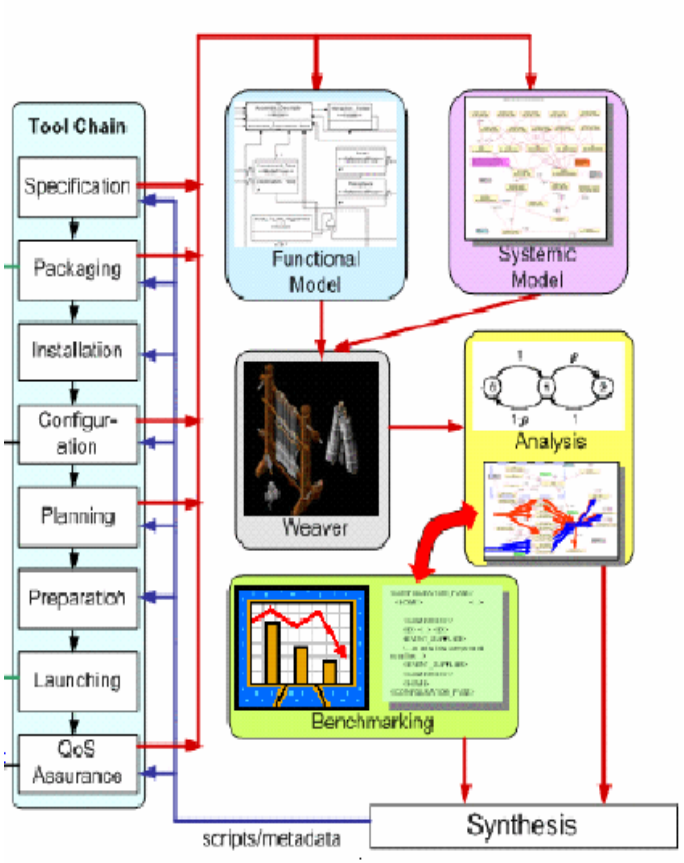
- Moving binaries to identified entities of target environment

Launching

- Triggering installed binaries & bringing app to ready state

QoS Assurance & Adaptation

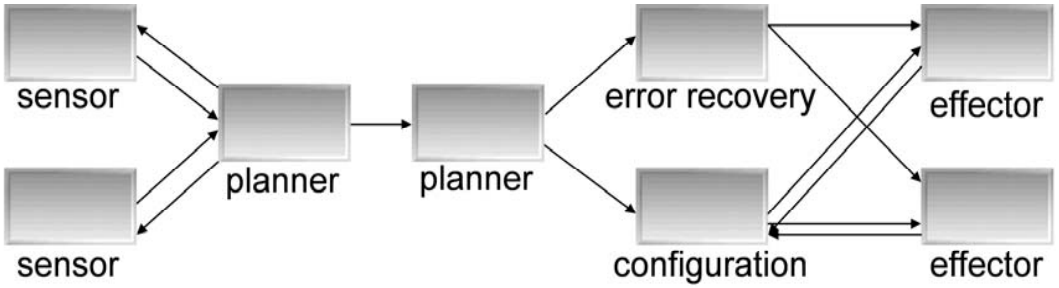
- Runtime (re)configuration & resource management to maintain end-to-end QoS

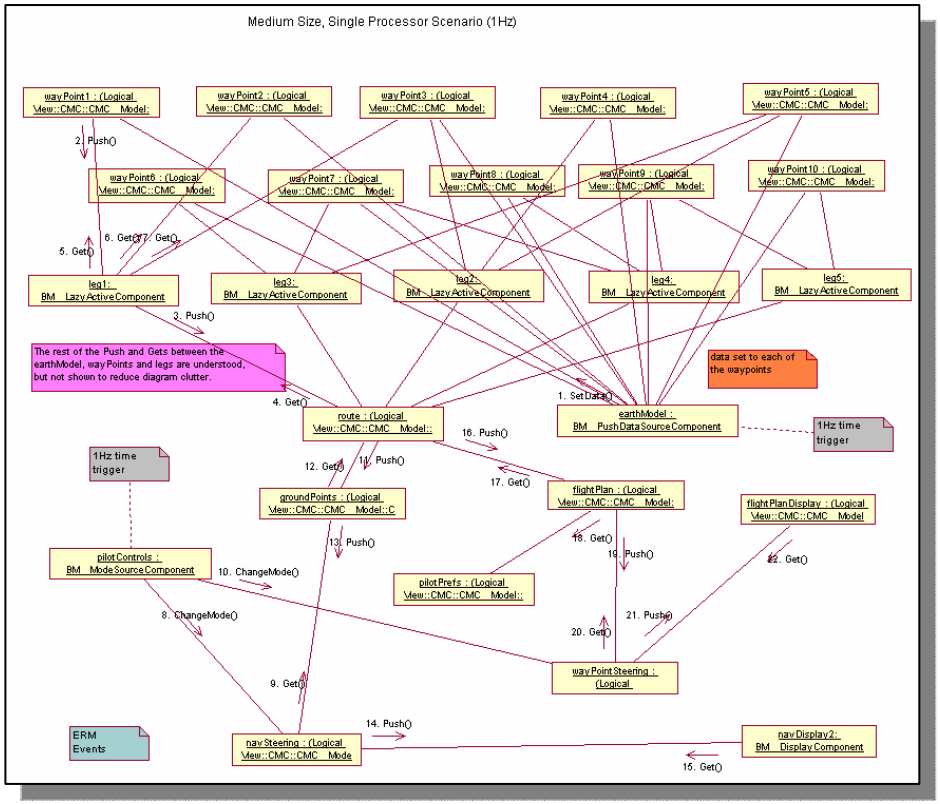


Example D&C specifications include

- OMG Lightweight CORBA Component Model (CCM) &
- IBM Service Component Architecture (SCA)

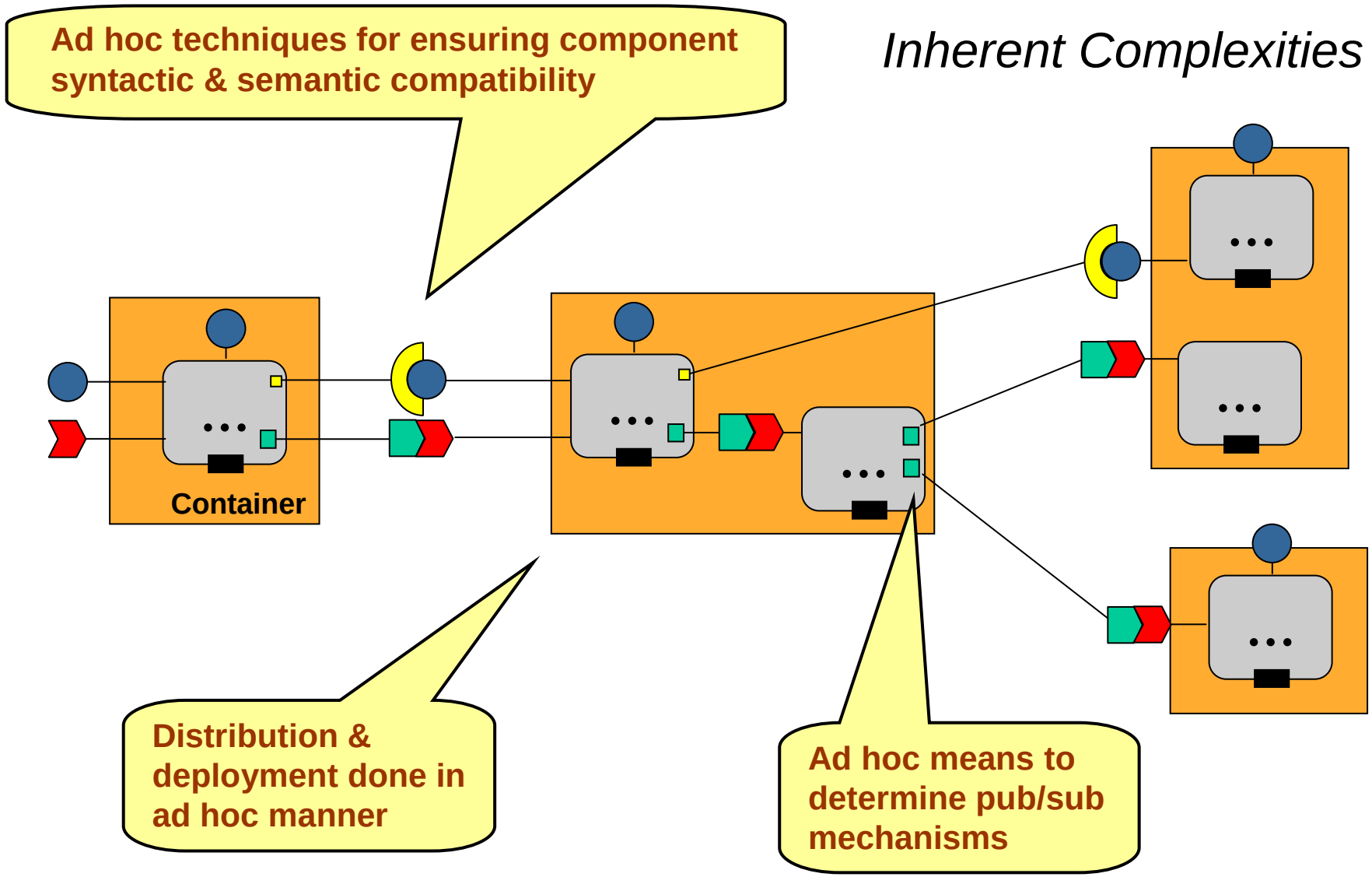
Challenge 1: The Packaging Aspect





- Application components are bundled together into *assemblies*
- Different assemblies tailored to deliver different end-to-end QoS and/or using different algorithms can be part of a package
- ULS systems will require enormous # (10^5-10^7) of components
- Packages describing assemblies can be scripted via XML descriptors

Packaging Aspect Problems (1/2)



Packaging Aspect Problems (2/2)

Accidental Complexities

```
<!– Associate components with impls -->
<assemblyImpl>
   <instance xmi:id="Sensor">
     <name>Sensor Subcomponent</name>
     <package href="Sensor.cpd"/>
   </instance>
   <instance xmi:id="Planner">
     <name>Planner Subcomponent</name>
     <package href="Planner.cpd"/>
   </instance>
   <instance xmi:id="Effector">
     <name>Effector Subcomponent</name>
     <package href="Effector.cpd"/>
   </instance>
 </assemblyImpl>
```

Existing practices involve handcrafting XML descriptors

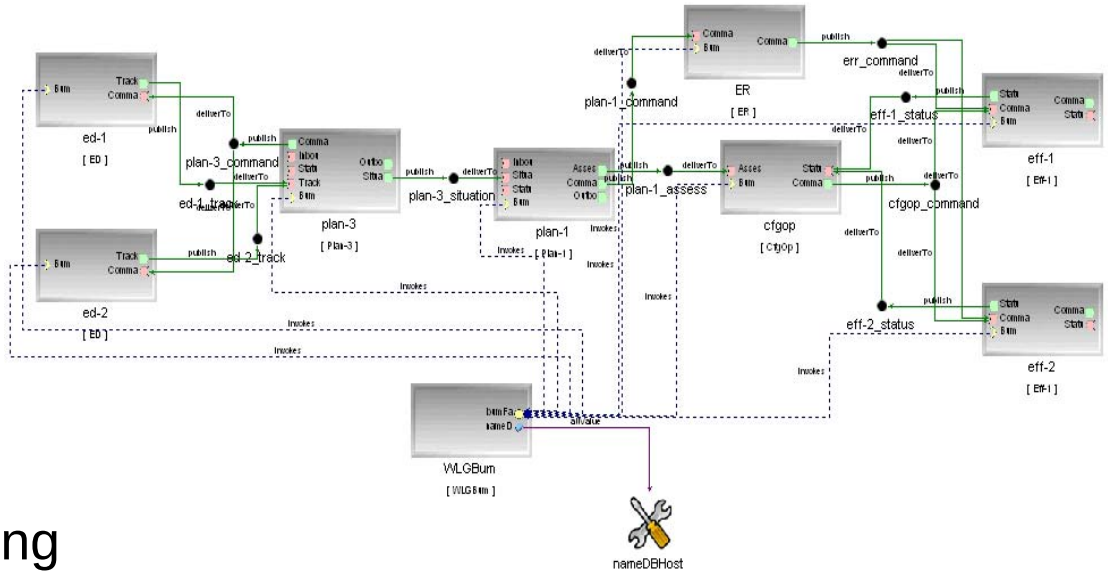
XML file in excess of 3,000 lines, even for medium sized scenarios

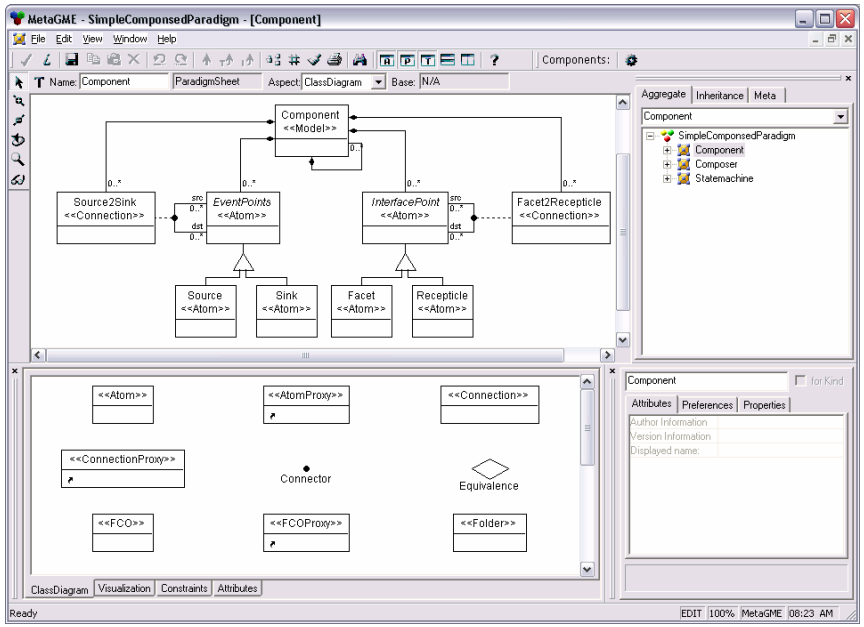
Modifications to the assemblies requires modifying XML file

SEM Tool Approach for Packaging Aspect

Approach:

- Develop the ***Platform-Independent Component Modeling Language*** **(PICML)** to address complexities of assembly packaging
- Capture dependencies visually
- Define semantic constraints using constraints
  - e.g., Object Constraint Language (OCL)
- Generate domain-specific artifacts from models
  - e.g., metadata, code, simulations, etc.
- Uses Generic Modeling Environment (GME) to meta-model & program

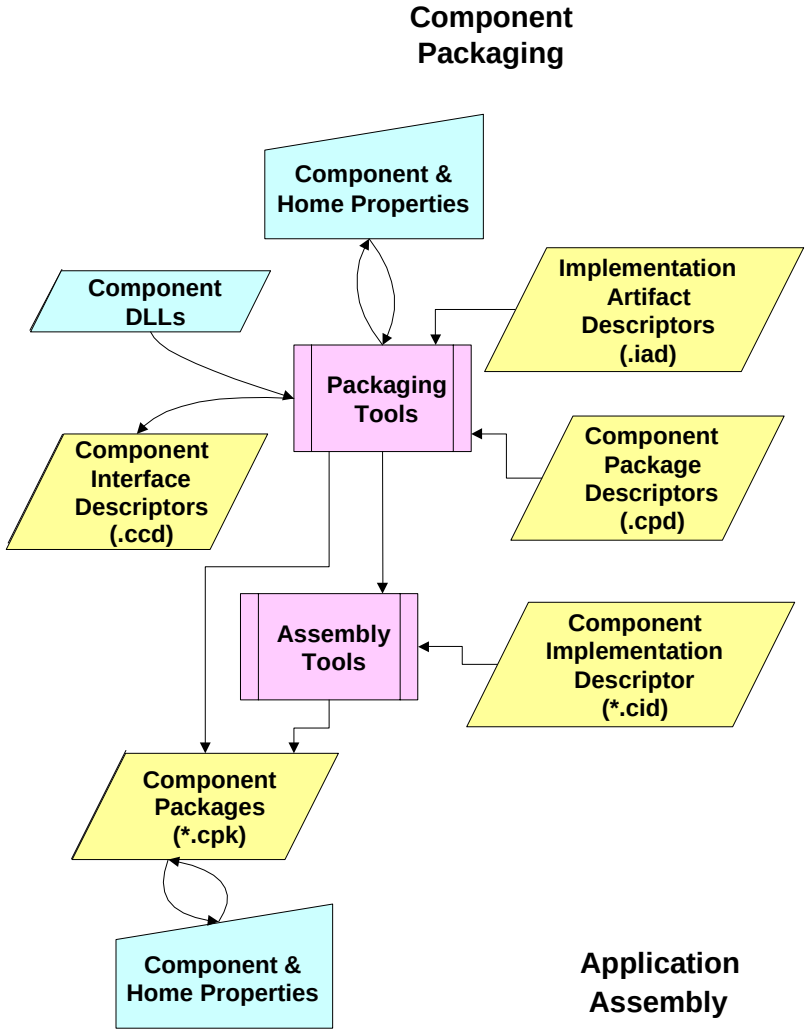




PICML helps to capture & validate design rules for assemblies

Example Metadata Generated by PICML

- **Component Interface Descriptor (.ccd)**
  - Describes the interface, ports, properties of a single component
- **Implementation Artifact Descriptor (.iad)**
  - Describes the implementation artifacts (e.g., DLLs, OS, etc.) of one component
- **Component Package Descriptor (.cpd)**
  - Describes multiple alternative implementations of a single component
- **Package Configuration Descriptor (.pcd)**
  - Describes a configuration of a component package
- **Top-level Package Descriptor (package.tpd)**
  - Describes the top-level component package in a package (.cpk)
- **Component Implementation Descriptor (.cid)**
  - Describes a specific implementation of a component interface
  - Implementation can be either monolithic- or assembly-based
  - Contains sub-component instantiations in case of assembly based implementations
  - Contains inter-connection information between components
- **Component Packages (.cpk)**
  - A component package can contain a single component
  - A component package can also contain an assembly

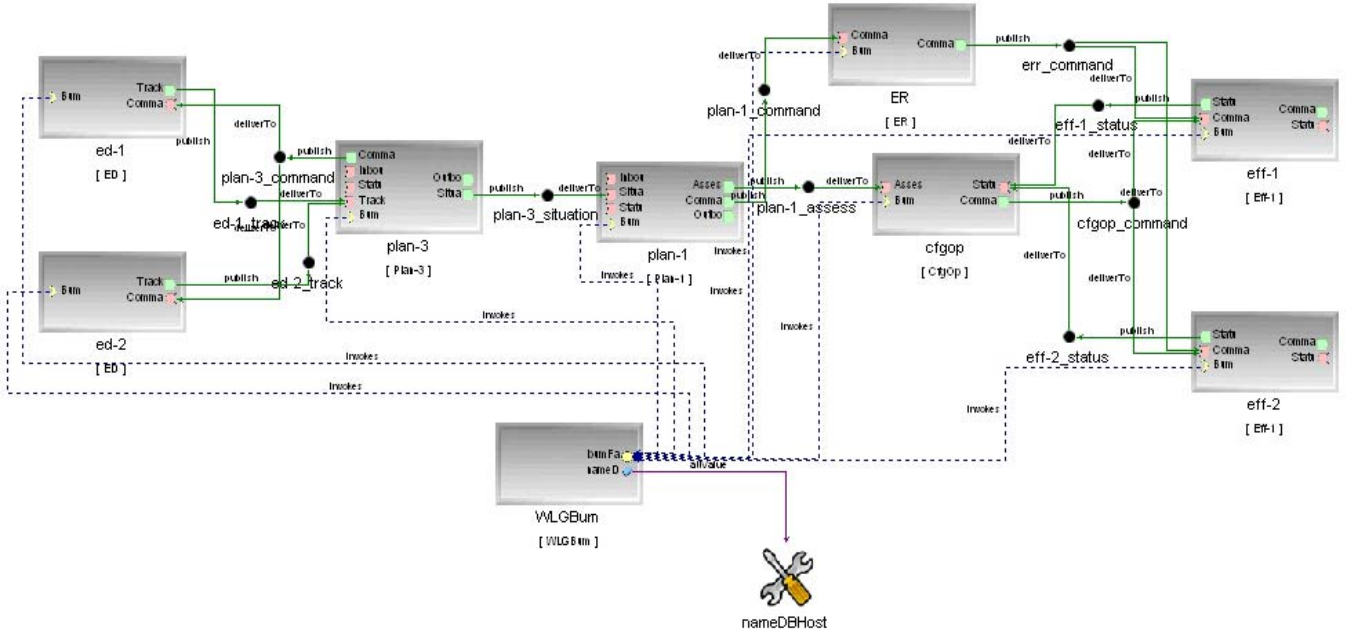


Based on OMG (D&C) specification (ptc/05-01-07)

Example Output from PICML Model

A Component Implementation Descriptor (*.cid) file

- Describes a specific implementation of a component interface
- Describes component interconnections



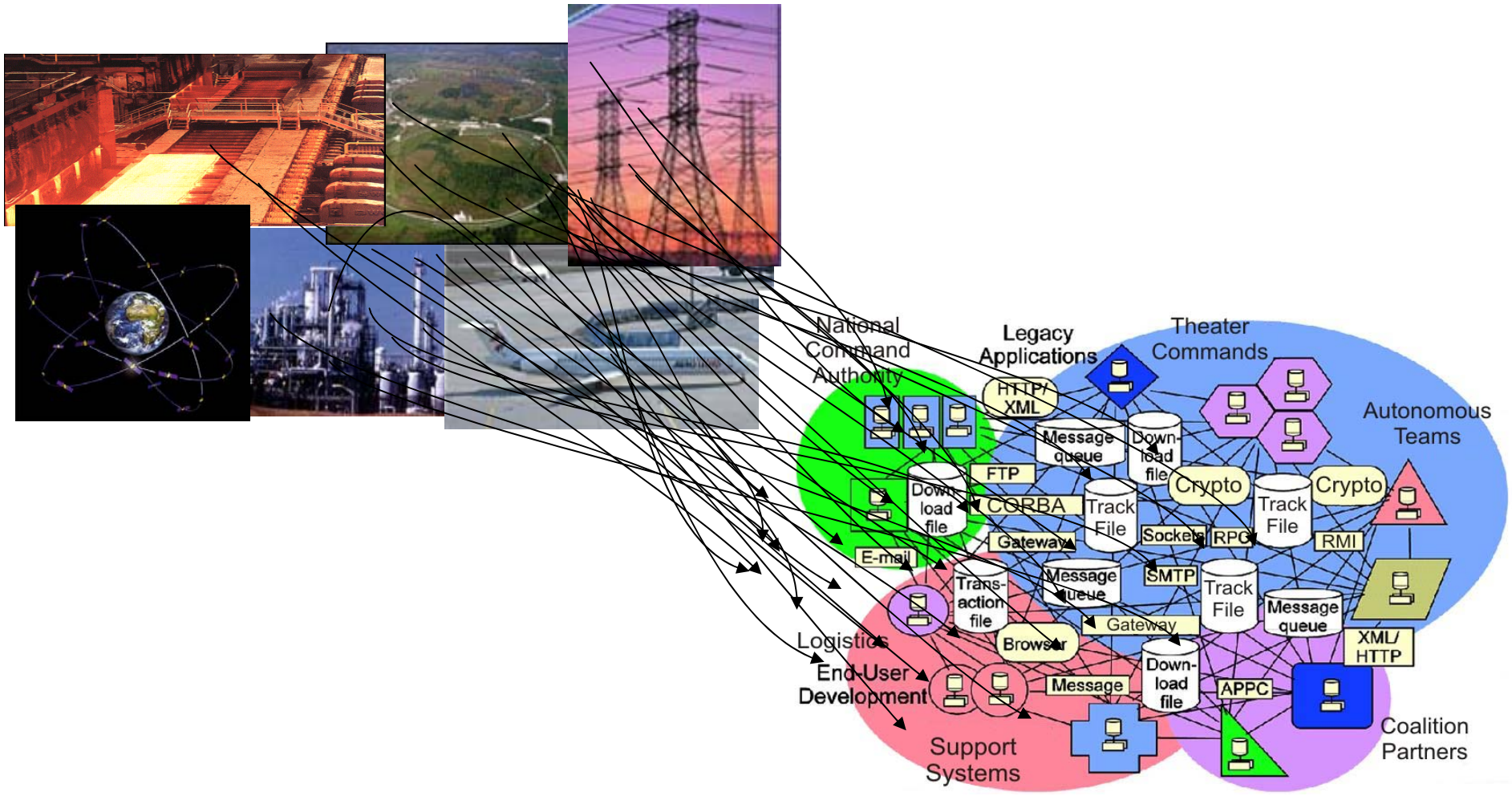

```
<monolithicImpl> [...]
   <deployRequirement>
     <name>Planner</name>
     <resourceType>Planner</resourceType>
     <property><name>vendor</name>
       <value>
         <type> <kind>tk_string</kind> </type>
         <value> <string>My Planner Vendor</string>
       </value>
     </property>
   </deployRequirement> [... Requires VxWorks ...]
 </monolithicImpl>
```

```
<connection> <name>Effector</name>
     <internalEndpoint>
         <portName>Ready</portName>
       <instance href="#Planner"/>
     </internalEndpoint>
     <internalEndpoint>
         <portName>Refresh</portName>
       <instance href="#Effector"/>
     </internalEndpoint>
</connection>
```

PICML supports better expression of domain intent & “correct-by-construction”

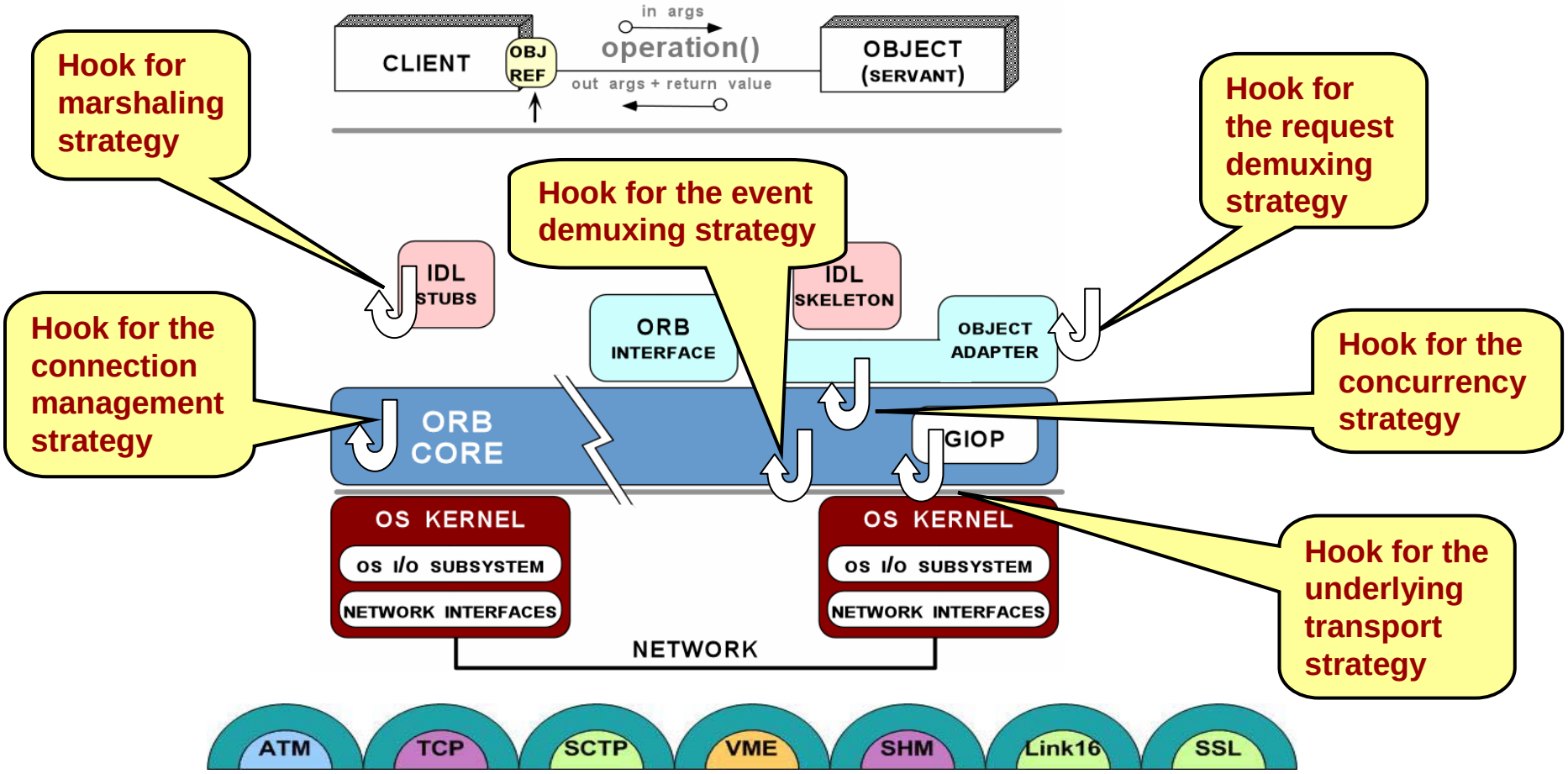
Challenge 2: The Configuration Aspect

ULS systems are characterized by a large *configuration space* that maps known variations in the application requirements space to known variations in the software solution space



Challenge 2: The Configuration Aspect

ULS systems are characterized by a large *configuration space* that maps known variations in the application requirements space to known variations in the software solution space

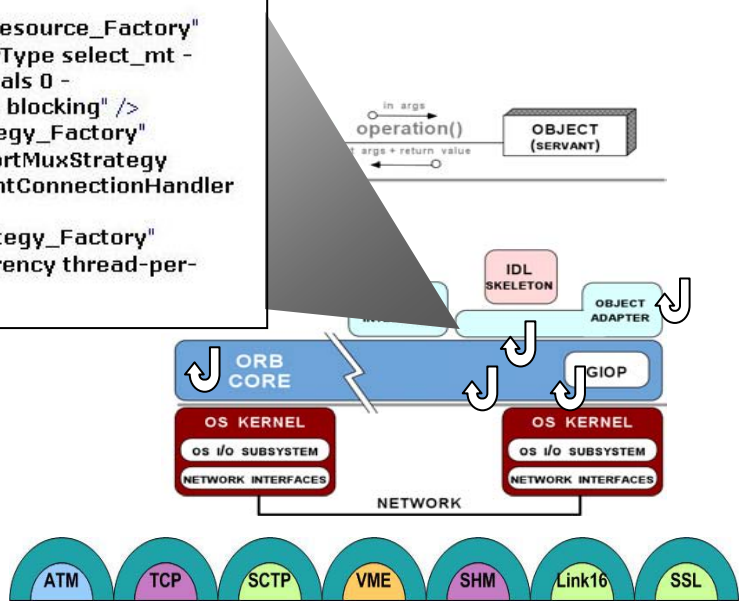


Configuration Aspect Problems

Middleware developers

- Documentation & capability synchronization
- Semantic constraints, design rules, & QoS evaluation of specific configurations

```
ests/Latency/Thread_Per_Connection/svc.conf
ACE_Svc_Conf>
 <!--     -->
 <!--   $Id: svc.conf.xml,v 1.1 2002/08/23
 22:23:04 nanbor Exp $   -->
 <!--     -->
 <static id="Advanced_Resource_Factory"
   params="-ORBReactorType select_mt -
   ORBReactorMaskSignals 0 -
   ORBFlushingStrategy blocking" />
 <static id="Client_Strategy_Factory"
   params="-ORBTransportMuxStrategy
   EXCLUSIVE -ORBClientConnectionHandler
   RW" />
 <static id="Server_Strategy_Factory"
   params="-ORBConcurrency thread-per-
   connection" />
/ACE_Svc_Conf>
```



Application developers

- Must understand middleware constraints, rules, & semantics
  - Increases accidental complexity
- Different middleware uses different configuration mechanisms
- e.g.

Microsoft .net — XML Configuration Files

JAVA — XML Property Files

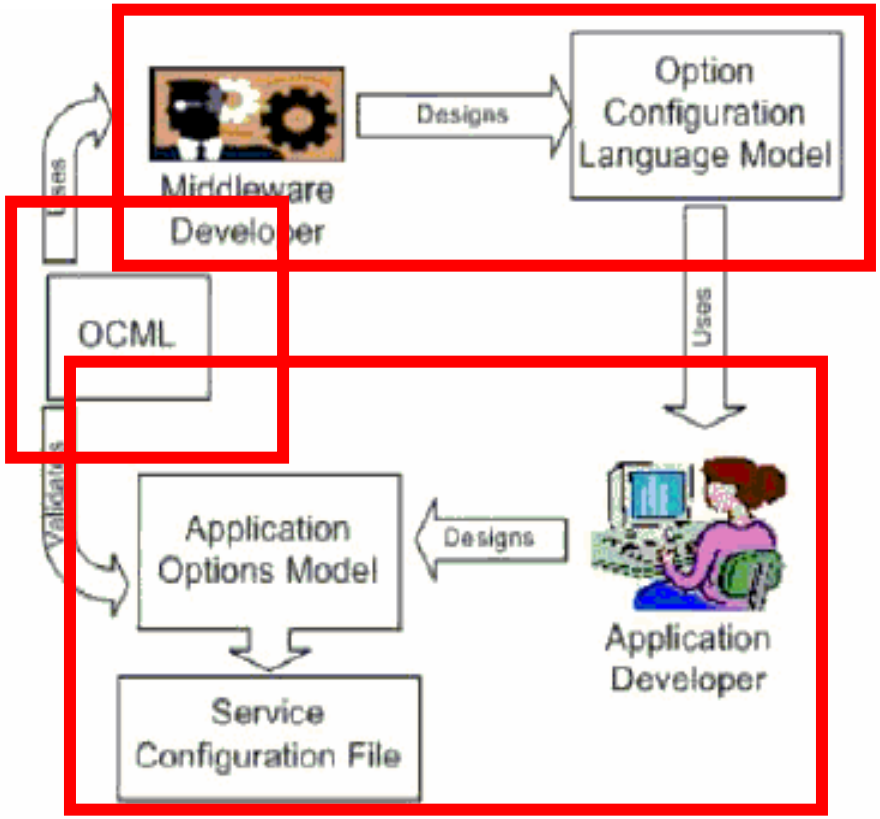
CORBA — CIAO/CCM provides ~500 configuration options

SEM Tool Approach for Configuration Aspect

Approach:

- Develop an ***Options Configuration Modeling Language (OCML)*** to encode design rules & ensure semantic consistency of option configurations

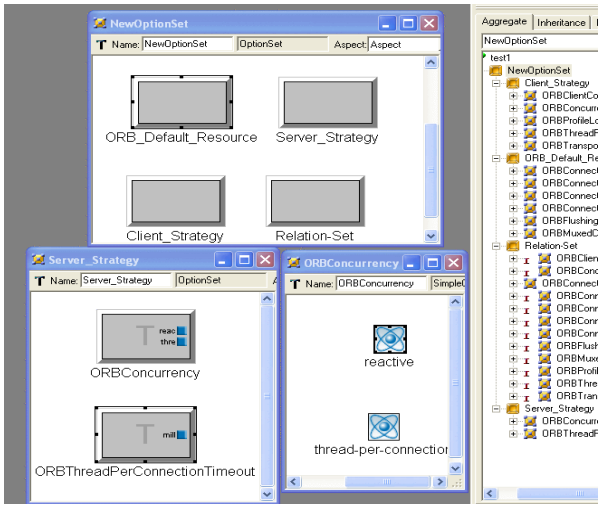
- OCML is used by
  - **Middleware developers** to design the *configuration model*
  - **Application developers** to configure the middleware for a specific application
- OCML *metamodel* is platform-independent
- OCML *models* are platform-specific

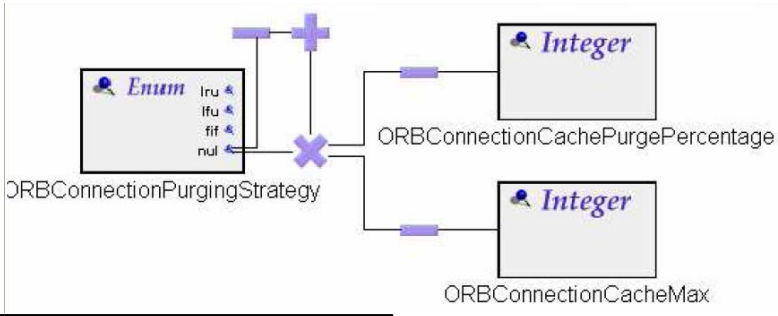


OCML helps to ensure design conformance

Applying OCML to CIAO+TAO

- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model





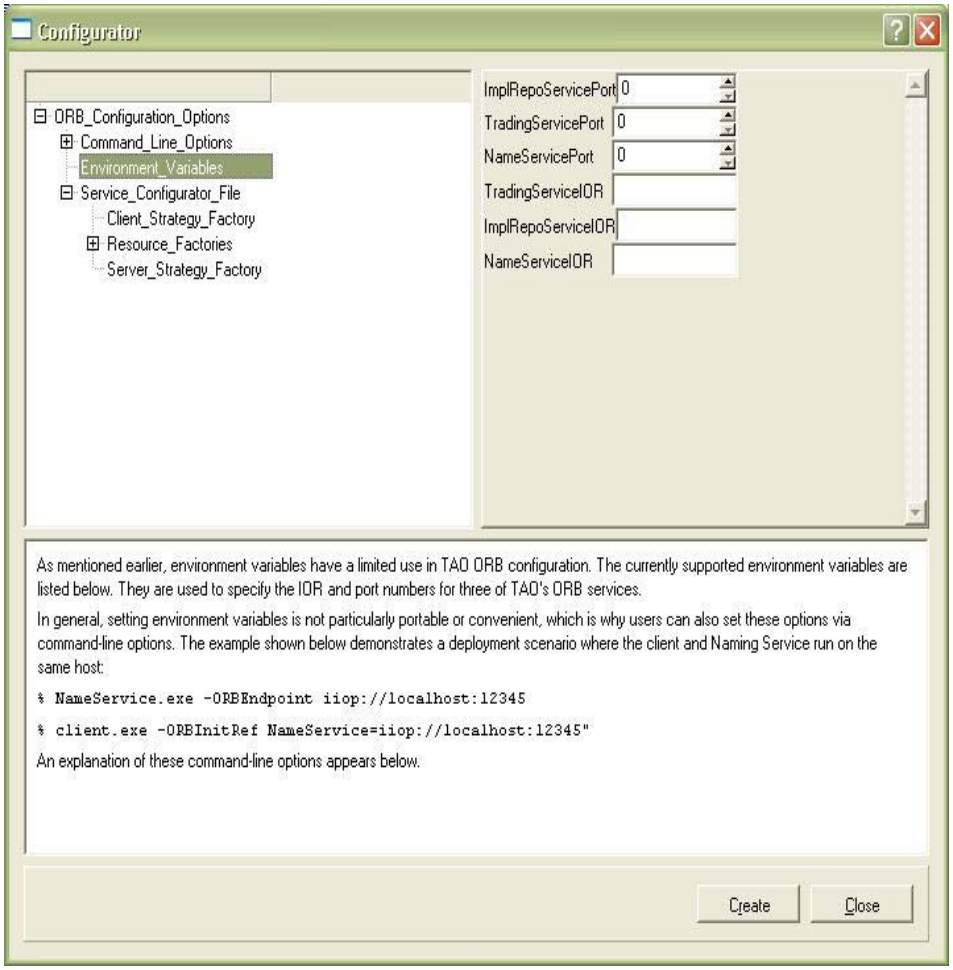

```
<CONFIGURATION>
   <PROCESSOR>
      <NAME>OCP_P1</NAME>
      <CONFIGURATION_PASS>
         <HOME>
            <HOME_TYPE>  BM__OPEN_ED_COMPONENT </HOME_TYPE>
            <ID>
               <GROUP_ID> 20 </GROUP_ID>
               <ITEM_ID>  22 </ITEM_ID>
               <NAME>     BM__OPEN_ED_COMPONENT  </NAME>
            </ID>
            <COMPONENT>
               <ID>
                  <GROUP_ID>200</GROUP_ID>
                  <ITEM_ID>220</ITEM_ID>
                  <NAME>WAYPOINT_PROXY</NAME>
               </ID>
               <DISTRIBUTION>
                  <DIST_WRITE>
                     <PROXY>
                        <ID>
                           <GROUP_ID>200</GROUP_ID>
                           <ITEM_ID>221</ITEM_ID>
                           <NAME>WAYPOINT</NAME>
                        </ID>
                     </PROXY>
                  </DIST_WRITE>
               </DISTRIBUTION>
               <EVENT_SUPPLIER>
                  <EVENT_SET>
                     <EVENT>
```

```
/**
 * Return the last time the client sent a request associated
 * session, as the number of ms since midnight, Jan 1, 1970
 * GMT.  Actions your application takes, such as get or set
 * value associated with session, do not affect access time.
 */
public long getLastAccessedTime() {
  return (this.lastAccessedTime);
}

/**
 * Update the accessed time information for this session.
 * Method is called by context when request comes in for a
 * session, even if the application does not reference it.
 */
public void access() {
   this.lastAccessedTime = this.thisAccessedTime;
}
```

Applying OCML to CIAO+TAO

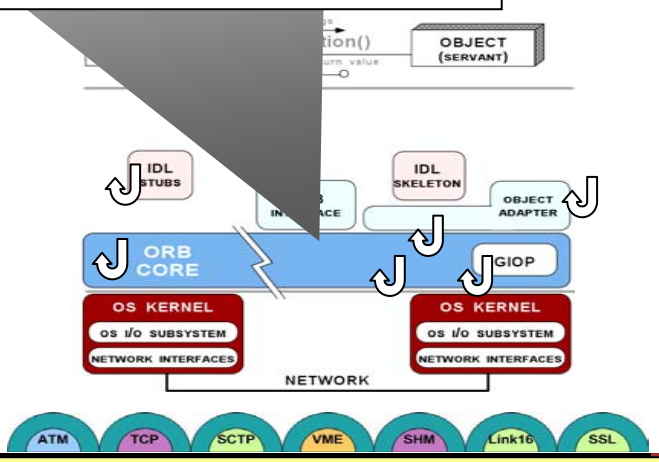
- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model
- Application developers provide a model of desired options & their values, e.g.,
  - Network resources
  - Concurrency & connection management strategies



Applying OCML to CIAO+TAO

- Middleware developers specify
  - Configuration space
  - Constraints
- OCML generates config model
- Application developers provide a model of desired options & their values, e.g.,
  - Network resources
  - Concurrency & connection management strategies
- OCML constraint checker flags incompatible options & then
  - Synthesizes XML descriptors for middleware configuration
  - Generates documentation for middleware configuration
  - Validates the configurations

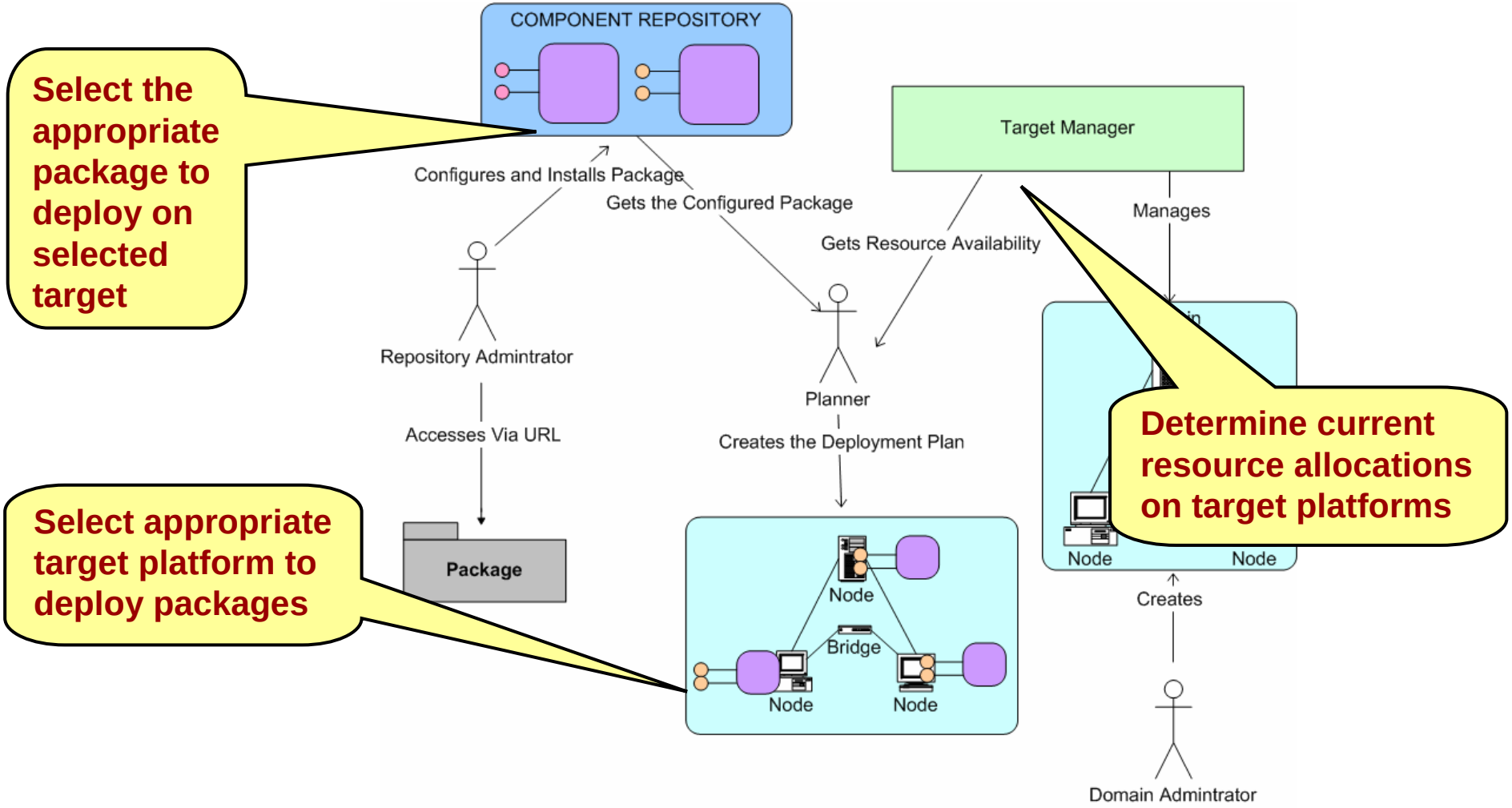
```
ests/Latency/Thread_Per_Connection/svc.conf
ACE_Svc_Conf>
 <!--     -->
 <!--   $Id: svc.conf.xml,v 1.1 2002/08/23
 22:23:04 nanbor Exp $   -->
 <!--     -->
 <static id="Advanced_Resource_Factory"
   params="-ORBReactorType select_mt -
   ORBReactorMaskSignals 0 -
   ORBFlushingStrategy blocking" />
 <static id="Client_Strategy_Factory"
   params="-ORBTransportMuxStrategy
   EXCLUSIVE -ORBClientConnectionHandler
   RW" />
 <static id="Server_Strategy_Factory"
   params="-ORBConcurrency thread-per-
   connection" />
/ACE_Svc_Conf>
```



OCML automates activities that are very tedious & error-prone to do manually

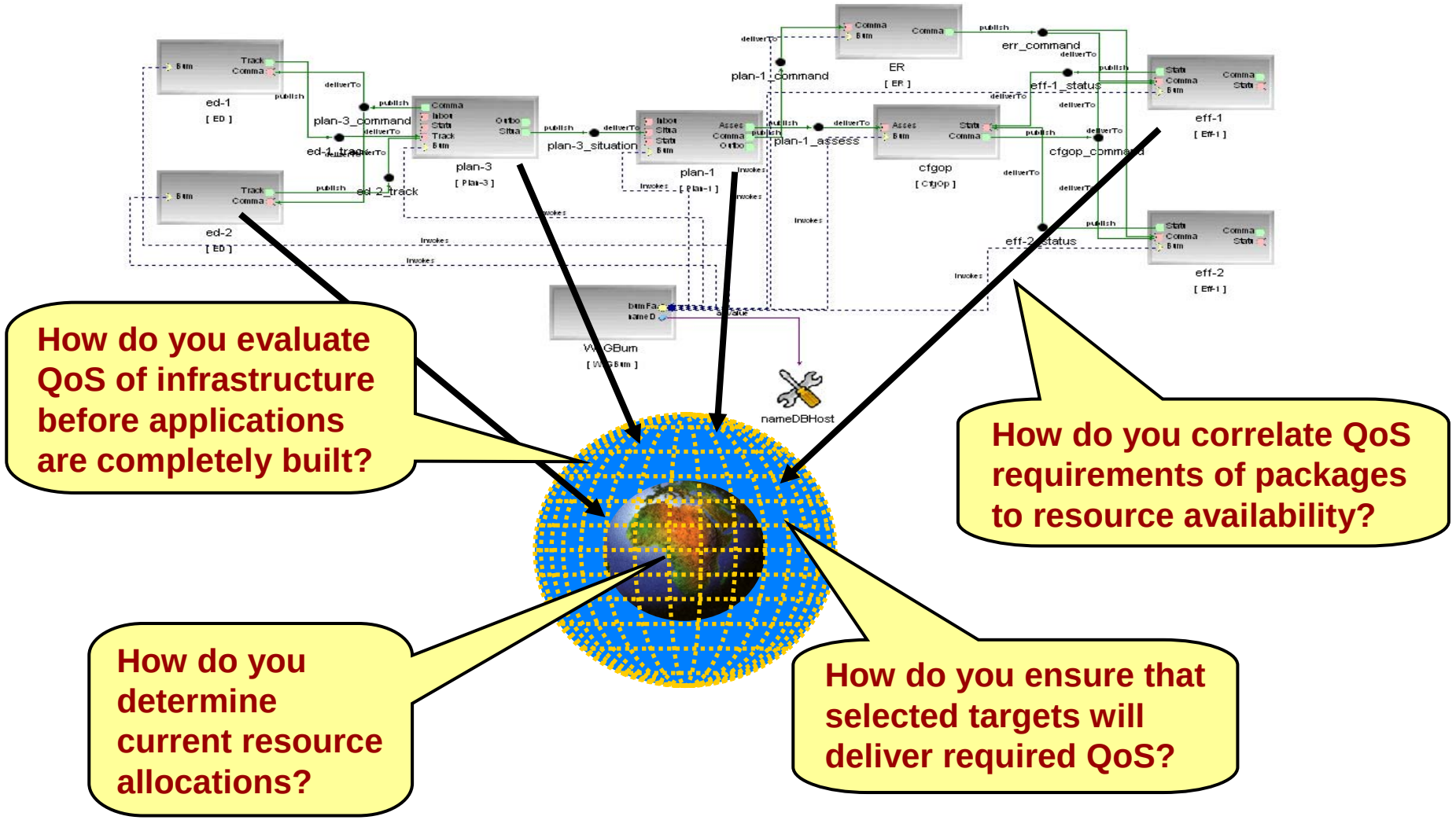
Challenge 3: Planning Aspect

System integrators must make appropriate deployment decisions, identifying nodes in target environment where packages will be deployed



Planning Aspect Problems

Ensuring deployment plans meet ULS system QoS requirements

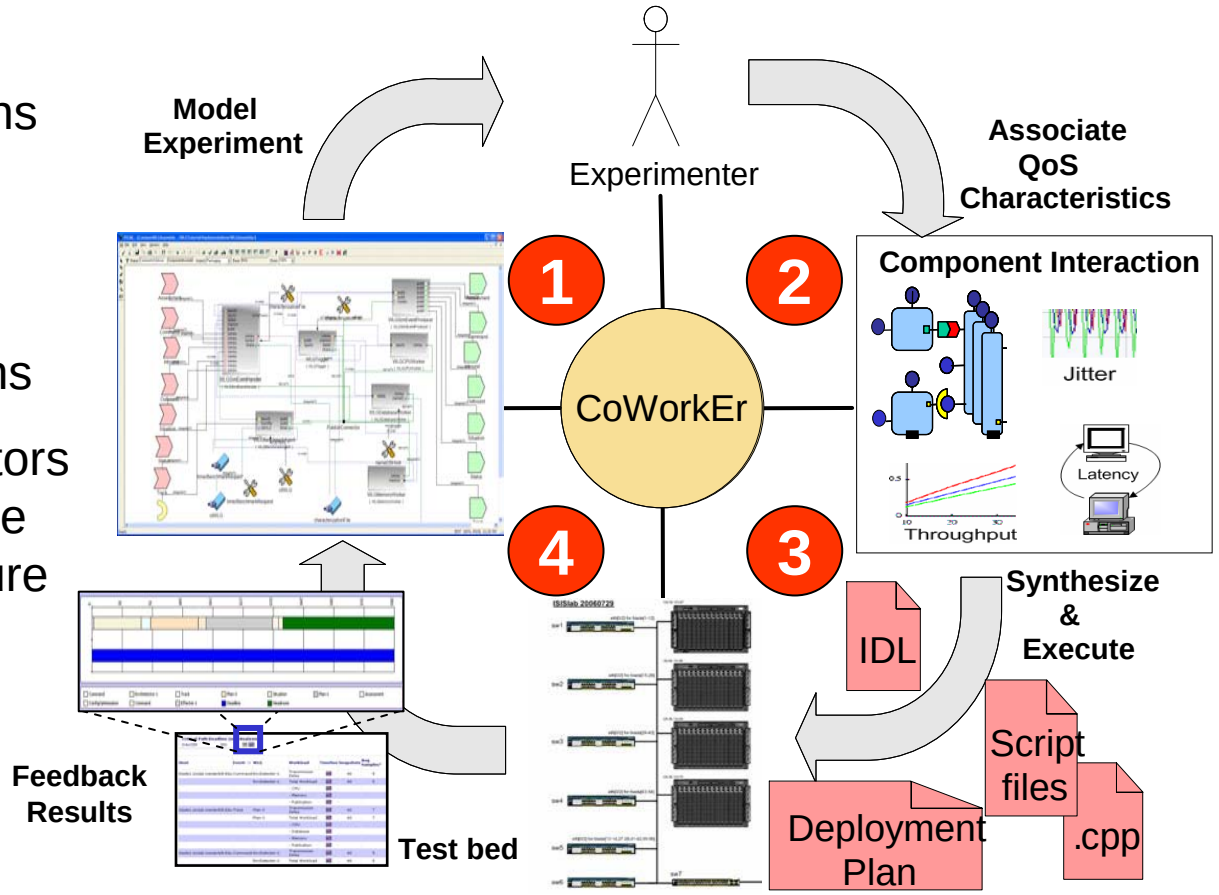


SEM Tool Approach for Planning Aspect

Approach

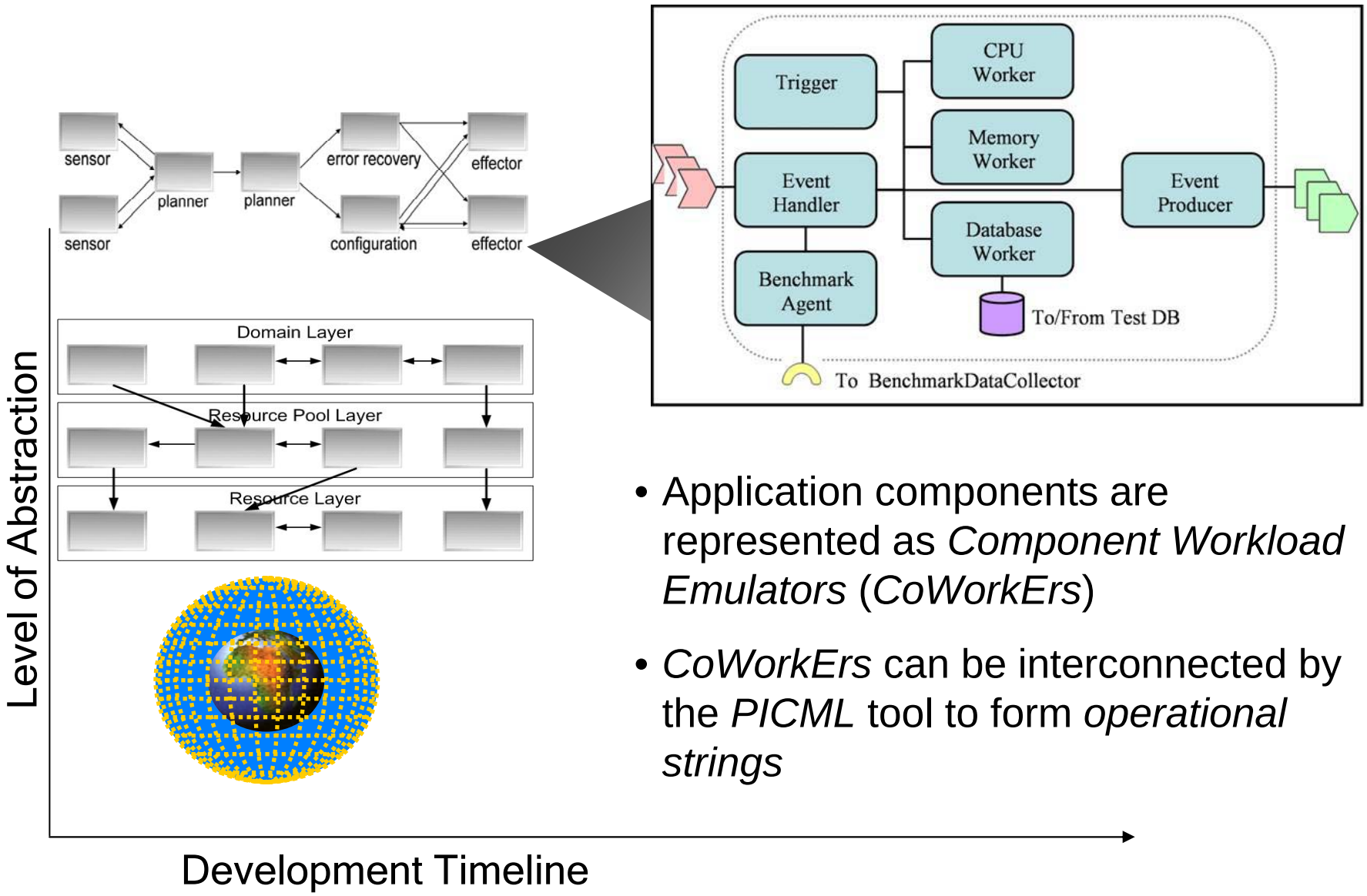
- Develop ***Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)*** to allow architects & systems engineers to

1. Compose scenarios to exercise critical system paths
2. Associate performance properties with scenarios & assign properties to components specific to paths
3. Configure workload generators to run experiments, generate deployment plans, & measure performance along critical paths
4. Analyze results to verify if deployment plan & configurations meet performance requirements



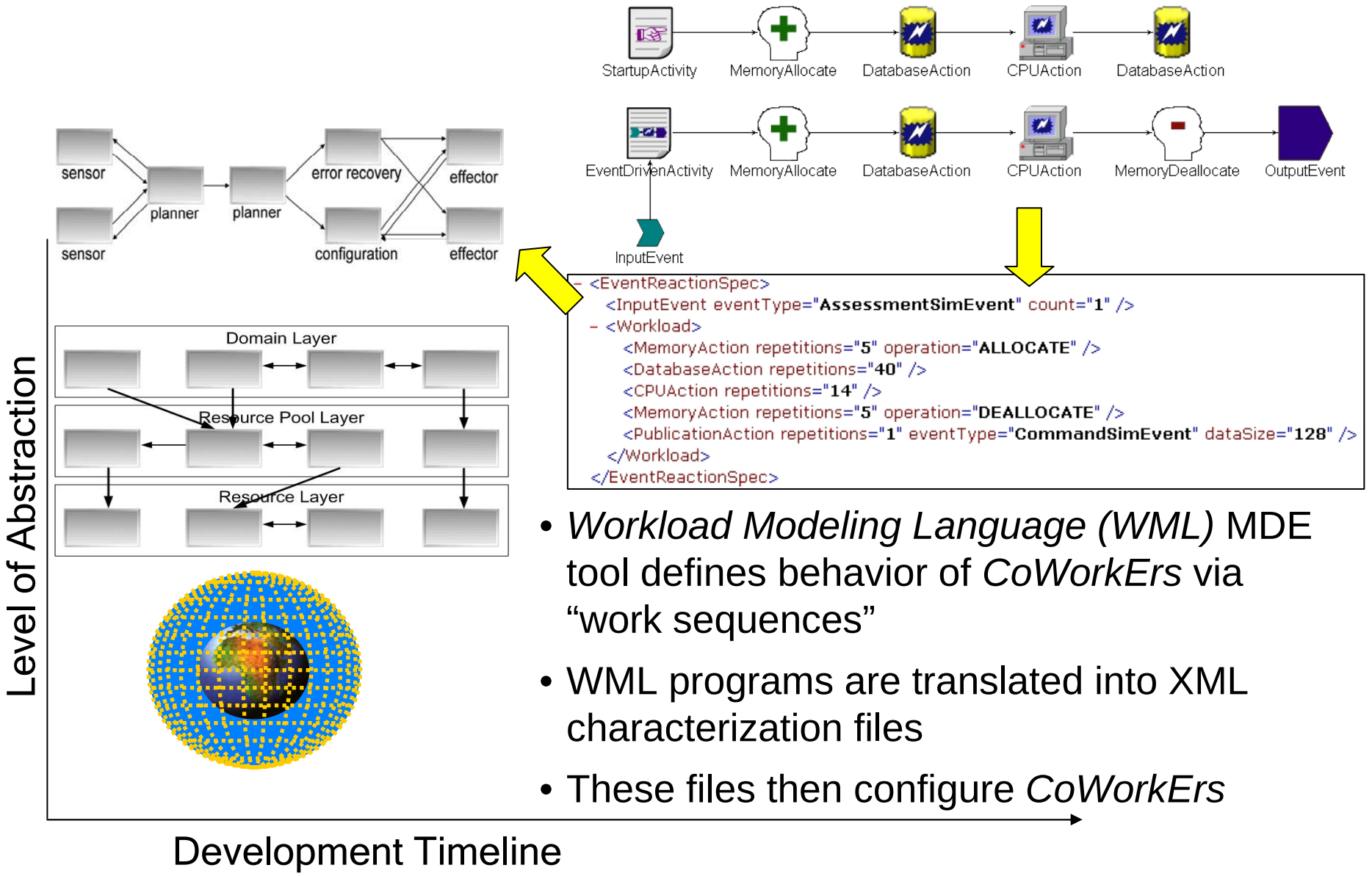
CUTS helps to conduct "what if" analysis on evolving systems

Emulating Computational Components in CUTS



- Application components are represented as *Component Workload Emulators* (*CoWorkErs*)
- *CoWorkErs* can be interconnected by the *PICML* tool to form *operational strings*

Representing Computational Components in CUTS

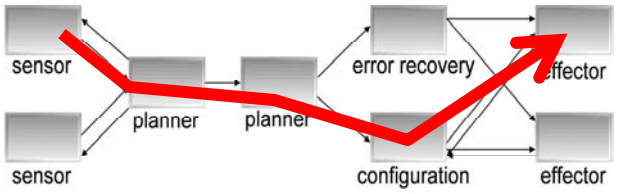


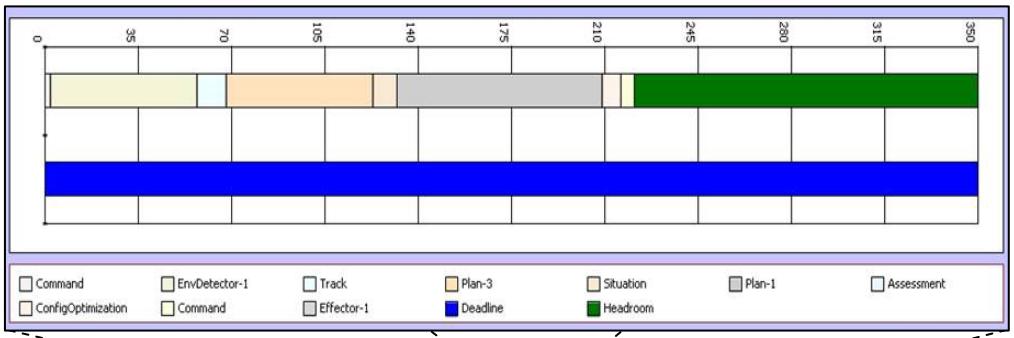

```
- <EventReactionSpec>
    <InputEvent eventType="AssessmentSimEvent" count="1" />
  - <Workload>
      <MemoryAction repetitions="5" operation="ALLOCATE" />
      <DatabaseAction repetitions="40" />
      <CPUAction repetitions="14" />
      <MemoryAction repetitions="5" operation="DEALLOCATE" />
      <PublicationAction repetitions="1" eventType="CommandSimEvent" dataSize="128" />
    </Workload>
  </EventReactionSpec>
```

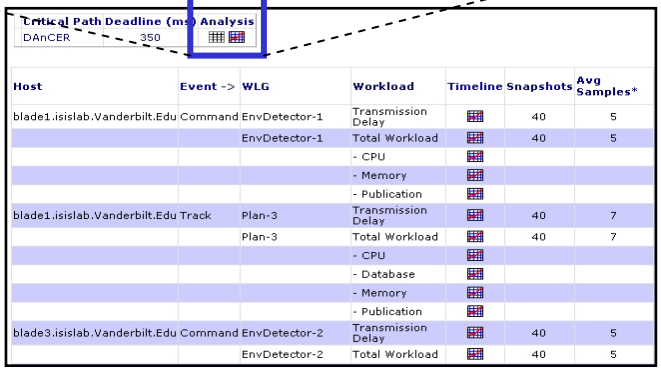
- *Workload Modeling Language (WML)* MDE tool defines behavior of *CoWorkErs* via "work sequences"
- WML programs are translated into XML characterization files
- These files then configure *CoWorkErs*

www.cs.wustl.edu/~schmidt/PDF/QoSPML-WML.pdf

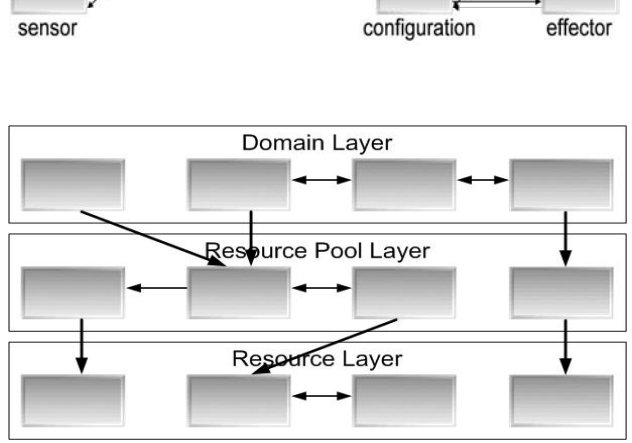
Visualizing Critical Path Performance in CUTS

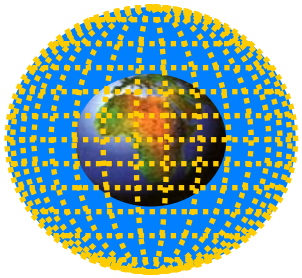






Host	Event ->	WLG	Workload	Timeline	Snapshots	Avg Samples*
blade1.isislab.Vanderbilt.Edu	Command	EnvDetector-1	Transmission Delay		40	5
		EnvDetector-1	Total Workload		40	5
			- CPU			
			- Memory			
			- Publication			
blade1.isislab.Vanderbilt.Edu	Track	Plan-3	Transmission Delay		40	7
		Plan-3	Total Workload		40	7
			- CPU			
			- Database			
			- Memory			
			- Publication			
blade3.isislab.Vanderbilt.Edu	Command	EnvDetector-2	Transmission Delay		40	5
		EnvDetector-2	Total Workload		40	5





Level of Abstraction

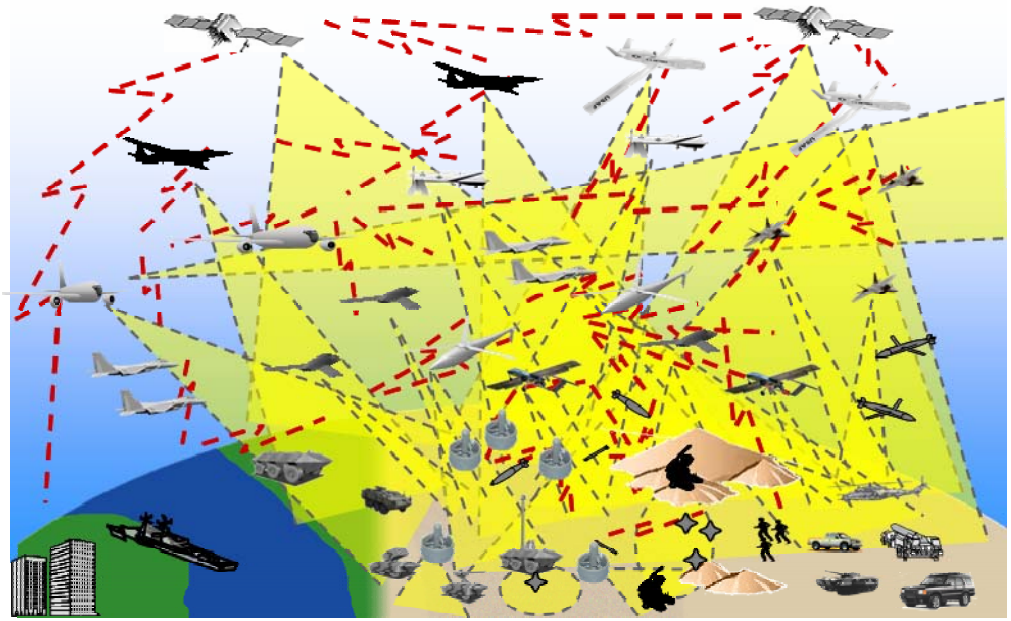
Development Timeline

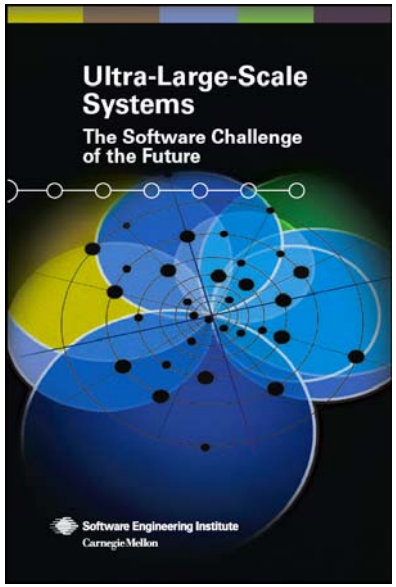
- *BenchmarkManagerWeb-interface (BMW)* MDE tool generates statistics showing performance of actions in each *CoWorkEr*
- Critical paths show end-to-end performance of mission-critical operational strings

CUTS integrates nicely with *continuous integration servers*

Concluding Remarks

- The emergence of ULS systems requires significant innovations & advances in tools & platforms
- Not all technologies provide the precision we're accustomed to in legacy real-time systems
- Advances in Model-driven engineering (MDE) are needed to address ULS systems challenges
- Significant MDE groundwork laid in recent DARPA programs





- Much more R&D needed for ULS systems
  - e.g., recent Software Engineering Institute study

ULS systems report available at www.sei.cmu.edu/uls