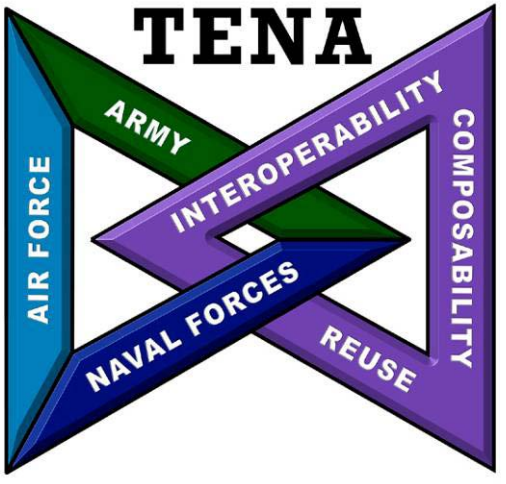


TENA Technical Introduction Course

Dr. Edward T. Powell

TENA Architect

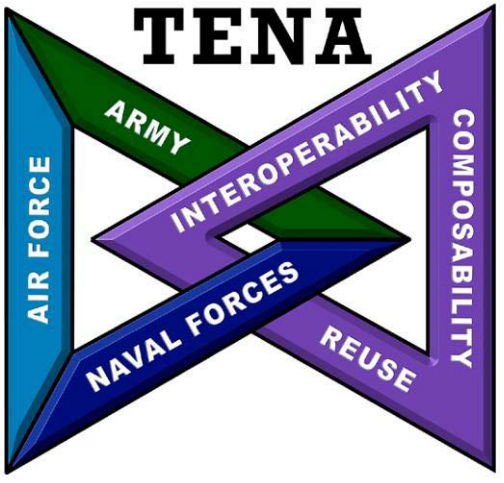
Friday, February 16, 2007

Schedule

5 Minutes	Introduction
10 Minutes	TENA Mission and Organization
15 Minutes	Some Uses of TENA
45 Minutes	TENA Architecture Overview
45 Minutes	Stateful Distributed Objects, the TENA Meta-Model, and the TENA Object Model
	Break
15 Minutes	TENA Integrated Development Environment
30 Minutes	TENA Definition Language
30 Minutes	Patterns Used in the Middleware API
	Break
2 Hours	TENA Middleware Release 5 API
15 Minutes	Where To Go From Here

Goals of the Course

- **Attendees**
  - Anyone who wants to know more about TENA, the TENA Middleware, or how to create interoperable "logical ranges"
  - Anyone who will be testing or using the TENA Middleware
- **Goals**
  - Provide an overview of TENA concepts and components
  - Provide users with information on the TENA Middleware software
- **What you should get out of this course**
  - Understand what TENA is
  - Understand how to create your Logical Range Object Model (LROM)
  - Understand what functionality is being provided by the TENA Middleware
  - Understand the key programming issues needed to use the TENA Middleware



TENA Mission and Organization

TENA Mission

Currently, range systems tend to be non-interoperable, "stove-pipe" systems

The purpose of TENA is to provide the architecture and the software implementation necessary to:

- **Enable Interoperability among Range systems, Facilities, Simulations, C4ISR systems in a quick, cost-efficient manner**
- **Foster Reuse for Range asset utilization and for future developments**

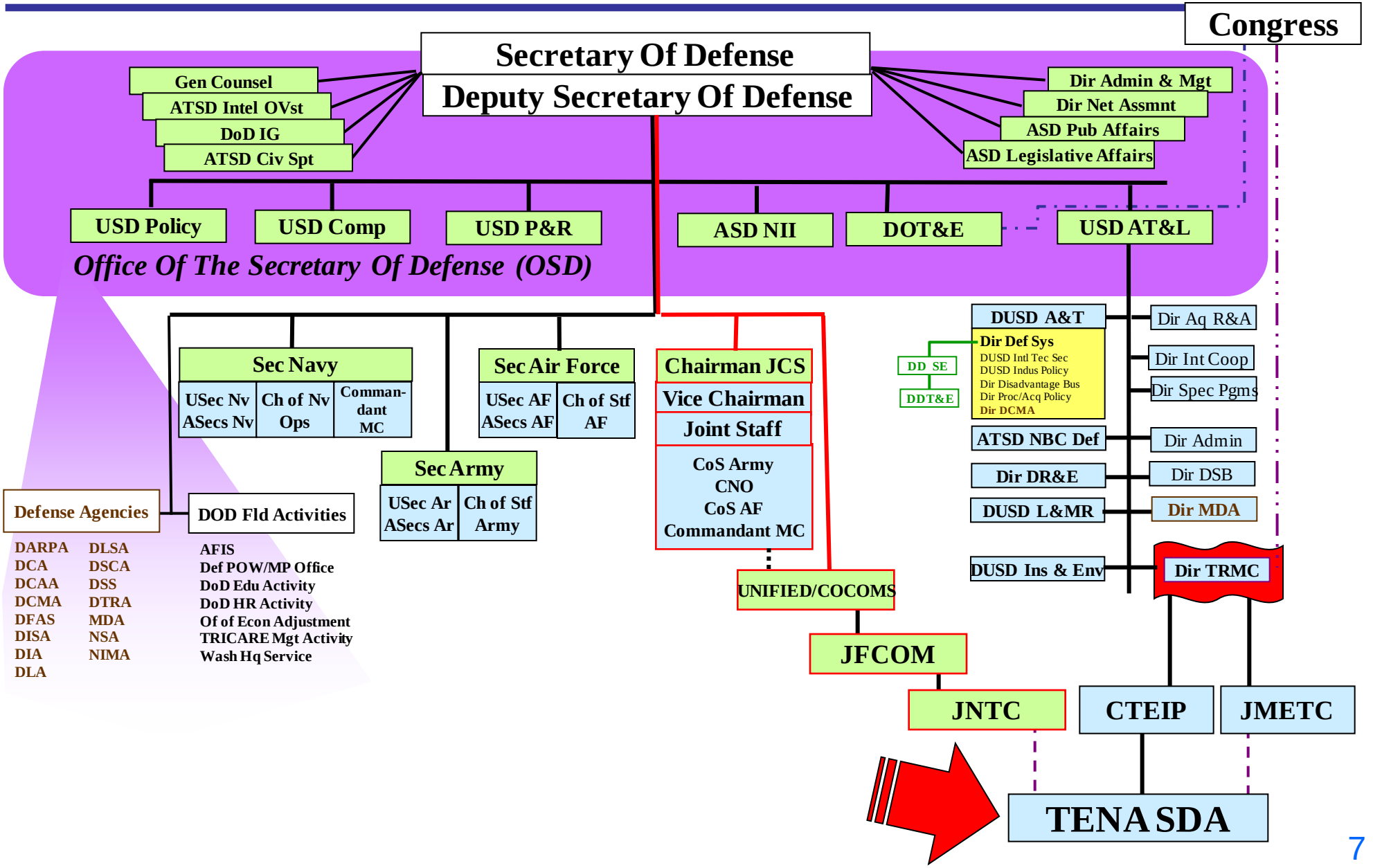
- **Support the Warfighter (Joint Vision 2010/2020)**
- **Enable SBA, STEP, CEE, JSB, and JDEP**
- **Foster Test and Training Integration**
- **In the long term: SAVE MONEY!**

Lay the Foundation for Future Test and Training Range Instrumentation

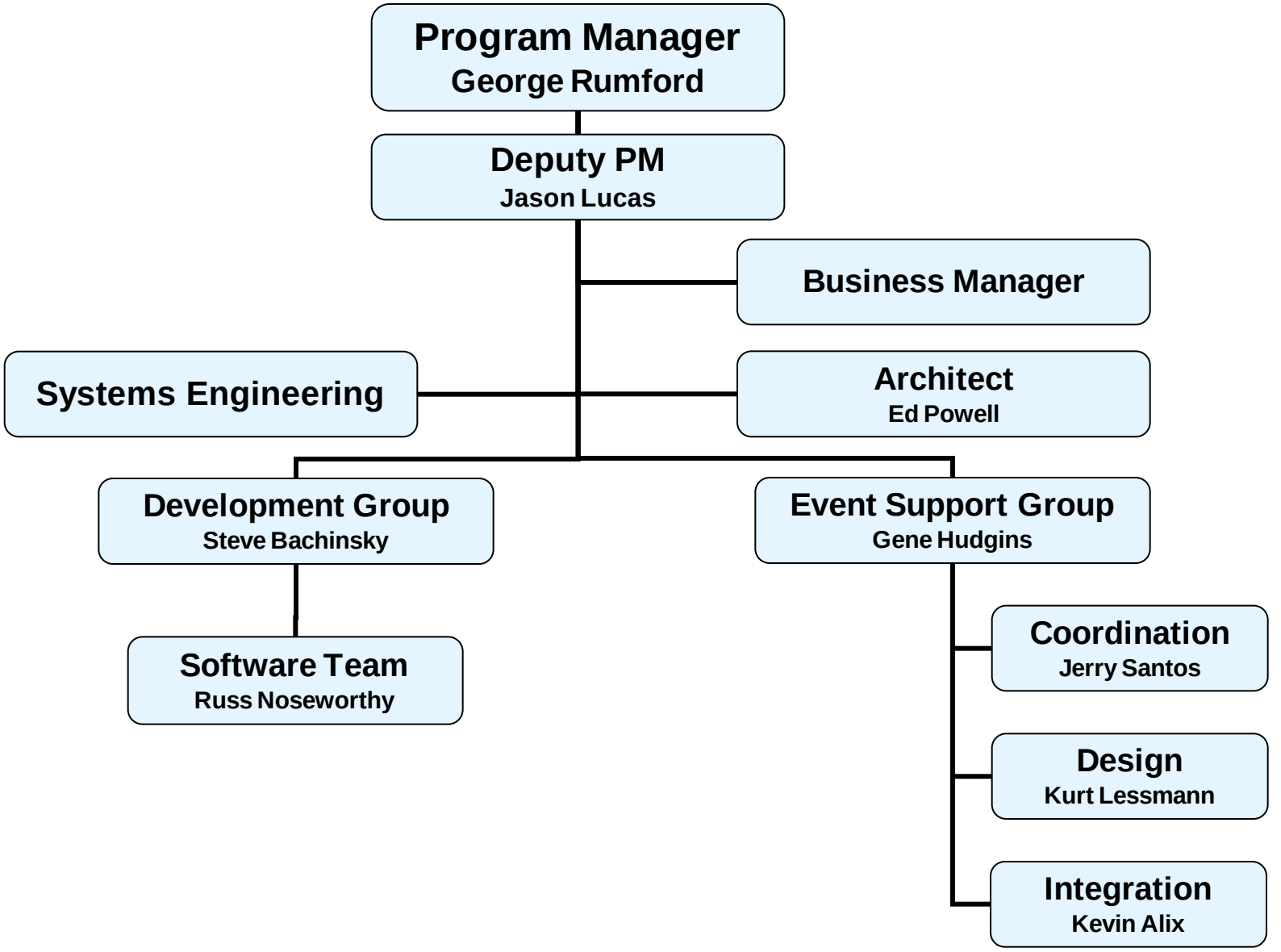
TENA Project Objectives

- **Define a common Architecture for the test/training range community – called "TENA" (Test & Training Enabling Architecture)**
  - Define a common **Object Model** to be used across the ranges
  - Define and build a common software **middleware** that will:
    - Use the object model
    - Enhance interoperability and reuse among the ranges
- **Common understanding of range processes – the Logical Range Concept of Operations**
- **Define and prototype common Tools to configure and conduct multi-range, synthetic test events or training exercises**
  - Create distributed, synthetic battlespaces with real weapon systems
  - Link multiple ranges together to form a larger, cohesive range
  - Enable testing, assessment, experimentation, and training of weapon system interoperability, C4ISR, and system-of-systems

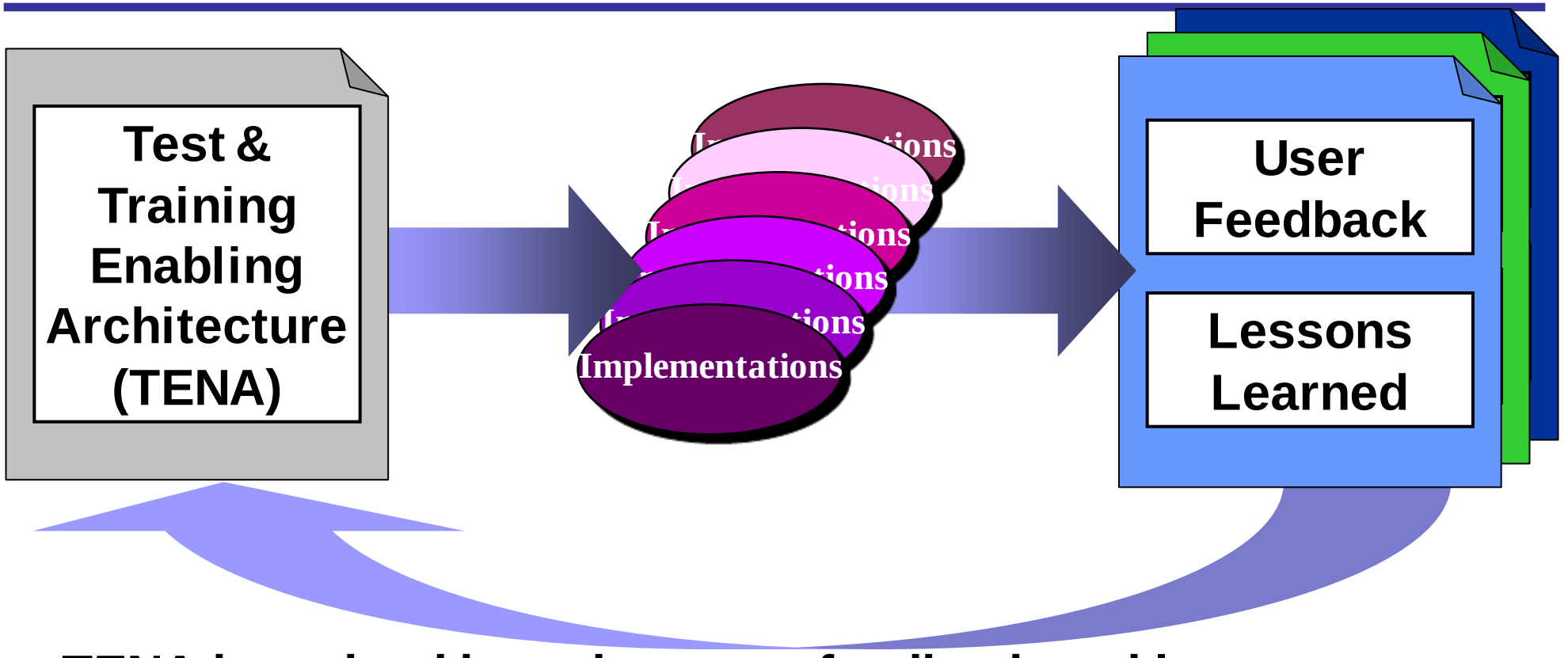
Where TENA SDA Fits in DoD



TENA SDA Organization



TENA Development Strategy



- TENA is revised based on user feedback and lessons learned from working software implementations
- TENA will be revised in the future based on future Implementations

TENA is based on real-world tests at real ranges

Architecture Management Team (TENA AMT)

- **AMT Members:**
  - 329 Armament Systems Group (329 ARSG)
  - Aberdeen Test Center (ATC), Aberdeen Proving Ground, MD
  - Air Armament Center (AAC), Eglin AFB, FL
  - Air Force Flight Test Center (AFFTC), Edwards AFB, CA
  - Army Operational Test Command (OTC), Fort Hood, TX
  - Common Training Instrumentation Architecture (CTIA)
  - Dugway Proving Ground (DPG)
  - Electronic Proving Ground (EPG)
  - integrated Network Enhanced Telemetry (iNET)
  - Interoperability Test and Evaluation Capability (InterTEC)
  - Joint Fires Integration & Interoperability Team (JFIIT)
  - Joint National Training Capability (JNTC)
  - Naval Air Warfare Center – Aircraft Division
  - NAWC – Weapons Division
  - Naval Aviation Training Systems Program Office (PMA-205)
  - Naval Undersea Warfare Center (NUWC)
  - NAVSEA Warfare Center - Keyport
  - P5 Combat Training System (P5CTS)
  - Pacific Missile Range Facility (PMRF)
  - Redstone Technical Test Center (RTTC)
  - T&E/S&T Non-Intrusive Instrumentation
  - White Sands Missile Range (WSMR)

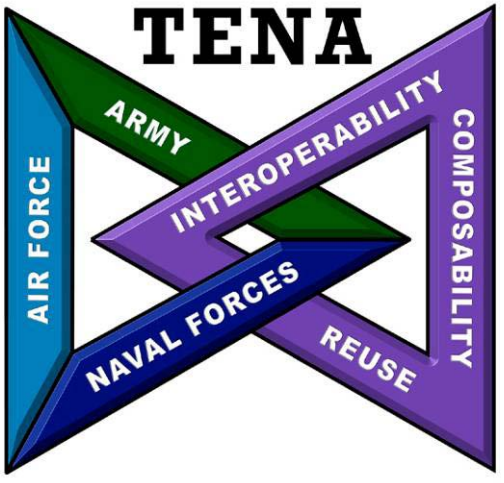
Meetings every 3 months

Advising Members:
- BMH Associates, Inc.
- Boeing
- Cubic Defense
- DRS
- Embedded Planet
- EMC
- Kenetics
- MAK Technologies
- NetAcquire
- Science Applications International Corporation (SAIC)
- Scientific Research Corporation (SRC)
- Scientific Solutions, Inc. (SSI)

- **Design Decisions / Trade-offs / Status / Technical Exchanges of Lessons Learned / Use Cases / Testing / Issues & Concerns Identification, Investigation & Resolution**

TENA is an Open Architecture

- **SEI defines an Open System as "a collection of interacting software, hardware, and human components designed to satisfy stated needs with interface specifications of its components that are fully defined, available to the public, maintained according to group consensus, in which the implementations of the components conform to the interface specifications."**
- **TENA is maintained according to a consensus of its users assembled as the TENA Architecture Management Team (AMT)**
  - TENA Architectural Specification is publicly defined and available on the web
  - TENA Middleware Specification (API) is publicly available on the web
  - TENA Object Model is publicly available and downloadable without restriction
    - An Event Designer can create or modify object models for a given event to satisfy their particular event requirements
- **TENA Middleware exists and is being used to support real events**
  - Built on open source software – CORBA ACE/TAO
  - Government owned, without proprietary software
  - Plans in place for open source release



Some Uses of TENA

Range Integration in Millennium Challenge 2002 (MC02)

Blue Forces

- Ships
- Ground forces
- Aircraft

Opposing Forces

- Aircraft & air targets
- Ships
- Ground forces

Nellis AFB
TENA Gateway

Land Range/China Lake
TENA Gateway

National Training Center/Ft. Irwin
TENA Gateway

Electronic Counter-measures Range/China Lake
TENA Gateway

Sea Range/Point Mugu
TENA Gateway

US Marines/So. California Logistics Airfield

Joint Network

Joint Training, Analysis, and Simulation Center

TENA Gateway

Integrating Software

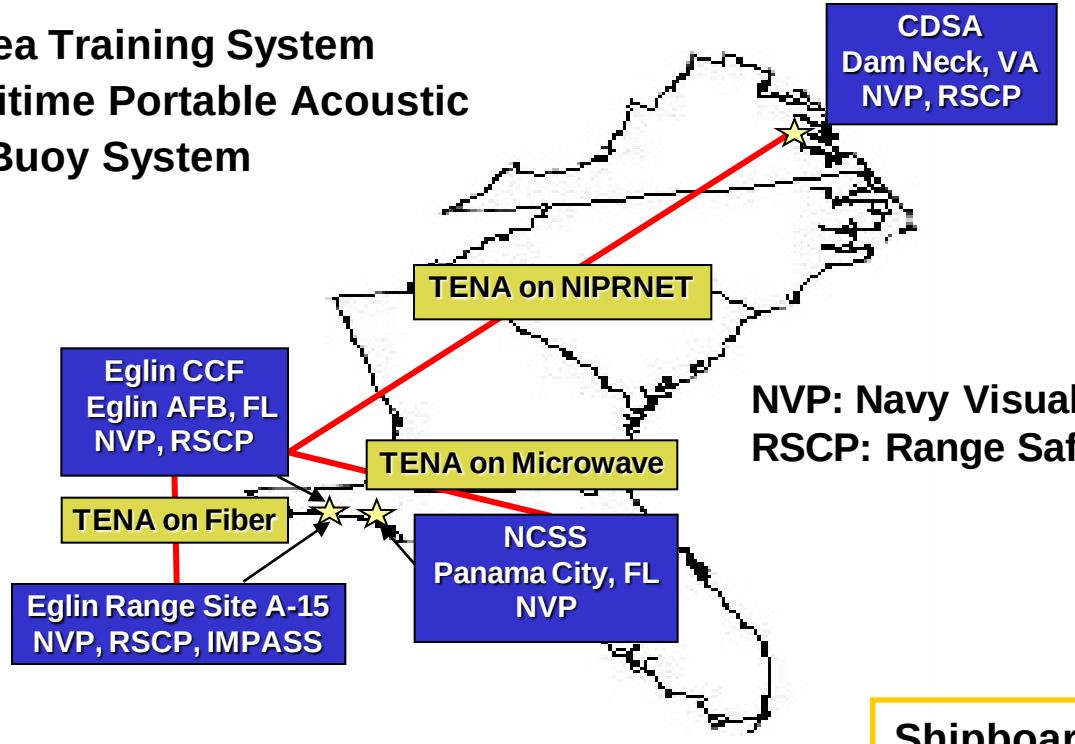
Global Command & Control System

Modeling & Simulation Feed

Command, Control, Communications, Computers, Intelligence Feed

VAST / IMPASS Over-the-Water Scoring

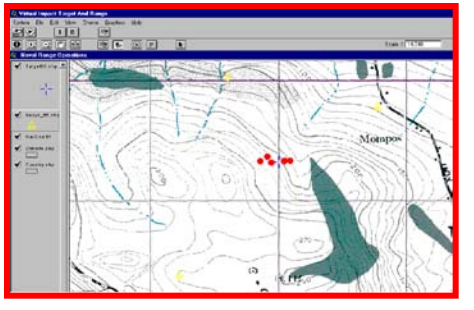
VAST: Navy Virtual At Sea Training System
IMPASS: Integrated Maritime Portable Acoustic Scoring and Simulator Buoy System



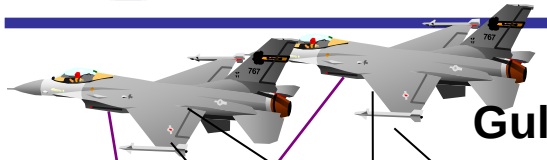
NVP: Navy Visualization Program
RSCP: Range Safety Control Program

GPS
Acoustic Processing
Communication Link

Shipboard Processing
Map Rendering
Virtual Target



JCIDEX 03 / TENA Activity



Live Infrastructure

Gulfport/Shelby/Camden MOA

ARDS GPS Pods

TACTS Pods

JTIDS Terminal

CRTC TACTS GND STN

CRTC LAN

ARDS GND STN

JTIDS TENA IF Gateway

TACTS TENA IF Gateway

ARDS TENA IF

Router

JCIET ADNET

Router

SA/AAR Display

- PCDS -

Camp Shelby MS

JECG Display

Rangeview

Ft. Rucker (opt)

SA/AAR Display

JECG Display

JECG Display

-Rangeview –

(Analysis

(AMO, TSPI, JTIDS,

Instrumentation)

Gulfport CRTC

Casualty Assessment Workstation

(A/G, G/G, A/A geo-pairing)

SA/AAR Display

- PCDS -

(TSPI)

Router

Router

TENA Display

Rangeview

Eglin AFB

SA/AAR Display

AUV Fest 2003
SIMDIS

Seahorse

Crawler

REMUS

CETUS

Acoustic

Range Craft

RF

Range Buoy

Range Control

RS-232

Range Data Gateway

TCP

Static Mine Locations

TENA

Range Information Display Center (Keyport)

AUV Fest Ops Center (Keyport)

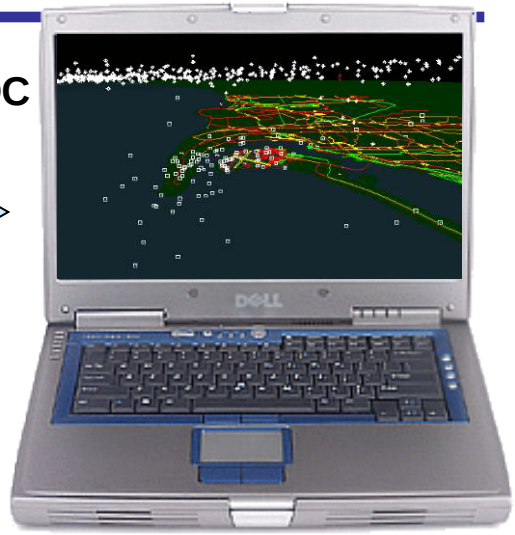
Newport

SIMDIS Use of TENA

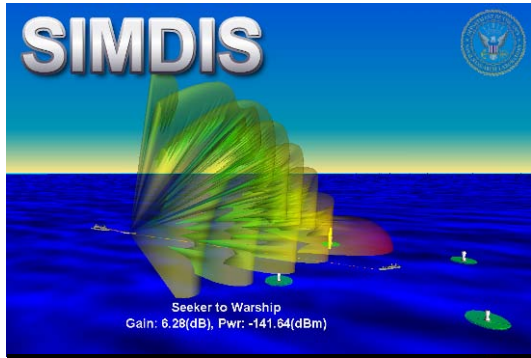
Southern California

TENA

NRL Washington, DC



- **Duration testing using SCORE TSPI data feed**
  - Four consecutive days
    - Win XP, Red Hat 9, Solaris 5.8
    - Processed 180,000+ entities
  - Two consecutive days
    - Win XP, Red Hat 9
    - Processed 53,000+ entities
- **Results and observations**
  - No issues with discovery latency
  - No issues with update latency
  - No issues with CPU usage
  - No issues with memory usage

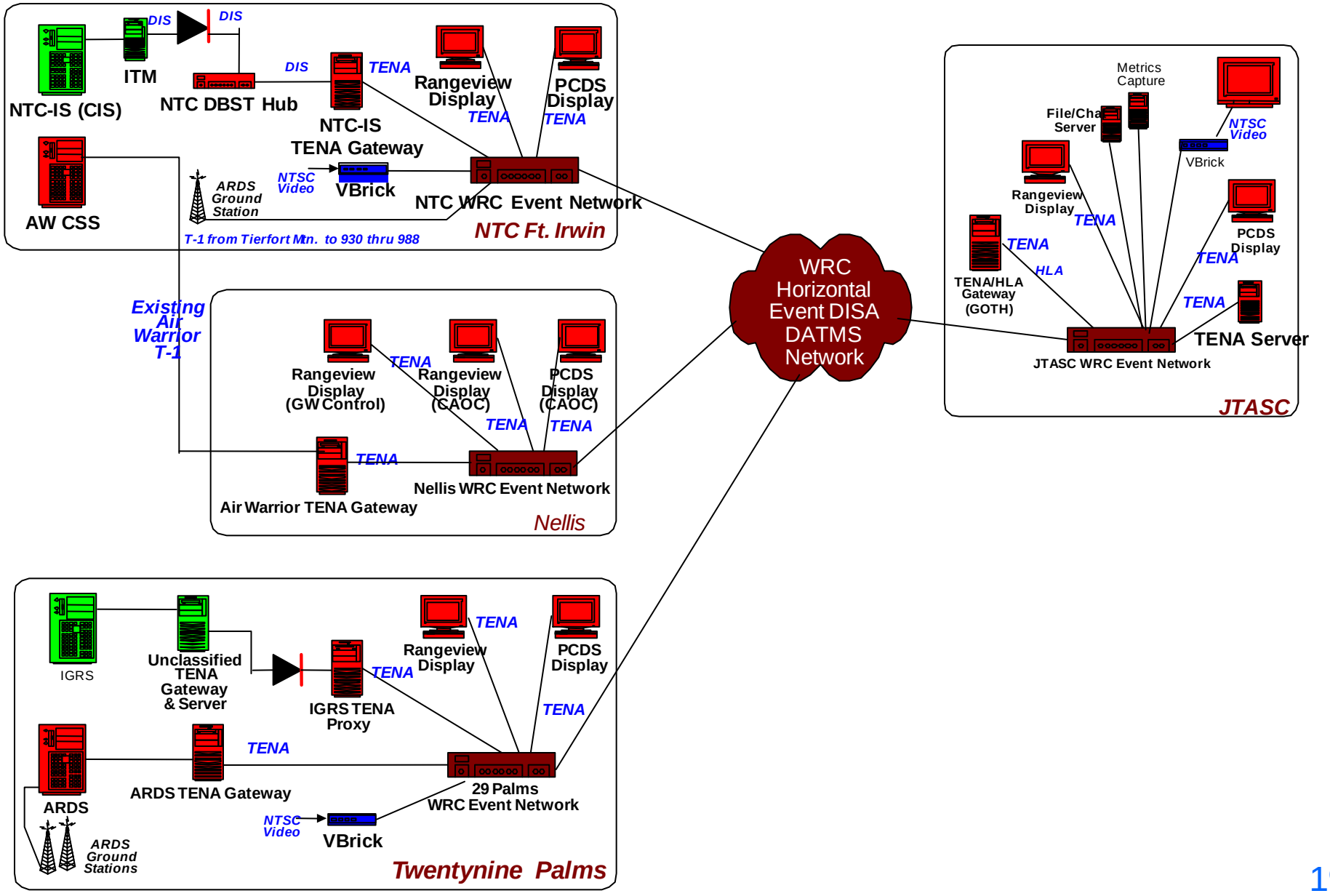


Threat Systems Test of TENA

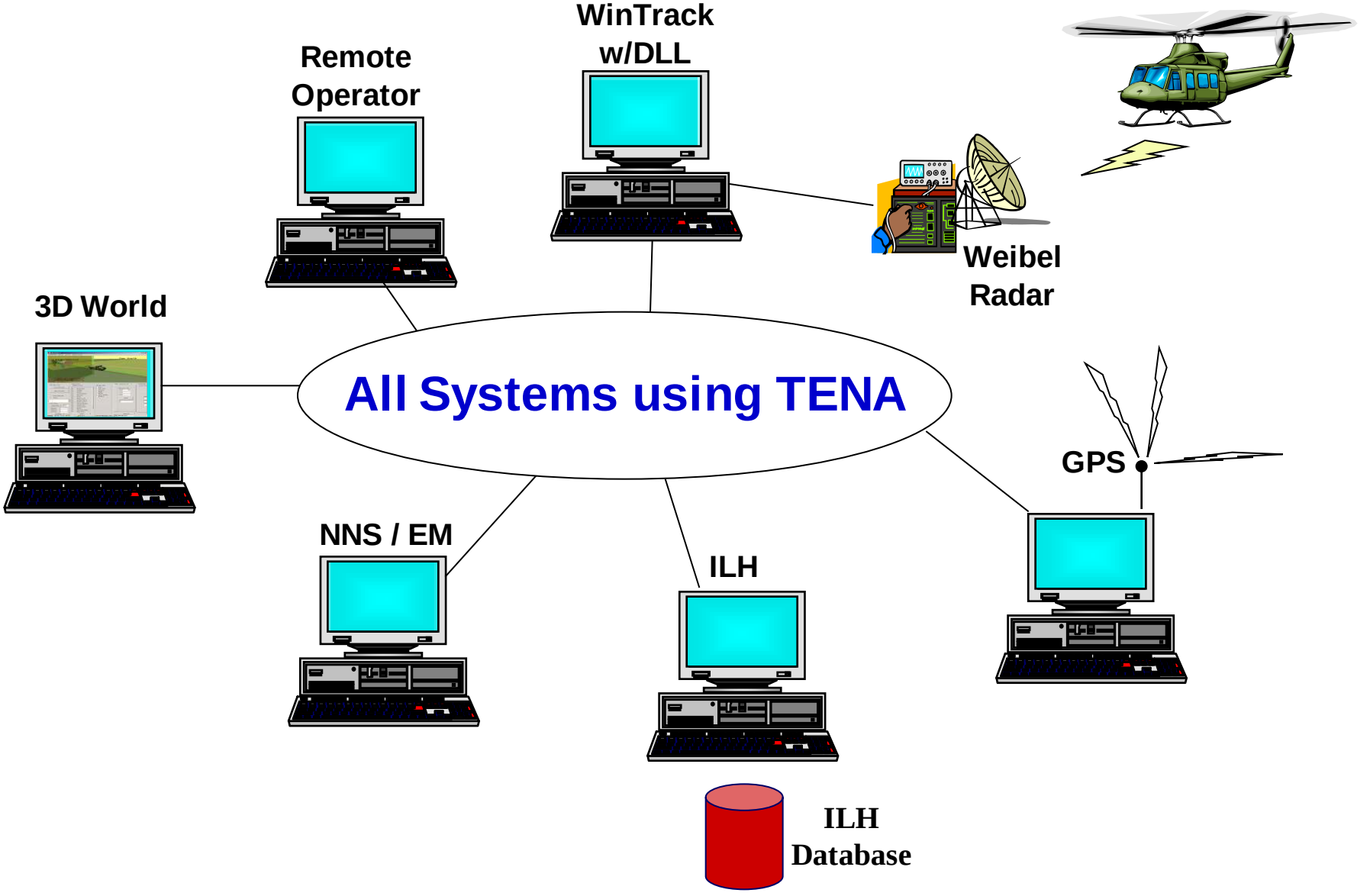
- **Testing and analysis by Scientific Research Corporation (SRC)**
- **Results and observations:**
  - TENA Middleware appears stable and predictable
  - TENA Object Model format is sufficient for representation of threat systems
  - TENA provides satisfactory functionality and performance to be utilized within a threat simulation scenario and for fielding threat simulations

JNTC Horizontal Thrust Event Jan 04

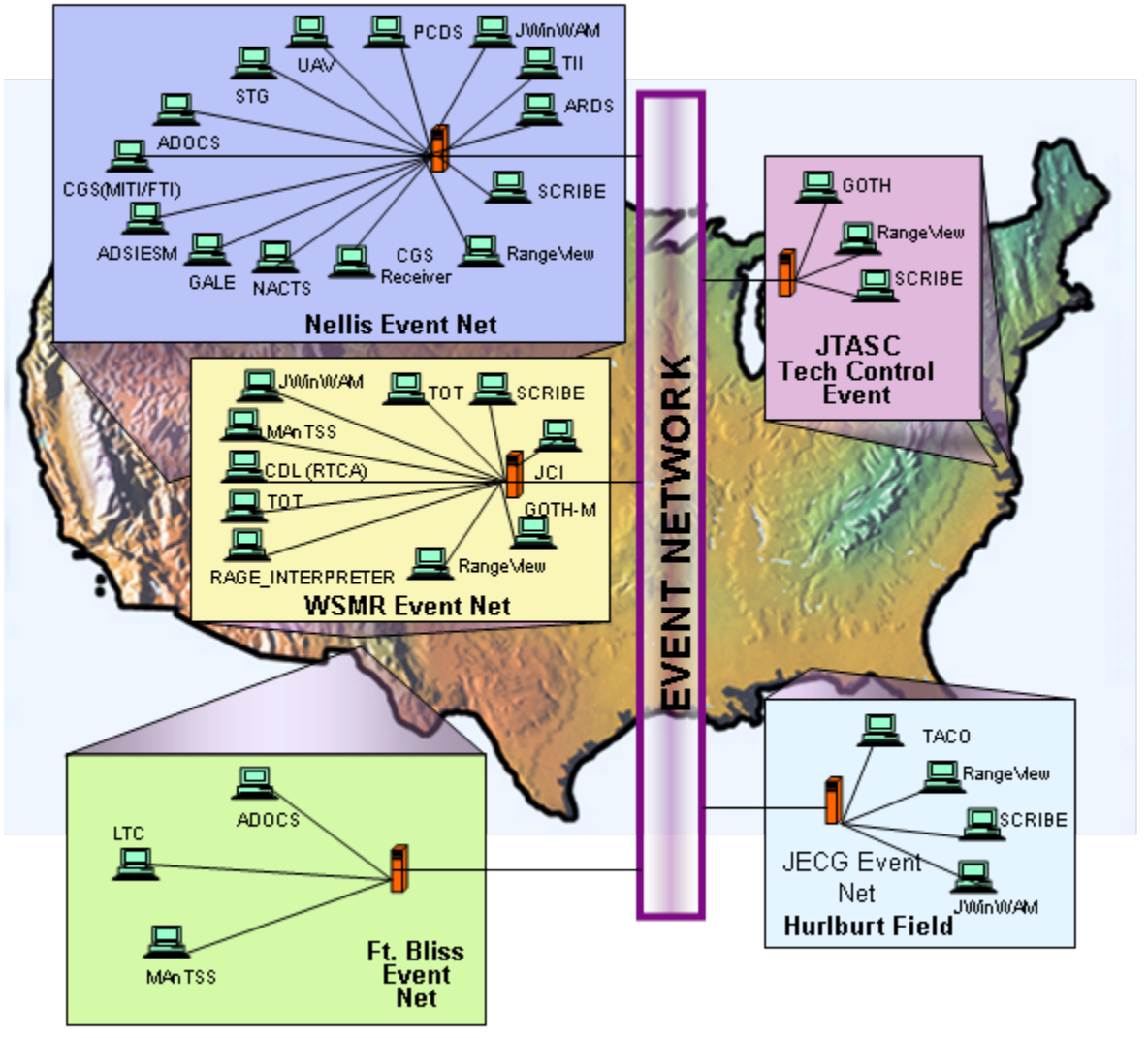
Range Integration & Instrumentation Solution



Weibel Radar Integration



Joint Red Flag 2005



- **ADOCS**: Automated Deep Operation Coordination System
- **ADSIESM**: Air Defense Systems Integrator
- **ARDS2TENA Gateway** (aka ARDS/ARDS-lite)
- **CDL**: Common Data Link (aka RTCA: Real-Time Casualty Assessment)
- **GALELITE**: Graphical Area Limitation Environment
- **GOTH**: Gateway of TENA/HLA
- **GOTH-M**: Gateway of TENA/HLA (Using Mak RTI)
- **JCI**: Joint CDL Interface (aka TENA ARDS/ARDS-Lite interface)
- **JLT**: JRF05 LROM Tool (TENA Interface Acceptance Test Tool)
- **JWinWAM**: Joint Windows-based Warfare Assessment Model (aka TENA2WAM)
- **LTC** (JFIIT): Link16 to TENA Converter
- **MAnTSS**: Modular Analysis and Test Support Software (White Cell) (JCMD)
- **CGS Receiver**: Common Ground Station (aka MTI/FTI: Moving Target Indicator/Fixed Target Indicator
- **NACTS**: Nellis Air Combat Training System Gateway
- **PCDS**: Personal Computer Debriefing System (aka Catch.TENA)
- **RAGE_INTERPRETER** (WSMR)
- **RangeView**: Display
- **SCRIBE**: TENA Logger
- **STG**: Static Target Generator
- **TACO**: TACtical Office
- **TTALC**: JFIIT TSPI Logger
- **TII**: TENA ICADS (Individual Combat Aircrew Display System)
- **TOSTADA** (aka DIS-TENA Gateway)
- **TOT** (aka OT-TES: Operational Test Tactical Engagement System)
- **UAV**: Unmanned Aerial Vehicle

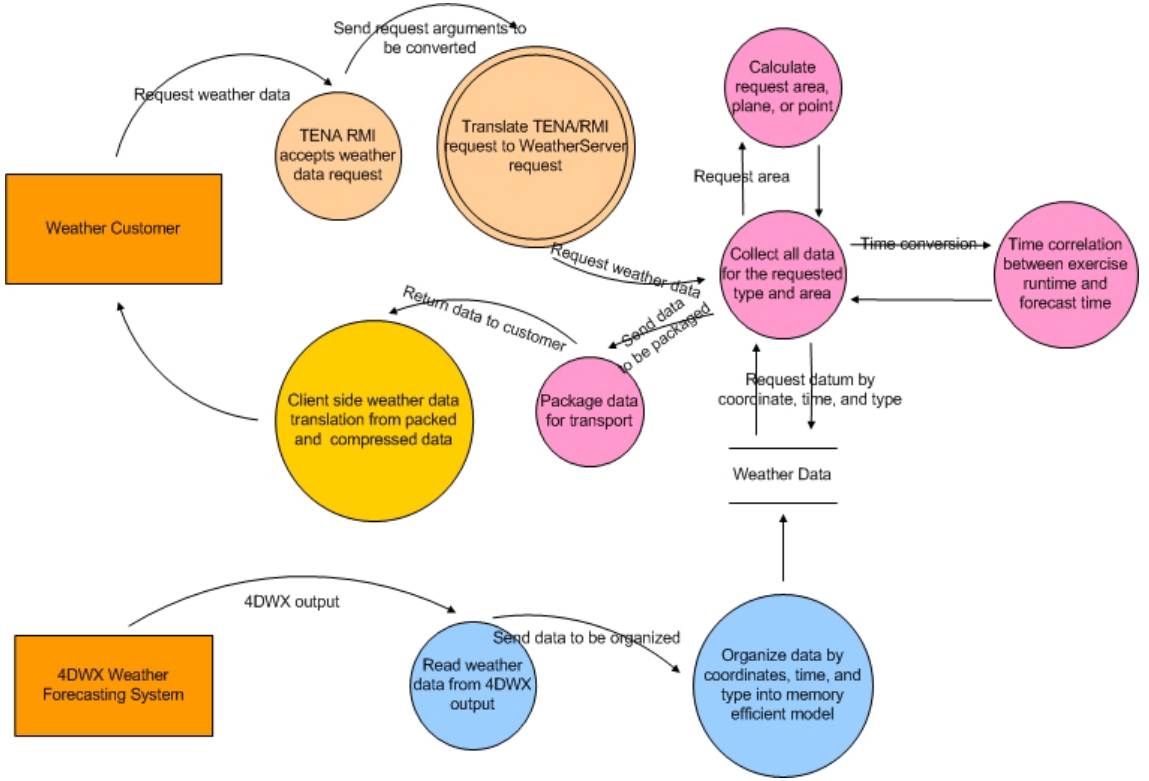
TENA in Real-Time Embedded Instrumentation by *NetAcquire*

- **Goal: demonstrate commercial-off-the-shelf (COTS) TENA operation in the following domains:**
  - **Real-time (strict constraints on data acquisition and response time)**
  - **Direct hardware interfaces not standard on COTS desktops**
    - **Aerospace serial I/O formats (synchronous, telemetry, special protocols, etc.)**
    - **GPS (time and position)**
    - **Analog input/output**
    - **Digital and pulse input/output**
    - **IRIG timing**
    - **Avionics buses (1553, ARINC, 1394)**
    - **GPIB (IEEE-488) instrumentation**
  - **Inexpensive, ruggedized, mobile form-factor**

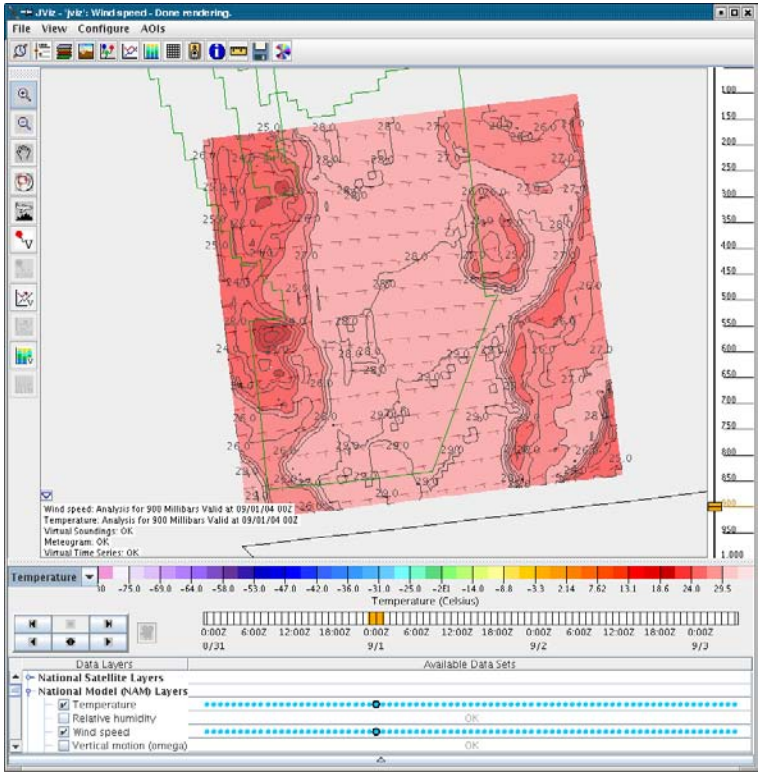
- **Accomplishments:**
  - **NetAcquire hardware/software product now successfully runs TENA**
  - **Direct synchronous serial hardware interface to FPS-16 radar system is supported**
  - **Radar system data auto-populated into TENA Radar SDO in real-time**
  - **Little or no programming required to support different radar data formats**
- **NetAcquire runs a true real-time operating system, device drivers, and application software**
  - **Provides TENA with deterministic and bounded response times**

TENA Used to Distribute 4-Dimensional Weather Data

- **Team from Dugway Proving Ground Meteorology Division, National Center for Atmospheric Research, and Keane Corporation developed a sophisticated weather server using TENA**
- **Weather information generated by real-time, 4D data acquisition is processed by the TENA Weather Server and made available to TENA-enabled test event clients**
- **Distributed Test Events need weather data:**
  - Wind, temperature, barometric pressure, precipitation, time (4th dimension)



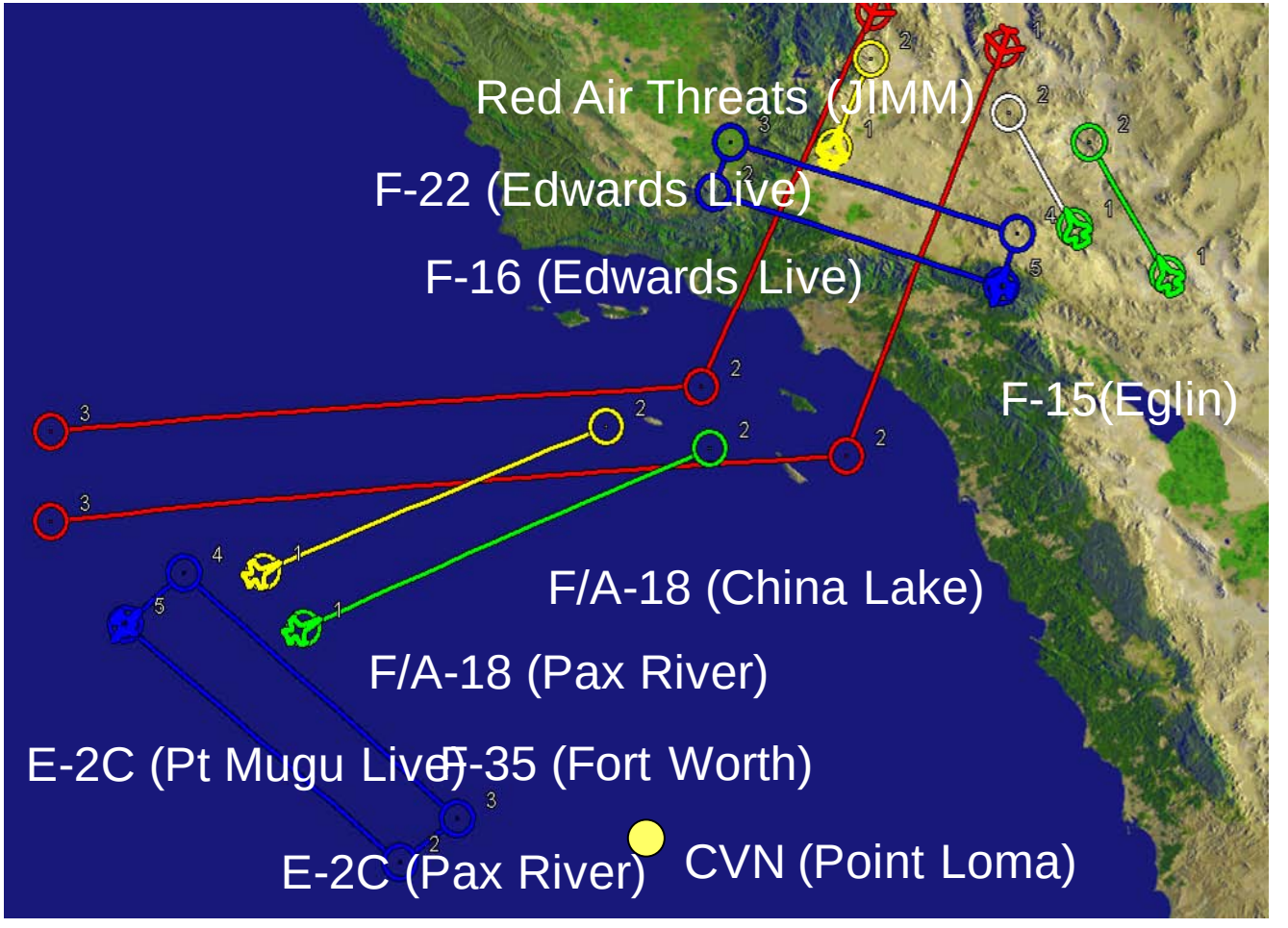
TENA Weather Server Data Flow



WSMR Temperature and Wind Fields

InterTEC Air Combat Mission JMETC Event

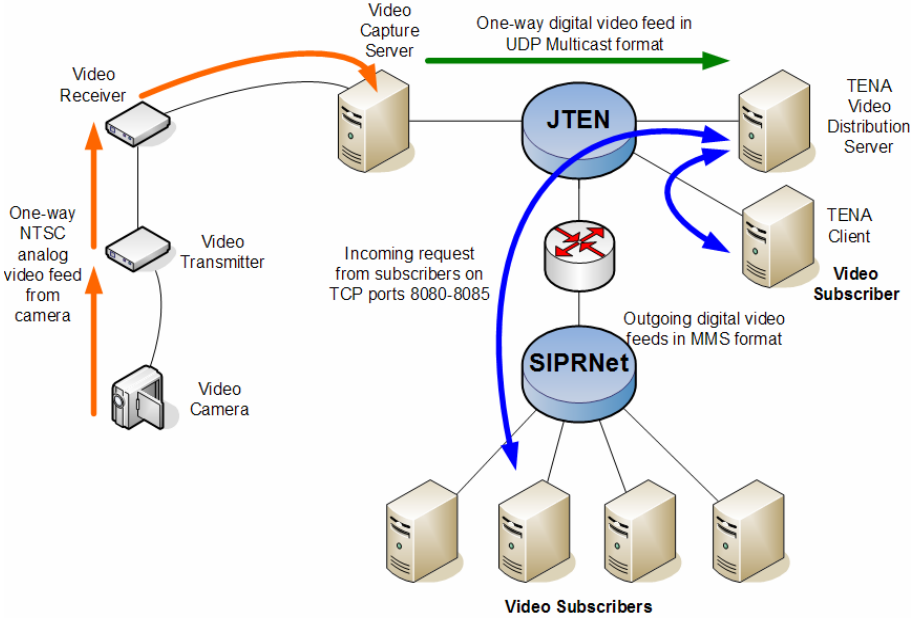
- TENA used in this large distributed LVC C4I Link-16 test event for data distribution of instrumentation, test control and distributed simulation between multiple sites



10 locations, 12 different applications, 56 instances of those apps linked together

TENA Used to Control Video Distribution Services with IO Range

- **TENA used to implement video distribution system for Information Operations (IO) Range in Austere Challenge 06 exercise.**
  - CONUS and OCONUS client terminals (30+) received video streams over SIPRNet.

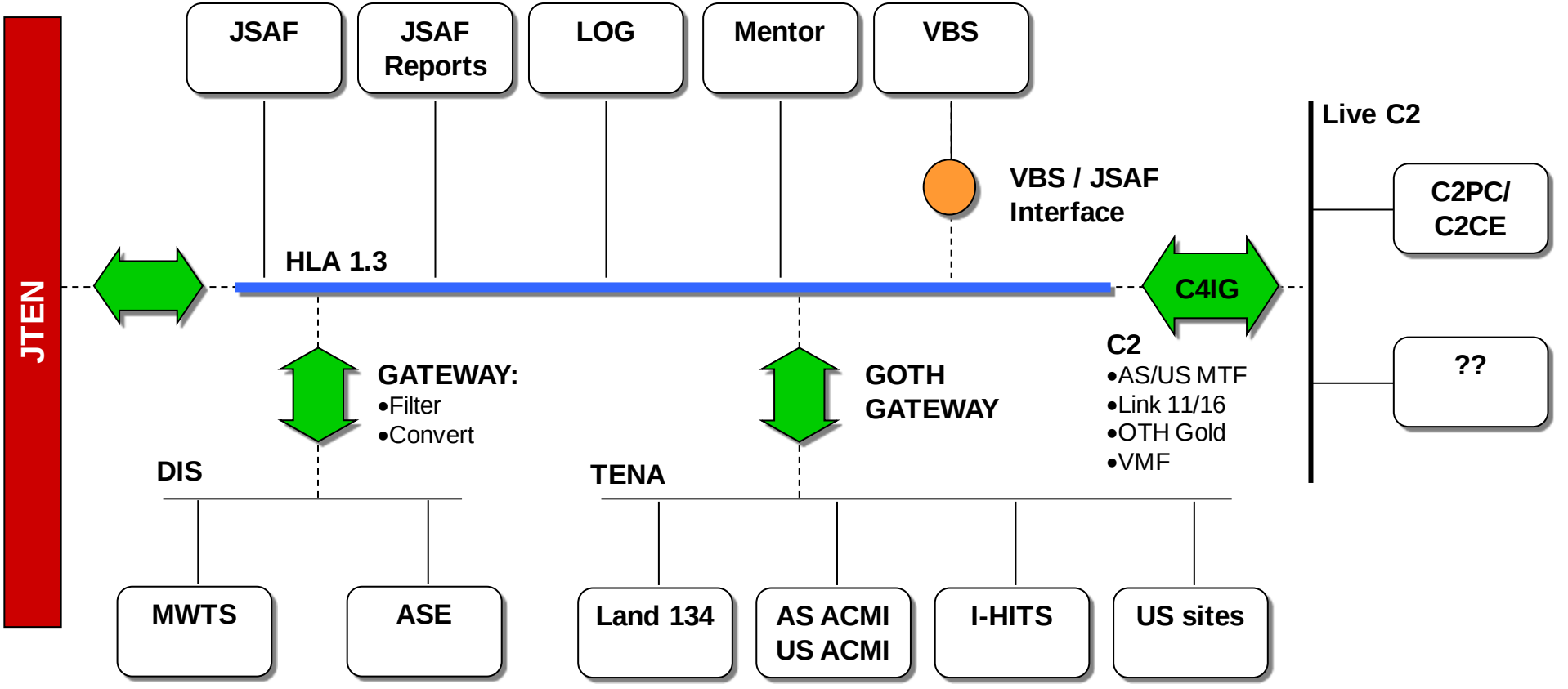


- **Video Distribution Server published availability of real-time and recorded data streams via Stateful Distributed Objects (SDO)**

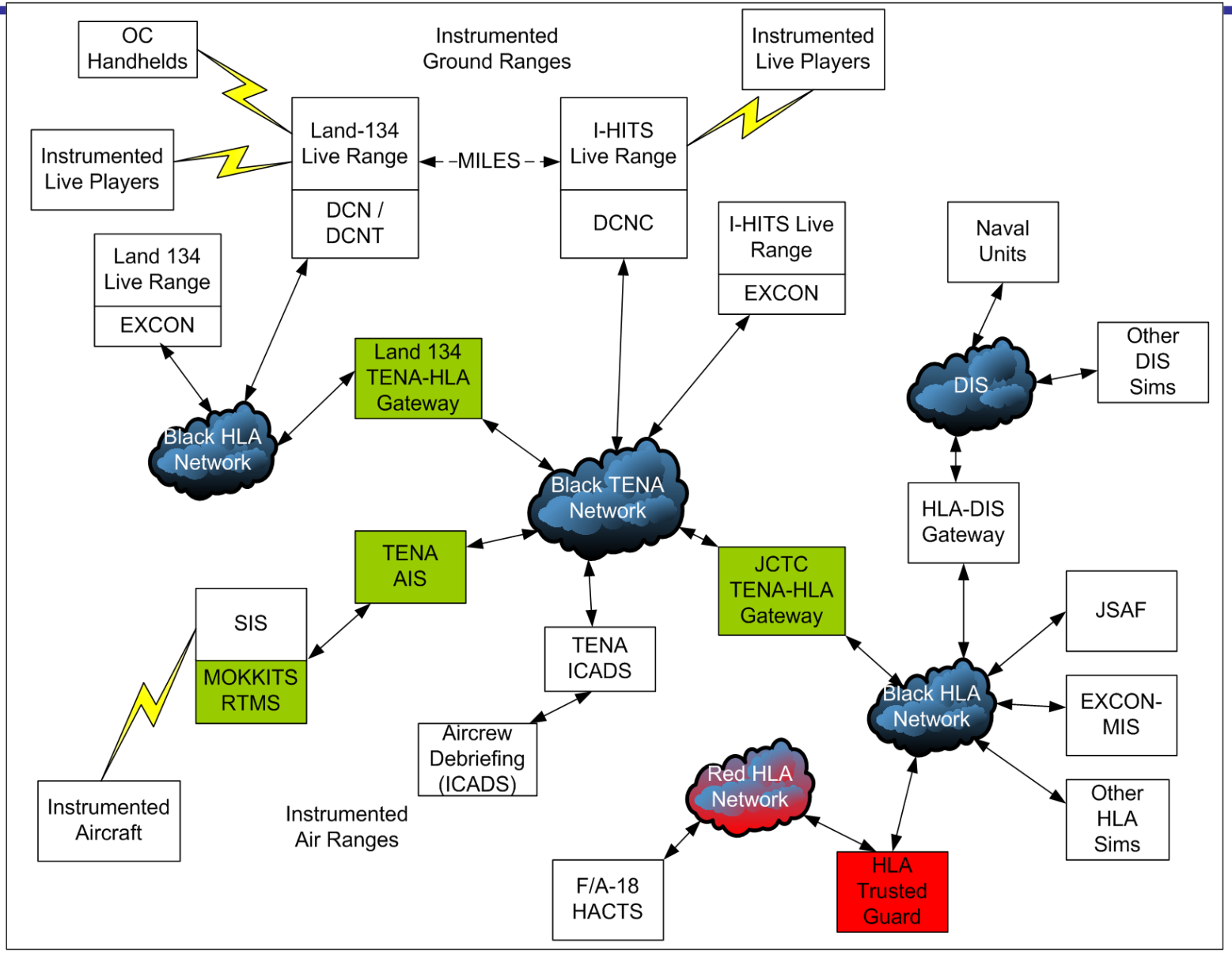
- **TENA auto-code generation enabled rapid development and integration of software**
  - Reduced technical risk and resulted in zero software failures during live fire event periods.

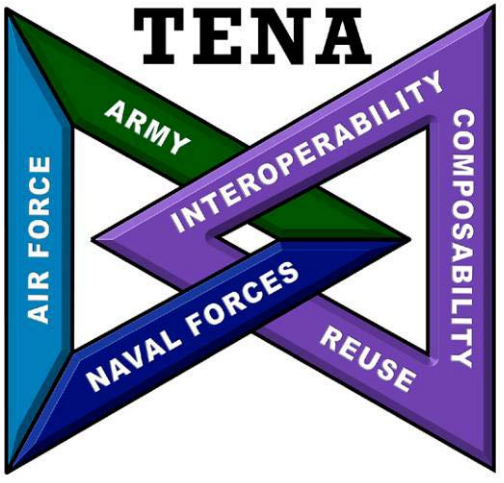
Talisman Sabre 07

TS07 Simulation Backbone



TS07 Extended ITQ-45 System





TENA Architecture

What is an Architecture?

- An architecture is a segmentation of a system (or system of systems) such that the primary pieces are identified, as well as their purpose, function, interfaces, and inter-relatedness, along with guidelines for their evolution over time
- Architectures put constraints on developers. These constraints make possible the achievement of higher level goals
- These higher-level goals are called the system's driving requirements
- An architecture is a bridge from requirements to design

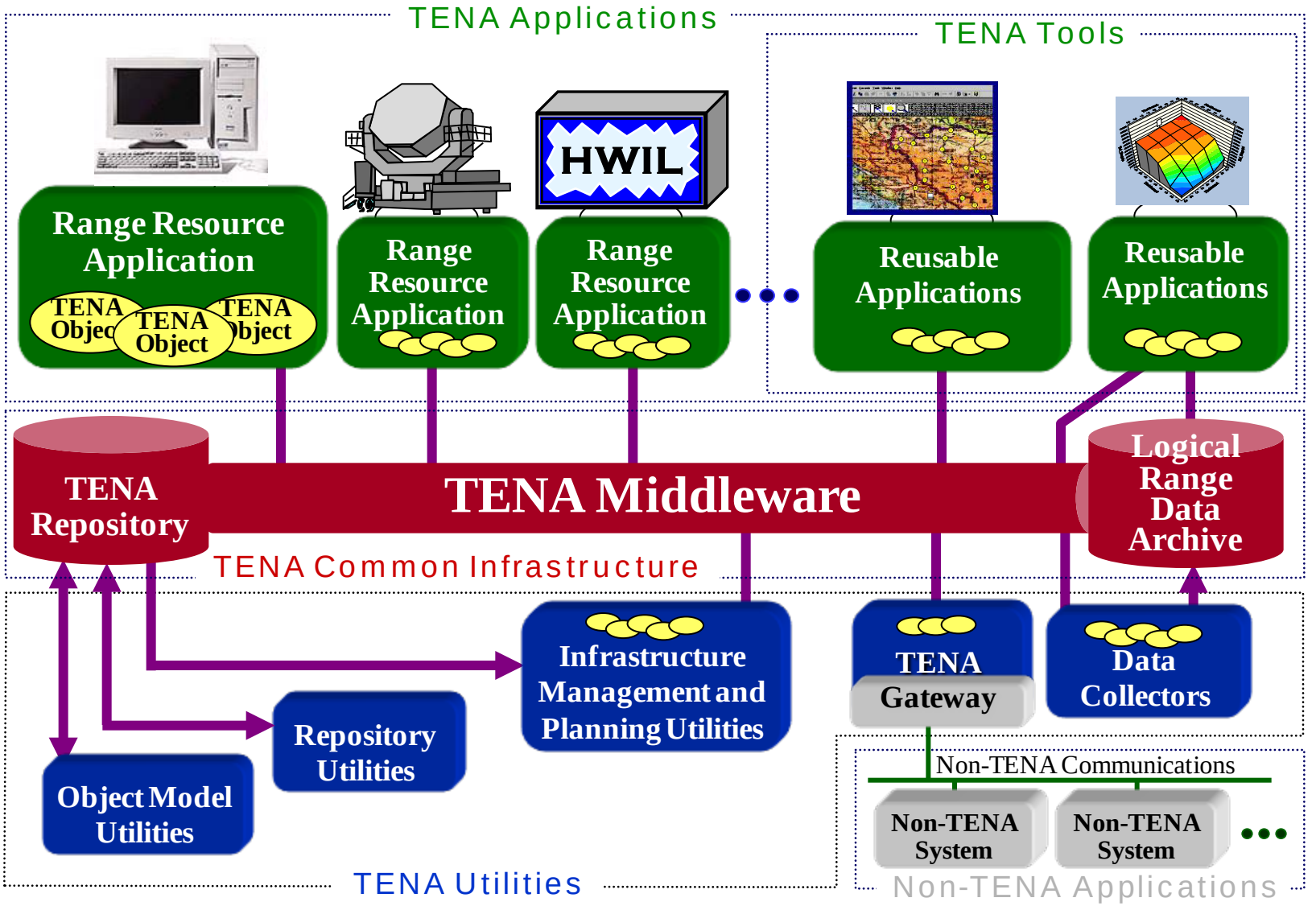
Technical Driving Requirements

1. **Interoperability**
   - The characteristic of a suite of independently-developed components, applications, or systems that implies that they can work together, as part of some business process, to achieve the goals defined by a user or users
2. **Reusability**
   - The characteristic of a given component, application, or system that implies that it can be used in arrangements, configurations, or in enterprises beyond those for which it was originally designed
3. **Composability**
   - The ability to rapidly assemble, initialize, test, and execute a system from members of a pool of reusable, interoperable elements
   - Composability can occur at any scale—reusable components can be combined to create an application, reusable applications can be combined to create a system, and reusable systems can be combined to create an enterprise

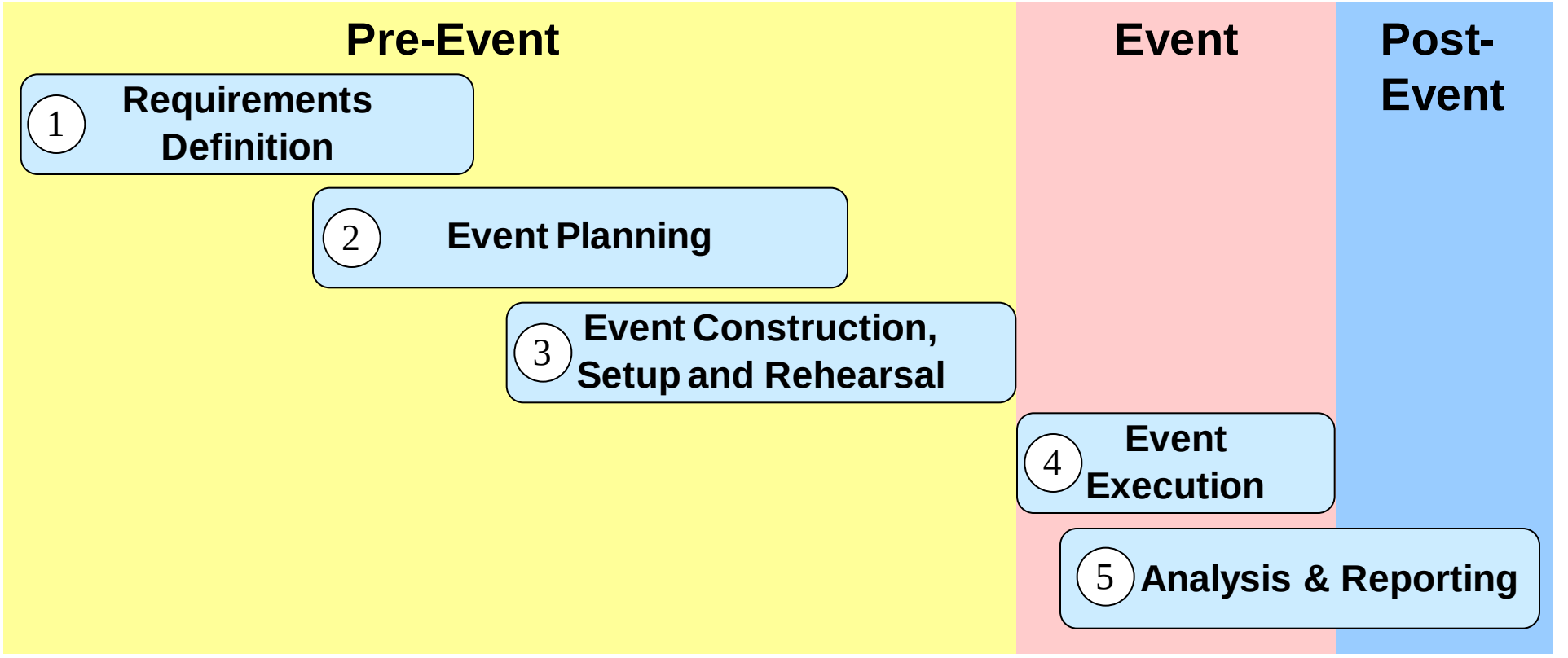
Achieving Interoperability and Reuse

- **Interoperability requires**
  - A common architecture → **TENA**
  - An ability to meaningfully communicate
    - A common language → **TENA Object Model (OM)**
    - A common communication mechanism → **TENA Middleware, LRDA**
  - A common context
    - A common understanding of the environment → **SEDRIS (as part of the TENA OM)**
    - A common understanding of time → **TENA OM, Middleware**
    - A common technical process → **TENA Technical Process**
- **Reuse and Composability require the above, plus**
  - Well defined interfaces and functionality for the application to be reused → **Reusable Tools, Repository**
  - Place to store reusable components → **Repository**

TENA Architecture Overview



Operational Architecture (including ConOps)



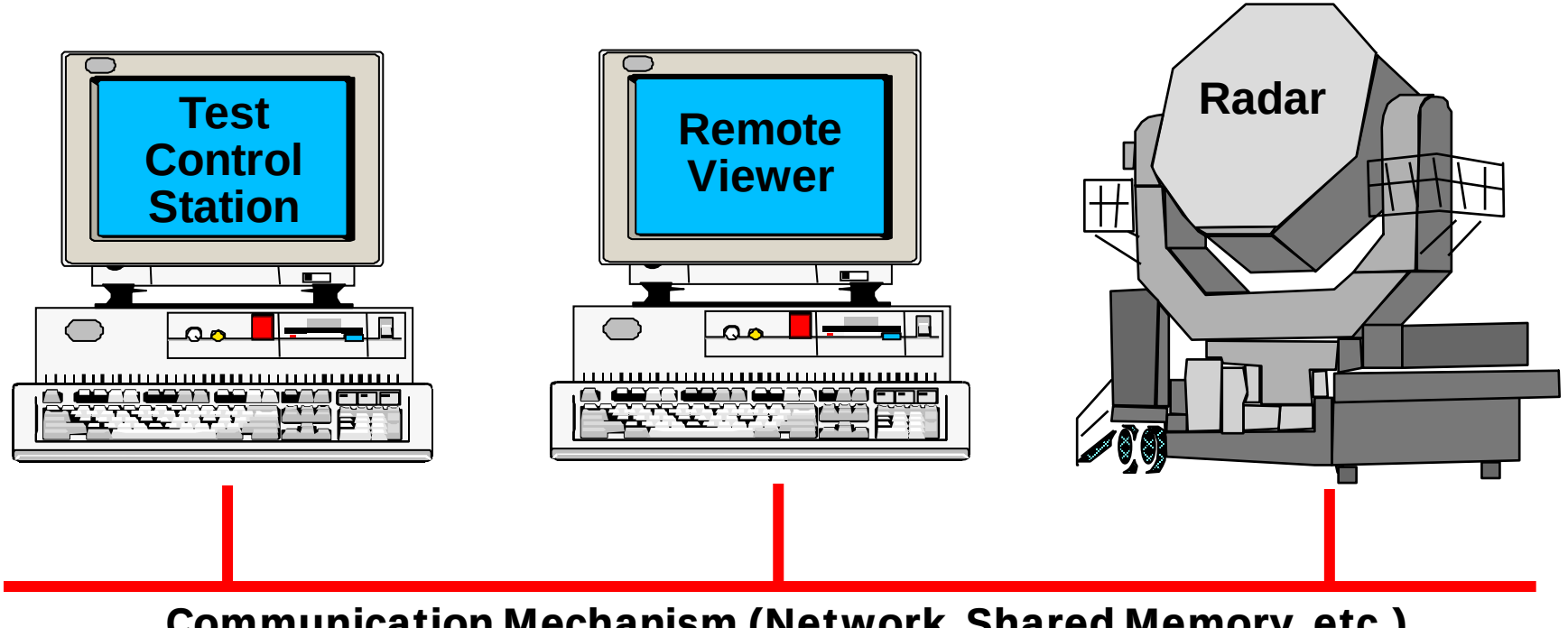
- **Three Phases**
  - Pre-Event / Event / Post-Event
- **Five Activities**
  - Requirements / Planning / Set-up / Execution / Analysis & Reporting

TENA Uses the Concept of a Logical Range

- **Logical Range – a suite of TENA Resources, sharing a common object model, that work together for a given range event**
- **TENA Resources are:**
  - Range Resource Applications - compiled to use the services provided by the TENA Middleware for interaction
  - Gateway Applications - to bridge TENA systems to legacy or other protocols or architectures
  - TENA Tools and Utilities - configured for a particular event
- **Common Object Model**
  - Logical Range Object Model (LROM) – the object definitions used in a particular event

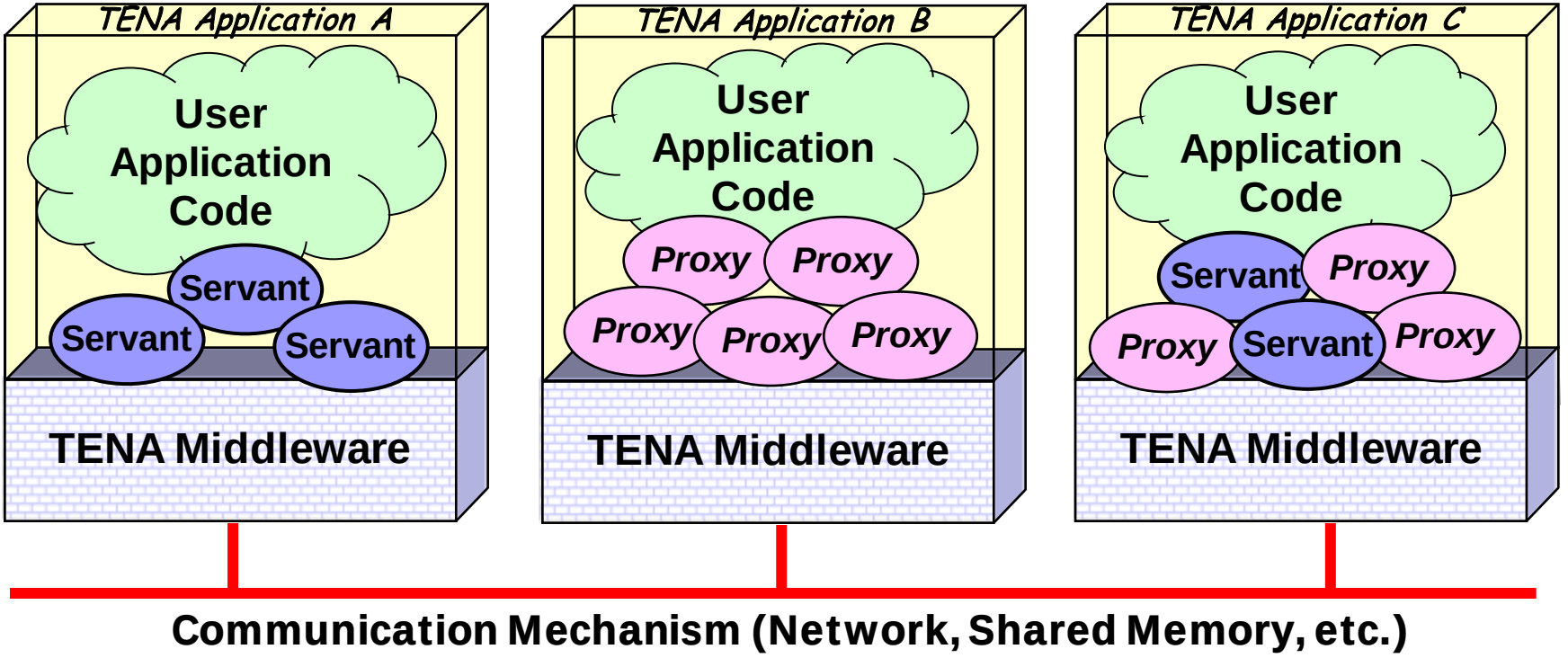
Logical Range Simple Example

TENA specifies an architecture for range resources participating in logical ranges



Logical Range Simple Example

- TENA specifies a peer-to-peer architecture for logical ranges:
  - Applications can be both clients and servers simultaneously
  - In their role as servers, applications serve TENA objects called "servants"
  - In their role as clients, applications obtain "proxies," representing other applications' servants
- The TENA Middleware, the TENA objects, and the user's application code are compiled and linked together



TENA Common Infrastructure

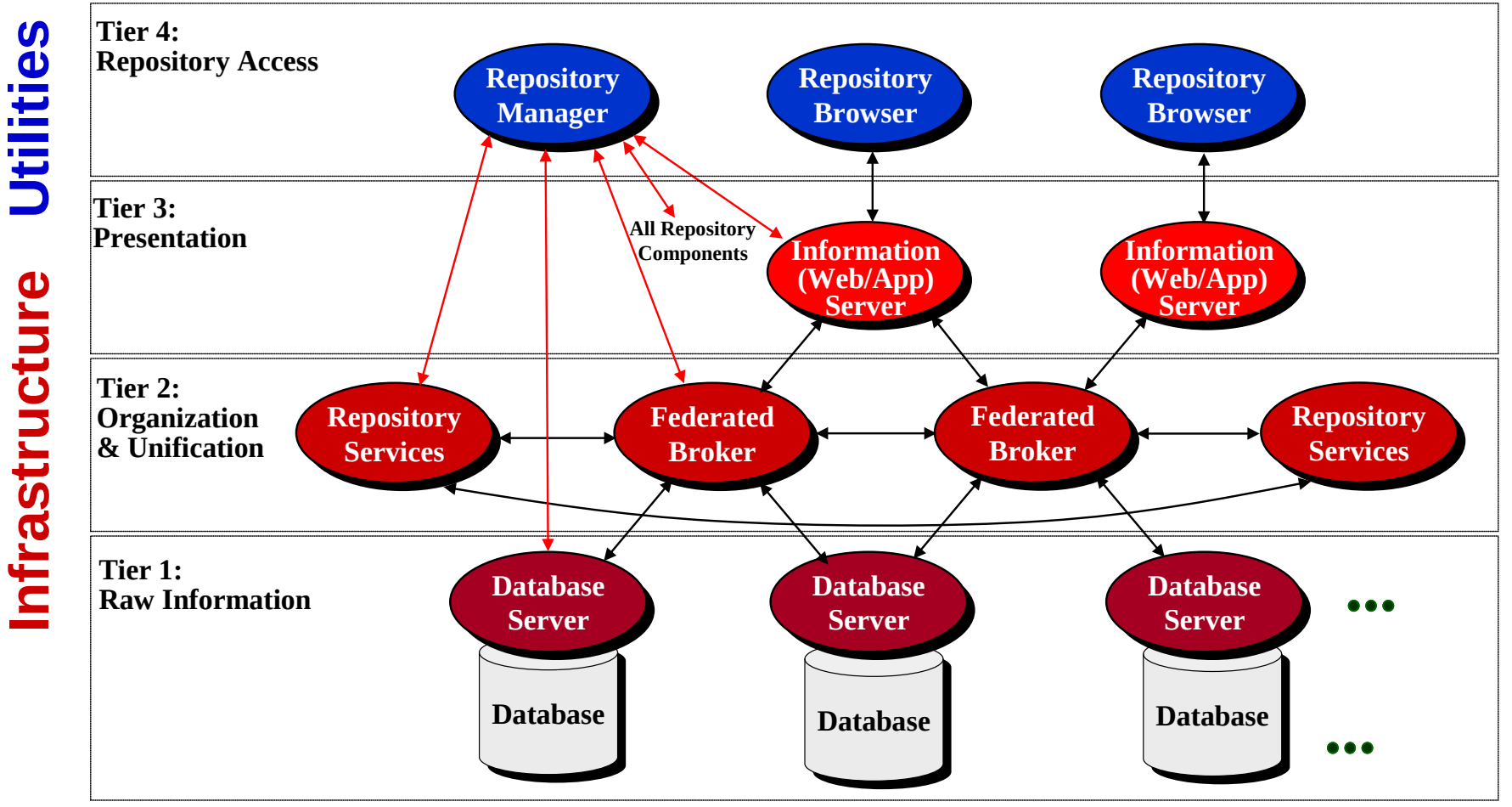
- **Components:**
  - Repository
  - Logical Range Data Archive
  - Middleware
- **Purpose:**
  - Provide the common, standardized, software mechanism that makes communication about objects in the TENA Object Model as efficient and simple as possible throughout the entire range event lifecycle

TENA Repository Purpose and Requirements

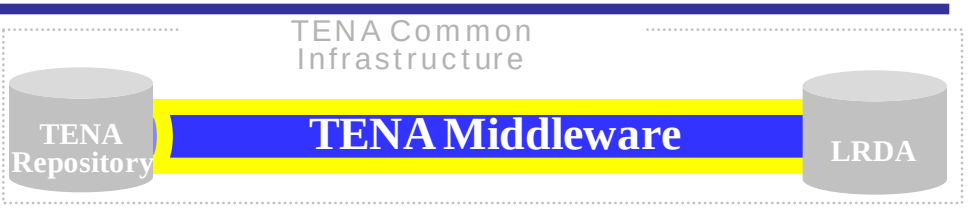
- **Purpose: to contain all the information relevant to TENA that is not specific to a given logical range**
- **Requirements:**
  - Store the TENA Object Model in all its forms including standard implementations
  - Store meta-data about all of its contents
  - Store TENA software (middleware, schemas, tools, gateways, reusable applications, and reusable components)
  - Store all TENA documentation
  - Store information from previous logical range executions for future reuse (including lessons learned)
  - Provide an easy-to-use secure interface to all of this information
- **The Repository is a database-of-databases, like the world-wide web.**
  - Except it has more meta-data, more security, more unification

TENA Repository Multi-Tiered Straw-Man Design

- This design is not "part of the architecture" — it is included to help illustrate the concept
- Obviously a web-based solution is the first step

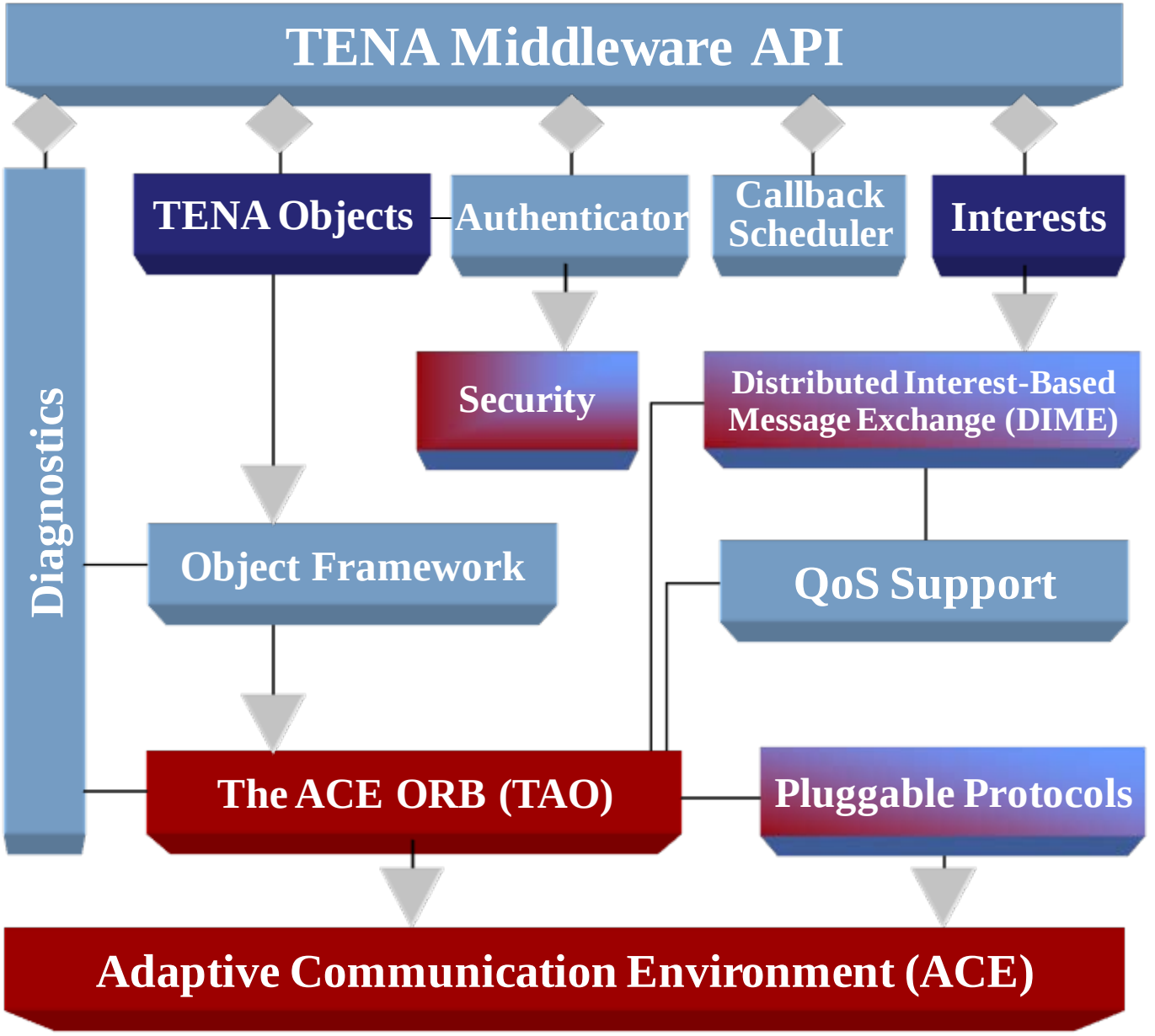


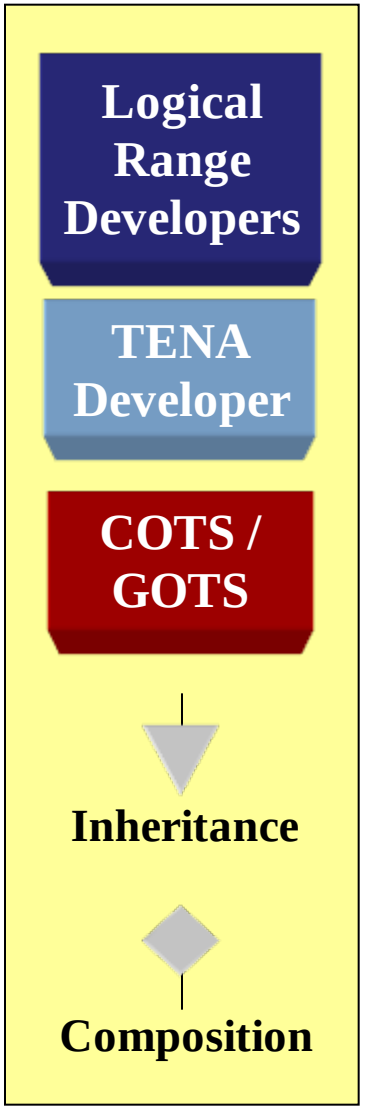
TENA Middleware Purpose and Requirements



- **Purpose: high-performance, real-time, low-latency communication infrastructure used by range resource applications and tools during execution**
- **Requirements:**
  - Fully support TENA Meta-Model
  - Be easy to use
  - Be highly reliable
  - Many varied communication strategies and media
    - Including management of quality-of-service
    - Including object-level security services
  - Be high-performance, including
    - Support multiple information filtering strategies
    - Support user-defined filtering criteria
  - Support a wide variety of range-relevant platforms (HW/OS/compiler)
  - Be technology neutral

TENA Middleware Current Design Overview

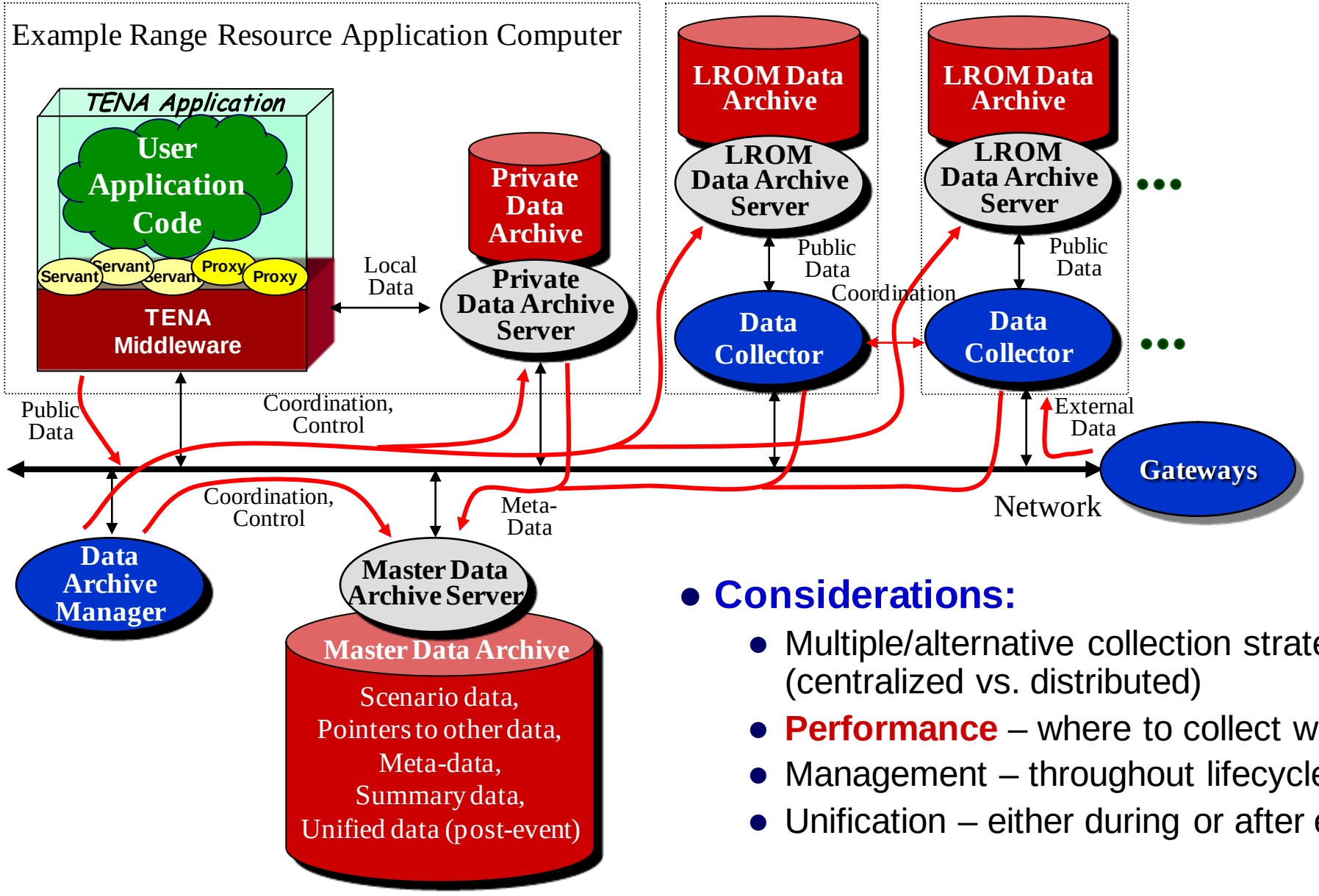




Logical Range Data Archive Purpose and Requirements

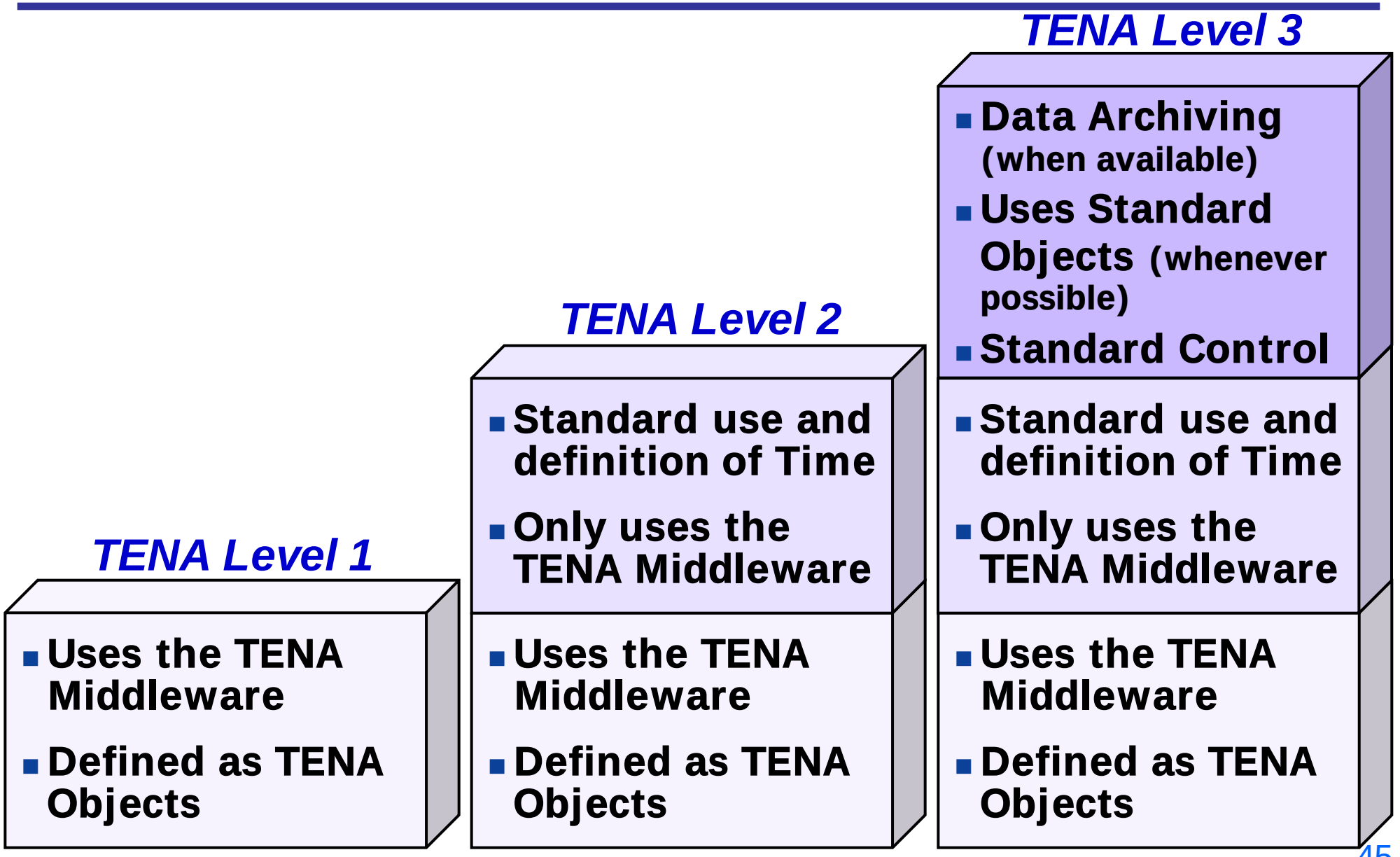
- **Purpose: store and provide for the retrieval of all of the information associated with a logical range execution**
- **Requirements:**
  - Store and serve initialization information
  - Store all data created in a logical range execution → **high-performance**
  - Store information at (possibly) multiple collection points → **distributed**
  - Support a "temporal" understanding of collected information → **temporal**
  - Support run-time queries as much as possible → **real-time**
  - Support post-event analytical queries
- **These things are non-trivial**
- **Does not have to be a single database running on a single computer (but could be)**
  - Perhaps a federated multi-database running on many computers throughout the logical range

Logical Range Data Archive Straw-Man Design



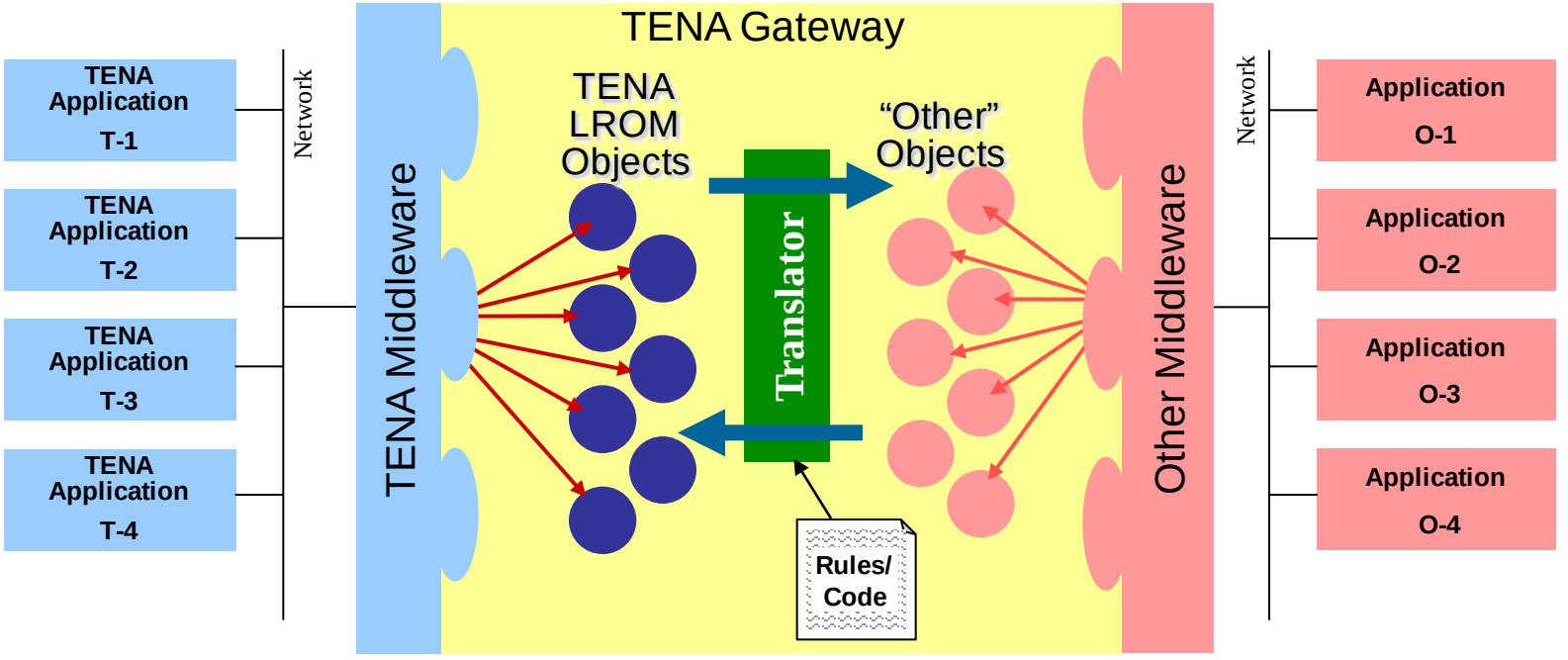
- **Considerations:**
  - Multiple/alternative collection strategies (centralized vs. distributed)
  - **Performance** – where to collect what?
  - Management – throughout lifecycle
  - Unification – either during or after event

TENA Compliancy Levels



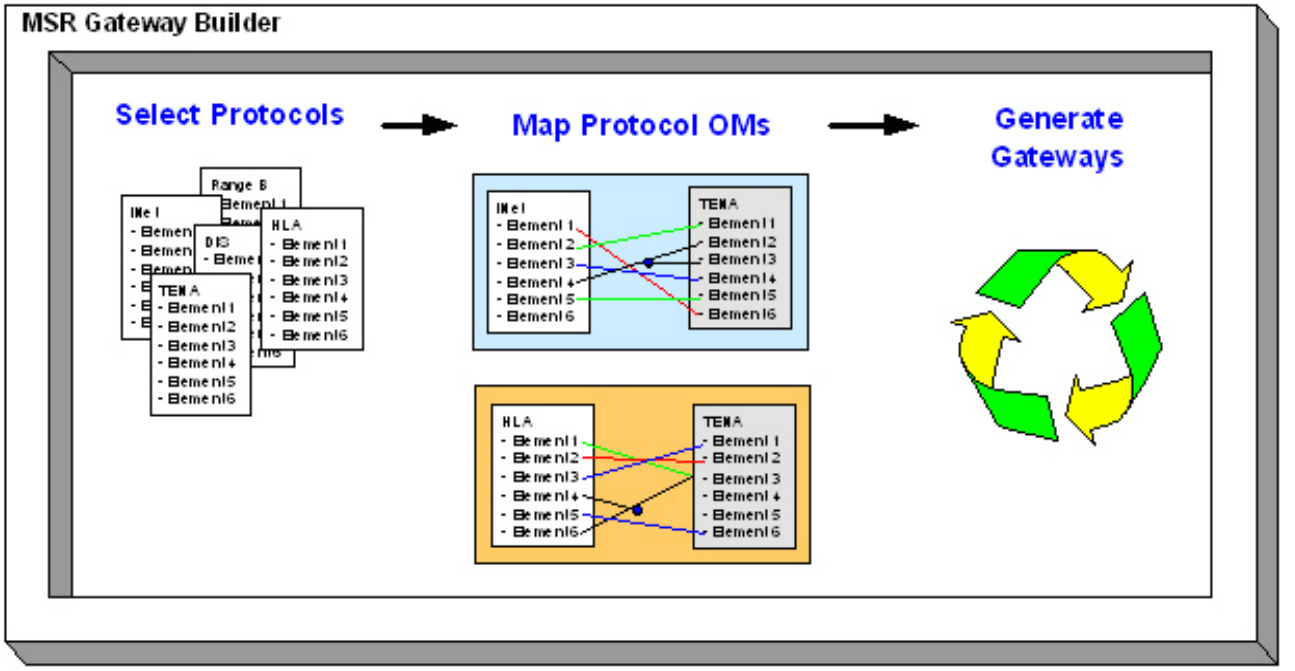
Gateways

- Gateways provide a means of bridging TENA systems to non-TENA systems
- Gateways are TENA applications but may also conform to other architectures
- The most important gateways will bridge TENA to the HLA and to C4ISR systems



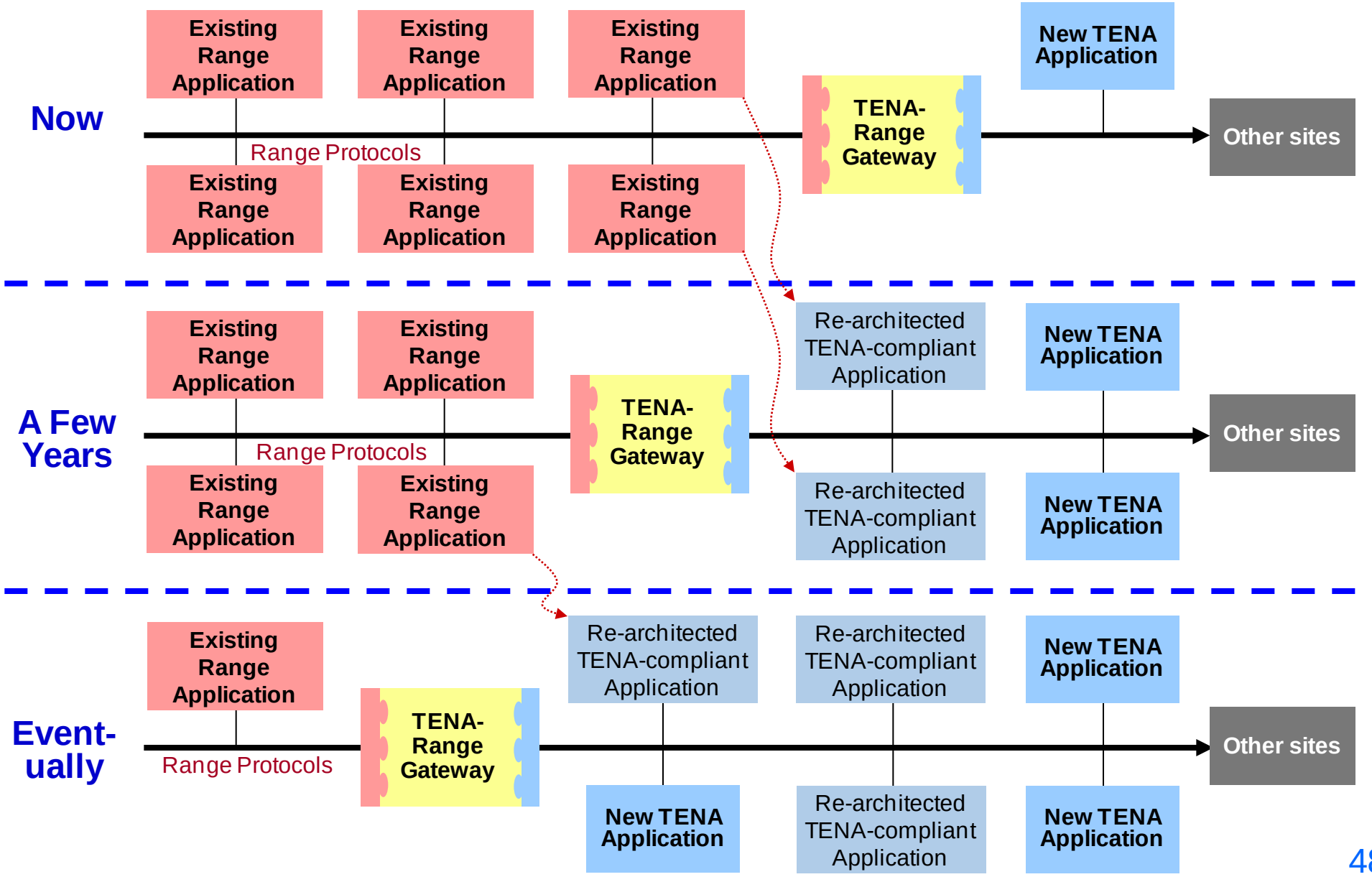
Gateway Builder

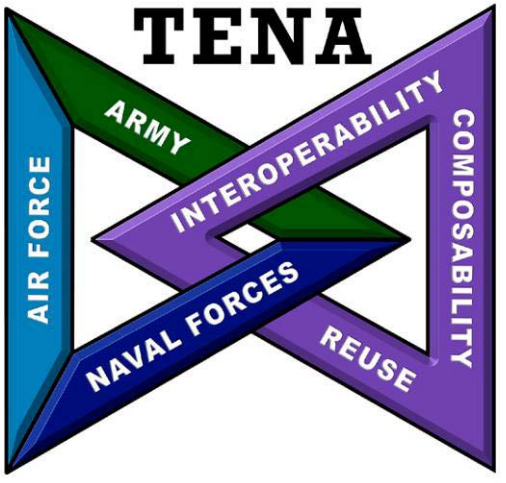
- MSR Program is focused on integration of distributed live, virtual, and constructive (LVC) systems into a common synthetic battle space that comprises various simulation protocols, training ranges, live systems and platforms
- Gateway Builder streamlines integration process and reduces time and effort of creating gateways
- Gateway Builder is a flexible, extensible, graphically driven tool that automatically generates gateways to bridge simulation and live protocols
- Gateway Builder supports mappings between TENA, DIS, and HLA and message-based protocols using any object model



Gateway Builder Simplified Block Diagram

Gradual Deployment of TENA

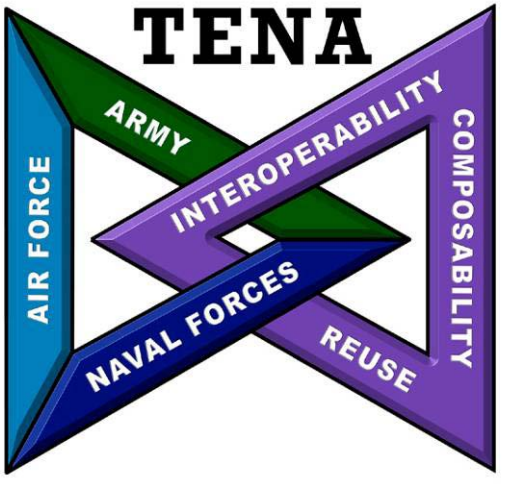



TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE


IMPROVING T&E
CTEIP
CAPABILITIES


UNITED STATES
JOINT FORCES
COMMAND
USJFCOM

Questions?



Stateful Distributed Objects, the TENA Meta-Model, and the TENA Object Model

What is a Meta-Model, and Why is it Important?

- **What is a Meta-Model?**
  - A meta-model is "a model that defines an abstract language for expressing other models," from Common Warehouse Metamodel specification by Dr. Daniel T. Chang.
  - All computer languages have meta-models
  - The TENA Meta-Model describes the features of objects defined in an LROM
- **Why is it important?**
  - The TENA Meta-Model is the architectural construct that specifies both the rules for defining an LROM and the requirements for the middleware

Every Computer Language Has A Meta-Model

(…and They're All Different)

- **C++**
  - Classes, structs == classes, abstract base classes, multiple inheritance, composition, generics, functions, methods, operators, fundamental types, exceptions, arrays, etc.
- **Java**
  - Classes, interfaces, exceptions
  - No structs, no functions, no generics, no multiple inheritance
- **CORBA IDL**
  - Interfaces, structs, valuetypes, sequences, enumerations, multiple inheritance of interfaces, unions
  - No classes
- **HLA**
  - HLA Classes (objects), interactions, attributes, single inheritance
  - No interfaces, no composition, no functions/methods, no …

Requirements for Defining the TENA Meta-Model

- **Must support distributed computing**
- **Must be rich enough in features to support the object modeling needs of the entire test and training range community**
  - Objects and Messages
- **Must provide a semantic unification of information amenable to the creation of a simple, yet powerful, standard TENA Object Model**
- **Must be as easy to use and understand as possible given the above requirements**

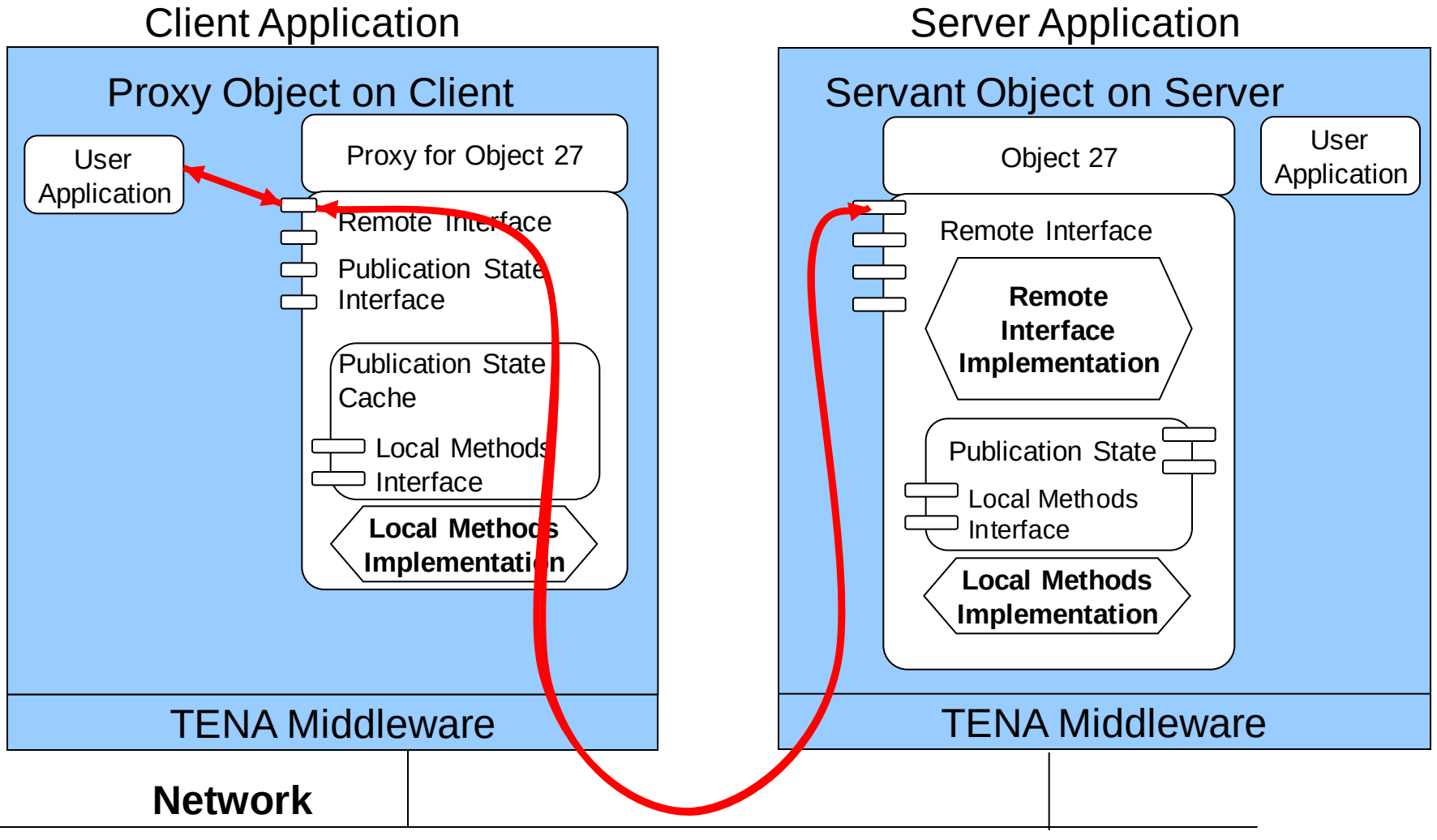
These requirements led to the invention of the Stateful Distributed Object, combining the best features of CORBA and the HLA in one easy-to-use concept

Stateful Distributed Objects

- **An SDO is an object that provides a location-transparent interface to its methods as well as the notion of state**
  - The state of an SDO is data that is disseminated from the creator of an instance of an SDO to all parties that have subscribed to that SDO
  - With an SDO **proxy**, a subscriber can invoke methods on its interface and read the state of the SDO as if it were local data
  - As modifications to a given SDO's state are disseminated from its publisher, subscribers are notified that new values of the state data are available
- **An SDO servant exists only in a single application, in a single process space**
  - This application is called the "**server**" or the "**owner**" of this particular SDO servant
  - There is only one owner application of any particular SDO instance at any one time

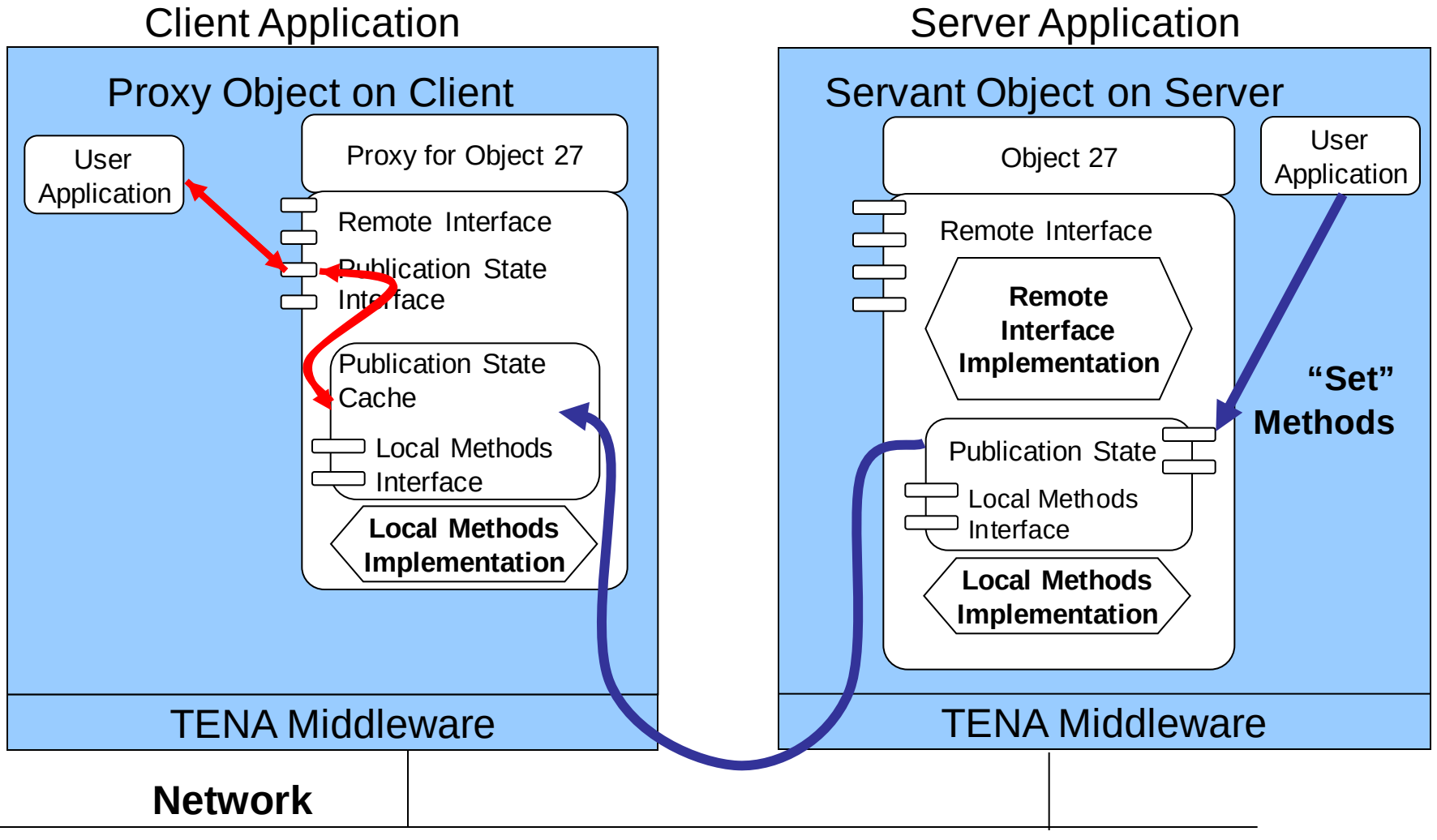
Clients and Proxies, Servers and Servants

- **Remote Method Invocation**
  - Work always performed on the server



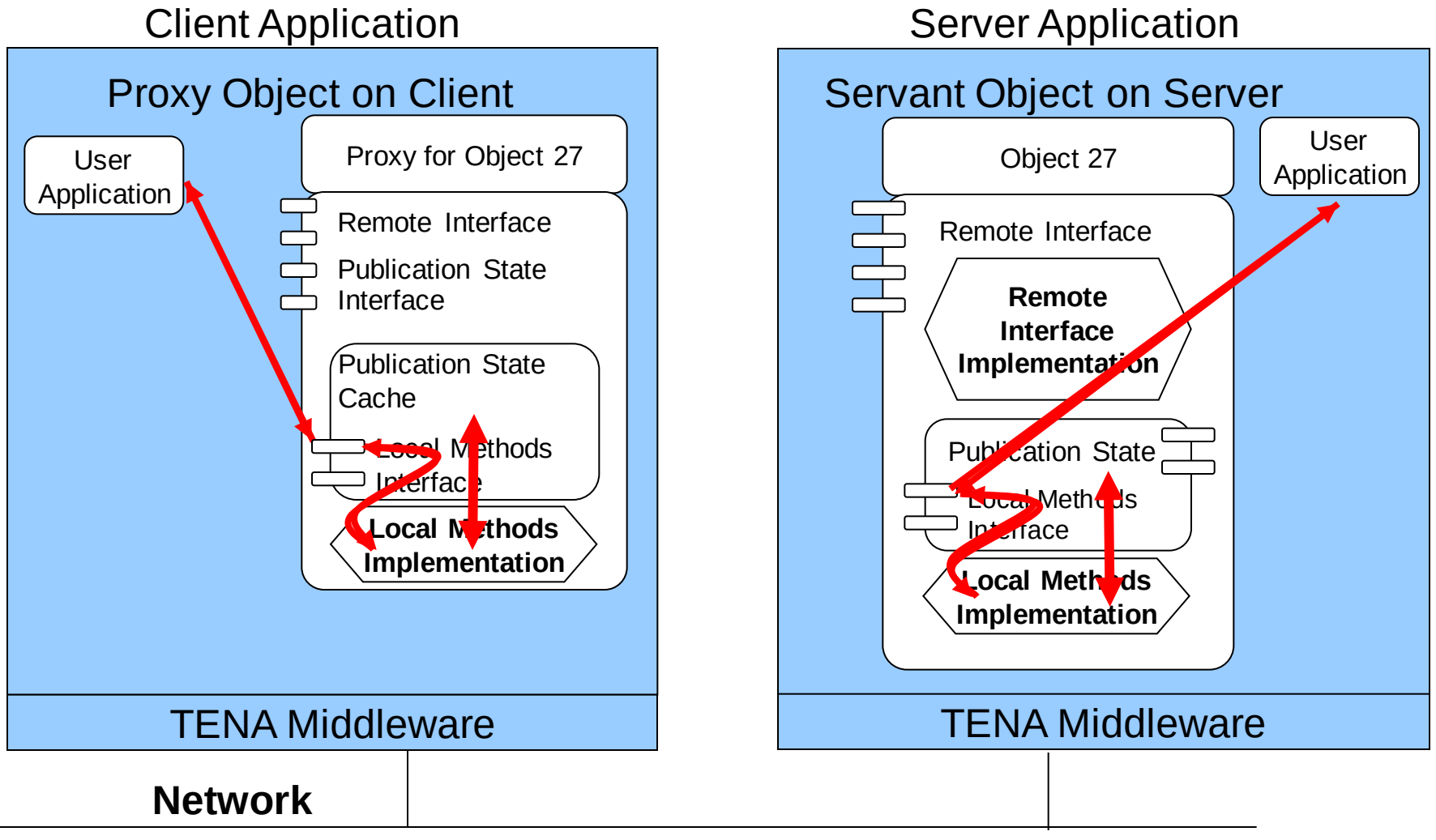
Clients and Proxies, Servers and Servants

- **Publication State Dissemination and Access**



Clients and Proxies, Servers and Servants

- **Local Methods used on both Client and Server**
  - Always performed locally on either client or server

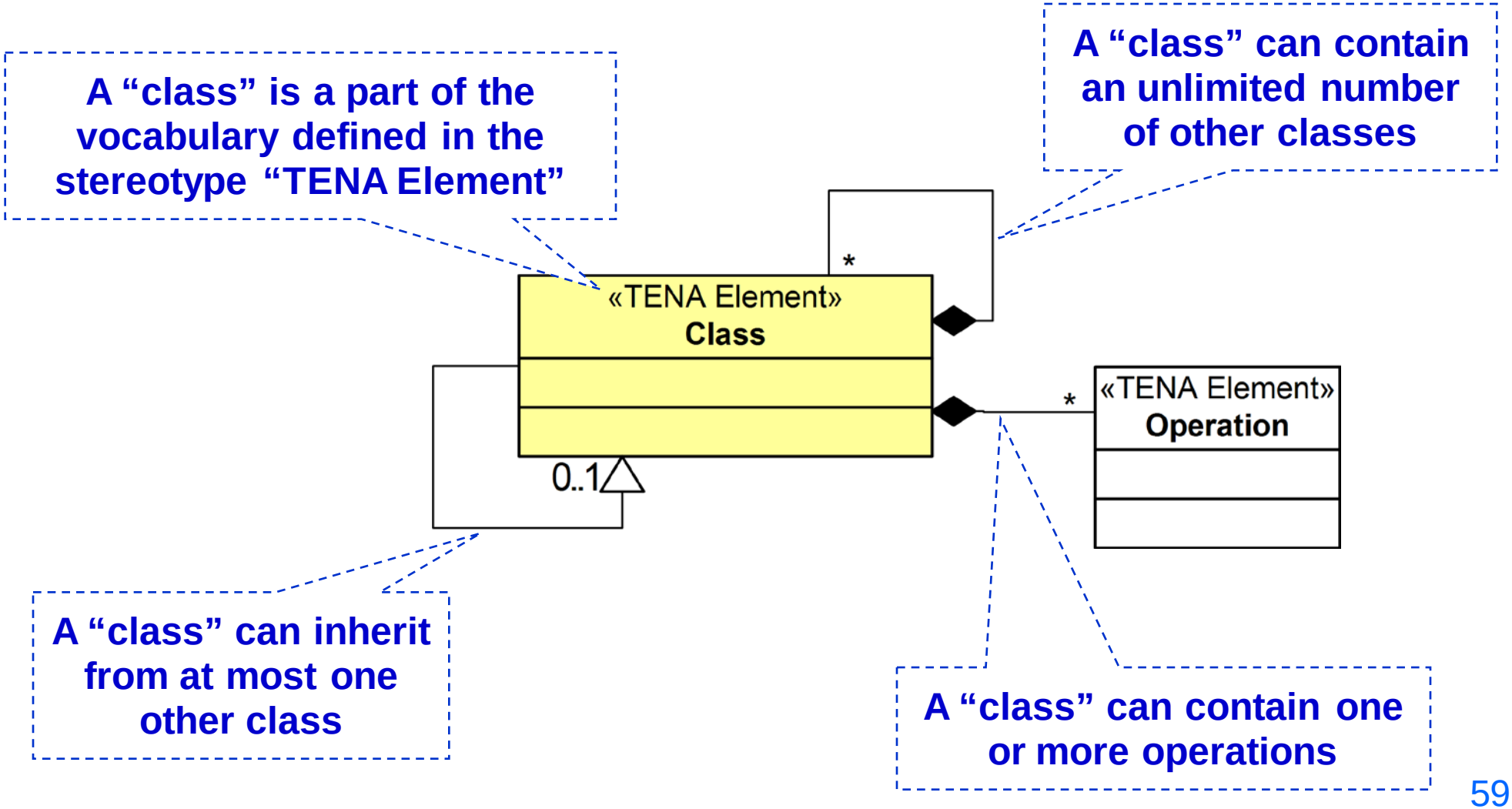


Local Classes

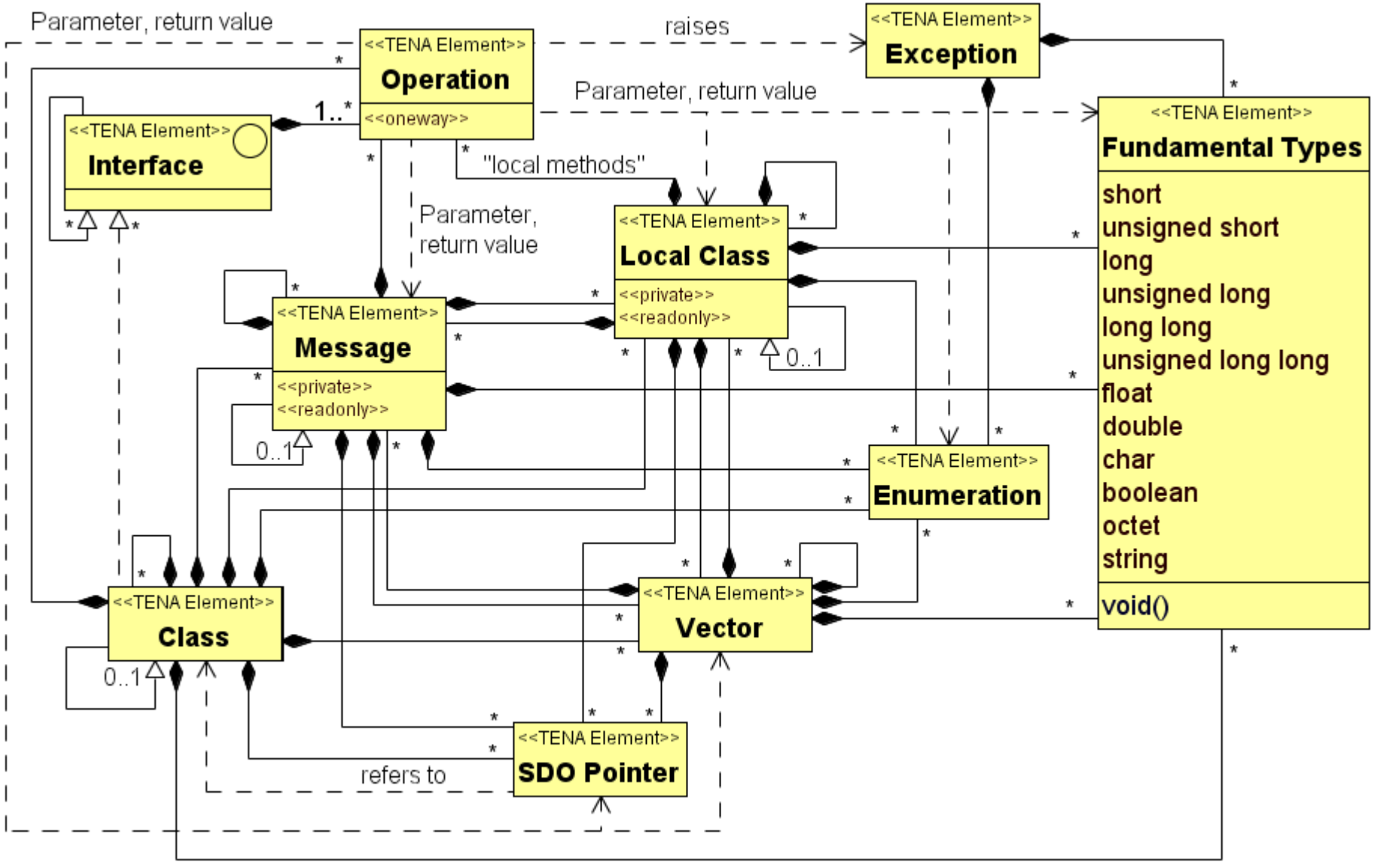
- **The concept of local methods are implemented in what are called "local classes"**
- **Local classes are simply classes that get moved in their entirety (identity, state, and behavior) from servers to clients**
- **Local classes can be contained in SDOs**
- **A "message" is a special type of local class that can be sent from an application to any subscribing applications**
  - Messages can contain other messages as well as contain local classes

Representing a Meta-Model

- **"Pseudo-UML" is used, since formal UML is not as compact or communicative**



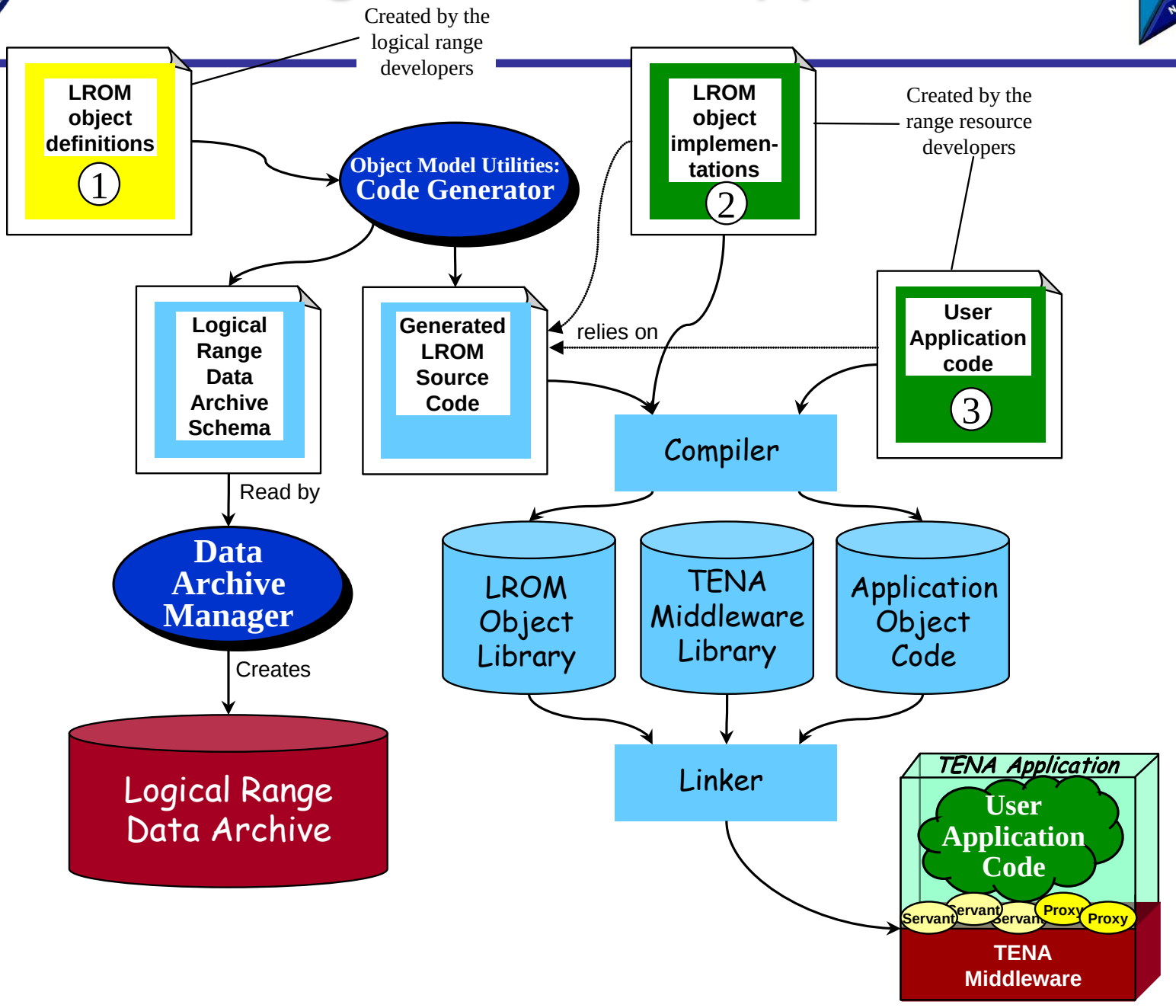
TENA Meta-Model Release 5.2.2



TENA Objects are Compiled In

- **Why use compiled-in object definitions?**
  - **Strong type-checking**
    - Don't wait until runtime to find errors that a compiler could detect
  - **Performance**
    - Interpretation of methods/attributes has significant impact
  - Ability to easily handle complex object relationships
  - Conforms to current best software engineering practices
- **How do you support compiled-in object definitions?**
  - Use a language like CORBA IDL to define object interface and object state structure
  - Use **code generation** to implement the required functionality
- **Thus the concept of the TENA Definition Language (TDL) was created**
  - Very similar to IDL and C++

Creating a TENA Application



The Logical Range Object Model

- **A Logical Range Object Model (LROM) consists of those object definitions, derived from whatever source, that are used in a given logical range execution to meet the immediate needs and requirements of a specific user for a specific range event**
- **The LROM is the common object model shared by all TENA resource applications in a logical range**
- **The concept of an LROM is necessary because it will not be possible to create the entire standard TENA Object Model before the first logical range is created.**
  - As time progresses, each LROM will contain more standard elements and fewer elements that are chosen on an *ad hoc* basis
- **TENA must be deployable *gradually* – the LROM concept supports this requirement**

The Standard TENA Object Model

- **Enables semantic interoperability among range resource applications**
- **Provides the "common language" that all range resource applications use to communicate**
  - It will eventually encode almost all information communicated among range resource applications
- **Object Model Stages**
  - **User-Defined Objects** – objects defined solely for the purpose of a given logical range by TENA users
  - **TENA SDA Supported Objects** – objects developed and supported by the TENA SDA, defined as potential standards, which are undergoing test and evaluation by the community prior to standardization
  - **TENA Standard Objects** – objects developed and supported by the TENA SDA, which have been approved for standardization by the AMT

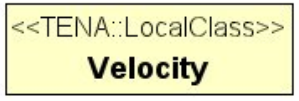
TENA Standard Object Models

- **TENA-Platform:**
  - TENA-Platform-v3.1
  - TENA-PlatformDetails-v3
  - TENA-Affiliation-v1
  - TENA-UniqueID-v2
  - TENA-PlatformType-v1
  - DIS-EntityType-v2
  - TENA-Munition-v2.1
  - TENA-Engagement-v3.1
  - TENA-Organization-v1
  - TENA-EmbeddedSystem-v2
  - TENA-EmbeddedSensor-v2
  - TENA-EmbeddedWeapon-v2
- **TENA-AMO:**
  - TENA-AMO-v1
- **TENA-TSPI:**
  - TENA-TSPI-v4
  - TENA-Time-v1.1
  - TENA-Position-v1
  - TENA-Velocity-v1
  - TENA-Acceleration-v1
  - TENA-Orientation-v1
  - TENA-AngularVelocity-v1
  - TENA-AngularAcceleration-v1
  - TENA-ORM-v1
  - TENA-SRF-v1
  - TENA-SRFserver-v1
- **TENA-Radar-v2**
- **TENA-GPS-v2**

TENA-TSPI-v4

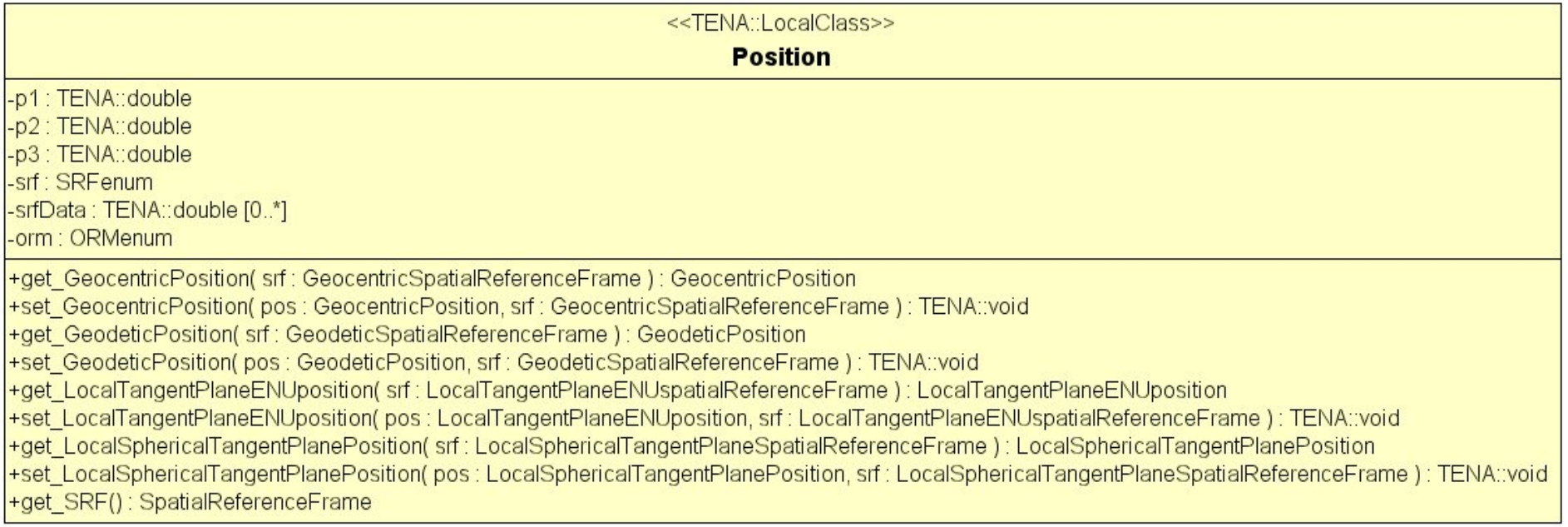

<<TENA::LocalClass>>
TSPI

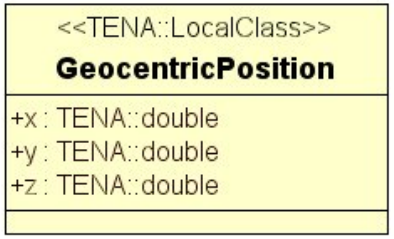

<<TENA::LocalClass>>
Time

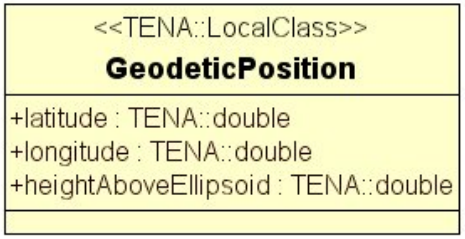

<<TENA::LocalClass>>
Velocity

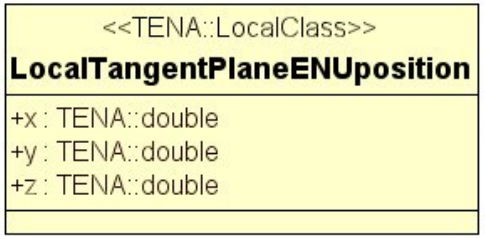

<<TENA::LocalClass>>
Acceleration

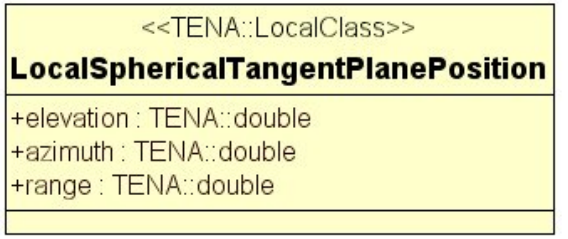

<<TENA::LocalClass>>
Orientation


<<TENA::LocalClass>>
Position
-p1 : TENA::double
-p2 : TENA::double
-p3 : TENA::double
-srf : SRFenum
-srfData : TENA::double [0..*]
-orm : ORMenum
+get_GeocentricPosition(srf : GeocentricSpatialReferenceFrame) : GeocentricPosition
+set_GeocentricPosition(pos : GeocentricPosition, srf : GeocentricSpatialReferenceFrame) : TENA::void
+get_GeodeticPosition(srf : GeodeticSpatialReferenceFrame) : GeodeticPosition
+set_GeodeticPosition(pos : GeodeticPosition, srf : GeodeticSpatialReferenceFrame) : TENA::void
+get_LocalTangentPlaneENUposition(srf : LocalTangentPlaneENUspatialReferenceFrame) : LocalTangentPlaneENUposition
+set_LocalTangentPlaneENUposition(pos : LocalTangentPlaneENUposition, srf : LocalTangentPlaneENUspatialReferenceFrame) : TENA::void
+get_LocalSphericalTangentPlanePosition(srf : LocalSphericalTangentPlaneSpatialReferenceFrame) : LocalSphericalTangentPlanePosition
+set_LocalSphericalTangentPlanePosition(pos : LocalSphericalTangentPlanePosition, srf : LocalSphericalTangentPlaneSpatialReferenceFrame) : TENA::void
+get_SRF() : SpatialReferenceFrame


<<TENA::LocalClass>>
GeocentricPosition
+x : TENA::double
+y : TENA::double
+z : TENA::double

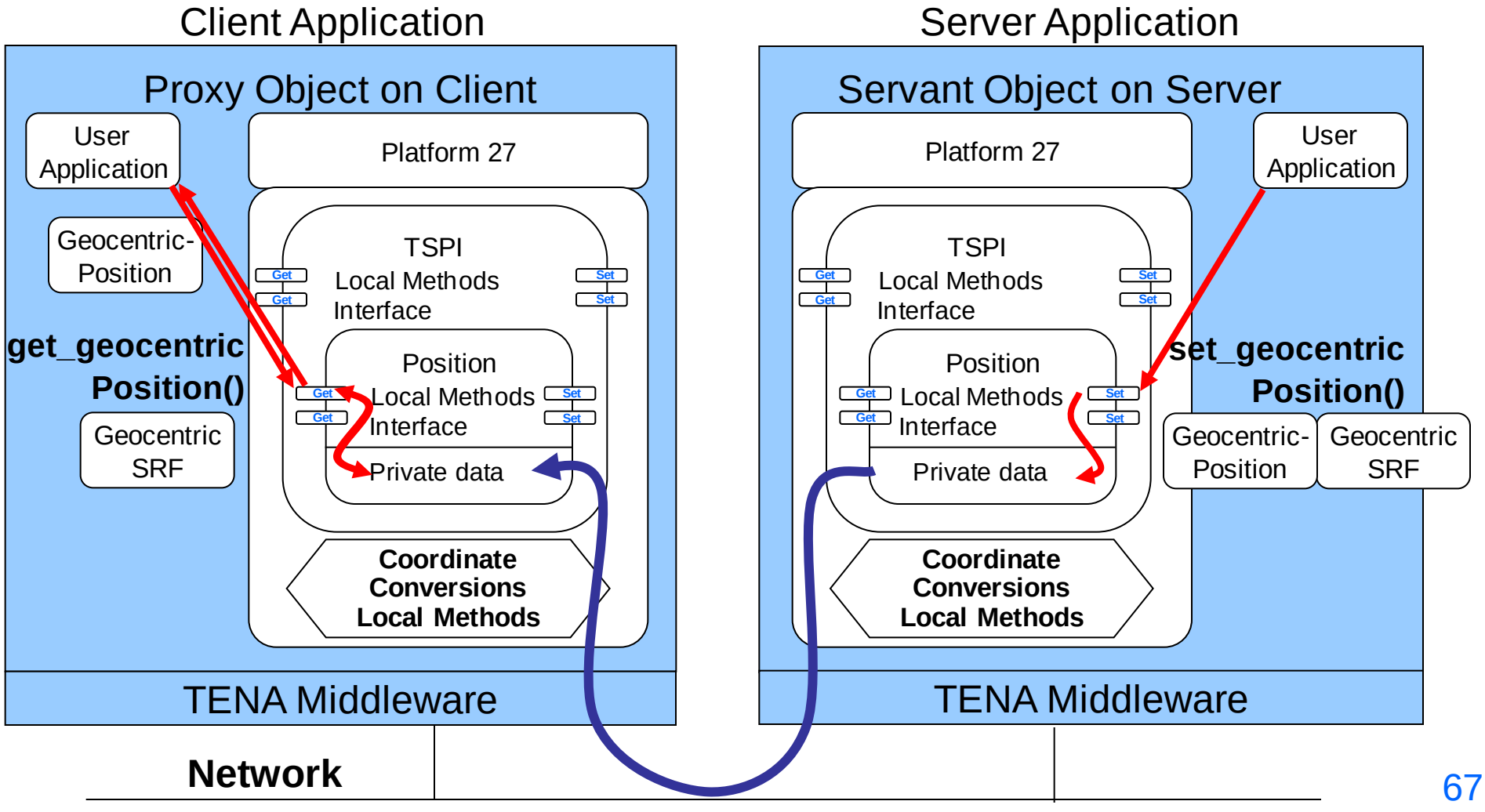

<<TENA::LocalClass>>
GeodeticPosition
+latitude : TENA::double
+longitude : TENA::double
+heightAboveEllipsoid : TENA::double


<<TENA::LocalClass>>
LocalTangentPlaneENUposition
+x : TENA::double
+y : TENA::double
+z : TENA::double


<<TENA::LocalClass>>
LocalSphericalTangentPlanePosition
+elevation : TENA::double
+azimuth : TENA::double
+range : TENA::double

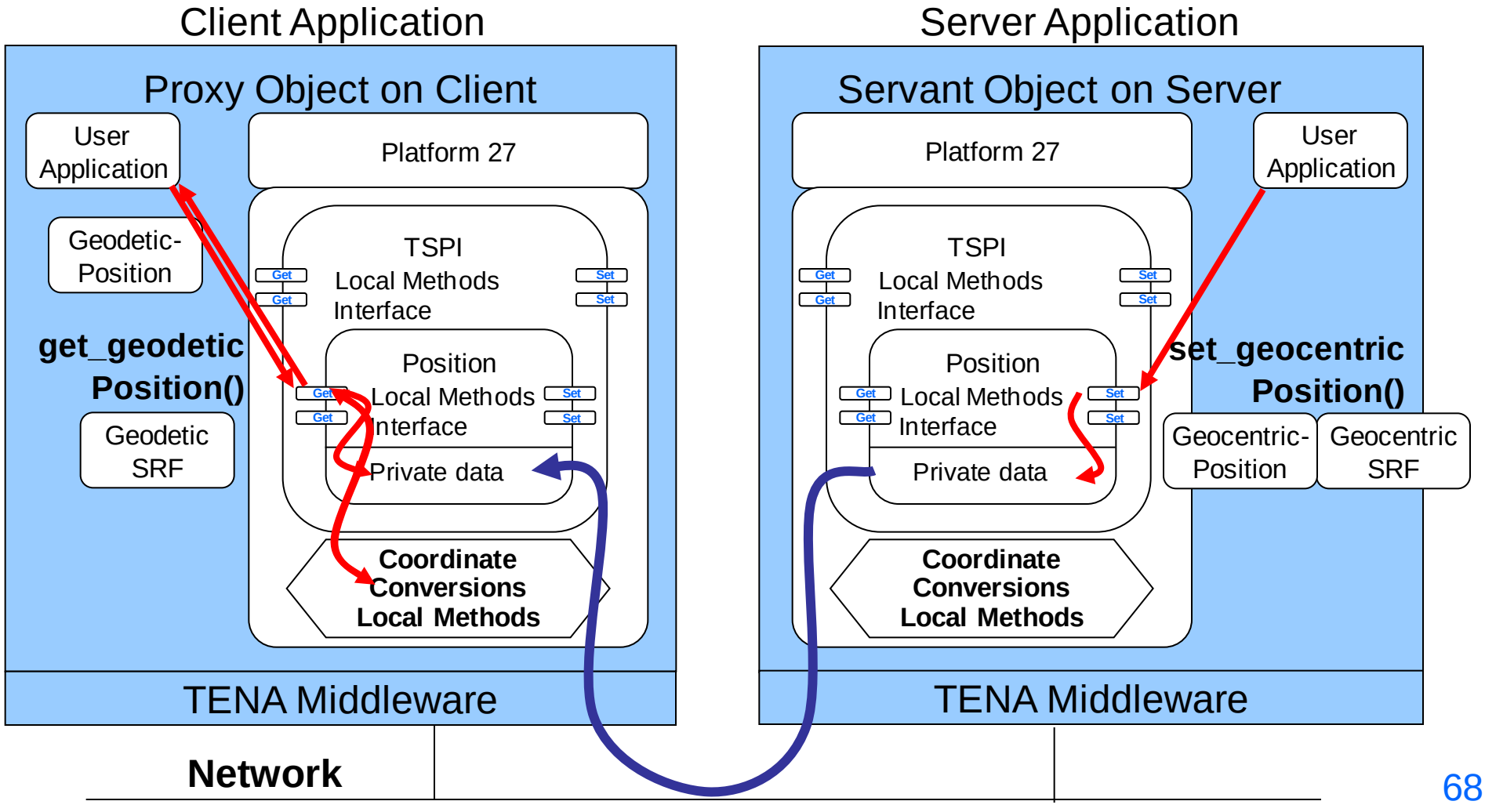
TSPI v4 with Coordinate Conversions

- **Case 1: Reading and writing in the same coordinate system**

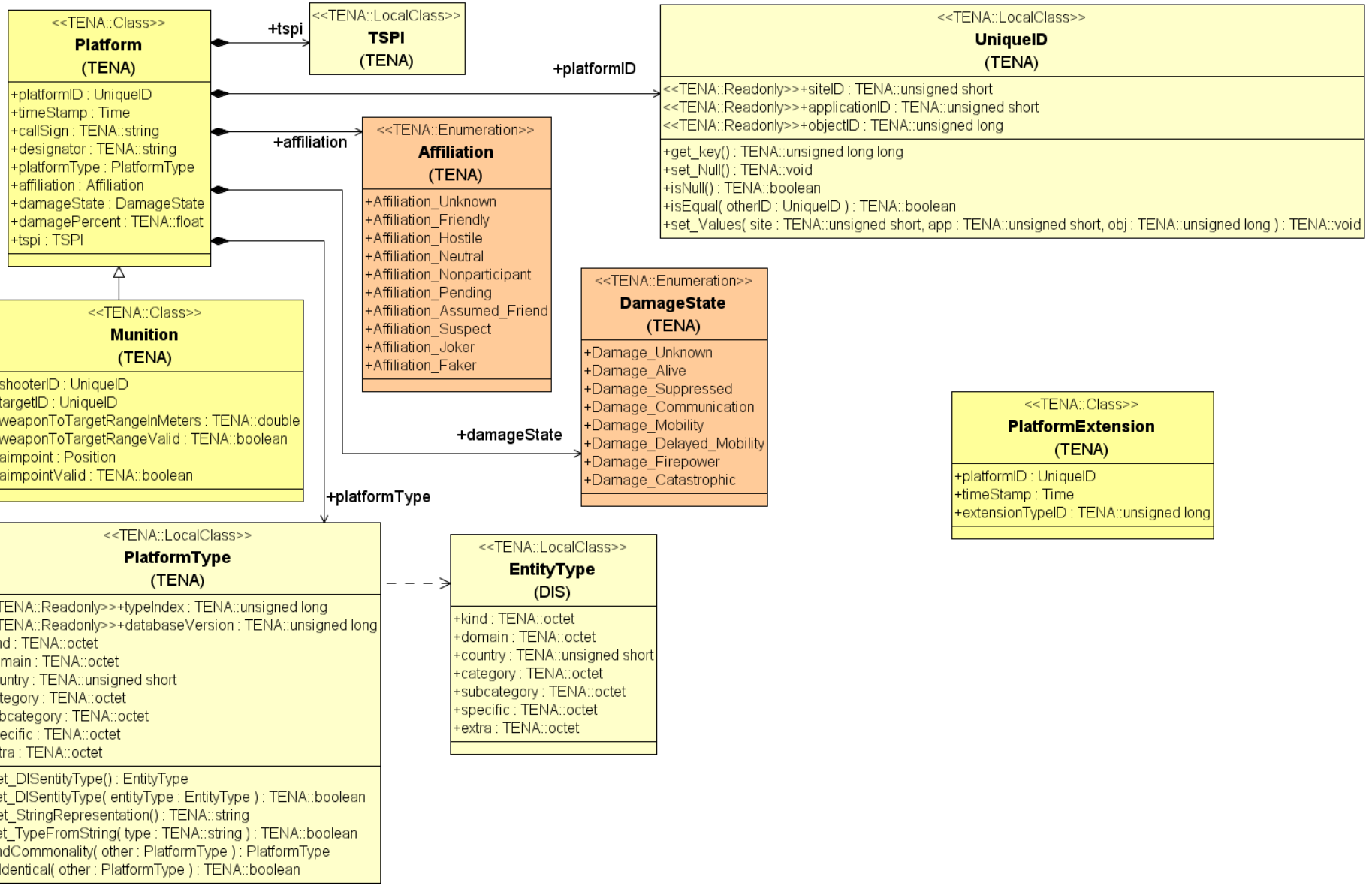


TSPI v4 with Coordinate Conversions

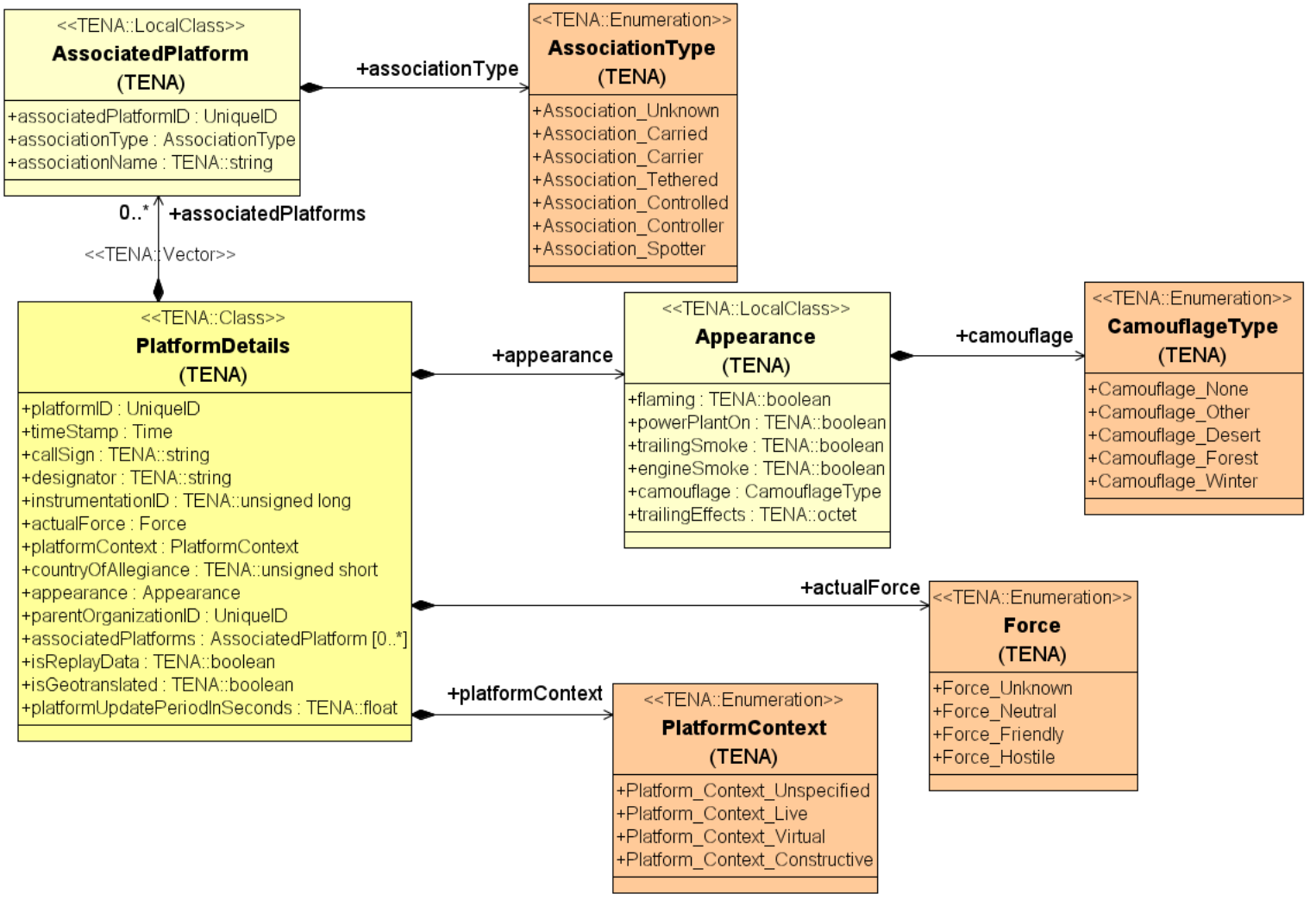
- **Case 2: Reading and writing in different coordinate systems**
  - Write in Geocentric (ECEF), read in Geodetic (latitude/longitude/altitude)



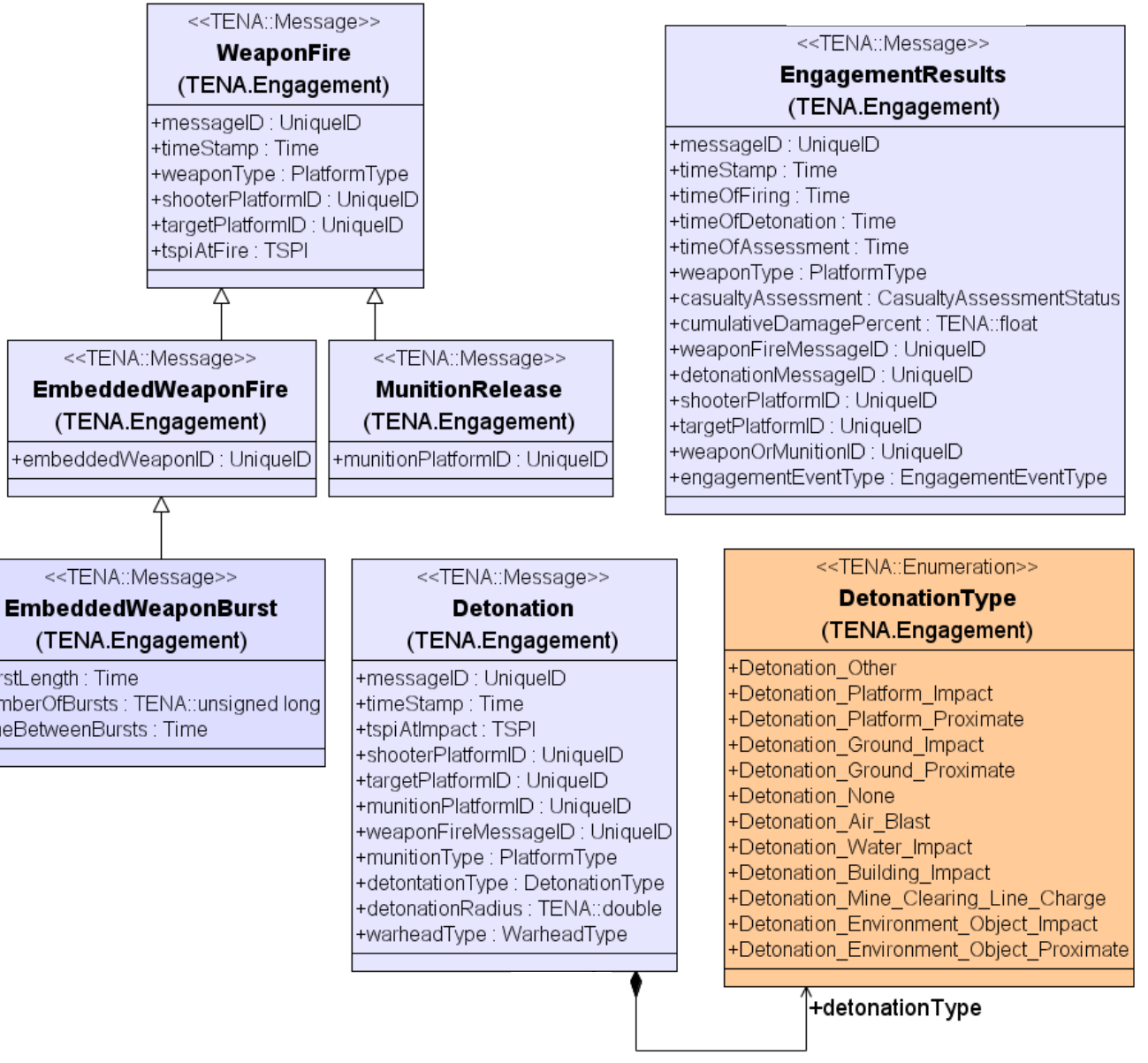
TENA-Platform-v3.1



TENA-PlatformDetails-v3



TENA-Engagement-v3.1



Web-Based Code Generation

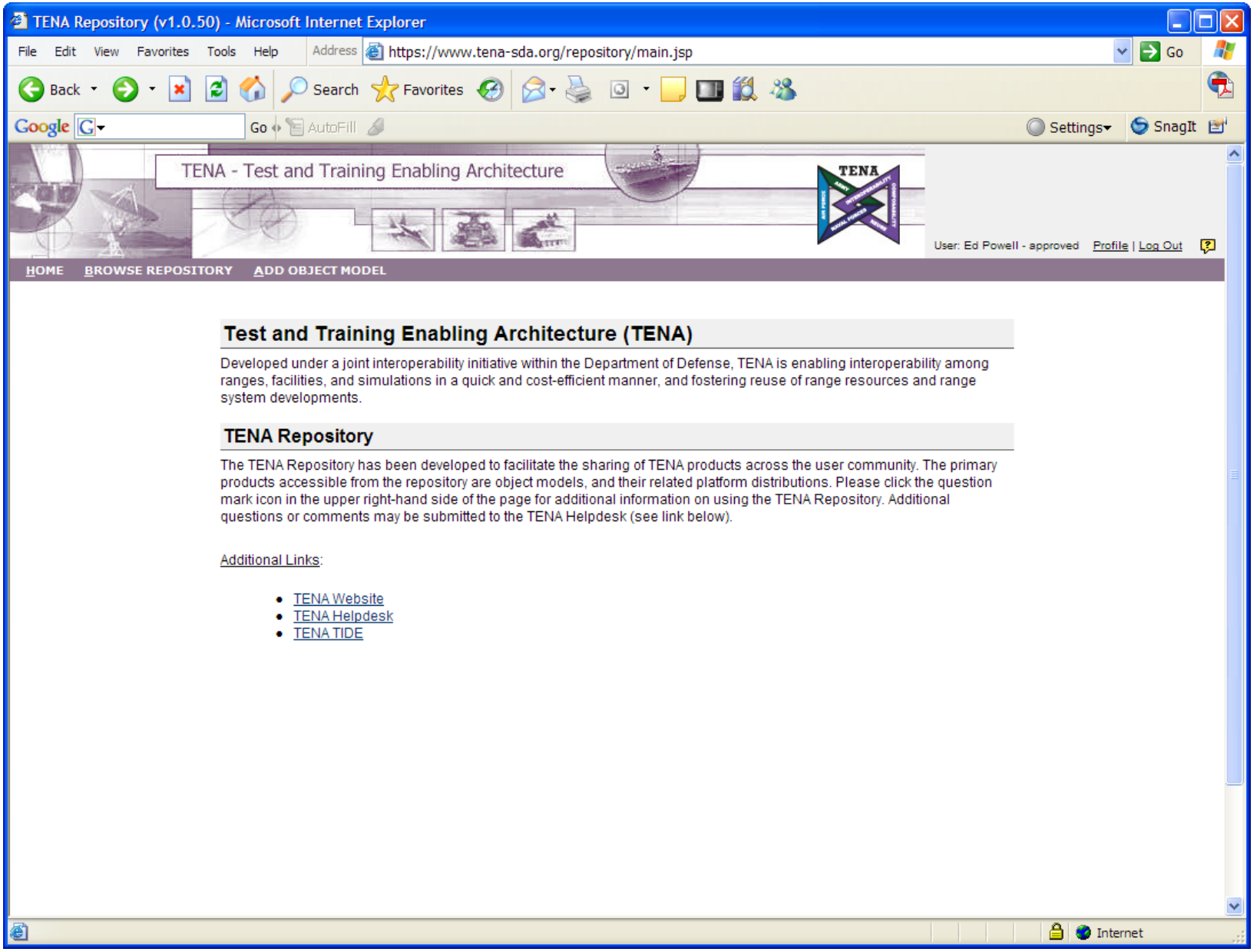
- **TDL-to-C++ compiler uses a Web front end, because it:**
  - Allows bug fixes and additions to the code generator without having to re-disseminate it to the community
  - Allows AMT to collect information on object models being designed so progress can be made toward the standard TENA Object Model
  - Allows collaboration with users on their object model designs
  - Allows code generator to be written for less than the full complement of TENA Middleware platforms, if necessary

Object Model Distributions

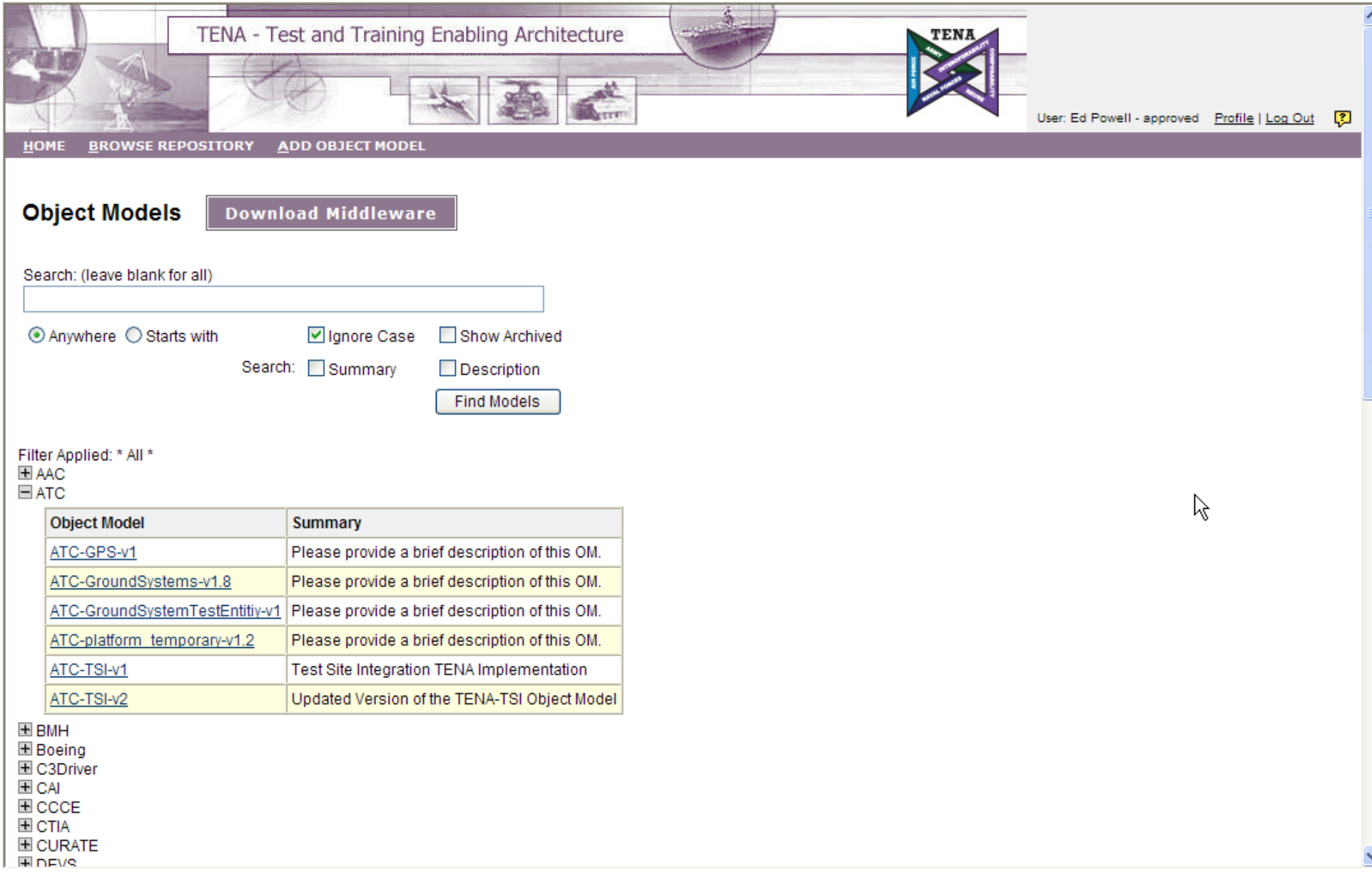
- **Two Types of Object Model Distributions**
  - Object Model Definition – specifies the types (e.g., classes, messages, enums) and their interface signatures and/or attributes
  - Object Model Implementation – Provides executable code that adheres to a particular definition
- **Object Model Components**
  - Object model definitions can "import" other definitions
  - Applications are required to install every object model definition and any pre-built implementations being used
  - Namespace changes with pre-built implementations complicates the automatic generation of "BasicImpl" applications
- **OM Distribution Bundles**
  - Currently developed mechanism for TENA Repository to bundle imported definitions and available implementations into a single downloadable file
  - Need to expand on this capability to automatically install all of the individual components

Web Site OM Support

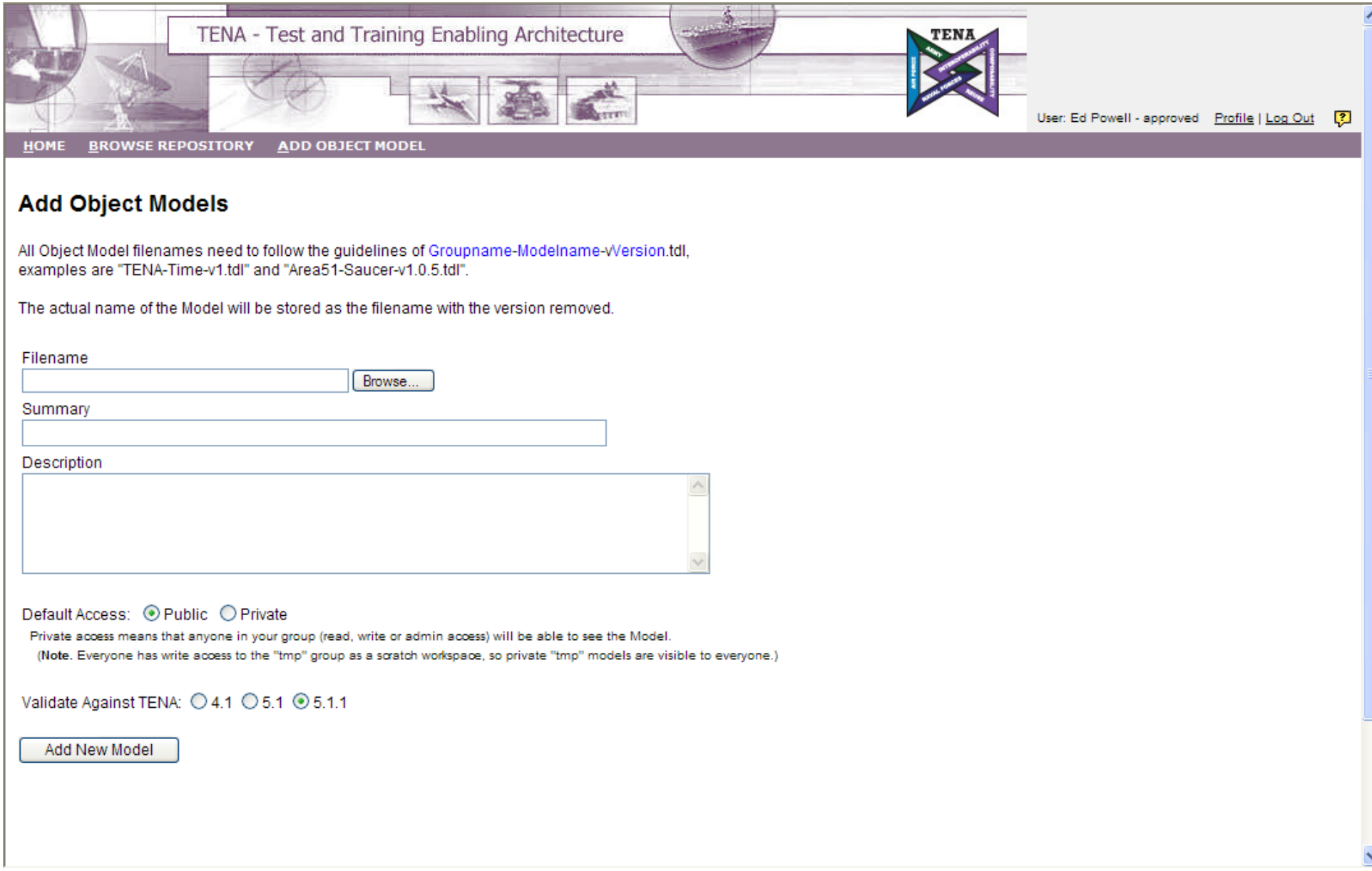
http://www.tena-sda.org/repository



Browse Repository



Upload TDL Files



Download Model Definition

Model Details

Model Name:	DIS-EntityType-v2
Access:	Public
Summary:	Version 2 of the DIS EntityType object model.
Description:	This is version 2 of the DIS EntityType object model developed by the TENA SDA project. This object model is based on the DIS standard, IEEE-1278.x. More detailed documentation of this object model is available in the TENA Wiki.
TDL File:	models/DIS/EntityType-v2/DIS-EntityType-v2.tdl View TDL
Uploaded by:	J. Russell Noseworthy on 09/23/2005 15:43:41

⊞ **Attachments**

Imports - *none*

Distributions: TENA-v5.1.1

⊞ *Select visible platforms*

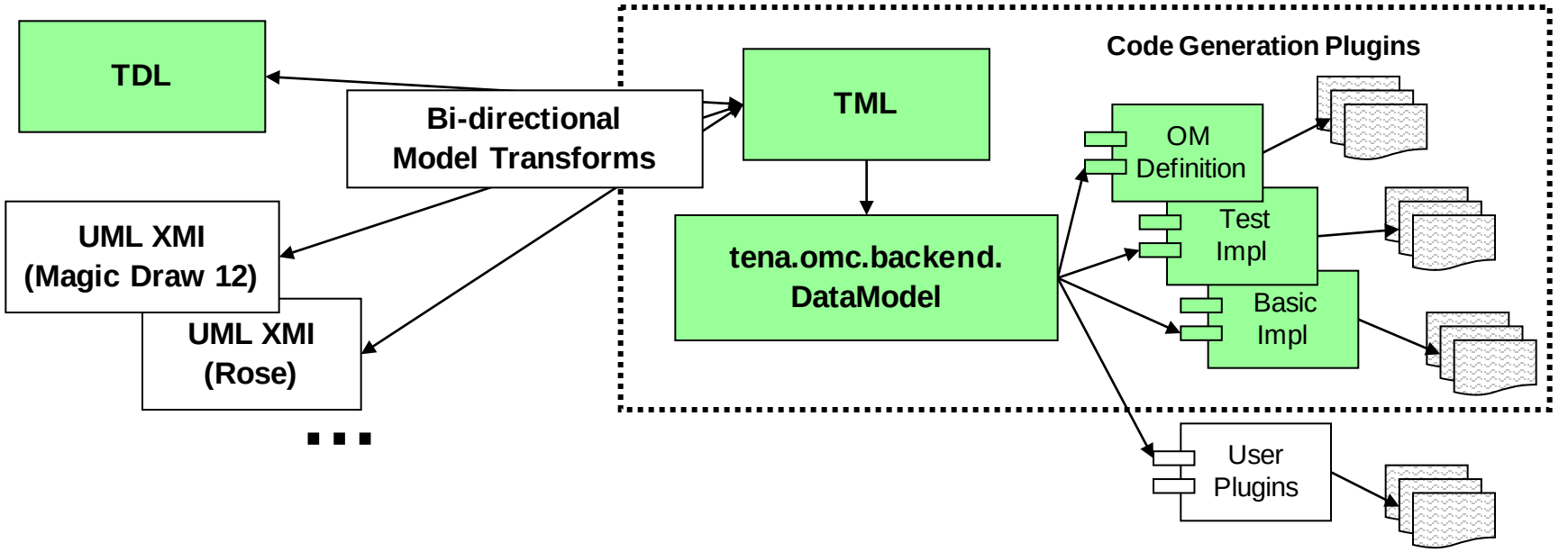
Distribution	Distribution State	Platform
Definition	Download (279.921KB) details	fc3-gcc34-d
Definition	Download (512.101KB) details	fc3-gcc34
Definition	Download (282.036KB) details	fc4-gcc40-d
Definition	Download (486.919KB) details	fc4-gcc40
Definition	Download (225.12KB) details	irix65-mipspro742m-d
Definition	Download (179.097KB) details	irix65-mipspro742m
Definition	Download (282.517KB) details	rhelws4-gcc34-d
Definition	Download (514.784KB) details	rhelws4-gcc34
Definition	Download (1.144MB) details	rh8-gcc32-d

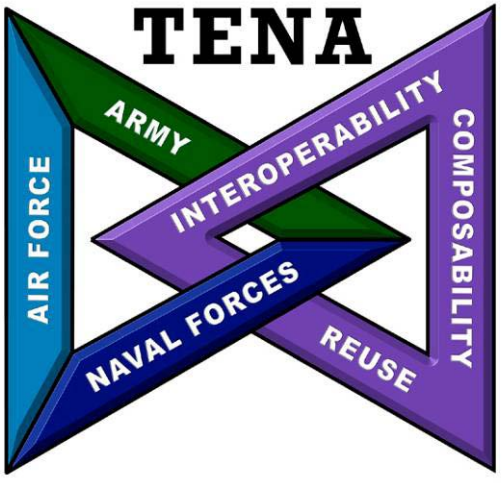
Remember: Need to Download Definition and Implementation

Distribution	Distribution State	Platform
Definition	Download (1.126MB) details	fc3-gcc34-d
Implementation Src	Download (26.322KB) details	fc3-gcc34-d
Definition	Download (1.395MB) details	fc3-gcc34
Implementation Src	Download (26.977KB) details	fc3-gcc34
Definition	Download (1.165MB) details	fc4-gcc40-d
Implementation Src	Download (27.82KB) details	fc4-gcc40-d
Definition	Download (1.388MB) details	fc4-gcc40
Implementation Src	Download (28.199KB) details	fc4-gcc40
Definition	Download (862.533KB) details	irix65-mipspro742m-d
Implementation Src	Download (26.621KB) details	irix65-mipspro742m-d
Definition	Download (554.146KB) details	irix65-mipspro742m
Implementation Src	Download (26.599KB) details	irix65-mipspro742m
Definition	Download (1.129MB) details	rhelws4-gcc34-d
Implementation Src	Download (27.309KB) details	rhelws4-gcc34-d
Definition	Download (1.399MB) details	rhelws4-gcc34
Implementation Src	Download (30.263KB) details	rhelws4-gcc34
Definition	Download (2.587MB) details	rh8-gcc32-d
Implementation Src	Download (26.175KB) details	rh8-gcc32-d
Definition	Download (2.664MB) details	rh8-gcc32
Implementation Src	Download (26.845KB) details	rh8-gcc32
Definition	Download (2.584MB) details	rh9-gcc322-d
Implementation Src	Download (26.165KB) details	rh9-gcc322-d
Definition	Download (2.661MB) details	rh9-gcc322
Implementation Src	Download (26.822KB) details	rh9-gcc322
Definition	Download (319.866KB) details	rh9-ppc82xxgcc322
Implementation Src	Download (26.506KB) details	rh9-ppc82xxgcc322

Future Auto-code Generation With TENA

- **Our desire is for the input to the TENA auto-code generator be standard XMI (generated from UML)**
- **Challenges: XMI not yet implemented in a standard way by tool vendors, and current auto-code generation capability is based on TDL**
- **Current Interim Solution – Use MagicDraw plug-in to create TDL from UML**
- **Next Steps**
  - Implement TENA Metamodel in Eclipse Modeling Framework using ECore representation – define TENA Modeling Language (TML)
  - Create XMI ↔ TML, TDL ↔ TML translators
  - API and framework being developed to support various "code generation plugins" used to automatically create specialized code based on FreeMarker templates





Summary So Far

TENA Solutions to Interoperability Challenges

- **On-the-Wire Specification vs. API Standard**
  - API Standard allows future technological advances for data transmission to be much more cost-effectively incorporated
- **Single Reference Frame vs. Multiple Reference Frames**
  - Multiple Reference Frames allow different range systems to operate in the coordinate system most optimum for their range
- **Single Level vs. Multiple Levels of Compliancy**
  - Multiple Levels of Compliancy allow a more meaningful definition of compliancy to be used among Range engineers & investment managers
- **Run-Time Interpreter vs. Compile-Time Integration**
  - Compile-Time Integration allows for inconsistencies to be discovered when the software is being upgraded vice during the event
- **Hand-Coded vs. Auto-Code-Generated Interfaces**
  - Auto-Code-Generated Interfaces can be produced more reliably and tremendously faster than traditional hand-coded interfaces
- **Centralized (Client/Server) vs. Peer-to-Peer**
  - Peer-to-Peer gives more flexibility to exercise designers – can emulate client/server if necessary

DoD Directive on TENA

Business Initiative Council TE-09
Common Test and Training Range Architecture Policy (CTTRAP)

- **Leverages lessons learned from past directives including Ada, HLA, and JTRS**
- **Establishes a flexible process where the Services make the final determination on TENA compliancy for their systems on a case-by-case basis**
  - TENA compliancy must not adversely impact cost, schedule, or performance of the individual range system
- **All new range systems will be required to use TENA**
- **All existing range systems that are having their data distribution mechanism upgraded will be required to use TENA**
- **The Directive applies if the current version of TENA satisfies the interoperability requirements of the new or upgraded range system. If not, the interoperability requirements for the new system will be identified so the appropriate upgrades to TENA can be made by CTEIP**
- **OSD(P&R) and DTRMC will oversee the sustainment of TENA**

Summary of What We Have

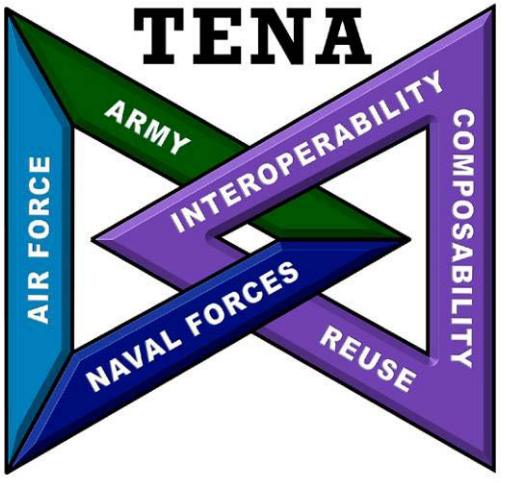
An Architecture for Ranges, Facilities, and Simulations to Interoperate, to be Reused, to be Composed into greater capabilities

- **A Working Implementation of the Architecture**
  - **TENA Middleware currently works on Windows, Linux, and Sun**
- **A Process to Develop and Expand the Architecture**
  - **CTTRA Workshops and AMT Meetings**
- **A Technical Strategy to Deploy the Architecture**
  - **Gateways provide interim solutions as TENA interfaces**
- **A Definition of Compliancy**
  - **Levels of compliancy to enhance communication among systems engineers and investment decision makers**

Important Contact Information

- **Project Website: http://www.tena-sda.org**
  - Download TENA Middleware
  - Submit Helpdesk Case (http://www.tena-sda.org/helpdesk)
    - Use for all questions about the Middleware

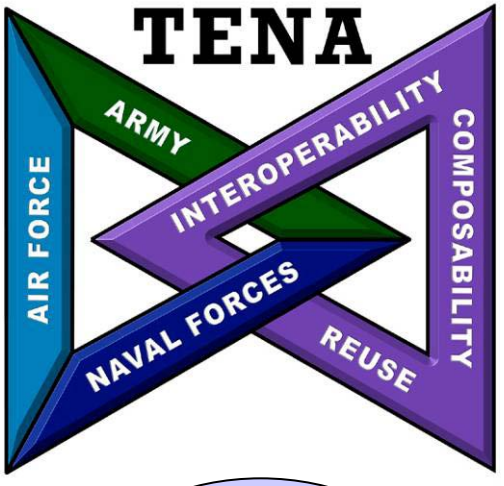
- **TENA Feedback: feedback@tena-sda.org**
  - Provide technical feedback on TENA Architecture or Middleware
  - Ask technical questions regarding the TENA architecture or project
  - Provide responses to AMT action items
  - Request TENA training

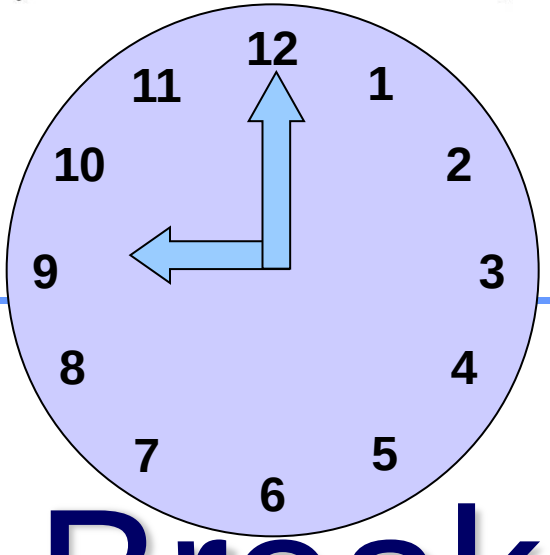

TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE

Questions?

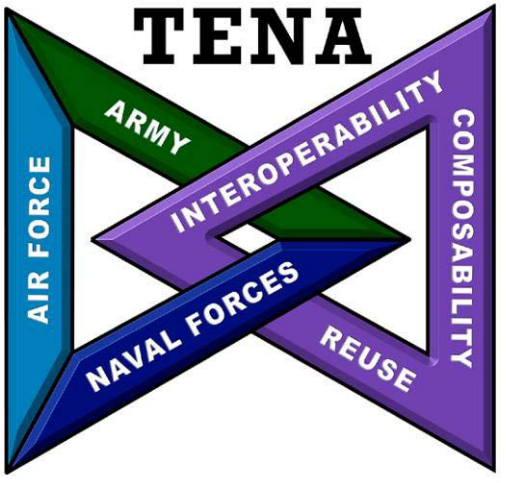

IMPROVING T&E
CTEIP
CAPABILITIES


UNITED STATES
JOINT FORCES
COMMAND
USJFCOM


TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE


12
1
2
3
4
5
6
7
8
9
10
11

Break



TIDE 1.1 Overview

TIDE-Based TENA Development Process

TENA Middleware Developer

Publish TENA Middleware Distributions

TENA Repository

Download Object Models

Publish Object Models

Download Object Models, Middleware

Publish OM Implementations, Applications

TIDE

Develop and Document Object Models

Object Model Developer

TIDE

Develop and Document OM Implementations, Applications

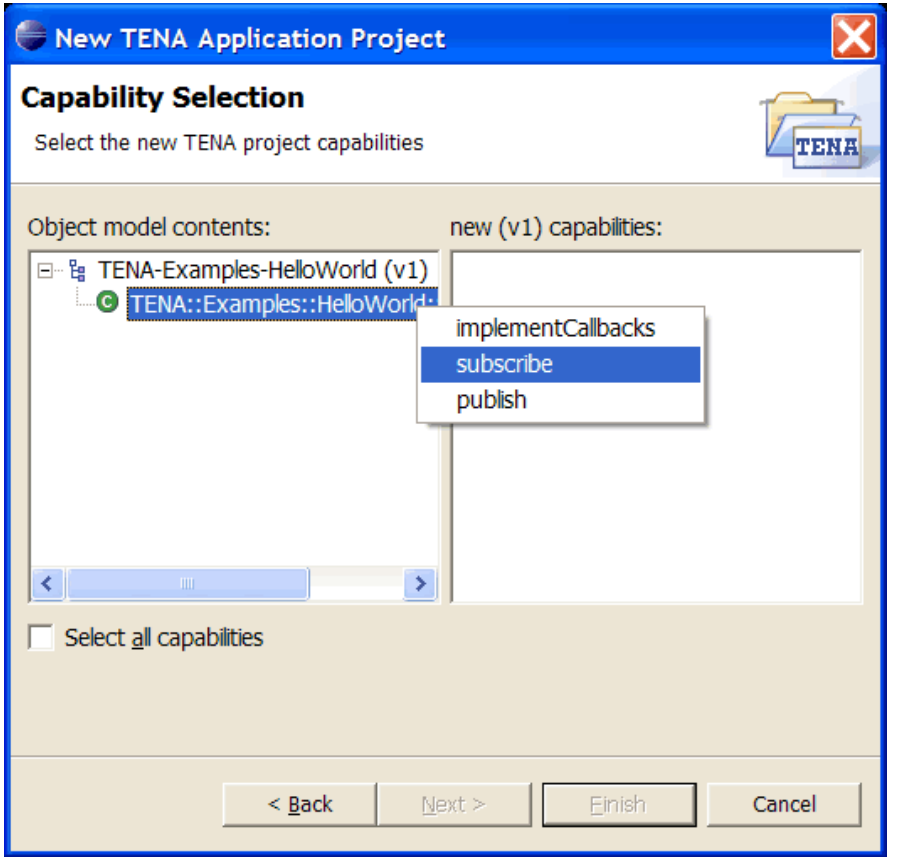
Application Developer

TIDE 1.1 Features

- Customizable TENA Projects – NEW!
- CDT Integration – IMPROVED!
- TENA Repository Exploring
- Installing/Uninstalling TENA Middleware
- Installing/Uninstalling and Requesting Object Models
- Installing/Uninstalling Implementations
- Creating TENA Projects – IMPROVED!
- Importing Existing TENA Projects
- Configuring TENA Projects – NEW!
- Building TENA Projects
- Comparing TENA Projects with their Pristine Copy
- Migrating TENA Projects between middleware releases
- Migrating TENA Projects between object model releases

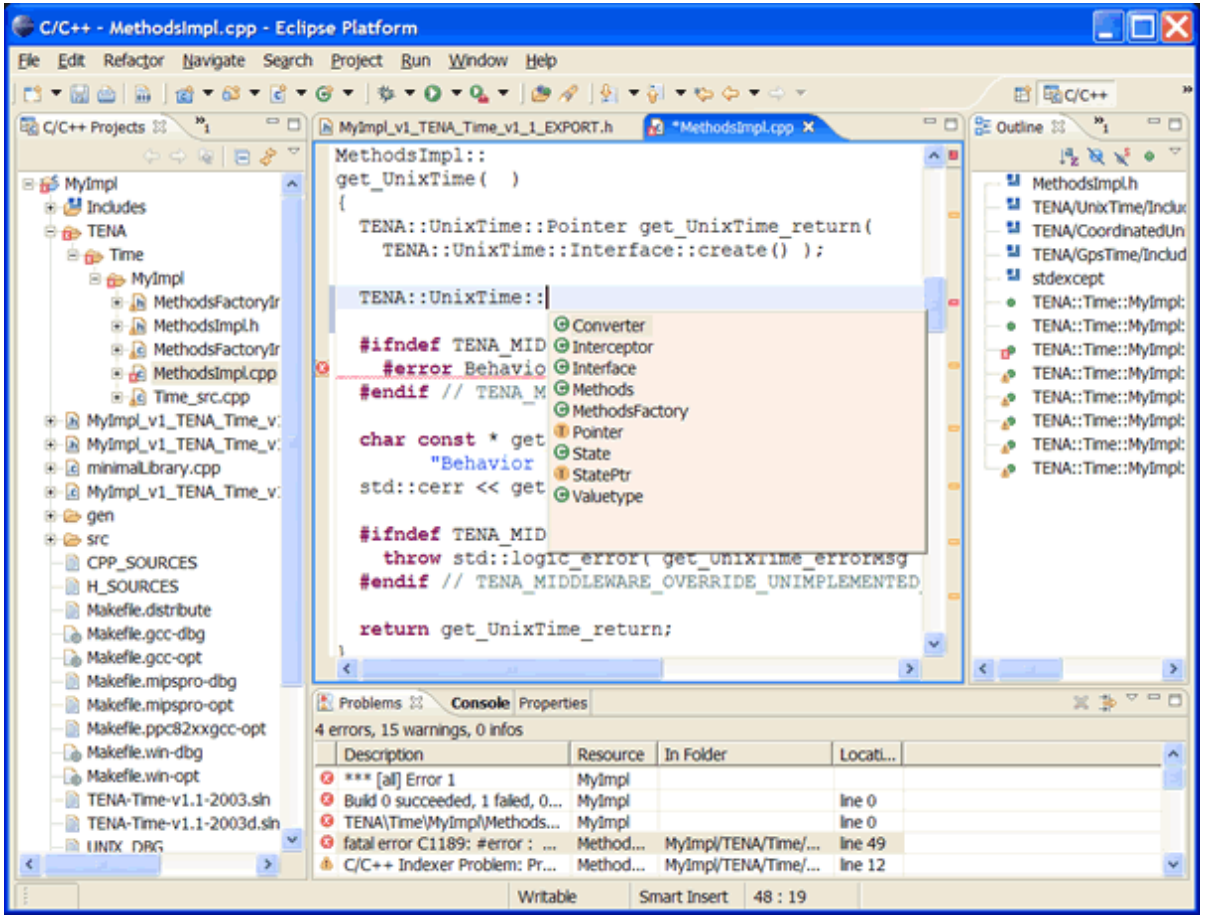
Customizable TENA Projects

- **Allows users to generate code based on specific capabilities (e.g., publish SDO Munition, subscribe to message Detonation, etc.) instead of all possible capabilities (i.e., publish and subscribe all SDOs and messages)**



CDT Integration

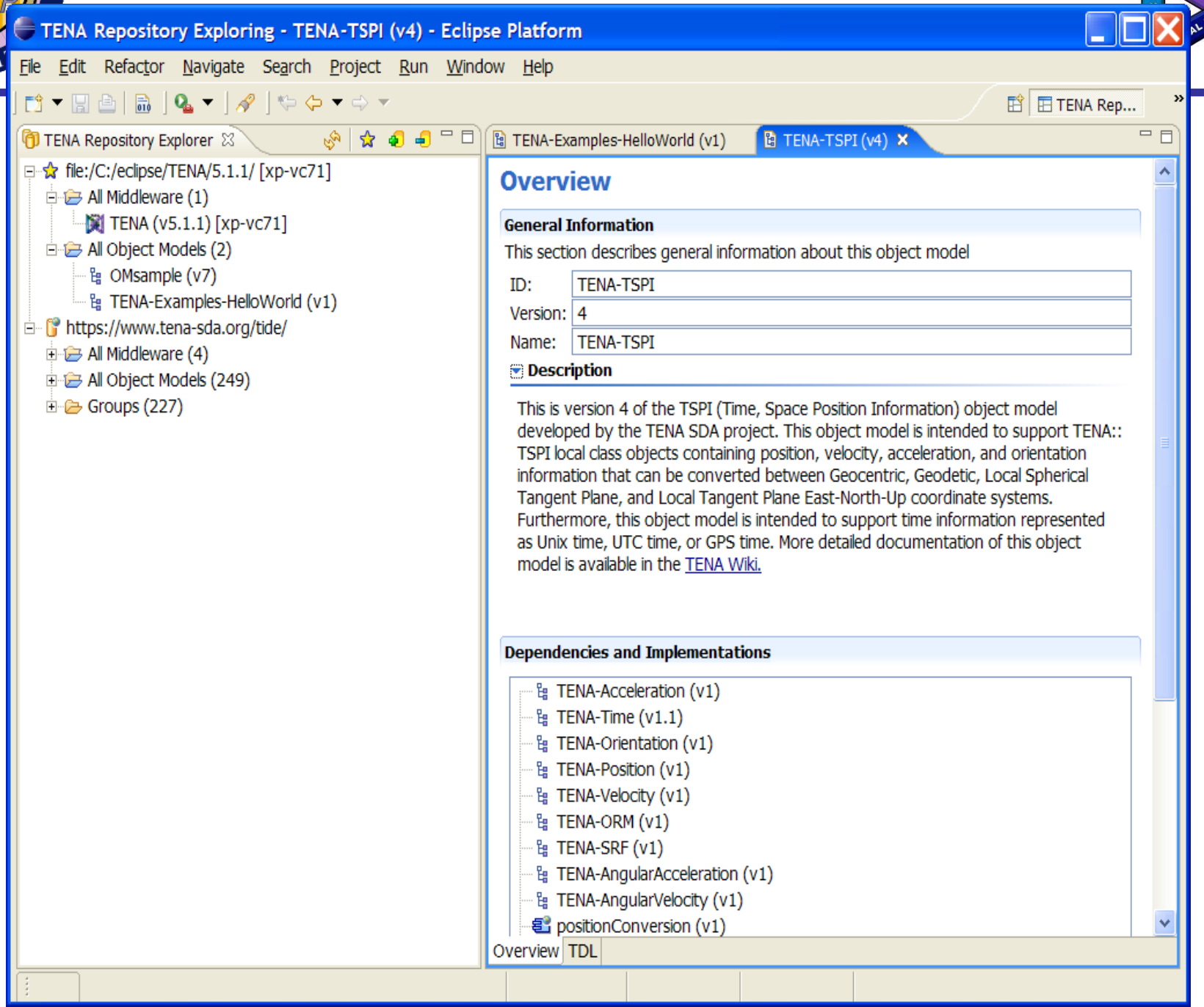
- **TENA Projects are now CDT (http://www.eclipse.org/cdt/) compliant C++ projects. This enables CDT features such as code completion, indexing, cross referencing, etc.**



TENA Repository Exploring

- **Allows users to browse locally installed TENA middleware, object models and implementations**
- **Allows users to browse remote repositories (e.g., https://www.tena-sda.org/tide/) for TENA middleware, object models and implementations**

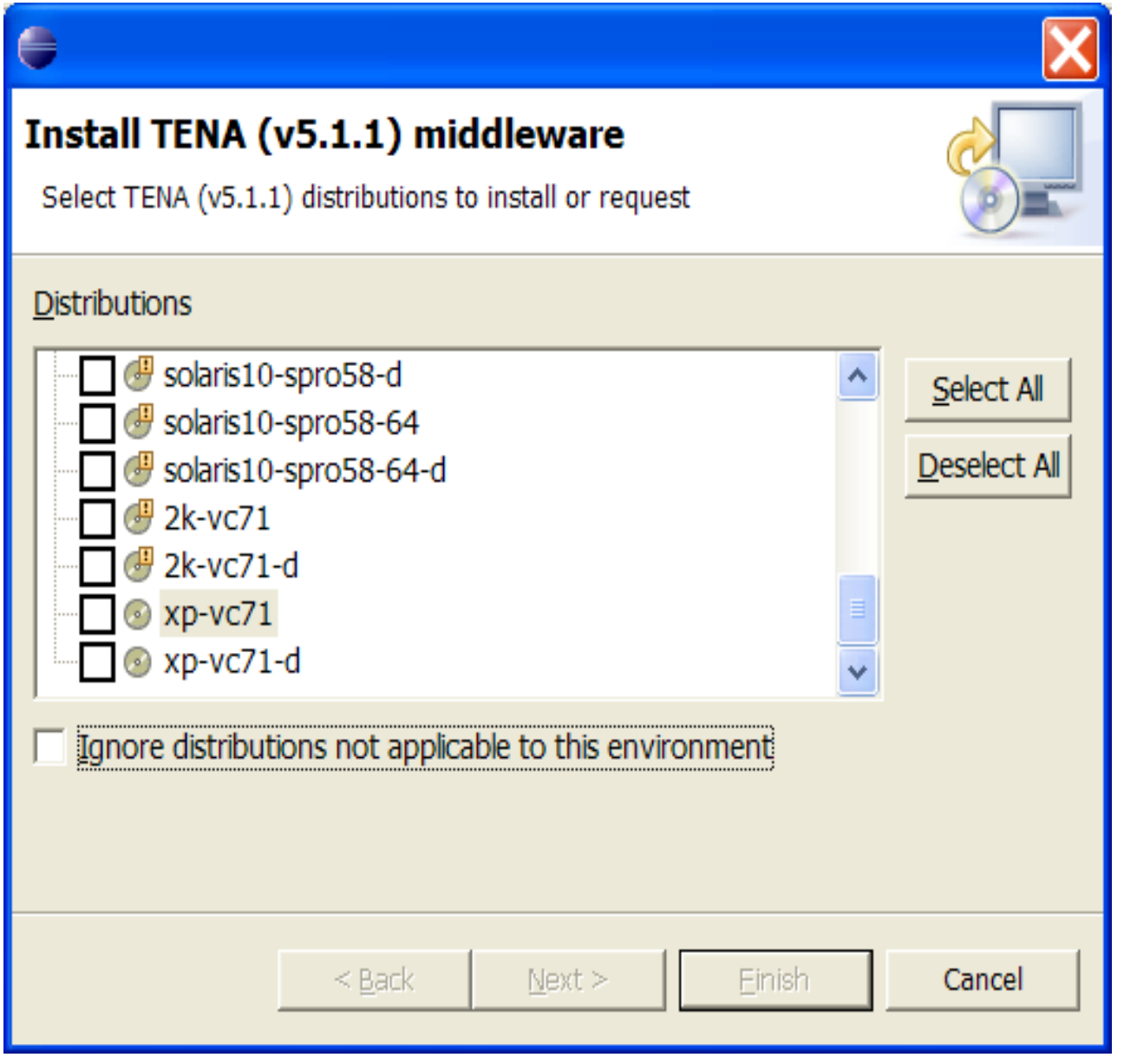
TENA Repository Exploring



Installing/Uninstalling TENA Middleware

- **Allows users to install the TENA Middleware from remote repositories locally**
- **Allows users to remove previously installed TENA Middleware**

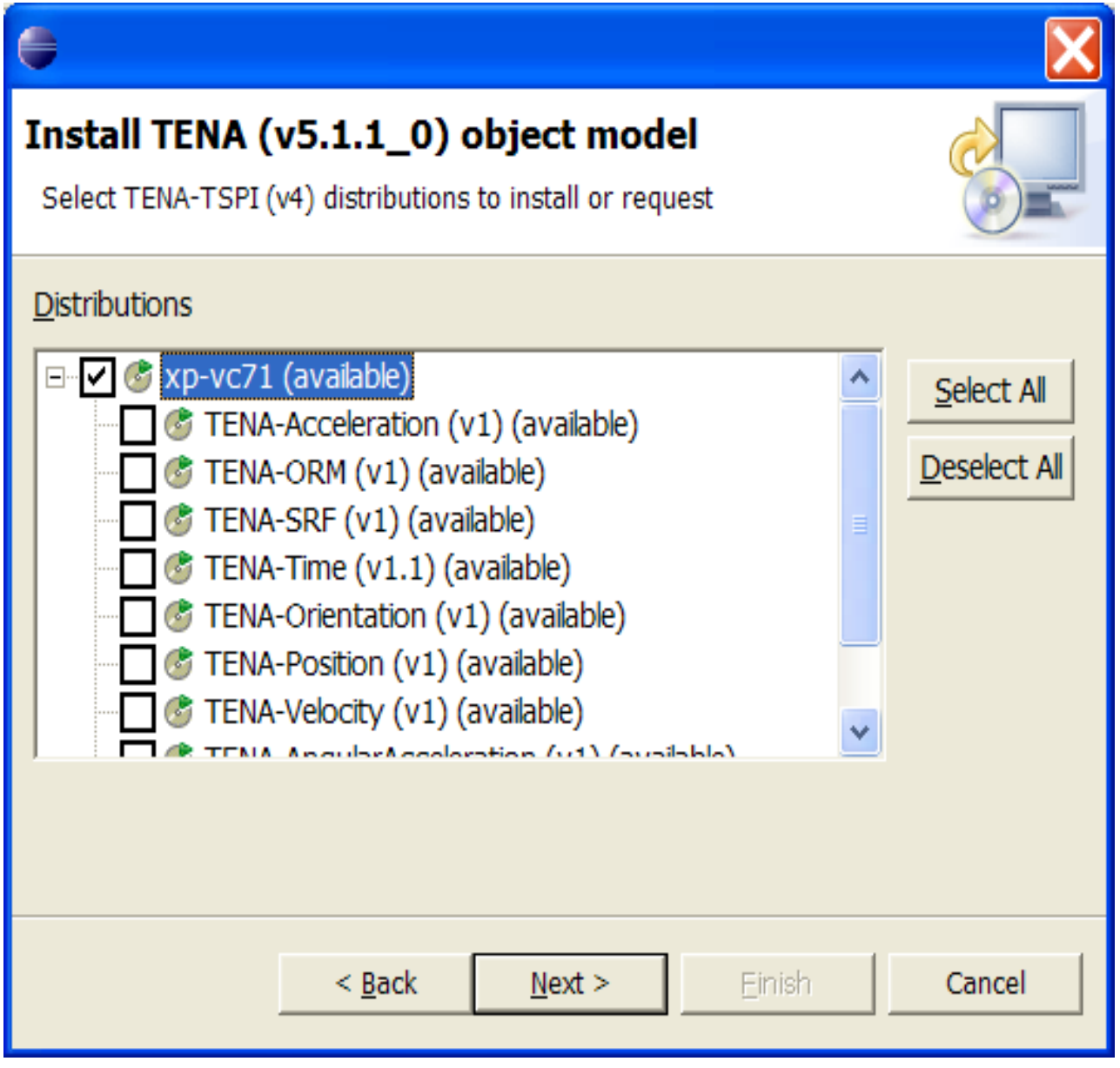
Installing/Uninstalling TENA Middleware



Installing/Uninstalling and Requesting Object Models

- **Allows users to install object model definitions and its dependencies from remote repositories locally**
- **Allows users to request object model definitions from remote repositories**
- **Allows users to remove previously installed object models**

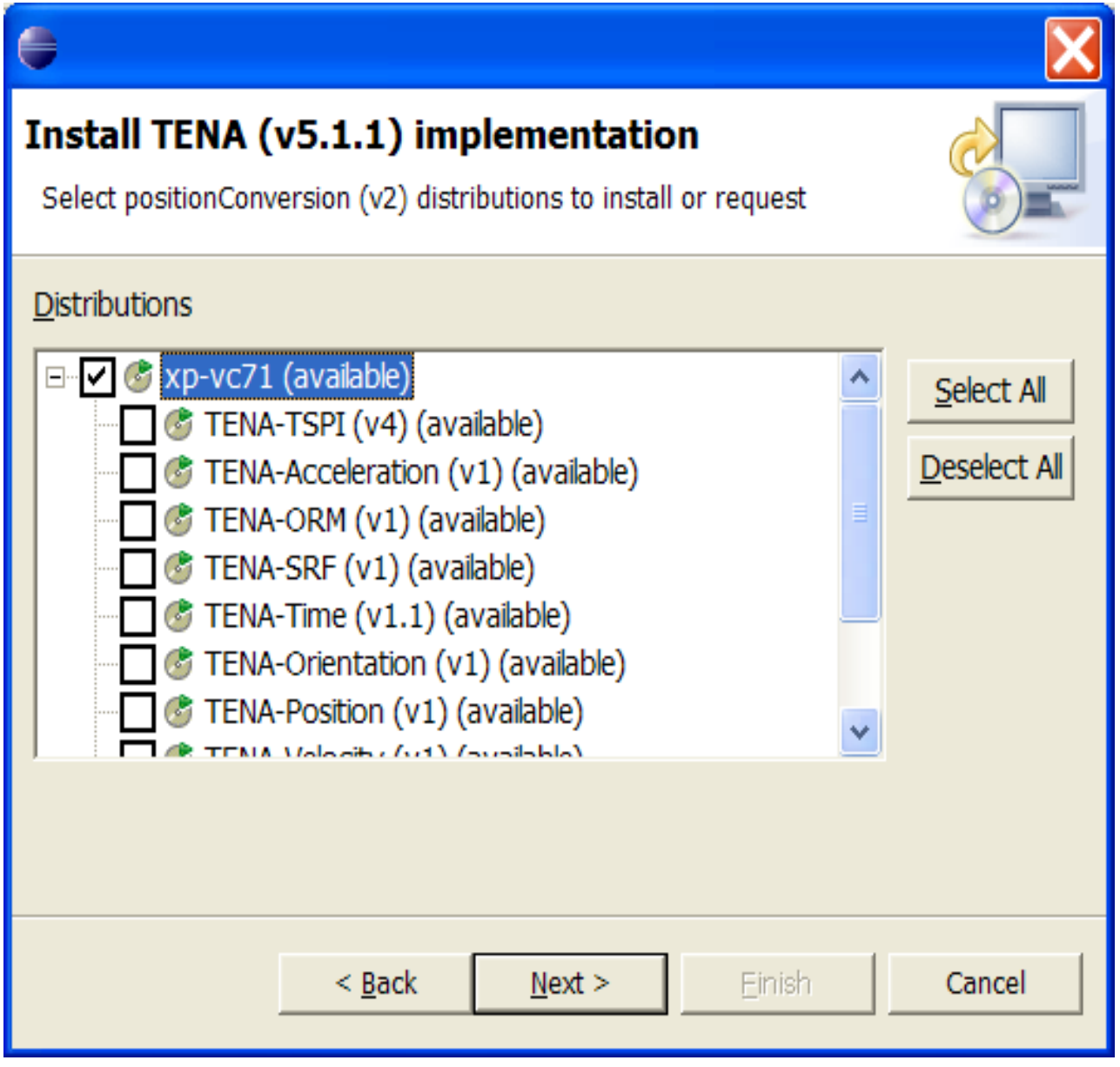
Installing/Uninstalling and Requesting Object Models



Installing/Uninstalling Implementations

- **Allows users to install object model implementations and its dependencies from remote repositories locally**
- **Allows users to remove previously installed implementations**

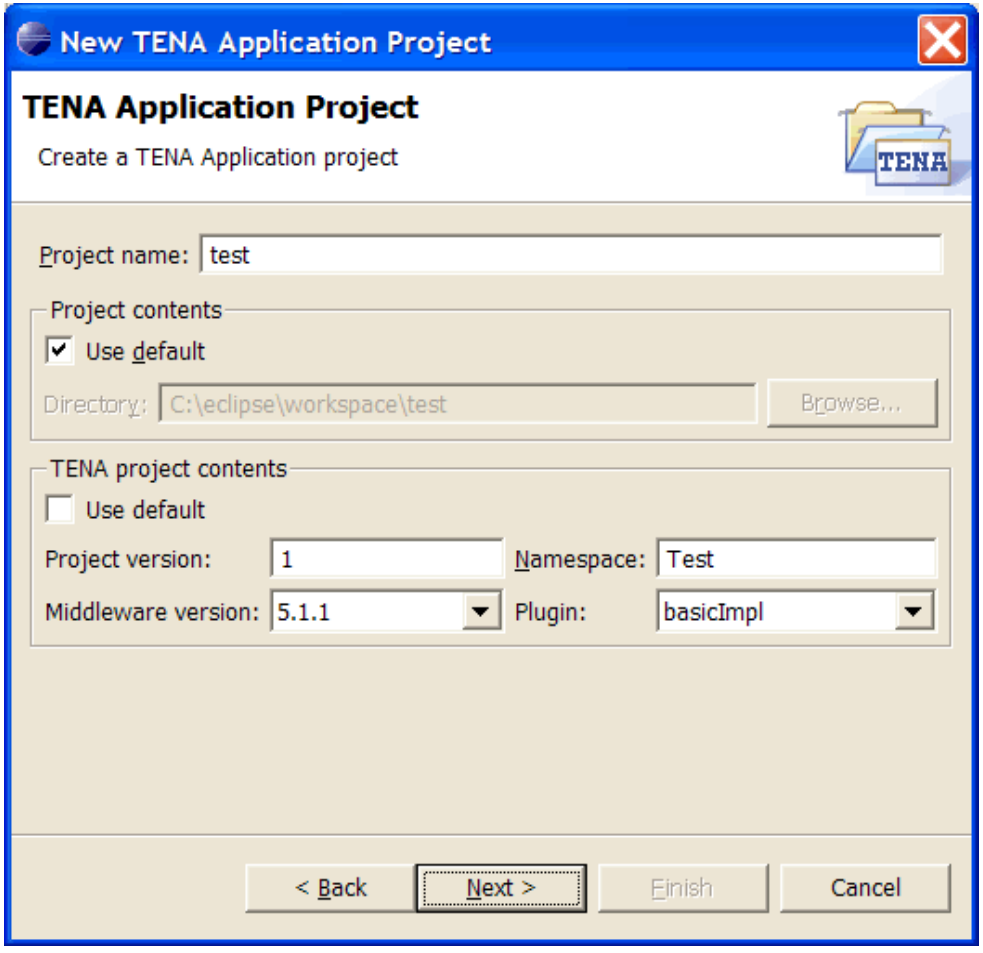
Installing/Uninstalling Implementations

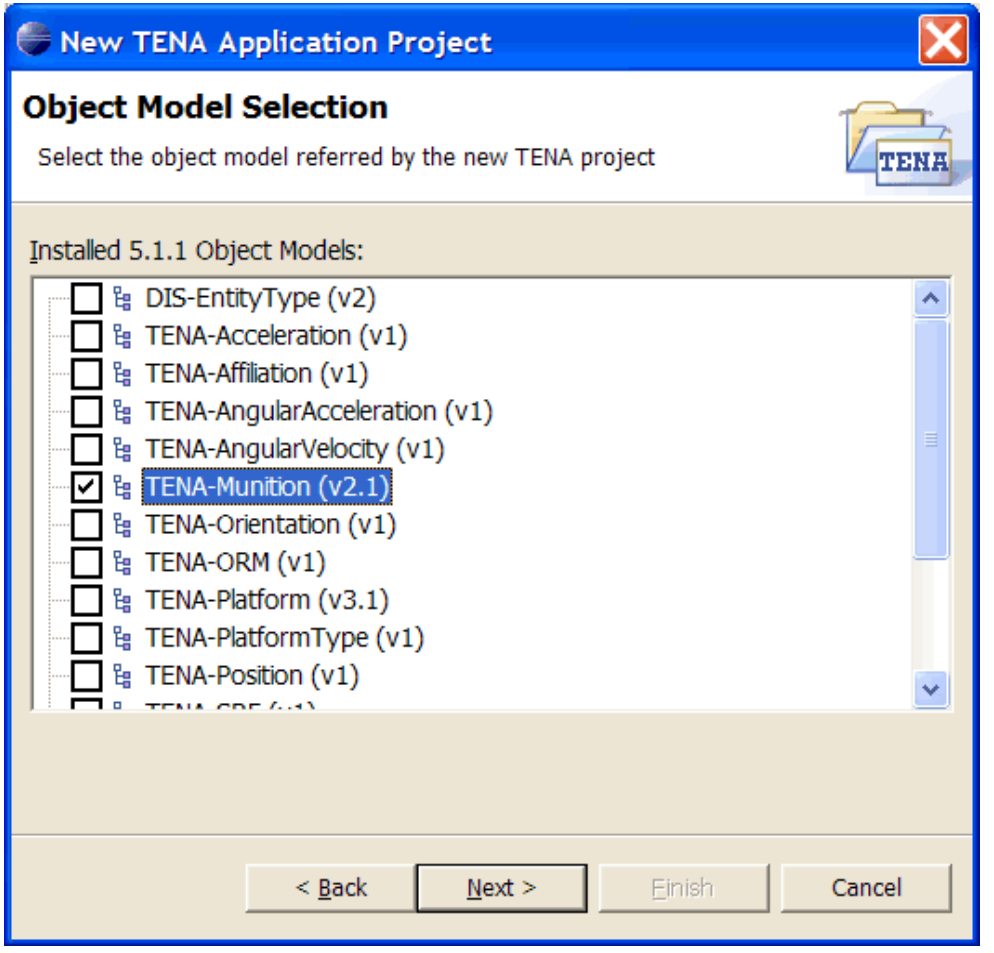


Creating TENA Projects

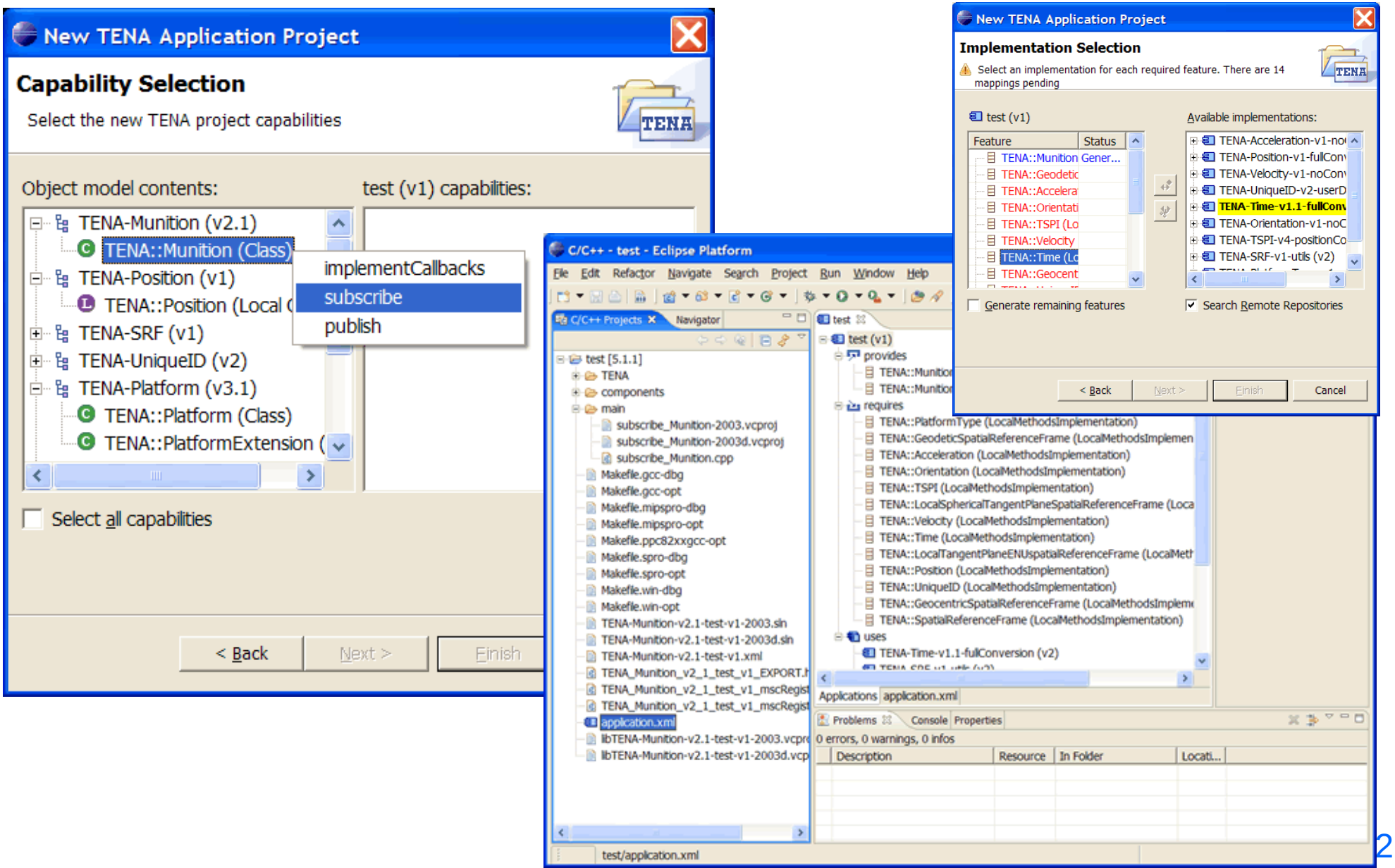
- **Allows users to customize (e.g., specify namespace, target middleware version, OMC plugin, capabilities and implementations to reuse) and generate TENA projects**

Creating TENA Projects





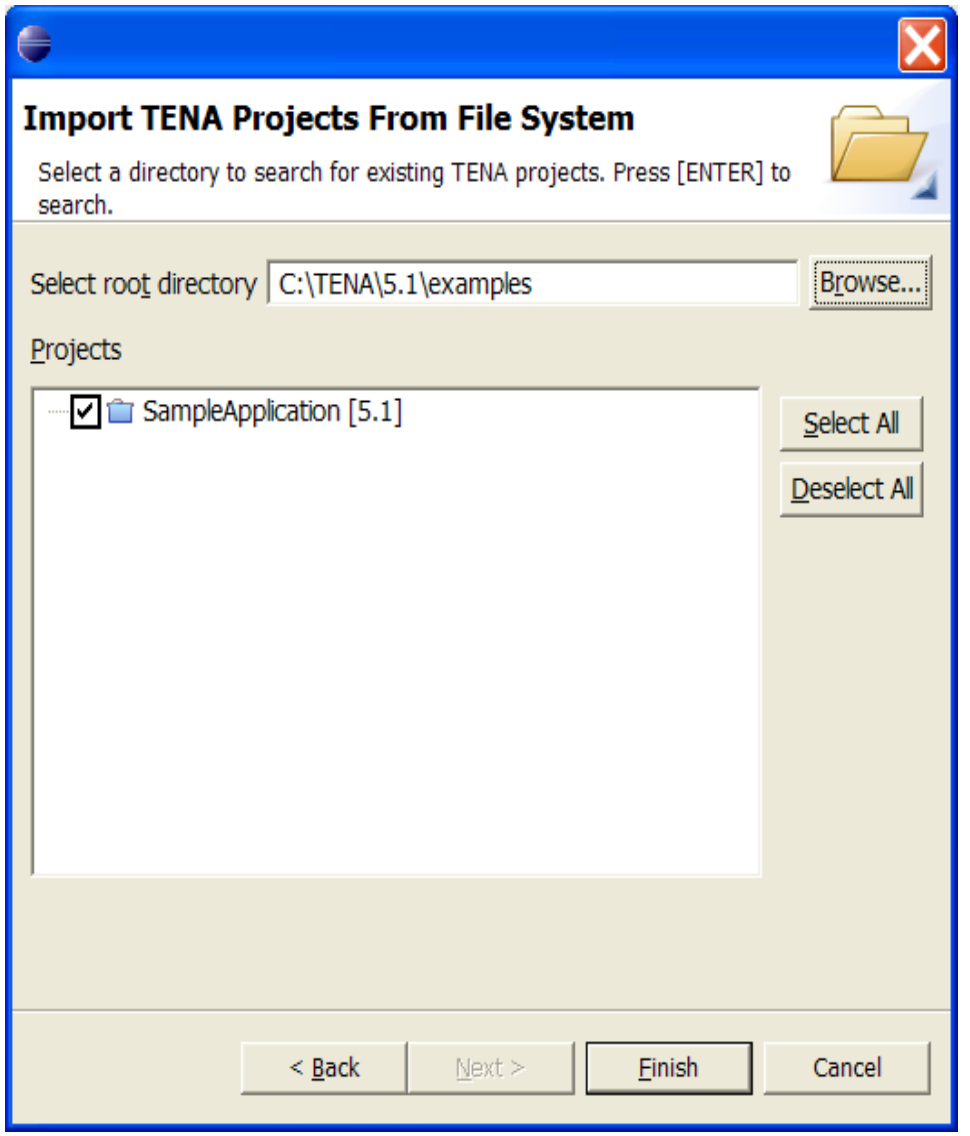
Creating TENA Projects

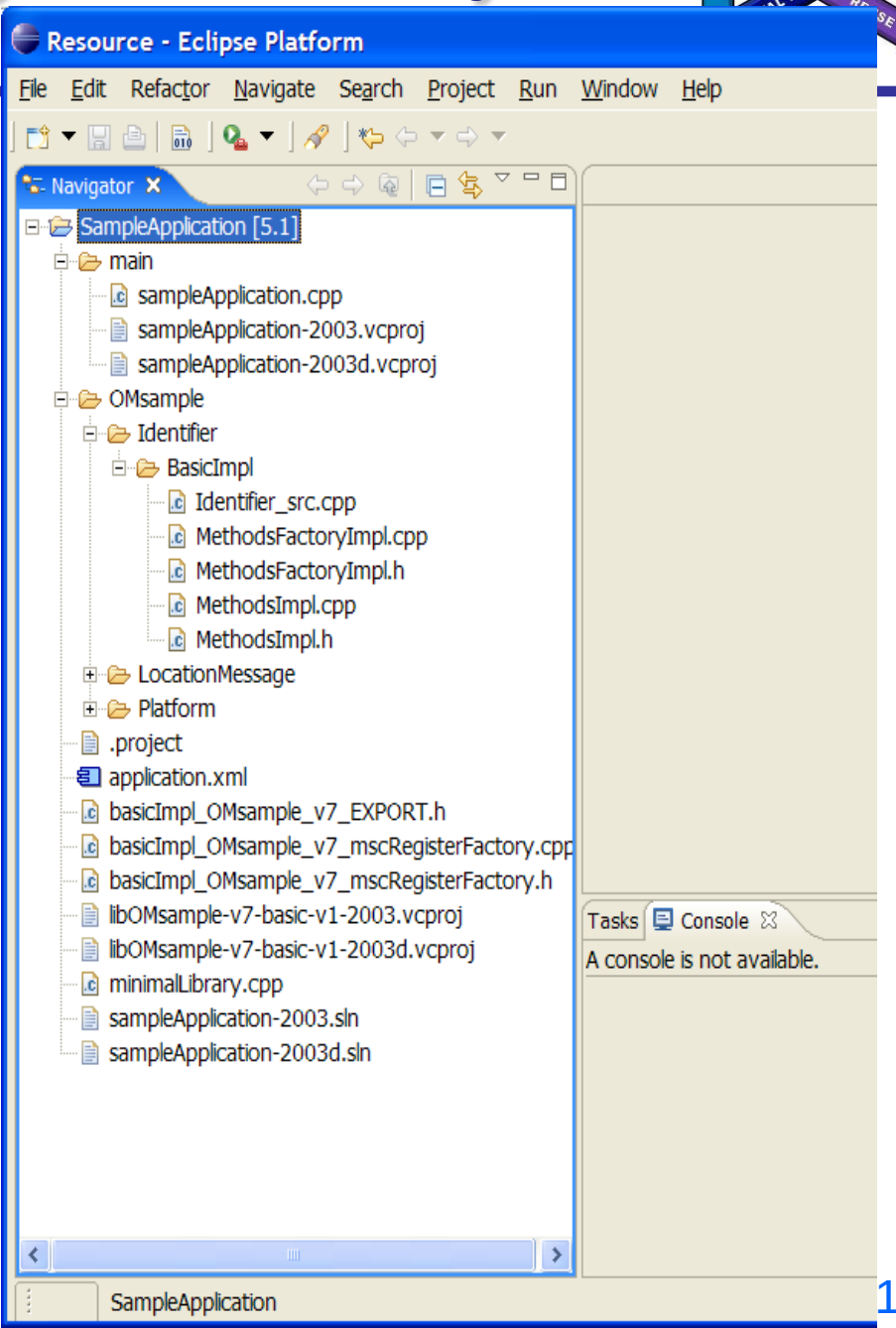


Importing Existing TENA Projects

- **Allows users to import into TIDE, existing TENA projects that were developed without TIDE**

Importing Existing TENA Projects

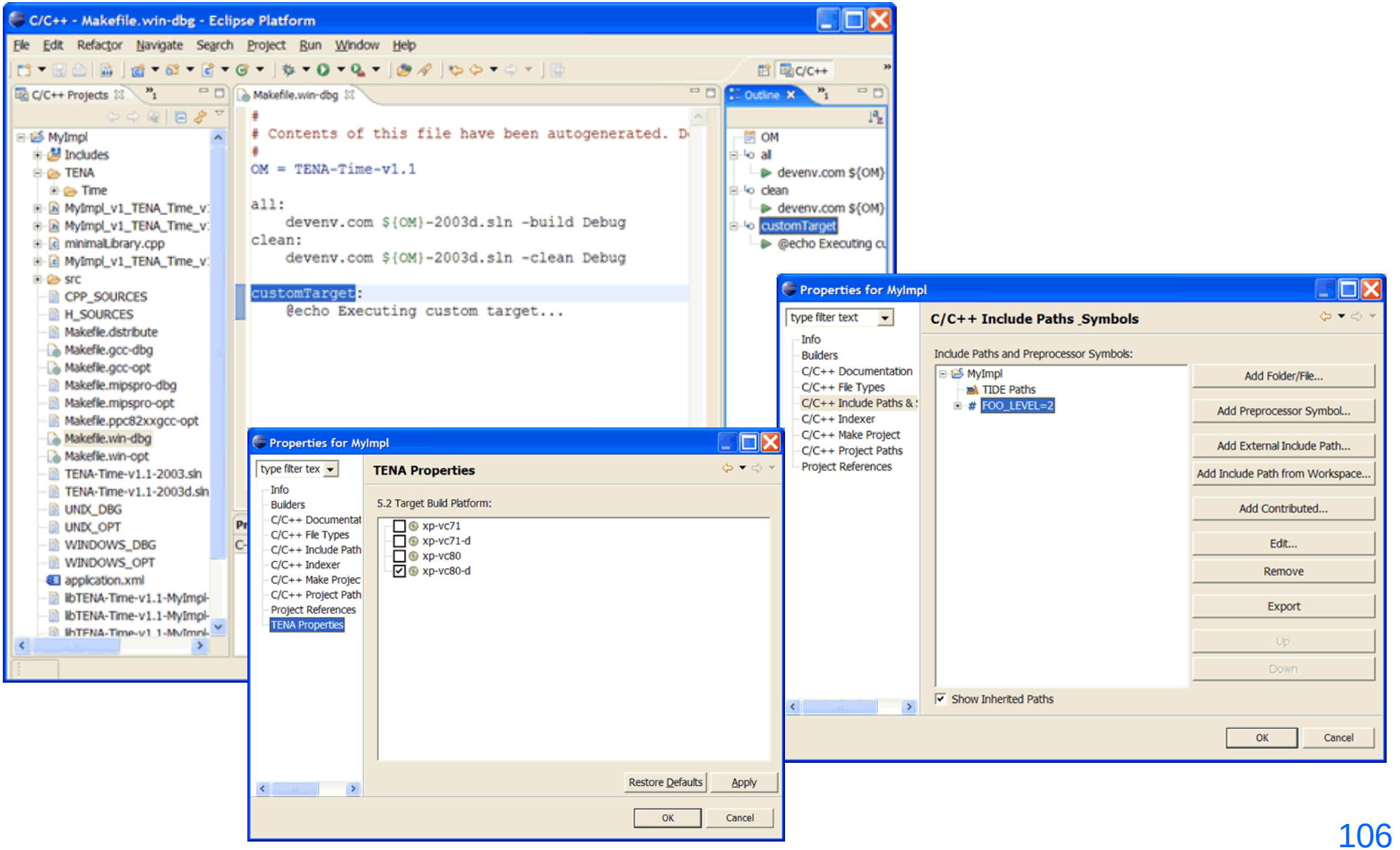




Configuring TENA Projects

- **Allows users to specify target build platform (e.g., xp-vc80), add custom preprocessor definitions, include paths, libraries, source files and make targets.**

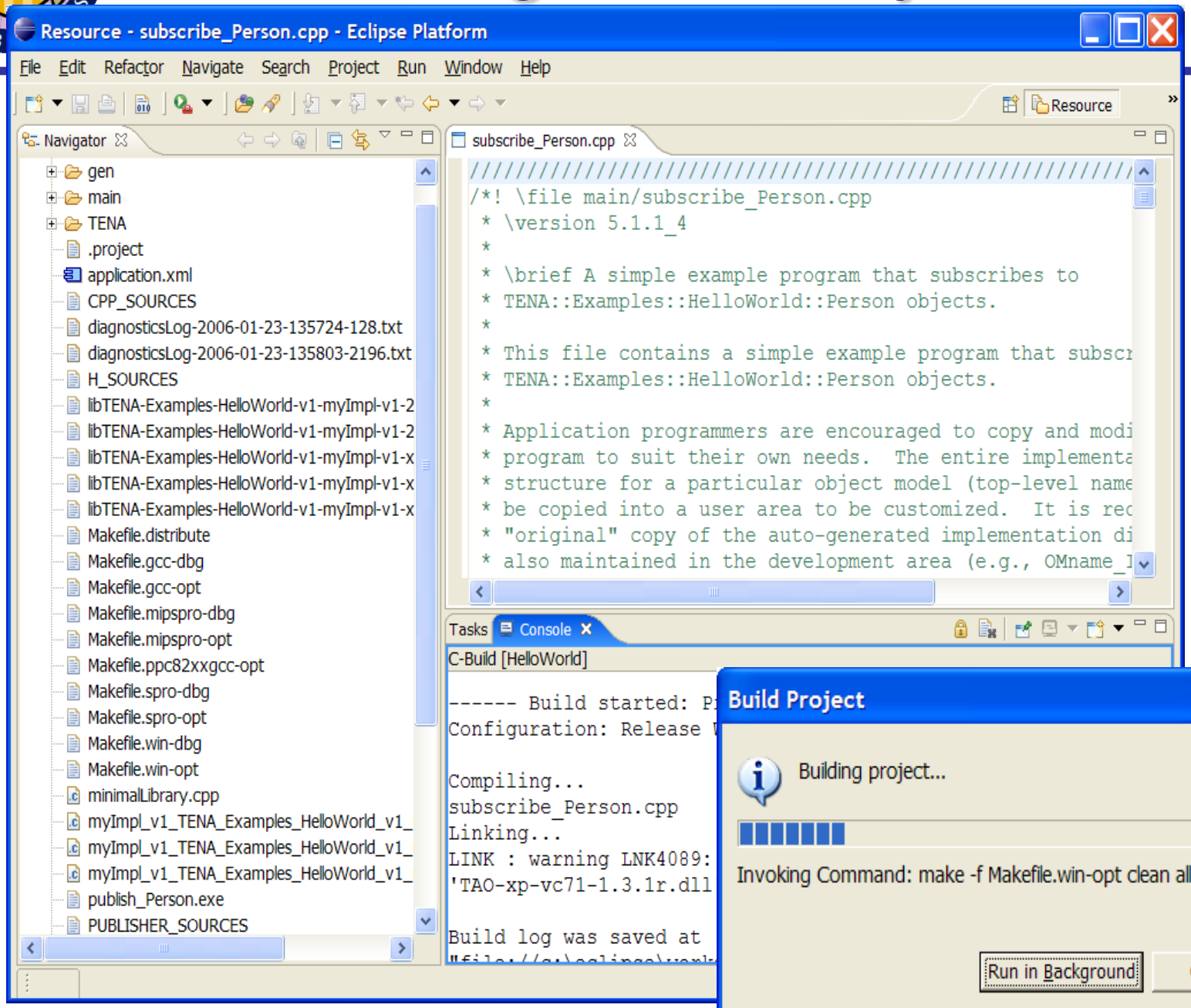
Configuring TENA Projects



Building TENA Projects

- **Allows users to compile and link TENA projects**

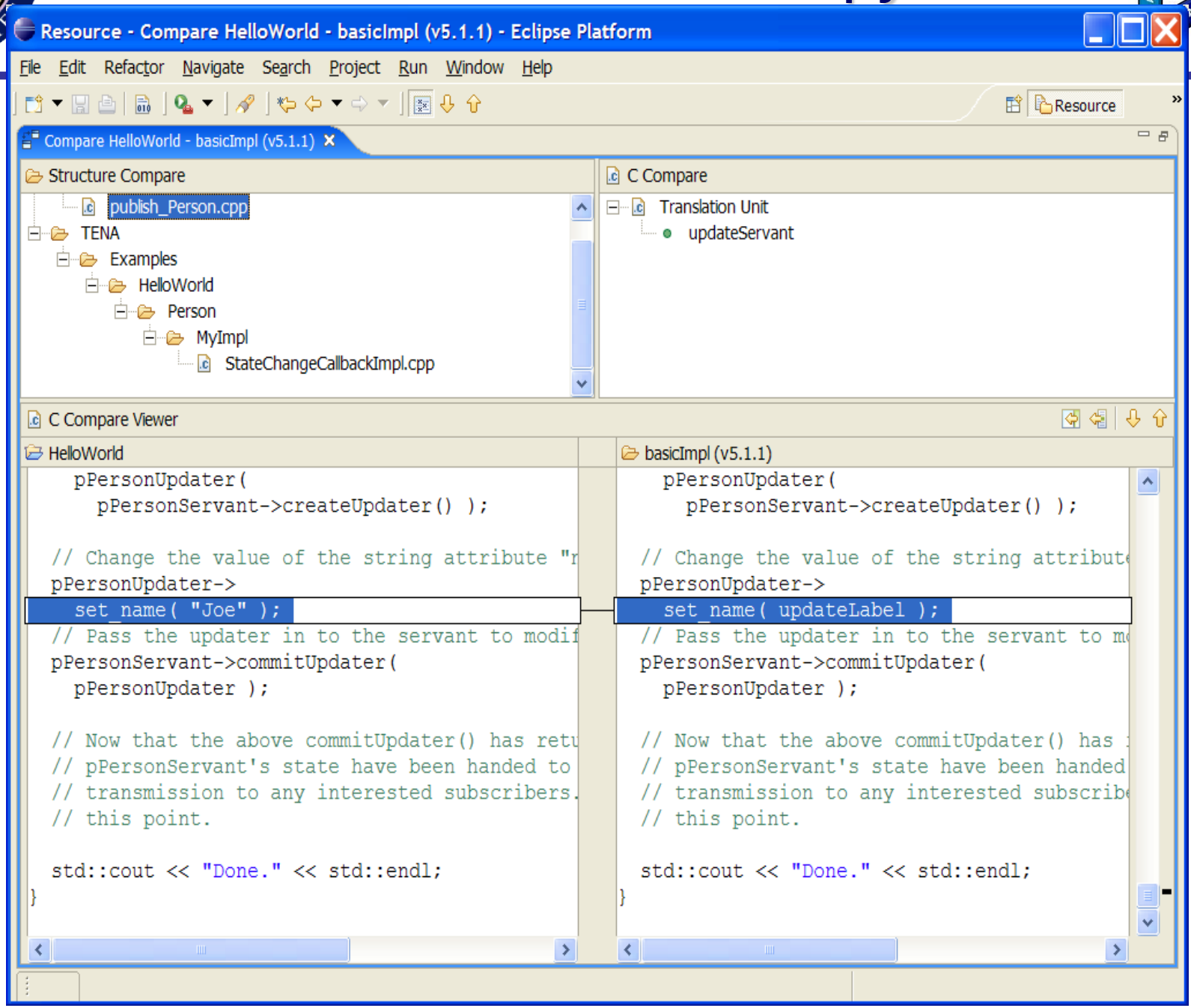
Building TENA Projects



Comparing TENA Projects with their Pristine Copy

- **Allows users to determine their changes to the generated TENA projects**
- **Number of changed classes is a good indicator of the conflicts that user will need to resolve when migrating to a different object model or middleware release**

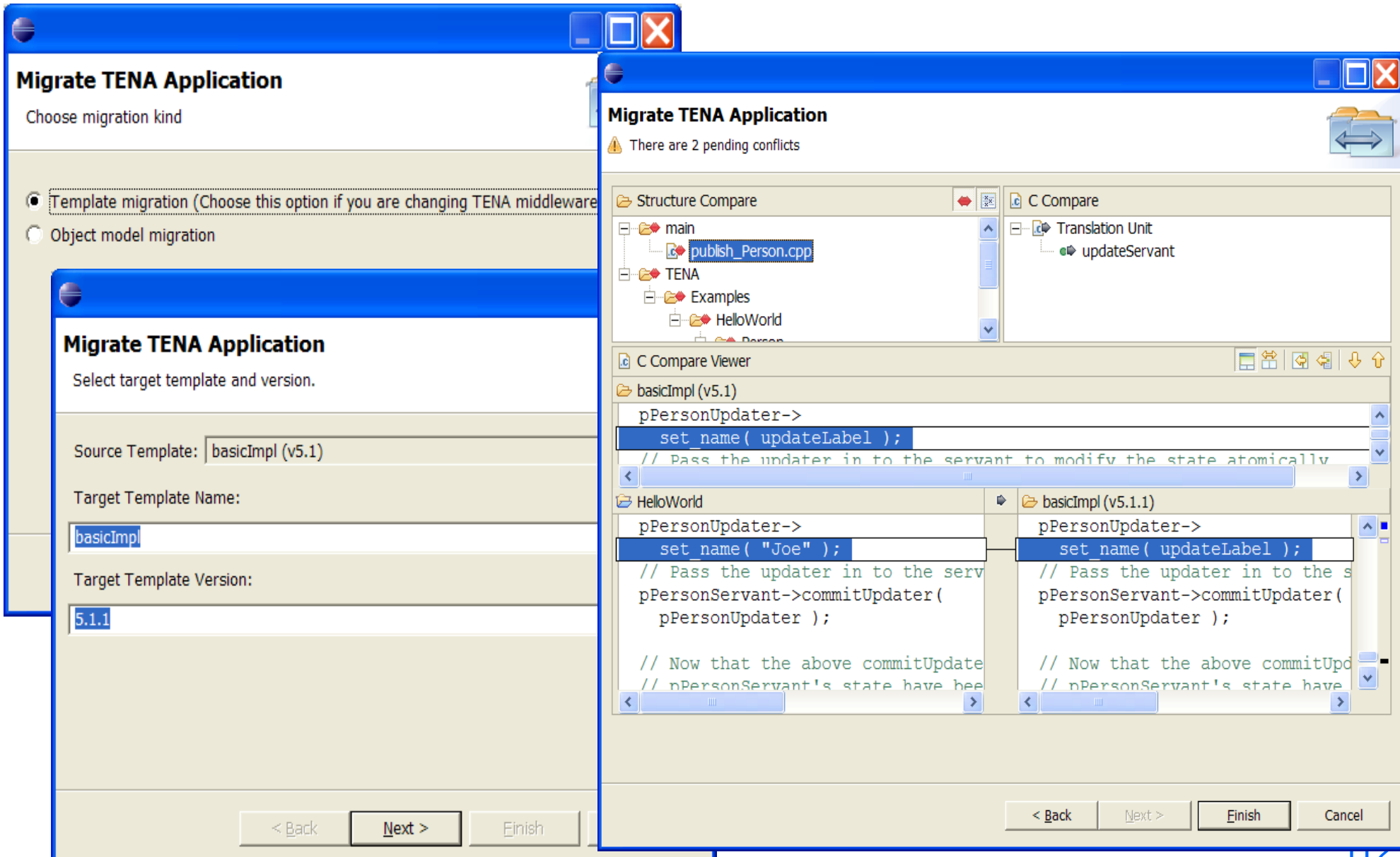
Comparing TENA Projects with their Pristine Copy



Migrating TENA Projects between Middleware releases

- **Assists users in porting their TENA projects to a different TENA middleware release**
  - (e.g., from TENA-v5.1 to TENA-v5.1.1)

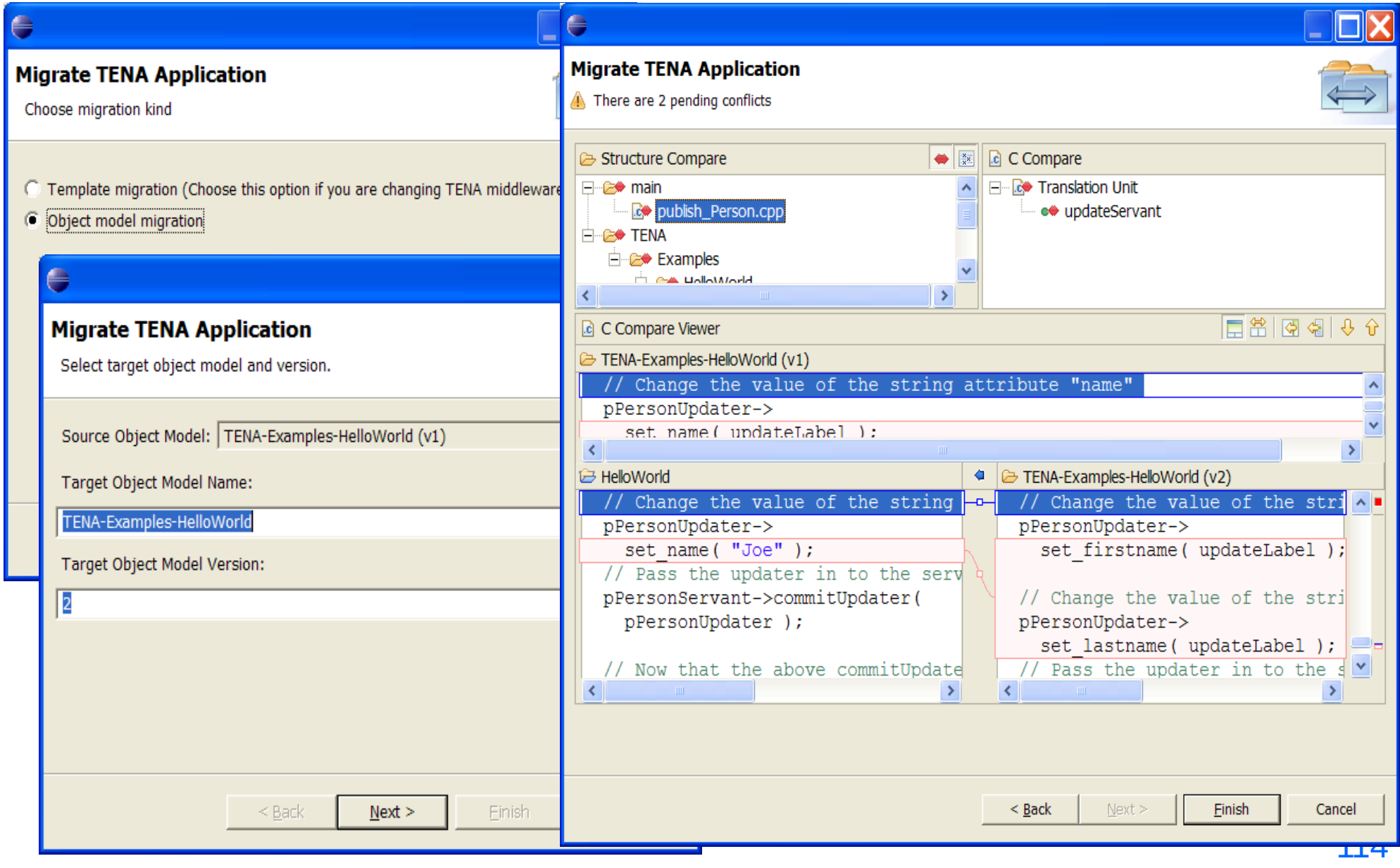
Migrating TENA Projects between Middleware releases



Migrating TENA Projects between object model releases

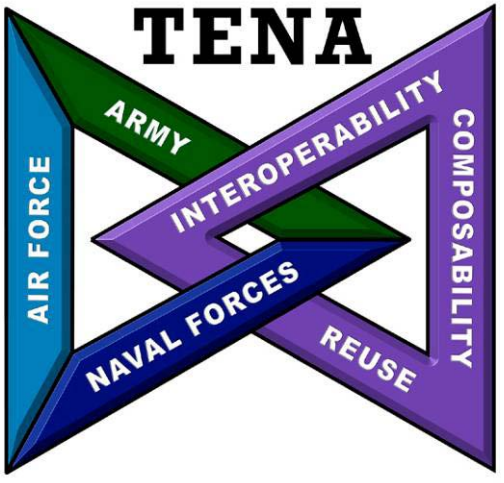
- **Assists users in porting their TENA projects to a different object model release**
  - (e.g., from TENA-Platform-v3 to TENA-Platform-v3.1)

Migrating TENA Projects between object model releases



Summary

- **TIDE is a developer utility that is expected to significantly enhance the usability of TENA**
  - Browsing local and remote object models
  - Building and installing object models
  - Managing changes with new object model and middleware versions
- **Download Now!**
  - https://www.tena-sda.org/display/TIDE



TENA Definition Language

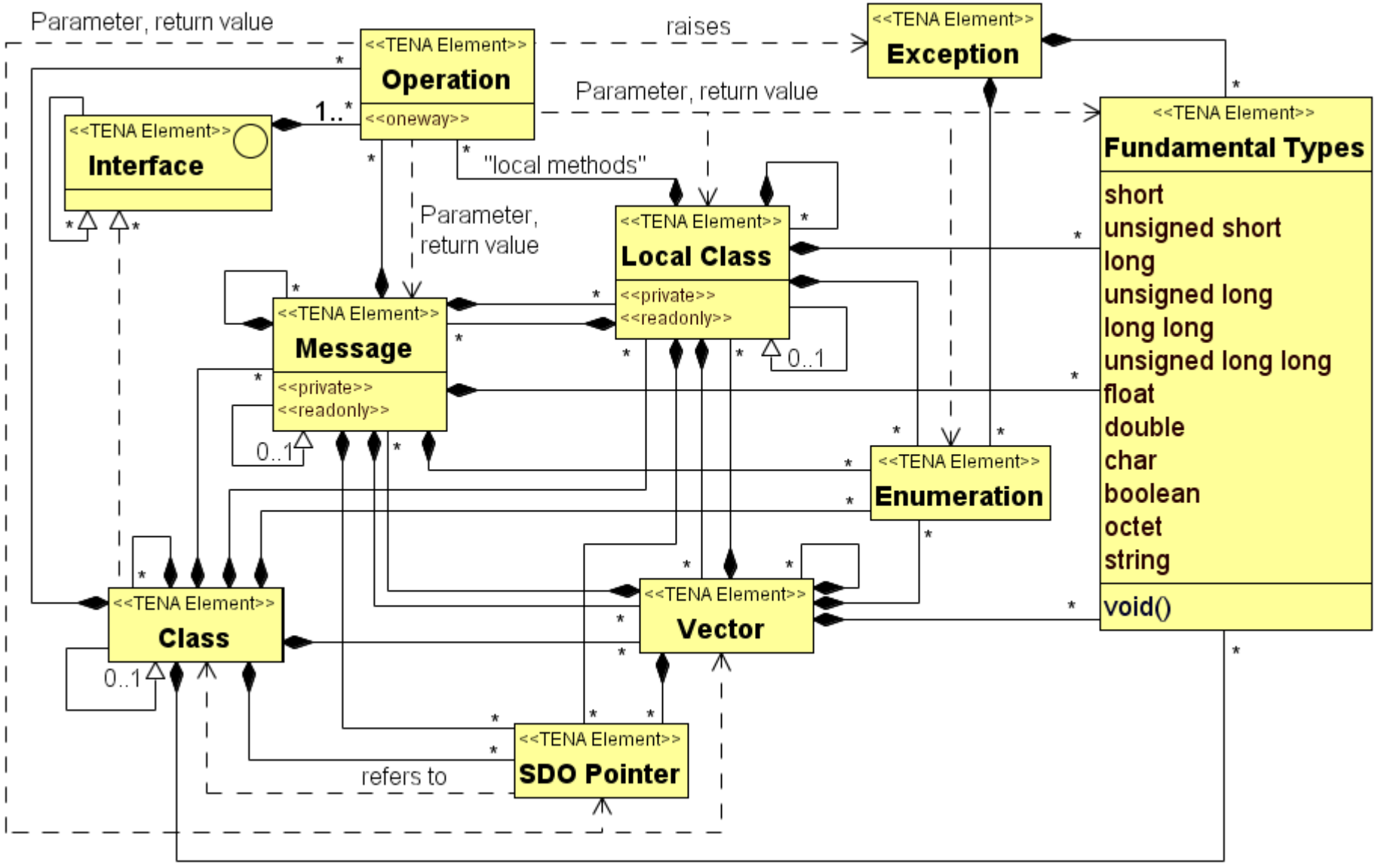
Designing Your Own LROM

- **Understand the information needs of your logical range**
  - What applications are serving what objects?
  - What applications are consuming what objects?
- **Define your LROM**
  - Use a UML tool to design your objects and their relationships
  - Remember the TENA meta-model focuses on SDOs, local classes, and messages
- **Encode your LROM manually in an ASCII text file in TDL**
  - Or, use the TDL Generation Tool (available on the TENA web site) to create a TDL file (MagicDraw 9.5 and 12 users)
- **Upload your TDL file to the Web-based code generator**
- **Retrieve the generated software from the web site**
- **Write a test application to test your LROM**

Why is TDL being used?

- **TDL is a text format for defining SDOs, messages, local classes, and all the auxiliary information needed for an LROM**
- **TDL was designed to be an unambiguous text representation of the LROM**
- **TDL is based on the OMG's Interface Definition Language (IDL), but has extensions to deal with SDOs**
  - The extensions are designed to be familiar to C++ and Java programmers
- **TDL is NOT C++**

TENA Meta-Model Release 5.2.2



Package

- The syntax for package is:

```
package packageName
{
  // all of the material
  // in this namespace
};
```

- All LROMs MUST be contained in at least one package
- An example package:

```
package Utility
{
  local class VelocityLTP
  {
    float x;
    float y;
    float z;
  };
};
```

- Fully qualified names are written using the "::" operator
- For example, the `VelocityLTP` struct defined above in package `Utility` is referenced in the following manner:

```
Utility::VelocityLTP
```

- Packages can be nested, like so:

```
package OMsample
{
  package Utility
  {
    local class VelocityLTP
    {
      float x;
      float y;
      float z;
    };
  };
};
```

- In this case, the fully qualified name for `VelocityLTP` is

```
OMSample::Utility::VelocityLTP
```

Fundamental Data Types

- **The following basic types are supported by TDL:**
  - **`short`** — 2-byte signed integers
  - **`unsigned short`** — 2-byte unsigned integers
  - **`long`** — 4-byte signed integers
  - **`unsigned long`** — 4-byte unsigned integers
  - **`long long`** — 8-byte signed integers
  - **`unsigned long long`** — 8-byte unsigned integers
  - **`float`** — 4-byte floating point numbers (roughly 7 decimal digits of accuracy)
  - **`double`** — 8-byte floating point number (roughly 17 decimal digits of accuracy)
  - **`boolean`** — can only take two values: TRUE and FALSE
  - **`char`** — a single byte interpreted as an ASCII character
  - **`string`** — a sequence of characters
  - **`octet`** — a byte with no intended interpretation, simply eight bits
  - **`void`** — a null return value for a method

Local Classes

- **Local classes are simply classes that get moved in their entirety (identity, state, and behavior) from servers to clients**
- **An example local class is:**

```
// Assume definition of class Entity and
// enumeration TrackingInstrumentationType

local class EntityTrackingData
{
  Entity * theEntity;                  // SDO Pointer
  short highestOrderDerivative;        // data element
  TrackingInstrumentationType type;    // Enumeration
  long numberOfArticulations;          // data element

  void increaseArticulations();        // local method
};
```

A local class may:

- be used as a parameter or a return value in an operation
- inherit from at most one other local class
- contain SDO pointers
- contain vectors
- contain enumerations
- contain fundamental types
- contain other local classes
- contain messages
- contain operations/methods that provide behavior—these methods have access to the other elements of the local class
- contain elements marked as private or readonly
- be contained in SDOs
- be contained in vectors
- be contained in messages

Messages

- **A message is a complex data type containing both operations/methods and state information, exactly like a local class, except that messages can be sent using the TENA Middleware's messaging service as bursts of information**
  - E.g., "Fire," "Detonation," or "Missile Away" messages

```
local class Point
{
  double x;
  double y;
  double z;

  string plainString();
};

message PointMessage
{
  Point location;
  long MessageID;
  void doit();
};
```

A message may:

- be used as a parameter or a return value in an operation
- inherit from at most one other message type
- contain SDO Pointers
- contain vectors
- contain enumerations
- contain fundamental types
- contain other messages
- contain local classes
- contain operations/methods that provide behavior—these methods have access to the other elements of the message
- contain elements marked as private or readonly
- be contained in SDOs
- be contained in vectors
- be contained in local classes

Private Local Class and Message Attributes

- **Local class and message methods have the ability to have private attributes**
  - private attributes are accessible only to local methods

```
local class Currency
{
  private double internalRepresentation_;

  float get_ValueInDollars();
  float get_ValueInYen();

  void set_ValueInDollars( in float dollars );
  void set_ValueInYen( in float yen );
};
```

Read-Only Local Class and Message Attributes

- **Local class and message methods have the ability to have read-only attributes**
  - read-only attributes can be directly read, but only written by local methods

```
local class UniqueID
{
  readonly unsigned short siteID;
  readonly unsigned short applicationID;
  readonly unsigned long objectID;

  unsigned long long get_key( );
  void set_Null( );
  boolean isNull( );
  boolean isEqual( in UniqueID otherID );
  void set_Values( in unsigned short site,
                   in unsigned short app, in unsigned long obj );
};
```

Enumerations

- **An enumeration represents a user-defined type that can take one of several pre-defined values**
- **Enumerations are used as type-safe alternatives to creating attributes as unsigned longs**

```
enum Months
{
  Januarius,
  Februarius,
  Martius,
  Aprilis,
  Maius,
  Junius,
  Quinctilis,
  Sextilis,
  Septembris,
  Octobris,
  Novembris,
  Decembris
};
```

- **Note that enumeration values are comma-delimited**

An enumeration may:

- be used as a parameter or a return value in an operation
- be contained in local classes
- be contained in messages
- be contained in SDOs
- be contained in vectors

Vectors

- **Vectors are resizable arrays of a specific type**
- **Vectors in TDL map to `std::vector< T >` in C++**

```
// simple vector example is:
class ArrayOfLongs {
  vector<long> myArray;
};

// vector of local classes
local class TestLocalClass
{
  long longVal;
  double doubleVal;
};

class ArrayOfTestLocalClasses
{
    vector<TestLocalClass> myStructArray;
};

// vector of vectors
class VectorOfVectorOfLongs {
  vector< vector < long > > myArray;
};
```

A vector may:

- be used as a parameter or a return value in an operation
- contain SDO pointers that point to a type of SDO
- contain fundamental types
- contain enumerations
- contain another vector
- contain local class instances
- contain message instances
- be contained in SDOs
- be contained in a local class
- be contained in a message

Note: Users may not limit vectors to a fixed size

SDO Pointers

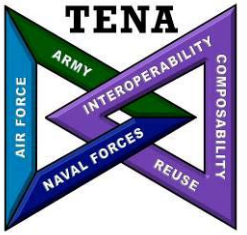
- **An SDO pointer represents a distributed "pointer" to an SDO class**
  - Using a pointer to an SDO, a user can navigate directly to that SDO
  - When a user uses (dereferences) a pointer, he gets a proxy to the SDO, including the current version of the SDO's publication state
  - An individual SDO Pointer refers to a particular type of SDO

```
class Entity
{
  unsigned long entityType;
  //  … other entity parameters
}

class EntityTrackingData
{
  Entity * theEntity;       // SDO Pointer
  short highestOrderDerivative;
  long numberOfArticulations;
  vector< long > data;
};
```

An SDO pointer :

- refers to a specific type of SDO
- is created in TDL using the C++ syntax for a pointer ("*")
- may be used as a parameter or a return value in an operation
- may be contained in vectors
- may be contained in local classes
- may be contained in messages
- may be contained in SDOs

SDOs
Defining SDO Classes



- **An example of an SDO class is:**

```
class Sensor
{
  string state;
  boolean onTrack;
  string trackingMode;

  vector<SensorTrack> sensorTracks;

  string point(in double azimuth,
               in double altitude,
               inout double power );
  };
```

- **Each parameter requires a direction indicator: "`in`," "`out`," or "`inout`"**
  - `in` for parameters going into the method
  - `out` for parameters coming out of the method
  - `inout` for parameters that go in, get changed, then come back out again

An SDO class may:

- inherit from at most one other SDO
- be referred to by SDO pointers
- implement (multiple) interfaces
- contain fundamental types
- contain SDO Pointers
- contain vectors, but SDOs may not be contained in vectors (only vectors of SDO pointers are permitted)
- contain enumerations
- contain local classes
- contain messages
- contain other SDOs
- contain operations

Inheritance

- Use "**: extends**" after the class or local class name to specify implementation inheritance

```
class Participant
{
  string name;
  string type;
  long ID;
  long displayColor;
  string iconScheme;
  long trackLength;
};

class Platform : extends Participant
{
  float fuel;
  string bestSource;
  Sensor longRangeSensor;
};
```

Interfaces

- **An interface defines a set of related method signatures**
- **An example interface is:**

```
interface Controllable
{
  string initialize();
  string start();
  string stop();
};
```

- **In this example, the remotely-invocable interface `Controllable` contains three method signatures, each taking no parameters and returning a string**
- **Interfaces inherit from one another using the following syntax:**

```
interface ExtraControllable : extends Controllable
{
  string destroy();
};
```

Interfaces:

- may inherit from (extend) one or more other interfaces
- may be implemented by SDO classes
- must contain at least one method signature

Implementing Interfaces in Classes

- **Use "`: implements`" after the class name to specify inheritance**

```
interface Controllable
{
  string initialize();
  string start();
  string stop();
};

class Participant : implements Controllable
{
  string name;
  string type;
  long ID;
  long displayColor;
  string iconScheme;
  long trackLength;
};
```

Interfaces and Inheritance

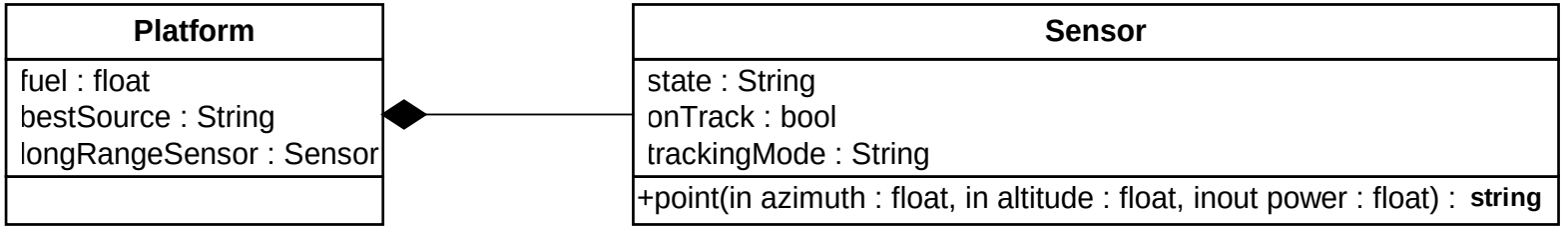
- An SDO class can inherit from (extend) only a single base class but can implement multiple interfaces
- All base classes or interfaces referenced must be either defined or declared before they are used
- The syntax is:

```
// Assume the interfaces “Controllable” and “Test”
// and the classes “Participant” and “Sensor” have
// already been defined

class Platform : extends Participant
               , implements Controllable, Test
{
  float fuel;
  string bestSource;
  Sensor longRangeSensor;
};
```

SDOs
Composition

- **Composition is the ability to include SDOs in other SDOs**
- **The `Platform` SDO contains in its publication state a `Sensor` SDO with the name `longRangeSensor`**

UML



TDL

```
class Sensor {
   string point (in float azimuth, in float altitude,
                 inout float power);
   string state;
   boolean onTrack;
   string trackingMode;
};

class Platform {
   float fuel;
   string bestSource;
   Sensor longRangeSensor;
};
```

User Defined Exceptions

- **Both local class methods and SDO methods support user-defined exceptions**
- **Exceptions in TDL are mapped directly into C++ exceptions**

Exceptions:

- may be raised from local class methods
- may be raised from SDO remote methods
- may contain fundamental types
- may contain enumerations

```
package test {
  exception BadRadarCommand {
    string messageFromRadar;
  };

  exception FaultyRadar {
  };

  class Radar {
     void sendCommand (in string command) raises BadRadarCommand;

     void checkRadar () raises (BadRadarCommand, FaultyRadar);
  };
};
```

Defining exceptions is only required for remote methods. For local methods, standard C++ exceptions can often suffice, e.g., `std::bad_alloc`

Oneway Remote Methods

- **Oneway methods were added to the meta-model for R5**
  - oneway methods must have void as their return type
  - oneway methods may not raise exceptions
  - oneway methods return immediately* after their invocation

```
package test {

   class Radar {
      oneway void sendCommand
           (in string command);
   };
};
```

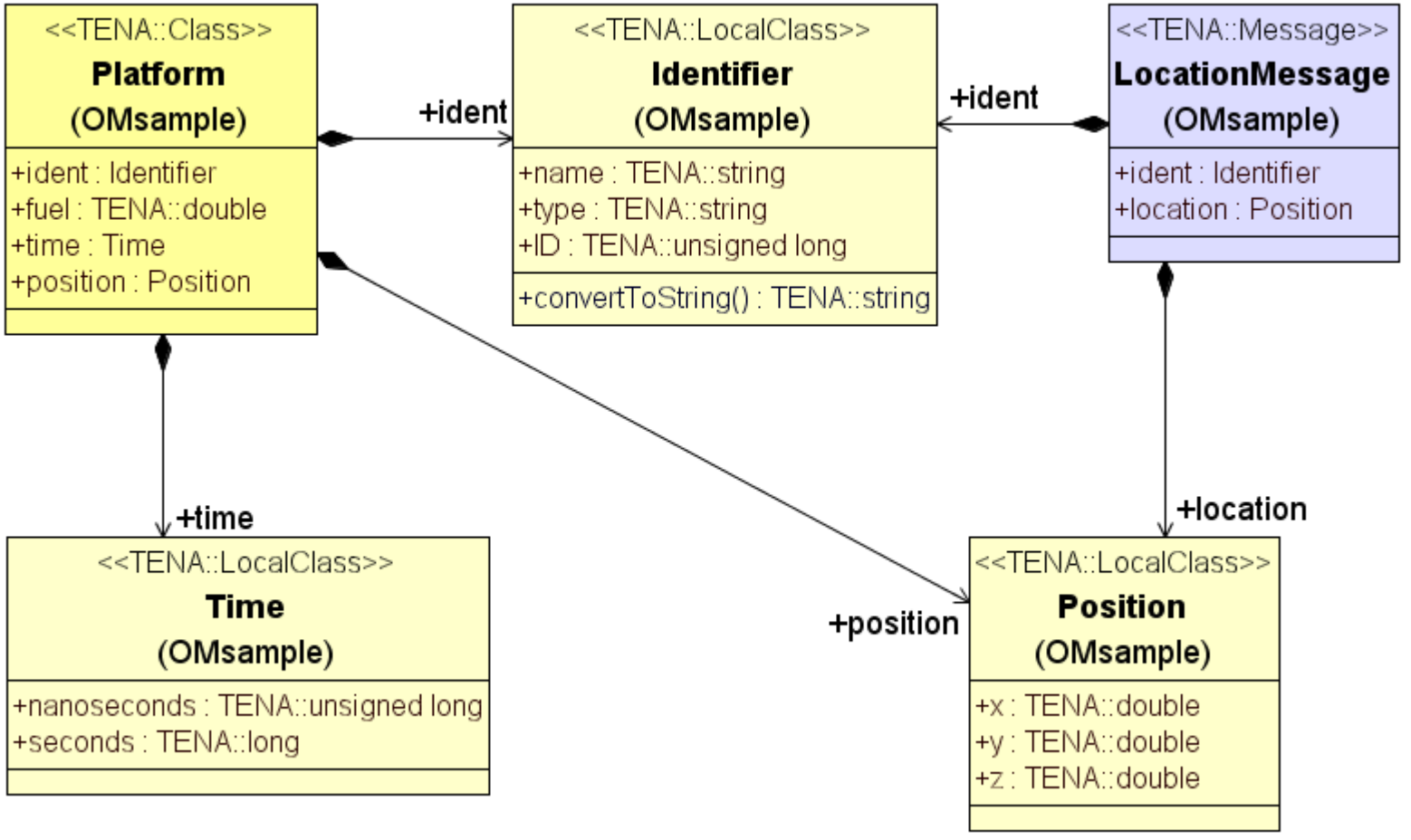
*** "Immediately" doesn't mean instantaneously, there is still work done to marshall the remote method and hand it to the operating system, but the method returns relatively quickly compared to a regular non-oneway method**

Possible Future Enhancements

- Constants
- Ability to recognize non-TDL types ("native")
- Local methods directly on SDOs
- Other data structures such as maps, deques, queues, trees
- SDOs containing vectors of SDOs
- Bitsets

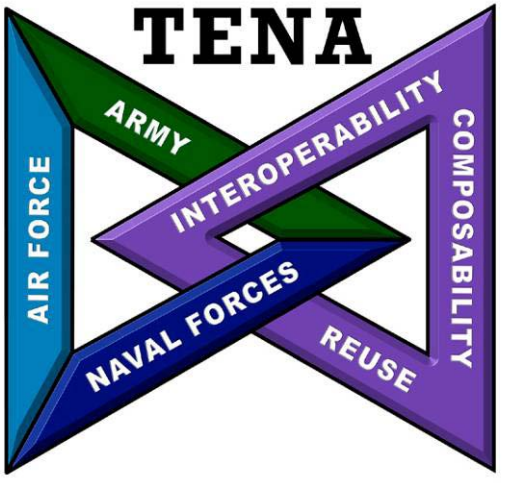
Sample Object Model

- **Delivered as part of the TENA Middleware Distribution**
- **Remember that this Object Model is only a sample to help teach the user community about TDL**



Sample OM in TDL

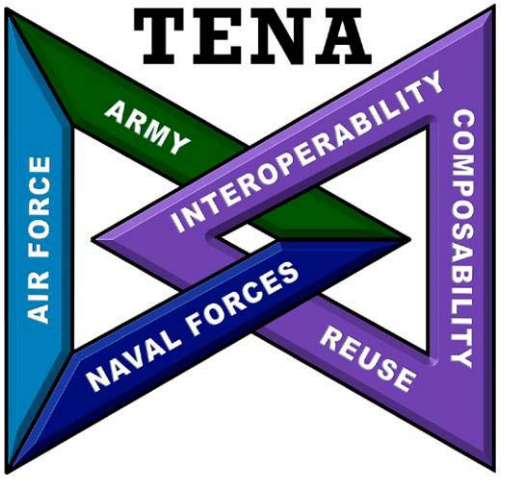
```
package OMsample
{
  local class Time
  {
    unsigned long nanoseconds;
    long seconds;
  };

  local class Position
  {
    double x;
    double y;
    double z;
  };

  local class Identifier
  {
    string name;
    string type;
    unsigned long ID;
    string convertToString();
  };

  class Platform
  {
    Identifier ident;
    double fuel;
    Time time;
    Position position;
  };

  message LocationMessage
  {
    Identifier ident;
    Position location;
  };

};
```


TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE

Questions?


IMPROVING T&E
CTEIP
CAPABILITIES


UNITED STATES
JOINT FORCES
COMMAND
USJFCOM



Patterns Necessary to Understand The TENA Middleware API

What are Design Patterns?

- **A Design Pattern is a pattern of inter-related software classes that can be used (and re-used) to solve a particular problem**
- **Patterns allow developers to codify knowledge about how to solve certain software problems**
  - Prevents having to "reinvent the wheel"
- **Design Patterns make it easier to reuse successful designs and architectures**
- **Design Patterns make it easier to communicate your designs and architectures to other developers**
- **Design Patterns are "discovered," not "invented"**

Design Patterns History

- ***Design Patterns: Elements of Reusable Object-Oriented Software*; Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides; Addison-Wesley, 1998**
- **Buy this book!!!**
- **First codification of the concept of design patterns in software**
  - Based on a similar concept in the realm of (building) architecture
- **The most influential book (and concept) in computer science in the last 15 years**
- **An entire patterns "industry" has grown up based on this original work**
- **Another good book: *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*; Douglas Schmidt et al., 2000**

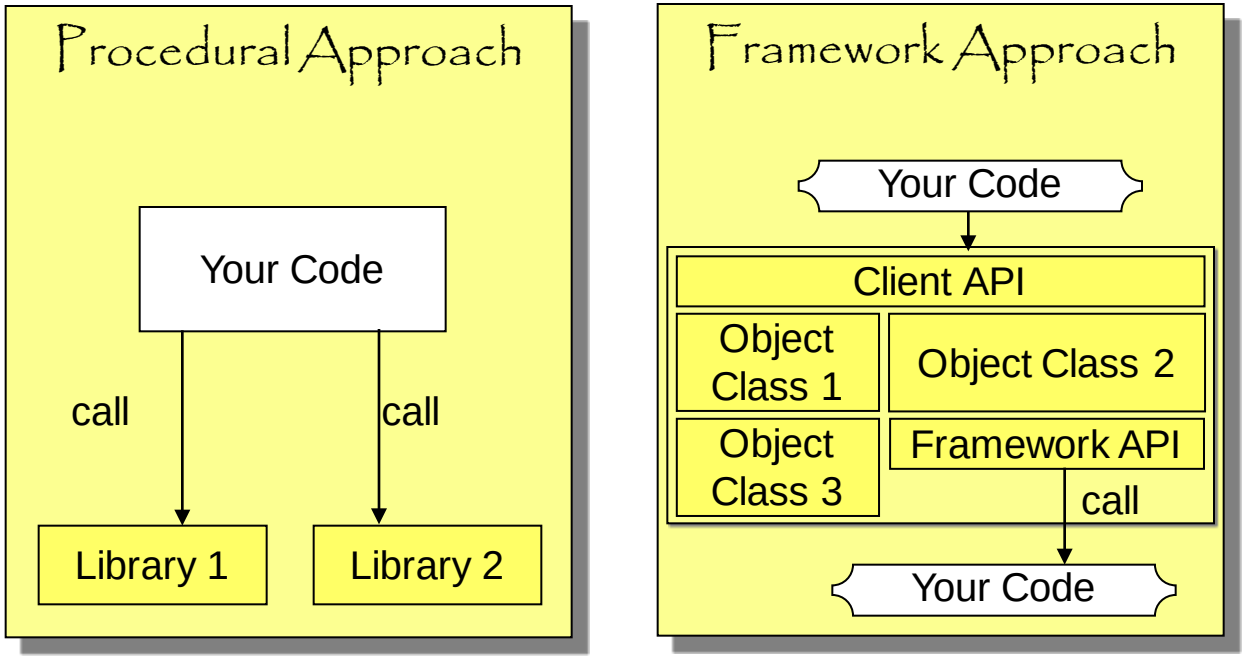
Patterns Catalogued in *Design Patterns*

1. Abstract Factory
2. Adapter
3. Bridge
4. Builder
5. Chain of Responsibility
6. **Command**
7. Composite
8. Decorator
9. Facade
10. **Factory Method**
11. Flyweight
12. Interpreter
12. Iterator
13. Mediator
14. Memento
15. **Observer**
16. Prototype
17. Proxy
18. Singleton
19. State
20. Strategy
21. Template Method
22. Visitor

(1) ***Design Patterns***, p 8.

Frameworks

- **Collections of classes that form the structure for the solution to a group of problems**
- **Generic and tailorable**
- **Flexible and extensible**

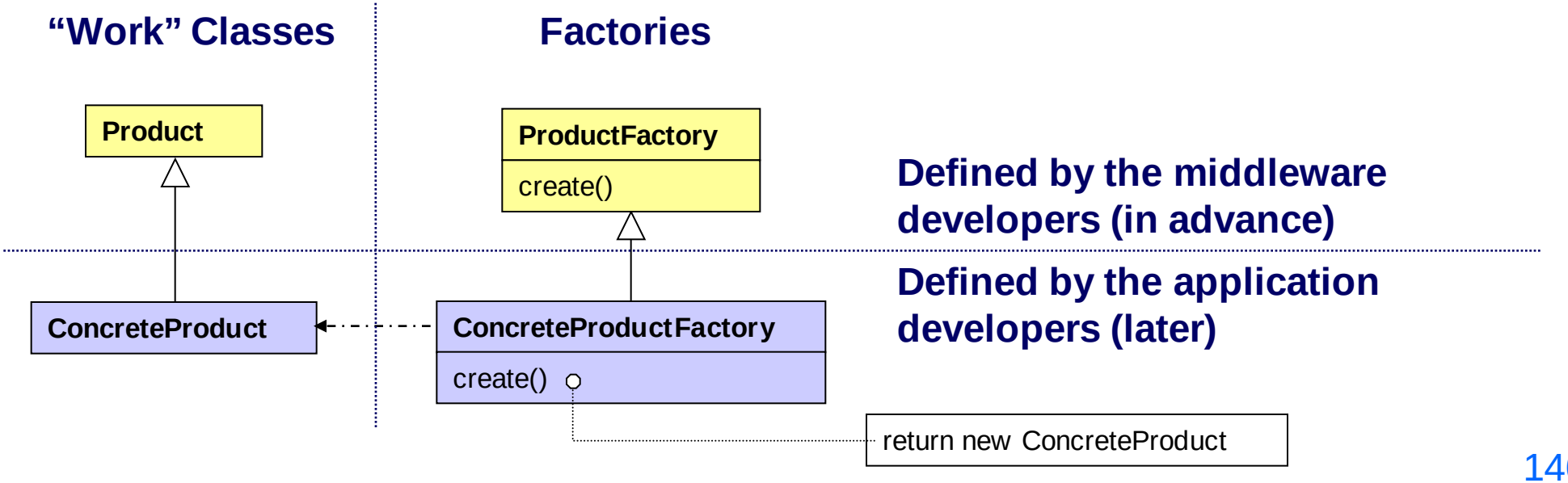


Frameworks obey the "Hollywood Principle": "Don't call us, we'll call you."

Factory Method Pattern

- **Intent – Define an interface for creating an object, but let subclasses decide which class to instantiate**
  - The Factory Method lets a class defer instantiation to subclasses
- **Motivation**
  - Factories are used when the one party knows what object to create and the other party knows when that object needs to be created

- **Structure – parallel class hierarchies**

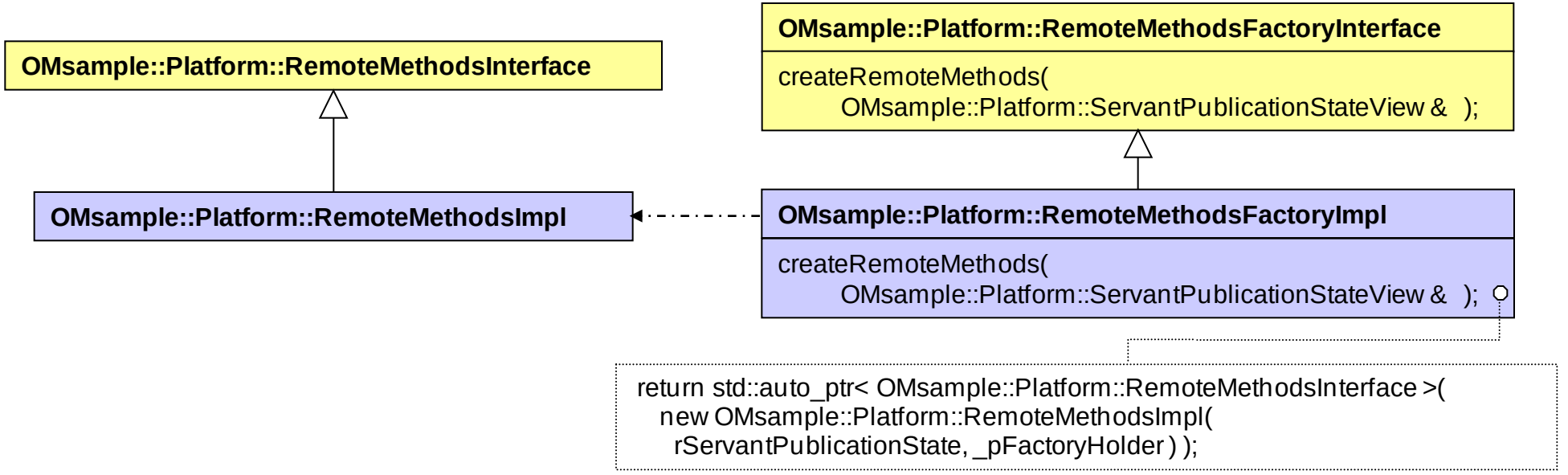


Factory Method Pattern Process

- **Framework (middleware) developers define**
  - `Product` base class
  - `ProductFactory` base class
- **Application developers write**
  - `ConcreteProduct` class
  - `ConcreteProductFactory` class
- **In their application program, application developers do the following**
  - Instantiate an instance of a `ConcreteProductFactory`
  - Hand this instance to the middleware
- **At the right time during execution, the middleware uses the `ConcreteProductFactory` object given to it by the application to instantiate an instance of a `ConcreteProduct`**
  - The middleware holds onto the `ConcreteProductFactory` as a `ProductFactory` and polymorphically invokes the `create()` method

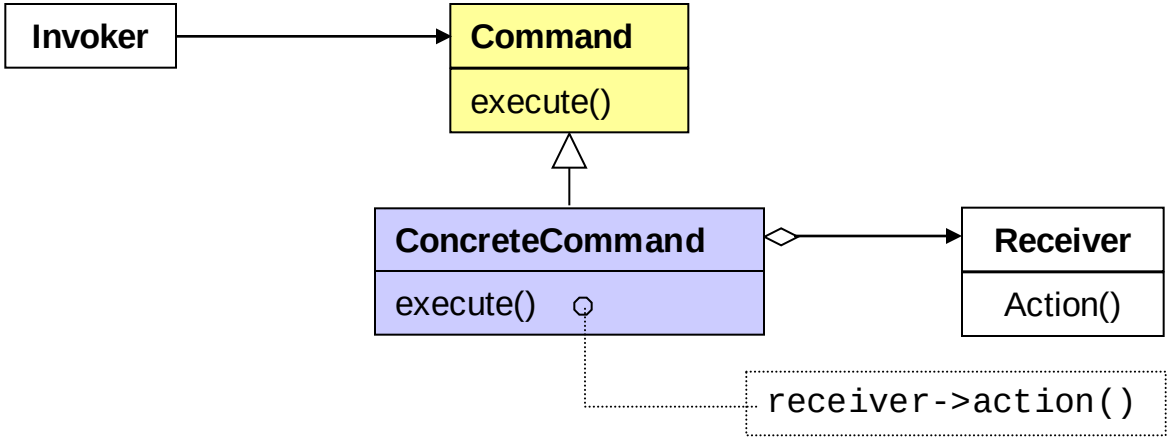
Factory Method Pattern
TENA Middleware Example

- **Each SDO may have its own methods that the user must write but that the middleware must instantiate and use**
- **The user must write their own "`SDOname::RemoteMethodsImpl`" class, and also a factory for this class so the middleware can instantiate it**



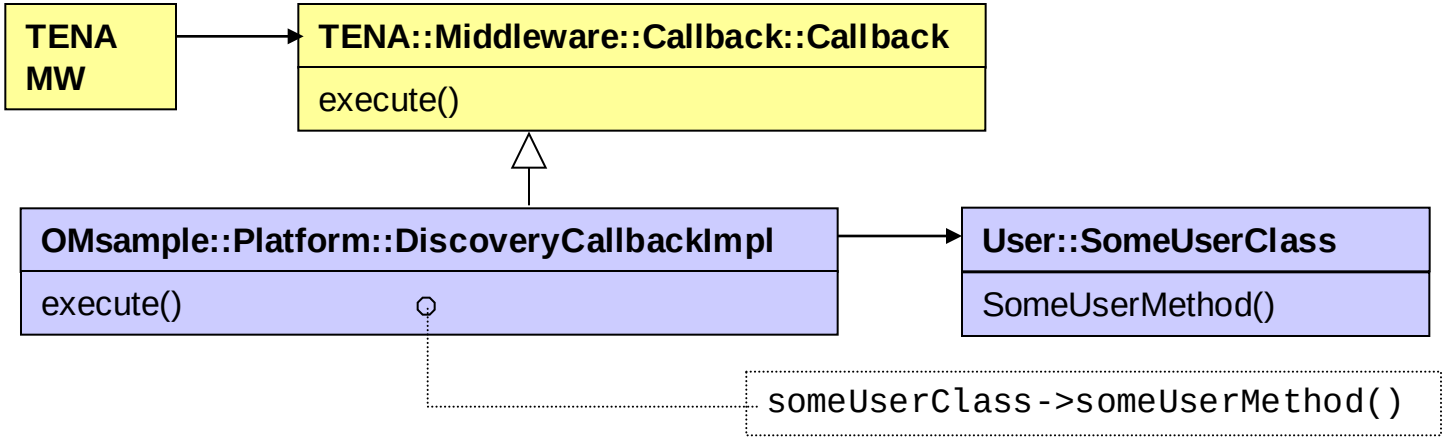
Command Pattern

- **Intent – Encapsulate a method call as an object**
- **Motivation – Sometimes it's necessary to issue requests to objects without knowing anything about the operation being requested or the receiver of the request**
  - Separates the intent of calling a method with the execution of calling the method, in time, code location, code developer, or caller
  - Method calls can be queued, logged, prioritized, etc.
- **Structure**
  - Invoker wants to call the **`Receiver::action()`** performed



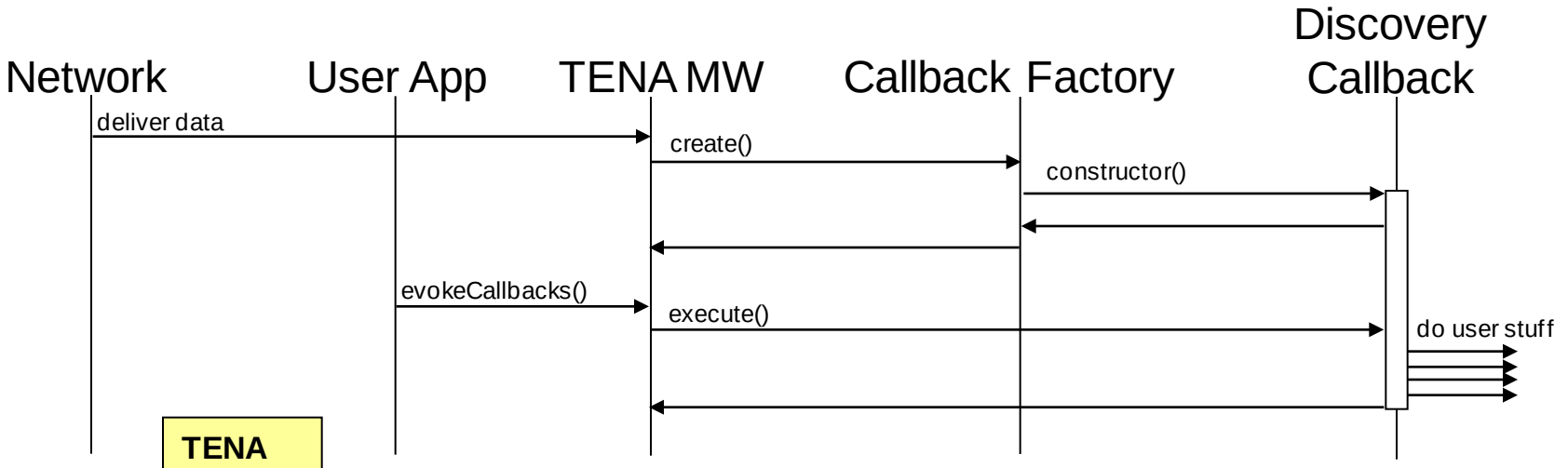
Command Pattern in the TENA Middleware

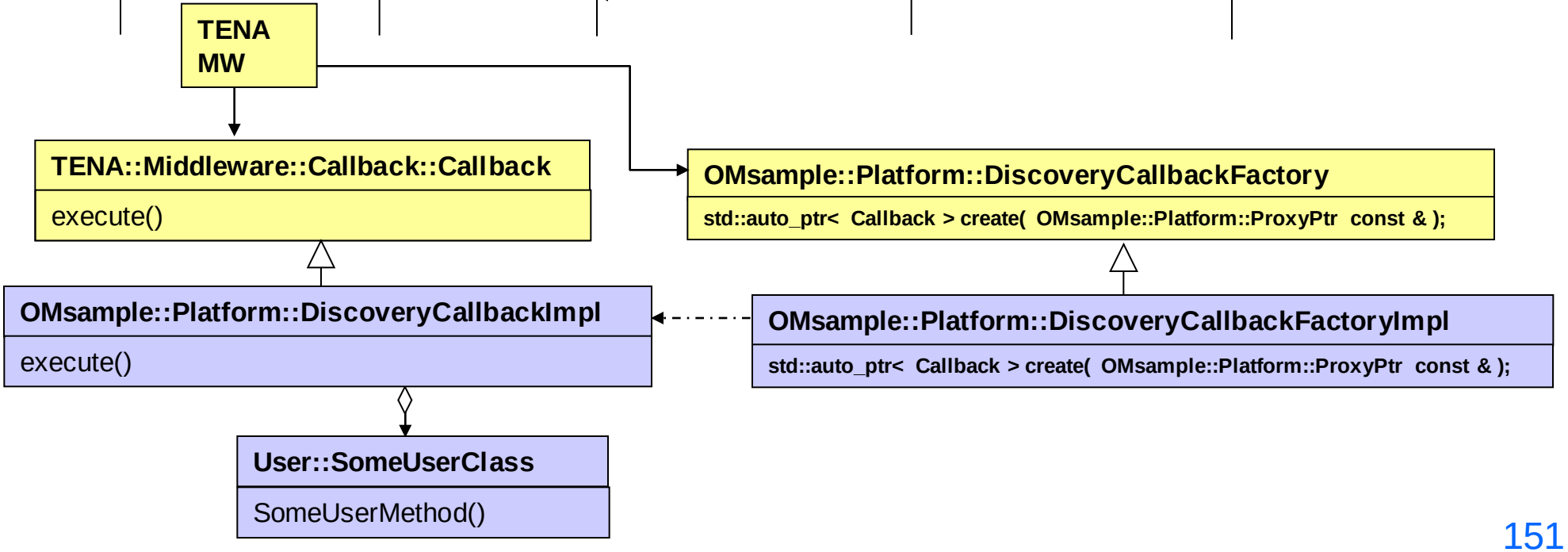
- **The TENA Middleware uses the command pattern to encapsulate callbacks to the application**
- **When the TENA Middleware needs to inform the application of something, it can't immediately call the user's callback code**
  - The TENA Middleware can't rely on the user's code being re-entrant
- **The TENA Middleware needs to use callbacks in the following situations:**
  - New proxy discovered
  - Servant for an already-discovered proxy has been deleted by its server
  - Publication state of already-discovered proxy has been updated



How does the Middleware Get the Callback Commands??

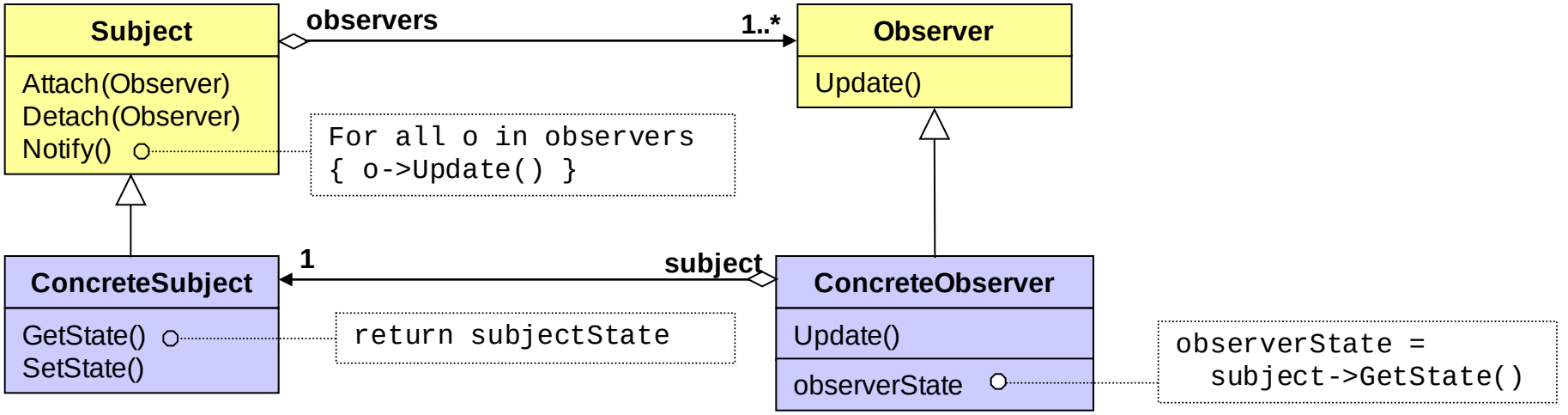
- **By merging the command pattern and the factory method pattern**





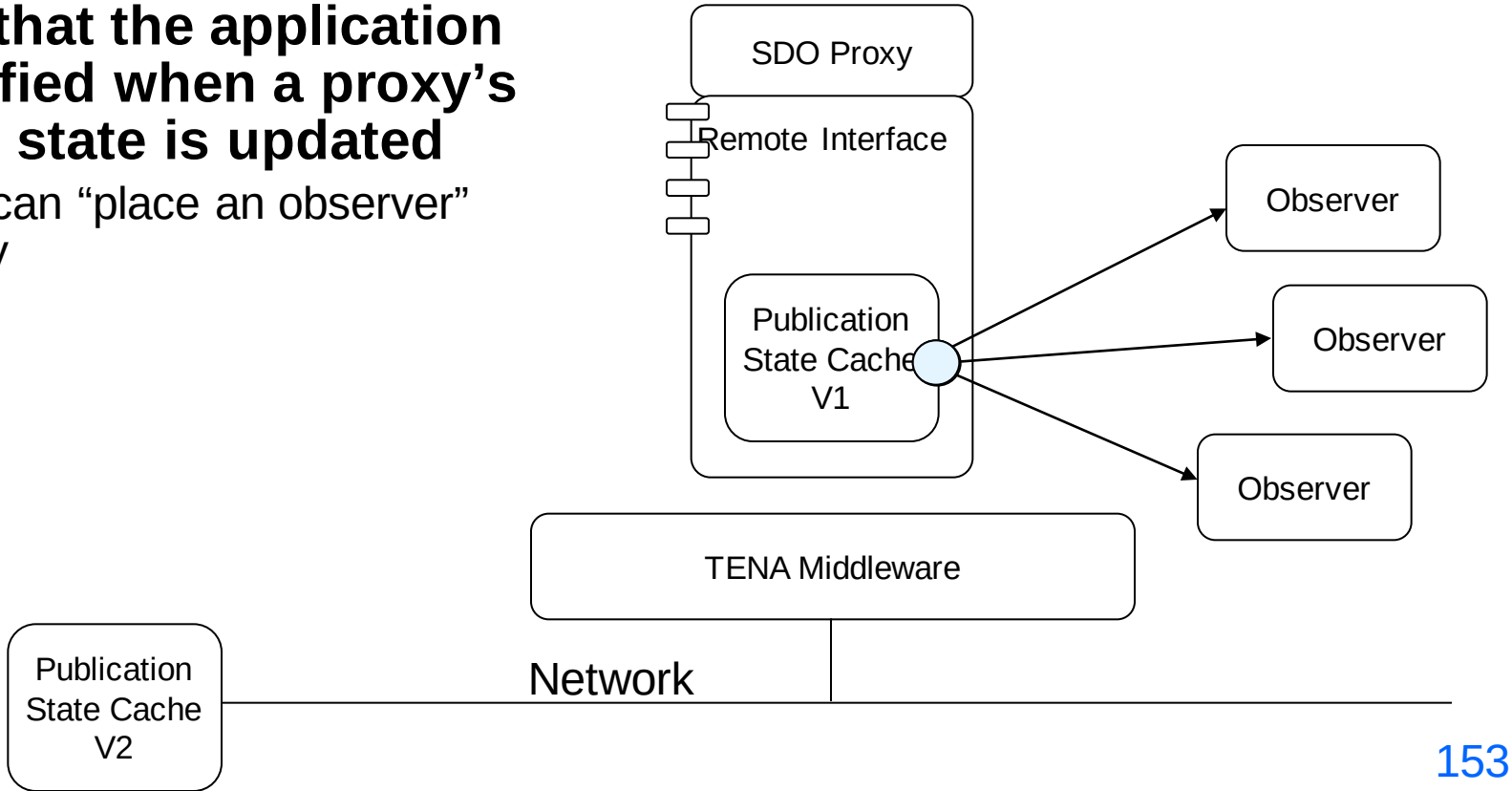
Observer Pattern

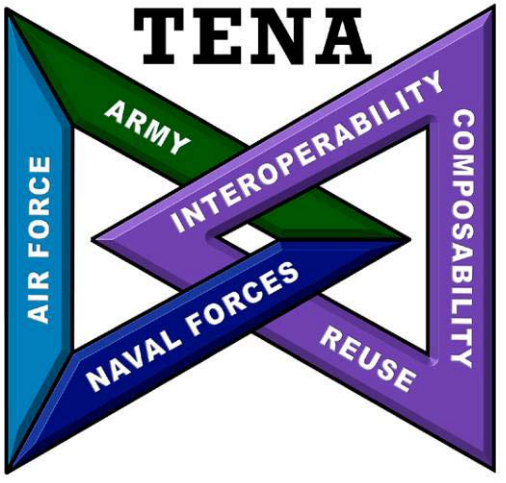
- **Intent**
  - Define a relationship between an object (the “subject”) and a number of other objects (“observers”) such that when the subject object is changed in some way, all observer objects are notified that the subject has changed
- **Motivation**
  - Need the ability to maintain consistency between classes without making the classes tightly coupled
  - Based on a portion of the Smalltalk “Model-View-Controller” paradigm
- **Structure – parallel class hierarchies again**



Observer Pattern in the TENA Middleware

- **An SDO's cached publication state (in proxies) is updated as soon as it is received off of the network**
- **Normally, an application would not know of this update until it next read the publication state**
- **The TENA Middleware provides a notification service so that the application can be notified when a proxy's publication state is updated**
  - The user can "place an observer" on a proxy

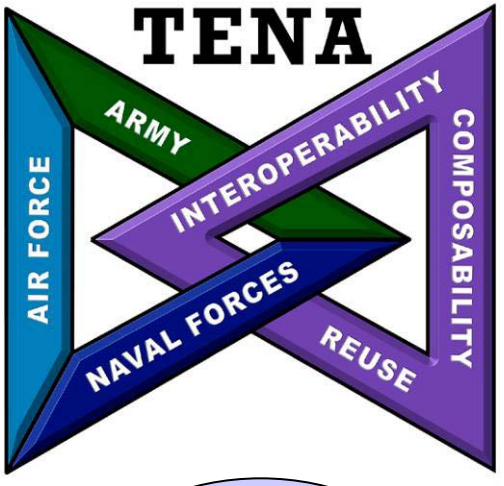


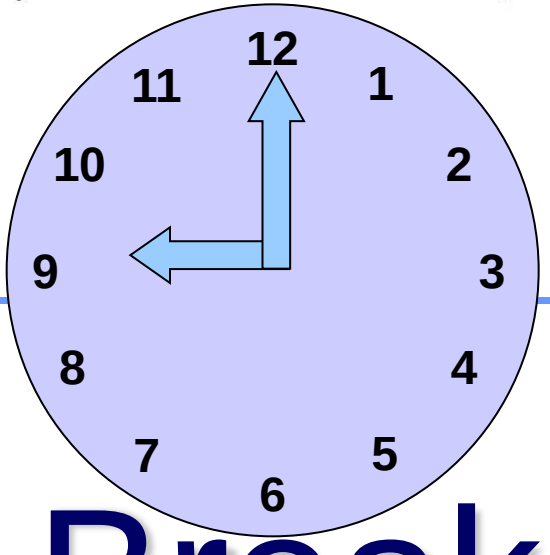

TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE

Questions?

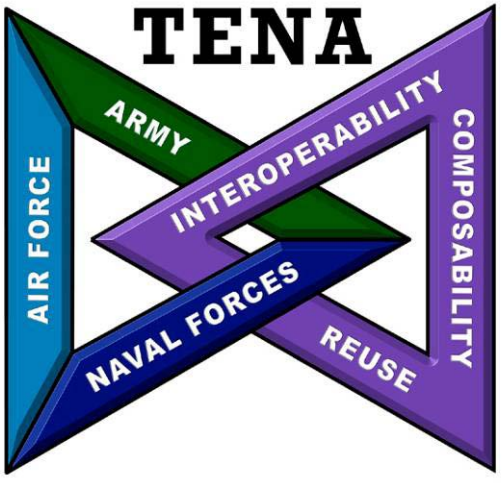

IMPROVING T&E
CTEIP
CAPABILITIES


UNITED STATES
JOINT FORCES
COMMAND
USJFCOM


TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE


12
1
2
3
4
5
6
7
8
9
10
11

Break



TENA Middleware Overview and the Release 5 API

TENA Middleware Overview and the Release 5 API

- **TENA Middleware Software Overview**
- **TENA Executions**
- **Application Development Basics**
- **Execution Management Services**
- **SDO Publication Services**
- **SDO Subscription Services**
- **Callback Services**
- **Local Classes**
- **Messages**
- **SDO Pointers**
- **Downcasting**
- **Diagnostics**

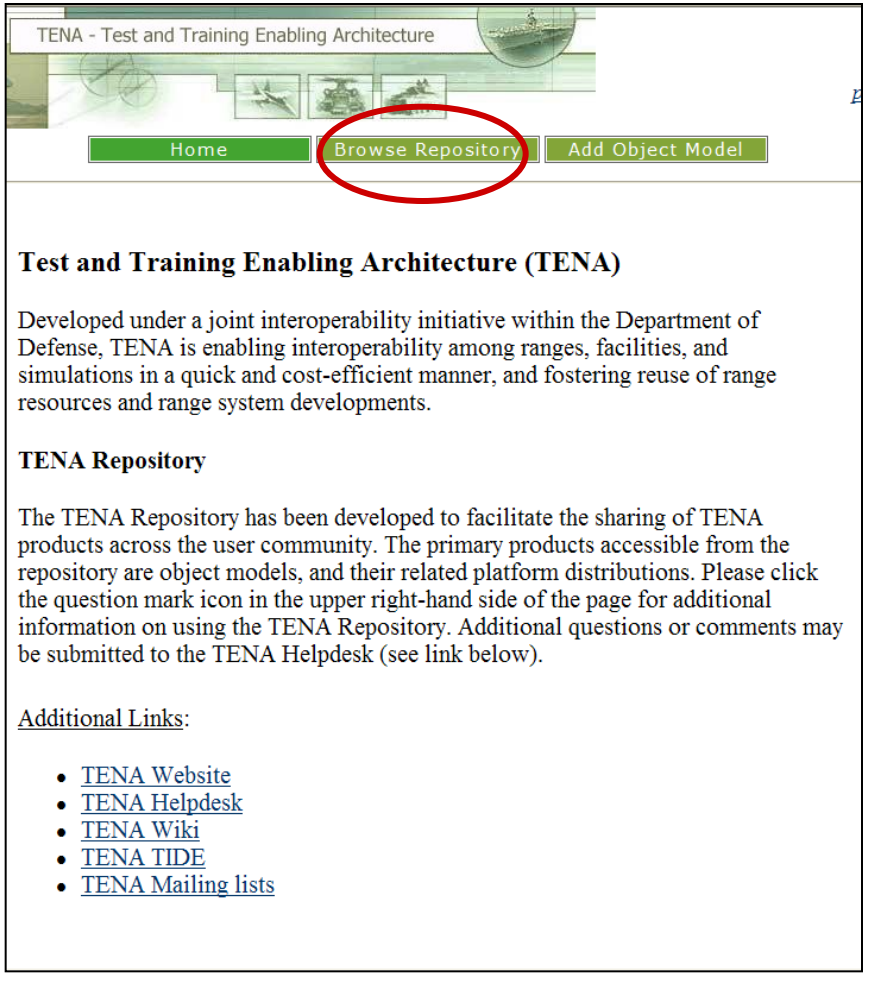
TENA Web Portal
http://www.tena-sda.org/

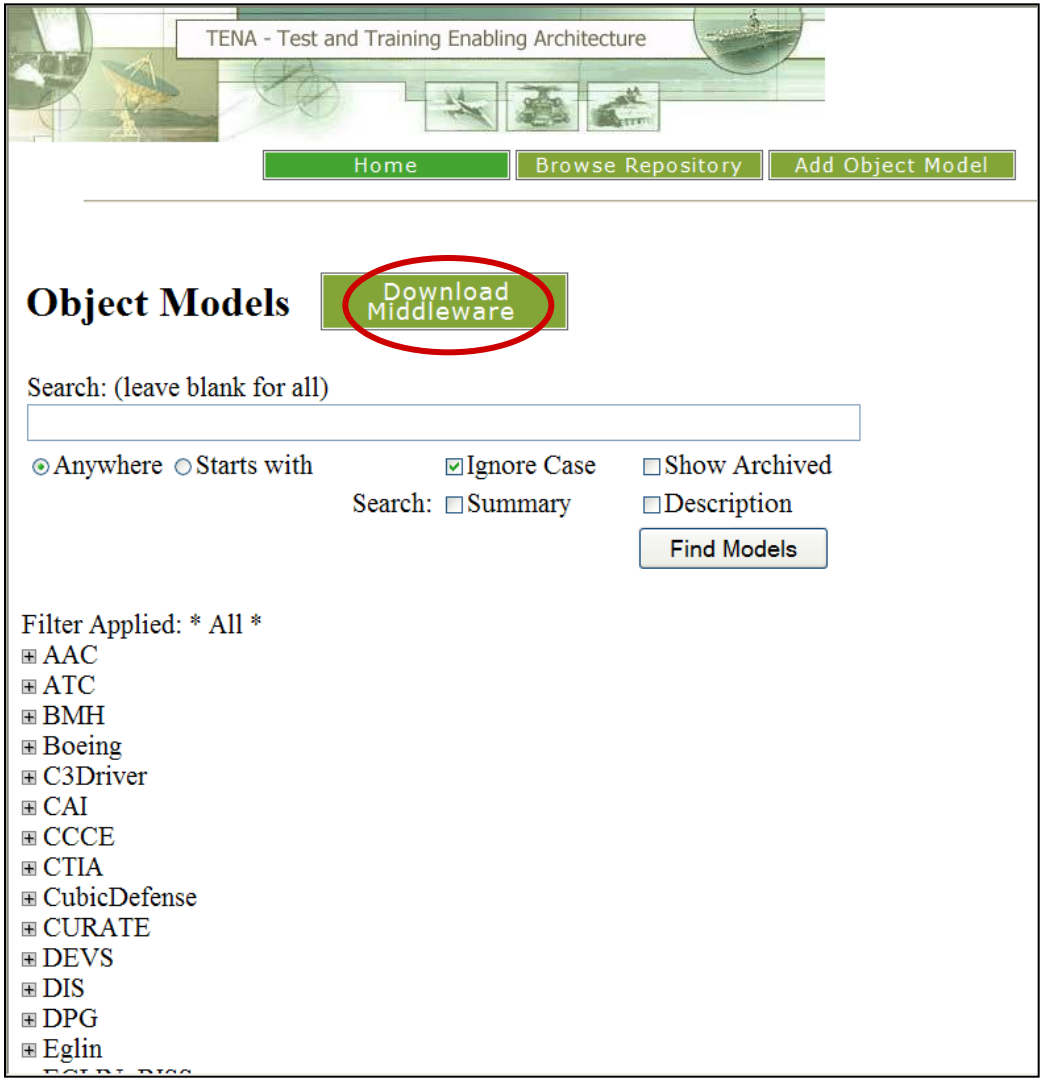
- **Registered user account required**

- **Contains**
  - News
  - Meeting Notices
  - Documentation
  - Middleware
  - Object Models
  - Training Materials

Downloading the Release

http://www.tena-sda.org/repository





Downloading the Release

TENA - Test and Training Enabling Architecture

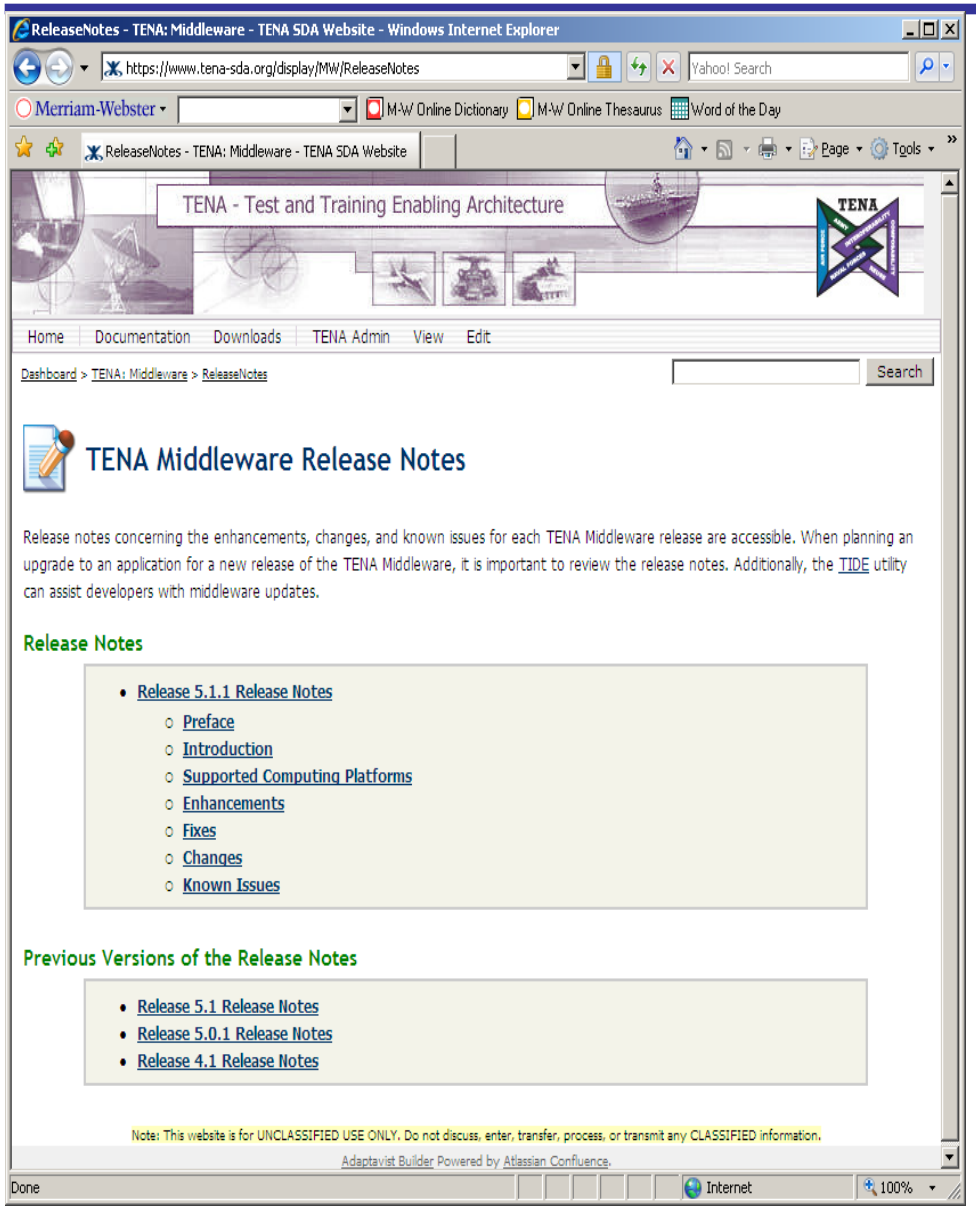
epowell-approved: profile | logout | ?

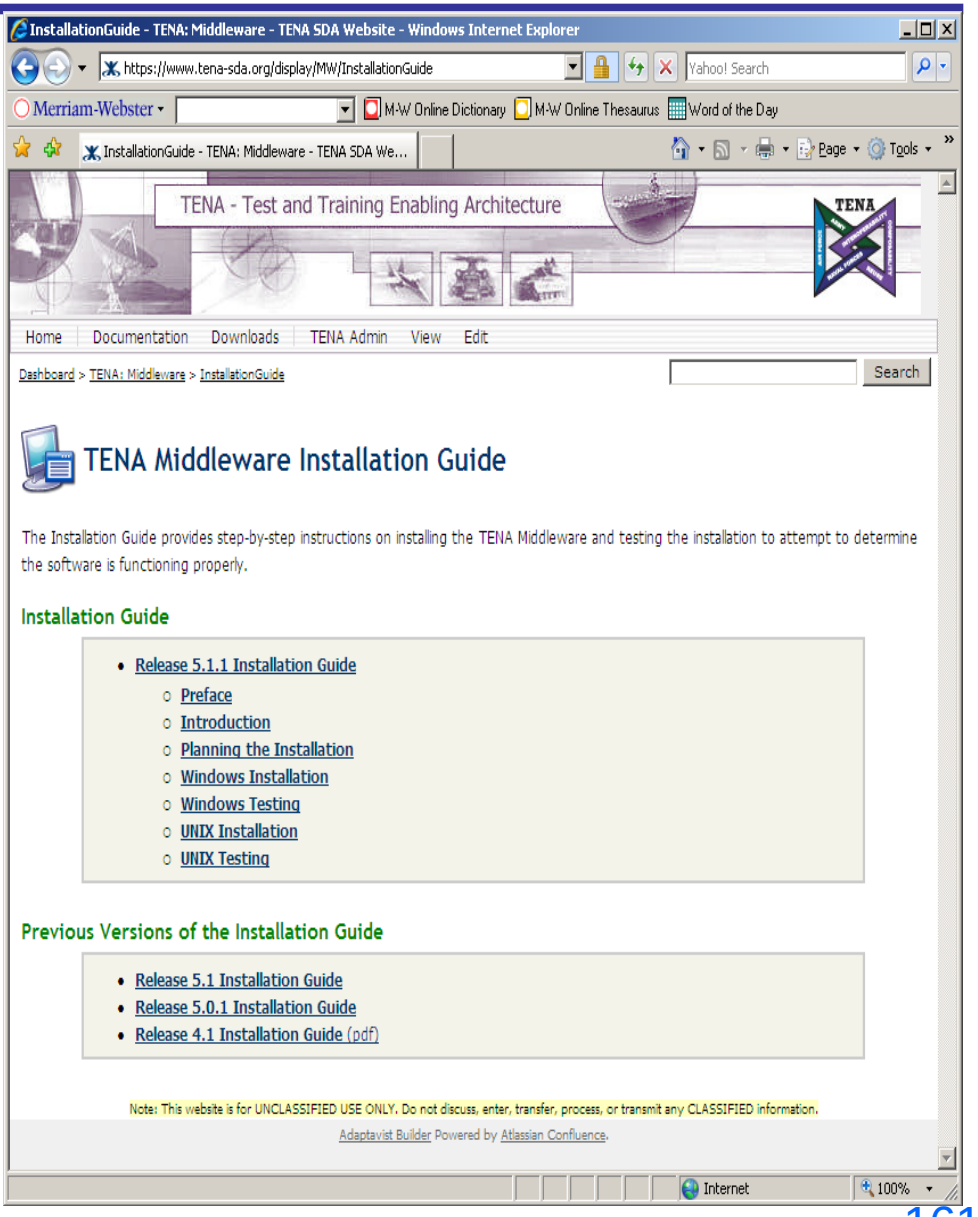
Home | Browse Repository | Add Object Model

Middleware: TENA-v5.1

Platform	Filename		
fc3-gcc343-d	TENA-fc3-gcc343-d-v5.1.bin	Download	(54.496MB)
fc3-gcc343	TENA-fc3-gcc343-v5.1.bin	Download	(27.424MB)
irix65-mipspro742m-d	TENA-irix65-mipspro742m-d-v5.1.bin	Download	(37.801MB)
irix65-mipspro742m	TENA-irix65-mipspro742m-v5.1.bin	Download	(21.75MB)
rh8-gcc32-d	TENA-rh8-gcc32-d-v5.1.bin	Download	(132.572MB)
rh8-gcc32	TENA-rh8-gcc32-v5.1.bin	Download	(40.312MB)
rh9-gcc322-d	TENA-rh9-gcc322-d-v5.1.bin	Download	(132.661MB)
rh9-gcc322	TENA-rh9-gcc322-v5.1.bin	Download	(40.548MB)
rhelws4-gcc343-d	TENA-rhelws4-gcc343-d-v5.1.bin	Download	(54.519MB)
rhelws4-gcc343	TENA-rhelws4-gcc343-v5.1.bin	Download	(27.107MB)
solaris8-gcc323-d	TENA-solaris8-gcc323-d-v5.1.bin	Download	(40.627MB)
solaris8-gcc323	TENA-solaris8-gcc323-v5.1.bin	Download	(23.368MB)
2k-vc71-d	TENA-2k-vc71-d-v5.1.exe	Download	(59.581MB)
2k-vc71	TENA-2k-vc71-v5.1.exe	Download	(24.365MB)
xp-vc71-d	TENA-xp-vc71-d-v5.1.exe	Download	(59.512MB)
xp-vc71	TENA-xp-vc71-v5.1.exe	Download	(24.361MB)

Installing the Middleware

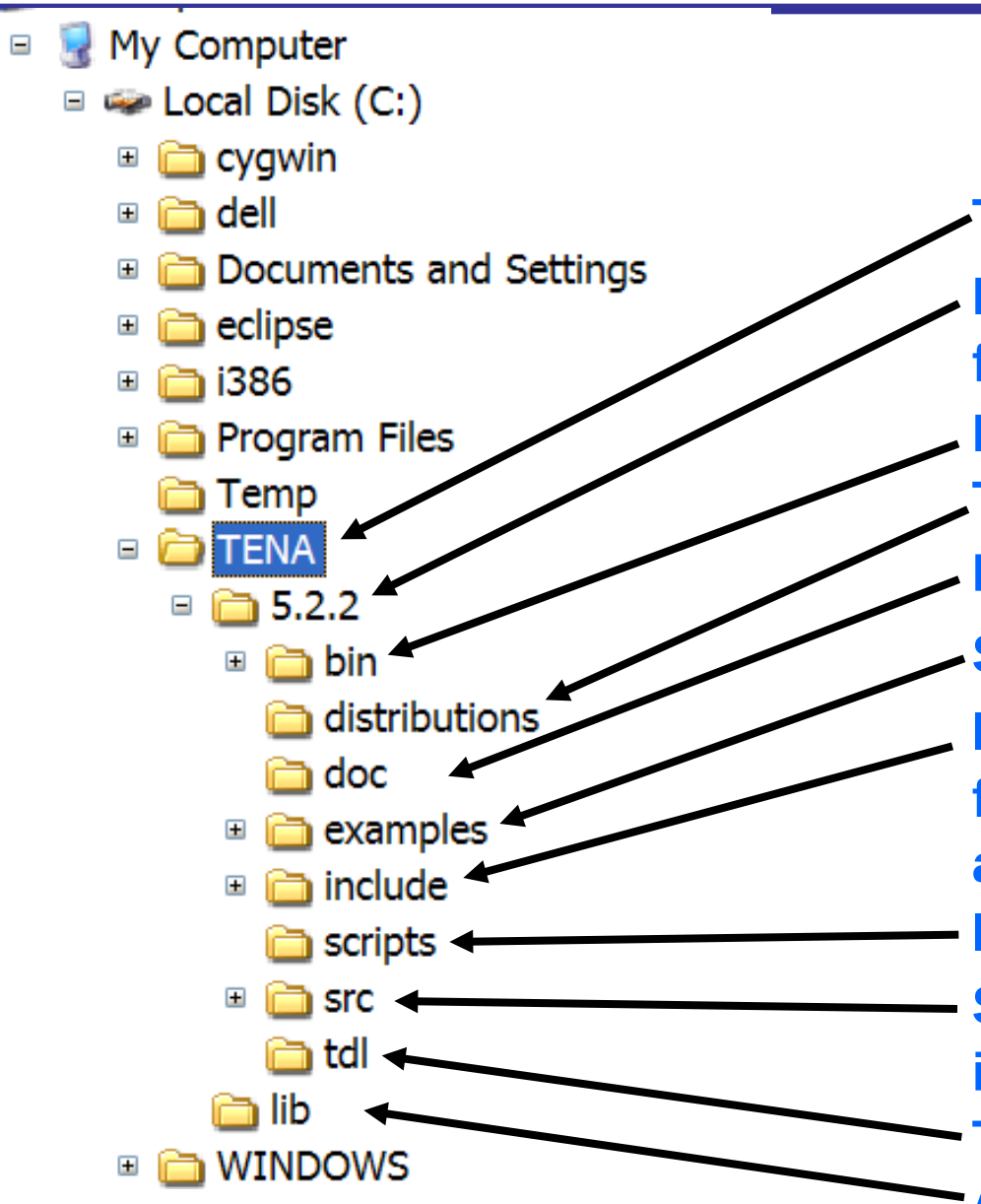




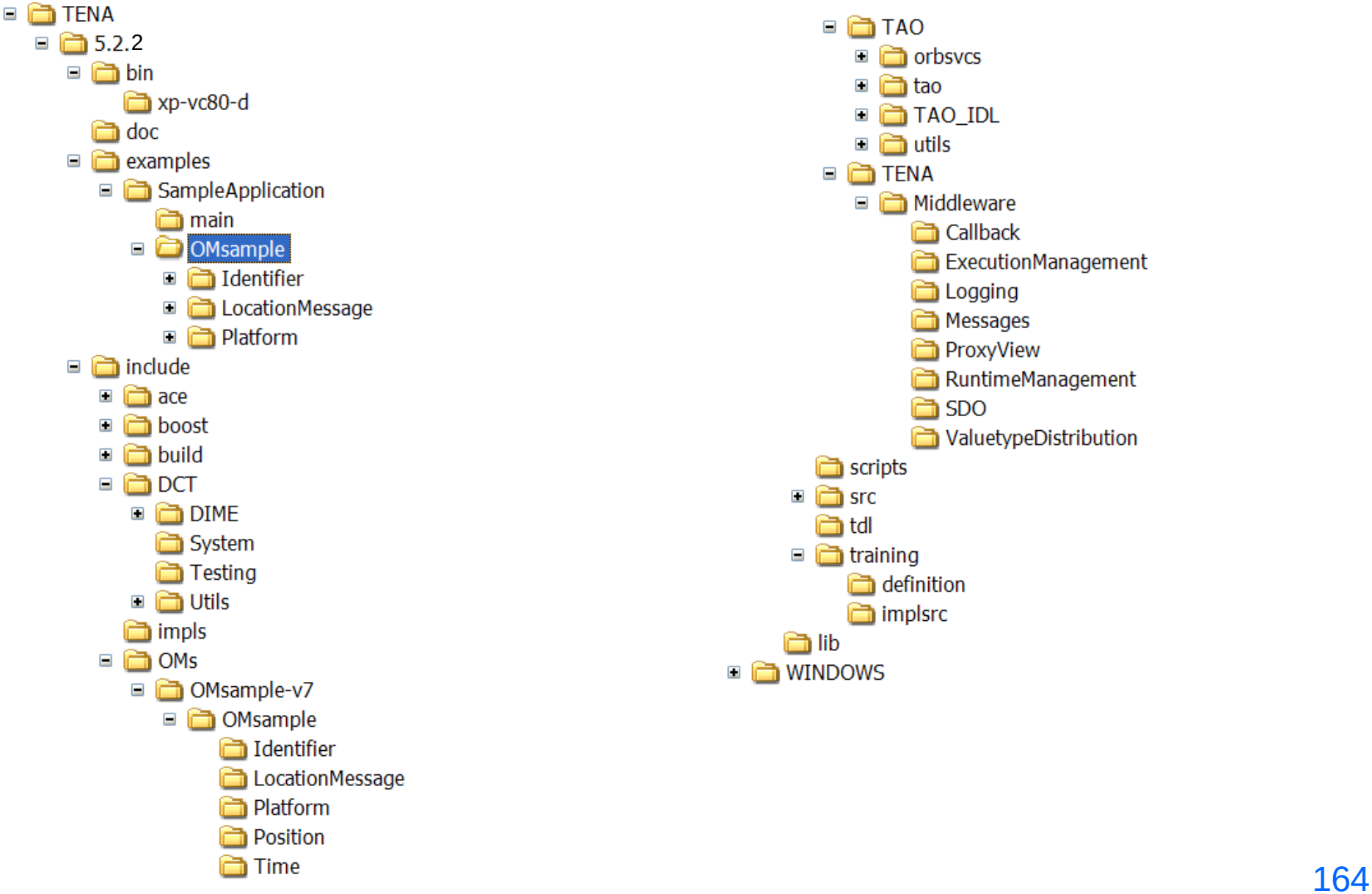
Supported Platforms

- Ardence ETS - NetAcquire (HW integrated Windows Real-Time OS) Microsoft Visual C++ 7.1 (bundled)
- Embedded Planet (embedded Linux OS) GCC 3.2.2 (bundled)
- Linux - Fedora Core 3 GCC 3.4.4 — Support for this platform is ending
- Linux - Fedora Core 4 GCC 4.0.2 — Support for this platform is ending
- Linux - Fedora Core 5 GCC 4.1.1
- Linux - Fedora Core 6 GCC 4.1.1 — New for R5.2.2
- Linux - Fedora Core 6, 64-bit GCC 4.1.1 — New for R5.2.2
- Linux - Red Hat 8 GCC 3.2 — Support for this platform is ending
- Linux - Red Hat 9 GCC 3.2.2 — Support for this platform is ending
- Linux - Red Hat Enterprise Workstation 4 GCC 3.4.4
- Linux - Red Hat Enterprise Linux 5 GCC 4.1.1 — New for R5.2.2
- Linux - SUSE 10.1 GCC 4.1.0
- Mac OS X 10.4.7 GCC 4.0.1 — New for R5.2.2 Universal Binary (Intel and Power PC) support
- Solaris 8 GCC 3.2.3 — Support for this platform is ending
- Solaris 10 Sun SPRO 5.8
- Solaris 10, 64-bit Sun SPRO 5.8
- Windows 2000 Microsoft Visual C++ .NET 2003 (aka Visual C++ 7.1) — Support for this platform is ending
- Windows Server 2003 Standard Microsoft Visual C++ .NET 2003 (Visual C++ 7.1)
- Windows Server 2003 Standard, 64-bit Microsoft Visual C++ .NET 2005 (Visual C++ 8.0) — New for R5.2.2
- Windows XP Microsoft Visual C++ .NET 2003 (Visual C++ 7.1)
- Windows XP Microsoft Visual C++ 2005 (Visual C++ 8.0)
- Windows Vista Microsoft Visual C++ 2005 (Visual C++ 8.0) — New for R5.2.2. User-specific HW/SW testing recommended prior to operational use

Middleware Installation



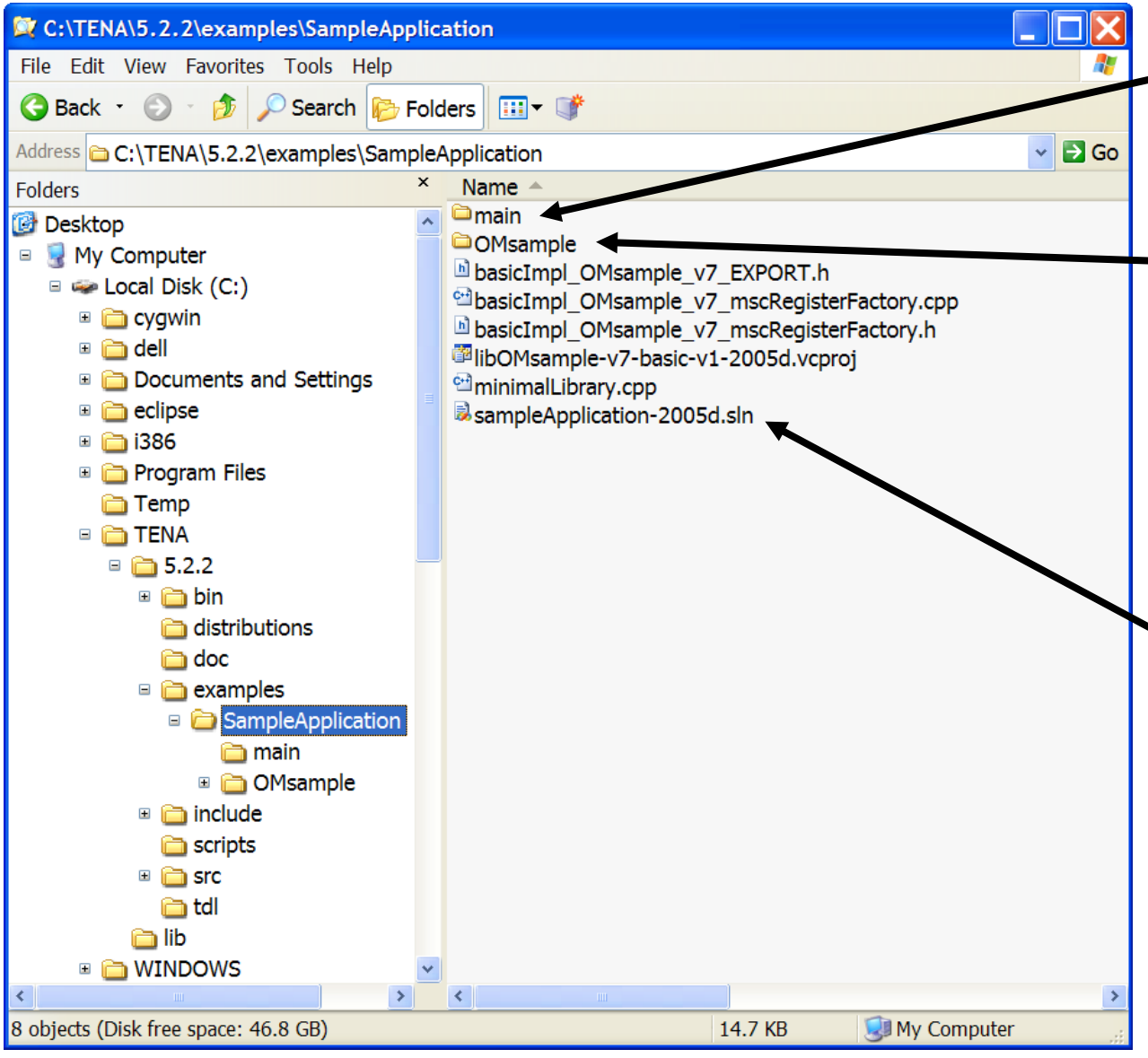
Directory Structure



`sampleApplication` – TENA Release 5.2.2 Example Program

- **Exercises some (but not all) TENA Middleware Features**
- **Comes with all you'll need to build it**
- **After installation is complete, build the `sampleApplication` executable**
- **Launch the Network Naming Service and Execution Manager**
- **Launch `sampleApplication`**
- **What `sampleApplication` does**
  - Joins the Execution
  - Publishes and Subscribes to a Platform SDO
  - Sets the initial values of the Platform's state
  - Goes into a loop changing state values
  - After loop completion, resigns from the Execution

Look into the `SampleApplication` Folder

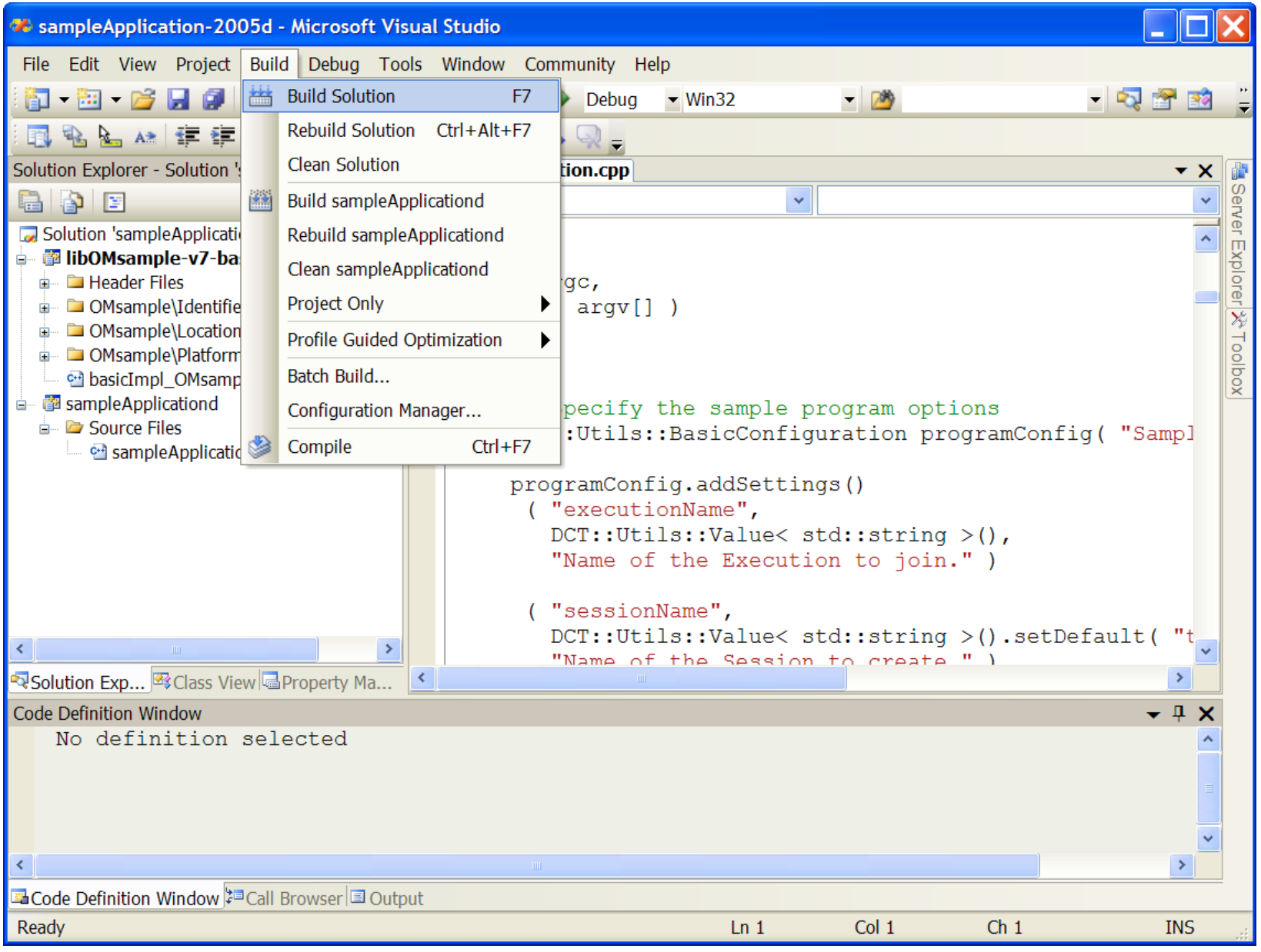


Where the sample application main program exists

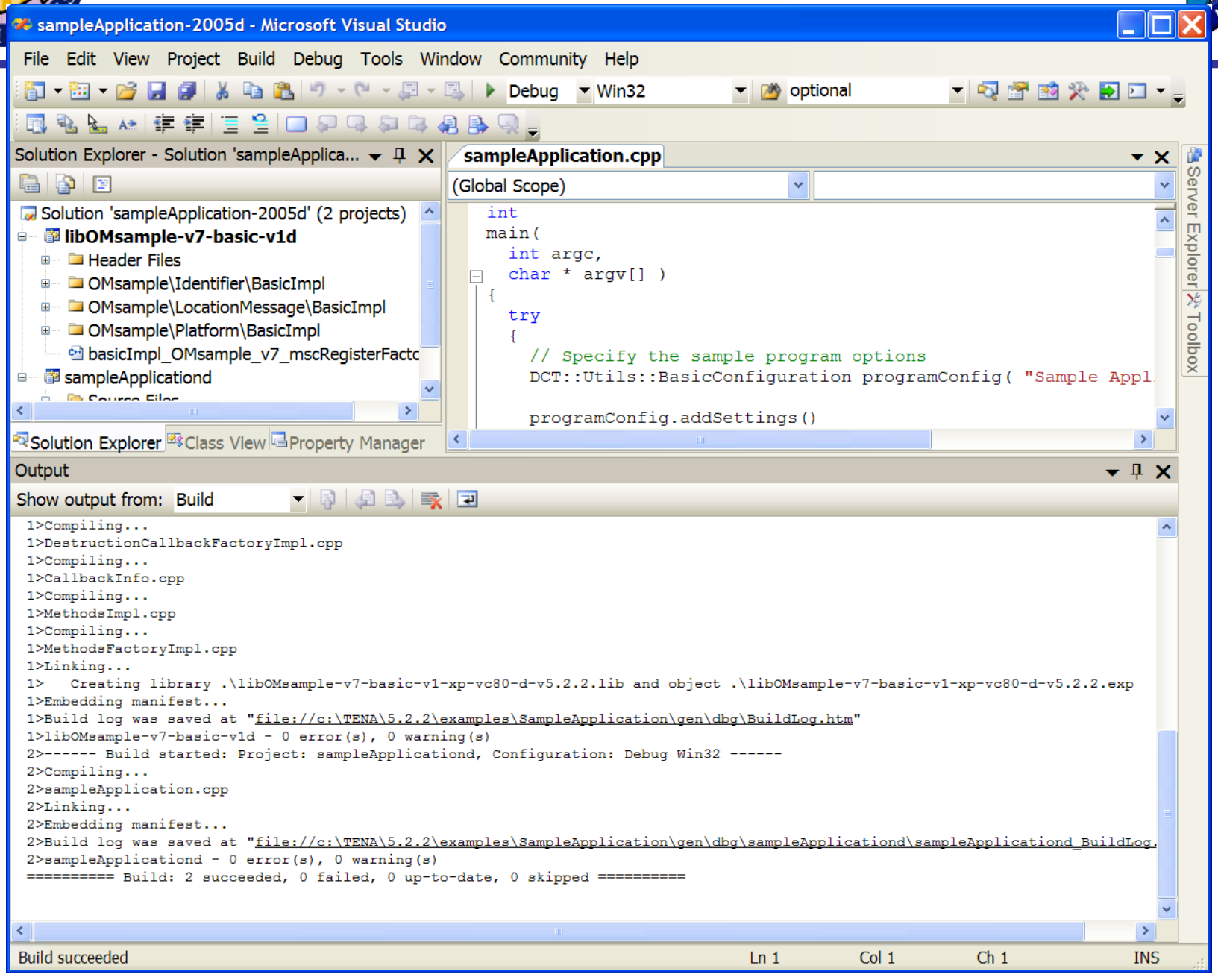
Where the object-model-implementation-specific software is located

Open the solution file (Windows)

Build **sampleApplication**



Build Results

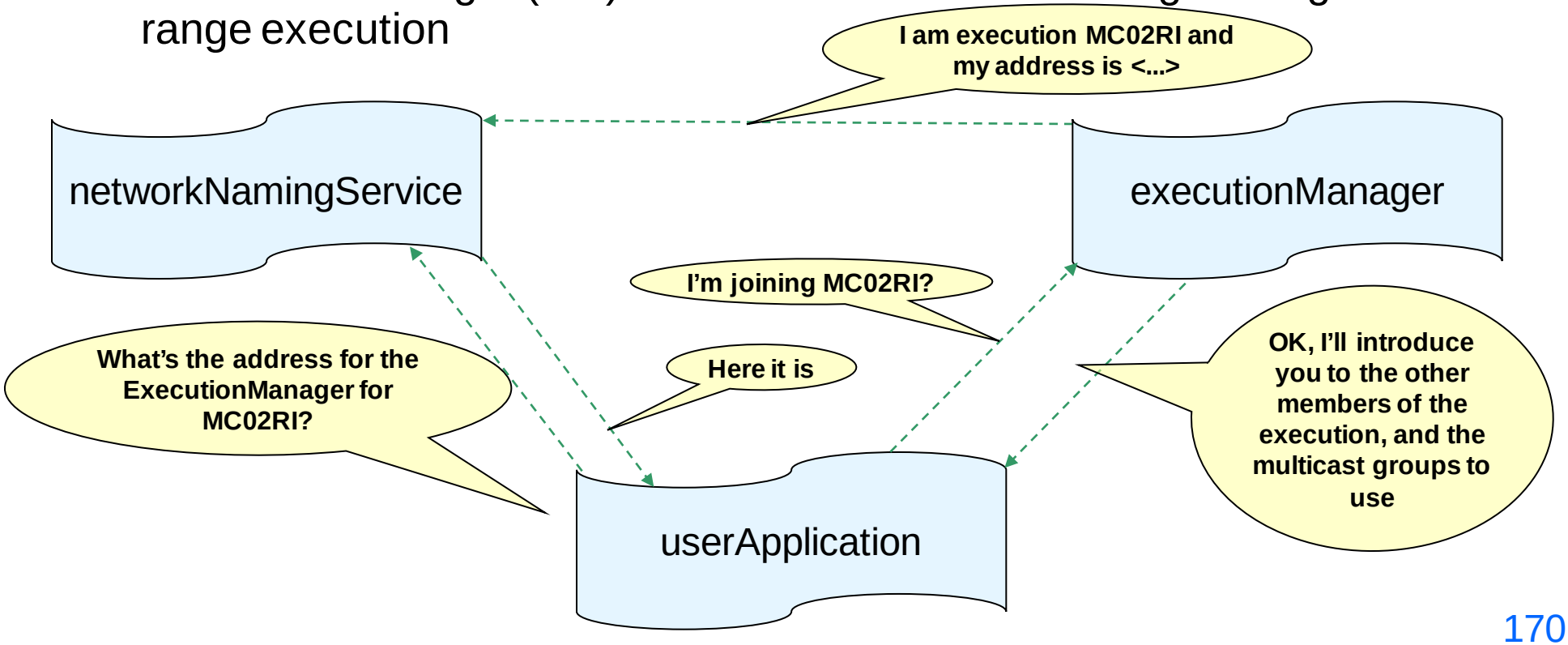


TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Logical Range Execution Applications

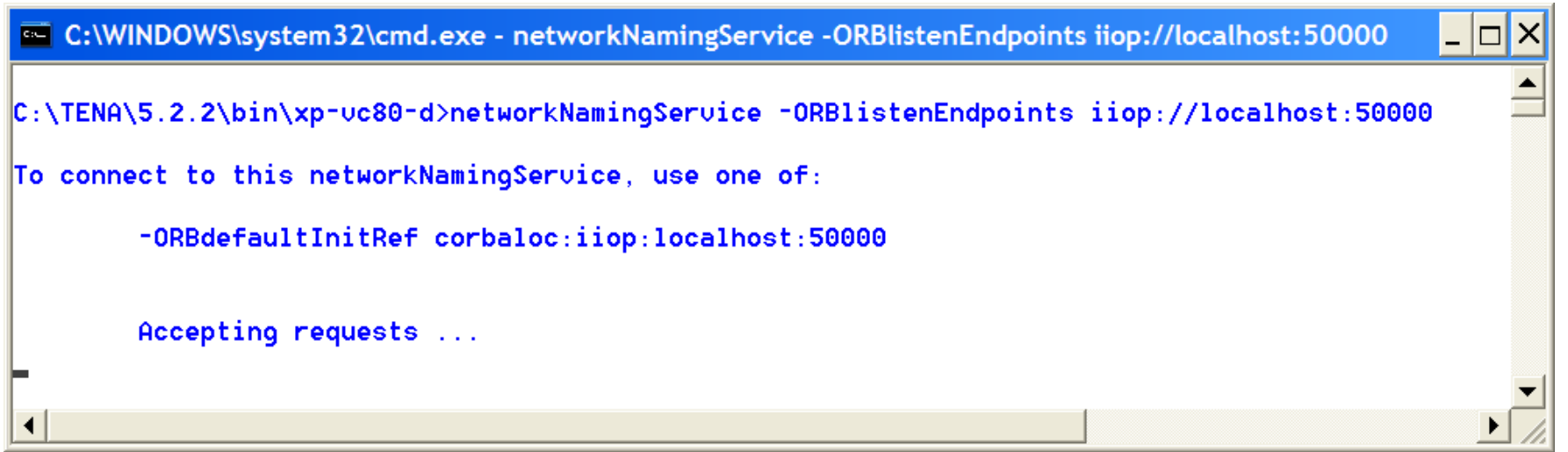
- **Logical Range Execution – the name for a given instance of a logical range**
- **Two Logical Range-wide applications provide the logical range with important functionality**
  - Network Naming Service (NNS) – Global directory
  - Execution Manager (EM) – contains information on a given logical range execution



networkNamingService (NNS)

- **Supports the requirement that each Execution must have a unique name**
- **Only one NNS process is required on any network**
  - Multiple independent NNS may co-exist if they use different "endpoints"
  - Endpoint = **`iiop://<host name or IP address>:<port>`**
- **The first step in creating an Execution is for the user to start the NNS process, if it is not already running**
  - Manually provide it with the "endpoint" reference

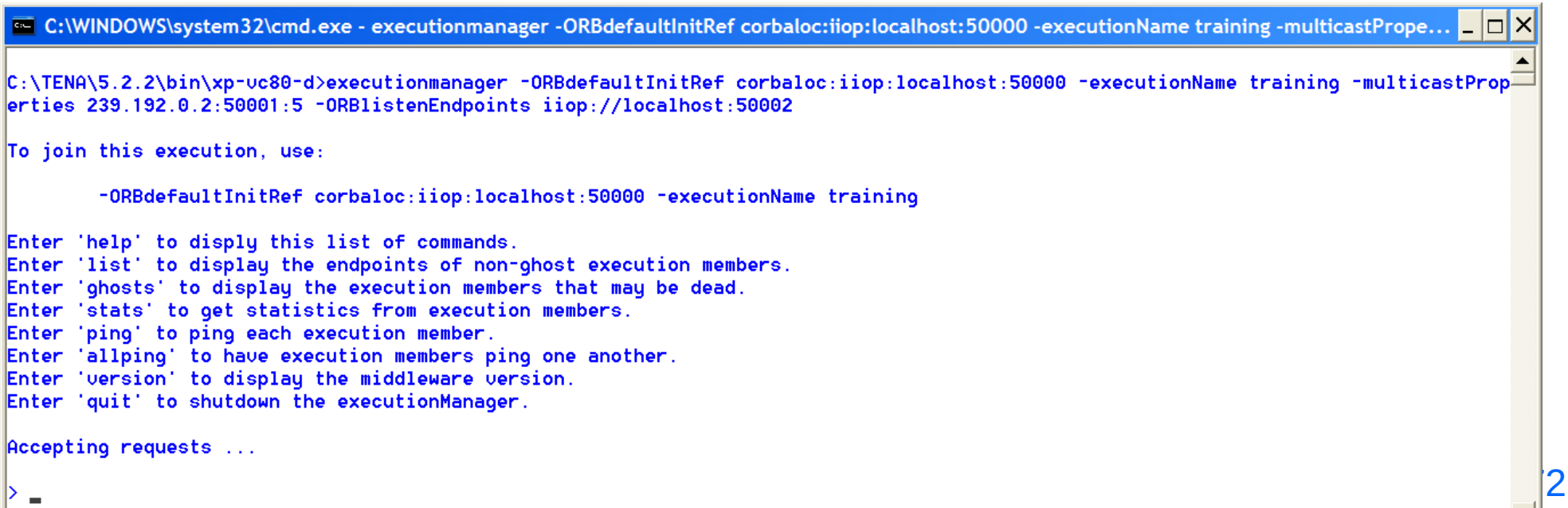
    `-ORBlistenEndpoints iiop://<host name or IP addr>:<port>`



executionManager (EM)

- **An executionManager process exists for each execution**
  - Keeps track of execution-wide information
  - Only used when applications join, resign, or change subscriptions; it is NOT used for data distribution
- **The second step in creating an execution is to manually start the EM for the specific LRE**

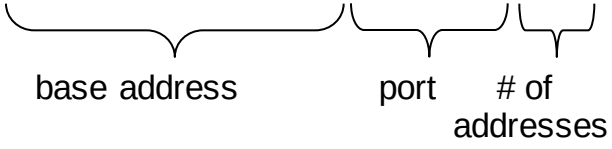
```
executionManager
   -ORBdefaultInitRef corbaloc:iiop:<Hostname>:<portNumber>
   -executionName <ExecutionName>
   [ -multicastProperties <baseMulticastAddress>:<portNumber>:
      <numberOfMulticastAddresses> ]
```



Provide the Execution Manager with Enough Multicast Groups

- **If you are going to use UDP multicast to disseminate SDO state, you must provide the Execution Manager with enough multicast addresses**
- **Currently, state information for each SDO type and interface is sent to a different multicast address (imperfect hashing algorithm is used)**
  - If you have 9 SDOs, 1 interface, and 1 message in your Object Model, you should provide the Execution Manager with at least 11 multicast addresses
- **You must provide multicast addresses in a contiguous block, for example:**

```
-multicastProperties 239.192.1.1:54321:11
```



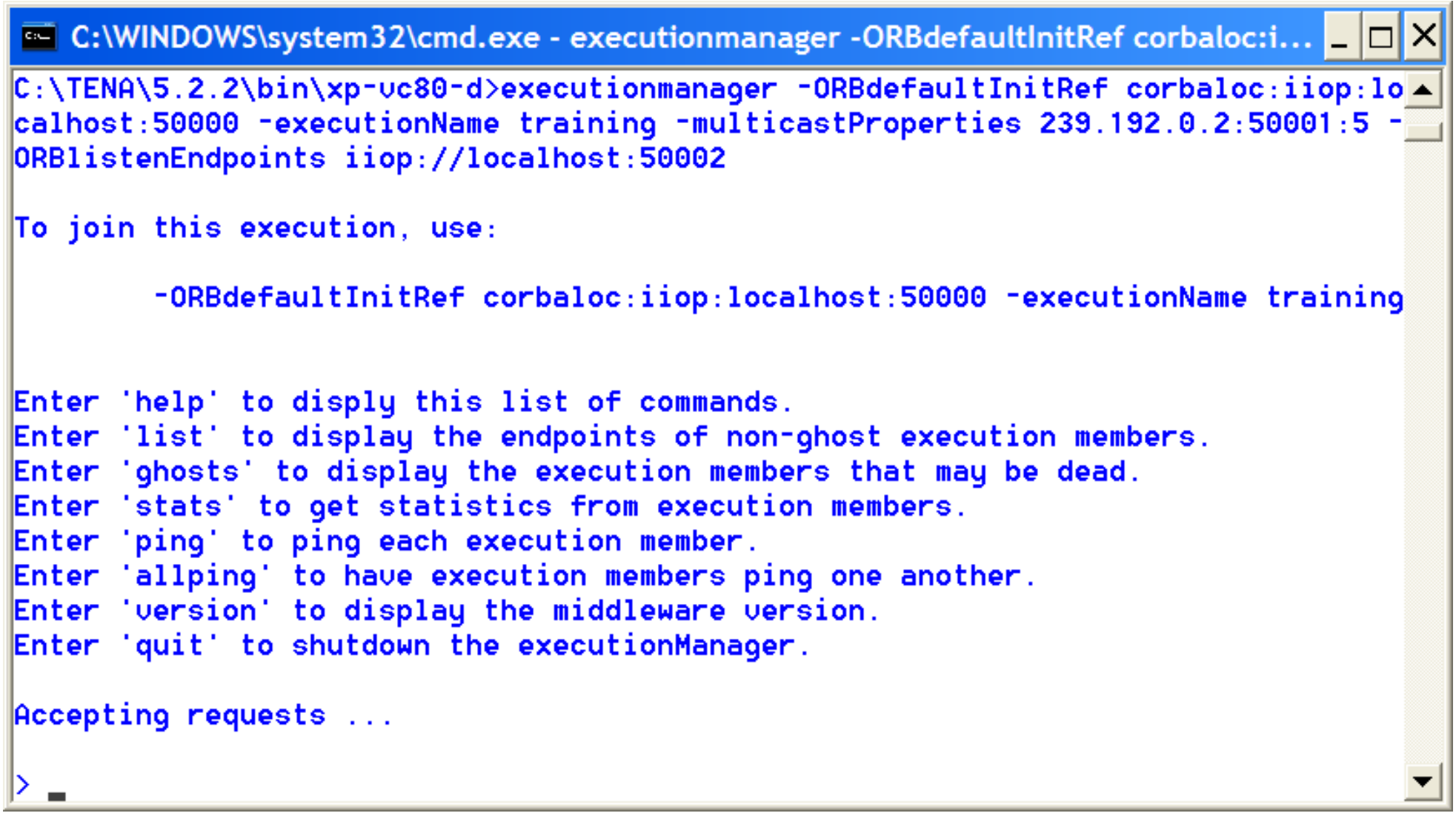
  - this provides addresses 239.192.1.1 through 239.192.1.11
- **IANA (www.iana.org) advises that multicast addresses range between 239.192.0.0 and 239.255.255.255 and that port numbers range between 49152 and 65535**
  - To be perfectly conformant, multicast addresses should be in the blocks between

```
239.192.000.000-239.251.255.255 Organization-Local Scope [Meyer,RFC2365]
239.255.000.000-239.255.255.255 Site-Local Scope [Meyer,RFC2365]
```

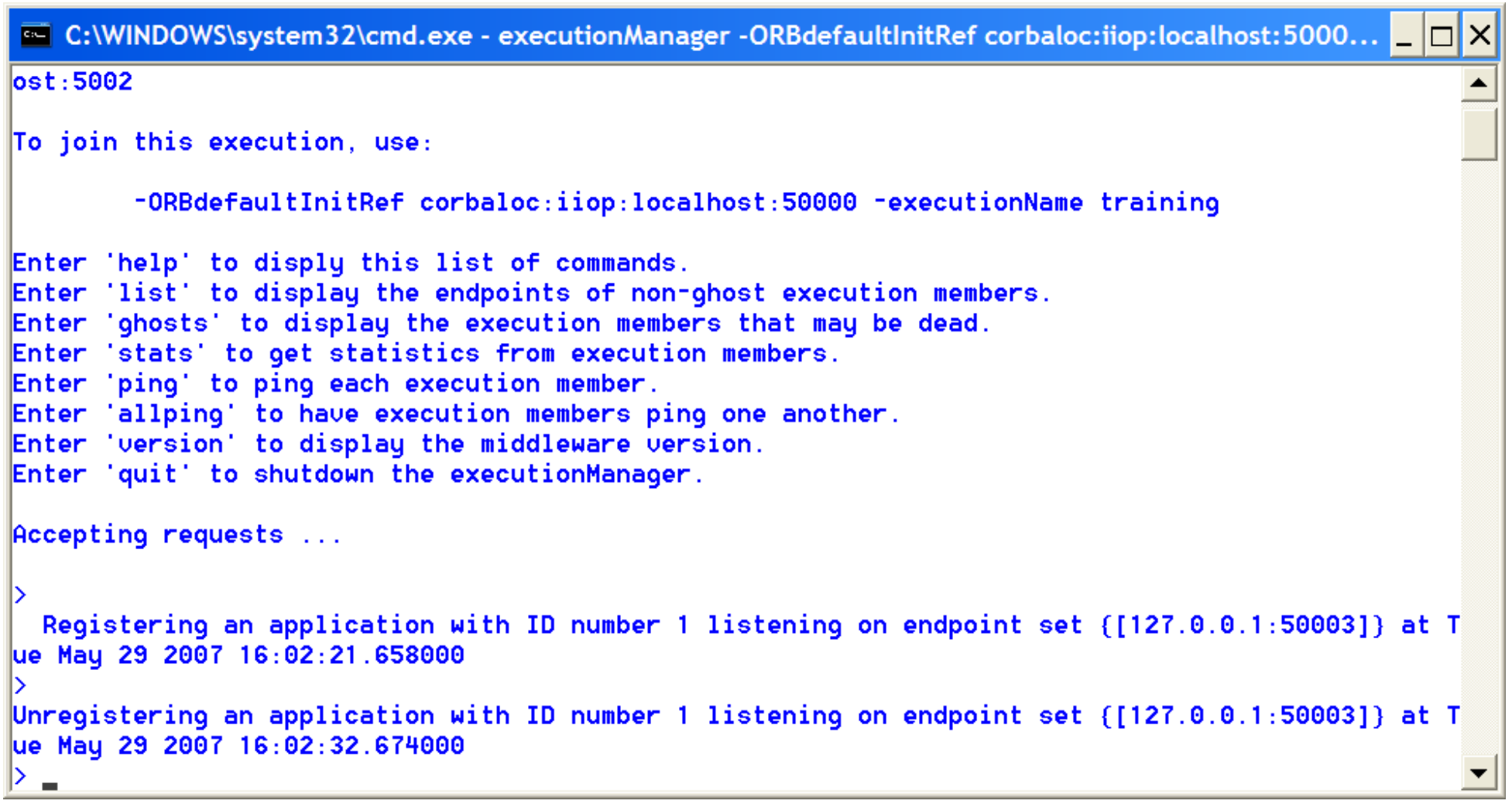
Firewall Detection

- **The Execution Manager verifies that each TENA application attempting to join an execution is able to receive network traffic**
- **Any application that is unable to receive network traffic is prevented from joining**
- **The Execution Manager will try for 2 seconds (default) to contact a joining execution**
  - This time can be changed using the TENA_MW_CONNECTION_TIMEOUT environment variable or the **`-connectionTimeout`** option
- **If a firewall is present, and the application can't receive network traffic, the user needs to:**
  - open the appropriate ports on the firewall
  - specify to the application behind the firewall the port to listen on using the **`-ORBlistenEndpoints`** option discussed previously
- **Look at the TENA Middleware FAQ for more info**
  - https://www.tena-sda.org/doc/5.1/FAQ/pg_faq.html

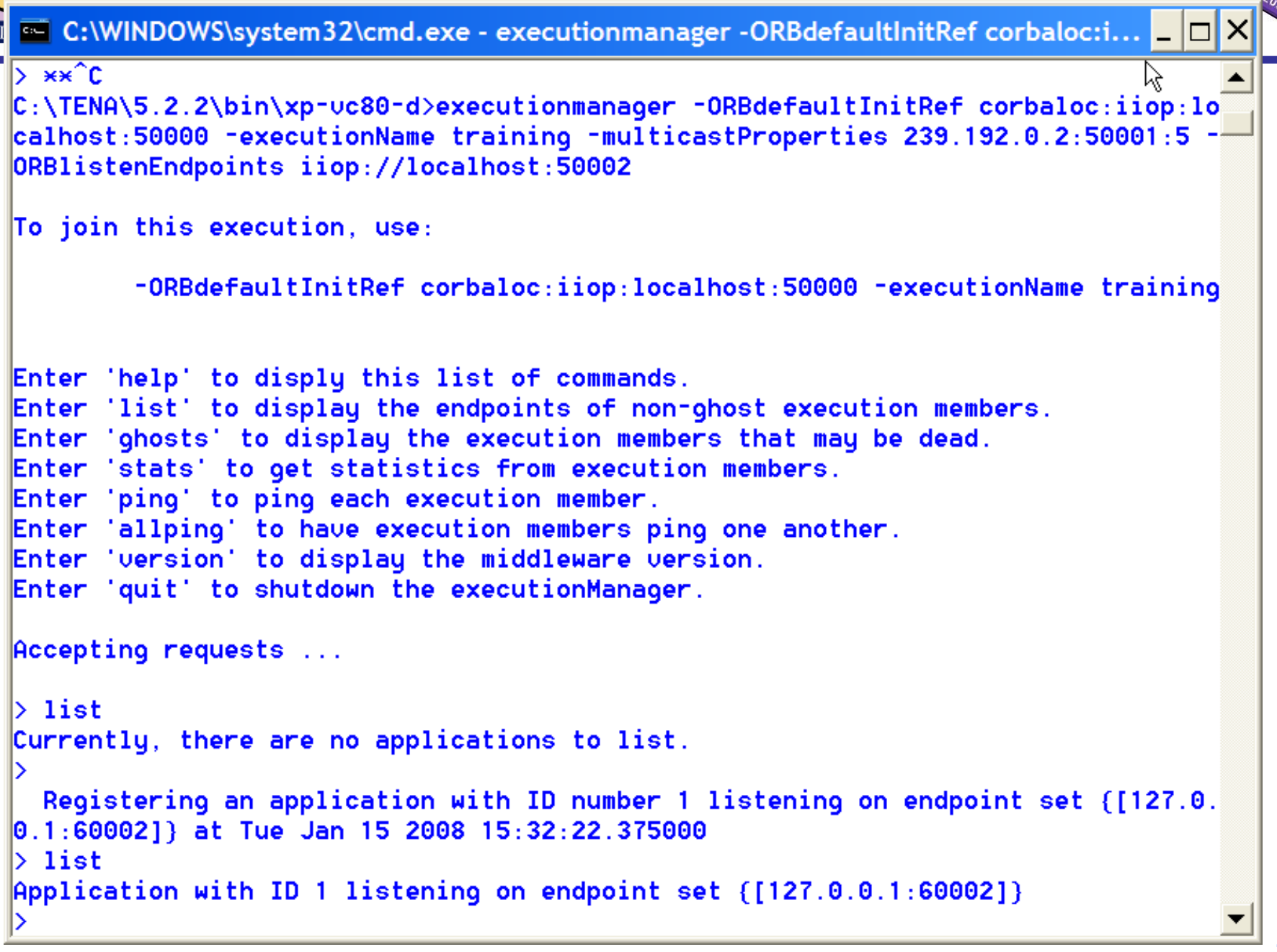
Execution Manager Takes Commands From the User



Execution Manager Records When Applications Join



List


C:\WINDOWS\system32\cmd.exe - executionmanager -ORBdefaultInitRef corbaloc:i...
> **^C
C:\TENA\5.2.2\bin\xp-vc80-d>executionmanager -ORBdefaultInitRef corbaloc:iiop:localhost:50000 -executionName training -multicastProperties 239.192.0.2:50001:5 -ORBlistenEndpoints iiop://localhost:50002
To join this execution, use:
-ORBdefaultInitRef corbaloc:iiop:localhost:50000 -executionName training
Enter 'help' to disply this list of commands.
Enter 'list' to display the endpoints of non-ghost execution members.
Enter 'ghosts' to display the execution members that may be dead.
Enter 'stats' to get statistics from execution members.
Enter 'ping' to ping each execution member.
Enter 'allping' to have execution members ping one another.
Enter 'version' to display the middleware version.
Enter 'quit' to shutdown the executionManager.
Accepting requests ...
> list
Currently, there are no applications to list.
>
Registering an application with ID number 1 listening on endpoint set {[127.0.0.1:60002]} at Tue Jan 15 2008 15:32:22.375000
> list
Application with ID 1 listening on endpoint set {[127.0.0.1:60002]}
>

Stats

```
To join this execution, use:

        -ORBdefaultInitRef corbaloc:iiop:localhost:50000 -executionName training

Enter 'help' to disply this list of commands.
Enter 'list' to display the endpoints of non-ghost execution members.
Enter 'ghosts' to display the execution members that may be dead.
Enter 'stats' to get statistics from execution members.
Enter 'ping' to ping each execution member.
Enter 'allping' to have execution members ping one another.
Enter 'version' to display the middleware version.
Enter 'quit' to shutdown the executionManager.

Accepting requests ...

>
  Registering an application with ID number 1 listening on endpoint set {[127.0.0.1:50003]} at T
ue May 29 2007 16:09:30.565000
> stats
Statistics for 1 Applications:
for {[127.0.0.1:50003]}:
        for type OMsample::Platform
                received: 2
                received_ignored: 0
                sent_discovery: 1
                sent_discovery_point_to_point: 1
                sent_update: 2

>
```

Diagnostics Log

- **All messages from the Middleware are redirected to a file named `diagnosticsLog-YYYY-MM-DD-HHMMSS-PID.txt`**
  - Note that this file is created in the directory where the application starts. That directory must be writable, or the Middleware will fail to initialize by throwing a **`std::runtime_error`** exception from **`TENA::Middleware::init()`**
- **The first line of every diagnostics log file is the name of the application, as reported by `argv[0]`. Every entry in the diagnostics log file is time-stamped.**
- **To redirect the log to the console, use the `–noDiagnosticsLog` command line option**

TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Object Model Distributions

- **It is important to understand how to use the code generated when an object model has been submitted and built for you**
- **We use as an example the USTB object model**

- **Distributions are given to the users in two (2) files**
  - The Definition File
  - The Basic Implementation File

What the OM Distributions Look Like

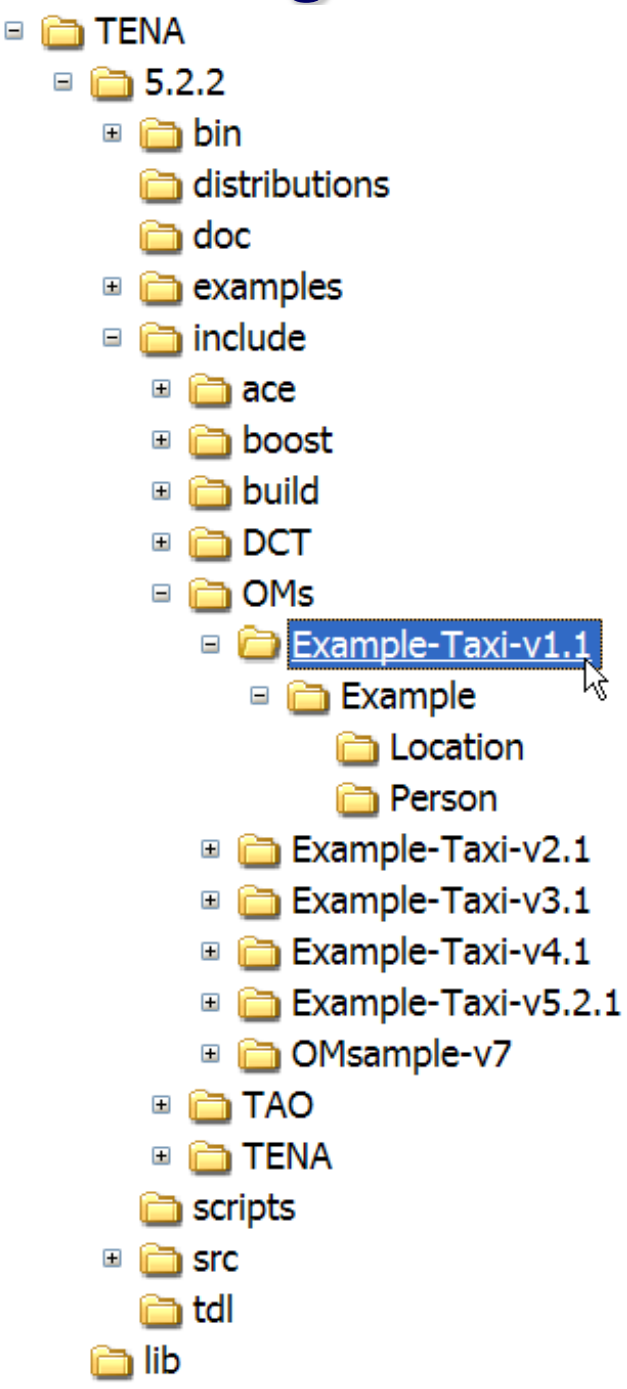
Distributions: TENA-v5.2.2

Select visible platforms

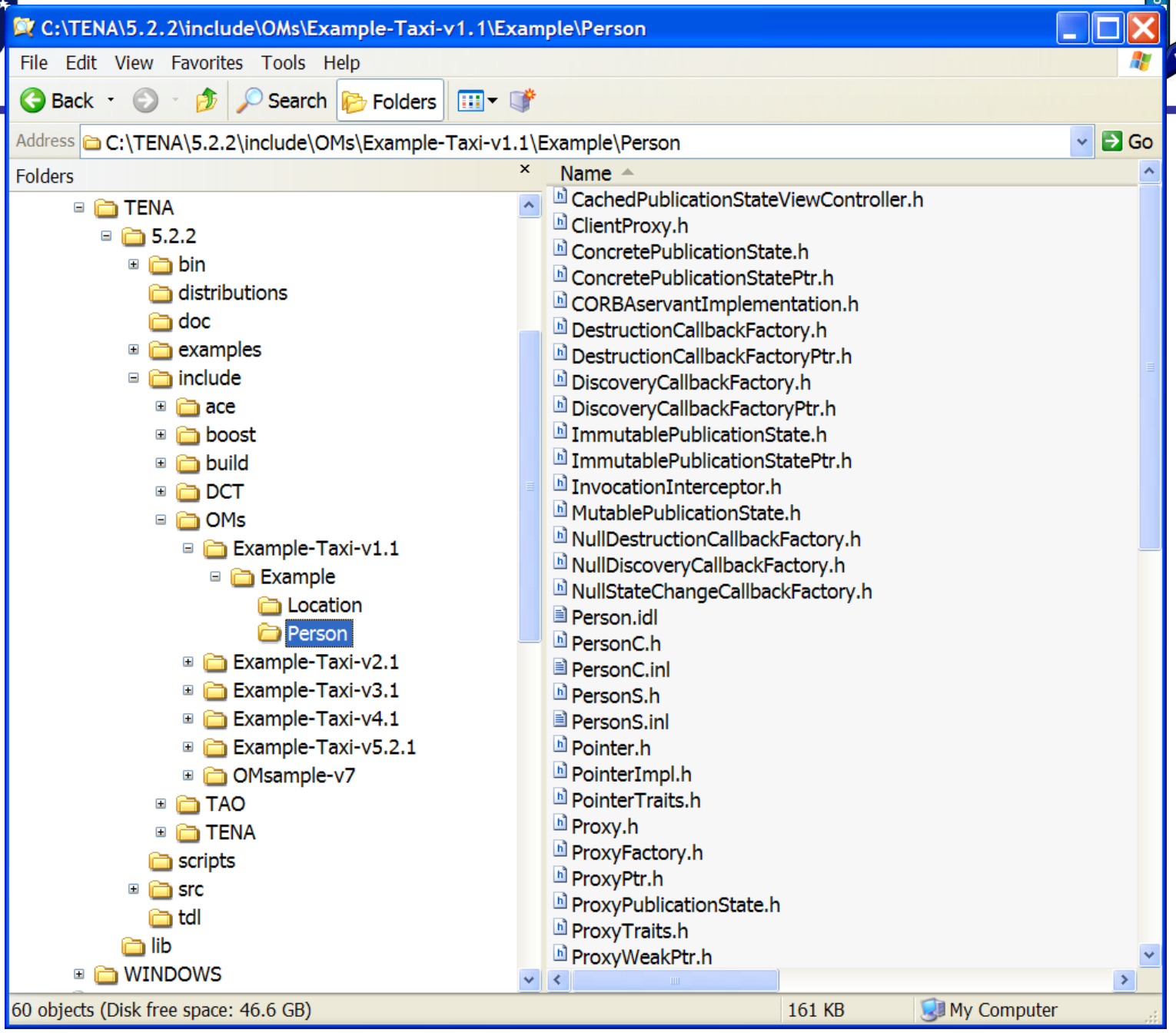
Several older platforms have been deprecated. To request access to a deprecated platform, please open a helpdesk case.

Distribution	Name	Version	Platform	State
Definition	---		fc5-gcc41-d	Download (1.136MB) details
Source Implementation	Basic	v1	fc5-gcc41-d	Download (68.014KB) details
Definition	---		fc5-gcc41	Download (1.277MB) details
Source Implementation	Basic	v1	fc5-gcc41	Download (71.532KB) details
Definition	---		fc6-gcc41-64-d	Download (1.161MB) details
Source Implementation	Basic	v1	fc6-gcc41-64-d	Download (68.014KB) details
Definition	---		fc6-gcc41-64	Download (1.298MB) details
Source Implementation	Basic	v1	fc6-gcc41-64	Download (71.532KB) details
Definition	---		fc6-gcc41-d	Download (1.142MB) details
Source Implementation	Basic	v1	fc6-gcc41-d	Download (68.014KB) details
Definition	---		fc6-gcc41	Download (1.284MB) details
Source Implementation	Basic	v1	fc6-gcc41	Download (71.532KB) details
Definition	---		rhel5-gcc41-d	Download (1.142MB) details
Source Implementation	Basic	v1	rhel5-gcc41-d	Download (68.014KB) details
Definition	---		rhel5-gcc41	Download (1.284MB) details
Source Implementation	Basic	v1	rhel5-gcc41	Download (71.532KB) details
Definition	---		rhelws4-gcc34-d	Download (1.118MB) details
Source Implementation	Basic	v1	rhelws4-gcc34-d	Download (68.014KB) details
Definition	---		rhelws4-gcc34	Download (1.232MB) details
Source Implementation	Basic	v1	rhelws4-gcc34	Download (71.532KB) details
Definition	---		rh9-ppc82xxgcc322	Download (409.824KB) details
Source Implementation	Basic	v1	rh9-ppc82xxgcc322	Download (68.348KB) details
Definition	---		osx104-gcc40-d	Download (2.511MB) details

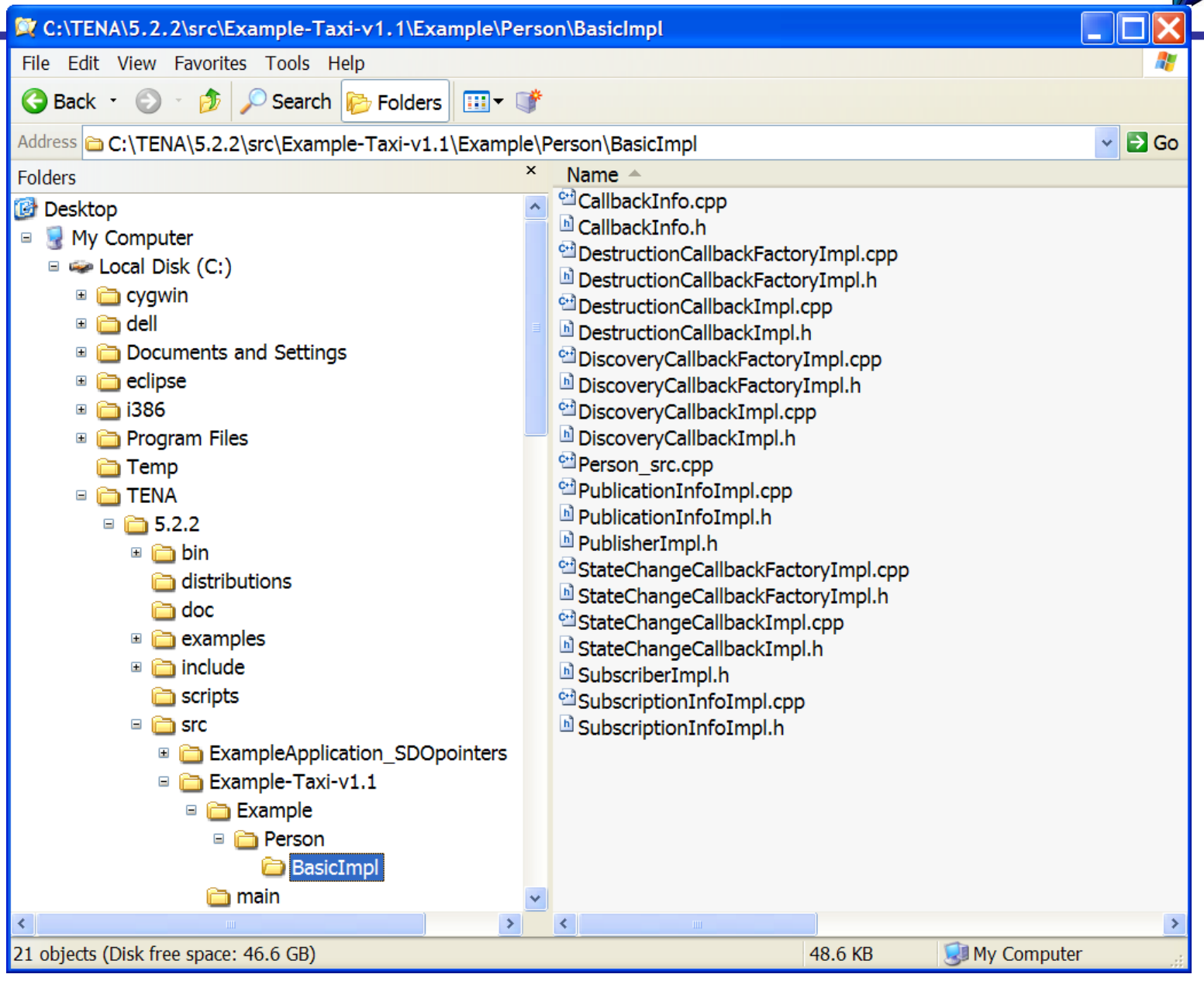
Installing the OMs



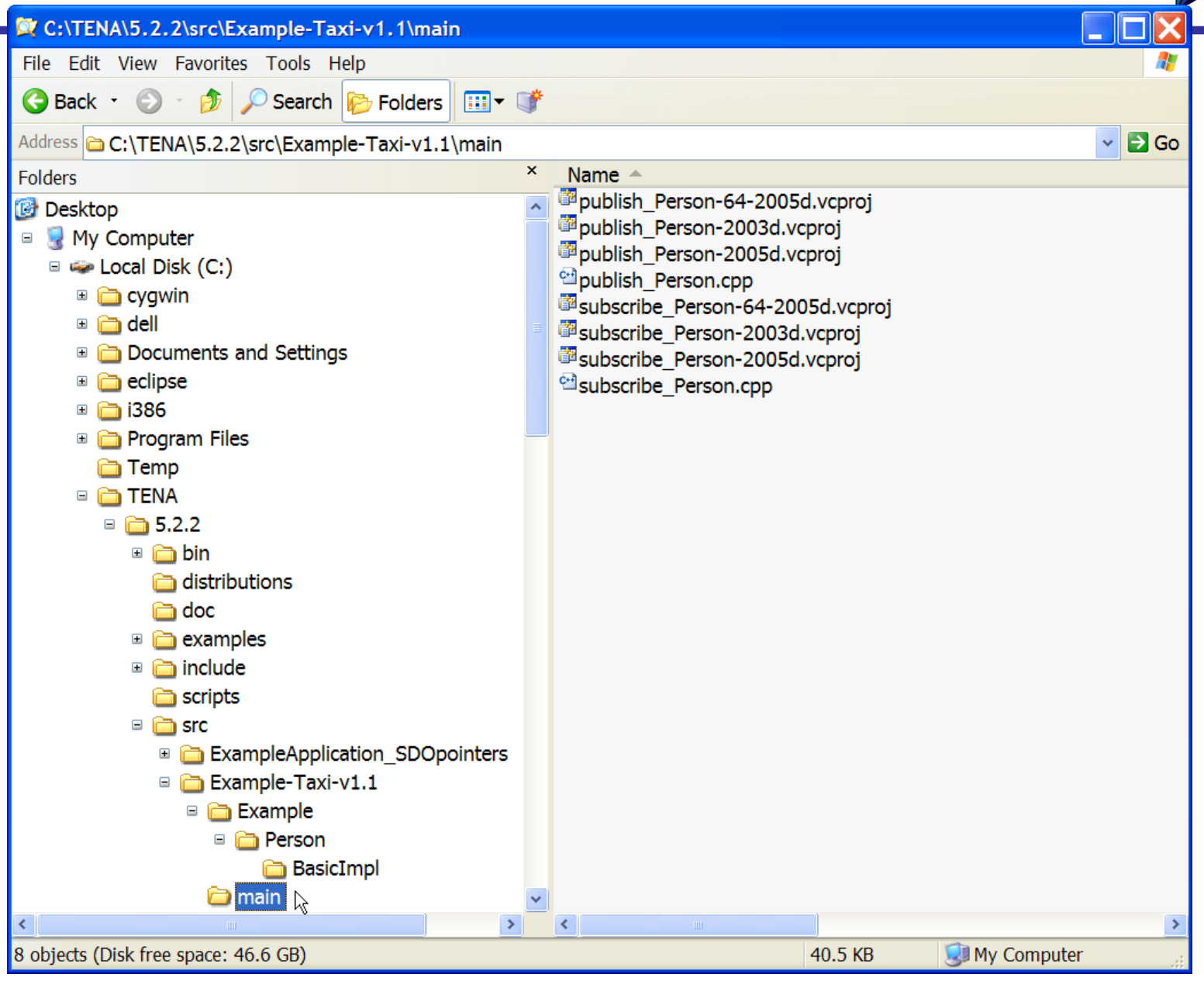
The OM Definition



The OM Basic Implementation



The OM Test Programs



TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Understanding the TENA Middleware API Calls

- **Many TENA Middleware API calls consist of instantiating a particular object and initializing it with an initial value that is the return value from a function**
- **C++ review**
  - `<type_name> <variable_name> ( <initial_value> );`

    `int i (1);` ←OR→ `float x ( sqrt (2) );`
  - `<type_name> <variable_name> = <initial_value>;`

    `int i = 1;` ←OR→ `float x = sqrt (2);`
- **Example:**

```
OMsample::Platform::ServantFactoryPtr
   pServantFactory (
    pSession->
      createServantFactory<OMsample::Platform::ServantTraits>(
       pPublicationInfo ) );
```

Steps to Join an Execution

- **Create a `TENA::Middleware::Configuration` object, passing in parameters**
- **Initialize the `TENA::Middleware::Runtime` by calling the `TENA::Middleware::init()` function**
- **Using this `Runtime` object, create an `Execution` object that represents your `Execution`**
  - Each execution is bound together by a single logical range object model
  - An application may join multiple executions if it wants (e.g., if it wants to "bridge" two different executions, or monitor two different executions, etc.)
- **Using the `Execution` object, get a `Session` object**
  - A "`Session`" object is an interaction point for an application with an execution
  - `Session`s contain publication/subscription information
  - Applications may use multiple sessions for each execution if they wish

API to Join an Execution

```
std::string executionName ( “MC02RI” );
std::string sessionName ( “MySession” );

// Construct new Configuration object
// Assumes argc, argv contain ORBdefaultInitRef info
TENA::Middleware::Configuration config (
   argc, argv, “config.txt”, “test” );

// Get pointer to Runtime object
TENA::Middleware::RuntimePtr pRuntime (
      TENA::Middleware::init( config ) );

// Get Execution
TENA::Middleware::ExecutionPtr pExecution (
      pRuntime->joinExecution( executionName ) );

// Get Session
TENA::Middleware::SessionPtr pSession (
      pExecution->createSession( sessionName ) );
```

TENA::Middleware::Configuration

- **Constructor:**

```
TENA::Middleware::Configuration::Configuration (
  int & argc,
  char ** argv,
  std::string const & filename = "",
  std::string const & prefix = "")
```

- **Function:**
  - parses options from command line, environment variables (starting with "**TENA_MW_**") and a file
  - All options are parsed using a case-insensitive match
  - Command Line arguments are parsed first and take precedence over the same argument specified in environment variables and/or the file
    - Unless the option takes multiple values, then all will be valid
  - Environment variables have "**TENA_MW_**" and all "_" characters removed before parsing
  - Arguments in the specified file are parsed with the string "prefix" removed from the front of the option

TENA::Middleware::Configuration Example

```
TENA::Middleware::Configuration config (
    argc, argv, "config.txt", "" );
```

- **What it Does**
  - Parses **argc** and **argv** for middleware configuration options
  - Parses any environment variables starting with "**TENA_MW_**"
  - Parses options contained in "config.txt" that begin with "test"

```
# Sample TENA configuration file
# Blank lines are ignored.
# The '#' character starts an EOL-terminated comment.

# ORBdefaultInitRef : Naming Service Endpoint, e.g.
#                     "corbaloc:iiop:<hostname>:<port>"
ORBdefaultInitRef=corbaloc:iiop:192.168.1.101:58725
```

Options Accepted by the Middleware

- **`-ORBdefaultInitRef <IOR prefix>`**
  - This option provides the TENA application with the information necessary to contact the **`networkNamingService`**, e.g., **`-ORBdefaultInitRef corbaloc:iiop:<hostname>:<port>`**
- **`-ORBlistenEndpoints <endpoint>`**
  - The endpoint on which the TENA application should listen for requests
  - **`-ORBlistenEndpoints iiop://192.168.1.10:9999`**
- **`-configFile <file name>` — for additional options**
- **`-multicastTTL <TTL integer>`**
  - When using multicast, to transmit data across multiple LANs, the routers that control the exchange of data between the LANs must be configured to pass IP multicast. As a protection against mistaken router configurations (e.g., router loops), every IP multicast datagram is tagged with a Time to Live (TTL) value. The TTL value is decremented every time an IP multicast datagram reaches a router.

Additional Middleware Options

- **`-multicastInterface <interface>`**
  - Sometimes it is desirable for hosts with more than one network interface to use a non-default network interface to transmit IP multicast data (i.e., when using **`TENA::Middleware::BestEffort`**). On UNIX or Linux, setting this option to the name of a network interface (e.g., "eth0") will cause that interface to be used for multicast traffic. On Windows, this option must be set to the hostname or IP address of the desired network interface.
- **`-multicastSendBufferSize <size in bytes>`**
- **`-multicastReceiveBufferSize <size in bytes>`**
  - This controls the size of the OS buffer used to hold data to be transmitted/received on an IP multicast socket. Since IP multicast is "best effort" (i.e., unreliable), attempts to transmit data faster than the network can support will result in a buffer overrun and data will be lost in some OS-specific way (most OSes will discard the entire buffer). Legal values are 1024 <= multicastSendBufferSize <= 8388608, i.e., from 1 KB to 8 MB. The default value is 131072, i.e., 128 KB.

Additional Middleware Options

- **`-disableStructuredExceptionTranslation`**
  - In a Microsoft application, several types of application runtime errors will raise what Microsoft calls "structured exceptions". These exceptions are incompatible with standard C++ exceptions (e.g., they can only be caught in a catch (...) block) and must be translated into a meaningful C++ exception. By default, the TENA Middleware translates all structured exceptions into a C++ exception class named **`DCT::Utils::StructuredException`**. Unfortunately, translating Microsoft structured exceptions hampers debugging a C++ application with Microsoft's debugger. This option disables the translation of Microsoft structured exceptions to facilitate debugging Microsoft application
- **`-noDiagnosticsLog`**
  - send diagnostics to **`stdout`** instead of a file
- **see http://www.tena-sda.org/doc/ for the documentation of the `TENA::Middleware::Configuration` class**

https://www.tena-sda.org/doc/5.1/ProgrammersGuide/df/d4c/classTENA_1_1Middleware_1_1Configuration.html

TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Steps to Publish a Servant SDO

- **Outside of your main program**
  - Define your object model
  - Implement each object's **`RemoteMethodsImpl.cpp`**
  - Create Factories for your **`RemoteMethodsImpl`** implementations (or just use the default factories provided by the code generator)
- **In the main program**
  - Instantiate the appropriate **`PublicationInfoImpl`** object
  - Inform the middleware you intend to create servants of this particular type by calling **`createServantFactory<>()`**:
    - You will obtain a **`ServantFactory`** from this step
  - Instantiate your SDO servant using the **`ServantFactory`** you just obtained
    - Each individual servant object needs to be created so that its publication state is disseminated using either unreliable UDP multicast (called "**`BestEffort`**") or multiple reliable TCP unicasts ("**`Reliable`**")
  - Update your SDO's publication state to give it its initial values

Remote Methods

- **For each type of SDO that the user needs to publish, the user needs to create a "`RemoteMethodsImpl`" class that inherits from an abstract base class generated from the object model**
- **Why?**
  - The "`RemoteMethodsImpl`" class needs to be written by the user because this class's methods are where the specific user behavior occurs
  - These are the methods that perform some action
    - Therefore, the TENA Middleware developers or the TDL compiler couldn't write these methods since only the users know what they want these methods to accomplish
- **Auto-code-generated `basicImpl` files are provided with each OM**
  - The TENA Hands-On Training course will cover in-depth the use of the **`basicImpl`** files

API for Publishing and Instantiating a Servant SDO

```
// Create a PublicationInfo object
OMsample::Platform::PublicationInfoPtr
 pPlatformPublicationInfo(
  new OMsample::Platform::BasicImpl::PublicationInfoImpl );

// Get a ServantFactory object
// Provide it the RemoteServantMethodsFactory and get
// back the PlatformServantFactory
OMsample::Platform::ServantFactoryPtr
 pPlatformServantFactory(
  pSession->createServantFactory<
                          OMsample::Platform::ServantTraits >(
   pPlatformPublicationInfo ) );

// Instantiate the servant SDO using the ServantFactory
// In this example the servant's state will be disseminated
// using UDP multicast (BestEffort) when updated
OMsample::Platform::ServantPtr pPlatformServant(
  pPlatformServantFactory->createServantUsingDefaultFactory(
    TENA::Middleware::BestEffort ) );
```

Updating the Servant's Publication State

- To change a servant's publication state, the TENA Middleware uses the concept of "updaters"
- Updaters allow sets of publication state attributes to be modified "atomically"

```
{
  // Get an updater
  std::auto_ptr<OMsample::Platform::PublicationStateUpdater>
   pUpdater ( pPlatformServant->createUpdater() );

  // Update the publication state
  pUpdater->set_name( "M1A1-001" );

  // Commit the changes atomically
  pPlatformServant->commitUpdater( pUpdater );
}
```

Transactional Behavior for State Updaters

- **Once an updater is obtained, a user can do three things:**
  - Commit the updater – make and disseminate the changes
  - Rollback the updater – do not make or disseminate the changes
  - Let the updater go out of scope – do not make or disseminate the changes
- **Methods on the servant:**

```
void
 commitUpdater (
  std::auto_ptr< PublicationStateUpdater > );

void
  rollbackUpdater (
    std::auto_ptr< PublicationStateUpdater > );
```

TENA Middleware Overview and the Current Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

SDO Subscription Services

- **Subscription services deal with how an application can become a client (consumer) of objects created and served by other applications**
- **To access objects created by other applications a programmer must make a subscription request to the TENA Middleware**
  - Programmers may subscribe only on the basis of an object's class (which must be defined in the LROM)
  - Class-level subscriptions can be made on any class within the OM, including contained classes
    - TENA objects (SDOs) are "atomic" units

Proxies

- **The result of subscription is that the TENA Middleware provides the application with one or more "proxies"**
  - Proxies "stand in" for servants, and represent them
  - Proxies have a similar interface to another object (the servant)
  - A proxy is, in effect, a read-only representative, in your process space, of a servant that exists elsewhere, typically in another process space
- **A user receives proxies in the form of what are called "`ProxyPtr`"s**
  - A `ProxyPtr` behaves much like a C++ pointer to a Proxy
- **There are three basic functions that one can perform with a proxy:**
  1. Invoke remote methods on the servant
  2. Get the publication state so that it can be read
  3. "Let go" of the proxy if you are no longer interested in it

Subscription Process

- **Outside of your main program**
  - Create the LROM
  - Write the **`DiscoveryCallbackImpl`** and **`DiscoveryCallbackFactoryImpl`** classes
  - Write the **`StateChangeCallbackImpl`** and **`StateChangeCallbackFactoryImpl`** classes (optional)
  - Write the **`DestructionCallbackImpl`** and **`DestructionCallbackFactoryImpl`** classes (optional)
- **Inside your main program**
  - Instantiate a **`CallbackInfo`** object
  - Instantiate a **`SubscriptionInfoImpl`** object
  - Subscribe to the SDO
  - Discover proxies (eagerly), i.e., process discovery callbacks
  - Read the newly discovered proxy's state
  - Invoke remote methods on a servant via the proxy

API for Subscribing to an SDO

```
// Instantiate a platform CallbackInfo object
OMsample::Platform::BasicImpl::CallbackInfoPtr
  pCallbackInfo( new
    OMsample::Platform::BasicImpl::CallbackInfo(
        std::cout ) );

// Instantiate a platform SubscriptionInfo object
OMsample::Platform::SubscriptionInfoPtr
  pSubscriptionInfo( new
    OMsample::Platform::BasicImpl::SubscriptionInfoImpl(
        pCallbackInfo ) );

// Subscribe to Platforms
 pSession->subscribeToSDO<
   OMsample::Platform::ProxyTraits >(pSubscriptionInfo);
```

Discovering a New Proxy

- **Inside the discovery callback object's `execute()` method, the new proxy is placed in the appropriate collection**
  - The callback object is constructed with a pointer to the proxy to the new object as a member variable ("**`_pProxy`**")

```
void OMsample::Platform::BasicImpl::DiscoveryCallbackImpl::execute()
{
    _pCallbackInfo->getDiscoveredSDOlist().push_back( _pProxy );
}
```

- **The list itself is also a member variable of the callback object**
  - How??? This is tricky, but accomplished because the user writes the appropriate classes:
    - **`main`**
    - **`OMsample::Platform::BasicImpl::CallbackInfo`**
    - **`OMsample::Platform::BasicImpl::DiscoveryCallbackFactoryImpl`**
    - **`OMsample::Platform::BasicImpl::DiscoveryCallbackImpl`**
- **The list is created in main() and passed in circuitously:**

```
main() ➡
 CallbackInfo::_discoveredSDOlist ➡
  DiscoveryCallbackFactoryImpl::constructor() ➡
   DiscoveryCallbackFactory._pCallbackInfo._discoveredSDOlist ➡
     DiscoveryCallbackImpl::constructor() ➡
      DiscoveryCallbackImpl._pCallbackInfo._discoveredSDOlist
```

CallbackInfo.h
Getting Information out of Callbacks

- **The callbackInfo class is used to get proxy information out of a callback, in this example using an `std::list`**

```
class CallbackInfo
{
  public:
    CallbackInfo( std::ostream & );
    ~CallbackInfo();
    std::ostream & getOutputStream();
    std::list< OMsample::Platform::ProxyPtr > &
      getDiscoveredSDOlist();

  private:
    CallbackInfo( CallbackInfo const & rhs );
    CallbackInfo & operator=( CallbackInfo const & rhs );
    std::ostream & _outputStream;
    std::list< OMsample::Platform::ProxyPtr >
      _discoveredSDOlist;
};

typedef DCT::Utils::SmartPtr< CallbackInfo > CallbackInfoPtr;
```

Reading the SDO's State

- Assume discovery occurred and the proxy was assigned to `pPlatformProxy`
- Using the "get" method to read the publication state

```
// using return value from the servant object
// instantiation create a ProxyRef
OMsample::Platform::ImmutablePublicationStatePtr
   pPlatformPublicationState(
     pPlatformProxy->getPublicationState() );

// read the attributes
std::string platformName (
  pPlatformPublicationState->get_name() );
```

Invoking a Remote Method

- **After getting the proxy, just invoke the method directly**

```
pPlatformProxy->movePlatformTo( 1.0, 2.0 );
```

- **This will result in a Remote Method Invocation on the servant**
- **Your process will block until this remote method returns**
  - In the future, "**oneway** methods" could eliminate the need to block

Unsubscribe

- **To unsubscribe to an SDO, invoke the appropriate method on the Session:**

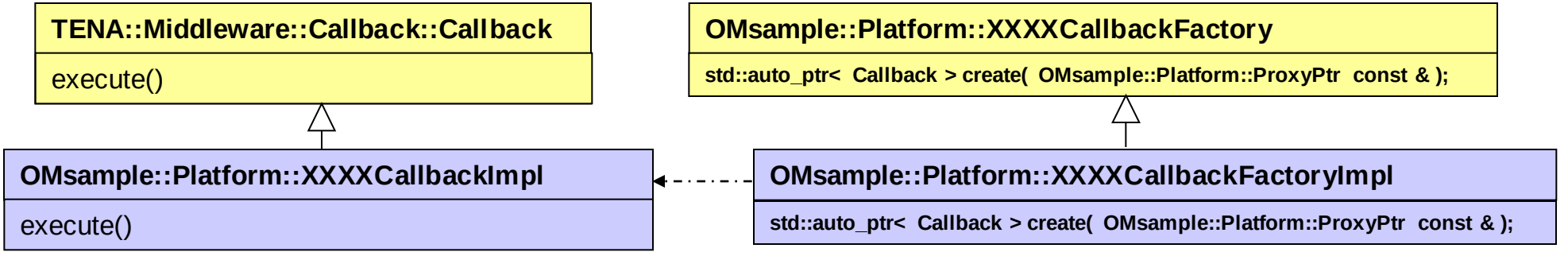
```
pSession->unsubscribeFromSDO<
  OMsample::Platform::ProxyTraits >();
```

TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

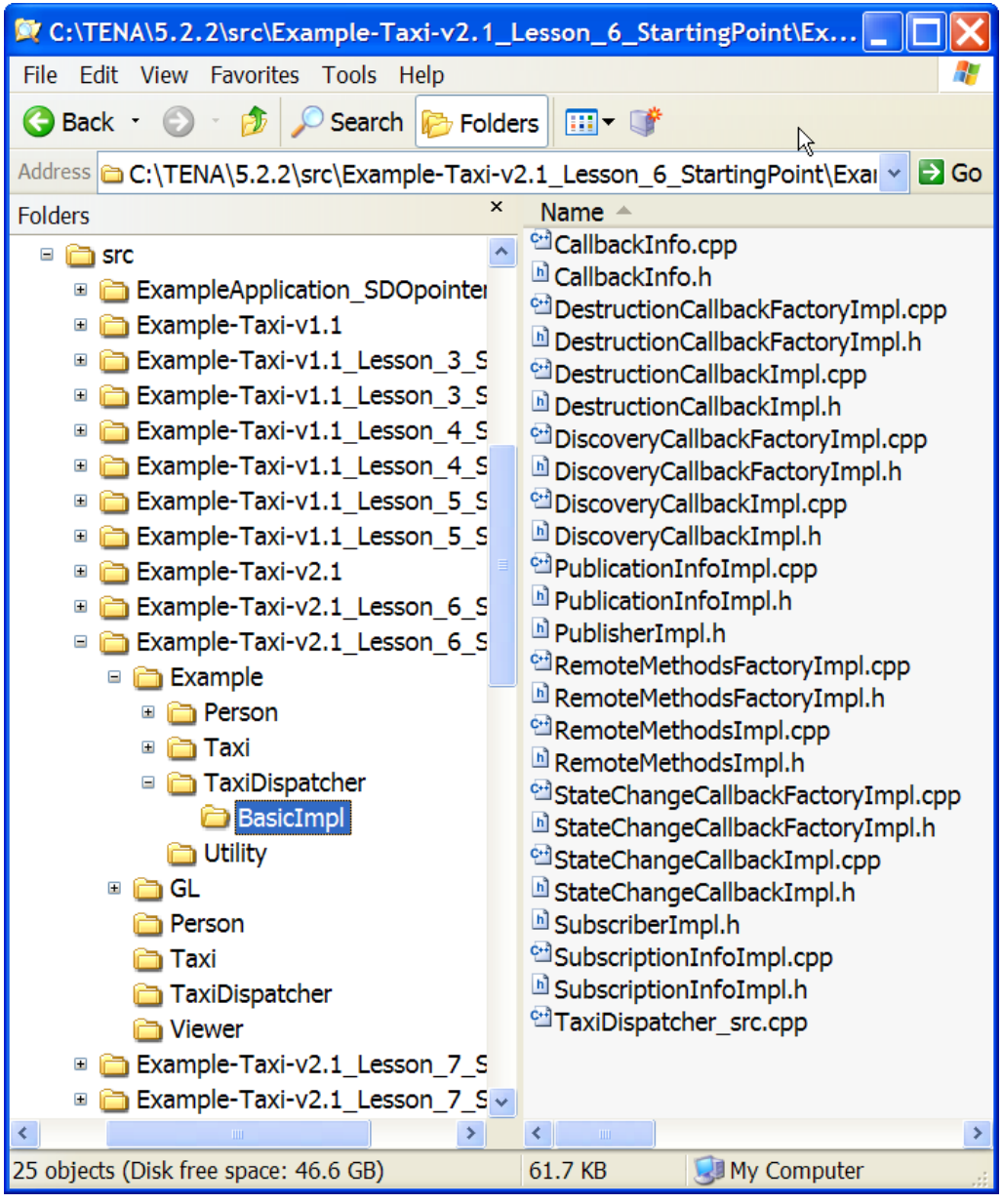
Callbacks

- **Issues:**
  - What callbacks are there?
  - How does the user write the callbacks they need?
  - During execution, how does the user get the callbacks delivered to them?
- **What Callbacks are there?**
  - **Discovery** – callback when a new proxy arrives
  - **StateChange** – callback when a currently known proxy's publication state changes
  - **Destruction** – callback when a servant whose proxy your application holds is deleted by its owner application
- **Each of these callbacks use the command and factory method pattern for their instantiation**



Writing Callbacks

- **Each user must write two classes for each of the three types of callbacks**
  - Factory
  - Actual Callback Implementation
- **Users should look at the auto-generated "basicImpl" code for guidance on how to use these callbacks**
  - For simple situations, the **`basicImpl`** code may be all a user needs to get his or her application running



Callbacks in Your Main Program

- **The user application must tell the middleware that it is ready to accept callbacks**
- **The TENA Middleware provides a unified `Callback` evocation interface to provide flexible control over the programming threading and concurrency issues**
  - When the middleware wants to inform the user of something, a `Callback` object is created and placed in a queue
  - The user is able to provide one or more threads of control to the middleware for executing these callbacks when they deem appropriate
  - This interface will support different application threading models (single threaded, multi-threaded) and different concurrency models (non-reentrant, reentrant)

Invoking a Single Callback
(These are methods on the Session)

`size_t evokeCallback()`

- Instructs the middleware to execute a callback if one exists in the queue
  - The middleware will execute only a single callback at a time – it will return immediately if no callbacks exist in the queue
  - A Boolean value of **`True`** returned by the method indicates that there are additional callbacks remaining on the queue

`size_t evokeCallback(`
`  unsigned int const maxWaitInMicroseconds );`

- Instructs the middleware to execute a single callback or wait a maximum duration of time for a callback if one does not exist initially
  - The middleware will execute only a single callback at a time – it will return immediately after processing the callback or expiration of the max time if no callbacks exist in the queue during the wait period
    - The middleware is unable to guarantee that it can honor the maximum duration since the execution control can be passed to the application for an indeterminate amount of time
  - A Boolean value of **`True`** returned by the method indicates that there are additional callbacks remaining on the queue

Invoking Multiple Callbacks
(This is a method on the Session)

```
size_t evokeMultipleCallbacks(
  unsigned int const maxWaitInMicroseconds );
```

- Used to execute callbacks in the callback queue until the maximum duration is exceeded
- Allows the middleware to execute multiple callbacks
- User applications may want to use this variation if they operate in a time-based loop in which they can provide any excess time for each loop to the middleware for handling callbacks
  - An adaptive approach can also be used where the application determines whether there are still callbacks pending at the end of an **`evokeMultipleCallbacks`**
  - If pending, a **`True`** return, the application could adjust the maximum duration thereby providing the middleware more time for processing

- The best routine to use for more than one callback

Callback Evocation Examples

- **The TENA Middleware provides a set of callback evocation methods**
  - These methods allow the user application to inform the middleware that callbacks can be executed
  - They return the size of the callback queue
- **There are three variations of the callback evocations services, each are methods on the session object:**

```
// Give me a single callback
size_t more ( pSession->evokeCallback() );

// Give me a single callback and wait for a maximum
// amount of time (an unsigned integer in microseconds)
size_t more ( pSession->evokeCallback( 1000 ) );

// Give me multiple callbacks and wait for a maximum
// amount of time (an unsigned integer in microseconds)
size_t more ( pSession->evokeMultipleCallbacks( 1000 ) );
```

TENA Middleware Overview and the Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Local Classes

- **The TENA Middleware also supports the abstract concept of a local class**
  - A local class is a generalization of a **`struct`**
    - A struct has state but not methods
  - A local class may contain methods *and* state
  - Unlike an SDO, the methods and state of a local class are entirely local to the process containing the local class instance
- **The local class concept allows object model definitions to support client-side methods on local classes contained in SDOs or Messages.**
  - Invoking a method on a local class results in local execution of the method, **never** a remote method call
- **Local class methods can be found in the files:**
  `src/.../<LocalClass>/BasicImpl/MethodsImpl.h`
  `src/.../<LocalClass>/BasicImpl/MethodsImpl.cpp`

Creating and Using a Local Class

- **Creating a local class**
  - Create the local class using its `Interface::create()` method

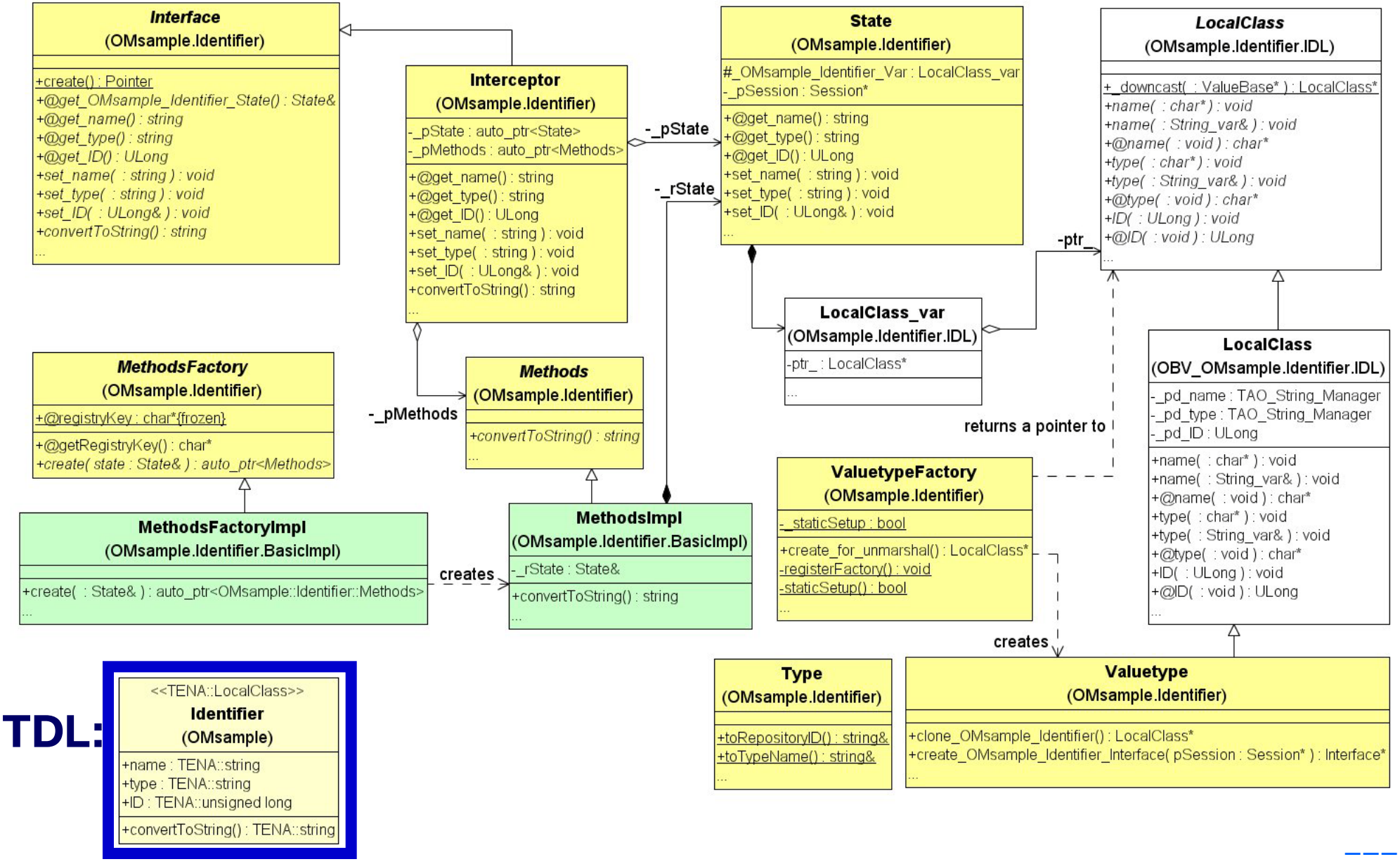
```
Example::LocalClassType::Pointer pLC(
  Example::LocalClassType::Interface::create() );
```

- **Using a local class is almost identical to using a smart pointer to a regular C++ object**

```
pLC->set_longMember( 7 );       // sets member variable
long l = pLC->get_longMember();// gets member variable
pLC->doLocalMethod();           // invoke local method
pLC.reset();  // Deletes the object pointed to by pLC
```

- **After `reset()`, the pointer `pLC` is no longer valid (i.e. it points to `NULL`)**

Internal Structure of a Local Class (Example: OMsample::Identifier)



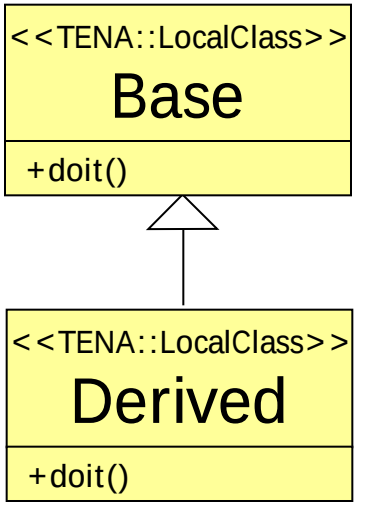
TDL:

Polymorphic Local Class Methods

- **Local class methods now behave polymorphically**
  - If you have a pointer to a **`Base`**, but the underlying object is really a **`Derived`**, then when you do:

```
pBase->doit();
```

the implementation of **`Derived::doit()`** is executed



Local Class Get/Set Issue

- **Issue: "I invoked set_foo() on my local class but the state of the SDO didn't change"**
  - `pUpdater->get_aLocalClass()->set_someAttribute( someValue );`
- **`get_aLocalClass()` returns a clone of the contained local class, not a reference**

Local Class Get/Set Issues (cont)

- **Doing this the right way:**

```
std::auto_ptr< OMsample::Platform::PublicationStateUpdater >
  pPlatformUpdater(
    pPlatformServant->createPublicationStateUpdater() );

// Get a Pointer to the Local Class attribute "time"
OMsample::Time::Pointer
  pTime( pPlatformUpdater->get_time() );

pTime->set_nanoseconds(2184);
pTime->set_seconds(2457);

pPlatformUpdater->set_time(pTime);

pPlatformServant->commitUpdater( pPlatformUpdater );
```

TENA Middleware Overview and the Current Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Message Publisher Supporting Classes

- **`MessageSenderPtr` - A message sender is the middleware object that is used for sending messages of a particular type with particular communications characteristics (`BestEffort` or `Reliable`)**

Sending a Message

```
// Create a MessageSender in the session,
// using the local methods factory
OMsample::LocationMessage::MessageSenderPtr
  pLocationMessageSender(
    pSession->createMessageSender<
        OMsample::LocationMessage::MessageTraits >(
          TENA::Middleware::BestEffort ) );

// Create a Message
OMsample::LocationMessage::Pointer pMessage(
    OMsample::LocationMessage::Interface::create() );

// Change the value of the integer attribute "MessageID"
pMessage->set_MessageID( 12 );

// Send the Message
pLocationMessageSender->send( pMessage );
```

Subscribing to Messages

- **Supporting classes:**
  - **`CallbackInfo`** - A helper class used to easily pass variables into the message callback classes that are created when a message is received
  - **`SubscriptionInfoImpl`** - helper class that assists with callback factories associated with subscribed message types
  - **`MessageCallbackImpl`** - The callback class in which the user provides the behavior that is invoked when a message is received

Subscribing to Messages

- **Subscribing:**

```
OMsample::LocationMessage::BasicImpl::CallbackInfoPtr
  pCallbackInfo ( new
    OMsample::LocationMessage::BasicImpl::CallbackInfo(
      std::cout ))

OMsample::LocationMessage::SubscriptionInfoPtr
  pSubscriptonInfo ( new
    OMsample::LocationMessage::BasicImpl::SubscriptionInfoImpl(
      pCallbackInfo ) );

pSession->subscribeToMessage<
  OMsample::LocationMessage::MessageTraits >(
    pSubscriptonInfo );
```

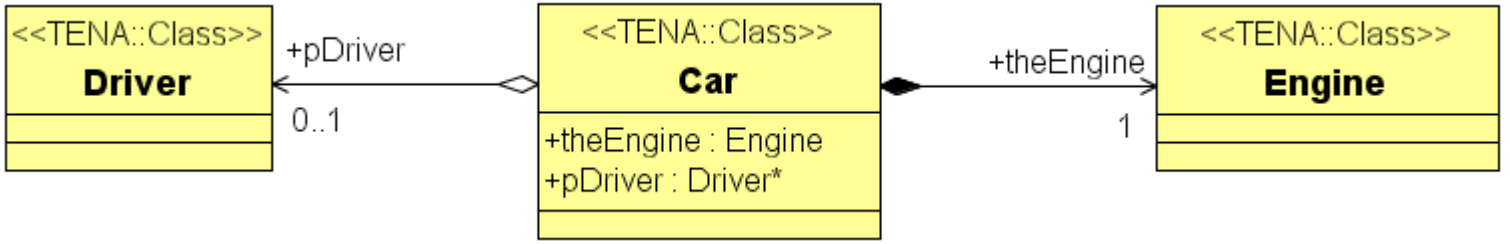
- **Unsubscribing:**

```
pSession->unsubscribeFromMessage<
  OMsample::LocationMessage::MessageTraits>();
```

TENA Middleware Overview and the Current Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

SDO Pointer Example



TDL:

```
class Driver {};
class Engine {};
class Car
  {
    Driver * pDriver;
    Engine theEngine;
  };
```

- **SDO Pointer Rules:**
  - An instance of an SDO pointer may be NULL
  - An instance of an SDO pointer may be de-referenced to obtain the instance of SDO to which it refers
  - An instance of an SDO pointer may be assigned to another instance of an SDO pointer of the same type
  - An instance of an SDO pointer may be set to refer to an SDO pointer of the corresponding type

Using SDO Pointers

- **Creating an SDO Pointer from a Servant (use the same method to get a pointer from a Proxy):**

```
CarExample::Driver::Pointer pDriver(
  pDriverServant->getPointer() );
```

- **De-Referencing an SDO Pointer**

```
CarExample::Driver::ProxyPtr pDriverProxy(
  pDriver->getProxyPtr(
    new CarExample::Driver::BasicImpl::SubscriptionInfoImpl );
```

- **Testing the Pointer**

```
if ( pDriver.isValid() ) { … }  // Is the Pointer Valid??
```

SDO Pointer Behavior

- **The dereferencing of an SDO pointer results in "instance-based subscription" to the pointed-to object**
- **Unless the pointed-to object's proxy has already been discovered (if that type has been subscribed to by the application), the dereferencing of an SDO pointer can be a very expensive operation with multiple network round-trips**
  - Your application is blocked while this is going on
- **If the pointed-to object's proxy is already being held onto by your application, then the dereferencing operation is very quick and optimized**

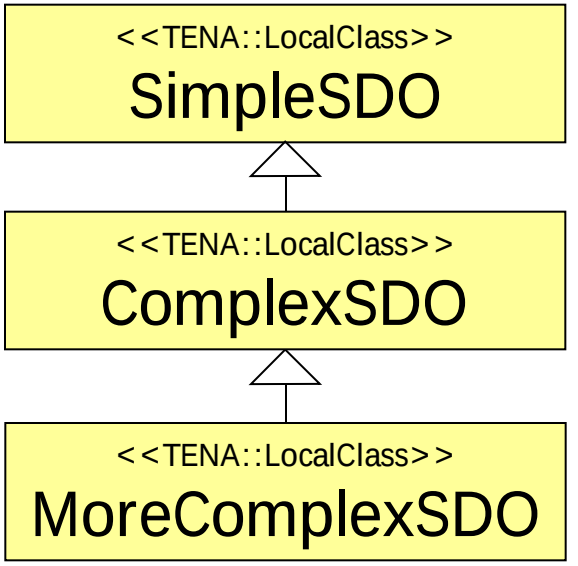
TENA Middleware Overview and the Current Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Casting Proxies

- **Upcast:**

```
// get pComplexProxy somehow…
SimpleSDO::ProxyPtr
  pSimpleProxy(pComplexProxy);
```



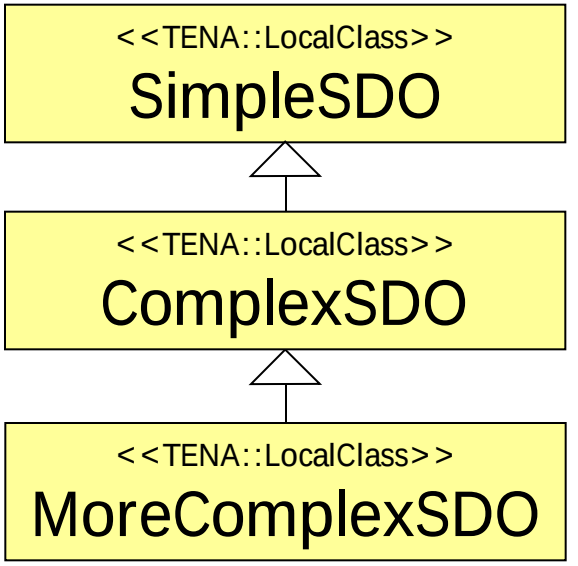
- **Downcast:**

```
MoreComplexSDO::ProxyPtr
   pMoreComplexProxy(
    TENA::Middleware::dynamicCast<
      MoreComplexSDO::ProxyPtr>(pSimpleProxy) );
```

- **Check:**

```
if ( pMoreComplexProxy.isValid() )
     {…}   // you've succeeded.
   else
     {…};  // you've failed, try again.
```

Casting SDO Pointers



- **Upcast:**

```
ComplexSDO::Pointer pComplexPointer(
  pComplexSDOServant->getPointer() );
SimpleSDO::Pointer
  pSimplePointer(pComplexPointer);
```

- **Downcast:**

```
MoreComplexSDO::Pointer
   pMoreComplexPointer(
    TENA::Middleware::dynamicCast<
      MoreComplexSDO::Pointer>(pSimpleProxy) );
```

- **Check:**

```
if ( pMoreComplexPointer.isValid() )
     {…}   // you've succeeded.
  else
     {…};  // you've failed, try again.
```

Casting Local Classes

- **Upcast:**

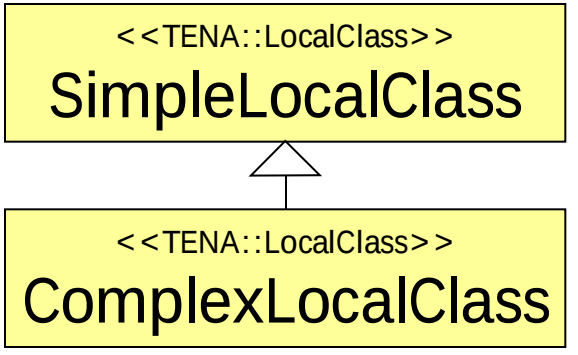
```
// create a ComplexLocalClass instance
ComplexLocalClass::Pointer pComplex(
  ComplexLocalClass::Interface::create() );

// upcast to SimpleLocalClass
SimpleLocalClass::Pointer pSimple( pComplex );
```

- **Downcast:**

```
ComplexLocalClass::Pointer pOtherComplex(
 TENA::Middleware::dynamicCast<
   ComplexLocalClass::Pointer>( pSimple ) );
```

- **Check:**

```
if ( pOtherComplex.isValid() )
    {…}   // you've succeeded.
  else
    {…};  // you've failed, try again.
```



Casting Messages

- **Upcast:**

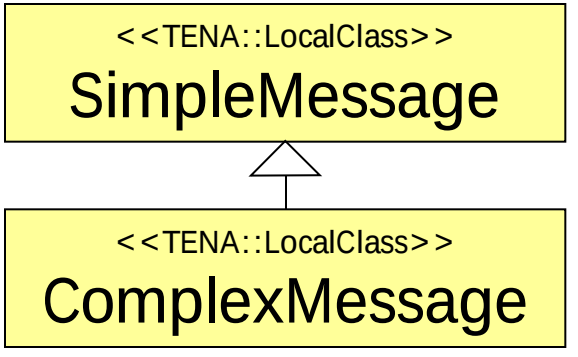
```
// create a ComplexMessage instance
ComplexMessage::Pointer pComplex(
  ComplexMessage::Interface::create() );

// upcast to SimpleMessage
SimpleMessage::Pointer pSimple( pComplex );
```

- **Downcast:**

```
ComplexMessage::Pointer pOtherComplex(
 TENA::Middleware::dynamicCast<
   ComplexMessage::Pointer>( pSimple ) );
```

- **Check:**

```
if ( pOtherComplex.isValid() )
    {…}   // you've succeeded.
  else
    {…};  // you've failed, try again.
```



Casting Notes

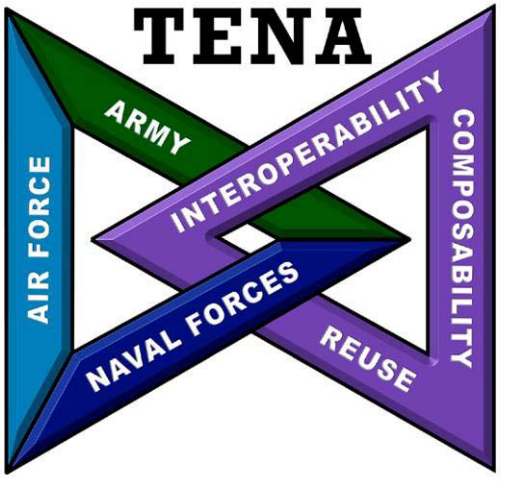
- **Servants can neither be cast upward nor downward (this is on the TENA "Things To Do" list)**
- **When downcasting, if you don't know what the underlying type of an object is, you need to continually downcast it to all the possible types and see which succeeds using the `.isValid()` method**
  - The program structure will be not much different from the case in which type information can be recovered
  - If your object model has type information encoded in it, then you can switch off this information to make the choice of a cast.
  - Remember to always check using the **`.isValid()`** method

TENA Middleware Overview and the Current Release 5 API

- TENA Middleware Software Overview
- TENA Executions
- Application Development Basics
- Execution Management Services
- SDO Publication Services
- SDO Subscription Services
- Callback Services
- Local Classes
- Messages
- SDO Pointers
- Downcasting
- Diagnostics

Diagnostics Log

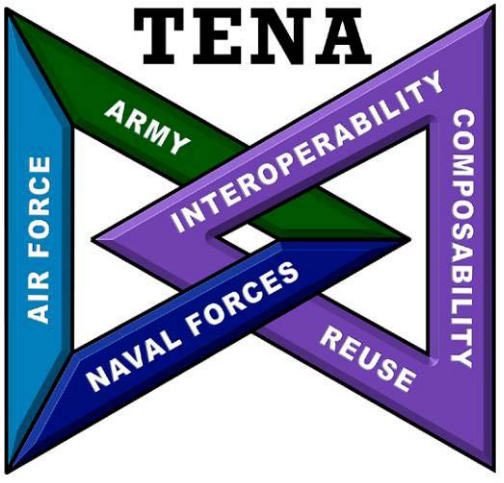
- **All messages from the Middleware have been redirected to a file named `diagnosticsLog-YYYY-MM-DD-HHMMSS-PID.txt`**
  - Note that this file is created in the directory where the application starts. That directory must be writable, or the Middleware will fail to initialize by throwing a **`std::runtime_error`** exception from **`TENA::Middleware::init()`**
- **The first line of every diagnostics log file is the name of the application, as reported by `argv[0]`. Every entry in the diagnostics log file is time-stamped.**


TENA
ARMY
AIR FORCE
NAVAL FORCES
INTEROPERABILITY
COMPOSABILITY
REUSE

Questions?


IMPROVING T&E
CTEIP
CAPABILITIES

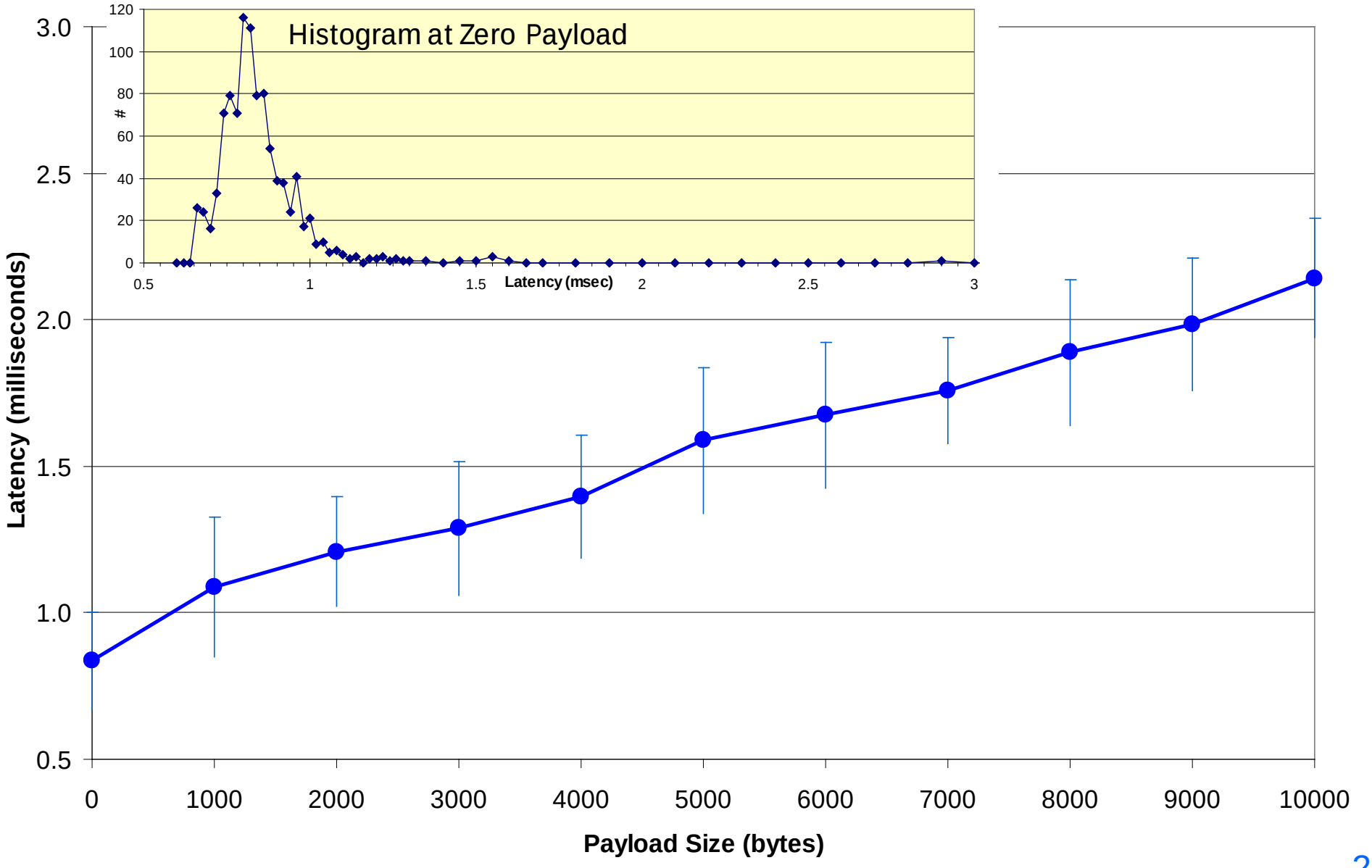

UNITED STATES
JOINT FORCES
COMMAND
USJFCOM

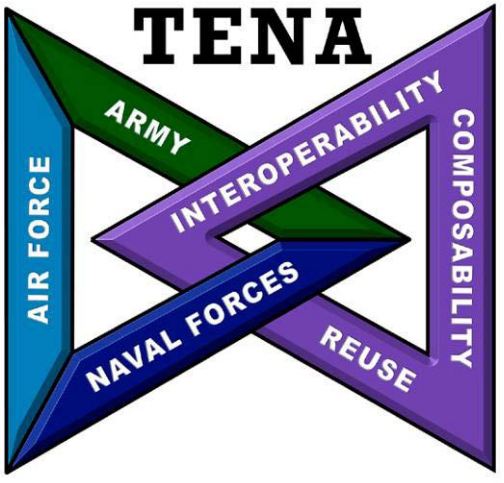


Where To Go From Here

- **Performance Test Results**
- **Getting Help From the Help Desk**
- **Additional Training and Feedback**

Release 4 Latency
(Peer-to-Peer, Windows 2000 Pro, 100 Hz Updates)

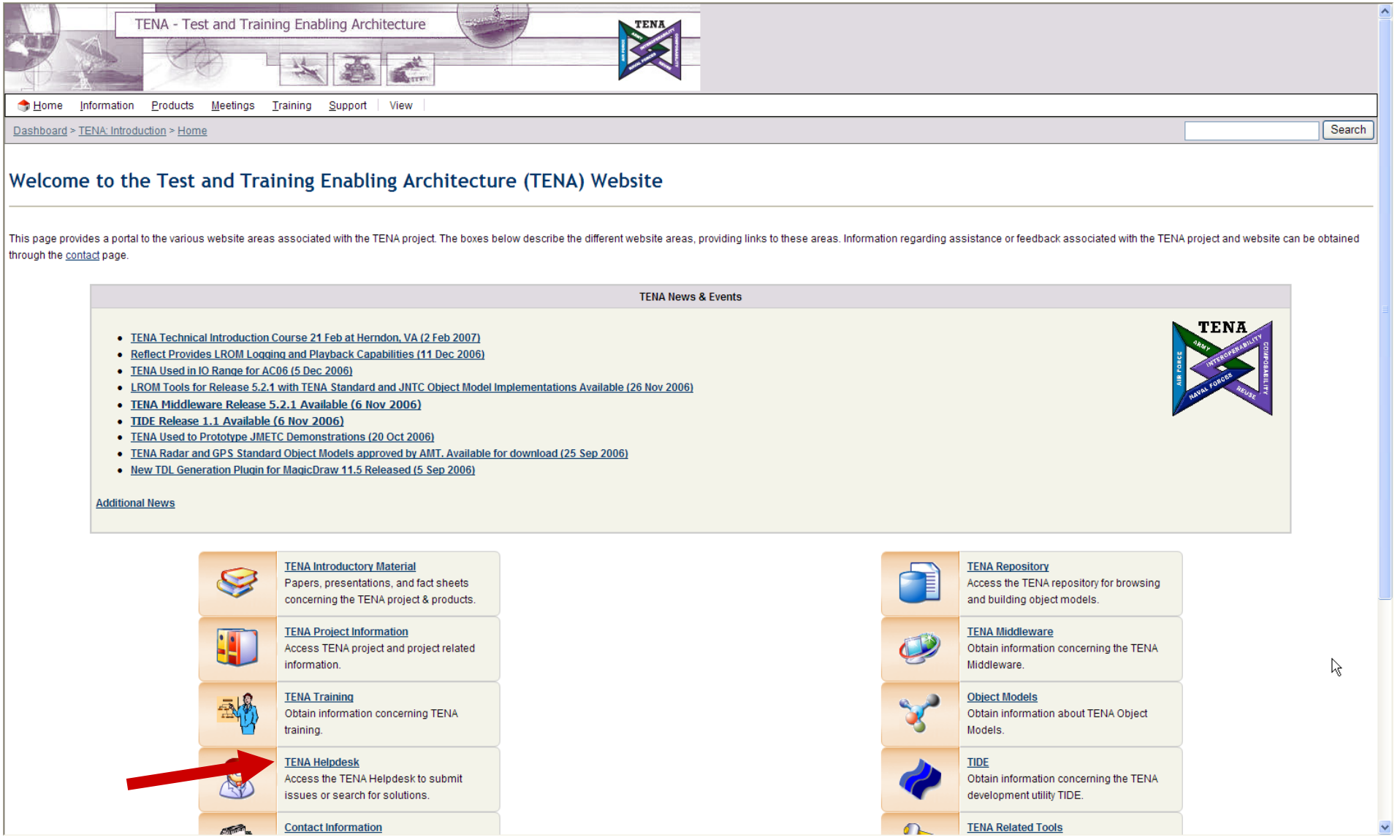




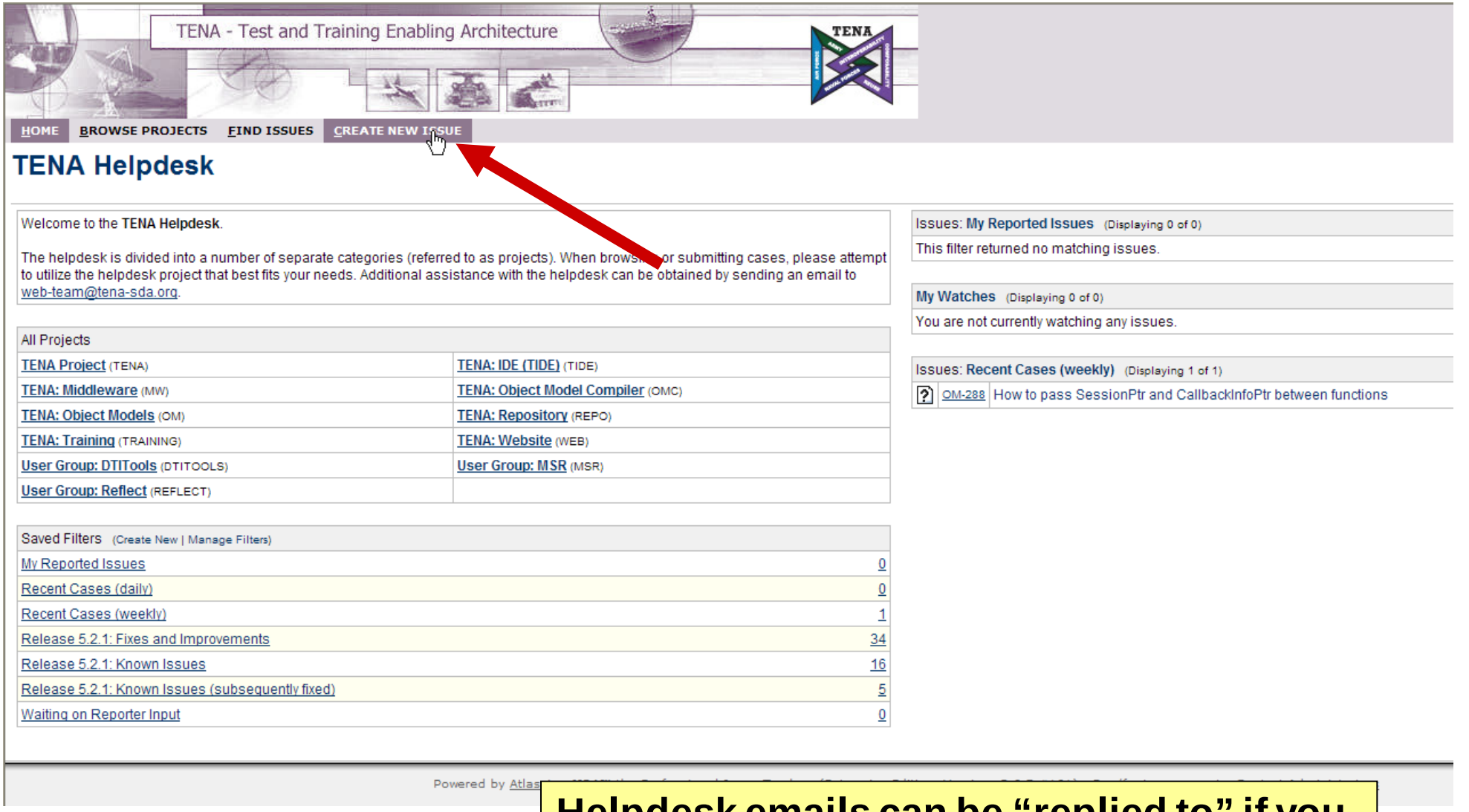
Where To Go From Here

- **Performance Test Results**
- **Getting Help From the Help Desk**
- **Additional Training and Feedback**

TENA Helpdesk

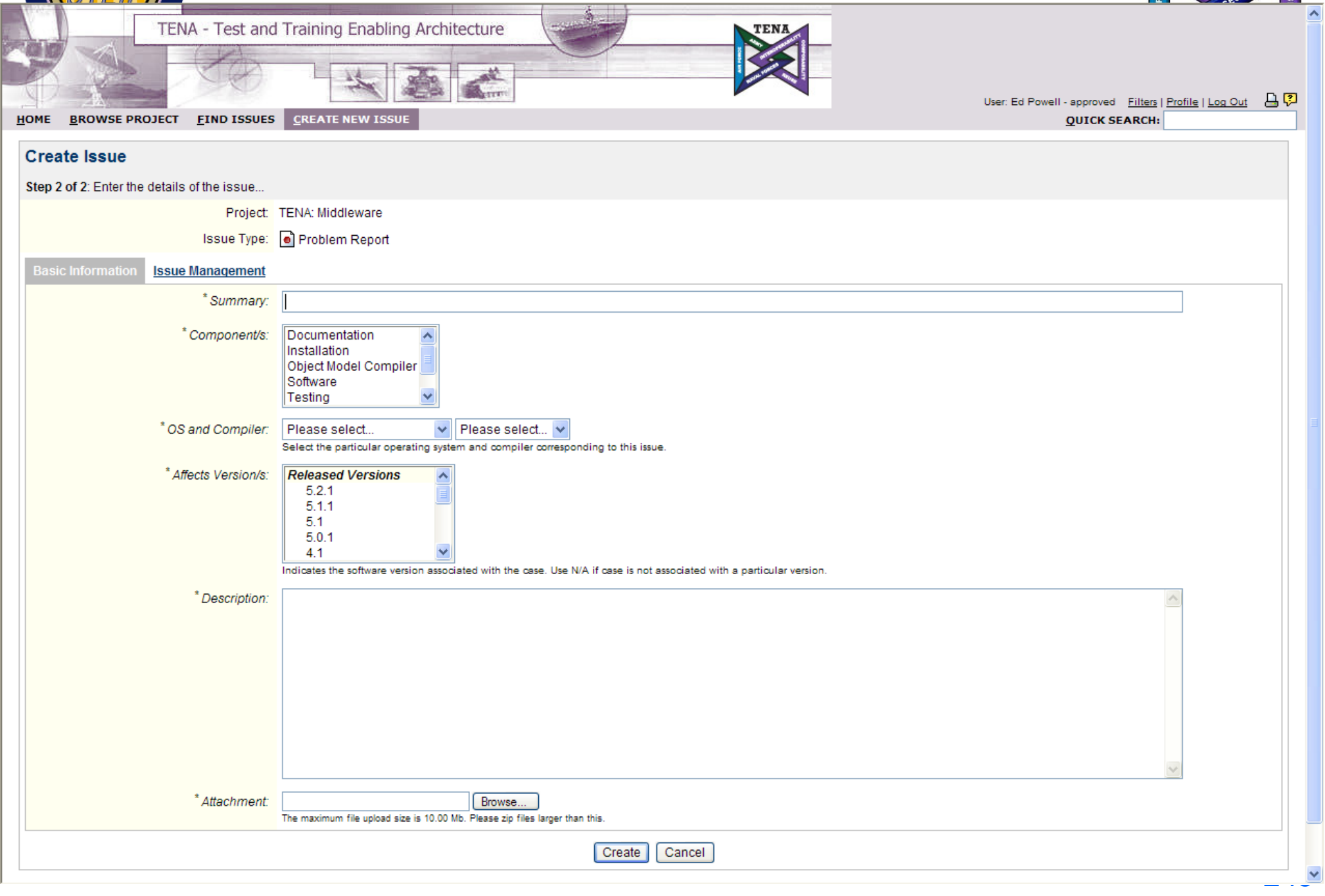


TENA Helpdesk



Helpdesk emails can be "replied to" if you have an account

Creating a New Issue



Where To Go From Here

- **Performance Test Results**
- **Getting Help From the Help Desk**
- **Additional Training and Feedback**

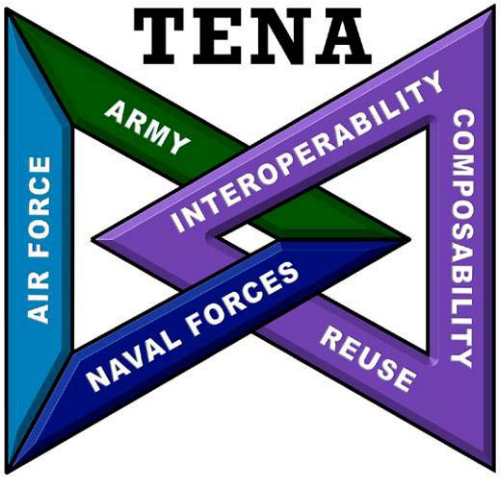
Hands-On Training Overview

- **The TENA Hands-On Training Course will provide software developers a hands-on programming experience with the current release of the TENA Middleware prototype**
- **The target audience for this course is Software Developers who have:**
  - Attended the Technical Introduction Course
  - A background in C++, Object-Oriented software, distributed computing, HLA, and/or CORBA is recommended, but not required
- **Development exercises will cover the gamut from joining a Logical Range Execution to Updating and Reading contained SDOs**

Important Contact Information

- **Project Website: http://www.tena-sda.org**
  - Download TENA Middleware (http://www.tena-sda.org/repository)
  - Submit Helpdesk Case (http://www.tena-sda.org/helpdesk)

- **TENA Feedback: feedback@tena-sda.org**
  - Provide technical feedback on TENA Architecture or Middleware
  - Ask technical questions regarding TENA
  - Provide responses to AMT action items
  - Request TENA training



Questions or Comments?