The Teton Project

DDS vs. DDS4CCM

July 13, 2011

Teton SNA Core Team
• Object Management Group (OMG)
  – International software open standards organization
  – DDS, CCM, CORBA, UML, SysML, MDA, IDL, D&C, MARTE, MOF, UPDM, XMI, many others…

• Data Distribution Service (DDS)
  – Popular real-time middleware open standard
  – Publish-Subscribe information exchange
  – Inherent support for data-oriented Event Driven Architecture (EDA)

• CORBA Component Model (CCM)
  – Component model extension of the classic CORBA 2 real-time middleware open standard
  – Also known as CORBA 3
  – Request-Response information exchange
  – Inherent support for Service Oriented Architecture (SOA) and Component Based Architecture (CBA)

Until very recently, the CCM & DDS complementary middleware standards were not integrated
DDS for Lightweight CCM (DDS4CCM)  
Component-Based DDS (CBDDS)

- New OMG open standard that integrates the CCM and real-time middleware standards
  - CCM+DDS real-time component framework
  - All components and interface ports are well defined and specified in OMG Interface Definition Language (IDL)

- Initial *Adopted* Specification in December 2008
  - At July OMG Real-Time Workshop, aided by RTI per an ongoing RTI Architecture Study for Teton that was targeting CCM+DDS
  - Primary/lead author was Thales, with co-sponsors RTI, PrismTech & Mercury
  - Entered two FTF (Finalization Task Force) phases, which NGC participated in
    - Influence over spec evolution primarily through NGC subcontractors
    - NGC funded the definitive & most advanced reference implementation of the DDS4CCM spec during FTF phase via RTI, Remedy IT and Vanderbilt ISIS
    - Three working drafts during the FTF phase

- Final *Available* Specification (v1.0) in June 2011
  - OMG: [http://www.omg.org/spec/dds4ccm/1.0/PDF/](http://www.omg.org/spec/dds4ccm/1.0/PDF/)
  - A draft v1.1 specification is already available on the OMG site
• The 5 key architectural tenets driving the Teton framework are:
  – **OA** – Open Architecture
    • More specifically, the DoD’s MOSA (Modular Open Systems Approach) initiative
  – **EDA** – Event Driven Architecture
    • Often associated with DOA (Data Oriented Architecture) as a complement to SOA, EDA primarily defines the programming model
  – **SOA** – Service Oriented Architecture
  – **CBA** – Component Based Architecture
    • Associated industry terms include CBD (Component Based Development) and CBSE (Component Based Software Engineering)
  – **MDA** – Model Driven Architecture

• CBA is an intrinsic element of SOA, MOSA/OA and MDA
OA – Open Architecture & MOSA

• Charter tenet for The Teton Project

• OA is the foundation of a number of current key contracts at NGES

• OA or MOSA in particular mandated in most new DoD RFPs
  – DoD directives mandate its use in acquisitions
    • DoD 5001.2: “A modular, open systems approach shall be employed, where feasible”
  – DoD program managers (our customers) expect it
    • MOSA is now a generally accepted standard approach
    • Makes government PMs job easier if contractor can assure MOSA compliance
  – Good for business
    • Reduces program execution risk
    • Our competitors are using and marketing it
### SNA Core Software Services (CSS) APIs for High Performance Computing (HPC) Applications

**Mission Independent Core Services & APIs for the Teton Scalable Node Architecture (SNA)**

<table>
<thead>
<tr>
<th>Logging</th>
<th>Pub/Sub Messaging</th>
<th>Pub/Sub Attachment Transfer (PSAT)</th>
<th>Data Record / Playback</th>
<th>Discovery Services</th>
<th>Config Parameter Access</th>
<th>Math Libraries</th>
<th>Service/Client Messaging</th>
<th>Time Mgmt</th>
<th>System Management (App Container)</th>
<th>OS Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>log4cxx</td>
<td>DDS</td>
<td>DDS4CCM</td>
<td>RTSP</td>
<td>DDS4CCM</td>
<td>DDS Topics</td>
<td>libConfig</td>
<td>VSIPL++</td>
<td>CCM</td>
<td>ACE POSIX</td>
<td>D&amp;C, CCM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDDS4CCM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMI4CCM</td>
<td>ACE POSIX</td>
<td>DDS4CCM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ACE</td>
<td>POSIX</td>
<td></td>
</tr>
</tbody>
</table>

- All CSS APIs based upon **open standards**
- The CSS Services & APIs provide the basic programming environment & foundation for all application component designs, offering universal software services
  - Including General Purpose (GP) and Signal Processing (SP) applications
- Core services support mission independent needs of new NGES programs
  - DDS a key contributor, but many more OA standards needed to address requirements
  - Component-Based DDS via DDS4CCM addresses even more, but still others are required

Both DDS and DDS4CCM Satisfy the Teton OA Architectural Tenet
### SNA APIs – OA Standards & Governance

<table>
<thead>
<tr>
<th>#</th>
<th>API</th>
<th>Description</th>
<th>Open Standard</th>
<th>Governance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Container (System Management)</td>
<td>Encapsulation of message event handler for service calls, service responses, subscriptions, timers &amp; other events. Encapsulates business logic/algorithms event callback registration and common component-based functionality such as security, startup, init &amp; shutdown. Embodiment of component ICD &amp; lifecycle management.</td>
<td>CCM (CORBA Component Model), D&amp;C (Deployment &amp; Configuration)</td>
<td>OMG (Object Management Group)</td>
</tr>
<tr>
<td>2</td>
<td>Service/Client Messaging</td>
<td>Request/response, synchronous/asynchronous, RPC/RMI-based messaging supporting SOA services and clients. Service interfaces defined in OMG IDL.</td>
<td>CCM, AMI4CCM</td>
<td>OMG</td>
</tr>
<tr>
<td>3</td>
<td>Pub/Sub Messaging</td>
<td>Publish/subscribe messaging API supporting streaming data and event streams per EDA architectural patterns. Topic data types defined in OMG IDL.</td>
<td>DDS, DDS4CCM</td>
<td>OMG</td>
</tr>
<tr>
<td>4</td>
<td>Pub/Sub Attachment Transfer (PSAT)</td>
<td>Wideband, high-throughput transport encapsulation extension of DDS for “large data.” Combination of DDS for signaling and OFED-based wideband fabric transport (SHMEM, RDMA, etc.) integrated into a DDS4CCM standard “connector” in the CCM container.</td>
<td>DDS4CCM</td>
<td>OMG</td>
</tr>
<tr>
<td>5</td>
<td>Logging</td>
<td>Text-based logging system for debug, development, I&amp;T</td>
<td>log4cxx</td>
<td>Apache</td>
</tr>
<tr>
<td>6</td>
<td>Config Parameter Access</td>
<td>Generic access to static [read at startup] user-defined configuration files of hierarchical key-value pairs. API lookup of a value string by passing a sequence of key strings.</td>
<td>libConfig</td>
<td>SourceForge .net</td>
</tr>
<tr>
<td>7</td>
<td>Data Record/Playback</td>
<td>Generic data record/playback service built on the pub/sub messaging system capabilities &amp; generic data type definition utilities.</td>
<td>RTSP (Real Time Streaming Protocol)</td>
<td>IETF RFC 2326</td>
</tr>
<tr>
<td>8</td>
<td>Discovery Services</td>
<td>Data &amp; Service directory services supporting dynamic system behavior. Metadata catalog access to available topic and data set descriptor names/attributes (metacards) for data, access to active service specifications to lookup service endpoints &amp; descriptors for services.</td>
<td>DDS Topics, DDS4CCM</td>
<td>OMG</td>
</tr>
<tr>
<td>9</td>
<td>Time Management</td>
<td>Access to current, multi-node synchronized system time. Timer event management encapsulation.</td>
<td>POSIX &amp; ACE Timers</td>
<td>IEEE/ISO/IEC &amp; DOC Group</td>
</tr>
<tr>
<td>10</td>
<td>Math Libraries</td>
<td>Mission independent math libraries supporting common signal &amp; image processing functions per a co-processor off-load model.</td>
<td>VSIPL++</td>
<td>HPEC-SI</td>
</tr>
<tr>
<td>11</td>
<td>OS Abstraction</td>
<td>Access to basic resources managed by the OS, for intra-process component design (vs. the 10 CSS APIs, which are primarily inter-process) and driver level access by Adapter components for handling external I/O. Prefer use of ACE OS abstraction layer in System Management API instead for future portability (Linux, Windows, VxWorks, etc.).</td>
<td>ACE, POSIX</td>
<td>DOC Group, IEEE/ISO/IEC</td>
</tr>
</tbody>
</table>

**MOSA APIs used directly by apps with no intermediate proprietary abstraction/shim layer in between**
SNA Platform v1.0 Run-Time HW/SW Stack

DDS4CCM Content in Red

Visualization Application Component Layer

Processing Application Component Layer

**CCM+DDS Container (CIAO)**

- **Logging Lib** (log4cxx)
- **Appenders** (DDS, Others)
- **Logging Protocol Aggregator** (to log4j XML)
- **DDS4CCM PSAT (CIAO)**
- **RTSP Data Record Service**
- **RTSP Data Playback Service**
- **DDS Libs & Threads (RTI)**
- **Wideband (OFED)**
- **DDRTTPS Message Protocol (RTI)**

DD4CCM Discovery Connector

- **Data & Service Discovery Topics (RTI)**
- **Time Mgmt Libs (ACE + POSIX APIs)**
- **PTTI & Time-Keeper Time Services**

AMI4CCM (CIAO)

- **CORBA 2.x Libs/ORB (TAO)**
- **CORBA IIOP Message Protocol (TAO)**

SMS (DAnCE)

- **OS Abstraction Layer (ACE)**

**Security Services** (SELinux, IPSec)

**Operating System, Device Drivers, Network Stack** (Linux - Fedora 12)

**HW & Boot Loader Firmware** (Intel x86-64, GPU, IBM Cell, PPC440)
EDA – Event Driven Architecture

• Architecture pattern supporting real-time system design
  – Hardware analogy: Interrupt-driven vs. Poll-driven design

• Primarily defined by the event-driven *programming model*
  – Leverage reactor pattern to dispatch incoming “events” to user-registered event handlers, a.k.a. “callback” routines
  – Most business logic embodied in callbacks, or code called from callbacks

• Business logic invoked by real-time events, including:
  – Incoming subscribed messages (pub-sub events)
  – Incoming service invocation messages (request-response events)
  – Asynchronous responses to service invocations by a client (AMI)
  – Timer events
  – Exceptions
  – Other registered activity on file descriptors (FDs) or equivalent (interrupts, AIO, file system activity, socket/pipe activity, etc.)

• Typically associated with data-driven design patterns
  – Pub-sub, streaming data, message passing

Both DDS and DDS4CCM (CCM) Satisfy the Teton EDA Architectural Tenet
Advanced real-time distributed processing systems utilize all of these patterns to some extent, and greatly benefit from combining and using them where appropriate.
### SNA Supports Pub-Sub & Request-Response Information Exchange Patterns

<table>
<thead>
<tr>
<th>Event Driven Architecture (EDA)</th>
<th>Service Oriented Architecture (SOA)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronicity</strong></td>
<td></td>
</tr>
<tr>
<td>Typically asynchronous, message-</td>
<td></td>
</tr>
<tr>
<td>driven</td>
<td></td>
</tr>
<tr>
<td><strong>Coupling</strong></td>
<td>Provides “Goods” (Nouns)</td>
</tr>
<tr>
<td>Looser coupling</td>
<td>“Services” (Verbs)</td>
</tr>
<tr>
<td>“data driven”</td>
<td></td>
</tr>
<tr>
<td><strong>Orientation</strong></td>
<td>Roles Publishers and Subscribers</td>
</tr>
<tr>
<td>Timely handling</td>
<td>Services and [Service] Clients</td>
</tr>
<tr>
<td>functionality</td>
<td></td>
</tr>
<tr>
<td><strong>Use</strong></td>
<td>Exchange Pattern</td>
</tr>
<tr>
<td>Typically used in</td>
<td>Smart-Pull information exchange</td>
</tr>
<tr>
<td>and data driven</td>
<td>initiated by a downstream app</td>
</tr>
<tr>
<td></td>
<td>(subscriber), requesting information</td>
</tr>
<tr>
<td><strong>Embedded</strong></td>
<td>from a topic (data set/stream)</td>
</tr>
<tr>
<td><strong>Analogy</strong></td>
<td>produced by an upstream app (publisher)</td>
</tr>
<tr>
<td>Analogous to “input driven”</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Dynamic Discovery</td>
</tr>
<tr>
<td>EDA component</td>
<td>Topics discovered as metacards in a</td>
</tr>
<tr>
<td>(RT) and near real time, C4ISR</td>
<td>data registry</td>
</tr>
<tr>
<td>systems,</td>
<td></td>
</tr>
<tr>
<td><strong>Interaction</strong></td>
<td>Registry Info</td>
</tr>
<tr>
<td>Best suited for</td>
<td>Metacards describe common</td>
</tr>
<tr>
<td></td>
<td>attributes for a set of</td>
</tr>
<tr>
<td></td>
<td>events/data instances that are/will</td>
</tr>
<tr>
<td></td>
<td>are made available, and provide a</td>
</tr>
<tr>
<td></td>
<td>topic identifier</td>
</tr>
<tr>
<td><strong>Binding</strong></td>
<td>Binding</td>
</tr>
<tr>
<td>Applications bind</td>
<td>Requester (subscriber) must</td>
</tr>
<tr>
<td>streams</td>
<td>explicitly specify a topic</td>
</tr>
<tr>
<td></td>
<td>identifier, either discovered in a</td>
</tr>
<tr>
<td></td>
<td>data registry entry or</td>
</tr>
<tr>
<td></td>
<td>known a priori</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Connection</td>
</tr>
<tr>
<td>Connection-Less</td>
<td>Connection-Oriented</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Coupling</td>
</tr>
<tr>
<td>Most appropriate (RT) and near</td>
<td>Loosely coupled information</td>
</tr>
<tr>
<td>real time, C4ISR systems,</td>
<td>exchange</td>
</tr>
<tr>
<td><strong>Used For</strong></td>
<td>Coupling</td>
</tr>
<tr>
<td>Event Handling, Message Passing,</td>
<td>Tightly coupled information</td>
</tr>
<tr>
<td>Data Streaming</td>
<td>exchange, although amenable to</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>loosely coupled workflow</td>
</tr>
<tr>
<td>Typically used in a “Data”</td>
<td>orchestration</td>
</tr>
<tr>
<td>architecture as the preferred</td>
<td></td>
</tr>
<tr>
<td>means of distributing information</td>
<td></td>
</tr>
<tr>
<td>that may be of interest to</td>
<td></td>
</tr>
<tr>
<td>multiple parties</td>
<td></td>
</tr>
<tr>
<td><strong>Mapping</strong></td>
<td>Mapping</td>
</tr>
<tr>
<td>Use for One-to-Many and Many-to-Many component interactions</td>
<td>Use for One-to-One and Many-to-One component interactions</td>
</tr>
</tbody>
</table>

Advanced real-time distributed processing systems utilize all of these patterns to some extent, and greatly benefit from combining and using them where appropriate.
SOA – Service Oriented Architecture

- Desired architecture pattern for most new systems/programs
- Specifically called out in DoD reference architectures and directives, particularly for Air Force and Navy systems
  - Primarily due to the strong influence of web-based enterprise architectures using Java EE or .NET, but the pattern is quite useful in distributed, real-time embedded (DRE) systems as well
- Inherently implements the Request-Response information exchange pattern
  - RPC vs. DOC programming style comes naturally and is desirable to most OO programmers
- Loose coupling and easy composition (workflows) of modular functionality
- Services as groupings of common functionality with a well-defined interface, defined by a single service specification
  - Captured and looked up in a service registry (e.g., UDDI, CORBA Naming Service, LDAP, or Directory Services built with DDS per the SNA Platform)
- Client-serv[er|ice] interfaces have proven popular on Teton-supported programs
  - Especially for defining/building “collocated” services (class libraries) in a language independent fashion using an IDL “interface” definition as the service specification

DDS4CCM Arguably is a Real-Time SOA, DDS Alone Only “Supports” a SOA
SOA Shortcomings When Using DDS Alone

• No concept of a “service” as a grouping of like functionality, and thus no service discovery
  – Only data discovery, topics and lots of message definitions per DDS’s data vs. service orientation

• No concept of an "interface" as a grouping of commonly handled DDS-only message types
  – The concept of a deployable service would need to be built as a non-standard custom entity

• Common request-response information exchange patterns would be much messier to implement due to potentially many required topics per “service”
  – Topic pairs for each request and response message type, and separate pairs for each service operation
  – Would need to be encapsulated under a proprietary, non-standard abstraction to be workable
  – Requires custom application layer header fields to support request-response message correlation

• No support for an RPC (Remote Procedure Call) programming model
  – RPC is the assumed IPC model in UML-based OO/distributed-object design tools, barring profiles/stereotypes to support events and message passing unique to pub-sub
  – Without custom DDS extensions, only “DOC” style, message-oriented services would be available

• No open standards based "well defined” service interface available for DDS
  – DDS does not support IDL “interface” definitions, so a custom documentation standard would be needed

• Cannot claim the politically advantageous mantle of being a “Real-Time SOA”

• SOA "workflows" would be more difficult to build
  – Best defined using branching sequences of synchronous RPC calls & returns combined with mediation
• Recent DoD OA/MOSA specs, program requirements and reference architectures usually mandate the development/use of software “components”
  – Typically interpreted as meaning “modular,” rather than necessarily mandating the use of a formal, standards-based component framework
  – However, NGC customer presentations proposing a standards-based component framework (DDS4CCM), rather than a custom one, have been very well received

• Adds the ability to formally define the Structural architecture model
  – SNA has adopted an architecture definition that encompasses separate Behavioral, Structural, Concurrency and Resource models (per SSCI, ATAM), supported by Module, Component & Connector (C&C) and Allocation architecture styles & views
  – Concurrency and Resource models greatly enhanced by DDS4CCM as well
    • Per C&C Assembly, Deployment and Domain diagrams, and standard XML artifacts generated from them, compatible with the OMG CCM and D&C specifications

• Brings many advantages to an architecture above and beyond powerful, interoperable messaging middleware such as DDS
Why Component Based Development (CBD)?

- **Modularity**
  - Components can be independently updated or replaced without affecting the rest of a system
  - CBD adheres to the time-tested “black box” design methodology
- **Reuse**
  - Software is reusable at the component level, vs. a full system level
  - A component reuse library such as OER is a key element
- **Interoperability**
  - Container-based development ensures interoperability between application components
- **Extensibility**
  - A CBA (Component Based Architecture) is inherently loosely-coupled, supporting easier extensibility of component and system functionality
- **Reduced Complexity**
  - Encapsulation, modularity, separation of concerns and the establishment of hierarchical component dependencies, or “layers”, all contribute to reduced design & system complexity
- **Reduced Design Time**
  - Faster time-to-market, shortened program/software development schedules
  - Focus changed to composition of a software-intensive system, vs. all new design
- **Lower Design & Maintenance Costs**
  - Result of shorter design times, reuse, less complexity
- **Quality & Reliability**
  - Reuse and test/maintenance at the component level vs. at a monolithic system level
What is a Software Component?

- Independently revisable unit of software with a well defined interface
- Well defined interfaces are called “ports”
- Able to be packaged into an independently installable set of hierarchically defined files
- Smallest decomposable unit that defines standard ports is called a “monolithic component”
- A “component assembly” is a higher level aggregation of monolithic components or other component assemblies
- A component/assembly defines an aggregation of multiple ports

Basic CBDDS CBA Symbology & Terms used on SNA
CBA Shortcomings When Using DDS Alone
(1 of 2)

• No formal concept and implementation of a software "component" as a reusable software module with well-defined and controlled interfaces/ports
  – CBA offers the promise of modularity, reusability, composition, and many other desirable system architecture features
  – Modularity and reuse, particularly when implemented via standards-based components, is the most desirable feature that our customers really want/assume when they say "SOA"
  – Formal specification of a component via IDL3 is valuable (vs. some ad-hoc/proprietary method)

• No support for the concept of an application "container" as a standard execution environment for component business logic
  – Encapsulation of tedious boilerplate setup code and lower level middleware details
  – Underlying interface with system management and security services
  – Ability to transparently initialize and integrate other core services
  – A consistent event-driven architecture (EDA) programming model built into the container
  – Encapsulation of event dispatch and threading issues (big portability advantage)

• No standard defined for deployment via System Management Services (SMS)
  – For instance the OMG D&C open-standard specification – no equivalent for DDS alone
  – Offers a standards-based method for starting/stopping/modifying a distributed software system
  – Use of the DAnCE implementation of the D&C spec Execution Model is a huge plus that comes with CCM and CBA
CBA Shortcomings When Using DDS Alone
(2 of 2)

• Lack of a rich tool suite for managing the full life cycle of software components in a CBA
  – Separation of design from deployment is particularly valuable for reuse
    • MDA tools exist to support these phases independently
  – Component attributes settable via a deployment planning tool offer excellent ability to easily configure system operation, post-design
  – CBA and its independent, formal deployment planning phase change the system development focus from design to composition of new high-level capabilities using pre-existing components as “software parts”

• Lack of a common graphical notation for describing the Structural architecture
  – See example on next slide – extremely useful for cross-program understanding and reuse

• No concept of component “assemblies”
  – Come with associated tool & file format standards for software packaging & deployment preparation
  – Useful for specification of deployable instances, as well as source code organization and sharing via component reuse libraries

• No standard file format for describing a deployment
  – OMG D&C spec CDD (Component Domain Descriptor) and CDP (Component Deployment Plan) files used for CBDDS
Real Time Software Component Framework
Layered Infrastructure – Notional System Design

Layered Architecture
View 1

Real-Time (RT) Software Component Framework

- MDE PIM design transforms to OMG CCM+DDS PSM
- Working with OA RT middleware community & OMG to advance CBDDS for HPC apps
- CBD tooling support
- Common framework used for both GP & SP apps
- Supports both SOA & EDA architecture patterns
- Well defined port interfaces specified in IDL3+
- OA APIs used directly by components
- External I/O & device adapters may access OS layer in lieu of proprietary isolation layer (full OA)
- Portable VSIPL++ math
- OA sensor applications successfully being built on this emerging OA RT component framework
New CBDDS Tooling Offers a Formal Architecture Description Language (ADL) Enabling the Graphical Specification of a Structural Architecture

• Formal ADL enables auto-generation of a large percentage of the operational software for a software intensive system

• CBA CCM graphical notations and component diagrams are proving to be invaluable as a common cross-program means of describing and sharing architectures and components

• C&C Assembly, Domain and Deployment diagrams very useful as specific, standard architecture views
  – Component diagrams offer a “software schematic” view of intended system composition using standard software parts
    • Very similar to hardware schematics connecting standard hardware parts
    • Connections drawn in an MDA modeling tool are automatically established during the deployment launch phase – big time/code savings to developers
    – Domain diagrams capture aspects of system resource utilization
    – Deployment diagrams define key elements of both the Resource and Concurrency architecture models

• The next 4 slides are included to illustrate the power of having a common CBDDS graphical ADL to represent alternative SOA/EDA architecture patterns
  – System optimization, tradeoffs, alternative deployment of same component designs
Assumptions:

- Periodic state data (e.g., GPS) is arriving from outside the system – it must be ingested and stored (keep state history).
- The state data must be normalized to internal CBDDS formats and protocols (per Adapter) before use by client components.
- Clients exist throughout a distributed system that need on-demand access to the collected state data through a value-added service that returns massaged or indexed state data (e.g., interpolations, latest average, etc.).
- Examples 1a, b & c are concerned with optimizing on-demand client access latency to incoming state data, not overall latency.
1C) Embassy Pattern Approach

Data received & stored by ONE instance of an Embassy component on each node (smaller memory footprint)

Lowest combo of access latency & memory, but with highest complexity, most components & implicit interfaces

**NOTE:** The Shared State Memory interface between these components is IMPLICIT, vs. explicit via use of standard CBDDS ports. This is not generally recommended, but is sometimes [currently] necessary, pending future support for a Shared Memory DDS4CCM connector. When doing this, these two components MUST be deployed together as a deployment assembly, and the Deployment Planner must be aware of the hidden constraint that they must be deployed on the same node.
CBA Design Patterns – State Distribution 2
Pub-Sub with Single Stateless Service Approach

Optimizing Overall Latency w/Service Support
Distribute incoming state event all the way to client vs. memory

Assumptions:
• Periodic state data (e.g., GPS) is arriving from outside the system – it must be ingested and stored (keep state history)
• The state data must be normalized to internal CBDDS formats and protocols (per Adapter) before use by client components
• Clients exist throughout a distributed system that need immediate access to the incoming state data, as well as on-demand access to a value-added service that returns massaged or indexed state data (e.g., interpolations, latest average, etc.)
• Unlike the prior 1a, b & c examples, 2a & b are concerned with optimizing overall latency to incoming state data events so that client components get the state change event ASAP
  • In the prior examples, the incoming state change event propagated only so far as the State Memory and then stopped
  • A separate (unspecified) event prompted the client components to make a service call to access the collective stored state data (which could be quite stale)
• The goal in the prior 1a, b & c examples was to minimize the time from the client call until value-added data was returned, whereas for 2a & b we are trying to minimize the time from external state event arrival until the client has access to it
CBA Design Patterns – State Distribution 2
Pub-Sub with Distributed Stateless Service Approach

Optimizing Overall Latency w/Service Support
Distribute incoming state event all the way to client vs. memory

2B) Local Service Approach w/Pub-Sub
Same exact component designs as 2A - alternative deployment
Same as prior, but the stateless "Helper" service is collocated with the client
More components, but entire State Memory data set required by service call may be passed by reference
via a local virtual function call (no underlying RPC message transport required)
DDS4CCM Connector Implementation
Logical View Expanded to Basic Port View Showing Fragments

Component A
- DDS_Write
- DDS_Read
- ServiceB

Component B
- DDS_Listen
- DDS_Update
- ServiceA

TopicX using DDS_Event Connector
- TopicY using DDS_State Connector

Logical View
- DDS::DataReader
- DataListenerControl
- Listener

Basic Port View
- DDS::DataWriter
- DDS::DataReader
- DDS::DataWriter

Complexity Encapsulated
Boilerplate Code Auto-generated & Abstracted
MDA – Model Driven Architecture

- Improved support for MDA/MDE is a prominent initiative at NGES
  - Also popular with our DoD customers

- Design and Deployment Planning Tools for DDS are virtually non-existent
  - Sparx System’s Enterprise Architect (EA) and IBM’s Rhapsody tools support the OMG UML4DDS standard
    - Low level UML profile only - has little value in terms of expressing an architecture
    - Strong support exists for run-time DDS tooling, but not Design or Deployment Planning

- CBDDS per DDS4CCM, being component-based, is inherently amenable to modular CBD design methodologies, processes & architecture views
  - DDS is “just” a pub-sub middleware, albeit a very powerful one
  - UML-based design tools are inherently OO and extremely oriented toward SOA and distributed object architectures, without specific tailoring

- NGC continues to work with 3 tool providers/vendors to offer full fledged tooling support for CBDDS (DDS4CCM)
  - Design, IDE integration, Deployment Planning – not just I&T support
  - Atego’s Artisan Studio (design & deployment), Zeligsoft’s CX for CBDDS (design & deployment), DOC Group’s CoSMIC (deployment only)

**DDS4CCM has MDA Tooling & is Extremely Amenable to MDA, DDS Alone is Not**
A Three-Stage GUI-Based Tool Chain Supports the Six-Step Teton SNA CBD Software Lifecycle Process

Design Tool
- Artisan Studio
- Zeligsoft CX

IDE Tool
- Eclipse
- Artisan Studio

Deployment Planning Tool
- Zeligsoft CX

Architecture & System Design
- System Software Design & Component Definition

Component Interface Design
- Component Interface Design

Component Software Design
- Component Software Design

Component Implementation
- Component Implementation

Component Packaging & Assembly
- Component Packaging & Assembly

Component Deployment, Integration & Reuse
- Component Deployment, Integration & Reuse

SNA CBD Software Lifecycle Process

Key Artifacts
- IDL: Interface Definition Language (OMG)
- CPP, H, SO
- CDP, CDD

IDE: Integrated Development Environment
CBD: Component Based Development
SNA: Scalable Node Architecture
IDL: Interface Definition Language (OMG)
CDP: Component Deployment Plan
CDD: Component Domain Descriptor
In a DDS-only Solution, the Entire CBA Lifecycle Process and Almost All Tools Would be Lost

- DDS alone only has extensive tooling for run-time debug, but no design tools, no deployment tools.
- DDS alone has no components, no deployment framework, no structure around which to build modularity, portability, reuse or a value-added IDE.

Key Artifacts

- Architecture & System Design
- Component Interface Design
- Software Design
- Implementation
- Packaging & Assembly
- Component Deployment, Integration & Reuse
- System Integration, Test & Verification

- Artisan Studio
- Zeligsoft CX
- Eclipse
- All that’s left to build a tool chain and process are bare middleware libs/APIs amenable to standard OO code-level design.

IDL for raw messages only – not services or reusable components.
MDE Tools are Available to Support CBD for CBDDS per New OMG DDS4CCM Spec

Artisan Studio (Atego)
UML-based Design & Deployment Planning tool

Zeligsoft CX (Zeligsoft → PrismTech)
UML-based Design & Deployment Planning tool
Design – Hierarchical Component Diagrams in UML Using Artisan Studio
Design – Hierarchical Component Diagrams in UML Using Artisan Studio
Design – Hierarchical Component Diagrams in UML Using Zeligsoft CX
Design – Hierarchical Component Diagrams in UML Using Zeligsoft CX
Design – Hierarchical Component Diagrams in UML Using Zeligsoft CX
Deployment Diagram in Artisan Studio

Deployment diagram that maps component and/or deployment assembly instances to processes/containers, and processes to processing nodes (OS instances)
Domain Diagram and Deployment Plan in Zeligsoft CX
Deployment Diagram in CoSMIC
OMG IDL Used to Define SNA Software Components, Messages and Interfaces

- The Object Management Group (OMG) Interface Definition Language (IDL) defines pub-sub data types & request-response service interfaces
  - ISO/IEC 14750 | ITU-T Rec. X.920 standard as well
- Language independent format
  - OMG standards also exist for IDL to C++ mapping, IDL to Java, etc. (new IDL to C++0x in works)
- A given middleware standard implementation provides an IDL to language compiler
  - Auto-generates language specific code abstracting away transports, serialization/de-serialization & endian swapping issues from applications
  - IDL compiler called prior to platform specific language compiler as part of project build
  - Produces client stubs, service servants/executors, and pub/sub message classes
  - Model generated IDL -> IDL compiler generated source = large percentage of design code base
  - SNA currently uses tao_idl and rtiddsgen IDL compilers
DDS vs. CBDDS

**DDDS: Data Distribution Service**
- Popular, powerful pub-sub messaging DRE (distributed, real-time, embedded) middleware
- Offers:
  - OA, EDA
  - Interoperability, Performance
  - Location-independent messaging and state distribution

**CBDDS: Component Based DDS**
- Greatly enhanced version of DDS that addresses the standards-based integration of DDS with other OA common core services required by all software-intensive system designs
- Extends DDS to add:
  - SOA, CBA, MDA
  - Reuse, Modularity
    - Adds structure to your architecture, not just interoperable messaging
  - Portability
    - Standards-based OMG DDS4CCM abstraction layer for DDS (vendor neutrality, transparent use of alternative middleware standards – not just DDS)
    - Portable, “container” based execution environment (threading, event queue/dispatch, clean integration of Logging, Time Management and Security)
  - Additional core services – System Management, Service/Client Messaging, PSAT
DDS vs. DDS4CCM Summary

- DDS alone only addresses the OA and EDA architectural tenets
- DDS4CCM adds support for SOA, CBA and MDA
  - Critical additional features/requirements for Teton
  - DDS alone provides performance & interoperability, but not modularity
  - The additional CBA & MDA promise of reduced development costs & schedules not realizable with just DDS
  - Custom/proprietary SOA service definitions and protocols required with just DDS
- A good case can be made that DDS4CCM (CBDDS) is a Real-Time SOA
  - DDS alone only provides support for a Real-Time SOA
- No CBA means no standard execution container, auto-gen of boilerplate code and no end-to-end development process for SNA
  - No standard modularity – “free for all” process/thread level design and access of OS level system resources to duplicate CBA container functionality in a variety of custom ways
  - Greatly reduced software reuse, and thus system development cost & schedule
- No MDA means no design or deployment planning tools, no modeling support, no ADL, no extensive auto-code (IDL) generation
  - Higher development costs, longer program development schedules

Teton’s Driving Architectural Tenets
- OA Open Architecture (MOSA)
- EDA Event Driven Architecture (DOA)
- SOA Service Oriented Architecture
- CBA Component Based Architecture
- MDA Model Driven Architecture