Determining Whether CORBA is Suitable for Implementing Real-time Airborne Radar Applications

Carolyn Boettcher
Robert Moore
{cbboettcher,rjmoore}@raytheon.com
July 17, 2002
• Modern airborne radar software is distributed
  » Executes on a network of processors that communicate over a high-speed bus

• Radar itself is a subsystem within the distributed mission avionics system
  » Central computer (a.k.a. mission computer) is the controller
  » Various subsystems cooperate to achieve mission objectives

• Communication between various sensors can implement sensor fusion for improved situation awareness
  » E.g., fusion between objects in a radar map and objects in infrared image
Example Radar/FLIR Fusion
High Level Block Diagram
• Radar data processor is the controller for the distributed radar software and communicates with other avionics subsystems
• Signal processors perform the parallel, numeric-intensive processing
• Sensor front end receives control commands from software and sends signal data to the signal processing nodes for post-processing
Why CORBA?

- Advantages of using CORBA in airborne radar
  - Provides high level application programming interface
  - Improves application portability to new distributed processing architectures
  - Replaces custom middleware that is otherwise needed
  - Increases COTS content of software
  - Makes it easier to add/modify interfaces between systems, esp. from different contractors

- Disadvantages/impediments to using CORBA
  - Increases performance overhead
  - May decrease performance predictability
  - Must overcome institutional bias
Potential Use of RT/CORBA in Raytheon’s Airborne Radars

- Messages between CC and radar
  » Requirements can probably be met by RT/CORBA if an adequate physical network layer is available

- Internal radar control messages
  » Middleware is needed for messages between objects on same and distributed computers
  » RT/CORBA may meet requirements

- Internal radar signal processing data transfers
  » Middleware is needed for messages across distributed computers
  » Has the most stressing requirements for RT/CORBA

First step - convert existing non-realtime CORBA demo of signal processing application to use ACE/TAO
CORBA Demo - First Step

- Convert existing non-realtime CORBA demo of signal processing application to TAO v. 1.2
  - Execute on network of Windows 2000 workstations
  - Use and Assess Important CORBA Features and Services
    - Portable Object Adapter (POA)
    - Event Service
    - Life Cycle Service
    - TAO Implementation Repository
  - Use portable design
  - Demonstrate creation/control of distributed CORBA objects
- Do Preliminary Assessment of ACE/TAO
  - Features/Learning Curve/Portability/Performance Penalties
- Document lessons learned
Two Dimensional FFT Application (2DFFT)

- A prototypical Synthetic Aperture Radar (SAR) map application
- Represents the essentials of a real world distributed radar application
- End result of processing is a displayable image

Features

- Requires creation and destruction of remote distributed objects
- Requires barrier synchronization between interacting objects
- Requires many-to-many communications (i.e., for distributed corner turning)
- Easy to Validate
- Easy to Benchmark and Analyze
Number of parallel processing nodes can range from 10 to 100’s depending on the map coverage (i.e. number of pixels)

Size of data blocks also varies depending on coverage
  » Ex. 40 Kbytes blocks for input data
  » Ex. 10 Kbytes blocks for distributed corner turning

During corner turning, every node sends data to every other node
  » For each receiving node, timeline tries to avoid more than one node sending to it at any given time

Latency of corner turning is more critical than input data latency
  » Transmission time estimate 100 microseconds
  » Latency goal (including OS and middleware overhead) is 200 microseconds
  » Short control messages must also be accommodated
Radar Corner Turning Example (with 8 processing nodes)
Four Classes of Interacting CORBA Objects

- RootSAR
- ComputeSAR
- Controller
- FactorySAR*

* Used by Controller to remotely create RootSAR and ComputeSAR objects
**Functionality**

» Demonstrate remote creation/control/management of CORBA
  – Use CORBA Name Service to locate required objects

» Demonstrate mastery of required security features

**Develop GUI**

» The user may:
  – Specify input parameters
  – Start/Pause/Stop the 2DFFT application
  – Query state of any active 2DFFT object
  – Save specified data to disk

» Display image produced by the 2DFFT application
- One Instance created by the Controller object
  - Requires the location and use of a SARfactory object already resident on the specified node
- Receives input parameters from the Controller object
- Controls and synchronizes the processing done by the associated ComputeSAR objects
- Receives computed image segments from the associated ComputeSAR objects
- Assembles image segments into a single image and sends the latter to the Controller Applet for display
**Multiple instances are created by the Controller object**
   » Requires the location and use of SARfactory objects already resident on the specified nodes

**Each Instance**
   » Performs the current processing step and then blocks
   » Reports to RootSAR when the current step is completed
   » Begins the next processing step when commanded by RootSAR
   » Destroys itself when all processing steps are completed

**Two types of processing steps**
   » Local Computation (e.g. FFTs)
   » Communication with other 2DFFT objects
      – Data transfer to/from the RootSAR object
      – Data exchange between all ComputeSAR objects
2DFFT Objects use three *asynchronous* communication modes
- Broadcast
- Point-to-point
- Many-to-many (corner turn)

CORBA Event Service suitable for the required communication modes
- Avoids low level code
- Enhances portability
- Allows realistic test case for Event Service
- Uses untyped event channels
  - All data passed must be type CORBA::Any
Broadcast

- A message (actively) sent from Root is (passively) received by Compute Objects
  - Used for Root to send commands

Diagram:
- Root
  - Send: Message m
  - Recv: Message m
  - Compute_0
    - pull consumer (passive)
  - Compute_1
    - pull consumer (passive)
  - Compute_N-1
    - pull consumer (passive)
Point-to-point

- Distinct messages are sent by Compute instances to Root
  - Used for Compute instances to send final image segments to Root
Many-to-many

- Data is sent to and received from every Compute instance
  - Used by Compute instances to implement global transpose (corner turn)
Demonstrate Radar/CC interactions

- Radar to CC messages
  - 5 messages at 20 Hz
    - Total size 236 bytes
  - Track file updates at .5 Hz (2 secs)
    - Each track is 28 bytes
    - Maximum number of tracks varies (10-100 maximum)
    - Designated track has 25 ms. latency requirement for initiating transfer after update

- CC to Radar messages
  - 3 messages at 20 Hz
    - Total size 117 bytes
  - 1 message at 10 Hz is 68 bytes
  - 1 message at 1 Hz is 9 bytes

This list is not complete, but is representative
Radar and Central Computer applications are in C++

RT ORB components are installed on Radar and CC (ACE/TAO)

Both Radar and CC have synchronized real-time clocks which a C++ application can access via system call

Create data structure for each message rate for Radar and CC

Use CORBA calls or messaging service
ACE/TAO Lessons Learned [1]

● TAO v. 1.2 implements most of CORBA 2.6 and has additional features from CORBA 3.0 (asynchronous message invocation)

● Moderately steep learning curve
  » Even if familiar with CORBA
  » Need to spend significant time and effort to build, study, and modify sample programs included with the distribution

● High quality CORBA implementation
  » Quickly tracks and implements new standard CORBA features
  » Has some non-standard features, but these can be often be avoided with minimal effort
  » Good overall performance
  » Portable to a large number of platforms
    – May have build problems with non-supported C++ compilers
ACE/TAO Lessons Learned [2]

- Large amount of high quality documentation
  - May have to search diverse collection of help pages and white papers to find needed information
  - Very good commercially supported documentation available (OCI)
- Default settings well chosen
  - However, if some settings are not appropriate, may take some time to find the correct values.
- Summary
  - Expect to spend a significant amount of time getting familiar with the details of using the ACE/TAO ORB, but the effort is worth it.
Future Work

- Get performance metrics on CORBA SAR demo
- Implement CORBA CC/Radar demo
- Implement CORBA Sensor Fusion demo
- Analyze security issues/solutions with RT/CORBA
- Port demonstrations to RT/OS and embedded processing architecture
  - Raytheon’s RT/Secure OS
  - VxWorks™
  - Raytheon’s proprietary middleware (ACE has already been ported to this for a radar simulation environment)