Toward Lightweight Fault-Tolerance for Distributed Real-Time Systems

Nanbor Wang¹, Steve Osselton², Simon McQueen², Jai Balasubramanian³, Jeff Parsons³, Robert Kukura², Andy Gokhale³, and Andrew Foster²

¹Tech-X Corporation
²PrismTech Corporation
³Vanderbilt University

OMG Workshop on Distributed Object Computer for Real-time and Embedded Systems

July 15, 2008, Washington DC

Partially funded by the Navy Surface Warfare Center, Dahlgren Division, the US Navy Sea System Command under contract No. N00024-07-NR-52502
Motivating Examples

- Modern warfighting systems are moving from traditional custom-made system-of-subsystems toward enterprise management computing systems.
- Distributed middleware provides key support for integrating software components and subsystems.
- Mission-critical systems and components need to satisfy key requirements:
  - Real-time performance, e.g., hard real-time, soft real-time, or none.
  - High-availability and survivability, i.e., fault-tolerance.
Overview – FT & RT Support in Distributed Middleware

• **FT-CORBA**
  - Defines interfaces and mechanisms for implementing and managing FT applications
  - No full implementation available

• **RT-CORBA**
  - Interoperable and portable distributed RT applications
  - Widely accepted in modern warfighting system development
  - Mandatory Open Architecture Computing Environment (OACE) standard
The Need to Integrate FT Middleware with FT Devices

• RT CORBA is widely adopted
• Not so for FT CORBA
  – Overly complicated – requires large infrastructure to support FT CORBA
  – The CORBA-centric approach
    • Does not support systems using multiple middleware technologies
    • Presents limited view of global system issues
    • Does not map well to real-world problems
    • Ignores widely adopted FT technologies such as transactional middleware
  – FT CORBA spec is basically defunct
• There is a need to integrate distributed computing middleware with widely accepted and proven FT technologies
A Typical Hardware-based Fault-Tolerance Device

- Mission-critical services often require redundancy
  - With accompanying mechanisms for monitoring, analyzing and recovering from faults
  - Existing mechanism(s) for state synchronization
- How can we integrate distributed computing (CORBA) applications with such devices?
  - Without heavyweight CORBA infrastructure
  - Leveraging existing FT mechanisms and system-level knowledge
Lightweight FT CORBA Concept

- To integrate CORBA applications with existing FT mechanisms, not re-invent/re-package the FT mechanisms

**LwFT CORBA**
- Limits to managing invocation routing
- Allows redundant service
- Provides transparent connection/fail-over between client and server
- Detects and reports middleware-level faults to external FT mechanisms

**External FT mechanisms**
- Handle the majority of faults and cue LwFT CORBA how to handle faults (how to re-route invocations)
- Provide state consistency mechanisms
Key Considerations in LwFT Design

- Eliminating the need for large CORBA infrastructure
- Initialization of FAs and FPs
- Identity of FT objects
- Updating FP with new Primary
  - Ability to interrupt an ongoing invocation
- Describing an FT object reference
- Allowing mixed ORBs from different vendors
- Incorporating different multicasting mechanisms (local interfaces)
- Controlling granularity of fail-over
  - Object references, POA, Process endpoints
- Minimizing impact on clients
  - Minimizing efforts to support new clients

- Current approaches available
  - Implementation Repository
  - Object Reference Template
  - Simple lookup service + interceptor

- Our current efforts
  - Summarize pros and cons for each approach
  - Encourage dialogues
  - Propose improved solutions
  - Develop software-based FT strategies for future uses
    - Load-balance (FLARe)
Core LwFT Mechanism: LOCATION FORWARD

- Key goal of LwFT4DRTS RFP – to provide FT with little modification to applications and ORBs
  - IOGR will not be used
  - LOCATION_FORWARD offers the standard mechanisms for redirecting client invocations
  - Semi-active replication style

- Key parts:
  - Servers: Register availability of self, generate modified IOR pointing to Forwarder
  - Replica Registry: interact with the FT mechanism and maintain a list of available server endpoints
  - Forwarder: Redirect client invocations
  - Clients: Initial invocation to forwarder gets redirected to the primary server until fault
An Example Interaction Diagram

- Servers modifies generated IOR to point to Forwarders
- A new FT_Locate message type is suggested to acquire the full IOR with RT tags

If the ServerReplica crashes while a Client has a connection to it, or between successive connections, the Client will simply default back to the original IOR as explained in Case 5.
FT-Group Management

FT Group should fail over by process

- When a replicated process fails, all replicated objects get redirected
- Server ID is required in order to identify replica process

Different group management strategies:

- Manage only process endpoints
  - All server replicas must be alike
  - Only server replicas generate IOR (using endpoints from Registry)
  - Group ID = POA names + object id
  - 2 forwardings required (to acquire IOR with RT tags)
  - ImR styled IOR generation

- Manage group references
  - Can configure replicas differently
  - Server replicas or Registry can generate IOR
  - Group ID = flat name -> object key
  - 1 forwarding required (RT tag info are kept with Group management)
  - Replicas must register individual references for each group – can be a source of inconsistency
  - Process failover also handled (using server ID)
Deployment of LwFT Modules

- Many strategies to locate registries and forwarders
- Registries and forwarders can be made FT (eliminating the need to replicate them and select a “primary” registry/forwarder)
- FT Proxy (FP)
  - Encapsulates FT-related connection/reconnection and fail-over operations
  - Detects broken connection & restarts invocations when necessary
  - Reports middleware faults
  - Needs to be the least intrusive to client code
Prototype Efforts

- Tech-X, PrismTech, and Vanderbilt teams are working on various aspects discussed
- Using DDS as an external FT mechanism
  - For object state synchronization
  - For fault detection (using the deadline QoS policy)
  - Have tried 2 different DDS implementations
- Exploring different group management strategies
- Exploring collocating forwarder w/ client request interceptor
- Making LwFT modules themselves FT
  - Making states “distributed” using DDS (for example)
  - Using local interface to stress the “distributed states” nature of these modules
  - Using regular CORBA interfaces to provide more flexibility in deploying modules
- Developing examples to demonstrate how LwFT applications can be developed and deployed
- Exploring using DDS as alternative FT GIOP transport