Techniques for Dynamic Swapping in the Lightweight CORBA Component Model

Jaiganesh Balasubramanian
jai@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~jai

Dr. Aniruddha Gokhale
gokhale@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~gokhale

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee
Motivation: Highly Dynamic Distributed Systems

- Heterogeneous environments
- Large number of bursty clients
- Stringent QoS requirements, e.g.:
  - 24x7 availability
  - Low latency & high throughput
- Examples
  - Online trading systems
  - Mission-critical systems for critical infrastructure
    - e.g., air transportation, power grid control

Our R&D goal is to assure the adaptability of these types of distributed systems
Component middleware capabilities:

- Creates a standard “virtual boundary” around application **component** implementations that interact only via well-defined interfaces.

- Define standard **container** mechanisms needed to execute components in generic component servers.

- Specify a reusable/ **standard infrastructure** needed to configure & deploy components throughout a distributed system.
Overview of Lightweight CCM

• Many DRE CORBA applications can’t use “enterprise” CCM due to constraints like limited processing overhead for performance-intensive applications.

• CCM features supported by Lightweight CCM are:
  • All types of ports, i.e., Facets, Receptacles, Event sources and sinks and attributes.
  • Component homes.
  • Monolithic implementations.
  • Session components and containers.
General Approach: Adaptability via Swapping

• Goal: Improve the adaptability of component-based distributed systems.

• Requirement: Adaptive and reflective environment where distributed systems can adapt to changing operating conditions.

• Approach: Providing such an environment to dynamically swap component implementations as and when the dynamic operating conditions change.
SwapCIAO Architecture (1/4)

Provides an execution environment where component implementations can be instantiated, removed and swapped.
SwapCIAO Architecture (2/4)

Dynamically opens and loads component implementations during the component swapping process.
Stores component implementations which are retrieved by the updatable component factory during swapping time.
Automatically generates glue code for swapping component implementations
Salient Features of SwapCIAO Architecture

• Updatable Container
  • Extends Lightweight CCM session container interfaces to support additional mechanisms for component creation, activation and swapping.
  • Automatically handles challenges involved in swapping components.

• Updatable Component Factory
  • Extends CCMHome by providing mechanisms for creating component implementations on demand.
  • Can be configured with sophisticated component implementation selection algorithms which can help selecting the right implementation to swap during the component swapping process.
  • Provides a portable interface for opening DLLs on heterogenous run-time platforms.
Providing Consistent and Uninterrupted Swapping (1/2)

- Clients can initiate new invocations when dynamic swapping takes place
  - These invocations should be blocked and allowed into the system after the swapping finishes.
- Ongoing invocations could be processed when dynamic swapping happens
  - These invocations must be allowed to complete before swapping starts.
Providing Consistent and Uninterrupted Swapping (2/2)

- Ongoing invocations need to complete processing.
  - Maintain a dispatch table at the POA and track how many requests processed in each thread.
  - Use standard reference counting and deactivation mechanisms to delete a component only when the dispatch table count is zero.

- Block new invocations from entering the system.
  - Updatable container instructs the ORB to block invocations using standard CORBA Portable Interceptors.
Providing Client Transparent Swapping (1/2)

- Clients hold an object reference to the component to make an invocation.
- During swapping, the old component is removed and the new component implementation is loaded.
- Old component reference held by client no longer valid.
- Dynamic swapping needs to be transparent.
  - Clients should not update their references and still not receive invalid reference exceptions
Providing Client Transparent Swapping (2/2)

• POA intercepts client requests after the component is removed.
• Active Object Map will be empty if component is removed already.
• Associate the POA with servant activator.
• Register the updatable component factory with the servant activator.
• If no component available, the servant activator automatically calls the updatable component factory to create new implementation.
• Store the new implementation in the POA’s Active Object Map.
  • Activate component servants with unique user id, so that clients need not be updated with the new reference, preventing roundtrip delays informing the client about the new component.
Conclusion

• SwapCIAO is a component middleware framework based on Lightweight CCM that supports dynamic component updating.

• SwapCIAO is designed to handle dynamic operating conditions by swapping component implementations that are optimized for particular run-time characteristics.

• Standard Lightweight CCM interfaces can be extended to develop a scalable and flexible interface for supporting dynamic component updating.

• Hence client programming model and the client/server interoperability were unaffected by the new interfaces added.

• Developers of client/server applications need not do anything special to get the benefits of swapping component implementations.