Recommendations for a CORBA Language Mapping for RTSJ

Victor Giddings
Objective Interface Systems
victor.giddings@ois.com

Outline

- Real-time Specification for Java
  - Background
  - Memory Management
  - Thread Types
  - Thread Priorities
- IDL to RTSJ Mapping Issues
  - Stubs and Skeletons
  - Thread Type Matching
  - Priority Mapping

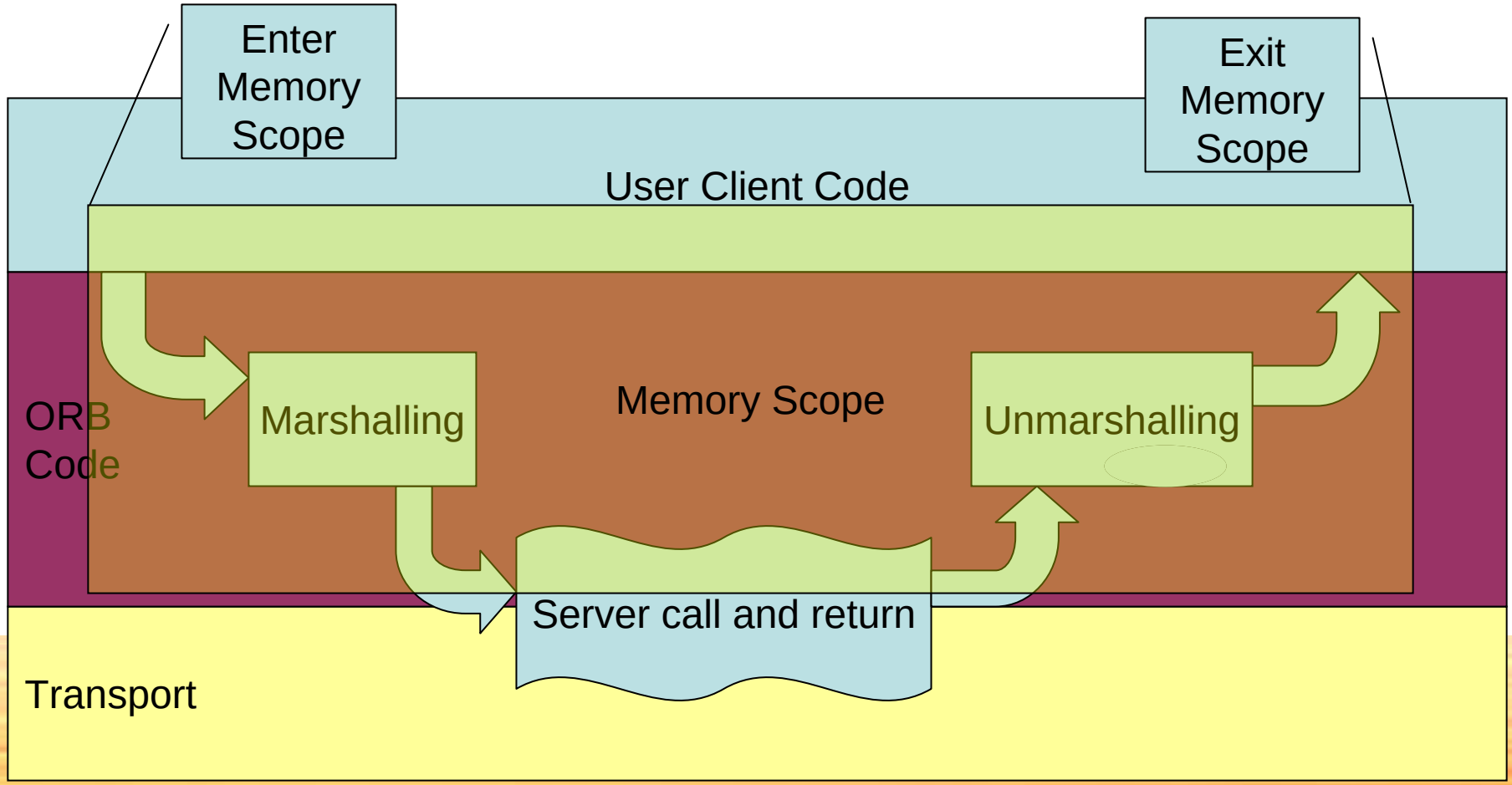
- Result of first Java Community Process effort (JSR-001)
  - Process authorized – December 1998
  - First specification approved – May 2000
  - Current version (1.0.1b) – March 2005
- Purpose – “enable the creation, verification, analysis, execution, and management of Java threads whose correctness conditions include timeliness constraints (also known as real-time threads).”
- Seven enhanced areas
  - Thread Scheduling and Dispatching
  - Memory Management
  - Synchronization and Resource Sharing
  - Asynchronous Event Handling
  - Asynchronous Transfer of Control
  - Asynchronous Thread Termination
  - Physical Memory Access

- Some of the features
  - New types of threads
    - Real-time threads
    - No Heap Real-time threads
  - Memory scopes
    - Predictable time of finalization
  - Scheduling includes priority inheritance
- Effect - *Not your father's Java*

RTSJ Developers must worry about memory management

RTSJ Memory Management

- Memory areas
  - Identifiable – presented as objects
  - Different types
    - Heap – subject to garbage collection
    - Immortal memory – not subject to deallocation
    - Scoped memory – reclaimed explicitly
    - Physical memory
- When memory scope is *left*
  - Object finalization will be performed
  - Object memory will be reclaimed
- ORB implementation challenge (common to all library-based products)
  - Allow users to control memory scope usage
  - While retaining state needed for correct operation


Enter Memory Scope
Exit Memory Scope
User Client Code
ORB Code
Marshalling
Memory Scope
Unmarshalling
Server call and return
Transport

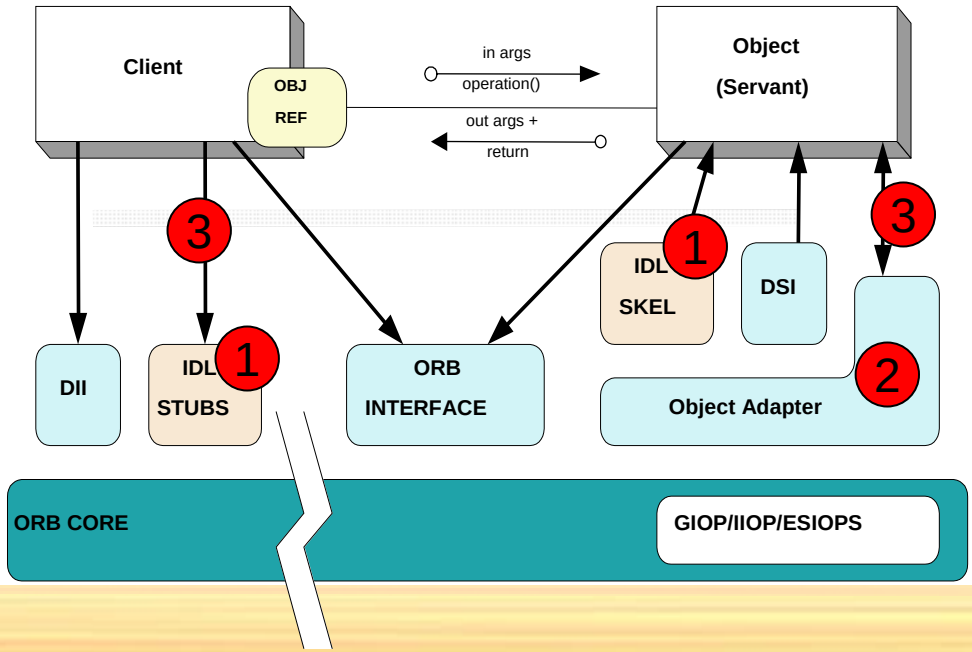
RTSJ Thread Types

- "Regular" Java threads
  - Allow
    - Concurrent execution
    - Blocking and synchronized methods
  - "Relaxed" priority preemptive scheduling
- RealtimeThread
  - Additional scheduling parameters
  - May be assigned a memory scope
  - May access heap, subject to blocking by garbage collector
  - Stricter scheduling
- NoHeapRealtimeThread (NHRT)
  - No access to heap
  - Must be assigned a memory scope
  - Blocks garbage collector

- Priority pre-emptive scheduling required
  - At least 28 unique priorities required
  - RealtimeThreads and NoHeapRealtimeThreads run at higher priority than “regular” threads
  - NoHeapRealtimeThreads not subject to blocking by garbage collector
- Implementation of *synchronized* keyword
  - By default, unbounded priority inversion must be avoided
  - Priority ceiling emulation must be provided if supported by underlying system

Mapping Considerations

- Implementation experience points to 3 areas of IDL to Java mapping
  1. Stubs and skeletons – portable stubs and skeletons not adequate for RTSJ
  2. Thread type matching – should requests from NoHeapRealtimeThreads be executed on a NoHeapRealtimeThread on the server?
  3. Priority Mapping – native Operating System priority not visible, not useful



Stubs and Skeletons

- Current IDL to Java mapping requires allocation of certain parameters in the stub and skeleton
  - Arrays
  - Sequences
  - Strings
- Stubs and skeletons need to be changed
  - To use factories and memory managers for these types
  - Allow re-use or hoarding of previously allocated memory
- Skeletons may need to allow user to specify memory scope to be used for allocation of parameter memory

Thread Type Matching

- Argument
  - Client request
    - Can come from one of three thread types – Thread, RealtimeThread, NoHeapRealtimeThread
    - Subject to different preemption according to type
  - Principle of “location transparency”
    - Server-side processing should be subject to same preemption
    - Especially for “local” or co-located call
  - Thread type must be carried in request
- Counter argument
  - Not language independent – what if server is C++, etc.?

Priority Mapping

- RTCORBA provides for mapping between
  - “Global” RTCORBA priority
  - Native Operating System priority
- RTSJ provides uniform portable language-specific view of Operating System priority
- More useful to map RTCORBA priority to Java priority

- RTSJ programming is different than Java programming
- Needs identified for IDL to Java mapping changes for RTSJ
  - Stubs and skeletons
  - Thread type matching
  - Priority mapping
- Draft RFP under consideration for these changes