

Design and Performance of an Asynchronous Method handling Mechanism for CORBA

Mayur Deshpande, Douglas C. Schmidt &
Carlos O'Ryan

`{deshpanm, schmidt, coryan}@uci.edu`

Department of Electrical & Computer Engineering
University of California, Irvine

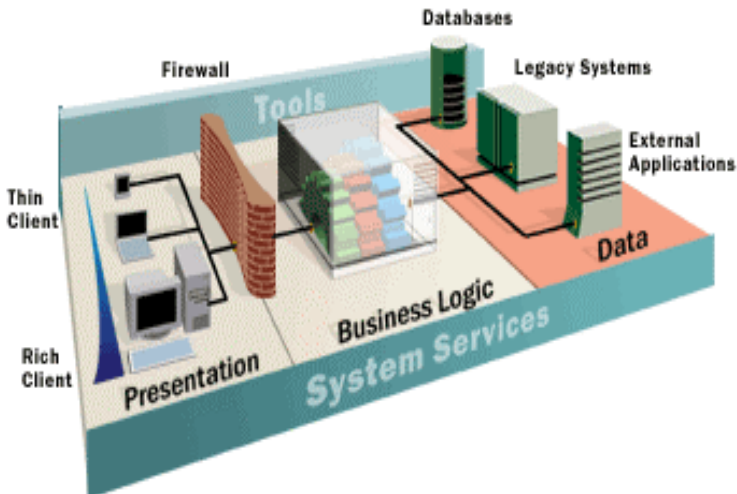
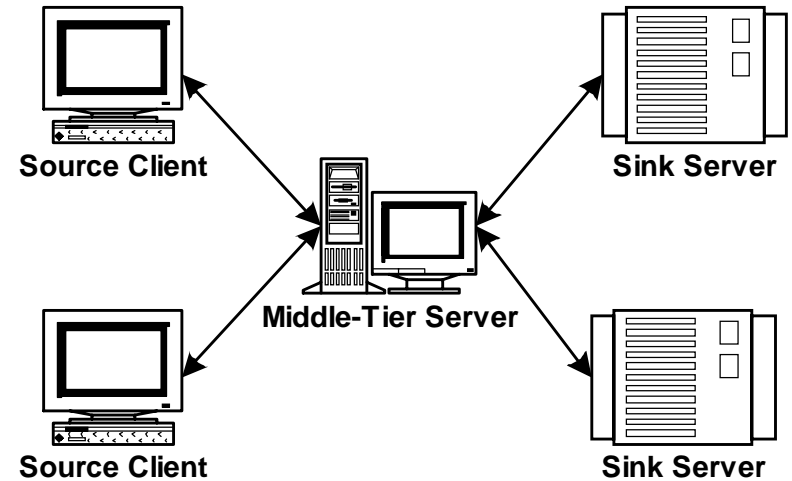


Monday, 09 September 2002

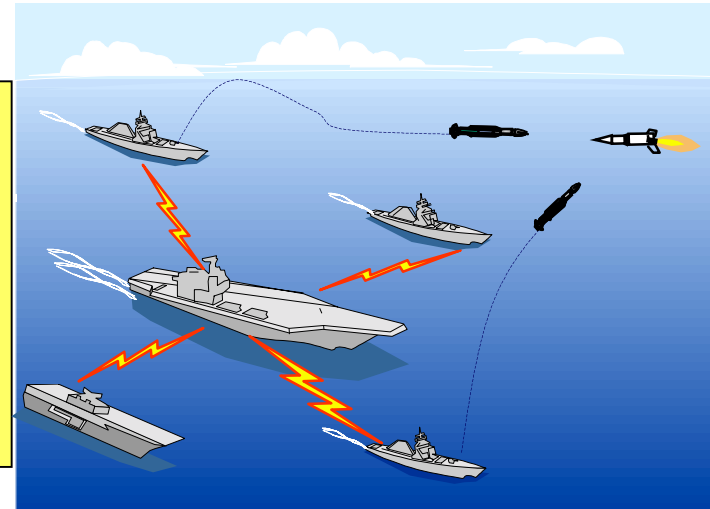
Research Sponsored by AFOSR, ATD,
Cisco, DARPA, Raytheon, SAIC, & Siemens

Motivation: Middle-Tier Servers

- In a multi-tier system, one or more “middle-tier” servers are placed between a *source client* & a *sink server*
 - A source client’s two-way request may visit multiple middle-tier servers before it reaches its sink server
 - The result then flows in reverse through these intermediary servers before arriving back at the source client

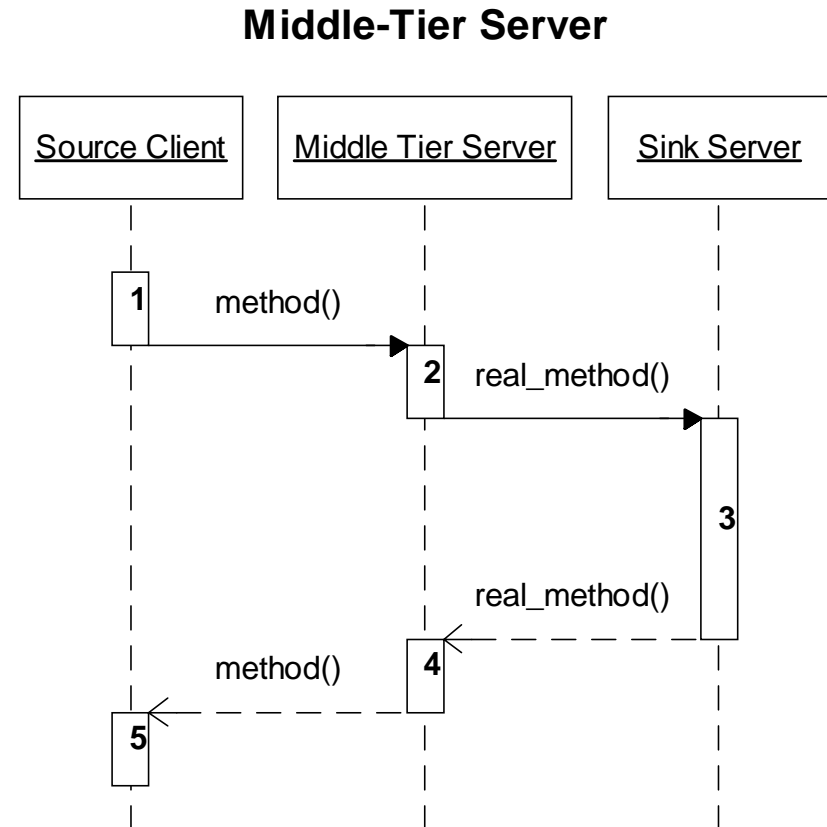


Middle-tier servers are common in both business & real-time/ embedded systems



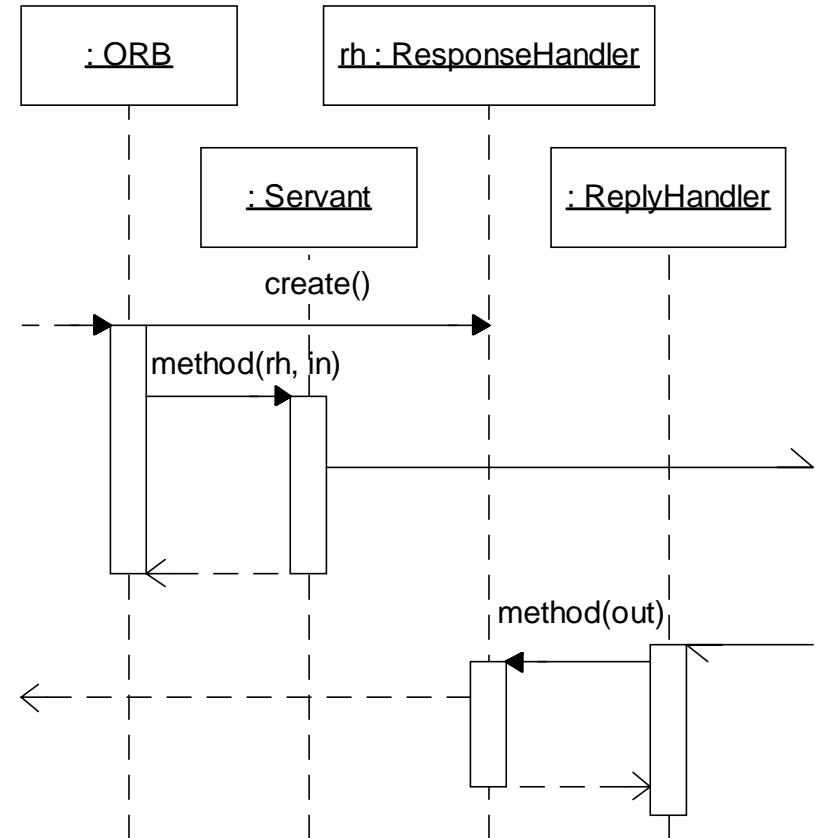
CORBA Limitations for Middle-Tier Servers

- Hard to implement scalable & convenient middle-tier servers using standard CORBA
- **Tight coupling** between receiving request & returning response *in the same activation record*:
 - Restricts request/ response pair to a single thread in standard CORBA
- Not scalable to dedicate a separate thread for each outstanding client request:
 - **thread creation**
 - **context switching**
 - **synchronization**
 - **data movement overhead**



Solution: Asynchronous Method Handling (AMH)

- AMH decouples the existing CORBA 1-to-1 association between
 1. An incoming request to the run-time stack and
 2. The activation record that received the request
- This design allows a server to return responses asynchronously, *without* incurring the overhead of multi-threading
- AMH is inspired by
 1. The CORBA asynchronous method invocation (AMI) model
 2. Continuations



AMH Implementation : Request

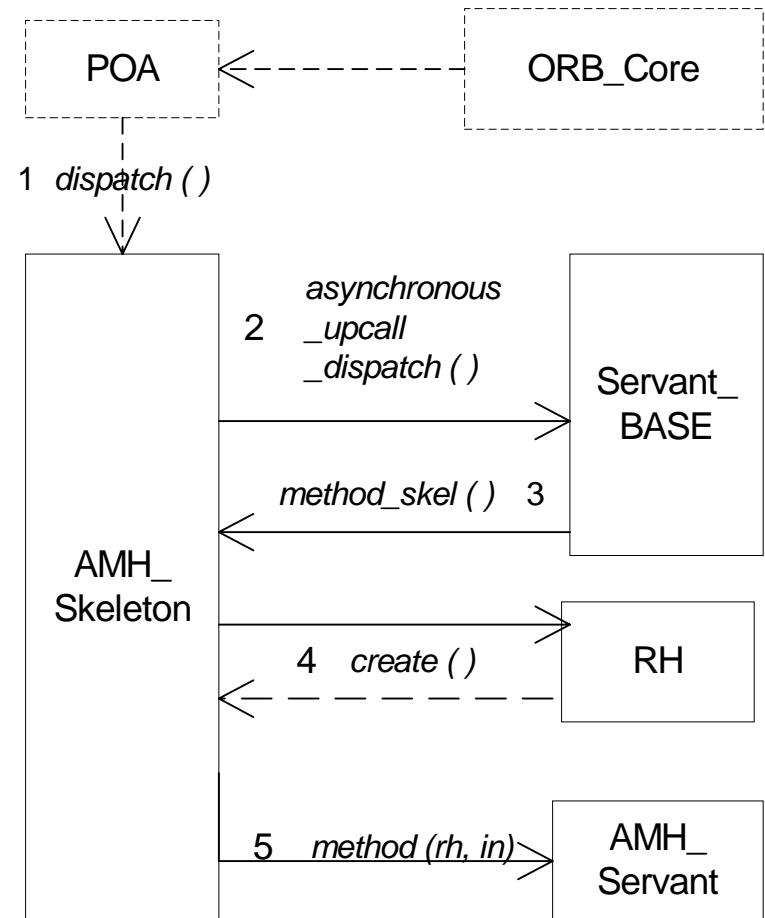
- *Asynchronous_upcall_dispatch* (2) does not implement functionality to return values to client

- This functionality is now present in the RH

- POA interface to dispatch (AMH) skeleton (1) unchanged:

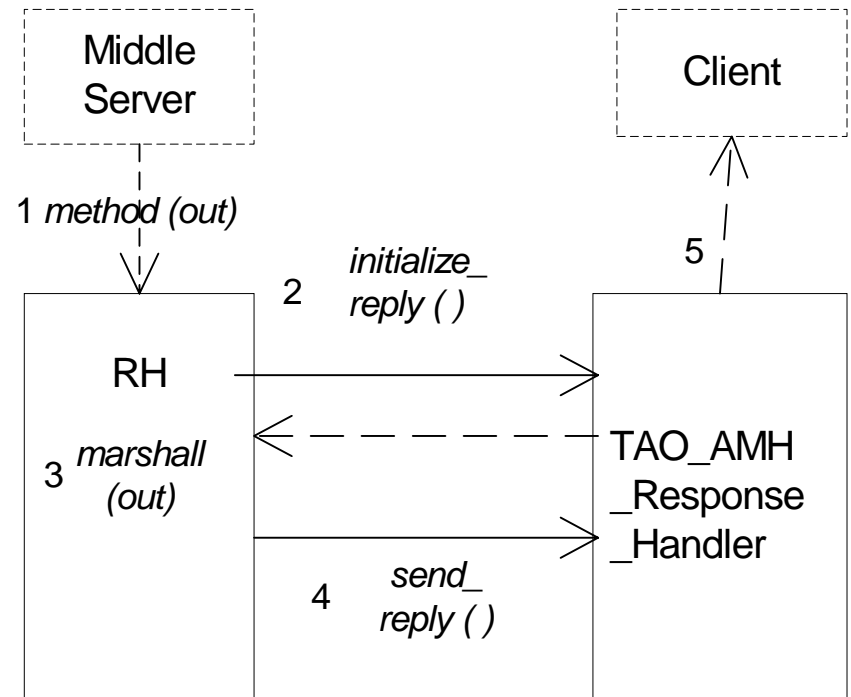
- AMH Skeletons can be registered in any POA

- No new POA policies



AMH Implementation : Reply

- Base ResponseHandler is responsible for all interactions with the ORB (2,5)
- Derived RH calls appropriate methods in base RH to initialize/send reply (2,4)
- Derived RH only implements marshalling of return values (3)



Design Challenge: Client Transparency

Challenge: AMH is a server-side mechanism. Hence it should be completely transparent to clients

Solution: AMH Skeleton registers an AMH servant as an implementer of the original (non-AMH) interface:

- *_this()* returns object reference of original interface
- Logic to demarshal operation parameters marshaled with stub of original interface generated by IDL compiler into AMH skeleton
- *is_a()* in AMH skeleton returns true against test for original interface

Design Challenge: Memory Footprint

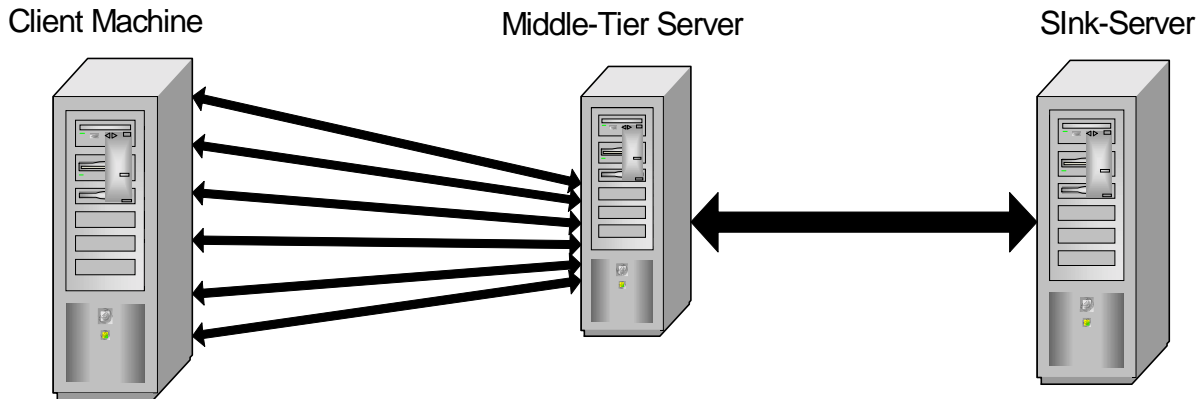
Challenge: Clients resident in embedded systems that need to contact an AMH server should not pay any memory overhead

Solution: Three pronged strategy:

1. All changes to ORB required for AMH confined to server-specific library that pure clients do not link
2. Restrict IDL generated code for AMH mechanism to AMH skeleton files
3. If IDL generated code needed in stub files, make classes in stub files abstract

Benchmark: Testbed

10 Mbps
LAN
through-
out



Spawns specified
concurrent clients

Forwards requests
to sink-server /
replies to client

Replies delayed by
specified amount
of time

•Intel XEON Dual @
1700Mhz, 1GB RAM,
RH 7.2 (2.4.7)

Intel P-III @733Mhz,
512 KB RAM,
Debian (2.2.18)

•Intel P-III Dual @
733Mhz, 512 KB RAM,
Debian (2.4.16)

Benchmark: Test Description

Middle Server runs various concurrency models:

- **Thread Per Connection**

(TPC): One thread created for every new client connection

- **Single Threaded Reactive**

(TPR-ST): One thread dispatched by ‘Reactive’ ORB: no threading locks

- **Thread Pool Reactive (TPR-2)**

Same as TPR-ST but two threads share the work

- **Single Threaded AMH**

(AMH-ST):

- Each request stored in Hash table (RH, Req-ID)

- Upon receipt of reply, RH extracted from Hash table and invoked

- **Multi Threaded AMH (AMH-MT)**

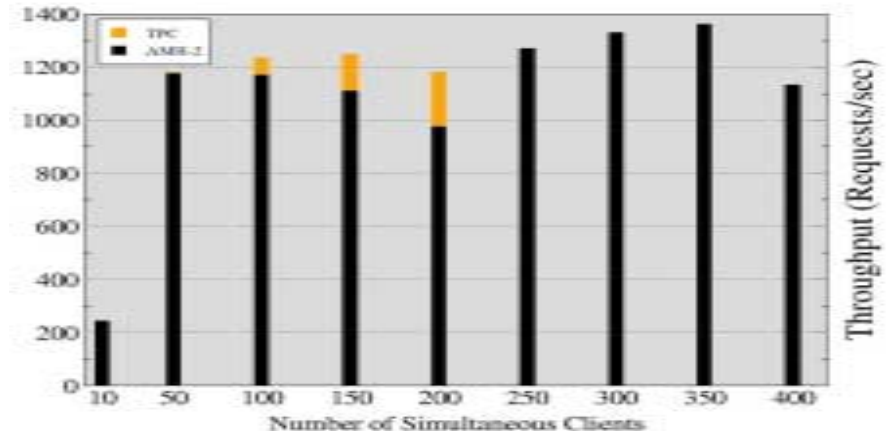
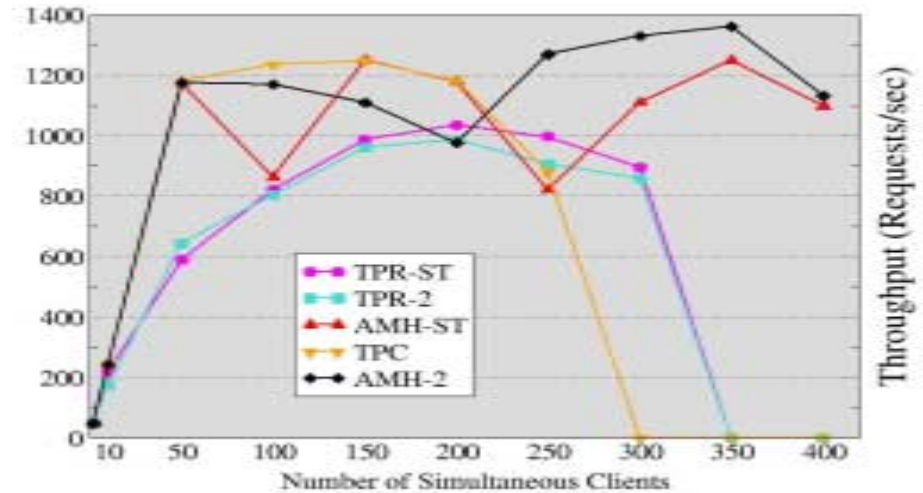
Same as AMH-ST but two threads share the work

Benchmark: Client Scalability

• **Test Description:** Throughput (requests/sec) of middle-server measured with increasing number of concurrent clients

• **Result Synopsis:**

- Under heavy load, AMH servers deliver highest throughput
- For light/medium load, AMH servers are slightly below par compared to other models
- **For some cases (more than 300 clients), AMH is only viable option!**



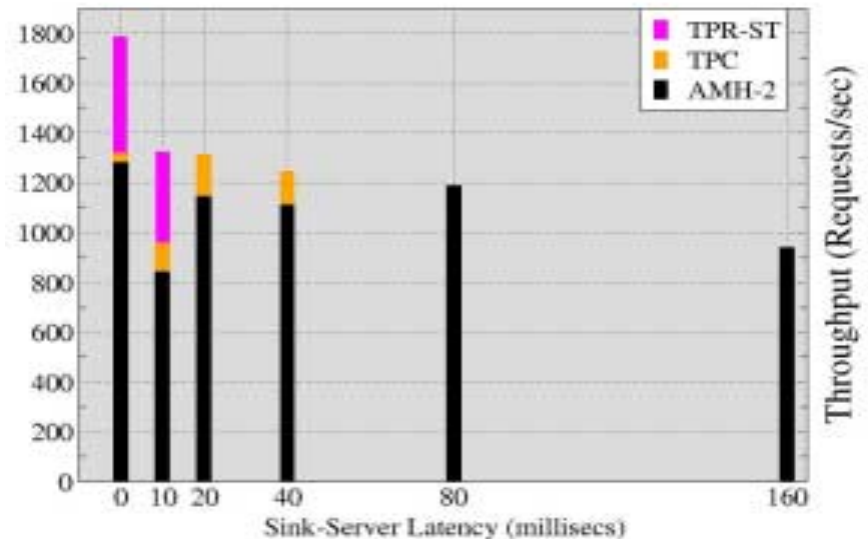
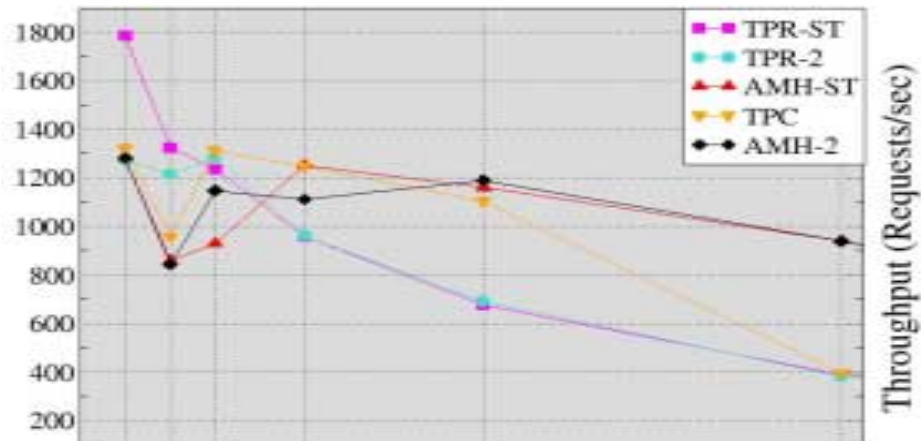
Benchmark: Latency Scalability

• **Test Description:** Throughput (requests/sec) of middle-server measured with increasing delay (in milliseconds) from sink-server

• **Result Synopsis:**

- Long latency requests: AMH servers deliver highest throughput
- Short latency: TPR-ST is best.
- Medium latency: TPC is best.

If server model can be dynamically switched, throughput can be made constant across all loads/latencies!



Future Directions

Research:

- Benchmark AMH for Real-time systems:
- Creating ResponseHandler on the heap may lead to jitter, heap-fragmentation and possible priority inversions
- Integrate AMH into common middle-tier services such as Load Balancing Service and Event Service

Enhancements:

- Support for operation-level control of AMH as compared to current interface-level control
- Control over creation policy of ResponseHandlers. Currently, one ResponseHandler is created for each request

Concluding Remarks

- Middle-tier servers need a scalable asynchronous programming model
 - **The current AMI models don’t suffice for middle-tier servers**
- For long latency requests or heavily loaded middle-servers, AMH offers a more scalable solution
 - **AMH offers tangible benefits for middle-tier servers**
- Our asynchronous method handling (AMH) model supports efficient server-side asynchrony with relatively few changes to CORBA
 - **AMH is similar to standard CORBA AMI, focusing on the server rather than the client**
- A paper on AMH is also available
 - **www.cs.wustl.edu/~schmidt/PDF/AMH_DOA-02.pdf**

