Service Configurator

A Pattern for Dynamically Configuring Network Services

Prashant Jain and Douglas C. Schmidt
pjain@cs.wustl.edu and schmidt@cs.wustl.edu

Washington University, St. Louis

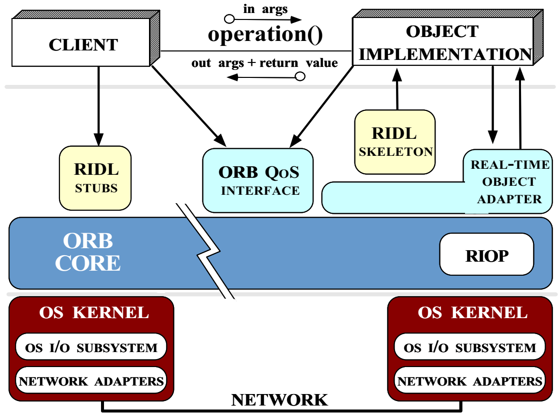
June 19, 1997

This work is sponsored by Siemens MED

Motivation

- Common server activities include:
  - *Service (re)configuration and run-time control*
  - *Daemonization and comm. endpoint initialization*
  - *I/O port demultiplexing and dispatching*
  - *Process and thread creation*
- Conventional server designs are overly *static*, *i.e.*:
  - Must modify, recompile, and relink existing code
  - Must terminate and restart running processes
- The **Service Configurator pattern** increases server extensibility by *dynamic configuring* network services

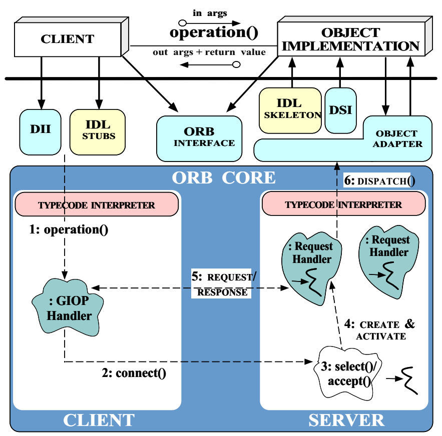
Example: The ACE ORB (TAO)



http://www.cs.wustl.edu/~schmidt/TAO.html

- **TAO Overview**
  - High-performance, real-time ORB
    - ∗ Telecom and avionics focus
  - Leverages the ACE framework
    - ∗ Runs on VxWorks, POSIX, and Win32
  - Small memory footprint
  - Highly configurable

Original SunSoft IIOP Reference Implementation



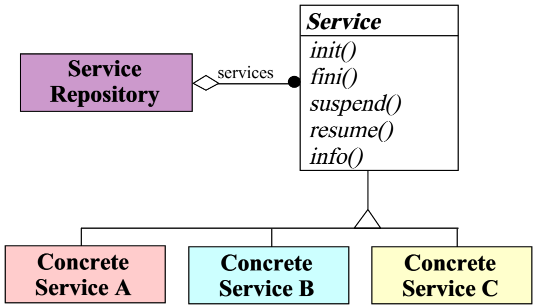
www.cs.wustl.edu/~schmidt/IIOP.ps.gz

- **Limitations with SunSoft IIOP**
  - Not a complete ORB
  - Inefficient TypeCode interpreter
  - "One-size fits all" design
  - Functionality was entirely *static*
    - ∗ *i.e.*, all enhancements require changing the ORB source code

Key Strategies and Patterns in TAO

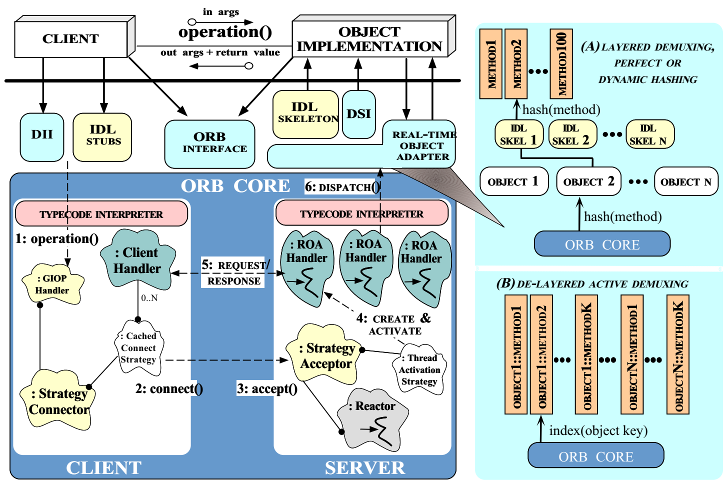
- **Key ORB Strategies**
  - Concurrency strategy → *e.g., Thread-per-Request, Thread-per-Connection*
  - Demultiplexing strategy → *e.g., Dynamic Hashing, Perfect hashing, Active Demultiplexing*
  - Dispatching strategy → *e.g., Rate Monotonic, Earliest Deadline First*
- **Key ORB Patterns**
  - Service Configurator
  - Strategy
  - Abstract Factory
  - Reactor
  - Active Object

Structure of the Service Configurator Pattern



- **Participants**
  - *Service* → specifies abstract *hook method* API
  - *Concrete Service* → implements hook methods
  - *Service Repository* → controls groups of services

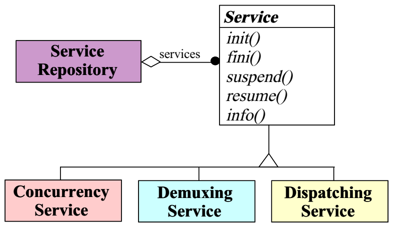
Increasing ORB Flexibility with Patterns and Frameworks



Overview of the Service Configurator Pattern

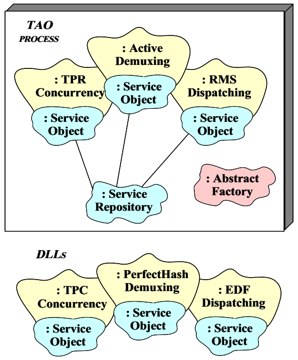
- **Intent**
  - *Decouples the behavior of services from the point in time at which service implementations are configured into an application or system.*
- **Forces resolved**
  - How to defer the selection of a particular type, or a particular implementation, of a service until very late in the design cycle
  - How to build complete applications by composing multiple independently developed services
  - How to optimize, reconfigure, and control the behavior of the service at run-time

Using the Service Configurator Pattern in TAO



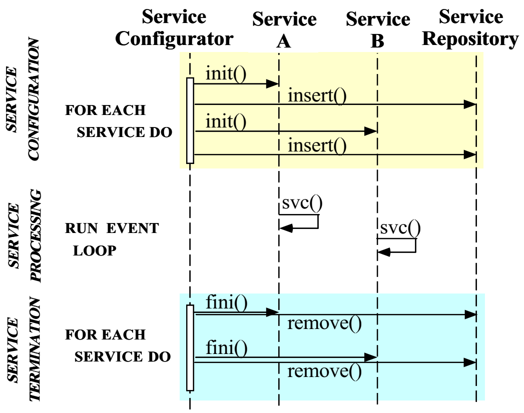

```
int main (int argc, char *argv[])
{
  // Configure the ORB.
  Service_Config tao (argc, argv);

  // Perform ORB services updates.
  tao.impl_is_ready ();
}
```



Run-time Configuration

Participants in the Service Configurator Pattern



- **Interaction Steps**
  - *Service configuration*
  - *Service processing*
  - *Service termination*

Concluding Remarks

- **Benefits of patterns, in general**
  - Facilitate design reuse
  - Preserve crucial design information
  - Guide design choices
  - Document common traps and pitfalls
- **Benefits of Service Configurator pattern**
  - Increases flexibility and extensibility of networking apps.
  - Centralizes administration and control
- **URLs**
  - http://www.cs.wustl.edu/~schmidt/patterns-ace.html
  - http://www.cs.wustl.edu/~schmidt/TAO.html