

Transparent Resource Management for QoS-Enabled DRE Systems

Ossama Othman <ossama@uci.edu>

University of California, Irvine

RTWS 2002

DRE Systems

- Have difficulties similar to other types of distributed systems
 - Heterogeneous environments
 - Concurrency
 - Etc.
- Found in many types of applications
 - E-commerce
 - Mission critical systems
 - Defense
 - Ship systems
- Often have more stringent QoS requirements
 - “QoS-enabled”

Motivation

- Development and maintenance of QoS-enabled DRE systems
 - Non-trivial
 - Requires expertise that DRE system developers often lack
- Solution: **Middleware** (*e.g.* CORBA)
 - Can shield DRE system developers from the complexities involved with developing distributed applications
 - Can facilitate manipulation of QoS requirements and management of resources

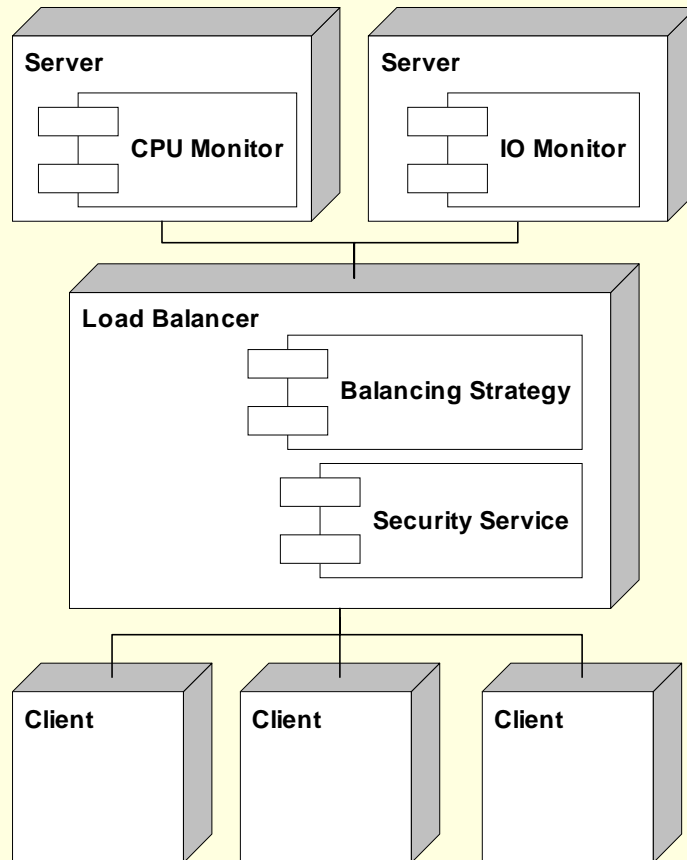
DRE System Resource Management

- Middleware alleviate resource management difficulties
- Doing so in a transparent and efficient manner is difficult
- Managing resources transparently is important for legacy DRE systems
 - Often cannot be easily modified to introduce support for improved distributed resource management
 - May not be feasible to do so
- Middleware in use may need to be enhanced to support this functionality
 - Such an enhancement can be found in a CORBA-based load balancing service
 - Currently being standardized by the Object Management Group

Middleware-Based Load Balancing

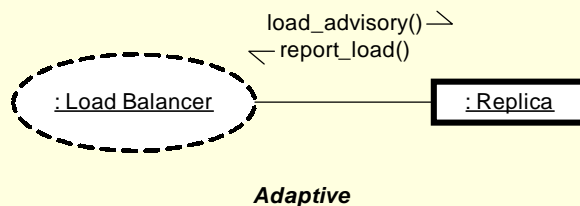
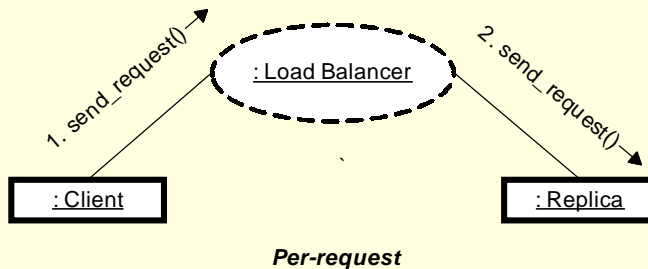
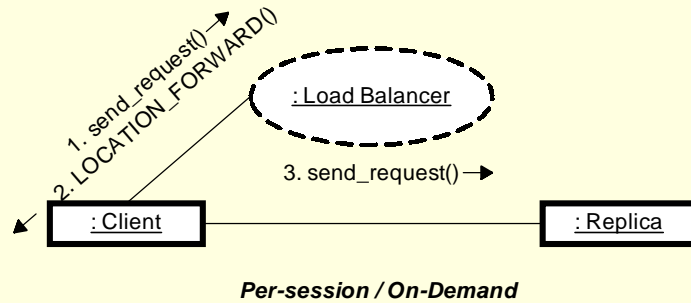
- Load balancing service can be used to manage resources for middleware-based DRE systems
 - Can improve the efficiency and overall scalability of a DRE system
 - Allows additional components to be added to DRE systems with minimal impact to performance
 - Improves availability due to inherent redundancy

Load Balancing Model



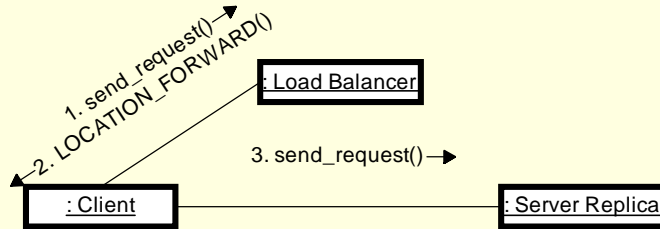
- Portable and interoperable
 - Can be administered via CORBA requests
- Easily composed with other services (e.g. fault tolerance, security, etc)
- Can take into account request content
- Customizable load metrics
 - Load metric neutral
- Extensible load balancing strategies
- Transparent to clients

Load Balancing Strategies

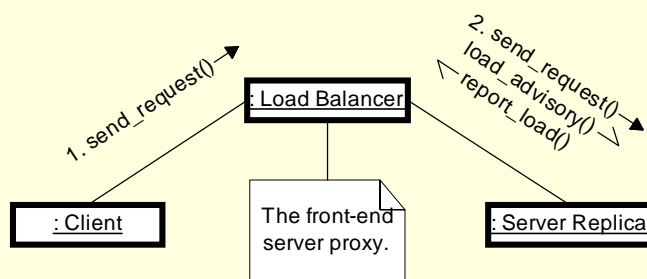


- Client binding granularity
 - Per-session
 - Client permanently forwarded to a replica
 - Per-request
 - Requests forwarded on client's behalf
 - On-demand
 - Client can be rebound to another replica whenever necessary
- Balancing policy
 - Non-adaptive
 - No load feedback used when binding clients
 - Adaptive
 - Load feedback taken in to account

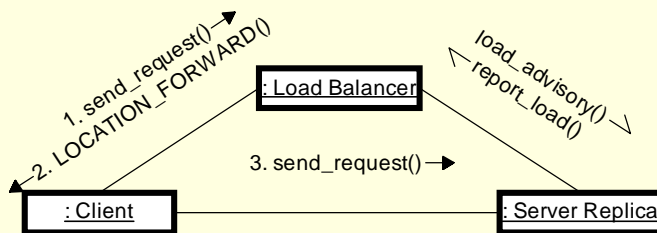
Load Balancing Architectures



Non-adaptive Per-Session



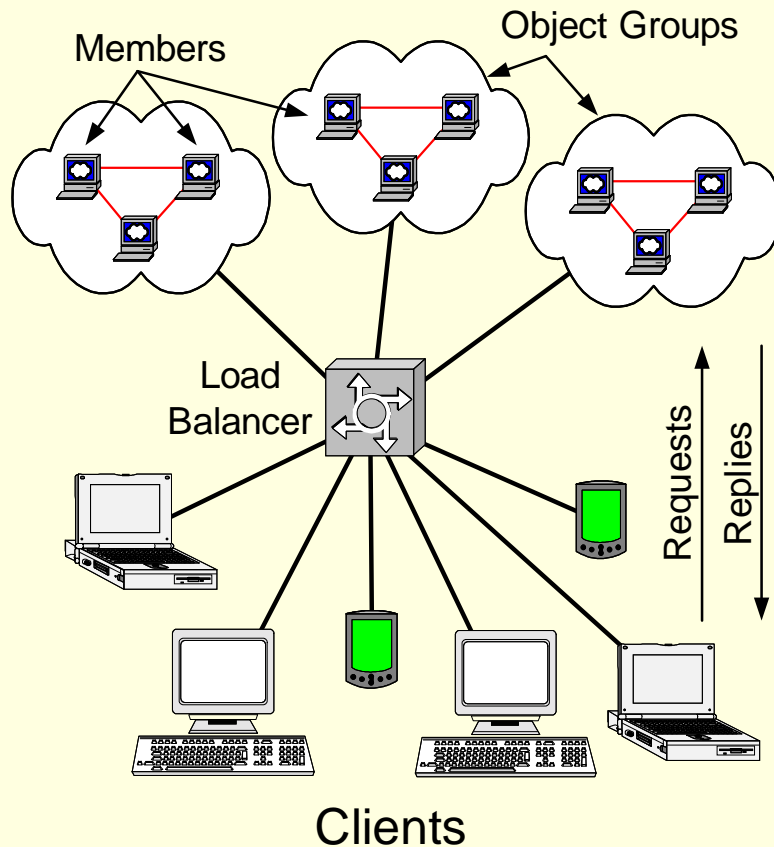
Adaptive Per-request



Adaptive On-demand

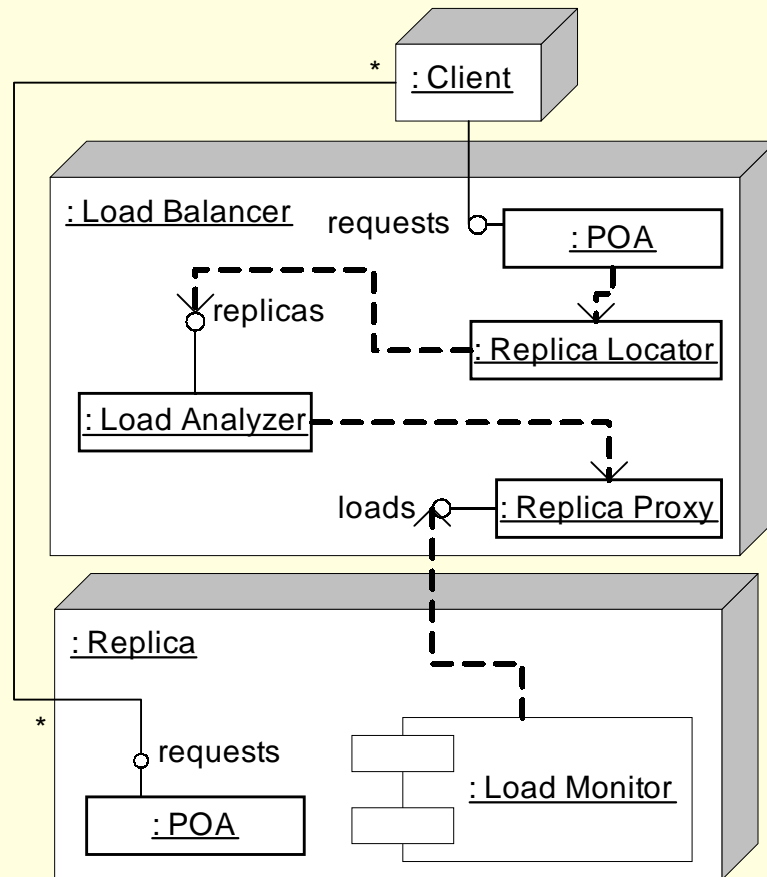
- Load balancing architecture comprised of a combination of *client binding granularity* and *balancing policy*
- Given the strategies just described, there are six possible architectures
- Three common architectures
 - Non-adaptive per-session
 - Adaptive per-request
 - Adaptive on-demand

Basic Scenario



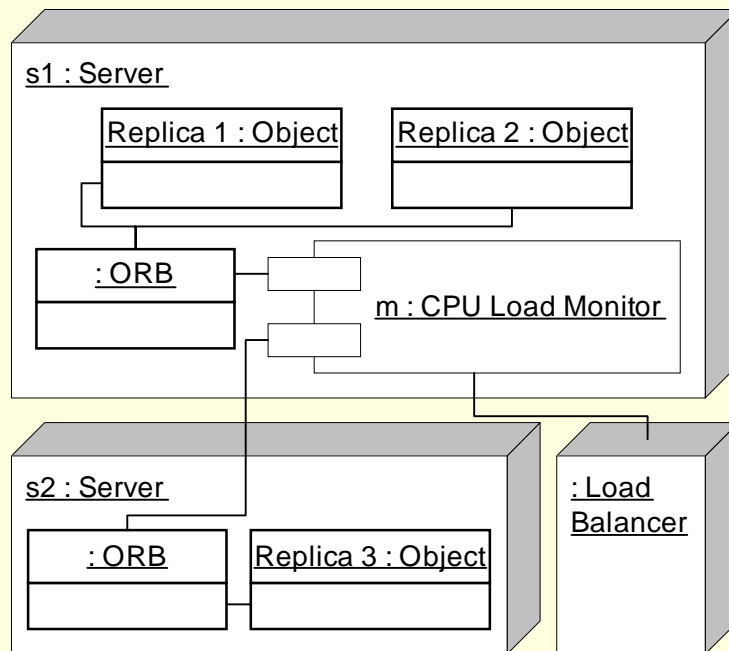
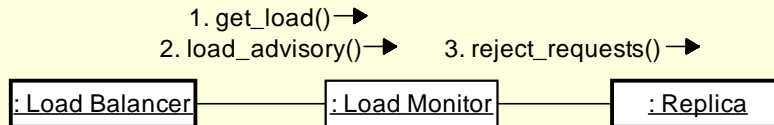
- Multiple clients making request invocations
 - Potentially non-deterministic
- Members
 - Multiple instances of the same object implementation
- Object groups
 - Collections of members among which loads will be distributed equitably
 - Logically a single object
- Load balancer
 - Transparently distributes requests to members within an object group

Load Balancer Components



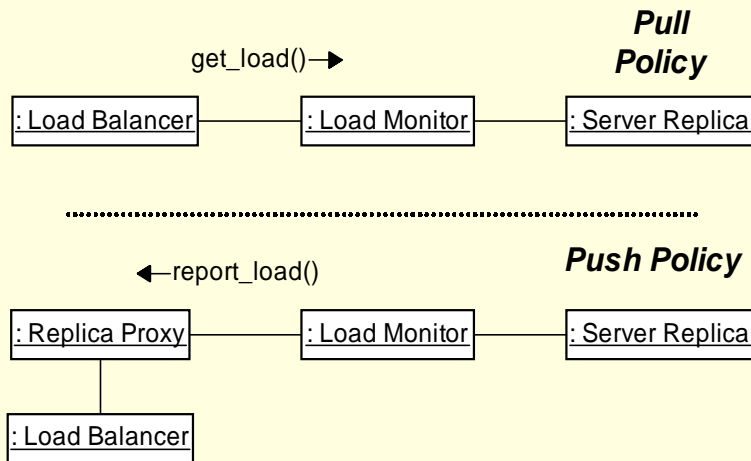
- Load Monitor
 - Facilitates load feedback and control
- Load Analyzer
 - Determines member load conditions
- Replica (Member) Locator
 - Binds client to appropriate member
- Replica (Member) Proxy
 - Intermediate object
 - Uniquely identify replicas in “push” load reporting model

Components – Load Monitor



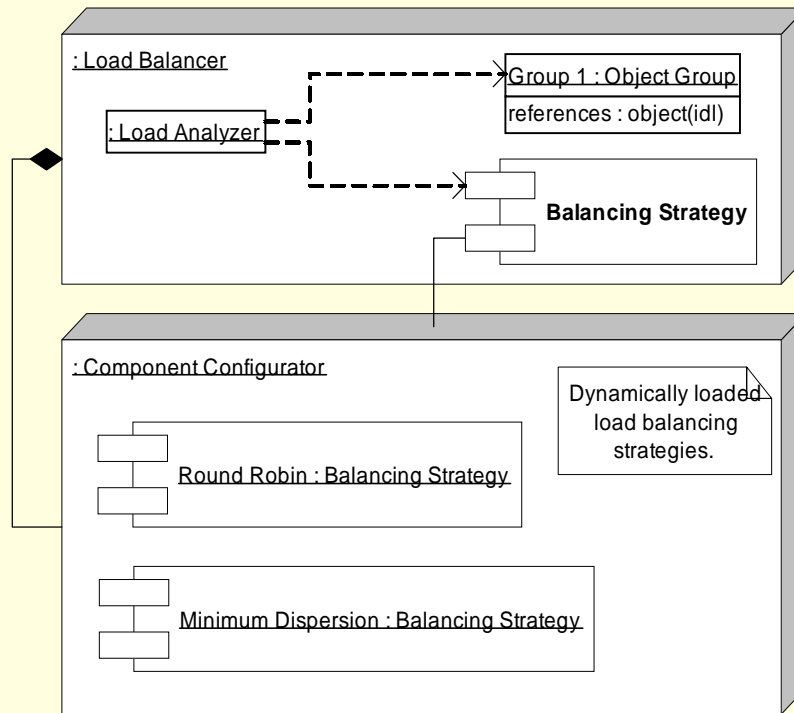
- Facilitates feedback and control
 - Monitor and report loads
 - Respond to load advisories sent by load balancer
 - Make replicas redirect or accept requests
- Can be embedded into replicas
 - Employ the Component Configurator and Interceptor design patterns to embed transparently
- Load monitor can be an interceptor (e.g., a CORBA request interceptor)
- Load monitors may also be shared by multiple replicas
 - Multiple replicas at a given location

Components – Load Monitor (cont'd)



- Load monitor can be configured with either of two policies
 - Pull policy
 - Load balancer can query, i.e. “pull,” loads on-demand
 - Push Policy
 - Load monitor can “push” loads to the load balancer

Components – Load Analyzer



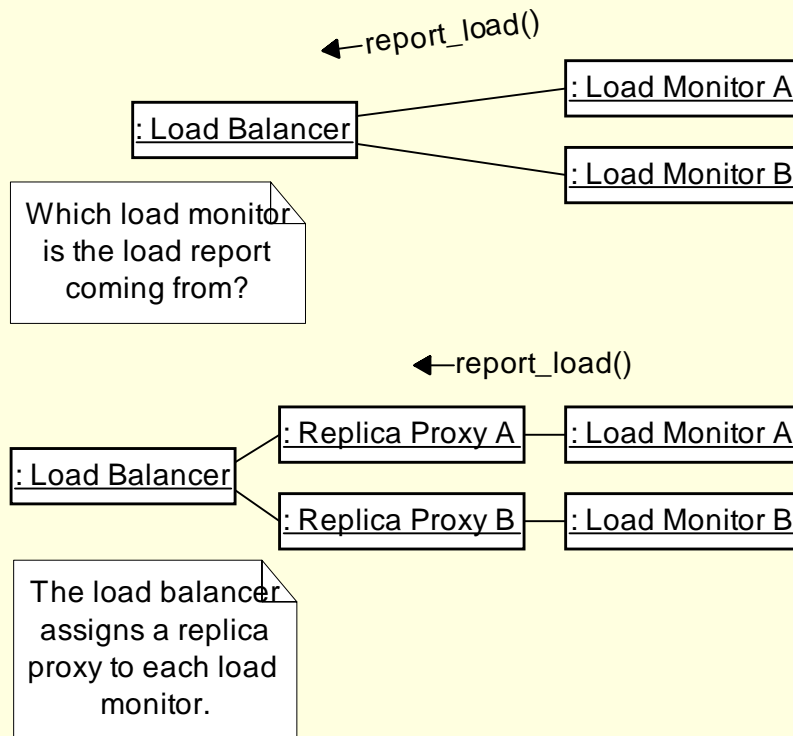
■ Load Analyzer

- Decides which replica will receive the next client request
- Determines load condition on each replica
 - Low, nominal, high, etc.
- Extensible load balancing strategies
 - Employ Component Configurator design pattern
 - Each replica group may use a different strategy

Components – Replica Locator

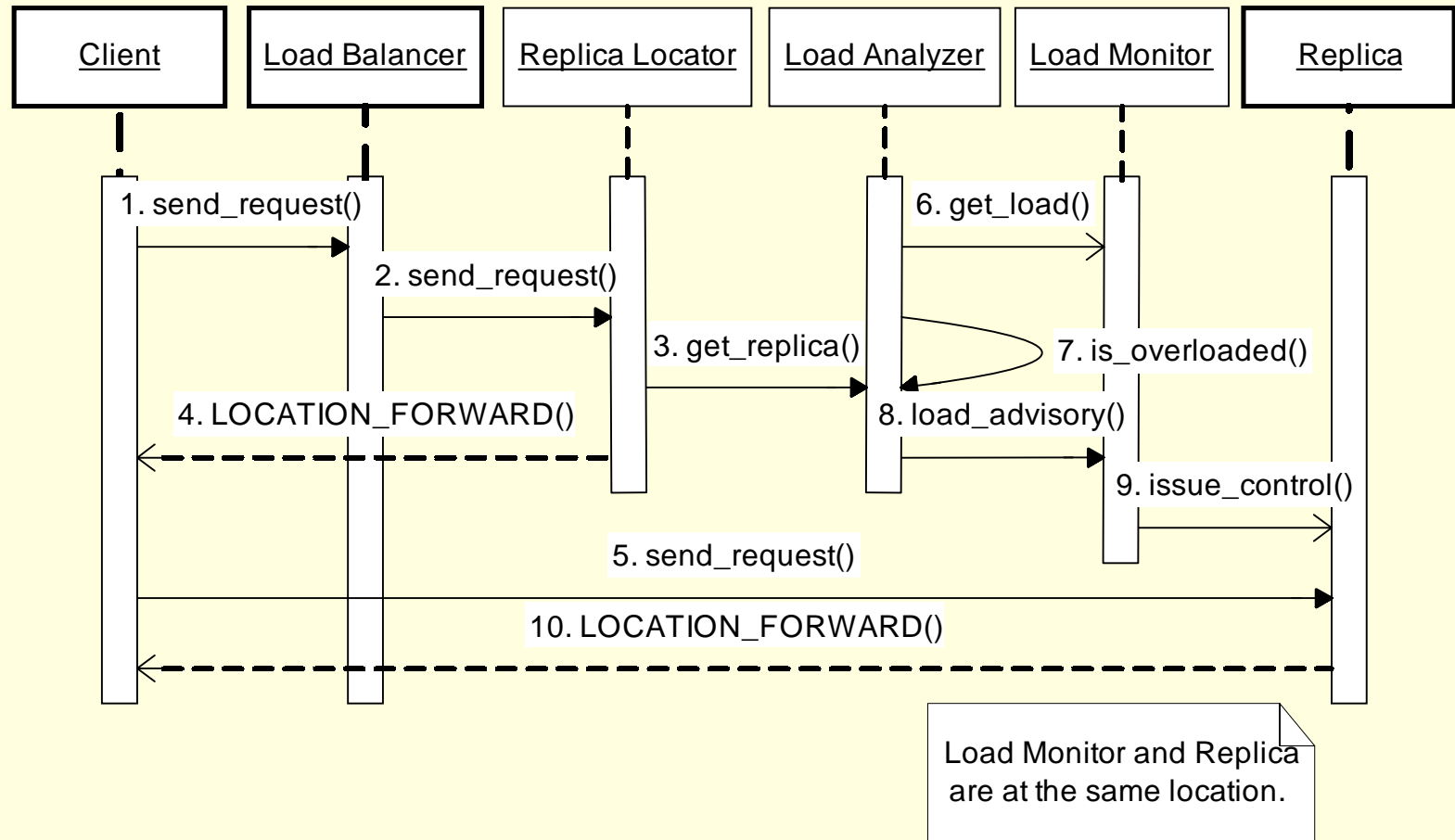
- Implements the Interceptor design pattern
- Typically implemented as a Servant Locator
- Forwards client requests to replica retrieved from the load analyzer
 - Redirection induced via standard GIOP `LOCATION_FORWARD` message
 - Conforming client side ORB will transparently re-issue request to replica chosen by load balancer

Components – Replica Proxy

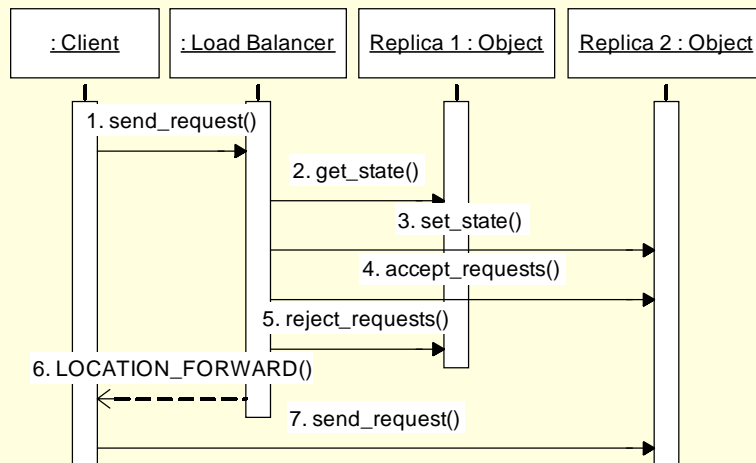


- Used to uniquely identify replicas and/or load monitors in “push” load reporting model
- Implements the Asynchronous Completion Token design pattern
- May be implemented as a logical object
 - Identifier assigned by load balancer

CORBA Load Balancing Interactions

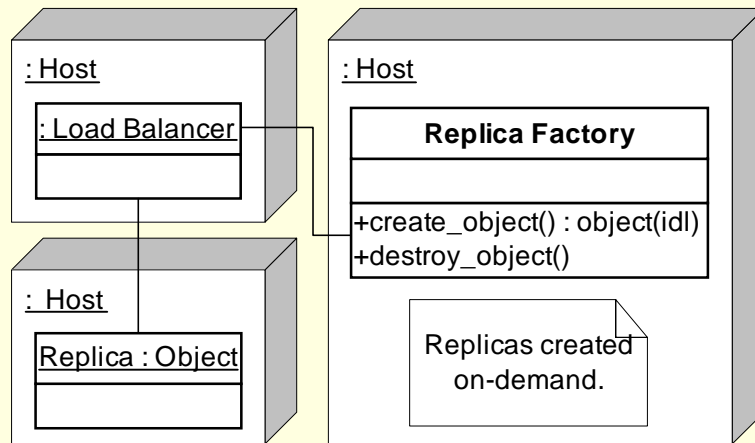


Stateless vs. Stateful Load Balancing



- Unit of Load Balancing
 - Stateless members
 - Requests
 - Stateful members
 - Object
- Object Migration
 - Stateful members must implement the Memento design pattern
 - Inherit from appropriate IDL interface

Object Group Membership



- Infrastructure controlled
 - Load balancer creates members on-demand
 - Application must supply “factories”
- Application controlled
 - Application creates object group via the load balancer
 - Application creates members and registers them with object group

Benchmarks

- *To be added prior to workshop*
- *If not available by that time, updated slides will be available at:*
 - *<http://doc.ece.uci.edu/~ossama/RTWS2002/>*

Closing Remarks

- Described Load balancing architecture and model is
 - Flexible
 - Extensible load balancing strategies
 - Freedom to implement in a variety of ways
 - Centralized
 - Decentralized / Federated / Cooperative
 - Hierarchical
 - Generic
 - Supports multiple object groups, not application-specific
 - Portable
 - Does not introduce changes to CORBA core and model
 - Familiar
 - Uses basically same group management concepts and IDL found in the Fault Tolerance chapter in the CORBA spec

Related Documents

- Joint Load Balancing and Monitoring submission
 - OMG document ...
- Slide updates available at:
 - <http://doc.ece.uci.edu/~ossama/RTWS2002/>