Design Patterns

MSc in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

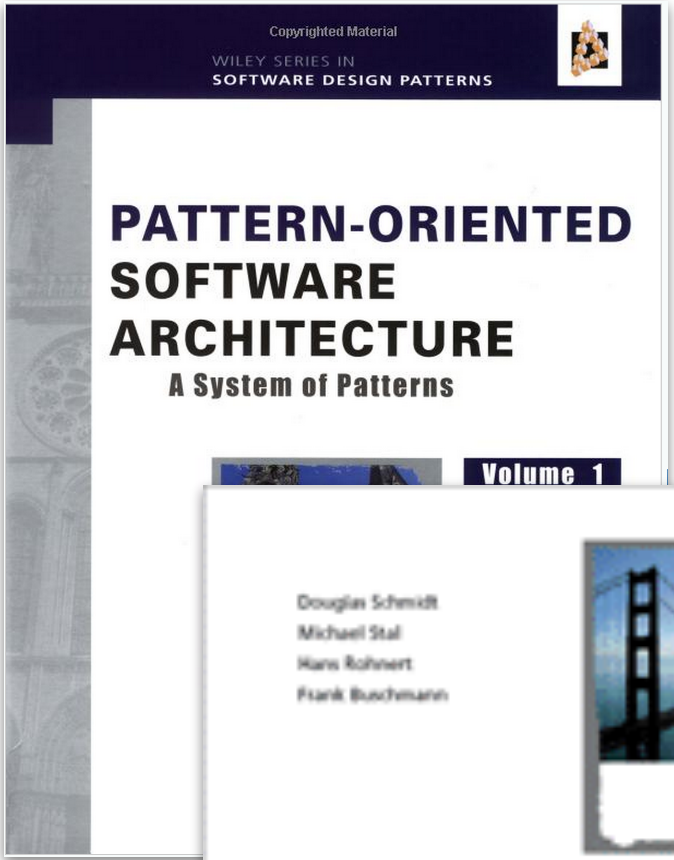
http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

POSA Patterns

Other sources - POSA - 5 Volumes!


Copyrighted Material

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
A System of Patterns

Volume 1

Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann

PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
Volume 2 Patterns for Concurrent and Networked Objects

WILEY

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
A Pattern Language for
Distributed Computing

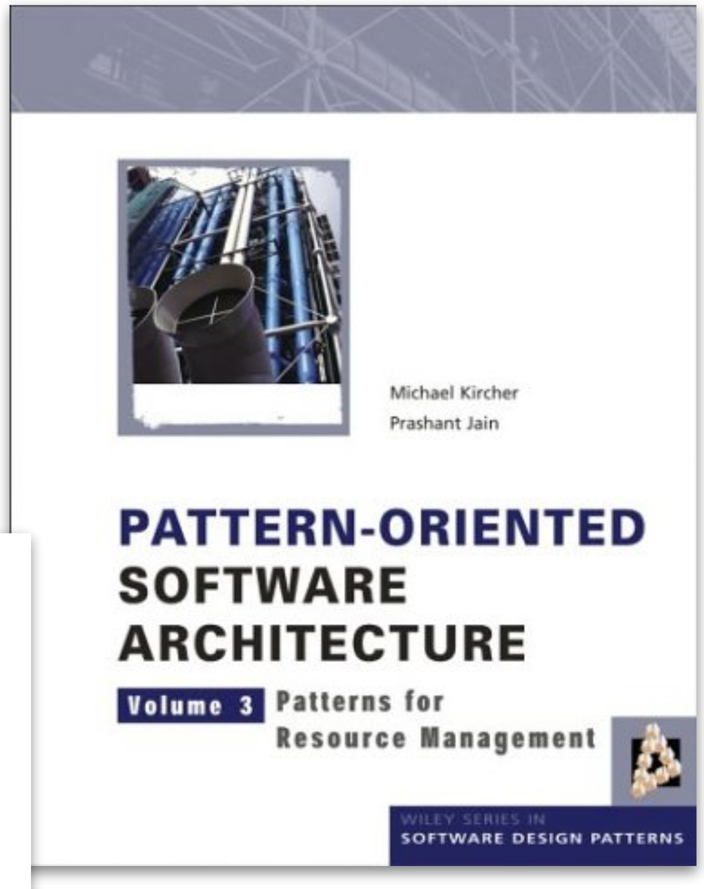
Volume 4

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
On Patterns and Pattern Languages

Volume 5

Frank Buschmann
Kevlin Henney
Douglas C. Schmidt


Michael Kircher
Prashant Jain

PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
Volume 3 Patterns for Resource Management

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

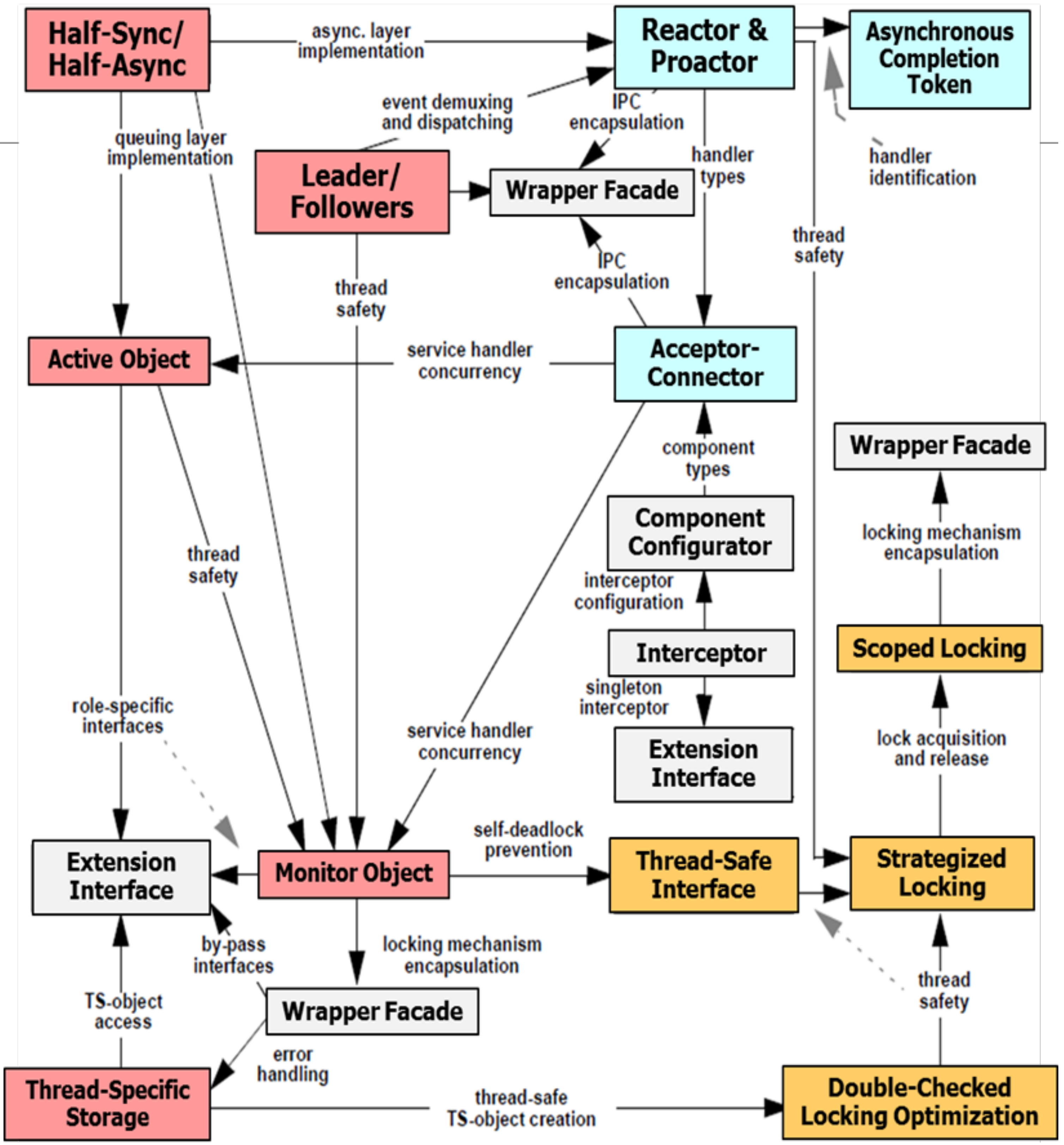
Volume 4

- Collates a terse description of large collection of patterns associated with Distributed Systems
- Repeats patterns from earlier volumes in series + patterns from GoF and PoEAA
- Extensive discussion on how multiple patterns participate in solving a given problem

WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

Catalog (Extract)

- Sophisticated and Specialised



Pattern Language

From Mud To Structure: DOMAIN MODEL (182), LAYERS (185), MODEL-VIEW-CONTROLLER (188), PRESENTATION-ABSTRACTION-CONTROL (191), MICROKERNEL (194), REFLECTION (197), PIPES AND FILTERS (200), SHARED REPOSITORY (202), BLACKBOARD (205), and DOMAIN OBJECT (208).

Distribution Infrastructure: MESSAGING (221), MESSAGE CHANNEL (224), MESSAGE ENDPOINT (227), MESSAGE TRANSLATOR (229), MESSAGE ROUTER (231), BROKER (237), CLIENT PROXY (240), REQUESTOR (242), INVOKER (244), CLIENT REQUEST HANDLER (246), SERVER REQUEST HANDLER (249), and PUBLISHER-SUBSCRIBER (234).

Event Demultiplexing and Dispatching: REACTOR (259), PROACTOR (262), ACCEPTOR-CONNECTOR (265), and ASYNCHRONOUS COMPLETION TOKEN (268).

Interface Partitioning: EXPLICIT INTERFACE (281), EXTENSION INTERFACE (284), INTROSPECTIVE INTERFACE (286), DYNAMIC INVOCATION INTERFACE (288), PROXY (290), BUSINESS DELEGATE (292), FACADE (294), COMBINED METHOD (296), ITERATOR (298), ENUMERATION METHOD (300), and BATCH METHOD (302).

Component Partitioning: ENCAPSULATED IMPLEMENTATION (313), WHOLE-PART (317), COMPOSITE (319), MASTER-SLAVE (321), HALF-OBJECT PLUS PROTOCOL (324), and REPLICATED COMPONENT GROUP (326).

Application Control: PAGE CONTROLLER (337), FRONT CONTROLLER (339), APPLICATION CONTROLLER (341), COMMAND PROCESSOR (343), TEMPLATE VIEW (345), TRANSFORM VIEW (347), FIREWALL PROXY (349), and AUTHORIZATION (351).

Concurrency: HALF-SYNC/HALF-ASYNC (359), LEADER/FOLLOWERS (362), ACTIVE OBJECT (365), MONITOR OBJECT (368).

Synchronization: GUARDED SUSPENSION (380), FUTURE (382), THREAD-SAFE INTERFACE (384), DOUBLE- CHECKED LOCKING (386), STRATEGIZED LOCKING (388), SCOPED LOCKING (390), THREAD-SPECIFIC STORAGE (392), COPIED VALUE (394), and IMMUTABLE VALUE (396).

Object Interaction: OBSERVER (405), DOUBLE DISPATCH (408), MEDIATOR (410), MEMENTO (414), CONTEXT OBJECT (416), DATA TRANSFER OBJECT (418), COMMAND (412), and MESSAGE (420).

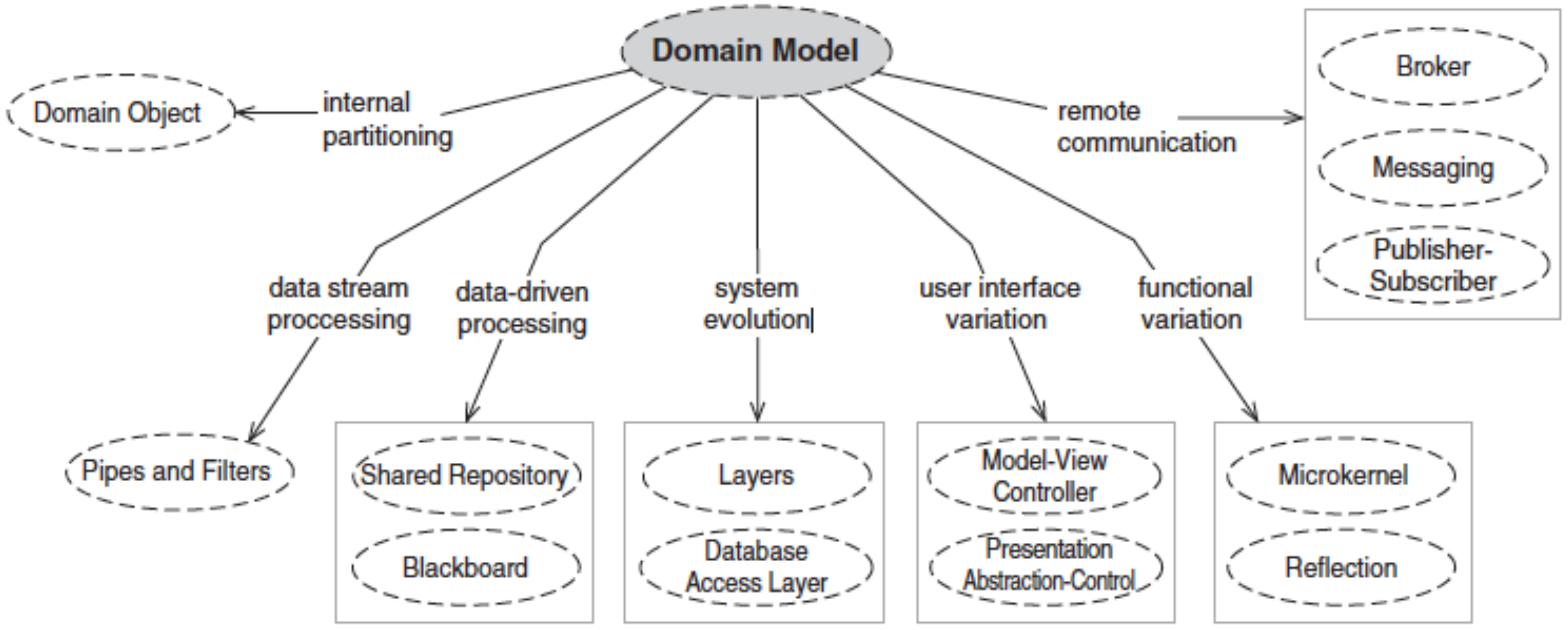
Adaptation and Extension: BRIDGE (436), OBJECT ADAPTER (438), INTERCEPTOR (444), CHAIN OF RESPONSIBILITY (440), INTERPRETER (442), VISITOR (447), DECORATOR (449), TEMPLATE METHOD (453), STRATEGY (455), NULL OBJECT (457), WRAPPER FACADE (459), EXECUTE-AROUND OBJECT (451), and DECLARATIVE COMPONENT CONFIGURATION (461).

Object Behavior: OBJECTS FOR STATES (467), METHODS FOR STATES (469), and COLLECTIONS FOR STATES (471).

Resource Management: OBJECT MANAGER (492), CONTAINER (488), COMPONENT CONFIGURATOR (490), LOOKUP (495), VIRTUAL PROXY (497), LIFECYCLE CALLBACK (499), TASK COORDINATOR (501), RESOURCE POOL (503), RESOURCE CACHE (505), LAZY ACQUISITION (507), EAGER ACQUISITION (509), PARTIAL ACQUISITION (511), ACTIVATOR (513), EVICTOR (515), LEASING (517), AUTOMATED GARBAGE COLLECTION (519), COUNTING HANDLE (522), ABSTRACT FACTORY (525), BUILDER (527), FACTORY METHOD (529), and DISPOSAL METHOD (531).

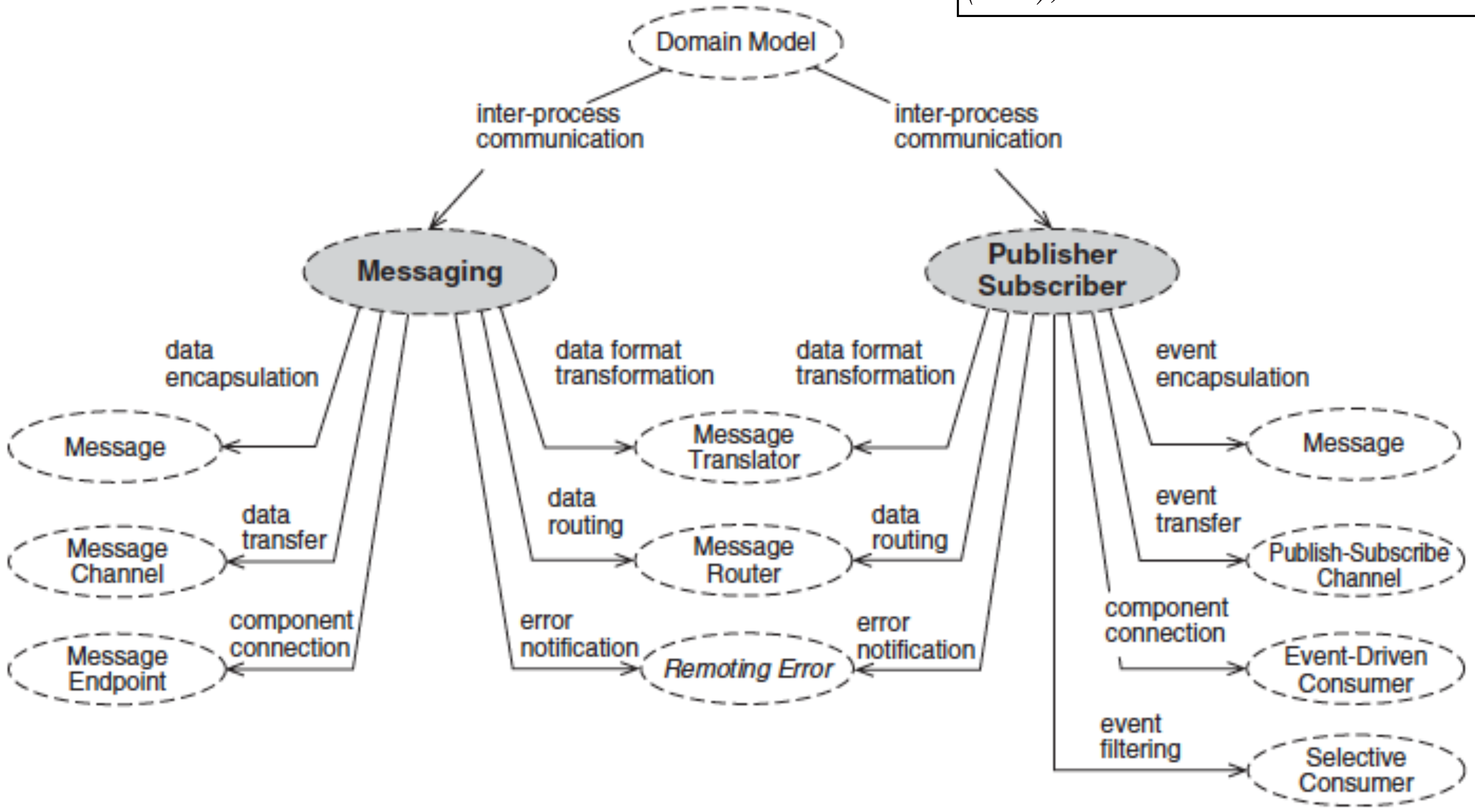
From Mud To Structure

From Mud To Structure: DOMAIN MODEL (182), LAYERS (185), MODEL-VIEW-CONTROLLER (188), PRESENTATION-ABSTRACTION-CONTROL (191), MICROKERNEL (194), REFLECTION (197), PIPES AND FILTERS (200), SHARED REPOSITORY (202), BLACKBOARD (205), and DOMAIN OBJECT (208).



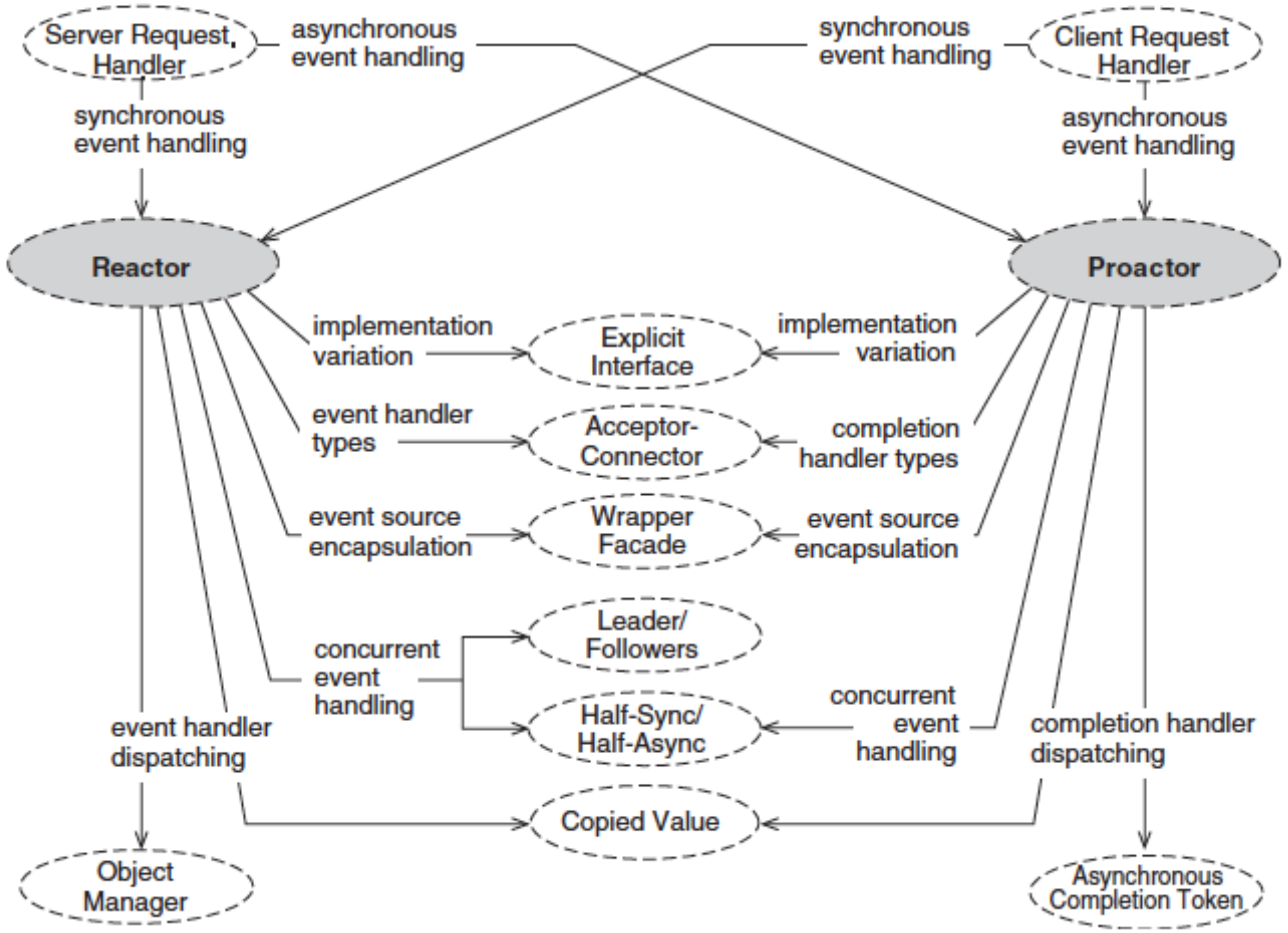
Distribution Infrastructure

Distribution Infrastructure: MESSAGING (221), MESSAGE CHANNEL (224), MESSAGE ENDPOINT (227), MESSAGE TRANSLATOR (229), MESSAGE ROUTER (231), BROKER (237), CLIENT PROXY (240), REQUESTOR (242), INVOKER (244), CLIENT REQUEST HANDLER (246), SERVER REQUEST HANDLER (249), and PUBLISHER-SUBSCRIBER (234).



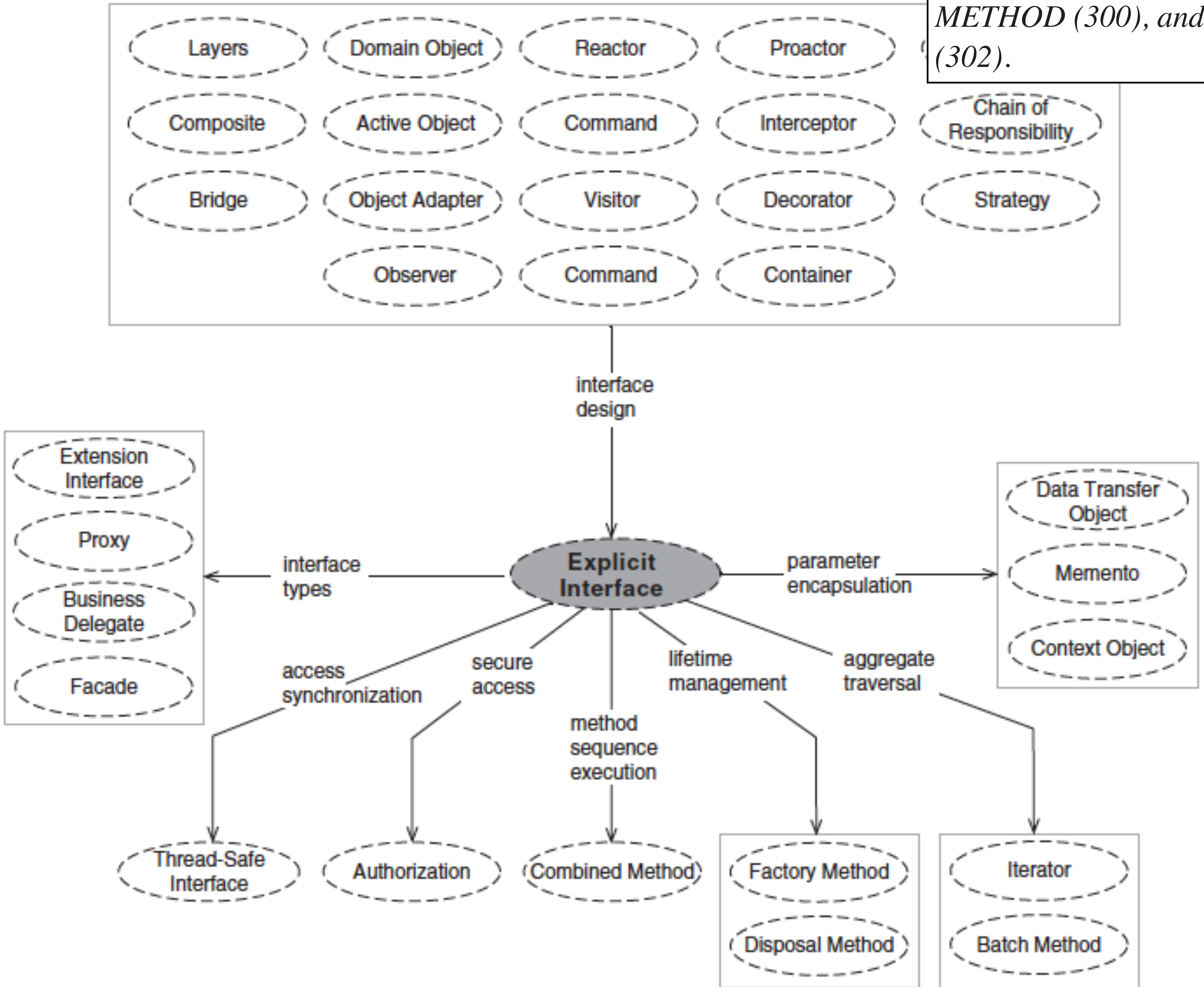
Event Demultiplexing and Dispatching

Event Demultiplexing and Dispatching: REACTOR (259), PROACTOR (262), ACCEPTOR-CONNECTOR (265), and ASYNCHRONOUS COMPLETION TOKEN (268).


Server Request Handler
asynchronous event handling
synchronous event handling
Client Request Handler
synchronous event handling
asynchronous event handling
Reactor
Proactor
implementation variation
Explicit Interface
implementation variation
event handler types
Acceptor-Connector
completion handler types
event source encapsulation
Wrapper Facade
event source encapsulation
Leader/Followers
concurrent event handling
Half-Sync/Half-Async
concurrent event handling
event handler dispatching
completion handler dispatching
Copied Value
Object Manager
Asynchronous Completion Token

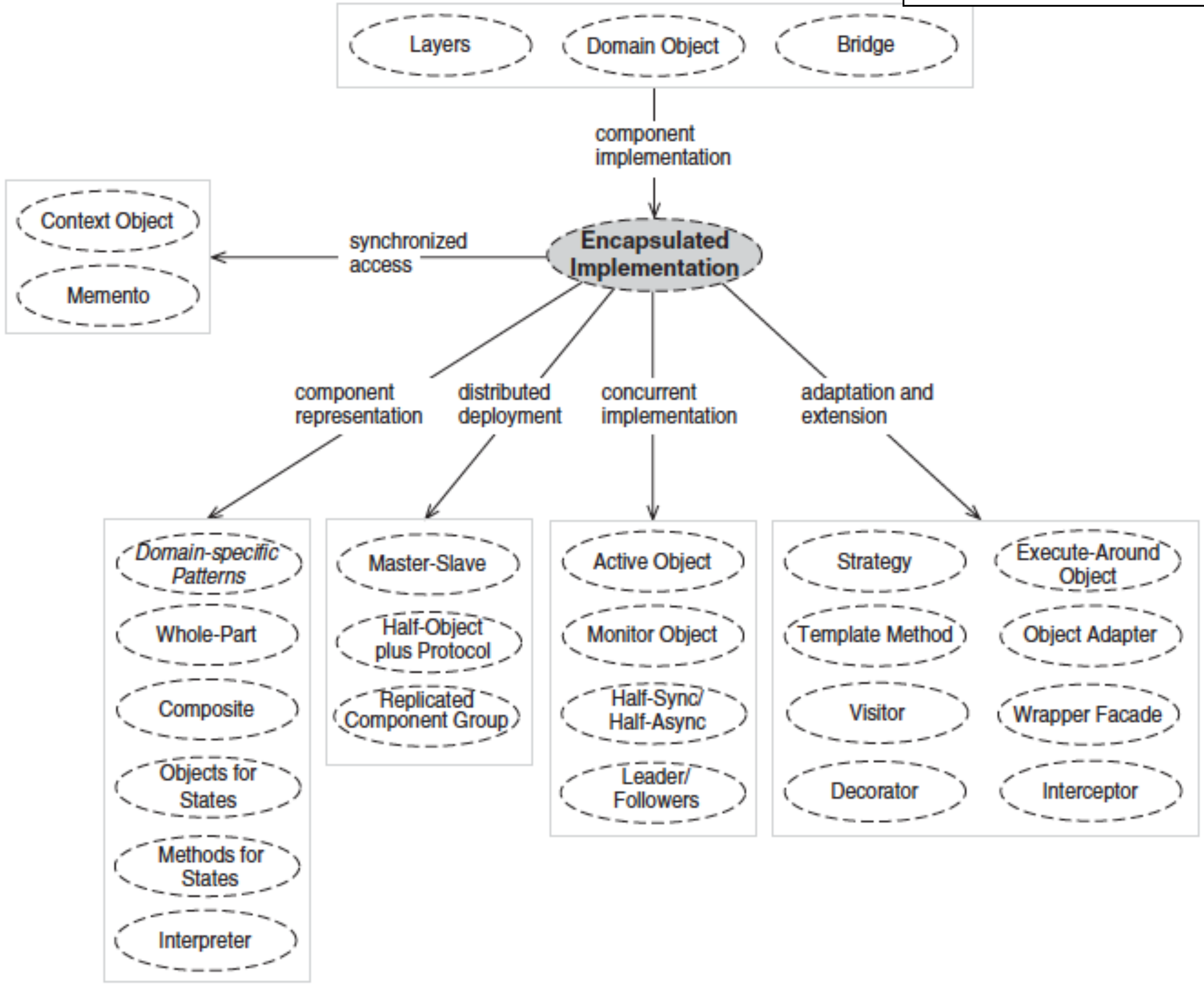
Interface Partitioning

Interface Partitioning: EXPLICIT INTERFACE (281), EXTENSION INTERFACE (284), INTROSPECTIVE INTERFACE (286), DYNAMIC INVOCATION INTERFACE (288), PROXY (290), BUSINESS DELEGATE (292), FACADE (294), COMBINED METHOD (296), ITERATOR (298), ENUMERATION METHOD (300), and BATCH METHOD (302).

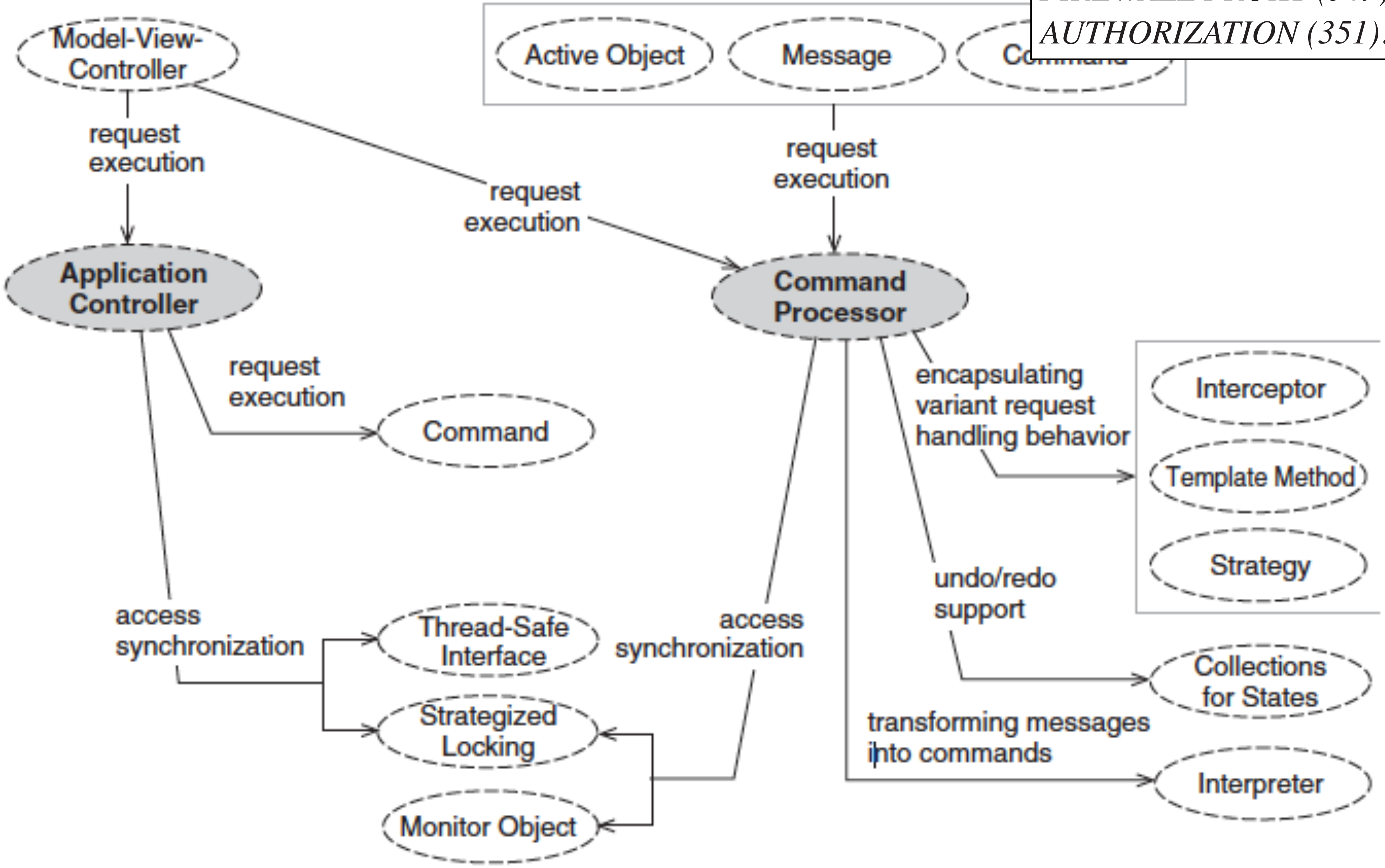


Component Partitioning

Component Partitioning: ENCAPSULATED IMPLEMENTATION (313), WHOLE-PART (317), COMPOSITE (319), MASTER-SLAVE (321), HALF-OBJECT PLUS PROTOCOL (324), and REPLICATED COMPONENT GROUP (326).

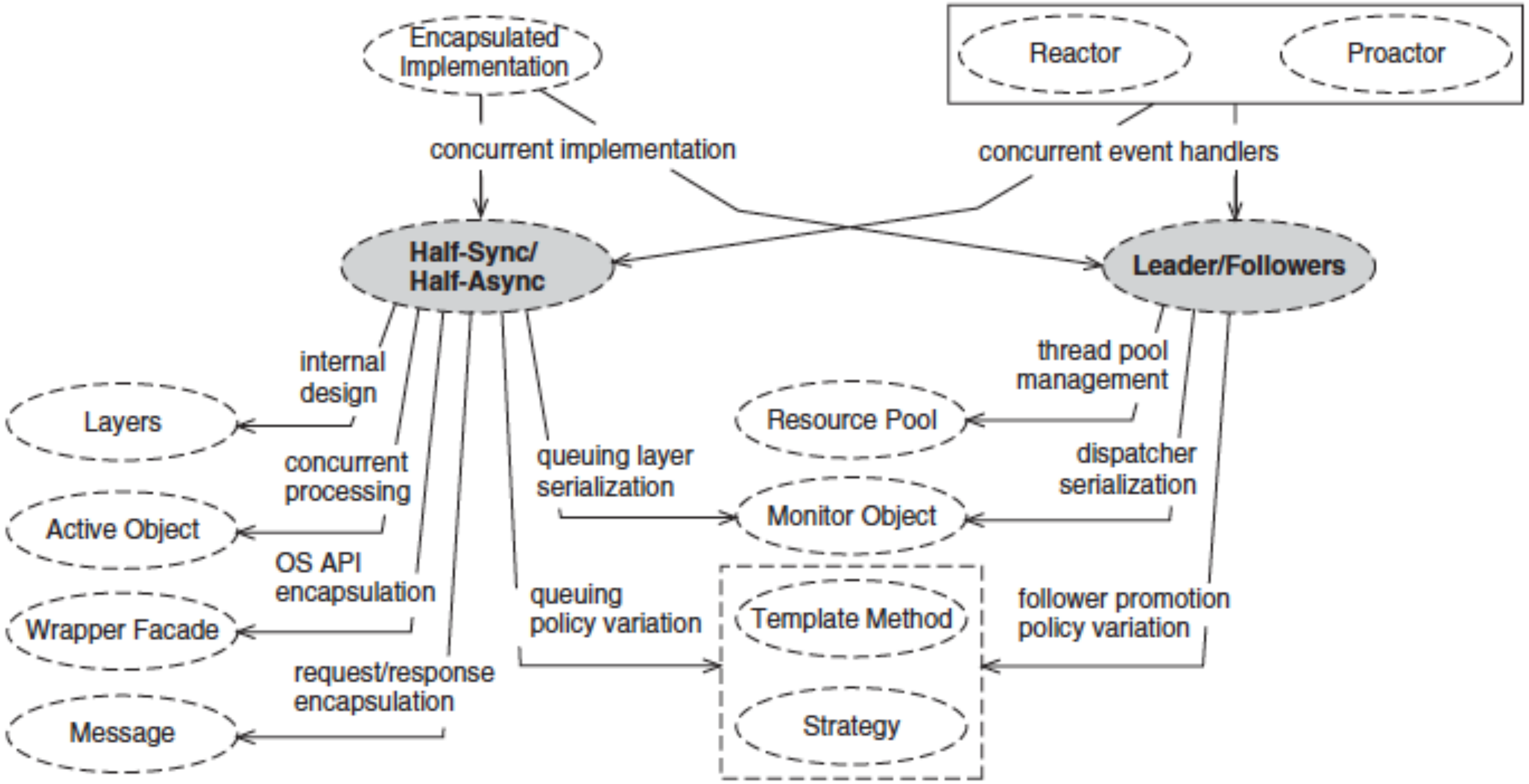


Application Control: PAGE CONTROLLER (337), FRONT CONTROLLER (339), APPLICATION CONTROLLER (341), COMMAND PROCESSOR (343), TEMPLATE VIEW (345), TRANSFORM VIEW (347), FIREWALL PROXY (349), and AUTHORIZATION (351).



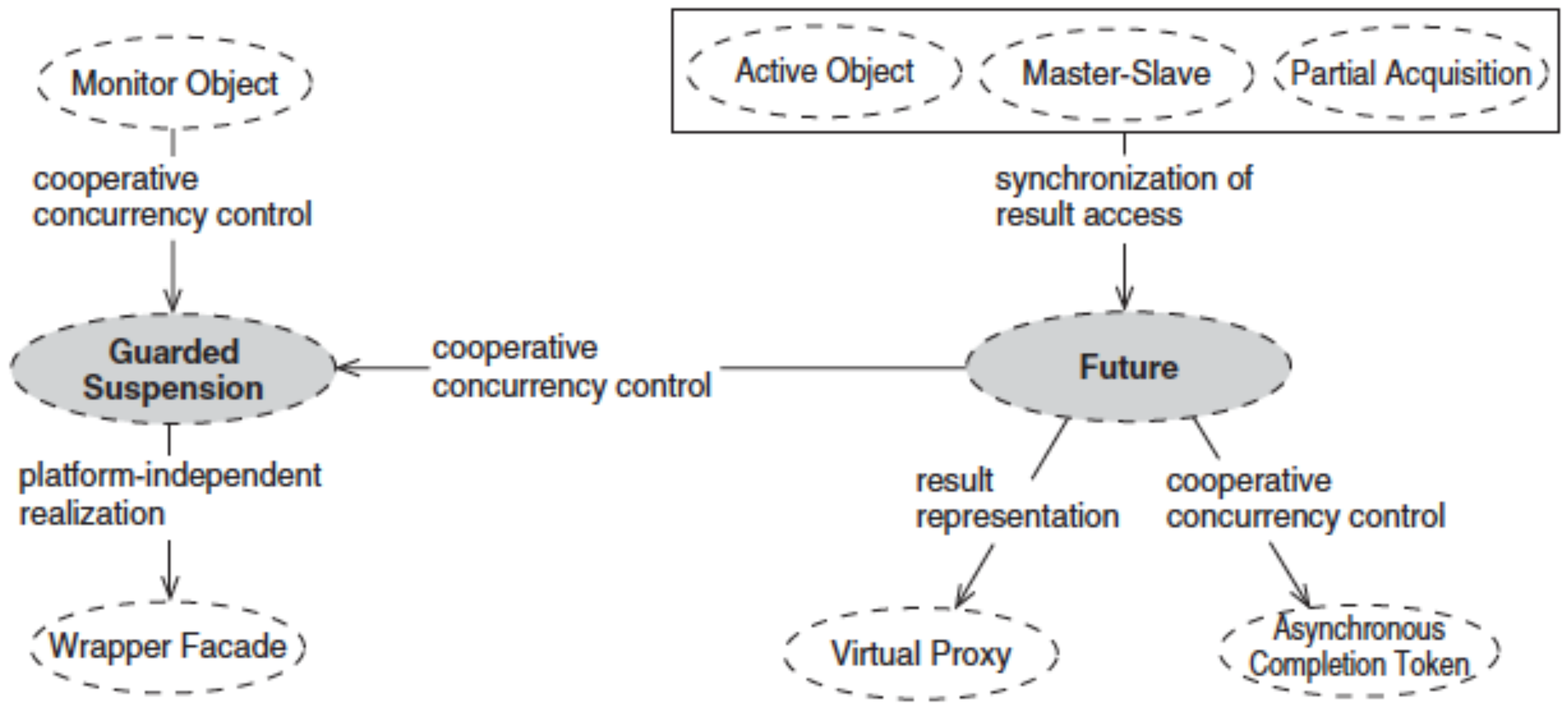
Concurrency

Concurrency: HALF-SYNC/HALF-ASYNC (359), LEADER/FOLLOWERS (362), ACTIVE OBJECT (365), MONITOR OBJECT (368).



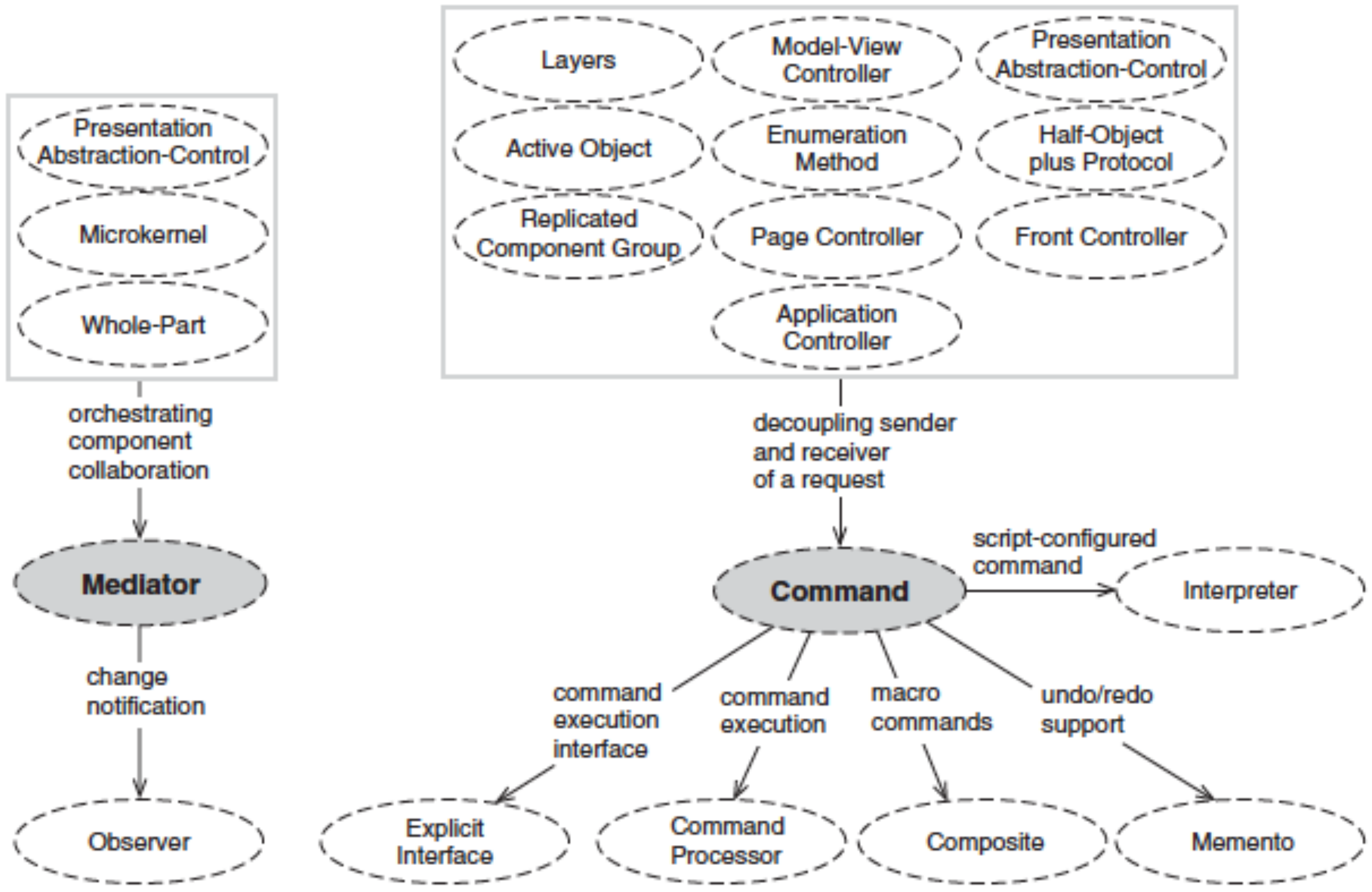
Synchronization

Synchronization: GUARDED SUSPENSION (380), FUTURE (382), THREAD-SAFE INTERFACE (384), DOUBLE- CHECKED LOCKING (386), STRATEGIZED LOCKING (388), SCOPED LOCKING (390), THREAD-SPECIFIC STORAGE (392), COPIED VALUE (394), and IMMUTABLE VALUE (396).

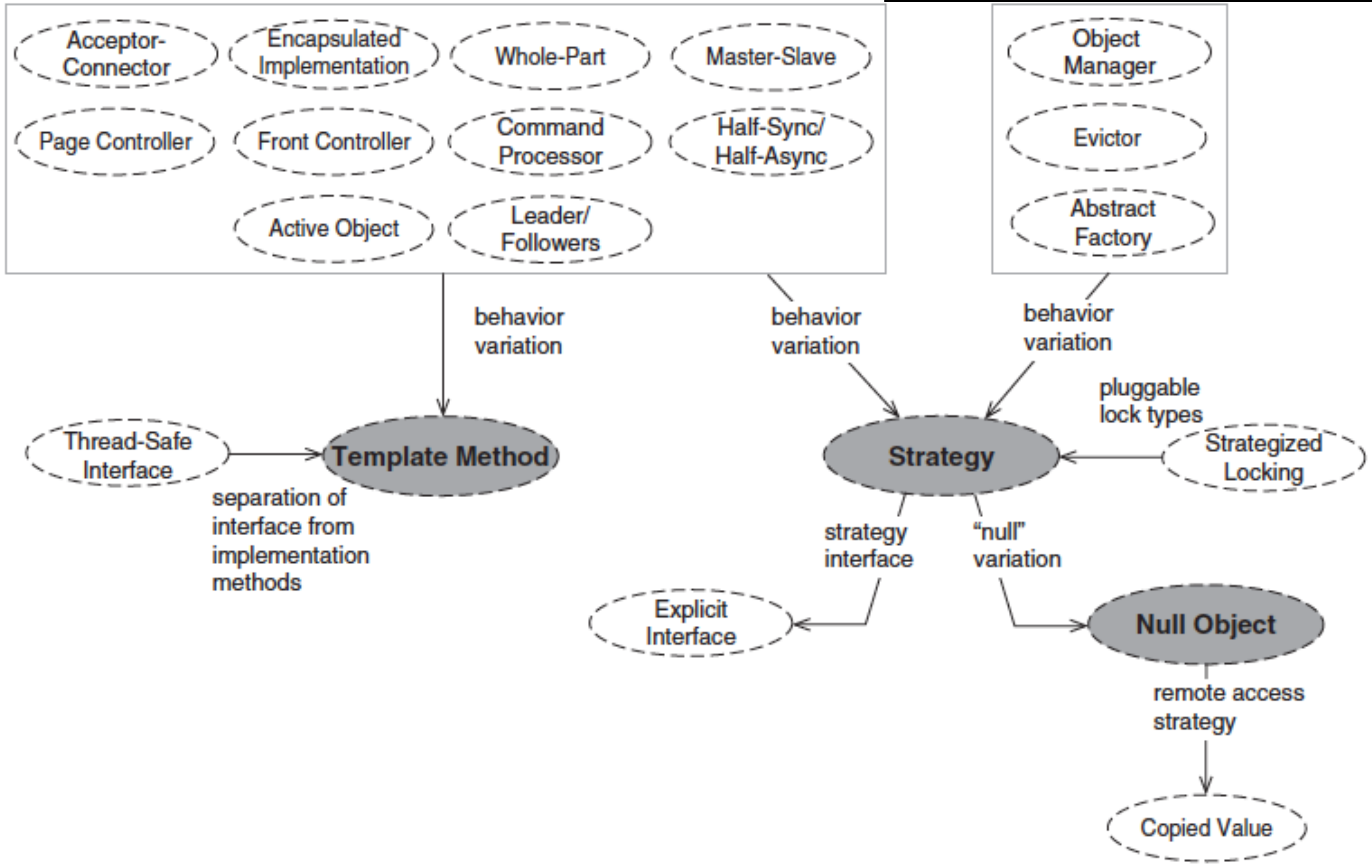

Monitor Object
cooperative concurrency control
Active Object
Master-Slave
Partial Acquisition
synchronization of result access
Guarded Suspension
cooperative concurrency control
Future
platform-independent realization
result representation
cooperative concurrency control
Wrapper Facade
Virtual Proxy
Asynchronous Completion Token

Object Interaction:

Object Interaction: OBSERVER (405), DOUBLE DISPATCH (408), MEDIATOR (410), MEMENTO (414), CONTEXT OBJECT (416), DATA TRANSFER OBJECT (418), COMMAND (412), and MESSAGE (420).

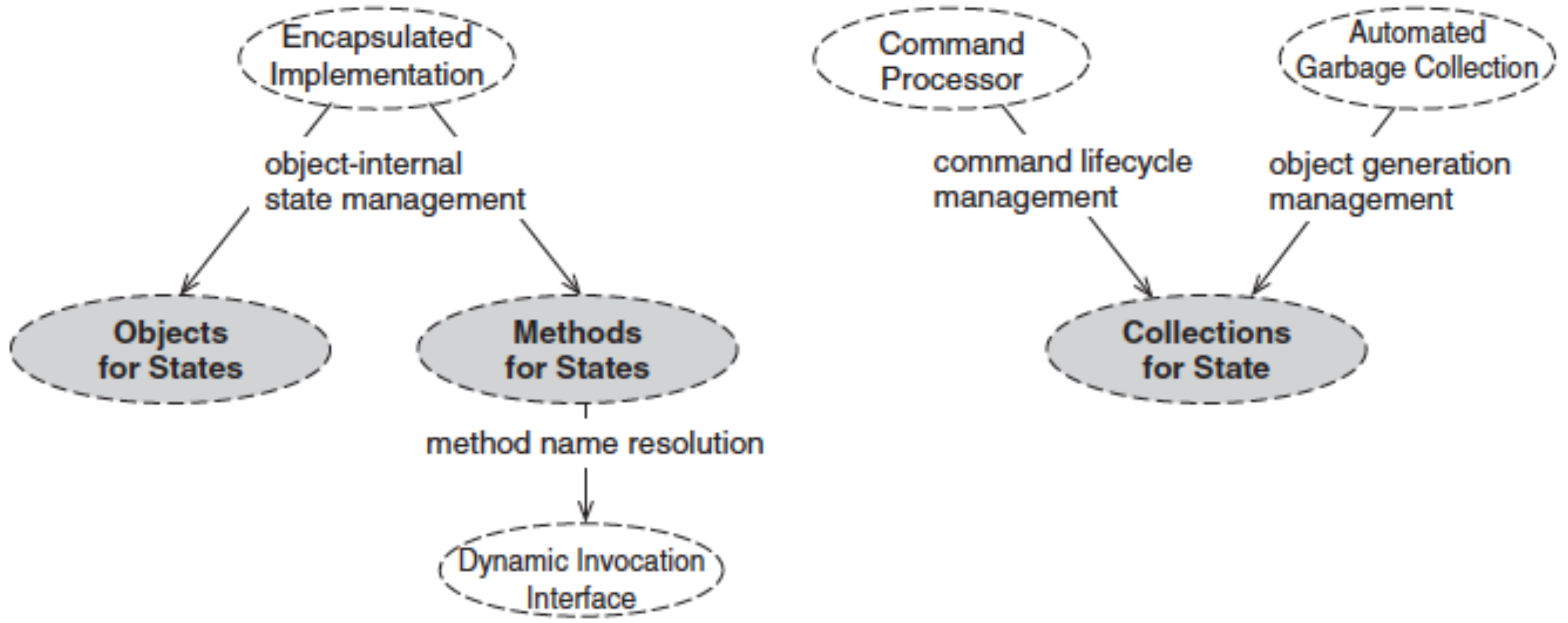


Adaptation and Extension: BRIDGE (436), OBJECT ADAPTER (438), INTERCEPTOR (444), CHAIN OF RESPONSIBILITY (440), INTERPRETER (442), VISITOR (447), DECORATOR (449), TEMPLATE METHOD (453), STRATEGY (455), NULL OBJECT (457), WRAPPER FACADE (459), EXECUTE-AROUND OBJECT (451), and DECLARATIVE COMPONENT CONFIGURATION (461).

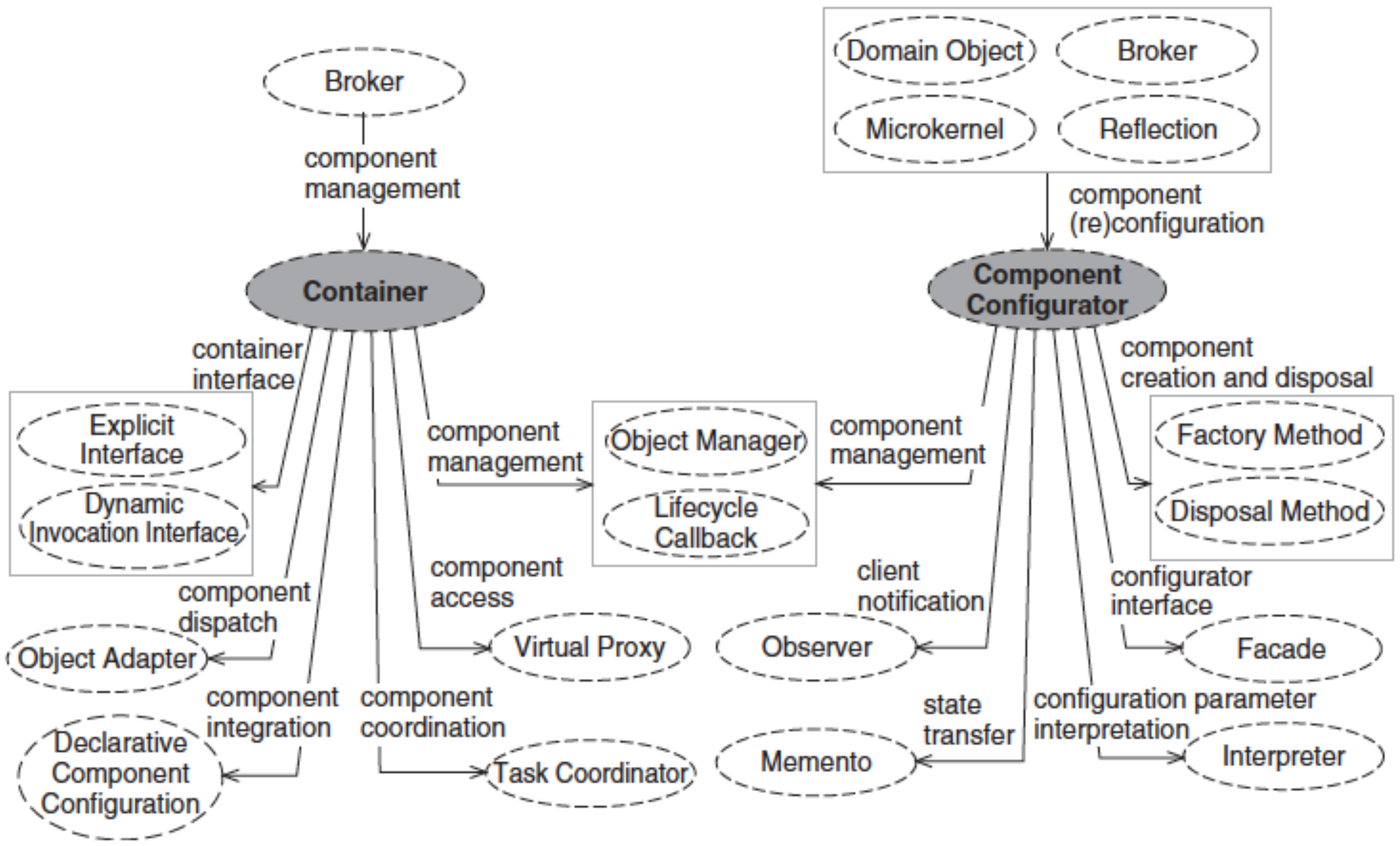


Object behaviour

Object Behavior: OBJECTS FOR STATES (467), METHODS FOR STATES (469), and COLLECTIONS FOR STATES (471).

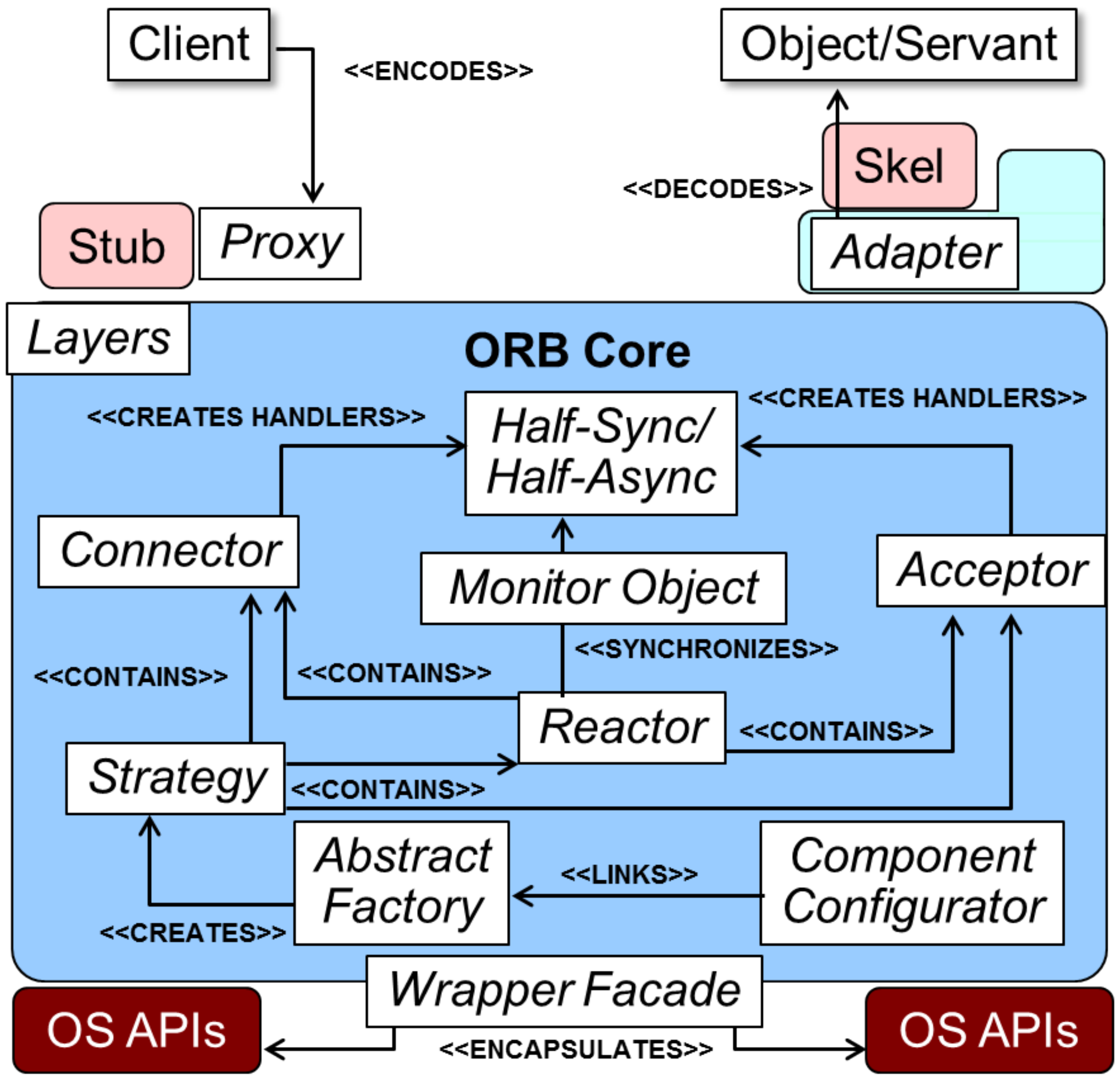

Encapsulated Implementation
object-internal state management
Objects for States
Methods for States
method name resolution
Dynamic Invocation Interface
Command Processor
command lifecycle management
Automated Garbage Collection
object generation management
Collections for State

Resource Management: OBJECT MANAGER (492), CONTAINER (488), COMPONENT CONFIGURATOR (490), LOOKUP (495), VIRTUAL PROXY (497), LIFECYCLE CALLBACK (499), TASK COORDINATOR (501), RESOURCE POOL (503), RESOURCE CACHE (505), LAZY ACQUISITION (507), EAGER ACQUISITION (509), PARTIAL ACQUISITION (511), ACTIVATOR (513), EVICTOR (515), LEASING (517), AUTOMATED GARBAGE COLLECTION (519), COUNTING HANDLE (522), ABSTRACT FACTORY (525), BUILDER (527), FACTORY METHOD (529), and DISPOSAL METHOD (531).



Composed Patterns Example

- 14 Patterns involved in implementing a Object Request Broker



Half Sync / Half Sync Example

- Android Activities are synchronous - single threaded in the context of user interaction
- Service access is asynchronous - inherently unreliable access to remote application service
- Half Sync/Half Async an appropriate pattern to tackle this problem

Half Sync/ Half Async Pattern - Context

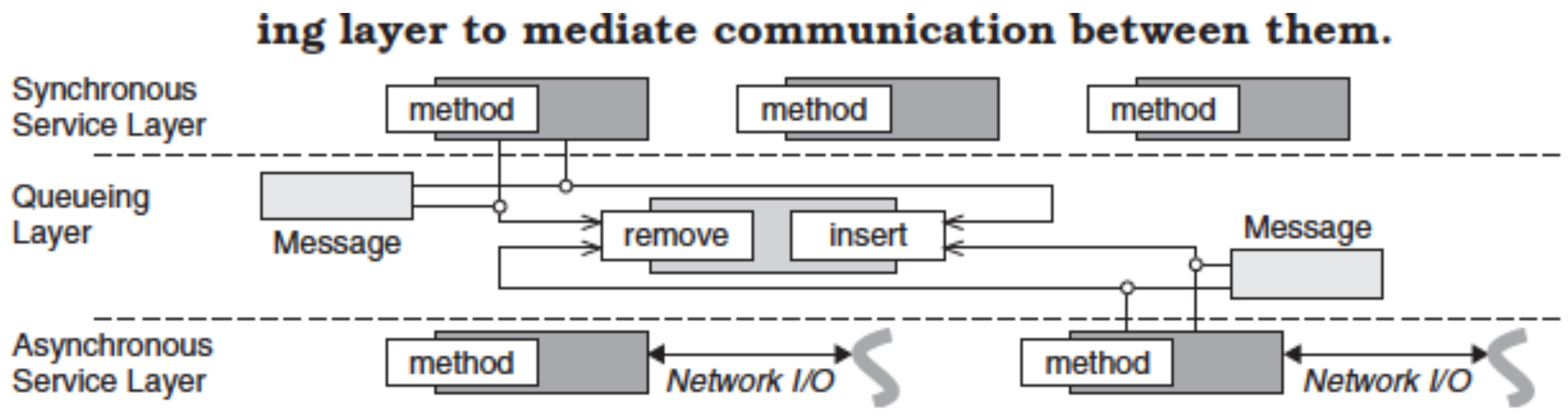
- When developing concurrent software, specifically a concurrent ENCAPSULATED IMPLEMENTATION (313) or a network server that employs a REACTOR (259) or PROACTOR (262) event handling infrastructure . . .
- . . . we need to make performance efficient and scalable while ensuring that any use of concurrency simplifies programming.

Half Sync/ Half Async Pattern - Synchrony

- Concurrent software often performs both asynchronous and synchronous service processing.
  - Asynchrony is used to process low-level system services efficiently,
  - Synchrony to simplify application service processing.
- To benefit from both programming models, however, it is essential to coordinate asynchronous and synchronous service processing efficiently.

Half Sync/ Half Async Pattern - Structure

- Decompose the services of concurrent software into two separated layers—synchronous and asynchronous—and add a queueing layer to mediate communication between them.



ing layer to mediate communication between them.
Synchronous Service Layer
method
method
method
Queueing Layer
Message
remove
insert
Message
Asynchronous Service Layer
method
Network I/O
method
Network I/O

Half Sync/ Half Async Pattern - Responsibilities

- Process higher-level services, such as domain functionality, database queries, or file transfers, synchronously in separate threads or processes.

- Conversely, process lower-level system services, such as short-lived protocol handlers driven by interrupts from network hard- ware, asynchronously.

- If services in the synchronous layer must communicate with services in the asynchronous layer, have them exchange messages via a queueing layer.

Half Sync/ Half Async Pattern - Benefits

- *Simplification & performance*
    - The programming of higher-level synchronous processing services are simplified without degrading the performance of lower-level system services
- *Separation of concerns*
    - Synchronization policies in each layer are decoupled so that each layer need not use the same concurrency strategies
- *Centralization of inter-layer communication*
    - Inter-layer communication is centralized at a single access point, because all interaction is mediated by the queueing layer

Half Sync/ Half Async Pattern - Limitations

- *May incur a boundary-crossing penalty*
  - Arising from context switching, synchronization, & data copying overhead when data transferred between sync & async service layers via queueing layer
- *Higher-level app services may not benefit from async I/O*
  - Depending on design of OS or application framework interfaces, higher-level services may not use low-level async I/O devices effectively
- *Complexity of debugging & testing*
  - Apps can be hard to debug due to concurrent execution

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/