Resource Access Decision Server: Design and Performance Considerations

Konstantin Beznosov and Luis Espinal

{beznosov,lespin03}@cs.fiu.edu

CADSE

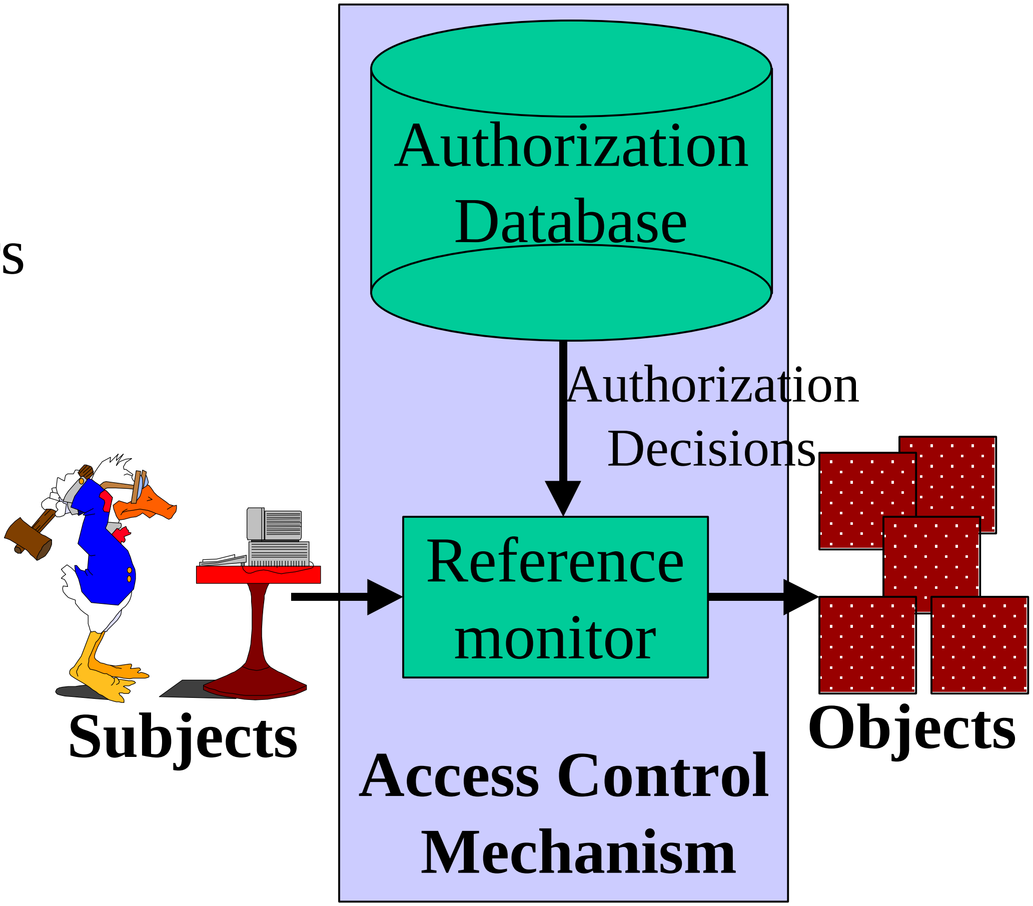
October 22, November 5, 1999

Presentation Overview

- Introduction
- RAD Specification Overview
- RAD Prototype Design
- Performance Measurements
  - Model, Measurements, Results
  - Implementation Considerations
- Conclusions

Introduction: Access Control, etc.

- Access control
  - concerned with limiting activity of legitimate users
  - enforced by a reference monitor
- Authorization
  - concerned with making access control decisions



Authorization Database

Authorization Decisions

Reference monitor

Subjects

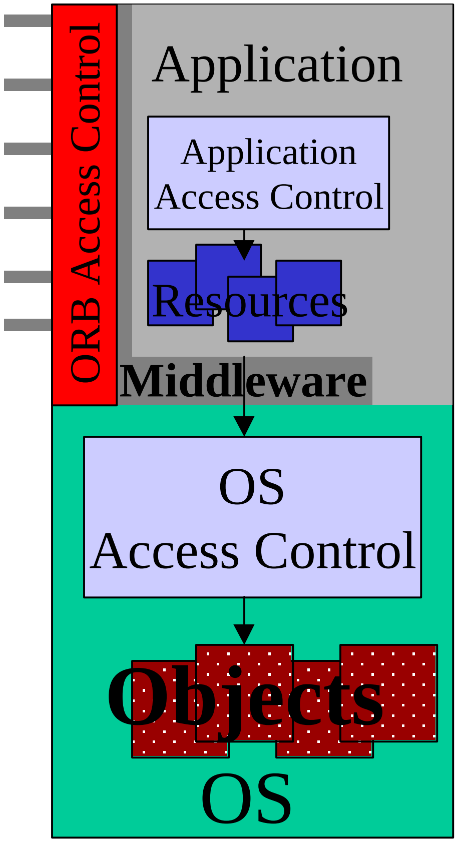
Objects

Access Control Mechanism

Access Control: Stand Alone vs. Distributed Systems

Stand Alone

- Primitive operations on objects controlled by OS (create, read, write, delete, use)
- Objects are homogenous (files, processes, memory)
- Single point of control
- Application access control is mangled with application logic



Distributed OO

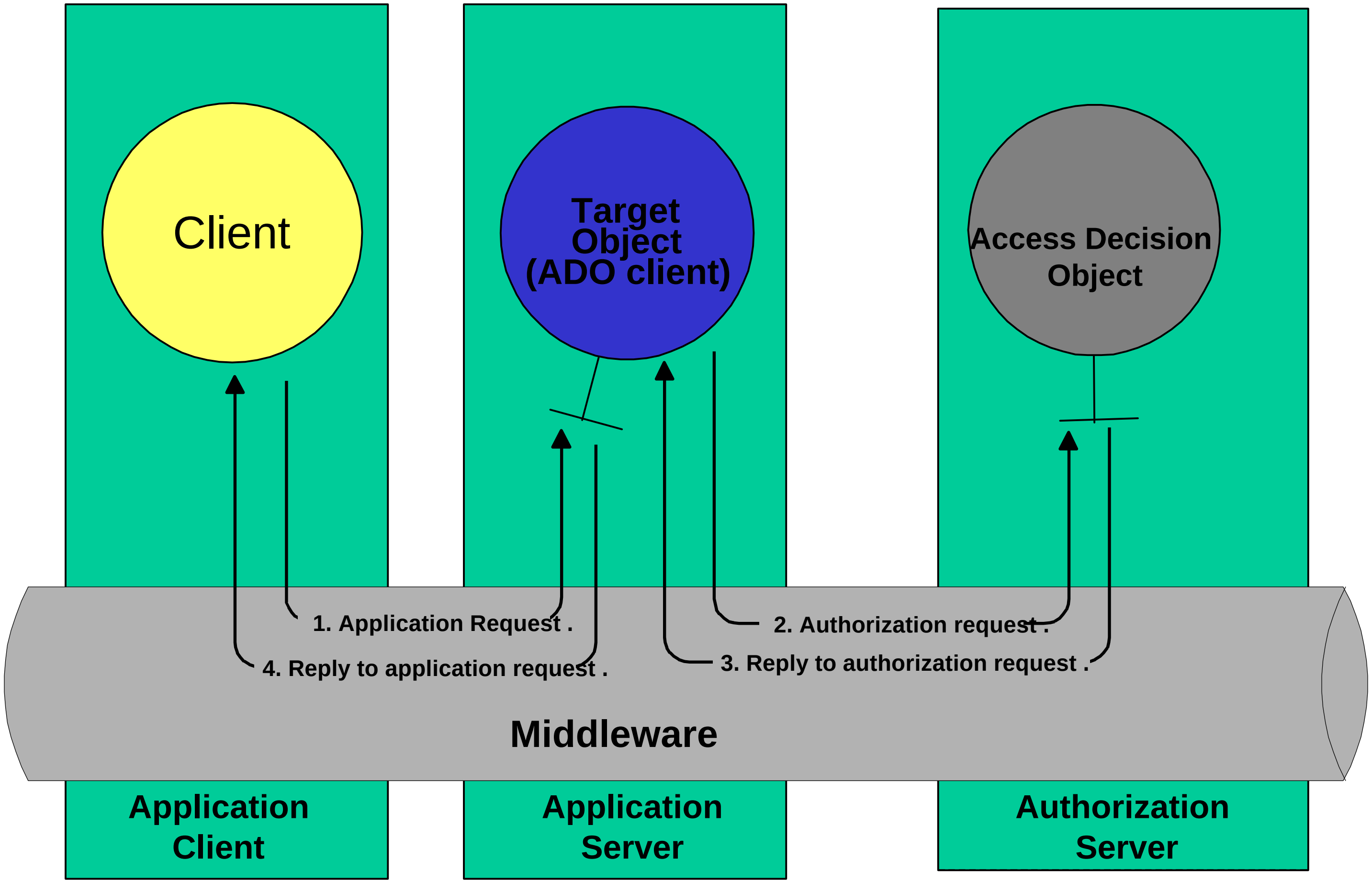
- Stand alone systems, +
- Complex operations on interfaces
- Resources are heterogeneous (different interfaces),
- Many points of control (commonality, consistency, administration issues)

The Problem with Access Control in Distributed Systems

It is **difficult** to **develop** distributed systems that:

- insure commonality and consistency of policies
- perform security administration
- support access control for fine-grain resources
- allow changing policies without changing systems
- easy to verify and test

A Possible Solution


Client
Target Object (ADO client)
Access Decision Object
1. Application Request .
2. Authorization request .
3. Reply to authorization request .
4. Reply to application request .
Middleware
Application Client
Application Server
Authorization Server

Objective Statement

Study validity of the approach from the following perspectives

- Performance and scalability
- Ability to separate application logic from authorization logic (it works and performs)
- Ability to enforce complex policies and change them without pain
- Ability to test and verify application and authorization functionalities independently

Objective Analysis

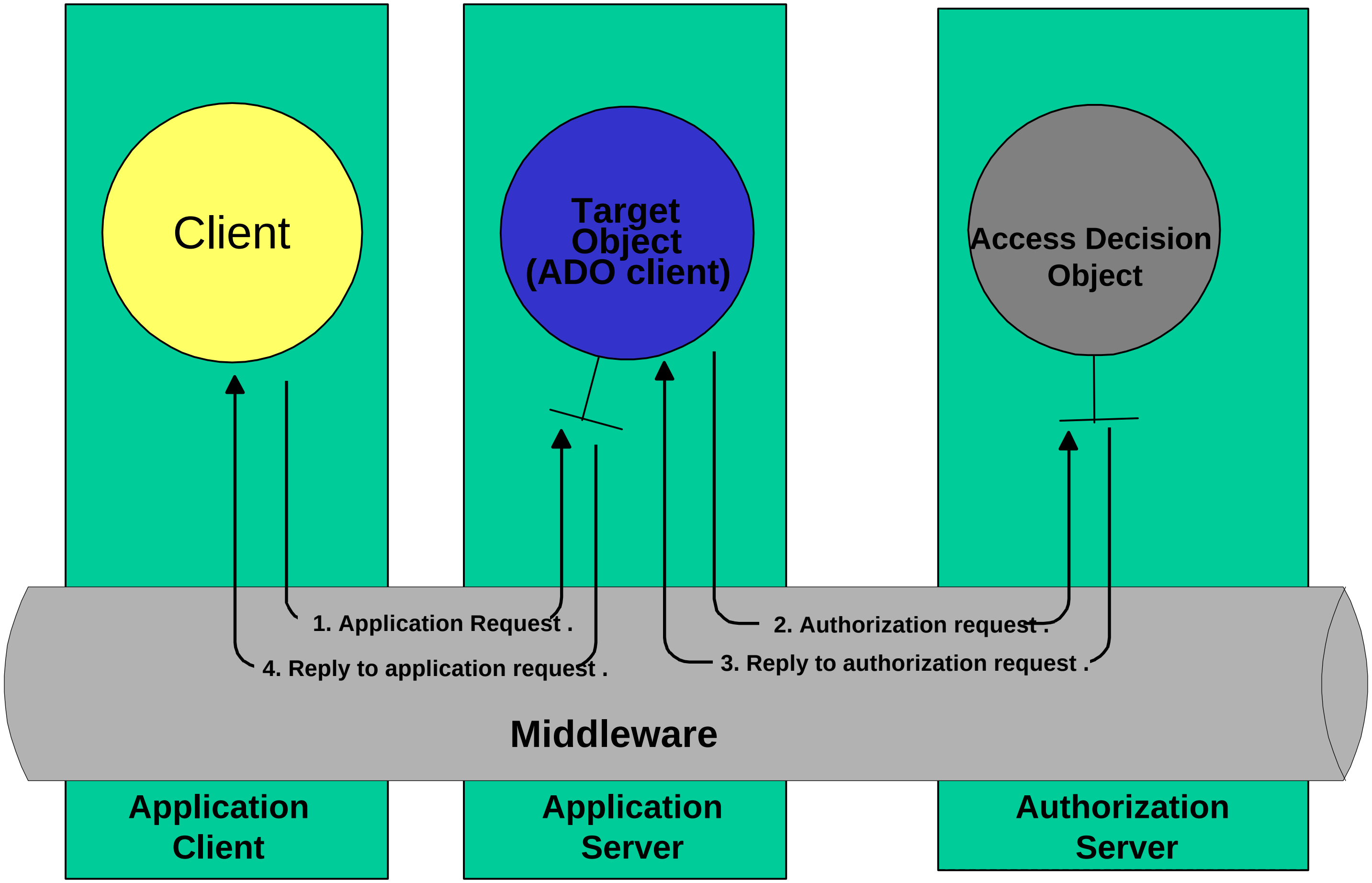
- Why is this the right goal?
  - By solving it, we will be able to assess the validity of the approach
    - Help system designers and enterprise architects in constructing, verifying, and testing distributed systems.
- Why is the goal worth addressing?
  - It is doable
  - Its results could be applicable to other security policies and mechanisms (audit, quality of protection, non-repudiation)

Research Directions

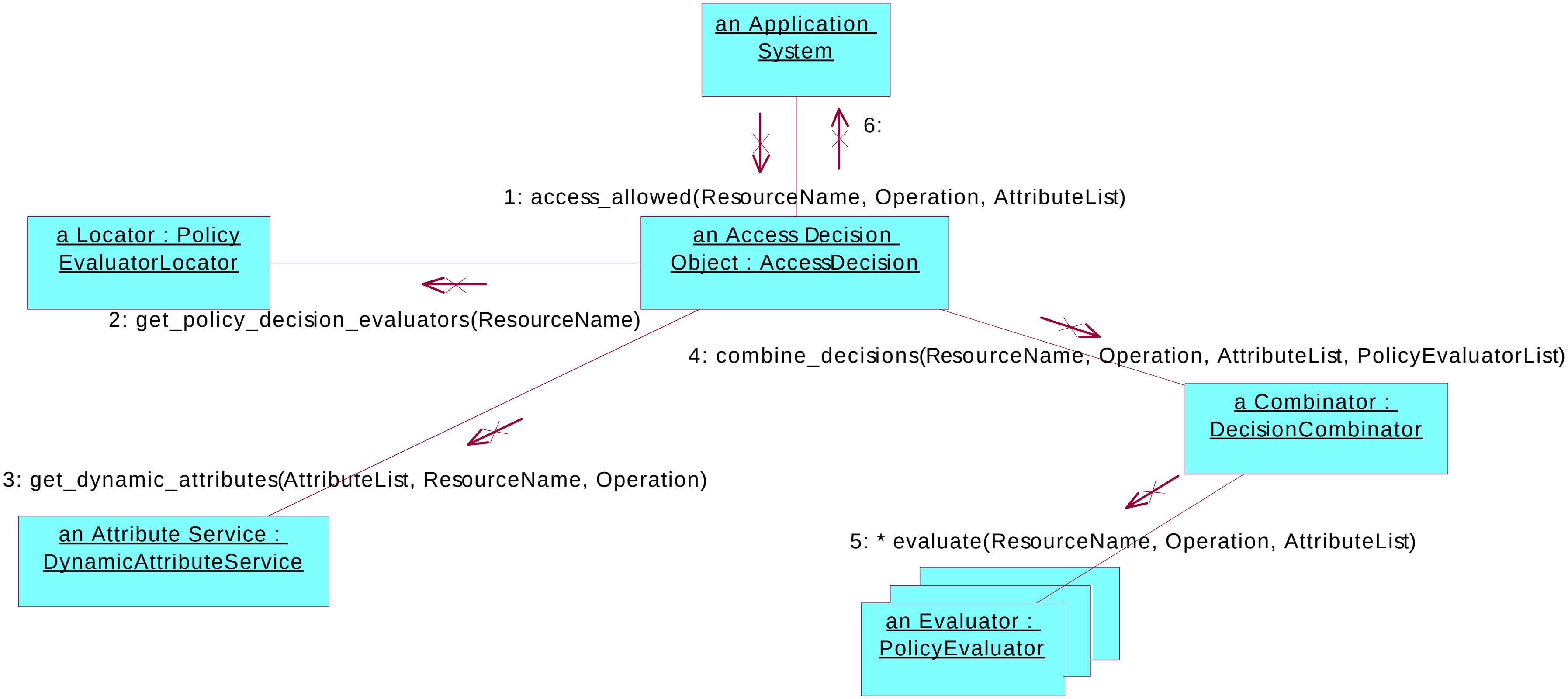
+ Develop a prototype
+ Measure performance

- Study the validity of the main claims
  - – support for different access control policy types
    - extend the prototype to support various policy types?
  - – consistency and commonality of access control policies
- ???

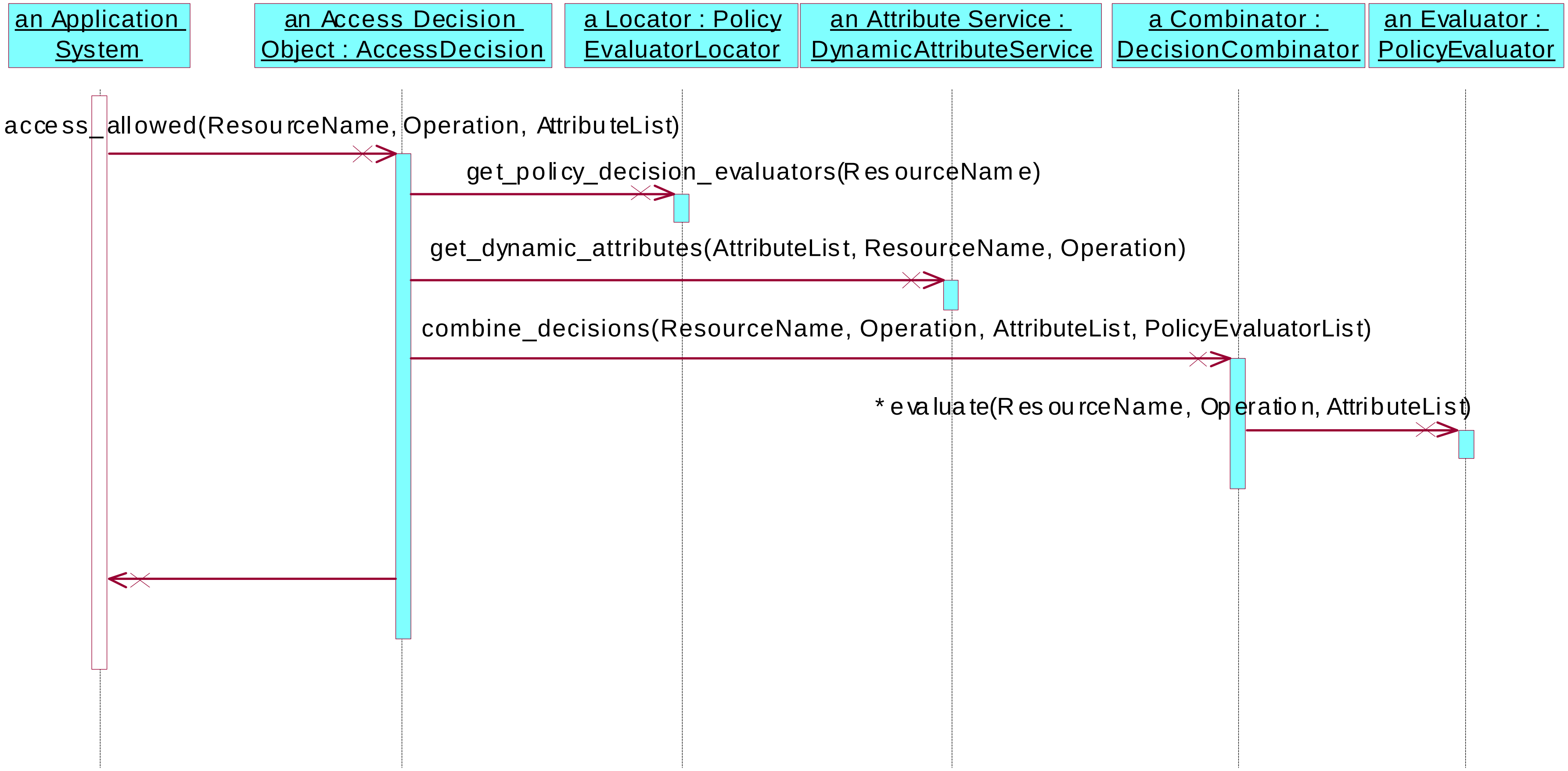
RAD Specification


Client
Target Object (ADO client)
Access Decision Object
1. Application Request .
2. Authorization request .
3. Reply to authorization request .
4. Reply to application request .
Middleware
Application Client
Application Server
Authorization Server

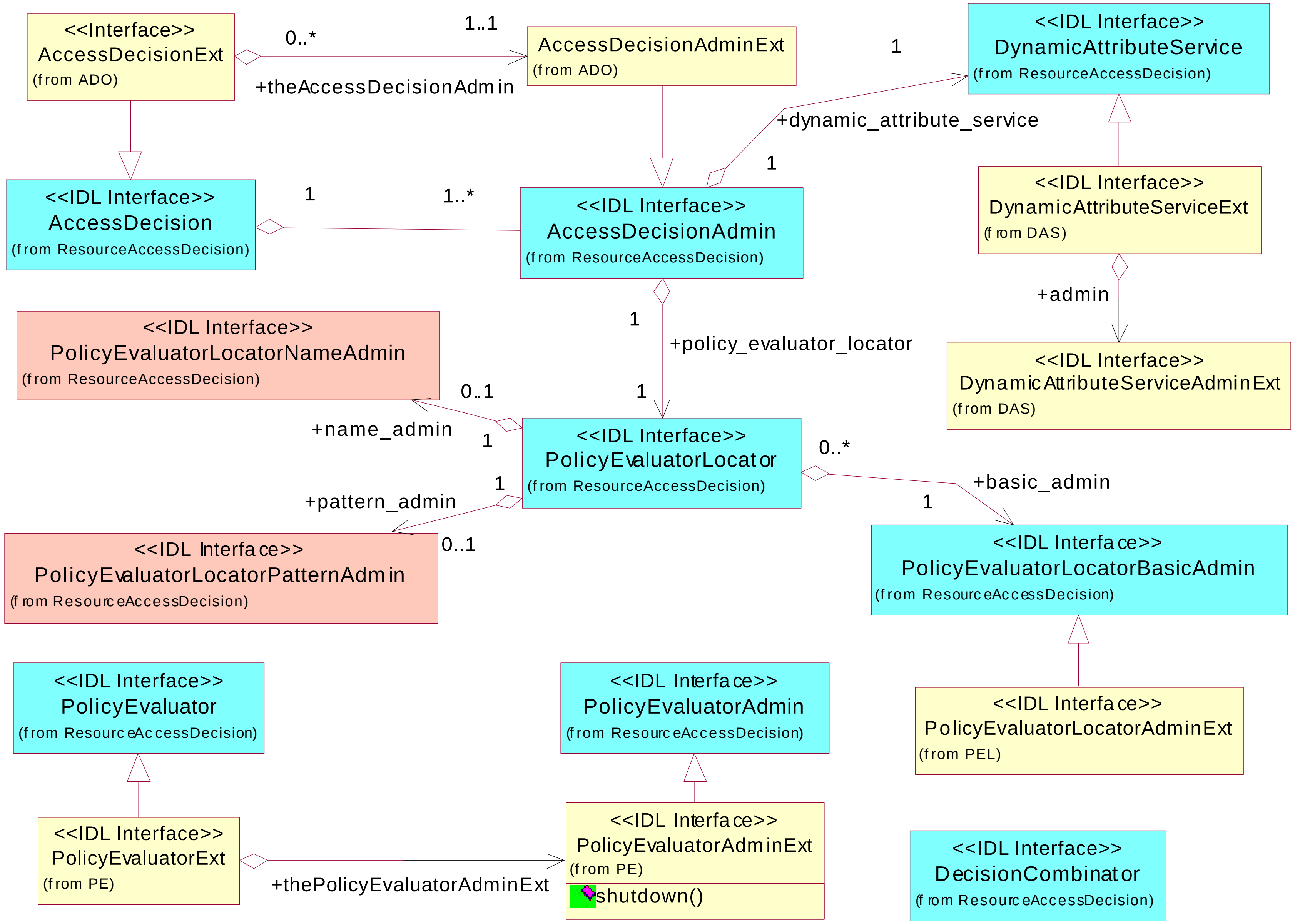
RAD Specification: Component Collaboration


an Application System
6:
1: access_allowed(ResourceName, Operation, AttributeList)
a Locator : Policy EvaluatorLocator
an Access Decision Object : AccessDecision
2: get_policy_decision_evaluators(ResourceName)
4: combine_decisions(ResourceName, Operation, AttributeList, PolicyEvaluatorList)
a Combinator : DecisionCombinator
3: get_dynamic_attributes(AttributeList, ResourceName, Operation)
5: * evaluate(ResourceName, Operation, AttributeList)
an Attribute Service : DynamicAttributeService
an Evaluator : PolicyEvaluator

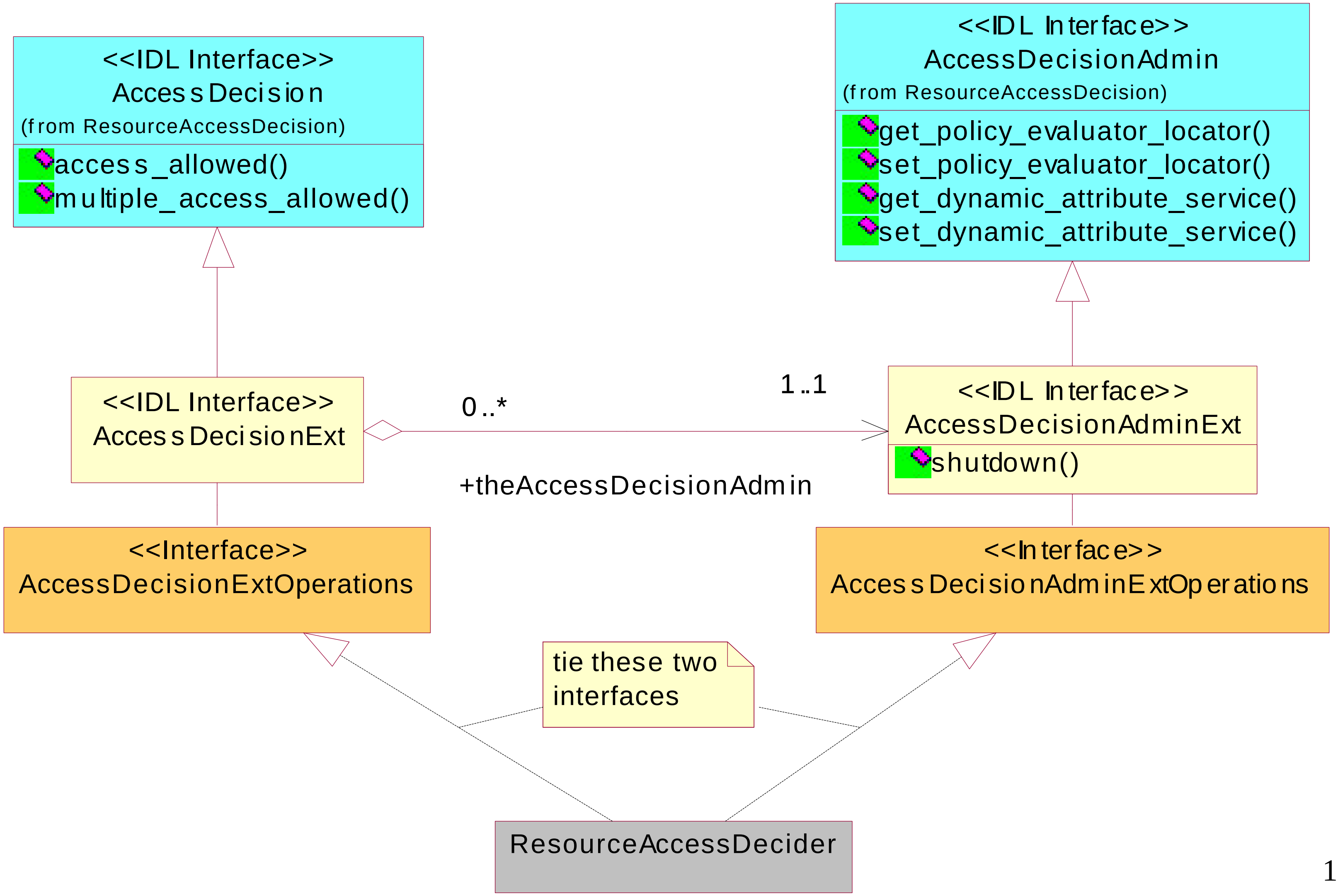
Resource Access Decision Specification Overview


an Application System
an Access Decision Object : AccessDecision
a Locator : Policy EvaluatorLocator
an Attribute Service : DynamicAttributeService
a Combinator : DecisionCombinator
an Evaluator : PolicyEvaluator
access_allowed(ResourceName, Operation, AttributeList)
get_policy_decision_evaluators(ResourceName)
get_dynamic_attributes(AttributeList, ResourceName, Operation)
combine_decisions(ResourceName, Operation, AttributeList, PolicyEvaluatorList)
* evaluate(ResourceName, Operation, AttributeList)

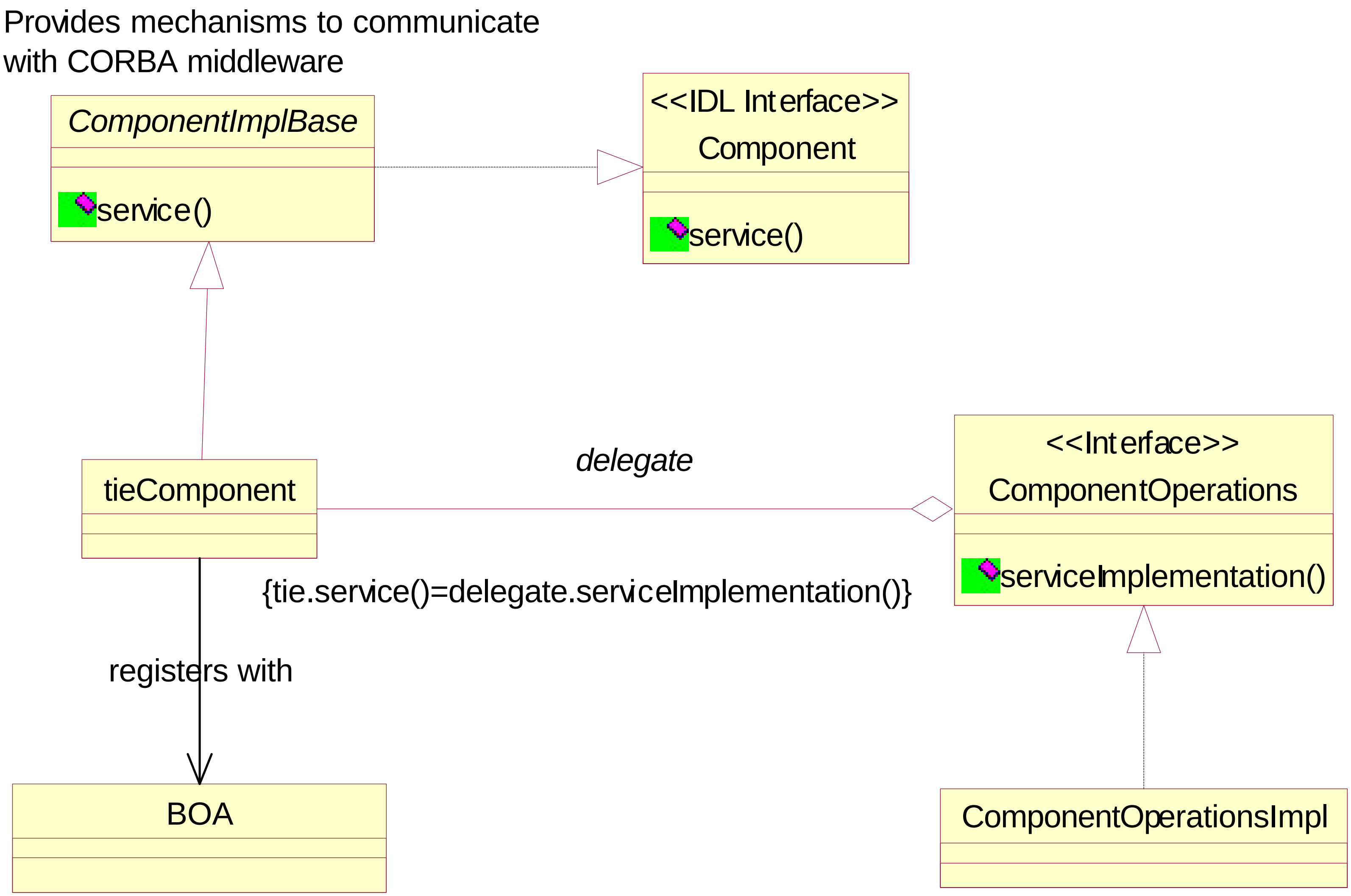
RAD Interfaces



Access Decision Object



Tie Approach



Provides mechanisms to communicate with CORBA middleware

ComponentImplBase

service()

<<IDL Interface>>

Component

service()

tieComponent

delegate

<<Interface>>

ComponentOperations

serviceImplementation()

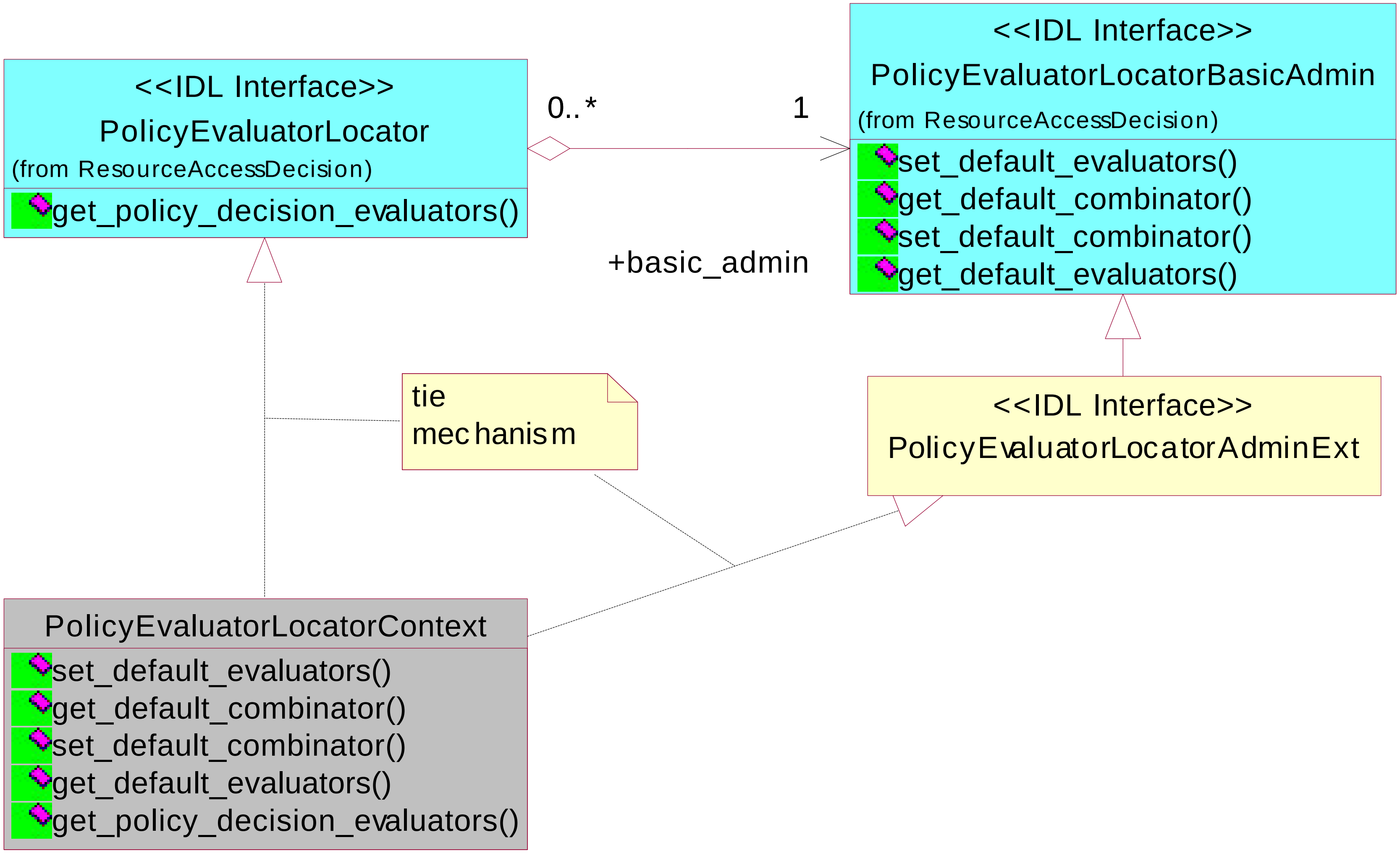
{tie.service()=delegate.serviceImplementation()}

registers with

BOA

ComponentOperationsImpl

Policy Evaluator Locator



<<IDL Interface>>
PolicyEvaluatorLocator
(from ResourceAccessDecision)

- get_policy_decision_evaluators()

<<IDL Interface>>
PolicyEvaluatorLocatorBasicAdmin
(from ResourceAccessDecision)

- set_default_evaluators()
- get_default_combinator()
- set_default_combinator()
- get_default_evaluators()

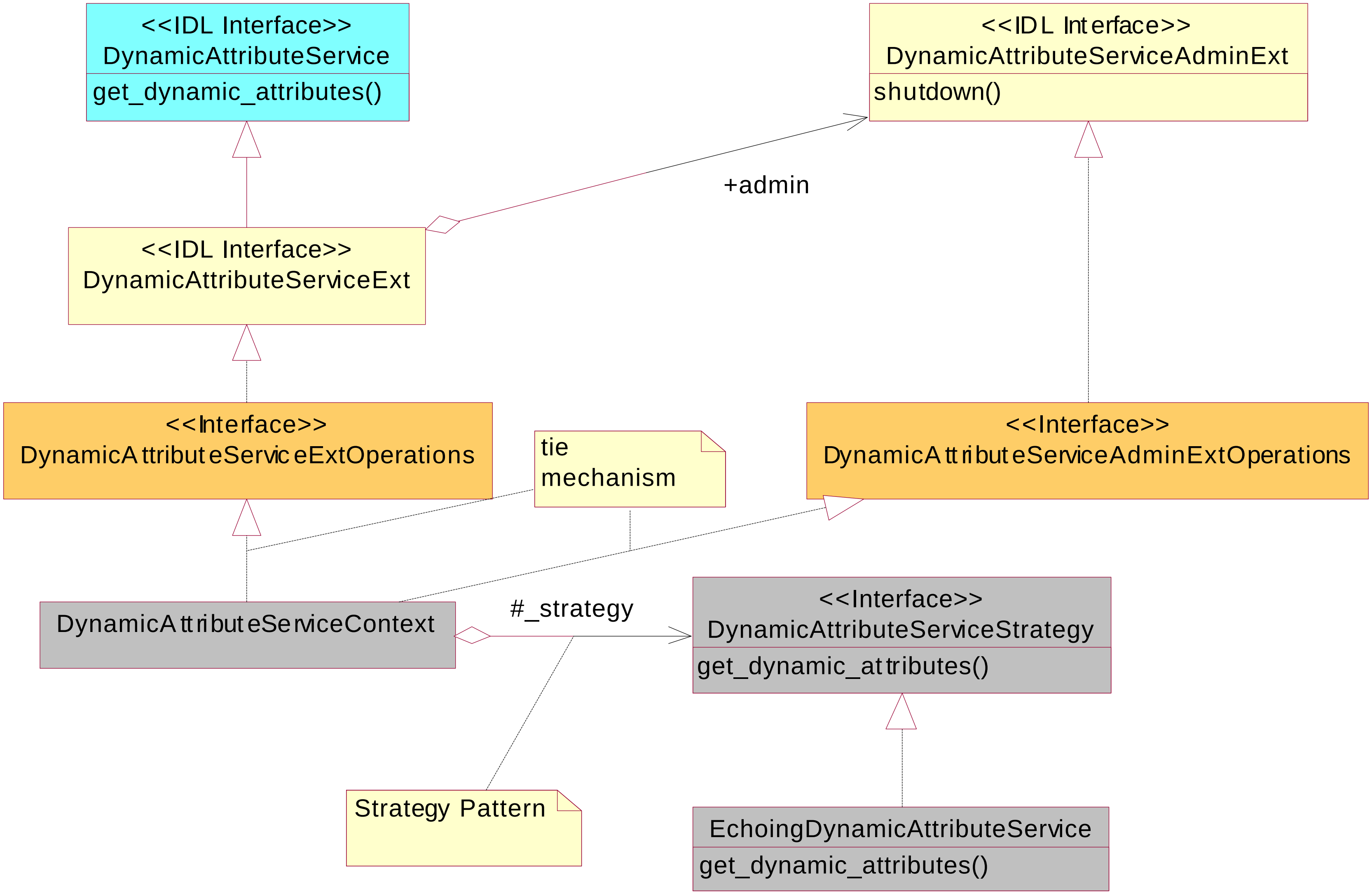
tie mechanism

<<IDL Interface>>
PolicyEvaluatorLocatorAdminExt

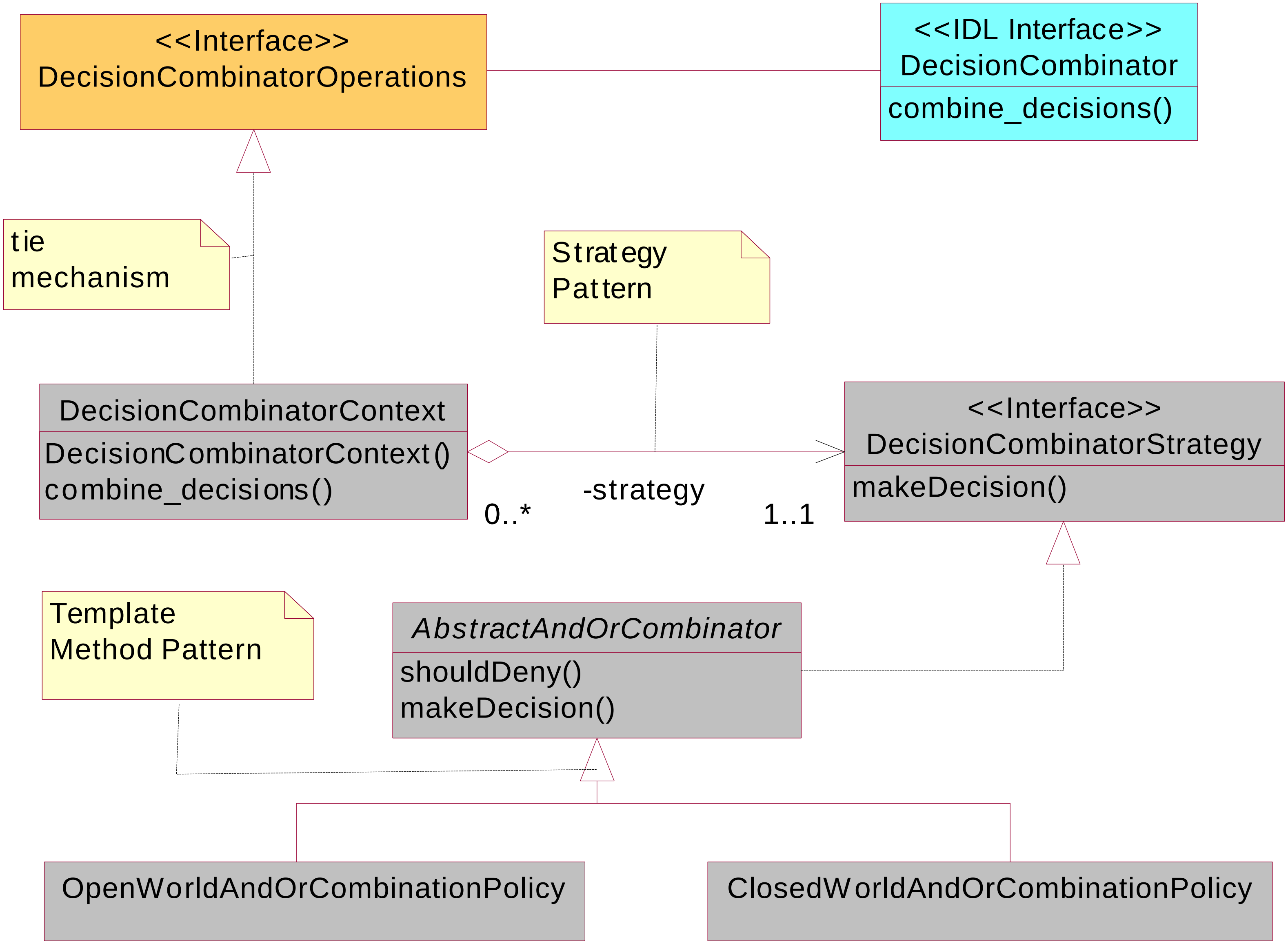
PolicyEvaluatorLocatorContext

- set_default_evaluators()
- get_default_combinator()
- set_default_combinator()
- get_default_evaluators()
- get_policy_decision_evaluators()

Dynamic Attribute Service



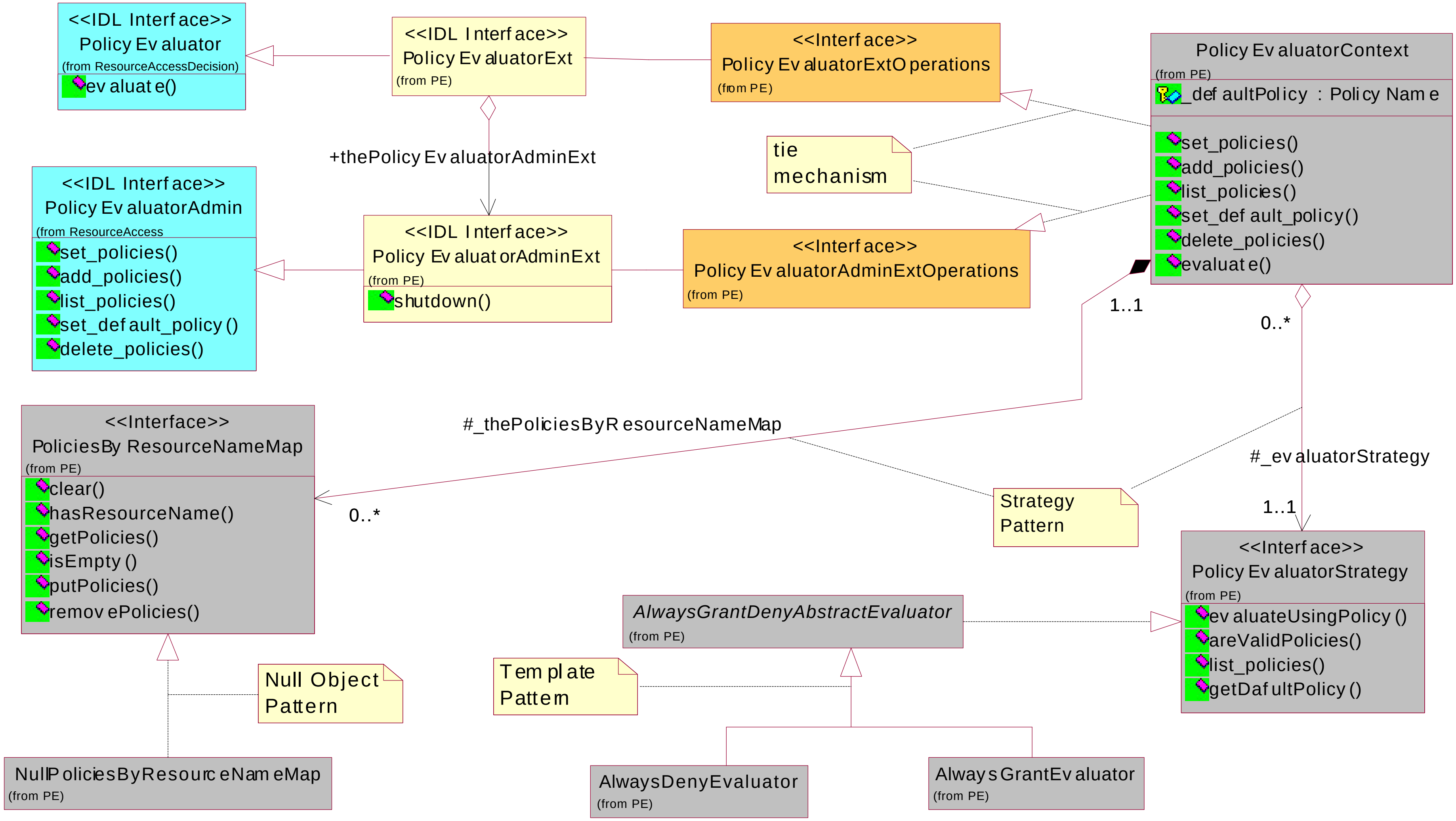
Decision Combinator



grant access if no PE returns "NO"

grant access if all PE's return "YES"

Policy Evaluator



<<IDL Interface>>
PolicyEvaluator
(from ResourceAccessDecision)
evaluate()

<<IDL Interface>>
PolicyEvaluatorExt
(from PE)

<<Interface>>
PolicyEvaluatorExtOperations
(fom PE)

PolicyEvaluatorContext
(from PE)
_defaultPolicy : PolicyName
set_policies()
add_policies()
list_policies()
set_default_policy()
delete_policies()
evaluate()

+thePolicyEvaluatorAdminExt

tie mechanism

<<IDL Interface>>
PolicyEvaluatorAdmin
(from ResourceAccess
set_policies()
add_policies()
list_policies()
set_default_policy()
delete_policies()

<<IDL Interface>>
PolicyEvaluatorAdminExt
(from PE)
shutdown()

<<Interface>>
PolicyEvaluatorAdminExtOperations
(from PE)

1..1

0..*

#_thePoliciesByResourceNameMap

#_evaluatorStrategy

<<Interface>>
PoliciesByResourceNameMap
(from PE)
clear()
hasResourceName()
getPolicies()
isEmpty()
putPolicies()
removePolicies()

0..*

Strategy Pattern

1..1

<<Interface>>
PolicyEvaluatorStrategy
(from PE)
evaluateUsingPolicy()
areValidPolicies()
list_policies()
getDafultPolicy()

AlwaysGrantDenyAbstractEvaluator
(from PE)

Null Object Pattern

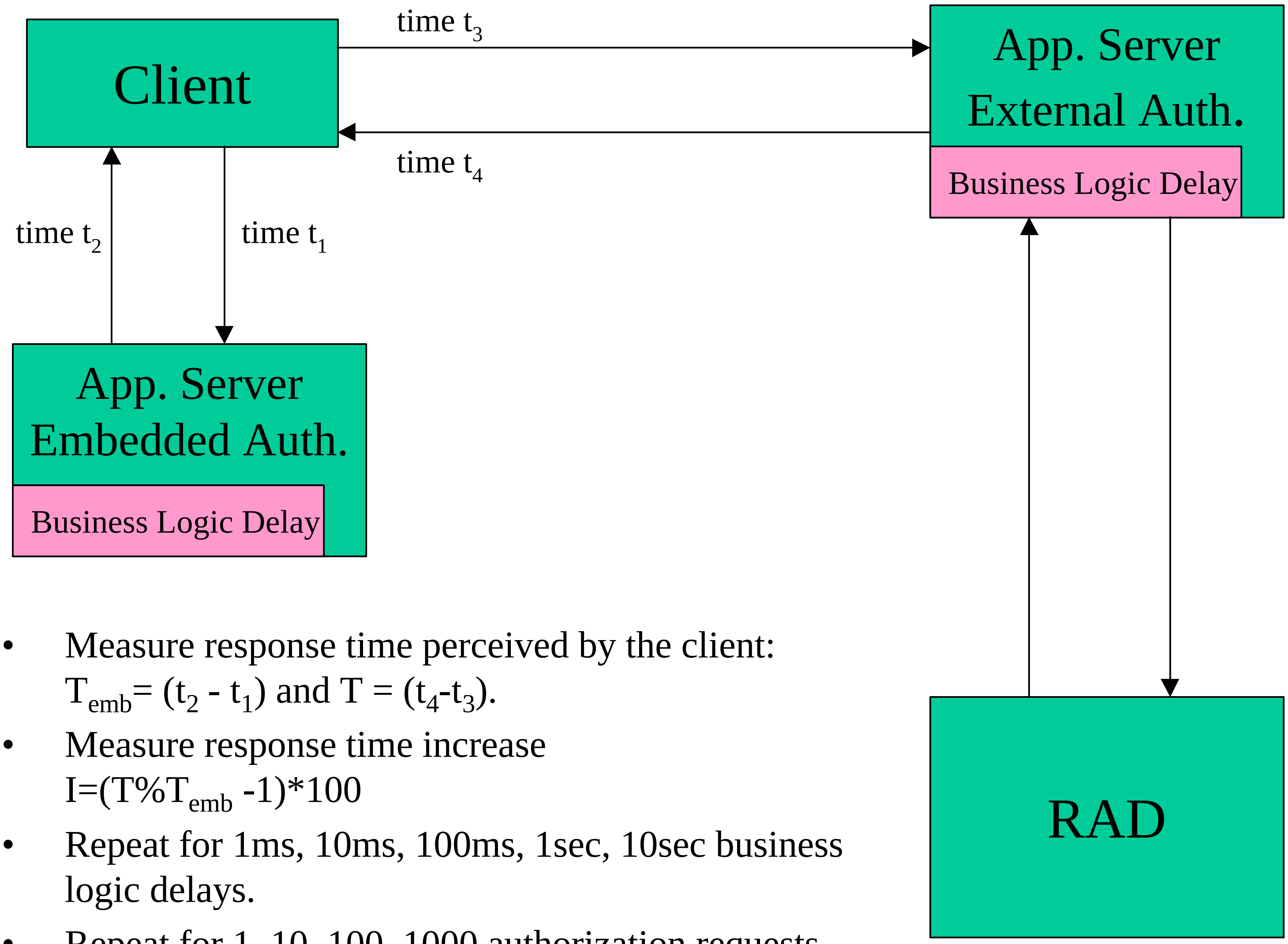
Template Pattern

NullPoliciesByResourceNameMap
(from PE)

AlwaysDenyEvaluator
(from PE)

AlwaysGrantEvaluator
(from PE)

Conducting Performance Measurements

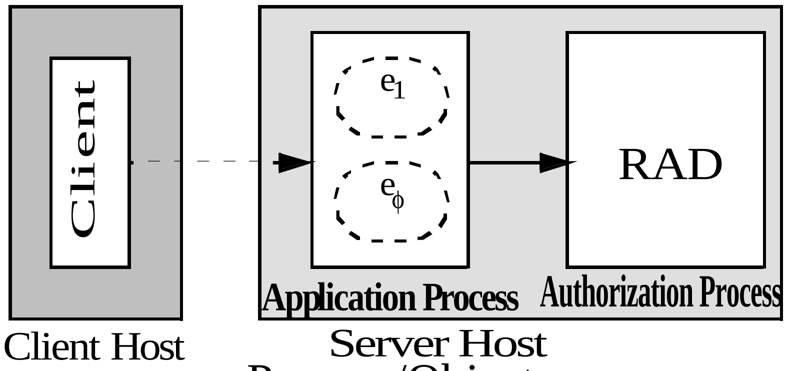


- Measure response time perceived by the client: $T_{emb}= (t_2 - t_1)$ and $T = (t_4-t_3)$.
- Measure response time increase $I=(T\%T_{emb} -1)*100$
- Repeat for 1ms, 10ms, 100ms, 1sec, 10sec business logic delays.
- Repeat for 1, 10, 100, 1000 authorization requests.
- Repeat for different configurations.

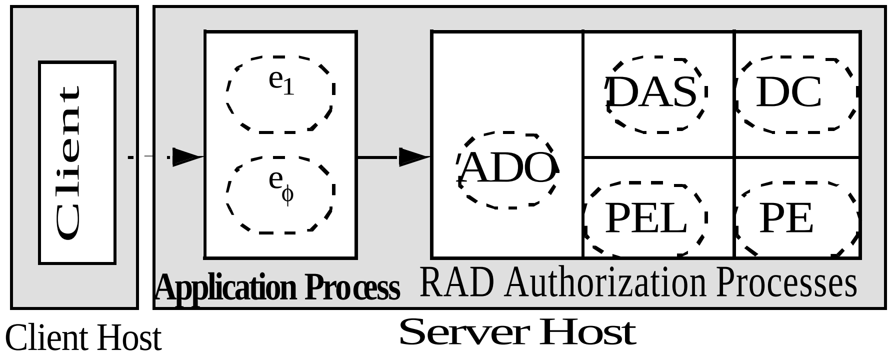
Test Configurations

Boundaries crossed: Application -> RAD/RAD Components

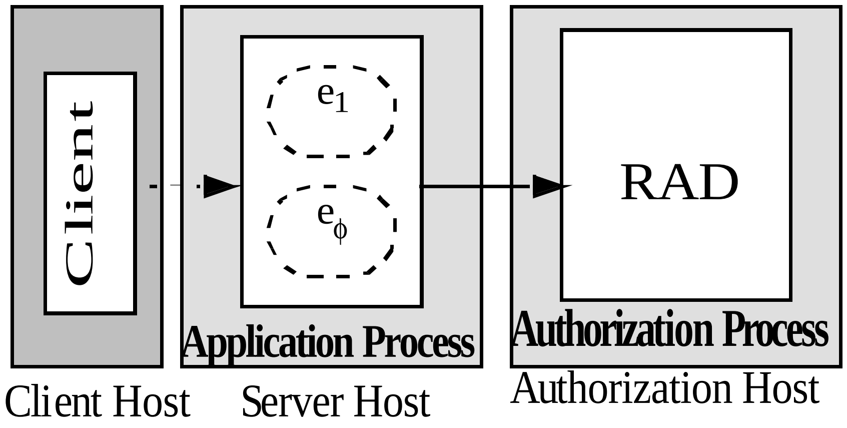
Host=ORB+network; Process=ORB+process; Object=function call



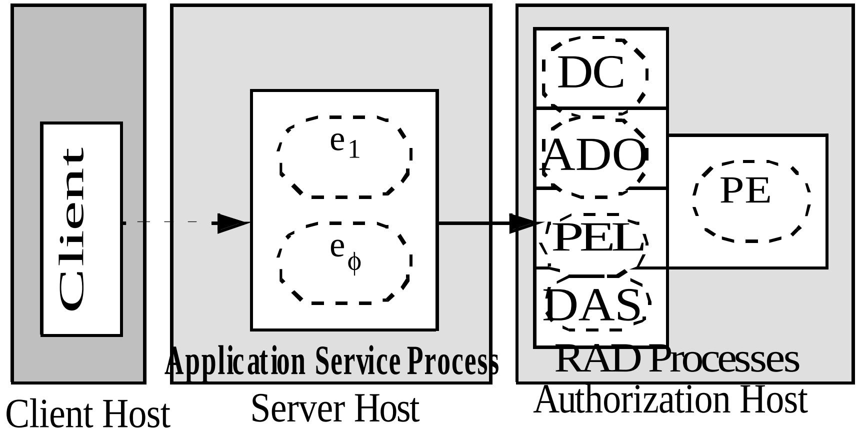
Process/Object



Process/Process

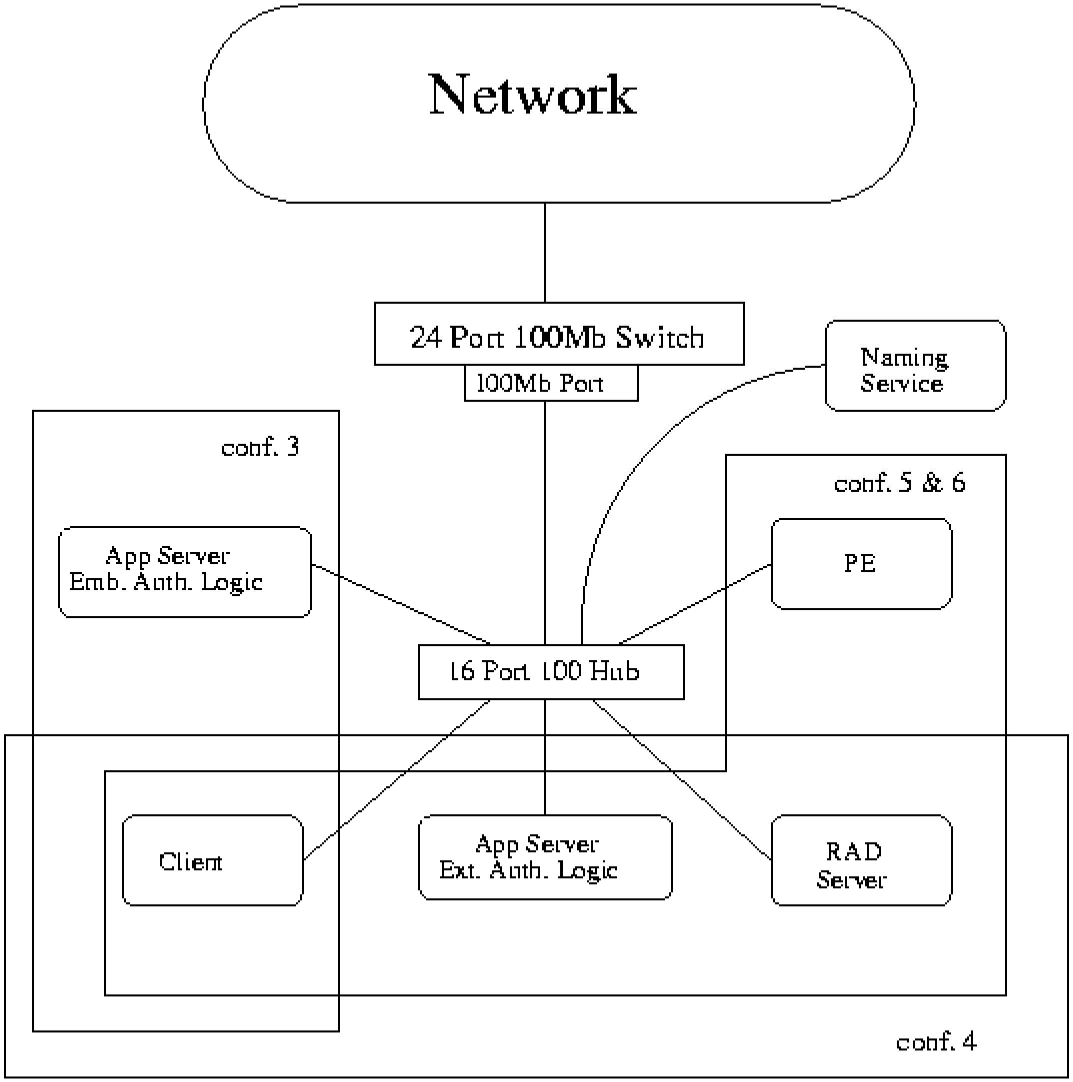


Host/Object

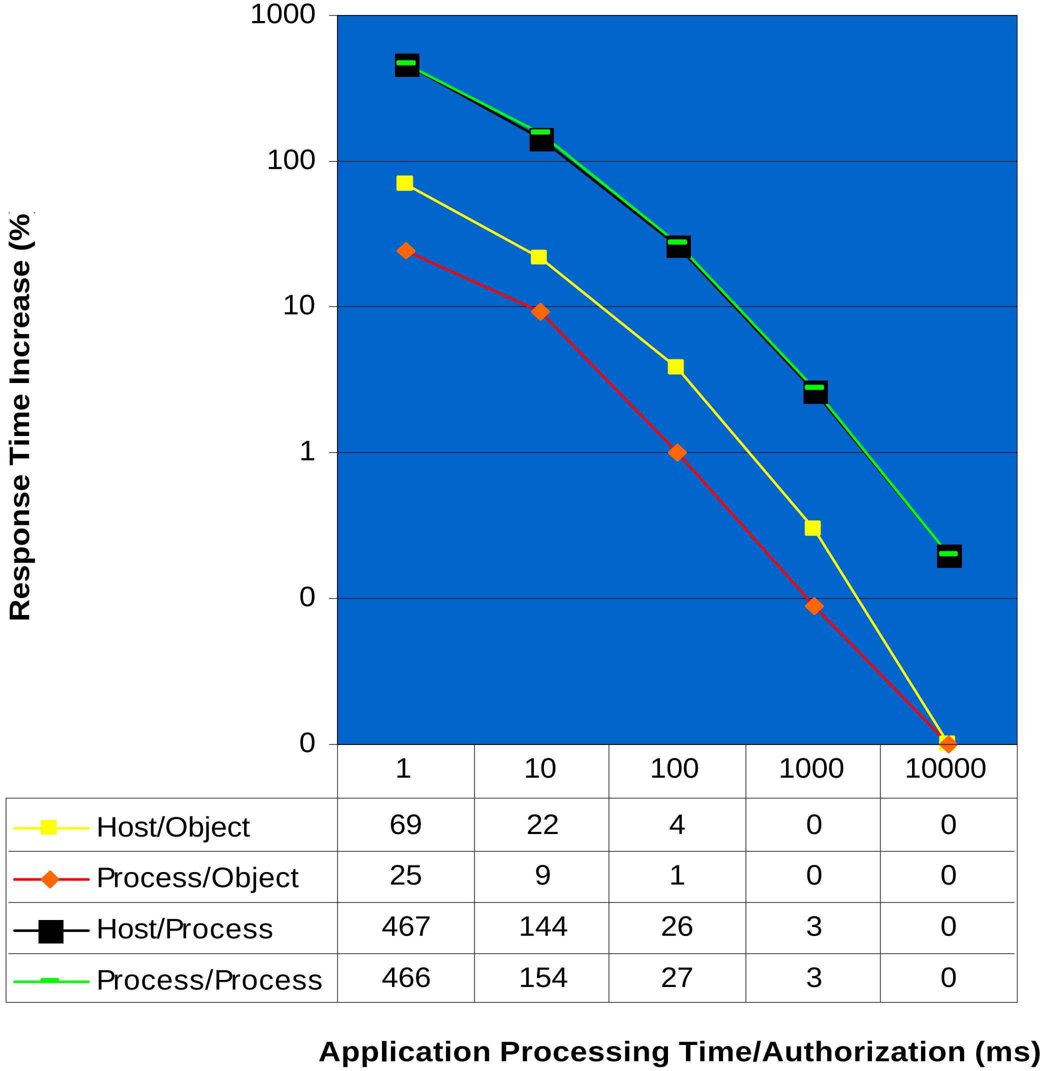


Host/Process

Conducting Performance Measurements



Measurements Results



	1	10	100	1000	10000
🟨 Host/Object	69	22	4	0	0
◆ Process/Object	25	9	1	0	0
■ Host/Process	467	144	26	3	0
▬ Process/Process	466	154	27	3	0

Application Processing Time/Authorization (ms)

$$I = (T \div T_{emb} - 1)*100$$

Factors affecting performance

- process co-location and direct (skipping middleware layers) invocations among RAD components
- host co-location of application and authorization servers

Conclusions

+ Prototype developed
+ Performance measurements collected

• Preparing results for publication

• Doing modeling of RAD and support for advanced access control policies