Extensible Contract Broker for Performance Differentiation

Pedro Furtado, Celso Santos

Univ. of Coimbra

CISUC

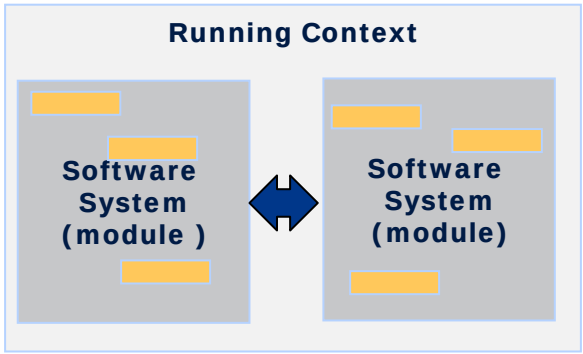
Portugal

The Context of this work

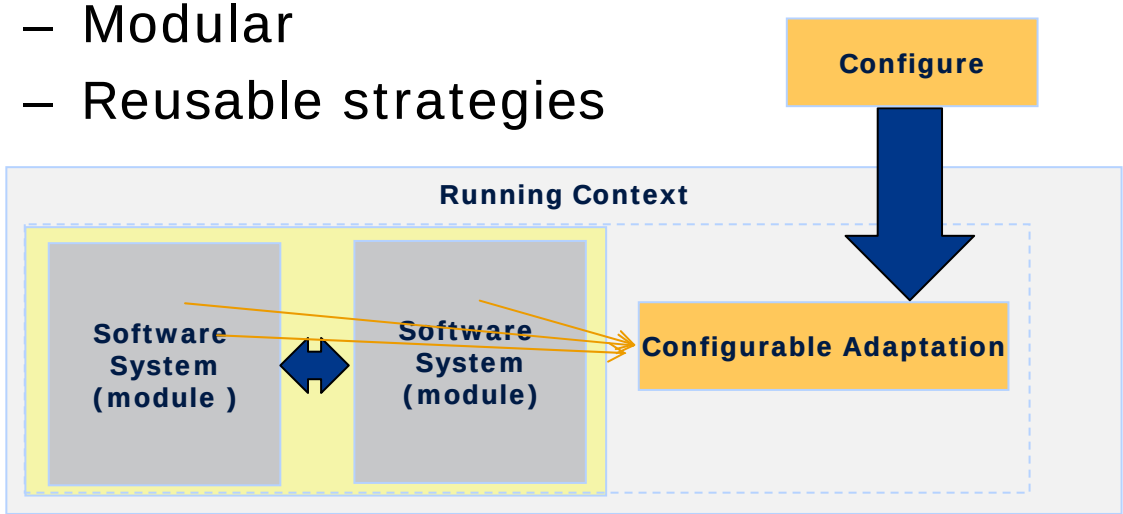
- Initial: to build an external middleware for database backend that:
  - allows the backend to adapt to different runtime contexts
  - provides differentiated service for applications
- Pursuing
  - Strategies for re-configuration and adaptation
  - Platform to define those strategies and apply them
- Currently:
  - Generalizing the concept for generic applications, and to distributed environments with hierarchies of Brokers

Differentiated Service and Adaptation

- Hard-Coded, where needed



- Configurable Broker
  - Modular
  - Reusable strategies



Configurable adaptation

Design-time concerns

Runtime concerns, protocols

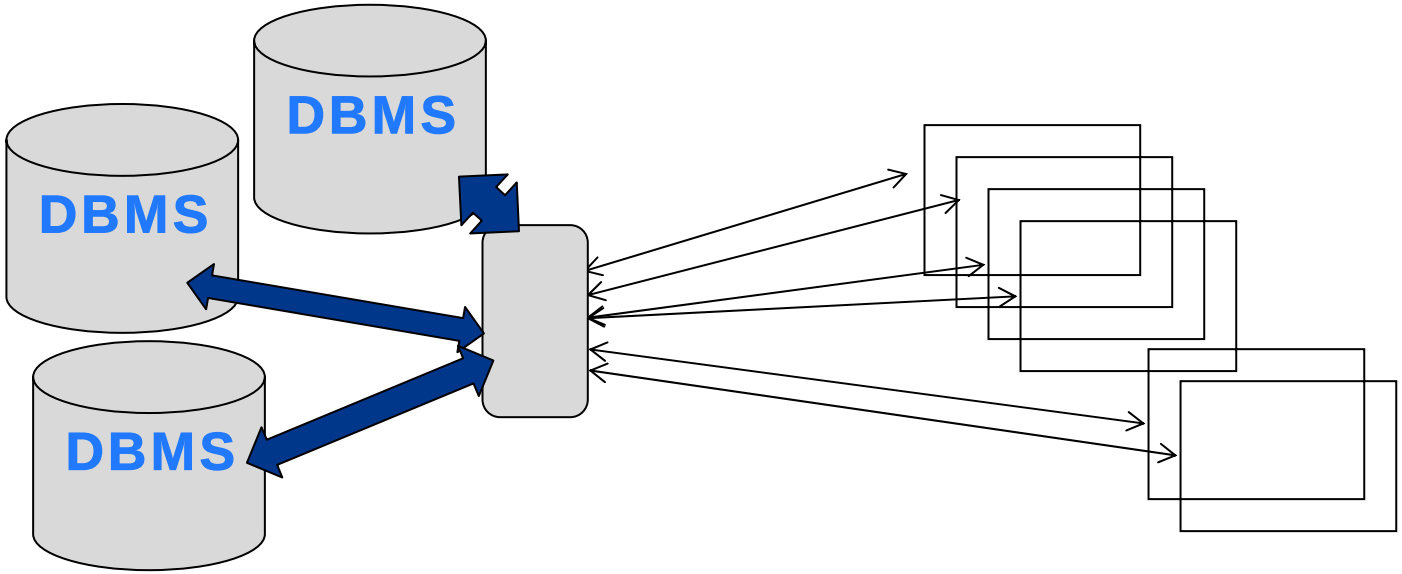
Configuration

Some Related Projects and Issues

- IBM work on autonomic systems
- C-JDBC (sequoia)
  - Clustered JDBC
- RT-QoS
  - QoS Management for Real-Time Databases

- QuO - Quality Objects
  - Framework for providing quality of service (QoS) in network-centric distributed applications
- Rainbow
  - Architecture-Based Self-Adaptation with Reusable Infrastructure

DB-servers and Transactions

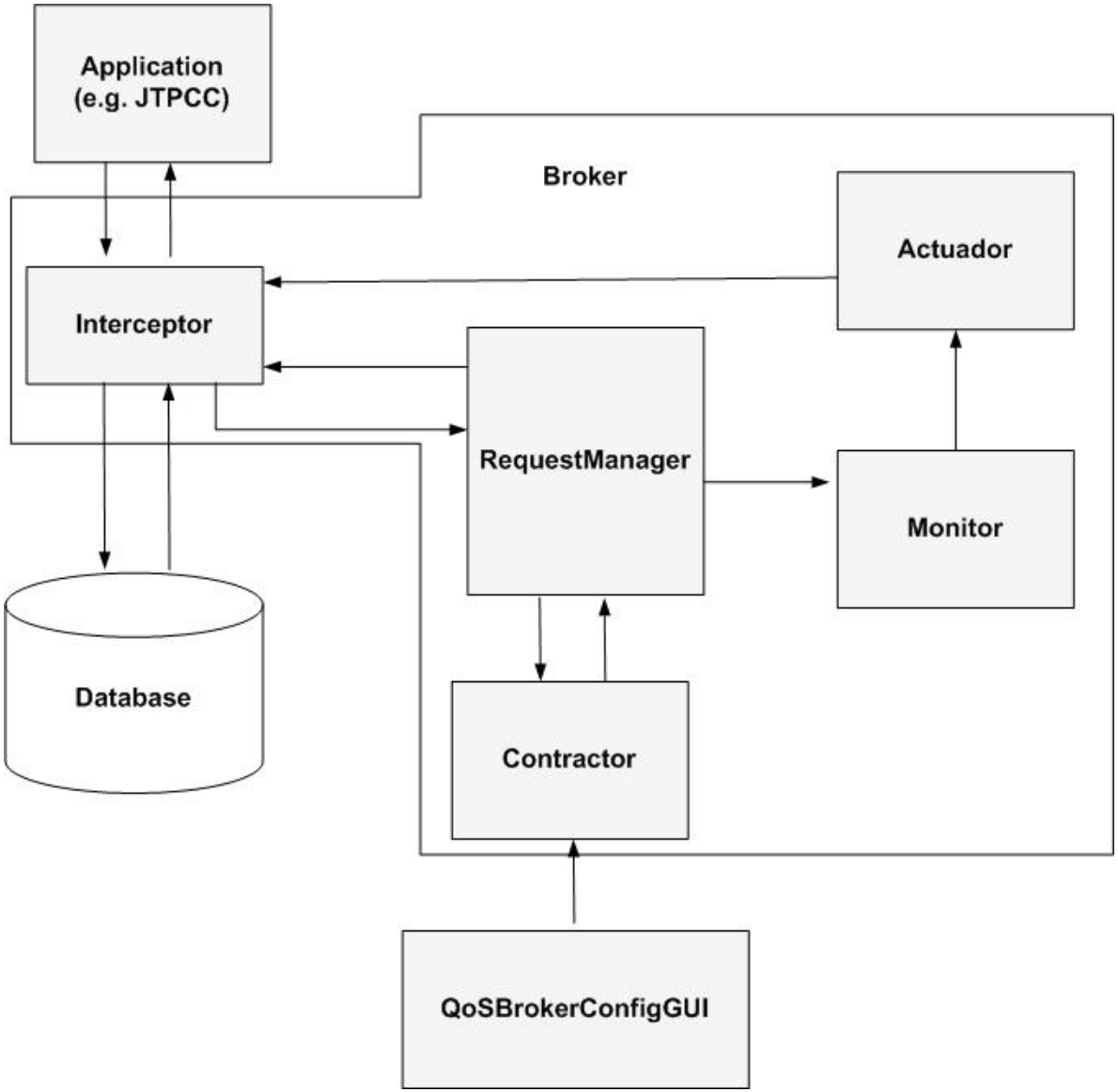


- Configure QoS (e.g. Performance) differently for different client characteristics
- Modify server parameters automatically depending on conditions and rules?
  - Admission control, scheduling
  - Replicate partially, optimize replication
  - Configure replication and load-balancing parameters depending on context
  - Provide availability guarantees
  - Mobile operation

Some Broker concepts

- Applications can use Broker services, configuring for their targets:
- Targets
  - Application items over which to apply features
- Service Features (SF)
  - service and resource use differentiation strategies (QoS) that the Broker offers
- Contracts
  - Conditions and rules of action for application of service features to targets

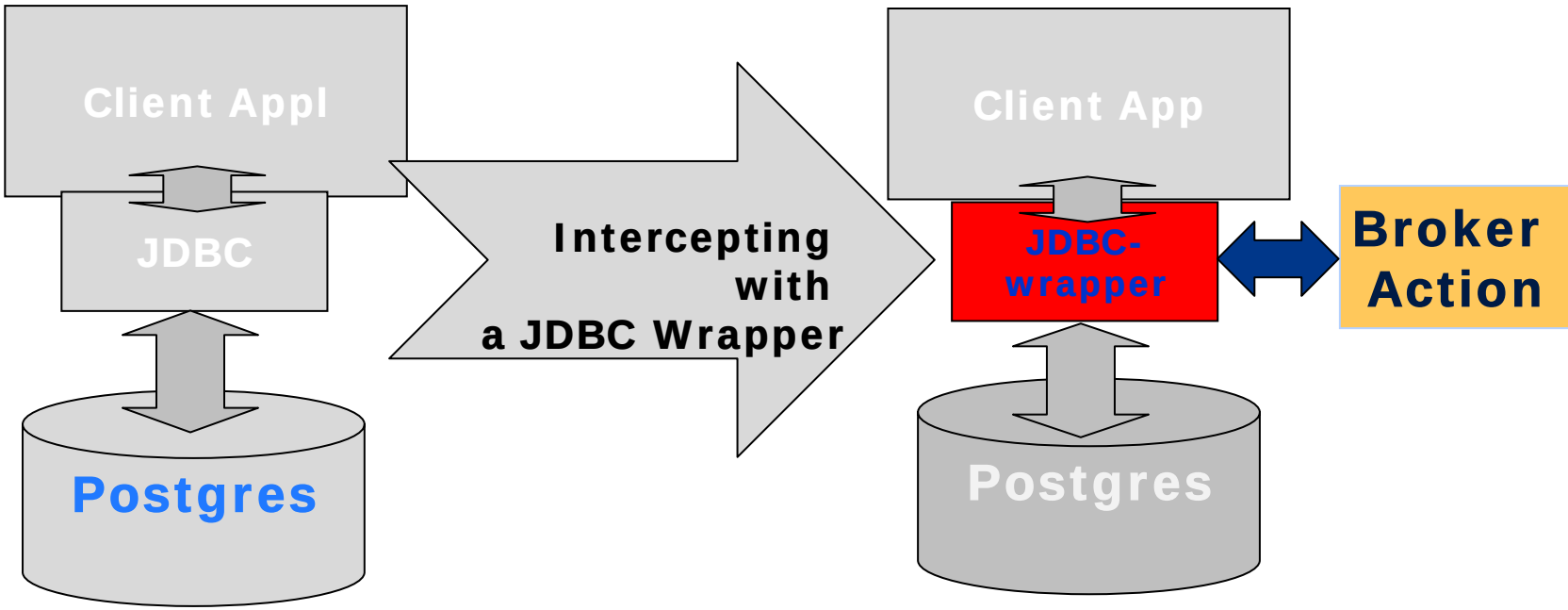
The Basic Broker Design



Brokering Actions

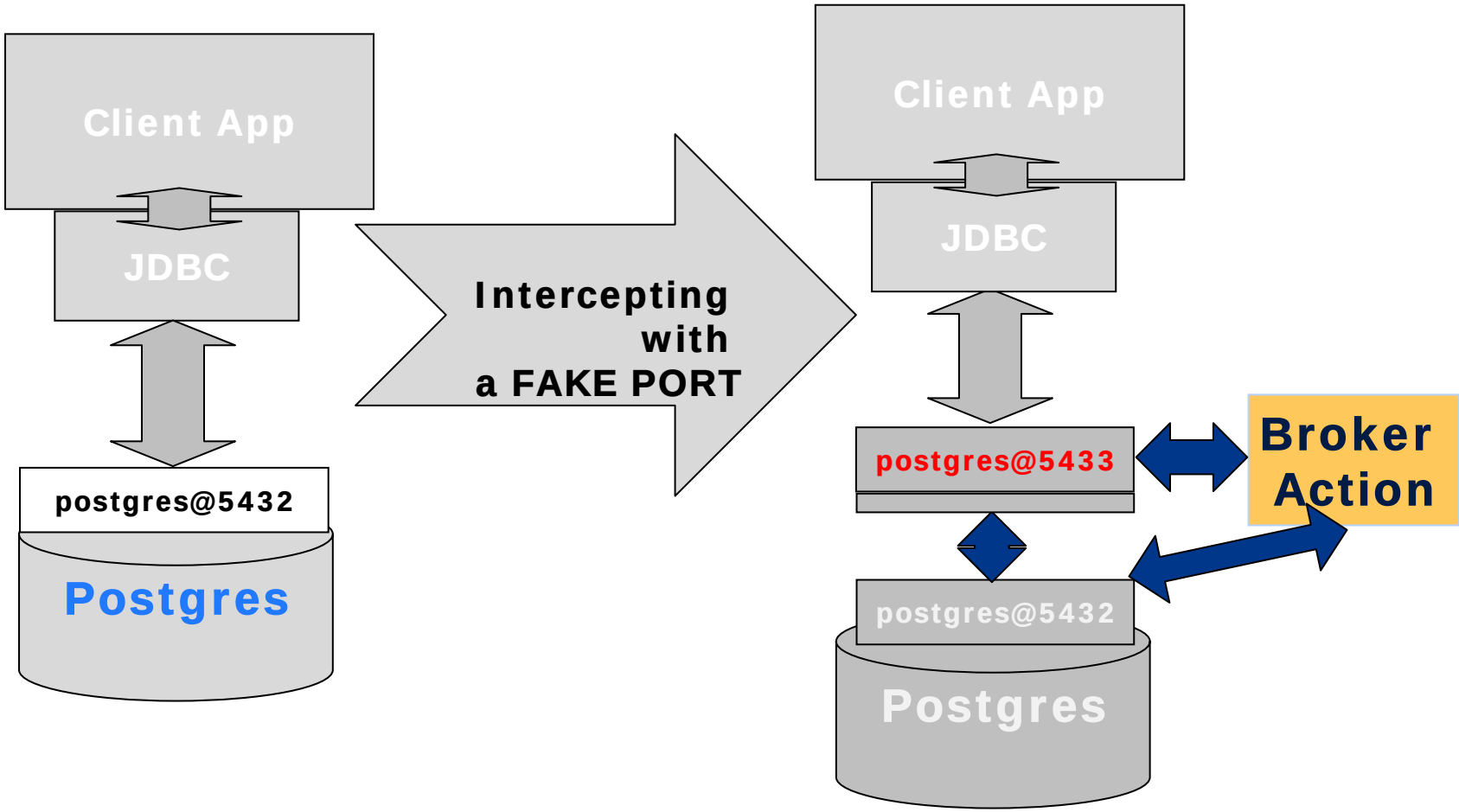
- Tagging of targets
  - Identification of application items that are subject to QoS
- Contracting
  - specification of conditions, rules, SF and parameters that are to be applied to targets
- Intercepting Application Targets
  - Identifying defined targets at runtime, retrieving contracts
- Monitoring and acting
  - Collect runtime statistics, monitor contracts for conditions
  - Managing resources, Acting at runtime

Targets, intercepting for DB backend



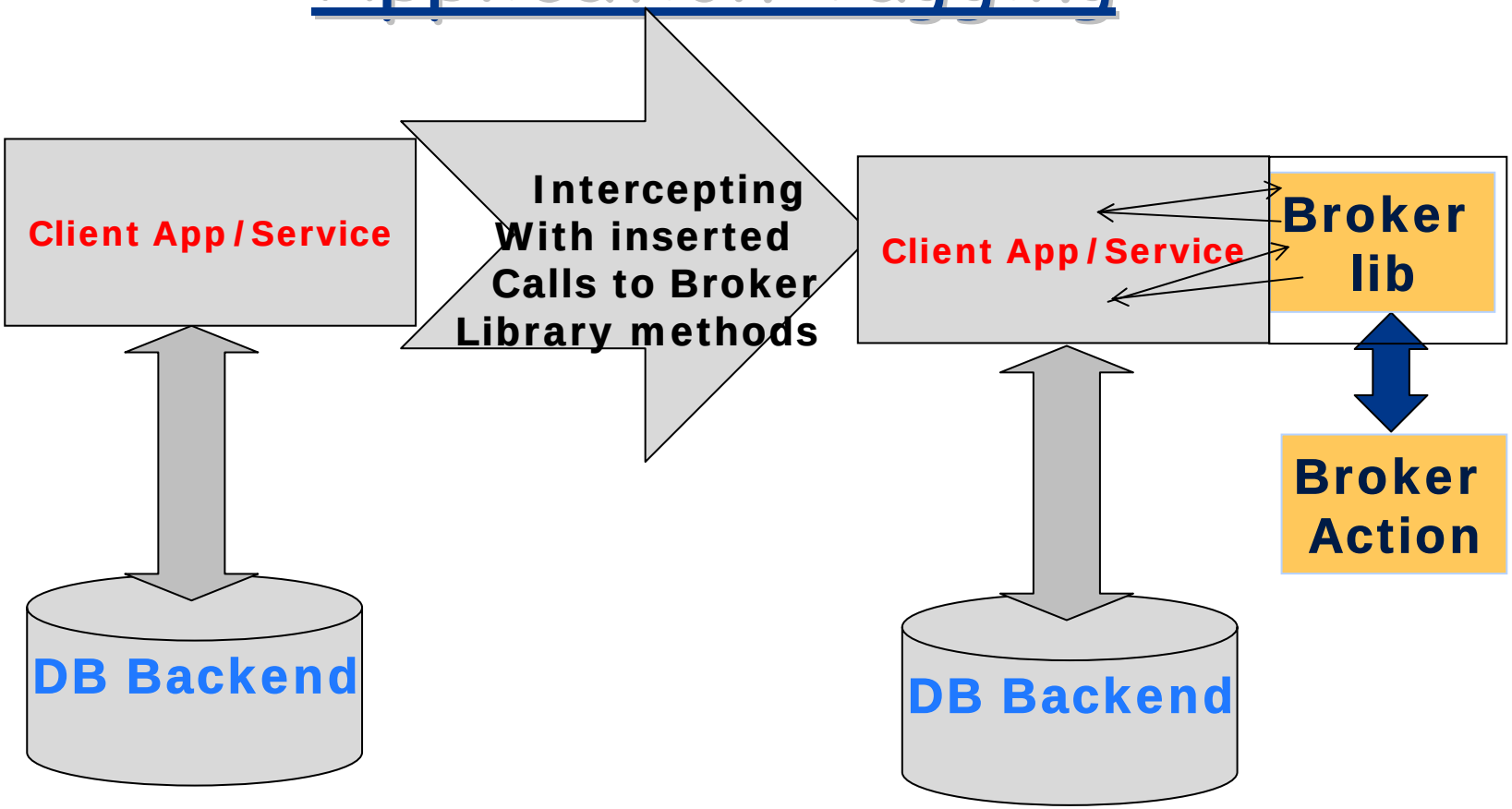
- Broker identifies transactions
  - SQL statement patterns, commits, etc
- Only changes to application:
  - Jdbc library replaced by jdbcWrapper library
  - Connection con; -> ConnectionWrapper con;

Alternative



- Only changes to application:
  - Indicate different port in JDBC connection establishment statement

Application Tagging



- Changes to application:
  - Broker Library methods
  - Calls to Broker methods

Application Tagging

- Naming
  - User/Client indicates methods that are targets
  - May also specify conditions (e.g. variable=value)
- Example

  type=thread

  :user_profile=loggedCustomer

  method=payment
- Tagging (and interception)
  - Tagger tool searches and embeds calls to Broker lib, testing conditions and obtaining variables as well

Contracts

- Identify a target, Specify service features and Condition-Action rules:

```
<CONTRACT>
    <TARGET>payment</TARGET>
    <USE-SERVICE>Premium</USE-SERVICE>
<ACTION-LIST>
    <ACTION>
        <DEADLINE>10000</DEADLINE>
            <DO>Warning</DO>
        <DEADLINE>20000</DEADLINE>
            <DO>Kill</DO>
    </ACTION>
 </ACTION-LIST>
</CONTRACT>
```

target

Service Feature

Condition -Action

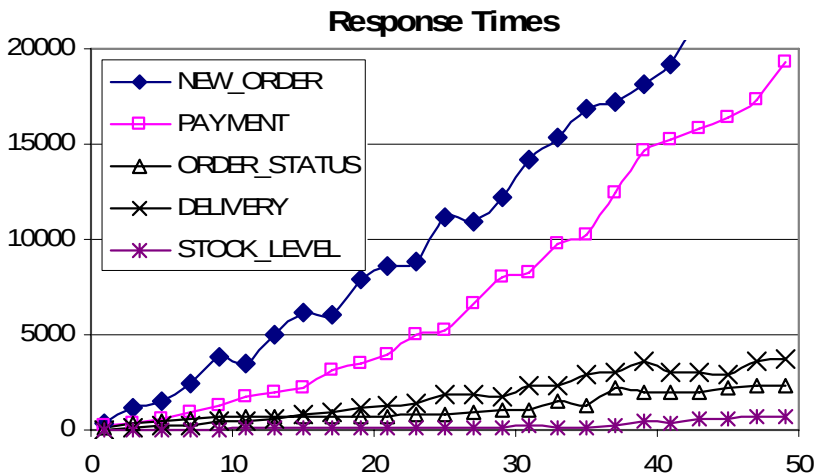
- Target may be any combination of:
  - Individual client thread
  - Method, code block
  - DB session, transaction, SQL statement

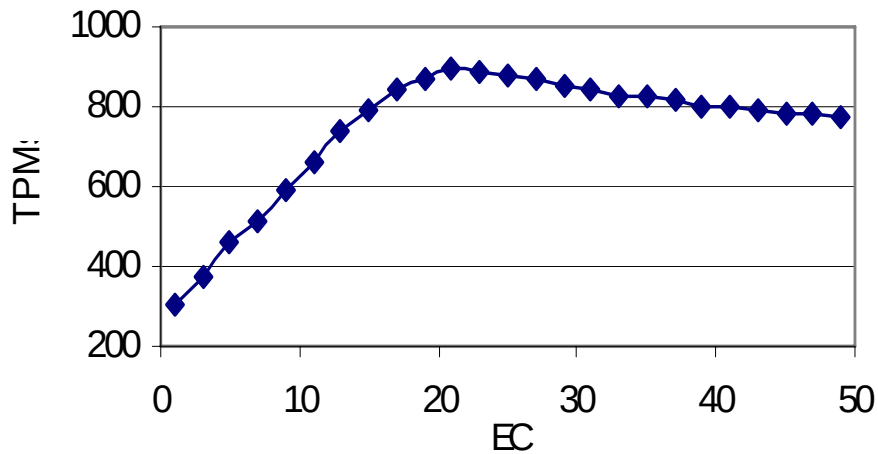
Target Tracking

- Request Manager and Monitor
  - Sessions hashtable
    - maintains information on which target each client thread or DB session is running (if any)
    - Grabs and monitors contracts related to running target
  - Monitorization and Statistics Collection
    - Verifies contract conditions for currently active items
    - Collects statistics for use by adaptable service features
  - Service features
    - Applies service features according to the contracts
  - Action
    - Acts if necessary, according to contract

Statistics, Workload Analysis

- Monitor and characterize target runtime characteristics (e.g. execution times)
- Identify number of simultaneous threads for maximum throughput (for a workload)
- (...)





Service Features

- Service and Resource Use differentiation strategies (QoS)
  - Extensible (can code additional SF)
  - Broker is applicable to any application
  - Given targets, we may wish, for instance:
    - Differentiate requests based on client sessions or combinations of:
      - Method, Client thread, DB session, transaction, SQL statement
    - Deadline hints, for the system to try to meet them
    - Deadline Warning and Notification
    - Schedule and Load Balance with multiple servers
    - Resource reservation
    - Feedback-based admission control (based on the miss rate)
    - Availability differentiation
    - (...)

Service Features (SF) need parameterization ...

- SF configuration parameter
  - e.g. There are three queues (Premium, Fast Service, Normal) with weights (50%, 30%, 20%)
  - e.g. Load balancing may be RR or WRR or LWR
  - Reserve 20% of server for Premium requests
- What SF to use for a specific target:
  - Within contracts
    - SF to apply
    - Specific SF parameters to apply

SF with configuration parameters

```
<SERVICE >
    <NAME>DIFFERENTIATED-SERVICE</NAME>
    <QUEUE-LIST>
        <QUEUE>
            <SERVICENAME>Premium</SERVICENAME>
            <WEIGHT>50</WEIGHT><RESV>20</RESV>
        </ QUEUE>

        < QUEUE>
            <SERVICENAME> Fast Service </SERVICENAME>
            <WEIGHT>30</WEIGHT><RESV>10</RESV>
        </QUEUE>
        <QUEUE>
            <SERVICENAME> Normal </SERVICENAME>
            <WEIGHT>20</WEIGHT><RESV>0</RESV>
        </QUEUE>
    </QUEUE-LIST>
</SERVICE>
```

Examples:

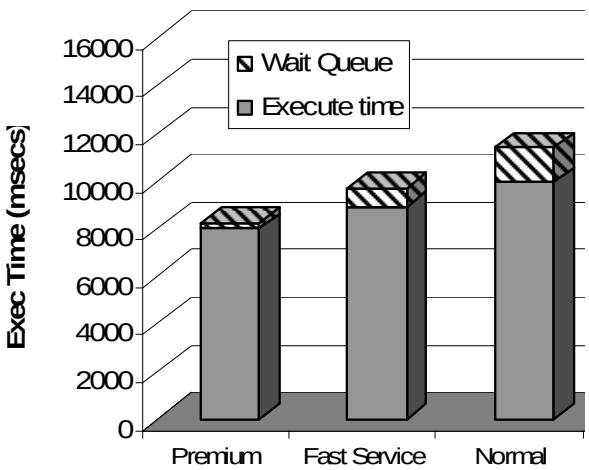
- Number of simultaneous clients limited - queue
- Priority Queuing Example:
  - NEW ORDER transaction on 10 sessions (out of 90) runs in Priority Queueing

```
<CONTRACT>
        <TARGET>
                <TRANSACTION>NEW-ORDER</TRANSACTION>
                <SQL>ALL</SQL>
                <SESSION>1-10</SESSION>
        </TARGET>
    <USE-SERVICE>PRIORITYQUEUE</USE-SERVICE>
</CONTRACT>
```

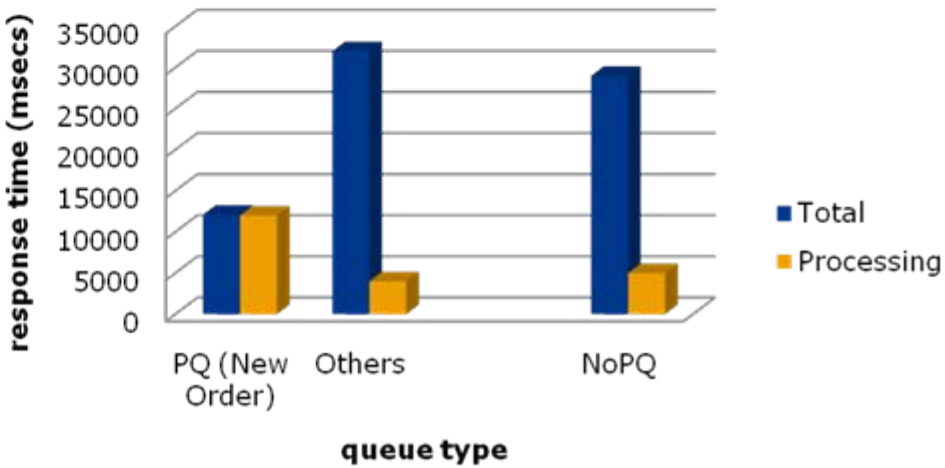
- Weighted Round Robin:
  - 90 terminals with 1/3 of those Premium, 1/3 Fast Service and 1/3 Normal service

- With WRR wait times were divided unevenly as desired
- PQ allowed priority transactions to have small total time in a congested state (where wait times were large)

Weighted Round Robin

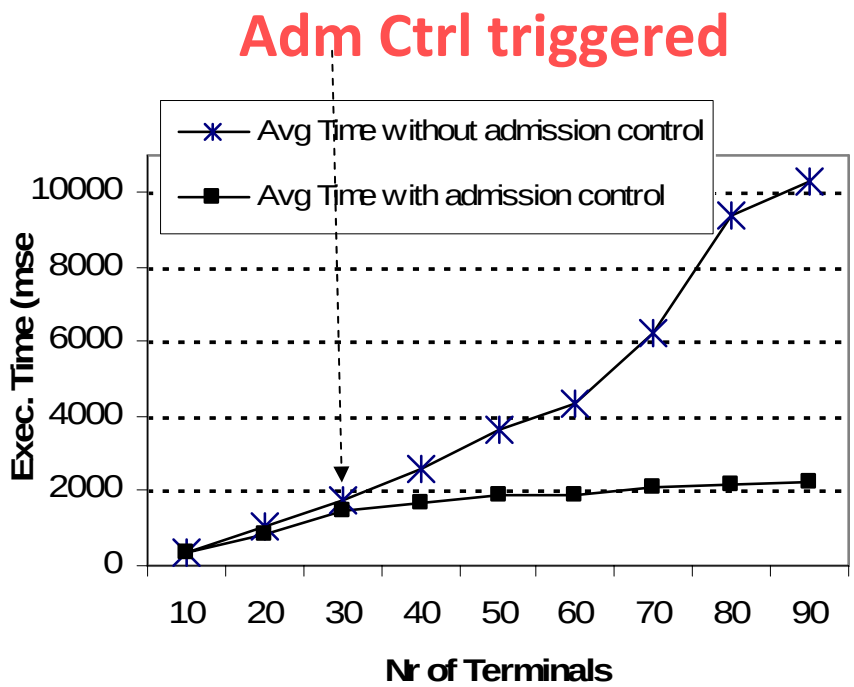


Priority Queueing



Admission Control Rule

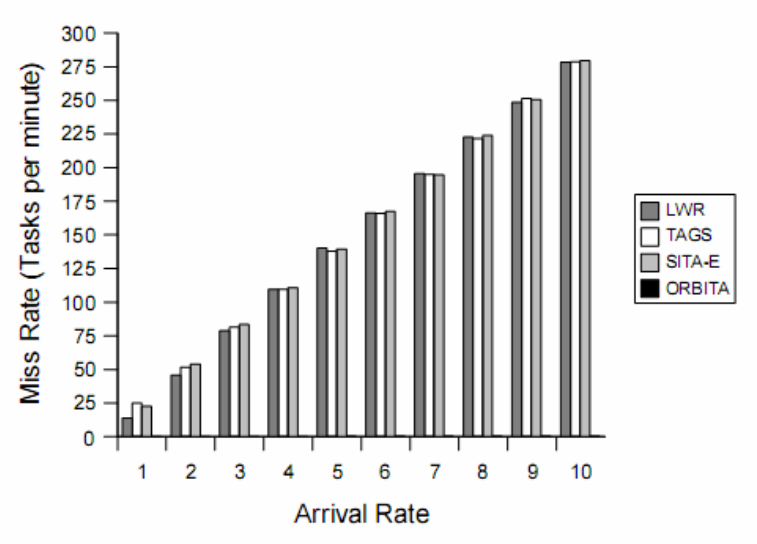
- Restrict admission of new clients when miss rate is above 10% in the last minute



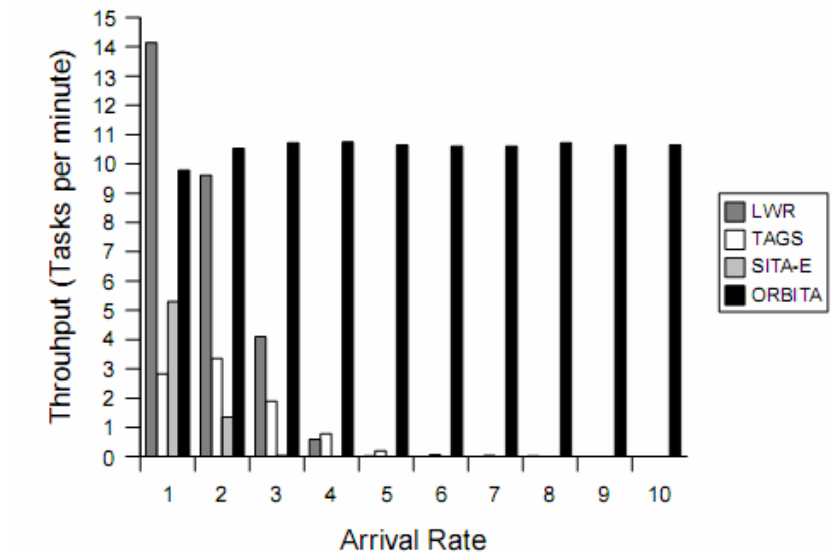
Load-Balancing with runtime constraints

- Load-balancing ORBITA approach:
  - Characterize reponse times of targets (monitor, statistics)
  - Schedule targets differently to servers base on statistics (SF)

Miss Rate



Throughput of big tasks



Current and Future Work

- Adaptation strategies to meet contracts
  - Already have some interesting results in load balancing and admission control
- Applying and extending WSLA in our context
  - We are redefining most of the sintax using WSLA
  - www.research.ibm.com/**wsla**/
- Systematic language for features, architecture elements, concerns and rules of action
  - e.g. RainBow
- Broker Architecture and Service Features to Broker and provide differentiation within services architecture

- Thank You!

- eden.dei.uc.pt/~pnf
- pnf@dei.uc.pt

Broker Overhead

- The Broker overhead varies with options
- Typically it varies between 2% and 10%