Performance Modeling of a JavaEE Component Application Using LQN: a Case Study

Alexander Ufimtsev

Liam Murphy

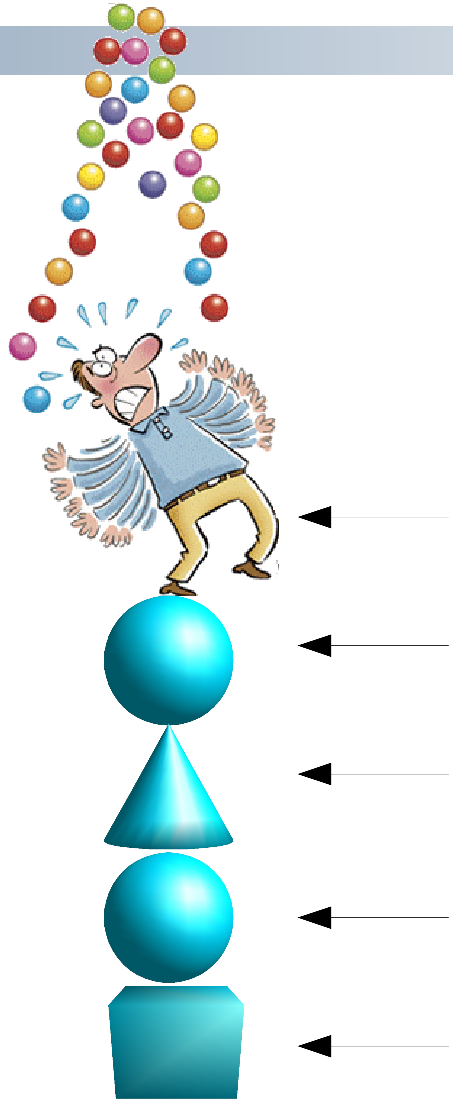
Performance Engineering Lab
School of Computer Science and Informatics
University College Dublin
Ireland

www.perfenglab.com

Motivation

- Software development projects fail (time, budget, QoS, altogether) for multiple causes
- Bad design contributes to approx. 20% of problems in enterprise systems [Ptak *et al*]
- *Performance* analysis should be done at the early stages of the design to avoid failures
- However, it is difficult to check outside proper test environment

Motivation



Component App

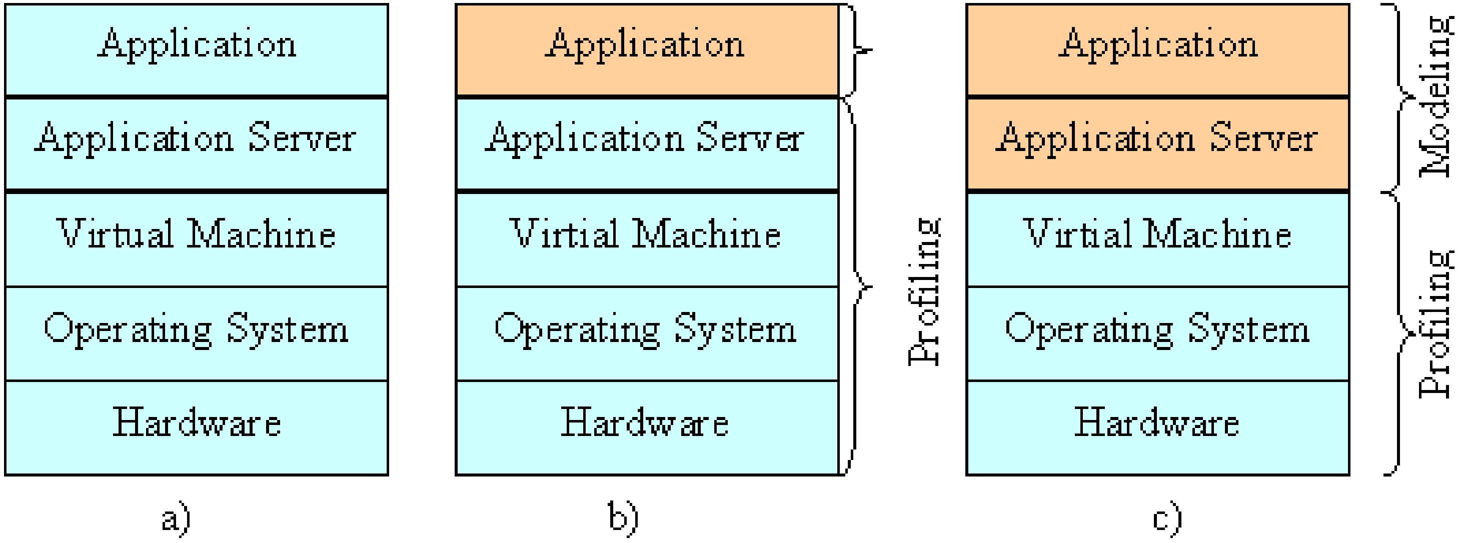
Application Server

Virtual Machine

Operating System

Hardware

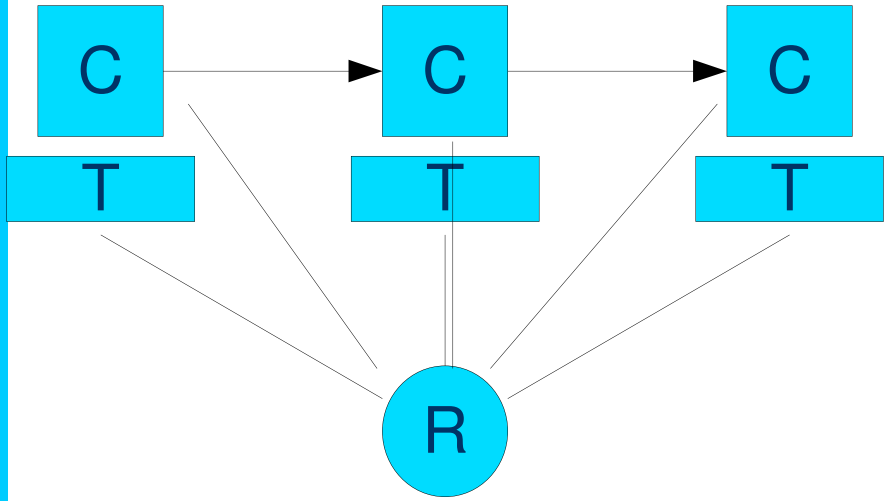
Three types of Approaches



Approach

- Construct a model of real-life application by instantiating the templates and composing them
- Perform measurements on the real running application
- Profile and calibrate the model from app. traces
- Compare model prediction with measurement results

LQN Templates Overview



(C)omponent interaction is augmented with instanciated (T)emplates of container services

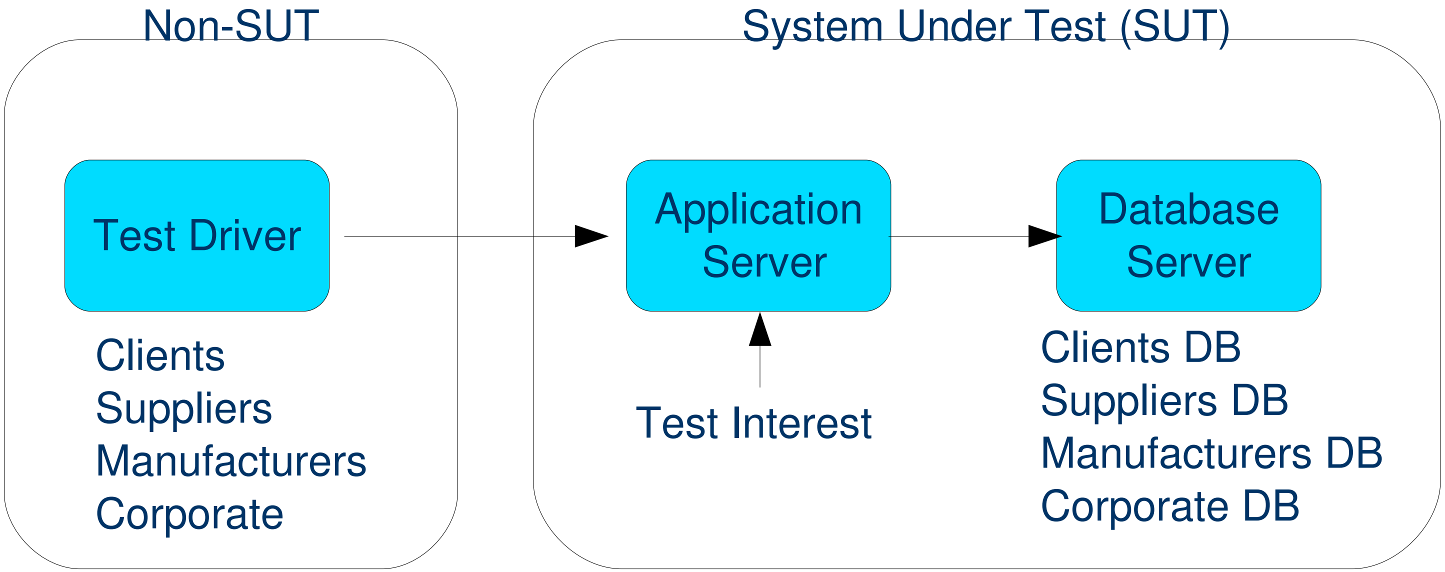
Why LQN?

- LQN (Layered Queuing Network)
  - Is a performance modeling language
  - Models system resources and behaviour in an intuitive way
  - Allows nested software structure and composition with component concepts
  - Captures resource contentions effectively
  - Does not suffer from state explosion problem
  - Provides both Analytical & Simulation solver

A Better Case Study: ECPerf

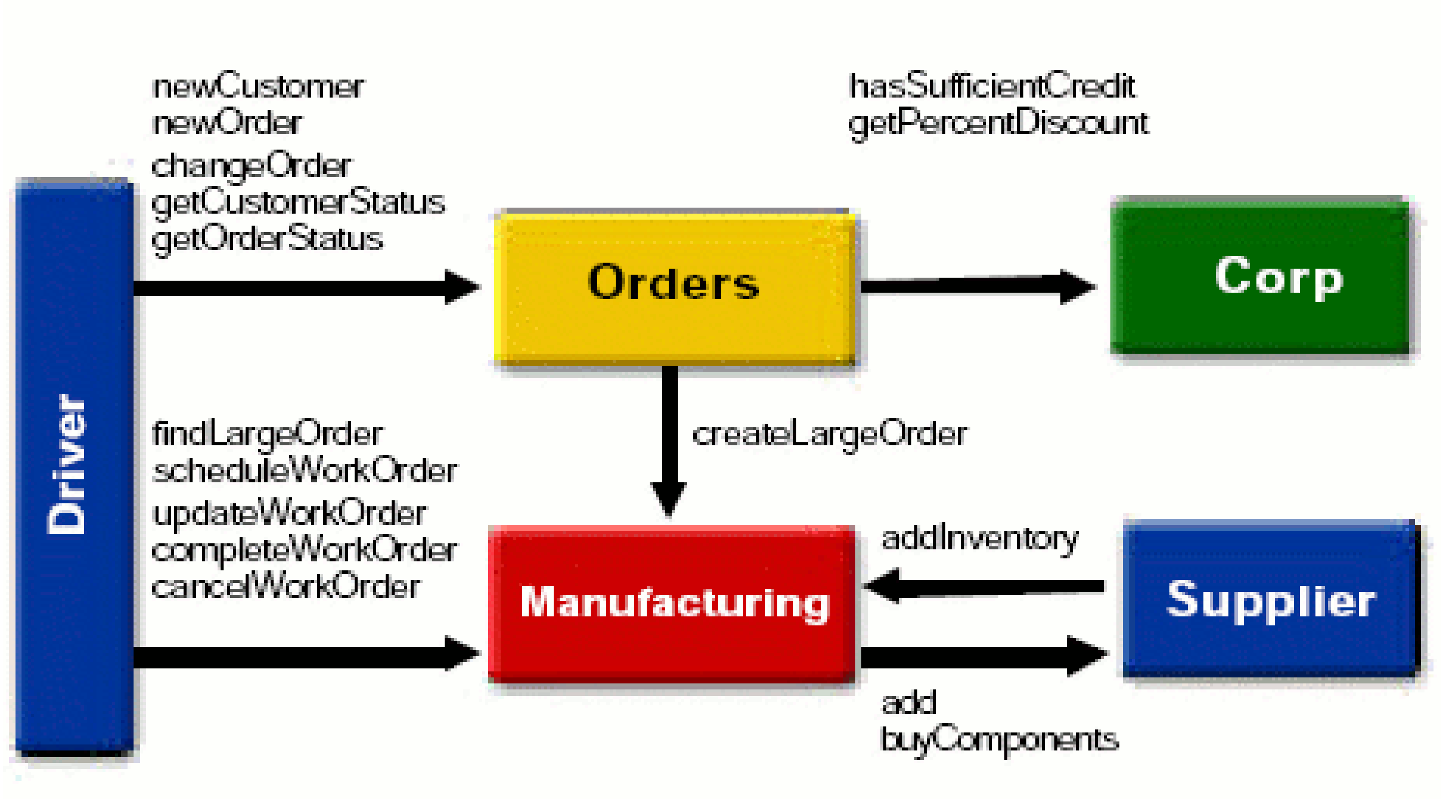
- ECperf is a Enterprise JavaBeans (EJB) benchmark meant to measure the scalability and performance of J2EE servers and containers.
- ECperf stresses the ability of EJB containers to handle the complexities of memory management, connection pooling, passivation/activation, caching, etc.

ECPerf Overview



- ~30 beans, not including helper classes
- 50K LOC

ECPerf Overview Cont'd



© JavaPassion.com

ECPerf Startup Parameters

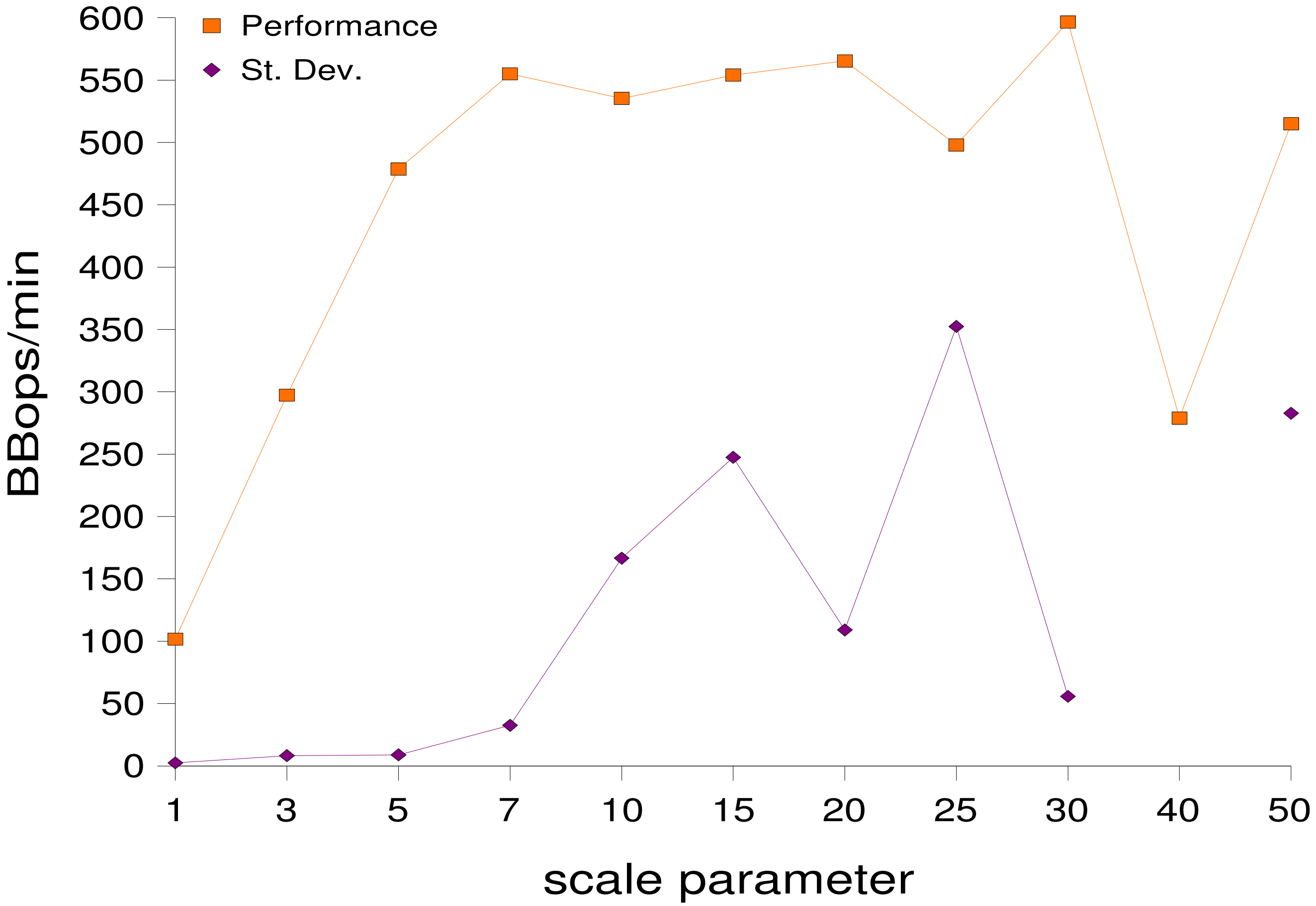
- rampUp = 480, stdyState = 600, rampDown = 180
- runOrderEntry = 1, runMfg = 1
- Transaction rate (txRate) was set from 1 to 50 in different tests. Orders=5*txRate, Manufacturing=3*txRate
  - txRate = 5 (40 threads: 25 order entry, 15 planned line)

Application Profiling and Measurement: Hardware

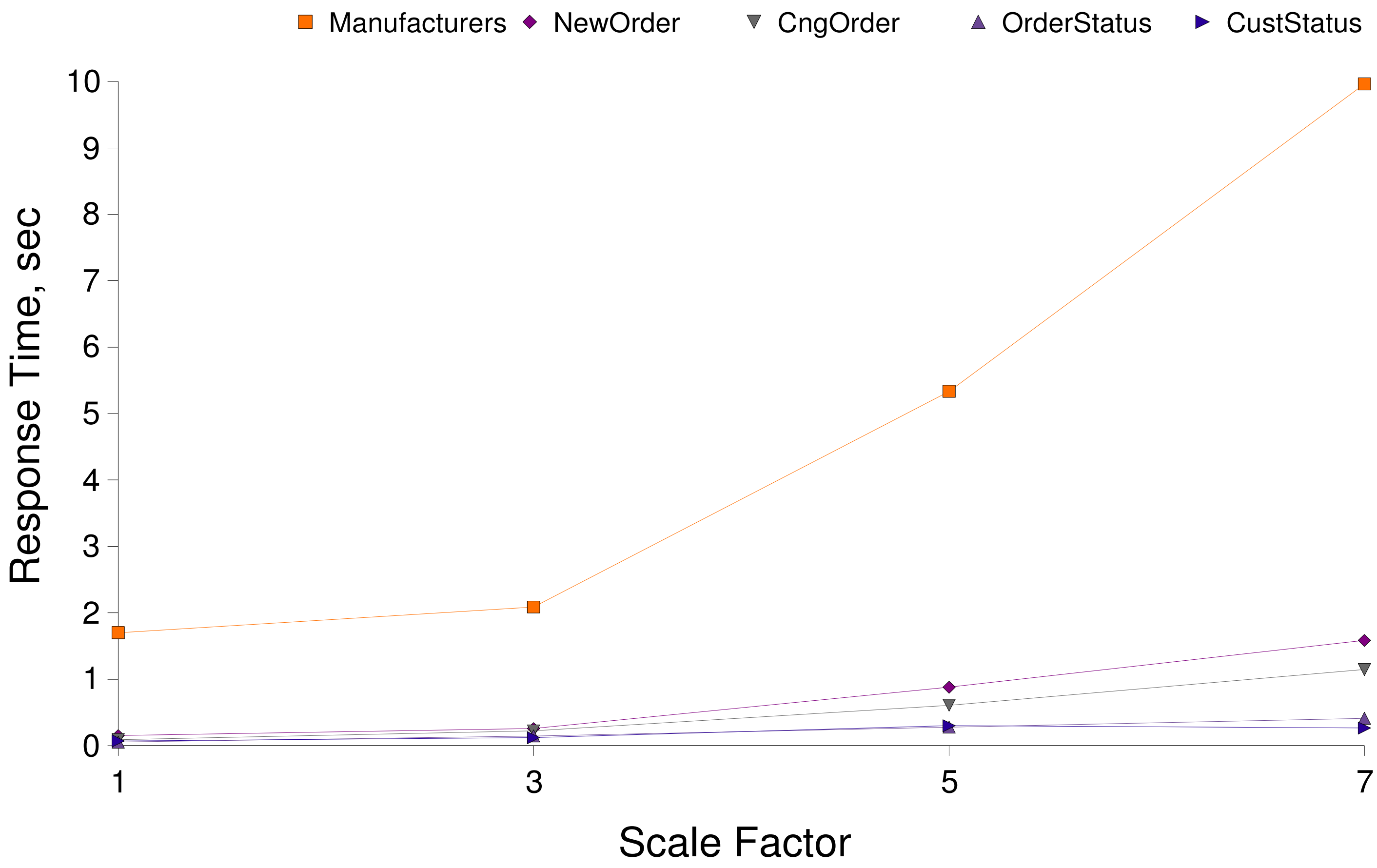
The testing environment includes three x86 machines:

- app server (PIII-866 Mhz / 512 Mb RAM),
- database (PIII-800Mhz / 512 Mb RAM)
- client (PIV-2.2 Ghz / 1024 Mb RAM)

Performance Test: Throughput



Performance Test: Response Time



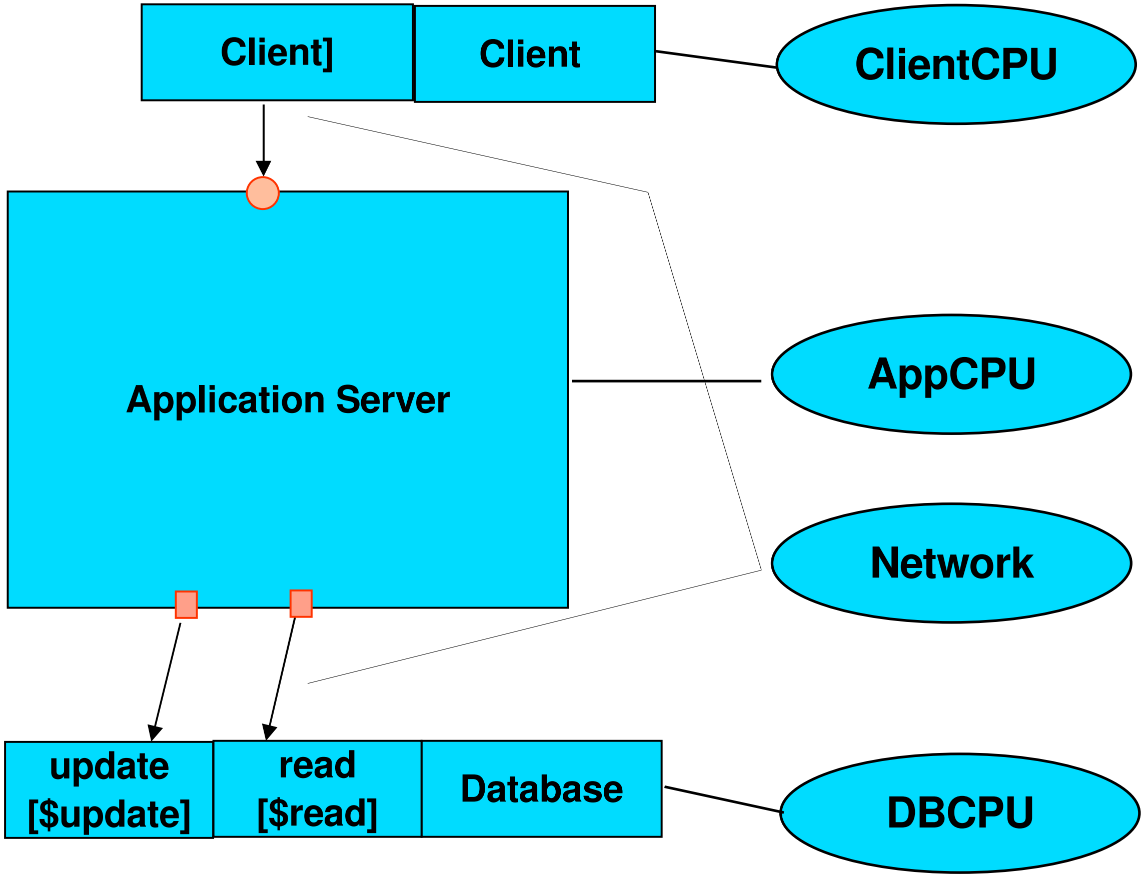
Approach Refining

- Communication – local & remote
- Container Services
- Connection Pooling
- Transaction Management
- Security
- Garbage Collection
- Naming
- Database

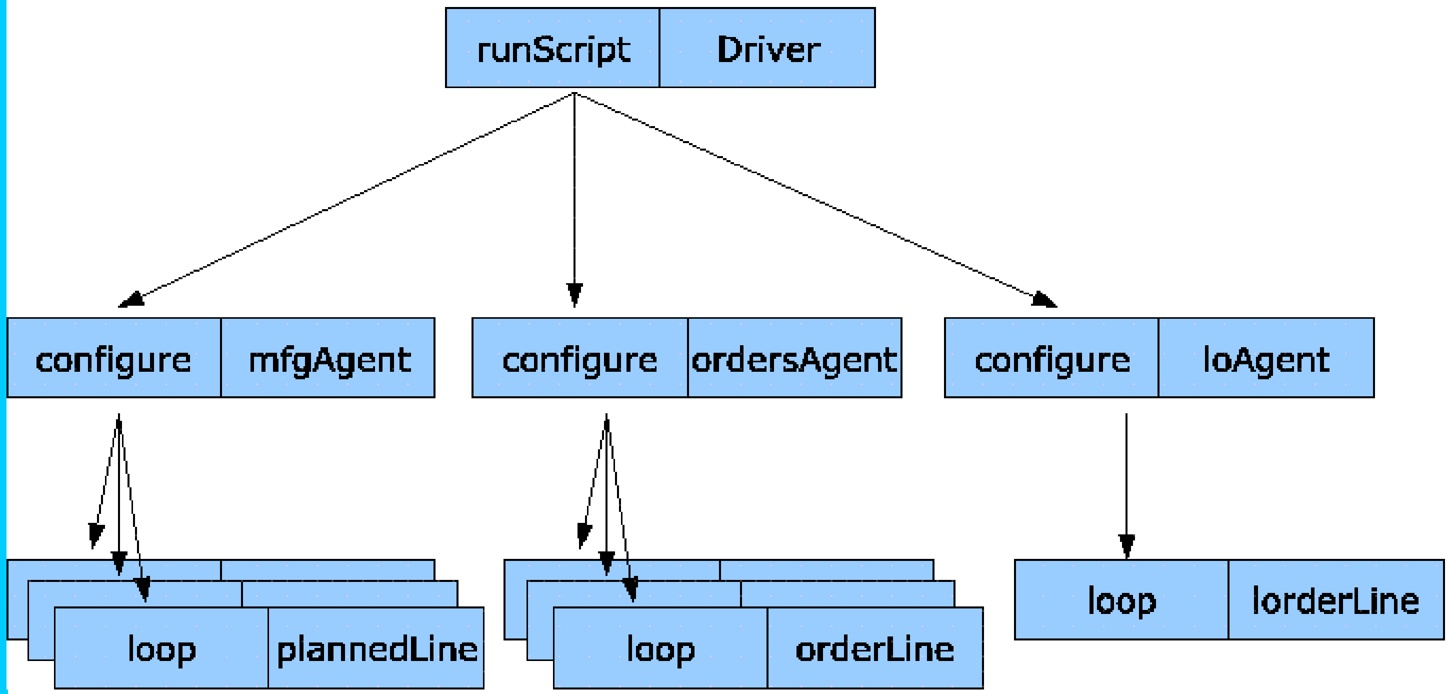
Addressing Ambiguity

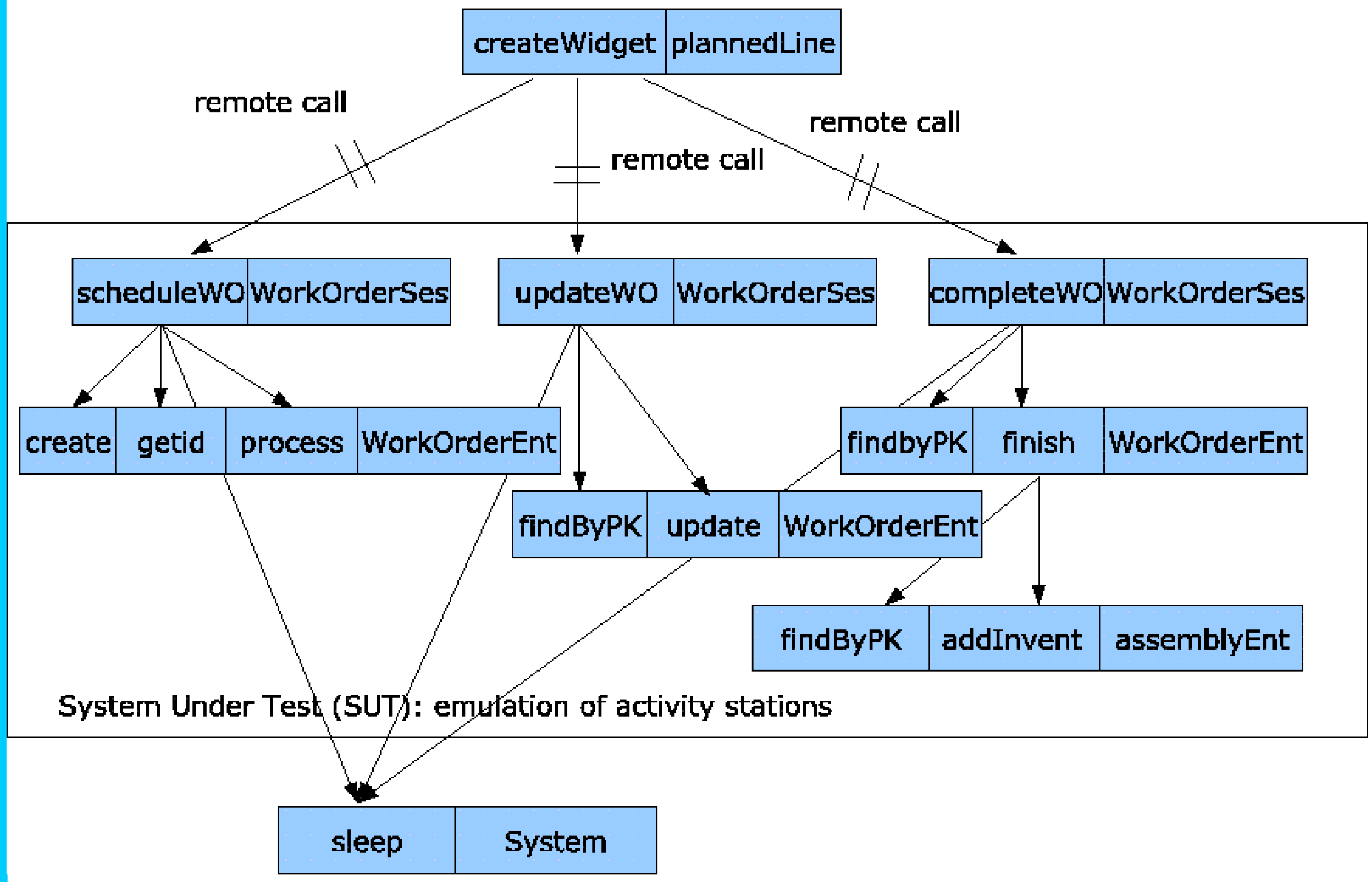
- Growing DB size
- Initial number of clients in DB depends on the load
- Transactions are retried 5-20 times when failed

Model Overview

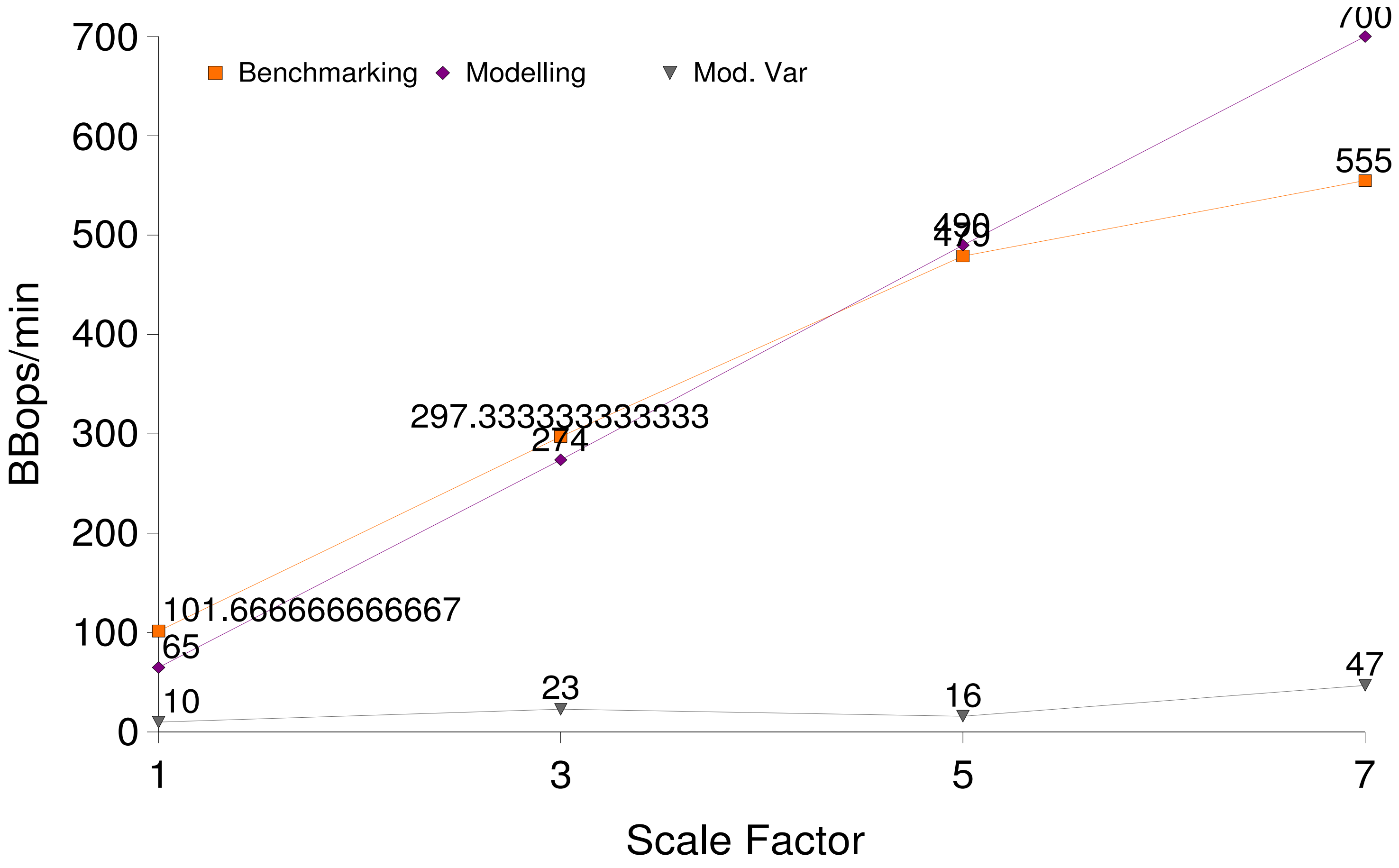


Workload


runScript
Driver
configure
mfgAgent
configure
ordersAgent
configure
loAgent
loop
plannedLine
loop
orderLine
loop
lorderLine


createWidget
plannedLine
remote call
remote call
remote call
scheduleWO
WorkOrderSes
updateWO
WorkOrderSes
completeWO
WorkOrderSes
create
getid
process
WorkOrderEnt
findbyPK
finish
WorkOrderEnt
findByPK
update
WorkOrderEnt
findByPK
addInvent
assemblyEnt
System Under Test (SUT): emulation of activity stations
sleep
System

Modeling vs Testing Results

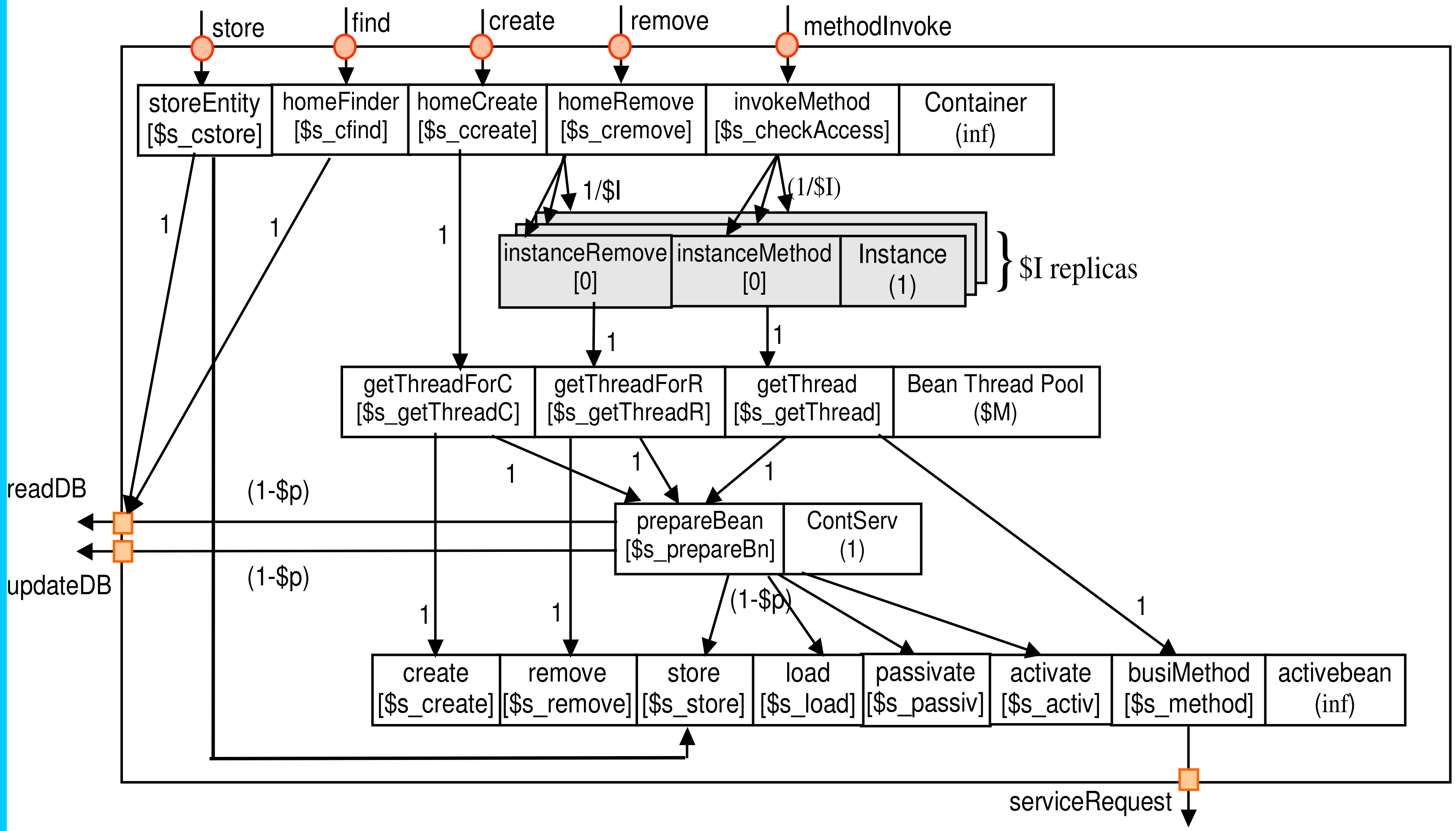


Conclusion & Future Work

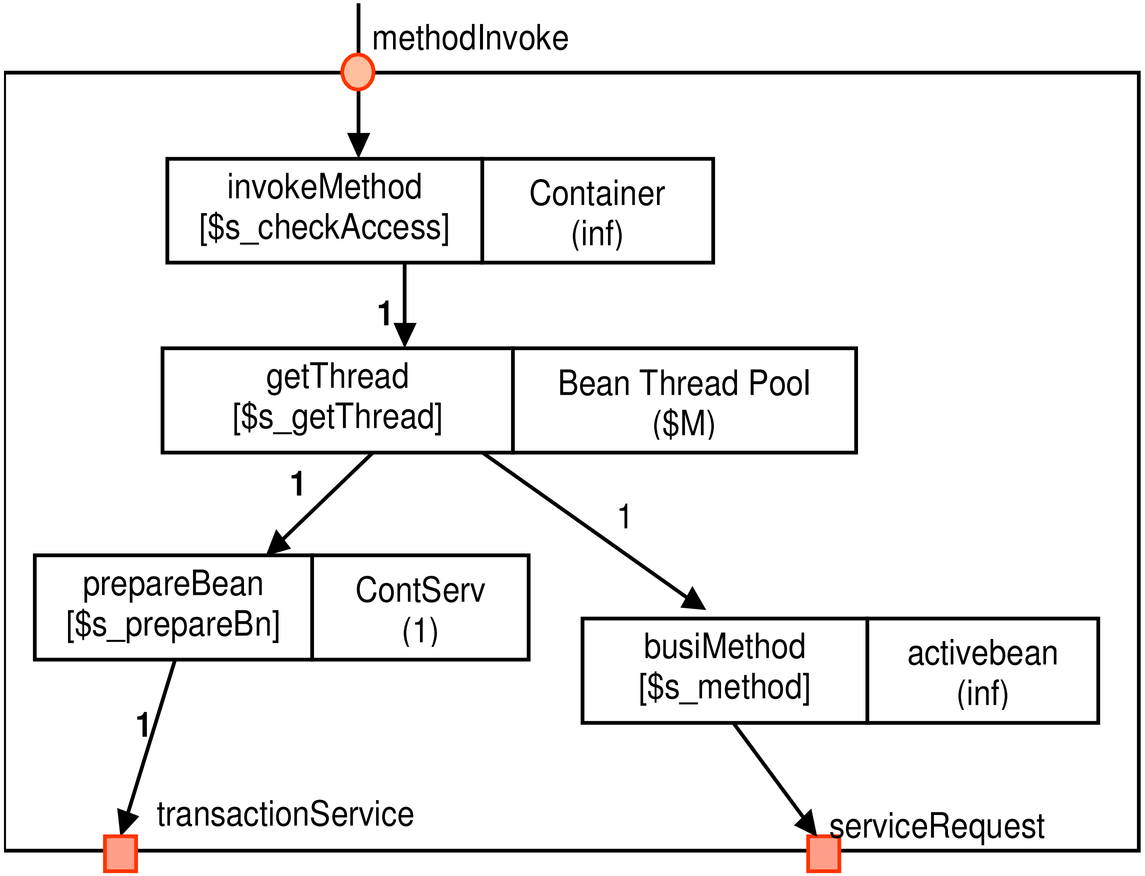
- More automation required when modeling real-life systems :-)
- Model works until systems starts approaching peak performance state
- More work is needed to understand why results go wrong at the 'border area' – system changes dynamically with load or hidden bottleneck

Questions?

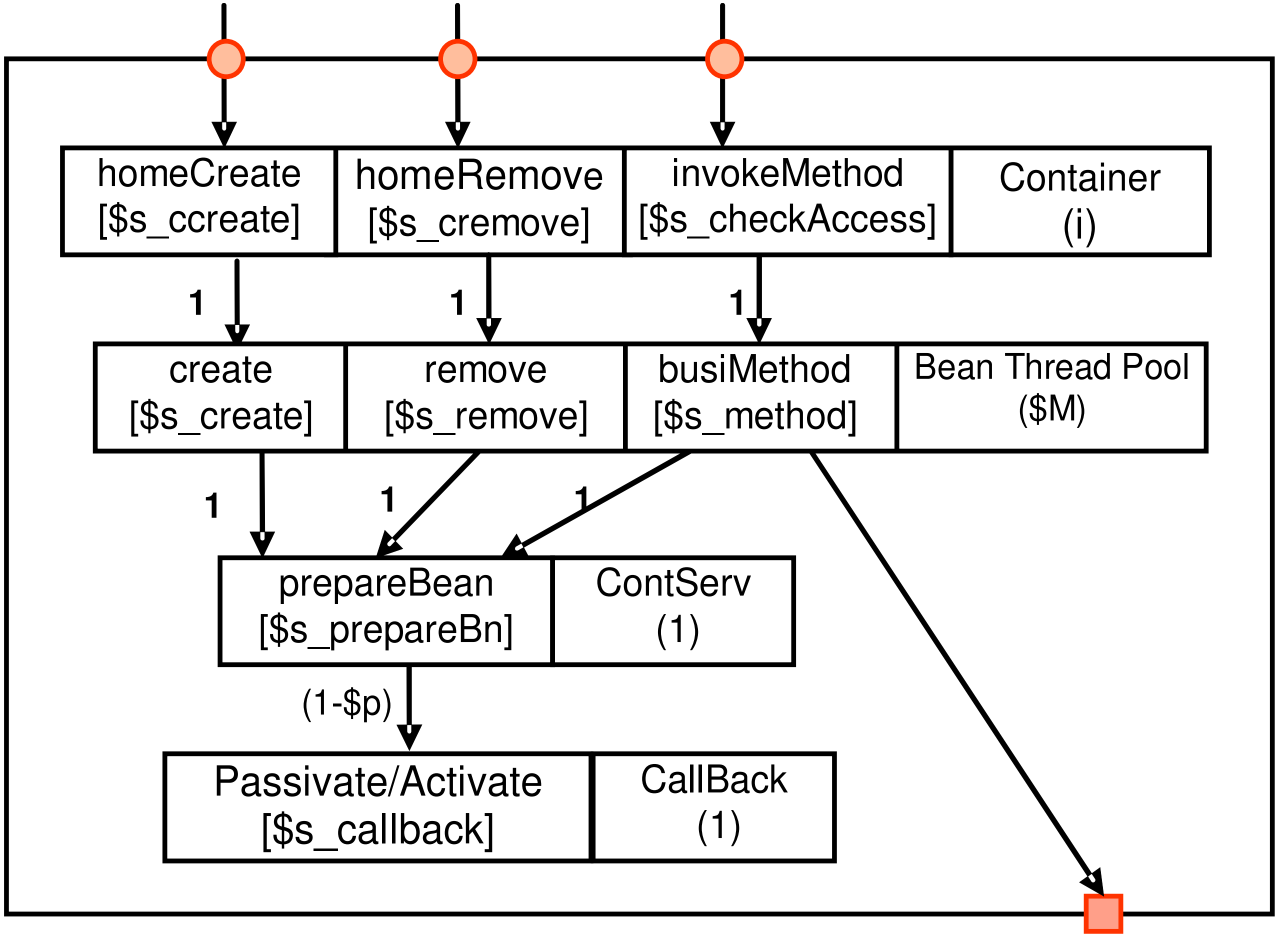
Appendix A: Entity Bean Template



Appendix B: Stateless Session Bean



Appendix C: Stateful Session Bean


homeCreate
[$s_ccreate]
homeRemove
[$s_cremove]
invokeMethod
[$s_checkAccess]
Container
(i)
1
1
1
create
[$s_create]
remove
[$s_remove]
busiMethod
[$s_method]
Bean Thread Pool
($M)
1
1
1
prepareBean
[$s_prepareBn]
ContServ
(1)
(1-$p)
Passivate/Activate
[$s_callback]
CallBack
(1)