WebSphere Application Server: A foundation for on demand computing

E.N. Herness
R. H. High, Jr.
J.R. McGee

IBM Systems Journal
Vol 43, No 2, 2004

Presenter: Barbara Ryder

Motivation

- Want to understand the analyses useful for applications written on frameworks and middleware
- Need to understand structure of such systems and part played by application code vs. library/framework vs. middleware

IBM's Websphere

- Web Application Server
- Example of middleware for "on-demand" computing
  - Separates data persistence and behavior, presentation, and control, to facilitate clean SW design
    - Manages application components and the resources they depend upon
  - For component-based, service-oriented, distributed, and message-driven computing

J2EE

- Java platform for building distributed enterprise applications
- Components:
  - Enterprise JavaBeans (EJBs), JavaServer Pages (JSPs), Java servlets
  - Interfaces for linking to databases
  - Deployed in "containers" that provide services for those components (e.g., servlets are deployed into a Web container)

Model-view-controller (MVC) Architecture

- Common to J2EE applications
- Centralized controller
  - Implemented as Java servlet and associated classes
  - Mediates between presentation view and business logic
  - Co-ordinates application flow

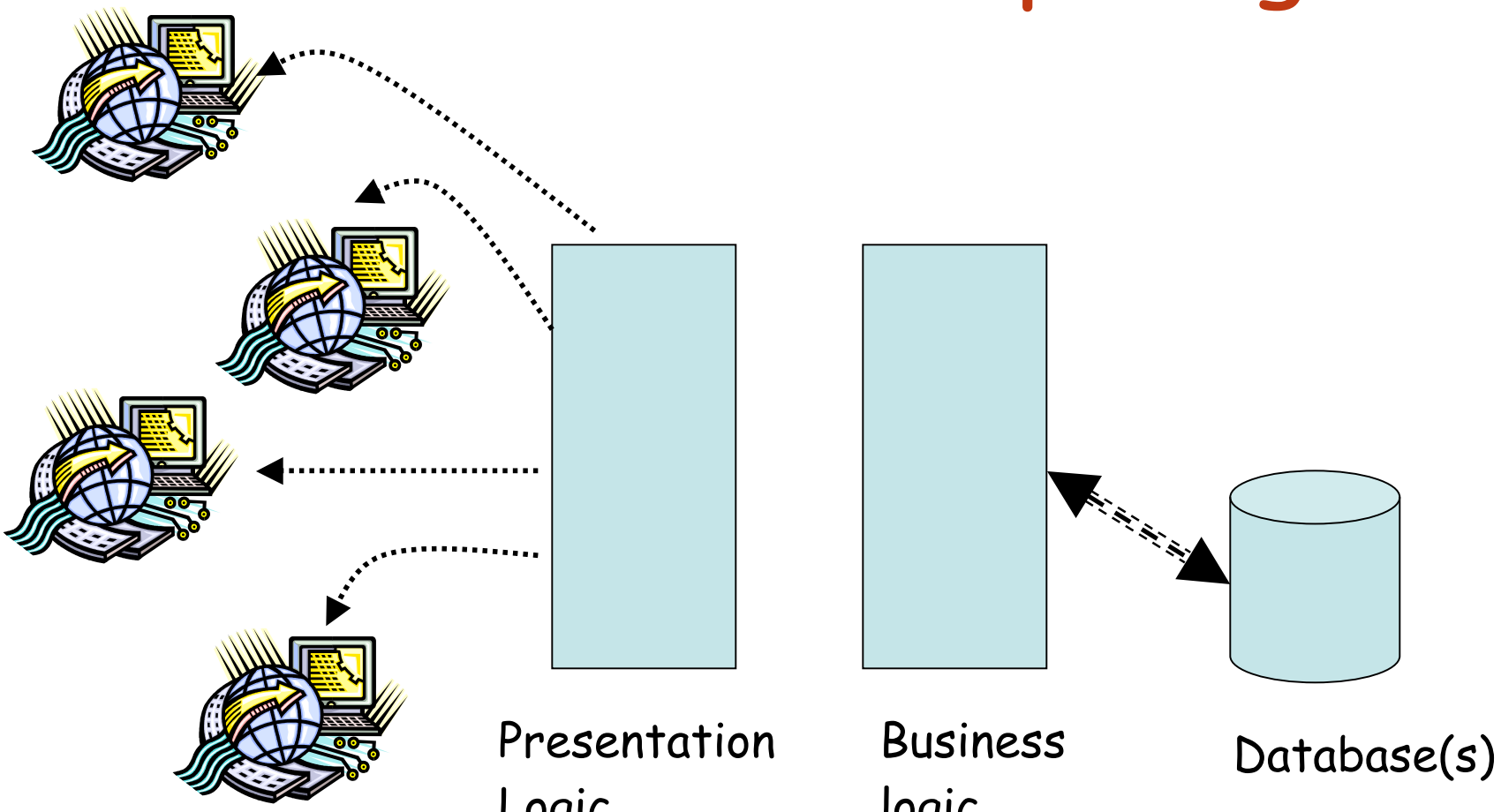
Models for E-Business Sys

- Multi-tiered distributed business computing
  - Clean separation between presentation, data, business elements of system
  - Often uses Enterprise Java Beans (EJBs) formal component model for business logic
    - Has runtime manager for bean objects to control caching of state (for efficiency) and bean creation, use, etc
    - Protection of access
    - Distribution, communication, addressing
    - Maintains single-level-store programming model

Models, cont

- Web-based computing
  - Access through Web browser or hand-held device (e.g., mobile phone) - fixed function devices
  - Originally for extending web browsers with page content derived dynamically through interaction with business logic and databases
    - Dynamic content
  - Has presentation logic and business logic in the application server layer

Web-based computing

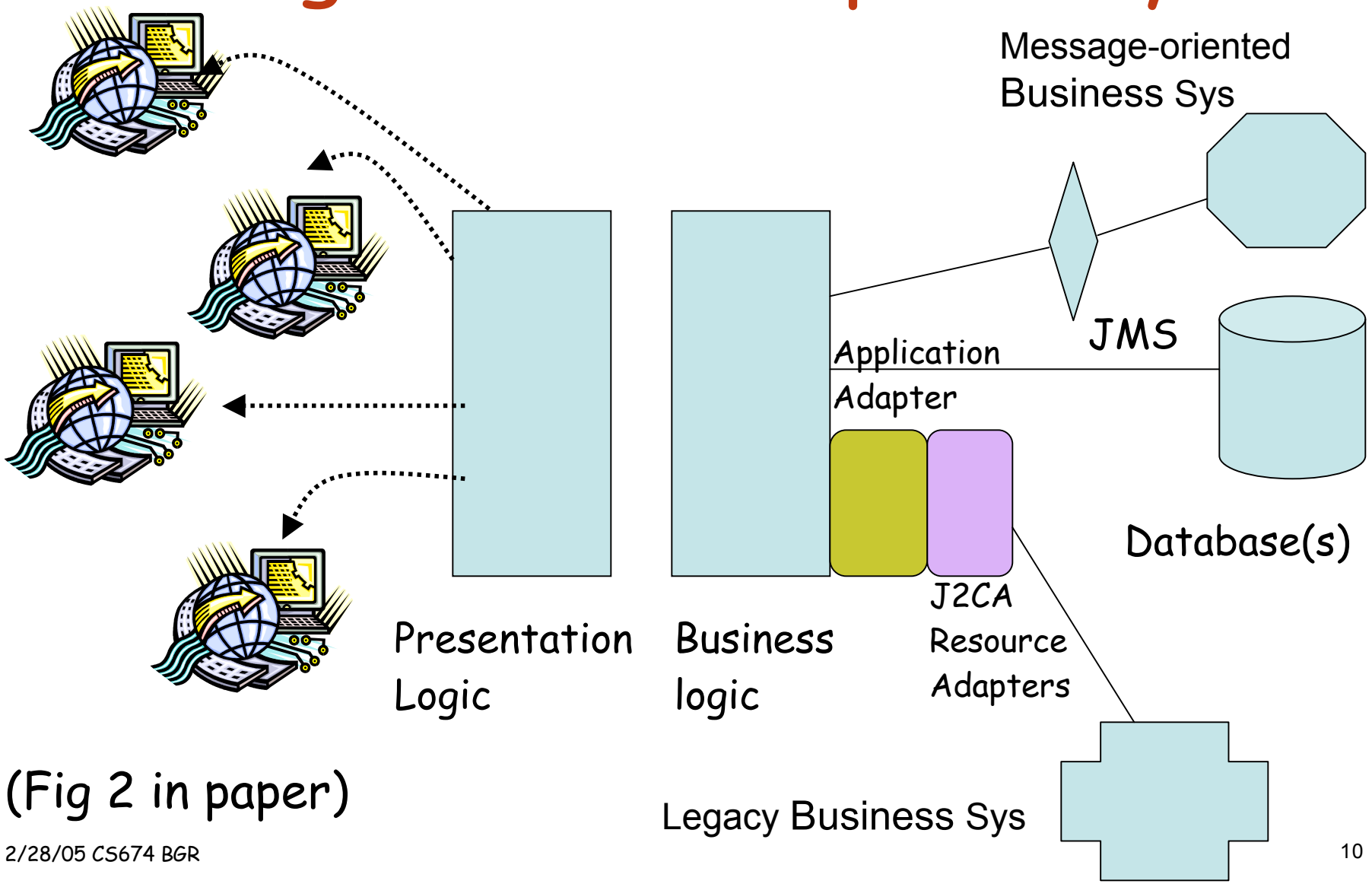


(Fig 1, in paper)

Models, cont.

- Integrated enterprise computing
  - Problem: how to introduce new apps into established enterprise system (with assumptions)
  - Issues: data integrity, security, traceability, configuration (I.e., deployment, management)
    - Java 2 Connector Architecture, Java Messaging Service(JMS)
    - Scripting languages for business processes

Integrated Enterprise Sys



(Fig 2 in paper)

Enterprise JavaBeans, R. Monson-Haefel, O'Reilly 2004.

Enterprise Java Beans

- Standard server-side component model for distributed business applications
  - Designed for interprocess components
  - RMI-based distributed object support
  - Asynchronous messaging support (JMS)
- Distributed object system layers
  - First tier - Presentation
  - Second tier - Business logic
  - Third tier - Backend

Models, cont.

- Service-oriented architecture (SOA)
  - Business services exposed for use within and outside of an organization
  - Policies to monitor and secure services
  - Standardized interfaces for ease of composition using
    - Web Services Defn Lang (WSDL), Simple Object Protocol (SOAP), Universal Description, Discovery and Integration (UDDI)
    - Loosely coupled distributed computing model

Enterprise JavaBeans, R. Monson-Haefel, O'Reilly 2004.

Web Services

- Network apps that use SOAP and WSDL to exchange information in XML docs
  - SOAP - XML grammar under W3C appln protocol for RPC and asynch messaging
    - Extensible and endorsed by all vendors
  - WSDL - XML grammar under W3C for interface defn lang to describe web services message format, Internet protocol used, Internet address, etc.

Coupling

- Temporal affinity
  - Measure of effect on system of temporal constraints among components
    - Loosely coupled systems avoid temporal constraints
      - Avoid resource contention among components even at expense of longer execution times of requests
- Organizational affinity
  - How changes in one part of system affect rest of system
    - E.g. versioning of components
    - Loosely coupled systems have high tolerance for mismatches between components

Coupling

- Also a measure of uniformity of administrative policies across a system
  - Loosely coupled systems have federated, separate policies
- Technology affinity
  - Degree of same technology base across system
    - Loosely coupled systems expect to accomplish integration with relatively few assumptions about underlying technology

Programming with Frameworks

- Temptation to work outside the model
  - E.g., directly invoke system calls in application rather than through framework interface
    - Ties application to a particular set of service technologies and database schema
    - Builds a brittle system
    - Middleware can 'do it better' (e.g., caches)

Developing an Application on Websphere

- Need to create
  - Logic elements (code in servlets and classes)
  - Declarative metadata (info in XML to control deployment and execution of app)
    - Indirection in specification of needed resources allows late binding of logical resource by administrator to specific physical resource
    - Can include caching policy, security policy, performance setting, persistent field info, description of application elements (e.g., which EJBs)

Developing an Application on Websphere

- Deployment
  - Process of installing application on application server and making it available for execution
  - Steps:
    - Make app understandable by runtime
    - Generate additional logic elements where needed
    - Bind to environment after code generation
    - Choose servers for components
    - Specify configuration and tuning params
    - Distribute app to all machines and start

Example

- Demand manager monitors system load
  - Sees server A is underutilized and B is overloaded with requests on multiple applications
  - System could move an app from server B to A for load balancing
    - Automatically stops app on A, removes binaries from A, moves the binaries to B, starts new app on B,
    - Performs changes transparently to user and app

Rapid Deployment Features

- Annotation-based programming
  - Metadata in the source code to cause generation of additional element for app
    - E.g., change in EJB implementation class automatically reflected in EJB remote interface

```
/**
*@efj.interface-method view-type= remote
*/
```

    - Allows override by external XML metadata
- Deployment automation
  - Handles code generation, compilation, installation automatically using actively monitored directory

Management System

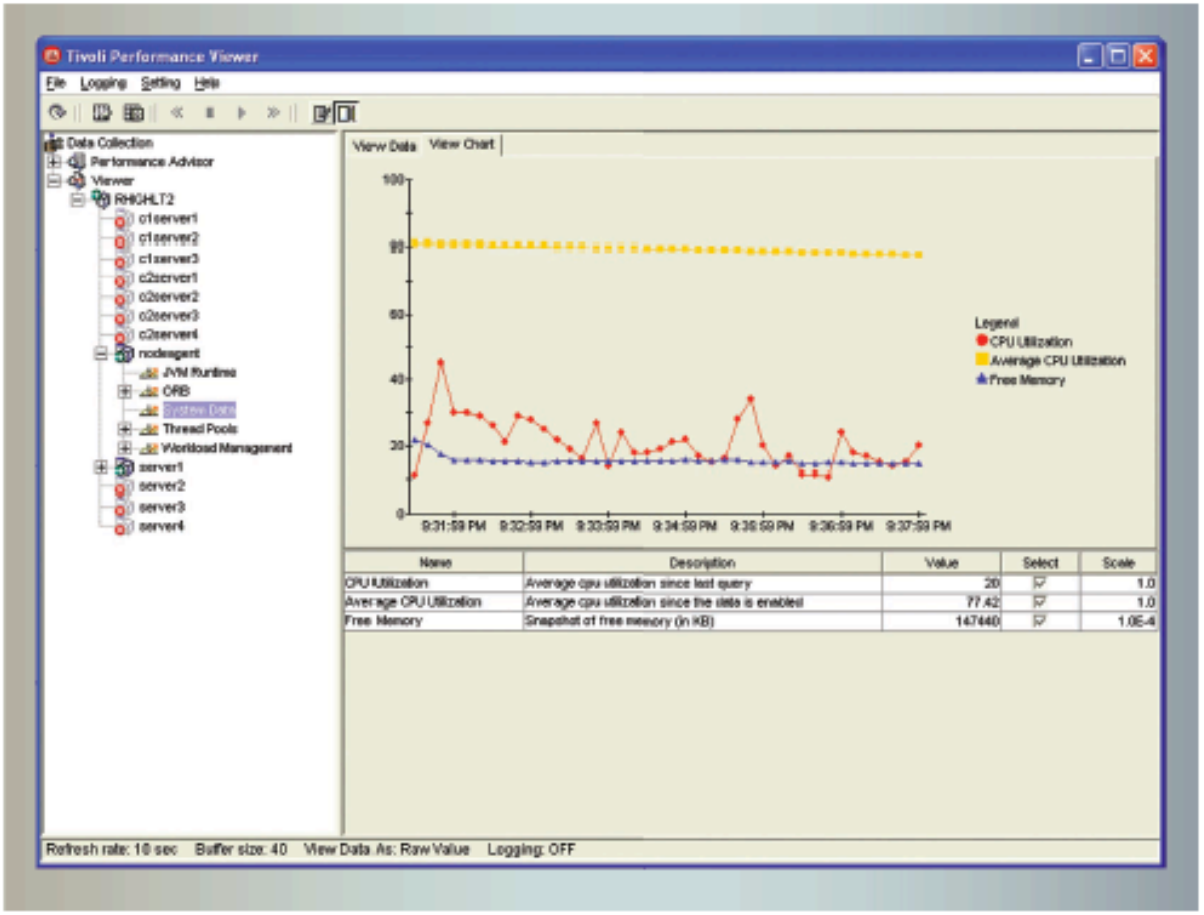
- Key idea - system has to be dynamically changeable
  - Need for accurate, consistent data about current state of environment (collection of systems)
    - Java Management Extensions (JMX) Mbeans allow polling of status of associated component
      - Logs of errors and events availble
      - JMX notifications of critical problems

Performance Monitoring

- PMI - set of configurable counters in Websphere runtime that track statistics on all requests
  - E.g., measure average wait time to obtain JDBC database connection from pool
  - Can be tuned for how much info to collect
  - Log time spent in major subsystems (e.g., Web container, database) can be written to ARM agents used by monitoring tools
  - IBM Tivoli performance viewer
- Performance advisors
  - Use data collected over time and rules from real-world systems to make concrete recommendations for performance improvement

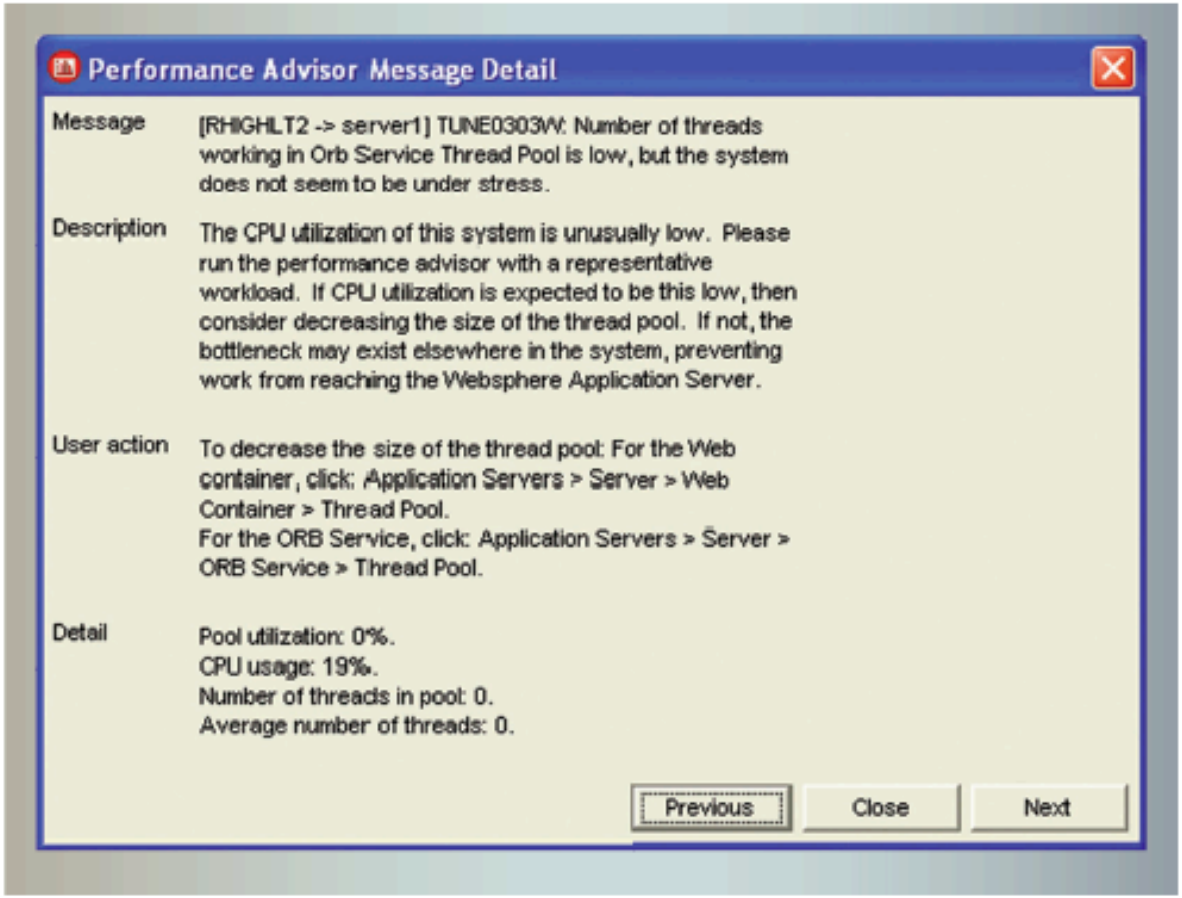
Tivoli Performance Manager

Figure 6 Viewing PMI data through Tivoli Performance Viewer



Performance Advisor

Figure 7 Example of Performance Advisor display



Workload Management

- Load balancing is main optimization for performance
  - Maximize utility from resources and soften impact of large spikes in workload within computing goals
  - Cluster- <=2 app server instances w own JVM
    - Granularity: can start/stop indiv server instances, indiv apps, or entire cluster
    - Client thinks of cluster as 1 app instance running on 1 app server instance
  - Multiple app server instances on one computer (Vertical) or different computers (Horizontal) or combo

Many Faces of a Reusable Component

- Customer Account object
  - Maintains balance of account, operations to getBalance, creditAcct, debitAcct
  - Banking app - needs balance data to be updated for every access to catch ALL updates (including system)
  - Demographic app - needs to read large number of balances efficiently to track trends; value can be approx (avoid going to disk)
  - Interest computation app - needs to have AccountHistory and Account objects with simultaneous values

Ensuring High Availability

- Outages
  - Handled by same technology as cluster workload management
  - What to do about 'orphaned' work?
    - (future) Message queue manager on failed machine must be moved automatically with appropriate fixups
  - Worst failure - entire data center fails
    - Correction: build more than one geographically dispersed data centers with broadband connectivity

Research Issues

- Many layers of these systems make static analysis limited given loose coupling and dynamic binding used
  - Also issues of scalability
- Dynamic analysis of app-only seems insufficient
- Possible combination of techniques for specific problem areas
- Possible pre-analysis of frameworks layers?
- How to accomplish good performance? Code reuse? App understanding?
  - With the aid of program analysis?
  - Need to include O/S or HW measurement?
- Relation to autonomic computing?