
# Introducing eBay's Scalable Innovation Platform

Dan Douglas, Chris Kasten, Richard Ragan

Java

Sun microsystems

# Gain exposure to the concepts and technology in eBay's Scalable Innovation Platform

GOAL

# Agenda

- Overview of the eBay “world”
- V4 Presentation framework
- Data Access Layer (DAL)
- µKernel

# Why a Scalable Innovation Platform?

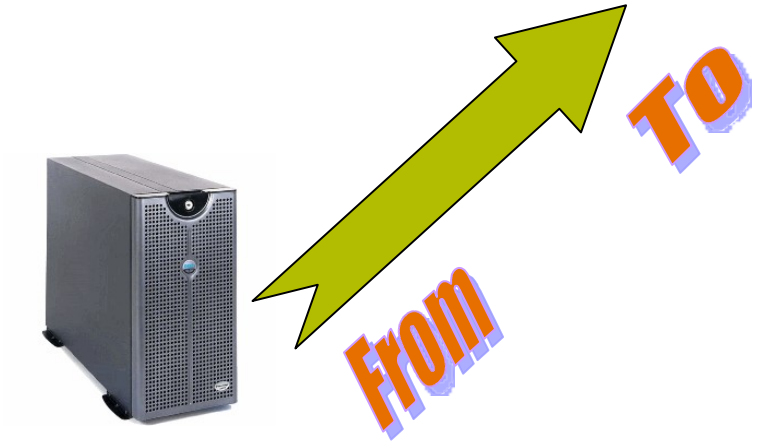
- eBay is more than you think (26 companies,39 countries)


- Some eBay companies are huge (eBay, PayPal, ...)
- Some eBay companies are small but growing fast
- All eBay companies need ability to try new ideas quickly

So, a small platform solution for our small companies that could scale to the massive traffic levels of the eBay site was needed. The solution had to support trying innovative applications with fast concept to delivery cycles.

# What do you mean by *scalable*?

- Over 2 billion page views/day
- 2 Petabytes of data storage
- 40+ Billion SQL executed / day
- Over 16,000 application servers
- 12 Million+ lines of code
- 438,000 site images
- 80,000 static pages x 39 countries
- Release whole site weekly
- 24 x 7 operation




**1 Web Server with Database and Application**

# What's *innovative* about it?

## *EASY* to try out ideas

- Reusable presentation layer components
- Visual testing without deploying to a server
- Integrated DAL wizard/editors

## *EASY* to use

- Provides a "convention over configuration" style app framework
- Lots of tools to speed you up

## *EASY* to get

- Simple installer for customized IDE
- Relies on standardized artifact repository

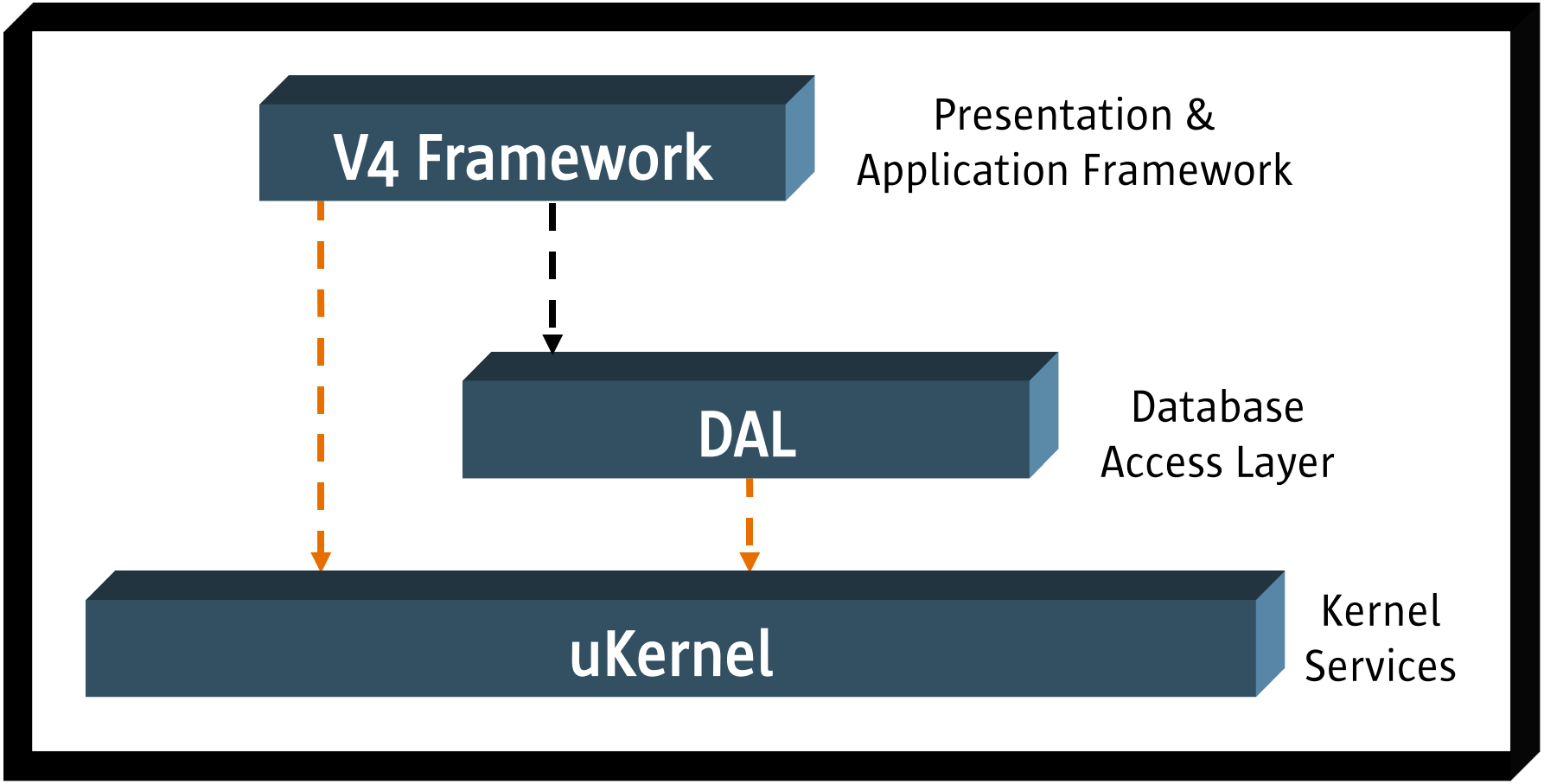
# What makes it a platform?

*EASY* to share ideas

- Repository of proven components
- Culture of reuse
- Standardized environment
- Common deployment/operational/management models

# Core Subsystems

## Building blocks of our scalable platform



# Agenda

- **Overview of the eBay “world”**
- V4 Presentation framework
- Data Access Layer (DAL)
- µKernel

# eBay Presentation Issues: Our Problems

- Business Logic was in Java,  Presentation in XSL
- These separate technology stacks caused problems
  - Running out of memory
  - Performance: (Java → XML → DOM) + (XSL → DOM) → Html String
  - XSL being used for business logic (expediency)
  - Content strings embedded in XSL code
  - XSL not suitable for software engineering (refactoring, components, tools, …)
  - Developers isolated into separate “tribes” (Java, XSL, Javascript). Lack of shareable tools, knowledge, development environments

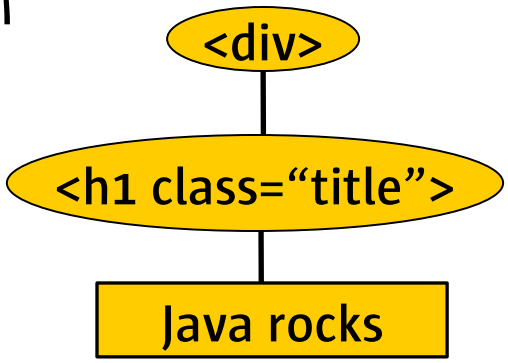
# V4 Solution

- Java Everywhere!
  - HTML Generation
  - UI Components expressed in Java
  - Full web resource management for safety and traceability
  - Content system
  - Javascript and Css (in Java)
  - Ajax and Portal infrastructures
  - Templating Solution
- Application Framework to integrate all the above
- Fast, rich set of IDE-based tools, emphasis on Quality Engineering
- Lots and lots of Docs and Training material

# V4 Java-based Presentation Approach

- Lightweight DOM 2.0 implementation in the basement
- Html 4.01 tags to construct a server-side DOM

```
DDiv div = new DDiv();
DH1 h1 = new DH1("Java rocks")
    .setHtmlClassName("title");
div.appendChild(h1);
```




- DOM can also hold a UI "component" that produces a subgraph (e.g. Navigation Tabs)
- After construction, render the DOM to HTML
  - Alternative rendering possible for things like text-based email

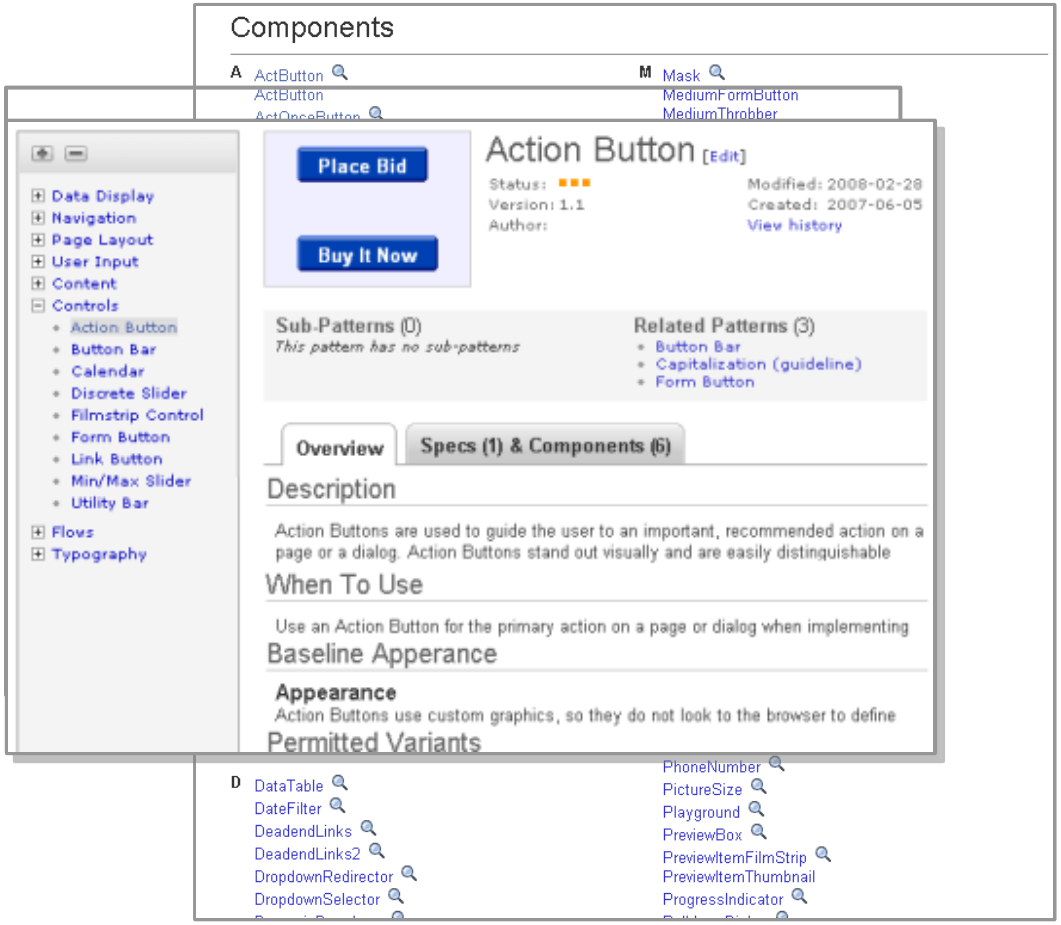
# V4 Java-based UI Components

- Reusable self-contained UI component
  - Generates Html subtree for its structure
  - Owns its Css, Javascript, Content
  - Has properties that control visual appearance option
  - Static definition of resources it uses (Images, static links, content, Css, Javascript)
  - Declares dependency on other components if a composite component
  - Has Interface-based bean data model + concrete implementation
  - Has Sample data model implementation for tools and testing


- To use a component:
  - Create and populate the component's Data Model
  - Instantiate the component with it's data model
  - Append it to the DOM you are building

# V4 Component Catalog

- Developed jointly with UI designers to showcase all components
- A Reference for Component
  - Description
  - When to Use
  - Appearance & Behavior
  - UI mockups/specs
  - Appearance generated by running actual code
- Facilitates Component Discovery & Reuse



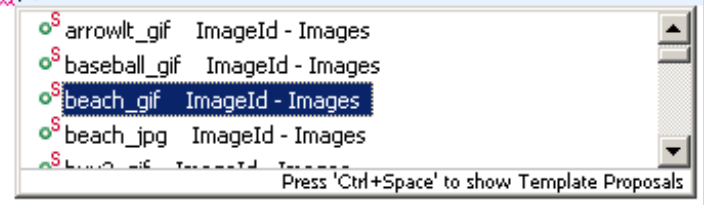
# Resource Management: Safety and Tracking

- Web pages depend on resources (images, links, external Javascript, Css, content, ...)
- Knowing all uses of a resource allows assessing impact of changes
- Type-safe resource references give compile errors if deleted/changed. No typos with Eclipse type-down.
- Resource uses are statically and dynamically traceable

```
MemberBadgeContent memberBadge = content(MemberBadgeContent.FACTORY);

LinkRef powersellers = link(links.services.buyandsell.Links.powersellers_html);
LinkRef feedbackUnverified = link(links.help.feedback.Links.unverified__feedback_html);

ImageRef powerSeller = image(pics.icon.Images.psIcon_50x25_gif);
ImageRef beach = image(pics.aboutebay.report.Images.);
```




Type-down Image Example

# Javascript (VJO)

- Javascript semantically closer to Java for safety (but still Js)
  - Packages, Classes, Interfaces and Imports
  - Constructors, overloading, super()
  - Instance members and functions
  - Static members, functions, and static initializers
  - Typed functions, return types, and argument types
  - Access control (class, interface, member, functions), final
- Code-generated type-safe Java proxies from Vjo. Used to bind Javascript event handlers to DOM elements
- If Javascript changes, Java code gets compile errors if incompatible

# Java and VjO side by side

```
package x.y.z;
public class Person {

public static final int MAX_NAME = 12;

    private String m_name;
    private int m_age;

    public Person(String name, int age){
        setName(name);
    }

    public void setName(String name) {
        m_name = name;
    }
    public String getName() {
        return m_name;
    }
}
```

Package + Class = Type

Static property

Instance members

Constructor

Instance Methods

```
vjo.type(x.y.z.Person')

.props({
    MAX_NAME:12
})
.protos({
    m_name:null,
    m_age:0,
/**
 * @JsClass Person
 * @return void
 * @access public
 * @param {String} name
 * @param {int} age
 */
    constructs:function(name,age){
        this.setName(name);
        this.setAge(age);
    },
    setName:function(name){
        this.m_name=name;
    },
    getName:function (){
        return this.m_name;
    }
});
```

JsDoc for type/access

# Text-based Css problems we saw

- Class or id name can change and the referencing code never knows 

- Class/id name collisions between components 

- Image name typo/wrong path in Css 

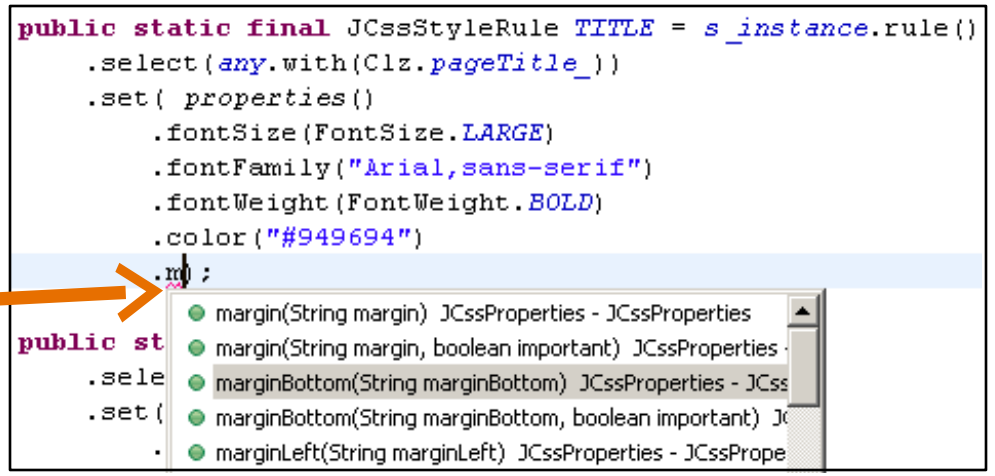
- Users get Css properties wrong (typos, ignorance) 

- Override a rule of some other Css and it changes unbeknownst to you 

- Country-level or browser-specific override of default Css by copying. What if original Css changes? 

# Java-based Css: JCssDef Solution

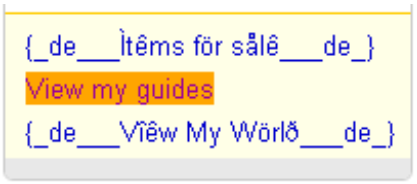
- Define Css in Java
- All Css 2.1 Selectors & properties supported
- Css Property type-down
- Class/id name reservation to avoid collisions




- Code-gen reference proxy for Css class/id name use. Compile time error when Css changes/vanishes.
- Programmatic reference to a style rule allows extension of the rule, and scoping (rather than copying)
- Tool to convert from .css to JCssDef form

# V4 Content System

- Content separated from code
- Hot content updates to production (40 minutes to thousands of servers in multiple data centers)
- Rich capabilities with high performance
  - Typed dynamic placeholders (String, Integer, Long, Date, Money,…)
  - Text styles in content element (bold,  italic, underline…)
  - Links and images references in content element
  - Structured  types for content (List, Map, Structure, Block)
  - Full internationalization supported
  - Content creation tools + Direct In-Browser In-page content editing
  - Find Localization problems before translation (pseudo localization with highlighting for text strings not coming from content system)

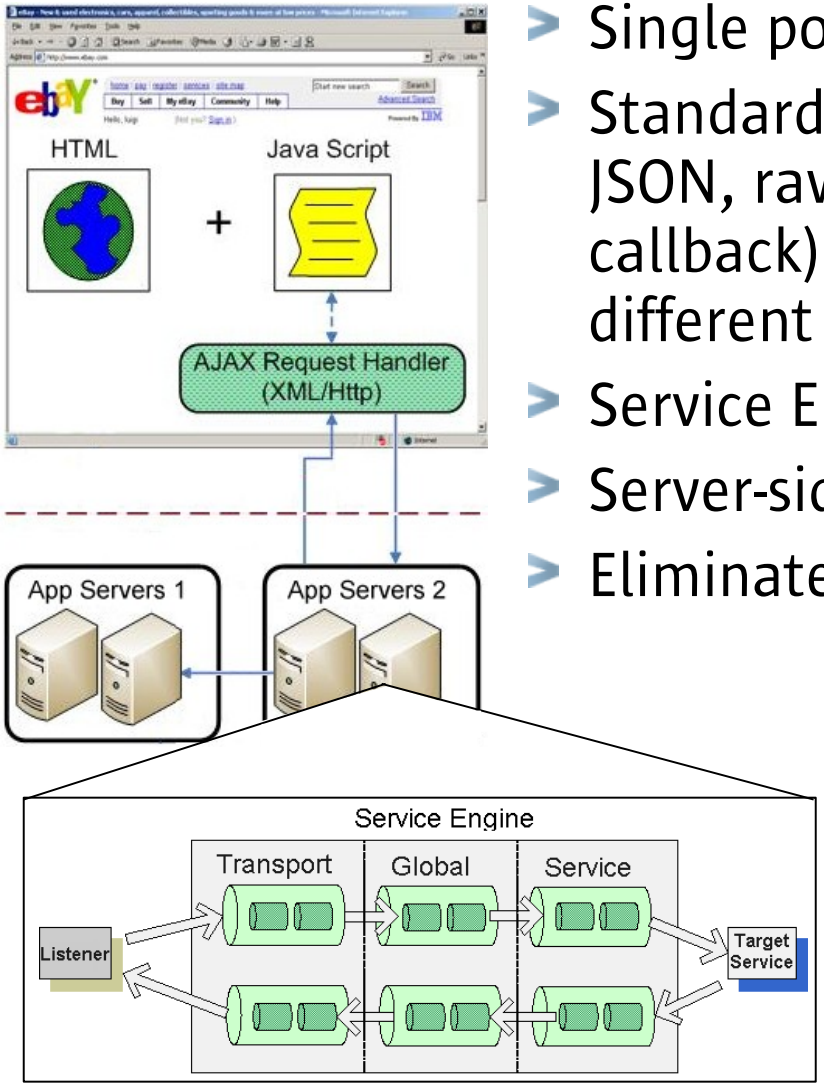


# V4 Templating (ESF – eBay Server Fragments)

- Templating is sometimes the right solution so we offer it
- ESF extends JSP 2.1/EL 2.1 specs. ESF = JSP minus J2EE Servlet
  - Removed dependency on J2EE servlets and on specific protocol (HTTP). Runnable in Java main.
  - Removed dependency on XML declarations
  - Supports Standard Taglibs including JSTL 1.1
- Pure Peering Model with V4 Components
  - Components can directly use ESF and vice-versa
  - V4 and ESF can use Tags. Tags can use V4 Components and ESF
- ESF can access all V4 type-safe resources (images, Content…)
- ESF Templates are themselves a V4 resource
- Simple example

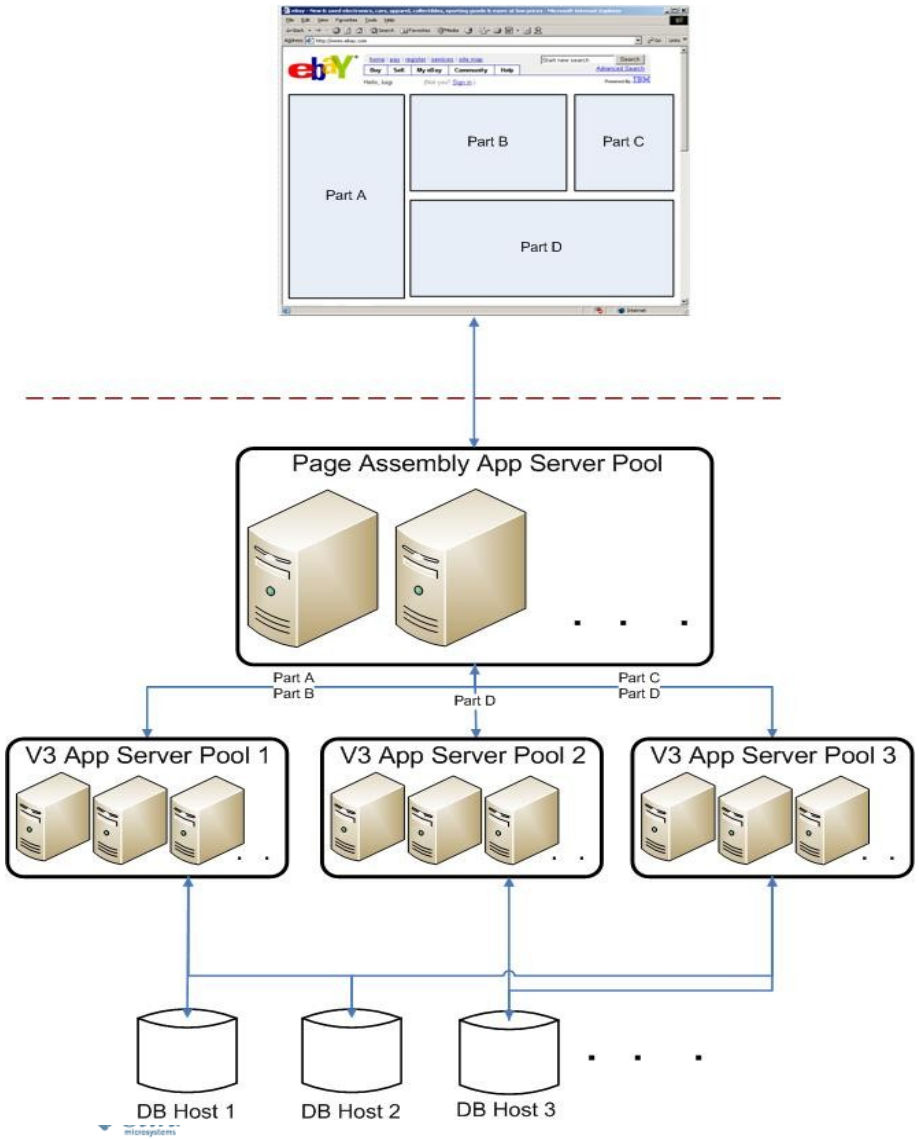
```
EsfTemplate esf = new EsfTemplate("Date is:
  <%=(new java.util.Date()).toString()%>") ;
System.out.println(esf.execute()) ;
```

# V4 Ajax




- Centralized registration for services/handlers
- Single point security management
- Standard marshalling/un-marshalling (Xml, JSON, raw data, name-value pairs, JSON with callback) and Request/response can have different formats (Req: XML, Resp: JSON)
- Service Engine on Server and client
- Server-side proxying of Ajax for mixed domains
- Eliminates duplicate requests

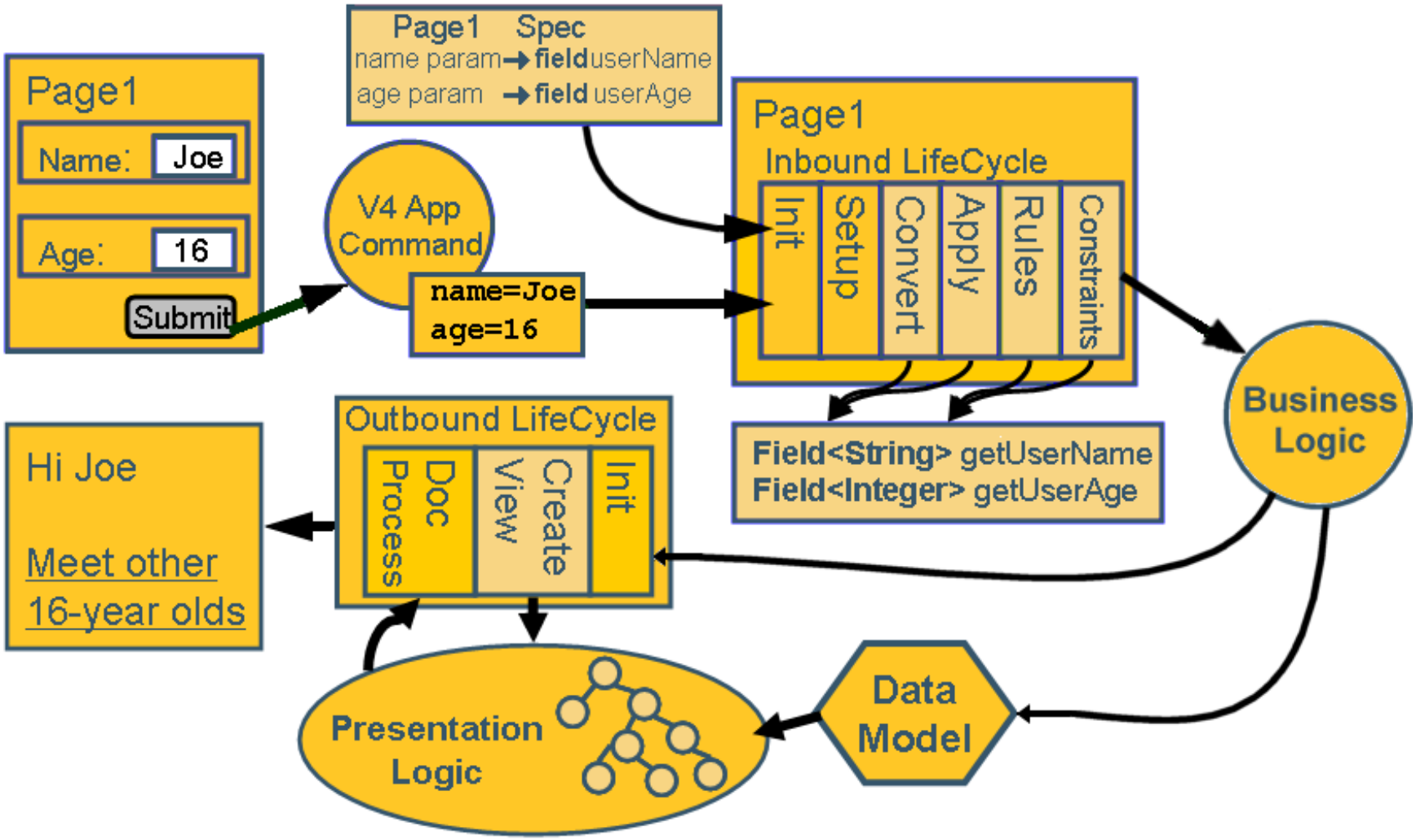
# V4 Portal




- Server-side assembly into wireframe from local and remote portlets
- Page config determines layout

- Duplicate Css/Js eliminated automatically
- Buildable from ordinary V4 page and components
- Centralized security control
- Can support dual Js libraries at once as pools are rolled out

# V4 App Framework: Request/Response Cycle



# V4 in Action

# Agenda

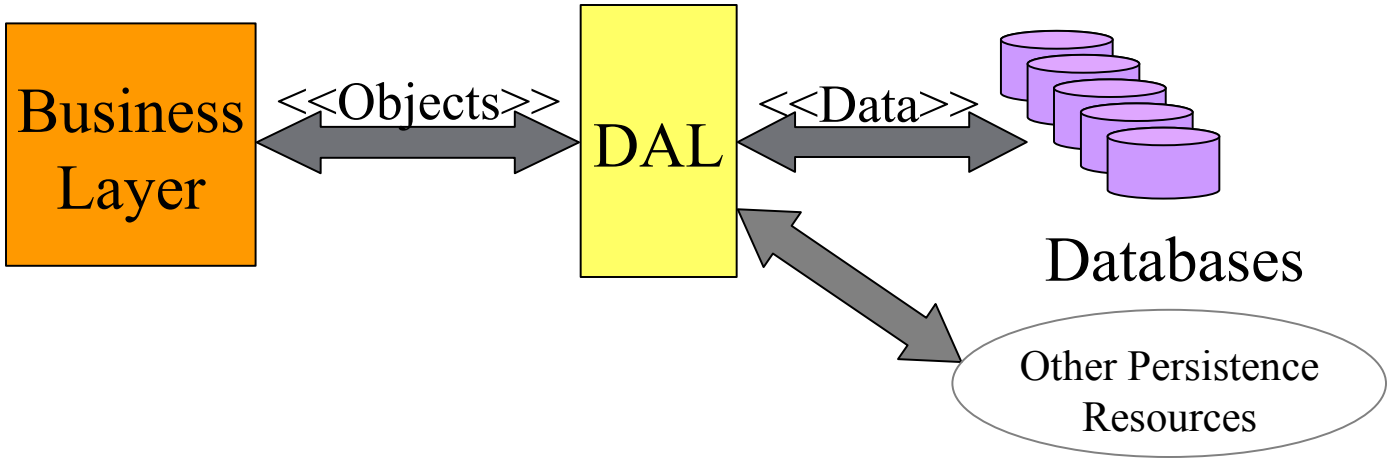
- **Overview of the eBay “world”**
- V4 Presentation framework
- Data Access Layer (DAL)
- µKernel

# eBay's Data Access Layer (DAL)

- DAL Overview
- General DAL Services
- DAL Architecture Basics
- DAL Services and Features Details
- DEDE

# DAL Overview

- The Data Access Layer (DAL) is an object-oriented abstraction for accessing and updating data in distributed persistence resources
- All persistence resource CRUD (Create Read Update Delete) operations are performed through the DAL's abstraction of the data.



# Why is a DAL needed?

- The DAL abstraction
  - Decouples the persistence layer
  - Encapsulates translation from persistence resource format (e.g. relational database fields) to Java objects
  - Encapsulates persistence resource interactions
  - Encapsulates mechanisms to locate distributed data
  - Encapsulates logical to physical decoupling
  - Encapsulates persistence failover mechanisms
  - Encapsulates caching mechanisms
- The DAL provides a consistent, transparent, object oriented API for accessing and updating data in distributed persistence resources

# Guiding Principles

- Performance and scalability is a major consideration in everything
  - Performance drives capacity, which impacts the bottom line
  - eBay's very large scale rapidly multiplies the littlest of things in to significant costs and impacts
  - We measure to the millisecond on all SQL
  - We measure to the microsecond on all cache accesses
  - We collect stats on every SQL executed
- Operational management, monitoring, and control is considered up front and not an after thought
  - A site cannot scale and be manageable and stable without operational considerations baked in up front

# Guiding Principles

- Developer productivity and maintainability is critical to longevity
- Provide a platform that can be simple and quick to develop in, but also has inherent eBay level of scalability built in under the covers for later graduation and usage.
- Provide power features and flexibility for when it is needed
- Full encapsulation of all DAL responsibilities for plugability, flexibility, and future evolution
- Always consider that the persistence mechanisms used by the DAL can be changed out and no business code above it should have to change

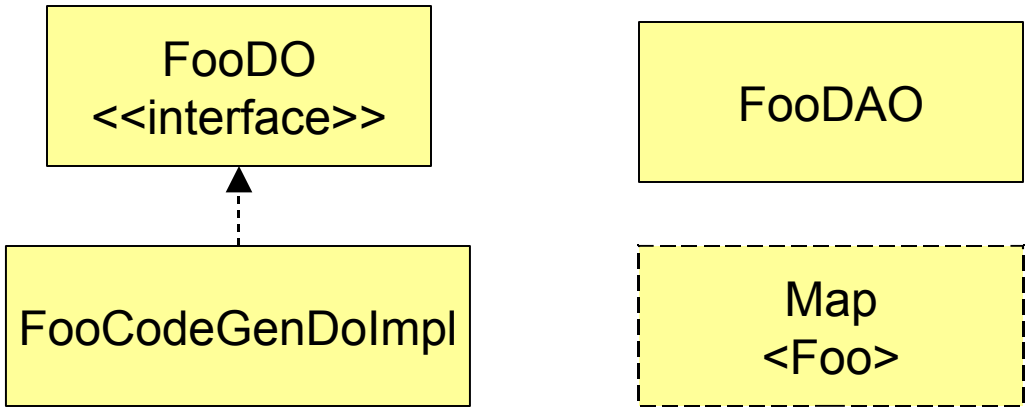
# General DAL Services

- Object to Relational Mapping
- Templatized SQL
- Data Dependent Routing (DDR)
- Logical to Physical Mapping
- Partial Object Hydration/Persistence
- Connection/Statement Management (DCP)
- Connection Storm Suppression (STR)
- Automatic Markdown
- Object Caching Technologies
- Session/Personalization Cache Tier
- DAL Eclipse Development Environment (DEDE)

# DAL Architecture Basics: Major Components

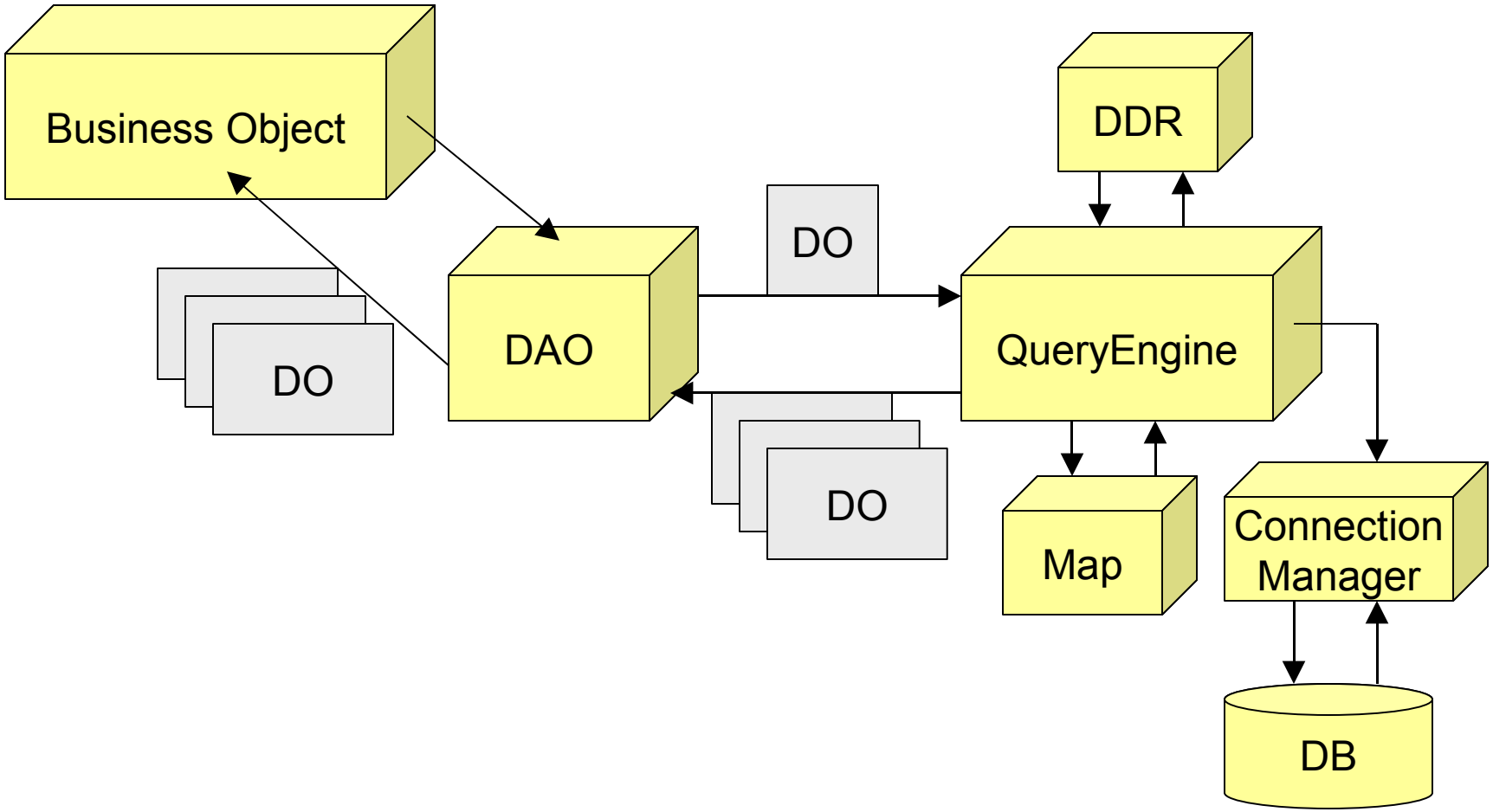
- Data Object (DO)
  - Represents a persistent object
  - JPA annotations on the interface
  - Consists of an interface and a generated Impl class of that interface
- Map
  - Runtime interrogation of DO interface for O/R mapping component creation
  - Owner of SQL, routing hints, read sets, and update sets registered by the DAO during initialization
- Data Access Object (DAO)
  - Provides methods to insert, update, delete, or find data objects
  - Code is source of truth for SQL, routing hints, read sets, and update sets
  - Content presented in Eclipse plugin editors for ease of development
- Query Engine
  - Instantiated and used by the DAO to coordinate and perform execution of the actual CRUD operations
  - Collaborates with the Map, connection management system, data dependent routing system, etc.

# DAL Architecture Basics: Specialized Components




- JPA annotated DO interface
- Eclipse plugin generates
  - CodeGenDoImpl (fully regenerated)
  - DAO (eclipse editors for development life cycle)
- Generic Map class instance for each DO type
- Customized behavior possible in subclass of DoImpl

# DAL Architecture Basics: Flow



# Templatized SQL Approach

- Standard SQL with template place holders imbedded in it
- Reduces maintenance of SQL by having the appropriate field names dynamically determined and substituted into the query
- Allows for substitution of physical table name returned from DDR configuration, rather than being hardcoded
- Allows for intuitive binding of DO attributes and hints directly in the SQL (typically the where clause)
- Allows for flexible use of a single SQL definition with different readsets (if template place holders are used)
- Allows for the full expressiveness of SQL when needed
- Intuitive look and feel of real SQL

# Example Templatized SQL

- Select Statement Example
  - Templatized SQL:

    SELECT /*<CALCOMMENT/>*/ <SELECTFIELDS/> FROM <TABLES/> WHERE a.ID=:m_id and (<JOIN/>)

  - Example Resulting SQL:

    SELECT /* FooMap.FIND_ALL.-2 */

    a.ID, a.TYPE, b.DESC FROM EBAY_FOO a, EBAY_BAR b WHERE a.ID=? and (a.ID = b.ID)

# Data Dependent Routing (DDR)

- A service used by the QueryEngine during runtime to determine the location of data in a distributed data persistence system
- Provides full decoupling from physical datasources via logical to physical mapping facility for deployment in to different environments
- Routing rules implemented as Java code in Touple Providers that return a physical table name and physical datasource (a touple) given a logical table name and routing hints
- Logical to physical map defined via a config file

# Connection/Statement Management

- eBay Database Connection Pool (DCP)
  - Highly scalable connection pooling and prepared statement caching
  - Pluggable adapters for any database JDBC driver
  - Reads external datasource configuration file (pluggable)
  - Significant configurability for both connection pool and prepared statement cache (runtime modifiable)
  - Prepared statement cache has LRU with also time based eviction
  - Integrated Stagger, Throttle, Retry (STR) connection storm suppression system
  - Web page viewable config and statistics during runtime

# Markdown

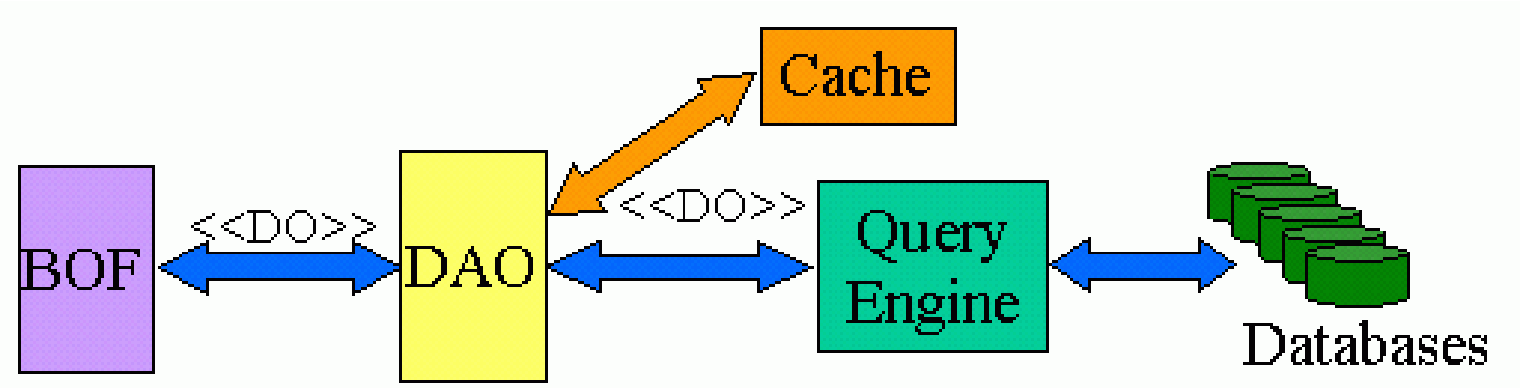
- Markdown is a historical eBay term referring to the ability to shutdown access to a database resource inside of the application code.
- The DAL provides the ability for external site operations tools to mark a datasource as not being available for use by the application. (also mark it back up)
- The DAL further has the capability to auto-markdown a datasource using a set of heuristics during runtime that are configurable from a configuration file.
  - Protects application server capacity from being impacted by a slow responding, internally failing, dying, or dead database

# Parallel Query Engine

- Ability to automatically divide up a QueryEngine request based on physical datasource destinations and run the queries in parallel on each host
- Supported for select, update, insert and delete
- Supports full scan aggregation in parallel
- Configurable on a query bases via config file

# DAL Cache System

- Generic caching system that is flexible and extensible
- Two types of caching:
  - Shared Cache (homogeneous cache shared by multiple threads)
  - Thread Local Cache (heterogeneous cache exclusive to a thread)
- Multiple indexes (unique and collection keys)
- Different options for L1 and L2 cache technologies



# DEDE

- DEDE is an Eclipse plugin for rapid DAL development:
  - Bottoms up generation of DO interface from database schema
  - Top down generation of database schema from DO Interface
  - A wizard to generate CodeGenDoImpl and DAO based on the DO interface
  - A multi-page (tabbed) editor to easily define and edit the major components of the DAO (queries, read and update sets, DDR hints, etc).
  - An annotation processor to flag invalid annotations on the DO
  - Detection of DO changes to prompt for regeneration of CodeGenDoImpl class
  - Generation of ready to run code based on general assumptions using an annotated primary key field(s)

# Agenda

- **Overview of the eBay “world”**
- V4 Presentation framework
- Data Access Layer (DAL)
- µKernel

# What does the µKernel provide?

- Application foundation layer
  - Logging – both server-local and distributed
  - Initialization – runtime dependency and sequence determination
  - Configuration – JMX-based configuration beans
  - Monitoring – status via polling
  - Command Runner – configurable pools of worker threads

# Logging

- Server-local logs
  - Thin layer on top of JDK logging
  - Uses dynamic configuration to control logging level output
  - Pre-configured output destinations
- Distributed logging
  - Centralized publisher gathers transaction data from multiple servers
  - Published on message bus for harvesters that collate and write log files
  - Report generation runs against files for pool-based summary

# Initialization

- System divided into modules
- Each module defines its dependent module
- Each module defines how it is initialized
- Application identifies “top level” modules
- Initialization system walks the tree and initializes modules

# Configuration and Monitoring

- Configuration
  - Embedded JMX Mbean server
  - Lightweight configuration beans bound to JMX categories
  - Change listener and Vetoable Change Listener patterns
- Monitoring
  - Component status can be registered
  - Callbacks to get status of component
  - Monitoring web interface returns data in HTML or XML format

# Command Runner

- Configurable pool of threads for executing work
- Can be dynamically updated
- Provides basic usage statistics
- Logs work as transaction units

# Scalable Innovation Platform Summary

- Java Everywhere works great
- ¼ Billion Hits per day on V4 pages on eBay
- 20-40% faster than pre-V4 pages
- All developers (Pres & Biz) on same language, IDE and tools
- Shopping Comparison page built with V4, DAL, µKernel: From concept to production on eBay site in a week
- Belgium: Micro-project team built Customer Service Application in 6 days
- Scalable + Innovation + Platform = Wow!

# THANK YOU

Dan Douglas, Chris Kasten, Richard Ragan