



# Service-Oriented Programming and Project “Openwings”

# Overall Presentation Goal

Learn more about the newly forming discipline of Service-Oriented Programming and how project “Openwings” embodies this vision



# Session Objectives

- Provide a high level overview of the various service-based architectures
- Discuss how the Service-Oriented Programming concepts and techniques differ from the traditional development approach
- Provide an overview of the Openwings service-based framework
- Demonstration of key service elements



# Speaker's Qualifications

- Pat Vessels is Director of Architecture and Technology Development and Visioneer at Motorola Integrated Systems Division
- Guy Bieber is Lead Architect of Openwings and has previously been Lead Architect of a Military Command and Control Systems product line
- Jeffrey Carpenter is an Openwings Architect, leading Openwings software development effort and several Openwings Expert Teams



# Benefits of This Session

- Greater understanding of what it means to be and think Service Oriented
- A basic understanding of what Openwings and the Openwings Community is
- A better understanding of how to apply Service-Oriented Programming and why it is important



# Presentation Agenda

- Introduction (Pat Vessels)
- Introduction to Service Oriented Programming (Guy Bieber)
- Project “Openwings” (Jeffrey Carpenter)
- Demonstration
- Summary



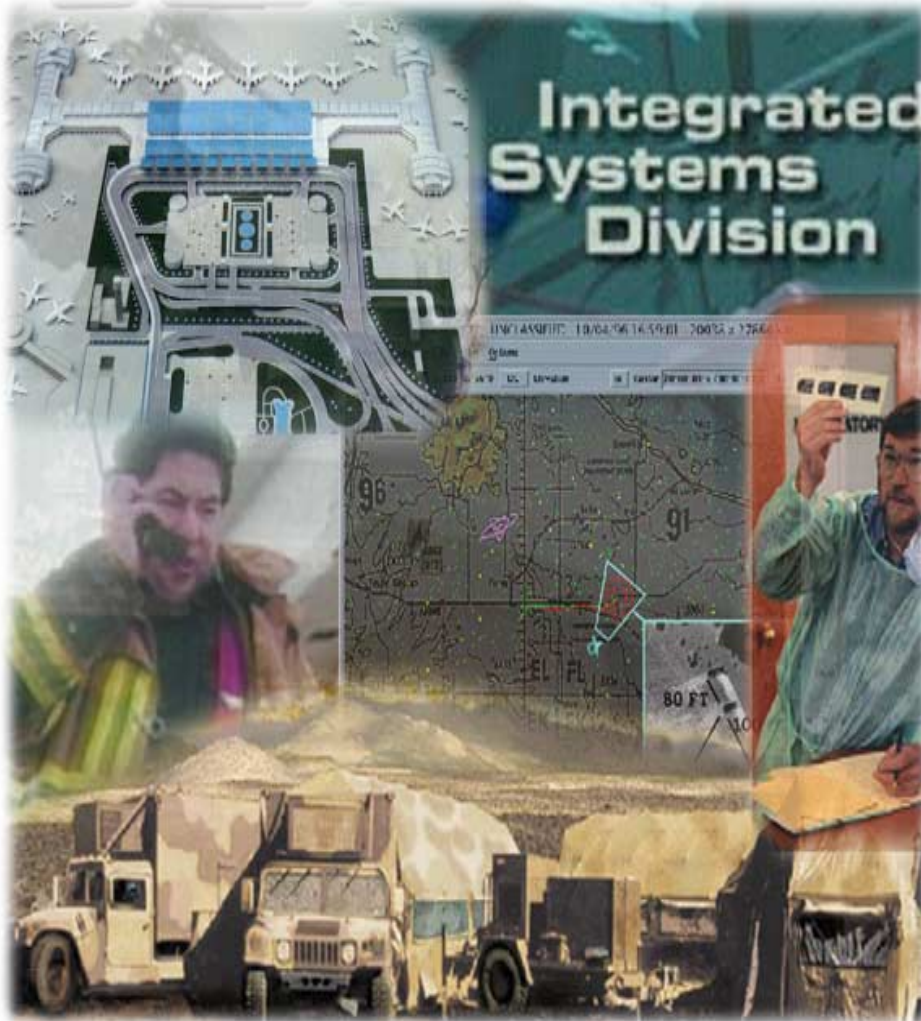


# Introduction

**Pat Vessels**

Director, Architecture and Technology  
Development Motorola, Integrated  
Systems Division

# Background—Our Business



- Key Focus Areas
  - System of Systems Integration
  - Secure Information delivery
  - Multi-Sensor Integration
  - Real-time Situational Awareness
  - Network-Centric Operations

***Delivering the right information to the right people at the right time***



# Business Need

- Issues that drive industry toward a Service-Based Architecture
  - Need for greater software reuse
  - Need for Network-based solutions
  - Standards-based solutions (rather than product-based)
- Issues that drove us to a Service-based architecture
  - Systems of Systems integration and interoperability
    - Interoperability with legacy systems
    - Dynamic, Ad-Hoc integration of systems and components
  - Distributed Operations
    - Need for a distributed architecture that can accommodate elements joining and leaving the network in a ad-hoc fashion
  - The need for a network-centric service architecture to support the DoD's Future Concept of Operations





# Introduction to Service Oriented Programming

**Guy Bieber**  
Lead Architect  
Motorola, Integrated Systems Division

# Service-Oriented Programming

*SOP represents new computer science for today's dynamic networked world*

- **SERVICE**—A service is a contractually defined behavior that can be implemented and provided by any component for use by any component, solely based on the contract
- Service-Oriented Programming is the fulfillment of distributed computing design from the early 90s
- SOP is the evolution of Component Based Architectures
  - Systems are defined by packaging services together (either physically or logically across a network)
  - Erases the traditional boundaries



# SOP Elements

- **Contracts**—An interface that defines the syntax and semantics of a behavior of a service
- **Components**—A third party deployable computing element that is reusable due to independence from its environment
- **Connectors**—An abstraction of a transport mechanism for a particular contract
- **Containers**—An execution environment for components that manages availability and code security
- **Contexts**—A system environment for deploying components that prescribes the details of installation, security, discovery, etc.
- **Discovery**—The technique for locating services

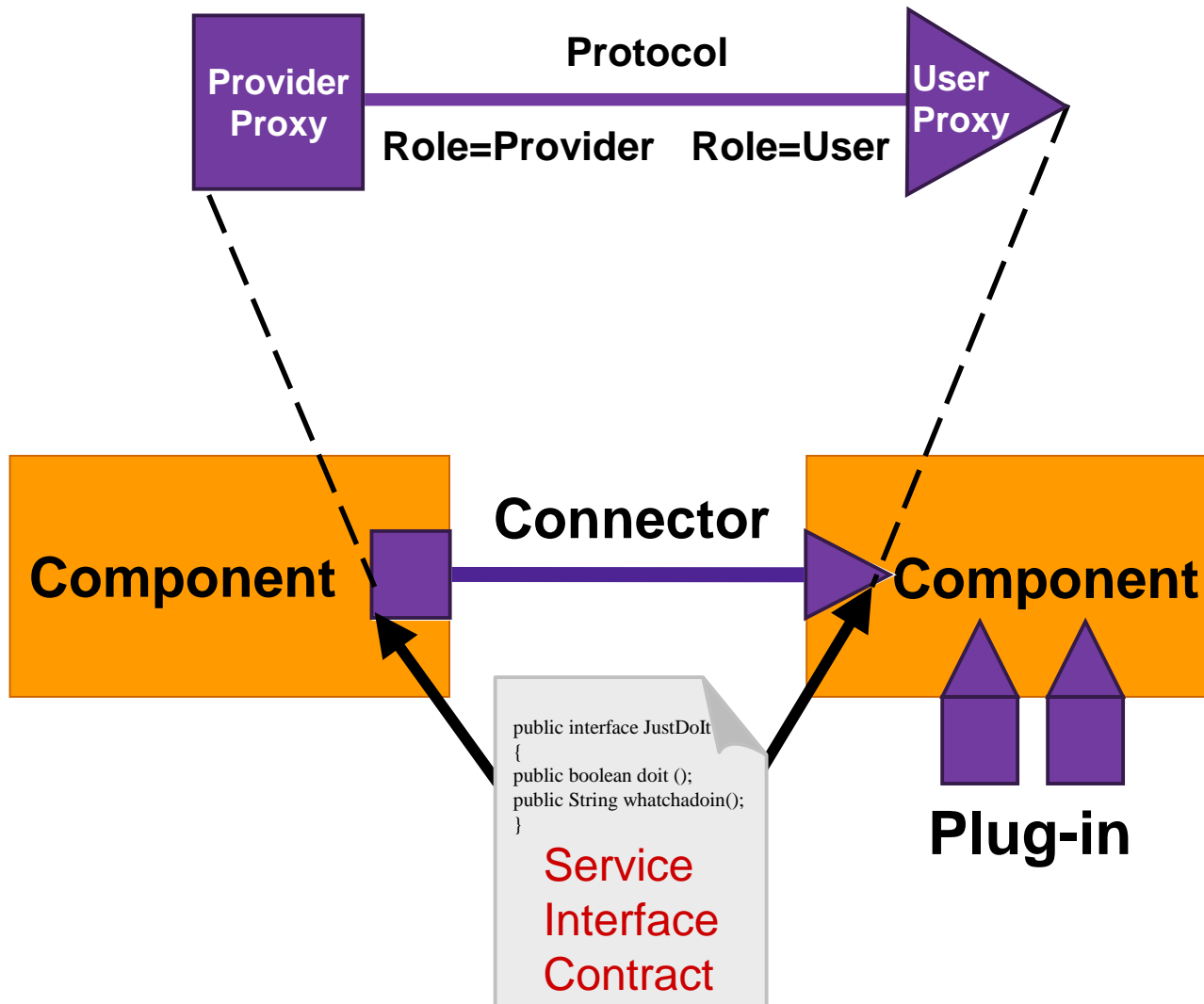


# A Modeling Language for SOP

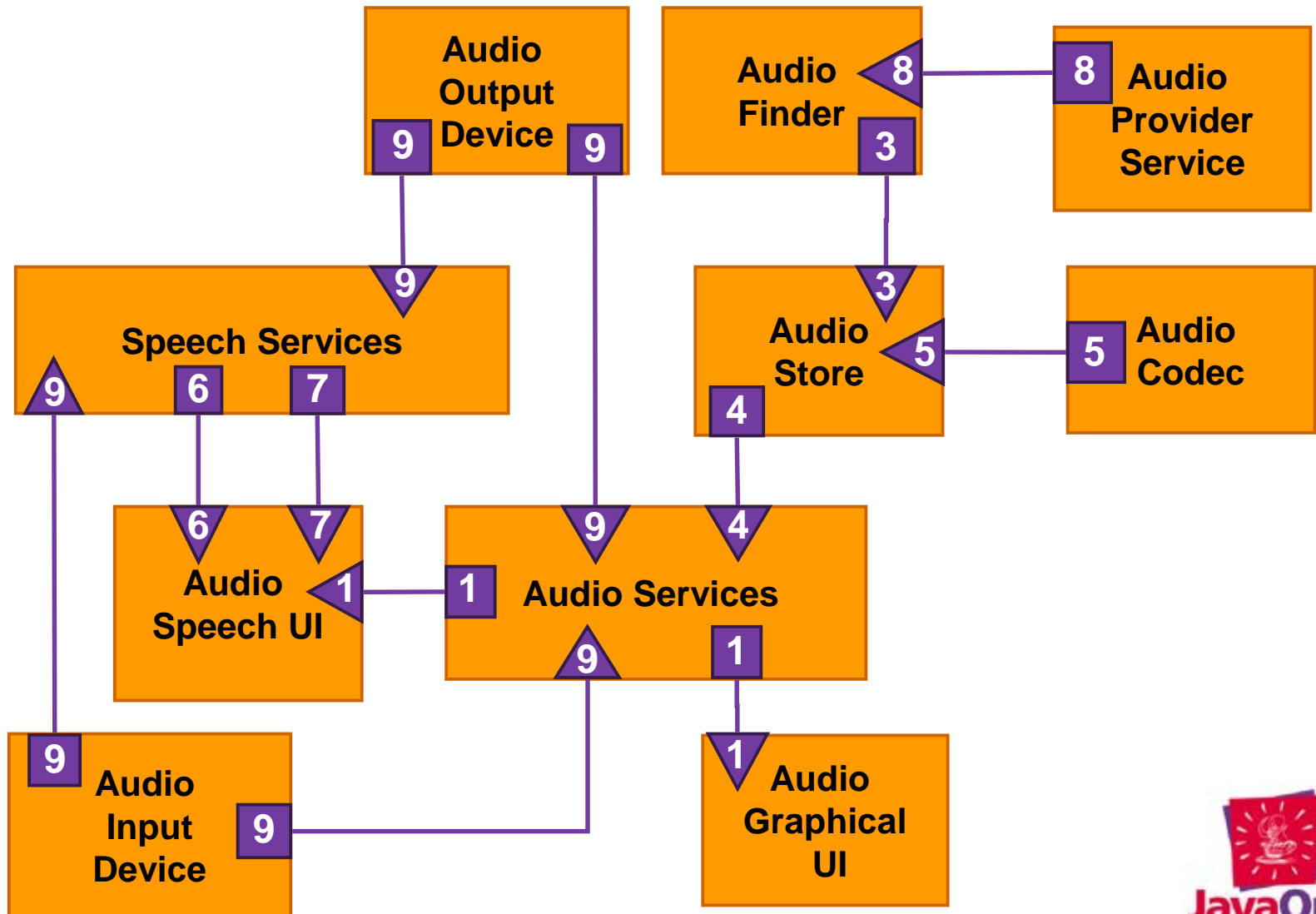
- Object-Oriented Programming
  - Unified Modeling Language
- Service Oriented Programming
  - Architecture Definition Language (CMU)
- Which do you use when
  - Use UML to describe the construction of a component, i.e., how is it built
  - Use ADL to describe interaction between components, i.e., how does it behave



# Architectural Definition Language



# Example ADL Diagram



# SOP Aspects

- **Conjunctive**—Service can be combined in ways not conceived by their authors
- **Deployable**—The ability to deploy components in any environment
- **Mobile/Portable**—The ability for code to work across platforms and to be moved from platform to platform
- **Secure**—The ability secure code, transports, and services
- **Available/Adaptive**—This includes many availability features such as fail-over, rolling upgrades, etc.
- **Interoperable**—Interoperability is enabled by two things: Interfaces and code mobility



# Thinking SOP

- Java™ technology-based Mobile Code and Interfaces are the enablers of interoperability for SOP
- Think in terms of services (behaviors defined by interfaces) not in terms of particular implementations
- Don't hard code protocols
- Don't hard code endpoints, discover services
- Design for availability

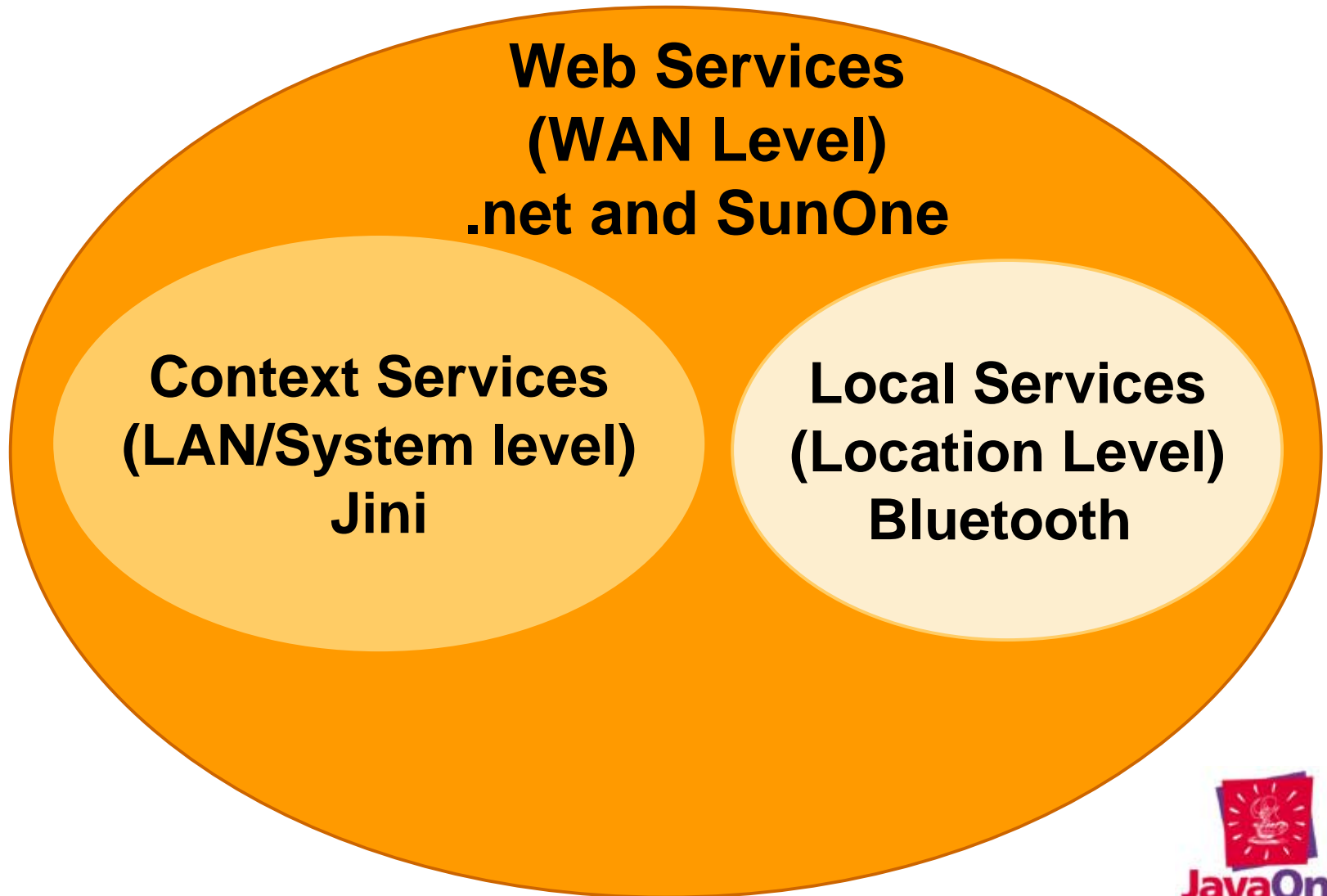


# Thinking SOP

- Your components can be deployed in any environment; don't hard code environmental details
- Self Describing Services—Use common interfaces; publish your interfaces
- Interfaces, connectors, service implementations, and service User interfaces are independently deployed components



# Service-Oriented Technologies and Frameworks



# Web Services—.NET vs. Sun™ ONE

Element	.NET	Sun ONE
Contracts	WSDL	WSDL (JSR 109)
Connectors	no (SOAP)	EJB JSF (JSR111)
Components	no	EJB, JSF
Containers	.net server	EJB, JSF
Context	no	no
Discovery	UDDI	UDDI (JSR 110) (web search)

*Microsoft Meets XML and Sun Responds*



# Bluetooth Services

<b>Element</b>	<b>Bluetooth</b>
<b>Contracts</b>	<b>Profile / Service Class</b>
<b>Connectors</b>	<b>Bluetooth Protocols (OBEX, PPP, etc.)</b>
<b>Components</b>	<b>no</b>
<b>Containers</b>	<b>no (Bluetooth devices)</b>
<b>Context</b>	<b>partial (ad hoc locality based)</b>
<b>Discovery</b>	<b>Service Discovery Protocol (peer to peer)</b>



# Jini™ Network Technology: Services

<b>Element</b>	<b>Jini Technology</b>
<b>Contracts</b>	<b>Java Interface</b>
<b>Connectors</b>	<b>no (RMI / JRMP)</b>
<b>Components</b>	<b>no</b>
<b>Containers</b>	<b>no</b>
<b>Context</b>	<b>partial (Jini Federation)</b>
<b>Discovery</b>	<b>Distributed Repository (step 1- discover lookup service, step 2- lookup desired service)</b>



# Services Are Services: Why Not One Model: Openwings

<b>Element</b>	<b>Openwings</b>
<b>Contracts</b>	<b>Java Interface</b>
<b>Connectors</b>	<b>Plug-in (async-jms, soap, etc.; synch-rmi, corba, etc.)</b>
<b>Components</b>	<b>yes</b>
<b>Containers</b>	<b>yes (generic handles any Java app)</b>
<b>Context</b>	<b>yes</b>
<b>Discovery</b>	<b>plug-in (Jini, .net, Bluetooth, etc.)</b>





# Project “Openwings”

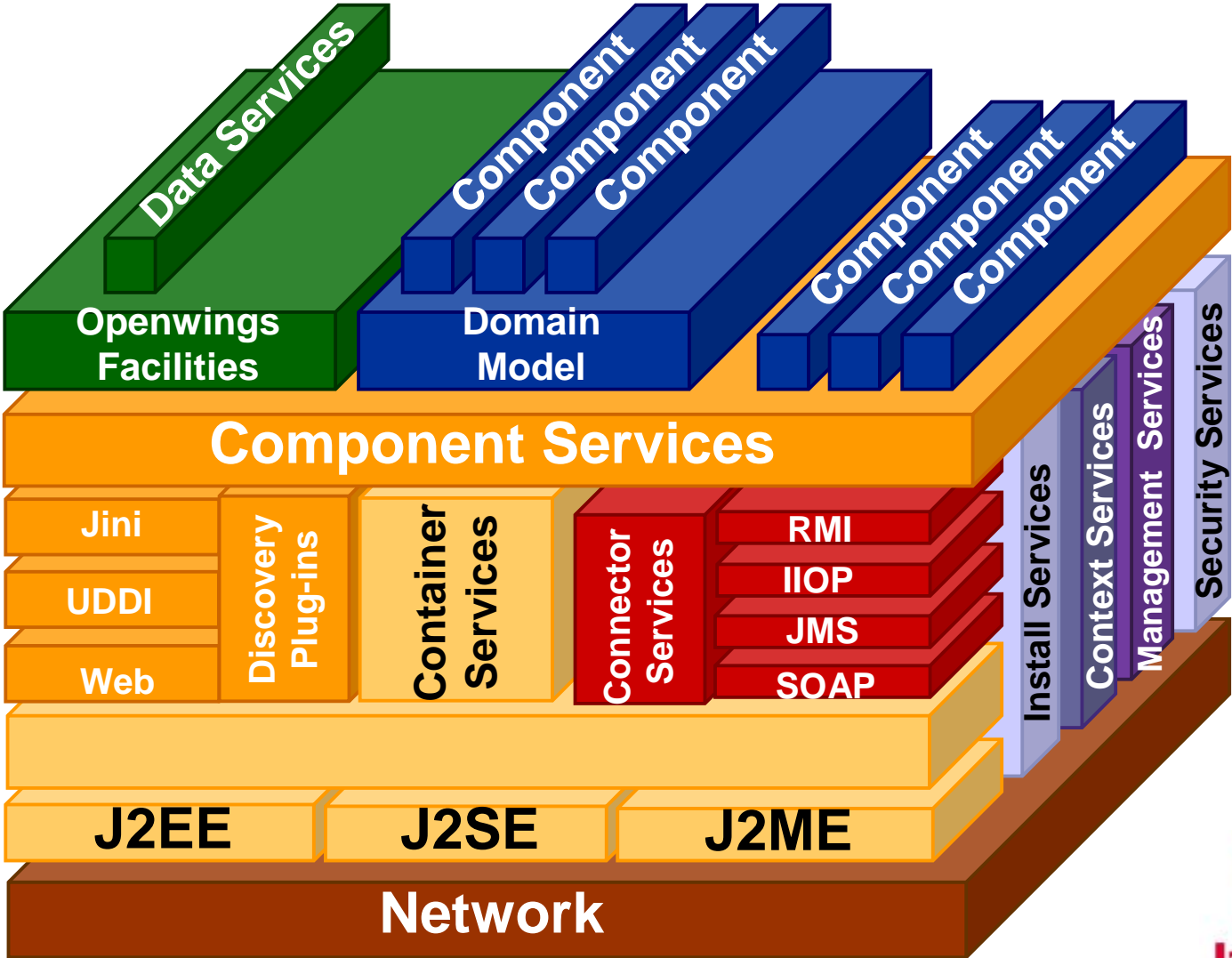
**Jeff Carpenter**  
Senior Software Engineer  
Motorola, Integrated Systems Division

# What Is Project “Openwings”?

- Openwings is an open systems framework that provides the constructs needed to build Service-Oriented systems
- Openwings ties together the other SOP technologies and frameworks
- Openwings enables the development of highly available, secure, distributed systems for mission critical applications



# Openwings Architecture



# Openwings Services

<b>Component Services</b>	Provides the infrastructure for locating desired services and providing services for others to use
<b>Connector Services</b>	Hides the details of inter-process communication allowing applications to be transport-independent
<b>Container Services</b>	Provides services for computational resources to hold components
<b>Context Services</b>	Provides services for system formation



# Openwings Services

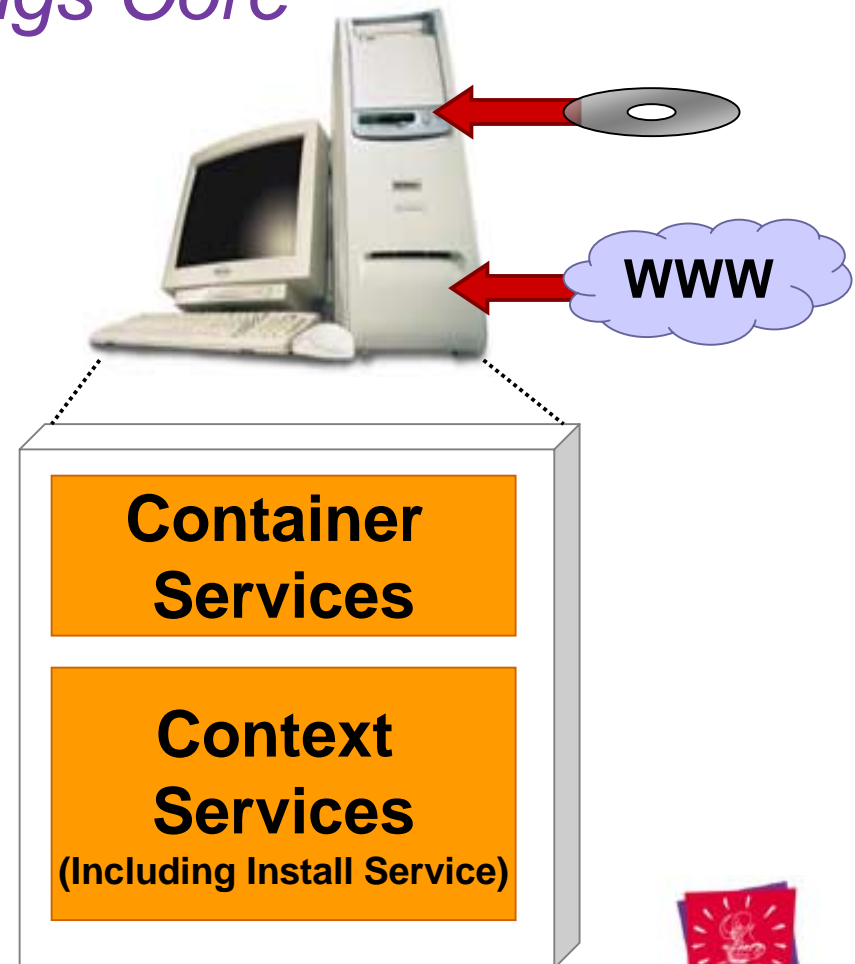
<b>Install Service</b>	Provides a service for installation of components
<b>Management Services</b>	Provides policy-based management to automatically administer systems
<b>Security Services</b>	Provides services for authentication, authorization and privacy
<b>Data Services</b>	Provides a 3-tier model that allows applications to be database independent



# Service-Oriented Scenario

## *Installing the Openwings Core*

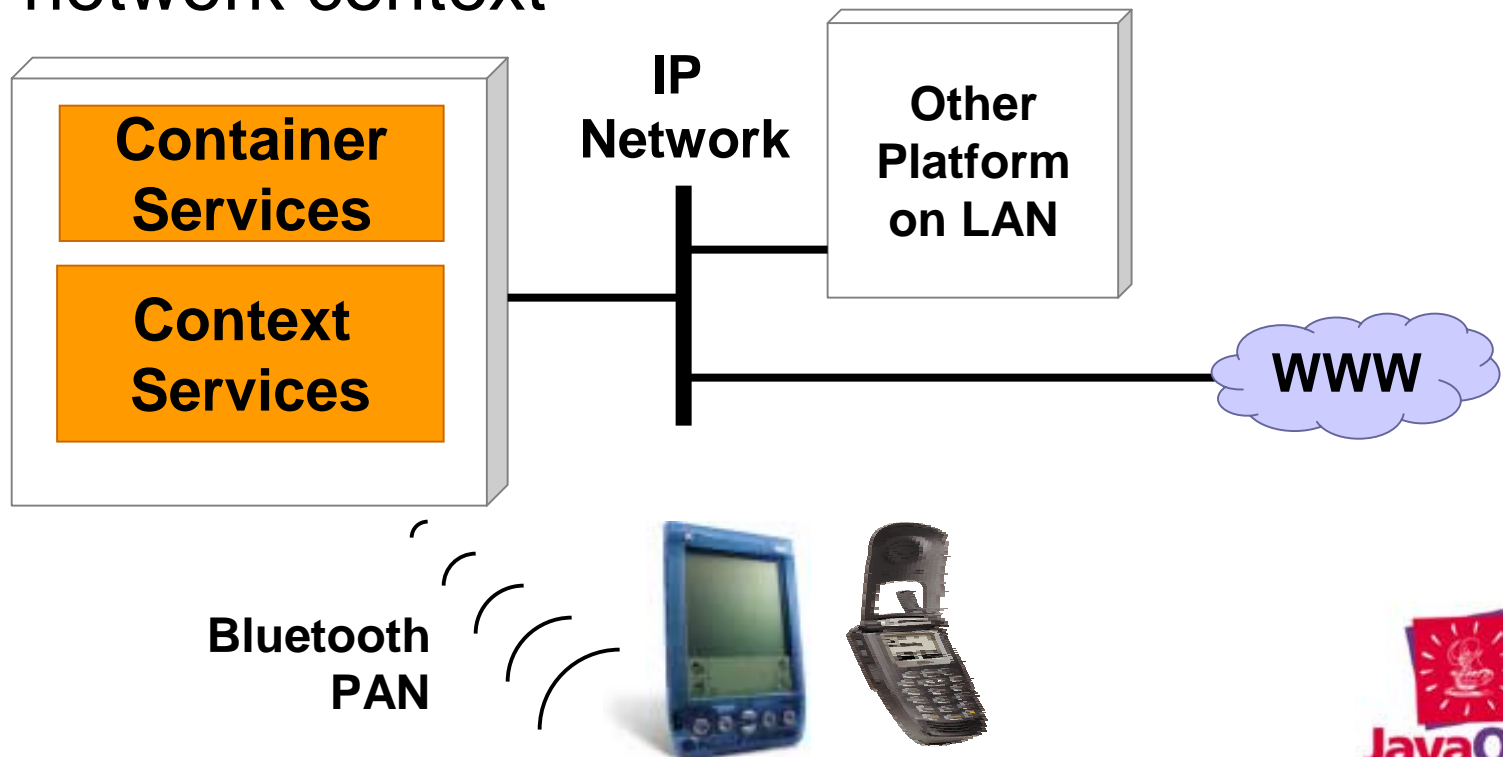
- This scenario shows how the different services of the Openwings architecture can work together to form service-oriented systems...
  - First, Openwings is installed on every platform, either from the web or from media
  - The Core services are started, including Context Services, Container Services and any supporting infrastructure



# Service-Oriented Scenario

## *Forming the Network*

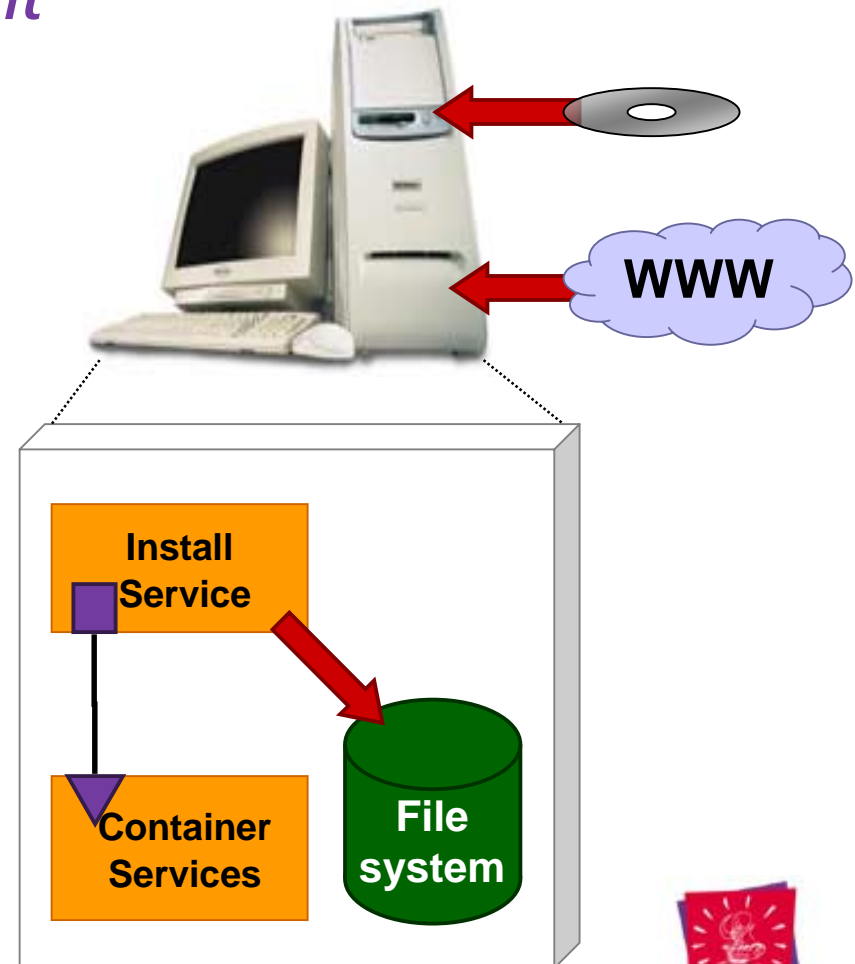
- When the platforms are connected via the network, Context Services forms the network context



# Service-Oriented Scenario

## *Installing a Component*

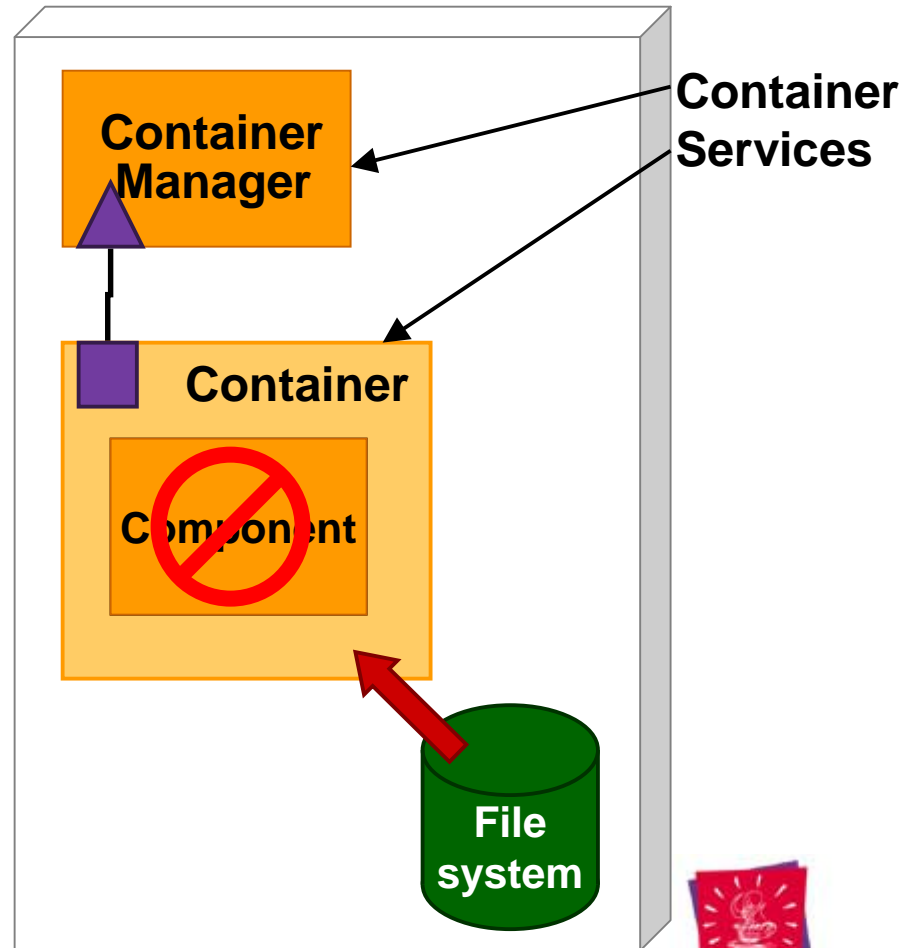
- Install a Component by plugging media into a platform or clicking on a web link
  - The Install Service detects the Component image
  - The Install Service reconciles the Component's default policies and creates the Component image on disk
  - The Install Service notifies Container Services of the installed Component



# Service-Oriented Scenario

## Running a Component

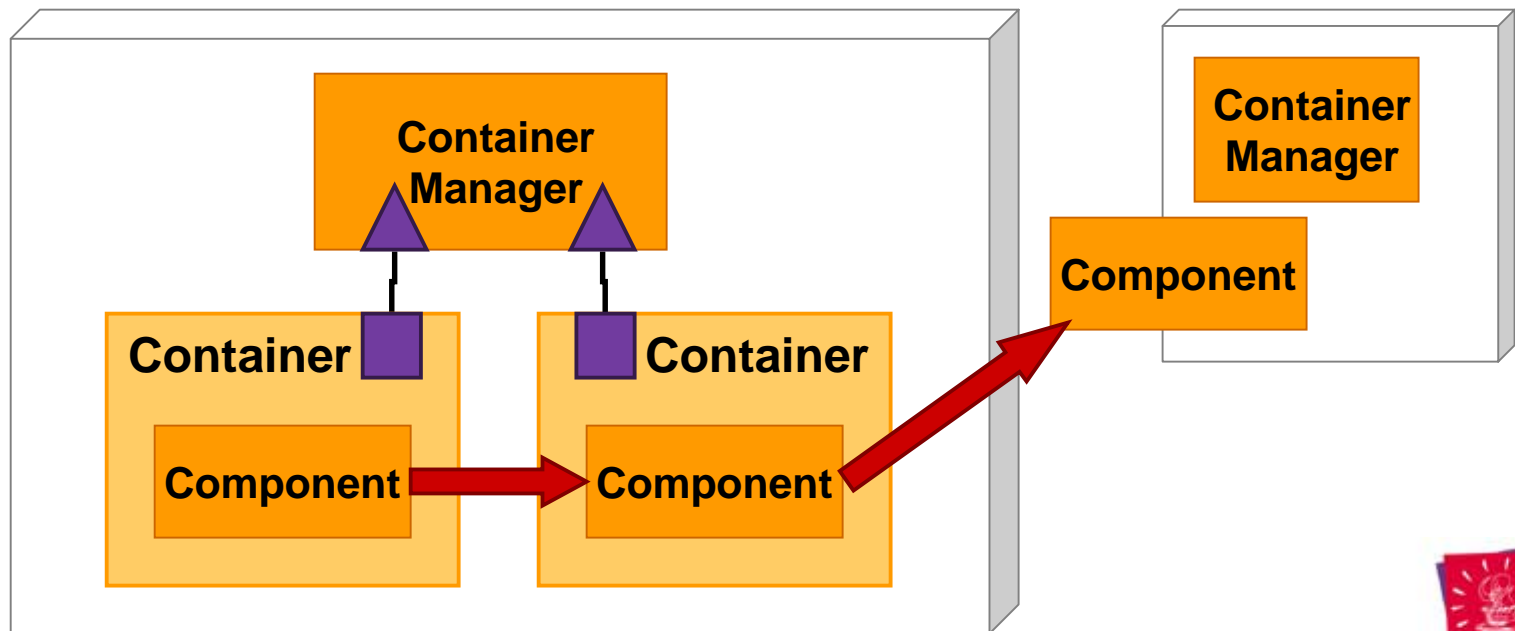
- Container Services manages the runtime lifecycle of the Component
  - The Component is started in a Container (at boot time or on user's request) from the image stored on disk
  - The Container enforces security policy for the Component code
  - The Component is restarted if it fails



# Service-Oriented Scenario

## *Moving a Component*

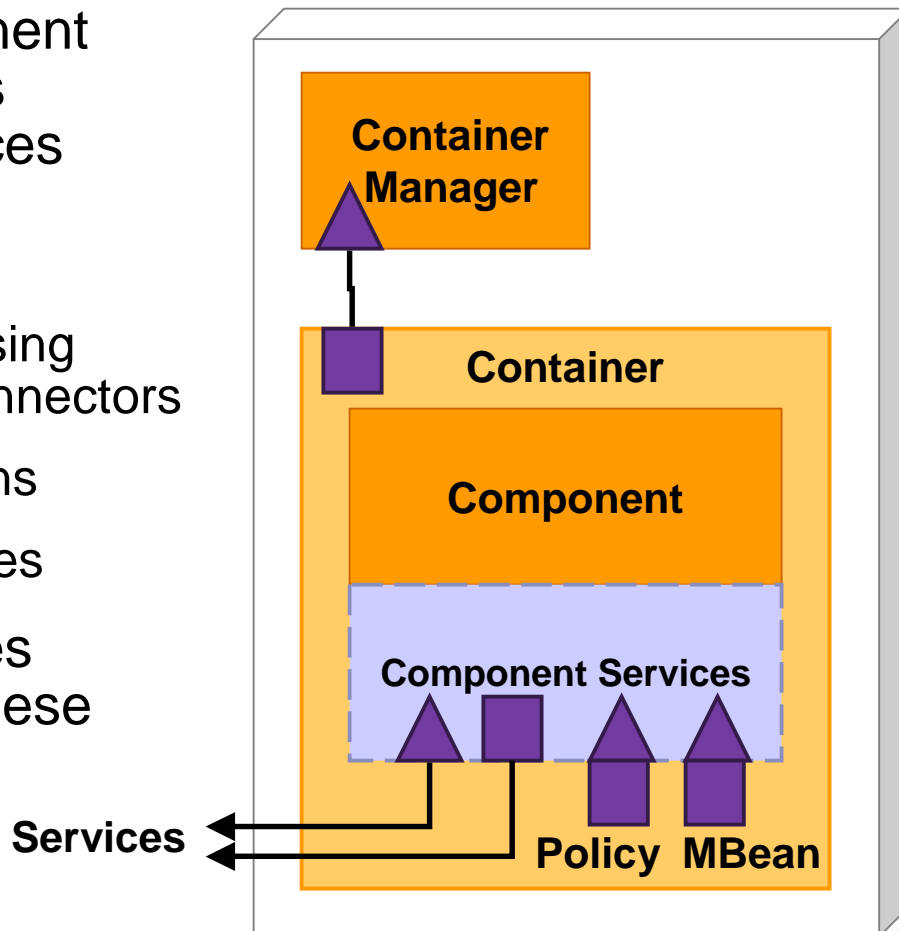
- Container Services moves Components between Containers and platforms for load balancing



# Service-Oriented Scenario

## *A Component Interacts With Its Environment*

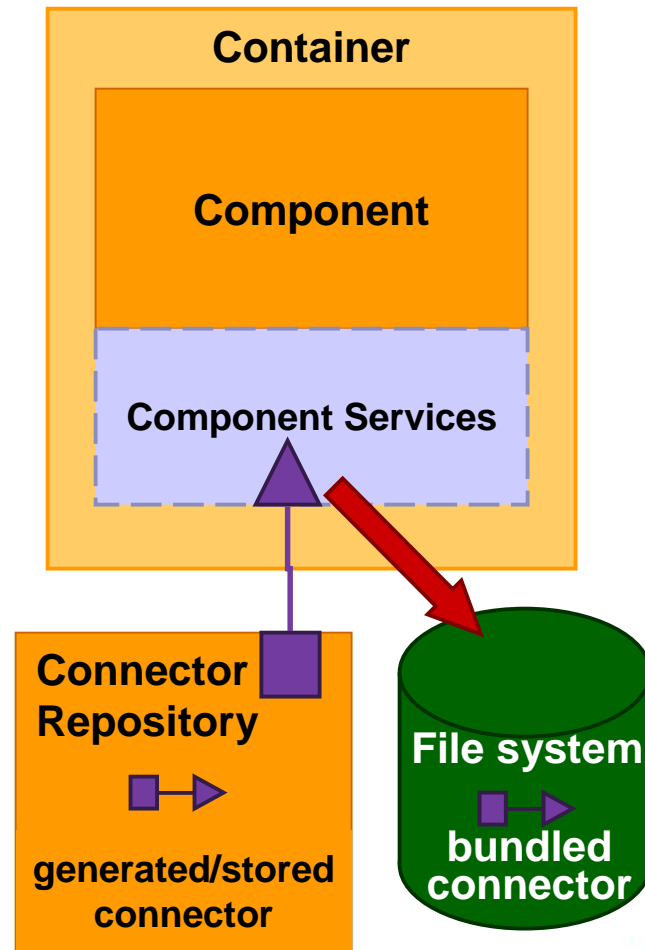
- When the Component starts up, it utilizes Component Services to interact with its environment
  - Providing and using services with connectors
  - Exposing MBeans
  - Accessing policies
- The following slides develop each of these interactions



# Service-Oriented Scenario

## *A Component Provides a Service*

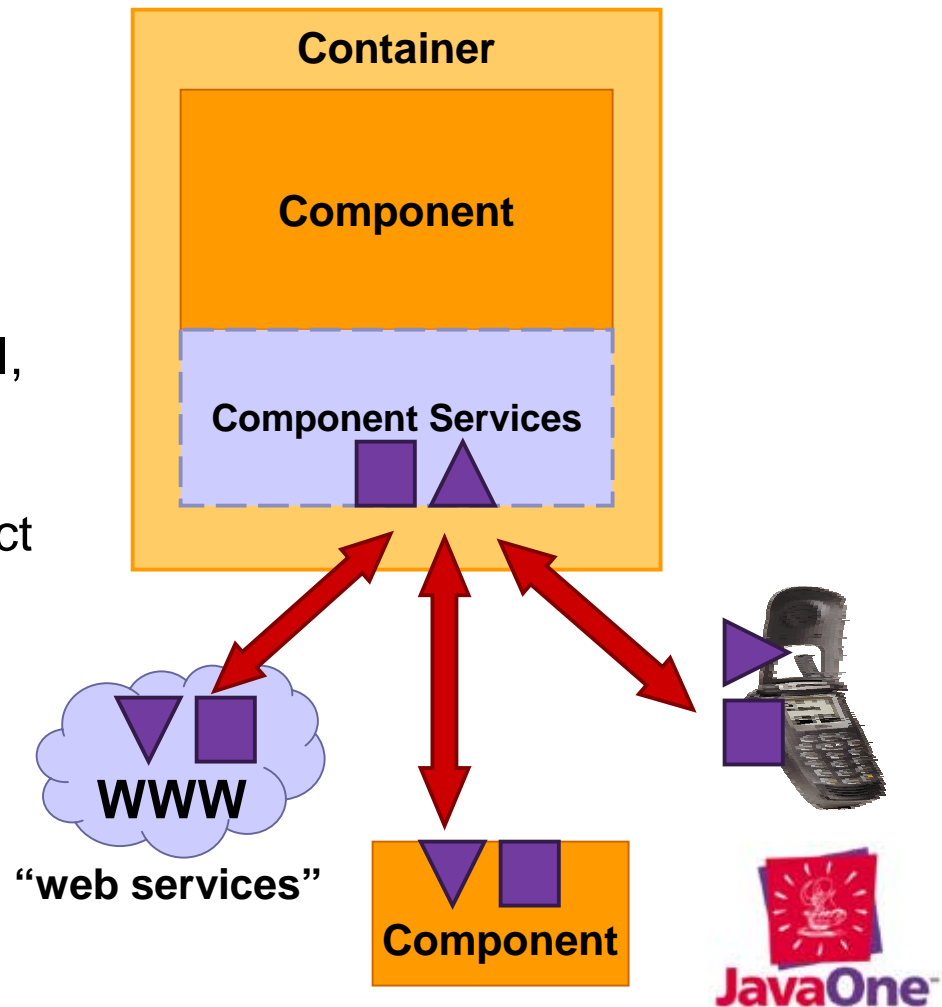
- The Component utilizes Component Services to provide services
  - Component Services uses Connector Services to implement communications
  - Component Services can use connectors bundled with components, or request them from a Connector Repository
  - The user proxy of the connector is published as the service object
  - Secure connectors use encryption



# Service-Oriented Scenario

## *A Component Provides and Uses Services*

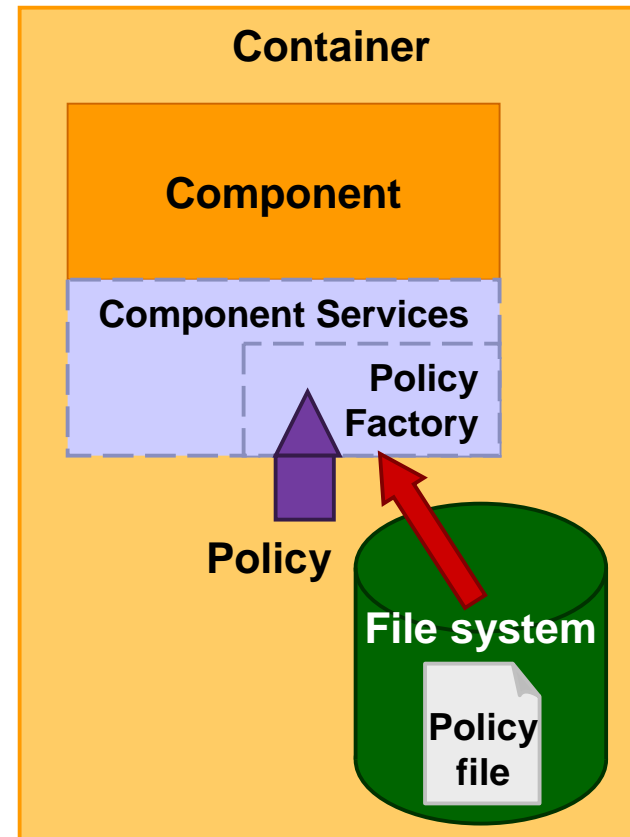
- The Component utilizes Component Services to provide and use services
  - Component Services provides/uses services via Jini, Bluetooth, UDDI, or whatever discovery plug-ins are available
  - Using a service is subject to Role-Based Access Control (RBAC)



# Service-Oriented Scenario

## *A Component Accesses Configuration Data*

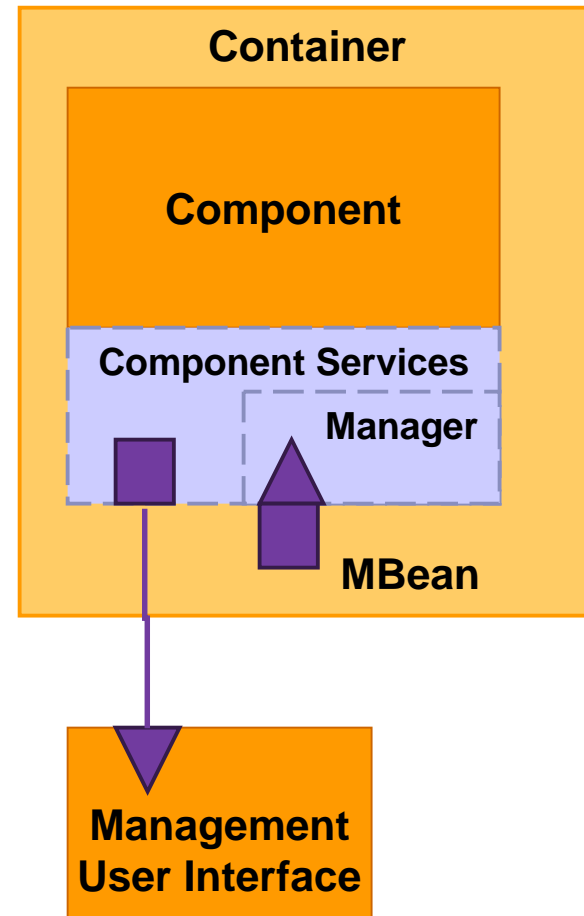
- The Component uses Policy Services to access its configuration data
  - The Component obtains a PolicyFactory object through Component Services
  - The PolicyFactory object loads stored configuration data from the installed image of the component



# Service-Oriented Scenario

## *A Component Exposes Manageable Aspects*

- The Component utilizes Management Services to expose its dynamically changeable aspects
  - The Component obtains a Manager object through Component Services
  - The Component registers MBeans with the Manager
  - Component Services provides the Manager as a service



# Openwings Expert Team Status

	OSR Approved	Expert Team Formed	Participant Draft	Participant Review	Public Review	Beta Release	Maintenance
<b>Systems</b>	★	★					
<b>Security</b>	★	★					
<b>Component</b>	★	★					
<b>Connector</b>	★	★					
<b>Container</b>	★	★					
<b>Context</b>	★	★					
<b>Management</b>	★	★					
<b>Data</b>	★						

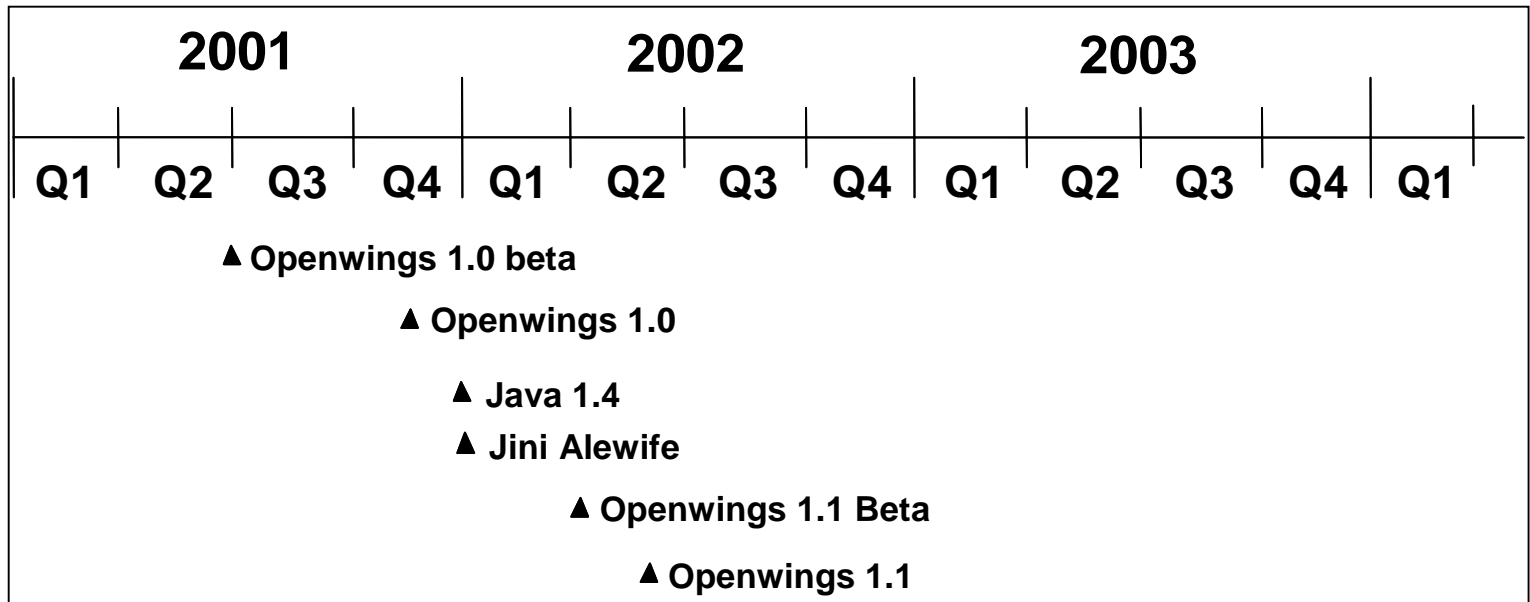


# Openwings News

- Motorola's Openwings Compliant Implementation
  - Support for .NET and OpenNet
  - Support for Bluetooth Discovery
- Partnerships
  - Sun and SIGNAAL—Real-time Extensions
  - Mercury—High Performance Processing



# Openwings Roadmap



**Java 1.4—XML object translation, logging, IPv6, JVM Sharing**

**Jini Alewife—Surrogate, ServiceUI, Multicast Configuration**

**Java Futures—WSDL, UDDI, JSF, J2ME RMI, USB, Bluetooth,  
Isolation API**

**Jini Futures—RMI Security, Security**





# Demonstration



# Summary

# Summary

- Service-Oriented Programming is the next big revolution in Software Engineering
- Learning SOP is much like learning OOP
- Openwings is an open standard that bridges the emerging SOP programming frameworks together into a complete and interoperable system





**JavaOne**<sup>SM</sup>  
Sun's 2001 Worldwide Java Developer Conference

**Q&A**

# Are You Thirsty for More?

- Check out our website  
[WWW.OPENWINGS.ORG](http://WWW.OPENWINGS.ORG)
- Become a Openwings Community Member
- Sign up for an Expert Team
- Get the Beta—Coming soon
  - It's Free





# JavaOne<sup>SM</sup>

Sun's 2001 Worldwide Java Developer Conference