JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

Jini™ Network Technology-Enabled Service-Oriented Architecture

A Low-Cost Alternative to Enterprise JavaBeans™ (EJB™) Architecture

java.sun.com/javaone/sf

Leon Chism/Steve Hoffman

Chief Internet Architect/Engineering Fellow

Orbitz

www.orbitz.com

Goal

Increase your understanding of Jini™ network technology distributed computing architectures, how they can facilitate competitive advantage, and how they can be used to replace/augment more expensive alternatives.

Orbitz's Commitment to Jini™ Network Technology

Or why am I listening to you?

- Orbitz runs over 1300 Jini™ services used by over 300 client VMs
- Orbitz' Jini™ network technology-based architecture powers Orbitz.com, AA.com and NWA.com, making it one of the highest-volume processors of airline tickets
- Jini™ services have achieved 99+% uptime (best uptime of all sub-systems at Orbitz)

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

Distributed Computing Pitfalls

What's Next?

Wrap up

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

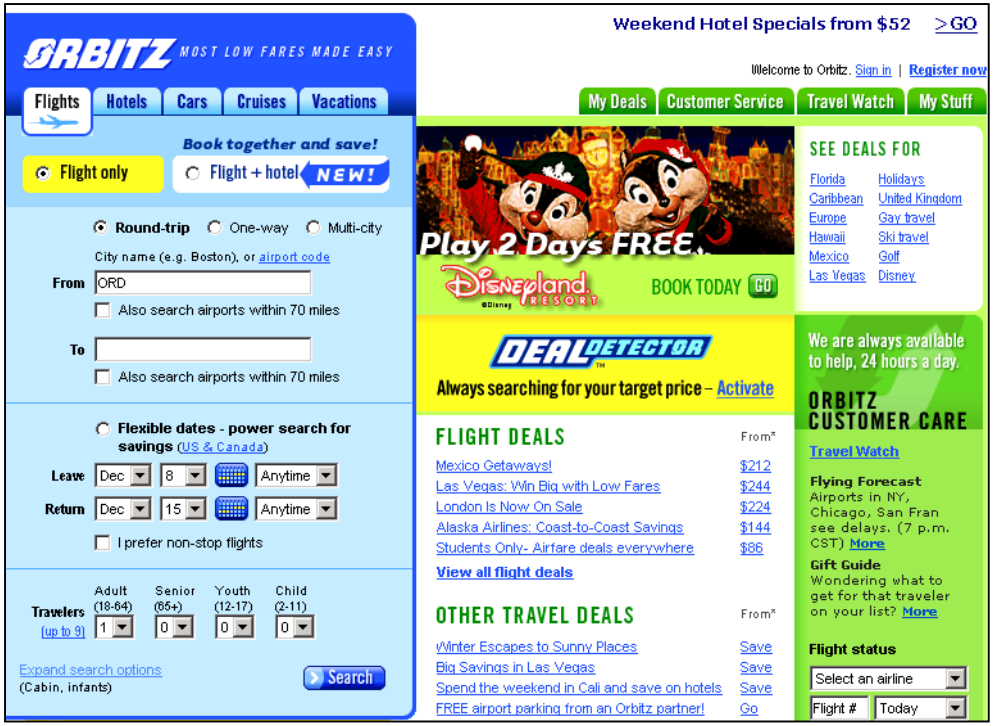
Distributed Computing Pitfalls

What's Next?

Wrap up

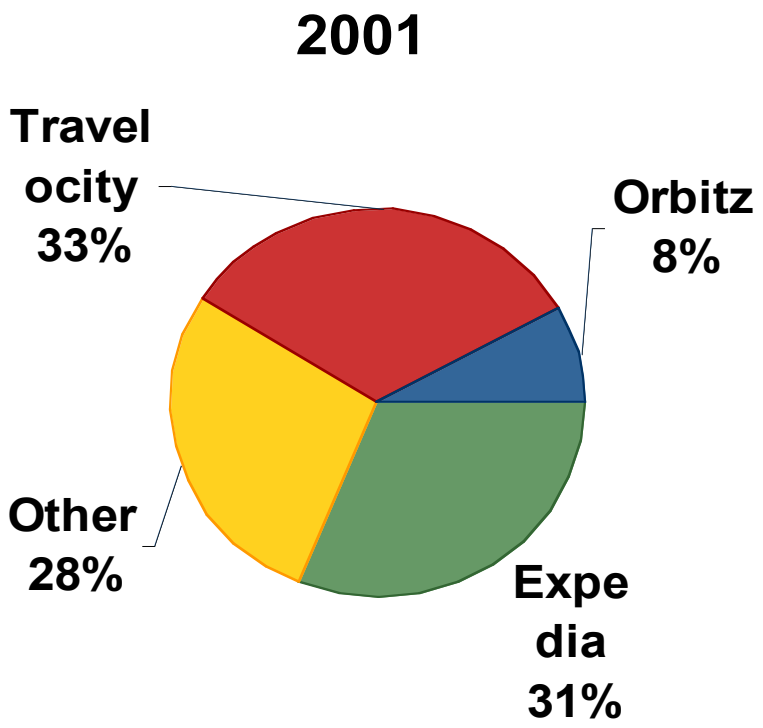
Orbitz Overview

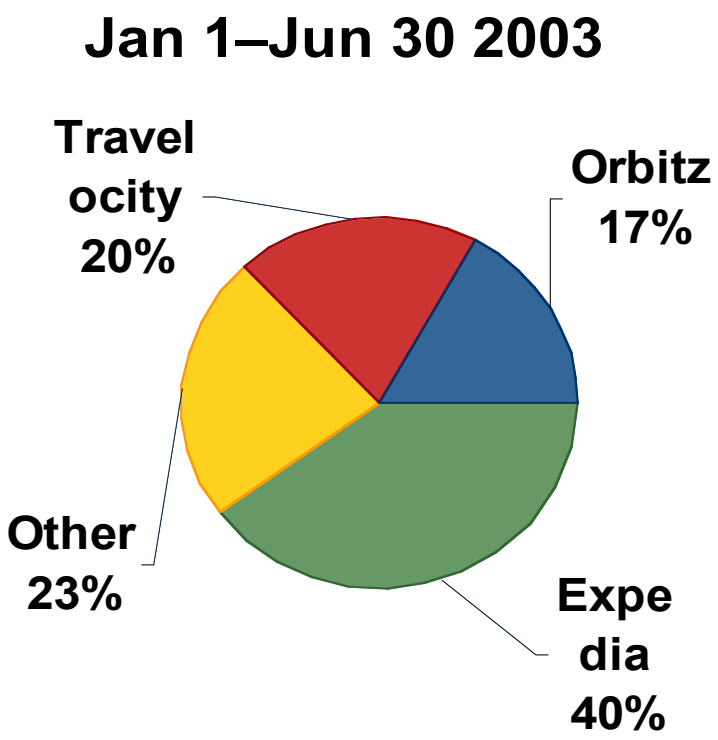
- Founded in 2000 by 5 leading airlines
- Site launched in 2001
- Top 3 brand travel site in just over two years
- 22 MM registered users and strong monthly traffic
- Poised to extend strength in air to other products
- Positive cash flow from operations since Q4 2002
- Public offering on 12/17/03



Rapidly Approaching #2 Position

Based on gross travel bookings





Source: *PhoCusWright*

Architectural Drivers

- Low-cost distributor
- Speed to market is critical
- Flexibility of the application
- Robustness
  - Availability
  - Reliability

Why Use Jini™ Network Technology?

- Enables automatic failover and recovery
- Scales horizontally
- Lookup capabilities
  - Typed
  - Interface-based
- Functionality didn't require transactions
- Automatic load balancing
- Self-healing
- Low-cost (no license costs)

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

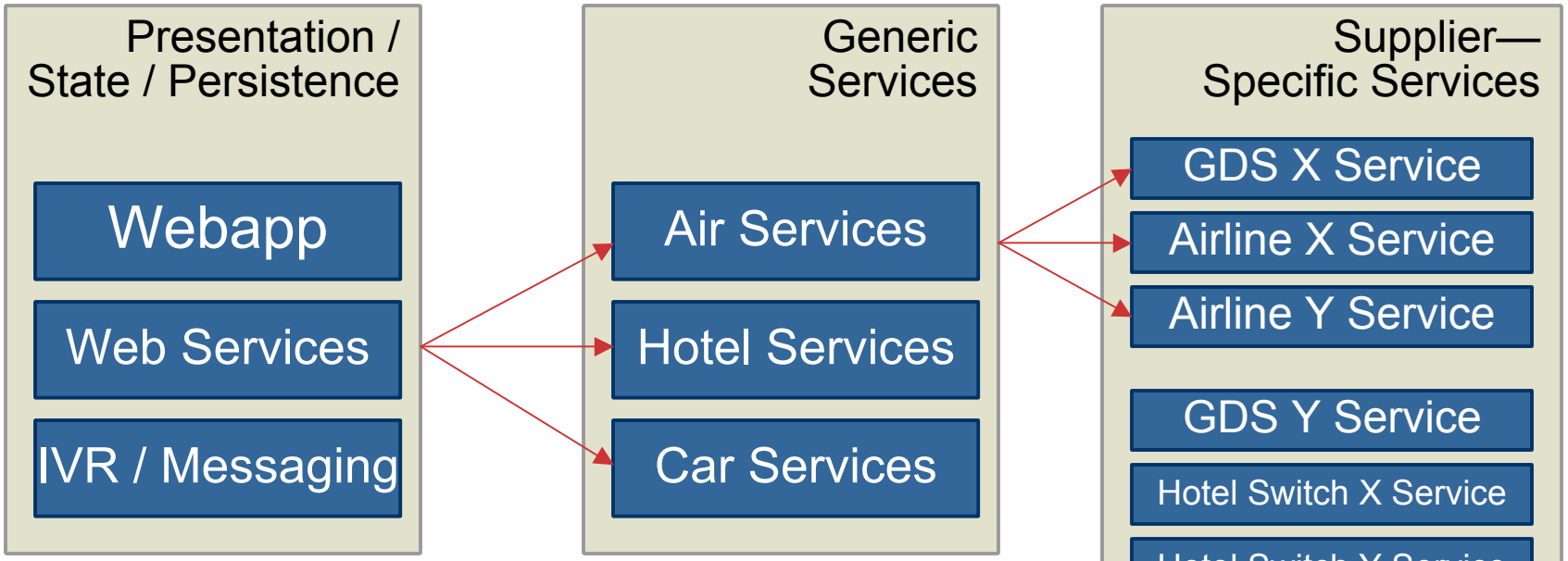
Distributed Computing Pitfalls

What's Next?

Wrap up

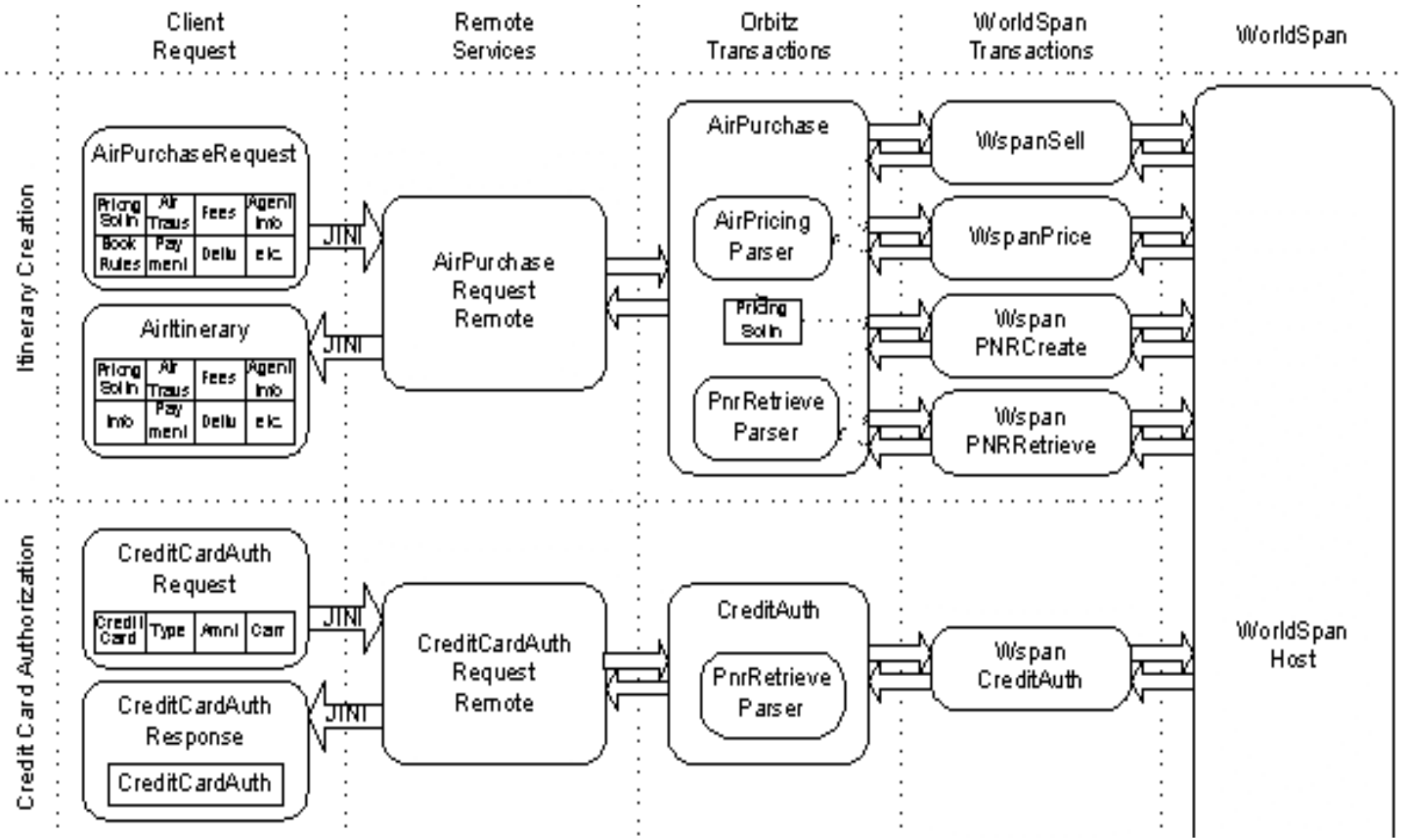
Orbitz Architecture

Mile-high view



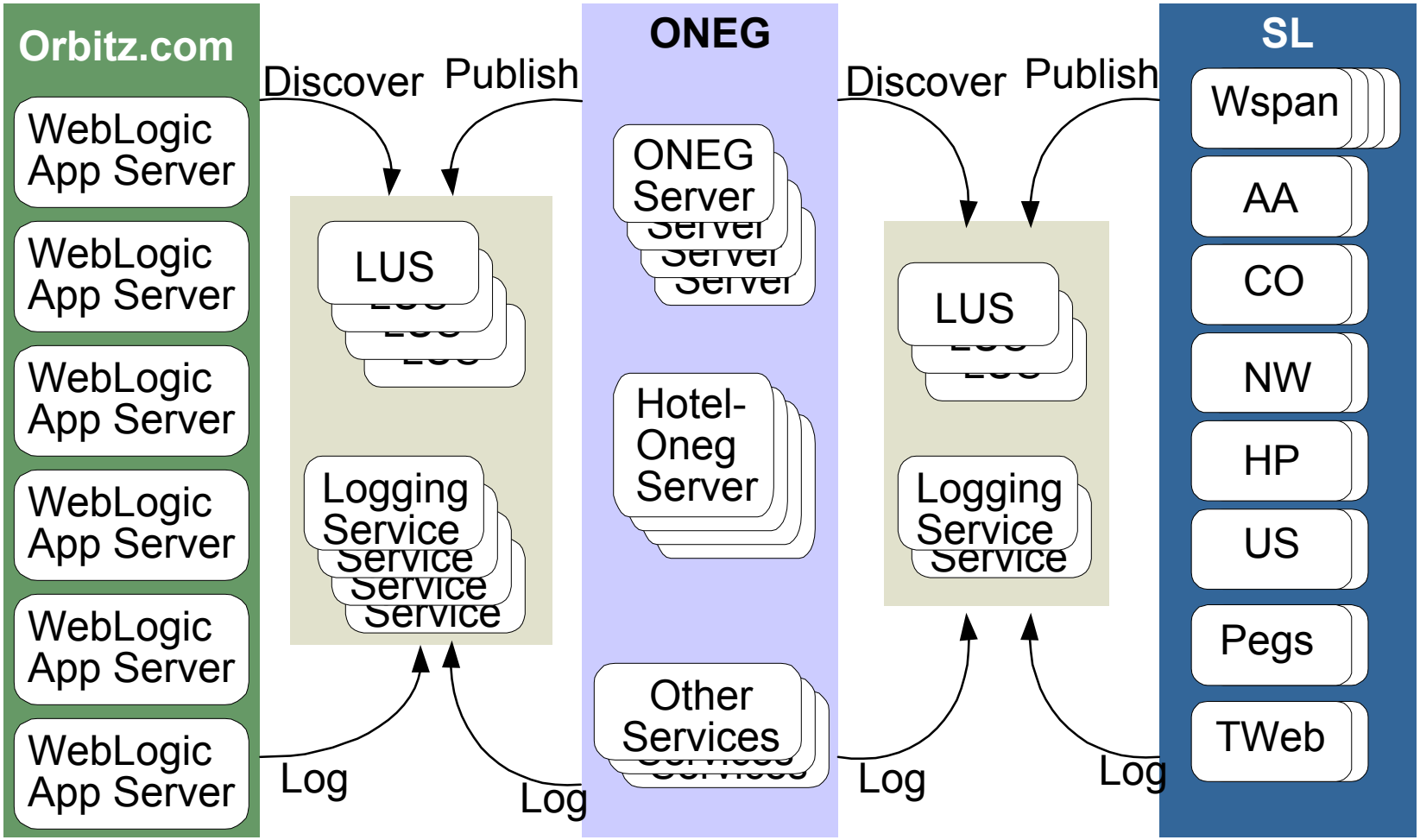
Supplier Link

Single host system (i.e. AA.com/NWA.com)



ONeG—The Big Switch

Putting it all together

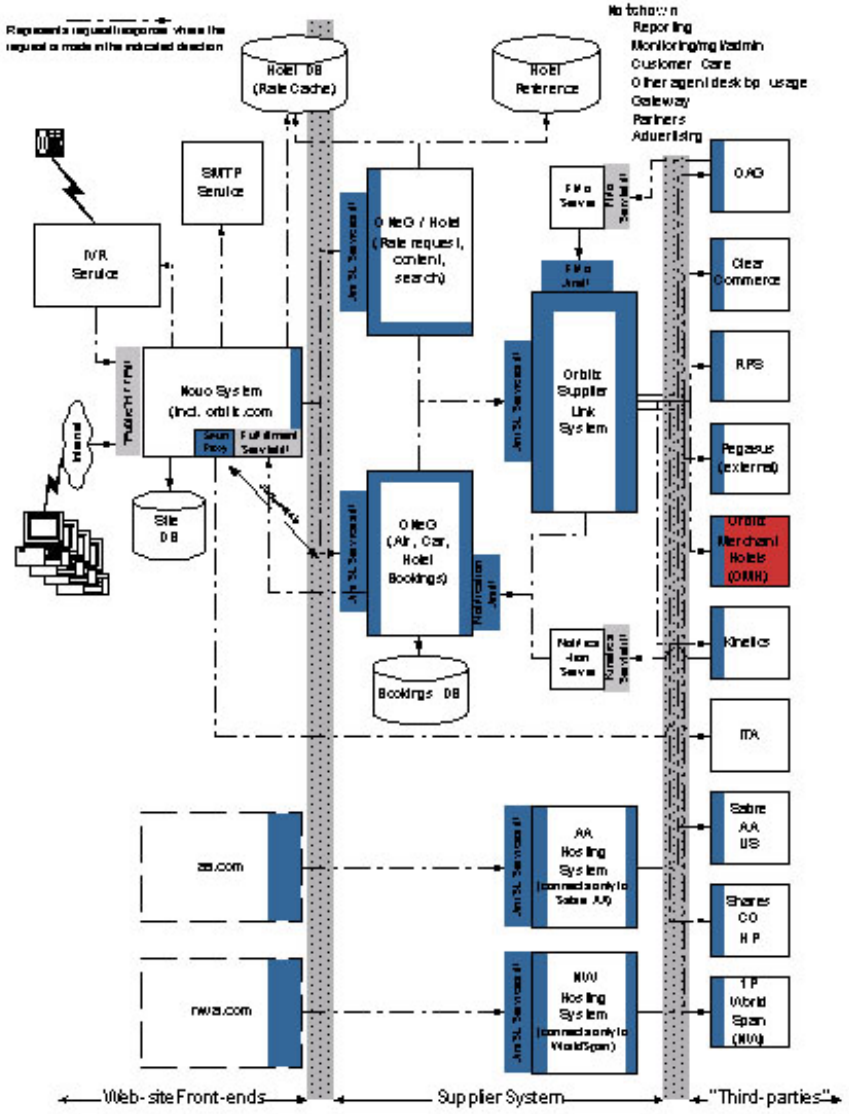


ONeG—Design Decisions

Switch or router?

- Client could have contacted host specific services directly
- Was decided to put failure/retry logic in the booking engine, not push on the client due to hot reconfiguration requirements
- Business event logging at ONeG layer
- Use network layer security to restrict access to supplier specific layer

High-Level Architecture Diagram



J2EE

Jini

The General Idea

Combine common set of objects, service interfaces and Jini[SM] services to create blocks of reusable functionality.

The General Idea (Cont.)

- Common objects
  - **`AIRPORT`**
  - **`CARRIER`**
- Service interfaces
  - **`LowFareSearchRequest`**
  - **`AirPurchaseRequest`**
- Mask framework details with simple Factory
  - **`ServiceImpl = OrbitzFactory.get(serviceInterface);`**
- Simple configuration (at least for client)
  - VersionAttribute **`String`**
  - List of lookup servers OR list of multicast groups

The General Idea (Cont.)

Things to note:

- Response (and objects contained) are core objects or interfaces are immutable
- Underlying implementations typically have much more information than is exposed to client
- Services are all transient and non-activatable hence completely interchangeable
- No transactions

Let's See Some Code!

Client's perspective:

- Startup

```
ClientStarter.start();
```

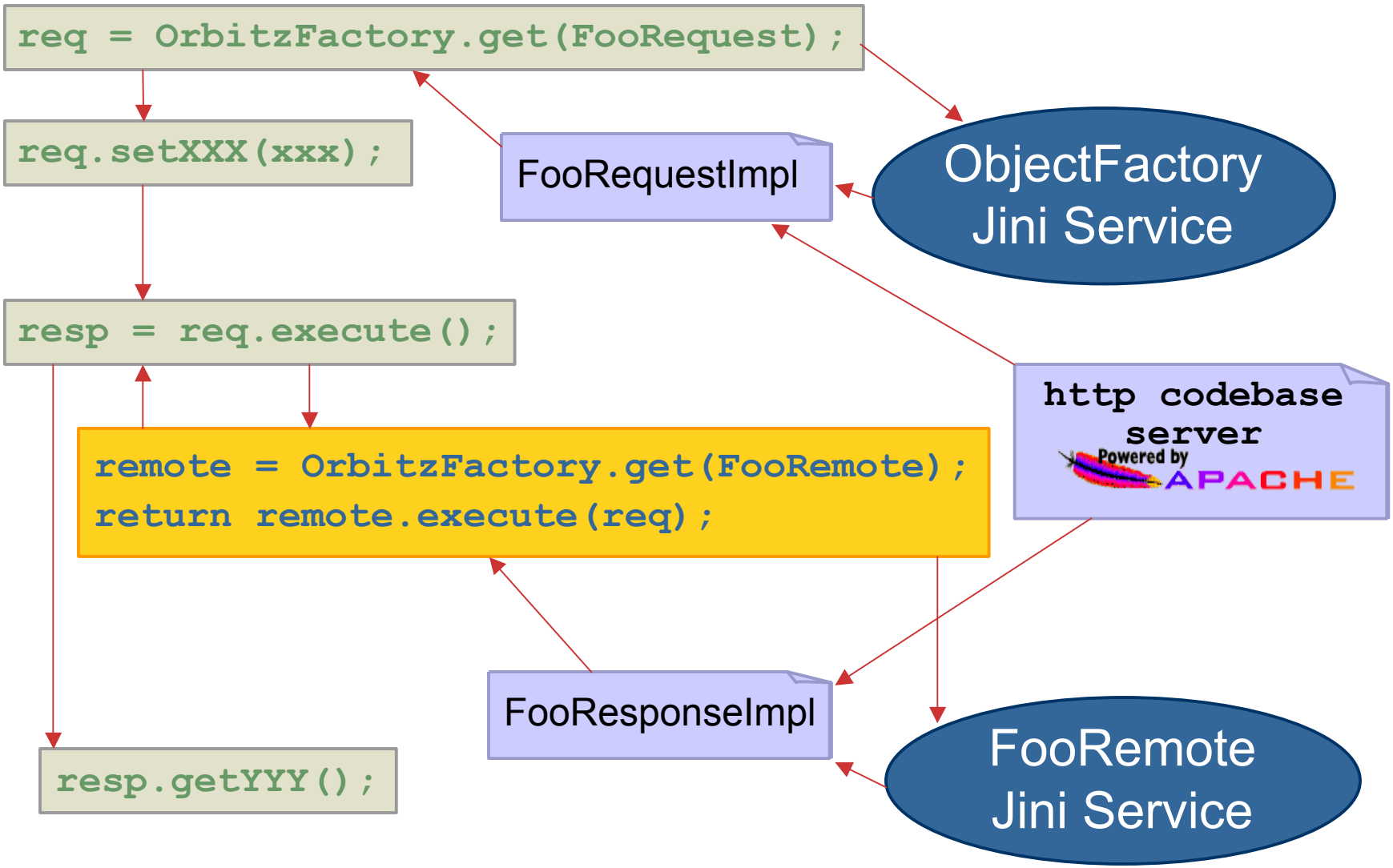
- Shutdown

```
ClientStarter.stop();
```

- Sample transaction

```
SampleServiceRequest req =
  OrbitzFactory.get(SampleServiceRequest.class);
req.setFoo(foo);
req.setBar(bar);
SampleServiceResponse resp = req.execute();
baz = resp.getBaz();
```

Say What? Show Me a Picture….



Let's See More Code!

Publishing a Service (Jini 1.X)

- Startup

```
ServerStarter.start();

SampleService ss = new SampleServiceImpl();
   // SampleServiceImpl extends UnicastRemoteObject
   // Need to run rmic to generate _Stub class

DiscoveryManagment ldm =
   ServerStarter.getDiscoveryManager();
   // Most likely LookupDiscoveryManager

JoinManager jm = new JoinManager(ss, attributes,
   null, ldm, null);
   // attributes is an Entry[] which includes
   // things like VersionAttribute
```

Let's See More Code!

Publishing a Service (Jini 1.X) (cont.)

- Shutdown

```
UnicastRemoteObject.unexport(ss, false);
jm.terminate();
ServerStarter.stop();
```

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

Distributed Computing Pitfalls

What's Next?

Wrap up

Jini™ Technology Benefits

Self-healing

- Designed to expect network failure (not ignore that it happens)
- Multicast discovery mechanism of Lookup Server ensures <1 minute discovery (heartbeat)
- Services come and go. Have many places to go to get serviced

Jini™ Technology Benefits

Automatic load balancing

- **`LookupCache`** and **`ServiceDiscoveryManager`** use random selection when single service is requested and there are multiple matches
- Over time this results in an even load distribution of load on identical services

Jini™ Technology Benefits

Horizontal scalability

- Capacity is directly related to the number of places a client can go to fulfil a service request
- If you need more capacity of a particular service, just start more instances
- Each **`LookupCache`** gets notified of new service and load is redistributed across new set

Jini™ Technology Benefits

Lightweight

- Easy to publish a service
- Easy to find a service
- Catch **`RemoteException`** and discard proxy when something goes wrong

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

Distributed Computing Pitfalls

What's Next?

Wrap up

Distributed Computing Pitfalls

RMI Distributed Garbage Collection

- We don’t use remote objects
- No way to turn it off in JRMP implementation of RMI (Jini versions 1.X)
- Default 1 minute DGC interval kills any heap optimizations
- Accounts for anywhere between 50-75% of the threads in our system

Distributed Computing Pitfalls

RMI DGC—possible remedies

- Make DGC interval longer or disable explicit GC (we saw a 1.5% improvement in VM throughput when using **`-XX:+DisableExplicitGC`**— time spent running code vs doing GC)
- Upgrade to Jini 2.X and use JERI so we can disable DGC

Distributed Computing Pitfalls

Reggie lockouts

- Contributed implementation designed to wait until things aren't changing before answering
- Implemented with big reader's/writer lock
- Not common, but annoying when LookupCache/ServiceDiscoveryManager drops all discovered services when lease to lookup server cannot be renewed

Distributed Computing Pitfalls

Reggie lockouts—possible remedies

- Longer lease times
- Custom implementation of lookup server that doesn't have reader's/writer lock. Won't get *best* picture of state of services, but isn't necessary in our system

Distributed Computing Pitfalls

Randomized load balancing

- If a machine gets slow, random selection will eventually put all requests on the slow machine
- Also a problem in heterogeneous environments

Distributed Computing Pitfalls

Randomized load balancing—possible remedies

- Switch from sync. calls to async so a push to a service becomes a pull. Use a JavaSpaces™ or JMS™ queue
- Publish **`Attribute`** with notion of relative capacity in conjunction with **`ServiceItemFilter`** on client side to adjust spread of service calls more in-line with machine's relative abilities

Distributed Computing Pitfalls

Event storming using the LookupCache

- LookupCache uses event notification to keep local VM cache up to date
- If large number of clients/lookup servers/services, the math gets big—e.g.:
  - 100 app servers (service clients) * 2 lookup servers * 30 services/VM * 20 VMs = 120,000 messages
- Contributed implementation of Lookup Server does in-order delivery of notifications which involves a linear scan of a single list of messages to send out

Distributed Computing Pitfalls

Event storming—remedies

- Custom **`LookupCache`** that doesn't use event notifications:
  - Every client probably doesn't need more than 3-4 places to go for a particular service type at any time
  - Auto-expire entries based on time or number of calls to get same randomization
  - Backfill with more services as current ones become invalid
- Custom Lookup Server with event module optimized for throughput of in-order event notification

Distributed Computing Pitfalls

Event storming—remedies (cont.)

- Coarser grained services
  - —Group services living in single VM (usually due to resource sharing *i.e.* DB pool) into single (or fewer) Jini™ service(s). Fewer event notifications to be sent
- Partitioning of service registrations
  - —Group service registrations to reduce the number of messages sent by a single instance of the Lookup Server

Jini™ and J2EE™

Two great tastes that go great together

- Have Jini environment, but want to introduce some J2EE™ servers (or vice-versa). What do you do?
- Deploy webapp that publishes Jini™ service interface similar to EJB™ interface, but service impl makes EJB local call
- Jini service lookups take the place of JNDI lookups
  - Multiple copies give same result as clustered directory service
  - Can live side-by-side with other APIs: XML/SOAP, JNDI EJB lookup, etc.

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

Distributed Computing Pitfalls

What's Next?

Wrap up

What's Next?

Jini 2.X upgrade

- JERI
  - Disable distributed garbage collection
  - Plugable transports (UDP, SSL)
- Security (per service)
  - Authorization
  - Authentication
  - Replace 2 layers of Jini services with 1 now that we can isolate/restrict access to services individually

Let's See More Code!

Publishing a Service (Jini 2.X)

- Startup

```
ServerStarter.start();

SampleService ss = new SampleServiceImpl();
   // SampleServiceImpl implements Remote

Exporter e = new BasicJeriExporter(
   TCPServerEndpoint.getInstance(0),
   new BasicILFactory());

Remote stub = e.export(ss);

DiscoveryManagment ldm =
   ServerStarter.getDiscoveryManager();

JoinManager jm = new JoinManager(stub, attributes,
   null, ldm, null);
```

Let's See More Code! (Cont.)

Publishing a Service (Jini 2.X)

- Shutdown

```
e.unexport(stub);
jm.terminate();
ServerStarter.stop();
```

What's Next? (Cont.)

Dynamic redeployment

- If Jini services use downloaded code, why can't all the code be downloadable?
- Introduce standard deployment platform utilizing benefits of transportable byte-code
  - Reconfigure capacity as needed
  - Ease deployment across many machines
- Interesting work along these lines:
  - Rio—http://rio.jini.org/
  - Jini Service Container—http://chiron.jini.org/

What's Next? (Cont.)

Asynchronous messaging

- Perhaps change from sync-client-push to async-server-pull model
  - Using JavaSpaces™
  - Using JMS™ queue
- Would keep servers from getting overloaded since they wouldn't pull more work than they could handle
- Could have a more heterogeneous environment of hardware while maximizing utilization
- Downside is introducing a single point of failure if the space/queue isn't *really* fault-tolerant

Agenda

Introduction

Orbitz Architecture

Jini™ Technology Benefits

Distributed Computing Pitfalls

What's Next?

Wrap up

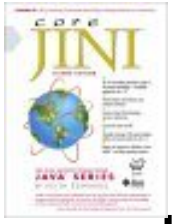
Summary

- Try Jini™ services its:
  - A great way for distributed applications to discover and communicate with each other
  - Lightweight, flexible, adaptable, self-healing
  - Production ready
  - The best kept secret in the Java™ community program
- J2EE™ platform is not the solution for *every* problem

For More Information

- http://www.orbitz.com/
- http://www.jini.org/
  - Strong and helpful community group. Presentations from 7th community meeting are online for free
  - Mailing list/archives
  - Download Jini 2.0

- http://research.sun.com/techrep/1994/abstract-29.html (A Note On Distributed Computing)
- *Core Jini*, W. Keith Edwards
  - ISBN: 0130894087 (Jini 1.X)



- *Hard Landing*, Thomas Petzinger Jr.
  - ISBN: 0812928350 (Airline lore)

Q&A

JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

Jini™ Network Technology-Enabled Service-Oriented Architecture

A Low-Cost Alternative to Enterprise JavaBeans™ (EJB™) Architecture

java.sun.com/javaone/sf

Leon Chism/Steve Hoffman

Chief Internet Architect/Engineering Fellow

Orbitz

www.orbitz.com

Sun microsystems