Son of SOA
Resource-Oriented Computing
Event-Driven Architecture

Eugene Ciurana
Director, Systems Infrastructure
LeapFrog Enterprises, Inc.

eugenex@leapfrog.com
pr3d4t0r @ irc://irc.freenode.net ##java, #esb, #awk, #security

About Eugene

- 15+ years of experience building mission-critical, high-availability systems infrastructure
- 12+ years of Java work
- Open-source evangelist
  - Official adoption of open-source / Linux at Wal-Mart Stores
  - State-of-the-art tech for main-line of business roll-outs
- Engaged by the largest companies in the world
  - Retail
  - Finance
  - Oil industry

AUGUST 27, 2007 VOL. 24, NO. 27 • $6
eWEEK
www.eweek.com
Jason Brooks: Windows goes Xen—by proxy. 57
Leaping into Open Source
Eugene Ciurana, LeapFrog director of systems infrastructure
eWEEK ROAD MAP
BY JASON BROOKS
Best-of-breed planning approach leads LeapFrog Enterprises to open source 31
NEWSPAPER PERIODICAL
enterprise

What You Will Learn...

- How to develop complex apps within very tight deadlines
- Formalize integration around a resource-oriented model
- Develop event-driven apps based on existing production tech and services
- Turn SOA-based systems into callbacks as an evolution of the provider/consumer model
- Define application processing in terms of compositions and asynchronous sequences of resource requests

So... What is the Problem?

- Very tight deadlines
  - Typical 12-month project rolled out in 90 days
- Development team built at the same time as application design work
- No history of developing Web applications
- Rigid IT infrastructure and policies
  - SOX and other compliance issues
  - IT guys used to rule the world
- Integration with financial and other legacy systems is a must

Advantages

- Very tight deadlines!
  - We gotta do what we gotta do...
- Dev team grows at the same time as design work proceeds
  - Technology adoption driven by team member selection and viceversa
- Very few legacy issues to deal with in Web applications
  - Adoption of best-of-breed technology from open-source community
- IT doesn’t do Web systems
  - Technology adoption policy evolves along with design and development
- No need to reinvent the wheel for existing systems
  - Financial, CRM model, etc.

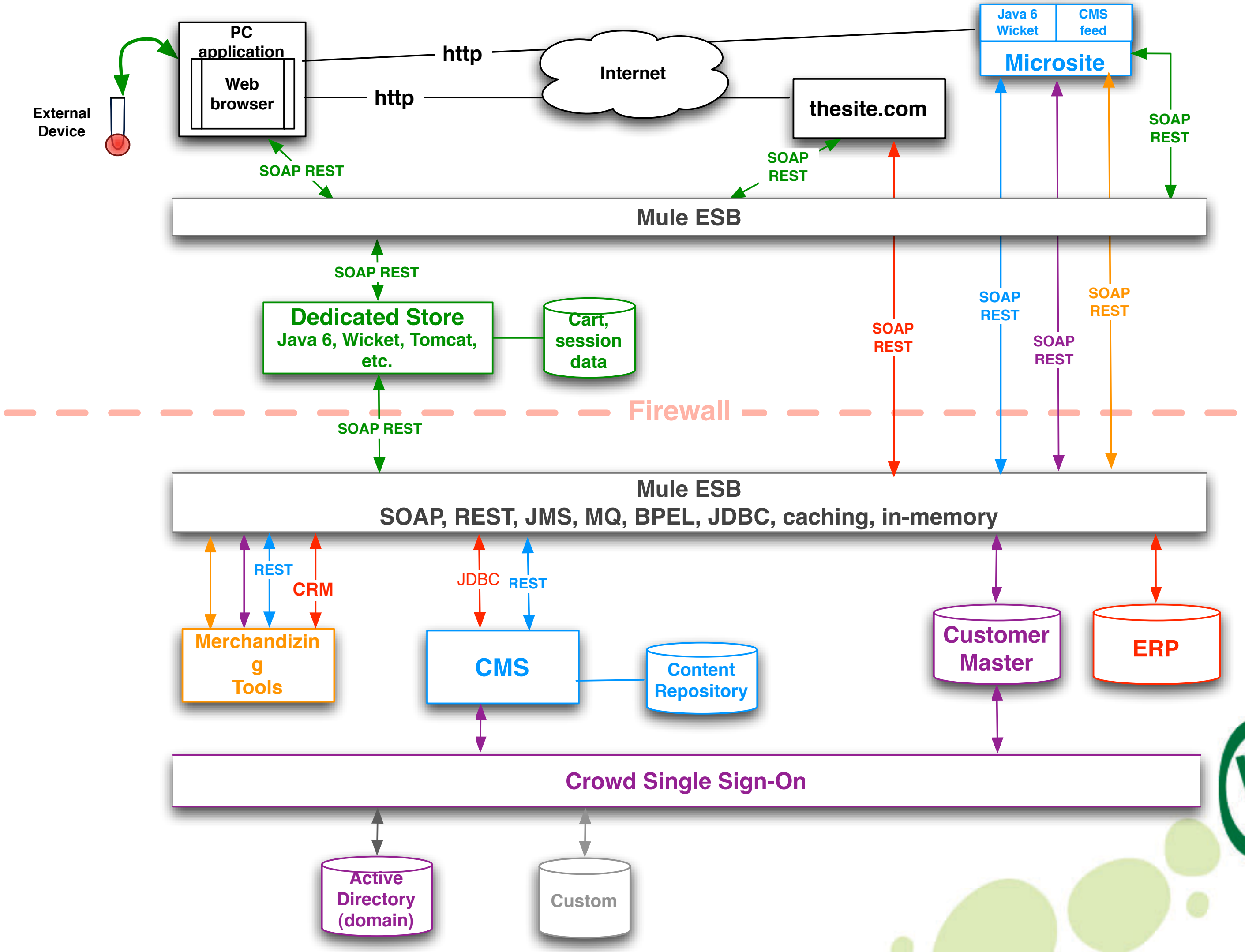
Integration Through Services

- SOA = Services-Oriented Architecture
- Collection of services that communicate with one another
  - No dependencies on other services
  - Self-contained
- Messaging: mechanism for communication between two or more services
- Real-time, asynchronous, synchronous
  - May occur over different transports
    - HTTP, FTP, JMS, RMI, CORBA, etc.

SOA Limitations

- Not all systems can be mapped as services
- Workflow issues
- Development team coordination
- Programmer skill levels
  - Do your programmers grok SOA?
- System coupling
  - System dependencies
  - Organizational dependencies

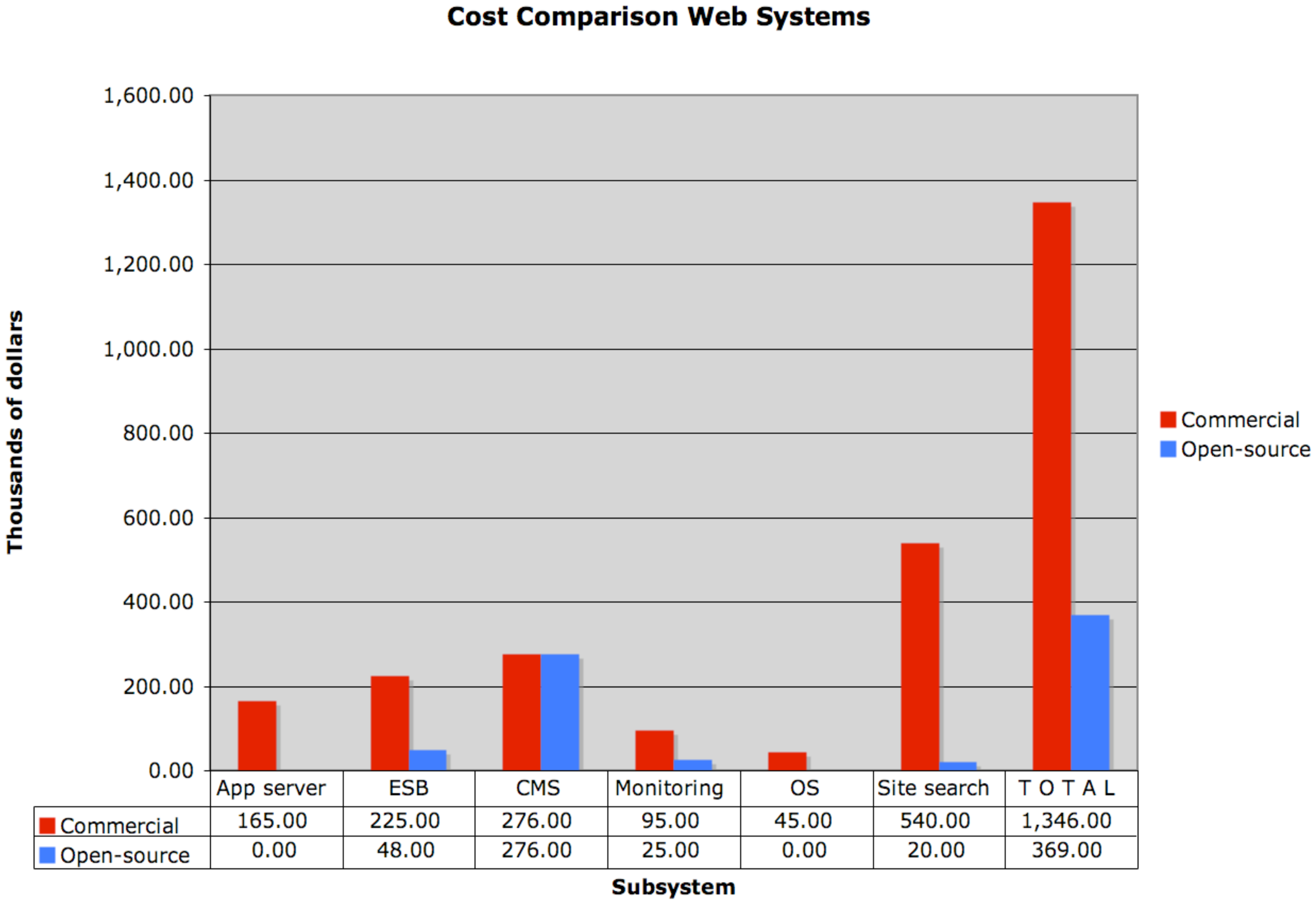
Environment - First Iteration



Technologies Deployed

- Best of Breed
- Mule ESB - the backbone
- Crowd Single Sign-on
- GWT for front end AJAXy stuff
- Wicket for Web applications
- Day Communiqué / CRX for CMS
- All open-source development tools
- Java 5 and Java 6

How Well Did This Work?



Cost Comparison Web Systems

	App server	ESB	CMS	Monitoring	OS	Site search	T O T A L
🟥 Commercial	165.00	225.00	276.00	95.00	45.00	540.00	1,346.00
🟦 Open-source	0.00	48.00	276.00	25.00	0.00	20.00	369.00

What’s Next?

- Integration of third-party systems
  - 2007 - two
  - 2008 - ten or more
- International sites
- Real-time device data processing
- Multiple data sources
  - Databases
  - Financial systems
  - CRM
- Support for millions of devices “in the wild”

Shift Toward Consuming Resources

- Conscious decision to blur the distinction between “services” and “data sources”
- Everything is a resource
  - SOAP, REST, JMS, files
  - Web apps back-end
  - Computational data
- Resources are available through a well-defined protocol
- Resources are always available through a common transport to simplify development and deployment

What is Resource-Oriented Computing?

- All components of a system are viewed as resources to be consumed synchronously or asynchronously
- There is no distinction between “data”, “objects” or “services”
- There is no dependency on a programming language or framework
  - Mix and match is the reason why you want to move toward ROC
- Resources are located through URIs
- Software identifies resources through logical rather than physical mappings

What is Resource-Oriented Computing?

- Programs map logical and physical locations through identifiers in traditional computing models
  - String resource = “I am some useful, non-trivial text.”;
- ROC defines resources through verbs and logical identifiers
  - Yes, it sounds like REST
- An identifier ALWAYS returns the CURRENT representation of a resource
- Each logical identifier is resolved for every request
  - Resource implementations can change dynamically, resource consumers need not care about where or how a resource is implemented

Java vs. REST vs. ROC

	Java	REST	ROC
Identifier	private int nX;	URI	URI
Fetch	out.printf("nX = %d\n", nX);	Method GET URI	Protocol fetch + URI
Resolve	Compiler, reflection	DNS + app server	ROC kernel or backbone
Compute	Java Virtual Machine	App server	Endpoint and service object
Low-level operation	JVM, method, initializer	HTTP method + URI	Verb + URI pair

Defining Resources

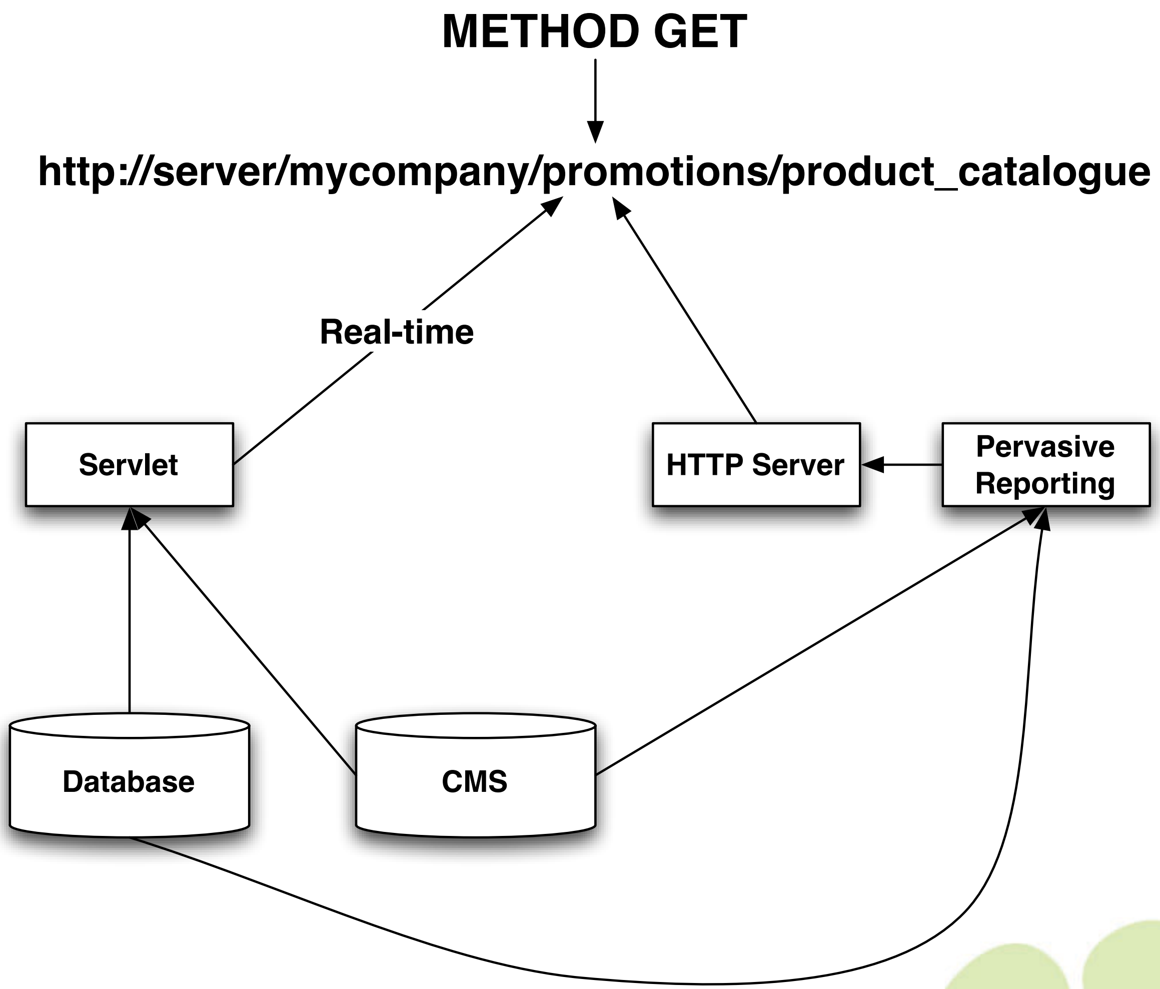
- Resources don’t exist in the context of an application until they are requested
- Resources lack typing
  - Typing is relevant only to the consumer
- Endpoint URIs may convert types for individual data elements or complex data structures
- URIs may encode the desired operation to perform on the data
  - protocol://servername/subsystem/operation/resource

Resource Abstractions

http://server/mycompany/promotions/product_catalogue

- The promotions resources may be generated...
    - cron periodically
    - On-demand
    - Aggregated
- The promotions system of record is independent of the ROC platform or the consumer
- The “verb” here is “promotions”, when combined with a GET
- There may be two or more aggregators that produce the resource

Resource Abstractions



METHOD GET

http://server/mycompany/promotions/product_catalogue

Real-time

Servlet

HTTP Server

Pervasive Reporting

Database

CMS

ROC Platforms

- Full ROC platform by 1060 Research
    - Custom distributed kernel
- GridGain, GigaSpaces
    - Distributed Computing
- Homebrew ROC
    - Are you in the business of building one from scratch?
- Off-the-shelf integration
    - Best-of-breed strategy: find the best components and integrate them

ROC Platforms

VENDOR LOCK-IN!!!!

- Homebrew ROC
  - **Are you in the business of building one from scratch?**
- Off-the-shelf integration
  - **Best-of-breed strategy: find the best components and integrate them**

ROC Architecture

- The systems are built around a backbone that provides resources via URI
- The backbone acts as an resource container or as a conduit between resources or resources and consumers
- URI mapping is done by the backbone
- Resource containers can exist in the same memory space as the backbone or in a separate system
- Resource providers may be written in any programming language
- Resource providers are stateless

ROC Architecture

- Modularity is attained through logical separation of resources
  - Resource providers as .jar, .war, or other entity
  - Localized backbones
  - Localized resource providers
- Logical separation may obey organizational policy, technology policy, or both
- Implementation can be done with off-the-shelf components in any combination that makes sense, as long as the backbone is protocol-, language-, and vendor-independent

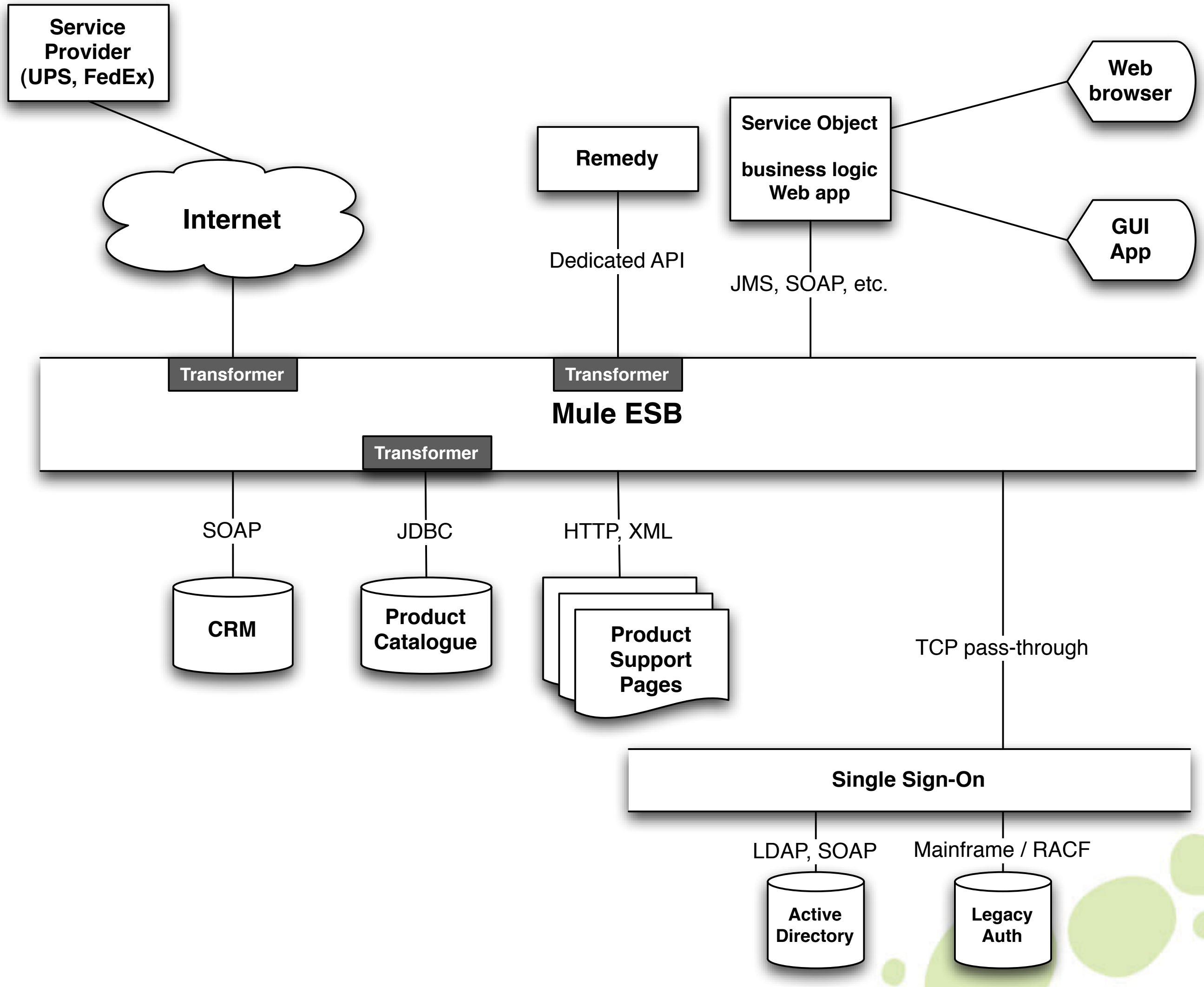
ROC Architecture

- Backbone: Mule ESB
  - Provides full independence from the kind of crap that vendors like to create lock-in for
  - Open-source
  - Workflow, transactions, transformations, logging, routing
- Resource container: Mule ESB
  - UMOs (service objects) implement business logic independently of protocol or data formats by design
  - Transactional, app server and workflow logic built-in
  - UMOs are just POJOs
- Synchronization
  - In-memory endpoints

ROC Architecture

- Original architecture had lots of best-of-breed software
  - Tomcat
  - Dedicated application/service providers
  - Web servers
- ROC architecture only has two basic building blocks
  - Mule acting as a resource service provider (i.e. Mule is the application container)
  - UMOs as computationally active entities
- Existing and off-the-shelf systems plug into the architecture through SOAP, REST, JMS, etc.
- Mule allows us to define our own protocols, if necessary!

ROC Architecture



Service Provider (UPS, FedEx)

Internet

Remedy

Dedicated API

Service Object

business logic Web app

JMS, SOAP, etc.

Web browser

GUI App

Transformer

Transformer

Mule ESB

Transformer

SOAP

JDBC

HTTP, XML

CRM

Product Catalogue

Product Support Pages

TCP pass-through

Single Sign-On

LDAP, SOAP

Mainframe / RACF

Active Directory

Legacy Auth

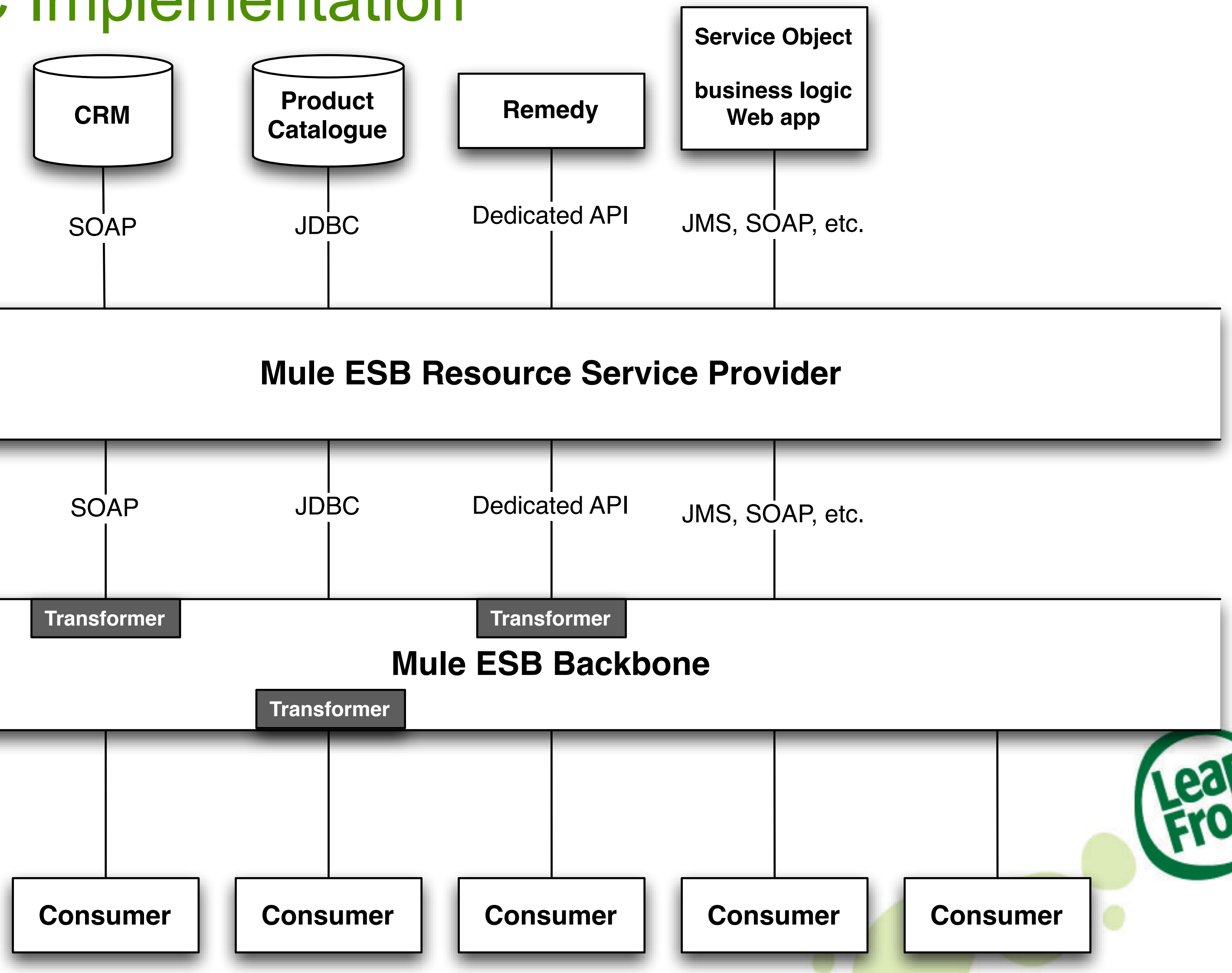
ROC Implementation

- Dedicated protocols
  - vm://mycompany/subsystem/resource_name
  - http://mycompany/subsystem/resource_name
- Easy to extend to handle ROC:

verb:protocol://mycompany:port/organization/subsystem/resource_name

- Easy to implement!

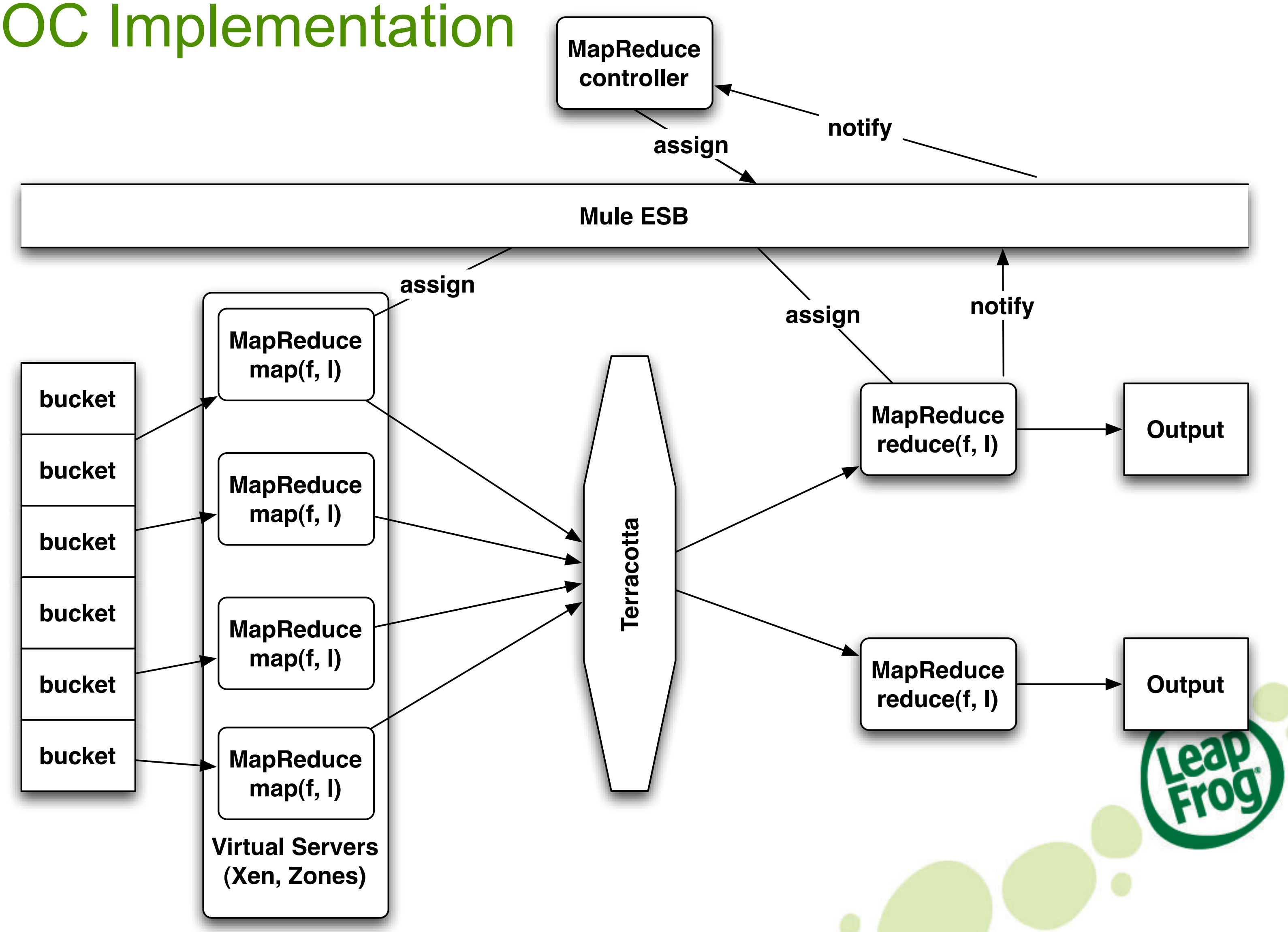
ROC Implementation



ROC Implementation

- Resource providers
  - SOAP API to CRM
  - JMS API to transactional pieces
  - Download app repository
  - OpenLaszlo dynamic rich Internet application provider
- Interfaces to existing systems
  - Epsilon direct mail interfaces
  - FTP, sftp, other data transfer
- Computational resources for ad hoc new functionality
  - MapReducers (2008, 2009)

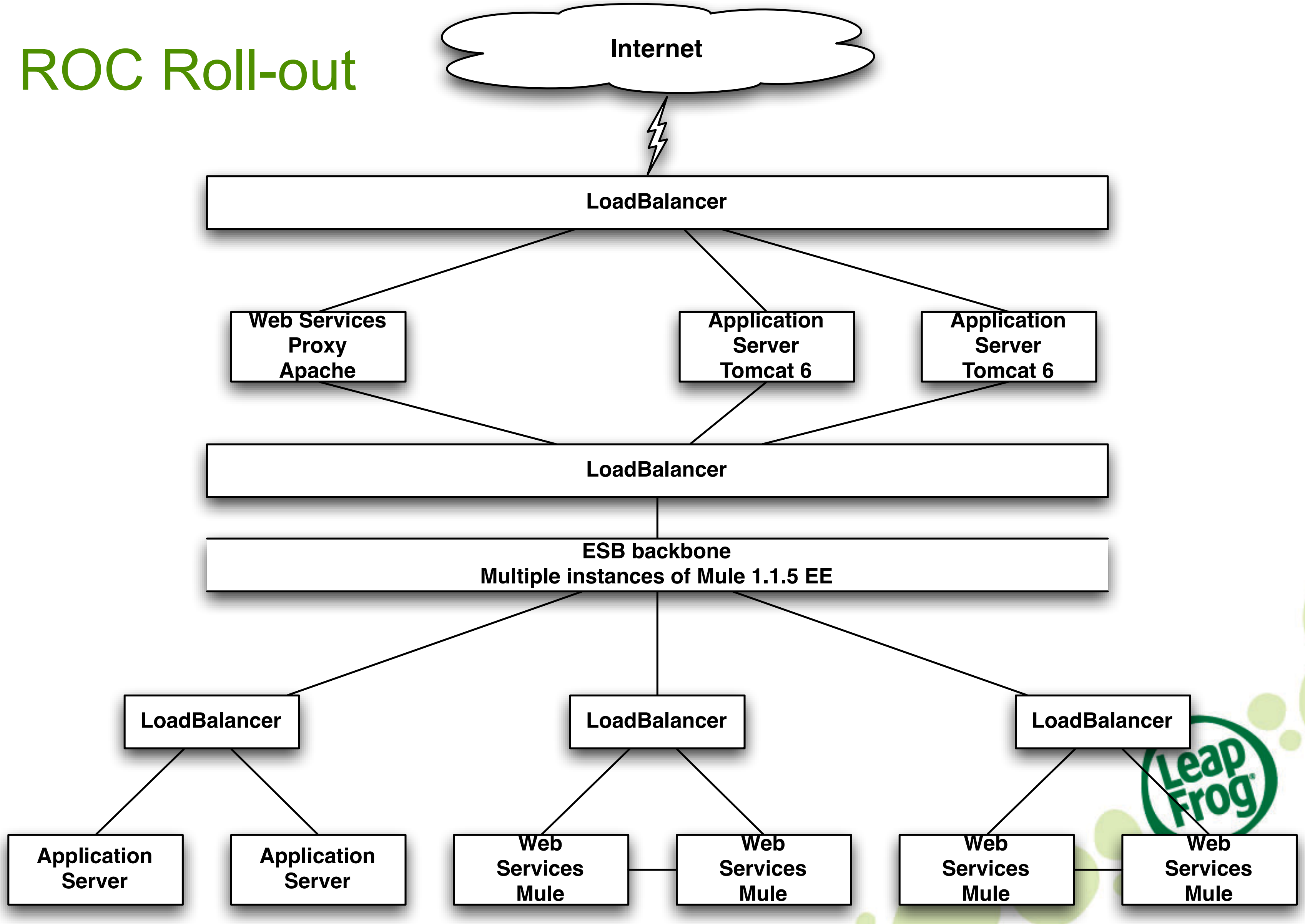
ROC Implementation



ROC Roll-out

- Quick, turnkey roll-out
- The fewer systems to maintain, the better
- Use Java or JVM-hosted languages wherever possible
- Integrate with third-party or non-Java systems over standard or custom protocols with as quick a turnaround as possible
- EASY TO SCALE QUICKLY!!!!

ROC Roll-out


Internet
LoadBalancer
Web Services Proxy Apache
Application Server Tomcat 6
Application Server Tomcat 6
LoadBalancer
ESB backbone
Multiple instances of Mule 1.1.5 EE
LoadBalancer
LoadBalancer
LoadBalancer
Application Server
Application Server
Web Services Mule
Web Services Mule
Web Services Mule
Web Services Mule

Conclusions

- Complex systems are easier to code and maintain if implemented as small blocks
- Small blocks can be mapped as resources that can be consumed in a stateless fashion
- Applications can be built as an aggregation of resources
- ROC techniques improve time-to-market
- ROC techniques combined with open-source offerings can reduce deployment costs by 70%, and ongoing maintenance by 30-40%
- Complex systems can be integrated as a combination of best-of-breed software whether commercial, open-source, or homebrew
- ROC is the logical evolution of applied SOA

Q&A

Thanks for coming!

This presentation is at:
http://eugeneciurana.com/MuleCon2008/ROC.pdf

Eugene Ciurana
Director, Systems Infrastructure
Leap Frog Enterprises, Inc.

eugenex@leapfrog.com
pr3d4t0r @ irc://irc.freenode.net ##java, #esb, #awk, #security