REST, Reuse, and Serendipity

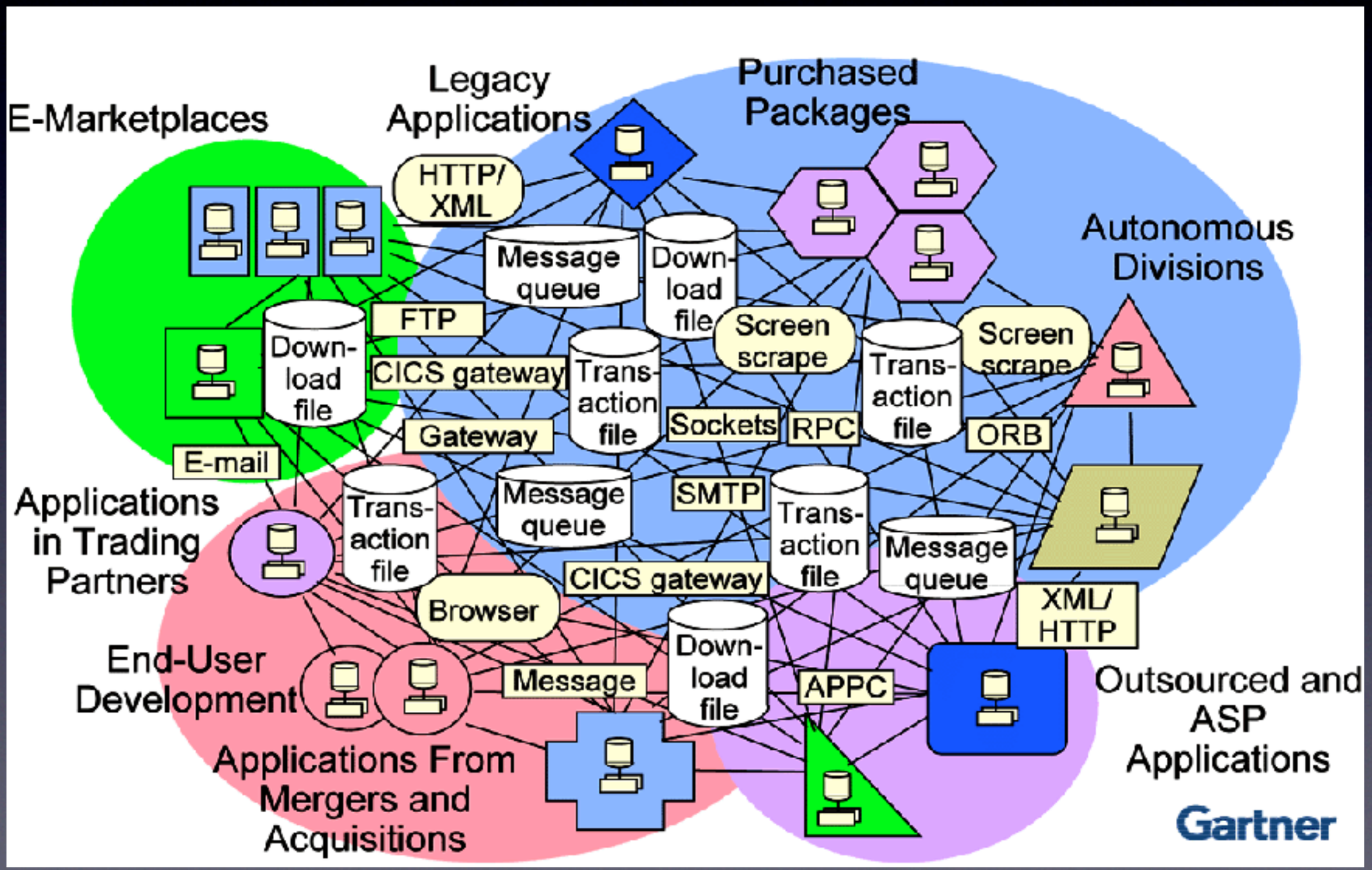
Steve Vinoski
Member of Technical Staff
Verivue
Westford, MA USA
QCon London 2008

serendipity (noun): the occurrence and development of events by chance in a happy or beneficial way.

Motivation

- Many folks in the web world already understand the capabilities and power of REST
  - if this describes you, this talk may bore you
- But to many involved in enterprise integration and middleware, REST is entirely new and misunderstood
  - this talk is for you

Enterprise Integration



Some Solutions

- Descendants of Remote Procedure Call (RPC)
  - CORBA, EJB, .NET, SCA
  - SOAP, WSDL, WS-*
- Enterprise messaging
  - Messaging queuing
  - Enterprise Application Integration (EAI)

Problems with MQ

- Generally proprietary, except for:
  - JMS: open messaging interfaces
  - AMQP: open messaging protocol (and open source Apache implementation)
- Can be expensive

Problems with EAI

- EAI systems are typically:
  - proprietary and expensive
  - centralized hubs
  - costly to customize and maintain
- Some ESBs have fallen into this category
  - old EAI products re-labeled

Problems with RPC Approaches

- Focus on language first
  - tries to fit the distributed system to the language, not vice-versa
- Tries to make distributed calls appear local
  - ignores partial failure and latency issues
- Exposes language-specific objects directly as language-independent services

More RPC Problems

- Code generation
  - traditionally, stub code generated from a definition language, e.g. IDL or WSDL
    - today WSDL is often reverse-generated from annotated language definitions
- Both approaches can create deceptively significant consumer-service coupling

Type System Illusions

- RPC-oriented systems offer the illusion of type safety
  - define interface types
    - define data types to pass via methods
- But there is no type safety across the wire
- This type specialization is costly for scalability and reuse

Interfaces are Protocols

- In RPC-oriented systems, a new service interface is a new application protocol
  - consumers hard-code knowledge of method names and semantics
  - consumers must inherently know which method to call, possibly in what order
  - no semantic constraints on methods

Data Specialization

- RPC-oriented systems encourage specialized data definitions
  - same as defining regular classes/methods
- Using XML is better than using IDL types or programming language types
  - but benefits disappear if you generate code from it

Integration Problem Summary

- Proprietary approaches too expensive
- Standard approaches focus on implementation languages, not distributed systems issues
- New interface == new application protocol
- Ad hoc data formats coupled to interfaces
- All these problems inhibit reuse

A Detour: UNIX Reuse

- Consider the UNIX shell pipe
  - chain output of one tool to input of another
  - old tools and new can interact, even though independently developed
  - easily combine existing tools into new ones

UNIX Pipes

- The pipe is based on two key features
  - the uniform interface of the “file-like object”
  - the standard file descriptor framework for applications: stdin, stdout, stderr
- Standard ways to get data to/from applications
- The pipe results from modularity and simplicity (and serendipity, perhaps?)

REST's Uniform Interface Constraint

- Generalized resource interface
  - in HTTP, methods are the protocol verbs

Method	Purpose	Idempotent?
GET	Retrieve resource state representation	Yes (no side effects)
PUT	Provide resource state representation	Yes
POST	Create or extend a resource	No
DELETE	Delete a resource	Yes

Uniform Interface Benefits

- Enables visibility into interactions
  - including caching, monitoring, mediation applicable across all resources
- Provides strong implementation hiding, independent evolvability
- Simplified overall architecture

Generic Invocation

- The uniform interface makes reusable generic invocation libraries possible
  - python urllib, urllib2, httplib, httplib2
  - curl command-line tool
  - many others, in many languages
- Server-side dispatching simplified as well
- No need for generated code

Four Sub-Constraints

- Resource identification via URIs
- Resource manipulation through the exchange of resource state representations
- Self-describing messages with potentially multiple representation formats
- Hypermedia as the engine of application state (HATEOAS, or hypermedia constraint)

Representations

- Method payloads are representations of resource state
- Methods often support multiple representation formats
- Representation format is not method-specific as with RPC-oriented approaches

Media Types

- Representation formats identified using media (MIME) types
- These types are standardized/registered through the IANA (http://www.iana.org/assignments/media-types/)
- Allows reusable libraries and tools to handle various MIME types

Hypermedia Constraint

- Resources keep resource state, clients keep application state
- Resources provide URIs in their state to guide clients through the application state
- Clients need "know" only a single URI to enter an application, can get other needed URIs from resource representations

Separation of Concerns

- RPC-oriented systems conflate methods and data
  - Many ad hoc methods and data types, but just a single data format on the wire
- REST separates methods and data formats
  - Fixed set of methods, many *standardized* data formats, multiple formats possible per method

Enhancing Reuse Possibilities

- Separating concerns
  - libraries/tools for dealing with methods
  - separate libs/tools for dealing with data
- Uniformity
  - libraries for caching
  - libraries for interception and mediation

For Example

- Consider a bug-tracking system
  - HTML representations for interactive viewing, additions, modifications
  - Excel or CSV representations for statistical tracking by importing into other tools
  - XML (e.g. AtomPub) or JSON to allow use by other tools, for extensions and integration
  - Atom feeds for watching bug activity
- Existing clients that understand these formats can easily adapt to use them

Independence

- Each of the resources and formats on the previous page could be added independently without disturbing existing resource and formats
- You might add them “just because,” without an immediate need
  - plant the seeds to see what grows

Summary

- RPC-oriented systems try to extend programming language paradigms over the wire
  - encourages variation, which can't scale
- REST is purpose-built for distributed systems
  - properly separates concerns and allows constrained variability only where required
  - encourages combinations of orthogonal solutions into larger applications

Engineer for serendipity.

Roy T. Fielding

For More Information

- Fielding's thesis
  - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Read various blogs: Mark Baker, Bill de hÓra, Joe Gregorio, Paul Downey, Benjamin Carlyle, Stu Charlton, Mark Nottingham
- Sign up to the *rest-discuss* Yahoo mailing list
- My "Toward Integration" columns in IEEE Internet Computing (all columns are available from http://steve.vinoski.net/), for example:
  - *Serendipitous Reuse* (Jan/Feb 2008)
  - *Demystifying RESTful Data Coupling* (Mar/Apr 2008)