SOAP vs. REST: Complements or Competitors?

David Chappell

Chappell & Associates

DavidChappell
www.DavidChappell.com

Web Services Today

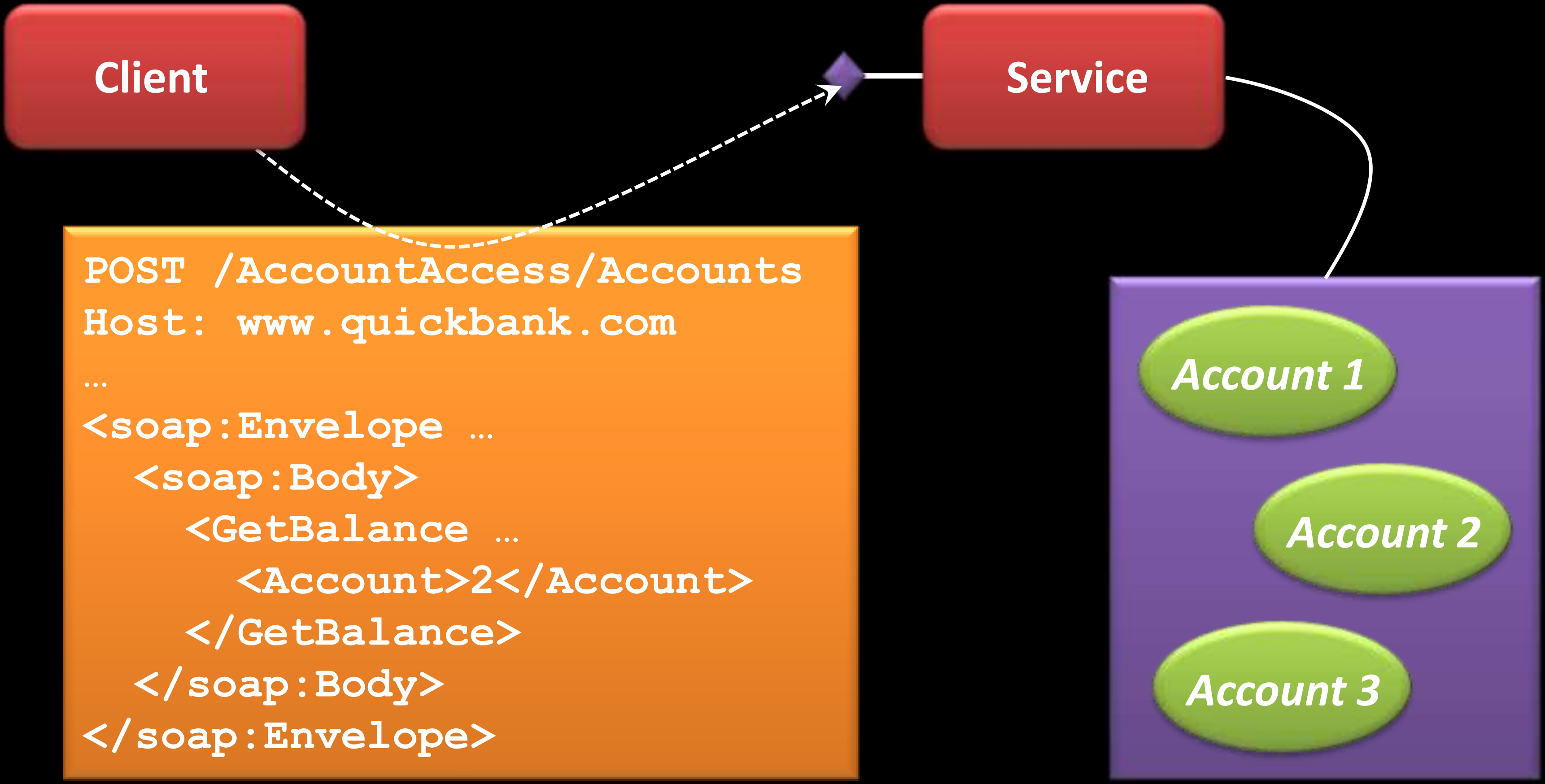
- Two approaches to Web services exist today:
  - SOAP and the WS-* specifications
  - Representational State Transfer (REST)
- There is some competition between proponents of each approach
- Yet both have value
  - The challenge is to determine when to use each one

Describing SOAP

David Chappell
& Associates

Access via SOAP

Illustrating the approach



```
POST /AccountAccess/Accounts
Host: www.quickbank.com
…
<soap:Envelope …
  <soap:Body>
    <GetBalance …
      <Account>2</Account>
    </GetBalance>
  </soap:Body>
</soap:Envelope>
```

Access via SOAP

A Windows Communication Foundation (WCF) interface

```
[ServiceContract]
interface IAccount
{
    [OperationContract]
    int GetBalance(int account);

    [OperationContract]
    int UpdateBalance(int account,
                      int amount);
}
```

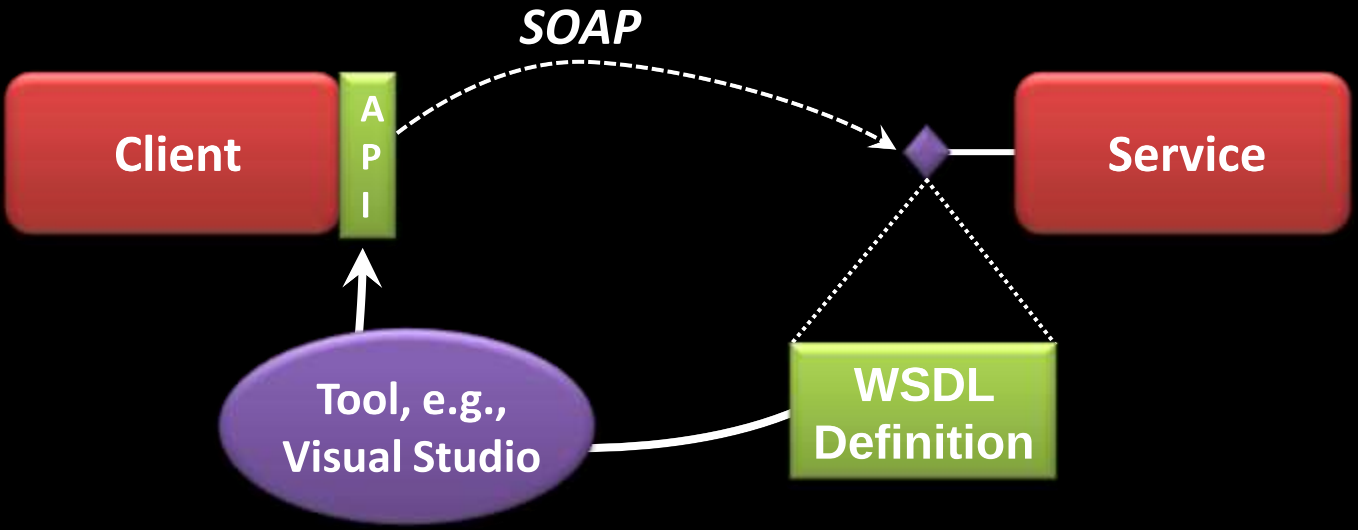
Indicates that this interface should be exposed as a service

Indicates that this method should be exposed as a remotely callable operation

Access via SOAP

Creating clients

- SOAP services are typically defined using the Web Services Description Language (WSDL)
  - This lets tools create client APIs
  - Client developers see methods with parameters



Access via SOAP
Representing data

- SOAP typically represents information using XML

- Pros:
  - There’s one common, expressive format
- Cons:
  - XML isn’t especially efficient
  - XML isn’t a good fit for some languages

Describing WS-*
Messaging and security

- Messaging
  - WS-Addressing: Allows using SOAP over protocols other than HTTP
- Security
  - WS-Security: Defines how to convey various security tokens and more
  - WS-Trust: Defines how to get security tokens
  - WS-SecureConversation: Allows establishing a security context

Describing WS-*
Reliability and transactions

- Reliability
  – WS-ReliableMessaging: Allows reliable end-to-end communication through SOAP intermediaries

- Transactions
  – WS-AtomicTransaction, WS-Coordination: Define how to do two-phase commit for ACID transactions

Describing WS-*
Policy and metadata

- Policy
  - WS-Policy: Allows defining policies in various areas, e.g., security

- Acquiring interface definitions
  - WS-MetadataExchange: Allows accessing a service’s WSDL definition and more

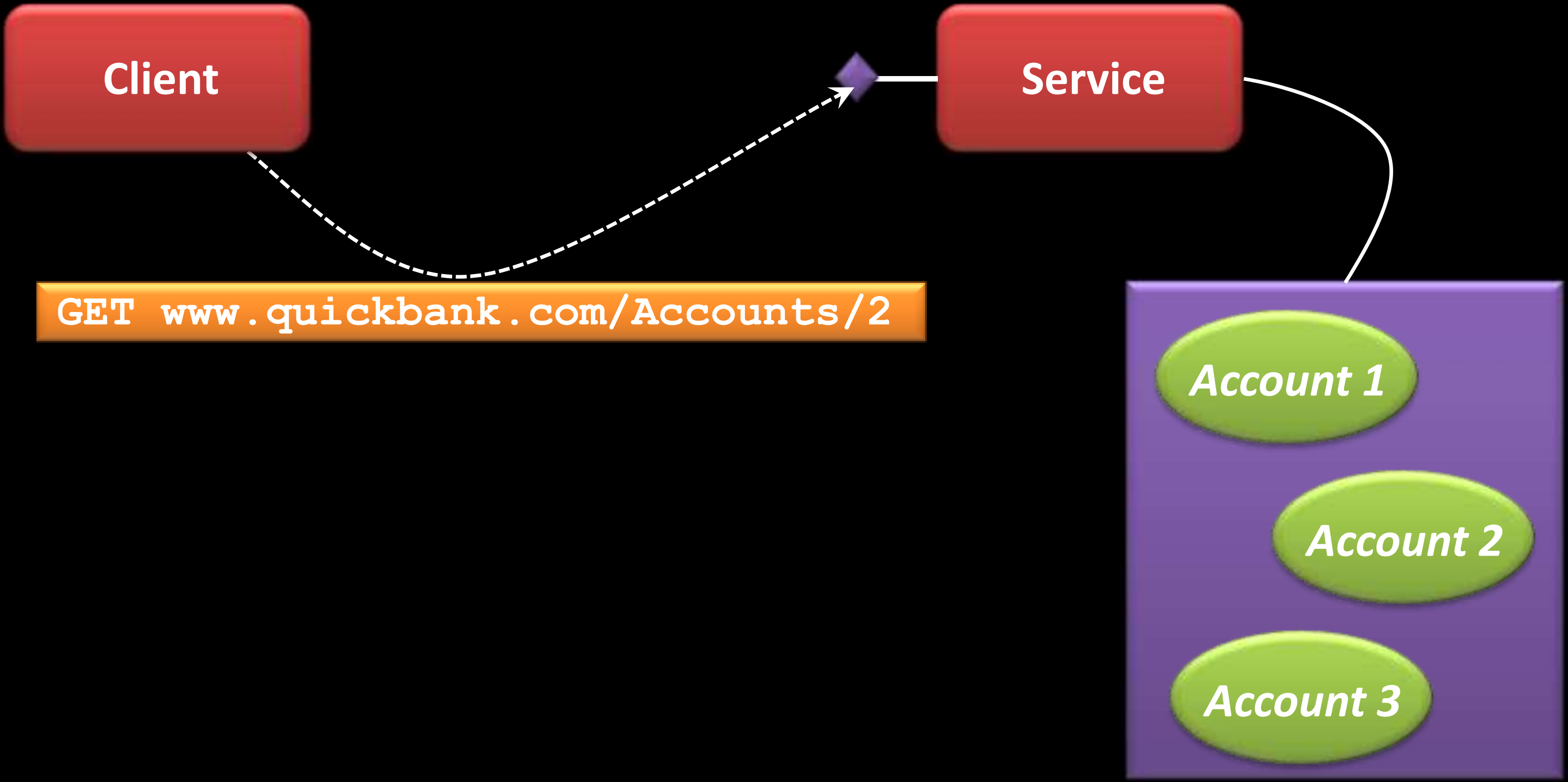
WS-* in the Real World
Pragmatic issues

- SOAP/WS-* aren't universally supported today
  - For example, WCF isn't (yet) the dominant technology for Web services on Windows
- Cross-vendor interoperability for SOAP and the WS-* technologies isn't perfect
  - Contract-first design can help
    - But WSDL is hard to work with

Describing REST

Access via REST

Illustrating the approach



Defining REST
An architectural style

- Two core principles
  - Everything is accessed through a uniform interface
    - GET, PUT, POST, DELETE, …
  - All resources are identified with a URI
- Some subsidiary principles
  - Be cacheable whenever possible
  - Be stateless whenever possible
  - More . . .

Truth In Naming

An aside

- Calling SOAP-based services “Web services” makes no sense
  - – SOAP has little to do with Web technologies

- REST-based services truly deserve the name “Web services”
  - – They’re entirely based on HTTP and URIs

Access via REST

A WCF interface

```
[ServiceContract]
interface IAccount
{
    [OperationContract]
    [WebGet]
    int GetBalance(string account);

    [OperationContract]
    [WebInvoke]
    int UpdateBalance(string account,
                      int amount);
}
```

Sends request using HTTP GET

Sends request using HTTP POST (by default)

The Semantics of HTTP Verbs

A closer look

- The semantics of GET, PUT, and DELETE are well-defined
- The semantics of POST are less clear
  - From the HTTP 1.1 spec:

POST is designed to allow a uniform method to cover the following functions:

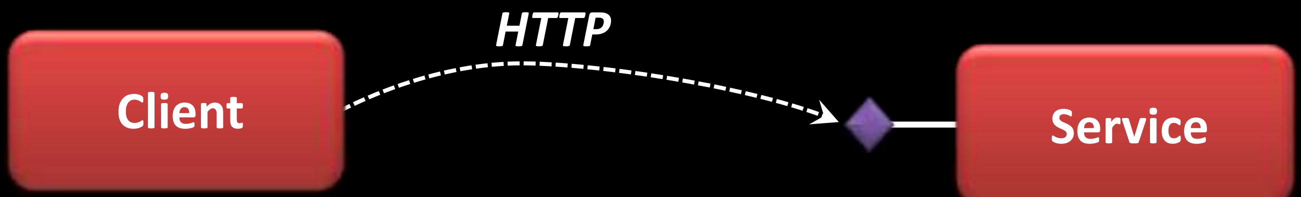
- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server ...

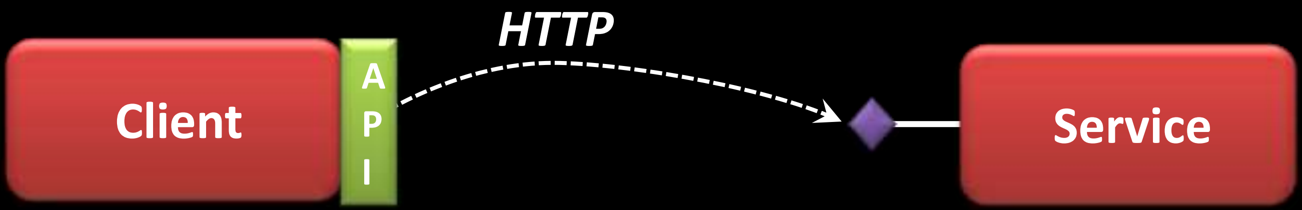
Access via REST

Creating clients

- There is no standard definition language for defining RESTful interfaces
- Option 1: Clients write raw HTTP calls



- Option 2: A RESTful service provides a client library
  – Clients see methods with parameters



Access via REST
Representing data

- REST defines no standard data representation
  - A RESTful service can use XML, JavaScript Object Notation (JSON), and other formats
- Pros:
  - Data formats can better match clients
    - Such as using JSON with JavaScript clients
  - Different formats can be chosen to match different performance requirements
- Cons:
  - Options increase complexity

REST in the Real World

Pragmatic issues

- No formal way to describe a service interface means more dependence on written documentation

- Client issues
  - Most developers don’t like writing raw HTTP calls
  - But providing a client library requires:
    - Choosing what languages and programming environments to support
    - Dealing with versioning

Comparing SOAP and REST: Making the Right Choice

DavidChappell
& Associates

Areas For Comparison

- Exposing operations vs. exposing resources
  - SOAP/WS-* and REST emphasize different things
- Capabilities
  - SOAP/WS-* and REST provide different functions

Resources vs. Operations

What is exposed?

- REST
  - Focused on accessing named resources
    - Each of which typically represents some data
  - Every application exposes its resources through the same interface
- SOAP
  - Focused on accessing named operations
    - Each of which typically implements some logic
  - Different applications expose different interfaces

RESTful Data Access

Example: Amazon’s Simple Storage Service (S3)

- S3 allows storing Objects in Buckets
  - Similar to storing files in directories

- Example operations:
  - GET Object: Returns the contents of this object
  - GET Bucket: Returns a list of objects in this bucket
  - PUT Object: Creates a new object
  - PUT Bucket: Creates a new bucket
  - DELETE Object: Deletes an object
  - DELETE Bucket: Deletes a bucket

RESTful Data Access
The benefits of caching

- For many (most?) services, the majority of client requests are reads
  – In a RESTful service, all reads rely on HTTP GET

- The results of a GET are commonly cached
  – This can allow better performance and more scalability for RESTful services exposed over the Internet

SOAP-Based Operation Access

Example: The banking interface shown earlier

- A service for banking functions might include operations such as
  - `GetBalance(Account)`
  - `UpdateBalance(Account, Amount)`
- These work well with either REST or SOAP
- Suppose the interface also includes
  - `Transfer(FromAccount, ToAccount, Amount)`
- This maps naturally to a SOAP operation
  - It doesn’t map as well to REST’s resource-oriented model

SOAP/WS-* and REST

A capability summary

	SOAP/WS-*	REST
Protocol for invoking operations	SOAP	HTTP
Transport protocol	HTTP, TCP, others	HTTP
Language for describing interfaces	WSDL	No standard
Data formats	XML	XML, JSON, others
Conveying security tokens	WS-Security	HTTP, SSL
Acquiring security tokens	WS-Trust	No standard
Establishing a security context	WS-SecureConversation	SSL
Providing end-to-end reliability	WS-ReliableMessaging	No standard
Supporting distributed ACID transactions	WS-AtomicTransaction, WS-Coordination	No standard
Defining policy	WS-Policy, et al.	No standard
Acquiring interface definitions	WS-MetadataExchange	No standard

Broad Standardization vs. YAGNI

Two views of the world

- Broad standardization
  - – Provides a wide range of capabilities
  - – Increases the odds of correct implementation, since vendors implement the capabilities
  - – Allows interoperability, since everyone provides the capabilities in the same way

- YAGNI
  - – You Ain’t Gonna Need It, so keep things simple

Security
REST

- RESTful services commonly use SSL
- Standards for carrying security tokens:
  - HTTP for username/password
  - SSL for X.509 certificates
- This is sufficient for many scenarios
  - Such as point-to-point Internet communications

Security
SOAP/WS-*

- SOAP-based services can use SSL

- SOAP-based services can also use WS-Security, which provides:
  - Support for identity through SOAP intermediaries
    - Not just point-to-point
  - Broader standards for carrying security tokens
  - A standard way to provide data integrity and data privacy

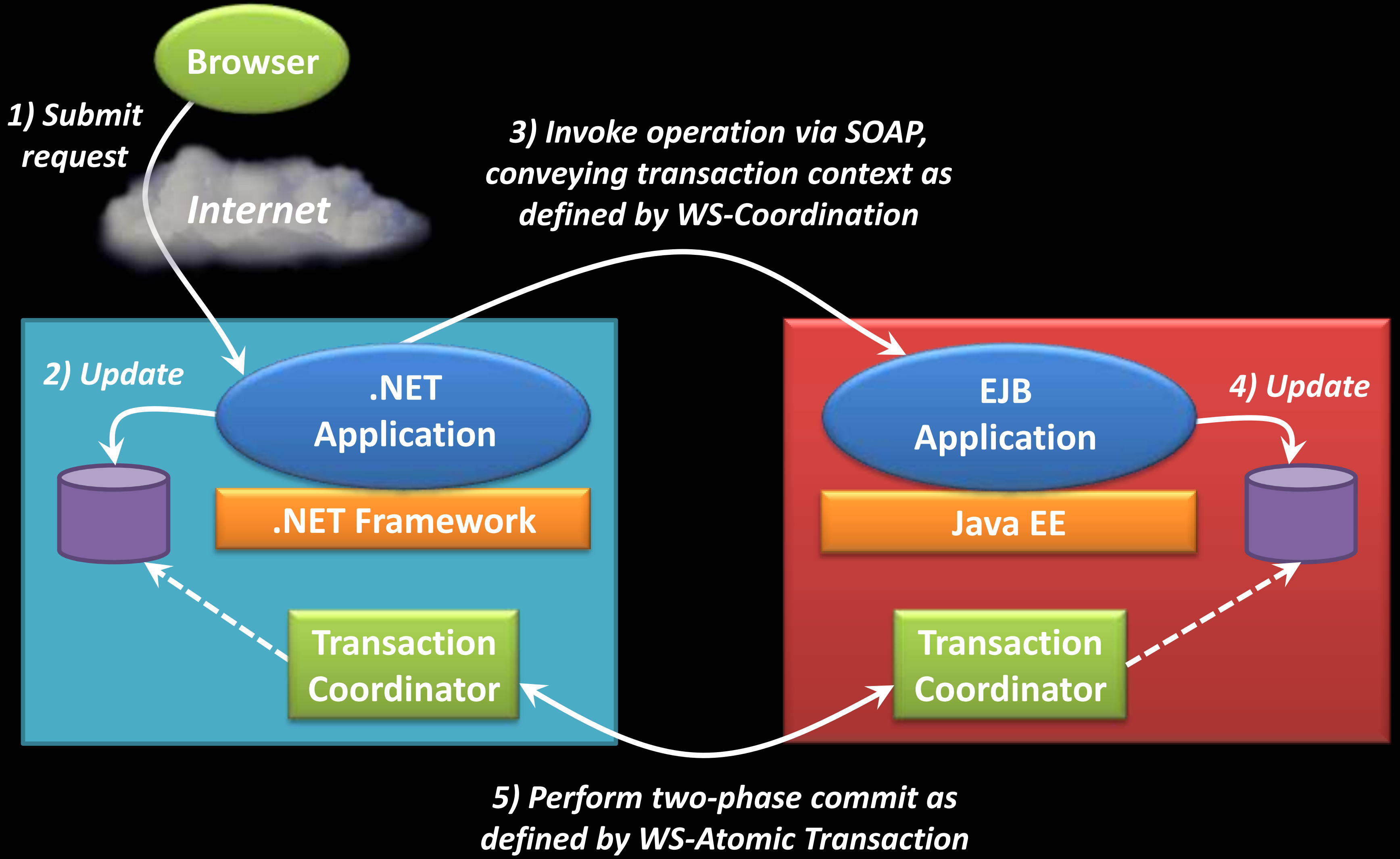
Transactions
Using WS-AtomicTransaction

- ACID transactions that span multiple applications are important in enterprise computing
  – ACID transactions don’t usually make sense across the Internet

- WS-AtomicTransaction addresses this problem
  – It relies on WS-Coordination

Transactions

A simplified WS-AtomicTransaction example


Browser
1) Submit request
Internet
3) Invoke operation via SOAP, conveying transaction context as defined by WS-Coordination
2) Update
.NET Application
.NET Framework
Transaction Coordinator
EJB Application
4) Update
Java EE
Transaction Coordinator
5) Perform two-phase commit as defined by WS-Atomic Transaction

Reliability

- REST
  - Assumes the application deals with communication failures via application retries
- SOAP with WS-ReliableMessaging
  - Builds acknowledgement/retry logic into the communications stack
  - Can provide end-to-end reliability through one or more SOAP intermediaries

Reliability

The challenge of idempotency

- An operation is *idempotent* if invoking it once has the same effect as invoking it more than once
  - Example: A GET that reads an account balance
- POST might not be idempotent
  - Example: A POST that transfers money between bank accounts
- There’s no guaranteed reliability in HTTP
  - What does a RESTful client do when a POST fails?

A Case Study: ArcGIS
The evolution of exposed services

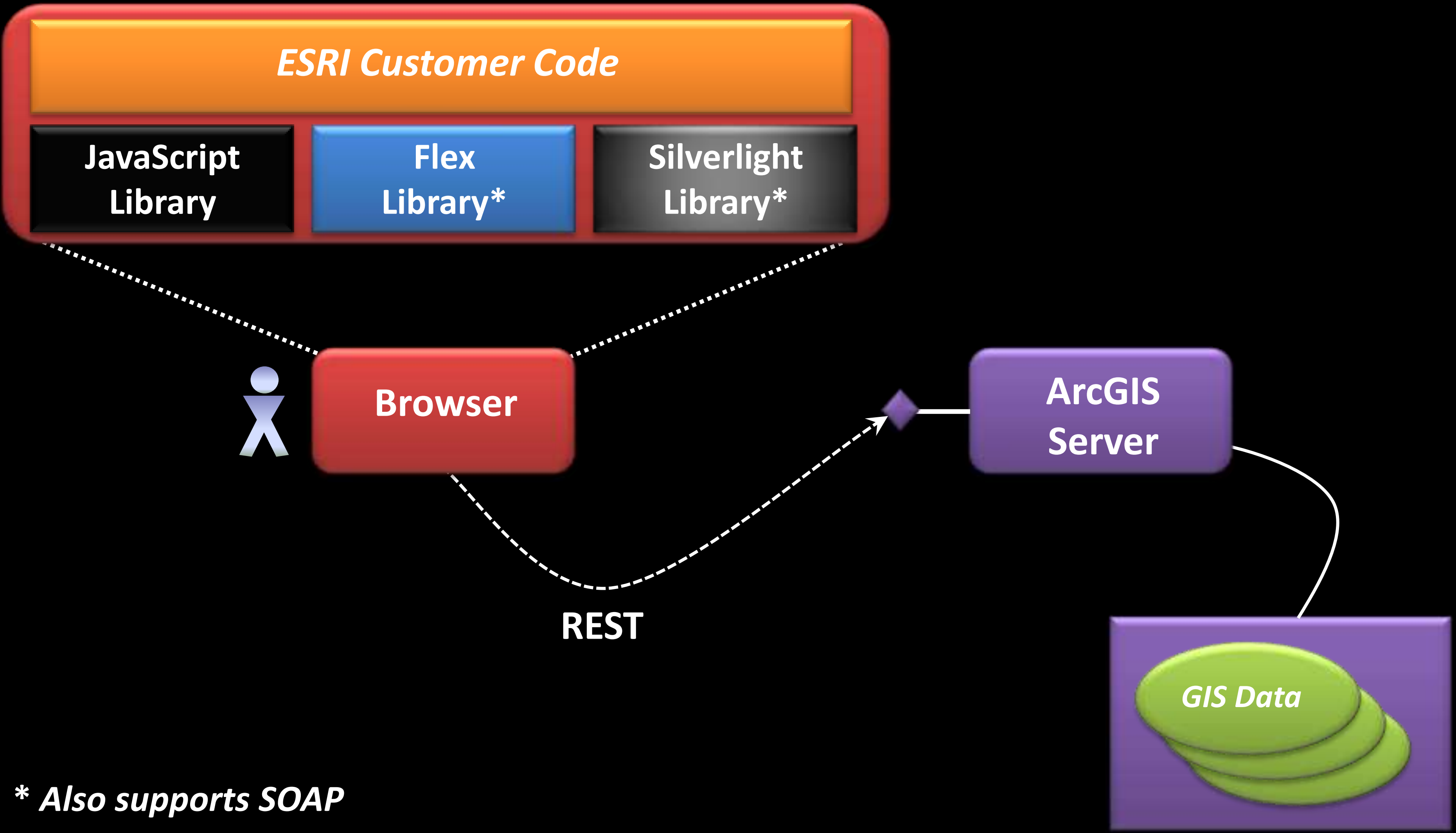
- Circa 2003: SOAP only
  - No WS-*
- Circa 2006: SOAP and REST
  - The SOAP interfaces provided greater functionality
- Moving forward: An emphasis on REST
  - With the SOAP and REST interfaces offering equal functionality
  - Both are documented and can be accessed directly

ArcGIS
Why change?

- REST is simpler
  - ArcGIS doesn’t need everything SOAP/WS-* provides
- REST has better performance and scalability
  - SOAP-based reads can’t be cached, for instance
- REST allows better support for browser clients
  - Because it allows diverse formats, e.g., JSON

ArcGIS

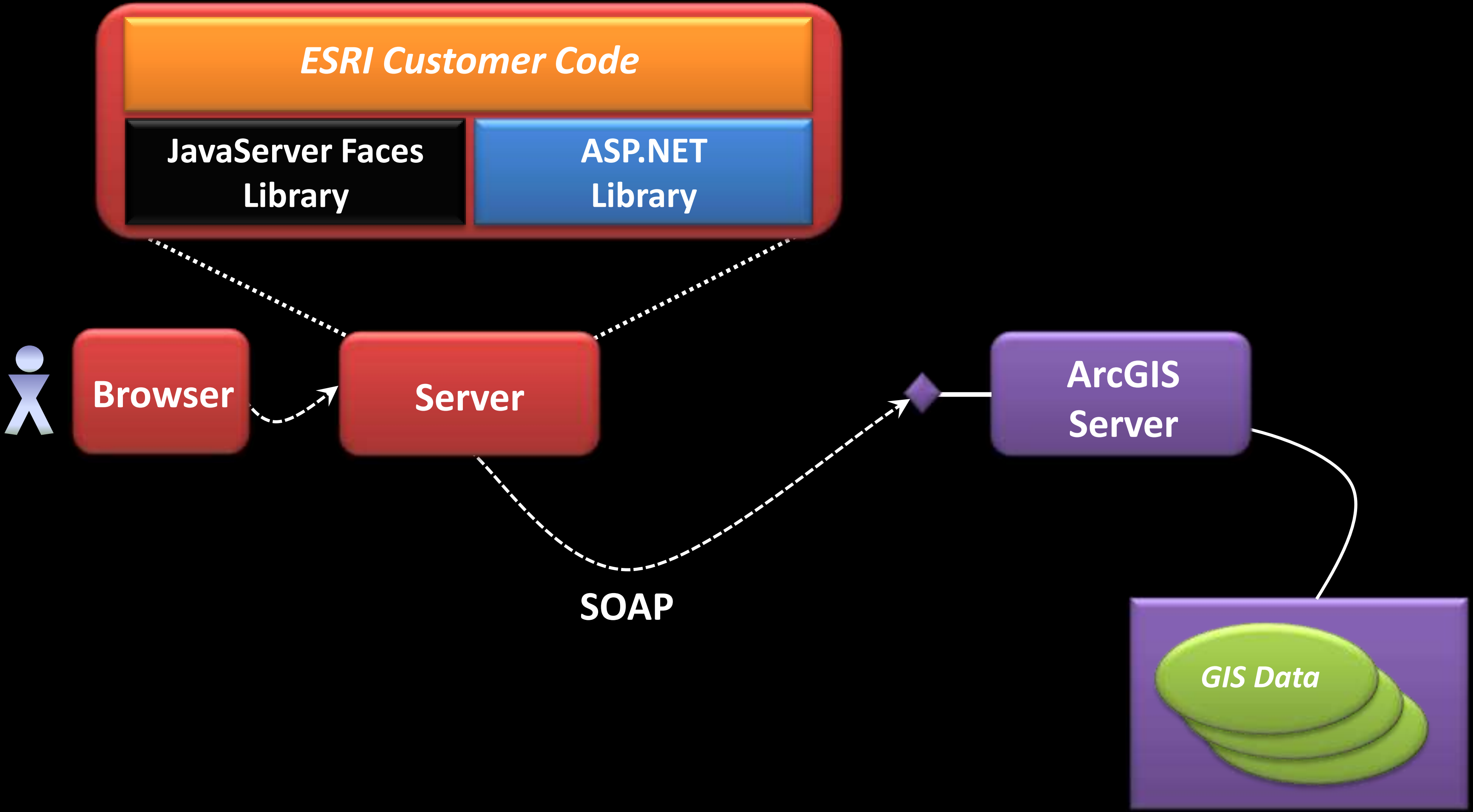
Client libraries for RESTful access from a browser



** Also supports SOAP*

ArcGIS

Client libraries for SOAP access from a server



Making A Choice
SOAP/WS-* or REST?

- Neither is right for every situation
  - Each has its place
- Some questions to ask:
  - Does the service expose data or logic?
    - REST can be a good choice for exposing data
    - SOAP/WS-* might be better for exposing logic
  - Does the service need the capabilities of WS-*, or is a simpler RESTful approach sufficient?
  - What’s best for the developers who will build clients for the service?

Conclusion

- In a service-oriented world, how services are exposed is important

- Both SOAP/WS-* and REST have good futures
  - There's good support for both approaches in .NET, Java EE, and other frameworks
  - And in ArcGIS

- The best decisions come from reason, not emotion

About the Speaker

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technology. David has been the keynote speaker for many events and conferences on five continents, and his seminars have been attended by tens of thousands of IT decision makers, architects, and developers in forty countries. His books have been published in a dozen languages and used regularly in courses at MIT, ETH Zurich, and other universities. In his consulting practice, he has helped clients such as Hewlett-Packard, IBM, Microsoft, Stanford University, and Target Corporation adopt new technologies, market new products, train their sales staffs, and create business plans. Earlier in his career, David wrote networking software, chaired a U.S. national standards working group, and played keyboards with the Peabody-award-winning Children's Radio Theater. He holds a B.S. in Economics and an M.S. in Computer Science, both from the University of Wisconsin-Madison.