A Comparison of Service-oriented, Resource-oriented, and Object-oriented Architecture Styles

Jørgen Thelin
Chief Scientist
Cape Clear Software Inc.

Abstract

- The three common software architecture styles commonly used in distributed systems and XML Web Services are compared and contrasted. In particular, the key differences between traditional SOAP and REST styles are explored. Guidelines are presented on which style is most applicable for certain application scenarios, and when a combination of styles is necessary.

Agenda

- Architecture Patterns and Styles
- Distributed System Types
  - Request / Response
  - Message passing
- Architecture styles
  - Object-oriented architectures
  - Resource-oriented architectures
  - Service-oriented architectures
- REST
- Choosing Architecture Style and Implementation Technology

Setting the Scene – Architecture Patterns and Styles

What is a Pattern?

- Martin Fowler defines a “Pattern” as:
  - An “idea” that has been useful in one practical context and will probably be useful in others”

    [Martin Fowler, “Analysis Patterns”, 1997]
- The concept of patterns can be applied at many levels in software projects:
  - Design / Code Patterns
  - Analysis / Model Patterns
  - Architecture Patterns / Architectural Styles

Pattern Levels – Design / Code Patterns

- Lowest level of patterns in software
- Based around a reusable chunk of code to solve a particular problem
- Typically implemented through source code templates and/or code generation
- Provides a "template" for implementing a system function, but requiring elaboration to complete

Pattern Levels – Analysis / Model Patterns

- Reusable object models (for example UML)
- Typically implemented through UML model templates or perhaps meta-models
- Provides a "template" for a group of related system functions, but often within a specific domain (for example Finance)

Pattern Levels – Architecture Patterns / Architecture Styles

- Reusable system structures, interconnections and interactions
- Typically implemented through architecture standards and policies
- Provides a “template” for subsystem structure and communications between subsystems

What is Software Architecture?

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

[Bass, Clements & Kazman, "Software Architecture in Practice", 1998]

What is a Software Architecture Style? #1

- An architectural style defines:
  - a family of systems in terms of a pattern of structural organization
  - a vocabulary of components and connectors, with constraints on how they can be combined

[Shaw & Garlan, "*Software Architecture: Perspectives on an Emerging Discipline*", 1996]

What is a Software Architecture Style? #2

- An architecture style:
  - Describes a **class** of architectures or significant architecture pieces
  - Is **found repeatedly** in practice
  - Is a coherent package of **design decisions**
  - Has **known properties** that permit **reuse**

  [Clements, Kazman & Klein, "Evaluating Software Architecture", 2002]
- In other words, architecture styles are like "design patterns" for the structure and interconnection within and between software systems.

Distributed Systems Architecture Styles

Distributed Systems Types

- Two main types of distributed software systems:
  - Request / Response type systems
    - Also known as “call & return” type systems
  - Message passing type systems
    - Also known as “document passing” type systems

Distributed System Type #1 – Request / Response Systems

- Request / Response type systems are:
  - Call oriented systems
  - *Usually* synchronous in nature
- Approach:
  - Operations have input parameters and output / return values
- Focus is on:
  - The particular operation to be invoked
  - The set of input values
  - The set of output values
  - The correlation of replies with requests
- No real focus on:
  - How the individual values are marshalled as a unit
  - How the output values are produced from the input values (assuming the correct output is produced!)

Distributed Systems Type #2 - Message Passing Systems

- Message passing type systems are:
  - Data oriented systems
  - *Usually* asynchronous in nature
- Approach:
  - Messages are constructed and send to a destination
- Focus is on:
  - Constructing the message payload in the correct format
  - How to dispatch the message (transport medium)
  - Where to dispatch the messages to (endpoint)
- No real focus on:
  - What will happen to messages after they are dispatched
  - Whether there will be a corresponding reply message

Architecture Styles for Distributed Systems

- For "call-based" distributed systems, there are three main architecture styles commonly used:
  - Object-oriented
  - Resource-oriented
  - Service-oriented
- Service-oriented architecture styles are frequently used with "message-passing" systems too

[but further discussion is outside the scope of this presentation]

Object-Oriented Architectures - 1

- Involve communicating with
  - A particular object instance
- Specific operations for object lifecycle management
  - E.g. EJB create, EJB remove methods
- Communications are implicitly **stateful**
  - Talking to a particular previously-created object instance

Object-Oriented Architectures - 2

- Use middleware specific protocols for communication
  - For example: IIOP, DCOM or RMI
  - Usually not Internet-friendly protocols
- Usually require **pass-by-reference** facilities
- Marshalling object references generally precludes using different types of software on client-side and server-side

Object-Oriented Architectures - 3

- All state information is held on the server-side
- Each access to the object involves a network call and round-trip communication
- Design patterns have evolved to provide ways to minimise network calls through bulk data retrieval
  - For example "Value Objects" in EJB programming

    http://www2.theserverside.com/patterns/thread.jsp?thread_id=79

Resource-Oriented Architectures - 1

- Involve retrieving particular resource instances
  - Some examples are:
    - Retrieving a HTML page using HTTP GET request
    - Retrieving a database table row using a SQL SELECT command
- Usually have operations for resource lifecycle management
  - E.g. HTTP PUT, HTTP DELETE verbs
- Requests are usually **stateless**
  - No link between one request and the next
  - Client manages any concept of "conversation state"

Resource-Oriented Architectures - 2

- Resource instances are identified by some sort of “address” data included with the request
  - Some examples are:
    - A HTTP URL
    - a WHERE clause in a SQL SELECT statement
- Encoding “parameters” into addresses can become difficult for complex query resources

Resource-Oriented Architectures - 3

- Retrieving a resource creates a (detached) snapshot of its current state on the client-side
- "Master copy" of the resource data remains on the server
- Usually can "cache" the resource data for later reuse within specified expiration times without having to re-retrieve the data

Resource-Oriented Architectures - 4

- Updates to resources:
  - Typically involve replacing the previous copy of the data with a new copy (for example HTTP PUT)
  - May also be some command verbs for doing "partial updates" (for example HTTP POST or SQL UPDATE)
- Subsequent changes to the "master copy" of the resource on the server-side are **not** automatically duplicated in the detached copies of the resource on the client-side

Resource-Oriented Architectures - Some Variations

- Variations in resource-oriented architecture style involve "distributed resource copies"
  - Multiple copies of the resource data exist
  - Changes and amendments are broadcast to keep all copies in synchronization
  - Often done using Publish/Subscribe messaging techniques
  - May have single-master, or multiple-masters

Service-Oriented Architectures - 1

- Involve communicating with
  - A specific application service
    - All messages/requests are sent to the service "endpoint"
- **No** operations for service lifecycle management
- Communications are implicitly **stateless**
  - All requests are sent to the same service endpoint
- SOA are generally more **scalable** due to stateless nature

Service-Oriented Architectures - 2

- Service endpoint decides how to process request
  - Inspects the message data content – either an "envelope" or the actual "payload" itself
- Each service has an **interface description**
  - Completely defines the message and payload formats (for example, a WSDL file)
  - Creates a **loosely-coupled** contract between client and server due to late binding

REpresentational State Transfer (REST)

- The REST approach is one of the major resource-oriented approach to building distributed systems using “pure” web technology (HTTP, HTML)

- REST (REpresentational State Transfer) is a term coined by Roy Fielding in his PhD dissertation describing a resource-oriented architecture style for networked systems

  http://www.ebuilt.com/fielding/pubs/dissertation/top.htm

REST

- Roger Costello has written a useful Tutorial and Introduction to REST:

  http://www.xfront.com/REST.html

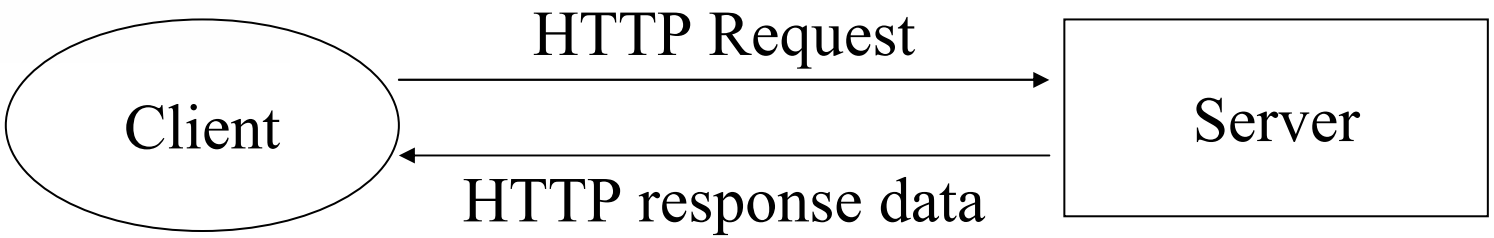
- Summary of a REST-style interaction:

  - Find or work out the resource address or **URL**
  - Retrieve the web resource using the URL
  - A **representation** of the resource is returned (such as a HTML page or an XML document)
  - The returned data is processed to place the client in a particular **state** (perhaps by rendering the HTML page)
  - Hyperlinks in the resource data can be used to retrieve related resources, which **transfers** the client to a new state (such as by rendering a different HTML page)

- The client application changes state with each resource representation retrieval --> "Representation State Transfer"

REST Example – Stage 1

```
GET http://www.TheArchitect.co.uk/weblog/index.xml HTTP/1.1
```



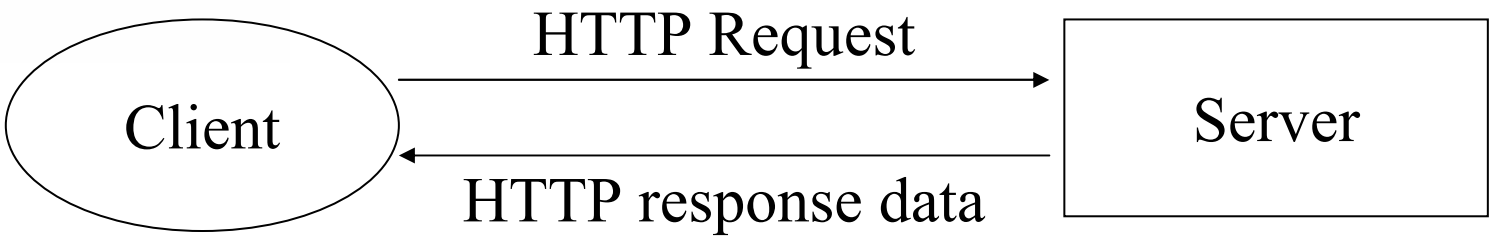
Eg. RSS data for my weblog

REST - Example return data - 1

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
<channel>
   <title>TheArchitect.co.uk - Jorgen Thelin&amp;apos;s weblog</title>
   <link> http://www.thearchitect.co.uk/weblog/ </link>
   <item>
      <link> http://www.thearchitect.co.uk/weblog/archives/2003/03/000106.html </link>
      <pubDate>Sat, 22 Mar 2003 00:01:00 GMT</pubDate>
      <guid> http://www.thearchitect.co.uk/weblog/archives/2003/03/000106.html </guid>
      <title>Internet Radio is Cool</title>
      <description>
I am sitting here in my hotel room in Salt Lake City finishing off some outstanding work,
and I can listen via the Internet to my local radio station at home - London&amp;apos;s Capital FM
Internet radio is so cool!
      </description>
      <comments>
      http://www.thearchitect.co.uk/cgi-bin/mt/mt-comments.cgi?entry_id=106
      </comments>
   </item>
</channel>
</rss>
```

REST Example – Stage 2

```
GET http://www.thearchitect.co.uk/weblog/archives/2003/03/000106.html
```



Eg. HTML page for one page in my weblog

SOAP vs. REST

- SOAP is often seen as a direct rival to a REST-based architecture, as SOAP v1.1 used a solely Service-oriented approach, and the debate from both sides has been savage!

  http://lists.w3.org/Archives/Public/www-tag/2002Apr/0235.html

- In fact, support for a more REST-based architecture style have been included in the SOAP 1.2 Specification with the new SOAP Web Method facilities:

  http://www.w3.org/TR/soap12-part2/#WebMethodFeature

- Using "RESTful SOAP" can be useful for exposing cacheable (typically read-only or idempotent) SOAP operations

- Sam Ruby has written an article comparing SOAP and REST and showing how they can co-exist peacefully together:

  http://www.intertwingly.net/stories/2002/07/20/restSoap.html

Web Services vs. REST - 1

- There is no real conflict between the general idea of Web Services and the REST approach
  - From W3C "Web Services Description Requirements" document:
    - Definition:
      A **Web Service** is a software application identified by a URI [IETF RFC 2396], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.

Web Services vs. REST - 2

- Web Service standards already support many RESTful features, and are adding more:
  - HTTP GET bindings in WSDL v1.1
  - SOAP Web Methods in SOAP v1.2
- The total set of Web Service specifications provide a superset of the REST approach – supporting Service-oriented as well as Resource-oriented mechanisms
- WSDL v1.2 should add facilities to allow the full description of the payload formats for REST / Resource-oriented approaches based in URLs
- Roger Costello has written an article on "Building Web Services the REST way" :

  http://www.xfront.com/REST-Web-Services.html

Choosing an Architecture Style

Comparison of 3 Distributed Architecture Styles

Attribute	Object-oriented	Resource-oriented	Service-oriented
Granularity	Object instances	Resource instances	Service instances
Main Focus	Marshalling parameter values	Request addressing (usually URLs)	Creation of request payloads
Addressing / Request routing	Routed to unique object instance	Unique address per resource	One endpoint address per service
Are replies cacheable?	No	Yes	No
Application interface	Specific to this object / class – description is middleware specific (e.g. IDL)	Generic to the request mechanism (e.g. HTTP verbs)	Specific to this service – description is protocol specific (e.g. WSDL)
Payload / data format description	Yes – usually middleware specific (e.g. IDL)	No – nothing directly linked to address / URL	Yes – part of service description (e.g. XML Schema in WSDL)

Choosing – Object Oriented Architectures

- Involve **tight coupling** between client and server, due to:
  - Object reference semantics
  - Object serialization
  - Early binding to interfaces
- Usually best for "closed" systems controlled by a single organization

Choosing – Resource Oriented Architectures

- Involve **loose-coupling** between client and server, due to:
  - Late binding to resource data
- Successful use revolves around the cache-ability of resource data
- So most typically used for operations which are:
  - For read-only or read-mostly data
  - Involve idempotent (repeatable) operations,
  - Return results with a "validity window" or "expiration period"
- Tend to scale well due to their stateless nature.
- Usually best for "linking and referring" across organization boundaries

Choosing – Service Oriented Architectures

- Involve **loose-coupling** between client and server, due to:
  - Late binding to service interface
  - Full interface and payload descriptions in interface contract
- Generally the most flexible
  – can support request/response and message passing systems
- Tend to scale well due to their stateless nature.
- Usually best for "shared" systems crossing organization boundaries

Combining Architecture Styles

- Usually best to stick to a single architecture style, but combinations are technically possible
- For example, a Web Service application could use a combination of architecture styles:
  - Resource-oriented approach for simple data reads
  - Service-oriented approach for complex data retrieval operations or data updates

Summary and Conclusion

Summary

- Two main distributed system types are:
  - Request / Response
  - Message passing
- Three main Request/Response architecture styles are:
  - Object-oriented
  - Resource-oriented
  - Service-oriented
- The choice of architecture style is an important decision for any software system
- Choice of architecture style can have implications on scalability, re-usability and ease of interconnection with other systems
- Web Services can be written using both Resource-oriented or Service-oriented approaches
  - SOAP v1.2 and WSDL v1.2 are helping to unify this