04 – Webservices

- Web APIs
- REST
- Coulouris
  - chp.9
- Roy Fielding, 2000
  - Chp 5/6
- Aphrodite, 2002
- http://www.xml.com/pub/a/2004/12/01/restful-web.html
- http://www.restapitutorial.com

Webservice

- "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
  - It has an interface described in a machine-processable format (specifically WSDL).
  - Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Web service

- Collection of operations
  - Used by a client over the Internet
  - Provided by various resources
    - Programs, objects DB
- Managed
  - Along a web server
  - Separate service
- Used to manipulate resources

Web Service API Styles

- RPC API
  - Xml-rpc
- Message API
  - SOAP+WSDL+..
- Resource API
  - REST

RPC on the web

- How can clients execute remote procedures over HTTP?
  - Define messages that identify the remote procedures to execute and also include a fixed set of elements that map directly into the parameters of remote procedures.
  - Have the client send the message to a URI designated for the procedure.

RPC on the web

- A service makes available a Service Descriptor
- The Service Descriptor is used to generate a Service Connector (proxy) on the client.
  - The descriptor might be coded with WSDL, XSDL or a non--XML approach (JSON--RPC)
- The client calls operations on the Service Connector as if it were the service.
- Frameworks such as JAX--WS (Java) and WCF (Microsoft) makes all of this easy

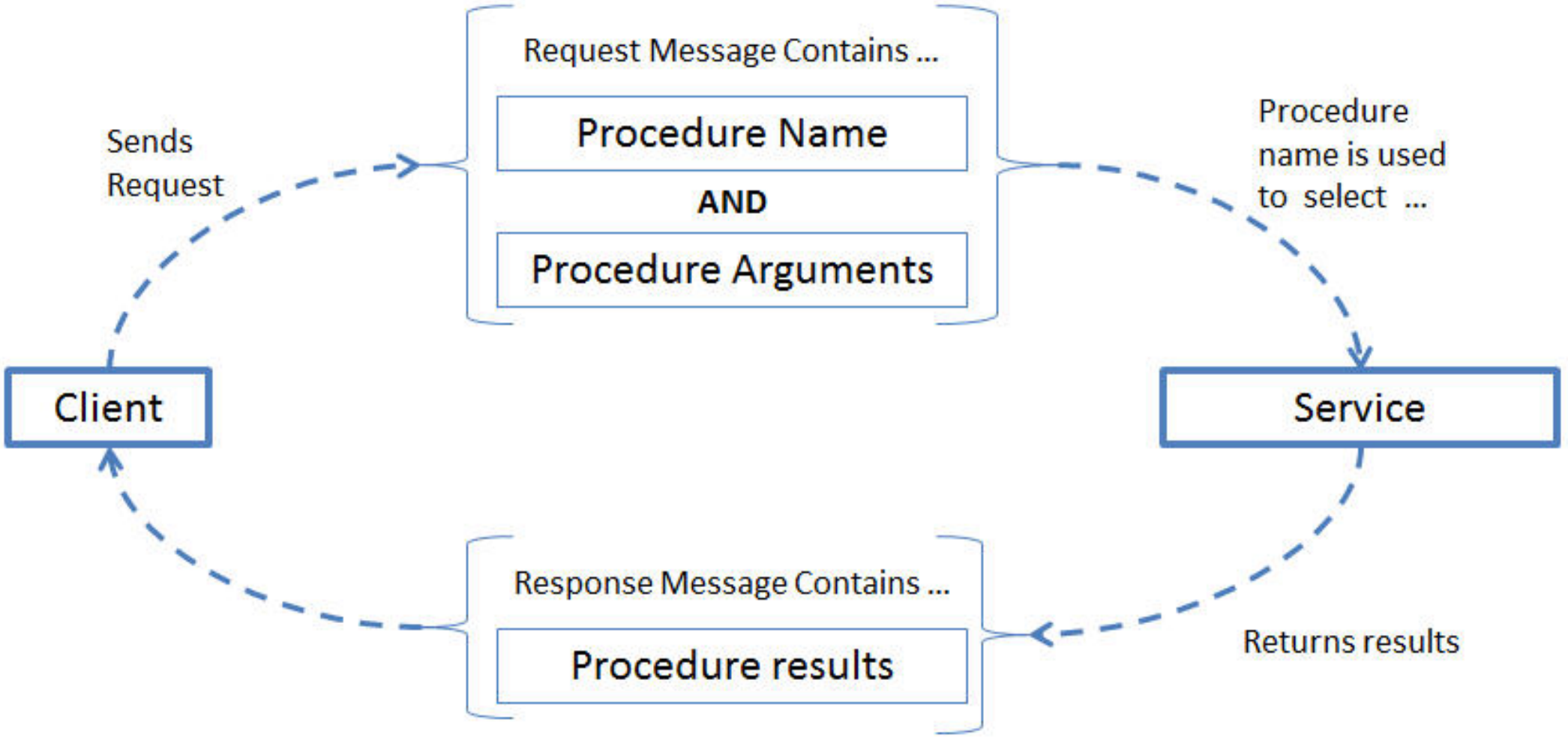
RPC on the web

- Methods or procedures may contain a list of typed parameters.
  - This is a tightly coupled system.
    - If the parameter list changes this approach breaks clients.
    - A Descriptor change forces the Connector to be regenerated on the clients
- A less tightly coupled system would contain a single Message Argument

RPC on the web

- Request/Response is the default but it may be replaced by Request/Acknowledge
- Request/Acknowledge is less tightly coupled in time.
  - The request can be queued for later processing.
  - May improve scalability.
- The response may still be received with
  - Request/Acknowledge/Poll
  - Request/Acknowledge/Callback
- Clients may use an Asynchronous Response Handler

RPC on the web



http://www.servicedesignpatterns.com/WebServiceAPIStyles

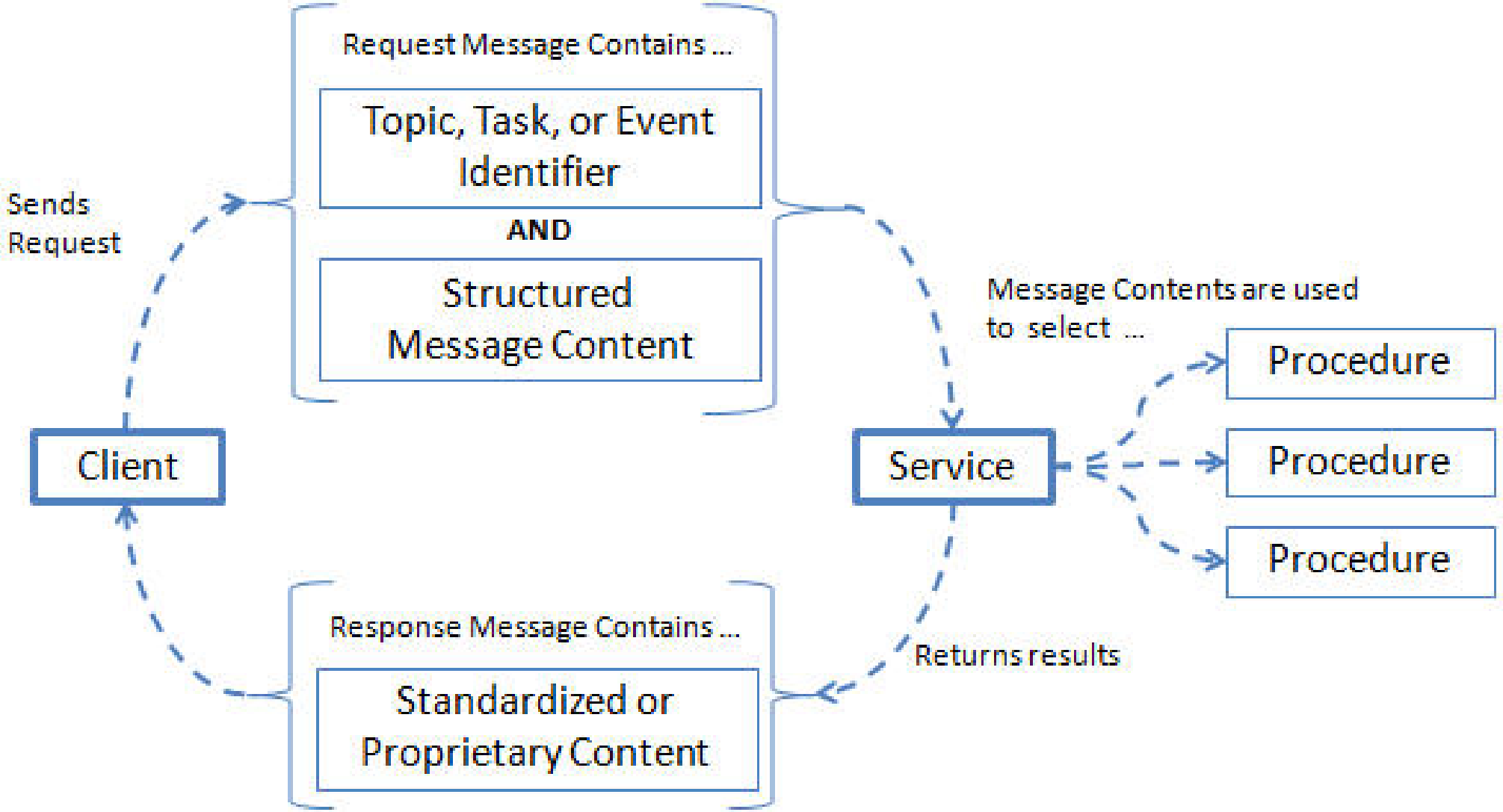
Messages API on the web

- How can clients send commands, notifications, or other information to remote systems
  - over HTTP
  - avoiding direct coupling to remote procedures?
- Using messages that are not derived from signatures of remote procedures.
- When the message is received, the server examines its contents to determine the correct procedure to execute.
- The web service is used as a layer of indirection by insulating the client from the actual handler

Messages API on the web

- The web service serves as a dispatcher and is usually built after the message is designed.
- No procedure name or parameter list is in the message.
- The service descriptor is often WSDL and XSDL.
- The services descriptor is used to generate the service connector (proxy).
- XSDL, UDDI, SOAP

Messages API on the web



http://www.servicedesignpatterns.com/WebServiceAPIStyles

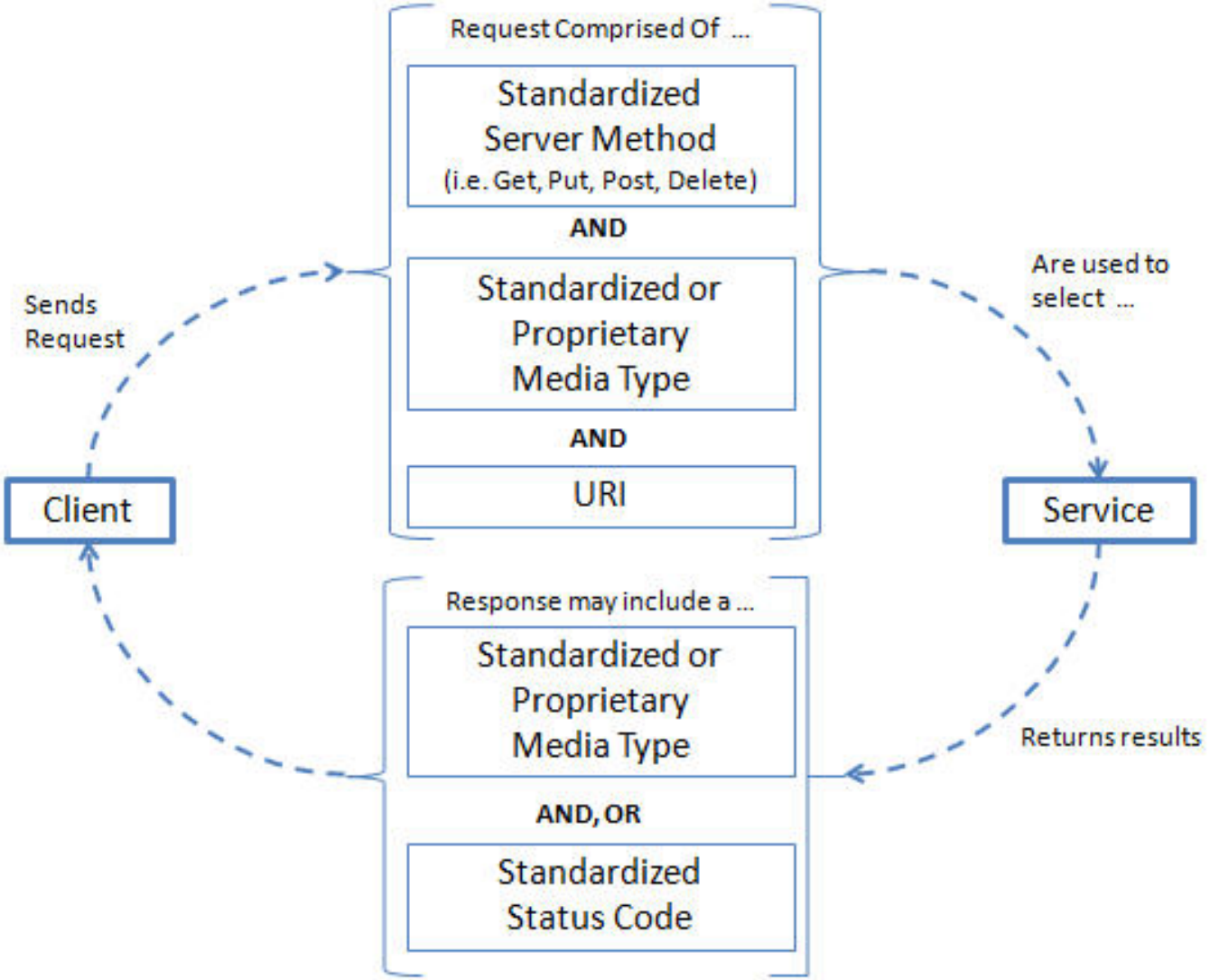
Resources on the Web

- A client application consumes or manipulates
  - text, images, documents, or other media files
  - managed by a remote system.
- How can a client manipulate data managed by a remote system?
- Avoiding direct coupling to remote procedures
- Minimizing the need for domain specific API's?
  - Assign all procedures, instances of domain data, and files a URI.
  - Leverage HTTP as a complete application protocol to define standard service behaviors.

Resources on the Web

- Exchange information by taking advantage of standardized media types and status codes when possible.
- The client's intent is determined by
  - a) the HTTP method used
  - b) the URI
  - c) the requested or submited media type.
- These services often adhere to the principles of representational State Transfer
  - Not all Resource API is to be considered RESTful

Resources on the Web



http://www.servicedesignpatterns.com/WebServiceAPIStyles

WS considerations

- Combination of multiple Web services
  - In various ways (sequencing, delegation....)
  - Replication
  - Offer new functionalities
- Loose coupling
  - Use of interfaces
  - Simple generic interfaces
  - Selected communication paradigms

WS considerations

- Representation of messages
  - XML / JSON
    - Less performance
    - Human readable
  - XML allows extensibility
- Service references
  - URI (not necessarily URL)
- Activation
  - Accessed from a computer (on the URI)
  - Always running vs activated on request

WS considerations

- Transparency
  - Not offered by infrastructure
  - Explicit data marshaling
  - Explicit remote call
- Local APIs
  - Language level
  - Hide implementation details
  - Data marshaling / communication
- Proxies/Dynamic invocations

REST

- Representational State Transfer
- Architectural style
- Derived from
  - Existing architectural styles
  - Additional constrains
- Rest derivation
  - Definition of constrains

Client server constrain

- Separation of concerns
  - User interface (client) separated from data storage
- Increases portability of the UI
- Increases scalability
  - Simple server components
- Separated evolution of the components
  - Supporting Internet scale
    - Multiple organizational domains

Stateless constrain

- Communication should be stateless
- Each request
  - contains all necessary information
  - Cannot take advantage of context on the server
- Session state is kept on the client

Stateless constrain

- Increases visibility
  - Monitorin only needs request data
- Increases reliability
  - Eases faillure recovery
- Increases scalability
  - No storage between requests
    - Resources freed quickly
    - Less server state
  - Lower processing
    - Stored state is not processed in sequential requests

Stateless constrain disadvantages

- Design choice
- Decrease network performance
  - Repetitive session data
- Reduces control over consistency
  - Application state on the client
  - Multiple client versions
    - Multiple semantics

Cache constrain

- Improve network efficiency
- Data in responses should be explicitly labeled
  - Cacheable
  - Non-cacheable
- Cache client can reuse response data

Cache constrain

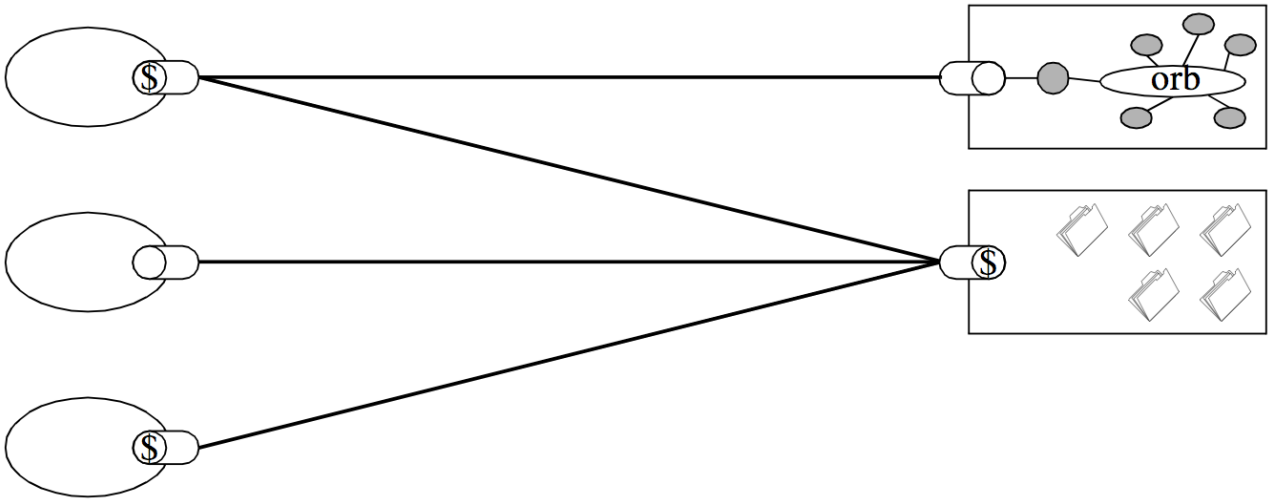
- Improves efficiency
  - Eliminates some interactions
- imp[roves scalability
- Improves performance
-
- Increases inconsistencies
  - Between cached values and real values

Modern web architecture

- Client-cache-stateless-server
  - Prior to 1994
  - Exchange of static documents
- Extension
  - Dynamically generated responses
    - Image-maps, server-side script
- Require further constrains

Uniform Interface constrain

- Uniform interface between components
- Generality of components interface
  - Simplifies architecture
  - Improves visibility of interactions
- Implementation is decoupled from service



Uniform Interface constrain

- Efficient for large-grain hypermedia transfer
  - Optimizes common case
  - Degrades efficiency
    - Standard ways to transfer data
- Uniform interface constrains
  - Identification of resources
  - Manipulation of resources trough representation
  - Self descriptive messages
  - Hypermedia as engine application state

Layered system constrain

- Allows hierarchical layers
  - Compositions of systems
  - Encapsulation of other layers
    - A layer only sees upo to the interface of lower service
- Bound to overall complexity
- Promotion of substrate independency
- Good fit with interface uniformity

Layered system constrain

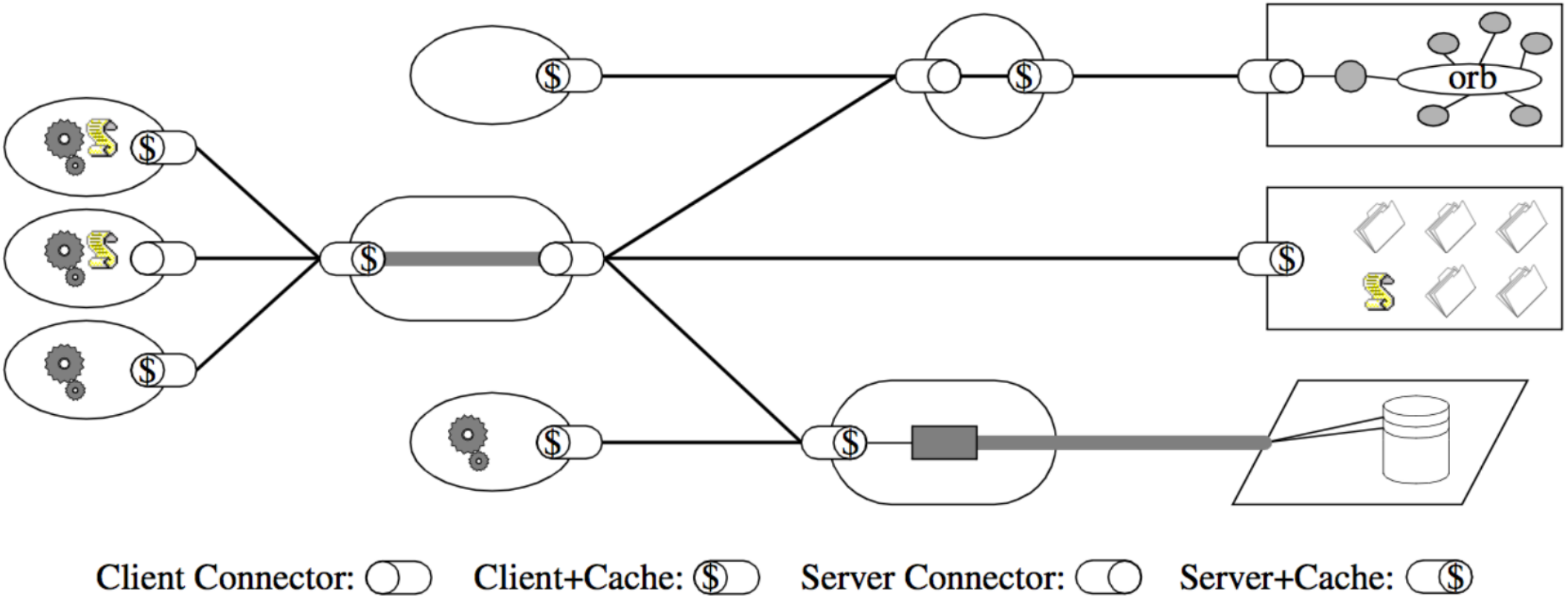
- Add overhead and lattency
  - Reducinf user-perceived performance
  - Offset by cacheing
- Caches on organizational domain boundaries
  - Increase performance
- Security policies can be enforced
  - On data crossing boundaris

Code on demand constrain

- Optional constrain
- Rest allows client functionality to be extended
  - Applets or scripts
- Simplifies clients
- Improves extensibility

REST

- Client-server
- Stateless
- Cache
- Uniform interface
- Layered
- Code on demand



REST Data elements

- Distributed objects (RMI)
  - Data is encapsulated by processing elements
- REST
  - State is a fundamental element
  - As in distributed hypermedia
- Selection of a link
  - Information moves from storage to user

Information transfer

- Render data on server
  - Send fixed-format result to recipient
  - Common in Client-server
- Encapsulate data + processing engine
  - Send both to recipient
  - Common in mobile-code
- Send raw data + metadata to recipient
  - Recipient chooses how to process

Information transfer

- Render data on server
  - Encapsulation
  - Simple client
- Encapsulate data + processing engine
  - Information hiding
  - Specialized processing
  - Limits functionalities
- Send raw data + metadata to recipient
  - Simple sender
  - No information hiding
  - Requires client to know formats

Rest Information transfer

- Hybrid
  - Shared understanding of data types
    - Metadata
  - Limiting scope of what is revealed
    - Using standard interface
- Components communicate
  - Transferring representations of a resource
  - Standard format
  - Selected dynamically

Rest data elements

- resource
  - the intended conceptual target of a hypertext reference
  - Any information that can be named can be a resource
- resource identifier
  - chosen to best fit the nature of the concept being identified
  - URL, URN
- resource metadata
  - source link, alternates, vary
- representation
  - HTML document, JPEG image
- Representation metadata
  - media type, last-modified time
- control data
  - if-modified-since, cache-control

Representation

- A representation is a sequence of bytes plus representation metadata
- May also include resource metadata
  - Information about the resource not specific to the representation
- Data format of a representation known as a media type
  - Design of a media type may influence user perceived latency

Connectors

- Encapsulate the activities of accessing resources and transferring resource representations
  - Provide clean separation of concerns
  - Provide substitutability by hiding implementations and allowing them to be replaced

Connector	Modern Web Examples
Client	libwww, libwww-perl
Server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
Resolver	bind (DNS lookup library)
Tunnel	SOCKS, SSL after HTTP CONNECT

How to create REST interfaces?

- Answer in order:
  - What are the URIs?
  - What is the format?
  - What methods are supported at each URI?
  - What status codes could be returned?
-
- http://www.restapitutorial.com

What are the URIs

- sensible resource names
  - /customers/12345/orders vs /api?type=user&id=12345%info=orders
  - provide context
  - increasing understandability
- Implement hierarchies
  - hierarchical nature of the URL to imply structure
  - Design for your clients
- Resource names should be nouns
- Use pluralization to indicate the collection
  - Avoid using collection verbage in URI's (eg. 'customer_list')

What is the format

- What are the resources representations?
  - HTML
    - Human readable
  - XML or json
    - Machine readable
  - Images
- Single/multiple representatiuon by resource?
- Textual description
  - Json or XML
- Binary
  - Any
- Use HTTP content-type / content negotiation

What methods at each URI?

- Universal verbs
  - Create Read Update Delete
  - Map into HTTP protocol
  - Assigned to each resource (URI)
- POST
  - Create new information
- GET
  - Read information
- PUT
  - Update information
- DELETE
  - Delete information

HTTP methods

- Safe methods
  - GET, HEAD, OPTIONS and TRACE
    - intended for retrieving data
- Unsafe
  - May change result
  - POST, PUT, and DELETE
- Idempotent
  - PUT and DELETE (and all safe methods)
- Not idempotent
  - Post (cdeates multiple resources)

Returned Status Codes

- 200 OK
  - This response code indicates that the request was successful.
- 201 Created
  - This indicates the request was successful and a resource was created. It is used to confirm success of a PUT or POST request.
- 400 Bad Request
  - The request was malformed. This happens especially with POST and PUT requests, when the data does not pass validation, or is in the wrong format.
- 404 Not Found
  - This response indicates that the required resource could not be found. This is generally returned to all requests which point to a URL with no corresponding resource.
- 401 Unauthorized
  - This error indicates that you need to perform authentication before accessing the resource.
- 405 Method Not Allowed
  - The HTTP method used is not supported for this resource.
- 409 Conflict
  - This indicates a conflict. For instance, you are using a POST request to create the same resource twice.
- 500 Internal Server Error
  - When all else fails; generally, a 500 response is used when processing fails due to unanticipated circumstances on the server side, which causes the server to error out.

Returned Status Codes

Resource	Method	Representation
Employee	GET	Employee Format
Employee	PUT	Employee Format
Employee	DELETE	
All Employees	GET	Employee List Format
All Employees	POST	Employee Format

Resource	Method	Representation	Status Codes
Employee	GET	Employee Format	200, 401, 404
Employee	PUT	Employee Format	200,401, 404,
Employee	DELETE		200, 204
All Employees	GET	Employee List Format	200, 301
All Employees	POST	Employee Format	201, 400

URI

- No REST
  - GET /adduser?name=Robert HTTP/1.1
- RESTful
  - POST /users HTTP/1.1
  - Host: myserver
  - Content-Type: application/xml
  - <?xml version="1.0"?>
  - <user>
  - <name>Robert</name>
  - </user>
  -

Directory like URI

- Intuitive URI's
  - Self documented interface
  - Structure
    - Straightforward, predictable easy to understand
- Tree of subordinate and superordinate nodes
  - http://www.myservice.org/discussion/topics/{topic}
  - http://www.myservice.org/discussion/2008/12/10/{topic}
  -

URI

- Hide the server-side scripting file extensions
  - .jsp, .php, .asp
  - so you can port to something without changing URIs.
- Keep everything lowercase.
- Substitute spaces with hyphens or underscores (one or the other).
- Avoid query strings as much as you can.

Content Negotiation

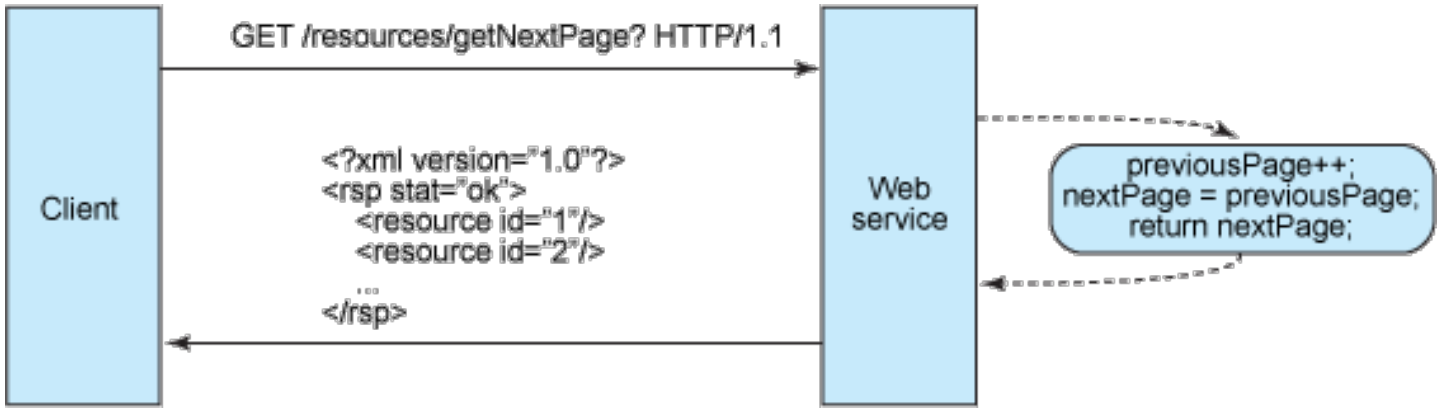
- No REST
  - GET /users/Robert?format=xml HTTP/1.1
- RESTfull
  - GET /users/Robert HTTP/1.1
  - Host: myserver
  - Accept: application/xml
  -

Data updates

- No REST
  - GET /updateuser?name=Robert&newname=Bob HTTP/1.1
- RESTful
  - PUT /users/Robert HTTP/1.1
  - Host: myserver
  - Content-Type: application/xml
  - <?xml version="1.0"?>
  - <user>
  - <name>Bob</name>
  - </user>

Be stateless

- Statefull design



- Stateless design