Mapping the RESTful Programming Model to the DDS Data-Centric Model

Rick Warren, Principal Engineer rick.warren@rti.com

Conclusion

- REST is architecture of the web, works great for the web
- RESTful view of DDS object model is very powerful
  - Network effect makes hyperlink-able DDS data more valuable
  - Single data model can support clients with variety of needs
  - RESTful DDS compatible with strict per-user access control
- (Many ways to create such a view; we'll consider a few)

- *Sorry, no demo in this talk—see http://www.youtube.com/user/RealTimeInnovations*

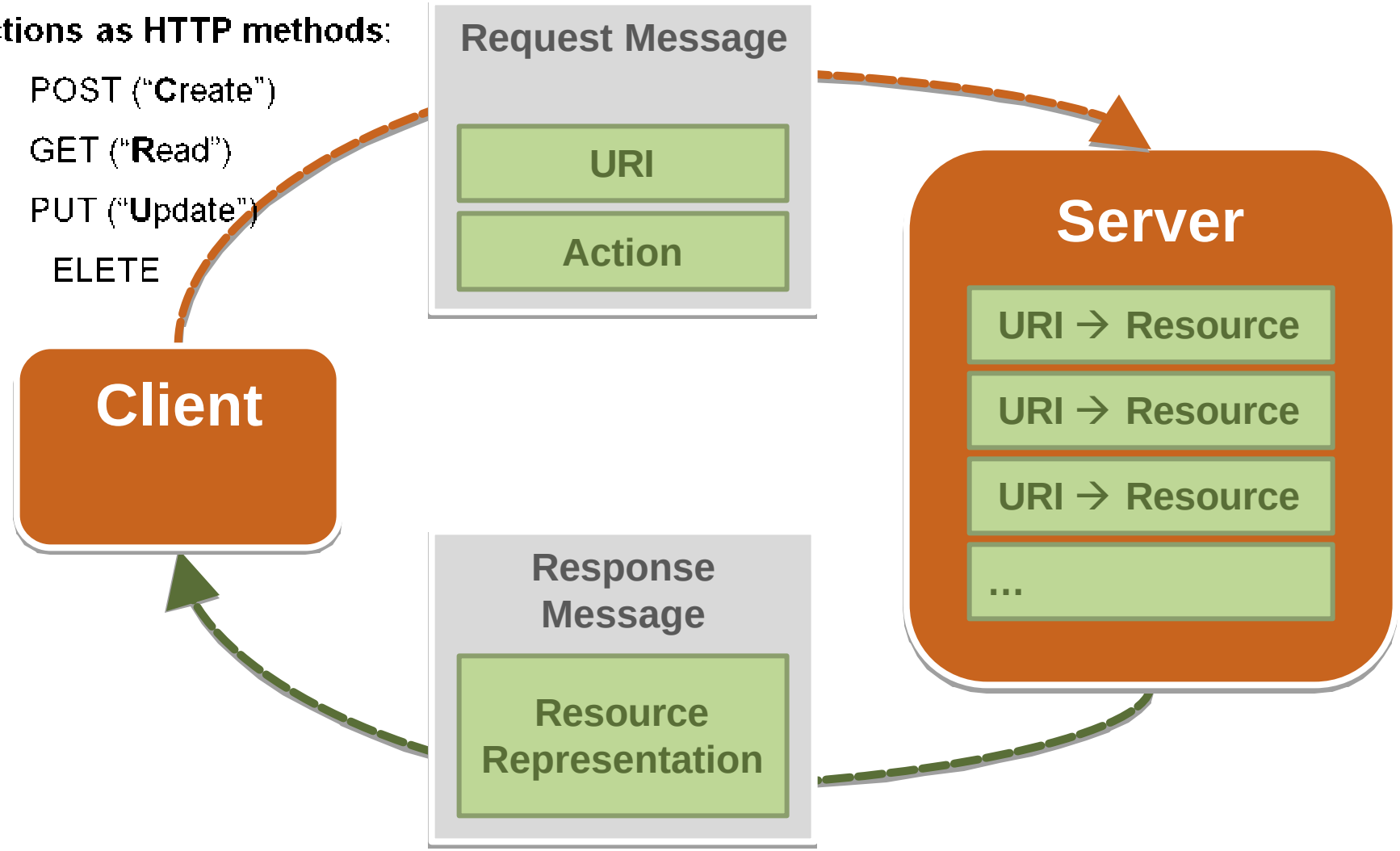
What is REST?

- **RE**presentational **S**tate **T**ransfer
  - Coined by Roy Fielding, co-inventor of HTTP, URIs, Apache web server, fire, and the wheel
- A **system architecture**
  - Designed for large distributed systems like the web
  - Emphasizes statelessness, interface simplicity and uniformity
- **Misuse**: a **technique of sending messages** with HTTP
  - Frequently contrasted with SOAP
- **I will both use and misuse this term in this talk.**

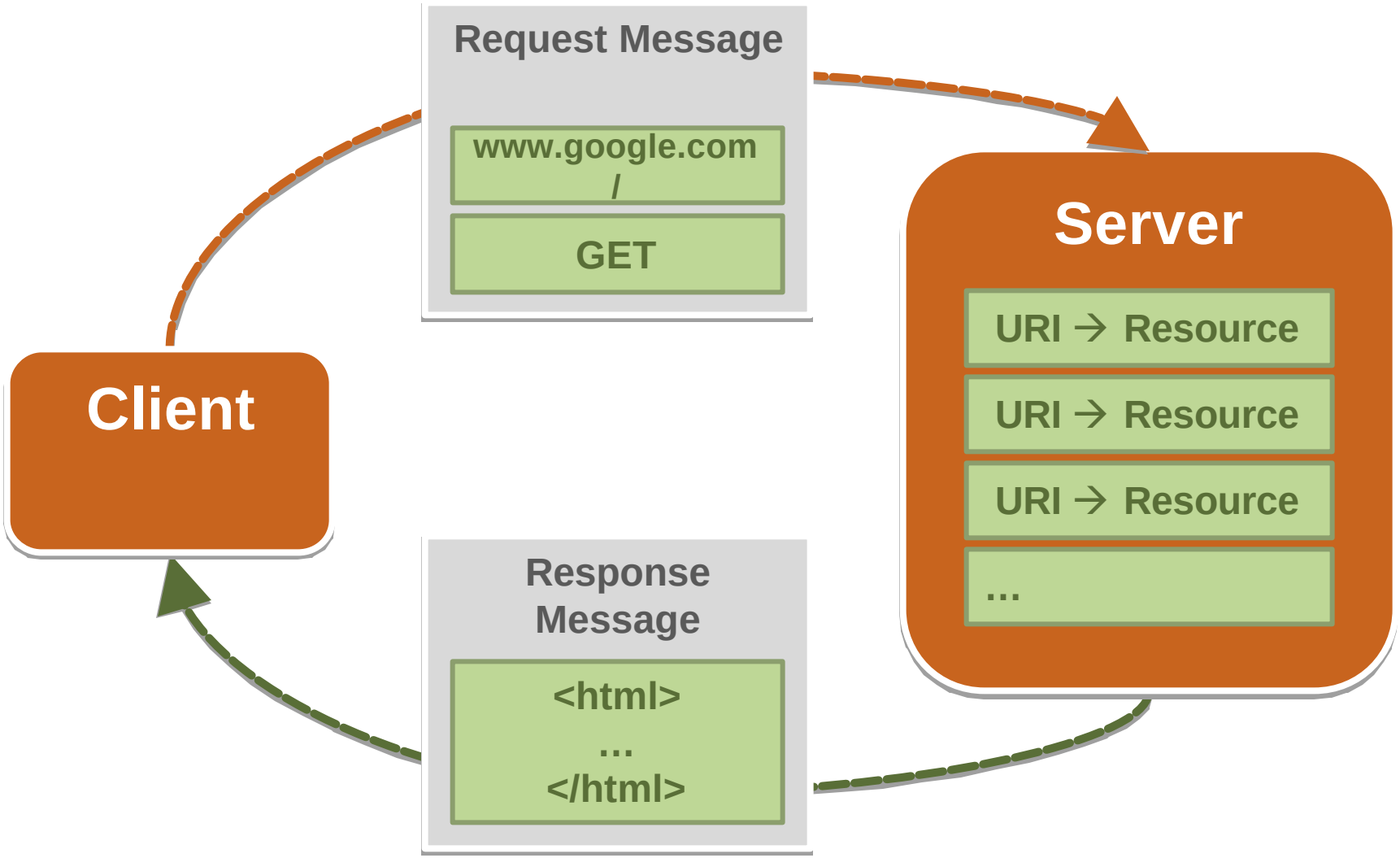
What is REST/HTTP?

Actions as HTTP methods:

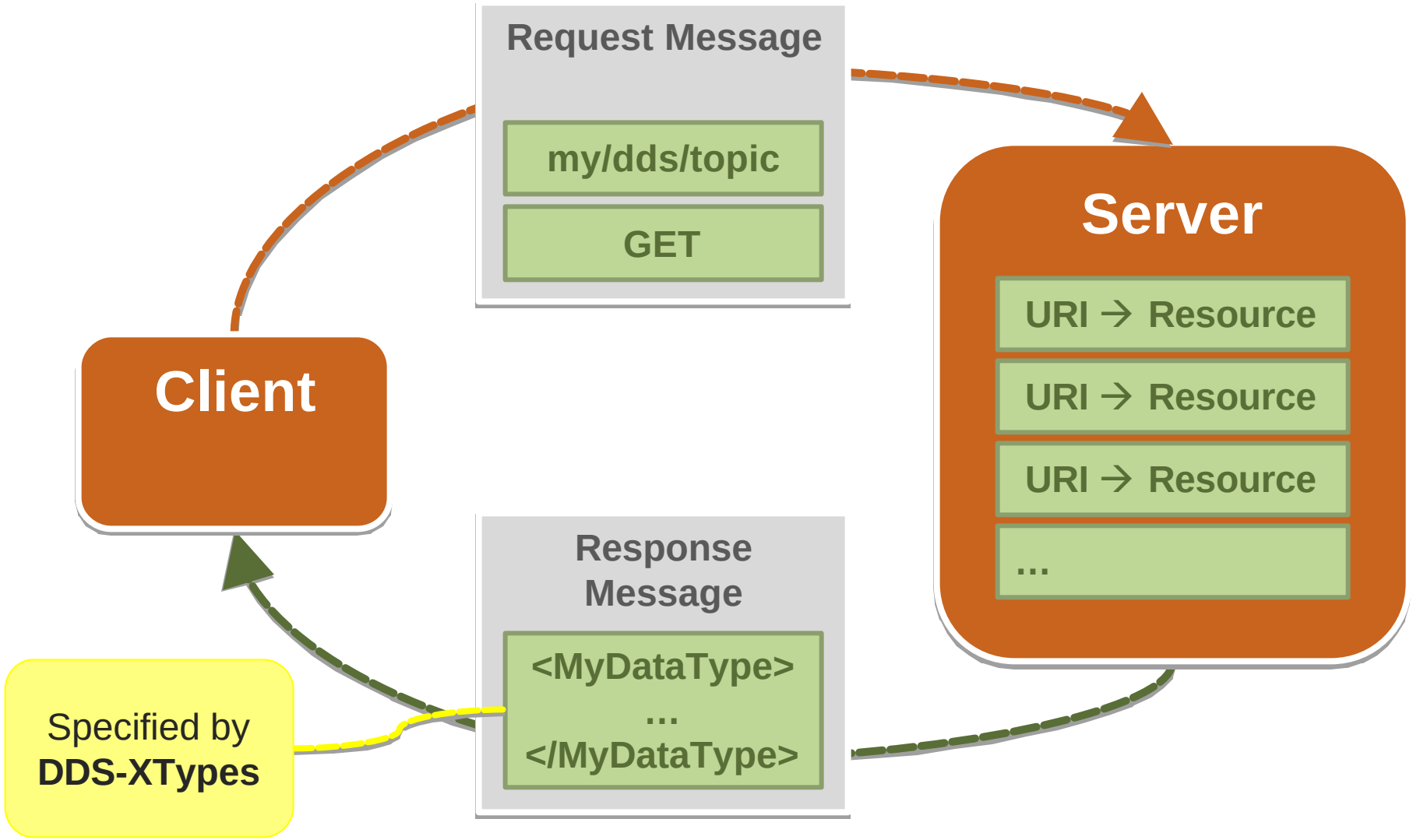
- POST ("**C**reate")
- GET ("**R**ead")
- PUT ("**U**pdate")
- ELETE



What is REST/HTTP? Example (Web Browser)



What is REST/HTTP? Example (DDS)



Why Is DDS-REST Integration Promising?

- **Vision of the web**:
  - All information everywhere could be accessible by anyone
  - …if only they knew where it is and how to get it.
  - *BTW: "where it is" = URI; "how get it" = HTTP*
- **Why should information inside DDS be different?**
  - Approach #1: Convince everyone to replace HTTP with DDS
  - Approach #2: Allow poor misguided HTTP users to get DDS data anyway
  - *Web-Enabled DDS standard initiative at OMG will help us capitalize on this vision*
- "Give me a URI and a protocol to dereference it and I shall download the world."
  - *Archimedes, 250 B.C.*

DDS-REST Integration Teaser

- `myReader.read()`
  - `GET http://…/myTopic`
- `myWriter.write()`
  - `POST http://…/myTopic`
- Any web page can be a DDS application

Implementing Web Services: SOAP vs. REST

SOAP/WSDL

- *Technology*
- Define domain-specific nouns (services) and verbs (operations)
  - Meaning is explicit
  - Can lead to wheel reinvention
- Only service itself is linkable
- Standard message envelope
  - Layer metadata onto any service
    - e.g. for governance
  - Complicates processing

REST

- *Architecture/design pattern*
- Universal syntax for nouns (URIs) and verbs (e.g. HTTP methods)
  - Polymorphism allows broad reuse
  - Ad hoc def'ns vary widely
- Whole resource tree is linkable: network effect
- Each service defines own formats, soup to nuts
  - Get exactly what you need in user-friendly format (e.g. JSON)
  - Difficult to apply metadata across services

Procedural REST: a "Hybrid" Approach

- Some REST services embed custom verbs in URI
- Example: Amazon Simple Queue Service

  `http://queue.amazonaws.com/`*`namespace/myQueueName`*
  `?Action=ReceiveMessage``&...`

- **Pro**: Obvious mapping to/from SOAP for easy skills xfer
  - Base URI is the service endpoint
  - Operations and args are parameters
- **Con**: Encourages thinking "invoke this operation" instead of "access this resource"
- **Why is this bad?** Encourages GET-only interfaces
  - *Style*: like a language with only one word; don't discard rest of HTTP
  - *Substance*: GET must be *safe*, *idempotent*
  - *Best practice*: resources nouns; verbs come from HTTP—don't reinvent things HTTP already provides

RESTful DDS: It's All About the Resources

- **Resources** (w/ URIs)
  - Data
    - Application data
    - Discovery data
    - Natural hierarchy is `<domain>/<topic>[/<instance>]`
  - Metadata
    - Entities (participants, readers, writers, topics…)
    - Data types
    - QoS
    - Status
- **Non-resources**
  - *Conditions and WaitSets*: concurrency is server's business
  - *Listeners*: status is a resource; access like any other
- **Other concerns**
  - Access control
  - Session management

RESTful DDS Entities: Open Questions

- **Q**: DDS object model or simplified model?
  If DDS, are all entities useful to web clients?
- **Q**: Publish/subscribe to single topic in multiple ways?
  (e.g. multiple readers w/ different QoS)
- **Q**: Require explicit client entity management?
  (e.g. POST to /datareader before GET from /topic)

- **Trade-off**:
  - Simplicity for the user, flexibility for the implementation
    vs.
    Power for the user, constrained implementation
  - *Can we have it both ways?*

RESTful DDS Resource Proposal: Data View

- /dds/<*domain ID*>
  - /topic/<*topic name*>
    - /data
      - ?sample_state=<*value*>&filter_expression=<*expr*>&…
      - /<*preconfigured instance name*>
    - /qos
    - /status
      - /inconsistent_topic
    - /type
    - /datareader
      - /qos
      - /status
        - /requested_deadline_missed
        - /requested_incompatible_qos
        - /…
    - /datawriter
      - /…

RESTful DDS Resource Proposal: Entity View

- /dds/<*domain ID*>
  - /participant/<*entity name*>
    - /qos
    - /topic/<*entity name*>
      - /…
    - /subscriber/<*entity name*>
      - /qos
      - /datareader/<*entity name*>
        - /data
        - /qos
        - /status
    - /publisher/<*entity name*>
      - /…

RESTful DDS Resource Proposal: Examples

Data View

- POST http://…/dds/0/topic/MyTopic/data
  ```
  <MyType>
    Hello
  </MyType>
  ```
  - Response: 200 OK
- GET http://…/dds/0/topic/MyTopic/data
  - Response:
    ```
    <MyType>
      Hello
    </MyType>
    ```

Entity View

- PUT http://…/dds/0/participant/MyParticipant …
- PUT http://…/dds/0/participant/MyParticipant/topic/MyTopic …
- PUT http://…/dds/0/participant/MyParticipant/publisher/MyPub …
- PUT http://…/dds/0/participant/MyParticipant/publisher/MyPub/datawriter/MyWriter
  ```
  <datawriter>
    <topic>MyTopic</topic>
    <qos>…</qos>
    …
  </datawriter>
  ```
- …

RESTful DDS Resource Proposal: Examples

- **Create entire tree of entities**:
  - *Request*:

```
PUT http://www.example.com/dds/0/participant/
    MyNewParticipant
<participant name="MyNewParticipant">
  <qos>…</qos>
  <subscribers>
    <subscriber name="MyNewSubscriber">
      <datareader>…</datareader>
    </subscriber>
  </subscribers>
  …
</participant>
```

  - *Response*: `200 OK`

Access Control

- **Requirement: Control access** by each principal
  - Publish, subscribe on certain topics
    - …with certain contents
    - …with certain QoS

- **Implication: Principals' pub/sub entities must be separate**
  - To avoid data "leakage"
  - To avoid contention/concurrency coupling
  - To manage entity lifecycles: DDS is not stateless

Access Control → Session Management

- **Approach: Associate request w/ principal's *session***
  - Client requests new session, providing identity and credential
    - Service responds with access token on success ("session ID")
  - Client includes session ID in subsequent requests
    - Service authorizes request; proceeds or fails
  - Client requests session end, or service cancels session

1. **Q#1: How to get session identifier?**
2. **Q#2: How to send session ID back again?** Options:
   a) Query parameter
   b) Resource URI
   c) Request body
   d) Cookie

Getting a Session ID ("Logging In")

- **Possible approaches**:
  - `/dds/log_in`
    - **No**: Doesn't imply how to use it (GET? POST?)
    - **Remember**: Resources are nouns. Verbs are CRUD.
  - `/dds/session_id` (GET)
    - OK, so now it's a noun.
    - We want to "get" a new session ID, right?
    - …But still **No**: need to provide credential, but GET can't have body
    - …and logging in is not "safe" operation, nor idempotent
  - `/dds/session` (POST)
    - **Now we're talking**: Want to create new session in a collection—that's what POST does.
    - Request body contains credential(s)
    - Implies administrative `GET /dds/session`: list all current sessions

Handing Back Session ID: #1, Query Param

- **Example**:
  `GET /dds/0/topic/MyTopic/data`**`?session=123`**
- **Good**
  - Simple, explicit
  - Works when GETting resources (very common)
- **Bad**
  - May not work well when POSTing resource
    (e.g. HTML forms don't allow POST with query)
  - URIs not shareable
    - *Bob*: "Get my data from here: http://bob.com/dds/…?session=Bob."
    - *Fred*: "Ooh, look at this…" ::*takes actions with Bob's credentials*::

Handing Back Session ID: #2, Resource URI

- **Example**:
  `GET /dds/`**`MySession`**`/0/topic/MyTopic/data`
- **Good**
  - Simple, explicit
  - Works equally well with all HTTP methods
- **Bad**
  - URIs still not shareable
    - Even harder to “cleanse” URI than with query parameter

Handing Back Session ID: #3, Request Body

- **Example**:
  `POST /dds/0/topic/MyTopic/data`
  `(body: session=123)`
- **Good**
  - URLs shareable without editing
  - Works well with POST, PUT, DELETE
- **Bad**
  - Doesn't work when GETting resources (very common)

Handing Back Session ID: #4, Cookie

- **Example**:
```
GET /dds/0/topic/MyTopic/data
Cookie: session=123
```
- **Good**
  - URLs shareable without editing
  - Works equally well with all HTTP methods
  - Time-out built in
  - Proven approach
- **Bad**
  - Cookie stored with client's user agent (e.g. web browser)
    - Harder to share across agents
      - …or maybe Good: forces re-authentication
    - Shared across users that share agent
      - Separate user accounts of multi-user platforms work around this
- ***This is the recommendation***

Conclusion

- REST works best when taking full advantage of HTTP
  - Think in resources
  - Use GET, POST, PUT, DELETE as intended
  - Respect safety, idempotency expectations
- DDS and REST are a powerful combination
  - Network effect makes hyperlink-able DDS data more valuable
    - Single DDS data set available via DDS network—
      or from any browser
  - DDS resource model supports clients with various needs
    - Simplified basic data access
    - Complete control over DDS entities and their configurations
  - RESTful DDS can provide strict per-user access control

Thank You