RESTful Moonshot

A framework approach

Executive Summary

REST (Representational State Transfer)[1] is a design suggestion. The suggestion is (simply): In a Client-Server architecture, do not allow "state" to be held on the server. Active state is to be held on the Client and persisted in a repository.

State is the opposite of computation. State can be created (by a human) or computed. Once data is stable then it is static (i.e. immutable or unchanging). Therefore, state is represented by the attributes of a thing (i.e. object attributes).

The idea of REST becomes more understandable when read backwards: Transfer of State Representation (e.g. attributes). Transfer from what to what? From Client to Server for the purpose of performing work on the Server (in a service—that is—command or query).

Transfer Semantics

If the word semantic means purpose or reason, then we must ask: For what purpose or reasons do we transfer state (attributes) from a Client to a Server and what are the mechanisms of that transfer?

1. Command
    a. Using **Arguments_n** compute **Data_collection_n**
    b. Store Data_collection_n in a repository
2. Query
    a. Fetch **Data_collection_n** from a repository based on **Query_n**
    b. Using **Arguments_n** compute **Result_n**

Command (**Arguments_n**) / Query (**Arguments_n**)

1. Client sends request with **Arguments_n**[2] data as **JSON** to Server
    a. Code pattern to turn data into **JSON**
    b. Code pattern for making request with **JSON**

---

[1] See also: URI-to-HTTP relationship table.

[2] For any semantic, the **JSON** being sent for any reason looks precisely the same (e.g. **Arguments_n** and **Data_collection_n** differ only in name—they are both **JSON**).

2. Server handler unpacks **Arguments_n**
    a. Use the json_ext library to deserialize **JSON** to Object(s)
3. Server handler performs calculation[3]
    a. Compute **Data_collection_n** as attributes on Server Object(s)
4. Server packs up **Data_collection_n** into response
    a. Use the json_ext library to serialize Server Objects to **JSON**
    b. Form into WSF_PAGE_RESPONSE
        i. Will the data be cached or not?
5. Server sends response
    a. Typical a_response.send (l_data)
6. Client receives response and unpacks **Data_collection_n**
    a. JS function that receives data as a *callback*
    b. *Callback* function unpacks **JSON** and distributes

Store **Data_collection_n** in **Data_repository_n**

1. Client sends request with **Data_collection_n** data as **JSON** to Server
    a. Code pattern to turn data into JSON
    b. Code pattern for making request with **JSON**
2. Server handler unpacks **Arguments_n**
    a. Use the json_ext library to deserialize **JSON** to Object(s)
3. Server stores Object(s) in **Data_repository_n**
    a. Examples:
        i. **JSON** in files
        ii. EAV Database
        iii. RDBMS Database

[3] This is a model-level computation involving only data sent from the Client or data stored in the repository (or both).