Architecture Principles

Description of the Deliverable for the Development Phase

The architecture principles aim to help implementers understand how to create conformant solution architectures through the implementation of the DDF Study Definition Reference Architecture (RA). They inform solution architects of the approach expected by the RA stakeholders to ensure consistency across Study Definition implementations and to ensure alignment with the business and technology objectives. Architecture principles define the fundamental assumptions regarding the RA and aid in developing a framework for decision making by solution architects implementing the RA.

Summary of work to be performed during scoping

A framework for the architecture principles was developed during the scoping period.

REST Architecture Principles

The Digital Data Flow (DDF) Unified Study Definition Model (USDM) Reference Architecture (RA) has an Application Programming Interface (API) component developed using the REST (REpresentational State Transfer) architecture. REST defines an architectural style that includes design constraints used to specify how the architecture of a system like the DDF USDM software should function. The *Representational* element of REST means that the API represents the systems underlying resources, or data objects, using different media types that can be requested by clients. For example, a client may request a study objective endpoint resource as JSON or XML. A resource is an object with a type, associated data, relationships to other resources, methods that operate on it, and is the fundamental concept in a REST API. The *State Transfer* element of REST means that the client stores the state in it's own application/session. When a client requests a resource from the server using the API, the client sends the relevant state information to the server so that the server has the context needed to provide the correct response.

REST Architecture Principles

- REST implements a client server architectural style that creates a layered system for which REST provides the interface. REST supports the separation of concerns principle by requiring that clients use the API to access the application and data layers of the DDF USDM architecture. Clients cannot see beyond the provided interface. This enables the other layers to evolve independently from the interface, simplifies each of the layers, and can improve scalability. It also simplifies access to USDM content and allows the widest range of clients to consume the content and services. Participants in loosely coupled distributed applications, such as those developed using the DDF USDM and its RESTful API, are free to work with the content they receive in any way they wish.
- REST exposes data as objects, called resources, and typically uses the HTTP verbs for actions on those objects. A resource provides the source of specific information and can include a global permanent identifier such as can be provided by an HTTP Uniform Resource Identifier (URI). A URI uniquely identifies a resource, and makes it addressable, or capable of being manipulated using a protocol such as HTTP. Endpoints are URIs or Uniform Resource Locators (URLs).

- REST requires a standard interface, such as that provided by HTTP, to support the principle of generality. The DDF USDM REST API anticipates HTTPS as the primary protocol for making requests to the server and uses HTTP verbs to convey actions. HTTP uses a relatively small number of verbs with well-defined and widely accepted semantics. The interface provides the means to exchange representations of resources. The HTTP methods used by the API may include:
    - GET retrieves a representation of the resource at the specified URI, and the body of the response contains the resource details.
    - POST creates a new resource at the specified URI. The body of the request message provides the details of the new resource. Note that POST can also be used to trigger operations that don't create new resources.
    - PUT either creates or replaces the resource at the specified URI. The body of the request message specifies the resource to be created or updated.
    - PATCH performs a partial update of a resource. The request body specifies the set of changes to apply to the resource.
    - DELETE removes the resource at the specified URI.
- REST servers should be stateless to support the architecture principles of visibility, reliability, and scalability. Stateless servers require that each request from client to server contain the information necessary to understand and complete the request without requiring that state information be stored on the server.
- REST responses should be cacheable to improve performance.
- REST supports content negotiation such that clients can request specific content and media types.
- REST uses hypermedia as the engine of application state (HATEOAS) where responses include URLs relevant to the context of the requested resource. A hypermedia-driven system provides information to navigate the system's REST interfaces dynamically by including hypermedia links with the responses.

Technology Independence

Any client should be able to call the DDF USDM API, regardless of the specific API implementation as long as it conforms to the specification. APIs should de-couple source from target, reducing dependence on specific technologies and thus reducing the cost of change. The API uses standard protocols, such as HTTP, and a mechanism for the client and the server to agree on the data exchange media type. API clients specify their preferred media types in the HTTP Accept header. As long as clients and servers adhere to the API specification and data model, implementations that pass the conformance tests can be developed using a wide-range of technologies, supporting the principle of technology independence. Application independence from the underlying technology allows software to be developed and updated independently without being subject to continual obsolescence and vendor dependence.

Interoperability

The DDF USDM software conforms to defined standards to promote interoperability for data, applications, and technology. Standards improve interoperability and help to ensure that multiple vendors implement products based on the standards, and facilitates supply chain integration. Standards help ensure consistency reducing the cost to produce, update, and manage systems thereby protecting existing IT investments and maximizing return on investment.

Data Standards and Controlled Vocabularies

The use of existing industry data standards and controlled vocabularies improves interoperability and increases the availability of supporting software tools. The use of data standards ensures data is defined consistently across organizations, and that the definitions are available and understandable to all users. The use of data standards and controlled terminologies facilitates the data exchange, simplifies systems integration, and enables data aggregations to support analytics.

Contract-first Design

The OpenAPI Initiative was created by an industry consortium to standardize REST API descriptions across vendors. OpenAPI promotes a contract-first approach, rather than an implementation-first approach. Contract-first means you design the API contract by defining the interface in a specification first and then write code that implements the contract.

Filter and Paginate Data

Exposing a collection of resources through a single URI can lead to applications fetching large amounts of data when only a subset of the information is required. This impacts performance, causes client usability problems, wastes network bandwidth, wastes server processing power, and is highly inefficient. The API allows the client to pass a filter in the query string of the URI that filters the information requested. The server application parses and handles the filter or page parameter returning the filtered results to the client

Discoverability and Follow-your-nose

With HATEOAS, a client interacts with a DDF USDM API whose application servers provide information dynamically through hypermedia. A DDF USDM REST client needs little to no prior knowledge about how to interact with the API or server beyond a basic understanding of hypermedia. The client makes an HTTP request of the API through a simple URL and all subsequent requests are discovered inside the responses to each request. The media types used for these representations, and the link relations they may contain, are standardized. The client transitions through application states by selecting from the links within the resource representation or by manipulating the resource representation in other ways afforded by its media type. In this way, hypermedia drives the REST API interaction, and not out-of-band information. The server should be descriptive enough to instruct the client how to use the API via hypertext only.

Governance

An important aspect of governance is API versioning. Careful versioning strategy allows consumers enough time to update their software to adopt to new API versions. With that, an industry standard of at least two versions of backward compatibility is recommended. API should be versioned numerically using single digit increments, i.e., 1, 2, etc. Each API version may have multiple releases. A new API version should be considered when significant functionalities are being added or drastic changes to response structure are being introduced. API version should be clear in either a path parameter (e.g., /v1/endpoint) and/or the response structure. API version as query parameter may be less ideal as stated in RESTful principles above a standard interface is preferred. Careful

consideration should be given to supporting multiple API versions simultaneously, as this certainly presents operational challenges.

Specification for each API release should be carefully reviewed for consistency including, but not limited to style, meaningful reuse of repeatable components, and examples.