Protocol

The possible protocol to be regarded here is a kind of meta-protocol in the style of the OData protocol being implemented over HTTP. Thus, for example, will implement a representation framework (as HTML is for a ‘document’ web oriented application stack) via a dynamic hypermedia aggregation of path traversals.

It should to be able to retrieve knowledge, information, facts about ‘subjects’: 4D (where, when, who, state dimensions: categories / profiles). Patterns. Node responses. Index servive.

Hierarchical contexts provides query / assertion / reply interfaces. Graph / nested contexts (tree / lists) models. Paths. Node endpoints dynamically allocated into graphs according identity, attributes and contextual comparison dimensional alignments compose resolvable paths semantics. Registry service.

Dialog: ask / assert / reply pattern in context path location. RDF Message based representations. Dialog variables, placeholders. Pattern based subscriptions. Naming service.

‘Accounts’ (Consumer / Subscriber context, interactions): backend, user clients and agents consolidated and syndicated dispatch of ‘gestures’ (paths messages plus command statements) translated to any given protocols / IO. Context graph ‘reactive’ dataflow activation.

Flows / scenarios (contexts / roles / state / transitions). Discovery. Subscriptions. Ontology alignment.

Protocol layer (over HTTP):
URI scheme (paths, node patterns, subscriptions HATEOAS HAL / JSONLD browseable applications).
Representations: RDF Message.
Command Statement (via HTTP methods defines how representation messages are handled):

C: Path.
S: Assert.
P: Query.
O: Reply.

Protocol 'semantics' (Discovery, Subscriptions, REST):
Subscription: REST Resource (feed / queue), declaratively stated patterns (Binding).

Nodes. Resources / Patterns. Data (Fact, Event), Information (Kind, Rule), Knowledge (Class, Flow) instance / schema dataflow activation (metamodel reactive IO).

Contexts. Endpoints. Paths. Hierarchical / graph aggregation of node resources identifiers. Resolvable 'patterns' to nodes in hierarchy according context and contents. ‘Posting’ (requests) occur in the scope of a context and ‘activates’ (async, message oriented) reactions with corresponding data, information and knowledge handled by the node.

Accounts (contexts). Dialogs (interactions). Subscriptions (data). Declaratively stated by 'patterns' (resource templates with 'paths': model data, application information and domain knowledge levels).

Protocol: submit 'paths' to 'paths' (functional semantics). CSPO Statements reified / encoded as 'paths'. Return 'paths', rel discovery by comparison alignment (referrer). CRUD is performed on the requesting side by means of returned results augmenting node metamodel. Intermediate requests augments requested nodes metamodel.

Example: from ‘Peter’ resource in ‘Employment’ (referrer context) to ‘Country’: all Peter’s Countries and the relationship with them (i.e.: countries where Peter has worked in).

CSPO Paths: Patterns. C: Path, S: Assertion, P: Query, O: Answer 'patterns'. Discovery by comparison alignment of obtained resource 'paths' rel with goal 'pattern' (referrer).

Example encoding:

C: Context / Path (instance identifier aggregates SPO into pattern.

SPO: /subjectPath[path]/predicatePath[path]/objectPath[path]

De referenceable resource: aggregated Message (IO).

Domain translation: fulfill (dynamic) template.

Representations (requesting client metamodel resources) are built upon aggregating and aligning protocol dialog 'path' resources into data (Fact, Event), information (Kind, Rule) and knowledge / behavior (Class, Flow) in the requesting node, maybe by multiple ‘posts’ / traversals of activated contexts. Those are the same models which get 'activated' in the requested side by means of async messages IO.

Application layer (Message encoded as RDF. Reactive dataflow). Bindings: Client APIs (DOM / JAF / DCI).

Dashboard: actionable domain translation of problem spaces flows.

Lab

OntoClean. Primitives (Number). Upper ontology. Bit map coding (digit, position). Octal comparison results mask (order rel expressed in three bits).

Number (Base, Digit*). Base : Number. Digit : Number, Position. Position : Number.

(Instance: Base, S: Number, P: Position, O: Digit);

(Occurrence, Resource : Number, Attribute : Base, Value : Digit);

(Fact1, Peter, wife, María);

(Fact2, Man, relationshipWith, Woman);

Patterns. Comparisons. Order / contexts. Comparing two facts (resources) in a given context must shield a third resource (alignment, discovery).

Knowledge to compare Fact1 as instance of Fact2. Knowledge to compare two facts (in functional context) and sort them ('causeOf', 'after', 'before', 'partOf', 'equality' example contexts) according resulting resource.

Resource Number: aggregated attribute/value pairs (define resource by extension).

Comparison encoding: Render (recursively) compared occurrences SPOs as Number / Base / Digit. Convert compared resources SPOs into functional context SPOs. Obtain comparison result operating converted resources (more specialized hierarchy match). Return functional contexts compared resource.

URI encoding (SPO instance, occurrence):

/resource[attr, val]/context[attr, val]/resource[attr, val]#instanceId

Example:

person[name, Doe]/country[code, ar]/employment[salary, highest]#johnJobs

Resources: comparable / function (Doe, highestSalaryJobs) : Jobs (high salary);

Context: referrer (Doe, highestSalaryJobs, Argentina) : Jobs (high salary, Argentina);