URI: Universal Resource Identifier

**Protocol:** // Location ? Query

- Specification of fixed identifier language.
- Location of resource on internet with file path.
- Sequences of key/value pairs of strings.

The URI combines choice of protocol with a path location and a query string of key/value pairs.

**UI: Universal Information Identifier**

- Specification of Query language, type structure, and interpretation protocol.
- Declarative logic formula specifying information in terms of typed variables.

The UI combines a logic formula query language with a specification metalinguage for the query.

---

**Breaking down The Cloud**

**Monolithic App / Service**

- Hypertext UI: Interaction through browser or mobile app.
- RESTful API: HTTP GET, POST, PUT, or DELETE.
- Server determines semantics of the URI, requestor does not know.
- Local account required, data not directly accessible.

**Modular Apps / Services**

- Hypertext UI: served as template and bundled with queries.
- RESTful API: exposed as set of queries each encoded in a URI.
- Server determines semantics of requests, including possible side effects, expressed in the URI.
- Fine grained access control: authorization per query.

---

**Old Topology**

Independent proprietary databases

- Data
- Information requests to databases are performed through service providers.
- Information to be applications and services.

Web application and service providers

- Interactions with web applications and services, returning content involving personal information.

**New Topology**

Secure personal, public, and proprietary datatstores

- Data
- Information requests to datatstores are performed by users, not service providers.
- Syndication also possible between datatstores.

Web application and service providers

- Interactions with web applications and services, returning content templates with request schemas for personal information.

---

**Problem: URI Query Semantics**

Query → Kv & Kv & ...

Kv → string = string

There is no general syntax or typing semantics for query strings other than a sequence of key/value pairs. There is no general way for the client to syntactically check or type check queries sent to web services.

APIs and DSLs must be embedded in strings. Web apps must then parse them out and check them. Since this is done differently for every platform, there is no common way to learn the syntax of a DSL through reflection.

---

**New Architecture**

The UI provides universal access, sharing is not limited to a specific Cloud platform.

- All personal data exists at one logical location, the datastore.
- Application development becomes less expensive: data platforms do not have to be independently developed.
- Richer possibilities for data acquisition and cross domain aggregation open up.

A common interface metalinguage yields better interoperability.

- Users are not locked into a single service since personal data remains independent.

---

**Example: Online Social Networking**

SNP = friend(AtMe, X) ∧ location(X) = “Berkeley” ∧ status(X) = Y

(Y, Y) ∈ ("(@Chris, "Reading Nietzsche -D3", (@Marten, "Debugging Polymy 2-/")..)...

SNP = (X, (friend(AtMe, X) ∧ event(X, E)) Y pubEvent(E, “Berkeley") ∧ date(E) = today()

(Y, ("Marten’s dinner party") ∧ (Chris’s Lauren working group meeting") ∧ ("UCB Chancellor's lecture")...

SNP = VX, (friend(AtMe, X) ∧ friend(AtMe, Y) ∧ friend(AtMe, Y)

(Y) ∈ ("Roger Paradox", "Enrico Generoso",..)

SNP = friend : Person → Person = 0 isToday = ¬(e) ∧ date(e) = today()

location : Person → Location pubEvent = λ(e, 1).

status : Person → Text event(e, null) ∧ location(e) = 1
date : Event → Date

event : Person → Event = 0 (... etc)
isToday = Event = 0

pubEvent : Event → Location = 0