SEC-13: Rethinking the Cloud

Marten Lohstroh, Chris Shaver @ UC Berkeley

Connect

Marten Lohstroh Chris Shaver

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.

UII : Universal Information Identifier

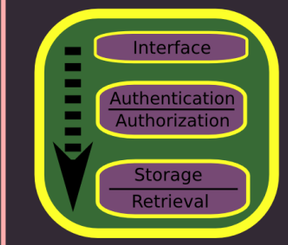
`Context ⊨ Query`

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

The UII combines a logic formula query language with a specification metalanguage 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 the semantics of the URI, requestor does not know.

Local account required, data not directly accessible.

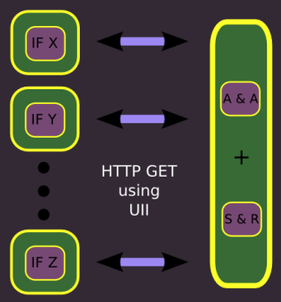
Modular Apps / Services

Hypertext UI: served as template and bundled with queries.

Restful API: exposed as set of queries each encoded in a UII.

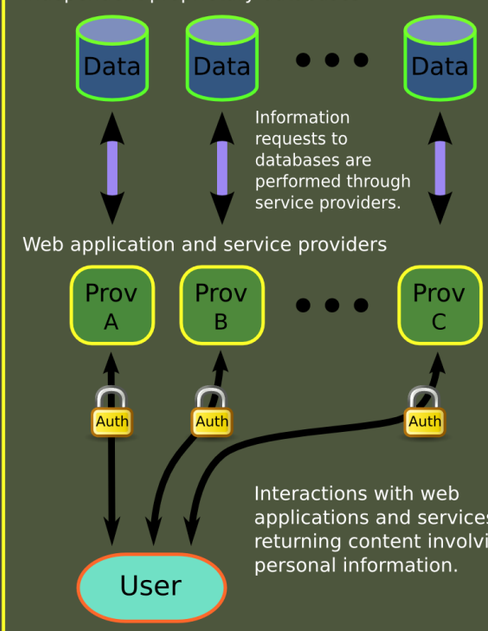
Semantics of requests, including possible side effects, are expressed in the UII.

Fine grained acces control: authorization per query.



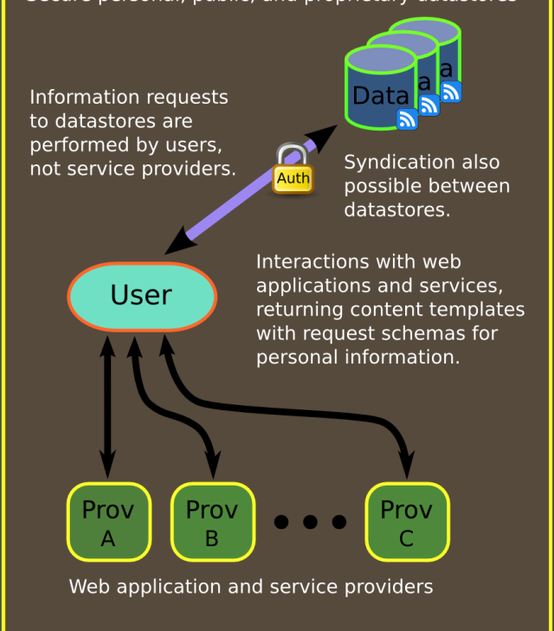
Old Topology

Independent proprietary databases



New Topology

Secure personal, public, and proprietary datastores



Context

Defines the language used to make queries and specifies their interpretation.

Declarations *Typing* `term : term`

Definitions *Substitution* `term ⇛ term`

Type term syntax with standard constants and constructors

```
type ⟶ ⊥, T, Ω, type × type, type + type, type ⇒ type,
       Int, String, Float, ... (standard primitives),
       ... (types declared in context)
```

Protocol *Interpretation*

The protocol specifies how to find valuations of free variables in queries. Satisfying solutions can be returned together as a set, as a stream, the first solution can be returned, etc...

Query

Query formula syntax with standard operators and quantifiers

```
term ⟶ var, term term, (term, term, ...), λ var.term,
       ... (standard primitive terms),
       ... (terms declared in context)

query ⟶ a, term = term, false, true,
        query ∧ query, query ∨ query, query ⇒ query,
        ∀ var.query, ∃ var.query  where a:Ω
```

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 UII 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 metalanguage yields better interoperability.
- Users are not locked into a single service since personal data remains independent.

Example: Online Social Networking

```
SNP ⊨ friend(@Me, X) ∧ location(X) = "Berkeley" ∧ status(X) = Y
      (X, Y) ∈ {(@Chris, "Reading Nietzsche :D"), (@Marten, "Debugging Ptolemy II :-/"), ... }
SNP ⊨ (∃X.(friend(@Me, X) ∧ event(X, E)) ∨ pubEvent(E, "Berkeley")) ∧ date(E) = today()
      (E) ∈ {("Marten's dinner party"), ("Chris's Lacan working group meeting"), ("UCB Chancellor's lecture"), ... }
SNP ⊨ ∀X.(friend(@Me, X) ⇒ friend(X, Y)) ∧ ¬friend(@Me, Y)
      (Y) ∈ {("Roger Paradox"), ("Enrico Generoso"), ... }

SNP = friend   : Person ⇒ Person ⇒ Ω      isToday  ⇛ λ (e) . date(e) = today()
      location : Person ⇒ Location        pubEvent ⇛ λ (e, l) .
      status   : Person ⇒ Text                 event(e, null) ∧ location(e) = l
      date     : Event ⇒ Date
      event    : Person ⇒ Event ⇒ Ω       (... etc)
      isToday  : Event ⇒ Ω
      pubEvent : Event ⇒ Location ⇒ Ω
```