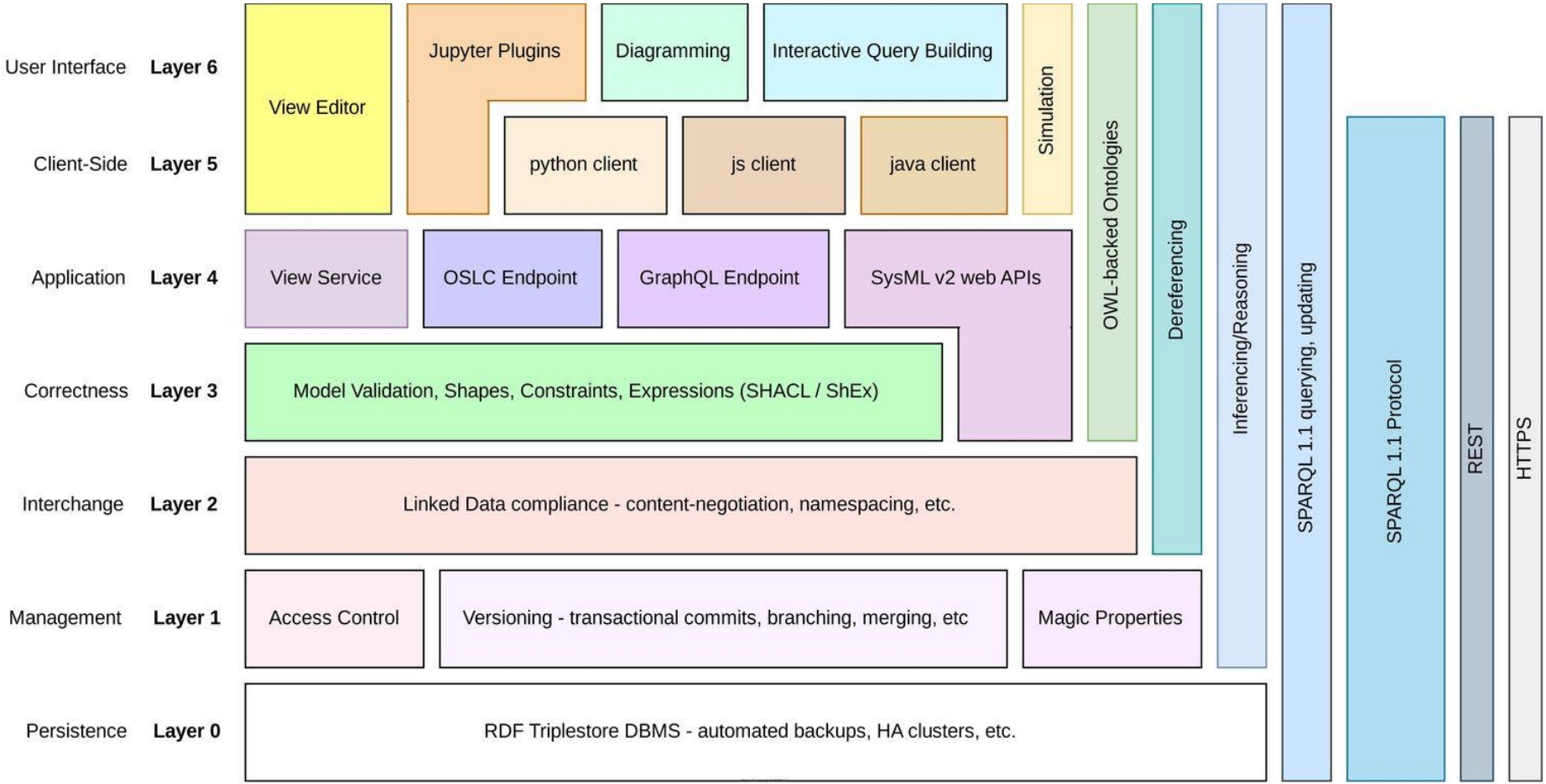
Generating Flexo APIs & SDKs for "model" developers

A generic, schema-driven approach to Flexo Layers 4 + 5

Blake Regalia
OpenMBEE


User Interface
Layer 6
Client-Side
Layer 5
Application
Layer 4
Correctness
Layer 3
Interchange
Layer 2
Management
Layer 1
Persistence
Layer 0
View Editor
Jupyter Plugins
Diagramming
Interactive Query Building
python client
js client
java client
Simulation
View Service
OSLC Endpoint
GraphQL Endpoint
SysML v2 web APIs
Model Validation, Shapes, Constraints, Expressions (SHACL / ShEx)
Linked Data compliance - content-negotiation, namespacing, etc.
Access Control
Versioning - transactional commits, branching, merging, etc
Magic Properties
RDF Triplestore DBMS - automated backups, HA clusters, etc.
OWL-backed Ontologies
Dereferencing
Inferencing/Reasoning
SPARQL 1.1 querying, updating
SPARQL 1.1 Protocol
REST
HTTPS

What are the givens?

- Graph-based data store with a self-describing data model
- Extensible microservice architecture
- Very broad ecosystem of tools supporting fractions of the “big picture”
- Existing frameworks and UIs in Layers 5 and 6 already familiar to end-users
- Vendors exposing endpoints using OpenAPI spec

Where are the gaps with reaching “model” developers?

- SPARQL is a steep learning curve
- Graph-based data models and querying are not intuitive to Excel users
- “There is never a one-size-fits-all solution to Linked Data user interfaces” ™
- Post-process-ability, data flexibility, computability

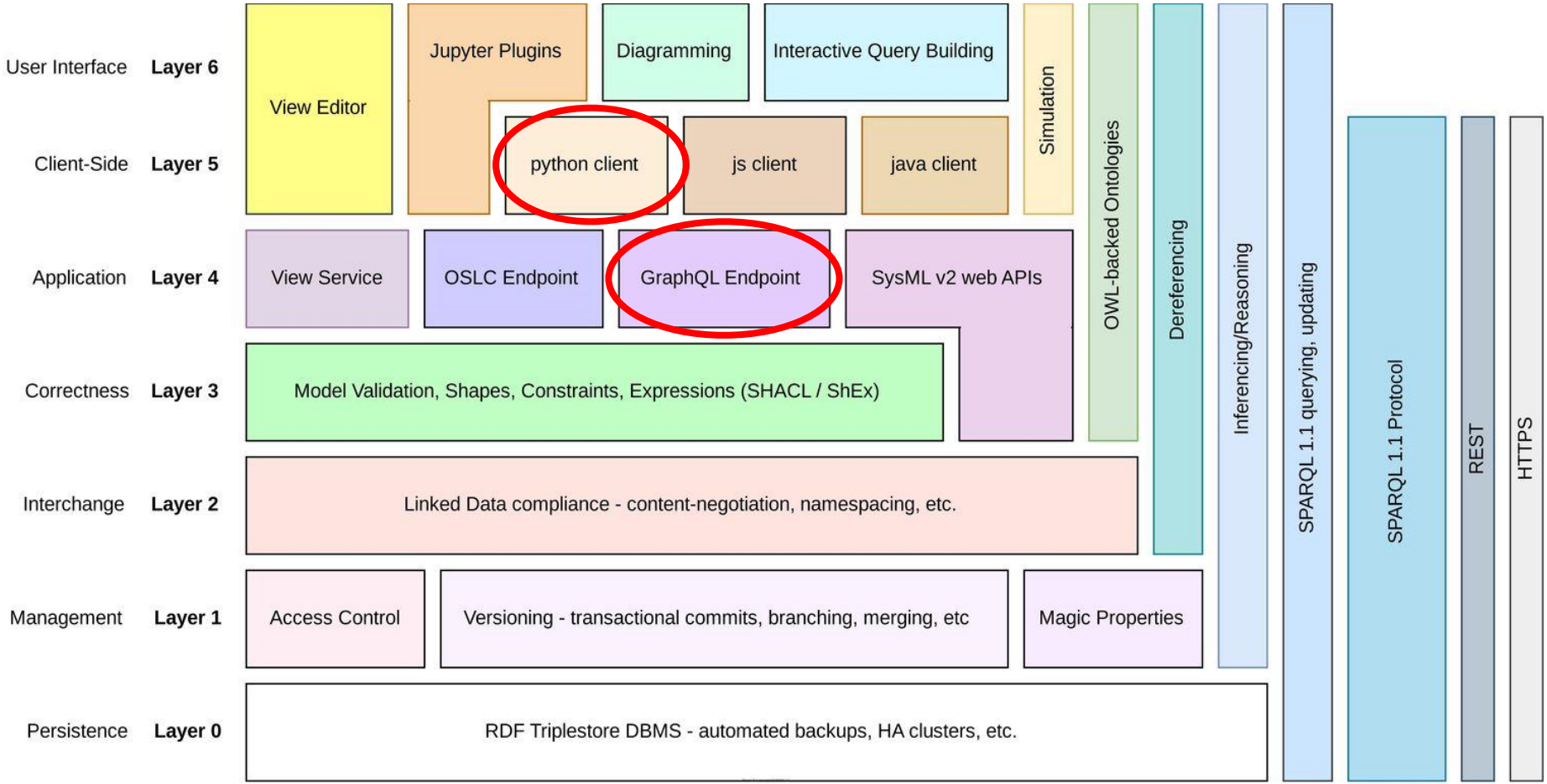
What are some desirable outcomes?

- Ability to fetch “objects” out of MMS-5
- ^^ without writing SPARQL
- ^^ without writing a specific SDK for each vendor, project, data model, etc.
- ^^ without needing to deeply learn/study the schema (i.e., discoverable)
- ^^ without sacrificing substantial query performance
- ^^ without sacrificing substantial query flexibility
- Leveraging mature open-source software
- Maintaining modularity throughout the layer cake stack
- Creating a solution capable of being reused/generalized across regimes

Generic Flexo ingestion: A precedent for success

OpenAPI Graph Extractor facilitates the semantic lifting of (optionally linked) structured data from a web service described by an OpenAPI document. In addition to capturing object structure and rich data type information (such as integer, date-time, etc.), the tool is able to create linked data from services that use Linked Description Objects as defined by the JSON Hyper-Schema

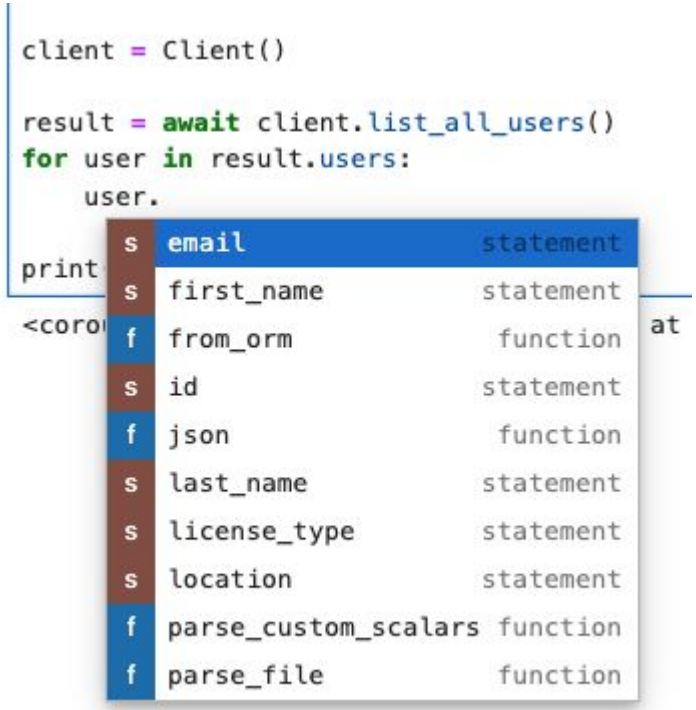
- Highly reusable & schema-agnostic. No external data dependencies
- Automatically adapts to changes in vendor’s schema
- Minimal configuration needed between redeployments


User Interface
Layer 6
Client-Side
Layer 5
Application
Layer 4
Correctness
Layer 3
Interchange
Layer 2
Management
Layer 1
Persistence
Layer 0
View Editor
Jupyter Plugins
Diagramming
Interactive Query Building
python client
js client
java client
Simulation
View Service
OSLC Endpoint
GraphQL Endpoint
SysML v2 web APIs
Model Validation, Shapes, Constraints, Expressions (SHACL / ShEx)
Linked Data compliance - content-negotiation, namespacing, etc.
Access Control
Versioning - transactional commits, branching, merging, etc
Magic Properties
RDF Triplestore DBMS - automated backups, HA clusters, etc.
OWL-backed Ontologies
Dereferencing
Inferencing/Reasoning
SPARQL 1.1 querying, updating
SPARQL 1.1 Protocol
REST
HTTPS

Working backwards

Objective: a strongly-typed python ORM-like query client for object retrieval

- Data flexibility, computability
- Discoverable schema (autocomplete)
- Intuitive, chainable operations



Generating the client!?

- Schema is already fully defined; object shapes are prescribed by vendor
- Maintains the propagation of schema from vendor to client; forwards-compatible
- Range of query operations is limited, which is ideal; data originates from a document store model
- Readonly nature (querying / object retrieval) simplifies the ORM. no need for writing back to data store

Generating the API

- Schema is already fully defined; object shapes are prescribed by vendor
- Maintains the propagation of schema from vendor to client; forwards-compatible
- Service allows generated clients to retrieve objects, handles the transformation from DSL or ORM query language to SPARQL
- Highly modular; generation is not driven by client. Compatible with multiple different clients (generated or otherwise)

Exploring the State of the Art

- **GraphQL** - **https://graphql.org/**
- **graphql-ld** - given graphql query & json-ld context, generates sparql query
  - user must know the schema and how to write graphql. no feedback on invalid queries
- **semantic object modeling (ontotext)** - allows dev to define mappings that generate graphql schema which can be used to query the triplestore
  - only works with graphdb, commercial licensing only
- **ariadne-codegen** - given a graphql schema, generates a python client with ORM-like classes and methods for querying a graphql server
  - client-sided only, no other apparent downsides
- **openapi-to-graphql** - typescript lib capable of generating graphql schemas from openapi documents. developed by IBM
  - very opinionated, but configurable thru API
- Other unmaintained projects that attempted to bridge RDF and GraphQL

Putting into practice

- Generate a GraphQL schema based on either an OpenAPI document or RDF ontology (RDFS+OWL, FOAF, DCT, Schema.org, etc.)
- Create or “fit” a service that transforms GraphQL queries to SPARQL (against that schema) such that it produces the appropriate bindings
- Generate a strongly-typed python ORM-like query client against the GraphQL schema

Generated service works as its own GraphQL endpoint, independent of client

Open-source tooling takes care of 80~90% the work above

Jama Case Study