k.LAB: a semantic web platform for science

Technical note – Version 1.0, 2021-02-20

Table of Contents

1. Architecture of the k.LAB platform
  1.1. The software stack
  1.2. The k.LAB logical layers
2. Resources
3. Semantic modeling
  3.1. An example of semantic resolution
  3.2. The worldview
    3.2.1. Authorities
  3.3. Models
  3.4. Resolution and contextualization
  3.5. Machine learning
4. Behaviors and applications
  4.1. Applications, Client software
5. Current status

Integrated modeling is a practice meant to maximize the value of scientific information by ensuring its *modularity*, *reusability*, *interoperability* and *traceability* throughout the scientific process. The k.LAB software, discussed here, is a full-stack solution for integrated modelling, supporting the production, curation, linking and deployment of scientific artifacts such as datasets, data services, modular model components and distributed computational services. The purpose of k.LAB is to ensure — by *design* rather than intention — that the pool of such artifacts constitutes a seamless *knowledge commons*, readily actionable (by humans or machines) through the full realization of the *linked data* paradigm [REF] augmented with semantics and powered by machine reasoning. This design enables automation of a wide range of modeling tasks that were previously only performable by experts and on an ad-hoc basis.

The k.LAB platform directly addresses the four FAIR goals (Findable, Accessible, Interoperable and Reusable), introducing innovations particularly in the practice of semantic annotation, which is reviewed into a modern, expressive approach meant to ease adoption by providers and users. To the four dimensions in FAIR, k.LAB adds a *reactivity* dimension, in line with the original vision of a semantic web: this dimension enables knowledge to also be *deployed* in an *"internet of observations"*, creating *live* artifacts that can interact, improve and evolve as new information appears on the network.

The central service in the k.LAB modeling API !!!! wraps the RESOLUTION ALGORITHM !!!! receives as input a logical query of the form "observe *concept* in *context*" (e.g., "observe *change in land cover type* in *Colombia, 2015-2020*", only slightly paraphrased from k.LAB's near-natural query formalism) and, in response, assembles, documents, initializes and runs a computation (called a *dataflow*) that produces the **observation** of the concept that best fits the context, based on the integration of data and model components available in the k.LAB network. The observations output by the API request, along with the dataflow assembled to generate them, are themselves scientific artifacts — automatically augmented with provenance records and user-readable documentation — that can be exported and curated as needed. @!!!! The logical queries are also used to state model dependencies so that …. !!!!

Artificial intelligence, driven by both semantics (*machine reasoning*) and the analysis of previous outcomes (*machine learning*), satisfies the request using a shared, communally owned and curated knowledge base (the *worldview*, a set of ontologies) and the resource pool available at any given moment on the k.LAB network, by ranking, selecting, adapting, and connecting data and model components made available by independent and uncoordinated providers.

This document is a brief sketch of the k.LAB main principles and architecture. Detailed documentation for k.LAB is in development and is referenced where available.

1. Architecture of the k.LAB platform

The open source k.LAB software stack includes five components that support the creation, maintenance and use of a distributed *semantic web platform* where scientific information can be stored, published, curated and connected. The software is licensed through the Affero General Public License (AGPL) v.3.0 and is available for the most part at the k.LAB git repository.

1.1. The software stack

- **Server** components are deployed by certified *partners* to publish resources and semantic content (**k.LAB Node**) and/or provide modeling services and applications (**k.LAB Engine**) to online users. Published resources can include both static data and dynamic computations, both of which may be hosted in source form at the node or linked to external data (e.g. WCS, WFS, OpenDAP) or computational services (e.g. OpenCPU). The k.Node software is deployed in containers that can be configured to host dedicated instances of Geoserver, PostgreSQL, Hyrax or other services; these are transparently managed through server adapters inside the node, virtually eliminating the need for alphabetization of node administrators.
- **Client** components are used by contributors to develop, validate and publish resources and semantic content (**k.Modeler**, an Integrated Development Environment (IDE) for semantic modeling), and by end users (**k.Explorer**) to access modeling services and specialized applications built for the platform and delivered through the web.

Additional server components serve specific needs on the k.LAB network and are of less common application in partner sites. Among them the following are noteworthy:

- The *hub* server, **k.Hub**, manages authentication and organizes node access for authenticated engines. The Integrated Modeling Partnership manages a set of nodes and a main hub, and releases site certificates that enable nodes to be connected to form the platform. Partners that need to manage users locally may also deploy and connect a hub, although this is normally only convenient in large deployments.
- A *semantic server* collects and indexes the semantic knowledge from the worldview and all public projects, constantly compiling and revising documentation and use cases to assist users in the semantic annotation process. Users can look up annotations made by others and access hyperlinked, evolving descriptions of each concept and predicate. The semantic server can be connected to the k.Modeler editor to provide inline logical validation of logical expressions in models being developed, and a suggestion service that can find and propose comparisons with use cases extracted from peer-reviewed public projects. Through the use of specialized metadata inserted in k.IM source files, the server can be integrated with the editors so that assistance is available directly, to ease the development of semantic content as much as possible. The semantic server is in development and is not available to the general public yet.

Other, less critical server components are in development and are not discussed here. Among these, a statistics server collects anonymized information from successful and unsuccessful resolutions and processes them using machine learning techniques to improve the resolution algorithm.

Usage configurations: [DISCUSS END-USER (+k.Apps) vs MODELER vs SITE ADMINISTRATOR] The k.LAB engine, a server-side component, can also be run at the client side in a local configuration, so that new content can be developed and tested in a sandboxed environment before publishing, with full access to public resources. Such client use is supported and facilitated by a small, downloadable control center application that removes the complexities linked to installing, upgrading, starting and stopping the engine or the IDE. The k.LAB distributed paradigm supports and enforces a model where information remains under the ownership of its authoritative sources while maximizing its availability and interoperability, compatibly with both public and commercial services thanks to careful attribution and to state-of-the-art encryption, access control and security.

1.2. The k.LAB logical layers

The set of active, connected nodes and engines at any given time forms what can be seen collectively as a distributed container, where scientific knowledge is found in **three layers** handling information at increasing levels of abstraction: the *resources*, *semantic* and *reactivity* layers. The first can be seen as a data curation platform based on modern linked data concepts, optimized for generality of the data model, semantic annotation and deployment. Semantic and reactive content for the platform is developed in the respective layers using two specialized languages, *k.IM* for semantic resources and *k.Actors* for rective behaviors and applications. The modeler IDE (*k.Modeler*) provides drag-and-drop interfaces to build resources and specialized editors for k.LAB projects, containing both k.IM and k.Actors resources.

- The **resources** layer provides a *protocol* for conventional data and computational resources or services to interoperate at the data level, matching identifiers, data types and metadata through a uniform API. Nodes and client applications include interfaces to manage development and submission of knowledge to the resources layer, to be published and curated either on-site or through hosting providers with full control of licensing and access.
- The **semantic** layer provides a *language* that enables annotating resources through semantically explicit logical expressions, ensuring findability, interoperability and accessibility through purely logical queries, validating consistency and producing mediation strategies through machine reasoning and logical inference. The semantic layer uses the **k.IM** language to specify semantic knowledge (compatible with W3-endorsed OWL 2) and models; these specifications, collected into namespaces and projects, can be deployed to k.LAB Nodes for the k.LAB inference engines to find, rank and use.

- The **reactivity** layer provides *behaviors* for the scientific artifacts produced by running queries in the semantic layer TURNING THEM INTO AGENTS. Reactive observations exist in k.LAB Engines and can react to each other either locally (within the same engine) or remotely. The reactivity layer enables distributed agent-based simulations and computations that automatically adapt to changing conditions or states. The **k.Actors** language is used to define behaviors for the reactivity layer. As a special case, behaviors bound to users and sessions can be used to quickly develop specialized interactive applications that run on the platform through web browsers.

The full separation of concerns and APIs in the three layers maximizes their value: for example, the resources layer can be seen through different worldviews, therefore serving different purposes in different networks by reinterpreting it through the logical "lens" of a differently configured semantic layer.

---

2. Resources

Resources have URNs (data, computations), geometry, and data type. One or more inputs/outputs and attributes. Simple API encompasses all conventional computations. Main service: resolve URN in context → return non-semantic artifact. If resource is computed, submitting input values may be necessary and the geometry specifies the form.

Adapters (for data, data services, computations, modeling services from WPS to large models). Current adapters for x…. I/O: modeler, node web UI Lifecycle and rolling peer review. URN or DOIs (eventually). In special cases handled by the k.IM language, also literals or function calls. Clients. Permissions and access.

Besides data: resources are also "internally" implemented algorithms (from the core software or plug-in extensions) called as functions; literal values; or special URNs handled directly w/o referring to external or imported data (e.g. random). Uniformity with "function calls" makes it possible to insulate all the semantics within the semantic layer. The k.IM language also allows defining "non-semantic models" which are resources. No semantics at this level: what comes next

---

3. Semantic modeling

Semantic modeling enables the *semantic annotation* of non-semantic resources provide a shared *worldview* (a logically organized knowledge base of concepts and relationships) and allow its linking to resource URNs by way of *models*, i.e. semantic annotations that specify the meaning associated with resources and, when applicable, their inputs, outputs and attributes. The pool of models constitutes the semantic layer, which is mined by the resolution algorithm to resolve a logical query to a computed scientific artifact.

Relies on the k.IM language, is the linguistic framework: separation of attributes, traits etc using composition rules. While the underlying knowledge model is OWL 2.0, k.IM simplifies use and acceptance due to clarity coming from separation of concerns and traceability of meaning. Examples.

In a departure from every other ontology platform, k.LAB does not rely on individual concepts for the description of observables, but on logical expressions that combine predicates, operators and nouns in a fashion reminding the grammar structure of the English language. This is key not only to the usability and parsimony of the underlying knowledge base, but also to the functioning of the machine reasoning underlying the resolution algorithm, which can reason independently on different logical dimensions of a query and infer computations that would require complex (and largely impossible) logical breakdown to function. Each logical dimension of a query can be *resolved* to a model which handles its specific target; the computations specified by the model are automatically assembled into the *dataflow* (algorithm) that produces the corresponding scientific artifact. The next example breaks down the process for a complex query.

As seen in the example below, is typical of k.LAB models to be very short, simple and readable. Every model, with few exceptions, is written for *one* concept, with all inputs stated merely as semantics; each model, by design, can be run and tested independently as a self-contained module. When the model logics require that certain dependencies are satisfied in a specific way, scoping rules can be used to ensure that specific models (or models for a specified subset) are picked to satisfy the desired dependencies. It is also possible to use (libraries of) *non-semantic models* to refer to specific computations whose semantics is highly technical and not worth exposing (E.G. INDEX OF …), ensuring linkage with conventionally used quantities without sacrificing modularity. In most situations, models can be written independent of the spatial and temporal context in which they will be run, and often even in ways that are compatible with different interpretations of space and time. Negotiation of inputs, outputs, data format, units/currencies, visualization and contextual validation are by default left to the k.LAB runtime. Writing models this way enforces discipline and maximizes clarity, readability and parsimony: contributors only write the core of the algorithm that leads to one specific observation, leaving every other aspect (including the selection and computation of any inputs) to the resolver and the k.LAB runtime.

3.1. An example of semantic resolution

So, a hypothetical query like [EXAMPLE WITH CLASSIFICATION biomass of eucalyptus tree] operated for example in a country context would be hypothetically addressed by the following strategy and models: …

All these models compete and are independently developed and published. The resolution algorithm can use both objective and success-driven heuristics so it can improve with time. Scoping rules allow controlling the selection at each branching point and full transparency is given both of the actual provenance and of the rationale of the choices made, including a log of the alternative paths that were discarded when they were close.

3.2. The worldview

Collaborative and shared. Meant for *observation* of reality, not reality itself: therefore, aware of scale and differentiating not only by phenomenological nature but also by the nature of the observation process applicable to them. For example, events vs processes. Scale of observation (range thereof) is key to semantics and to compatibility of worldviews. More than 1 possible but we're working on one, scaled around human observation (wouldn't fit large or small, such as field of application of relativity or quantum physics. Must be shared, can't be owned.

3.2.1. Authorities

Vocabularies and their usage.

3.3. Models

Subjects and qualities Processes and Change Relationships and configurations Attributes/identities: classification, identification Roles

3.4. Resolution and contextualization

Main process (figure!) Choice of models: the resolver (distributed) Scale handling in space and time Automatic recontextualization (watershed example)

3.5. Machine learning

Model vs. Learn - produces a computable resource (dataflow) that can be stored with a URN, independent of semantics. This includes "calibration" and "validation". Standard machine learning (show example). Calibration or other model inference. Model for many applications

---

4. Behaviors and applications

4.1. Applications, Client software

5. Current status

Version 1.0