

[] | *resources_handling/imgs/KLAB_LOGO.png*

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
1.1. The software stack	2
1.2. The k.LAB logical layers	3
2. Shared resources	4
3. Semantic modeling.....	4
3.1. Semantic mediation and inference in support of modeling.....	5
3.2. The worldview.....	7
3.2.1. Authorities	7
3.3. Models	8
3.4. Learning models	8
3.5. Views, documentation, provenance.....	8
4. Behaviors and applications	8
4.1. Applications, Client software	8
5. Current status	8

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 PRODUCTION API USAGE vs PROVIDER, NODE/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 reactive behaviors and applications. The modeler IDE (*k.Modeler*) provides drag-and-drop interfaces to build resources and a specialized editing and debugging environment for k.LAB projects, supporting both k.IM and k.Actors development.

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

GENERAL ARCHITECTURE: non-semantic resources (data, model components) annotated with semantics to produce *models* which can be *contextualized* in a space and time *context* to produce *observations* that describe those concepts in the context. These can be potentially provided with *behaviors*. Subito: EXAMPLE OF USE (model x as). Source (model urn as). Internals first: literals (model 1 as), processing (model X as using).

2. Shared resources

Shared resources available on the network 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. For example, the k.IM statement `im:Net value of ecology:Pollination` contains a predicate (im:Net), an operator `value of` which affects the meaning of the process `ecology:Pollination` and transforms it into its quantifiable value. This semantic articulation is key not only to the usability and parsimony of the underlying knowledge base, which can remain small and learnable because of the reuse of terms, 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 otherwise require complex logical breakdown to function. Lacking specific models for a complex observable, each logical dimension of it may be *resolved* to one or more models which handle a specific component, and the set of models is ranked for best fit to the context before selecting the most appropriate. The computations specified by the model are automatically assembled into a *dataflow* (algorithm) that produces the desired scientific artifact.

It is typical of k.LAB models to be very short, simple and readable. Every model, with few

exceptions, is written for *one* concept, with any required inputs stated merely as semantics; each model, by design, can be run and tested independently as a self-contained module. For example, the model below

```
model occurrence of earth:Region with im:Still earth:PrecipitationVolume
  observing
    earth:Upstream im:Area in m^2 named contributing_area,
    geography:Slope in degree_angle named slope
  set to [
    def sloperadians = Math.tan((slope*1.570796) / 90)
    return Math.log((contributing_area+1) / Math.tan((sloperadians+0.001)));
  ];
```

requires observations of geographical slope and upstream drainage area to compute its output, a commonly used hydrological quantity (topographic wetness index) interpreted here through the semantics of "occurrence of region with retained precipitation". None of the complex calculations required to compute the inputs is required in the model, as their semantics (*earth:Upstream im:Area* and *geography:Slope*) is resolved at run time to the most appropriate model for the context. The latter can consist of a single point in space or of as a gridded or polygon-based spatial coverage, without any modification to the model. 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 chosen to satisfy the desired dependencies. It is also possible to use (libraries of) *non-semantic models* to refer to specific computations whose semantics is deemed not worth exposing, ensuring linkage with conventionally used metrics without sacrificing modularity or requiring overly difficult semantic characterization.

In many situations, models can be written independent of the specific 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. Semantic mediation and inference in support of modeling

In simple cases, the query "observe *concept* in *context*" is answered by locating a model annotating a data source with the specified *concept*. For example, setting the context to a geographical region (e.g. a country's extent with a spatial grid model at 100m resolution and temporal context, e.g. the year 2010) and querying an observable such as *geography:Elevation in m* may retrieve, among others, the following annotation written as such:

```
model im.data:geography:morphology:dem90 as geography:Elevation in m;
```

which annotates a network-available resource specified by the URN

`im.data:geography:morphology:dem90` as an observation of the `geography:Elevation` concept. The URN gives access to metadata including extent, resolution and temporal coverage, so that the model, whose semantics is identical to the query's, can be assessed for match to the context. If the model is deemed to be the best match for the context, the k.LAB engine will translate it into a set of processing steps (in this case simply a resource retrieval operation) and pass the resulting *dataflow* to the runtime to compute and produce the resulting *observation*, in this case a raster map of elevation, with 100m resolution, reflecting the boundaries and time of the context (the dataflow will include any necessary reprojection, resampling, or unit transformation to match the query and the context). Other models may compete for the choice, made on the basis of criteria such as resolution and extent match, specificity, semantic match, and including criteria such as peer review results or usage feedback for the original data. A partially covered context may use other resources to be completed, if the ranking is close enough.

Besides such simple and direct matches, machine reasoning backed by an observation-centered (as opposed to reality-centered) ontological framework can enable more sophisticated observation tasks that do not correspond to readily available annotations and are normally only possible through specialized manual work. For example, attributes such as `im:Normalized` may be prepended to another observable to affect the result, which would be resolved to an independent model (`model im:Normalized using <normalization algorithm>`), possibly restricted to certain classes of observables (`model im:Normalized of im:Quantity ...` to restrict its application to numerically quantifiable observables). More interestingly, resolution strategies may cross inherency barriers to infer the best observation strategy when a direct match is not available. For example, a hypothetical query like `(ecology:AboveGround ecology:Biomass) of biology:Eucalyptus biology:Tree` operated in the same country context would refer, by virtue of the inherency operator `of`, to a quality (above-ground biomass) inherent to a particular subset (Eucalyptus) of a secondary object (Tree) located in the primary context of the query (a geographical region). It would be resolved by the following strategy:

1. Locate a model for the original observable, `(ecology:AboveGround ecology:Biomass) of biology:Eucalyptus biology:Tree`. If found, resolve using it. Otherwise
2. Locate a model of the inherent subject, `biology:Eucalyptus biology:Tree`; if found, resolve each eucalyptus tree in the region, and for each of them locate a model of `(ecology:AboveGround ecology:Biomass)`; if found, redistribute the result over the set of trees in the region. If eucalyptus tree model cannot be resolved
3. Locate a model capable of instantiating every `biology:Tree` in the region; if found, locate a classifier model capable of attributing the abstract identity (`biology:Species`) of which `biology:Eucalyptus` is a subclass, and apply to classify the trees, only keeping the eucalyptus. If successful, locate a model of `(ecology:AboveGround ecology:Biomass)` to complete the observation.

Similar reasoning strategies can be applied to a large set of situations, using semantic inference driven by the phenomenological understanding of the observation process. For example, a query for `presence of biology:Tree` could be satisfied, when not directly resolvable, by an observation of model of `(ecology:AboveGround ecology:Biomass) of biology:Tree` because biomass (a `im:Mass` in a higher-level ontology) is an extensive property whose non-zero value implies the existence of its inherent subject, so the presence can be computed as a true/false value attributed to each area where the biomass of any tree is nonzero. In another commonly encountered use case, qualities that can only be correctly computed in specifically delineated contexts (for example hydrological

qualities, such as "upstream area", which require computation in a correctly delineated river basin) can be automatically computed in arbitrary contexts by first looking up a model to delineate all the relevant contexts (river basins) intersecting the areas, then applying the necessary models to compute the qualities inherently to those, then re-distributing the values over the desired context. The same considerations hold for more complex observables such as processes which have the ability of affecting the value of qualities through time and to generate events or other objects, which in turn can be the context for other qualities or processes. The ability of automatically negotiating mediations based on inherency and phenomenological reasoning multiplies the capability of connecting diverse models without error, offering integration possibilities that stand orders of magnitude above those allowed by semantic matching alone. Such tasks require specific planning and significant technical expertise and time to perform in conventional ways.

3.2. The worldview

Both annotation and inference, as described above, require a set of *ontologies* that define the realm of knowledge that can be integrated and conform with the foundational principles of k.LAB's observational model. We refer to this set of ontologies as the *worldview*, a set of k.IM projects that are automatically synchronized to all users that adopt it. Worldviews are linked to user profiles and may be as many as needed; the development of a worldview, however, is a large collaborative endeavor and to date, there is one worldview (the *im* worldview, for Integrated Modeling) that is under development within the k.LAB team and an extended group of collaborators. As a worldview is meant to describe *observation* of reality, not reality itself, it is naturally aware of *scale* and its semantics differentiates observables 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

Identifying identities such as taxonomic or chemical species presents a challenge as their number is virtually infinite: as a result, ontologies often provide *some* of the ones most likely needed by the communities of reference, but it is impossible to address all use cases and even importing specialized ontologies (such as CHEBI for chemical identities) risks overwhelming the reasoner with too many (and still often not enough) concepts, or creating unnecessary usage conflict with the same concepts from other ontologies. In k.LAB, this problem is obviated through the introduction of *authorities*, a mechanism to interface with external vocabularies that enjoy broad community acceptance, fully integrated in the k.IM language. Such vocabularies are seen by contributors and users as externalized namespaces. An authoritative identity takes the form *IUPAC:water*, easily distinguished from other concepts by its uppercase namespace tag (a regular concept would have a lowercase namespace, e.g. *geography:Slope*). Its use in k.IM triggers validation of the concept ID (*water*) using an online service tied to the authority (*IUPAC*) which is advertised by nodes in the k.LAB network. Upon successful validation, an identity concept is produced for the statement whose definition is identical and stable at all points of use. This mechanism allows externalizing large vocabularies (such as the IUPAC catalog of chemical species or the GBIF taxonomy identifiers) and structured specification conventions (such as the World Reference Base for soil types) that are validated and turned into stable, k.LAB-aligned semantics at the moment of their use. Another

advantage of many authorities is flexibility of usage: for example, **IUPAC:water** and **IUPAC:H2O** are valid identifiers that can be used in k.IM observables as written [EXAMPLE] and translate to the same concept (the chemical identity corresponding to water, encoded internally as the standard InChI key) using a IUPAC-endorsed catalog service. The k.LAB stack provides content contributors with assisted search interface and intelligent editor support with inline, "as-you-type" validation and documentation.

3.3. Models

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

3.4. Learning models

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

3.5. Views, documentation, provenance

4. Behaviors and applications

4.1. Applications, Client software

5. Current status