SciMesh: RDF graphs for scientific knowledge

Release 1.1.0

Torsten Bronger Michael Flemming
Hartmut Schlenz Michael Selzer
Manideep Jayavarapu

Apr 04, 2023

Contents:

1 Status of this document **2**
1.1 Todo list . . . 3

2 SciMesh at a glance **3**
2.1 Introduction . . . 3
2.2 Current scope of SciMesh . . . 5
2.3 Data model overview . . . 5

3 SciMesh vocabulary **6**
3.1 External vocabulary used in SciMesh . . . 7

4 Detailed description of the data model **7**
4.1 The core . . . 7
4.2 Concurrency . . . 8
4.3 Unknown chronological order . . . 11

5 Prototype: JuliaBase **11**
5.1 Give it a try! . . . 14

6 Note about content addressing **14**

7 Data provenance **15**
7.1 Bulk data . . . 15
7.2 Data input . . . 15
7.3 Data output . . . 16

8 Implementation **16**
8.1 Workflow . . . 16

8.2 Getting the graph . . . 18
8.3 Considerations about visualisation . . . 19
8.4 Good URIs . . . 19
8.5 Authentication . . . 20

Date 2023-04-04

Abstract SciMesh is a set of specifications that define the representation of scientific results in form of a knowledge graph. While many such data formats focus on simulation data, SciMesh explicitly includes physical specimens as first-class citizens. Thus, provenance of both data and specimens can be documented. SciMesh's immediate purpose is to represent content of electronic lab notebooks (ELNs) for sharing scientific results across ELN instances. This way, collaboration partners running different ELNs (even different ELN software) can have a uniform view on their collective results without media disruptions.

Authors

- Torsten Bronger[1], t.bronger@fz-juelich.de
- Michael Flemming[1], m.flemming@fz-juelich.de
- Hartmut Schlenz[2], h.schlenz@fz-juelich.de
- Michael Selzer[3], michael.selzer@kit.edu
- Manideep Jayavarapu[3], manideep.jayavarapu@kit.edu

Note: The authors would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) – 442146713.

1 Status of this document

This is a work in progress, which, once finished, is supposed to be a gentle introduction into SciMesh as well as a reference for it.

It is also a "request for comments". Thus, if you want to contribute ideas or content, or if you found a mistake, don't hesitate to get in touch.

[1] Forschungszentrum Jülich, ZB, Jülich, Germany
[2] Forschungszentrum Jülich, IEK-1, Jülich, Germany
[3] Karlsruhe Institute of Technology KIT, Karlsruhe, Germany

1.1 Todo list

Without significant ordering:

- Add further triplets to the RDF output, even it is makes it highly redundant. For example, inverse relations like "is_specified_input_of" are interesting. There are probably tons of vocabulary to investigate, especially in the OBO ecosystem.
- Create proper entities for people (operators, "currently responsible person" of a sample). Currently, only a string literal containing the login name is emitted.
- Highly JuliaBase-specific: Create proper entities for sample topics.
- The current scheme has no persistent way to identify a sample. (This may turn out to be an implementation detail as JuliaBase does have a persistent identifier for each sample.)
- Find a way to deal with sample renames.
- Currently, most predicates are home-brew (i.e., they use our URI). They should be replaced with existing, widely adopted third-party URIs wherever possible.
- We should assure that the kind of properties is clear, either by using an established predicate or by other means. Eventually, a re-using party must be able to derive that e.g. a temperature is a temperature.
- Connect process classes (in other words, experimental methods) with third-party entities representing the underlying method(s).

2 SciMesh at a glance

2.1 Introduction

SciMesh represents scientific insights as a knowledge graph. In its current realisation, it focuses on sample-based workflows, in which samples (physical ones or data artefacts) undergo a sequence of processing and measurement steps. However, it is not limited to that. Most generally, it represents scientific insight by declaring a relationship between cause and effect. In other words, *if* certain prerequisites are set, *then* certain observations will be made.

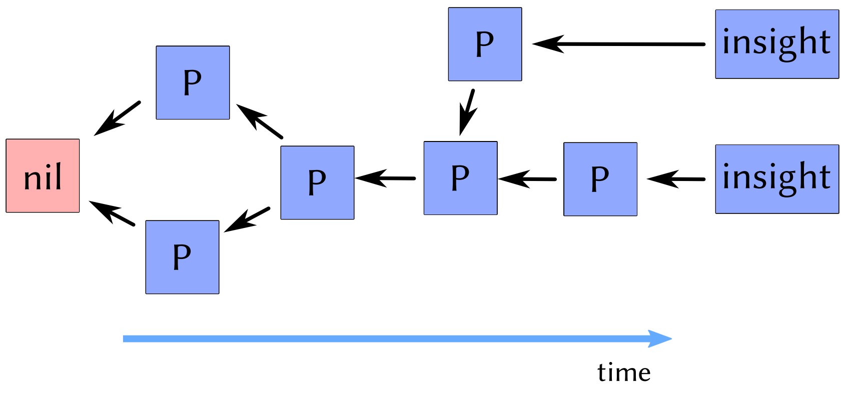


Fig. 2.1: Example graph of processes.

Fig. 2.1 shows a graph illustrating this line of thought. Starting from an initial state called "nil", processes change this state. There is a monotonic increase in time from left to right. Consequently, this time axis also is inherent in a chain of processes. A process may be "take glass substrate out of rack", "heat sample", "mix two substances together", or "wait for solar eclipse". It is really that general. Every process has got one of more causes. If it is a single one, it may be the initial state. The initial state is totally void. It bears no information at all, which is why the chain of processes must define the states in-between as completely as necessary to be useful for scientific conclusions. Conversely, one process may be the cause of multiple others. This way, chains can be branched off, possibly by different scientists years after the work on the main trunk.

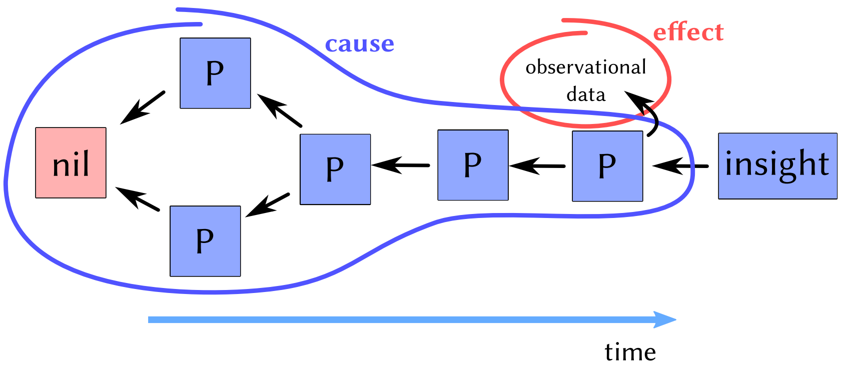


Fig. 2.2: Example graph of processes, showing cause and effect.

Fig. 2.2 shows how the essence of scientific work, the relation of effects to causes, is represented in the graph: The processes up to a certain point are the cause, and the observations at that point are the effect. Therefore, it is valid to call that point an "insight", which is a graph node of its own. Two things are important here. First, the process to which the observational data is attached is a measurement, and quite frequently, a measurement does not change state. Still, it is a process proper, with all bells and whistles. We think that it is not necessary to distinguish between measurements and processes which actually change something. Besides, a measurement does change state, albeit the change may not be significant. And secondly, a certain graph of processes might have many insights pointing to it. In particular, the processes following a measurement may lead to a second measurement with new results, meaning a new insight.

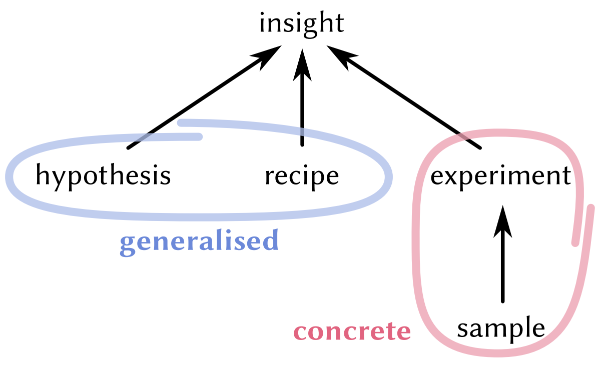


Fig. 2.3: Is-a relations of insight-like entity classes.

Speaking of insights, Fig. 2.3 shows the relationships of things that may point to a certain state (a.k.a. process) in a graph. All of them are *insights* but if it is about a chain of concrete processes at certain points in time, this should be labelled as an *experiment*. If all of this happened to the same sample, the experiment may be identified with this *sample*.

If the processes are not concrete but generic blueprints of processes, the experiment is actually a *recipe* for experiments. Or, by bringing cause and effect together, it is a scientific *hypothesis*.

To illustrate experiment versus hypothesis, consider the following two examples, respectively:

1. On 21 October 2019, John threw a ball from the Eiffel Tower, and it moved downwards.
2. A body is released in a gravitational field, and it moves according to the force field.

2.2 Current scope of SciMesh

The currently specified RDF data model of SciMesh is narrower than the very general concept outlined above. The reason is very simple: Our work is in its early stages, and we have to focus on specific domains of research in order to avoid frittering away our resources. Since we work in task area "Caden" of NFDI4Ing, the domain of research of choice are sample-based workflows.

We hope to be able to give guidelines for how to extend SciMesh to other domains, eventually.

2.3 Data model overview

The RDF data model is build around two concepts: *Sample* and *process*. Both are meant in their broadest senses. A sample may represent a state of a physical specimen as well as a data set. A process may create a new sample state (i.e., change the sample, for example an etching process), or create new data (e.g. a measurement on a sample), or both.

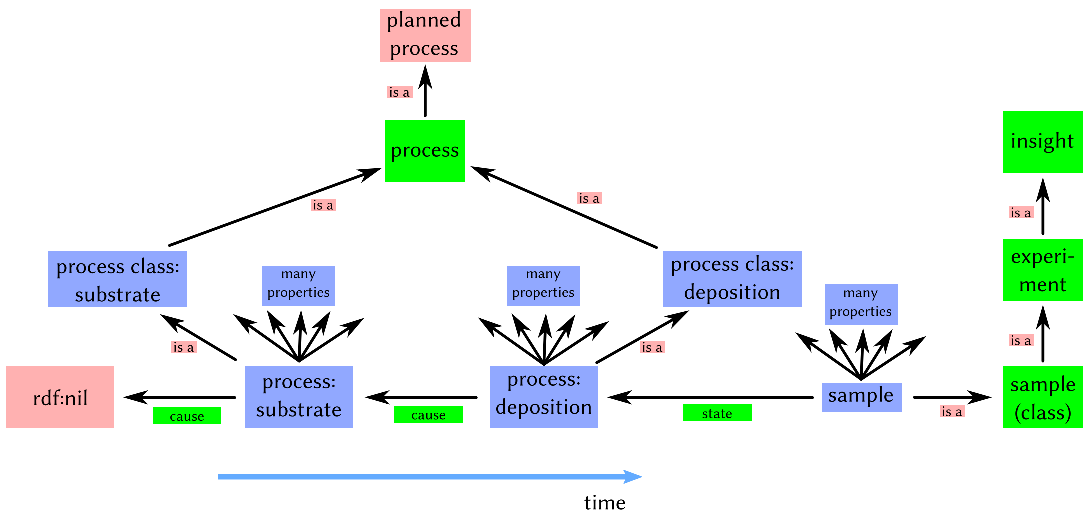


Fig. 2.4: Simplified example topology of a SciMesh knowledge graph. Colours denote namespaces: Blue is the ELN namespace, green is the SciMesh namespace, and red are external namespaces (RDF, OWL, OBO etc).

Fig. 2.4 gives an overview of the anatomy of a knowledge graph in SciMesh. It is a very simple graph yet contains most of the basic concepts. At its heart, there is a sequence of processes (at the bottom) that work on the sample. The first process, “substrate”, creates the sample (the sample starts its life as a bare substrate). Its RDF properties determine the basic physical properties of the sample (e.g. material and size). Then, further layers of material are deposited on the substrate in the deposition process. Common properties of most processes are name, method, timestamp, operator, and comments.

Note that three different namespace domains are involved:

1. The domain of the ELN instance: the processes, the process types, the sample and its intermediate instances. This is in blue.
2. The domain of the ELN software: the sample type. This is in green.
3. External domains like BFO/OBO and RDF. This is in red.

Note: Some may have noticed a striking resemblance to Git’s data model. Indeed, there is a very close mapping possible between both. The processes are commits, and the sample is a branch. You may split a sample in half, creating a new branch. At the same time, one process may have more than one cause. This way, pouring two chemical substances corresponds to merging in Git.

---

3 SciMesh vocabulary

Default namespace: `http://scimesh.org/SciMesh/`

Process (class) A process in the sense of Fig. 2.1 and Fig. 2.2.

Insight (class) An insight in the sense of Fig. 2.1 and Fig. 2.2.

Experiment (class) A specialised insight (see Fig. 2.3) which represents a concrete experiment. It points to a graph of equally concrete processes, i.e. processes that happened at a certain place at point in time, with certain ingredients and result state and/or observational results. Normally, all such processes bear a timestamp.

Hypothesis (class) A specialised insight (see Fig. 2.3) which represents a general hypothesis, in contrast to a concrete experiment. It points to a graph of non-concrete processes which represent a sequence of generic state changes and observations.

Recipe (class) A specialised insight (see Fig. 2.3) which represents a general recipe, in contrast to a concrete experiment. It points to a graph of non-concrete processes which represent a sequence of generic state changes.

Sample (class) A sample (see also Fig. 2.3), which may be a physical specimen as well as a simulation result artefact. It is represented by the process it points to, and all of its ancestors. This process chain may also contain process *after* the sample. Those are not part of the sample.

Concurrent (class) This is a concurrent process, a subclass of Process. This means that in absence of schemas, you have to tag an entity as a Concurrent by tagging it as both Concurrent and Process. A concurrent process does not represent a well-defined state.

Therefore, you should not refer to its URI from elsewhere for communicating something that others can or should reproduce. It can only be used as a container for further process data for another process that points to it with a “cause” relation. See section “*Concurrency*” for further information.

cause (property) This contains a cause of the subject in the object. Both subject and object are of type “Process”.

The object may also be `rdf:nil`. In this case, it must be the only cause apart from concurrents, and it documents that no causes of this process are known and will ever be known.

state (property) This connects an insight to a process of its process chain, in particular, the latest process. If the insight is a sample, the latest process represents the current state of the sample.

However, there may be more than one state of an insight. Multiple sates help coping with gaps in the process graph (e.g. because external servers are down).

operator (property) The operator of a process. The object may be a string with the person’s name or email address, or the URI of the operator.

timestamp (property) The point in time when the process was done. The object of this triplet is a blank node which must have at least the property `time:inXSDDateTimeStamp`. The `time` namespace stems from https://www.w3.org/TR/owl-time/. Concretely, in Turtle, it may look like this:

```
<http://inm.example.com/5-chamber_depositions/14S-005> a⤶
↪sm:Process,
    …
    sm:timestamp [ time:inXSDDateTimeStamp "2014-10-
↪02T14:10:00+00:00"^^xsd:dateTime ] .
```

3.1 External vocabulary used in SciMesh

http://www.w3.org/2000/01/rdf-schema#label (property) The name of a process or sample instance. It should be human-readable.

4 Detailed description of the data model

4.1 The core

The basic component of SciMesh is the *Process*. It describes some action that leads to a new state of a certain specimen. This can be different or not to the state before. For example, if the specimen hasn’t been actually changed, it is probably a non-invasive measurement process with data output. In either case, each process has got a distinct URI. This URI also represents the state that that process generates.

Processes are connected to all of their parameters, data outputs, measurement devices, methods, operators, timestamps etc. The process must be the subject of such triples.[1]

Moreover, processes are chained. Every "Process" points with "cause" relationships to their immediate cause processes, which the current process is an effect of. This reflects the cause–effect relationships of nature.

At some point in the past, no further causes are known. There are some, of course, because only the creation of the universe has no cause. (Probably.) But the are no *known* causes, and if it is sure that they will never be known, one can make this explicit in SciMesh by adding a "cause" relationship to rdf:nil.

At the other end of the chain, i.e. in the present, there may be a sample pointing with a "state" relationship to the current state of the sample, which is the latest process that happened to the sample. This way, the current state as well as its complete provenance is documented in detail.

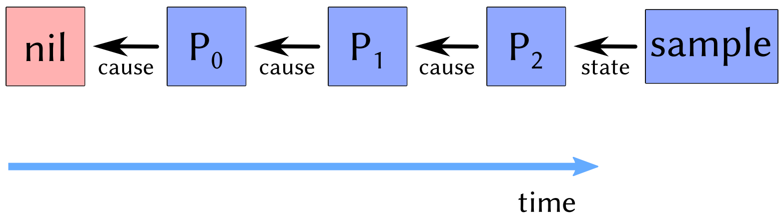


Fig. 4.1: Basic process chain with sample.

This data model is simple. Too simple, therefore, we introduce an additional concept: Concurrent processes.

4.2 Concurrency

So far, we've had *sequential* processes. Each process has a timespan which does not overlap with any other process. While this reflects the course of time perfectly, it is flawed for mere practical reasons: Sometimes, we need to document actions which work on a specimen at the same time in different processes. For instance, if some processes take place in the same vacuum, the vacuum should be its own process running parallelly to the things happening in it. Then, vacuum parameters can be connected with that process and need not be replicated.

This is expressed with a "cause" relation to a subclass of Process: "Concurrent". This is a process the running time of which overlaps with all of its effect processes. In the following, P denotes a Process proper (i.e., *not* a Concurrent), C a Concurrent, and X a Process or Concurrent. Further, $T(X)$ is the timespan of X, i.e. the set of points in time at which X has effect. (s, p, o) is an RDF triple consisting of subject, predicate, object. Then:

$$\begin{aligned} (X, \text{cause}, C) &\Rightarrow T(X) \cap T(C) \neq \varnothing, \qquad &\text{(concurrent)} \\ (X, \text{cause}, P) &\Rightarrow T(X) \cap T(P) = \varnothing. \qquad &\text{(sequential)} \end{aligned} \tag{4.1}$$

[1] The underlying reason is that SciMesh-Relations always point to the past. This makes it easier to detect whether additions to a graph manipulate causality in retrospect.

If the cause is a Concurrent, SciMesh makes no stronger assertion! You may consider C a super- or a subprocess of X, and C may start before, with, or during X. It may even be that only a subset of $T(C)$ is known, as long as the condition (4.1) holds.

A Concurrent does not represent a well-defined state. Thus, do not put a reference to a Concurrent anywhere if you want people to reproduce it. They couldn't. Instead, cite the URI of the closest following (in time) Process proper.

The reason why a Concurrent is not a well-defined state is because it holds only partial aspects of the subsequent Process(es) proper. While its influence on those Processes is well-defined and leads to states, the Concurrent alone does not. For instance, a vacuum is a well-defined ambient condition for many kinds of processing, and may be one ingredient for a concrete, referenceable sample. However, just the vacuum itself does not end in a referenceable result.

Application of concurrency: Subprocesses

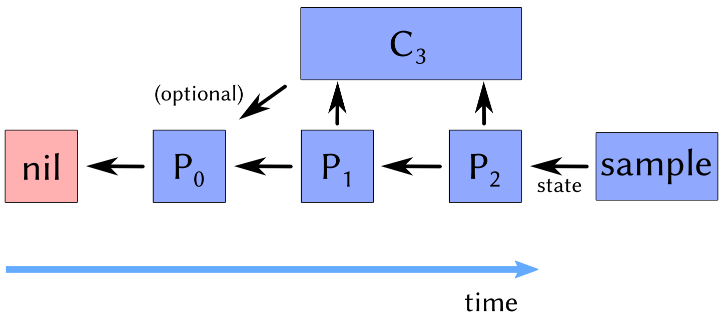


Fig. 4.2: Basic concurrency example.

Fig. 4.2 contains a process C_3 which runs at least while P_1 and P_2 are running. Moreover, because there is a "cause" relationship between $P_{1,2}$ and C_3, C_3 is also acting on the sample.

Note that the "cause" relationship from C_3 to P_0 only makes sense if C_3 is significantly (in the sense of this research) influenced by the sample. For example, if C_3 contains sensor data of sample properties, the relation is necessary. Otherwise, leave it out. You even must not add this relation if C_3 contains multiple samples (see compound process below) because then you would intertwine the history of all the samples.

Also note that the Concurrent has two processes pointing to it, which a visualising agent may interpret as a relationship between a super-process (C_3) and subprocesses ($P_{1,2}$).

Application of concurrency: Compound processes with multiple samples

SciMesh's explicit states for each sample make an extra step necessary for compound processes. With "compound process" we mean a process that has been applied to more than one sample. Since every sample has its very own provenance, this means that the compound process needed to be duplicated multiple times. This would be wasteful and should be avoided. Instead, place a so-called sample-specific process in the process graph which only contains information that is really special to the sample, e. g. its position in the apparatus. Then, give all these sample-specific processes a common Concurrent in a “cause” relation, with all the details of the compound process.

Properties of the sample-specific process like the above mentioned “position in the apparatus” must be in this sample-specific process even if in a particular run only one sample was processed. It might be tempting to merge the sample-specific process into the concurrent and don’t create a concurrent in the first place. However, this is only possible of (1) only one sample was processed and (2) none of the fields of the sample-specific process are needed. Otherwise, queries on the graph would have to be different in both cases.

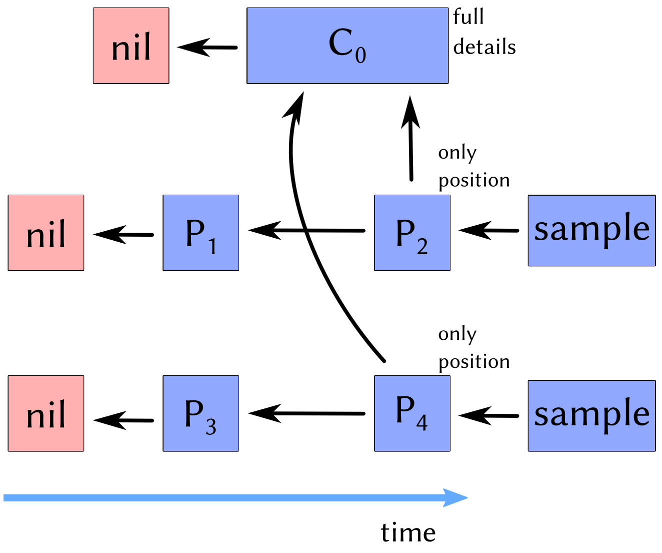


Fig. 4.3: Representation of a compound process: Each sample has only sample-specific process data in its direct graph ($P_{2,4}$, here only the position in the apparatus, e. g. a fridge). These sample-specific processes have a common predecessor C_0 which contains all process details (temperature, heating time etc).

Fig. 4.3 gives an example. The “only position” processes contain only the position of the sample (e. g. in the fridge). At the same time, the compound process C_0 contains fridge brand, temperature, whether the fan was active, etc. Cooling duration may be attached to the compound process or the sample processes $P_{2,4}$, whatever makes more sense.

Warning: Do not make the compound process the effect of older sample processes as it is done by the “cause (optional)” relation in Fig. 4.2! This would have to be done with all samples in the fridge, and this would mix together the provenance of all samples. It is highly unlikely that this is what you want. Besides, it is wrong conceptually: While indeed all sample are together in the fridge and will have influence on each other in an extremely slight way, they do not affect each other from the point of view of the respective research. And SciMesh maps the latter.

Application of concurrency: Sample splits

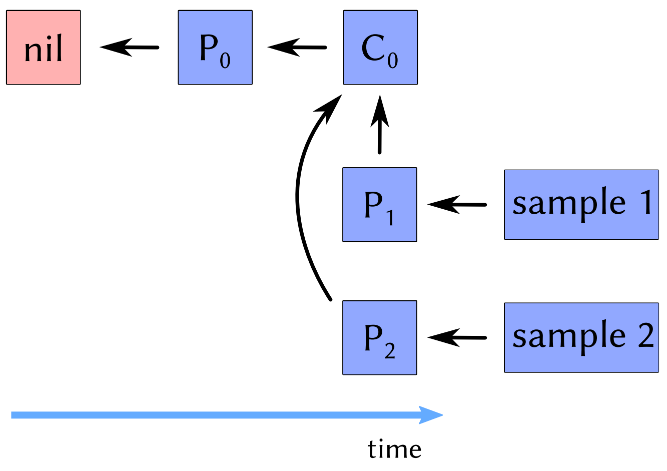


Fig. 4.4: Representation of a sample split: Each sample has only sample-specific process data in its after-split processes ($P_{1,2}$, the position within the parent sample, the size after the split, etc). These sample-specific processes have a common predecessor C_0, which contains all general split properties (split method, whether there still is remaining material of the parent sample, etc).

Fig. 4.4 shows how to represent a sample split. The concurrent C_0 must be the effect of the last process of the parent sample. Then, there are detail processes $P_{1,2}$, each of which represents the state of each child after the split, respectively. As with other applications of concurrents, the concurrent contains general properties, and the sample processes individual properties of the split. See the figure caption for examples.

4.3 Unknown chronological order

If it is unknown which process comes first in a groups of processes, they are organised in parallel in the graph. Note that this documents that information is missing. In particular, if the processes are invasive, i. e. they may change the sample significantly enough to alter results of subsequent processes, this means that the trustworthiness of the following processes is reduced.

5 Prototype: JuliaBase

JuliaBase is a Python/Django framework for creating ELNs, or same databases, with a high degree of customisability. Because it realises a process/sample-based workflow in a highly-structured manner, it is a good candidate for prototyping SciMesh.

Fig. 5.1 shows a simple sample data sheet. In chronological order, you can see what has been done with the sample. In this case, only one thing: The “5-chamber deposition” is the only experiment here. It consists of three layers of silicon which have been deposited on the substrate, each with its setup configuration (temperature, gas flow rates).

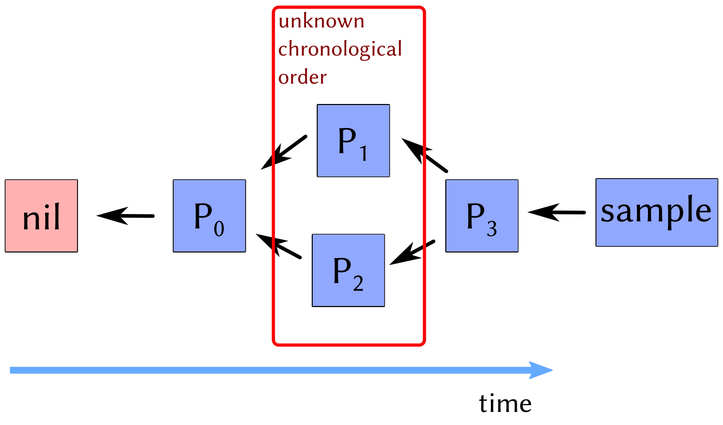
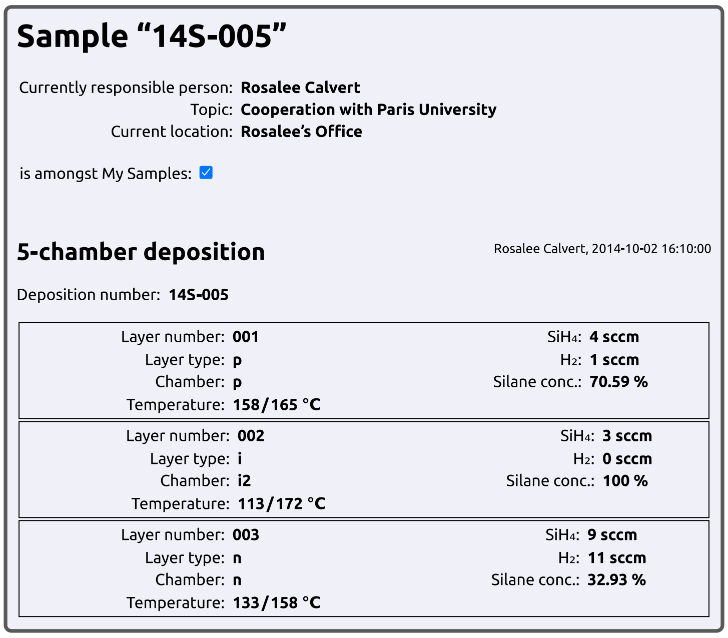


Fig. 4.5: Representation of unknown chronological order: The affected processes are placed in parallel in the graph. All arrows are "cause" relations.



Sample "14S-005"

Currently responsible person:	**Rosalee Calvert**
Topic:	**Cooperation with Paris University**
Current location:	**Rosalee's Office**

is amongst My Samples: ✓

5-chamber deposition Rosalee Calvert, 2014-10-02 16:10:00

Deposition number: **14S-005**

Layer number:	**001**	SiH_4:	**4 sccm**
Layer type:	**p**	H_2:	**1 sccm**
Chamber:	**p**	Silane conc.:	**70.59 %**
Temperature:	**158/165 °C**		
Layer number:	**002**	SiH_4:	**3 sccm**
Layer type:	**i**	H_2:	**0 sccm**
Chamber:	**i2**	Silane conc.:	**100 %**
Temperature:	**113/172 °C**		
Layer number:	**003**	SiH_4:	**9 sccm**
Layer type:	**n**	H_2:	**11 sccm**
Chamber:	**n**	Silane conc.:	**32.93 %**
Temperature:	**133/158 °C**		

Fig. 5.1: Data sheet of sample "14S-005", as seen in a JuliaBase instance by the browser.

```
@prefix jb: <http://juliabase.org/jb#> .
@prefix jb-s: <http://juliabase.org/jb/Sample#> .
@prefix ns1: <https://inm.example.com/FiveChamberLayer/> .
@prefix jb-p: <http://juliabase.org/jb/Process#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix s.o: <https://schema.org/> .
@prefix sm: <http://scimesh.org/SciMesh/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://inm.example.com/samples/14S-005> a <https://inm.example.com/Sample> ;
    jb-s:currentLocation "Rosalee's office" ;
    jb-s:currentlyResponsiblePerson <https://inm.example.com/User/7> ;
    jb-s:name "14S-005" ;
    jb-s:topic "Cooperation with Paris University" ;
    sm:state <http://inm.example.com/5-chamber_depositions/14S-005> .

<http://inm.example.com/5-chamber_depositions/14S-005> a sm:Process,
        <https://inm.example.com/FiveChamberDeposition> ;
    rdfs:label "5-chamber deposition 14S-005" ;
    jb-p:comments "" ;
    jb-p:finished true ;
    jb-p:timestamp "2014-10-02T14:10:00+00:00"^^xsd:dateTime ;
    sm:cause () .

ns1:13 a <https://inm.example.com/FiveChamberLayer> ;
    jb:isSubprocess <http://inm.example.com/5-chamber_depositions/14S-005> ;
    ns1:number 1 ;
    ns1:chamber "p" ;
    ns1:sih4 [ a s.o:QuantitativeValue ;
            s.o:unitText "sccm" ;
            s.o:value 4.000 ] ;
    ns1:temperature1 [ a s.o:QuantitativeValue ;
            s.o:unitCode "CEL" ;
            s.o:unitText "°C" ;
            s.o:value 158.000 ] ;

ns1:14 a <https://inm.example.com/FiveChamberLayer> ;
    …
```

Fig. 5.2: Turtle representation of sample “14S-005”.

Although this sample data sheet is so simple, its RDF representation is rather complex, see Fig. 5.2. This RDF is in turtle format, which is human readable (well, with some experience). After the namespace prefixes (the lines that start with `@prefix`), which serve merely to abbreviate common prefixes with very short names, you can see the sample with its properties, living in the `jb-s` namespace. The `cause` property points to the last process made with this sample.

This process is the only one at the same time (`sm:cause: ()`). It is the deposition process. It ends with the empty line. By the way, the instance-specific entities (in other words, the things that are special for the respective institute like the experimental methods) live in the namespace `ns1`.

What follows is the first layer with its data. It links to its deposition with the `jb:isSubprocess` property. The data of the second layer and the complete third layer are elided here for clarity.

5.1 Give it a try!

The prototypical implementation is kept in sync with this document. It is made against the JuliaBase software in its graphs branch. There is also a short howto for getting RDF data out of a JuliaBase test instance.

6 Note about content addressing

"Content addressing" means naming a content after its … well … content. By using the checksum of the content as the name, name and content become one. This is the most persistent identifier for data you can get. No ambiguities, no manipulations.

Such identifiers can be used to cite an element of a SciMesh graph. For example, one can cite a SciMesh insight in a paper. Or, one can use such identifiers in a blockchain to document absolutely reliably that a certain scientist made a certain discovery at a certain point in time.

The previously mentioned Git, for example, uses hash values for naming each commit. Those hashes are not random: They are the checksum of the current source tree content, and *all* changes that have led to this content. So, the complete history of a project is encoded into the commit's name. This way, one cannot tamper with a Git repository without changing all names, too. Conversely, if I refer to a certain Git commit by its name, I can reliably check that I got unmanipulated content. Content addressing at its best.

Similar methods are used in IPFS or various (all?) blockchains.

However, in an RDF triplet store, you can change anything as you like at any time (given you have the necessary permissions).

In order to get content addressing in SciMesh, one may capture a snapshot of a SciMesh graph in IPLD. IPLD would return a name for it, which will be persistently linked to that graph with exactly that content. However, this method has never been tried so far, so for the time being, content addressing and immutability in SciMesh should be considered future tech.

7 Data provenance

Data processes convert input data to output data. They can contain simulations, but also data conversion, evaluation, aggregation, and visualisation. They should be atomic, i.e. not consist of sub processes but this is not a requirement.

In SciMesh, data processes are of class "Process" just like experimental processes. You can chain them with "cause" relations, meaning that a data process has as its potential input all the data produced by its predecessors.

7.1 Bulk data

With "bulk data", we mean an opaque octet stream of data at a specific URL. In order to be referred to in a SciMesh graph, the response from the web server should include a correct content type. Moreover, the URL must contain the checksum of the data. If the protocol schema itself does not provide this (e.g. IPFS URLs do), the URL fragment (the part behind the "#") must contain a hash using Multiformats. In particular, the format is:

```
<base>base(<version><multihash>)
```

In other words, the binary <multihash> is encoded by the function "base()" (e.g. base32), and the character <base> denoting that function ("b" in case of base32) is prepended. <version> is always the byte 0x01.

7.2 Data input

In order to see the exact data that is used, you have to have a deeper look into the process (e.g. by inspecting the inputs manifest in the processing program). In SciMesh, URLs to bulk input data are not explicit. (Of course, you can make them explicit with your own vocabulary.) Analogously to physical samples, the input is the whole graph of processes (and in particular, their data outputs) that led to this process.

While technically, the program that does the data processing can download any input data, a valid SciMesh graph makes sure that all of that is output of a preceding process. Violating this is like not including all sample-influencing parameters into a physical process.

In some cases that could mean that you have to create a preceding process just to connect it with bulk output data URLs. Just do so, it is fine.

7.3 Data output

Any data output is represented by URIs that resolve to retrievable URLs with that data, which are connected with the process using custom vocabulary (as it is with measurement data for experimental processes). The process must be the subject of such triples.

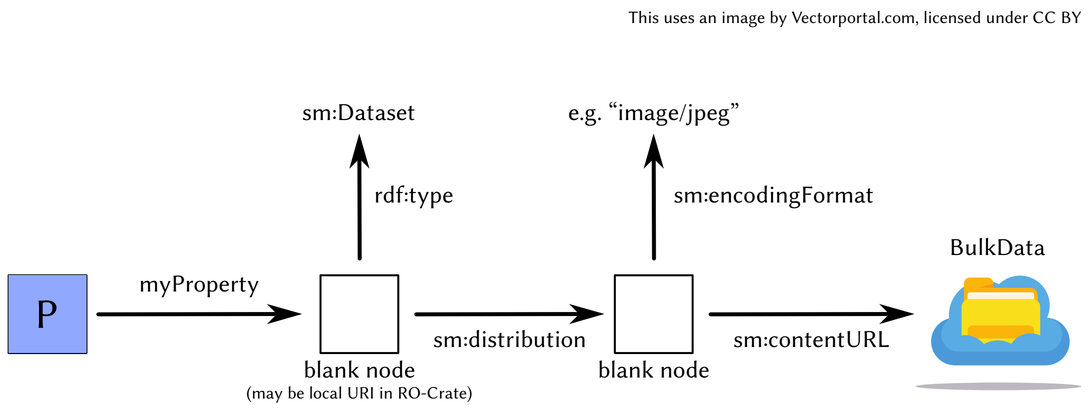


Fig. 7.1: Representation of bulk output data in SciMesh. Here, "sm" is the namespace "`http://schema.org/`".

Fig. 7.1 shows

8 Implementation

The following gives best practise and normative requirements for how to implement SciMesh at an institution, in particular, in an ELN or samples database.

8.1 Workflow

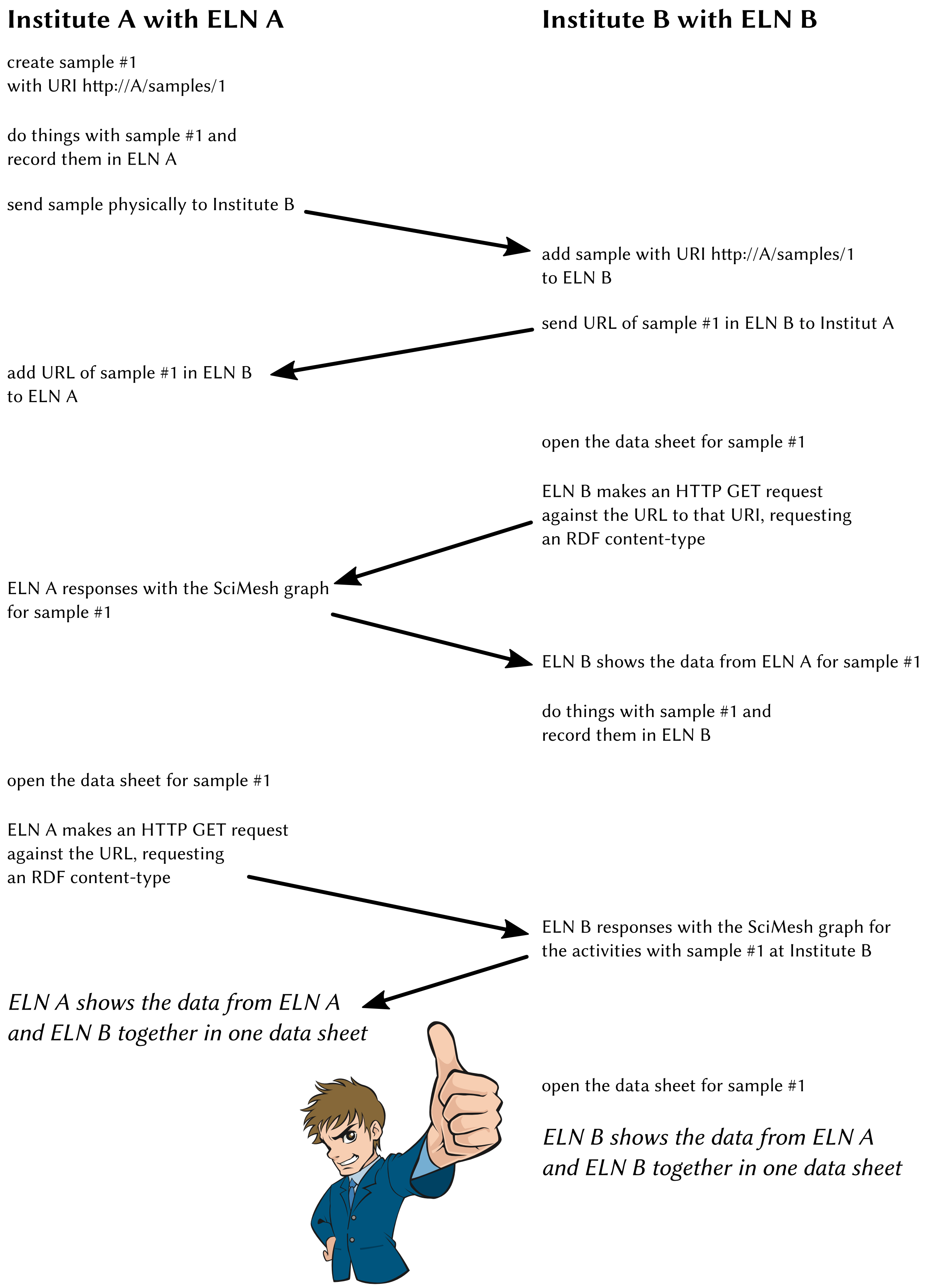
Fig. 8.1 shows the bouncing of samples and data between two institutes working together. The problem at hand is the experimental activities with sample #1, which is created in Institute A, but also investigated in Institute B. At the same time, it is necessary to be able to look at *all* data in *both* institutes.

The text-heavy figure contains most of the detail. Therefore, I will only make some remarks in the following.

Both institutes have their own ELN. Those two ELNs are not only different instances on different computers; they may even be different software.

It is important to see that Institute A is the "real" home of the sample since the URI of the sample is actually a URL in the domain of Institute A. However, there is a URL (*not* URI) for sample #1 at Institute B. Any party needing to collect all data of sample #1 must poll both URLs. Instances may deliver data also found at other instances but this is not guaranteed.

Institute A keeps a list with all URLs that also have data of sample #1, and exports it as part of the SciMesh graph of sample #1.



Institute A with ELN A

Institute B with ELN B

create sample #1
with URI http://A/samples/1

do things with sample #1 and
record them in ELN A

send sample physically to Institute B

add sample with URI http://A/samples/1
to ELN B

send URL of sample #1 in ELN B to Institut A

add URL of sample #1 in ELN B
to ELN A

open the data sheet for sample #1

ELN B makes an HTTP GET request
against the URL to that URI, requesting
an RDF content-type

ELN A responses with the SciMesh graph
for sample #1

ELN B shows the data from ELN A for sample #1

do things with sample #1 and
record them in ELN B

open the data sheet for sample #1

ELN A makes an HTTP GET request
against the URL, requesting
an RDF content-type

ELN B responses with the SciMesh graph for
the activities with sample #1 at Institute B

*ELN A shows the data from ELN A
and ELN B together in one data sheet*

open the data sheet for sample #1

*ELN B shows the data from ELN A
and ELN B together in one data sheet*

Fig. 8.1: Possible workflow for two institutes working together with the same samples.

Two aspects are not covered at all in this graphics: caching and permissions. While the first is an optional yet important optimisation, the latter is essential and will be covered later.

8.2 Getting the graph

A major challenge in SciMesh is the fact that in general, the data of a sample or an insight is scattered over many instances, possibly in different institutions and countries.

The two most important things to see here are:

1. All URIs of samples and processes are constant. They *never* change.
2. All of these URIs are URLs at the same time.

So, if an ELN wants to show all the data for a certain sample, it first makes an HTTP GET against the sample URI. This yields the sample entity, and possibly some process entities. The ELN then traverses the process graph back in time. Whenever it hits a missing cause process, it makes an HTTP GET against its process URI. This yields a new graph that is merged into the existing one. Then, traversal is continued. At some point, there are no causes to look for any more (all remaining cause fields contain `rdf:nil`), or their URLs cannot be retrieved (because the servers don't respond or we don't have the required permissions). Then, the graph is displayed to the user.

Requirements for ELNs

Participating databases or ELNs need to implement the following:

1. Every process and its process history (i.e. the graph back in time) must be the response to an HTTP GET to that process URI. External processes (i.e. with URIs under the control of other systems) needn't be included.
2. An HTTP GET to the sample URI must return the sample entity, the processes it points to in the "state" properties, and the whole process graph. Again, external processes needn't be included.
3. An HTTP POST to the sample URI with a JSON payload of the form

   ```
   {"state": ["http://example.com/processes/1",
              "http://example.com/processes/2"]
   }
   ```

   adds the containing process URIs to the sample, i.e. adding "state" properties with these URIs as objects.

Note that all of these requests – including the POST – may be answered by HTTP 30x codes and need to be repeated with the new URL.

8.3 Considerations about visualisation

Grouping of processes

Visualising agents should consider all processes that form a connected graph via “concurrent” relations a *group*.

In Fig. 4.2 and Fig. 4.3, the latest non-concurrent process in the group (P_2 in the first figure and $P_{2,4}$ in the second) represents the state of the group. Consequently, its URI is used to refer to the entire group. Despite that, the concurrent should be the top-level or outermost component in the visualisation. The reverse way from the concurrent through the “cause” relations leads to the inner regions of the visualisation. I can be thought of as an onion-like structure, although it needn’t be displayed that way.

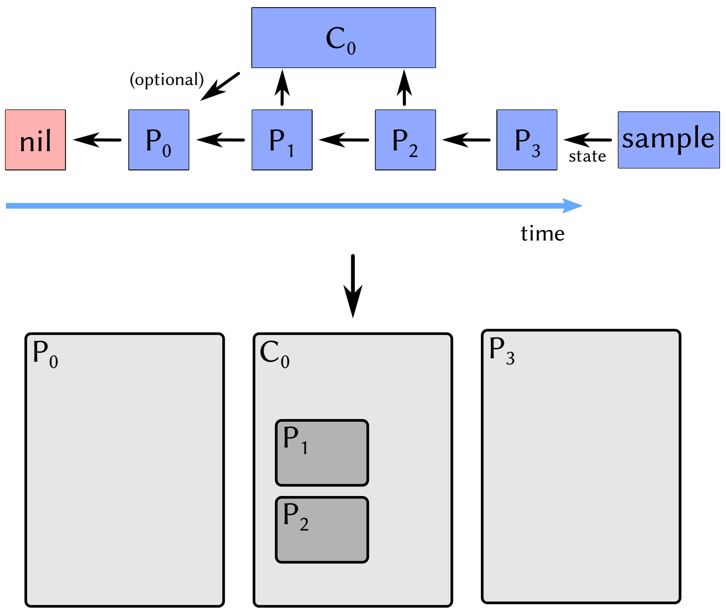


Fig. 8.2: Graph on the top, resulting visualisation on the bottom. This visualisation is just a serving suggestion, of course. For example, arranging P_1 and P_2 horizontally would be more consequent but possibly less aesthetical.

8.4 Good URIs

We recommend to follow the guidelines explained in “Cool URIs for the Semantic Web” when you create URI of any kind, but in particular for samples and processes. Moreover, you may or may not use any of the ubiquitous PID (persistent identifier) services out there, some of which are specialised to physical samples.

In any case, URIs of samples and processes must be URLs at the same time that yield the corresponding SciMesh graph via HTTPS. (Note that the URI should start with `http://`, though.)

8.5 Authentication

Scientific data might be sensitive for a couple of reasons. Anyway, an ELN will generally refuse to deliver knowledge graphs without authentication and authorisation.

Currently, SciMesh-compliant ELNs must implement the following method of authorisation, called “mutual trust”.

Mutual trust

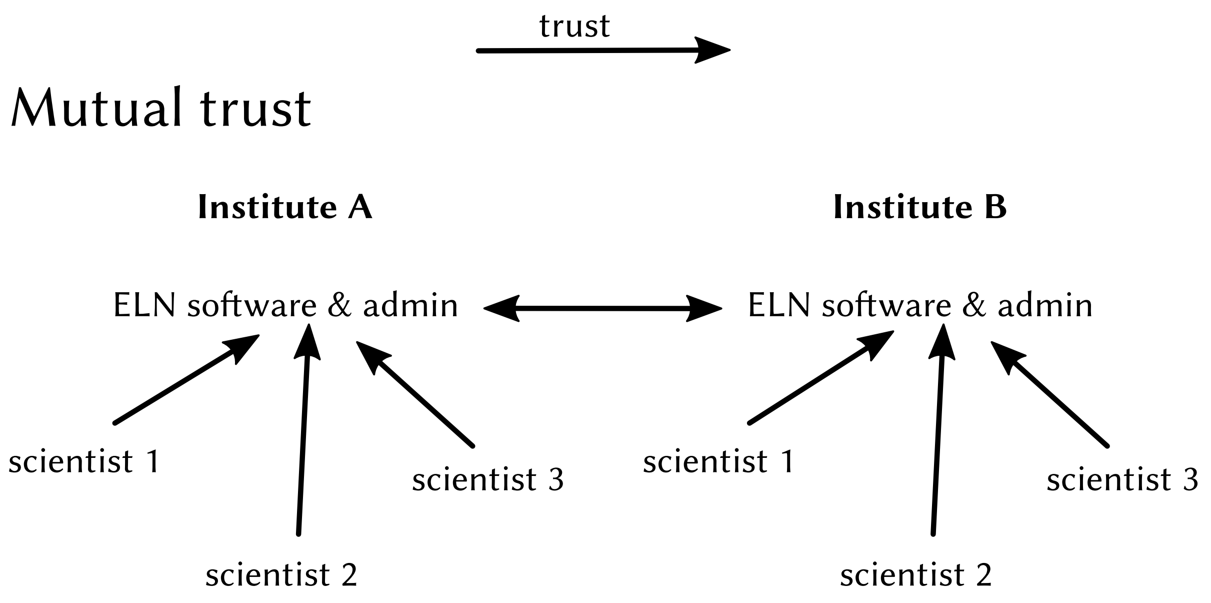


Fig. 8.3: Trust graph for the “mutual trust” method.

This method is very simple. The ELN instance contains a list of peering ELNs that it trusts. In practise, this trust is established by a bilateral agreement between the managements of the respective institutions.

Technically, both ELN instances must send TLS client certificates at each HTTPS request, which are used to authenticate the ELN by the peer.

Note that this scenario does not mean that every scientist can access all data of the peer ELN. It only means that the ELN instance can access all data on the peer ELN, but in general will show only a subset of it to the scientist.

Fig. 8.3 shows the underlying trust graph. As said, both ELNs trust each other, and the scientists trust their respective ELN.

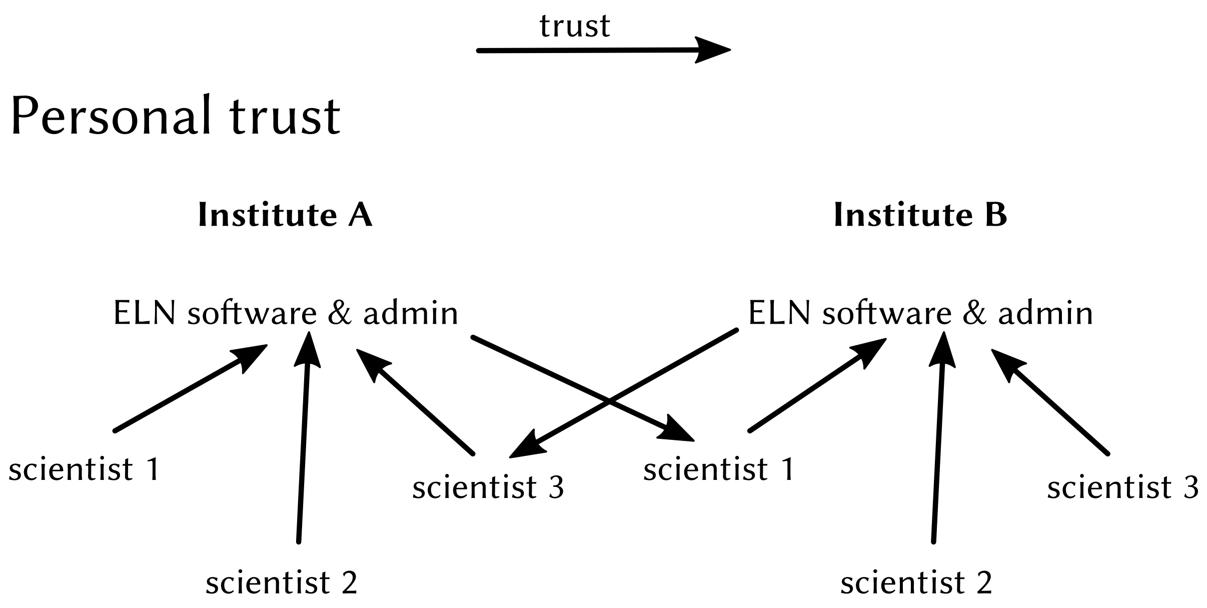


Fig. 8.4: Trust graph for the “personal trust” model.

The “mutual trust” may seem to be all too trustful. Therefore, let’s have a look at a more restricted option.

Fig. 8.3 shows an alternative trust graph. Here, the ELNs do not need to trust each other. Instead, they trust certain scientists of the other institute.

It is not easy at all for an ELN to implement this. Since no data must be relayed through the peer ELN (which we don’t trust), the sample data sheet with the joint data needs to be compiled in the browser by JavaScript or WebAssembly code (or a proxy web service, for that matter) that can be trusted by the scientist not to forward any data to third parties, in particular, not to the ELN of the scientist’s own institute.

This does not make much sense, given that the scientist and the ELN is under the very same governance, therefore, SciMesh does not include an authorisation method for such a trust model for the time being. It may make sense, however, if at least one of the ELNs provides its service to scientists of different institutions (governances).