DFC standard documentation

(presented as a Gitbook to improve readability)

A. Why we need a standard ?

The problems are:

- Multiple duplication of production information by producers, and inability to seamlessly manage stock
- Difficulty to adopt a big data approach and build new services due to data incompatibility (ex: logistics)

The objective is:

- Enable producers to seamlessly manage their product and inventory catalog while selling their products through multiple channels using multiple platforms.
- To enable the development of shared logistics solutions using logistic information from all the platforms in use
- To allow the implementation of interoperable management tools allowing for example a producer to generate integrated reports gathering sales data from different platforms
- Allow the reporting or visualization (via maps for instance) of data coming from multiple platforms
- To allow actors with different business models and technologies to work together without changing their way of working and tools
- Etc.

For this we need:

- That the data follows a shared semantic standard so that it can be shared from one actor to another even if the actors have different data models, speak different languages.

- To be able to link the data to unique identifiers allowing their reconciliation when necessary (eg unique product identifier, unique company identifier, unique person identifier, unique place identifier)
- An architecture allowing this ecosystem to work in a decentralized manner, including the areas of identification/authentication, segmentation of the API on high or low granularity service logic, etc.

B. What is the DFC standard ?

The DFC standard is composed of 2 complementary blocks :
1- a semantic block which is a unified and agreed upon sectoral activity description, presented both in a human (model) and machine (ontology) readable format. This description can be modularized : in the case of DFC, we have separated the “business part” (what we do, how, who, where) and the “product part” (how to describe the nature of the things manipulated in the model) and have joined them in a “full model and ontology”. This is the “common language” that enable the actors using the platforms to communicate.
2- a technical specification on how data should be exchanged between actors implementing the standard. This is the technical language that enable platforms to communication (= share or mutualize data).

C. General strategy to build that standard

a. Remain independent

We could have setup a standardization group within GS1. We chose not to as GS1 were asking us to sign a form to tell we commit to use the GS1 standards. How can we commit to use a specific solution when we have not yet investigated our problem, the potential solutions available, etc. ? We are pretty happy we made that decision, as we remained free to work with Open Food Facts for products identification, which wouldn’t have been possible in the context of a GS1 standardization group.

b. Collaborate with Open Food Facts for product identifiers and ontologies

We need a general common universal product id base to be able to match products between platform, and know if we talk about the same product, or another. In the agro-industry and retail industry agents use the EAN, GS1 product id codes. In the context of local and alternative food systems, short food chains, very few producers have subscribed to GS1 services, they don't have EAN. And probably most won't want to subscribe and pay the 85€ min yearly fee. So we have chosen to collaborate with Open Food Facts and use their identification system as our universal products identifiers. When there is an EAN, Open Food Facts use it as product identifier. When there is not they attribute one randomly, for free. We can call those ids "pseudo EAN".
As we don't want to force all actors to open their data (this should remain the agent decision), we will have our own "DFC universal product" catalog but we will use Open Food Facts API to generate ids for products which don't have EAN. We also have chosen to use the same taxonomies as Open Food Facts to describe products, so a product file in the DFC catalog will be super easy to open and duplicate into Open Food Facts anytime the agent want to open their data.

c. Collaborate with "The Commons" for shared data storage

We also choose a "neutral" agent to host the shared data of the consortium : this DFC universal products catalog. That avoid any conflict of interest among the consortium partners.

d. Iterative process and prototype base development

When building both the semantic and technical specifications of the standard, we need real life cases to test if what we imagine is adapted, works well, enable to do what we want to do. So we are developing a prototype to not only make a "proof of concept" and show the potential of the standard to solve the problems mentioned above, but also to build and improve step by step the semantic and technical standard.

D. Status of the project

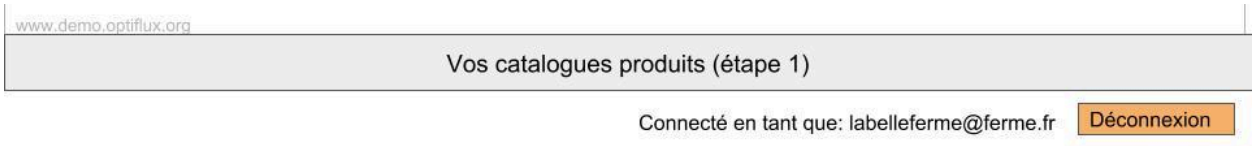
a. First releases

We released in November 2017 a first version of the semantic standard, then a second version in June 2018, and we just released a new version in May 2019. You can access the corresponding ontology through Github. For the technical specification, we started to work on it in october 2018 and we just released a first specification through this document.

b. The POC to test the v.0 of the standard

We needed to start working on the prototype to build a first version of the technical standard, understand the current status of the data and of the first platforms willing to adopt the standard. We built a prototype on the use case of sharing of products catalogs between platforms and mutualization of logistic flows.
This prototype will be developed in 3 steps:

1- Producers can access their universal catalog and visualize what they propose to each of their distribution channels. Before going further and talk about products transportation, we need to make sure the ontology enable us to correctly manage products information between platforms.



Mes produits et quantités proposées par circuit de distribution

Type de produit	Nature	Conditionnement	Quantité La Ruche Vitry (LRQDO)	Quantité Ekybio (Panier Local)	Quantité AMAP du lutin (Cagette)	Quantité Micromarché (OFFrance)
Tomate	Tomate coeur de boeuf	1kg vrac	20	20	-	0
Tomate	Tomate marmande	1kg vrac	À volonté	À volonté	-	-
Tomate	Tomate marmande	5kg vrac	À volonté	À volonté	-	-
Miel	Miel de thym	1 pot de 500g	-	10	10	10
Miel	Miel de lavande	Lot 2 pots de 500g	-	10	10	10

2- Producers will be able to see, beyond their cross-platforms catalog, logistic flows planned whatever their distribution channel.

www.demo.optiflux.org

Vos livraisons à venir (étape 2)

Connecté en tant que: labelleferme@ferme.fr Déconnexion

Livraisons à effectuer sur les 7 jours à venir

Mes livraisons

Date de livraison	Lieu départ	Lieu arrivée	Poids	Volume	Hub distributeur	Plateforme outil
22/11/17 18:00 CET	12 rue Vignon 14120 Le Petit Pré N 66° 33.122'	25 rue de Rivoli 75001 Paris N 66° 24.033'	125Kg	2m3	La Ruche Rivoli	La Ruche Qui Dit Oui
23/11/17 09:00 CET	12 rue Vignon 14120 Le Petit Pré N 66° 33.122'	27 rue de la Goutte d'Or 75018 Paris N 66° 13.323''	250Kg	5m3	La Louve	Odoo

3- Producers will be able to define some filters to display compatible logistics flows from other producers within the ecosystem using the same standard.

www.demo.optiflux.org

Vos livraisons à venir, et les trajets compatibles (étape 3)

Connecté en tant que: labelleferme@ferme.fr Déconnexion

Filtres

Date début Date fin

Montrer les trajets dont lieu de départ à moins de 20 km Montrer les trajets dont lieu d'arrivée à moins de 5 m

Montrer les trajets dont date de livraison estimée à moins de 5 eures d'écart

Mes livraisons

Date de livraison	Lieu départ	Lieu arrivée	Poids	Volume	Hub distributeur	Plateforme outil
22/11/17 18:00 CET	12 rue Vignon 14120 Le Petit Pré N 66° 33.122'	25 rue de Rivoli 75001 Paris N 66° 24.033'	125Kg	2m3	La Ruche Rivoli	La Ruche Qui Dit Oui
23/11/17 09:00 CET	12 rue Vignon 14120 Le Petit Pré N 66° 33.122'	27 rue de la Goutte d'Or 75018 Paris N 66° 13.323''	250Kg	5m3	La Louve	Odoo

Trajets compatibles

Date de livraison	Lieu départ	Lieu arrivée	Poids	Volume	Hub distributeur	Plateforme outil
22/11/17 15:00 CET	6 rue de maux 14250 Bayeux N 66° 33.135'	3 rue du chatelet 75001 Paris N 66° 24.043'	50Kg	1m3	AMAP du Chatelet	Cagette

We have now specified more precisely the UX of the first step of the prototype (document in French). And already it raises a lot of questions, about synchronization of data for example : when a producer has connected and imported data from various platforms, solved conflicts, etc. If he modifies a data in one platform, and make another change for the same product in another one, etc. How do we synchronize ? For now we don't have platforms reading and updating info from a universal shared catalog. We don't know yet how we are going to manage such issues, but we'll move forward one step at a time !

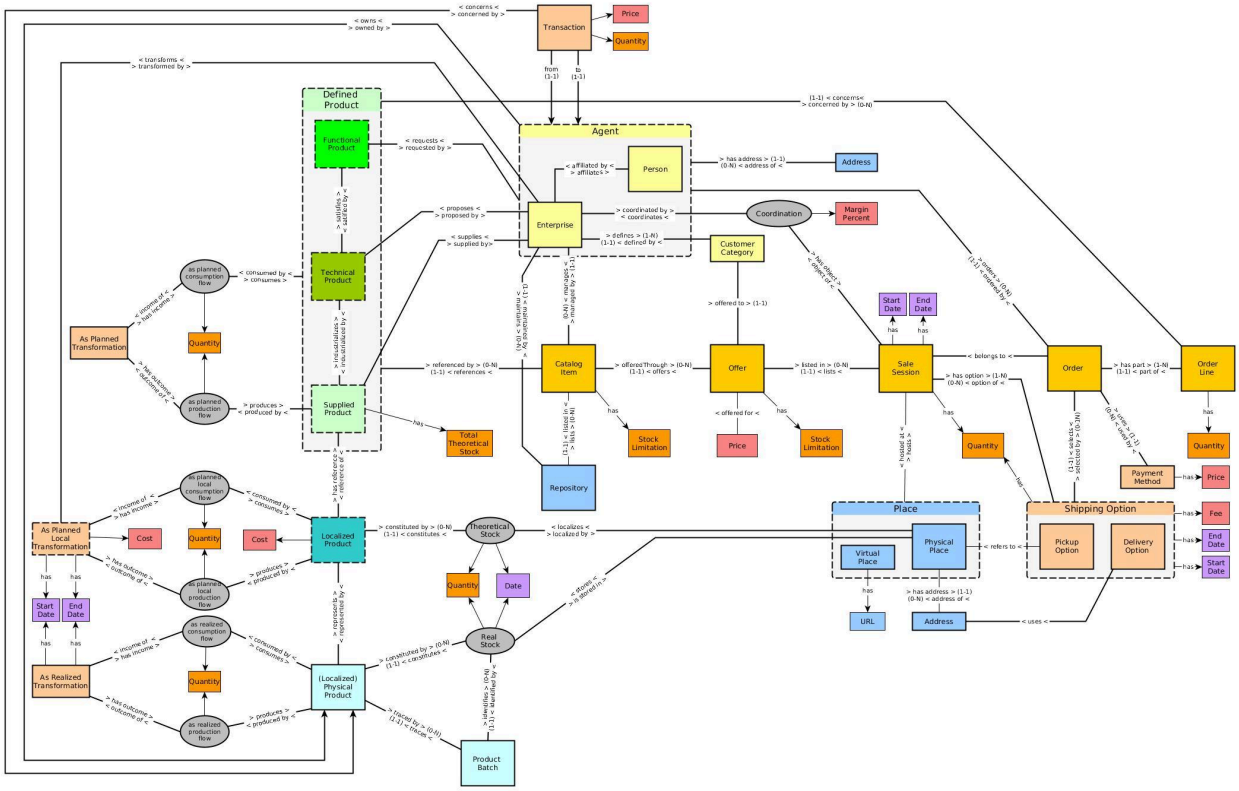
So developing this prototype will enable us to test real life the semantic specification of the standard (the ontology) and the technical specifications (the protocols, etc.) described below, and improve them step by step.

E. Semantic specification

This specification has been built since March 2017, through alternation between field interviews and modelization. We have been supported in that work by an ontologist, Bernard Chabot.

a. Business model and ontology

The human readable modelization can be represented as follow:



Click here to access a zoomable version of the document.

i. Products

- We have identified the different concepts of products we manipulate. We distinguished the product “need” (what I want as a customer), from the product “answer” (what I propose as a distributor to satisfy your need), from the product “supply” (what I propose

as a producer that enable distributors to meet their promises to customers), all those products being manipulated without any “location” notion.

- Then a producer identifies a location where their products are supposed to be when they become real product. We call them “localized products”, which is a combination between an ID product (what product it is) and an ID place (where it is). For instance, the potatoes of “Awesome farm” are in theory localized in the farm itself.
- When the potatoes get harvested they become “physical products”, real products that you can hold in your hand. A physical product is also always located somewhere, and belongs to a product batch.

ii. Transformation

- Some products are “composed products”, they are made of other products, processed in some way to make a new product, like a tomato sauce for instance. There is a theoretical transformation that plans a transformation process without any notion of where the products are located, the “recipe” that connects products with one another independently from where they could be. For instance as an artisan cookies maker I can tell that I put 100g flour, 20g chocolate, 2 eggs, etc. in my cookies. In that case I express the recipe in term of “functional products”, I need flour and chocolate. I can be more precise and tell which type exactly of flour and chocolate I use and talk more in term of “technical products”, like wheat flour T110. And I can even tell exactly the flour from which farmer I used in my recipe, so express the recipe in terms of other “supplied products”. This is what we call the “as planned transformation flow”.
- Then this recipe becomes something more like a “production workflow” that adds some notion of location. I have to move the tomato from “Awesome farm” to “TheKitchen”, the onions from “The Other Farm” to “TheKitchen”, etc. When all my components are in “TheKitchen” I can start the production process, cook, mix, bottle, etc. And get some jar of tomato sauce as output, which are located in “TheKitchen”. But this is still only a plan, a production map, I still don’t have the products, I’m just organizing and planning the operations. We realized when iterating that **transforming the nature of a located product was exactly the same flow as transforming the place where this product is supposed to be located**. As the localized product is a combination of an ID product and an ID place, one flow was transforming the ID product, the other the ID place. So **we are treating transportation as a specific transformation flow**. Input will be for instance 100 x potatoes 1kg located in “Awesome Farm” and output will be 100 x potatoes 1kg located in “The Great Town Shop”.
- Then when physical products are concerned this flow becomes a “realized transformation flow”.

iii. Sales operations

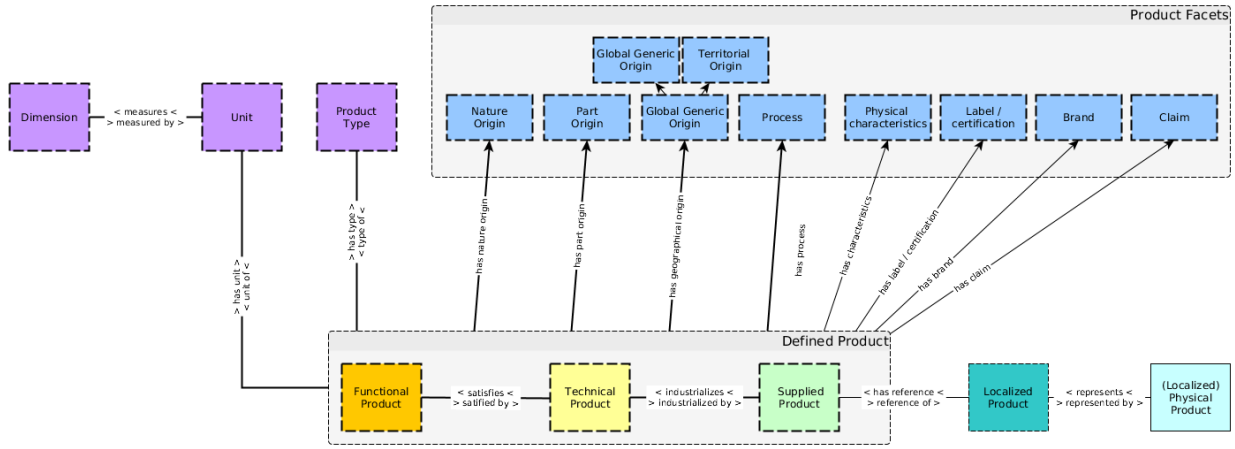
- A distributor (an enterprise) constitutes its catalog, made of “catalog items”, and build offers for them given their customer categories. The product offered can be, depending on the sales and marketing strategy of the distributor, a functional product (ex: tomatoes to stuff), a technical product (ex: beefsteak tomatoes) or a supplied product (ex: beefsteak tomatoes from Awesome Farm). A given agent can have a catalog on various repositories (i.e. platforms) so a same enterprise can have multiple catalog items for the same product, if they use multiple platforms for instance. We will be able to match them using the unique product identifier.
- A sale session aggregates offers under certain shopping conditions (opening / closing dates, etc.)
- The customer makes an order with various order lines in a specific sale session and choose a shipping option - that can be delivery (they are delivered to their home or business address) or pick-up (they need to collect the product in a location defined by the distributor) - and a payment option among those defined by the distributor.
- The sale happens in a place that can be physical (a physical store) or virtual (an online store). Note that this works as well for a physical store: technically each day the store opens and close at a define time, and each day can be considered as a specific sale session. In the case of physical store sales, the shipping option is implicitly “collect on site”.

iv. Transaction

- When an agent has ordered products and products has been delivered through a last transformation flow (transport), the ownership of the product changes hands. The transaction is then officially happening and the previous product owner can invoice the customer.

b. Product ontology

Here is a simplified representation of the product ontology:



Click here to access a zoomable version of the document.

The question of products identification is treated in the technical standard and won't be discussed here. We came up with that model through various iterations, you can especially check that spreadsheet illustrating the cases we used to illustrate and validate our last iteration. We chose to characterize products through various orthogonal facets that enable precise understanding, rich research and comparisons. This will be really useful for several use cases. They will enable platforms to order products received from automated data exchange in their own appropriate taxonomy. Also, actors will be able to make searches in a pull of products from various platforms using various custom taxonomies.

So we choose the following "criteria" to uniquely identify products:

- **Product type**: this is a basic sales oriented taxonomy (are you selling carrots or soap ?).
- Then some **facets help identify more precisely** the products:
    - If the product has a unique **"living/mineral" source**, what is the source? For instance a steak has as source a living cow from a specific breed. If a product is composed it can be decomposed through the "transformation flow" mentioned above and each component can be described following the same process.
    - If the product has a unique "living/mineral" source, **what part or product of this source is used**? For instance honey comes from a unique living source which is a bee from a specific breed. The part of product of source concerned here is "honey". If we talk about a carrot, the source will be the carrot variety, and the part concerned will be "the root". If I talk about carrots seeds, the part concerned will be the seeds. If I sell carrots with the leaves it will be "whole plant".
    - Every product sold will have gone through some sort of **processes**, at a minimum a tomato has been "harvested" from the living tomato plant. So even what we call "raw products" have undergone some basic process. Salt will have been "dried", etc.
    - Each product has a **geographical origin**, it comes from somewhere. It can be a **territorial origin** (ex: France, Nice, etc.) or a **general global origin** (ex: north west atlantic sea) so this facet can have values from two taxonomies even though there is only one facet. A cocoa bean can be from the same producer (big farmer owning lots of parcels for instance), go through the same harvesting,

fermenting, drying processes, come from the same variety of cocoa plant, BUT come from a different location.
    - Products can have certifications/labels
    - Products can have some physical characteristics (soft, tender, etc.)
    - Products can have some specific claims (“gluten free”, “zero salt”, etc.)
    - And products can have a specific brand
- And to finish a product is also identified with a **dimension and unit**, like potatoes are sold by 1 x kg, a jar of tomato sauce is sold by 1 x item, the item here being a “jar of 500 ml tomato sauce”.

Behind each of those facets, we need taxonomies, or it can be a free field but then it’s hard to make searches !
As we understand how complex it is to maintain taxonomies alone, we decided, just as we did for product identification (see technical specification / metadata repository), to use Open Food Facts taxonomies. They are not totally aligned with the way we wanted to describe products so for the first real life test (prototype) we will ignore some of our facets. We are working hand in hand to make our complementary approaches converge.

Depending on the use case, we might have to ask a user to fill in some unfilled info if they are needed by the integrated platform to process the data. For instance, if a product is in the category “apple” but no variety was filled in. And the data receiving platform has two categories “acidic apple” and “sugary apple”, how can the receiving platform know where to put the product ? So in that case, the UX will require to as the user to fill in missing data.

F. Technical specification

a. Overall strategy for building a reliable spec

The overall strategy is to implement the prototype in roughly two phases:

- Phase 1 : implementing an MVP of the prototype, to showcase the potential of the project and help raise additional funds.
- Phase 2 : the full implementation of the prototype. It will complete phase 1 to enable industrialization and professionalization of the standard through improvement of the tools, of the technical architecture, and of the possibilities of integration with external actors.

b. Design decisions

	Phase 1	**Phase 2**

Protocole	Data must be expressed in a semantic way in the API and must respect the OWL DFC model User-friendly and easy to implement	Phase 1 + professional protocol
Stateless or stateful	Stateless	
Granularity	low granularity service	Phase 1+ high granularity via queries
URL	REST resource driven sémantic, without parameter	Phase 1 + parameters enabling queries (SPARQL or HyperGraphQL)
Service specifications	OpenAPI except for the input/output data structure, for which use OWL. LDP specification compliant.	- Complete OpenAPI specifications for standard services. LDP specification compliant - SPARQL spec for high granularity service by Query.
Serialization	JSON-LD LDP specification compliant	JSON-LD (JSON-LD in the data attribute if HyperGraphQL) LDP specification compliant
Transport layer	HTTP + LDP	HTTP + LDP HTTP + SPARQL or HyperGraphQL
Single or multi-source	Simple access to one logical source	Query on multiple sources
Right delegation	No (all or nothing, all decided by the platform)	Yes (web ACL if SOLID used)
Identification and authentication	OIDC (hosted by “les communs”)	OIDC (or web-idOIDC if mature via Solid)
Data storage	Distributed	Phase 1 + centralized cache to improve performances
User data	ID centralized by user	Phase 1 or web-id/OIDC if mature enough in order to achieve a decentralized authentication
Product data	ID provided by Open Food Facts	Decentralized ID management using the semantic web and SOLID
Federation or syndication	Ontology: Federation Taxonomy: Federation Storage: Syndication	

	Identification and authentication : Federation Validation: ? Synchronisation / caching: Federation Notification: ? Serialisation : Federation Others aspects of the protocol: Federation	
Interface	Native web components	Phase 1 + Startin'blox or SemViz

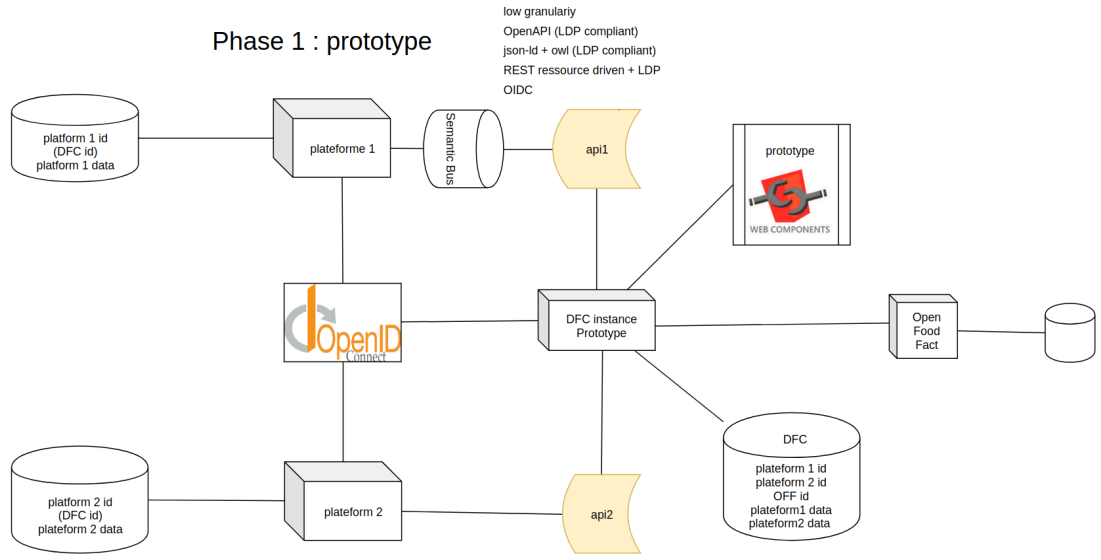
Federation: all entities follow the same protocol
Syndication: entities may have different protocols

c. Architecture representations

Here we try to represent some of the designs choices above between phase 1 and phase 2:

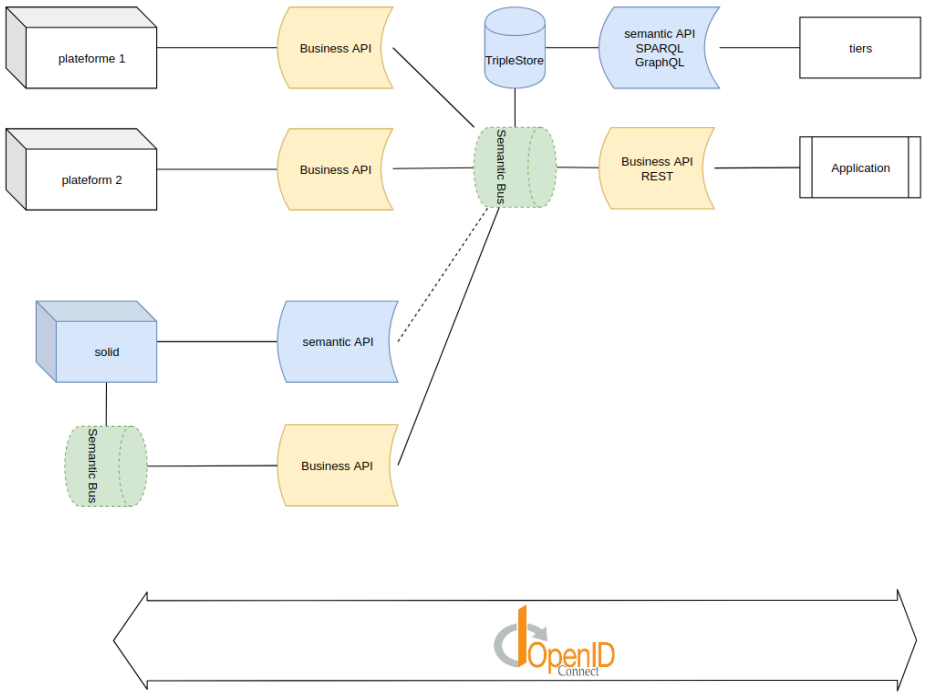
- Single or multi-source
- Identification and authentication
- Data storage

Phase 1

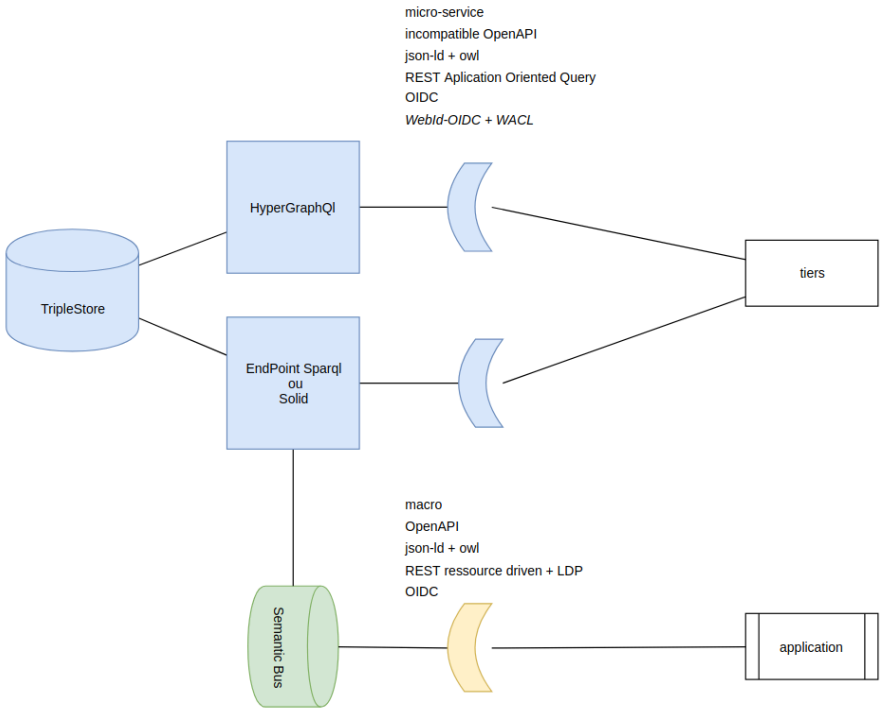


Phase 2

Phase 2 : industial proof of concept



Phase 2 : cache detail



d. Appendix 1. Decisions points and choices

i. Stateless or stateful?

When several requests are made, should they be independant or can they rely on each other?

- In a “stateless” setting, no state information is retained on the server itself. If we do 2 requests, the second one does not need to know the results of the first one, but it will have to contain all the required information for the server to be able to return an answer from scratch.
- In a “stateful” setting, state information (session) is retained on the server. The 2nd request will be done on the basis of the input and output of the 1st request .

Some of the main advantages of each approach impact performance and scaling:

- A stateless architecture makes it much easier to scale horizontally, which means to deploy additional servers (often identical) hosting the same application on which the load is spread. Whereas if stateful, the servers store a state and in order to perform horizontal scaling, either this state will need to be synchronized between servers or the same user will always have to be directed to the same server - which makes it harder and less efficient to implement.
- In a stateful architecture each request does not need to repeat steps that were already performed or provide the same information again. The most common use of this is for identification/authentication, which then only needs to be done on the first request saving significant processing time.

The common best practice today is to use stateless services, making each request independant. It makes for easier debugging, maintenance and scaling but needs additional development for identification and authentication especially.

Conclusion: implement stateless service

ii. Service granularity

The notion of service granularity describes what functional level will a service address, rather high level like an entire user or very details like a specific attribute of this user.
A highly granular service will split the interactions into low level functional blocks, whereas a standard service will provide richer and more complete interactions, but not splittable.

For instance:

- A service that manage the product catalog including product categories, stock, etc. This would be a **low granularity service**
- A **highly granular service** would just be the stock information. The name of the product. Etc. Everything is split in small high granularity services, that can either be assembled on the service side as one low granularity service, or let the client assemble these high granularity services itself. For the use case of mutualized logistic flows, we could have a specific service on "routes" without any information on what is transported. Another

service on orders, to find out what was ordered. And then have a view that combines the different services.

Some pros and cons of each approach:

	Pros	**Cons**
High granularity service	- Easier to maintain and debug - Less data being transmitted as only what is needed is requested - More flexible and future proof - **Easier autorisation management**	More services to keep track of
Low granularity service	- Less service calls - Easier to develop - Performance gains when it comes to assembling data via for instance joints at the DB level	- Services are more complex and difficult to maintain - Often requires ad-hoc development when new usage appear

The main advantage of highly granular services in the context of interoperability with authorization management is the simplicity of development on the server side. For instance it is easy to answer in an interface that the owner of some data denies access to it for a specific client. The service will just use the standard HTTP responses (401 or 403). This becomes more complex if this entity is nested in another (as for a low granularity service). In this case, the server will respond well to the interface with the root data but will have to express one way or another that its answer is not complete because it lacks a part for which the user has no rights.

Conclusion: implement the needed low granularity services for phase 1 and in phase 2, implement in addition more granular query based services

iii. Service standard

An old standard exists to describe API, coming from the XML world but today applicable to most data format: WSDL. This norm is complex as it is very generic and is able to describe almost any kind of API, not only HTTP, and not only synchronous or unidirectional. Working hand in hand with WSDL, and nearly as complex, the UDDI standard allowed for registration and discovery of services.

OpenAPI on the other hand focuses on HTTP and on the service usage from a resource point of view, which makes it much simpler. **It allows to document et specify the API (the endpoint, the URLs structure, the serialisations...).**

The capability of being able to specify by a given standard, the form of the url, the ability to work in a Query logic, the serialization, the use of a model and its visibility with each request is summarized in this table:

	Spec OpenAPI	URL	Query	Sérialisation	Data model accessible per API	Data model in the API response
REST + OpenAPI	Yes	No	No	Yes (json)	Yes but OpenAPI specific	Yes but OpenAPI specific
REST resource driven semantic	**Yes**	**No**	**No**	**Yes (rdf/ttl/json-ld)**	**Yes but OpenAPI specific**	**Yes (owl)**
REST application driven (SPARQL)	**Partial**	**Yes**	**Yes (SAPRQL)**	**Yes (rdf/ttl/json-ld)**	**Yes (owl)**	**Yes**
GraphQL	Partial	Yes	Yes (GraphQL)	Yes (json))	Yes but OpenAPI specific	No
hyperGraphQL	**Partial**	**Yes**	**Yes (GraphQL)**	**Yes (json-ld in data property)**	**Yes (owl)**	**Yes (owl)**

1. URL structure

Resource driven (logique REST)

A URL is a unique address pointing to a single resource. This resource can be a container that contains several atomic resources.

```
1: http://serveur/idContainer/idRessource1
2: http://serveur/idContainer/idRessource1/attribut1/
3: http://serveur/idContainer/idRessource1/attribut1/attribut2/
4: http://serveur/idContainer/idRessource1/attribut1/idRessource2/attribut2/...
```

A URL may also contain parameters, using the special characters ‘?' to mark the start of the parameters and '&' to separate each parameter. Previous examples translated with parameters:

```
1: http://serveur/idContainer?idRessource=idRessource1
2: http://serveur/idContainer?idRessource=idRessource1&attribut=attribut1
3: http://serveur/idContainer?idRessource=idRessource1&attribut=attribut1.attribut2
```

4: difficile à exprimer voir 6

With pagination parameters to load data by package rather than all at once:

5: http://serveur/idContainer?idRessource=idRessource1&start=0&length=100

Application driven

The more parameters an API can handle, the more generic the API becomes that encompass the entire application. This type of URL changes the design of an API since it no longer creates an API by type of resource but a generic API capable of meeting the different needs of the application.
The tendency is to make parameters that contain json objects which makes APIs much more generic (search, projection (choice of fields), sorting ...). The syntax use is usually based on the MongoDB NoSql database “standard”.

6 :
http://serveur/idContainer?search={attribut1:{$gte:value1}}&projection={attribut1:1,attribut2:1,attribut3:0}&sort={attribut1:1,attribut2:2}&start=0&length=100

The request above means retrieving the resources related to the container idContainer :

- Whose attribute1 is greater than value1
- Only retrieve attribute1, attribute2 but not attribute3,
- Sorted according to attribute1, then attribute2
- Starting with the first document found and returning 100 documents

The more complex parameters the API contains, the less it is possible to use the OpenAPI specification because the result and structure of the query are variable. If the strategy is moving towards this type of url, it is probably better to take the step towards using queries.

Query

The ultimate use of advanced parameters is called Query. This is a normalized query passed in a single parameter with all the information needed for server using this standard to execute the request. In the semantic world it is SPARQL. GraphQL offers another standardization specific to this technology for queries.

The disadvantage of SPARQL queries is the inherent complexity of the semantic web. GraphQL can be an alternative, but it will require to transform the OWL ontology into GraphQL types. The drawback with the graphQL approach is that the model consisting of types is not exposed with the data returned.
HyperGraphQL is an implementation of GraphQL dedicated to the semantic web in order to connect to a tripleStore. The API provides data in their json-ld form with context in owl schemas.

2. Gateways between protocols

Ideally Both protocols (de facto standard and semantic standard) should be accepted by the platforms. The API in de facto standard could then be based on the semantic API to avoid duplicated maintenance (putting SPARQL in front). This represents a significant cost to setup and configure.

The following table shows where it is possible to go from one standard to another (if the de facto standard is used in phase 1 and the semantic one in phase 2):

Source	Destination	Comment
SPARQL	GraphQL	- It is possible to convert a GraphQL query into a SPARQL query. - It seems possible to overlay graphQL on a SPARQL API but it's quite experimental. - There is HyperGraphQL also which allows to connect directly to a Jena semantic base. the SPARQL and GraphQL API can then coexist based on the same database.
SPARQL	REST resource driven	REST APIs can be based quite easily on SPARQL APIs to simplify their use, through the semantic bus for example.
GraphQL	REST resource driven	The REST APIs can be based quite easily on GraphQL APIs to simplify their use, through the semantic bus for example
GraphQL	SPARQL	A SPARQL API can only be based on a semantic database (jena or rdf for example)
REST resource driven	SPARQL	A SPARQL API can only be based on a semantic database (jena or rdf for example)
REST resource driven	GraphQL	GraphQL is designed to connect to resource-driven APIs by adding resolvers to the server. It also allows dynamic aggregations / searches on several APIs of several actors.

To build the prototype, we need a user-friendly protocol that is easy to set up quickly while offering the possibility of making a more professional API available. For the sake of maintainability of it all, the user-friendly API should be pluggable into the professional API.

Conclusion: the actors of the consortium agree on a shared vision on the need to have 2 APIs:

- **a "professional" API using SPARQL**

- **A more user friendly API in JSON-LD only on "read" data permissions.**

3. De facto standards or semantic web ?

Do we want to respect the standards of the semantic web, or develop a simpler API, potentially more intuitive, enjoyable, and easy to use?

The standard syntax for querying semantic data is SPARQL (W3C). The standard for semantic APIs takes this query standard and is based on an HTTP url to which is added a parameter (usually endpoint or search) that contains the request. From this, the server request will be done via HTTP (LDP at the margin) and SPARQL, and the answer will be given in semantic data (json-ld or rf or ttl) with a granularity that depends on the request SPARQL but in a generally high granularity service logic.

In the non-semantic world and low granularity services, still predominant, the most used API standards today are

- OpenAPI (managed by a large consortium)
  It is not compatible with json-ld as it is and probably will not be, because the concepts are competing especially in terms of model. It respects the logic of the common good with a broad consortium and shared governance.
- Graphql (pushed by Facebook while trying to build a community) .
  Very centralized by Facebook without trying to make a consortium.

They do not respect the semantic web standards and are not managed by the W3C.

There is here a fine balance to be found between potential for interoperability, coverage of a wide variety of use cases, etc. and usability and ease of usage.
Here we find ourselves in a situation where we can choose to develop an API using web-semantic standards (http + json-ld + sparql) standardized by the W3C but complex and not or more accessible technologies (OpenApi, graphQL) which are de facto standards.

Conclusion:

All the actors in the consortium agree that compared to our goal, our willingness to allow flexible cooperation between actors on all issues where it makes sense, we want to move towards the use of the API standard SPARQL to ensure the flexibility of use cases and to use the power of the semantic web. The development in addition of a second API based on OpenAPI would make it easier to reuse and guarantees easy access to the standard. This vision is of course coupled with the vision of a high granularity services architecture, necessary to use the power of the semantic web.

They warn nevertheless about the complexity of implementation of high granularity services and the risk of slowing the project if we started on this option for the prototype.

They are also realistic about a low skill level by the current teams of the SPARQL API standard.

The API of the DFC standard will be "REST resource driven semantics". This means HTTP requests (a URL per usage / resource) documented by writings and in OpenAPI that return the Json-LD. This contains the OWL semantic model and not the model under the Open-API standard. The API can return multiple nested resources without respecting the high granularity service logic.

Platforms can use the implementation they want including solid servers but must expose the API of the standard.

Eventually, a cache server will increase search performance and use SPARQL and / or GraphQL for this. This cache will also expose the API of the standard.

iv. Serialization: JSON-LD vs XML-RDF vs TTL …

In our case data serialization is the process of converting the semantic data to a format that allows sharing it in a form that then allows recovery of its original structure.
Historically the standard used was RDF (XML), which has evolved to TTL which simplifies the syntax. The last evolution is JSON-LD, which is like JSON but expresses a context which allows an equivalence with RDF or TTL, but which is not serialized on files. TTL simplifies RDF, while JSON-LD offers another radically different way of doing things using the most common standard (JSON).
When we enter the semantic world, we can have trouble reading RDF or TTL, while JSON-LD is easy to read. JSON-LD is the standard that has been chosen by Google and Facebook in their journey towards the semantic web.

Conclusion: the consortium's vision is therefore that we must adopt the JSON-LD standard.

The consortium, however, anticipates that this choice will require a bit of thinking around the server implementing, because standard SOLID servers return semantic data in RDF or TTL. We could also contribute to the global common good so that SOLID servers can provide JSON-LD.

v. Transport layer

HTTP has become the unquestioned standard to exchange data unidirectionally.
FTP is still relevant to exchange files containing data like ods or xlsx values but is not adapted to our needs.
HTTP can be improved with the REST logic. This good practice makes the best use of the HTTP standard to harmonize APIs.

In the semantic world, another standard complements the HTTP: Linked Data Platform. This standardizes the structure of the requested semantic content. It is based on 2 concepts: containers and resources. Containers contain resources. A resource can reference containers that contain data linked to it. A resource can itself be a container.
Some contradictions exist between REST and LDP. The header link for example is used to indicate the links that a resource has with another resource in REST while it is used to indicate what is the nature of the resource (Resource, BasicContainer, DirectContainer ...) in LDP.
JSON-LD is compatible with LDP provided that it complies with the Resource / Container logic.
LDP does not impose a form of url and OpenApi remains relevant to describe the API.

Conclusion: use HTTP and LDP while following a REST logic as much as possible

vi. Directionality

One key factors to differentiate between protocols is their directionality:

- **Unidirectional** = a client requests from a server and the server responds. The server cannot initiate the communication.
- **Bi-directional** = the client and the server can send messages without being requested.

The advantage of bidirectionality is that one can also adopt a push mode, where the server sends data to the client. For example regarding inventory: a product on platform A is purchased, but this product is also sold on platform B, platform A could "push" the information that the available stock has changed to platform B.
The need to have uni- or bi-directionality is therefore related to the need to have data synchronization and notifications in real time. An alternative in the example above using unidirectionality could be that servers send pull requests, every minute for example, to know if the stock has changed. **Is it sufficient ? Do we want real time or near real time to 1 or 5 minutes?**

So far the protocol used in the semantic web is rather unidirectional, HTTP or LDP. It can be a simple request about a resource (file: LDP or base of triplestore: HTTP) or more complete queries with Sparql.

There is today very little R&D on bidirectionality in the semantic world, it is still very experimental (AMQP + SPARQL, XMPP + SPARQL, HTTP2 + SPARQL, WebSocket). The terrain is slippery, not yet mature. The theoretical ideal would of course be bi-directional, **but the rational and practical solution of the consortium is rather to use unidirectionality**. We prefer to focus our vision towards new and proven standards and technologies. We will see later when technologies will mature if it makes sense to switch to bidirectional.

Conclusion: implement a unidirectional solution

vii. Multi- or single-resource requests?

The ability to make multi-resource requests (a query that looks for information across multiple files) is related to using a TripleStore to store information, not just files. The TripleStore is then the aggregation of all the triplets of all the files. We can not make sparql multi-resource requests today in the SOLID server implemented by the MIT (node solid server) but it is possible on a JENA server. JENA does not implement authentication and rights management solutions, and these are the points that SOLID offers a solution for. In the SOLID standard, SPARQL multi-resources is planned, but not yet implemented.

The need for multi-resource queries will influence the choice of server (next point).

In our vision, and in the continuity of the architectural vision described above, to be able to carry out research on a set of resources (eg a search on all the catalogs or all the logistical needs) we need to ability to make multi-resource SPARQL queries.
We do not yet know how to put it in place, but we know in our vision that we need to be able to make multi-resource requests.

Conclusion: we need to be able to do multi-resources requests, but it is unclear how to achieve this

viii. Identification and authentication

Rights management covers the issues of authentication, identification, and rights management granted by an agent to another agent.

- **Identification** = we know who is the user trying to query the data. **Who we are**.
- **Authentication** = a technical way to check that the person is who he pretends to be and not a hacker. **It's the validation of who we are**.
- **Authorization** = **what data and functionality we have access to**.

In our case we need a way to perform these steps in an universal way and using the same credentials across platform, aka SSO. We need it because if we take the case of the prototype, a producer who wants to visualize his overall catalog will have to request data from all the platforms he uses. Without SSO, he would have to connect from the prototype to each platform specifically which in terms of UX is not satisfactory.

Who performs these steps? Is there a single or several servers that know who is who and how to verify it? This is the difference between centralized and decentralized identification and authentication.

1- Centralized, it is possible to use a secure access to the server that hosts the data as a certificate (private key, public key). This is not the case for this project.

2- Delegation to a centralized entity. Protocols exist that enable multiple servers to perform identification/authentication, sharing a common OIDC or OAuth or SAML standard. This solution involves technical difficulties to become a reliable and trusted server. If an unreliable server is used, it becomes a risk to the entire system. The most known current servers of identification and authentication are, Google, Facebook, Github, the French State in France. They are recognized as authentication providers and used by many sites.
This is the famous "connect with Facebook".

3- **Decentralized**, we can move towards fully distributed identification and authentication. The DID specification is a standard on the subject. We can identify and authenticate ourselves on any SOLID server, thanks to a solution combining webID and OIDC: Webid-OIDC. It is an associated identification (Webid) and authentication (OIDC) protocol. The WebId carries the url of the server on which to make the OIDC. Webid-OIDC does not claim to respect DID.

But how to match producer accounts from different platforms if the system is totally decentralized? The reality is that today producers have one account per platform.
In the spirit of SOLID, a producer does not have accounts on two platforms. His account is on a single SOLID server. eg if the producer has a SOLID server, he can create his identity on his own SOLID server, and it is valid on the other servers too. Other servers will ask his server "Do you authenticate this person?"

If you are connected to a SOLID server, you can connect to any other SOLID server. But how do we know if this server is in the community or not?
It's a matter of ergonomics, UX: The user will have a list of servers validated by DFC for example on which he chooses to identify himself / authenticate. When we arrive in the application, unlike a traditional login with password, we ask with which server we identify. Like Oauth! But with Solid any server can be server of reference, it is possible to propose to the producer to identify with any of the platforms in the network. **A new server / authentication node means a manual addition to the list of servers on which authentication is possible - this is a political decision to be made by the consortium. The maintenance of the list of servers can be automated by crawling.**

If we want to decentralize authentication via federation, it means that an actor who does not wish to federate on authentication could not join. It's easy to validate the authentication once in the federation, but it's hard to enter it. Others have to trust the trustworthiness of the new entrant, because if he mistakenly authenticates himself, he poses a risk to the entire federation. The choice to decentralize the identification and authentication would imply the choice of the use of SOLID servers or an equivalent that manages a DID protocol, and thus the "transfer" of a certain amount of information from the platforms to their SOLID server or equivalent. Knowing that today producers already have multiple accounts on multiple platforms it would mean that they are left with several unique identifiers unless they are forced re-create an account in the new system and migrate the data.

Conclusion: the vision shared by the consortium is towards decentralized identification, authentication and authorization, via federated SOLID servers. But there is still uncertainty about how this can be implemented. In particular, decentralized right management require to have a consistent autorisation semantics across platforms, which in itself is a significant work. **We will use OIDC for the prototype.**

ix. Right delegation between platforms and DFC

In the prototype, there is no request for consent, but we can clearly indicate what data will be used just for information.

For the prototype, we will use the OIDC server managed by the commons.

By choosing OIDC centralized authentication / identification, the user consents to DFC having access to their data when a DFC account is created and linking it to their platform account. **The right management is decentralized on the platforms that can implement their own system.**

On the long term WACL (Web Access Control) should be the ideal solution for platforms that choose SOLID servers.

x. Centralized or decentralized data storage

Do we want the data we store to be on multiple servers, or on only one?

- Federation, all actor must respect the SOLID standard, each having a SOLID in front, that integrated with his servers on the data to be shared.
  Advantage = as we all use the same standard, it simplifies the distributed storage of data on all SOLID servers in the federation.
- Syndication, a translator is needed between the technology of the actors and the common technology. This option respects more the identity of each one, the actors are not obliged to use the same storage technology. Both are complementary and compatible.

Our vision pushes us to favor decentralization, so to move towards a logic of federation, with a semantic server per actor. But we defend the freedom of choice of each actor, so the vision is a hybrid model of federation and syndication for those who do not wish to adopt a standardized semantic server. This also allows for a much easier migration from the existing state, with multiple different technologies, to a federated one where all use SOLID server.

Conclusion: it seems that we need a centralized server for 3 uses

- ID repositories

    - Users
        - OIDC server of the commons
        - Depending on SOLID's technological evolutions, this repository could be decentralized thanks to web-id/OIDC
    - Products
        - Open Food Facts
    - Places
        - To define
- The semantic cache
    - To guarantee the performances and to be able to make queries (SPARQL, GraphQL) which cover the data of all the platforms
- The Open Food Facts Taxonomy Not Open-Data
    - For producers to map their catalog based on the Open Food Facts taxonomy. This mapping from the original taxonomy of the platforms to the Open Food Facts taxonomy has to be done once manually and can be automated after, which should facilitate the adoption of DFC.

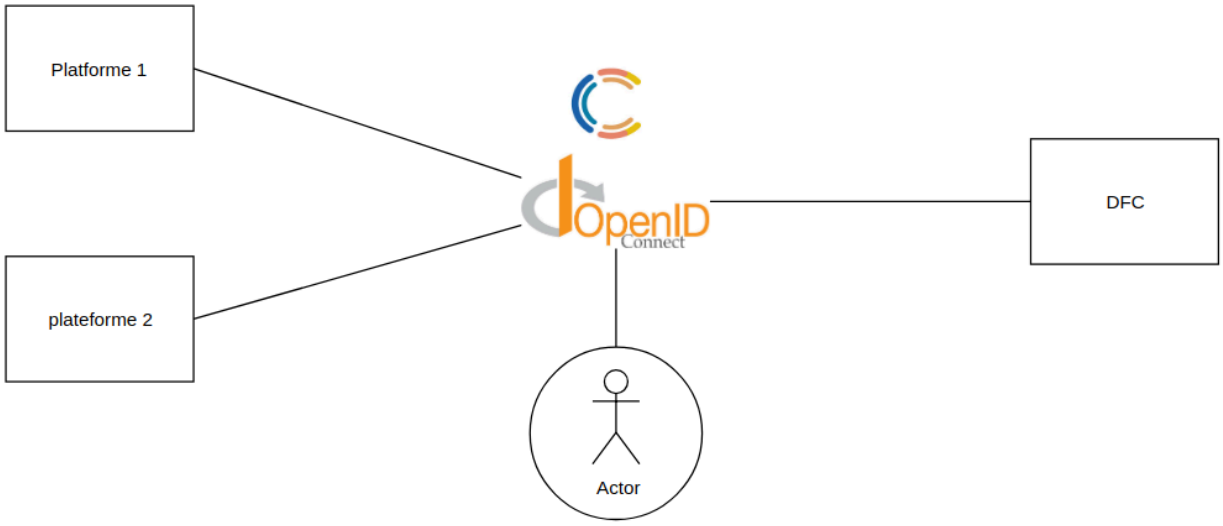
xi. Metadata repository

1. Users

To interoperate data between platforms, we need to know that user X on platform 1 is user Y on platform 2, that they are the same agent. For that we need either a unique identifier for users on top of a way to authenticate them. The user repository is essential here. It usually includes for each user a login, a password (hashed of course), an ID, and information about the user.

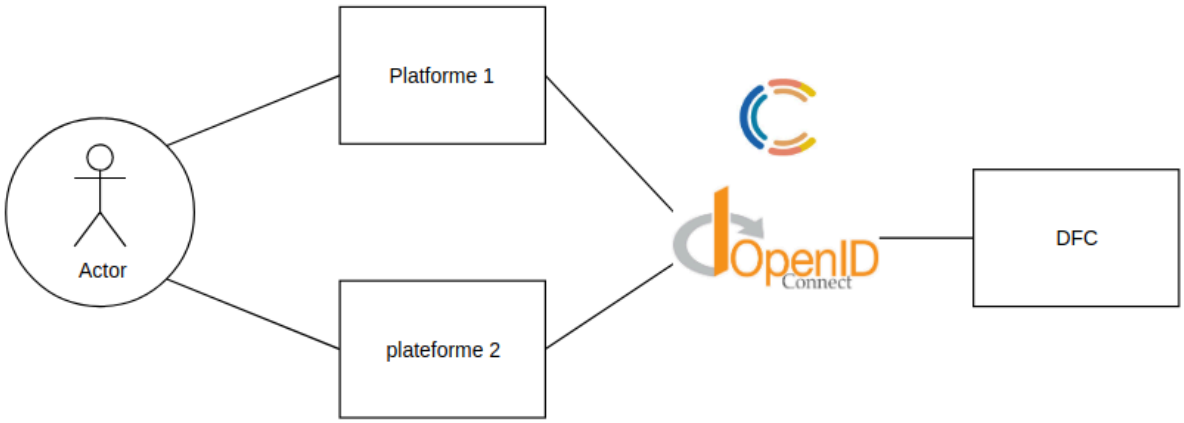
Centralized repository

One ID per user



- Principle: All platforms allow connection with the OIDC server of the commons that users perceive like the DFC server. A platform user must have associated their account with an OIDC common account to take advantage of DFC features.
- Advantage: One OIDC account for the user. Simplicity architecture and implementation for DFC. Users can continue without the DFC OIDC account but will not be able to enjoy the associated features.
- Disadvantage: Platforms need to perform developments to be able to recognize the OIDC server of the commons as a means of identification / authentication and allow existing users to associate their existing account with a DFC OIDC account

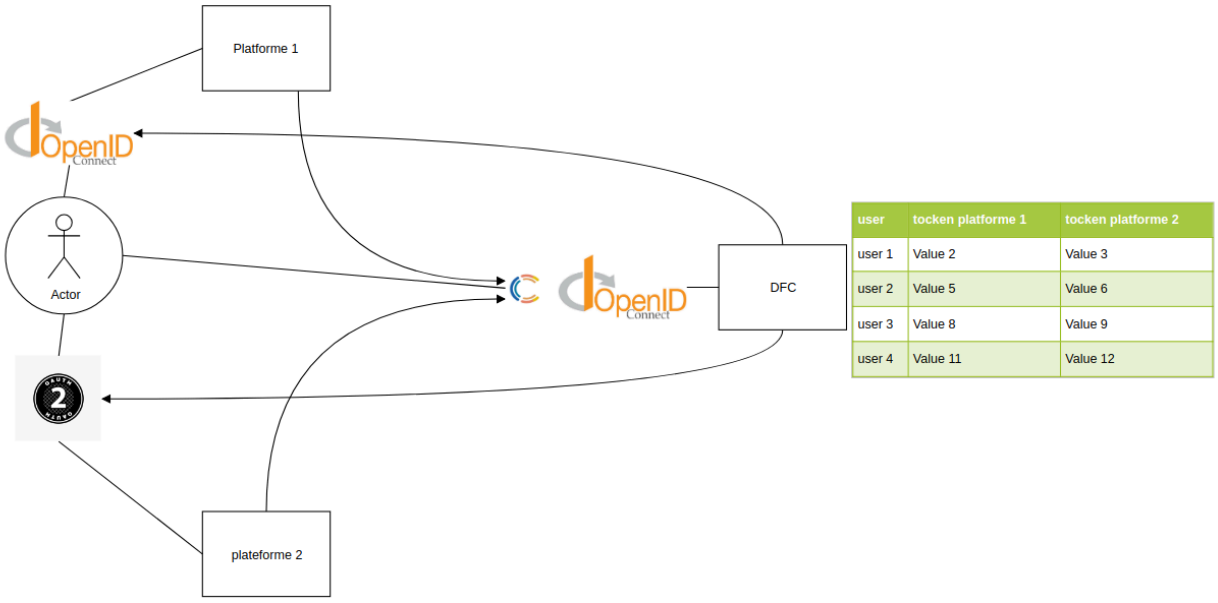
One ID per platform



- Principle: The platforms (including DFC) each have an identifier on the oidc server of DFC.
- Advantage: Simple architecture and implementation for DFC. No need to ask users to associate their account with the OIDC DFC server. Platforms can communicate with each other without DFC. transparent for users.
- Disadvantage: It is not possible to differentiate rights by incoming users. A platform 1 grants a set of rights to the platform 2 for example without user distinctions. It is possible, however, that the user specifies on platform B what is entitled to see the platform A. The platforms need to realize developments to be able to validate the identification / authentication OIDC

Decentralized repository

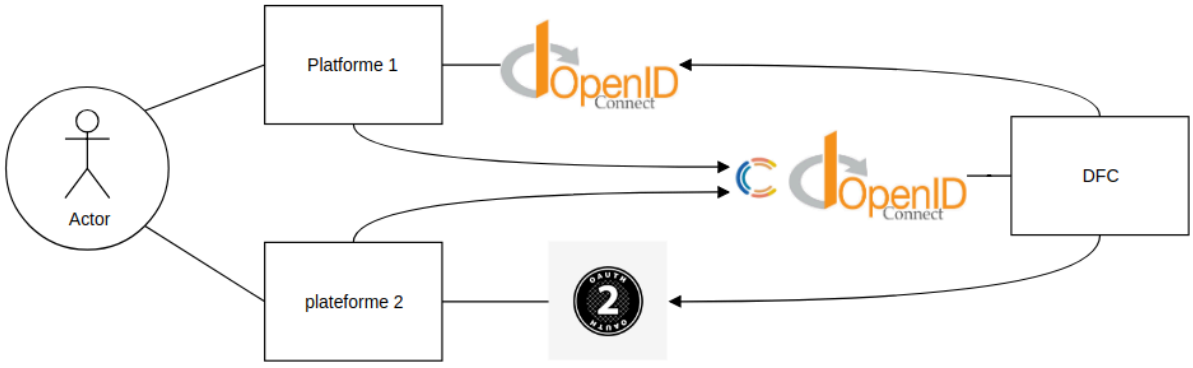
One ID per user



user	tocken platforme 1	tocken platforme 2
user 1	Value 2	Value 3
user 2	Value 5	Value 6
user 3	Value 8	Value 9
user 4	Value 11	Value 12

- Principle: Each platform and DFC have an authentication server. The user must inform DFC about his access to his own data (user, password token ...). DFC can then mediate and platform 1 can request data from platform 2 through DFC as it is the only place where the mapping between the 2 accounts is specified. The identification and authentication protocols may be different between platforms but this will increase the complexity of the DFC code.
- Advantage: Platforms do not need to reimplement / improve their authentication / authentication
- Disadvantage: multiple accounts for the user. Complex code DFC side

One ID per platform



- Principle: The platforms (including DFC) each have an identifier on each platform
- Benefit: No need to ask users to associate their account with the OIDC DFC server. The platforms can communicate with each other without DFC but with a technical cost (see disadvantage). transparent for users.

- Disadvantage: It is not possible to differentiate rights by incoming users. A platform 1 grants a set of rights to the platform 2 for example without user distinctions. However, it is possible for the user to specify on platform B what is entitled to see platform A. Code relatively complex DFC side but less than solution 1. In the case of direct access (without DFC), if a new platform arrives in the ecosystem: the new platform will have to configure all the protocols / token of the others and the others will have to add the protocol / token of the new one.

Conclusion:

- **Short term**
  The platforms are ready to make an effort to integrate an OIDC authentication into their platform and allow their users to complete their platform-specific authentication via OIDC. **So we chose a centralized user repository with one user ID.**
  The chosen OIDC server is that of the collective "Les Communs (the commons)". This group is a legitimate leader in strategic thinking for the commons and the provision of IT tools to enable the commons to organize. This server will then be our unique source of user universal identifiers. To start with and identify matching users on Open Food Facts, as Open Food Facts is not going to complete their platform authentication via OIDC, we will use SIRET to identify the corresponding users and user their data. It is possible because all data of Open Food Fact are Open-Data and don't need authentication.
- **Long term**: we hope that web-id / OIDC or other DID protocol will mature and that we will be able to rely on it for decentralized authentication. That would enable us to manage those unique universal identifiers in a decentralized way. It would also enable to manage agent characterization information, that are not managed through OIDC (only basic info like login and password are currently managed). If we need later on to manage conflicts between agent description facets and decide which info is right, we might need also some shared database to store those universal conflict-free / trusted information.

>>> *Access the source for the drawings*

2. Products

To be able to determine that two products on two platforms are the same, it is essential to have a common identifier.

Decentralized repository

If several servers are able to manage product identifiers, this means that the product must only be created on one platform. If this product is to be used by a second platform, it must refer to the identifier created on the first platform. This requires an infrastructure capable of

crowdsourcing different sources and is difficult to do without a semantic web. The semantic web is based on the uri that contains the address of the server on which the initial resource was created. Other servers can manage information about this product but the identifier remains linked to the original server.
It is also possible to issue ID ranges per platform. This is the logic of GS1 but it means that it is not possible for a platform to create an identifier without having previously requested its range from a centralized entity that realizes the assignment of the identification ranges. It is therefore a false decentralization because there is a centralized entity from which all the entities of the consortium depend.

Centralized repository

It is easier to entrust to a single entity the creation of product identifiers. This does not prevent each platform from managing its own internal identifiers, but platform products will not be interoperable with other platforms unless they are linked to a central repository identifier. This means that there is a need for a simple and ergonomic solution so that platform users can easily link their products that they manage on a platform with a centralized identifier. When choosing this trusted third party, it is important to make an alliance with a third party that already has a legitimate leading position and is able to generate identifiers recognized by all market players. Open Food Fact is able to reference GS1 managed EANs for packaged products and is also able to generate identifiers in a reserved range (let's call them "pseudo EAN"). They have a healthy governance and in line with the values of DFC. They have a leading position in the field of factual information on food products.

Conclusion:

- Short term: given the pre-existence of several platforms that already have their product identifiers, it seems complex and inconsistent to start with decentralized identifiers. The most consistent strategy is to use Open Food Fact IDs and generate one for products that do not have an API. We pay great attention to ergonomics so that platform users have the least possible difficulty in associating the identifier of their platform with an Open Food Facts ID.
- Long term: moving to completely decentralized management of product identifiers would be possible thanks to the Semantic Web and Solid but it requires a complex migration work and a difficult strategy to identify. It is therefore a solution that remains in the vision but still requires a lot of research and development.

Operational implication

DFC would have a specific universal products catalog, in a separate database from Open Food Facts, hosted by a trusted third party (Les Communs in France). This will enable DFC to not force users to open their products data, as any info on Open Food Facts is open data. When a product is imported on DFC catalog, we would check (using enterprise ID SIRET first, then hopefully only OIDC) if the user already have corresponding products in Open Food Facts, and in that case, match the product and use the corresponding existing id (EAN or pseudo EAN). If no matching product is found, and the product doesn't have an EAN, we would use the Open

Food Facts API to generate a new pseudo EAN. Products on the DFC catalog will be easy to push to Open Food Facts whenever a user wants to open their data.
On a first step the mapping between products will happen through the prototype, which will send back universal id information to the original platform. On a second step, each platform might want to integrate on their own interface a module to link /create a given product to it's universal DFC entry.

3. Places

Platforms reference places, but there can be cases where a same place is not referenced the same way in two platforms, so mapping them to GPS coordinates might not give the same coordinates even if it's the same place. As for users and products, we need a way to know that a place in platform A is the same place in platform B. Especially when we move on in the development of the prototype, step 2 and 3, we will need to be able to identify places, so we can search places within x km distance of departure or arrival.
We could at some point, like users and products, have ids for places. Open Street Map could be a good candidate to manage universal location ids, but it seems it is not the case for the moment, as they reallocate ids when places are delete. Also some addresses are not recognized today by Open Street Map. GS1 did work on the topic as well, we would need to investigate better their solution.
This problem has not fully been investigated yet.

e. Appendix 2. General principles.

i. Federation vs syndication

The nuance between syndication and federation applies to all the notions where several servers jointly concur to achieve an objective.

- Federation = all servers respect the same protocol. This facilitates interaction between the actors of the federation. But if all aspects of the protocol are federated (including identification and authentication) it can make access to the federation more complicated because to enter it, one must implement the same protocol and often migrate to common servers / solutions.
- Syndication = each actor has his own protocol, which is translated into the common protocol to allow interaction. Moving the complexity to the integration layer.

The architecture may also be an hybrid with federation in some limited areas, for example on the semantic standard, the unique identifiers and the authentication, and syndication otherwise.

ii. Libraries to develop in semantic

A question was raised about the existence of mature libraries in different programming languages to manage semantic data in SPARQL. Are there open-source libraries and code sample in the different languages to query and process web-semantics? As it can exist for OpenAPI where there are libraries that can generate code samples.

There would be libraries in almost all languages, but these are more or less mature. In javascript in particular, the standardization is taking place, RDF-JS.. The situation is that with SPARQL it is more difficult that with OpenAPI or Graphql, who invest a lot of money for the community, and thus also finance the libraries that will interact with these standards. The libraries around SPARQL are more "fresh paint", less documented, more difficult to setup. **There is a risk, and to check by each of the actors, so that it does not generate too much pain.**

iii. Transition strategy from current to ideal

Base principle: Improve the legacy rather than migrate to new technology

High vs Low granularity services

The reality of the platforms around the consortium today is that they are absolutely not designed for high granularity services. At best, as some low granularity services connected to each other via API. If each platform were to implement a standard based on a high granularity services architecture, each platform would have to re-assemble these high granularity services in low-granularity corresponding to their own low granularity services to be able to share their services. data via API ... Re-develop in their API a kind of "connector" between their services and high granularity services of the standard.

So the approach for the prototype is rather not to go to a 100% high granularity service state, but rather only on some relevant perimeter, without pushing the high granularity service logic to the extreme, but granular enough so that the API is simple to use. Depending on the potential of reusability of the service by others. For example, separating the product and inventory information is quite clear.

To try to go as granular as possible, we can use as soon as necessary an aggregation bus (like the semantic bus developed by Simon), for example to switch from several high granularity services to a low granularity service, according to the respective configurations of each platform.

Semantic web

A semantic approach in high granularity service would also require platforms to transform their database by a BDD TripleStore for example and that it is not the purpose of DFC to impose it. Also, developers do not master SPARQL today

Conclusion: the actors of the consortium finally preferred to start by first developing a user friendly API using OpenAPI, in JSON-LD.

Of course, we keep in mind the goal of moving towards a fully semantic and professional API when the DFC project budget has been secured. This approach being a little less experimental, it will allow us to connect 2 data sources to the prototype. This will require us to code existing API transformations into "REST resource driven semantic" APIs that respect the DFC ontology.

G. Access sources

a. Usage licences

b. Github repositories

c. Meeting notes

Notes:
https://docs.google.com/document/d/17ADh0ygn5soLPDB4MPDOptEhW4_pPtOb79WFQVczBL8/edit
PPT:
https://docs.google.com/presentation/d/1wqUUbnY_HX8G0r7MpS6W_gFt-kewiGpoxnr5A8xoInE/edit#slide=id.g470b61b977_0_0

H. Contact us