IDS Architecture Implementation using FIWARE

Acknowledgements

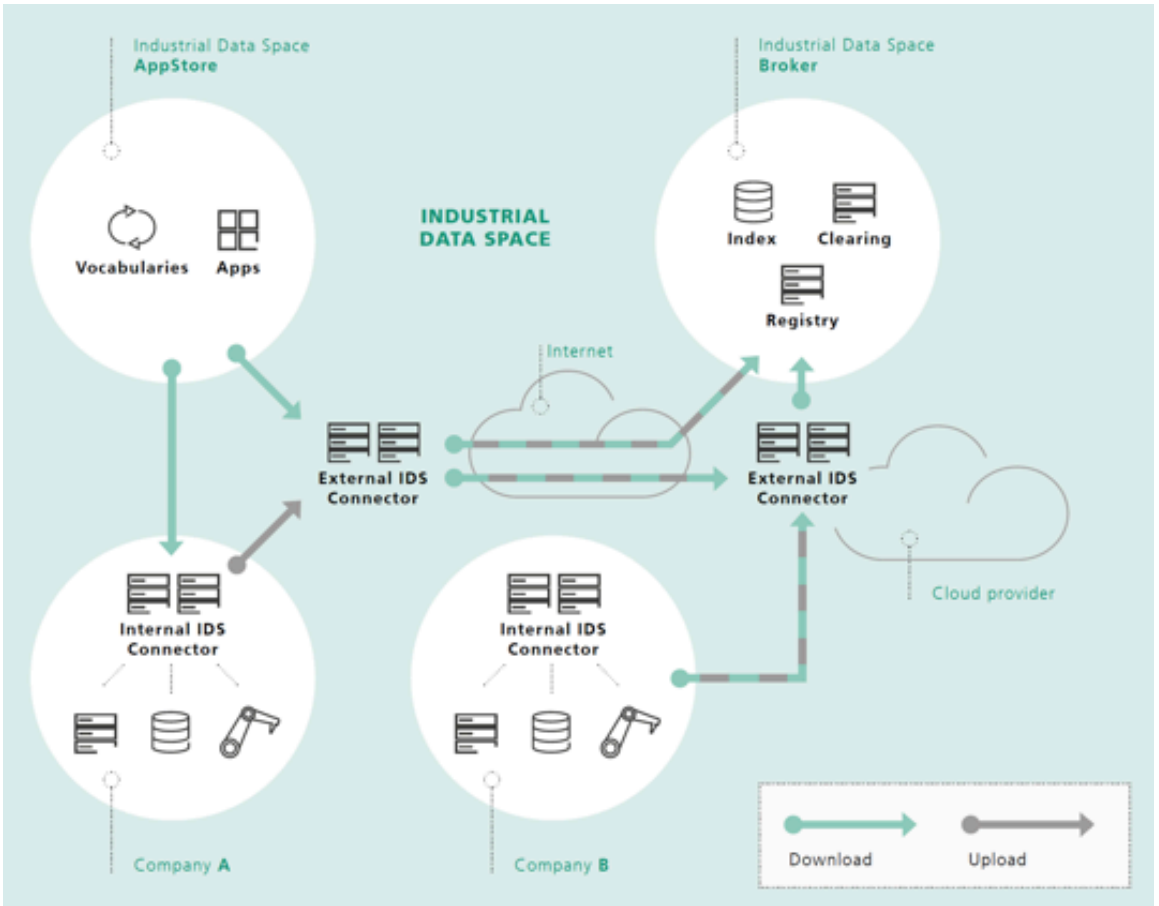
<This section will include references to people who has contributed to edition of the document and their role within the FIWARE OS Community>

Introduction

Basic IDS Architecture concepts

<Come with basic description of IDS Architecture concepts but without getting into too much details since text should incorporate a reference to the Reference Architecture Model for the Industrial Data Space>.

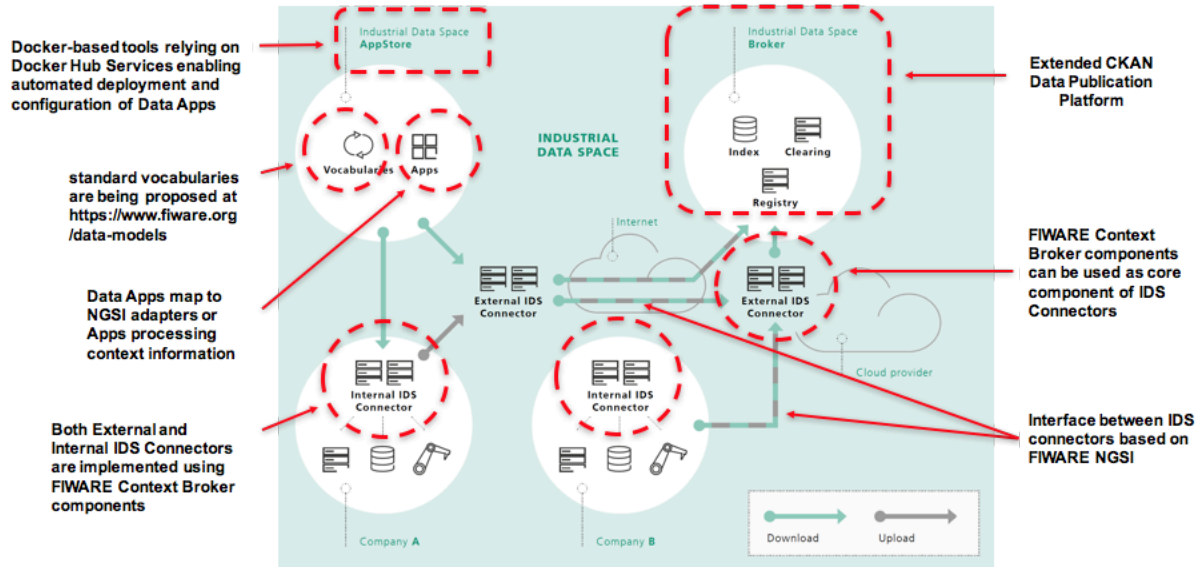
<A picture like the following describing the main IDS architecture elements will be included>



High-level mapping of IDS Architecture elements

<This section will provide a high-level description of how of IDS Architecture elements can be implemented using FIWARE technologies>

<A picture like the following will be included>

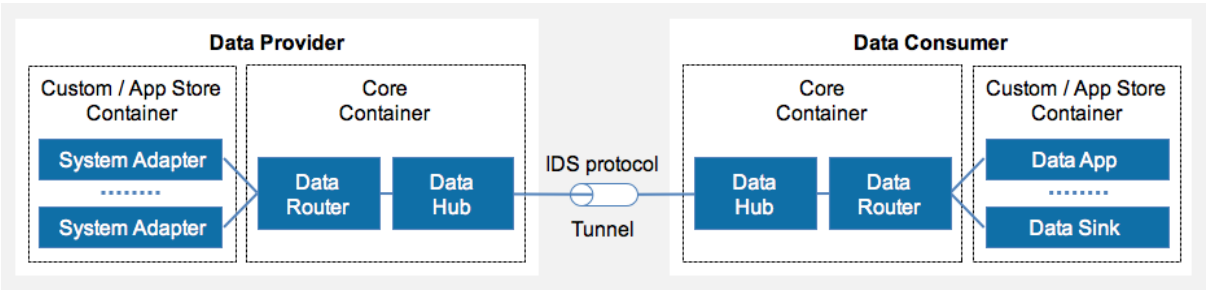


Implementation of IDS Architecture elements

IDS Connector

IDS Connector overall architecture

<First, we will elaborate on main elements of an IDS Connector based on IDS specifications, including a picture like the following (derived from IDS specifications)>



The picture above is derived from the IDS specifications but uses the terms “Data Router” and “Data Hub” instead of “Message Router” and “Message Bus” to avoid that a mapping with Message middleware is somehow suggested. “Data Hub” is kind of a more neutral term which may map into multiple implementations, one of them being the FIWARE Context Broker.

<Following is text taken from the IDS spec describing the picture displayed above in which we provide some suggested edits (to be submitted to IDSA under formal review of the specs>

The Message Bus stores data between services or Connectors. Usually, the Message Bus provides the simplest method to exchange data between Connectors. Like the Message Router, the Message Bus can be replaced by alternative implementations in order to meet the requirements of the operator. The selection of an appropriate Message Bus may depend on various aspects (e.g., costs, level of support, throughput rate, quality of documentation, or availability of accessories).

<We then shall introduce a picture which described our proposed IDS Connector architecture using FIWARE components. We shall cover the ability to deploy system adapters and Data Sinks>

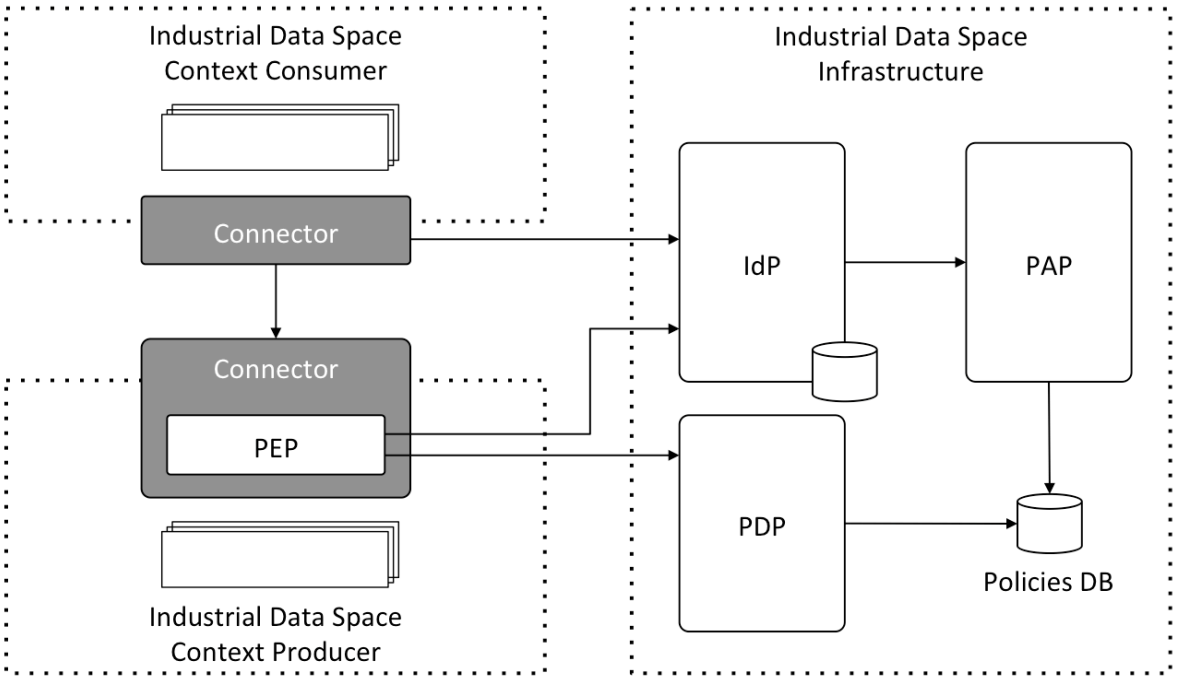
<We have to further elaborate the pictures in this slide set>

Basic Access Control

<Incorporate description of how Access Control works, the role of API umbrella, how connection to global IdM and PDP servers will work as well as generation of CDRs will be created>

According to IDS Architecture Model, to be able to make access control related decisions based on reliable identities and properties of participants, a concept for Identity and Access Management (IAM) is mandatory. This includes identification (i.e. claiming an identity), authentication (i.e. verifying the identity) and authorization (i.e. making access decisions based on an identity). Every participant may possess attributes apart of its identity, and these attributes could vary dynamically. Thus, Connectors regulate access to data basing on different criteria: the specific identity of Connectors, Connectors attributes or security profile requirements.

The following figure shows the proposed architecture to achieve the Identity and Access Management IDS requirements using FIWARE components.



The proposed model uses OAuth 2.0 protocol as authorization framework. Using OAuth 2.0 delegates the authorization process saving load in IoT devices and enables an application-scoped feature. Thanks to this feature, Access Control policies can be defined in the scope of an application/service. By fostering this behavior, a participant could have different permissions for different services, thus making it possible for the same participant to be reused among different services, being under different security conditions in each of them.

In the Figure we can see and IDS Context Consumer that accesses data provided by an IDS Context Producer. Of course, the communication between them is established through their respective IDS Connectors and using a secure channel. For protecting the access to the provided resources, every request is intercepted by a Policy Enforcement Point (PEP). On the other hand, in the IDS global infrastructure are deployed the rest of components that take part in the architecture. These components are deployed once and used by every IDS Connector in the environment. The Policy Administration Point (PAP) and the Policy Decision Point (PDP), together with the set of PEPs included in each Connector compose the widely-known Access Control architecture. The PAP stores the defined access control policies in the Policies DB, where PDP checks them at decision time. Finally, the Identity Provider (IdP) is in charge of identification and authentication.

In order for the Access Control architecture to be used in all security contexts, the framework for describing authorization policies should fit a level of granularity and flexibility. For this purpose, OASIS (a global nonprofit consortium that works on the standards definition for security, IoT and other areas) standardized the eXtensible Access Control Markup Language (XACML), which allows the definition of fine-grained policies. XACML serves as a standard not only for the format of authorization policies and evaluation logic, but also for that of the request/response interactions that take place during an authorization decision.

Following the XACML terminology, policies are composed by a set of rules (as well as other items that are out of the scope of this paper). Rules are in turn made of a target (e.g., the resource), an effect (e.g., allow/deny) and a condition. In our approach, rules are used to

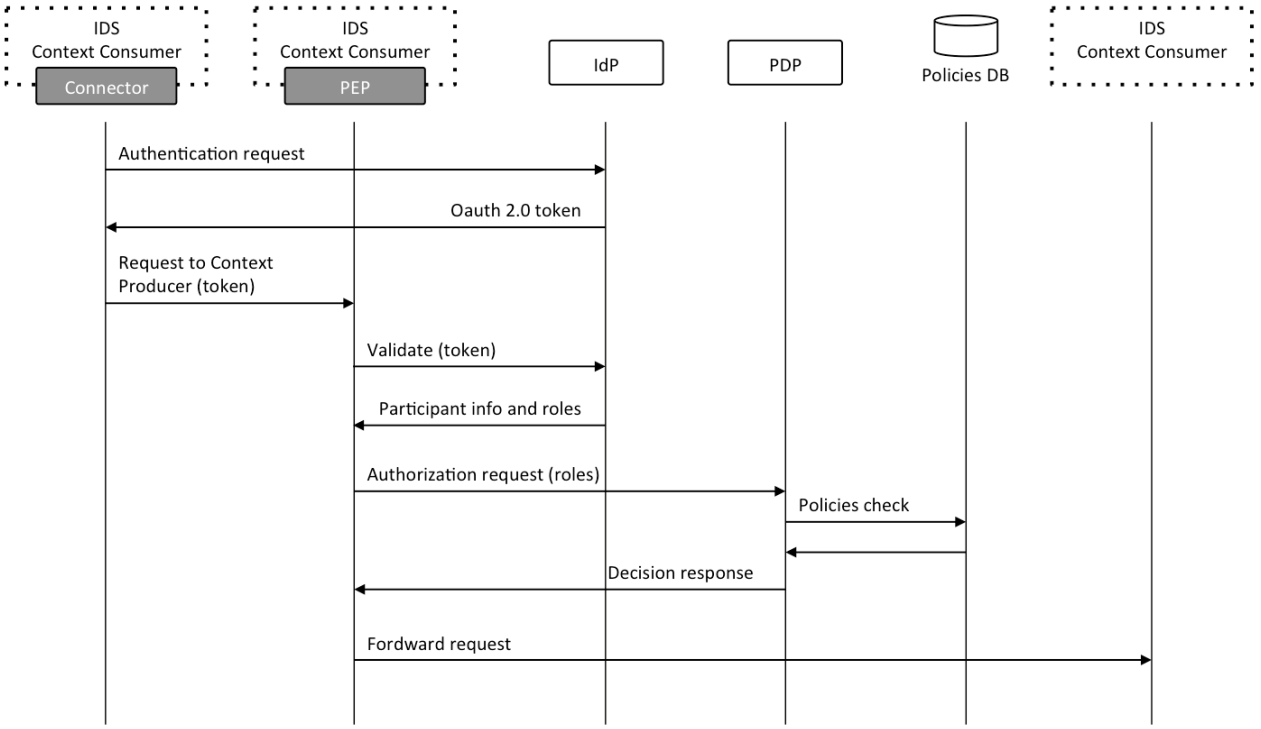
implement permissions, in which the target is an action (e.g., an HTTP verb) plus a resource of the Context Producer. Of course, more complex policies may be also defined, provided that they are supported by the policy-description language. Roles are in turn sets of Permissions that serve as a container so that more than one permission can be assigned at once. Both Permissions, Roles and their relationships are defined in the PAP. Note that this approach allows both a simpler Role-Based Access Control (RBAC) scheme and a more complex, more flexible Attribute-Based Access Control (ABAC) scheme. It all depends on how the Permissions are defined when creating the XACML rules.

As introduced above, the Identity Provider provides identification and authentication following an Identity as a Service (IDaaS) approach. Thus, every IDS participant need to be registered in the IdP, so that it gets a set of credentials (usually a username and a password), which it can use to authenticate and identify itself against the AC system. Groups of participants can be created in the IdP, to allow more complex authorization scenarios. Granting a permission to a group of participants, rather than to a single one, has the benefit of linking the given permission to the participants belonging to the group.

Once a participant is registered in the IdP, it can create an OAuth 2.0 access token for accessing data in an specific Context Producer. In OAuth 2.0 terminology that means creating a token in the scope of a consumer. This token represents the participant in the system and has to be included in every request sent to other Connector. As outlined before, these requests are intercepted by the PEP, that extracts the participant's access token and validates it with the IdP. This validation can be performed in three levels of security:

1. Authentication: Using this level of security, the PEP just checks if the participant has been correctly authenticated against the IdP. Thus, at this level, every participant with an active account would be able to access the protected data. The check is performed by sending a validation request to the IdP.
2. Basic authorization: In this case, the PEP also checks if the participant has the required roles to perform the corresponding action (defined by an HTTP verb) in the corresponding data source (defined by an HTTP path). After the first check with the IdP, the PEP obtains the roles the participant has assigned in the scope of the Context Producer where the token was created. Once roles have been retrieved, the authorization check is sent to the PDP. PDP fetches the policies associated with the participant's roles from the Policies DB and decides whether or not access should be granted based on them.
3. Advanced authorization: This is the most complex, powerful case, because the authorization check is not only based on the HTTP verb and path, but also on other more advanced, customizable parameters, such as the request body or headers. To perform the check, a custom XACML policy request is sent to the PDP.

Next figure illustrates the interaction described below:



Every component present in this architecture is implemented as a FIWARE Generic Enabler:

- Identity Provider: the Identity Management GE is named KeyRock (https://catalogue.fiware.org/enablers/identity-management-keyrock) and has been implemented using Openstack technology as a starting point. KeyRock has two main components, named after their Openstack's counterparts: Python-based, back-end Keystone and Django-based, front-end Horizon. Both components have been extended for supporting the specific features needed by the IDS Security Architecture.
- Policy Administration and Decision Points: this implementation integrates both PAP and PDP components and is called AuthZforce (https://catalogue.fiware.org/enablers/authorization-pdp-authzforce). It provides an API to get authorization decisions based on authorization policies and authorization requests from PEPs. The API follows the REST architecture style and complies with XACML v3.0.
- Policy Enforcement Point: the policy enforcement point GE is named Wilma (https://catalogue.fiware.org/enablers/pep-proxy-wilma) and is developed using Node.js. In the context of IDS FIWARE Architecture, Wilma is deployed as part of API Umbrella, an API Management system that enriches the PEP functionalities with features like accounting, API documentation or catching.

Automated deployment of components in IDS Connectors

General vision

Deployment base IDS Connector components

<translate description given in minutes of kick-off f2f meeting>

Deployment of System Adapters: IoT Agents

<to be discussed and description incorporated>

Deployment of System Adapters: XapiX-based Agents

<to be discussed and description incorporated>

Deployment of Connectors to Data Sinks

<to be discussed and description incorporated>

Management of subscriptions and registration of Context Providers

<translate description given in minutes of kick-off f2f meeting>

For getting data from a Context Provider, a Context Consumer can proceed basically in two ways. It can send a request each time it wants to get the data or it can subscribe to a specific data field to periodically receive its updates.

When using the first approach, the query is intercepted by the PEP in API Umbrella and the pertinent accounting (for billing, statistics production, etc) is performed. However, if the Context Consumer subscribes to a data source, when the Context Provider sends an update, the request is sent directly to the consumer and the accounting is not performed.

To solve this problem, a *Second level proxy* (2lp in advance) is deployed between API Umbrella and the Context Provider. The 2lp acts as follows:

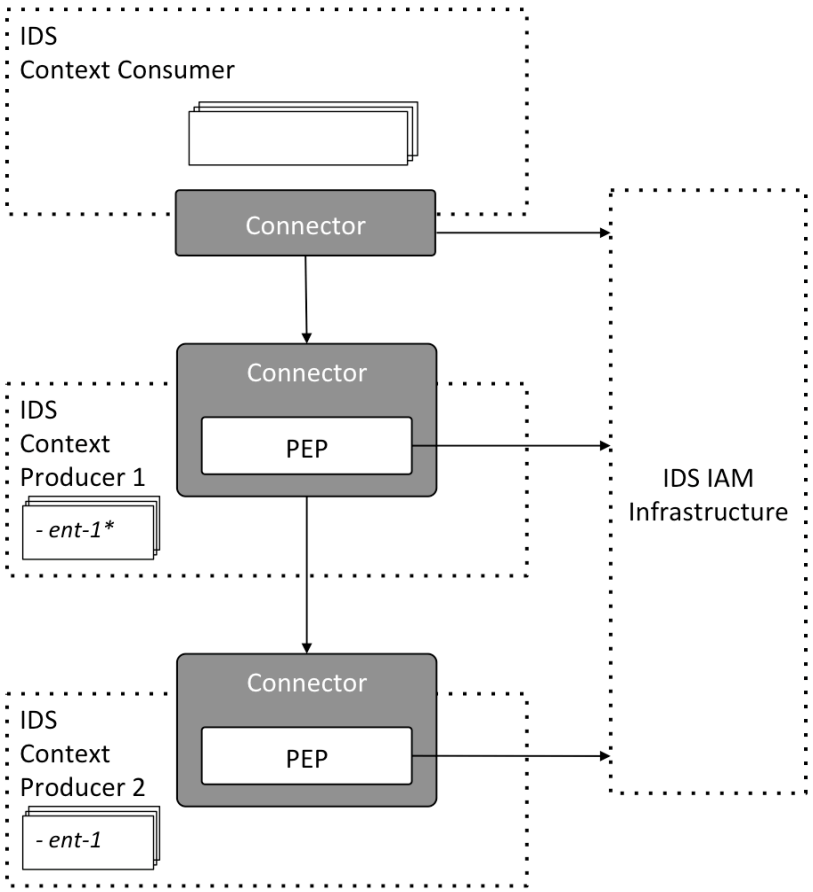
- When configuring a Context Provider as a backend in API Umbrella, 2lp endpoint has to be used instead of the real Context Provider endpoint
- 2lp intercepts every request from API Umbrella to the Context Provider. If the request is different than a subscription request it just redirects it to the Context Provider. If the request is a subscription request to the endpoint http://endpoint:port/path
    - 2lp creates an identifier for the subscription (subs_id)
    - 2lp modifies the content of the request changing the subscription endpoint to http://api_umbrella/subs_id/path
    - 2lp calls API Umbrella API to register a new backend associating requests to path /subs_id/* to the endpoint http://endpoint:port
- After this configuration, a request from the Context Provider to http://api_umbrella/subs_id/path will be intercepted by API Umbrella and redirected to http://endpoint:port/path

- Note: when setting up a subscription, an expiration time is included in the request. This expiration time is also stored in the 2lp. Thus, when the time is over, it removes the subscription backend registration from API Umbrella.

Management of Access Control for trusted applications

<translate description given in minutes of kick-off f2f meeting>

Establishing trust between participants in the Industrial Data Space is crucial when talking about Security IDS Architecture. It happens that some Context Providers offer data that is not directly managed by them (i.e. stored in their databases) but provided by a third party Context Provider. In such cases, a participant could be authorized to get this data from the first Context Provider but not from the second one. The second provider, when receiving the delegated request, has to be able to authorize the request basing on the permissions the participant has in the scope of the original provider. For allowing this interaction, a trusted relationship has to be established between both Context Providers.



In the previous figure we can see an example of the explained scenario. The Context Consumer has the needed permissions to access a data entity provided by *Context Producer 1* (*ent-1)*. However, although *Context Producer* 1 is offering this entity, it is actually provided by *Context producer 2* so it has to delegate the Context Consumer request to it. When receiving the request, PEP of *Context Producer 2* will check with the IAM

infrastructure that the consumer has the needed permissions to access the entity in the scope of the *Context Producer 1*. Therefore, it should reject the request.

In FIWARE implementation we solve this issue with a mechanism that allows Context Producer to have a list of *Trusted Context Producers.* When receiving a request from a participant, the PEP will check if it has the required permissions to access the data in the scope of itself or in the scope of one of the Context Producers included in its Truster Context Producers list. In the previous example, *Context Producer 1* has to be included in the *Context Producer 2* list. This way, when receiving the request it will check that having the participant no permissions defined in its scope, it actually have the permission to access the entity in the scope of a trusted Context Producer.

Ensuring trust on data end to end

<Check how things could work end to end considering the Use Case on Zero breakdown/defects manufacturing. How we can ensure the security and trust requirements?>

IDS AppStore

In the FIWARE reference implementation of IDS, the AppStore is implemented using the FIWARE Business API Ecosystem GE. I.e. MasterMind, will support the deployment of components (such as IDS Connectors) from the marketplace provided by the Business API Ecosystem GE.. This solution provides as well some additional features such as the support for accepting term and conditions or enabling the monetization of such components.

The workflow to purchase a new component is from the AppStore is as follow:

1. For each component to be purchased in and used in MasterMind, the owner should follow the registration process in the Business API Ecosystem, including the component package (simpler solutions may be used in preliminary versions that does not require storing files in the App Store), the credentials to access a private docker registry (if any), the license code (if any), the terms and conditions, characteristics, pricing, etc).
2. Once the offering is created, the Business API Ecosystem will automatically register the component in the MasterMind catalogue, using its API, providing all the included configuration files.
3. To ensure that only those users that purchased a particular component in the IDS AppStore can see and deploy it MasterMind, the security framework is used. In particular, the different components will be bound to Identity Manager roles, so when a user acquires a component in the AppStore, the BAE will grant the role bound to the component to the user. This way, when the user logs in MasterMind using the IDM, it will retrieve user information (including its roles in MasterMInd), that can then be used internally to build create a custom catalog including only the components purchased by a user (or organizations she/he is member of).

IDS Broker

<structure this section so that basic description according to IDS specs is provided, then elaborate in proposed implementation of IDS Brokers as extended CKAN instances>

<we have to check how monetization is covered within the IDS specs and create subsections dedicated to explain how monetization will be achieved>

<Text from IDS specs:

- Data Providers can offer data to other participants of the Industrial Data Space. The data therefore has to be described by metadata.
- The metadata contains information about the Data Provider, syntax and semantics of the data itself, and additional information (e.g., pricing information or usage policies).
- If the Data Provider wants to offer data, the metadata will automatically be sent to one or more central metadata repositories hosted by the Broker.
- Other participants can browse and search data in this repository.

(page 16, section 3.2 “Functional layer”>

<which may map to the following:

- Context Broker Providers can offer NGSI query datasets to other participants in FIWARE-based Architectures. The NGSI query datasets therefore can be described by metadata.
- The metadata contains information about the Context Broker Provider, Context Producer/Provider, syntax and semantics of the data itself, and additional information (e.g., pricing information or usage policies).
- If the Context Broker Provider wants to offer NGSI query datasets, the datasets with their metadata can be published on one or more federated CKAN instances.
- Other participants can browse and search for NGSI query datasets in CKAN instances.

<One topic we have to cover is how the user who is deploying an IDS connector and the system adapters connected to the Context Broker associated to that IDS connector can, in the same shot, register datasets into the CKAN instance>

<José Manuel raises the point about whether NGSI-9 can be used to support IDS Broker functionality>

Example IDS Use Cases

Predictive Maintenance of Fleet Vehicles in Smart Cities

Zero breakdown/defects manufacturing