LF Energy

Data Architecture - Working Document

v0.5 - 7 July 2020

Table of Contents

1. Introduction

In the age of the energy transition, different devices, actors, and systems must share increasing levels of data to keep electricity and energy systems stable and working reliably.

This document outlines a shared data architecture; for now, with more questions than answers. As a goal, we would like the data architecture to build agreement and coherence between energy systems, regardless of geography or role. This document proposes common definitions based on generic principles in order to support LF Energy open source projects. The document is a collaborative work in progress undertaken by members of LF Energy representing numerous utilities, vendors, suppliers, and organizations in the energy landscape.

The document does not seek to replicate the work of standards bodies from which it heavily draws. Non-differentiating software refers to open source software in the public domain, and which we collectively believe has the potential to accelerate the energy transition and invite innovation at scale.

LF Energy and related Linux Foundation projects focus on open source development or implementations of de facto standards which, through adoption and joint investment, can become the basis for the grid of the future. This does not mean the software is perfect; merely that we share a collective agreement to make it the best in the world for the purpose of higher (or lower) levels of abstraction and connectivity.

Data occupies a central place in the grid of the future. While we cannot literally network electrons, we can network metadata about electrons. A canonical approach to data lets us ensure interoperability between business functions and the objectives of network operators.

1.1. Problem statement

Network operators need to manage large amounts of data to effectively choreograph, orchestrate, or control supply and demand in highly distributed and complex systems. This document proposes how to develop and maintain a semantic framework that supports the LF Energy functional taxonomy while ensuring interoperability and providing visibility to stakeholders.

2. Goals

The LF Energy data architecture provides development and architectural guidelines for:

- Interoperability of data exchange between LF Energy projects
- Applications and data structures
- Defining high-level information flows between LF Energy functional categories

This data architecture defines general principles for interoperability and business and semantic contexts, a brief overview of data modeling, and information on provenance.

2.1. Approval

The LF Energy Technical Advisory Committee (TAC) will review this document for approval.

2.2. Target audience

This document's primary audience is the LF Energy Architecture Working Group.

LF Energy projects (and others) can apply this document's insights to their own projects.

2.3. Context

This data architecture is partly inspired by some of the GridWise Architecture Council (GWAC)'s layers of interoperability of data exchange: syntactic interoperability, semantic understanding, and business context.

The DAMA Data Management Book of Knowledge (DAMA DMBOK) describes many aspects of data management.

See also the Definition of a Global Architecture for Smart Grid Applications.

2.4. Scope

The data architecture provides direction and guidance for open source development.

This document does not address certification, market models, regulation issues, or standards development itself.

Functional architecture and technical architecture are covered by other LF Energy groups.

3. Principles governing LF Energy data projects

3.1. Foundational design

Foundational design principles guide the data architecture implementation of LF Energy projects. These principles cover the general rules of data architecture with developing LF Energy open source projects. Some sections contain statements; other sections contain questions to facilitate conversation and build consensus.

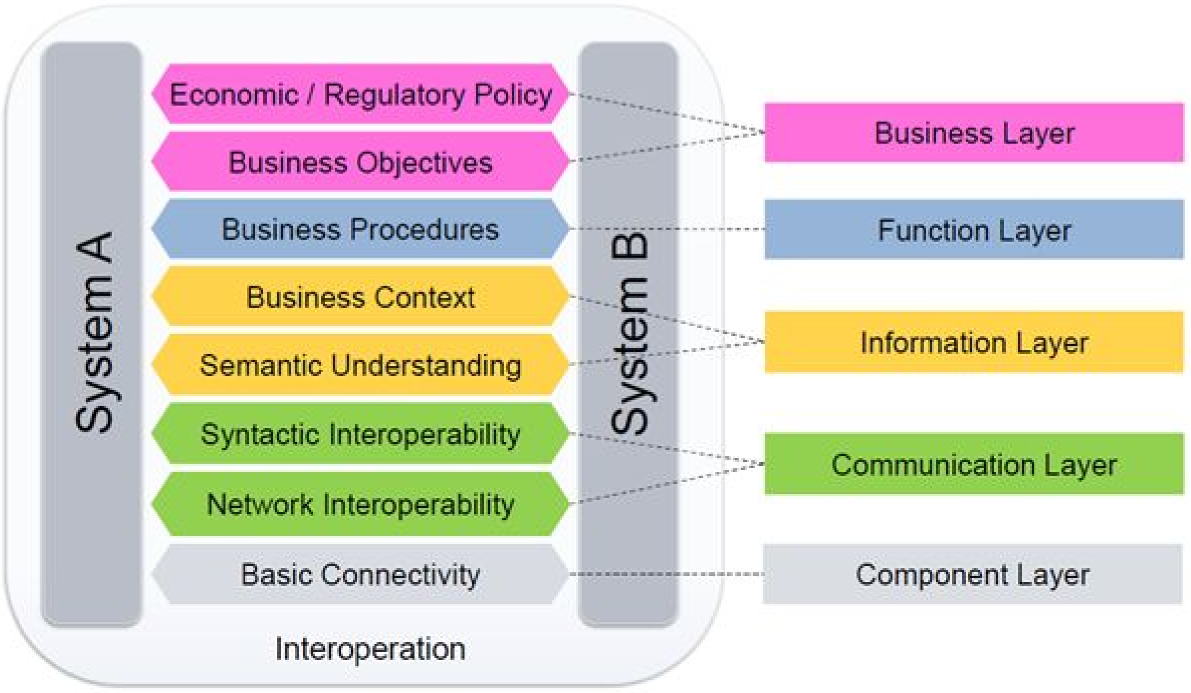
3.2. Layers

A data architecture must include data governance, technical choices, and interoperability.

Interoperability is key for LF Energy and other applications to work together without hassle. LF Energy uses the GWAC interoperability model of layering as a standard.

This LF Energy data architecture focuses primarily on the layers of business context, semantic understanding, and syntactic interoperability.

This data architecture uses GWAC/IEC/M490 layers to define interoperability between systems.



Source: CEN-CENELEC-ETSI Smart Grid Coordination Group CEN-CENELEC-ETSI Smart Grid Coordination Group figure 6 / EU

The GWAC interoperability model is designed by the Gridwise Architectural Council and adopted by the International Electrotechnical Commission (IEC).

In technical reference architecture IEC62357-1 the IEC adopted the GWAC interoperability model with the layers *Business, Function, Information, Communication,* and *Component*.

For more information about interoperability, see:

- EG1 Standards and Interoperability for Smart Grids Deployment
- EG1 supporting documentation
- The Smart Grid Interoperability Maturity Model Summary
- The Decision Makers Checklist

3.3. Interoperability

Element	Description
Principle	Interoperability
Guideline	LF Energy prefers open standards over industry standards over proprietary standards.
Explanation	In the energy sector, development on and use of interoperable applications is best accelerated by using open standards. LF Energy adopts standards (for example, IEC standards, GWAC standards, CIM standards, ENTSO-E standards, W3C) when applicable in order to support interoperability.
Motivation	Interoperability lowers the barriers for connection and co-creation within the ecosystem of applications, especially for platforms and the products and services platforms support. Interoperability also supports: • extensibility (add new systems easily) • replaceability (systems can be replaced) • loose coupling (systems can evolve independently) • low intrusiveness (legacy systems do not need to change their internal architecture to integrate) • recursiveness (systems of systems)
Deviation	LF Energy strongly recommends using open standards. At some stage the LF Energy community must realise and drive new open standards ourselves.

3.4. Business context and semantic understanding

These guidelines cover the layers of business context and semantic understanding.

Element	Description
Principle	Use standardised information models to interface with LF Energy solutions.
Guideline	Use existing information models and standards whenever possible to define interface semantics between applications.
Explanation	The energy sector uses standardized IEC models: IEC standards: ● http://smartgridstandardsmap.com/ & IEC 57 Poster ● IEC Common Information Model (CIM) ● IEC 61850 Others: ● ebIX UML Model for the European Energy Market ● OASIS Universal Business Language (UBL) ● W3C standards (e.g. PROV for lineage; SKOS for definitions) ● IFC (industrial foundation classes/ Building information model) ● SARAF Other semantic standards can be added if needed to define an interface, or for inspiration for an internal data model. You can find standards related to the smart grid at http://gridstandardsmap.com/.
Motivation	● Easier interfacing with existing applications ● Save time by not bootstrapping a new information model
Deviation	LF Energy strongly recommends using standardised semantics. When standardised semantics aren’t adequate, it may be necessary to create new

	semantic definitions or invent new elements or segments of an information model. Examples: ● Current information models do not cover your problem in enough detail ● An current information model is incomplete for your use case LF Energy encourages projects to supply standardisation bodies with their extensions to existing standards. LF Energy projects don't have to wait for extensions to be approved by standardisation bodies to continue the speed of innovation.

3.5. Provenance

Data exchange often requires participants to know the origins of data. This "how to" explains the purpose of provenance (also called *lineage*). The goal for LF Energy is to guide projects on how to implement provenance. LF Energy bases guidance on the W3C standard for provenance.

Provenance can be used for:

- Understanding how data was collected, so it can be meaningfully used
- Determining ownership and rights over an object
- Determining whether certain information is trustworthy
- Verifying that the process and steps used to obtain a result complies with given requirements
- Reproducing how something was generated

Element	Description
Principle	Every application can (optional) publish the lineage/provenance of the data where the application has control (input to output).
Guideline	Use PROV model to publish the provenance if needed

Explanation	Data can be distributed over multiple systems, *in* organisations and *beyond* organisations in many ways. If every application can publish its sources, an organisation gains insight into all of that organisation's data flows. Information model: PROV An alternative would be a centralised system to publish provenance information. This could lead to slower development speed due to dependency on a central system.
Motivation	Data lineage/provenance may be required to: ● Diagnose and repair errors ● Governance ● Compliance
Deviation	Not every application or business needs all data provernance. OMG CWM could be considered for data warehouses.

Source: https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/

3.6. Information identification

Data identification is key in distributing data within the landscape. Identification can help determine whether applications refer to the same data/class. Naming (conventions) could enable unique identification, increasing human usability. A global unique identification method is preferrable (for example, a UUID) because it allows human names to be changed.

Element	Description
Principle	Forward and backward compatibility by using data identification of existing reference models
Guideline	Reuse existing information identification provided by the relevant information models/standards for forward and backward compatibility. The UUIDs can be generated decentralized by each separate application.

	The UUID generation option needs to be available for software that could act as a single source of a (a subset) of data. The applications should be able to import UUIDs from other systems.
Explanation	Information identification for software consumption is key in information exchange. A clear understanding of semantics is key to know the object/element you want to identify. The UUIDs can be generated by the source repository for the data. An example: IEC CIM uses the mRID as identification for computer consumption. It is recommended to use a UUID. Human readable names are modeled differently and not suitable for computer based reasoning.
Motivation	No need for an central UUID generator Compatible with existing information models
Deviation	Some standards are using unique names for identification (for example, IEC 61850). Base-58 encoded UUIDs can be used for presentation and short database fields.

Alternatives for decentralised UUID generation is a central UUID generator. This has advantages to guarantee uniqueness within the scope of the server. The disadvantage is the possible requirement of complex infrastructure.

An alternative for saving keys in a decentralised server is an index and reference server. This server stores the internal identification and stores the external identification name. The advantage of this approach is the flexibility to change UUIDs more easily. The disadvantage is the dependency of this server.

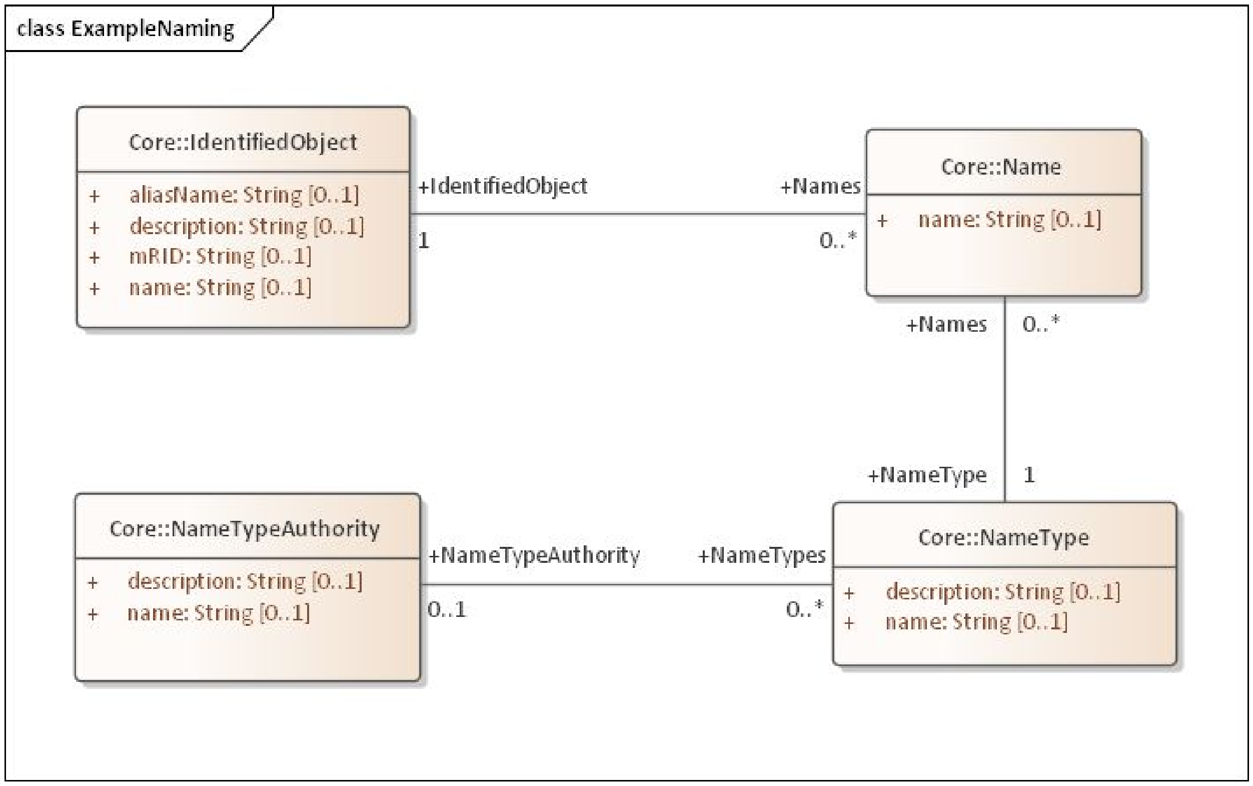
3.6.2. Index and reference server

An example of what an index and reference server could store:

Internal ID	External ID
123456	1115e9d1-25db-4930-9e56-20646d7bcf35
654321	24787b25-0978-4804-8598-833add58f93d

3.6.3. Identification in IEC CIM

The IEC CIM classes for naming and identification in IEC CIM:



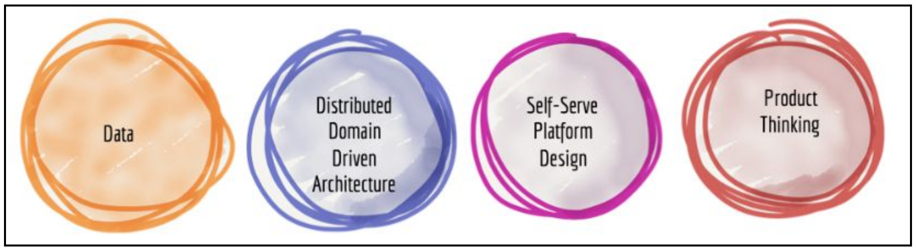
3.7. Information modeling

Information can be modeled in different modeling languages. Some examples of data modeling languages are UML and OWL.

Element	Description
Principle	modeling languages
Guideline	Reuse modeling languages provided by the standards; convert modeling if needed to combine information models or for other good reasons.
Explanation	Use of the original modeling languages allows easy consumption of the standard. The standard usually describes the way to implement it.
Motivation	● Don't reinvent the wheel if we can reuse ● Easiest way to consume the standard
Deviation	Models can be converted to other models to generate artifacts or to harmonize/combine existing standards.

4. Data practices

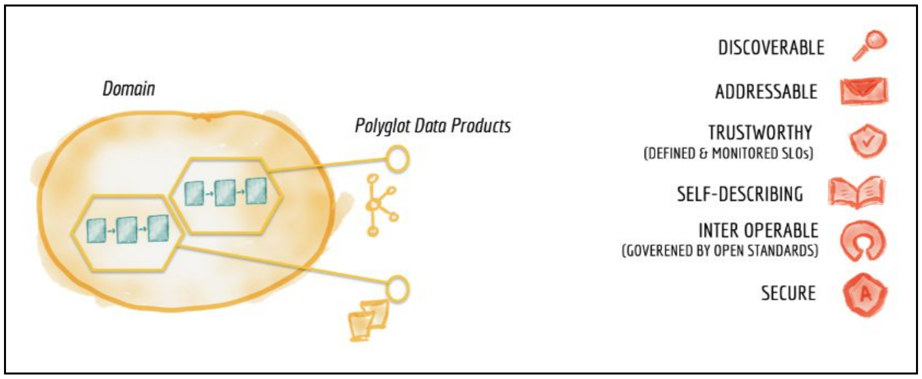
4.1. Adopt a data mesh pattern


Data
Distributed Domain Driven Architecture
Self-Serve Platform Design
Product Thinking

Source: https://martinfowler.com/articles/data-monolith-to-mesh.html

For a definition of terms, see Defining data architecture.

Product thinking and data driven delivery distributed to domains are useful patterns with the LF Energy data architecture. Data mesh patterns combine these concepts. Demands in a complex, emerging energy landscape suggest that a Domain Driven design combined with a Data Driven architecture requires a microservices environment that pushes business processes and functional domains to the edge.



Distributed polyglot data products delivered in functional business domains

Design aspects to drive from the data architecture:

- Scalable polyglot data storage
- Encryption for data at rest and in motion
- Data product versioning
- Data product schema
- Data product de-identification

- Unified data access control and logging
- Data pipeline implementation and orchestration
- Data product discovery, catalog registration and publishing
- Data governance and standardization
- Data product lineage
- Data product monitoring/alerting/log
- Data product quality metrics (collection and sharing)
- In memory data caching
- Federated identity management
- Compute and data locality

4.2. Data definition for interfaces and applications

This data architecture helps LF Energy projects apply the principles of *interoperability* and *business context and semantics*. Determining applicable data definitions early in a project helps define interfaces; these can provide inspiration for an application's internal models.

The following steps show you how to define data. Note that some steps can be combined or revisited.

4.2.1. Create a definition

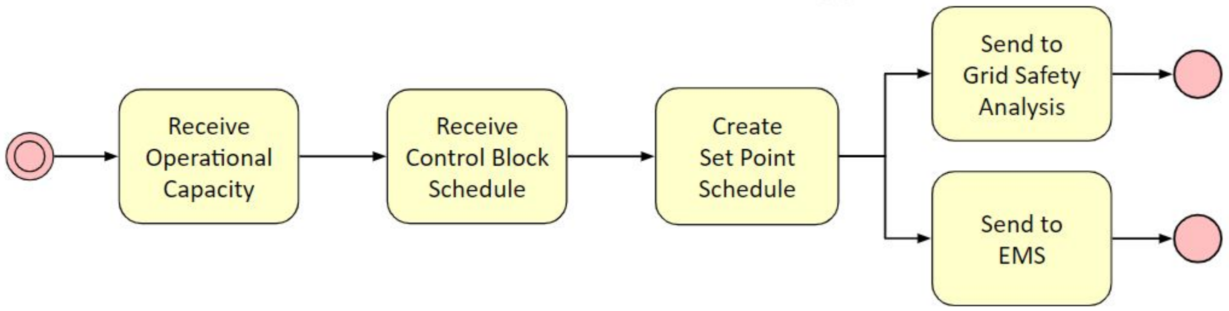
1. Define the business context of a LF Energy project, LF Energy application
2. Define exchange and interaction based on sequence diagrams
3. Determine semantics model
4. Define the business context

Building an interface or an internal application model starts with the requirements.

- What parties/roles are involved?
- Who is communicating with who?
- What is the context of the communication? What process is supported?
- What information needs to be exchanged or used?

Building process diagrams (BPMN) can help to give insights in the questions above. What information is needed in what step? What steps are needed to finish a use-case. Another way to do this is using use-case diagrams.

Example BPMN model: set point schedule process (HVDC interconnectors)


Receive Operational Capacity
Receive Control Block Schedule
Create Set Point Schedule
Send to Grid Safety Analysis
Send to EMS

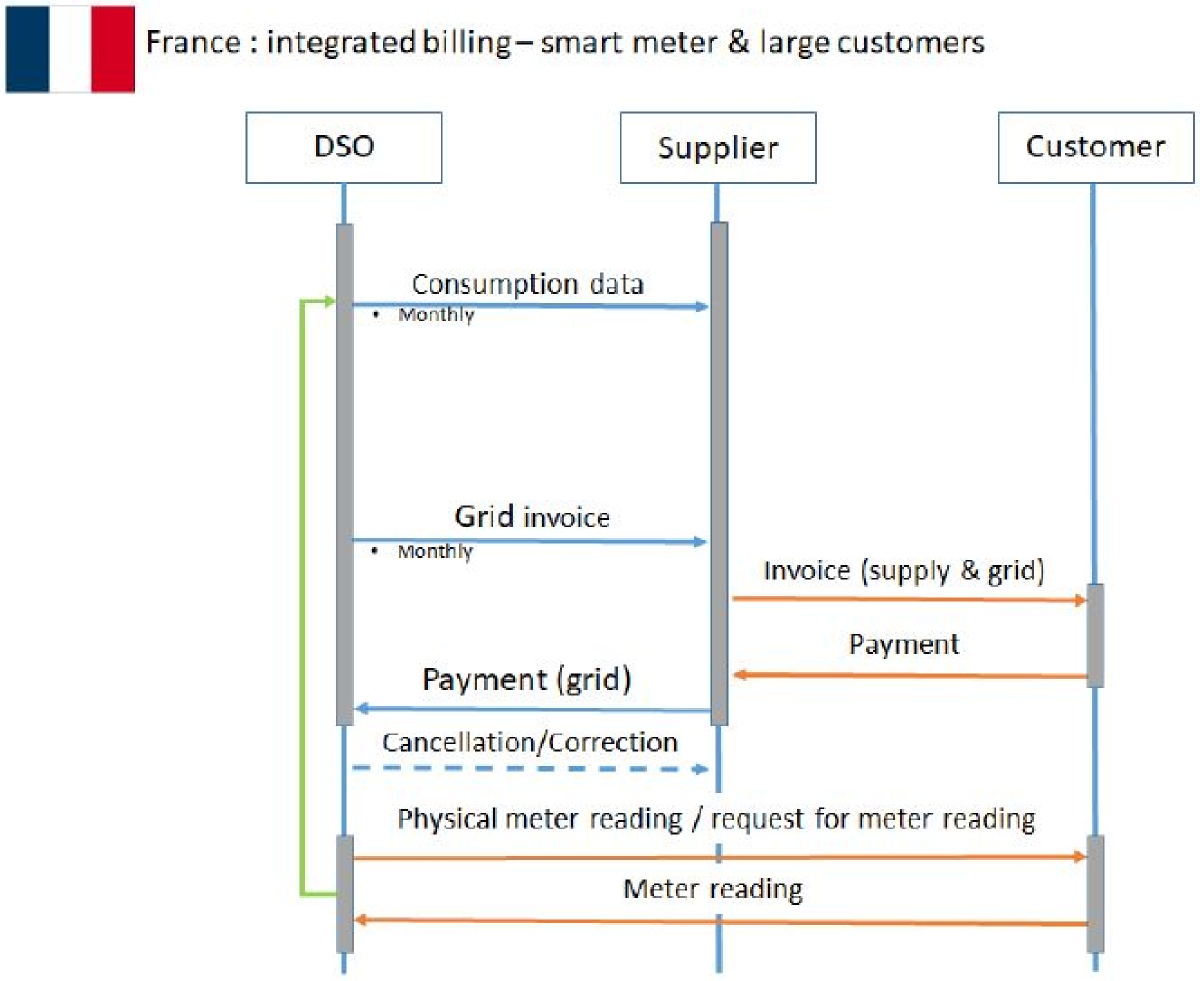
Source: TenneT

Another (more standardised way) to model business context is using use-cases. An example of this is the Use Case Methodology presented in IEC 62559 series of standards which has its origins in EPRI's IntelliGrid project. In addition to the Use Case Repository (UCR) available at EPRI's site, there are some others which have been developed internationally.

4.2.2. Define sequence diagrams

Based on the process flow, a sequence diagram can be defined to describe the interaction between systems. The sequence diagram defines the integration pattern (e.g. request/response or publish/subscribe). The integration pattern defines messages needed on interfaces.

Example:



Source: Towards Interoperability within the EU for Electricity and Gas Data Access & Exchange -European Smart Grids Task Force Expert Group 1 – Standards and Interoperability Working Group on Data Format & Procedures

4.2.3. Determine semantic models

We envision to deliver a decision tree, helping the project answering the following questions, to determine the semantic models from the applicable and available standards.

4.2.3.1. What standard, what part of standard

Once it is known what information needs to be exchanged/is relevant for an application. Relevant semantic models can be searched for a specific domain of problem. Some standards (e.g. IEC CIM) are providing so-called profiles. A profile is a subset of the standards used (e.g. CGMES in IEC CIM) built for a specific use-case.

4.2.3.2. Existing profiles

The IEC CIM profiles can be found in the related IEC documents (e.g. via the IEC webstore). XSD's or profiles in case of IEC CIM can be found on the CIMug website (membership required). In case of a usable profile, just follow the standard as defined.

4.2.3.3. New profiles

If existing profiles do not meet your requirements, you can search the relevant concepts in the relevant standards. In case of IEC CIM, the UML model can be downloaded from the CIMug website (registration required). To understand the model better on specific topics, look for (CIMug) presentations, the CIM primer, study the notes, consult relevante IEC standards or find someone with knowledge. W3c standards for example are freely available. See 5.2 for all relevant standards that could be used within the LF Energy context.

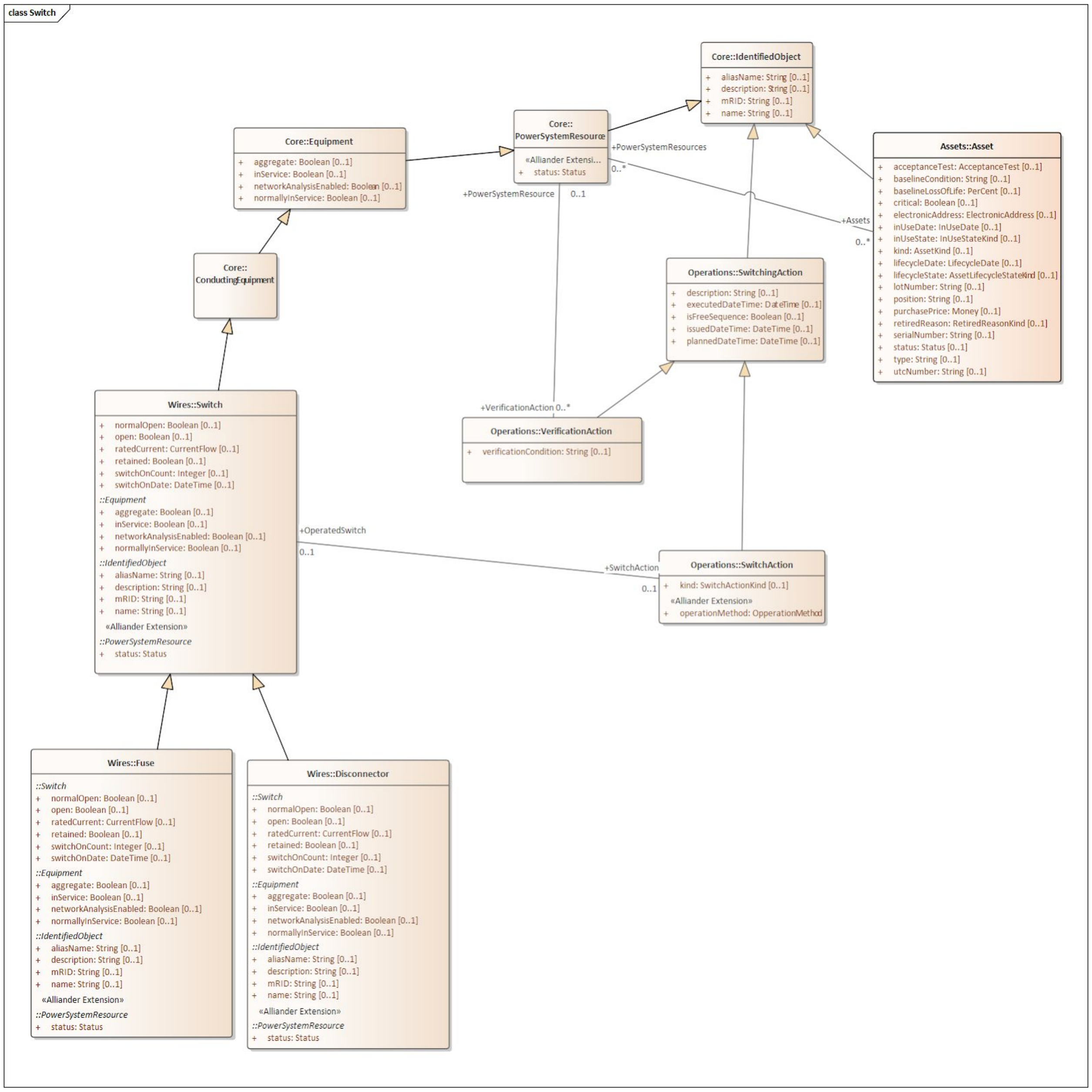
4.2.3.4. Chosen concepts

Relevant concepts could be placed on a diagram (e.g. enterprise architect). A spreadsheet could be used to map requirements on relevant (standardised) models of to (non standard) existing/internal application models.

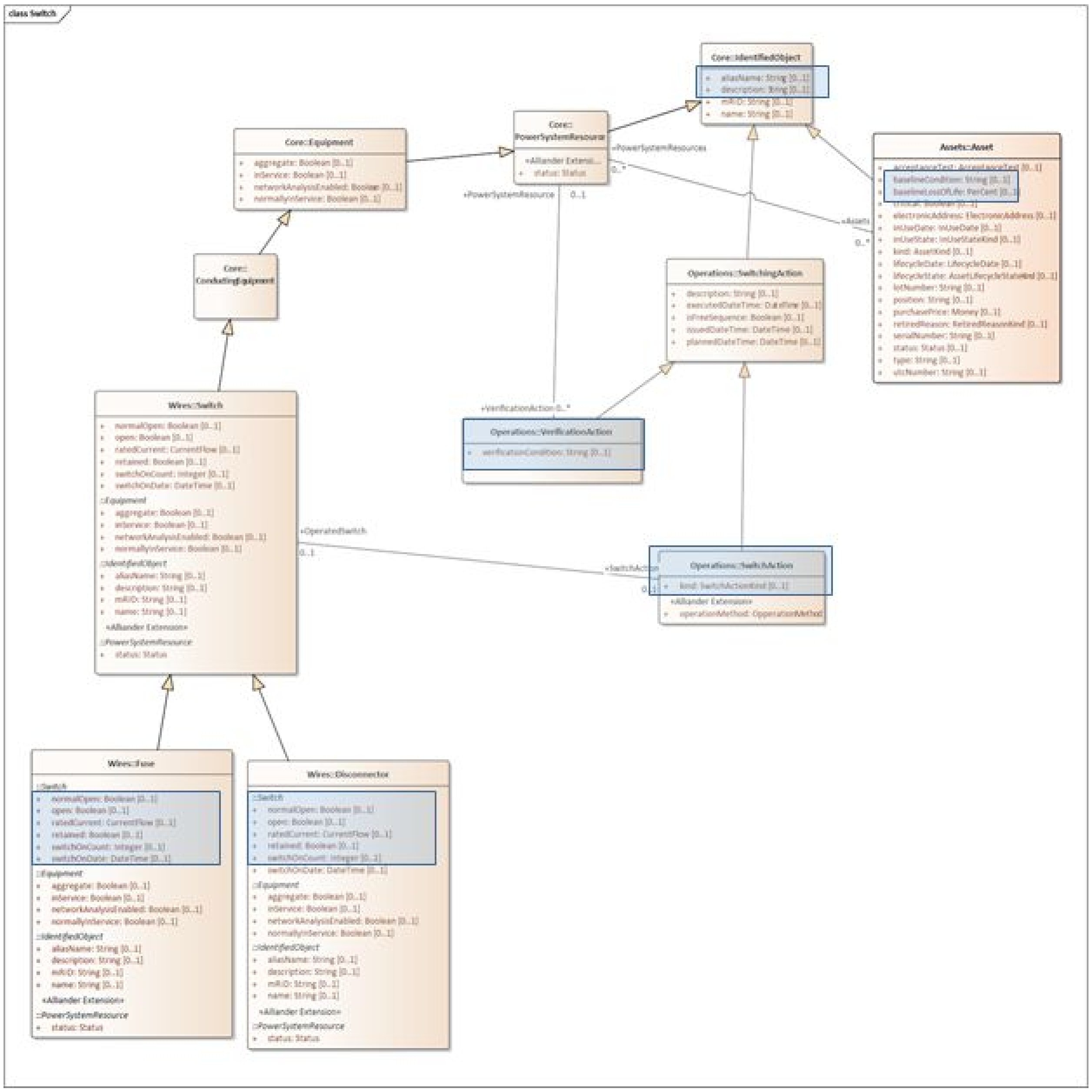
4.2.3.5. No current standards

If a concept can not be found in the existing standards, look for other standards and/or relate multiple standards. If this still does not work. Build your own extensions to existing standards or build your own information model (least prefered).

Example classes diagram with attributes for an interface or application:


class Switch
Core::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
Core::PowerSystemResource
«Alliander Extensi...
+ status: Status
+PowerSystemResources
0..*
+PowerSystemResource
0..1
Core::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
Assets::Asset
+ acceptanceTest: AcceptanceTest [0..1]
+ baselineCondition: String [0..1]
+ baselineLossOfLife: PerCent [0..1]
+ critical: Boolean [0..1]
+ electronicAddress: ElectronicAddress [0..1]
+ inUseDate: InUseDate [0..1]
+ inUseState: InUseStateKind [0..1]
+ kind: AssetKind [0..1]
+ lifecycleDate: LifecycleDate [0..1]
+ lifecycleState: AssetLifecycleStateKind [0..1]
+ lotNumber: String [0..1]
+ position: String [0..1]
+ purchasePrice: Money [0..1]
+ retiredReason: RetiredReasonKind [0..1]
+ serialNumber: String [0..1]
+ status: Status [0..1]
+ type: String [0..1]
+ utcNumber: String [0..1]
+Assets
0..*
Core::ConductingEquipment
Operations::SwitchingAction
+ description: String [0..1]
+ executedDateTime: DateTime [0..1]
+ isFreeSequence: Boolean [0..1]
+ issuedDateTime: DateTime [0..1]
+ plannedDateTime: DateTime [0..1]
+VerificationAction 0..*
Operations::VerificationAction
+ verificationCondition: String [0..1]
Wires::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status
+OperatedSwitch
0..1
+SwitchAction
0..1
Operations::SwitchAction
+ kind: SwitchActionKind [0..1]
«Alliander Extension»
+ operationMethod: OpperationMethod
Wires::Fuse
::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status
Wires::Disconnector
::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status

Example where relevant attributes are marked:


class Switch
Core::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
Core::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
Core::PowerSystemResource
«Alliander Extensi...
+ status: Status
+PowerSystemResources
0..*
+PowerSystemResource
0..1
Core::ConductingEquipment
Assets::Asset
+ acceptanceTest: AcceptanceTest [0..1]
+ baselineCondition: String [0..1]
+ baselineLossOfLife: PerCent [0..1]
+ critical: Boolean [0..1]
+ electronicAddress: ElectronicAddress [0..1]
+ inUseDate: InUseDate [0..1]
+ inUseState: InUseStateKind [0..1]
+ kind: AssetKind [0..1]
+ lifecycleDate: LifecycleDate [0..1]
+ lifecycleState: AssetLifecycleStateKind [0..1]
+ lotNumber: String [0..1]
+ position: String [0..1]
+ purchasePrice: Money [0..1]
+ retiredReason: RetiredReasonKind [0..1]
+ serialNumber: String [0..1]
+ status: Status [0..1]
+ type: String [0..1]
+ utcNumber: String [0..1]
+Assets
0..*
Operations::SwitchingAction
+ description: String [0..1]
+ executedDateTime: DateTime [0..1]
+ isFreeSequence: Boolean [0..1]
+ issuedDateTime: DateTime [0..1]
+ plannedDateTime: DateTime [0..1]
+VerificationAction 0..*
Operations::VerificationAction
+ verificationCondition: String [0..1]
Wires::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status
+OperatedSwitch
0..1
+SwitchAction
0..1
Operations::SwitchAction
+ kind: SwitchActionKind [0..1]
«Alliander Extension»
+ operationMethod: OperationMethod
Wires::Fuse
::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status
Wires::Disconnector
::Switch
+ normalOpen: Boolean [0..1]
+ open: Boolean [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ retained: Boolean [0..1]
+ switchOnCount: Integer [0..1]
+ switchOnDate: DateTime [0..1]
::Equipment
+ aggregate: Boolean [0..1]
+ inService: Boolean [0..1]
+ networkAnalysisEnabled: Boolean [0..1]
+ normallyInService: Boolean [0..1]
::IdentifiedObject
+ aliasName: String [0..1]
+ description: String [0..1]
+ mRID: String [0..1]
+ name: String [0..1]
«Alliander Extension»
::PowerSystemResource
+ status: Status

4.2.4. Building profiles

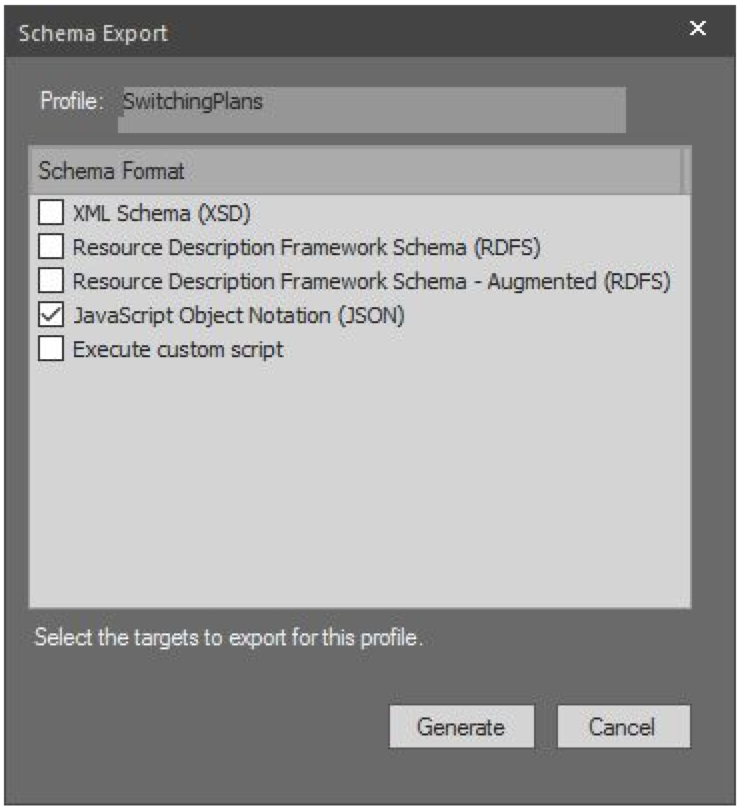
Tools can be used to add constraints to the profile and convert them into usable artifacts. This can be done manually or by using tools. It is important that the artifacts follow the structure as described in the UML if it's used for interfaces. The classes and their relations define the semantics of the model.

Some tools examples for the IEC CIM UML model:

- Enterprise architect (EA) schema composer (sometimes buggy)
- CIMtool (no longer supported)
- Schema composer (Prototype)
- CIMcontextor (maintained by the Entso-e)
- JcleanCIM (UML validation tool)

Consult the manuals to see how these tools can be used.

Example: EA schema composer export options:



Schema Export

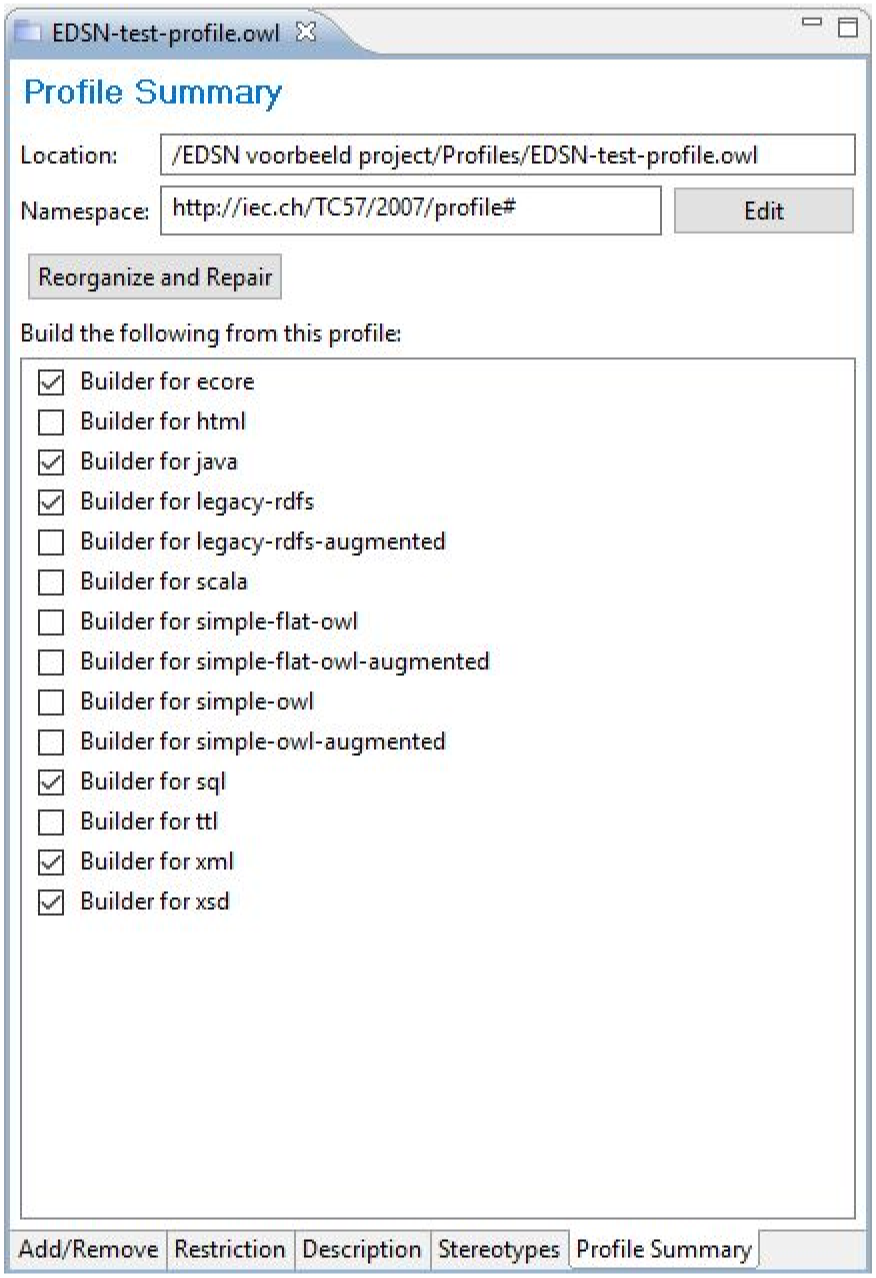
Profile: SwitchingPlans

Schema Format

- [ ] XML Schema (XSD)
- [ ] Resource Description Framework Schema (RDFS)
- [ ] Resource Description Framework Schema - Augmented (RDFS)
- [✓] JavaScript Object Notation (JSON)
- [ ] Execute custom script

Select the targets to export for this profile.

Example: CIMtool export options



Profile Summary

Location: /EDSN voorbeeld project/Profiles/EDSN-test-profile.owl

Namespace: http://iec.ch/TC57/2007/profile#

Edit

Reorganize and Repair

Build the following from this profile:

- [✓] Builder for ecore
- [ ] Builder for html
- [✓] Builder for java
- [✓] Builder for legacy-rdfs
- [ ] Builder for legacy-rdfs-augmented
- [ ] Builder for scala
- [ ] Builder for simple-flat-owl
- [ ] Builder for simple-flat-owl-augmented
- [ ] Builder for simple-owl
- [ ] Builder for simple-owl-augmented
- [✓] Builder for sql
- [ ] Builder for ttl
- [✓] Builder for xml
- [✓] Builder for xsd

For internal application information structures, the profiles can be used as inspiration. Internal data structures could make different choices to make the application perform. Some technologies/databases have restrictions on the allowed data structure.

Profiles can also be used to generate software code (e.g. CIMverter)

4.2.5. References

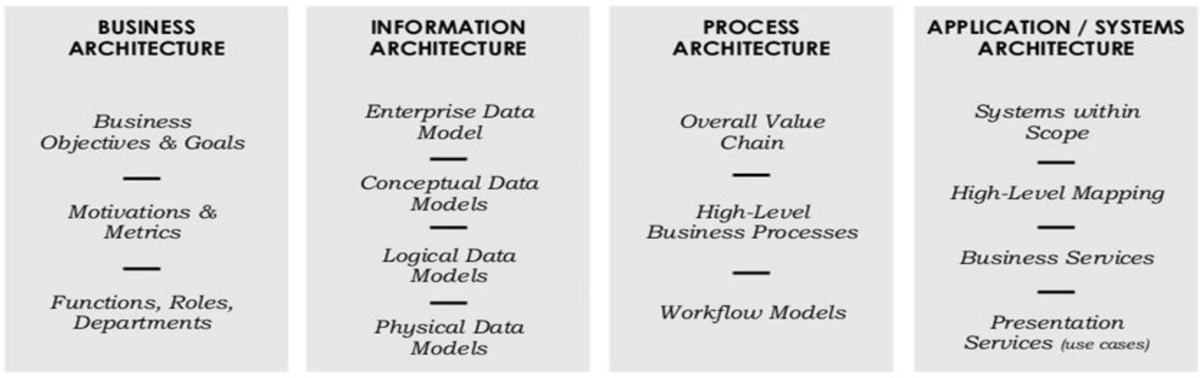
- Supporting material for the final report on Standards and Interoperability Working Group on Data Format & Procedures.
- Definition of a Global Architecture for Smart Grid Applications

5. LF Energy data project proposal

5.1. Defining data architecture

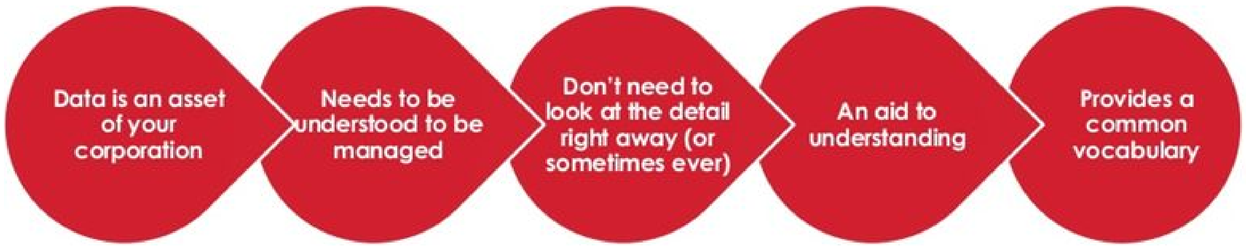
This LF Energy data architecture adopts the Wikipedia definition of data architecture:

> "In information technology, data architecture is composed of models, policies, rules or standards that govern which data is collected, and in what form it is stored, arranged, integrated, exchanged, and put to use in data systems and in organizations. Data is usually one of several architecture domains that form one of the pillars of an enterprise architecture."
>
> Source : https://en.wikipedia.org/wiki/Data_architecture



Data architecture is part of Information Architecture. While information architecture includes functional application architectures, they are out of scope for the data architecture.

Data matters because:


Data is an asset of your corporation
Needs to be understood to be managed
Don't need to look at the detail right away (or sometimes ever)
An aid to understanding
Provides a common vocabulary

5.2. Data architecture vs Data driven architecture

5.2.1. Data architecture

“In information technology, data architecture is composed of models, policies, rules or standards that govern which data is collected, and in what form it is stored, arranged, integrated, and put to use in data systems and in organizations. Data is usually one of several architecture domains that form one of the pillars of an enterprise architecture”

Source : https://en.wikipedia.org/wiki/Data_architecture

5.2.2. Data driven architecture

“Applications reacting on data ‘coming by’. Data served by streaming, event driven architectures in combination with historical, integrated datastores (distributed). Think of Kappa architectures or Lambda architectures implemented by f.i. an event (streaming) distribution solution and an event store solution. Data preparation and data governance like lineage and linkage by means of metadata for instance, is included as well. The data mesh paradigm is combining domain driven and data driven architectures realising a domain driven distributed implementation of data driven architectures.”

Source: Eric Houmes

5.3. Project description

This section contains ideas for an LF Energy data project. Subjects are based on the LF Energy project charter.

This LF Energy project consists of multiple sub-projects:

- Create an LF Energy vocabulary
- Create an LF Energy data architecture and methods (this document)
- Create software for this data architecture

5.3.1. Create an LF Energy vocabulary

In order to establish and maintain a canonical LF Energy vocabulary, determine as early as possible how to choose infrastructure, how to consolidate and store shared vocabulary, and project governance. Consider the LF Edge glossary as a successful example of infrastructure, consolidated vocabulary, and governance.

The goal is to help others understand LF Energy semantic models.

5.3.2. Create an LF Energy data architecture and methods

Make reference model data as portable as possible. The goal of reference model portability is to create a common understanding of LF Energy data architecture, and to improve semantic and syntactic interoperability between LF Energy projects.

Consider the following tools for portability:

- XSD schemas
- Apache AVRO schemas
- Code
- SQL schemas
- Open API specifications
- Cloudevents
- GraphQL
- Odata
- NGSI-LD

5.3.3. Create software for this data architecture

The third step is to create software for automating LF Energy data architectures and methods. The goal of this tool is to build profiles using familiar models (DX-prof/OWL/RDFS+) and generate artifacts.

Extensible frameworks and software (and combinations thereof) to build this functionality include:

- Extend and build an EA (enterprise architect) plugin;
- Build something similar to the Alliander schema composer prototype
- Extend web protegé
- Implement the 61970-501 profiling standard

New tools could replace existing profiling tools like:

- Enterprise architect (EA) schema composer (sometimes buggy)
- CIMtool (open source but no longer supported)
- Schema composer (Prototype)
- CIMcontextor (maintained by the Entso-e / open source but no public collaboration on the source code)

There is already a group thinking about software creation: Alliander, Stattnet, and PNNL.

Appendix A: Future work for this data architecture

These are the placeholders for future work.

- Examples how the model driven data architecture can work in de the modern cloud native environment
- Legal check on collaboration scenarios with the IEC
- Legal check on compatibility on IEC supporting documents with open source licenses "The (slightly limited) IEC supporting documents are available for development under an IEC license (free of charge). "
- UUID generator requirements
- Do we have preferred ways of modeling? UML, OWL of all?
- How do we want to model prov on IEC CIM objects?
- How can we map the formal semantics on the business vocabulary? SHACL?
- Do we want a repository with the LF Energy extensions to the standards?
- Do want want rules to convert from IEC CIM to these formats?
- Information security on interfaces? Is this a topic we want to address in this community?
- Hierarchical & performance requirements for data exchanges & hierarchical interfaces >> for example, Performance based on application

Syntactic interoperability

- How do we serialize information models? Avro/OAS/GraphQL etc? Research: cloud events and serialisation formats https://cloudevents.io/ What are the rules to convert IEC CIM to cloudevents.io?
- What is our URI strategy? Do we need an URI strategy at all for our project?
- What are our API guidelines? **API template?**
- What is the namespace strategy?

Recommended Tools

Schema composer (Alliander): https://github.com/Alliander/schema-composer
Entso-e profiling tool: https://www.entsoe.eu/digital/cim/#cim-tools

- Application internal guidelines
- Conceptual:UML?
- Any thoughts on data storage?
- How to convert a standard information model to an application specific data structure?
- Life cycle management of the data?
- Next research topics:

- <<research micro services architectures and how they related data architecture -- https://12factor.net/>>

Appendix B: Glossary

The exact way to publish this glossary can be discussed later.

General Terms

Term	Meaning
Semantics	The study/science of definitions of words
MDA	Model driven architecture
Business context	The business context where information is exchanged e.g. the electrical domain
Semantic understanding	The meaning of semantics
Syntactic understanding	How the semantics are serialised in formats
IEC	International Electrotechnical Commission
TSN	Time Sensitive Networking
RTOS	Real-Time Operating System

Data Store Terms

Streams: Streams are collections of sequentially occurring events indexed by a single property, typically time series data. To define a stream, you must first define a type, which defines the structure of the data you want to stream.

Types: define the shape or structure of the event and how to associate events within the stream. Possible types can be simple atomic types, such as integers, floats, strings, arrays, and dictionaries, or complex or nested types.

Metadata: Data about data, is information you assign to data objects. The value of metadata itself lies in its capacity to enrich other data, and to facilitate logical segregation and contextualization of data, thus supporting analytical, visualization, organizational and searchability efforts.

Stream Views: Stream Views provide flexibility in the use of types. While you cannot actually change the properties of types themselves, the stream views feature enables you to create a view of a selected stream that appears as if you had changed the type on which it is based. Types themselves cannot actually be changed; the “changing” of a type is described in a stream view. You create a stream view by choosing a source and target type as well as a set of mappings between properties of those two types. Using a stream view to leverage existing type properties is preferable to creating an actual new custom type, because the affected stream continues with its previously archived stream data intact.

Indexes: Indexes speed up and order the results of searches. A key uniquely identifies an event within a stream of events. Keys are unique within the stream. Single vs. compound.

Units Of Measure: defines the magnitude of a quantity (for example, length).
Compression: To more efficiently utilize network bandwidth.
Quality:

Source:
https://ocs-docs.osisoft.com/Documentation/SequentialDataStore/Data_Store_and_SDS.html

Asset Framework Terms

(Redefining old definitions)

Enables to build a representation of your equipment and processes that can give you tremendous insight into your data. Equipment and processes that you want to monitor are called assets. The representation of all your assets and processes together is called an asset model. The asset model organizes all your equipment into a structure that makes it easy to find information.

Asset: In plant management, a physical entity that is a unit of equipment, such as a mixer, hopper, tank, or meter.

Template: Set of properties and values. Assets created from the same template share a common set of properties with the same values. Typically a template defines a type such as a pump or tank. Templates remain linked to assets, any changes made to a template propagate to the elements based on that template.

Event: the fundamental unit of information that consists of a value and a time stamp.

Event Frames: An event frame encapsulates the time period of the event with relevant, comprehensive asset data. Each event frame has a name, start time, end time, one or more assets and streams. As with assets, you can create event frame templates to standardize and manage types of events.

Asset analytics and notifications: Transform simple process data into decision-making information. Referred to as analytics or calculation tools, such tools produce values from simple or complex calculations. You can use the results of these calculations to create new data streams, to make business decisions, as inputs to other calculations, or to trigger notifications.

Source: https://livelibrary.osisoft.com/LiveLibrary/content/en/glossary-v1/
https://livelibrary.osisoft.com/LiveLibrary/web/pub.xql?c=t&action=home&pub=server-v13&lang=en#

Data Views Terms (Business layer integration)

Data views are subsets of data from one or more streams, which can serve as a bridge between raw stream data and data-driven applications. A data view is a declarative query and shape for stream data. Required selected data by specific time period and interpolation interval.

Source: https://ocs-docs.osisoft.com/Documentation/DataViews/Data_Views_Overview.html

Other terms

Security

Account Management

Namespaces
Roles (Administrator, Member)
ACL: Access Control List
Tenant: Multi Tenant

Identity

Role-Based Access Control

Message Format

Headers:

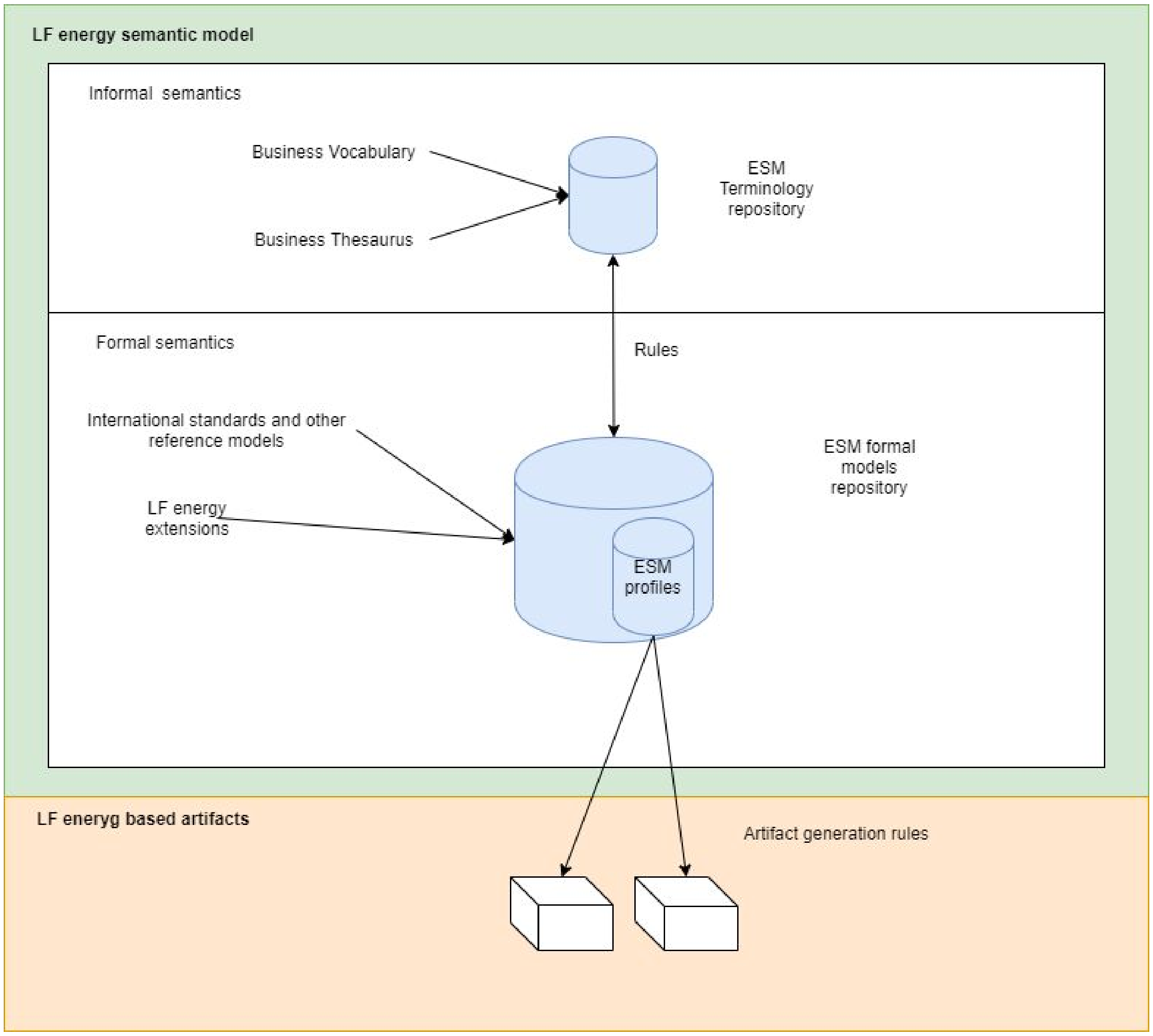
Name	Value
`producertoken`	A unique token used to identify and authorize a given producer.
`messagetype`	Describes the type of message contained in the message body. One of: `type`, `container`, or `data`.
`messageformat`	Describes the data serialization format employed in the message body. E.g. `json`.
`version`	Specifies the version of the Message Format used in the message.
`action`	Optional: One of: `create`, `update`, or `delete`. Describes the action to be performed using the data in the message body. If omitted, `create` is assumed.
`compression`	Optional: The compression algorithm used to compress the message body. e.g. `gzip`. If not specified, the message body is assumed to be uncompressed.

Message Types:

Modeling languages

Information models can be used on multiple abstraction levels. Multiple stakeholders might require different levels of detail.

What kind of models would we define? First suggestion:


LF energy semantic model
Informal semantics
Business Vocabulary
Business Thesaurus
ESM Terminology repository
Formal semantics
Rules
International standards and other reference models
LF energy extensions
ESM formal models repository
ESM profiles
LF eneryg based artifacts
Artifact generation rules

ESM: enterprise semantic model (models used in the enterprise/LF Energy)

Informal semantics

Informal semantics can be used with business people. Most people want a basic understanding of terms. They are most of the time not interested in all modeling details.

Business Vocabulary

Terms used in human interaction can be modeled using a business vocabulary. This is typically a list of terms and their definitions. Due to the nature of human interaction. The terms are sometimes not very specific.

One of the ruling ways to model a vocabulary is SKOS. (Simple Knowledge Organisation System). SKOS allows to build a taxonomy as well.

Potential sources for vocabularies input:

- http://www.electropedia.org/
- CIM model

Formal semantics

Preferably based on international standards e.g. IEC CIM. IEC CIM is modeled in UML classes. Formal semantics can be used to define interfaces and exchange information. It can also be used as inspiration for application internal models.

LF Energy based artifacts

Artifacts that are based on formal semantics. Example artifacts:

- XSD schemas
- Apache AVRO schemas
- Code
- SQL schema's
- Open API specifications
- Cloudevents
- GraphQL
- Odata
- NGSI-LD

Lineage/Provenance

Different people may have different perspectives on provenance, and as a result different types of information might be captured in provenance records.

"agent-centered provenance" : What people or organizations were involved in generating or manipulating the information in question.

For example, in the provenance of a picture in a news article we might capture the photographer who took it, the person that edited it, and the newspaper that published it.

"object-centered provenance" : By tracing the origins of (parts of) a document to other documents.

For example, a web page that was assembled from content from a news article, quotes of interviews with experts, and a chart that plots data from a government agency.

"process-centered provenance" : capturing the actions and steps taken to generate the information in question.

For example, a chart may have been generated by invoking a service to retrieve data from a database, then extracting certain statistics from the data using some statistics package, and finally processing these results with a graphing tool.

Provenance records are metadata. There are other kinds of metadata that are not provenance. For example, the size of an image is metadata of that image but it is not provenance.

Capturing provenance

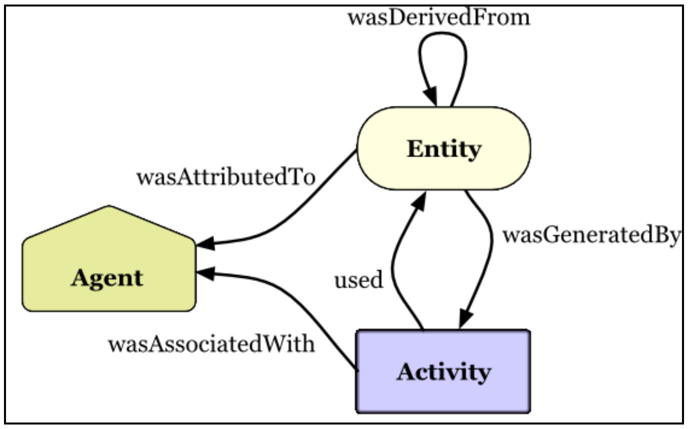
For capturing provenance in a standardised way, we need a data model, an ontology, and a vocabulary. W3C PROV has specified all of these aspects:

- A high-level explanation is explained in this document. A more detailed description of all the concepts and relations in the PROV Data Model is provided in [PROV-DM].
- Ontology based on RDF standard [PROV-O], in a notation designed for human consumption [PROV-N], and in PROV's XML format [PROV-XML].
- Also combining PROV standard with other popular vocabularies such as FOAF [FOAF] and Dublin Core [DCTERMS].

Overview of W3C PROV

This section provides an explanation of the main concepts in PROV. We refer back to the PROV document itself to understand all aspects of the specification and the model. The PROV data model document [PROV-DM] provides precise definitions and constraints [PROV-CONSTRAINTS] to be followed.

The following diagram provides a high level overview of the structure of PROV records, limited to some key PROV concepts. Note that because PROV is meant to describe how things were created or delivered, PROV relations are named



Entities

In PROV, physical, digital, conceptual, or other kinds of things are called *entities*. Examples of such entities are a web page, a chart, and a spellchecker. Provenance records can describe the provenance of entities, and an entity's provenance may refer to many other entities. For example, a document D is an entity whose provenance refers to other entities such as a chart inserted into D, and the dataset that was used to create that chart. Entities may be described as having different attributes and be described from different perspectives. For example, document D as stored in my file system, the second version of document D, and D as an evolving document, are three distinct entities for which we may describe provenance.

Entities are denoted using ovals, as shown in examples below.

exg:dataSet1
exc:regionList1
exc:composition1
exc:chart1
exn:article

Activities

Activities are how entities come into existence and how their attributes change to become new entities, often making use of previously existing entities to achieve this. They are dynamic aspects of the world, such as actions, processes, etc. For example, if the second version of document D was generated by a translation from the first version of the document in another language, then this translation is an activity.

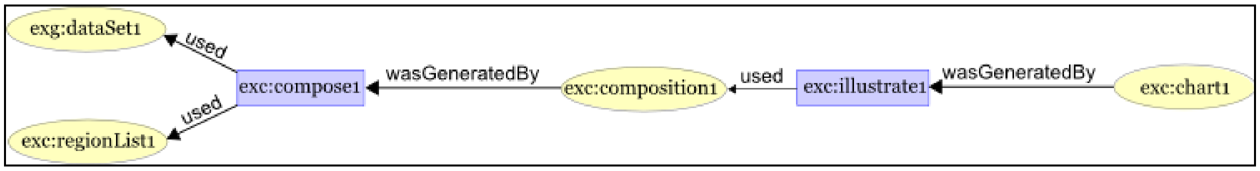
In visualizations of provenance, activities are depicted as rectangles, as below.

exc:compose1
exc:illustrate1
exc:compile1

Usage and Generation

Activities *generate* new entities. For example, writing a document brings the document into existence, while revising the document brings a new version into existence. Activities also make *use* of entities. For example, revising a document to fix spelling mistakes uses the original version of the document as well as a list of corrections. Generation does not always occur at the end of an activity, and an activity may generate entities part-way through. Likewise, usage does not always occur at the beginning of an activity.

In visualizing the PROV data, usage and generation are connections between entities and activities. The arrows point from the future to the past.



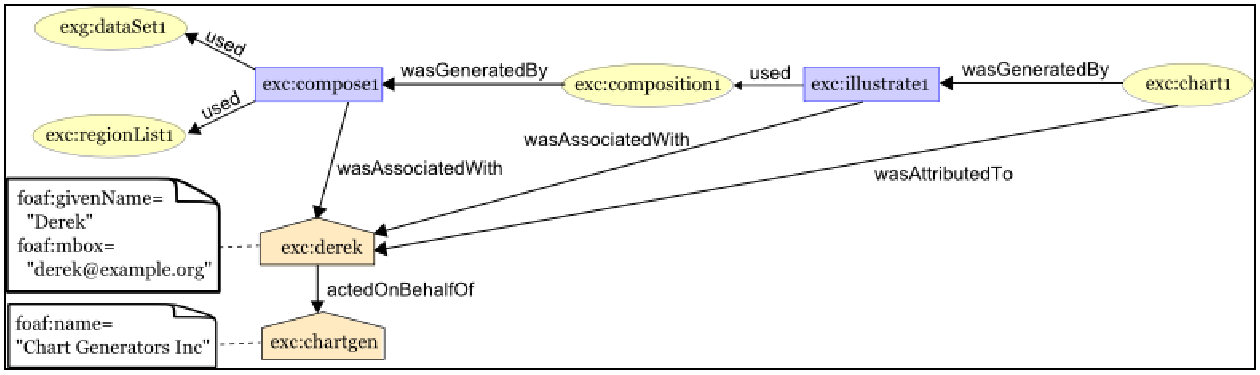
Agents and Responsibility

An *agent* takes a role in an activity such that the agent can be assigned some degree of responsibility for the activity taking place. An agent can be a person, a piece of software, an inanimate object, an organization, or other entities that may be ascribed responsibility. When an agent has some responsibility for an activity, PROV says the agent was *associated* with the activity, where several agents may be associated with an activity and vice-versa.

We can also describe that an entity is *attributed* to an agent to express the agent's responsibility for that entity, possibly along with other agents. This description can be understood as a shorthand for saying that the agent was responsible for the activity which generated the entity.

One may want to describe the provenance of an agent. For example, an organization responsible for the creation of a report may evolve over time as the report is written as some members leave and others join. To make provenance assertions about an agent in PROV, the agent must be declared explicitly both as an agent and as an entity.

We can extend our graphical depiction to show the agents, association and attribution links.



Roles

A *role* is a description of the function or the part that an entity played in an activity. Roles specify the relationship between an entity and an activity, i.e. how the activity used or generated the entity. Roles also specify how agents are involved in an activity, qualifying their participation in the activity or specifying for what aspect of it each agent was responsible. For example, an agent may play the role of "editor" in an activity that uses one entity in the role of "document to be edited" and another in the role of "addition to be made to the document", to generate a further entity in the role of "edited document". Roles are application specific, so PROV does not define any particular roles.

Derivation and Revision

When one entity's existence, content, characteristics and so on are at least partly due to another entity, then we say that the former was *derived* from the latter. For example, one document may contain material copied from another, and a chart was derived from the data that it illustrates.

PROV allows some common, specialized kinds of derivation to be described. For example, a given entity, such as a document, may go through multiple *revisions* over time. Between revisions, one or more attributes of the entity may change. In PROV, the result of each revision is a new entity. PROV allows one to relate those entities by making a description that one was a revision of another. Another kind of derivation is to say that one entity, a quotation, *was quoted from* another entity, commonly a document.

Plans

Activities may follow pre-defined procedures, such as recipes, tutorials, instructions, or workflows. PROV refers to these, in general, as *plans*, and allows the description that a plan was followed, by agents, in executing an activity.

Time

Time is often a critical aspect of provenance. PROV allows the timing of significant events to be described, including when an entity was generated or used, or when an activity started and finished. For example, the model can be used to describe facts such as when a new version of a document was created (generation time), or when a document was edited (start and end of the editing activity).

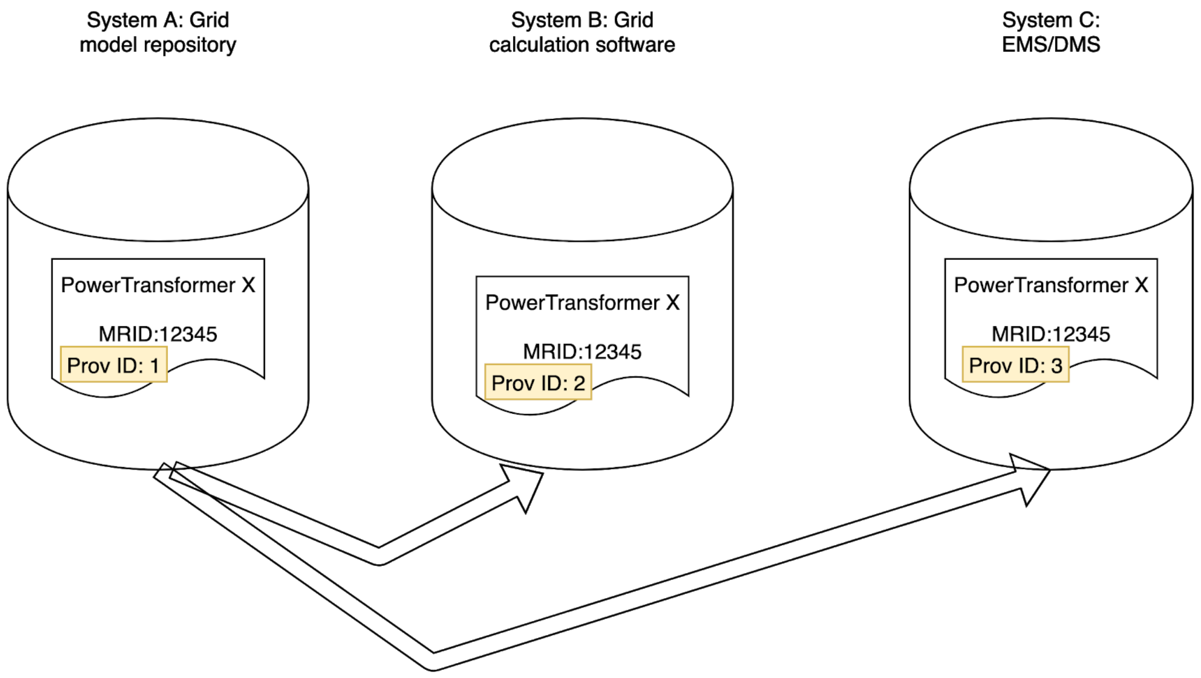
1. Implementation

The implementation of the PROV ontology on interfaces is described by the W3C as well:
JSON: https://www.w3.org/Submission/2013/SUBM-prov-json-20130424/
XML: https://www.w3.org/TR/prov-xml/

Note:
Prov-O en Prov-DM different in the use of camel cases

2. Example

In this example, digital twin presentation in IEC CIM is modeled as PowerTransformer. The information is stored in multiple applications but has one source. For the readability, the UUID is reduced to a simple number.



Let stay that system A is the source. System B and C can point to System A for the source. Since the mRID is the same everywhere, a identification per system is needed to publish the provernance. In this example the ProvID is added to make this work.

The PowerTransformer is the provenance “entity” in this example.

Appendix D

This document previously contained a section that's better suited as a letter to the IEC. For more information, see: https://docs.google.com/document/d/1yx0Y8yf2HoMk_Q56p-y7bHBvR-eBiyZfk1aVmZ0B3v0/edit#heading=h.s6lmgum5iyk