Characteristics of IIoT Information Models

An Industrial Internet Consortium White Paper

Takuki Kamiya (Fujitsu), Andrei Kolesnikov (IOT Association), Brett Murphy (RTI),
Kym Watson (Fraunhofer IOSB), Niklas Widell (Ericsson)

TABLE OF CONTENTS

FIGURES

TABLES

Part I: Core

1 Introduction

Interoperability between applications, subsystems and devices in Industrial Internet of Things (IIoT) systems requires agreement on the context and meaning of the data being exchanged. This agreement is generally captured as an information model, which enables semantic interoperability. Following the definition in the IIC Vocabulary [IIC-2020c], semantic interoperability is the ability to exchange and use information such that its meaning can be understood by the participating systems. In this report, an information model is defined following ISO 10303-1:1994 to be "a formal model of a bounded set of facts, concepts or instructions to meet a specified requirement". An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data structures and semantics. An information model may be motivated by and designed for a certain domain.

There are multitudes of information models available or under active development for a variety of application domains or industries. It is a very active area of the technology stack. This whitepaper focuses on information models above the IIC connectivity framework layer (see Figure 1-1 taken from [IIC-2018]). It is here where semantic interoperability plays a key role, although of course services are needed from the underlying connectivity layer.

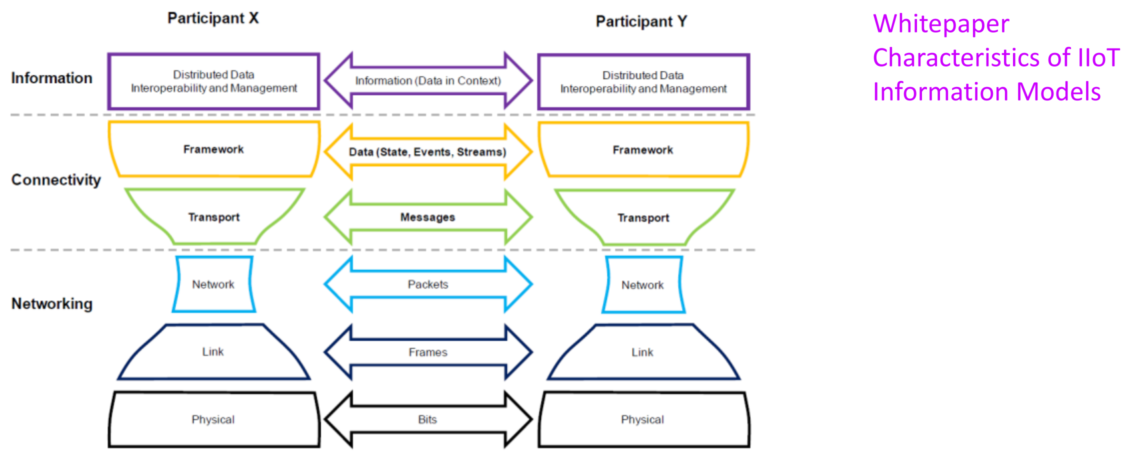

Participant X
Participant Y
Information
Connectivity
Networking
Distributed Data Interoperability and Management
Information (Data in Context)
Distributed Data Interoperability and Management
Framework
Data (State, Events, Streams)
Framework
Transport
Messages
Transport
Network
Packets
Network
Link
Frames
Link
Physical
Bits
Physical
Whitepaper Characteristics of IIoT Information Models

Figure 2-1: Industrial Internet Connectivity Stack Model. Each layer builds on the capabilities provided by the layer below. The 'Connectivity Framework' layer provides data sharing mechanisms among participants. The 'Distributed Data Interoperability and Management' layer relies on the mechanisms provided by the 'Connectivity Framework' layer to provide meaningful information sharing.

Source: IIC Connectivity Framework

Figure 1-1: Positioning of the whitepaper in relation to the IIC Connectivity Framework [IIC-2018]

The IIC Analytics Framework [IIC-2017] describes requirements on information models and several typical scenarios where they are important:

- Data semantics are needed to mitigate the difficulty of data analysis on unstructured data (Table 3-1 Industrial Analytics Requirements).
- Sharing of information models originating from various pieces of equipment (section 5.1 Design Considerations).
- Aggregation of data from event streams into information models with higher level semantics (section 7.1.1 Streaming Real Time Analytics)
- Components in safety-critical systems need well-defined and validated information models so components can be configured safely with no unintended consequences (section 8.1 Safety).
- The curated data originating from various assets (e.g. stored in a historian) should be common across tiers and accessible using a federated information model that supports search, classification and markup to enable rapid industrial analytics application development (section 8.3 Data Management).

[IIC-2020a] gives an overview of the characteristics of digital twins for industrial applications together with several typical use cases. This document also stresses the importance of information models: "The core element of digital twin is information, which is related to different lifecycle phases of the underlying entity".

[IIC-2020b] describes the general IIC approach to and aspects of digital transformation (DX) in applications. Standardized information models with defined semantics and APIs are an essential foundation for any form of DX. There must be a seamless integration across the system life cycles (especially engineering and operations). This applies in particular to all technologies for data sharing ([IIC-2020b, section 2.9].

This white paper surveys a subset of information models that are relevant to IIoT and characterizes those information models using a meta-model developed for this purpose. With this we capture commonalities and can begin to address the challenge of integrating subsystems that use different information models.

2 Meta-Model for Characterizing IIoT Information Models

There are many information models for IIoT systems. Most have been developed independently for a given application area with use cases in mind. This section describes the characteristics of information models with the aim of providing a basis to compare and contrast them.

2.1 Services Needed from Connectivity Framework Layer

2.1.1 Data format

According to the IIC Connectivity Framework, a data type is a syntactic constraint placed upon the interpretation of data and there shall be a data type system for representing data objects as structures in a programming environment and for formatting data to be communicated on some transfer medium.

Common generic base standards used in the Framework Layer to achieve syntactic interoperability are XML and JSON. The standards DDS, OPC UA and oneM2M have their own binary data formats related to their respective information models.

The Media Type (formerly Multipurpose Internet Mail Extensions MIME Type) defines the syntax of the data. See *https://www.iana.org/assignments/media-types/media-types.xhtml* for the extensive list of registered Media Types.

2.1.2 INTERACTION ABSTRACTION

Services: The IIC Vocabulary has adopted the service definition of ISO/IEC TR 14252:1996: “a service is distinct part of the functionality that is provided by an entity through interfaces”.

In IIoT systems RESTful web services are widespread at application level. A RESTful web service is a stateless client-server (or request-reply) interaction and uses HTTP methods to operate on an information resource on the server: POST to create a resource, GET to retrieve a resource; PUT and PATCH to change the state of or update a resource, and DELETE to remove it. These methods implement the four fundamental operations Create, Read, Update and Delete (CRUD). The resource is accessed through its Uniform Resource Locator (URL). The IIC Connectivity Framework uses the term “data object” for a resource.

Quality of service: The IIC Connectivity Framework, section 4.1.10 defines several quality-of-service characteristics, such as relating to reliable data delivery or timeliness, amongst others.

Design patterns: There are different types of data exchange patterns. Prevalent examples are:

- request-reply and
- publish-subscribe.

They may be supplemented by event-driven mechanisms. An event can trigger the activation of a data-exchange or service.

Protocols: A protocol prescribes rules for data exchange. The protocols considered in the IIC Connectivity Framework in the Framework Layer are HTTP, DDS, OPC UA and oneM2M from amongst the numerous protocols that have been defined. These protocols are built on lower level data exchange protocols in the Transport, Network, Link and Physical Layers.

A protocol may provide enhanced access to an information model through advanced queries, such as spatial-temporal and filter queries, pagination of voluminous response data sets.

2.2 Levels of Abstraction in Information Models

An information model may have elements from several layers of abstraction as shown in Figure 2-1. There is not a sharp boundary between the layers. The diagram is intended to provide a broad framework to help understand the scope and characteristics of an information model.

Level 0: The Serialization Layer defines the mapping of data onto bit and byte streams for transmission with a defined protocol over a communication medium. This is dealt with in the Industrial Internet of Things Connectivity Framework and is not in the scope of this report. It is shown here for clarity and deeper understanding of the overall system.

Level 1: The Base Layer defines the fundamental data types of the information model.

Level 2: Semantic Layer defines the meaning of data so that it can be understood by machines.

Level 3: The Conceptual Layer defines the conceptual model underlying the information model. A conceptual model is a meta-model or representation of a set of entities that helps to understand the entities' characteristics and how the entities behave and interact.

At Level 1, numbers and characters are described. These can then be combined into defined data structures such as tables (or data frames with named columns), arrays and lists. Named lists may provide additional metadata. A number as a mathematical concept (element of the set of real numbers) may be mapped in different ways in Level 0 depending on the number of decimal places and type of representation. Similarly, there are various encodings of character strings in Level 0.

The Level 2 Semantic Layer comprises more advanced entities in alignment with the characteristics of IIoT information models presented in this report. Key universal concepts in an IIoT information model are time, location, properties (of observations and other entities) and units of measurement. Properties such as mass and length are expressed in terms of units such as kilogram or meter respectively. Thus, there is a semantic reference between the property and the unit. States and events are common to any dynamic system interacting with its environment.

Note that time is in level 2 as it includes the format definition and the time zone. For example, the time "2020-11-10T11:23:45+0900" as in ISO 8601 means "Year 2020, Month 11 (November), Day 10, 11:23:45 in JST (UTC+9)".

Semantic links and unique identifiers for entities provide the foundation for semantic interoperability between information models. [IIC-2020c] defines an identity to be an inherent property of an instance that distinguishes it from all other instance; the identity information is expressed as an identifier.

The Level 3 Conceptual Layer covers the formal description of the information objects at concept or class level. At a high level, it can be considered as a set of concepts and their relationships. It is an abstraction of the Level 2 entities. A conceptual model in the IIoT domain describes the following:

- model structure in terms of entities, relationships and constraints,
- behavior in terms of states, state transitions and actions performed in states and transitions and
- interactions and interfaces in terms of actions, such as message exchange and information exchange.

Relevant examples of conceptual models at Level 3 are given in the Annexes in the respective sections “Ontology“ and “Metamodel”.

Going from top to bottom, the abstraction layers tend to change from being specific to a vertical or end-user application to being agnostic to (or independent of) the vertical and end user application as illustrated in Table 1.

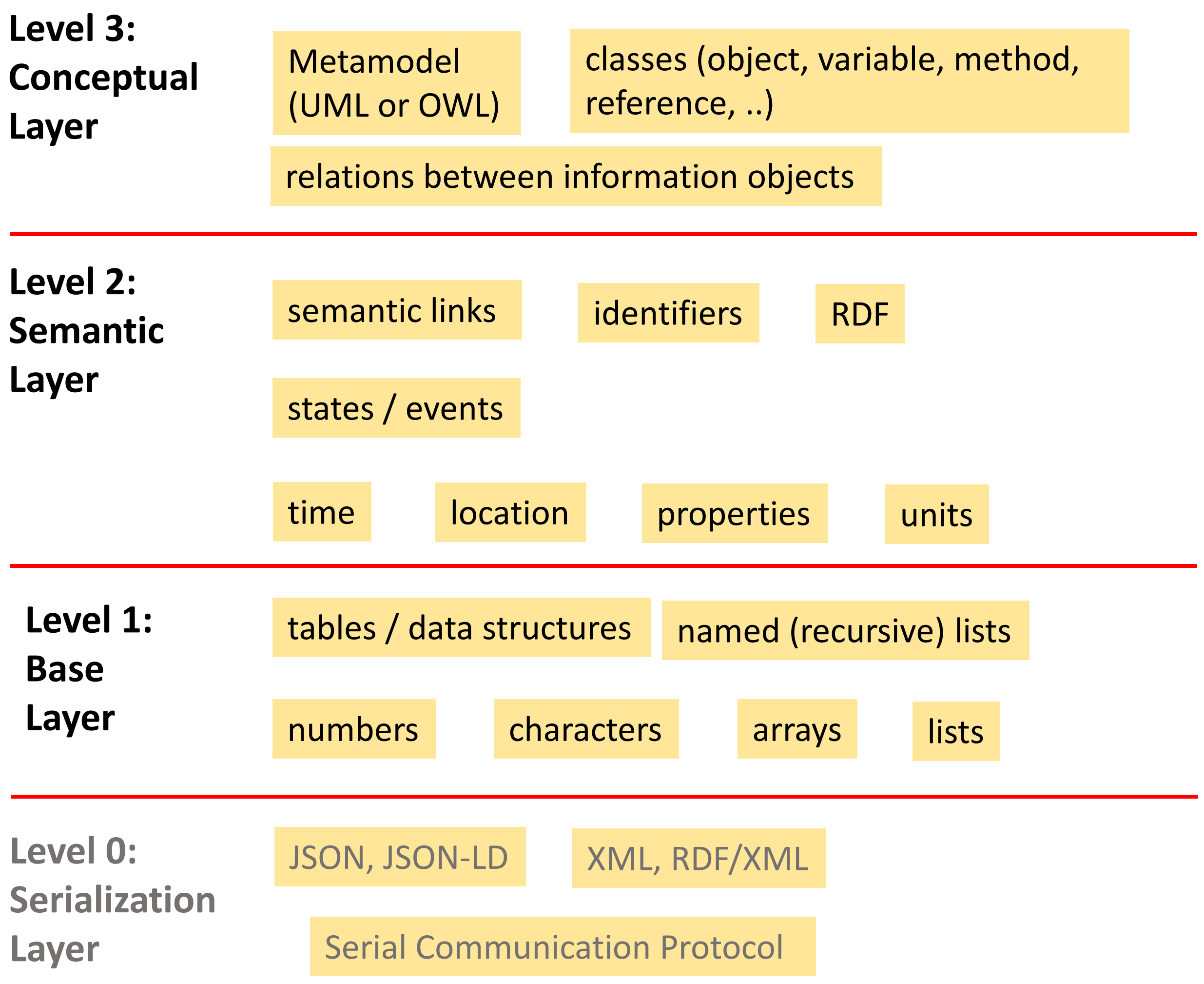


Figure 2-1: Abstraction Layers in an Information Model

Information Model	Vertical Abstraction	End User Application Abstraction
Level 3: Conceptual Layer	Often motivated by the requirements in an IIoT vertical, but with a high degree of generality transferable to other verticals	Agnostic to application (ideally)
Level 2: Semantic Layer	Often vertical specific	Agnostic to application (ideally)
Level 1: Base Layer	Agnostic to vertical	Agnostic to application
Level 0: Serialization Layer	Agnostic to vertical in principle, except to fulfill non-functional requirements such as performance	Agnostic to the application in principle, to fulfill specific non-functional requirements

Table 1: Vertical and End User Application Abstraction in relation to the Layers of Abstraction in the Information Model

2.3 IIoT System Information Model Types

2.3.1 Position in System Life Cycle

In general, the life cycle phases require specialized information models. Common phases in IIoT are development (or design and configuration) and production (or operation). Plattform Industrie 4.0 (Plattform Industrie 4.0)[1] defines life cycle phases by distinguishing types and instances for all assets (according to IEC 62890 Life-cycle management for systems and products used in industrial-process measurement, control and automation). Asset Types have the phases Development (design) and Usage / Maintenance. Asset Instances have the phases Production and Usage / Maintenance. The mining sector, for example, has phases “open pit/exploration”, “initial processing and refinement” and “final processing/finished goods”. The IIC Whitepaper ‘Digital Twins for Industrial Applications’ [IIC-2020a] defines a digital twin to be ‘a formal digital representation of some asset, process or system that captures attributes and behaviors of that entity suitable for communication, storage, interpretation or processing within a certain context.’ An entity may have several digital twins depending on the phase in its life cycle. The whitepaper considers the information flow between digital twins of an entity across its life cycle. Each digital twin needs an agreed information model to achieve interoperability.

2.3.2 Position in (Architecture Layer, System Layer, Hierarchy Level):

Specialized information models are required for the objects in the different levels of an IIoT system. For example: the levels in Plattform Industrie 4.0 are product, field device, control device,

[1] *https://www.plattform-i40.de/PI40/Navigation/EN/Home/home.html*

station, work center, enterprise and connected world (according to IEC 62264/IEC 61512). Each level has its own information objects of concern. Information objects may be application agnostic or highly dependent on the specific application. A few examples to illustrate the diversity are:

- product description data,
- sensor values of a field device,
- parameters, function blocks and state machines of a control algorithm in a controller,
- configuration data of machines in a work center and
- supply chain descriptions.

More complex things in an IIoT system require more advanced information models with well-defined underlying ontologies, extensible semantic description capabilities and data structures.

2.4 Core Characteristics

2.4.1 Ontology

Sharing and reuse of data between applications is a big challenge in building IIoT systems. Knowledge models for a specific application tend to be tightly coupled with the application-specific concepts, which makes moving data between applications a difficult task to undertake.

Knowledge models created using relational database schemas are limited to using one kind of relationship, the foreign key. This is because database schemas concerns with structural representation of data, not the semantics of data. Traditional methods such as database schemas and XML schemas provide limited expressivity in modeling knowledge, which forces business logic that should really be part of the knowledge model to be embedded into applications. This makes it difficult and expensive for users to switch to new software, often leaving them stuck with obsolete software.

Decoupling the knowledge model from the application is critical to enabling data sharing and reuse between systems or across knowledge domains. All of the business logic in the knowledge model should be captured in the model itself and not in the applications.

With two systems each using a different schema for its database, data sharing is achieved by translating from one schema into the other. Data sharing would occur through a process of translation. A dump of data would be mapped and translated from one schema to the other. The original information is replaced with new information in a form conforming to the target schema. This means any data that cannot be directly translated will be lost.

On the other hand, Information semantically described in a knowledge model can be shared across domain boundaries by defining the concepts of the foreign domain using the concepts of the local domain. This is a descriptive approach instead of the translational approach seen for data sharing based on databases. With the mapping being descriptive, the data has meaning in both domains and the full fidelity of the original data are maintained. For example, Semantic Web technologies such as Resource Description Framework (RDF), RDF Schema (RDFS), Web Ontology Language (OWL) and SPARQL support this descriptive, semantics-based approach of data sharing.

Resource Description Framework (RDF) provides a way to model information in a graph. However, it does not provide a way of specifying the meaning of that information by itself. Other means of defining a vocabulary of terms with semantics for describing the information are therefore necessary. A simple vocabulary is a collection of defined terms used in communication. A taxonomy is a vocabulary in which terms are organized in a hierarchical manner. In a taxonomy, for example, you can define that both Speedometer and GPS are sensors, or both mirrors and doors belong to a body in a vehicle. An ontology is vocabulary of terms to define concepts and the relationships between them. Taxonomies are more expressive than vocabularies, and ontologies are more expressive than taxonomies. Ontologies allow you to express the semantics behind vocabulary terms, their interactions, and context of use. In Semantic Web technologies, RDFS supports taxonomies, and OWL extends RDFS to provide a language for defining ontologies that capture semantics of domain knowledge. Ontologies are the core element of Semantic Web.

Not all existing information models are based on Semantic Web technologies. For example, the OPC UA information model uses object-oriented model paradigms with features such as classes and entities (Object Types and Objects), properties, attributes and methods. It is more descriptive than a taxonomy. There are successful examples of attempts to map those non-Semantic Web information models to a formal ontology such as OWL. Mature Semantic Web Frameworks such as Jena[1] are available, and mapping to OWL ontology allows complex queries on the information model and automatically apply reasoners to generate new knowledge using those frameworks.

2.4.2 CONTEXTUAL DATA

The Industrial Internet of Things Vocabulary Technical Report defines information to be 'data that within a certain context has a particular meaning'. Contextual data is additional metadata to qualify the data or to provide further information on its meaning. In a basic approach, the contextual data may simply be supplementary data fields, e.g. with the units, time, location and validity conditions of a sensor observation. Contextual data may also describe how the data was collected, by whom or what it refers to (e.g. water temperature 1m below the surface, or wind speed 10m above the ground). In the Semantic Web as defined by W3C, technologies such as RDF and OWL are used to create a web of linked data as explained above. The linked data is a "semantic annotation" that may provide (formalized, machine readable) contextual data.

Example of negotiation metadata: Contextual data is important in all kinds of interactions between things in IIoT systems, in particular for negotiations between resources and assets. Standardization is needed for business applications. The most important information entities are:

- negotiation protocol

[1] *https://www.w3.org/2001/sw/wiki/Apache_Jena*

- contextual information on the subject and status of the negotiation and
- Semantic expressions of contract content representing the tenders and results of the negotiation.

UN/CEFACT have worked on relevant standards:

- Business Requirements Specification for Electronic Tendering International Standardization; *http://www.unece.org/fileadmin/DAM/cefact/brs/BRS_eTendering_v2.8.0.pdf, 2008.*
- Business Requirements Specification for Cross Industry Scheduling Process v2.0; *https://www.unece.org/fileadmin/DAM/cefact/brs/BRS_Cross_Industry_Scheduling_Process_v2_FINAL.pdf*, 2017. Two central concepts in this specification are Demand and Supply Forecasts.

New standards may be required to provide enhanced functionality in negotiation applications.

2.4.3 Metamodel

OMG's Model Driven Architecture (MDA) is a conceptual framework that separates functionality description from platform choices. At the center of MDA concept is the modeling standard Unified Modeling Language (UML).[1] MDA promotes UML as the centerpiece of its vendor-neutral approach to system interoperability. Since UML is for describing user-domain models, it is a metamodel. UML is a popular modeling language that has been used for modeling concepts in various industry domains for decades. Its popularity not the least comes from the intuitive graphical representation that UML tools can provide for users to interact with models.

On the other hand, W3C's Semantic Web is a concept that builds on Web architecture to enable data integration, shared semantics and web of data. One of the key technologies to enabling these goals is Web Ontology Language (OWL).[2] OWL is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. Unlike MDA, which originated from the background of software development practices, OWL was initially derived from the need to integrate multiple heterogeneous datasets from various sources. In Semantic Web, ontologies developed using OWL are essential in integrating disparate datasets. OWL provides mechanisms to integrate and align and map different ontologies by which machines can understand heterogeneous data in an integrated way. One of the major appealing points of OWL is its ability to support automatic reasoning and inferences. This ability allows reasoners such as Pellet[3] to augment the knowledge base with the deduced implicit facts, which makes ontologies easier to maintain. Ontology development is a complex

[1] *https://en.wikipedia.org/wiki/Unified_Modeling_Language*, *https://www.omg.org/spec/UML/*

[2] *https://www.w3.org/OWL/*

[3] *https://www.w3.org/2001/sw/wiki/Pellet*

task that is usually done using ontology engineering tools such as Protégé[1]. Such tools allow to browse-and-edit ontologies. However, many of them are not very strong in supporting graphical editing features, often forcing designers to specify ontologies in terms of complex logic axioms.

As seen above, UML and OWL each has its own advantage. UML provides an intuitive graphical representation, and OWL is based on formal semantics to allow reasoning and inferences. A push towards ontologies based on Semantic Web technologies is observed recently. However, many of the existing modeling work was done in UML, and UML diagrams are still dominant as a way to communicate the modeling work. Ontology designers often end up having to draw UML diagrams for communication and create OWL ontologies simultaneously. The tasks are duplicate in many ways, and maintaining two models simultaneously and separately is error-prone.

OMG's MDA provides a facility to describe metamodels called Meta-Object Facility (MOF). Because UML and OWL are modeling languages, they can both be modeled using MOF. MDA also defines model transformations in Ontology Definition Metamodel (ODM) by which UML models can be transformed into OWL models. As ODM acknowledges, the specified non-normative transformation rules will often lead to less-than-ideal choices for mapping some structures. Therefore, the result of UML-to-OWL transformation based on ODM still should be reviewed by people who are versed in the domain knowledge in question.

Platform Industrie 4.0 defines its information model of the Asset Administration Shell (AAS) in UML as specified in the document: *Details of the Asset Administration Shell –Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0)*, 2019-November available at: *https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.html*

The AAS serves to provide a uniform interface to assets and to exchange asset information between partners in a value chain of Industry 4.0. The Reference Architecture Model Industrie 4.0 (RAMI4.0) defines a hierarchy of assets: Connected World, Enterprise, Work Center, Station, Control Device, Field Device and Product. The AAS UML model can be mapped to various standardized representations expressed in XML, JSON, RDF, OPC UA and AutomationML. The mappings are bi-directional, meaning that the UML model can be (re-)created by importing a file in one of the underlying formats (assuming that the file structure follows defined rules). Schema have been defined for XML and JSON. The AASX Package Explorer (*https://github.com/admin-shell/aasx-package-explorer*) is an open source tool designed to create an AAS; it supports the above mappings.

The OGC standard SensorThings API (cf. Annex B) also defines its information model in UML with a mapping to JSON.

[1] *https://protege.stanford.edu/*

The separation of the conceptual knowledge from technology dependent representations is illustrated schematically in Fig Figure 2-2. The UML model describes object classes, their attributes, methods or operations and the relationships between objects. This separation ensures that the UML model can be implemented in various technological platforms and facilitates achieving interoperability of evolving application software.

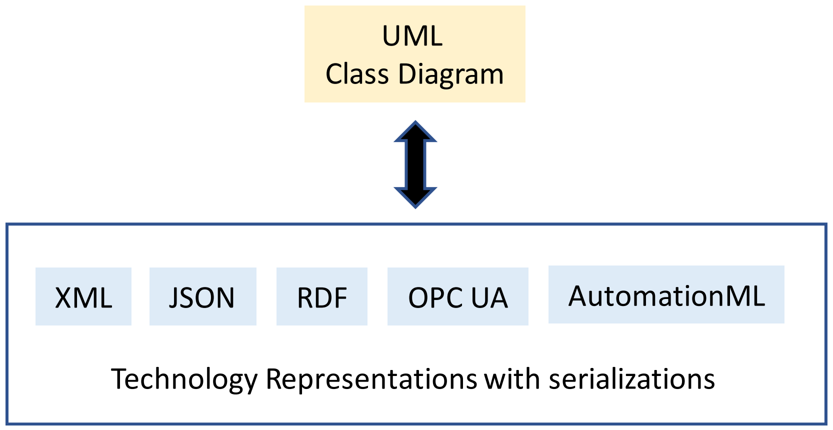


Figure 2-2: Mapping of a UML model to various representations

2.4.4 Observation & Measurement

The concepts of observation and measurement are defined in ISO 19156:2011 (Geographic information -- Observations and measurements).

An observation is the “act of measuring or otherwise determining the value of a property”. A property is a “facet or attribute of an object referenced by a name”, e.g. temperature or color. A measurement is a “set of operations having the object of determining the value of a quantity”.

This is a central capability of any IIoT system, viz. to be able to observe “hings” in the system.

Time is essential metadata for observations in many applications. An observation without full time information may well be meaningless or ambiguous. Real-time applications in particular must record time-related information. Time can be described as a time instance or time interval (open or closed at each end). It may be expressed as a number with a defined zero time relative to another time system, e.g. Unix time is the number of seconds since January 1, 1970 (UTC). Time may also be expressed as a string in ISO 8601 format to describe the date, time and time zone. Cross-region applications must include the time zone and apply a defined date-time format to achieve interoperability.

The *location* of a thing in an IoT system is typically essential information about the thing in order to interpret and exploit other thing related data such as sensor data or to realize location-based services. The location can be described as a point, line, polygon, surface or solid in a Coordinate Reference System (CRS). The CRS can be a geodetic coordinate system such as latitude/longitude

/altitude in the commonly used WGS 84[1]. In general, “outdoor” applications in smart cities, environmental monitoring and agriculture use geodetic coordinate systems. The CRS can also be expressed in cartesian coordinates of an X-Y-Z coordinate frame defined relative to an anchor point. This approach is common for applications in buildings and also in vehicles (cars, planes, boats etc). The vehicle has its own CRS independent of its position and orientation in a world coordinate system. A third common way to describe location is by a symbolic name, for example the name of an administrative region, city, factory or plant, piece of equipment in a factory or plant, room in a building. The actual location is then defined implicitly through the location of the referenced thing. Mobile things (e.g. vehicles, AGVs, smartphones) have a dynamic location that is typically tracked. A thing may have several location descriptions with differing resolution, e.g. the car is at a point with given latitude/longitude or the car is in a certain street.

Recommended further reading on standards dealing with location:

- ISO 19156:2011 (Geographic information -- Observations and measurements)
    - *https://portal.opengeospatial.org/files/?artifact_id=41579* Cf. in particular section 7.1.4 on the location of an observation and feature-of-interest
- OGC 15-078r6 SensorThings API Part 1: Sensing
    - *https://www.opengeospatial.org/standards/sensorthings*

[IIC-2020c] defines an *IoT Sensor* to be an IoT device that observes one or more properties of a physical entity and converts those properties into information.

The observation procedure applied by the sensor may be important in evaluating the information returned by the sensor, for example, its accuracy.

The *data format* in this context refers to the format of the observation or measurement value. Over time many different standards have emerged in various industries. They are often defined as a binary or JSON or XML data structure.

2.4.5 Actuators and Tasking

[IIC-2020c] defines an *IoT actuator* to be an IoT device that can change one or more properties of a physical entity in response to received information.

For example, a switch can be turned on or off, a valve can be partially opened or closed. A camera can be equipped with an actuator to control its pan, tilt and zoom settings.

Regulatory control is a well-established field in advanced process control. A control algorithm determines the target setting of an actuator depending on observed sensor values and the value of the process variable to be controlled.

[1] https://en.wikipedia.org/wiki/World_Geodetic_System#WGS84

Tasking is defined as parameterizing IoT actuators and sensors. This includes scheduling and configuration.

In general, actuators cannot execute the requested changes immediately or unconditionally. It is necessary to check if this is physically possible, safe to do so or even permitted. In addition, changes requested by different applications may pose scheduling or physical conflicts. Various sensors, especially cameras, need to be configured to make the desired observations. Mobile sensor carriers such as vehicles or satellites need to be brought into position before a sensor on the carrier can make observations. An application may request a sensor to make an observation at given time.

The standard 'OGC SensorThings API Part 2 – Tasking Core' of the Open Geospatial Consortium (17-079r1, *http://www.opengis.net/doc/IS/sensorthings-part2-TaskingCore/1.0*) defines basic mechanisms to parameterize so-called taskable IoT devices such as actuators and sensors.

The OGC® Sensor Planning Service Implementation Standard of the Open Geospatial Consortium (OGC 09-000, *http://www.opengis.net/doc/IS/SPS/2.0*, *https://www.ogc.org/standards/sps*) defines web service interfaces to provide information about the capabilities of a sensor, how to task the sensor with a feasible tasking request and to actually execute the sensor task.

2.4.6 Security

The 'Industrial Internet of Things Volume G4: Security Framework' [IIC-2016] gives a detailed description of system characteristics enabling trustworthiness as well as business, functional and implementation viewpoints of a security framework. Here we restrict our attention to several aspects relevant to information models.

2.4.6.1 Authorization and Access Control

Role-based access control (RBAC) for data and services in an IIoT system is a standard common practice. Users (humans as well as apps) acquire access through the assignment of defined roles. In attribute-based access control (ABAC), attributes of the user, data resource, action and context define access permissions. The access rules are expressed as policies, for example defined with the OASIS standard XACML (eXtensible Access Control Markup Language).[1] The main elements of XACML are *subject* (to describe who is accessing the data set), *action* (to describe what the subject wants to do with the data), *resource* (describing the data set) and *environment* (describing the context such as time and location).

Usage control is an extension of access control. It addresses the specification and enforcement of restrictions on how data may be used. This is relevant for IP protection and digital rights management, for example. Usage control is one of the core concepts of IDSA (International Data

[1] *http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html*

Spaces Association).[1,2] The IDS Reference Architecture Model lists further examples in IIoT systems where access control is not sufficient and usage control is required:

- secrecy: to ensure that classified data is forwarded only to users with the respective clearance,
- integrity: to ensure that critical data is not modified by unauthorized or untrusted users,
- time to live: deletion of data after a defined period,
- anonymization by data aggregation: a sufficient number of data sets is aggregated to prevent de-anonymization of sensitive data (e.g. personal health data),
- anonymization by data substitution: replacing part of the data (e.g. pixelization of faces or car number plates) to prevent misuse of the data.
- separation of duty: ensuring that data sets (possible from competitive entities) are never aggregated or processed by the same service and
- sage scope: ensuring that data may only be input to a Trusted Connector as defined by IDSA.

The IDS Information Model is expressed as an RDF Ontology to provide unambiguous identifiers with several programmatic representations in the native structure of the respective program environment. The central concept in the IDS Information Model is a Digital Resource of various types as defined in ISO/IEC 2382:2015 (data, text, software, audio, image, video, container, opaque). A Resource refers to a Concept (with a semantic annotation of the Resource content) and a Context (describing spatiotemporal and real-world entities linked to the Resource). Data usage restrictions are expressed in the technology-independent IDS Usage Policy Language that is based on the W3C recommendation Open Digital Rights Language (ODRL)[3].

2.4.6.2 Authentication

Identification of elements in an IIoT system is fundamental to ensuring their correct usage. Identifiers can be assigned not only to assets (such as machinery, SW and HW components), but also to properties and concepts. Identifiers serve a dual fold purpose, namely:

- to distinguish uniquely IIoT elements for their correct identification and selection and

[1] *https://www.internationaldataspaces.org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0.pdf*

[2] *https://www.internationaldataspaces.org/wp-content/uploads/2019/11/Usage-Control-in-IDS-V2.0_final-1.pdf*

[3] *https://www.w3.org/TR/odrl-model/*

- to bind semantics to these elements. This enables the semantic interoperability of applications.

Identity management (governance) is required to have a systematic and accepted procedures to assign identities and define the semantics of the associated element.

Identification of Digital Objects in ITU-T X.1255

The purpose of Recommendation ITU-T X.1255 is to provide an open architecture framework in which identity management (IdM) information can be discovered. This IdM information will necessarily be represented in different ways and supported by various trust frameworks or other IdM systems using different metadata schemas. This framework will enable, for example, entities operating within the context of one IdM system to have identifiers from other IdM systems resolved accurately. Without the capability for discovering such information, users and organizations (or programs operating on their behalf) are left to determine how best to establish the credibility and authenticity of a suitable identity, whether for a user, a system resource, information or other entities. Based on this information, it is up to the user or organization to determine whether or not to rely on a given trust framework or other IdM system for such purposes. The core components of the framework set forth in this Recommendation include:

- a digital entity data model,
- a digital entity interface protocol,
- one or more identifier/resolution systems and
- one or more metadata registries.

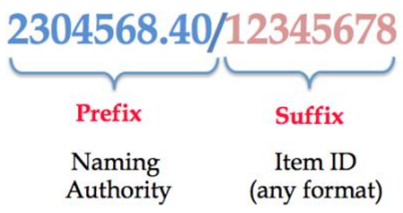
These components form the basis of the open architecture framework.

The X.1255 document has its roots in Digital Object Architecture (DOA) set of standards: RFC 3650 "Handle system overview", RFC 3651 "Handle system namespace and service definition", RFC 3652 "Handle system protocol specification".

The Digital Object Architecture's goal is to provide a solution to the following digital information management issues:

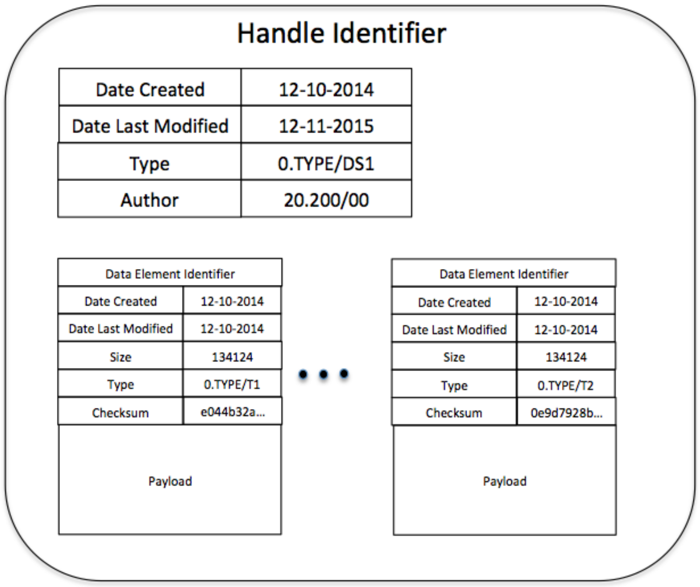
- provide standard access to heterogeneous information, carrying the functions of Identification, search and retrieval, while maintaining trust (including security),
- provide interoperability across heterogeneous information systems due to its independence from the specific underlying technologies that host and serve the information,
- maintain interoperability over long periods of time and
- support very large level of scalability due to its distributed and open architecture based on standard protocols and procedures.

DOA runs on a global Handle System resolution architecture, while every instance can be open or private. It requires the minimum set of object attributes needed to exist on a network. Every identifier is persistent, globally unique and secure. Unlike domain names, it is not semantically recognizable (not for humans, but for the machines, program agents, etc.). Each unique identifier resolves into metadata about the object. An object contains zero or more Data Elements. Each Data Element has a unique identifier. A Data Element has state metadata about itself.



- handle composition.

A Handle is a globally unique and resolvable identifier. Prefix is resolvable by the Global Handle Registry to a Local Handle System (LHS). The identifier is resolvable by Local Handle System into set of typed values. Identifier supports 2.0, UTF-8.



Date Created	12-10-2014
Date Last Modified	12-11-2015
Type	0.TYPE/DS1
Author	20.200/00

Data Element Identifier	
Date Created	12-10-2014
Date Last Modified	12-10-2014
Size	134124
Type	0.TYPE/T1
Checksum	e044b32a...
Payload	

Data Element Identifier	
Date Created	12-10-2014
Date Last Modified	12-10-2014
Size	134124
Type	0.TYPE/T2
Checksum	0e9d7928b...
Payload	

Figure 2-3: DOA Identifiers

Overall, DOA is a universal flexible schema for assigning and resolving any kind of identifiers for digital object and its metadata. Currently there are practical cases of re-using existing proprietary and open identifiers (such as MAC address, IMEI, serial numbers, etc.) in industrial systems. The most well-known DOA based system is DOI used for the global identification of books and scientific articles.

More information can be found here: *https://www.dona.net/digitalobjectarchitecture*

Identification in the I4.0 Asset Administration Shell

Identifiers are one of the focal subjects in [I4.0-ZVEI-2019]. They are defined for the Asset Administration Shell (AAS), assets, submodels (instances and templates), property definitions and concept descriptions in external repositories such as eCl@ss and IEC CDD.

For example, the Property "Number per minute" has the eCl@ss identifier 0173-1#02-AAT096#001 as an IRDI (International Registration Data Identifier).

As a second example, the Property 'Max. rotation speed' (e.g. of a motor) has the eCl@ss IRDI identifier 0173-1#02-BAA120#008. It is defined to be the greatest permissible rotation speed with which the motor or feeding unit may be operated and has the unit of measure 1/min.

[I4.0-ZVEI-2019]	Details of the Asset Administration Shell –Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0), 2019-November *https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.html*

2.4.6.3 CONFIDENTIALITY/PRIVACY

There are situations where sensitive data, for example, crucial and critical information gathered in a production process, need to be guarded safely. With the privacy protection for individuals by law such as GDPR (General Data Protection Regulation 2016/679, EU), personal data are not allowed to be released to third parties or to the public without adequate authorization.

Modern data technologies, such as encryption, can be used to maintain confidentiality and privacy protection, but they also create more complexity in both storage systems and computations and as a result, affect the efficiency of data utilization.

The method of data collection can also help protect user privacy. For example, the ITS Probe Data collected from vehicles gives detailed information about the vehicle for many ITS applications. The data are stored collectively with added data obscurity of vehicle specific information so that a user of the vehicle cannot be directly traced from the data.

A distinction must be made between different types of applications used as a top-down overview based on user types and working environment. There are three major pillars:

Confidentiality (of the information generated, transmitted, processed) is the property that information is not made available or disclosed to unauthorized individuals, entities or processes. Breaches of confidentiality can occur by word of mouth, printing, copying, emailing, or through software vulnerabilities that allow attackers to read or exfiltrate data. Data exfiltration is the unauthorized transfer of data read through exploits at another location under the control of the attacker. This data may be used for blackmail or other purposes. Confidentiality controls include access control and encryption technologies.

Privacy (protection of personally identifiable information): is the right of an individual or group to control or influence what information related to them may be collected, processed, and stored and by whom, and to whom that information may be disclosed.

Security (protection of data and systems) is the condition of the system being protected from unintended or unauthorized access, change or destruction.

Sensitivity of data is a function of privacy and confidentiality. In general, privacy is associated with personal (human) use for the cases where an IoT system generates data that directly contains the parameters associated with end users or personal information that can be derived from the data (geo tracking, medical records, in-house habits, etc). There are local regulations in most countries, including the GDPR of the EU on the collection, processing and transfer of personal data. A privacy framework is defined in ISO/IEC 29100.

Confidentiality of data models and the data itself for non-human IoT systems is regulated by corporate policies and governmental regulations, for example the requirement to transmit and store sensor or processed data within the country (EU GDPR, Russia's law on critical informational systems, etc). For instance, the GDPR may prohibit the export of personal data originating in the European Economic Area (EEA) to countries outside of the EEA. Countries also exercise export control on critical goods, including data and software (for example the United States Export Administration Regulations, or the regulations of the German Federal Office for Economic Affairs and Export Control). The EU-US Privacy Shield regulates transatlantic data flows.

By default, information models are open to access and defined by different standards bodies, such as international organizations, NGO's and professional associations.

An IIoT System Information model plays the role of "data container", whereby the data can be open or encrypted, depending on system setup and threat model. For more detailed information and a more comprehensive discussion, the reader is referred to "Industrial Internet of Things Volume G4: Security Framework".

3 Overview of Examples of Information Models

The annexes give an overview of several information models that are widely-applied in IIoT applications. The focus is on information models that are openly available with no intellectual property restrictions and ideally defined in international standards. They are described in terms of the characteristics explained in the previous sections. Some information models are primarily for particular application verticals as shown in Table 2.

Information Model	Reference	SDO	Principal verticals
Web of Things		W3C	
SensorThings API, Part 1 Sensing	OGC 15-078r6 V1.1	Open Geospatial Consortium	Environment, Transportation, Public Sector
OPC UA	IEC 62541	IEC SC 65E OPC Foundation	Manufacturing
OPC UA Companion Standards		VDMA, OPC Foundation	Manufacturing
Asset Administration Shell		Plattform Industrie 4.0	Manufacturing
Smart Objects		IPSO	At field level in various verticals
OneDM / Semantic Definition Format	SDF 1.0	IETF OneDM	At field level in various verticals

Table 2: Information Models considered in this Whitepaper

There are far too many information models to consider all in detail. The following have been mentioned in our discussions, in some cases with accompanying presentations, but they have not been considered to the same level of detail as those in Table 2:

- Digital Twin information models,
- ISO/IEC JTC1 WG11 Smart Cities,
- ETSI oneM2M (Manufacturing, Buildings & facilities, Public Sector),
- IEC / ISO TC 204 Intelligent Transport Systems (transportation),
- GENIVI (vehicles),
- OAGIS (Open App Group-Supply Chain),
- GS1: EPCIS (supply chain),
- IEC 62832 (digital factory),
- ISO 15926 (engineering lifecycle-process automation),
- Open Robotics Institute,
- ISO 10303 (manufacturing),
- AUTOSAR (AUTomotive Open System ARchitecture),
- NAMUR (process control),
- Oil & Gas: reliability and maintenance data for equipment ISO 14224,
- WITSML (energy and utilities),
- IEC 61850 (intelligent devices in electrical grid) and
- IEEE 11073 Medical device communication (health care).

	Interaction	Ontology / Metamodel	Contextual data	Observation & Measurement	time	location	security metadata
Web of Things	interaction affordances for Properties, Actions and Events; hypermedia principle	basic UML model; aligned with O&M	yes	yes	RFC3339 profile of ISO 8601	W3C note	yes
SensorThings API, Part 1 Sensing	RESTful API	UML, aligned with O&M	yes	yes	ISO 8601	GeoJSON	no
Asset Administration Shell	I4.0 language being defined; export to OPC UA etc	UML	yes	no	ISO 8601	not explicit	yes
OPC UA	services on UA nodes; request-response and publisher-subscriber	UML	yes	no	ISO 8601	not explicit	yes; fully specified
IP Smart Objects (IPSO)	RESTful; request-reply	no	yes	no	Unix time	partially	yes
OneDM / Semantic Definition Format	interaction affordances for Properties, Actions and Events; RESTful	Basic UML	no	yes	RFC3339 profile of ISO 8601	no	no

Table 3: Coarse Classification of Information Models considered in this Whitepaper

3.1 Related Work

The importance of information models to interoperability in IIoT systems continues to drive work on general standards and methods. Several relevant examples are mentioned below.

ISO/FDIS 30141 “Internet of Things (IoT) - Reference Architecture” gives a general description of a wide range of characteristics of IoT systems and an IoT Conceptual Model, Reference Model and Reference Architecture. The Conceptual Model is defined in UML and covers the high level IoT concepts: Digital Entity, Physical Entity, Domain, IoT User, Network and Identity. Information models are not addressed explicitly, but referred to in the context of supportive functions for accessing IoT system resources.

ISO/DIS 23247 Part 3 “Digital representation of manufacturing elements” considers the digital representation of several manufacturing elements (personnel, equipment, material, process, facility, environment, product and supporting documents) as a table of high-level attributes for each element. These attributes are mostly specific to the manufacturing vertical. ISO 23247 Part 3 has an informative annex listing several technologies that can be used to represent the manufacturing elements.

[JAC 2020] compares several standards for IoT and digital twins, including the Web of Things, SensorThingsAPI and Plattform Industrie 4.0 AAS in Table 3. It considers a similar list of characteristics as in this report and discusses meta-models with a focus on resource description, identification and discovery. [JAC 2020] proposes a metamodel hierarchy based on multilevel metamodeling as conceived by the Object Management Group [OMG 2002].

ISO/IEC JTC 1/SC 41 is working on the standard “Internet of Things (IoT) - Interoperability for IoT Systems - Part 3: Semantic interoperability” that is currently at Committee Draft stage ISO/IEC CD 21823-3:2019. The draft considers interoperability between IoT resource models and ontology-based data integration.

Part II: Annexes

Annex A WoT (Web of Things) Information Model

A.1 Background Information

The W3C (World Wide Web Consortium) Web of Things (WoT) specification work is an effort to standardize a common, web-based interface to IoT Things. The specifications are created by the W3C WoT WG (Working Group), while supporting work is performed in the W3C WoT IG (Interest Group). Full details available here *https://www.w3.org/WoT/*

On a high level, WoT is based on using web technologies such as URIs and hypermedia controls applied to the IoT domain. WoT introduces Thing Descriptions (TDs) as a way to describe a Thing with affordances, data schema, security configuration and protocol bindings. A TD allows for understanding what a Thing can do and how to interact with it in a generalized and machine processable manner, with the specifics of the interaction being managed by a protocol binding to the particular Thing implementation.

WoT WG has two main specifications that have reached Candidate Recommendation and are on track to become W3C Recommendations, namely WoT Architecture and WoT Thing Descriptions.

A.2 Services Needed from Connectivity Framework Layer

A.2.1 Data Format

WoT is neutral with regards to data formats. Data formats are identified by Media Types.

Regardless of the data format, when linked data is used to annotate the WoT Thing Description, the data items exchanged between system entities can be associated with semantic meaning as long as the Thing Description is properly annotated using the mechanism provided by JSON-LD.

A.2.2 Interaction Abstraction

A.2.2.1 Services

Services provided by IIoT entities are described in WoT Thing Description in terms of *Interaction Affordances* (i.e. properties, actions and events).

The hypermedia principle, which is one of the core foundations of the REST architectural style, demands that any piece of information available on the Web be linked to other pieces of information so that the consumer of the information gets explicit knowledge about how to navigate the Web and control Web applications. Here, the simultaneous presentation of information and control (provided in the form of hyperlinks) is a mechanism that affords Web clients the means to drive Web applications.

Drawn from this hypermedia principle, the Web of Things defines *Interaction Affordances* as metadata of a Thing that shows and describes the possible choices to clients, thereby suggesting how clients may interact with the Thing.

A hypermedia control is the machine-understandable description of how to activate an affordance. Hypermedia controls usually originate from a Web server and are discovered in-band while a Web client is interacting with the server. This is opposed to out-of-band interface descriptions that need to be preinstalled or hardcoded into clients (e.g., RPC, WS-* Web services, HTTP services with fixed URI-method-response definitions). Hypermedia controls enable loosely-coupled, dynamic and autonomous clients.

For data, WoT uses the CRUD+N interaction model for properties (i.e. data items).

A.2.2.2 Quality of Service

WoT is descriptive, not prescriptive, and so is generally designed to support the QoS mechanisms of the systems it describes, not introduce new ones.

A.2.2.3 Design Patterns

Each *interaction affordance* in WoT has one or more *interaction verb(s)* each representing a semantic intention of an operation on the affordance. For example, a *property* interaction affordance may provide one or more of *readproperty*, *writeproperty* or *observeproperty* operations. An *action* interaction affordance may provide *invokeaction* operation, and similarly, an *event* interaction affordance may provide *subscribeevent*, *unsubscribeevent* operations.

WoT allows those abstract operations to be bound to concrete protocol methods and options. In a WoT Thing Description, these concrete bindings are provided as Web forms. Forms in the WoT can be seen as request templates provided by the Thing to be completed and sent by the clients.

A.2.2.4 Protocols

The number of Protocol Bindings a Thing can implement is not restricted.

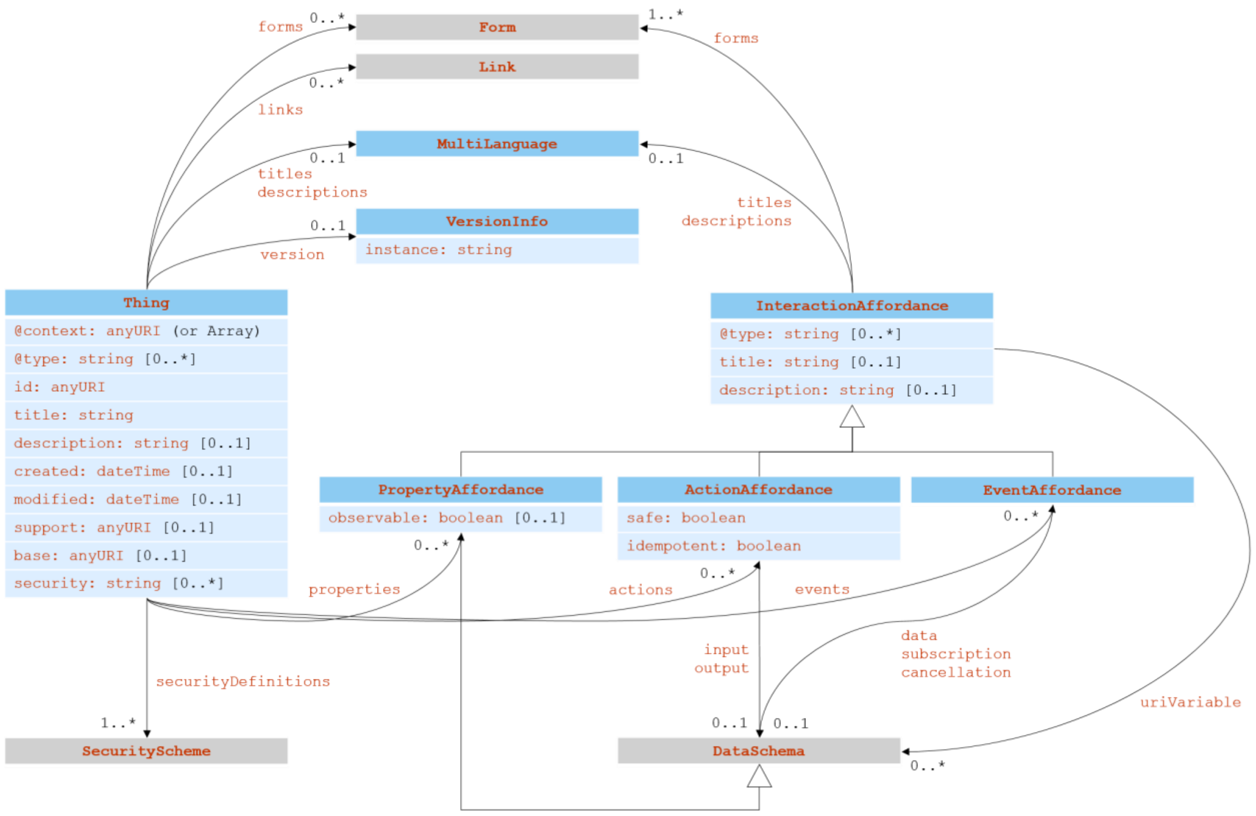
For example, the WoT Thing Description supports the HTTP protocol binding and includes the HTTP RDF vocabulary definitions from HTTP Vocabulary in RDF 1.0. This vocabulary can be directly used within WoT Thing Description instances in creating concrete bindings to HTTP.

Support for other protocols is enabled by the context extension mechanism provided by JSON-LD. JSON-LD is the default serialization format of the WoT Thing Description. The Web of Things (WoT) Protocol Binding Templates, which is a W3C WoT Editor’s Draft as of the time of this writing, defines how to indicate and specify the use of other protocols such as CoAP, MQTT and OPC UA in the WoT Thing Description.

A.3 Core Characteristics

A.3.1 Ontology

The UML diagram shown below gives an overview of the WoT Thing Description Information Model.



Figure_Apx 1: WoT information model WoT Information Model

The WoT Thing Description specification defines an information model based on a semantic vocabulary formulated in Web Ontology Language (OWL). The WoT Thing Descriptions provide rich metadata for Things in a way that is both human-readable and machine-understandable.

A WoT Thing Description describes Thing instances with general metadata such as name, ID, descriptions, and also can provide relation metadata through links to related Things or other documents.

The WoT Thing Descriptions also contain metadata on *Interaction Affordances*, which show and describe the possible choices to clients, thereby suggesting how clients may interact with the Thing. There are many types of potential affordances, but W3C WoT defines three types of Interaction Affordances: Properties, Actions, and Events. A fourth Interaction Affordance is navigation, which is already available on the Web through linking.

Today, typical existing API descriptions comprise a static mapping from an application intent to a resource address, method, request payload structure, response payload structure and expected errors. This imposes a tight coupling between Web client and Web service.

The Interaction Model of W3C WoT introduces an intermediate abstraction that formalizes the mapping from application intent to concrete protocol operations and also narrows the possibilities how Interaction Affordances can be modeled.

In addition, The WoT Thing Descriptions contain public security configuration metadata and communications metadata defining protocol bindings.

A.3.2 CONTEXTUAL DATA

The WoT Thing Description ontology is meant to be integrated into a larger ontology, as it would not suffice to describe physical world objects alone. An ontology is generally designed for a specific domain of application, like transportation or home automation. For the latter domain, the WoT Thing Description ontology can e.g. be integrated with SAREF. The Terms included in the WoT Thing Description ontology may be used to express what affordances should be expected from Things of a certain type, like saref:LightSwitch or, more generally, anything that has a state.

The WoT Thing Description provides Context Extension mechanism to extend Thing Descriptions with additional Vocabulary Terms. It allows for additional Vocabulary Terms to be used in a Thing Description instance. If the namespaces included by extension are based on Class definitions such as those provided by the RDF Schema or OWL, they can be used to annotate any Class instance of a Thing Description semantically by associating the instance to a such an external Class definition.

A.3.3 METAMODEL

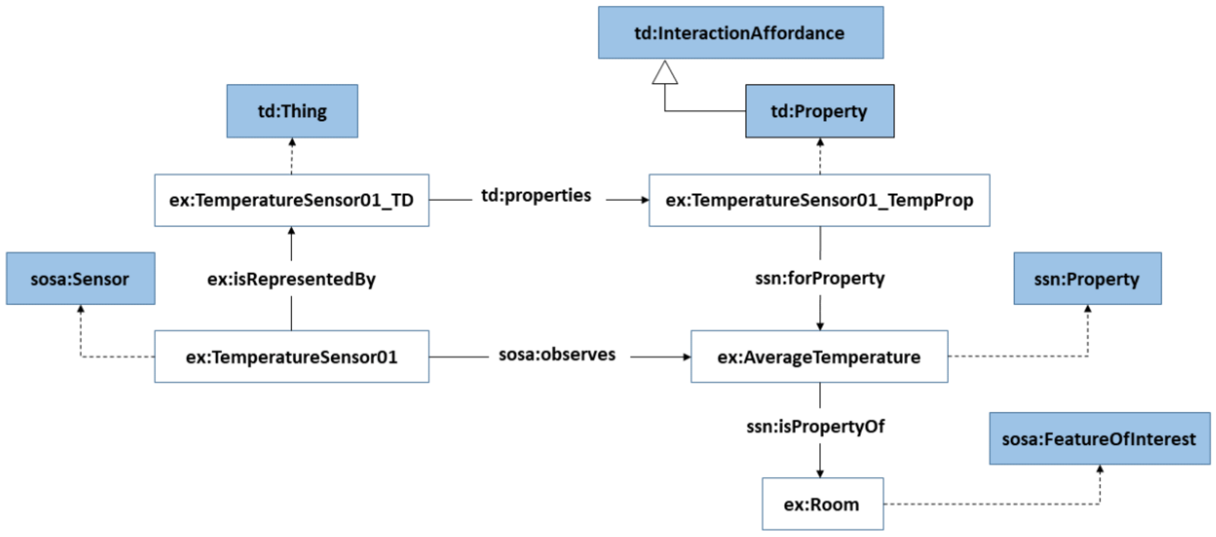
As described in the previous section, the WoT Thing Description ontology is meant to be integrated into larger ontologies each designed for a specific domain of application.

Since many ontologies for IoT are based on Observation and Measurement (O&M) model, the WoT Thing Description ontology is also meant to align with O&M-based ontologies such as the OGC/W3C SOSA/SSN ontology.

A *Thing* in WoT Thing Description (hereinafter simply called "*Thing*" or "*td:Thing*") is the abstraction of FoI (Feature of Interest), System or Platform, etc. Or, it can even be the abstraction of several such entities together.

For example, a lamp can be an instance of *ssn:FoI*, where *td:Thing* is a lamp Web representation. In addition, there could be many Web representation of such a lamp, which means a *td:Thing* is not the FoI itself. This is similar to the case of a person and their website.

Figure_Apx 2 describes an example in which a WoT Thing Description (i.e. ex:TemperatureSensor01_TD) is a Web representation of the sensor (i.e. ex:TemperatureSensor01).



Figure_Apx 2: WoT sensor representation

A.3.4 Observation & Measurement

A.3.4.1 Time

The WoT Thing Description ontology uses the dateTime datatype for value space constraints, and uses RFC3339 convention for constraining the lexical form.

A.3.4.2 Location

At the time of this writing, the WoT Thing Description ontology as well as SOSA/SSN is neutral as to the definition of location data. The WoT Working Group is currently discussing describing more on the use of location data in WoT Thing Description.

There is a W3C Working Note "*Spatial Data on the Web Best Practices*" published by the W3C Spatial Data on the Web Working Group, which describes the best practices related to the publication of location information on the Web. The practices described in the document can be used with the WoT Thing Description.

A.3.4.3 Sensors

See Section *Meta Model* for how a WoT Thing Description can represent a sensor.

A.3.4.4 Data Format

The WoT Information Model contains vocabulary (and corresponding ontology) for Data Schema. The Data Schema Vocabulary is compatible with a very common subset of the terms defined by JSON Schema.

The Data Schema is intended to be data-format neutral. It is designed to work well with a variety of data formats including JSON, XML, CBOR, EXI, etc. The WoT Binding Templates document further defines how the Data Schema binds to XML Schema. There are experiments underway by the WoT Working members to apply the Data Schema to other data formats such as CSV and ASN.1.

A.3.5 Actuators and tasking

A.3.5.1 Actuators

The WoT Thing can be a Web representation of an actuator. The section *Meta Model* describes how a sensor can be represented by a WoT Thing. Actuators can be similarly represented by a WoT Thing.

A.3.5.2 Tasking

For simple use cases such as that involves light switch in a room, the WoT Thing Description can be annotated with external ontology terminologies, so it can be discovered and operated by clients based on the application terminologies. This enables certain level of tasking by the client software, including those that run on gateways and clouds.

For more complex use cases involving sensors and actuators, it is considered a good practice to separate the state from interaction affordances, as illustrated in the *Meta Model* section. This enables deeper semantic understanding of the system that the WoT Things are part of, which in turn allows to make applications more autonomous in coordinating entities including sensors and actuators.

A.3.6 Security

Security metadata for accessing Web Things are described in WoT Thing Description. The WoT Thing Description by default supports a selection of well-established security mechanisms commonly used in the Web, such as Basic, Digest, Bearer and OAuth 2.0. The WoT Thing Description defines metadata for each of those security mechanisms describing the configuration of a security mechanism. This metadata includes information for authentication/authorization and payload encryption. Other security mechanisms can be used through Thing Description context extension mechanism.

A.3.6.1 Authorization

See the section preamble above.

A.3.6.2 Authentication

See the section preamble above.

A.3.6.3 CONFIDENTIALITY

See the section preamble above.

A.4 IIOT SYSTEM INFORMATION MODELS

A.4.1 POSITION IN SYSTEM LIFE CYCLE

The WoT Things typically goes through life cycles containing phases such as development, production, deployment, operation, reconfiguration, termination, etc. Each of those phases further consists of sub-phases. For example, the Thing needs to be published for discovery as part of the operation before the operation can be discovered and deliver its services to WoT clients.

The WoT Thing Description, in its initial specification release, is concerned mainly with the operation phase of the lifecycle.

A WoT Runtime needs to provide certain operations to manage the lifecycle of Things, or more precisely their software abstractions and descriptions. A lifecycle management (LCM) system may encapsulate those lifecycle operations and use internal interfaces to realize the lifecycle management. The details of such operations vary among different implementations. The WoT Scripting API includes certain LCM functionality, and hence represents one possible implementation of such a system.

Further lifecycle support including development, production, etc. is considered for the next version of WoT specifications.

A.4.2 POSITION IN (ARCHITECTURE LAYER, SYSTEM LAYER, HIERARCHY LEVEL):

As described in the *Meta Model* section, a *Thing* in WoT Thing Description is the abstraction of Feature of Interest (FoI), System or Platform, etc. Or, it can be the abstraction of several such entities together. It is therefore neutral to the layers of IIoT systems.

The WoT Thing Description ontology is meant to be integrated into larger ontologies each designed for a specific domain of application. Each specific domain (e.g. Plattform Industrie 4.0) needs to be concerned with the layers/levels of the architecture with its unique requirements and use cases.

Annex B OPEN GEOSPATIAL CONSORTIUM SENSORTHINGS API INFORMATION MODEL

B.1 BACKGROUND INFORMATION

SensorThings API is a standard of the Open Geospatial Consortium OGC comprising Part 1: Sensing (OGC 15-078r6) and Part 2: Tasking Core (OGC 17-079r1):

https://www.opengeospatial.org/standards/sensorthings

An overview of Part 1 is given in the paper [KOT 2018].

B.2 Services Needed from Connectivity Framework Layer

B.2.1 Data Format

JSON

B.2.2 Interaction Abstraction

B.2.2.1 Services

RESTful services for Create, Read, Update and Delete operations, notification of updates with MQTT extension. Enhanced access through advanced queries, e.g. spatial-temporal and filter queries, pagination (to return a subset of a large data set)

B.2.2.2 Quality of service

Not defined

B.2.2.3 Design Patterns

SensorThings API uses the pattern Request-Response (i.e. Request-Reply). The MQTT extension uses the pattern Publish-Subscribe.

B.2.2.4 Protocols

HTTP POST, GET, PATCH, and DELETE, MQTT to access the information model

B.3 Core Characteristics

B.3.1 Ontology

The entity types of the SensorThingsAPI Part 1 are shown inFigure_Apx 3.

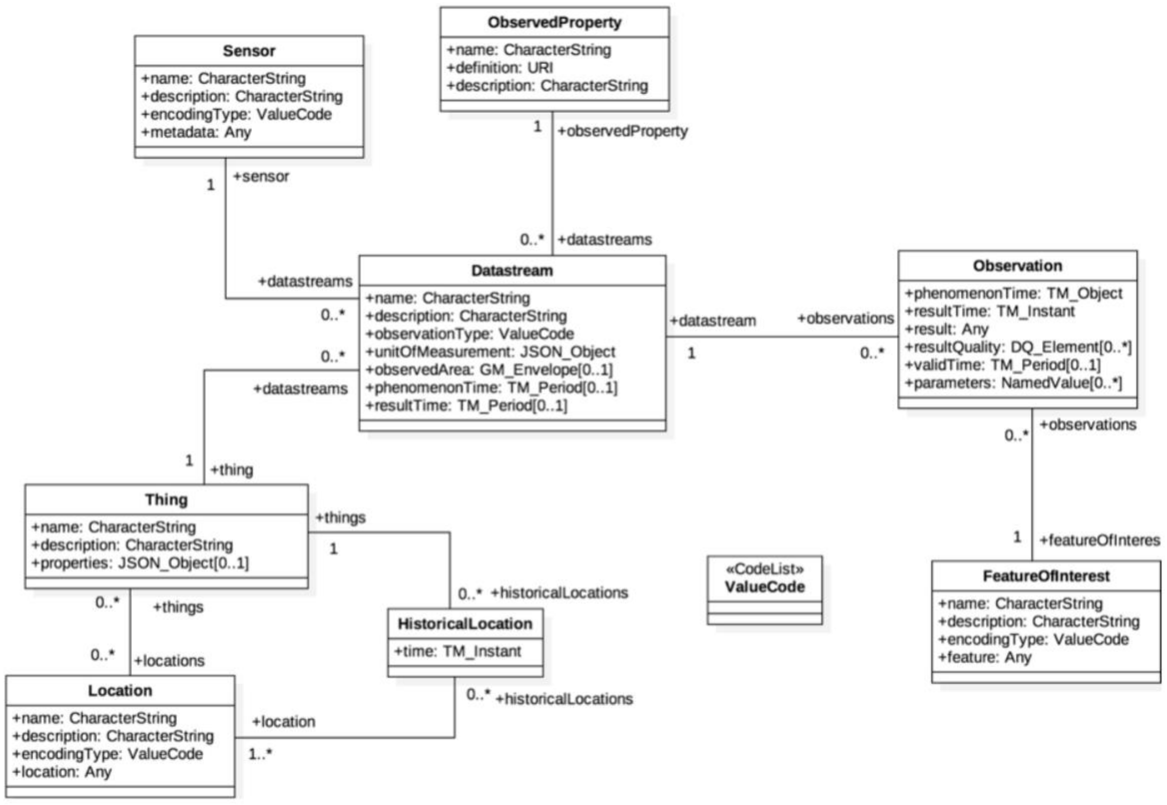
A '*Thing*' is the central entity in the data model. It can be physical or virtual, and is equipped with one or more 'Sensor' to collect Observations. Depending on the use case this can be the object being observed, or the sensor platform, such as a satellite.

A *Thing* is defined to be an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks [according to ITU-T Y.2060][1].

[1] *https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060*

- A *Thing* may have a Location and several Location representations, e.g. latitude/longitude, street address or reference to a building section.
- A *Thing* may have *HistoricalLocations* to trace previous locations (useful in case of a moving Thing).
- A Thing has zero-to-many *DataStreams*
- A *Thing* has a freely definable set of properties which can be used to add semantic information about the Thing. The properties are encoded as a JSON object containing user-annotated properties as key-value pairs.

Each Observation has a *FeatureOfInterest* that it observes, as well as a *Datastream*.



Figure_Apx 3: Sensing Entities in SensorThingsAPI Part 1

DataStream:

- A *Datastream* groups a collection of Observations measuring the same phenomenon (called the *ObservedProperty*) and produced by the same Sensor for the same Thing.
- observationType (e.g. category, count, measurement, truth,..)
- observedArea (bounding polygon of coordinates)
- resultTime (The temporal interval of the result times of all observations belonging to this Datastream. ISO 8601 encoding)

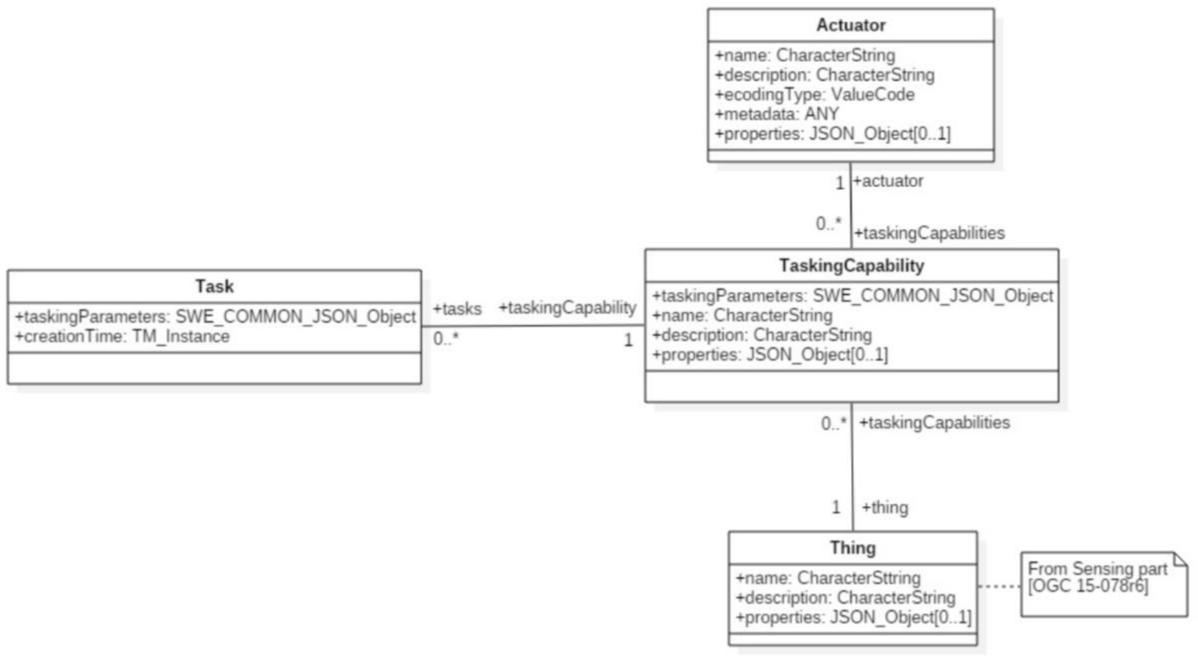
- unitOfMeasurement (expressed as a Unified Code for Unit of Measure (UCUM), e.g. degree Celsius)

Observation:

- An *Observation* is the act of measuring or otherwise determining the value of an *ObservedProperty*. An *Observation* results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the *FeatureOfInterest* of the *Observation*. The concept *Observation* is according to the Observations and Measurements (O&M) model [OGC 10-004r3[1] and ISO 19156:2011 Geographic information -- Observations and measurements].
- result (estimate of property value, e.g. temperature, wind speed and direction)
- phenomenonTime (time instance or interval when the observation was made)
- resultTime (when the result was generated)
- resultQuality may be defined to specify accuracy or validation status
- validTime (The time period during which the result may be used)
- parameters (used to make observation)
- Refers to exactly one *FeatureOfInterest*. In IoT, *FeatureOfInterest* is typically the Location of the *Thing*, and *Thing* is carrier of sensor. For example, the *FeatureOfInterest* of a WiFi-connected thermostat can be the Location of the thermostat (e.g. the living room where the thermostat is located). However, it is also permissible to consider the *FeatureOfInterest* to be an object to be observed (e.g. machine, robot, lake etc.) and the *Thing* to be the (mobile) carrier of a sensor such as a camera.

Figure_Apx 4 shows the tasking entities defined in SensorThingsAPI Part 2.

[1] *https://www.opengeospatial.org/standards/om*


Actuator
+name: CharacterString
+description: CharacterString
+ecodingType: ValueCode
+metadata: ANY
+properties: JSON_Object[0..1]
1 +actuator
0..* +taskingCapabilities
TaskingCapability
+taskingParameters: SWE_COMMON_JSON_Object
+name: CharacterString
+description: CharacterString
+properties: JSON_Object[0..1]
Task
+taskingParameters: SWE_COMMON_JSON_Object
+creationTime: TM_Instance
+tasks +taskingCapability
0..* 1
0..* +taskingCapabilities
1 +thing
Thing
+name: CharacterSttring
+description: CharacterString
+properties: JSON_Object[0..1]
From Sensing part [OGC 15-078r6]

Figure_Apx 4: Tasking Entities in SensorThingsAPI Part 2

The *TaskingCapability* entity contains information about the capabilities of the taskable device. It contains all the parameters that can be used for controlling the device; the parameters are encoded in JSON.

The *Task* entity contains the parameter detail of the specific control action that is to run on the Actuator. The taskingParameters property of a Task describes values for optional and mandatory tasking parameters. Clients use the definition to provide corresponding tasking parameter values. To ensure common understanding between client and server, a common exchange protocol is used to express both descriptions and tasking parameter values1.

An *Actuator* is a device that can be controlled or tasked. Typical actuators in the IIoT domain are switches, motors, valves, robots and cameras. The Actuator entity contains information and metadata about the taskable actuator. Each TaskingCapability has one Actuator and defines the parameters that can be set/tasked for the Actuator.

B.3.2 CONTEXTUAL DATA

Contextual and any other meta-data can be added to the property set of a *Thing*, *Actuator* or *TaskingCapability*, the parameter set of an *Observation* or to the description attribute of an entity.

[1] *https://www.opengeospatial.org/standards/swecommon*

B.3.3 METAMODEL

The entities and their relations are defined in a UML model (cf. figures above).

B.3.4 OBSERVATION & MEASUREMENT

B.3.4.1 ANNEX A SECTION 1, SUBSECTION 3, SUB-SUBSECTION 2

SensorThingsAPI is based on the Observation and Measurements Model (O&M v2.0) specified in OGC 10-004r3 and ISO 19156:2011 (Geographic information -- Observations and measurements).

B.3.4.2 TIME

There are concepts of time instance, time interval and time zones. Time values are encoded as ISO 8601 strings.

B.3.4.3 LOCATION

The Location of a *Thing* may be a point, line or polygon or a collection thereof. The Location may be encoded in GeoJSON[1] with longitude/latitude coordinates and optionally altitude according to WGS 84. In the future it is planned to add IndoorGML[2] and 'Well-known text representation of coordinate reference systems'[3].

B.3.4.4 SENSORS

SensorThings API Part 1 has an abstract concept of a sensor (cf. Figure_Apx 3).

B.3.4.5 DATA FORMAT

JSON

B.3.5 ACTUATORS AND TASKING

B.3.5.1 ACTUATORS

SensorThingAPI Part 2 has an abstract concept of an actuator (cf. Figure_Apx 4).

B.3.5.2 TASKING

SensorThingAPI Part 2 defines scheduling and configuration of sensors and actuators.

B.3.6 SECURITY

There is on-going work to extend the standard to define roles and access permissions.

B.3.6.1 AUTHORIZATION

[1] *https://tools.ietf.org/html/rfc7946*
[2] *https://www.opengeospatial.org/standards/indoorgml*
[3] *https://www.opengeospatial.org/standards/wkt-crs*

Not in scope; to be leveraged from other IoT capabilities.

B.3.6.2 Authentication

Not in scope; to be leveraged from other IoT capabilities.

B.3.6.3 Confidentiality

Not in scope; to be leveraged from other IoT capabilities.

B.4 IIoT System Information Model Types

B.4.1 Position in System Life Cycle

SensorThingsAPI is applicable to the asset production and asset usage / maintenance phases.

B.4.2 Position in (Architecture Layer, System Layer, Hierarchy Level):

SensorThingsAPI can be deployed on field devices (sensors and actuators, controllers) in production or operation as well as on products (e.g. with sensors to monitor product usage or transport).

Annex C PLATTFORM INDUSTRIE 4.0 ASSET ADMINISTRATION SHELL INFORMATION MODEL

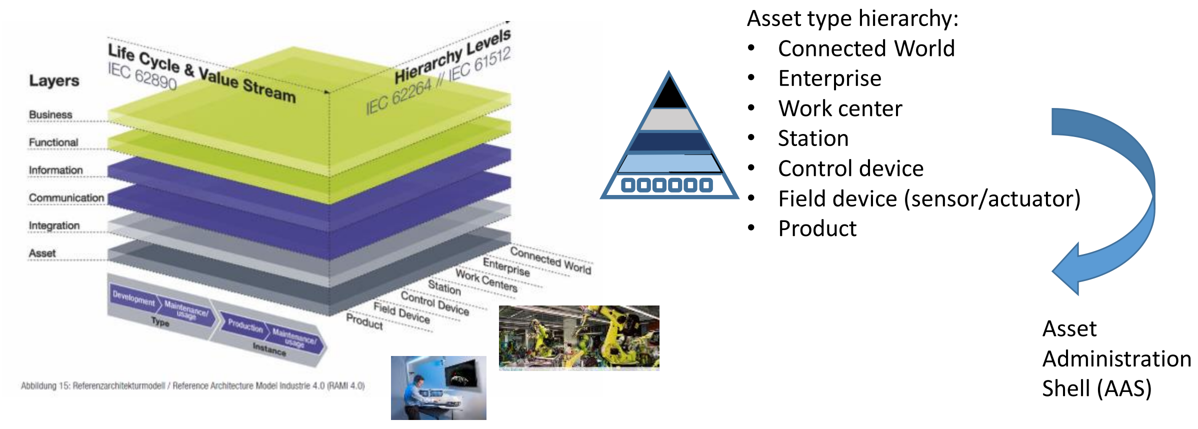
Plattform Industrie 4.0 has published its “Details of the Asset Administration Shell” – Part 1: The exchange of information between partners in the value chain of Industrie 4.0 [Plattform Industrie 4.0-2020a][1] and Part 2*:* Interoperability at Runtime – Exchanging Information via Application Programming Interfaces [Plattform Industrie 4.0-2020b][2].

The Asset Administration Shell (AAS) aims to create a uniform logical interface to assets such as manufacturing machines and factories. An asset is defined to be a “physical or logical object owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization”. An asset is represented in the virtual world by its administration shell and may have multiple virtual representations, i.e. multiple administration shells. An asset can be an idea, software program, an archive, service or physical item. An asset has a lifetime and a clearly defined identity.

[1] *https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html*

[2] *https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html*

In I4.0 the hierarchical levels in the framework RAMI4.0 are (from top to bottom): Connected World, Enterprise, Work Centre, Station, Control Device, Field Device supplemented and Product. The levels Connected World and Product go beyond the classic factory levels in the automation pyramid and represent a key aspect of I4.0.



Plattform Industrie 4.0/Hrsg. BITKOM, VDMA, ZVEI:
Umsetzungsstrategie Industrie 4.0 – Ergebnisbericht, Berlin, April 2015

Figure_Apx 5: Reference Architecture Model Industrie 4.0 (RAMI4.0)

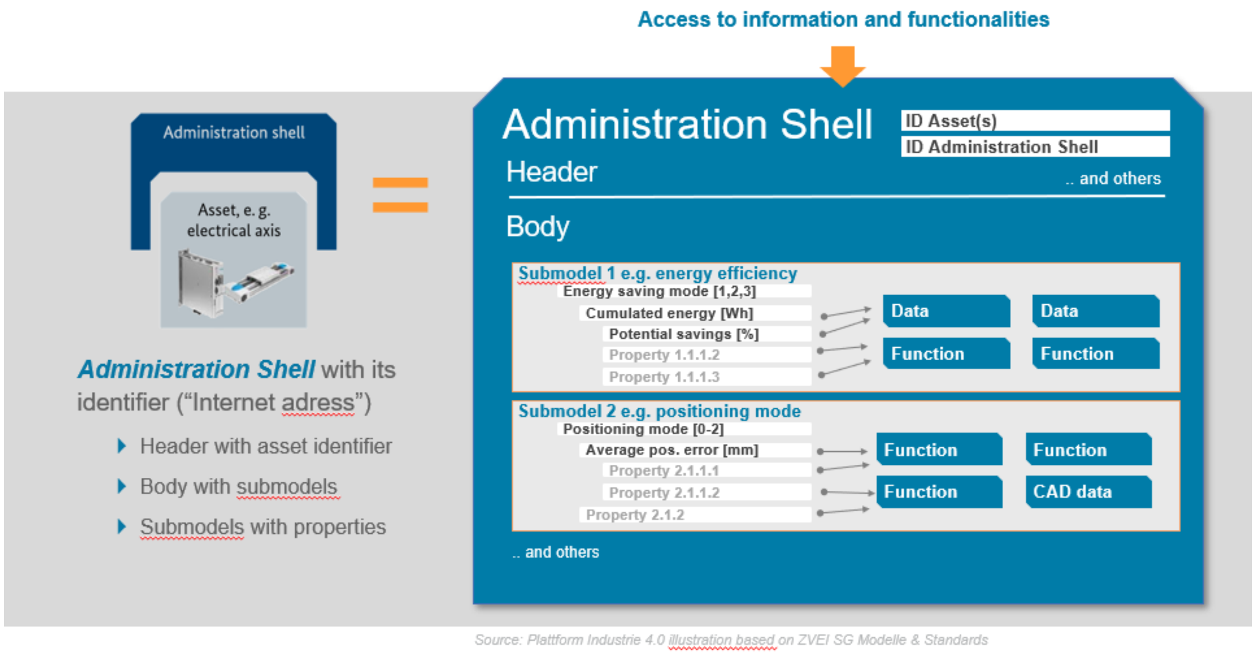
An Industrie 4.0 component is the combination of an Asset and its Administration Shell.

The document "Details of the Asset Administration Shell –Part 1" specifies the information model proposed for the exchange of information between partners in a value chain of Industry 4.0. In I4.0, the AAS may comprise several submodels, each with a structured set of properties (in a hierarchy): "Each submodel contains a structured quantity of properties that can refer to data and functions. A standardized format based on IEC 61360-1/ ISO 13584-42 is envisaged for the properties[1]. Thus, property value definition shall follow the same principles as also ISO 29002-10[2] and IEC 62832-2[3]. Data and functions may be available in various, complementary formats." The intention is that submodels will be standardized by organizations such as the Mechanical Engineering Industry Association (VDMA, *https://www.vdma.org/en/*). Full semantic interoperability will require standardized submodels. Nevertheless, the AAS information model provides a framework for facilitating the integration of assets into applications.

[1] i.e. hierarchy of classification from a collection of classes, each of which represents a technical concept of the domain / defining classes and properties of parts which characterize a part independently of any particular supplier-defined identification

[2] conceptual information model and exchange file format for characteristic data (UML and XML for file format)

[3] Requirements for model elements of digital factory framework. It defines the nature of the information provided by the model elements, but not the format of this information.


Access to information and functionalities
Administration shell
Asset, e. g. electrical axis
Administration Shell with its identifier ("Internet adress")
Header with asset identifier
Body with submodels
Submodels with properties
Administration Shell
Header
ID Asset(s)
ID Administration Shell
.. and others
Body
Submodel 1 e.g. energy efficiency
Energy saving mode [1,2,3]
Cumulated energy [Wh]
Potential savings [%]
Property 1.1.1.2
Property 1.1.1.3
Data
Data
Function
Function
Submodel 2 e.g. positioning mode
Positioning mode [0-2]
Average pos. error [mm]
Property 2.1.1.1
Property 2.1.1.2
Property 2.1.2
Function
Function
Function
CAD data
.. and others
Source: Plattform Industrie 4.0 illustration based on ZVEI SG Modelle & Standards

Figure_Apx 6: Asset Administration Shell of Platform Industrie 4.0

This document also includes a mapping of the AAS to XML, JSON, RDF, OPC UA and AutomationML. Further concise overview information on the AAS can be found in [IIC-I4.02020][1].

For an overview of the standards used in I4.0, see http://i40.semantic-interoperability.org/. The standards include OPC UA and AutomationML, both of which play a central role in the IIC testbed Smart Factory Web.

C.1 SERVICES NEEDED FROM CONNECTIVITY FRAMEWORK LAYER

C.1.1 DATA FORMAT

The AAS can be stored in a file in the package file format for the Asset Administration Shell (AASX). An AAS can also be mapped to the technology representations XML, JSON, RDF, OPC UA and AutomationML which each have their own serialization. There are XML and JSON schemas to support the respective representations.

C.1.2 INTERACTION ABSTRACTION

Platform Industrie 4.0 is preparing a draft directive VDI/VDE[2] 2193 on a language for the interaction between I4.0 components. Part 1 species the I4.0 language and the structure of messages. Part 2 defines an interaction protocol for the use case tendering. Details of the Asset

[1] *https://www.iiconsortium.org/pdf/Digital-Twin-and-Asset-Administration-Shell-Concepts-and-Application-Joint-Whitepaper.pdf*

[2] VDI: Association of German Engineers; VDE: Association for Electrical, Electronic & Information Technologies

Administration Shell – Part 2 defines an API for the Asset Administration Shell at a level above the connectivity framework layer.

C.2 Core Characteristics

C.2.1 Ontology

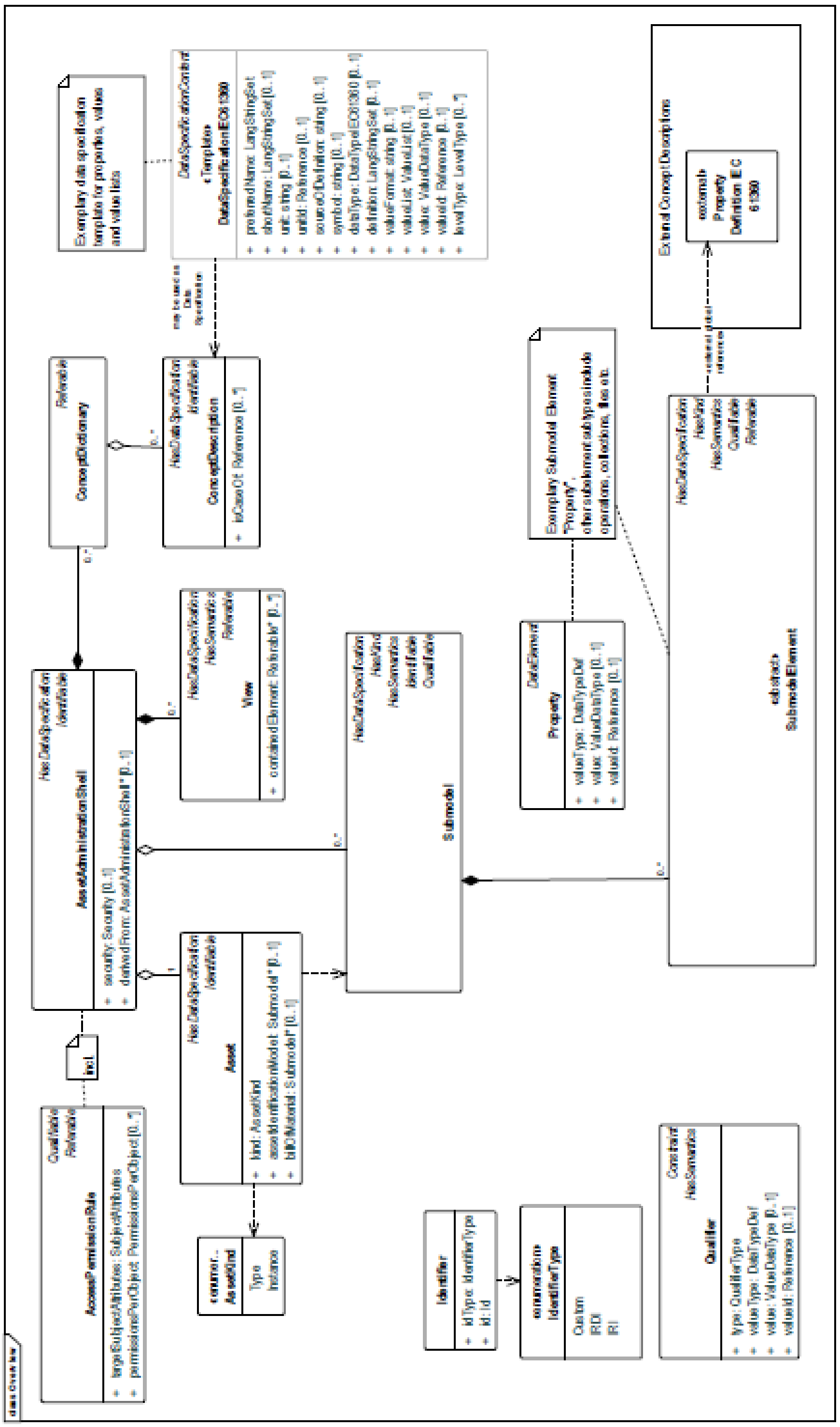
The UML meta-model is shown below in section C.2.3. The AAS meta-model is mapped onto an ontology expressed in RDF.

C.2.2 Contextual Data

The sub-models of the AAS allow for in principle arbitrary properties to express context information. All elements may reference a concept dictionary such as eCl@ss (*https://www.eclass.eu/*) or IEC CDD (Common Data Description; IEC 61360)

C.2.3 Metamodel

AAS has a metamodel defined in UML.



Figure_Apx 7: AAS Metamodel

C.2.4 Observation & Measurement

The AAS is not designed to handle data streams of observations or measurements, but rather to describe the devices making the observations such as sensors.

C.2.4.1 Time

Time is encoded following the ISO 8601 format in time zone UTC.

C.2.4.2 Location

Location is not defined explicitly, but can be included as a self-defined property.

C.2.4.3 Sensors

Sensors are a type of asset and hence may have an AAS. However, submodels for sensors are not yet standardized. An AAS may reference properties defined in eCl@ss.

C.2.4.4 Data Format

Not defined

C.2.5 Actuators and Tasking

C.2.5.1 Actuators

Actuators are a type of asset and hence may have an AAS. However, submodels for actuators are not yet standardized.

C.2.5.2 Tasking

A device that could be tasked such as a valve, motor of camera may be represented with an AAS. The AAS may contain submodels with tasking parameters. These are not yet standardized.

C.2.6 Security

The AAS defines a meta-model for attribute-based access control. An AAS has security attributes to describe Access Control Policy Points including access permission rules and certificates.

C.2.6.1 Authorization

C.2.6.2 Authentication

Identifiers are defined for the Asset Administration Shell (AAS), assets, submodels (instances and templates), property definitions and concept descriptions in external repositories such as eCl@ss and IEC CDD.

C.2.6.3 Confidentiality

C.3 IIoT System Information Model Types

C.3.1 Position in System Life Cycle

The AAS is designed to cover the complete system life cycle of RAMI4.0.

C.3.2 Position in (Architecture Layer, System Layer, Hierarchy Level):

The AAS may be applied in all layers of RAMI4.0.

Annex D OPC UA BASE INFORMATION MODEL

OPC Unified Architecture (OPC UA) is defined in the multi-part standard IEC 62541. The specifications are maintained and provided by the OPC Foundation.[1] In this section we describe the characteristics of the base information model of OPC UA. For full details, refer to OPC UA Part 3 (Address Space Model) and OPC UA Part 5 (Information Model).

In the following text a word in italics with an initial uppercase letter (such as *Node*) denotes a term defined in the OPC UA standard.

A growing family of so-called Companion Specifications[2] extends the base information model with elements specific to certain classes of machines and applications. An OPC UA Companion Specification defines *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent specific semantics. OPC has a template for creating a Companion Specification. A wide range of industries define their standards for information models building on the OPC UA base information model. The verticals covered by the Companion Specifications are in diverse process control domains, including manufacturing, oil & gas, building automation and utilities. For AutomationML (IEC 61714) there is a Companion Specification "OPC Unified Architecture for AutomationML" (DIN SPEC 16592) that defines how to generate an OPC UA Server from an AutomationML model; AutomationML defines an XML-based data format for the storage and exchange of plant engineering information in a heterogeneous, multi-disciplinary tool landscape.

The joint ZVEI, VDMA and OPC Foundation "I4AAS OPC UA" Working Group[3] is developing an OPC UA Information Model for the Plattform Industrie 4.0 Asset Administration Shell (I4AAS) in a "OPC UA companion specification I4AAS".

[1] *https://opcfoundation.org/developer-tools/specifications-unified-architecture*
[2] *https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/*
[3] *https://opcfoundation.org/markets-collaboration/i4aas/*

D.1 Services Needed from Connectivity Framework Layer

D.1.1 Data Format

OPC UA defines mappings for *DataEncodings*, *SecurityProtocols* and *TransportProtocols*. The *DataEncodings* specify the serialization of the information model in UA Binary, UA XML or UA JSON. The security protocol UA Secure Conversation defines how serialized data is encoded and transferred over a Secure Channel. The underlying Transport Layer applies the *TransportProtocols* UA TCP, HTTPS and AMQP to transfer the secured message. OPC UA defines an abstract Connection Protocol as a duplex channel between clients and servers. The OPC UA Connection Protocol defines the byte structure of a message on the transmission medium and interacts with the Secure Channel.

D.1.2 Interaction Abstraction

D.1.2.1 Services

OPC UA Part 4 defines services grouped into sets of related services.

Service Set	Usage
SecureChannel Service Set	retrieve endpoint and security configuration to establish a secure connection
Session Service Set	create and administrate user-specific connection between application
NodeManagement Service Set	modify the server's address space (if permitted)
View Service Set	navigate and follow (hierarchical) references in the server's address space, search for and filter information
Attribute Service Set	read and write Node attributes of (an) node(s), incl. the value attribute, historical data and events
Method Service Set	invoke methods which a server provides at the nodes in its address space
MonitoredItem Service Set	create attributes of nodes to be monitored by the server
Subscription Service Set	create, modify, or delete monitored items
Query Service Set	perform a filtered search for information in the server's address space

Table-Apx 1: OPC UA Service Sets

D.1.2.2 Quality of service

There are no explicit Quality of Service parameters.

D.1.2.3 Design Patterns

OPC UA defines request-response (client-server) and publisher-subscriber (OPC UA Part 14).

D.1.2.4 Protocols

OPC UA defines transport protocols over the UA Connection Protocol, TCP, SOAP/HTTP, HTTPS, and WebSockets.

D.2 Core Characteristics

The OPC UA information model is a connected graph with Nodes, Attributes of Nodes and References (directed edges) between Nodes. The nodes are coupled to OPC UA functionalities: Data Access, Historical Data Access, Alarms & Events and Commands.

The object-oriented modelling paradigm is applied not only to objects, but also to variables, data types and references. Objects typically represent system components (hardware or software) and are structured hierarchically. The information model of an OPC UA server is divided into groups of *Nodes* called *NodeSets*. A *Node* belongs to exactly one *NodeSet*. The base *NodeSet* defined in OPC UA Part 3 and Part 5 is the basis for all other *NodesSets*. Every *Node* has a URL to identify its unique name space. The name space *http://opcfoundation.org/UA/* is contained in all *Nodes* defined in the OPC UA standard[1]. OPC UA Part 6 describes the XML schema of the XML files of all *NodeSets*. An OPC UA server stores a list of all used NodeSet name spaces in a *NamespaceArray*. The *Nodes* and their references can then just index the respective element of the *NamespaceArray*. A *Node* in a *NodeSet* may be available in several OPC UA servers, but with different indices depending on the server. The name space index 0 is, however, reserved for the base *NodeSet* of the OPC Foundation.

Each *Node* contains attributes for identification, description or definition of access rights. The value of the attribute *NodeID* is unique over the complete server information model, i.e. over all *NodeSets* of the server. The *NodeID* is comprised of the name space URL (or index) and an identifier that is unique within the *NodeSet*. There are four types of identifiers:

- Numeric (a positive integer)
- String (max. 4096 Byte, case sensitive)
- Guid of format 00000000-0000-0000-0000-000000
- Opac: a free format of data type ByteString with max. 4096 Byte

[1] *https://opcfoundation.org/UA/schemas/1.04/Opc.Ua.NodeSet2.xml*

The *BrowseName* of a *Node* is a language independent and human readable name. The *BrowseName* of a *Node* is not necessarily unique in a *NodeSet*, but unique amongst Nodes of the same hierarchy level. The sequence of *BrowseNames* (*BrowseNamePath*) to a *Node* is unique if the information model is hierarchical. The *DisplayName* of a *Node* is used to visualize a *Node*. A *Node* may have several *DisplayNames*; they are language dependent and may include a language reference. Similarly, a *Node* may have language dependent descriptions.

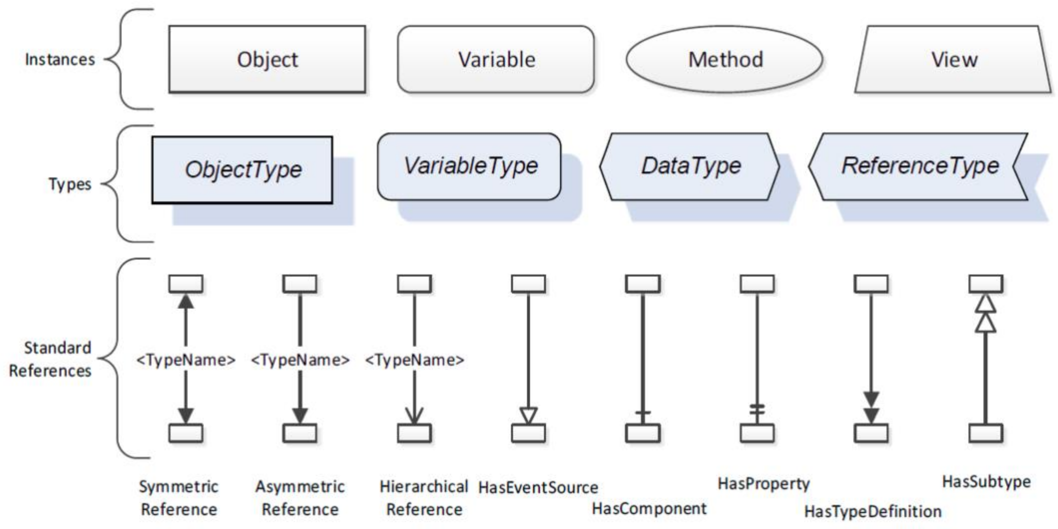
The optional property WriteMask can be used to define if the DisplayName or NodeID may be changed at run time. The property UserWriteMask defines the rights of a connected client or user. The RolePermissions is an array of access rights of type RolePermissionType. These rights refer to the services of the server, e.g. browsing, deletion or creation of Nodes, read and write of historic data.

The OPC UA information model has 8 Node types (*NodeClass*) as defined in OPC UA Part 3:

NodeClass	Usage
Object	Represent an object made up of variables, methods and further objects; the object may be a system, system component, real-world object or software object.
ObjectType	Define requirements for Object Nodes (contained variables, methods etc)
Variable	Represent a value (scalar, array, multi-dimensional array)
VariableType	Define requirements for Variable Nodes (value type, array size etc)
DataType	Define simple and structured data types
ReferenceType	Define a type for references between nodes (hierarchical or non-hierarchical)
Method	Define callable remote procedures
View	Provide access to a subset of nodes

Table-Apx 2: *NodeClass* types

OPC UA Part 3 Annex B provides a UML Model of the Address Space Model. OPC UA also defines a graphical notation for the OPC UA information model (in the normative OPC UA Part 3:Annex C).



Figure_Apx 8: The OPC UA Information Model Notation (OPC UA Part 100, Fig. 3)

D.2.1 Ontology

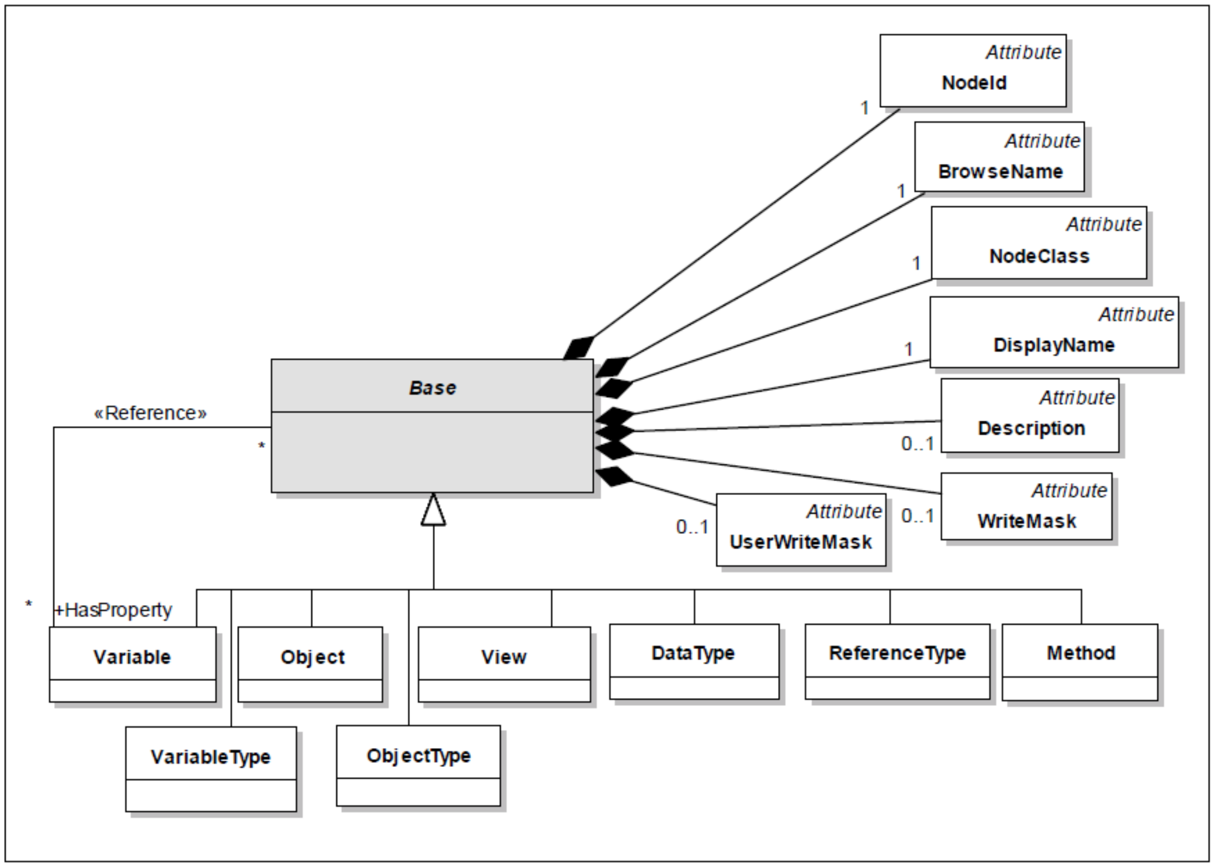
Defined through the UML meta-model as below.

D.2.2 Contextual Data

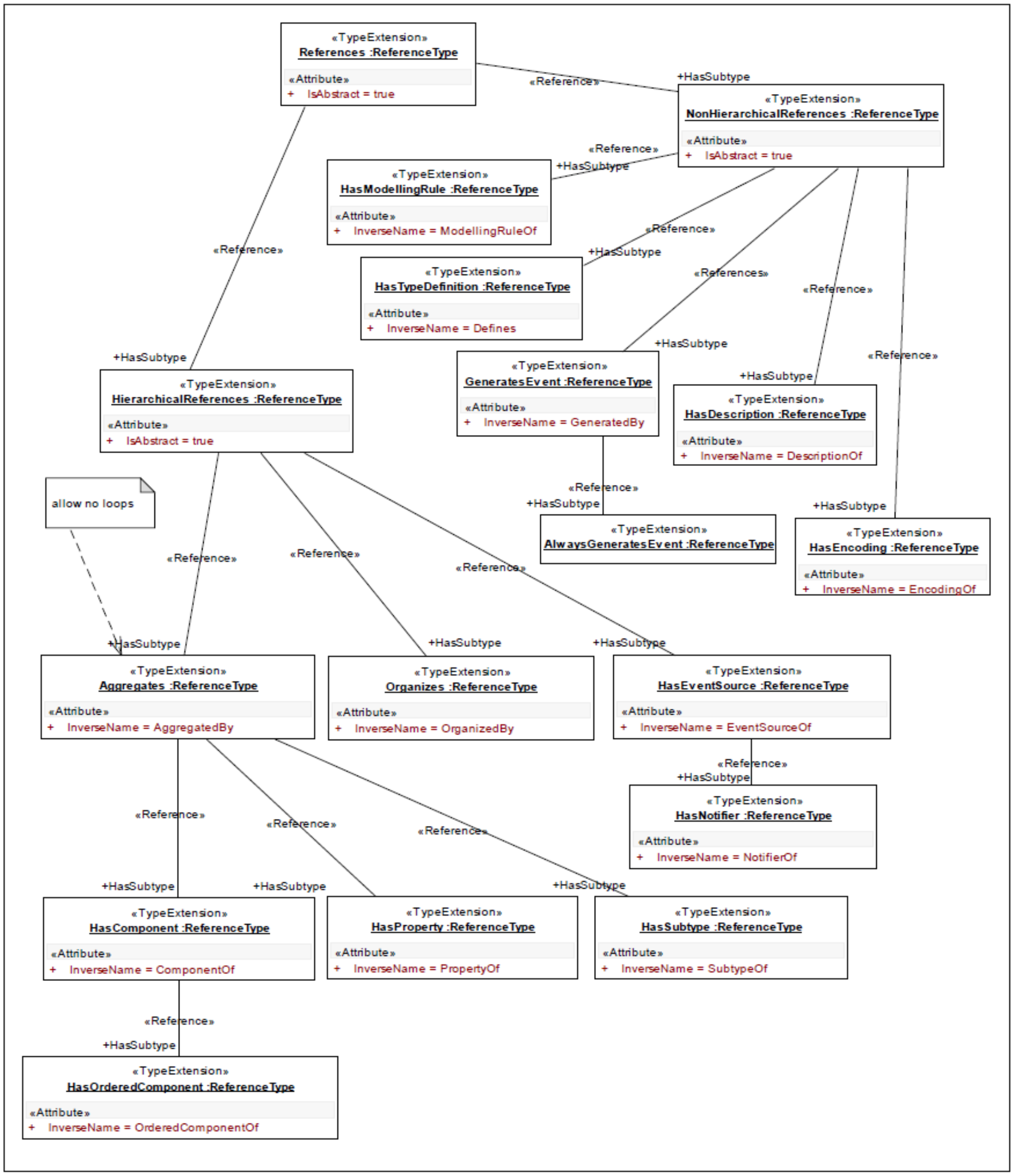
Context information may be included through the *Properties* of *Nodes* such as *Objects* and *DataVariables* and *Attributes* of *Nodes*. *Properties* characterize what a *Node* represents. *Attributes* define additional metadata for all *Nodes* of a *NodeClass*.

D.2.3 Metamodel

The informative Annex B of OPC UA Part 3 describes the OPC UA Meta Model as UML classes. This covers the meta-models of the Base model of *Nodes*, *ReferenceTypes*, *Attributes*, *Object* and *ObjectType*, *EventNotifier*, *Variable* and *VariableType*, *Method*, *DataType*. The UML for the Base model and *ReferenceTypes* are shown below as important parts of the Metamodel.



Figure_Apx 9: Base metamodel of *Nodes* (OPC UA Part 3, Fig. B.4)


«TypeExtension» References :ReferenceType
«Attribute» + IsAbstract = true
«TypeExtension» NonHierarchicalReferences :ReferenceType
«Attribute» + IsAbstract = true
«TypeExtension» HasModellingRule :ReferenceType
«Attribute» + InverseName = ModellingRuleOf
«TypeExtension» HasTypeDefinition :ReferenceType
«Attribute» + InverseName = Defines
«TypeExtension» HierarchicalReferences :ReferenceType
«Attribute» + IsAbstract = true
«TypeExtension» GeneratesEvent :ReferenceType
«Attribute» + InverseName = GeneratedBy
«TypeExtension» HasDescription :ReferenceType
«Attribute» + InverseName = DescriptionOf
«TypeExtension» AlwaysGeneratesEvent :ReferenceType
«TypeExtension» HasEncoding :ReferenceType
«Attribute» + InverseName = EncodingOf
allow no loops
«TypeExtension» Aggregates :ReferenceType
«Attribute» + InverseName = AggregatedBy
«TypeExtension» Organizes :ReferenceType
«Attribute» + InverseName = OrganizedBy
«TypeExtension» HasEventSource :ReferenceType
«Attribute» + InverseName = EventSourceOf
«TypeExtension» HasNotifier :ReferenceType
«Attribute» + InverseName = NotifierOf
«TypeExtension» HasComponent :ReferenceType
«Attribute» + InverseName = ComponentOf
«TypeExtension» HasProperty :ReferenceType
«Attribute» + InverseName = PropertyOf
«TypeExtension» HasSubtype :ReferenceType
«Attribute» + InverseName = SubtypeOf
«TypeExtension» HasOrderedComponent :ReferenceType
«Attribute» + InverseName = OrderedComponentOf
«Reference»
+HasSubtype

Figure_Apx 10: Predefined reference types (OPC UA Part 3, Fig. B.6)

References define a relationship between a source and a target *Node*. They are either hierarchical or non-hierarchical. Hierarchical references are used to create the structure of *Objects* and *Variables* and may not form a loop (cycle). Each *Node* has a hierarchical reference to a parent *Node*. Nonhierarchical references are used to create arbitrary associations and are essentially links to additional information about the source *Node*. For example, *Variables* have a *Reference* of type *HasTypeDefinition* to a *VariableType*.

There are constraints on the source and target *NodeClasses* of a *Reference* depending on its type. For example, for the reference type *HasComponent*, if the target *Node* is an *Object* or a *Variable*, the source *Node* shall be an *Object* or an *ObjectType*. Applications may define their own *ReferenceType* by creating subtypes of an existing *ReferenceType*.

D.2.4 OBSERVATION & MEASUREMENT

D.2.4.1 TIME

OPC UA has the simple data types *DateString*, *TimeString* and *DurationString* conforming to ISO 8601-2000. Events, server actions, state machine changes and data values may have a *TimeStamp*.

D.2.4.2 LOCATION

There is no explicit definition of location (of Objects) in the information model.

D.2.4.3 SENSORS

OPC UA Part 100 (Devices) defines the information model associated with *Devices*. A *Device* is defined to be an independent physical entity capable of performing one or more specified functions in a particular context and delimited by its interfaces. The device types include sensors, RemoteIO and programmable controllers. The information model includes metadata provided by the device vendor such as *Model*, *SerialNumber*, *HardwareRevision* and *SoftwareRevision*. In addition, there is a *DeviceHealth* Interface with *Properties* and *Alarms* to describe the health status of a *Device*. The communication interfaces and protocols of the *Device* can be described as well. The integration of *Devices* into server (host) applications is also covered.

D.2.4.4 DATA FORMAT

OPC UA Part 6 defines mappings of the information model to Binary, XML and JSON formats.

D.2.5 ACTUATORS AND TASKING

D.2.5.1 ACTUATORS

OPC UA Part 100 (Devices) defines several device types including RemoteIO and ProgrammableController (see section above on sensors). There are locking mechanisms to support control of actuators.

D.2.5.2 TASKING

Tasking can be modelled through *Method Nodes*. *Methods* have child *Variables* with *BrowseName InputArguments* and *OutputArguments*. Several *Nodes* may have a reference to the same *Method*.

D.2.6 SECURITY

D.2.6.1 AUTHORIZATION AND AUTHENTICATION

OPC UA Part 3 and Part 5 define user authorization and authentication with a role-based approach. *Roles* can be granted based on user identity, application identity or endpoint. There is a defined, extendable set of so-called well-known *Roles*: Observer, Operator, Engineer, Supervisor, ConfigureAdmin, SecurityAdmin, AuthenticatedUser and Anonymous.

OPC UA has audit parameters to be included in audit logs of access to *Nodes*.

The security mechanisms of OPC UA have been assessed by Kaspersky Labs and the German Office for Information Security BSI.[1]

D.2.6.2 CONFIDENTIALITY

Access rights to the *Nodes* (read, write, browse) can be restricted. The OPC UA data transport may be encrypted on an OPC UA SecureChannel over TCP.

D.3 IIOT SYSTEM INFORMATION MODEL TYPES

D.3.1 POSITION IN SYSTEM LIFE CYCLE

OPC UA can be applied in all phases of the IEC 62890 life-cycle referenced by RAMI4.0:

- development and usage / maintenance of asset types
- production and usage /maintenance of asset instances

D.3.2 POSITION IN (ARCHITECTURE LAYER, SYSTEM LAYER, HIERARCHY LEVEL)

OPC UA can be applied at the levels field device, control device, station, work center and enterprise according to IEC 62264 / IEC 61512 referenced by RAMI4.0.

Annex E IPSO INFORMATION MODEL

E.1 BACKGROUND INFORMATION

IPSO Smart Objects is a common design pattern and object model to enable data interoperability between IoT devices. The IPSO model is designed based on the OMA SpecWorks LwM2M

[1] *https://opcfoundation.org/security/*

(LightWeight Machine to Machine) object model and for use of CoAP, but is in practice independent of these, as any RESTful protocol can be used.

IPSO Smart Objects are simple and lightweight in design, in order for them to be easily handled by constrained devices over network links with potentially limited or intermittent connectivity. As such it fits into many industrial applications with sensors and actuators.

IPSO Smart Objects were previously defined in the IPSO Alliance, which has now been incorporated into OMA SpecWorks. The specifications are available under a MIT license at

https://www.omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/

The full object set is in the OMNA registry

http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html

E.2 SERVICES NEEDED FROM CONNECTIVITY FRAMEWORK LAYER

E.2.1 DATA FORMAT

The IPSO data model is defined in XML and human readable specification text.

Content formats are those specified by the OMA LWM2M specification:

- Resource values: text/plain, tlv
- Objects: text/senml+json, application/cbor, binary/tlv
- Attributes: link-format, link-format+json

E.2.2 INTERACTION ABSTRACTION

E.2.2.1 SERVICES

IPSO Smart Objects are designed for RESTful services. The device interaction is through simple Objects on the device. Objects and their resources are mapped into the URI path like this

Object ID/Instance ID/Resource ID

An object type semantically represents a single measurement, actuation, or control point, such as a temperature sensor. An object has resources that represent a particular view or aspect of the object, such as current value, max value or engineering type.

Numeric constants are used for IDs (e.g. Object ID 3303 is Temperature), with the available range split between standardized and non-standardized IDs.

As an example, the URI to read instance #0 of a temperature sensor would be /3303/0/5700 where 3303 is the standardized Object ID for Temperature sensor and 5700 is the standardized Resource ID for “Sensor Value”. A GET REQUEST to this URI would retrieve the current sensor value of the sensor.

Objects can have different Operations (Read, Write, Execute) that in turn map to what kind of CRUDN operations they receive.

Depending on what is allowed by the model, Objects can have multiple instances, and can then be created or deleted.

For more complex data models, it is possible to create composite objects using object linking, where a top object can have resources of link type, that in turn link to additional objects. In this way composite objects can easily be created and navigated, with a large degree of reuse and granularity, without resulting in large, nested complex data structures.

E.2.2.2 QUALITY OF SERVICE

Quality of service is not provided by the IPSO framework, but QoS support can be provided by other components of the stack.

E.2.2.3 DESIGN PATTERNS

The primary design pattern for communication is Request-Reply.

It should be noted that a LwM2M client (e.g., a device) will act as both a CoAP client and a CoAP server during different parts of its lifetime, it is a CoAP client during e.g., bootstrap procedure and a CoAP server during “normal operation”.)

E.2.2.4 PROTOCOLS

IPSO and LwM2M are primarily designed to be used with CoAP, which is a RESTful protocol similar to HTTP for constrained devices running over UDP. In addition to CoAP, LwM2M also supports TCP, MQTT, SMS and NIDD (non-IP data delivery).

E.3 CORE CHARACTERISTICS

E.3.1 ONTOLOGY

No formal ontology has been defined by OMA. However, the model is well specified, simple and coherent so there should not be a problem to define a formal ontology.

E.3.2 CONTEXTUAL DATA

IPSO Semantics are defined at specification time. There are various ways to add additional semantic information (e.g. application type). The concept of reusable resources also allows for semantic reuse.

E.3.3 METAMODEL

IPSO Smart Objects are based on the web architecture and RESTful principles. Objects contain resources that are referenceable with URIs, and those resources are operated with using the fundamental CRUDN operations, with representations in specified formats sent back to the requestor.

E.3.4 Observation & Measurement

E.3.4.1 Time

Unix time

E.3.4.2 Location

IPSO does not currently include a separate specific location description (i.e., no “left thermostat”).

The current set of standardized and defined objects includes several data models that are relevant for location aspects however, such as Depth, Altitude, Direction, Gyrometer and GPS Location.

E.3.4.3 Sensors

IPSO Smart Objects is very well suited for carrying sensor information. The currently specified objects include several kinds of basic sensors, such as Presence Sensor, Temperature Sensor and Humidity Sensor etc. Additional sensor types are being defined by IPSO WG in OMA SpecWorks. More information may be found on IPSO’s home page *https://oma.groups.io/g/ipso*

E.3.4.4 Data Format

The serialization formats JSON, CBOR and raw values are currently defined.

E.3.5 Actuators and tasking

E.3.5.1 Actuators

IPSO Smart Objects has current support for basic actuators (e.g. Power Switch), but more can be added.

Actuators are handled with either Write or Execute operations, depending on the action.

E.3.5.2 Tasking

Parameters can be set on executable resources, they are then passed along on resources. The type and values are known to the client based on the schema.

Higher level orchestration and e.g., conditional execution is not standardized, but instead intended to be provided by a higher-level application.

E.3.6 Security

The IPSO data model does not itself define a security model. However, LwM2M does provide full life-cycle security support for IoT devices, from bootstrapping to device operation.

E.3.6.1 Authorization

Granular access control based on roles can be set in LwM2M for resources, and this mechanism can be used to handle data access in IPSO. (Per management servers, that can set access control rules)

E.3.6.2 Authentication

Endpoint ID concept in LwM2M (URN). Identities are bound to security credentials, which for LwM2M includes certificates, pre-shared keys, public Raw keys and PKI deployments.

E.3.6.3 Confidentiality

Both transport and end-to-end object encryption is supported by (D)TLS and OSCORE respectively.

E.4 IIoT System Information Model Types

E.4.1 Position in System Life Cycle

IPSO Smart Objects can be used in the development and deployment of IoT devices such as sensors to describe IoT things. It can also be used in operation as a way to describe and structure transferred data.

E.4.2 Position in (Architecture Layer, System Layer, Hierarchy Level)

Mainly at the field device level (with resource constrained devices such as simple sensors and actuators)

Annex F One data Model and Semantic Definition Format (SDF)

F.1 Background Information

The One Data Model (OneDM) initiative intends to provide a general data model for IoT devices, with additional interaction semantics for a device’s affordances. The OneDM family of activities will over time include both a general format for describing the models, called Semantic Definition Format (SDF) and a set of common models for IoT devices hosted by the OneDM project.

The OneDM initiative grew out of a shared effort between several IoT SDOs, including OMA SpecWorks, OCF, Bluetooth SIG and Zigbee Alliance to find a way to harmonize their data models. The chosen path forward included the definition of a neutral format to describe the models, from which translations to and from the respective ecosystems is possible. SDF, which is the name of the neutral format, is now adopted into IETF for future development, but already today it is useful to define data models and translate to involved ecosystems.

SDF is a general format for describing IoT devices and their affordances, typically represented in JSON. SDF introduces definitions for fundamental device composition (Things and Objects) as well as for describing general interaction patterns, including Properties, Actions and Events together with the data that may be exchanged.

The description of SDF here is based on the version 1.0 of the format. That version was the starting point for the work in IETF, where work has now continued in the ASDF WG. It is expected that SDF will evolve and improve as it progresses through the IETF specification process, and this may in turn add or change functionality described in this annex.

One Data Model's Liaison Group's home page: *https://onedm.org*

One Data Model's Liaison Group's GitHub page: *https://github.com/one-data-model*

Current SDF 1.0 draft *https://datatracker.ietf.org/doc/draft-onedm-t2trg-sdf/*

IETF ASDF WG *https://datatracker.ietf.org/wg/asdf/about/*

F.2 Services Needed from Connectivity Framework Layer

None.

F.2.1 Data Format

SDF is defined in JSON, using some formats from JSON-schema.

The format is machine readable and can be validated with either CDDL (RFC8610) or json-schema based approaches.

SDF is translated to the target ecosystem, with tools provided by the respective ecosystem.

F.2.2 Interaction Abstraction

F.2.2.1 Services

SDF is not tied to a specific service model. However, it maps well to a RESTful approach as several of the ecosystems that contributed to the creation of SDF are built on RESTful principles.

The SDF file for a Thing defines Interaction Affordances as metadata that show and describe the possible choices for interaction that the Thing has. This is similar to how Web of Things Thing Descriptions describe the available interaction affordances of a WoT Thing.

F.2.2.2 Quality of service

SDF (currently) has no notion of QoS.

F.2.2.3 Design Patterns

SDF has three core interaction patterns: Property (sdfProperty, immediate get/set), Action (sdfAction, potentially multi-stage) and Event (sdfEvent, telemetry).

F.2.2.4 Protocols

SDF is independent of the underlying protocol. A binding to the target environment is used to translate the model into the target ecosystem, and the protocols that are used are therefore dependent on the target ecosystem.

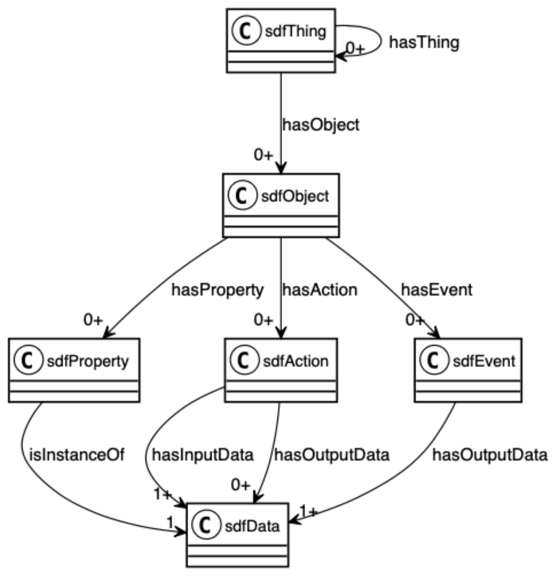
F.3 Core Characteristics

F.3.1 Ontology

SDF defines a vocabulary of terms that are relevant for modeling IoT devices. The vocabulary can be used as basis for translation into RDF models.

Based on the work of the OneDM group, the basic SDF vocabulary will be used to create a wider vocabulary describing the adopted IoT models.

Figure_Apx 11 shows the main classes of SDF being used to describe a device and its affordances. The top-level class is Thing, which represents a device or part of a device. One Thing may consist of multiple Things in turn, allowing for composition of complex Things. A Thing may also have Objects, that are the main reusable semantic components of definition in SDF. The use of the term Object is common for many IoT ecosystems to describe a point of interest, like temperature. An Object may have affordances of three different kinds: Properties, Actions and Events. A Property models a state in the object, like current or max for a measured value. A Property may be read, and in some cases written. An Action models more complex interactions than are possible with Properties only. For instance, an Action may model outside effects like turning on or off a switch. Finally, Events model telemetry, the capability to listen to the change of state and other things on the object, also possibly associating more capabilities like sequencing and consistency, that might not be possible with properties alone.



Figure_Apx 11: Main SDF Classes

F.3.2 Contextual Data

SDF does not (currently) define contextual data.

F.3.3 Metamodel

As the main ecosystems that SDF grew out of were based on web architecture and RESTful principles, that heritage is visible but not mandatory to implement. In addition to the RESTful principles, SDF adds Actions as multi-stage operations and Events as telemetry observation mechanisms.

It is important to note that SDF is created to enable translation and cross interoperability of different ecosystems, it is not intended to be a wire format. In deployment, SDF models will be translated into ecosystem-specific models for the target ecosystem.

F.3.4 Observation & Measurement

SDF uses Events to model observation.

Units of measurement come from SenML. (RFC 8428)

F.3.4.1 Time

RFC3339 profile of ISO 8601.

F.3.4.2 Location

Not currently specified.

F.3.4.3 Sensors

One of SDF’s key objectives is to model sensors.

Note that SDF only specifies the representation format for general sensors, actual sensor models represented in SDF are provided by the One Data Model initiative.

F.3.4.4 DATA FORMAT

SDF provides a way to define a data type (sdfData) that allows for reusable type definitions. Example of type definitions include inclusion of data constraints and addition of semantic anchors. The defined type is used for the payload used with the interaction affordances. For instance, it may be used to define the input to an Action as well as the output of an Action.

The available types for data type definition come from JSON, JSON-schema and CDDL.

Note that SDF is not directly serialized to data on the wire on its own, instead it is translated to a target ecosystem and uses the serialization of that ecosystem.

F.3.5 ACTUATORS AND TASKING

F.3.5.1 ACTUATORS

Actuators may be modeled as Properties or Actions, depending on the complexity and operational pattern of the actuator.

Note that SDF only specifies the representation format for general actuators, actual actuator models represented in SDF are provided by the One Data Model initiative.

F.3.5.2 TASKING

Actions are associated with a Data type that contains the data for the action.

F.3.6 SECURITY

SDF does not today define a security model, it is provided by the end ecosystem.

F.3.6.1 AUTHORIZATION

F.3.6.2 AUTHENTICATION

F.3.6.3 CONFIDENTIALITY

F.4 IIOT SYSTEM INFORMATION MODEL TYPES

F.4.1 POSITION IN SYSTEM LIFE CYCLE

SDF can be used in different ways in different parts of the system life cycle, for example:

1. Static translation at design/integration time. This is the basic use case where data model from ecosystem X is first translated into SDF, and then translated to another ecosystem Y without loss of semantic information.

2. Runtime semantic proxy. A use case under investigation where a dynamic mapping can be achieved in a proxy point between two systems, using translation to and from SDF from the different system's ecosystems.

F.4.2 Position in (Architecture Layer, System Layer, Hierarchy Level)

SDF is used to model data models for IoT devices, currently composable up to Thing level. From an architecture perspective that may describe field devices and potentially control devices.

Annex G Acronyms

AAS	Asset Administration Shell
ABAC	Attribute-based access control
AGV	Automated Guided Vehicle
CRS	Coordinate Reference System
CRUD	Create, Read, Update and Delete
CSV	Comma-separated values
DOA	Digital Object Architecture
DX	Digital Transformation
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IDSA	International Data Spaces Association
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IP	Intellectual Property
IRDI	International Registration Data Identifier
ISO	International Standards Organization
ITS	Intelligent Transportation System
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
MDA	Model Driven Architecture
NGO	Non-Governmental Organization
MQTT	Message Queuing Telemetry Transport
ODRL	Open Digital Rights Language
OGC	Open Geospatial Consortium
OMG	Object Management Group
OASIS	Organization for the Advancement of Structured Information Standards
OWL	Web Ontology Language
QoS	Quality of Service
RAMI4.0	Reference Architecture Model Industrie 4.0
RBAC	Role-based access control
RDF	Resource Description Framework
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language
UML	Unified Modeling Language
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business
URN	Uniform Resource Name
UTC	Universal Time Coordinated
WGS	World Geodetic System
W3C	World Wide Web Consortium

XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

Annex H GLOSSARY

Industrial Internet Consortium (IIC)	an open membership, international not-for-profit consortium that is setting the architectural framework and direction for the Industrial Internet. Founded by AT&T, Cisco, GE, IBM and Intel in March 2014, the consortium's mission is to coordinate vast ecosystem initiatives to connect and integrate objects with people, processes and data using common architectures, interoperability and open standards.
Industrial Internet of Things (IIoT)	describes systems that connects and integrates industrial control systems with enterprise systems, business processes, and analytics. Note 1: Industrial control systems contain sensors and actuators. Note 2: Typically, these are large and complicated system.
Internet Engineering Task Force (IETF)	The Internet Engineering Task Force (IETF) develops and promotes voluntary *Internet standards*, in particular the standards that comprise the *Internet protocol suite* (TCP/IP). It is an open *standards organization*, with no formal membership or membership requirements.

Annex I REFERENCES

[CAV-2020]	Cavalieri, S.; Salafia, M.G. Insights into Mapping Solutions Based on OPC UA Information Model Applied to the Industry 4.0 Asset Administration Shell. Computers 2020, 9, 28. *https://www.mdpi.com/2073-431X/9/2/28*
[IIC-I4.02020]	IIC and Plattform Industrie 4.0: Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0 *https://www.iiconsortium.org/pdf/Digital-Twin-and-Asset-Administration-Shell-Concepts-and-Application-Joint-Whitepaper.pdf*
[IIC-IIRA2015]	IIC: The Industrial Internet, Volume G1: Reference Architecture Technical Report, version 1.7, 2015-June-04 *http://www.iiconsortium.org/IIRA.htm*
[IIC-2016]	IIC: The Industrial Internet of Things, Volume G4: Security Framework, 2016-September-26 *https://www.iiconsortium.org/IISF.htm*
[IIC-2017]	IIC: The Industrial Internet of Things, Volume T3: Analytics Framework, 2017-October-23 *https://www.iiconsortium.org/industrial-analytics.htm*
[IIC-2018]	IIC: The Industrial Internet of Things, Volume G5: Connectivity Framework, IIC:PUB:G5:V1.01:PB:20180228, 2018-February-28 *https://www.iiconsortium.org/IICF.htm*
[IIC-2020a]	IIC: Digital Twins for Industrial Applications, 2020-February-18, *https://www.iiconsortium.org/stay-informed/digital-twins-for-industrial-applications.htm*
[IIC-2020b]	IIC: Digital Transformation in Industry White Paper, 2020-July-29, *https://www.iiconsortium.org/pdf/Digital_Transformation_in_Industry_Whitepaper_2020-07-23.pdf*
[IIC-2020c]	IIC: The Industrial Internet of Things Vocabulary, version 2.3, 2020-10-05 *http://www.iiconsortium.org/vocab/index.htm*
[ISO23247-3]	Automation systems and integration — Digital Twin framework for manufacturing — Part 3: Digital representation of manufacturing elements.
[JAC-2020]	Jacoby, M., Usländer, T. Digital Twin and Internet of Things—Current Standards Landscape. MDPI Applied Sciences 2020, 10(18), 6519; 2020. *https://doi.org/10.3390/app10186519*

[KOT-2018] Kotsev, A., Schleidt, K., Liang, S., Schaaf, Hylke van der, Grellet, S., Lutz, M., Jirka, S., Beaufils, M.   Extending INSPIRE to the Internet of Things through SensorThings API. *http://www.mdpi.com/2076-3263/8/6/221*

[OMG-2002] Object Management Group, Ed. Meta Object Facility (MOF) Specification, Version 1.4. Object Management Group: Milford, MA, USA, 2002 *https://www.omg.org/spec/MOF/1.4/PDF*

[Plattform Industrie 4.0-2020a]
German Federal Ministry for Economic Affairs and Energy (BMWi), "Details of the Asset Administration Shell –Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC012.0)", 2020-November,
https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html

[Plattform Industrie 4.0-2020b]
German Federal Ministry for Economic Affairs and Energy (BMWi), "Details of the Asset Administration Shell –Part 2: Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (Version 1.0RC01)", 2020-November,
https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html

AUTHORS AND LEGAL NOTICE

This document is a work product of the Industrial Internet Consortium Distributed Data Interoperability and Management Task Group, chaired by Andrei Kolesnikov (IOT Association), Niklas Widell (Ericsson) and Kym Watson (Fraunhofer IOSB).

Authors: The following persons contributed substantial written content to this document: Takuki Kamiya (Fujitsu), Andrei Kolesnikov (IOT Association), Brett Murphy (RTI), Kym Watson (Fraunhofer IOSB), and Niklas Widell (Ericsson).

The original document structure and motivation was proposed by the late Brett Murphy of RTI, a founding co-chair of the DDIM TG.

Contributors: The following persons contributed valuable ideas and feedback that significantly improved the content and quality of this document: Kazuhiro Takahashi (NEC) and Mitch Tseng (Tseng Infoserv).

Technical Editor: Stephen Mellor (IIC staff) oversaw the process of organizing the contributions of the above Authors and Contributors into an integrated document.

In Memoriam: Brett Murphy

We lost a beloved colleague recently. Brett Murphy has been a member of the Industrial Internet Consortium since Real-Time Innovations (RTI) joined within days of our launch in 2014. But not only was Brett a member, he was a trusted colleague, friend, key contributor, voice of reason and shining light to us all.

In consortia culture, where contributors offer their precious free time and intellect on top of their day jobs, Brett was always willing to help. He applied his rock-solid reliability to his own brilliant ideas, as well as when called upon to lead a new initiative, so others could learn and follow.

We know the dedication he shared with us was eclipsed by his dedication to his family. We share Brett's family's and the RTI family's grief and thank them for sharing Brett with us for these past 6 years. His contributions, and more importantly, his presence in our lives will never be forgotten.

Brett was a key contributor to this report. It is fitting that we memorialize him here.